

Evaluating Java for Game Development

*By Jacob Marner, B.Sc.
Department of Computer Science
University of Copenhagen, Denmark
(jacob@marner.dk)*

March 4th, 2002

1	PROLOGUE	4
1.1	CONVENTIONS	4
1.2	KNOWLEDGE REQUIRED OF THE READER.....	4
1.3	THE STRUCTURE OF THE REPORT	4
1.4	ACKNOWLEDGEMENTS.....	5
1.5	HOME PAGE.....	5
2	INTRODUCTION	6
3	GAME DEVELOPMENT.....	7
3.1	A DEFINITION OF COMPUTER GAMES	7
3.2	GAME PLATFORMS	7
3.3	THE BUSINESS OF GAME DEVELOPMENT	8
4	INTRODUCING JAVA	12
4.1	THE TERM JAVA	12
4.2	OVERVIEW	12
4.3	THE JAVA PROGRAMMING LANGUAGE	12
4.4	THE JAVA PLATFORM	13
4.5	THE JAVA STANDARD LIBRARY	15
4.6	THE JAVA DEVELOPMENT KIT	15
5	JAVA/C++ LANGUAGE COMPARISON.....	17
5.1	APPLICATION VS. SYSTEM PROGRAMMING	17
5.2	OBJECT ORIENTATION	17
5.3	PORTABILITY	17
5.4	DYNAMIC DATA ALLOCATION AND GARBAGE COLLECTION	19
5.5	ARRAY BOUNDS CHECKING	20
5.6	REFERENCES AND POINTERS	20
5.7	TYPE CASTING.....	21
5.8	THE STANDARD LIBRARY	21
5.9	PROTOTYPES AND HEADERS.....	22
5.10	DOCUMENTATION GENERATION	22
5.11	LANGUAGE SIMPLICITY	23
5.12	OPERATOR OVERLOADING	24
5.13	UNSIGNED INTEGRAL DATA TYPES.....	24
5.14	CONSTANT PARAMETER TYPES.....	24
5.15	PACKAGES AND NAMESPACES.....	25
5.16	MULTIPLE CLASS INHERITANCE	25
5.17	TYPE DEFINITIONS AND ENUMERATIONS.....	25
5.18	PREPROCESSING.....	26
5.19	TEMPLATES AND GENERICS.....	27
5.20	DYNAMIC CLASS LOADING.....	28
5.21	REFLECTION.....	28
5.22	DEBUGGING AND IDEs	28
5.23	SUMMERY	30
6	JAVA PERFORMANCE	32
6.1	THE ORIGIN OF THE JAVA MYTH.....	32
6.2	WHEN IS JAVA SLOW?.....	33
6.3	WHEN IS JAVA FAST?	34
6.4	WHY IS THIS JAVA PROGRAM SLOW?.....	41
6.5	MEMORY FOOTPRINT	42
6.6	SELECTING THE HEAP SIZE	43

6.7	THE PERFORMANCE OF JNI	44
6.8	TEMPLATES / GENERICS	44
6.9	STRINGS	45
6.10	TWEAKING AND READABILITY	45
6.11	CONCLUSION.....	46
7	BENCHMARKS	47
7.1	INTRODUCTION	47
7.2	CHOICE OF COMPILER/VIRTUAL MACHINES	47
7.3	THE SYSTEM SETUP	49
7.4	THE BENCHMARK APPLICATIONS	49
7.5	BENCHMARKING BOTH TWEAKED AND UNTWEAKED APPLICATIONS	50
7.6	WARM-UP PERIODS	50
7.7	GENERATION OF TEST DATA.....	51
7.8	A PARTICLE SIMULATOR	52
7.9	AN OBJECT-ORIENTED VERSION OF LIFE	57
7.10	A 3D DEMO.....	62
7.11	REVIEWS OF THIRD PARTY BENCHMARKS.....	68
7.12	CONCLUSIONS AND THOUGHTS	72
8	EVALUATING JAVA FOR A GAME PROJECT.....	74
8.1	WHY USE JAVA?	74
8.2	CONSIDERING THE RISKS	75
8.3	OPENGL, DIRECTX AND OTHER LIBRARIES	79
8.4	JAVA3D	79
8.5	DIFFERENT WAYS JAVA CAN BE USED IN GAMES	81
8.6	COMMERCIAL GAMES USING JAVA	81
8.7	PORTABILITY	83
8.8	THE PERFORMANCE OF JAVA IN GAMES.....	84
8.9	SHOULD I USE JAVA FOR MY NEXT GAME PROJECT?.....	85
9	CONCLUSION	87
	APPENDIX A: REFERENCES	88
	APPENDIX B: UNTWEAKED PARTICLES SOURCE CODE	92
	APPENDIX C: TWEAKED PARTICLES SOURCE CODE.....	118
	APPENDIX D: UNTWEAKED LIFE SOURCE CODE	145
	APPENDIX E: TWEAKED LIFE SOURCE CODE	154
	APPENDIX F: UNTWEAKED 3D DEMO.....	163
	APPENDIX G: TWEAKED 3D DEMO.....	226
	APPENDIX H: EDITED THIRD PARTY BENCHMARK	305
	APPENDIX I: BENCHMARK RESULTS.....	308

1 Prologue

This report is registered as report 01-11-11 at the Department of Computer Science, the University of Copenhagen, Denmark. The report was written by Jacob Marner with Associate Professor Jyrki Katajainen as supervisor. It was turned in on March 4th, 2002.

The purpose of this project is to examine whether the use of Java for games is advantageous compared to the traditional language of choice, C++. This is not an easy question to answer, and as we will see later the answer will depend on several project specific issues.

The target group of the report is professional game programmers with little or no knowledge of Java, who wonder whether Java would be beneficial in their future projects. The report generally assumes that the reader is skeptical about Java.

1.1 Conventions

When words or phrases written in `courier` are used in the report, they always refer to pieces of C++ code, Java code, or file names.

The first time a new special term is introduced and explained in the report, the term will be written in **bold**.

References will be denoted with square brackets.

1.2 Knowledge required of the reader

The reader of this report is expected to have at least a background in Computer Science equaling a Bachelor's degree (B.Sc.) and to be experienced in practical programming besides that taught at university undergraduate courses.

Furthermore, the reader should have a strong knowledge of C++ programming. Any prior knowledge of Java is not needed since the needed information this will be given in the report.

Although it is generally expected that the reader knows about game development in general terms, this is not traditional B.Sc. knowledge. This will therefore be briefly introduced early in the report.

It is recommended that readers without knowledge of C++ read something about it first. One good C++ primer is [Lippman]. [Kernighan] and [Stroustrup] are recommended as language references.

1.3 The structure of the report

Chapter 2 will introduce the main question that this report aims to answer.

Next, chapter 3 will attempt to define games and give an overview of the game market today and narrow down the focus of this report.

Chapter 4 will briefly introduce Java to the uninitiated. This will not be a programming tutorial. Rather it will introduce the various concepts and technologies that are important to understand the world of Java.

In chapter 5, the Java and C++ languages will be compared and the pros and cons of each will be discussed.

This will be followed up in chapter 6 with a discussion of the performance of Java compared to that of C++.

Chapter 7 runs some benchmarks measuring the relative performance of Java vs. C++.

Chapter 8 is perhaps the most important chapter in the report. Here it is considered how Java can be used for game developed and it is discussed whether doing so is a good idea.

Finally, in chapter 9 the conclusions of the report will be summarized in a short concise form.

If you do not have time to read all of the report, I recommend reading only chapters 8 and 9.

Following the conclusion is the reference list (Appendix A), the benchmark source code (Appendix B - Appendix H) and the benchmark results (Appendix I).

1.4 Acknowledgements

I would like to thank the people at the forums at <http://www.javagaming.org>. Without the many – sometimes heated – discussions I have had at those forums I would not have been able to write this report.

1.5 Home page

The home page of this report is:

<http://www.rolemaker.dk/articles/evaljava/>

On this page you can find the report itself, the source code for the benchmarks, the original Excel spreadsheet with the benchmark results, and any errata to the report.

2 Introduction

In the last couple of years the use of Java has become very widespread in the IT-industry. It is used mainly for Internet software, but it is also popular for regular applications and in embedded systems. In fact, according to [Galli] over half of all U.S. developers use Java and this share is projected to rise with additional ten percent during 2002.

This rise in the interest in Java is mainly due to the fact that developers have a higher productivity when using Java compared to traditional languages such as C or C++, especially when doing cross-platform or Internet development.

The use of Java is not widespread among developers of computer games sold in retail. It is therefore natural to ask why this is so, and ask whether game developers can benefit from using Java. If it is true that Java can reduce production cost, it might be advantageous to use it Java for game development.

This is the question that I aim at answering in this report. I will compare Java with C++, consider the performance of Java, clarify some common myths, run some benchmarks, and finally discuss various ways Java can be used in game development.

It would also have been interesting to evaluate the use of the Microsoft .NET platform (and C#) for game development, but doing such an evaluation is a large topic in itself and is therefore outside the scope of this report.

3 Game development

In this chapter I will attempt to give an overview of what game development is and then narrow down what segments of the game industry that will be considered in this report.

If you are a professional game developer then you can skip this chapter. All you need to know is that the remainder of this report only will consider games targeted at personal computers and game consoles.

3.1 A definition of computer games

A computer game is *a piece of interactive software created with the primary purpose of entertaining the user.*

This definition is not clear-cut but is still useful, since it characterizes the prototypical computer game.

Seen from a programming viewpoint, a game typically is an *interactive real-time simulation*, possibly distributed. A model is maintained of a virtual world and this model is updated and visualized in real-time according to the dynamics of the model and the interaction with the user or users.

Computer game development is normally not viewed as constituting a computer science field by itself. Computer game development is, however, an application of computer science that makes use of many different fields, including, but not limited to graphics, audio, HCI, operating systems, compiler design, algorithms, networks and distributed systems, software engineering, software design, and artificial intelligence.

3.2 Game platforms

Games are getting more and more popular each year and therefore appear on a multitude of platforms. To mention the most common:

- Personal computers. (Primarily for Microsoft Windows, but some are also being made for MacOS and Linux. Games for personal computers usually support networking)
- Dedicated Game Consoles that connect to a regular television. (This includes machines such as the Sony Playstation 1 and 2, the Nintendo 64, the Nintendo GameCube, and the Microsoft Xbox)
- Arcade machines.
- Web-based games.
- Dedicated hand held consoles with an embedded screen. (Primarily the Nintendo Gameboy Color and the Nintendo Gameboy Advance)

- Interactive Television. (Games hosted at a television broadcaster and controlled via a regular tone telephone)
- Set-top boxes. (Devices provided by television cable networks to enhance the capabilities of the television)
- Cellular phones.
- Other hand held devices. (This includes devices such as the Palm Pilot)
- Entertainment kiosks. (Is usually placed at bars and cafes)
- Slot machines. (Is usually placed at bars as a replacement for the older one-armed bandits)

If we ignore the games in which you play for money, then personal computers and dedicated consoles (including the hand held ones) represent the vast majority of the revenue involved in games. Of these, the dedicated consoles represent approximately 2/3 of the revenue and the personal computers the last 1/3. [Veronis]

On the personal computer scene, Intel Windows PCs dominates the market completely when it comes to games. Apple Macs have a small market share and Linux almost none. This low revenue for Linux users has probably to do with that Linux users are accustomed to free software. Furthermore, Linux is mostly popular as an operating system for professionals – not home users.

The dedicated game console market is currently dominated by the devices by Sony and Nintendo. Sega has a small market share but this has been diminishing for some time. Sega has taken the consequence of this trend and has stopped producing consoles. Microsoft has recently (November 2001) released its first console, the Xbox, and this is likely soon to become a dominating player.

There has been a rising in the popularity of web based games especially among groups that do not usually play games; woman and middle aged and older men. I have been unable to determine the impact these games has on the distribution of revenue, but I believe that it is insignificant compared to the personal computer and console market.

Since the personal computers and dedicated consoles dominate the game industry scene I will focus on only these platforms for the remainder of the report.

3.3 The business of game development

The game market, from a business point of view, is interesting because it is a market that is in rapid growth. In several years in a row the market growth was in excess of 20% a year [Veronis] although last year this growth was somewhat lower.

It is often said that the computer game market is larger than the film market. This is not entirely true. It is true; however, that the revenue of PC and console games *worldwide* in 2000 including the sales revenue of the actual consoles was higher than the *box office* sales in the *U.S.* that same year. To say that the gaming market is bigger than the film market one should also include the world wide sales of films in theaters, on VHS, DVD,

merchandising, the actual televisions and video players and so forth. When doing the comparison that way, games are far behind the movie industry [Wardell01].

In 2001 the U.S. movie industry has seen a growth in excess of 30%. This is more than the growth the same year in the game industry, so it is no longer clear if even the statement above holds any more.

Nonetheless, it sounds like there is a lot of money to be earned in games at the moment. Unfortunately this is not entirely the case, because the amount of developers has grown even more than the market. The result is that the competition has become harder. [DFC 1]

Because of this, issues such as marketing become vital to the success of a game. It becomes very essential to become one of those few games that retailers put in the front shelf and that magazines use many pages to cover. If this goal is not achieved then sales will be significantly lower [Wardell00]. Some analytics claim that it is a myth that shelf space is limited [DFC 2] but I think those analytics forget to take into account that many stores, such as Blockbuster, have no more than 10 different games on sale at any given time and that many stores have shelf space with varying levels of prominence.

In order to compete for prominent shelf space, each game attempts to become the one with the highest profile. This currently increases the cost of developing these games to the range of 2-3 millions US\$ combined with a fairly high risk of failure. This is the kind of games we will call **high profile games**.

Alternatively, some developers focus on niche groups of users and decrease cost equivalently or even just focus on producing a high quantity of low-budget games rather than a few high-budget games. Each of these games will have significantly lower sales than their high profile counterparts, but because of the lower budget they may still be profitable. We will call these games **low profile games**.

High profile and low profile games often have different goals in development. The high profile games must be technically advanced. To get the attention of retailers and the media, focus has traditionally been on graphics. Performance is therefore vital. Low profile games, on the other hand, cannot compete with this under any circumstances, so here the primary goal is to keep costs down.

It is here presented as if games either are fully high profile or fully low profile games. In practice, games may be anywhere on the scale between these two extremes.

We will consider both high and low profile game in the discussions in this report.

3.3.1 The business model of PC games

The typical business model for a PC game is similar to that known from the book or music industry in that there is a strong division between *developers* and *publishers*. The developer does the actual software development while the publisher markets the game, produces the actual physical media and packages and distributes it. There has been a recent trend for developers to sell their games directly on the Internet but this is generally only a viable solution for low profile games. Another solution for low-profile games is to

make the games web-based, in which case they are often free to play and financed via advertisements.

Usually a developer presents a demo to the publisher and then the publisher finances the development of the rest of the game. If the development team has previously created success titles then a demo might not be necessary. Since financing games usually is risky for the publisher, the publisher earns the majority of the resulting income.

3.3.2 The business model of console games

On the console market the business model is almost the same, but with some important differences.

First, the console manufacturers sell the consoles with a loss to increase their market share. In return they require royalties for each game sold for the platform.

Because of these royalties, the developer profit of selling console games is less than for PC games. So although two thirds of the revenue is on the console market, this does not mean that two thirds of *developer profits* are gained on this market.

Second, to ensure that they can get their royalties, the console manufacturers require that only developers with a development license may develop software for their specific console. Vital information about the software interfaces of the consoles is kept secret and is only given to licensed developers. Becoming a licensed developer is not an easy task. This also allows the manufacturers to control the content of games being published on their platform and thereby to avoid, for instance, adult content.

This licensing scheme also means that there are significantly fewer very low profile games available for consoles in comparison with what is available for PCs.

Side note: Linux on the Sony Playstation 2

In 2001 Sony released a limited number of Linux kits (including a hard disk, keyboard and mouse) for the Sony Playstation 2 in Japan only. They have furthermore announced a general release in Japan, the US, and UK in June 2002.

This move has led some people to believe that Sony will allow more people to access information about the Playstation 2 and to develop games for it. This is true to the extent that it is possible to develop standard Linux applications for the Playstation 2, but not that it will be possible to develop full Playstation 2 games. First, Linux on the Playstation 2 uses a significant amount of memory (the Playstation 2 has only 32 MB RAM) making it impossible to run competitive games. Second, the software SDK for the system is still kept secret. I have been told by Jeff Kesselman of Sun Microsystems, who works with Sony on the Sony Java initiative, that Sony *never* will release the specifics of their system as this would be they would loose control of the platform and the lucrative royalties that follows.

3.3.3 Who plays games?

In contrary to popular belief computer games are no longer played primarily by children and teenagers. Today, games are played just as much by adults, although the

main bulk still is below 35 years old and male. This change has occurred both because computer games has increased much in popularity in the last decade and because those game players that used to be children have grown up and still play games. [DFC 2]

The various platforms have chosen to focus on different target groups for their products:

- Nintendo focuses on family games, i.e. games that children can play with their parent. The target is pre-teenagers and to some extent also teenagers.
- Sony and Sega focuses on male teenagers and male young adults (up to 35 years old).
- The PC market has no specific focus but there is a tendency to focus on male young adults.

Worthy of note is that web-based low profile games on the PC are popular especially among woman and middle aged adults (over 30 years old). This is probably because many of these very low profile games are financed via advertisements and therefore free to play for the end-user. This makes this more accessible to groups that would not normally spend money on games.

Consoles are generally attractive to players that are either young or inexperienced with computers, because they are much simpler to use than a PC.

4 Introducing Java

In this chapter Java will be briefly introduced to readers with no knowledge of it; enabling them to understand the remainder of the report. This chapter will not teach how to program Java, but rather attempt to give an overview of the concepts involved.

For more detailed information about Java please go to the official Java web site at <http://java.sun.com/>.

4.1 The term Java

The word **Java** is trademarked Sun Microsystems and are used to cover several related concepts:

- **The Java programming language.**
- **The Java platform or implementations thereof.**
- **The Java standard library.**
- **The Java Development Kit (JDK)**

Although these concepts are related it is important to distinguish between them since they no necessarily need to be used together. Each of these will be introduced in the following sections.

4.2 Overview

Traditional Java applications are applications that are written in the Java programming language and then compiled into byte code form – called **Java byte code**.

These byte code files can then be executed on any platform that complies with the Java platform. This is usually done by the implementation of a **Java virtual machine** which emulates the Java platform on another platform. For instance, with the use of a Java virtual machine for Linux, one is able to execute Java byte code on Linux.

The major benefit of using a virtual machine over compiling directly to machine code is that the application in compiled form can be completely portable. Furthermore, the Java virtual machine protects the underlying machine from illegal instructions and erroneous memory access. This is important when using mobile code. [Coulouris]

4.3 The Java programming language

The Java programming language is an imperative object oriented programming. Its syntax is very similar to that of C++. In effect C++ programmers can usually read and learn Java source code without much trouble.

The language is of a slightly higher abstraction level than C++, but not so high that it like ML or Python rightly can be called a high level programming language.

The details of Java can be found in the Java language report [Gosling00].

A detailed discussion of the differences between the C++ and Java programming languages can be found in chapter 5.

4.4 The Java platform

The Java platform is a standard that tells how Java byte code should behave. A specific implementation of the Java platform is called a Java run-time environment (JRE).

The Java run-time environment usual consists of two parts:

- A Java virtual machine (JVM)
- An implementation of the Java standard library (see section 4.5).

4.4.1 Virtual machines

Traditionally Java applications have run very slowly because the virtual machines were simple interpreters that read and executed byte code instructions one at the time. Due to improvements in virtual machine technology this is no longer the case. We will discuss Java performance in more depth in chapter 6.

Fortunately, implementing a virtual machine as a simple interpreter is not the only approach available. Other approaches are:

- Virtual machines with **Just-in-time (JIT) compilers** that compile the byte code to platform specific machine code at run-time just before it is needed.
- Virtual machines with **Hotspot** compilation. This is similar to JIT compilers except that it focuses on compiling the hotspots in the code and supports adaptive optimization. Hotspot compilers are often said to contain a JIT compiler.
- **Static compilation** that at design time compiles Java byte code to an executable file that is machine specific. This approach is similar to that used by traditional languages such as C or FORTRAN and is therefore not appropriate for mobile code, but is still useful for games that are sold in retail.
- **Embedded implementations.** This approach uses specialized hardware to build a machine that complies with the Java platform. This approach is currently not well suited for games on the PC and consoles and will not be considered further.

Hotspot virtual machines and static compilers are currently the most efficient approaches for executing Java on PCs and will these therefore be the main focus on this report. Hotspot compilers used more commonly than static compilers so most focus will be on those.

Static compilers cannot be called virtual machines in the traditional sense, but was included were because they replace virtual machines in order to run Java applications.

When this report refers to virtual machines in the text, this does not usually include the static compilers.

These approaches will be discussed in more detail in chapter 6.

For more details about Java virtual machines please see the Java virtual machine specification. [Lindholm]

4.4.2 Platform editions

The Java platform comes in several variants, each targeting different uses:

- **The standard edition** (J2SE) This edition is intended for personal computers and is usual the edition referred to when people discuss the Java platform. This edition will be the edition discussed in this report unless otherwise noted.
- **The micro edition** (J2ME). This edition is a scaled down version of Java for use in systems that does not have the capacity to run the standard edition. Related to the micro edition is a whole suite of technologies such as PersonalJava and EmbeddedJava. These are mainly intended for embedded systems so small that they cannot even use the micro edition.
- **The enterprise edition** (J2EE). This extends the standard edition with several features well suited for large scale server side enterprise applications.

Implementations of the standard edition often require 5-10 MB of memory just to run. This is the same for the enterprise edition but significantly less for the micro edition.

The micro edition is actually not a single implementation, since different devices have different requirements and constraints. The micro edition is divided into a set of **profiles** that defines the characteristics of the edition on a given class of devices. Several such profiles exist. For instance, set-top boxes and digital assistants each have their own profile describing a certain subset of Java to be used on those devices. If two devices share the same profile then software will be portable between them.

4.4.3 Java Applets

Besides being used for development of regular applications, Java run-time environments can also be embedded in applications, most noticeably World Wide Web browsers. These browsers can execute (client side) applications embedded in web pages. These applications are downloaded dynamically and automatically from the web page. Such embedded applications are called **Java applets**.

Due to the high portability of Java byte code the programs can (usually) run without change no matter what machine the browser runs on.

It is common knowledge that Java is useful for web-based games so applets will not be discussed in much detail in this report.

4.5 The Java standard library

The Java standard library is a large collection of functionality that can be used many common tasks, such as math, strings, common data structures, I/O, time/date handling, audio, multithreading, networking, serialization, graphical user interfaces, and cryptography. The API that interfaces the standard library from the Java programming language is usually termed the **Java core API**.

One important part of the core API is the **Java Native Interface (JNI)**. This allows Java applications to interact with C and C-like C++ code allowing them to extend to functionality of the standard library with system specific functionality.

When people refer to 100% pure Java applications, they usually refer to Java applications that make use of the Java core API only. Extra Java libraries are allowed, but these may not use JNI.

Besides the core API, Sun provides a number of **optional packages** that extends the core API in various ways. An optional package is simply a programming library. These libraries often make use of JNI to access system specific features and must therefore be specially ported to each platform. Since the number of optional packages are high the availability of these libraries are usually low except on those platforms for which Sun provides implementations; i.e. MS Windows, Sun Solaris and to some extent also Linux. It is important to realize that optional packages are nothing but normal programming libraries with the special property that are officially supported by Sun.

The optional packages provide much new functionality. Examples are 3D graphics, e-mailing, advanced imaging, and access to computer communication ports. Of special interest to this report is **Java3D**, the optional package that provides 3D graphics.

4.6 The Java development kit

The Java development kit (JDK) is the name of the official Sun toolkit used to produce Java applications. It primarily consists of a compiler that compiles Java language source code to Java byte code and the full documentation of the Java core API.

Many other vendors provide their own version of the JDK, but all of them refer to their level of Java support by referring to the Sun version number.

The latest version of the JDK is version 1.4.0 which was released February 14th 2002 only few weeks before this report was published. The official Java virtual machine implementations follow the same version numbers. It is important to use latest versions of Java since older versions of the virtual machines are significantly slower than the new ones.

When working with Java it is important to know the major versions:

JRE/JDK version	Description	Release Date
1.0	A simple interpreter.	January 23, 1996
1.1	Introduced JIT compilation.	February 18, 1997
1.2	Changed to use the Java 2 platform. Matured JIT compilation.	December 8, 1998
1.3	Introduced Hotspot compilation.	May 8, 2000
1.4	Matured Hotspot compilation.	February 14, 2002

For more information about the time line of Java technology, see [Sun]. Note that although Java 1.0 was released in 1996, Java is normally said to have been released in 1995, because of the beta versions that went before it.

With each version the speed of Java applications has improved significantly. When making speed comparisons in this report we will only use version 1.4 since this represents the current state of Java most precisely.

It has been the source of some confusion that JDK versions 1.2 and later use the Java 2 platform. At least in theory these cover two different things. The first signifies the JDK while the other signifies the platform. Unfortunately, there has been made changes or additions to the platform in each JDK version so this does not hold completely. However, with JDK 1.2 major changes was made to the platform which is probably why it name was changed to the Java 2 platform.

Since JDK 1.3, Sun has provided two different virtual machines, the **client virtual machine** and the **server virtual machine**. The server virtual machine performs more optimizations than the client virtual machine but takes longer to warm up (see section 6.3.2).

5 Java/C++ language comparison

The syntax of the Java and C++ languages look very much alike, but this similarity is only partial. The similarity is made on purpose to make the transition to Java easier for C and C++ programmers.

There are a number of important differences between the languages and their development environments that will be reviewed in this chapter. Many of these differences contribute to difference in programmer productivity between the two languages. How much the increase is in Java will be discussed in chapter 8.

Not all differences between the two languages are covered here.

The review assumes the reader has a good knowledge of C++.

This chapter will generally not discuss the consequences the languages differences have on performance. That will be part of the general performance discussion in chapter 6.

5.1 Application vs. system programming

Java is an **application programming language**, while languages such as C or C++ are **system programming languages**. In essence this means that Java does not allow direct access to the underlying hardware.

The consequence is that Java is not suitable for writing low-level software such as hardware drivers or operating system kernels. It is intended for application programming only.

Java will therefore never completely replace C++. It should also therefore not be a surprise that Java virtual machines themselves are written in C++.

5.2 Object orientation

In C++ the use of object oriented code is optional. Users can ignore objects and write C-like applications without objects.

In Java the use of object oriented programming is mandatory. All variables and functions must be defined as part of a class.

On one hand this reduces flexibility during programming but on the other hand it helps to ensure code modularity.

5.3 Portability

Java source code is almost 100% portable between different compilers, and Java byte code is almost 100% portable between different platforms and different virtual machines. It is not exactly "write once, run anywhere" but it is the closest you get with any

programming language or environment that I am aware of and it is in fact *very* close to achieving it.

If you are developing for multiple platforms this high level of portability is an important benefit.

There are several language features that help to ensure portability:

- The sizes of Java data types are identical on all implementations. This is not the case with C++.
- Java uses only Unicode for characters. This removes code page problems and also eases adding internationalization support.
- The standard library is big enough to actually allow advanced programs being written without the use of external libraries that is not written in 100% pure Java.

If a program accesses native code (i.e. C or C++ code) then this native code must of course be manually ported to each target platform.

The result of this high portability is that applications very often can be compiled on multiple Java compilers and run using multiple Java virtual machines without changing anything. As long as the underlying computer does not change, switching Java compilers and virtual machine works even if the application uses JNI.

The effect of this is that Java developers often try out their application on several virtual machines and based on the result chooses the one that gives the highest performance.

In contrast, compiled C++ programs are not portable. C++ source code is somewhat portable, assuming that they do not use features outside the limited standard libraries, and that the compilers do not differ too much.

C++ portability often sounds better in theory than it is in practice. The C ANSI library is today fully adopted on all major C++ compilers but there are still several compilers that do not conform to the rest of the C++ standard library (including the STL containers) or even supports the full C++ language standard – or they support it with some variations. Furthermore, common functionality, such as multi-threading or graphical user interfaces, is not standardized with C++, making applications using such features much less portable.

The portability of Java relies on that assumption that the used virtual machines and standard libraries are fully standard complying. Fortunately, this is the case with most virtual machines. The Sun implementation is in practice the defining one. Third parties then license the Sun source code as the base of their own Java virtual machines. This makes the functionality of virtual machines very identical and portability among virtual machines a non-issue. The only exception to this is the static native compilers, since these works much differently from regular virtual machines. In practice this means that there can be some portability problems with some of those. I therefore recommend that you never rely on being able to use static compilation for your final release.

5.4 Dynamic data allocation and garbage collection

In C++ all dynamic data must be allocated and deallocated explicitly. If the programmer forgets to deallocate the data then there is a memory leak that might cause the system eventually to run out of memory. If the programmer on the other hand deallocates data too early and he then accesses it, the behavior will be undefined.

These problems can be solved to some extent with proper tools such as *Compuware/Numega Boundschecker*. In fact, using such a tool is highly recommended for any C++ developer. However, even in the presence of good tools, handling these problems is no trivial task.

Java does not support explicit deallocation of data. Instead, all data that is no longer referenced is automatically garbage collected preventing virtually all pointer allocation/deallocation related problems. (The single exception is discussed in section 5.4.2)

The performance consequences of using garbage collection will be discussed in chapter 6.

5.4.1 Garbage collection in C++

It can be argued that garbage collection is also possible in C++. This is correct but only to some extent.

If you build your own garbage collection routine in C++ then you can generally take one of two basic approaches:

- Allow the use of regular pointers and perform the garbage collection by using some kind of mark and sweep algorithm scanning the memory for data that looks like pointers. This type of C++ garbage collector may fail to recognize unused memory if you use the C++ memory in unexpected ways and may therefore not always work. One serious problem with this approach is that the garbage collector may not move objects at run-time, which would invalidate pointers. This significantly reduces the performance of the garbage collector since it makes it impossible to make generational garbage collection (see section 6.3.3). One free library using this approach is *Garbage Collector* by Hans J. Boehm. Using Garbage Collector is not easy, because it is quite complicated to set up.
- Encapsulate pointer operations and add a reference count to each object. When the reference count becomes zero, the object is garbage collected. This is the clean way of doing it, but is significantly slower than what is possible when using a mark and sweep algorithm. This algorithm fails if you use normal pointers instead of the encapsulating objects. A simple implementation of this approach is described in [Photon]. This is also the approach used within the Microsoft Component Object Model (COM).

For both types of garbage collection the system may fail, because third party libraries not always may observe the required restrictions. In Java these problems are nonexistent,

because all libraries support garbage collection automatically, and because garbage collection is guaranteed to work in all cases.

5.4.2 Memory leaks in Java

In one sense of the word Java does have memory leaks but it does not have it in the traditional sense.

In C++ you have to remember to call `delete` on every object that you have created with `new`. If not, the object will never be deleted and you have a memory leak.

In Java you just have to be sure to delete all references to an object in order for it to be deallocation. This can happen if you have some references to objects that you use throughout the application that contain some references to objects that no longer is in use.

The situation is much less severe in Java (compared to C++) because if you delete a reference to an object then the references that the object itself contains are also automatically deleted. This does not happen in C++, where it is vital that every single object is deleted exactly once.

To summarize, Java can in one sense be said to contain memory leaks, but they are much less severe and much rarer in practice than those in C++. Java profiling tools, such as the popular *OptimizeIt* can detect this kind memory leak.

5.5 Array bounds checking

C++ programs can access data outside the bounds of arrays. This may cause very hard-to-find bugs, but may also be used by the resourceful programmer to do specialized access to the system memory.

The use of tools such a NuMega Boundschecker can at the expense of extra overhead usually detect when this happens.

In Java an exception is always thrown when an array is accessed out of bounds, meaning that such errors always are caught immediately.

5.6 References and pointers

In C++, the programmer has to handle pointers to memory addresses explicitly. Since pointers may point to illegal memory addresses or even to addresses containing data of other types, many hard-to-find errors might occur. The use of pointers often causes confusion to beginning and intermediate C++ programmers, because of the many issues involved. For instance, in C++ all variables can be passed both by-value, by-reference, and as a combination caused by passing an address as by-value.

Java solves this problem by replacing pointers with **references**. A reference is internally the same as pointer in C++, but with added security:

- A reference cannot point to deallocated memory. To achieve this, Java does not support explicit allocation of objects on the stack – this is only possible for primitive types.
- A reference cannot point to an object of an incorrect type.
- Memory leaks in the traditional sense cannot occur, since an object is deallocated automatically if no references point to it any more.

Java distinguishes between **primitive types** and **object types**. Primitive types include data that can be stored in a single memory address. These are always passed by-value and are stored on the stack. Object types are always accessed via a reference and are always passed by-reference. Object types cannot be stored explicitly on the stack (virtual machines may, however, do this behind the scenes) but only on the heap. References are themselves primitive types.

5.7 Type casting

In C++, the user can freely change the type of a pointer without any run-time check to verify that the type cast is valid. This is useful in some special cases where data stored in one way is to be viewed in another. However, this is also the source of many errors.

In Java, type casting is always controlled. One can only type cast object references that are related by inheritance. If the type cast is a down cast (i.e. to a derived class) then a type check occurs at run-time to detect whether the type cast was legal. No type check is needed when up casting.

As discussed in section 5.19, Java does not support templates and must rely on common base classes in generic containers. The result of this is that the amount of explicit type casts needed in Java usually is far higher in that in equivalent C++ programs.

5.8 The standard library

C++ comes with a decent standard library. This consists of several components, including the STL (Standard template library) and a library extremely similar to the C standard library.

These libraries are general limited in functionality. Essential features such a multi-threading, networking and GUIs is not supported.

Because the C++ standard library includes a variation of the C standard library many idiosyncrasies occur within the library.

One prominent example is string support. The manipulation of strings is important to many applications, so this should be made as easy as possible. The C standard library does have support for strings, but they are very awkward to use and require manual allocation of data each time. The C++ standard library solves this by providing a dedicated string class, but unfortunately most 3rd party libraries do not support these,

thereby very often forcing the programmer to fall back on the traditional and awkward C strings.

All in all these issues cause the C++ standard library to be quite awkward to use.

Unlike C++, Java did not have to be compatible with a much older language and was able to do the library design more cleanly from the start.

Java supplies a lot of advanced functionality in its standard libraries relieving the programmer from much work. To mention just a some of them, the programmer gets (this includes features from the optional packages): strings, networking, math, data/time manipulation, multithreading, URLs, sound, a GUI system, graphics, various standard container types and algorithms, remote method invocation (a kind of RPC support), database access, CORBA, cryptography, speech, modem support, server side programming, e-mail, 3D graphics with hardware acceleration, communication port support, authentication, multimedia, a visual component architecture, and advanced imaging.

Furthermore, and not to be underestimated, this rather advanced set of features is used as the common denominator of many 3rd party libraries, which thereby often have better interoperability than C++ libraries.

5.9 Prototypes and headers

In C++ a programmer often has to place prototypes for functions or global variables in header files and place the definition in regular source code files. In practice maintaining these headers are tedious to do.

Java contains (unlike C++) a multiple pass parser/compiler so there is no need to use prototypes, header files or even to consider ordering of variables and methods other than for readability reasons. This increases productivity significantly.

5.10 Documentation generation

In C++, documentation is usually written separately from the code and is usually hard to keep up to date.

In Java, it has been convention from the beginning to use document generation tools. Such a tool reads the source code and turns it into html documentation. It automatically extracts class names, parameter types and names and generates pages with hyperlinks to make the documentation easy to navigate. The user can insert special comment tags that are used to generate the actual documentation content. These tags also allow Java development environment to provide instant documentation to the user about symbols in the source code. We will discuss development environments more in section 5.22.

Such documentation generation tools also exist for C++, but they are not widely used. With Java there is a strong advantage here because users can read the API documentation the same way no matter who made the library. It also makes it easier to maintain the documentation since it is stored just beside the source code it documents.

5.11 Language simplicity

Java is a simpler language than C++. Not that it has less features, but that it has less complexity. The main advantage of this is that it is easier to learn and that less time is used during programming to think about language details.

Personally, even after having programmed C++ for more than 7 years, I regularly learn new details about the language. C++ is a big and complex language. Much of this complexity comes from two sources:

- The legacy from C. Originally C++ was designed to be fully backwards compatible with C. Although C++ has evolved on its own and is no longer fully backwards compatible the legacy remains.
- The wish to allow the programmer to access to low level constructs in order to fine tune performance and access hardware directly.

Java, on the other hand, has no legacy and has chosen to remove many low-level details. This has been done in order to be more portable but also to increase programmer productivity by bringing programming to a higher abstraction level. The price is, of course, that Java cannot be finely tuned for performance on a low-level, but the programmer must rely on the optimizing compilers to do the job. This is compensated by the fact that Java is a simpler language and this makes the job of making a good optimizing compiler easier.

Below is a list of some of the strange details in C++ that programmers must take care of:

- Strange syntax when using pointers to functions as arguments to functions. Look in any C book and notice that the authors usually comment on the messiness of this syntax.
- Why use semicolons after classes and structs? Java does not need them.
- The need for reference variables. A lot of programmers get confused when they see statements such as: `int i; int j& = i;` (which in fact causes aliasing). Reference variables are included because this allows functions to pass parameters by-reference instead of by-value.
- The syntax for defining operator overloading is quite hard to remember. Especially the postfix version of the `--` and `++` operators contain a strange syntax as they need a dummy `int` parameter, e.g.: `X& X::operator--(X&, int);`
- The need for initializers in constructors for efficiency. Java just requires them to be in the beginning of the function block.
- You need to declare static member data twice. For instance, if I have a static member variable then it must be declared in the header file and defined in a source file.
- The need to declare private members in the header (i.e. the public interface) of a class.

- The need for the `friend` keyword. If C++ supported package (i.e. namespace) access modifiers this would not have been needed.
- The need to use `inline` and `register` keywords.
- That you have to know about all kinds of details such as declarations `__cdecl`, `__pascal`, `extern "C"`, far pointers and other concepts such as dll imports/exports resources, linking options, etc.
- Various `#pragma` declarations.
- The need to avoid that certain header files are included more than once and the problems with cyclic dependencies between header files.
- The importance of knowing that the order of execution of the initializers in a constructor depends on the ordering of the fields in the struct or class.

5.12 Operator overloading

C++ supports the use of operator overloading. Java does not.

The rationale for this Java limitation is that operator overloading often reduces code readability.

Personally, I am not sure I agree. Clever use of operator overloading of relevant classes, say vectors, can significantly increase readability.

5.13 Unsigned integral data types

In C++ all the primitive integral data types comes in both signed and unsigned variants.

Probably due to the many hard-to-find bugs related to the mixing of unsigned and signed types Java has chosen to avoid unsigned types and therefore only support signed data types. The exception is the `char` data type which is a 16-bit unsigned integral (recall that Java uses Unicode for characters) and which do not exist in a signed version.

5.14 Constant parameter types

C++ supports the passing of constant parameters and the declaration of constants, by the use of the `const` keyword. If a reference or a pointer to a constant is passed then the value of the object cannot be changed. This also means that all manipulations made on the object must be constant. Furthermore, a method can be declared constant, meaning that it is not allowed to change the `this` object. The compiler verifies that the constant declarations are respected.

This feature is a powerful one, since it can detect many errors at run-time where some code my mistake change something that it should not.

Similarly Java supports a `final` keyword. `final` seems to be identical with `const` but are not. A `final` parameter or reference just means that the *reference itself* is constant. This means that a final reference to an object still allows the object to change, but it does not allow the reference to point to another object.

Almost needless to say the `final` mechanism is much less useful than the C++ `const` mechanism and detects fewer errors at compile time. This is a clear disadvantage of Java.

5.15 Packages and namespaces

In C there are generally a lot of problems with name clashing between functions. To circumvent this, library writers often prefix function names with some identifier to make them stand out from the rest. But this does not solve the problem completely as authors sometimes use duplicate identifiers.

C++ improves this scheme somewhat by providing namespaces but this does not solve the problem completely.

A related problem is the clashing of source file names.

Java solves these problems by providing a standard package scheme. A package is basically the same as a namespace in C++, except that packages are organized hierarchically and the naming is based on Internet domain names but with reverse ordering. For instance, I own the domain *marnet.dk*, so when I publish code; this code is placed in the package *dk.marnet*, or a branch under it.

The source code must also be organized according to this scheme, meaning that code for the package *dk.marnet*, but be placed in the *dk/marnet/* subdirectory of the Java class root.

This scheme completely solves name clashing issues in Java. This is a clear advantage of Java.

5.16 Multiple class inheritance

C++ supports multiple class inheritance. Java only supports single class inheritance and multiple interface inheritance. An interface in Java is basically a class with no implementation, i.e. no method definitions and no fields.

Since multiple class inheritance in some cases is useful this is a clear advantage of C++.

5.17 Type definitions and Enumerations

In C and C++ you can give an already defined type a new name. This can be done either with `typedefs` or `#defines`. The purpose of this is usually either to:

- Make the code easier to port to other C/C++ compilers. For instance, OpenGL defined a type `GLuint`, which is supposed to be an unsigned 32-bit value no matter the compiler.
- Make new names that help the user to understand the purpose of values stored in that type. This is also essentially what enumerations in C and C++ do.

In Java it is not possible to rename types or to define enumerations. To make an enumeration in Java, you either have to define a set of constant integer variables or use the Enum design pattern [Meurrens].

The first type of use presented above is not needed in Java, because the language already is made to be portable between compilers, so the programmer does not have to care about it.

The second use, however, is still relevant in Java. The rationale for not including it in Java is that these name changes often causes more confusion than they are an advantage. Personally, I do not agree with this rationale and consider this limit a drawback of Java, but I acknowledge that opinions differ in this matter.

5.18 Preprocessing

Before C++ source code is passed through the actual C++ compiler it is first passed through a preprocessor. This preprocessing step allows, among other things, the use of macros and conditional compilation. Java supports neither.

In fact Java does not have a preprocessor at all, although some third party Java preprocessors do exist.

5.18.1 Conditional compilation

Conditional compilation is mostly useful for two purposes:

- To make code portable among different platforms. “Portable” C/C++ code often makes heavy use of this.
- To distinguish between different kinds of builds. For instance, one can include extra code in debug builds that is not included in release builds.

Because of the high portability of Java code, the first use is not relevant for Java. However, the second use is still relevant.

The rationale for avoiding conditional compilation is that debug checking never should be removed and that bug checking should remain even when the application is deployed to the end-user. This allows the team to get better bug reports from the users and make it easier to perform field debugging.

I disagree with this. If the code is performance critical, it is not acceptable to keep debug checks in final releases. Instead of enforcing its own views, a language should give this choice to the programmer.

Sun has acknowledged the need to distinguish between debug and release builds and with JDK 1.4 support for assertions has been included in Java. Assertions can be removed at run-time when one trusts that the code is bug free although it is still present in the byte code in order to make it easier to do field debugging. [Assert]

5.18.2 Macros

Likewise, definition of macros is not possible in Java.

This has been done for the sake of code readability. Although one is tempted to think that there is cases where the use of macros can be justified, it is probably true that this feature is much misused in C and C++ in the name of performance, even when optimizing compilers or templates could do just as well a job. So for the sake of code readability and maintenance I agree with this choice in Java and consider it an advantage of Java.

5.19 Templates and generics

Templates, also known as parameterized classes/functions, are a powerful feature in C++ that allows the use of strongly type checked polymorphy in classes/functions while being very efficient. If code of similar efficiency was to be made in C++ it would either require the use of macros or of a lot of code redundancy to write specialized versions for each function/class.

Java supports neither templates nor macros. This generally leaves only three options for creating polymorphic functions or classes:

- The use of a common base class. This is the solution used by the Java standard library. This solution is quite elegant except for the fact that the user of the polymorphic method has to make a lot of explicit down-casts from the base class to the derived class. Such down-casts have traditionally been expensive in Java but with JDK 1.3 and later they can sometimes be eliminated during optimization. A problem with this way of implementing polymorphy is it does not apply to primitive types.
- To write specialized versions of the classes. E.g. one would write a list of `ints`, a list of `doubles`, a list of `float`, etc. The disadvantage of this clearly is that the code becomes harder to maintain and will take longer to write in the first.
- The use of third party Java preprocessors. These can be used to define parameterized macros that in effect work like templates. This is how many generic algorithms were implemented in C++ before the introduction of templates. The disadvantage of this is that the code becomes less readable and somewhat messier to debug.

Sun is aware of this problem and has planned the support of Generics for a later version of Java [JSR14]. A prototype implementation has already been completed. Generics will probably be included with JDK 1.5 but there is no guarantee of this. Generics are similar to templates in syntax and the main purposes are to:

- Eliminate the need for the explicit down-casts that would be required without generics.
- Ensure consistency of types. For instance, without generics a list container may contain elements of differing types (derived from the generic class `Object`). With generics (or templates) it is ensure that the list only contains elements of the specified type.

Generics will not allow polymorphy between primitive types and object types.

5.20 Dynamic class loading

A strong feature of the Java language and platform is the support for dynamically loaded classes. This essentially means that code can be added and (with a few tricks) replaced at run-time.

C++ code on the other hand is completely static after it has been created. To obtain dynamic functionality in C++ one usually makes use of some scripting language.

It should be noted that not all Java static compilers support dynamic class loader. I know that it is supported in both Excelsior Jet and gcj.

5.21 Reflection

Dynamic class loading is taken a step further with the Java reflection mechanism. The reflection mechanism allows Java programs to query information about unknown classes to learn about what methods and fields they have and then later call it. This functionality is a central part of the Java component architecture, JavaBeans.

For instance, it allows components/classes to be plugged into a program and then made available to the user. This is useful in development environments where the user visually is able to set attributes of objects that the environment previously knew nothing about.

The C++ RTTI supports similar functionality but in a far more limited way.

5.22 Debugging and IDEs

The language itself is not the only factor that determines how productive a programmer is with it. Another factor is how good the available tools are for the language.

For C++ the absolutely best integrated development environment (IDE) is *Microsoft Visual Studio*^{*}, combined with *VisualAssist* from *Whole Tomato Software* and *NuMega*

^{*} For the purpose of this evaluation I have used Visual Studio 6.0 SP5 since I did not have experience nor access to Visual Studio .NET

Boundschecker. This environment provides the programmer with an excellent productivity.

For code editing this environment is to my knowledge unsurpassed by environments provided by any language on any platform. Some of the included features are variable, function and method tab-completion and visual indication of most syntax errors during editing (including the detection of unknown variables or variables not in scope).

This environment also supports pluggable compilers which allow it to be used to develop applications for both the PC and consoles.

However, for debugging, neither that nor any other C++ environment I know is perfect. Although the visual debugging tools are advanced they still have problems with in some cases, including allowing the user to:

- Visually navigate the data in dynamically allocated arrays. They can only be viewed by viewing specific fields in the array via a watch or by browsing the raw memory.
- Visually navigate data that are of a type derived from the type being navigated.
- Visually navigate complex templates, including those in the STL.

It is not easy to determine the best Java development environment at the moment, but good candidates are *Eclipse* (previously known as *IBM WebSphere Studio*) or *IntelliJ IDEA*.

These are both excellent development environments but do not provide the editing productivity of Visual Studio, although they are not far from it. They do, however, support things not available for C++ environments such as extensive tool support for refactoring [Fowler] which more than compensates for the lack of editing features. I have in fact been unable to find any tools at all for C++ that supports refactoring[†].

And for debugging, the Java IDEs are very powerful. This is likely only possible because

- Debugging support has been integrated into the Java virtual machine from the beginning.
- Java is a simpler language.

There are no limitations with the debugging features in Java like there is in C++. Combined with the protected environment this helps to detect bugs early in the development process and thereby increase productivity.

It should be noted that the advanced editing features of Visual Studio easily can be applied to Java, but this has yet to be done. This will probably happen before the visual debugging problems with C++ are fixed, if that even is possible.

[†] Although no refactoring tools exist for C++, there do exist one that has limited refactoring support for C. This is called Xrefactory and is an Emacs plug-in.

To summarize, the best IDEs of C++ and Java are of approximately the same quality today. The best C++ IDE is a little better at editing, while the best Java IDEs are superior when it comes to debugging and support for refactoring.

I think it is an important point that better tools are easier to make when a language is simple. We already have evidence of this by the fact that several refactoring tools exist for Java while none yet exist for C++ - although the refactoring principles can be applied to both languages. It is therefore a good guess that we will see more improvements in Java tools over the coming years than in C++ tools.

5.23 Summery

The major differences between Java and C++ that affects productivity are summarized below. We start by listing those features that is advantages of Java and then list those that are advantages of C++.

The advantages of the Java language:

- Java source code and byte code is significantly more portable than C++ code.
- Java applications runs in a completely protected environment, uses references instead of pointers, and uses garbage collection instead of explicit deallocation. This prevents virtually all pointer related errors from running unnoticed. This includes errors such as illegal memory access, type casting, arrays accessed out of bounds, traditional memory leaks, dangling pointers, access to deallocated memory and more.
- There is no need to use header files or prototypes in Java.
- The standard library included with Java is superior to that of C / C++.
- The Java syntax is simpler than that in C++ with fewer details including the elimination of many unsigned types.
- Java support dynamic addition and replacement of code at run-time and run-time inspection of such code.
- Java is easier to debug.
- Java supports packages, which removes all name clashing problems.

The disadvantages of the Java language:

- Java does not support templates and Java does not support polymorphy between primitive types and object types.
- Java does not support typedefs or enumerations.
- Java does not support constant objects or constant object parameters.
- Java does not support multiple inheritance of classes, but only single inheritance of classes and multiple inheritance of interfaces.

- Java does not support operator overloading.
- The editing tools C++ are currently a little better than those for Java.

Furthermore, Java is only suited for application development, not low-level system development such as development of hardware drivers. Low-level access can still be gained, though, with the use of JNI.

Most of the issues with C++ are due to its legacy from C. Given that C++ at its creation had to be backwards compatible with C, the design of C++ could probably not have been much better.

Many of the differences between the two languages have the result of increasing productivity. How much these productivity gains sum up to is discussed in chapter 8.

6 Java Performance

In this chapter we will discuss the performance of Java – especially as seen in relation to the performance of C++. This will be done both theoretically but also based on both the benchmark results in chapter 7 and my personal experience.

This chapter uses the results of chapter 7 a lot, but this chapter is presented first because proper writing and running of benchmarks requires good knowledge of Java performance in general.

6.1 The origin of the Java myth

Java has a strong reputation for being a slow language that cannot be seriously considered for real applications. It has not gotten this reputation by coincidence.

If you are a C++ programmer then you were probably very tired of all those Java evangelists back in 1995 that claimed that Java was superior in every regard and that 100% pure Java was the best thing there was. They even sometimes claimed that it ran practically as fast as C++, and that any measured differences were insignificant. Chances are that you tried it back then, saw how awfully slow it ran, and then dismissed it as a web development toy and decided that the Java evangelists either was liars or fools.

Fortunately, most the hype surrounding Java has since then died out and the compilers and virtual machines has improved significantly in meantime.

I heard the same hyped arguments back then and originally dispelled the use of Java as anything but a web applet language, because of the many promises that was clearly not true. I did not return to Java until late in 1998.

And if you even today do a quick web search on "Java games" you get nothing but small low quality applets that is not very impressive. These games just help to maintain the reputation of Java as being a language that cannot be used for real games.

However, all this does not mean that Java is useless for professional gaming purposes. As we will see in this chapter Java is in fact not as slow as its reputation claims; it all depends on how you use it.

6.2 When is Java slow?

It is in fact true that many Java applications to this day still are very slow. This has mainly to do with the fact that they use old versions of Java. What version one uses are very important for performance.

The performance difference between the old versions of Java (see section 4.6) and C++ is approximately (this is very rough numbers, determined by running a few simple benchmarks that is not documented in this report and will vary wildly depending on the actual application):

JDK/JRE version	Factors slower than C++ (approx.)
v1.0	20-40
v1.1	10-20
v1.2	1-15

That Java is a factor of x slower than C++, means that Java uses x times more time than C++ to perform the same computation.

As can be seen from the table above, old versions of Java is in fact *very* slow.

You have probably experienced this first hand by playing some small games on some web page with your browser. Unless users explicitly install a modern Java plug-in browsers use an old version of the Java technology that is very slow compared to what is possible today. Even modern browsers such as Netscape 6 and Microsoft Internet Explorer 6 still only comes with a Java virtual machine at version 1.1 unless the user explicitly installs a Java plug-in. This explains why web applets run so slow.

You may also have run your own tests using Microsoft J++. Judging Java based on that is not fair, since that is one of the slowest versions you can get of Java. It is no secret that Microsoft is no supporter of Java – especially since the announcement and release of C#.

You may have tried Java applications outside your browser and found them slow too. This is probably because it was a GUI application and the GUI system in Java *is* noticeable slower than native Windows or XWindows. It has gotten faster lately but it is still not fast enough. Fortunately, since games usually do not make heavy use of the built-in Java Windows system, this is not really a problem for games development.

6.2.1 Problems in the first Java virtual machines

As can be seen from the tables comparing the speed of Java and C++, the speed of Java applications has increased significantly with each major version.

Originally Java virtual machines were simple byte code interpreters that executed one command after another. Needless to say, this was very slow. With JDK 1.1 Just-in-Time (JIT) compilation was introduced to the virtual machines. These compiled each class to machine code before the first time it was executed. Seemingly this should cause Java programs to become as fast as any statically compiled programs.

Unfortunately, this was not the case, since traditional compiler technology relies heavily on inlining to gain their high performance. This was not easy in Java because each class in byte code form is completely independent of other classes. This flexibility is enhanced even more by the possibility of dynamically loaded classes. Since each class in JDK 1.1 and JDK 1.2 was compiled to machine code one at the time the consequence was that it was impossible to inline method across class boundaries or even within classes themselves if those methods were not private.

Two approaches have been used to solve this problem.

- Assume that all classes immediately present are static and compile the classes as was they written in any other statically compiled language. This is the approach used by the Java static compilers.
- Allow the compiled machine code to be recompiled at run-time if the involved set of classes changes. This is the approach use by Hotspot virtual machines (JDK 1.3 and later). This approach will be discussed in more detail in section 6.3.2.

6.2.2 Other issues that slow down Java

Other specific problems that traditionally have slowed down applications written in the Java languages are:

- All down-casts must be type checked at run-time. Since Java does not have templates, the number of explicit type casts is generally very high.
- At array accesses are checked at run-time to verify that they occur within bounds of the array.
- Objects cannot be stored explicitly on the stack, but must be allocated on the heap, causing many more objects to be allocated.

6.3 When is Java fast?

Modern Java applications are sometimes slow and sometimes not. How fast they are depends on several factors which we will discuss here.

One very important factor is the amount of tweaking that has been performed on the Java program. According to the conclusions of chapter 7, a non-tweaked program will be several factors slower than an equivalent C++ program. In my tests it was 2.5 to 4 times slower. The definitions of tweaked and untweaked code are presented in section 7.5.

However, if the Java program is tweaked (and the C++ program is tweaked too for fairness in comparison) this difference is reduced significantly. Many highly tweaked programs have in fact been shown to be able to run faster in Java than in C++. This is

also the case with one of the benchmarks in chapter 7 (The *Tweaked Life* application). Judging from the various benchmarks I would say that tweaked Java programs on the average runs 20-50% slower than tweaked C++ program (= a factor of 1.2 to 1.5 slower). The results vary a lot from application to application, so the conclusions are not clear. In some cases tweaked Java code is up to three times slower than C++ and in some case it is double the speed of C++ code – but in general the tendency seems to be that it is a little slower.

JDK/JRE version	Factors slower than C++ (approx.)
v1.3	0.7-4
v1.4	0.5-3 (typically 1.2-1.5)

The only major drawback is that it is often quite hard to tweak programs in Java to achieve this performance. If such tweaking is done the productivity falls and the main point of using Java in the first place will be lost. So naturally this tweaking should only be done in parts of the program that has been shown to be very computational intensive.

6.3.1 The amount of code that needs to be tweaked

When a Java programmer has written some code that has been written with readability and maintenance in mind, this code is classified as untweaked. When people talks about the high productivity programmers have with Java they usually assume this kind of programming.

As mentioned above I estimate that such programs are 2.5 to 4 times slower than untweaked C++ on the average. This is not good, but fortunately this not mean one has to tweak all of program code. When designing a program the programmer should try to determine which areas will have the highest throughput and then isolate those parts from the rest of the program so they can be easily tweaked for performance. This technique should be augmented with the use of a profiler after the program has been written.

In many programs as little as 10% of the code is run 90% of the time, so if the programmer tweaks those 10% the program will run almost as fast as had all of the program been tweaked.

For instance assuming the worst case that the tweaked part of the program is 50% slower than C++ and the rest is 4 times slower, the end result will be that the program is $(0.1 * 4 + 0.9 * 1.5 = 1.75)$ 75% slower – in the worst case.

Furthermore, if the program uses 3D hardware, as most contemporary games do, most of the execution time is spent in 3D hardware, which means that the performance difference between Java and C++ becomes even less. This is the case with the 3D Demo benchmark in section 7.10.

6.3.2 Hotspot technology

One of the major innovations in Java is the development of Hotspot technology (JDK 1.3 and later), a kind of adaptive optimizer technology. [Hotspot]

The idea behind this technology is that the application is analyzed at run-time and optimizations are performed based on these results.

The first many times some piece of code is run it is run interpreted, and when the system has gathered enough information about the run-time behavior of the application it aggressively compiles the sections of the code that are run most; the hotspots. This is how the technology got its name.

The idea is that it is better to use much time compiling the code that is run often rather than performing inferior compilation to it all as is usually done by JIT compilers.

This strategy introduces an initial overhead to the application as it starts up (the so-called **warm up period**) while it analyzes run-time behavior and runs the optimizing compiler.

The optimizations that can be performed on the code can be much more aggressive than the optimizations performed with traditional static compilers, such as C++ compilers, because the compilation can be undone and redone at any time. This means that if the general run-time behavior changes then the machine code can, and probably will, also change.

In general, the Hotspot virtual machine first attempts to in-line as much code as possible. Then the full range of traditional optimization techniques is applied to inlined code.

Special optimizations for Java are:

- Array bounds check elimination.
- Elimination of checked type casts.
- Non-final methods (i.e. virtual methods) can be turned non-virtual if they are not derived or never used polymorphically.
- Variable escape analysis that allows objects to be stored on the stack instead of on the heap. Escape analysis is not efficient with traditional compiler because they have to be more conservative. This optimization is according to [Hutcherson] not yet implemented in the Sun JVMs (JDK 1.4), but is planned for later releases.

Of optimizations, which are unique to Hotspot technology, are:

- The compiler can, based on run-time observations about the behavior of the code, provide accurate hints to the processor about what choice will happen in if-statements, thereby improving processor pipe-line utilization.
- Compilation occurs at run-time meaning that it can take into account performance characteristics of the specific architecture it runs on. For instance, an Intel Pentium 2 has and an Intel Pentium 3 has different performance characteristics. To fully

utilize the pipelines one must know the precise architecture. C++ compilers cannot know this.

Finally it should be noted that C++ compilers in principle also can choose to adopt adaptive optimization technology just like Java. It is, however, unlikely that this will happen for a long time, if ever. This is mainly because C++ is a more complex language, so making such a compiler for C++ is a more complex task than it is for Java.

Hotspot provides some unique possibilities for optimization that static compilers do not. This could lead us to believe that Java might quickly become faster than C++. Whether this will happen, only the future will show, since there are also things that slow Java down compared with C++ (see section 6.2.2).

6.3.3 Allocation and garbage collection

Another thing that often is said to be slow about Java is the garbage collector. This also used to be the true. But as can be seen from the benchmarks in chapter 7 the allocation and deallocation of a heap object is much faster in Java than it is in C++.

However, *heap* allocation in Java is still much slower than *stack* allocation in C++. This is currently a problem, because Java does not allow explicit allocation of objects on the stack. The current Hotspot virtual machines does not even detect whether specific objects are short temporary ones that can be placed in registers or on the stack.

The consequence is that an essential part of the current tweaking process for Java programs is to eliminate all heap allocations from performance critical code.

Allocation and deallocation in Java can be done faster than in C++ mainly, because Java does not allow direct access to heap memory. This means that the garbage collector at run-time is able to reorganize and fully defragment the heap; something which is impossible in C++.

The garbage collector used in modern versions of Java is a so-called **generational garbage collector**. In the implementation that is used with current Hotspot virtual machines the heap is divided into two sections; one for each generation. The first, often called the **nursery**, is where recently allocated objects are stored. The second is where older objects are.

When a new object is allocated is it allocated within the nursery. When the nursery is full this area is garbage collected and any surviving objects are moved to the next generation. The advantage of this strategy is that the nursery never becomes fragmented. Allocation can happen basically by returning the current heap pointer and then increasing the value of the heap pointer with the size of the allocated object. Allocation can be done in constant time, except in the special case where the nursery is full.

The nursery is garbage collected by marking all reachable objects and then moving live ones to the next generation. Except for the fact that many allocations causes the garbage collector to run more often, short lived objects are virtually free since they do not take time during the scan for live objects. Analysis has shown that most objects in Java programs are short lived, meaning that the number of objects that needs to be copied to the next generation is expected to be small.

The section of the heap used for old objects uses a classic mark-compact garbage collector. Fortunately the area of the heap used for old objects will only need to be garbage collected rarely.

I often hear people say that they dislike the idea of a garbage collector that may kick in at any time and pause the system for a few seconds. That would admittedly be a bad thing in many cases, especially in a high-performance real-time game.

Fortunately, there is only a very low risk of this actually happening unless the program allocates large quantities of objects. Garbage collection may be implemented in several ways. If it is asynchronous then garbage collection occurs in a low priority thread. This prevents the long pauses from occurring. However, if the garbage collector is synchronous it may cause pauses of a few seconds. Sun's JVM 1.4 uses an asynchronous garbage collector. This does not remove pauses completely but makes them much shorter. However, even short pauses are bad in a real-time application since even minor variations in the frame rate will be immediately noticeable by the user.

Fortunately, the garbage collector is only working when the heap is full (or almost full). If the program is tweaked to avoid heap allocation in time critical code the risk of this happening is virtually zero. Furthermore, if one invokes the garbage collector explicitly at times when it doesn't matter, such as when a new level has been loaded, the risk can be reduced even more. In essence it is often easy to write an application such that the garbage collector does not pose a problem.

In contrary to popular belief, garbage collection is not inherently slower than manual deallocation. It is actually often faster since you reduce the overhead of deallocating objects. Remember; when you use a sweeping garbage collector (which is by far the most common) only live objects take time to process during garbage collection. Objects that need to be deallocated are free.

6.3.4 The speed of Java in the future

The first beta version of the Java platform was released to the public in 1995, so the history of the system is fairly short compared to that of C and C++. It is therefore no surprise that Java was very slow in the beginning.

With almost every release we have seen vast improvements to speed of Java. Just within the last 2 years or so we have seen a 10-20% increase in speed (from JDK 1.3.0 to JDK 1.4.0). And there are still many potential optimizations and improvements that still can be implemented. It is therefore reasonable to assume that this improvement in speed will continue with future versions of Java.

At the same time C++ compiler technology is only seeing minor improvements meaning that Java rapidly is gaining on C++ in performance. I think it is reasonable to assume that they within a couple of years will become about the same speed – if not Java even overtakes C++.

When evaluating a language or system, it is in general dangerous to rely on guesses about future performance, and I will therefore only base the conclusions in this report on the current state of Java; not guesses about the future. If you begin a project today in Java

and Java is faster before your game is to be released that is only fine, but you should not rely on it.

6.3.5 Static native code compilers

Using Hotspot virtual machines is not the only way to improve the performance of traditional Java virtual machines. Another popular approach is to use static compilers. A static compiler is a traditional compiler like those for C or Fortran that converts source code into an executable file. To these compilers, Java is just yet another language that you can program in. A consequence of this is that programs compiled with static code compilers are no longer portable.

When running programs compiled with a static native code compiler the program is not interpreted, it is not compiled at run-time and does not need a Java run-time environment (JRE), although some run-time dynamic link libraries may be needed[‡]. Static code compilers create regular executable (*.exe) files and even dynamic link libraries (*.dll/*.so).

Several Java static compilers exist out there. These include:

- Excelsior Jet.
- gcj (A front-end to gcc and officially part of gcc. Limited AWT support. No Swing support. In addition to JNI it has its own native interface called CNI which is faster than JNI. If one uses CNI then the code is locked to gcj. Supported on all platforms that has a gcc port, i.e. a lot.)
- NaturalBridge BulletTrain
- Instantations JOVE
- Wind River Systems Diab FastJ
- Tower Technology TowerJ (does not support AWT or Swing)
- Metroworks Codewarrior for Java

Some Java development tools try to give the customer/user the impression that they contain a static native code compiler although they do not. This includes Webgain Visual Café (formerly Symantec Visual Café) and Microsoft J++. These products only wrap the Java byte code into *.exe files. They do little or no actual compilation to machine code.

Most of the mentioned Java static code compilers support everything from Java including all the standard libraries, AWT, Swing and all the standard extensions (such as Java3D). You can even import byte code supplied by third party developers and compile them directly to native code.

[‡] Note that some static native code compilers require the final executable to use Sun's Java DLLs. In this case, due to JRE license restrictions, the full JRE must be distributed with your application even if it is not used.

Static native compilation does not come without a price. By using static native compilation the following Java features are lost:

- No platform independence of the compiled code. You can, however, use a different compiler for each platform. The source does not need to be changed.
- No support for applets. Static compilers cannot be used with Java applets - only applications. If you think about it, this doesn't make much sense anyway, since applets are supposed to be put onto a web page and be able to run on many different types of machines.

Also, some of the static compilers do not support dynamic class loading, so if you need that feature you should make sure it is supported in the compiler you choose.

An important question to ask is whether the code they produce is faster than Hotspot virtual machines. The answer is that sometimes they do and sometimes they do not. I recommend that you try it out for your specific application to see if any performance is gained.

Sometimes static code compiler gives as much as a 30-40% speedup and sometimes they even slow code down. But if they give your application any extra speed they are certainly worth the effort to try out.

However, my personal guess is that in the long run Hotspot virtual machines are probably going to win out, because there still is a lot of potential optimizations that is not yet implemented using that technology.

As discussed in section 5.3, Java static compilers are often not fully standard compliant. In practice this means that one should not rely on static compilers for your specific product. If a static compiler can be used with success and it improves performance then this is fine, but one should not rely on it working or even improving performance over Hotspot virtual machines.

Some people prefer static compilers for deployment reasons. The deployment of Java programs are discussed in more detail in chapter 8.

6.3.6 Dirty Java

Often one has an application where certain parts of the program must run at a very high performance while it is less important with the rest. For such a program it can be risky to rely on Java for the performance critical sections of the code since to may turn out to be slow in that specific case.

This is probably the case if you are building a brand new high-profile first person shooter. Since Java sometimes, even when tweaked, is slower than C++ one may think that it is best just to ditch Java and write it all in C++. However, doing so would mean that one loses the productivity benefits on Java must also loses performance compared to those cases where Java is faster than C++.

One approach that gets the best of both worlds is to mix Java with C++ within a single application. This approach is usually termed **dirty Java**, as seen in contrast to Pure Java.

[Kreimeier] discusses dirty Java this from a game development point of view. This article is highly recommended reading.

Continuing the example from section 6.3.1 where we saw that a pure Java program in the worst case would be about 75% slower than C++. If the code that is run the most is written in C++ and the remaining 90% is written in Java and untweaked then the result is $(0.1 * 4 + 0.9 * 1 = 1.3)$ that the application is 30% slower (in the worst case) than had it been written fully in C++.

The mix of Java and C++ is done through what is called the Java Native Interface (JNI). JNI allows Java to interact with C/C++ in a controlled way. This maintains the protective nature of Java while allowing the Java programmer to access system specific features or third party libraries available only in C/C++. Accessing C/C++ functions from Java does, not surprisingly, have an overhead head compared to making the call directly from C/C++ itself. The performance of JNI is discussed in more detail in section 6.7.

6.4 Why is this Java program slow?

I often speak with people that has tried one or more Java applications and found them slow. In virtually all the cases it due to one or more of the following reasons:

- The application is poorly written or the bottle necks have not been tweaked. This may give a slow down of has much as a factor of four or more.
- The application uses an old version of the Java technology such a being run in the built-in virtual machine in popular browsers. That is very slow.
- The program uses AWT or Swing. (Swing is supposedly up to 40% faster in JDK 1.4 compared to JDK 1.3, but it is still slow compared to native Windows applications)

The first case, that the program is poorly written or not tweaked, happens more often than you think. I believe that it has to do with:

- Java is easy to learn, while C++ is not. This means that a lot of poor programmers learn Java and hence create poor applications and games. This is also a problem for Microsoft Visual Basic which also is plagued by poor programmers.
- Myths about tweaking. It is likely that certain optimization techniques that the programmer used to make programs faster in earlier versions of Java and in other languages do not apply to Java and actually make programs slower.
- Wrong use of the API functions. It is important to know what is fast and what isn't.

So basically, the main reason that Java programs are slow is that developers don't know how to use it properly. Writing efficient programs can be bit tricky especially since many of the optimization "truths" you know from C++ no longer is true in Java due to the Hotspot adaptive optimizer.

For instance, it has long been a Java "truth" that using pools of objects was much faster than allocating heap memory each time it is needed. This is only true to some extent today. Because of the generational garbage collector, pools may actually slow

things down in the long run since they will be moved to the second generation of the heap and cause subsequent garbage collection sweeps of the old object area to take longer to perform.

In the design of the Sun virtual machines, the focus was on optimizing their code for cleanly written code. This has, among other things, caused the allocation of heap objects to be much faster in Sun's virtual machine than in IBM's virtual machine (see chapter 7). This has led to the myth that cleanly written Java applications are the fastest kind of Java applications. This is simply not true. For instance, rewriting the code to eliminate heap allocations is still an efficient way of improving the performance of a Java application.

More information about Java code tweaking can be found in [Wilson] and [Shirazi].

6.5 Memory footprint

One issue that causes problems for Java performance today and probably also in the future is the large memory footprint.

A C++ program generally only needs to load the executable image, which is quite compact, into memory and then use stack and heap space as needed. The overhead of allocating data on the heap is usually small, often only a few bytes. It uses a few external libraries, such as the C standard library, but these are usually stored in dynamically linked libraries and usually already loaded by the operating system, because they are used by so many programs. Finally, C++ programs can allocate system memory as they see fit and at any given time only use what is needed. All this makes the utilization of memory for C++ programs very good.

A Java program, on the other hand, uses much more memory. A rough rule of thumb is that a large Java application usually use a little under twice as much memory as equivalent C++ counterparts.

This happens for several reasons:

- The Java virtual machine and associated libraries needs to be loaded into memory. Currently, Java virtual machines are not easily shared among multiple Java applications. The consequence is that this overhead is the same for each Java application that is running. Typical Java virtual machines use 5-10 MB memory each. According to [Hutchinson] a focus for the next version of Java might be to make it possible to share virtual machines automatically.
- Because there is more heap allocations in Java and because these each tend to use more space on the heap than their C++ counterparts, the memory footprint increases even more. The extra heap allocations tend to comes from two sources: 1. Java cannot allocate objects on the stack or even inside the other heap allocated objects. 2. When an array of objects is allocated in Java, one first has to allocate an array of references to the object and the explicitly allocate each object it contains.
- When a Java program starts the virtual machine is given a preset heap size that can be changed by a command line option. This is required because of the garbage collector. This means that Java programs use a lot memory just before garbage

collection starts. On the other hand, if a Java application momentarily needs more memory than available on the heap then the application runs out of memory even if there still is system memory available. This issue is discussed in more detail in section 6.6.

On PCs this high memory footprint of Java is not much a problem since many people today have system memory in abundance. However, on consoles this is a serious problem. For instance, a Sony Playstation 2 only has 32 MB of RAM. After the virtual machine is loaded, there is not much room left for actual programs and data.

Sun and Sony is aware of this problem and has, together with several other companies, begun work on what is called the Java Game Profile [JSR134]. This is a profile (see section 4.4.2) for the Java Micro Edition that specifies how consoles are to work – treating them as a kind of embedded devices. The Java Micro Edition is known for its low use of memory compared to the Java Standard Edition, so there is hope for the future of Java on game consoles. Java and game consoles are discussed in detail in section 8.7.2.

6.6 Selecting the heap size

One problem with garbage collected heaps is the amount of memory that is allocated by the operating system is increased compared to systems that use explicit deallocation. This happens because data is not deallocated immediately, but postponed until absolutely necessary.

Usually garbage collection is started automatically when the heap is about to be full. This means that even if the application uses few heap allocated objects it might eventually fill up the heap using system memory for it all.

Current virtual machines usually work with two values: the initial heap size (defaults to 2 MB) and the maximum heap size (defaults to 64 MB). The initial heap size may grow at run-time if garbage collection yields too little free memory until the maximum heap size is reached. At this point the system will simply run out of memory.

This means that the programmer actually must set a maximum amount of memory that the application may use at run-time. This is very unlike C++, which can access all the memory of the system.

If one simply increases the maximum heap size to something very large (say 1 GB) then the performance will almost certainly drop since the system will begin to swap the heap to disk if grows to more than the available system memory. If the initial heap size ever grows to more than the available system memory disk swapping will occur. This is a worse problem in Java than in C++ because Java provides less locality in its use of the heap.

Essentially, this means that it becomes vital to select a maximum heap size that is less than the available physical system memory.

This could pose a problem for regular applications where the amount of data is unknown until it is loaded – a hex editor for instance. For games it is not that much of a problem. The programmer would simply require that the computer had a certain amount

of memory, print this minimum amount on the outside of the game box, and then set the maximum heap size to this value.

Alternatively, one can write a small C++ program that detects the amount of available physical memory and that then starts the game with a heap size a little smaller than that.

So, in practice, setting the heap size explicitly for games is not much of a problem.

6.7 The performance of JNI

Some people claim that JNI is very slow and should be avoided at all costs. This is not entirely true. Accessing Java from C++ is slow, but accessing C++ from Java is quite fast. One Java developer I spoke with (Caspian Prince) showed me the profiling output of a partial OpenGL C++/Java interface wrapper that he had made and here the overhead involved with JNI was shown to be less than 0.5% of the overall processing time. With careful use this overhead can be brought down even further. In my own Master's thesis [Marner01a], which was written using a mix of Java and C++, the overhead was so small that it could not even be measured. For information about wrapping C++ libraries for use in Java see [Marner00] and [Marner01b].

It is, however, true that JNI should not be used mindlessly. In earlier versions of Java it was always an advantage to implement critical sections of the code in C++ (such as calculations in inner loops). However, in later versions of Java, Java is so fast that this usually no longer pays because of the overhead involved in JNI. [Wilson]

6.8 Templates / Generics

When discussing the pros and cons of the performance of Java vs. C++, one inevitable has to discuss templates (see section 5.19). Templates in C++ are provided primarily for one purpose: To make polymorphic functions/classes without the overhead of using a base class and having virtual methods.

Because the purpose of templates is accomplished by specialization, i.e. generation of machine code for each type that uses the template; this becomes just as efficient as had the code been written using a macro. The added benefit of templates over macros is improved type checking and ease of debugging.

This approach produces very efficient and elegant code and is therefore a much hailed feature of C++.

Java does not support templates. Generics (see section 5.19) are planned for JDK 1.5 but are not available in the current version. This is seen as a major draw back of Java by many C++ programmers and Java programmers alike.

Java allows polymorphy using virtual methods of common base classes. In fact, all Java objects are derived from a common base class called `Object`. This approach can be used to write code that is similar to template code with the only exception that it only works with object types – not primitive types. This limitation is a clear draw back in Java and this issue will not be handled even when Generics are implemented.

With regard to performance, the Java approach is much slower than the C++ approach in current virtual machine implementations.

But there is principally nothing that prevents the Java approach for being as fast as the C++ approach in future versions of Java. Just as virtual methods and explicit type casts can be eliminated, Hotspot should, in principle, also be able to detect if certain classes or functions only are used with certain types and then generate specialized versions of each class or function. Unfortunately, this type of optimization is not yet implemented in Java virtual machines.

The currently implemented optimization approach used in Java is to try to eliminate explicit type casts thereby making the code as fast as the C++ version. Unfortunately, this does not currently always work well enough, so using generic classes in Java is often quite slow (see section 7.11.2 for an example of this)

To summarize; although there is nothing that prevents Java from competing with C++ templates, current implementations of Java still lacks severely behind in this area.

6.9 Strings

In Java, strings are represented internally as Unicode characters (2 bytes per character) while C/C++ traditionally represents them as ASCII characters (1 byte per character).

String operations will therefore tend to be roughly twice as fast in C++ as in Java.

6.10 Tweaking and readability

The consequence of tweaking Java code is greater than that of tweaking C++ code both with regard to performance and to the appearance of the code.

In C++ tweaking usually does not change the appearance of the code much. A speed up of 20-30% can usually be gained by doing this and it does not reduce the readability of the code significantly.

This is not the case in Java. There are several reasons for this. The greatest problem is that objects in Java cannot be stored on the stack. This means that the Java programmer allocated objects on the heap in many cases when the C++ programmer does not:

- Every time some temporary object is needed. C++ programmers can often just allocate them on the stack.
- Every time a primitive data type needs to be passed by reference to a method. In Java primitive data types are passed by-value, so to pass it by-reference it must be encapsulated in an object that might be created for the purpose.
- Every time more than one value is to be returned from a method. In C++ the programmer would just create a temporary object to be returned on the stack, but with Java a new object must be allocated on the heap.

To eliminate all these cases can be a complex task. Examples of this can be found in the benchmarks of chapter 7. For instance, see the difference between the implementation

of `Frustum.java` in the tweaked vs. the untweaked version of the 3D Demo made with `GL4Java`.

Some of the tricks needed to eliminate the heap allocations are:

- Allocate temporary objects outside the critical loops and reuse them by changing their state. The drawbacks are (1) that this in some cases prevents the use of recursion, (2) allocations does not happen in the code where the object is needed, and (3) objects can no longer be immutable, since immutable objects by definition cannot change state.
- To return multiple values the programmer can let the return value be passed back to the caller in an object that is passed by reference by the caller. Unfortunately, this makes it less clear to the reader whether arguments are ingoing, outgoing, or both.

6.11 Conclusion

Today tweaked Java code is a little slower than tweaked C++ code on the average, typically around 20-50% slower, but this may vary a lot. In some cases Java is up to twice as fast as C++ and in some cases it up to three times slower.

Untweaked Java code is usually 2.5 – 4 times slower than untweaked C++ code but this varies much.

Tweaking the code causes the readability of the code to be significantly reduced and hence also the productivity of the programmer. Fortunately it is very often only necessary to tweak very little parts of the code, since very little of the code usually is run most of the time.

7 Benchmarks

In this chapter we will run some benchmarks that compare the relative speed of Java and C++. These results are used, together with the results from third party benchmarks, to reason about the speed of Java in relation to C++

The full benchmark results are all listed in tabular form in Appendix I.

7.1 Introduction

The only proper way to compare two languages is to come up with a problem and then solve the problem in the two languages using the same general algorithms. Code cannot be converted mechanically, but by focusing on the performance of *problem solving* rather than the performance of *program instructions* we will get more trustworthy results.

The best solution would be to write a major application or game in the two languages and then compare their speed when used by a regular user. However, doing so is a lot of work, so the approach generally chosen is to build some smaller applications that are believed to represent larger applications fairly well.

I have observed that people tend to become defensive whenever a benchmark shows that their favorite language or library looks bad. I can only assure the reader that I have made a serious effort to make the benchmarks such that they do not favor one language or library over another.

Benchmarks should always be taken with a grain of salt since they never completely represent full real-life applications. In fact, real-life applications may very well perform differently. Nonetheless, benchmarks can be a guide that helps to tell what the trend in performance seems to be.

7.2 Choice of compiler/virtual machines

When running benchmarks, it is essential to use the compilers and virtual machines that provide the best performance – since these are the ones that will be used for production code.

Although I would very much have liked to perform benchmarks on various game consoles this has not been possible, because I do not have a development license for those machines (see section 3.3.2). I have therefore chosen only to perform benchmarks on the primary desktop computer platform, namely the Intel Windows PC.

On this platform Microsoft Visual C++ is the de facto compiler, so this will be included in the test. The Intel C/C++ compiler is reputed as being the fastest C/C++ compiler available on the platform, so this will be included too.

Since gcc is one of the compilers that are used to compile for consoles, I have considered including it in the test, too. But it does not make sense to deduce anything about the speed of consoles based on the speed of gcc on the PC, so the benefit of

including gcc in the test is minimal. To save time I have therefore not included gcc in the benchmark.

For Java I will use the latest version of Java from Sun, since this is the de facto compiler/virtual machine used by the community. I will also include the latest IBM virtual machine since this generally is reputed to be fastest available. Finally, I have chosen Excelsior Jet to represent Java static native code compilers since it generally is considered among the best ones available.

The following compilers and virtual machines are used in all the tests:

Compiler	Run-time system	System Type	Abbreviation
Microsoft Visual C++ 6.0 SP5	<is an executable>	C/C++ compiler	MS C++
Intel C/C++ 5.0	<is an executable>	C/C++ compiler	Intel C++
Sun JDK 1.4.0 compiler	Sun JVM 1.4.0 client hotspot	Java VM	Java 1.4 client
Sun JDK 1.4.0 compiler	Sun JVM 1.4.0 server hotspot	Java VM	Java 1.4 server
Sun JDK 1.3.1_01 compiler followed by Excelsior Jet Professional 2.1	<is an executable>	Java static native code compiler	Jet
IBM JDK 1.3.0 preview edition 25th sep 2001 compiler	IBM JVM 1.3.0 preview edition 25th sep 2001 virtual machine	Java VM	IBM

I have also briefly tested the IBM Jikes compiler. It seems to perform identically to the IBM JDK 1.3.0 compiler so it is not benchmarked separately here.

All of the used compilers and virtual machine can be downloaded on the Internet for free (Excelsior Jet and the Intel C/C++ is only available for an evaluation period) with the exception of Microsoft Visual C++ which must be purchased in retail.

For both C++ and Java the code is compiled using full optimization, optimizing for speed using maximum inlining (i.e. both the /O2 and /Ob2 command line options). For Excelsior Jet maximum optimization, except for *JetPerfect*, was enabled including all cheats to make the program run faster. These cheats includes the deactivation of null checks, array bounds checking, and checking of type casts. JetPerfect was not used since

no documentation about its use was included with evaluation edition of Jet. The full set of compiler options used for Jet is listed in Appendix I.

7.3 The system setup

The benchmarks were run on a single AMD Athlon 700Mhz processor with 392 MB PC100 RAM running Windows 2000 Professional SP2. The system contained an nVidia Geforce 256 video card (from LeadTek) with 32 MB RAM using the nVidia 23.11 reference drivers from December 2001.

Note that the Intel C/C++ compiler is specifically optimized for Intel processors and should run somewhat slower on AMD processors. However, since AMD processors are very common among game players, I think using an AMD processor is acceptable for game benchmarking purposes.

7.4 The benchmark applications

In an attempt to produce trustworthy results, the goals of the benchmark applications should be:

- They should be representative for the behavior of real-world applications.
- They should be small enough to be worthwhile to implement in two languages.
- If computational intensive libraries are used then the *exact same* libraries should be used in both cases. If this not possible then the libraries should not be used.
- A least one benchmark should use data sets that are so large that the system as a whole cannot be placed within processor cache.
- The benchmarks should avoid string handling. Java uses Unicode while C uses ASCII - comparing these would be to bias the benchmarks towards C++.

We will not test the GUI system since it is a well-known fact that this is much slower in Java than in C++ Fortunately it is often not used in games.

I have implemented three benchmark applications to fulfill the goals above:

- A particle simulator: This measures raw computational performance.
- An object oriented implementation of John Conway's game of Life: This measures the performance of highly object oriented code where some heap allocations cannot be eliminated.
- A 3D demo using OpenGL: This measures the performance of 3D hardware-accelerated rendering.

7.5 Benchmarking both tweaked and untweaked applications

Traditionally, benchmarks compare only code that has been tweaked (i.e. hand-optimized) by the programmer for optimum performance. This has the strong advantage that we can objectively say that both programs are of an equal quality - the best.

When tweaking code I will require that it is done within certain constraints. For instance, if the point of a benchmark is to be representative for a big object oriented application then I should not eliminate the use of all objects for the sake of speed.

Unfortunately, tweaked benchmarks are often not very representative for real world applications. Tweaking code is, especially in Java, a slow process and something that is more easily done to small applications than large ones. Real-world applications will only be applied limited performance tweaking and only in bottle neck areas of the program. Because a program very often has many internal dependencies, it can become hard to tweak certain areas of the program without making the rest much harder to maintain. So in general, the larger a more complex a program is the harder it is to tweak. Traditional benchmarks are usually small so it is usually no problem tweaking them - but if I tweak them fully then they stop representing real applications and their credibility as benchmarks are reduced.

In order to compensate for this, I will benchmark both tweaked and untweaked code. I will let untweaked code be defined as code that is written according to the coding styles normally recommended for the language as taught in programming books. This will cause differences in performance that might seem unfair. For instance, in C++, users do not usually declare methods `virtual` that does not need to be so. Java users, on the other hand, rarely use the `final` keyword even if they don't expect to be able to use it as a virtual method.

The purpose of untweaked code is to make a test that is representative for the majority of the code being written and run in the world. The majority of a real application will consist of such code, so doing this kind of test is very relevant.

One might claim that an untweaked test cannot be compared fairly because of the differences in coding styles from programmer to programmer. I have nonetheless made an effort to write the applications in a fair way like I expect a typical programmer would do it (or like I would do it myself if the goal was cleanly written code).

To summarize, untweaked benchmarks are not perfect, but neither is tweaked benchmarks and to accommodate for both points of view, I have included both in this report.

7.6 Warm-up periods

When making benchmarks that make use of JIT compilers and especially Hotspot virtual machines, one must take special care of the effects of warming them up.

During the initial warm-up period these virtual machines are significantly slower than statically compiled code. This will make these virtual machines automatically look bad in benchmarks that are run for a very short period of time.

Since real-world applications are likely to run for long time, the warm-up period will have no significance in real-world applications. The benchmarks should therefore eliminate the effect of it in order to give a credible result.

Most other benchmarks of Java that I have seen on-line make this fatal mistake and can therefore often not be trusted. We will review some of these in section 7.11.

To eliminate the warm-up period, we will perform two steps:

First, for each benchmark application we will explicitly measure the warm-up period on Sun's 1.4 server JVM. This is the one that reputedly has the longest warm-up period. This can be done by measuring the execution times for linearly increasing problem sizes. When the growth in execution time becomes constant with each increase in problem size the warm-up period has ended. And just to be sure the test is not the victim of random variation, we will pad the measured warm-up period with some extra time.

The measurement of the warm-up period assumes the warm-up period for a piece of code is independent of its exact use. This assumption does not hold in all cases, which also is a reason to pad the warm-up period with some extra time.

Second, after the period has been measured, we will do the regular benchmark runs, running the runs for different problem sizes. In order to eliminate the effect of the warm-up period we will only observe the performance difference by observing the difference in the growth in execution time each time the problem size is increased. This difference in growth will be a direct indication of the relative speeds of the two systems.

7.7 Generation of test data

In order to make it easy to scale the benchmarked problems in an easy way, the used data is randomly generated. Some third party benchmarks choose to generate the same test data every time, but this approach runs the risk of making some systematic biasing towards one of the implementations. Randomly generated data also has a small risk of doing this, but it is much less than if the same data is used every time.

When using random data, one runs a random risk of producing data that will behave oddly when run. For instance, in the particles simulation below, if all particles start on top of each other they will collide and the result will not be very representative of a typical run.

To compensate for this, all runs are repeated five times and the median value of the execution times has been taken. It would be incorrect just to take the average of the measured values, since a rare single highly deviating run would influence the final result.

7.8 A particle simulator

As the first benchmark application I have chosen to write a particle simulator. A set of particles (in this case planetary bodies) are put into 3D space with a given position and impulse. Gravitational forces are then made to work between them. When two particles collide they are combined into a new and larger particle. When a particle leaves the bounds of the world it is removed from the simulation.

The computation has a time complexity of $O(n^2 * m)$, where n is the number of particles and m is the number of time steps. In order to avoid that the number particles are reduced late in the simulation we will set the size of each time step to something very small.

This application is very computational intensive, and it should be seen as a representation of those parts of larger applications that contain mathematical computations.

Only the actual simulation steps of the particle simulation is timed. The generation of the data set and the computation of the used time step are not. The benchmark does therefore not use I/O or any library intensively, with the sole exception of the standard library `sqrt ()` function which is needed during the calculations of the force working between the particles.

The source code can be found in Appendix B and Appendix C for the untweaked and tweaked versions respectively.

The C++ implementation is a simplified version of a particle simulator that I wrote together with a fellow student, Søren Skov, as part of a university course in parallel programming and distributed systems in the autumn 2001.

7.8.1 Implementation notes

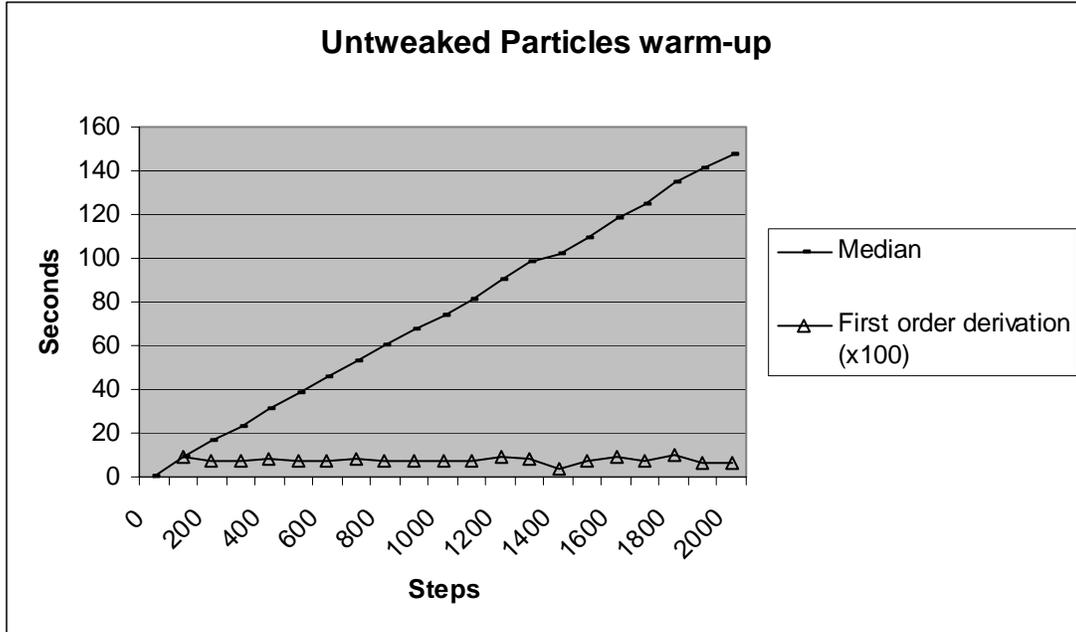
The set of particles is implemented as a linked list. This I have done because particles only are accessed in order and we need to be able to delete elements that we just have accessed. It could just as well have been implemented as an array. Doing so would likely have reduced the speed of the Java version (due to array bounds checking) but made no difference for the C++ version.

The C++ version uses the STL list which is doubly linked list while Java uses a singly linked list. The List from the Java standard library could not be used because we needed to be able to copy iterators.

The fact that C++ uses a doubly linked list while Java uses a singly linked list has extremely little effect on performance. The only consequence it has is that an extra pointer has to be set when a particle is deleted in C++. Particles are deleted extremely rarely in the benchmarks, since we, as noted in the previous section, set the time step to a very small value to avoid any particles from disappearing.

7.8.2 The untweaked version: Warm-up period

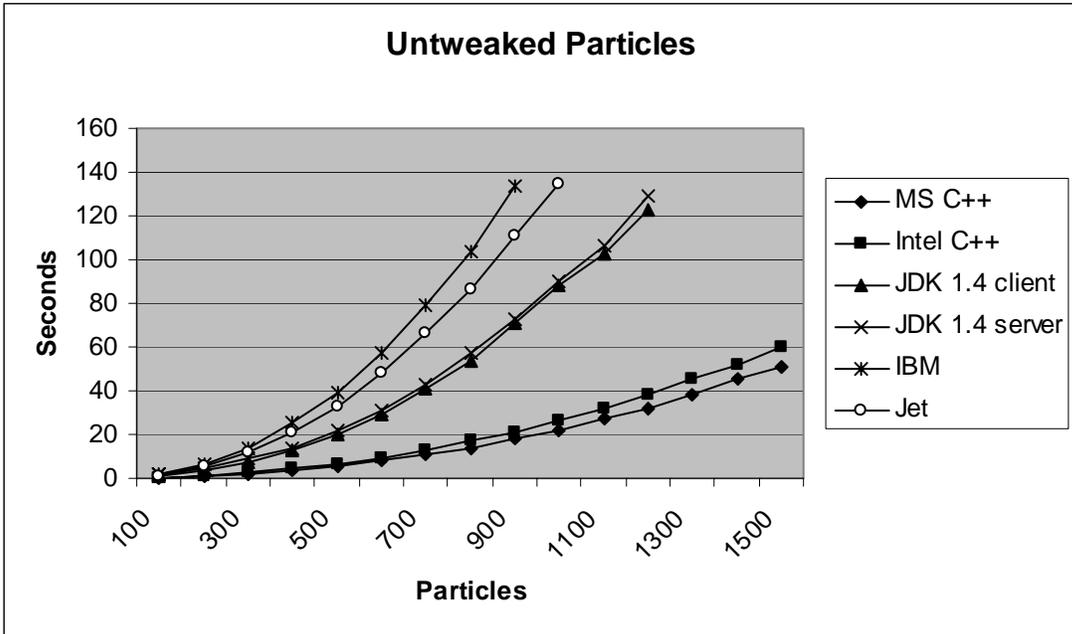
To measure the warm-up period of the untweaked Particles application I keep the number of particles constant at 300 and then increase the number of time steps gradually. Since this causes a linearly increase in time complexity, execution time should increase linearly too when the warm-up period is over.



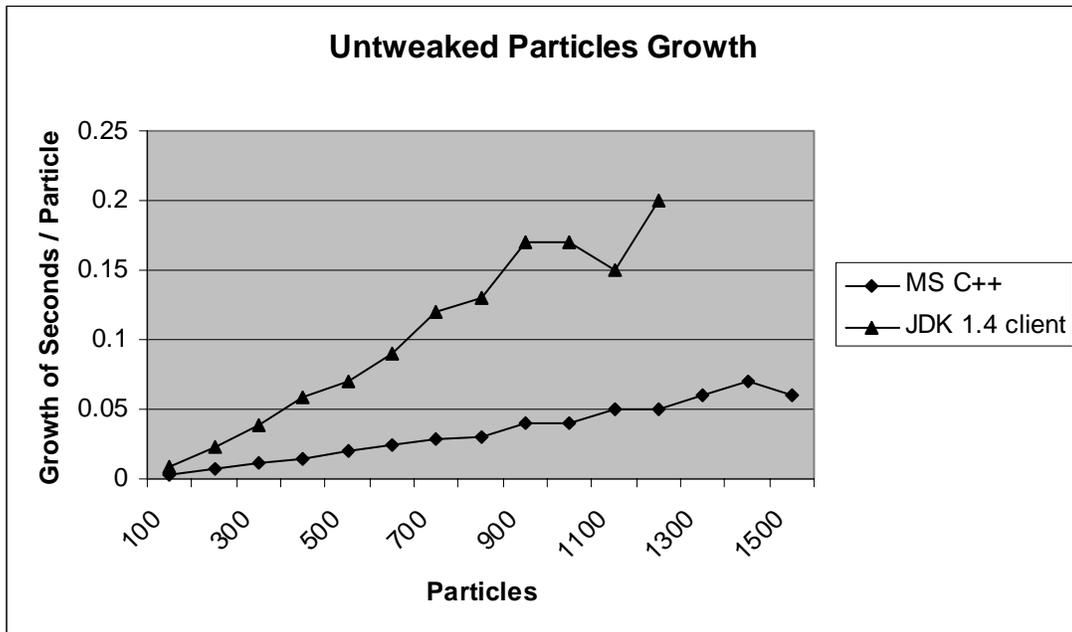
From the graph above there is very little indication of any warm-up period going on. If it is there, it is certainly over after about 20 seconds including padding.

7.8.3 The untweaked version: The results

The following graph shows the absolute timings measured when running the benchmark. I have kept the number of steps constant at 100 and have kept the step size (h) constant at the low value of 0.001.



The fastest C++ version is Microsoft C++ and the fastest Java version the Sun JDK 1.4 client virtual machine. In order to eliminate the influence of the warm-up period in the comparison of these two programs, we look at the growth in time per growth in problem size. In other words we differentiate the graph above with respect to the number of particles. In order to avoid cluttering the graph, we only look at the two fastest versions:



Sun's JDK has used 20 seconds when it is run with 500 particles and above so the comparison should be done only for 500 particles or above.

We observe from the graphs that:

- Here we see that the fastest Java version is approximately 3 times slower than the fastest C++ version.
- Nothing is gained in this case by using a static compiler like Jet over a Hotspot VM.
- The IBM JVM is in this case no faster than the Sun Hotspot JVMs. This is a bit surprising since it generally has the reputation of being the overall fastest JVM available.
- It is the Java 1.4 client that is the fastest JVM. This is surprising since we would normally have expected the Sun server JVM to be the faster of the two.

7.8.4 The Tweaked version: Introduction

Both the Java and C++ versions of the Particle program have here been tweaked for performance.

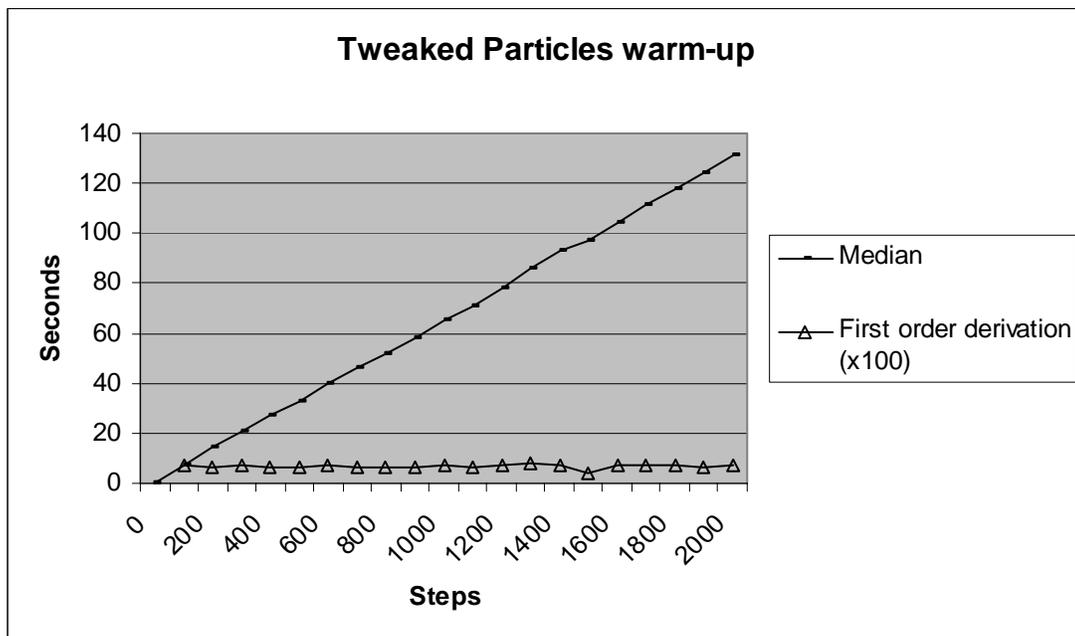
The change that caused the largest effect for the C++ version was to change the implementation such that as few as possible new vector objects were allocated at run-time. Instead old vector objects were reused when possible.

For Java the same change was made and during this process all heap allocations was eliminated from within the timed part of the program.

Readability was lowered more in the Java program than in the C++ program.

7.8.5 The tweaked version: Warm-up period

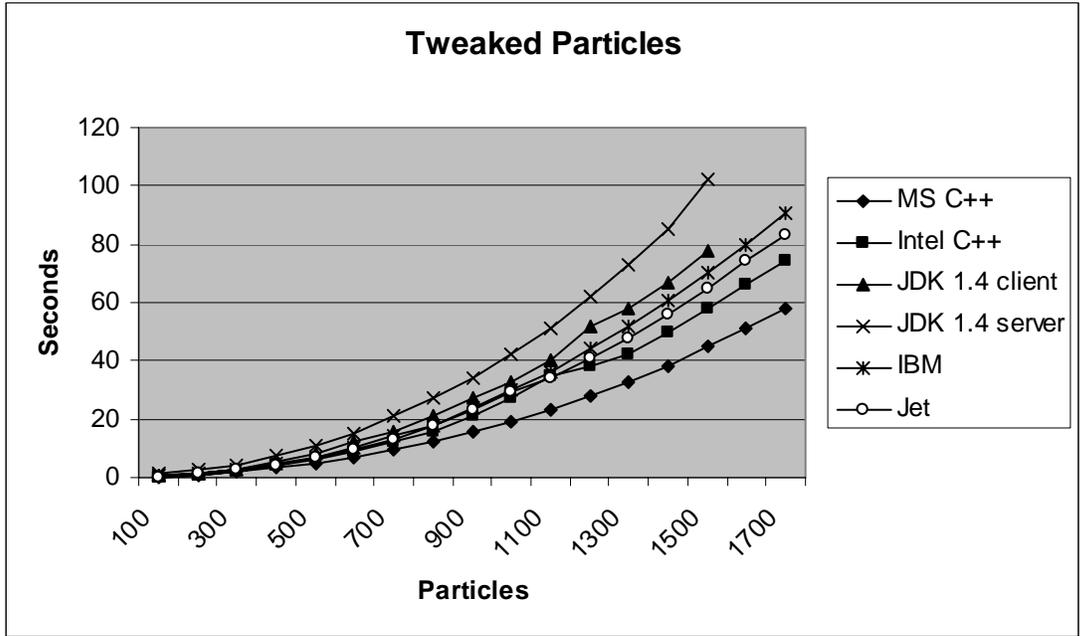
Repeating the warm-up measuring process using the tweaked version of the program using Sun's JDK server VM we get the following result:



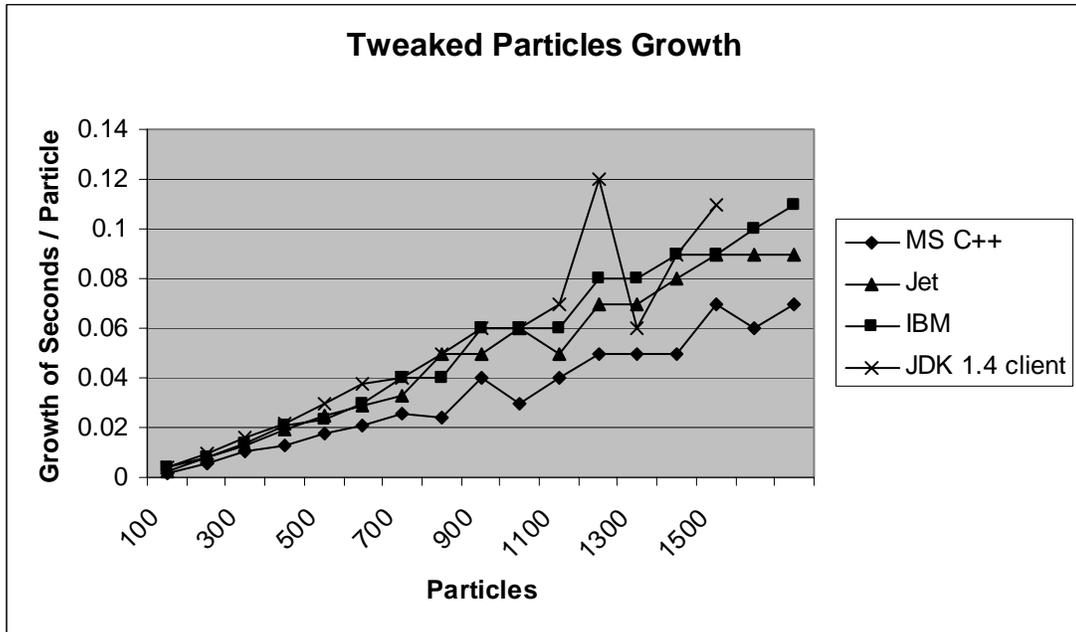
Here there is absolutely no indication of a warm-up period. We can therefore safely assume that it is over after 15 seconds, although it probably only lasts a few seconds.

7.8.6 The tweaked version: Results

Again the number of steps is 100 and the step size is 0.001.



Again Microsoft C++ is the fastest. The fastest of Java versions are this time Excelsior Jet closely followed by IBM's VM and Sun's JDK client VM.



The tweaking caused more than a double up in the speed of the Java program but only a 5-10% speed increase for the C++ program. That the speedup was minor for C++ was expected. The major speed up from Java comes from the removal of all heap allocations from the inner loops. So although Java heap allocations are not as expensive as they were in previous versions of Java, they are still something that should be avoided in optimized code if at all possible.

After I tweaked the programs, Java has seriously gained on C++. Java has not quite achieved C++ speed but is not very far of. This time the Jet version is the fastest of the Java versions, being about 30-50% slower than C++. The IBM VM and the Sun client VM follows just after being about 60% slower.

As expected, there is no effect of the warm-up period in this application so the relative positions of the programs in the growth graph directly mirror the positions in the graph with absolute timings.

The IBM VM is the fastest regular VM this time (Jet is a static compiler and therefore not what is considered a regular VM). This is a little surprising since it performed quite poorly in the untweaked version. I can only conclude that IBM VM does not handle things such as heap allocations very well. For raw computations, however, it is the fastest regular VM there is.

7.9 An object-oriented version of Life

The next application is an implementation of John Conway's game of Life [Gardner70] [Gardner71]. Traditional implementations usually represent the world as a 2D array of booleans and then update it procedurally. Such a third party benchmark suite comparing C++ and Java is reviewed in section 7.11.2.

However, in the benchmark in this report I wanted to focus on the performance of object-oriented code. I have therefore implemented Life such that each cell is an object and this object will, when time passes, check whether it should die or whether it should spawn new neighbors. Cells are derived from an abstract class `Entity` that can be used to implement cells with other kinds of behaviors. The purpose is to tell the speed of cleanly designed object-oriented applications.

The time complexity of the algorithm is $O(n^2 * m)$, where n is the side length of the two dimensional world and m is the number of time steps performed. The computation speed used per position in the grid per step will depend on whether there is a cell in it or not, but this does not change the complexity of algorithm.

The starting world is generated randomly. Each position in the world initially has 40% chance of containing a cell. In most cases the number of cells will rapidly drop within a few time steps and then end in a steady state or even oscillate between a few states.

Besides handling many heap allocated objects, this benchmark is characterized by having a high memory requirement.

7.9.1 Implementation notes

The default maximum heap size in the latest versions of Java virtual machines is 64 MB (this value is JVM dependant). This is not enough to run this benchmark when the world gets bigger.

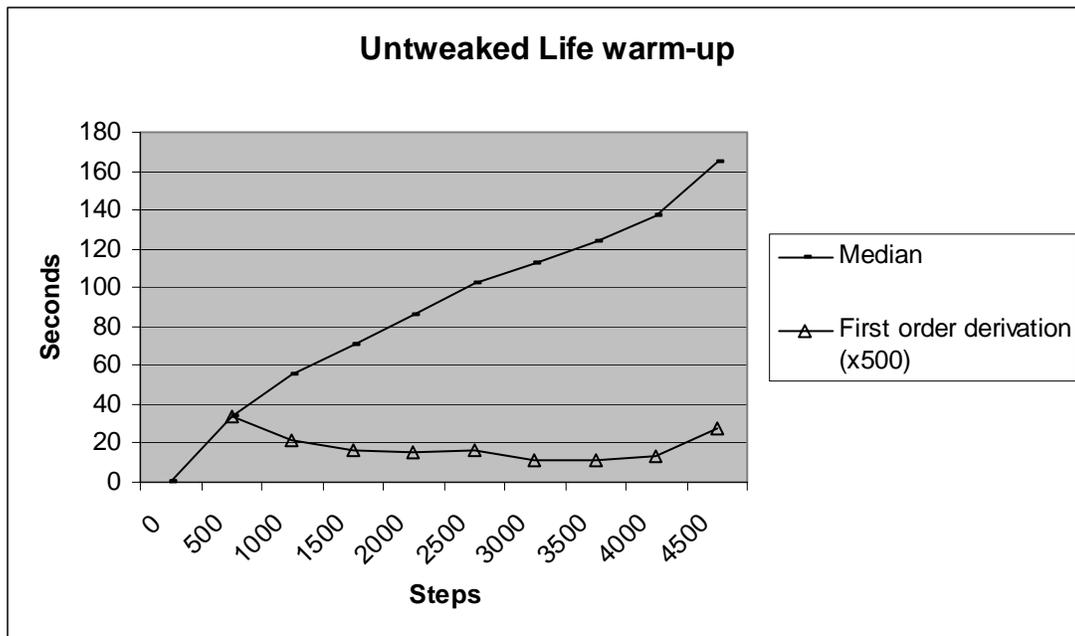
To handle this, the regular Java virtually machines are run using the non-standard parameter `-Xmx200MB`. This sets the maximum size of the heap to 200 MB. This means that if the initial heap becomes too small then the heap will grow until 200 MB in size until it accommodates the needed data. A little more performance could be gained for Java by also using the `-Xms` parameter, but doing so would cause much memory to be used even when it is not needed and has therefore been avoided in these tests to avoid causing a bias in favor of Java.

For Excelsior Jet the maximum heap size was set using the compiler parameter `-heaplimit=209715200`, which sets it to 200 MB.

The source code can be found in Appendix D and Appendix E for the untweaked and the tweaked versions respectively.

7.9.2 Untweaked version: Warm-up period

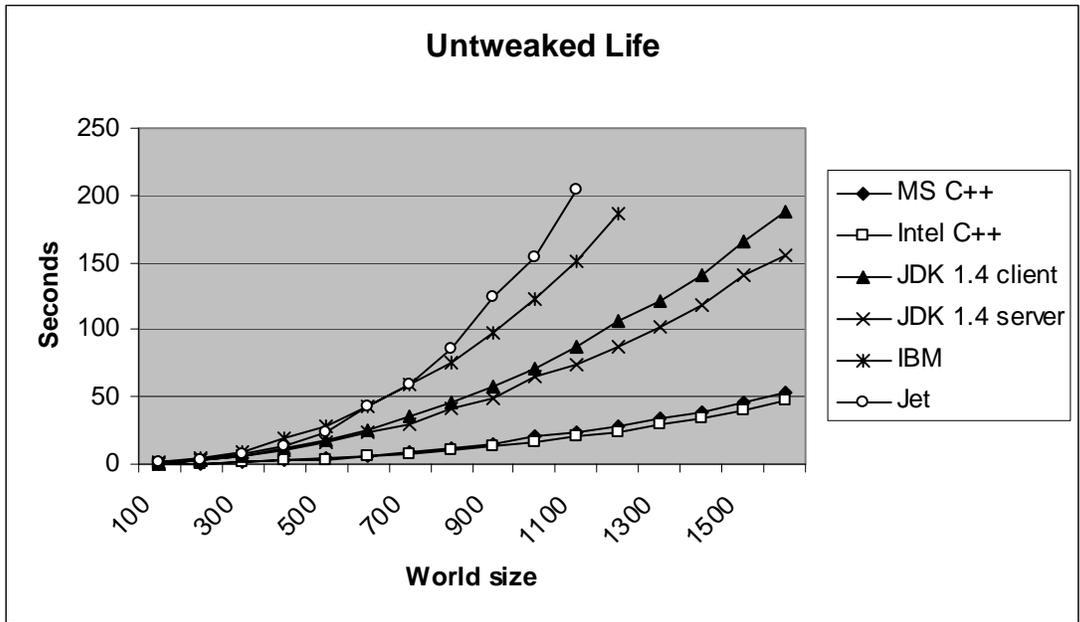
To measure the warm-up period we keep the side length of the world constant at 200 and vary the number of time steps computed. Since we expect the cells to oscillate between a low number of states after a few time steps we can assume that the computation time is linearly dependant on the number of steps. So when the growth in time becomes constant the warm-up period has ended.



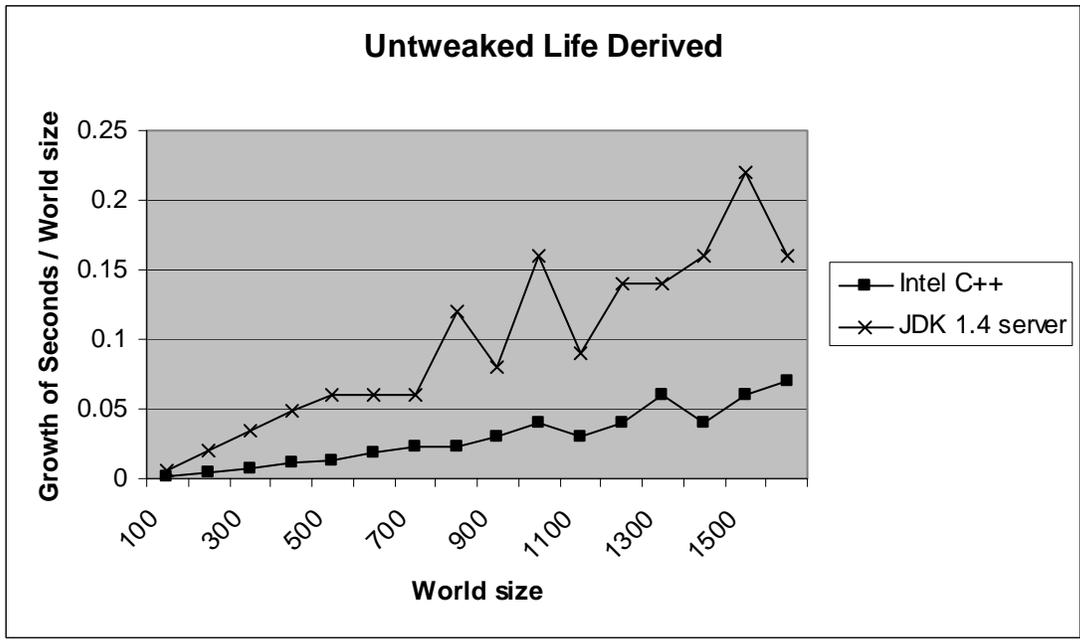
From this graph it is clear that there is a warm-up period. Unfortunately it is fairly hard to see when it ends. Although the program seems to become a little faster after 100 seconds this is only temporary, so I estimate that it in fact already is linear at this point. In

fact, the curve already seems to be linear after about 70 seconds. To be safe in case of variations in data we will assume that the warm-up period is 90 seconds.

7.9.3 Untweaked version: Results



The best C++ compiler for this benchmark is Intel C++, although there is not much difference between that and Microsoft's compiler. The best VM is Sun's JDK 1.4 server. These two are shown below, differentiated with respect to the size of the world:



The warm-up period has ended for the Java server VM at 90 seconds and this is at a world size of about 1300. We should only compare their relative speed after this period. From the differences in growths in the graph above we see that the fastest Java is 2 – 2½ times slower than the fastest C++.

In this benchmark we see the approximately same behavior as in untweaked Particle Simulator. Java compares better this time though. This is probably due to the fact that the C++ uses actual heap allocations this time and can't put all the data on the stack as it could in the Particle simulator. So in highly object oriented systems where C++ would also have virtual methods and abstract class and many heap allocations, Java compares better because its allocation and garbage collection is fast compared to explicit allocation/deallocation.

This benchmark also shows that static Java compilers should not be used blindly. Using Jet in this case made the program several factors slower and it seems to scale even worse.

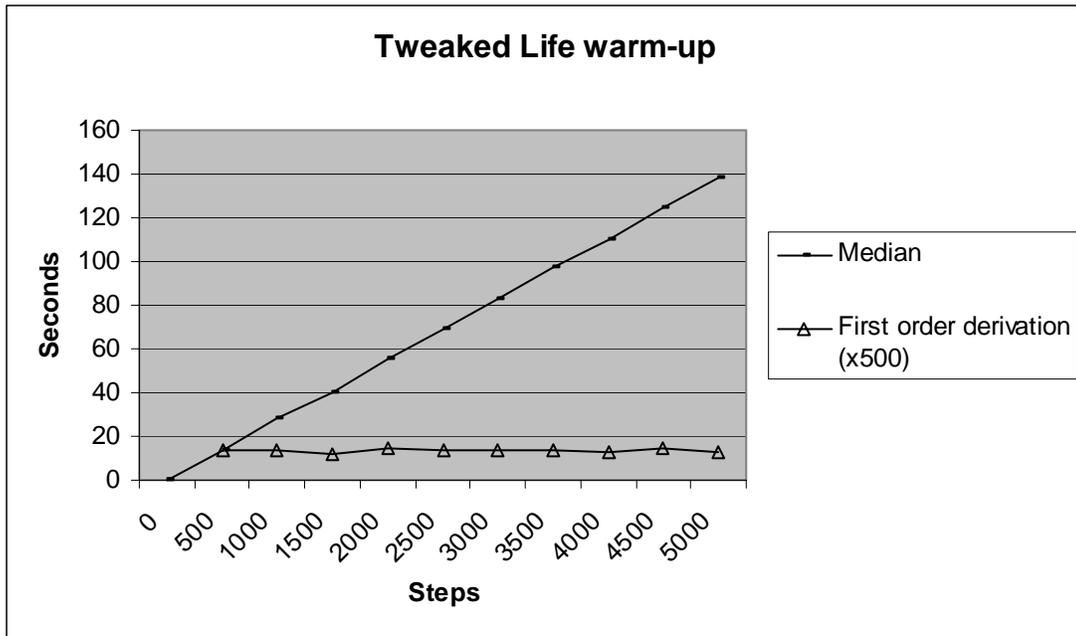
Like in the untweaked particle simulator, we again get a poor performance with IBM VM. This strengthens the hypothesis that the IBM is not very good at doing heap allocation.

7.9.4 The tweaked version: Introduction

The Life program has now been tweaked for performance. After the tweaking has been performed there are no longer any immutable objects. Cells are still allocated on the heap, though. These have not been eliminated on purpose, because such elimination often is impossible in large complex object oriented systems.

If we had allowed such elimination it would be possible to reuse the Cell objects (like in the Flyweight pattern [Gamma]) and this would probably have improved performance significantly.

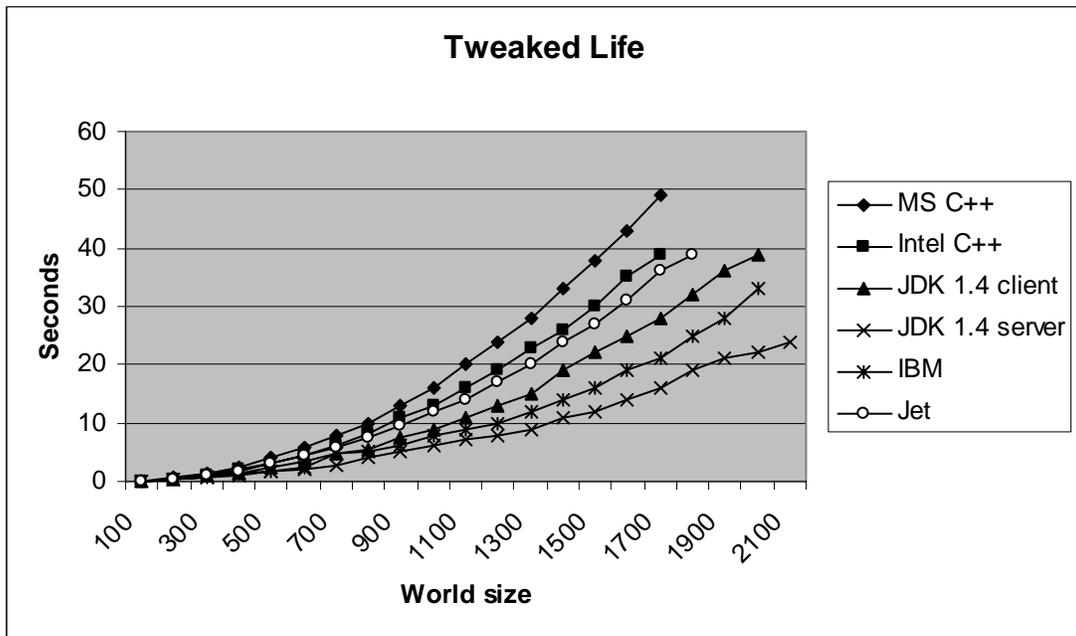
7.9.5 The tweaked version: Warm-up period



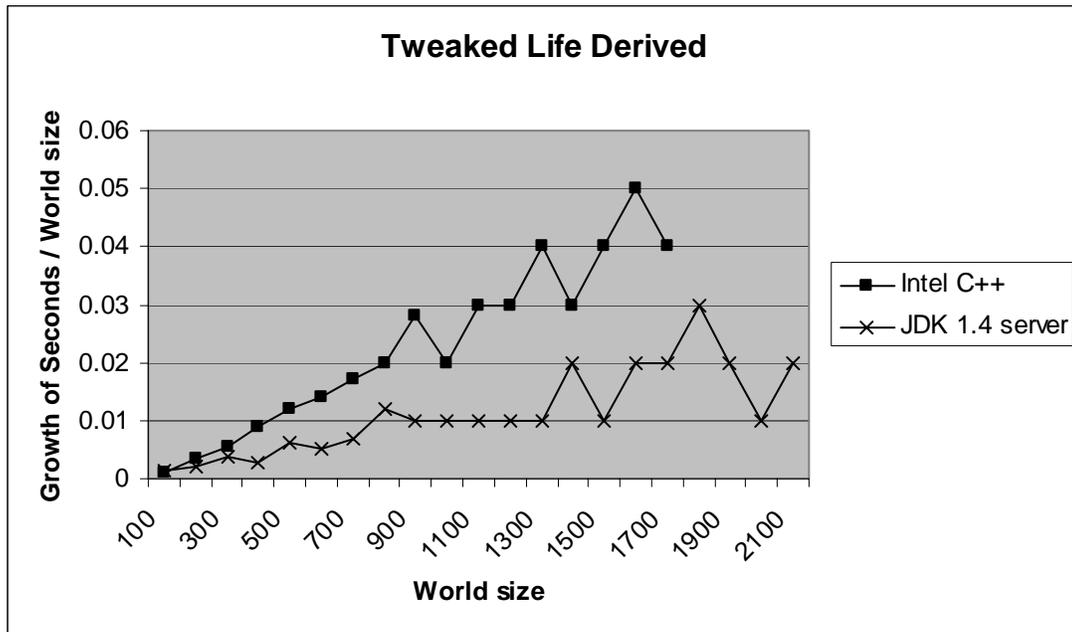
There is absolutely no indication of a warm-up period in this program.

For the remainder of this benchmark we will work with the assumption that the warm-up period is over after 30 seconds – it is probably much less.

7.9.6 The tweaked version: Results



The fastest Java is again the Sun 1.4 server VM and the fastest C++ is Intel C++.



After we performed the tweaking the C++ program gained about 25% in speed. The Java version became much, much faster.

The application running on the Sun JDK 1.4 server VM is 2-3 times faster than the fastest C++ version. Tweaking the Life application approximately yielded about a factor of 10 in speed! This shows just how important it is to perform tweaking of Java code, while it is not so important for C++ code.

7.10 A 3D demo

The third and final benchmark application used in this report is a 3D non-interactive demo that attempts to be representative of 3D applications or games.

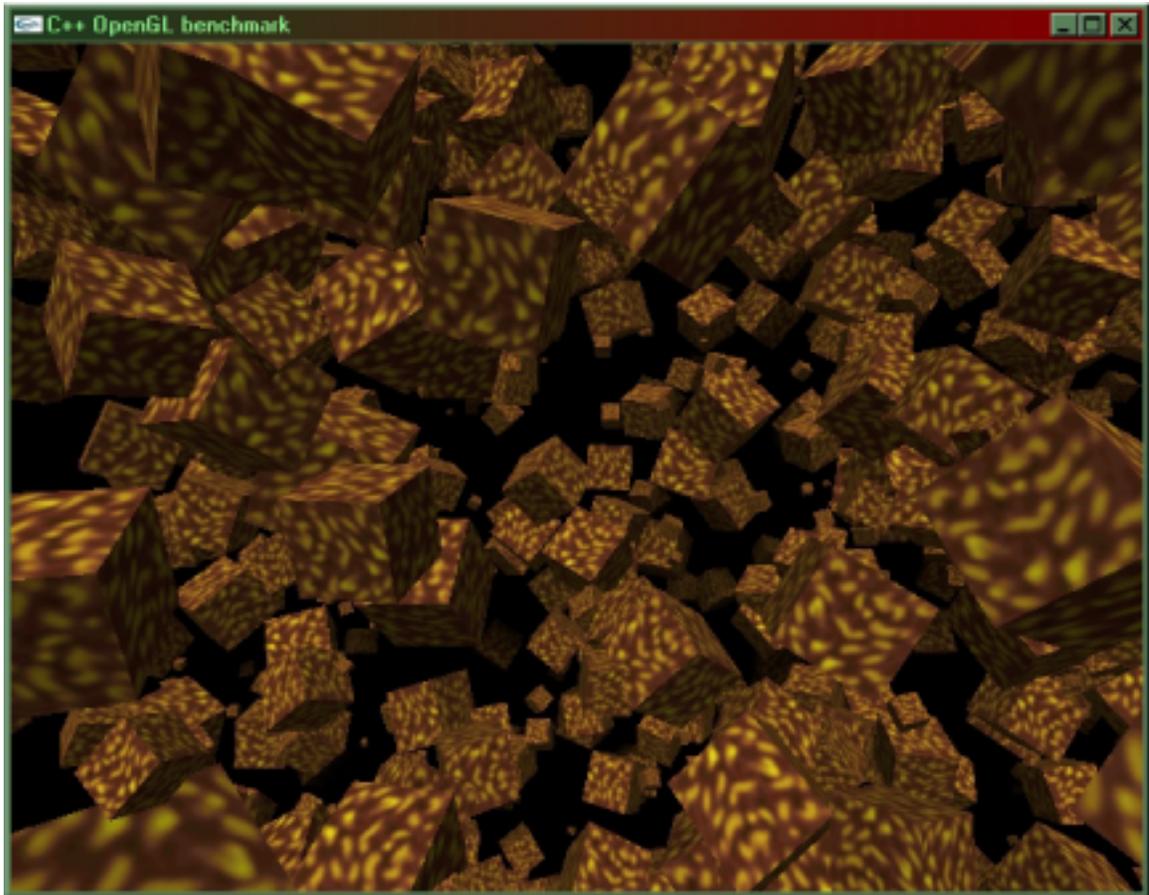
Since modern games, such as first person shooters, often consist of large static levels this is where I will focus this benchmark. A world is created and it is populated with a given number of randomly positioned, scaled, and rotated boxes. The camera then flies through the world in a circular motion rendering them as fast as possible.

In C++ the benchmark application will be implemented using OpenGL. It will use GLUT for portability. On the Java side it will be implemented in two versions. One will use the current de facto OpenGL bindings for Java, *Jausoft GL4Java*. The other will use Java3D. This also allows us to compare the relative speeds of GL4Java and Java3D.

The 3D hardware, drivers, and low-level graphics API can be important factors when running such a benchmark, and in order to eliminate the effect of these I will run them on the same 3D hardware using the same driver. To achieve this for the Java3D version I

will use the OpenGL version of Java3D. This is also reputedly faster than the DirectX version.

The following screenshot was taken from the C++ version in a world with 8000 boxes:



7.10.1 Implementation notes

All three implementations of the 3D Demo use the same 3D features:

- Linearly attenuating fog.
- Perspective projection with the same view frustum (a 90° wide field of view)
- Textures, mipmaps, and trilinear filtering.
- A global ambient light and a single directional light (phong lighting model) with no distance attenuation.
- Nicest perspective correction (The same terminology is used in both Java3D and OpenGL).
- Flat shading.
- Back face culling.

- The drawn surface colors are a modulation of the light and texture colors. This makes the texture look lit.
- The view port is 640 x 480 pixels.

To make the application more like a real 3D application, it also implements view frustum culling. Java3D already does this automatically so here only the correct bounding boxes needs to be set. The bounding boxes in both Java3D and OpenGL are identical.

For the OpenGL implementations a kd-tree data structure was used to subdivide the world [Berg]. This means that the C++ and GL4Java version, much like the Java3D version, use a scene graph.

The culling algorithm in the OpenGL versions scans the kd-tree and discards and boxes that lies outside the view frustum. The culling algorithm uses two steps: First it does bounding sphere collision detection between the view frustum and the bounding box (the sphere extends to the corners of the box). Then a regular bounding box vs. view frustum culling is performed. The plane of view frustum that was last used to cull the given bounding box is tested first. For a discussion on culling algorithms see [Assarsson].

The C++ and GL4Java versions supports asymmetric view frustums while the Java3D version only supports symmetric view frustums. The C++ and GL4Java versions do therefore not implement occlusion culling like I am told that Java3D does. This gives a small bias towards Java3D.

Since the world is static, the box positions in the OpenGL versions are not set each frame by changing the OpenGL modelview matrix. Rather, the actual vertex positions are changed in a preprocessing step and these are then entered directly into an OpenGL display list. Java3D does this automatically in our case. (This assumes that no capability bits are set and no geometry is stored by reference in that branch of the scene graph. These assumptions hold in the benchmark)

In the Java3D version I have not manually merged all Shape3D objects at each branch group. According to Chien Yang of the Java3D development team merging is done automatically during scene graph compilation as long as their Appearance object are shared – which they are in this benchmark.

A big effort has been made to make each of the implementations as identical as possible and for the tweaked versions to make them as fast as possible. For Java3D, version 1.3 beta 1 was used since this according to the developers is the fastest version available. To make the comparison fair to Java3D, the other implementations only uses OpenGL features available in OpenGL 1.1 and earlier. No vendor specific extensions were used, although the use of certain nVidia OpenGL extensions would have increased speed of the C++ and GL4Java implementations significantly.

GL4Java version 2.8.2 was used for the GL4Java version.

The Java3D version could not be tweaked much, so the performance is identical in its tweaked and untweaked versions. It has therefore only been included in the comparison of tweaked code.

The complexity of the algorithm used in this benchmark is $O(n * m)$ where n is the number of boxes and m is the number of frames. The reason why it is not $O(\log n * m)$, as would normally be expected, when one uses a tree culling algorithm, is that the world size stays constant when the amount of geometry increases. This causes the amount of geometry within the view frustum to increase linearly with the number of boxes.

The source code of the untweaked and tweaked versions can be found in Appendix F and Appendix G respectively.

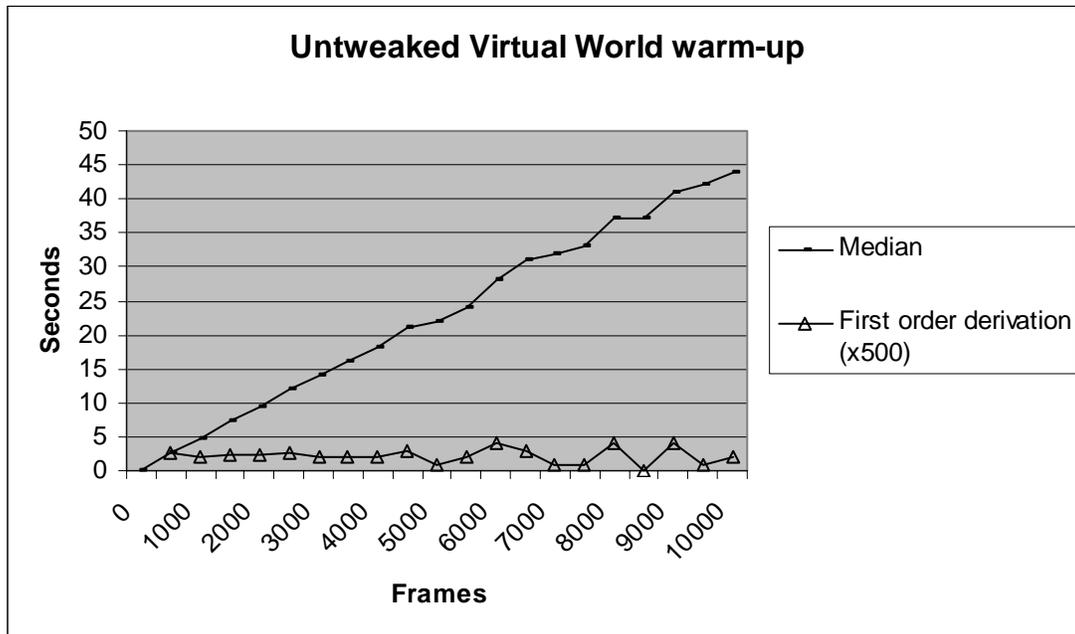
7.10.2 Excelsior Jet

For both the tweaked and untweaked versions of the GL4Java implementation and the Java3D implementation I was unable to get Excelsior Jet 2.1 to work.

This seems to be related with the fact that Excelsior Jet 2.1 only supports Java up to JDK 1.3 while both GL4Java and Java3D 1.3 beta 1 uses certain JDK 1.4 features if they are present. Whether this is actually the case, or whether it is something I have done wrong, I do not know.

This issue emphasizes, as discussed in section 6.3.5, that one should not rely on being able to use static native compilers for one's code. Static compilers are written from scratch by much smaller companies and have therefore not been as vigorously tested as the Sun virtual machine or derivatives.

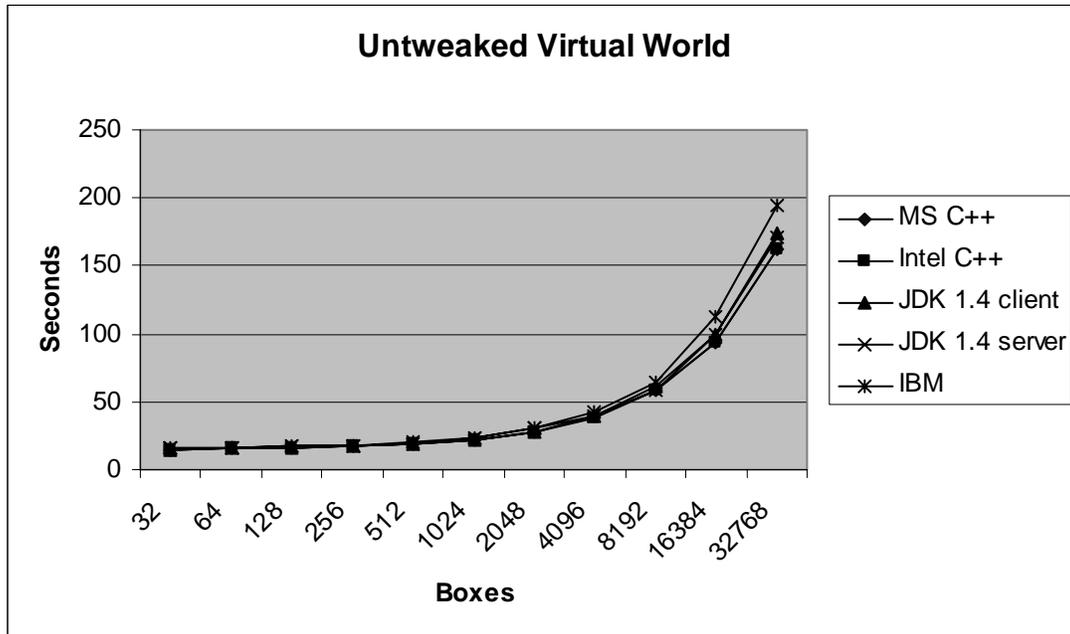
7.10.3 Untweaked version: Warm-up period



There is very little indication of any warm-up period in this program. This is expected since only very little of the execution time in a GL4Java program occurs in Java. Most of the time occurs in hardware. Just to be safe we will assume that the warm-up period is over after 10 seconds.

7.10.4 Untweaked version: Results

In this benchmark we keep the number of frames constants at 3000 and vary the amount of geometry. The x-axis is a logarithmic scale. This has been done so we can examine how the performance is at both very low and very high amounts of geometry.



Recall that the Java3D implementation is not tested here. That will only happen in the benchmark of the tweaked version in section 7.10.6 and 7.10.7.

The results above shows very little difference between the different languages and compilers/VMS. This is very likely due to the fact that most of the execution time is spent in 3D hardware.

This makes untweaked Java more interesting for 3D applications, because the overall performance consequence of using untweaked code will be minor.

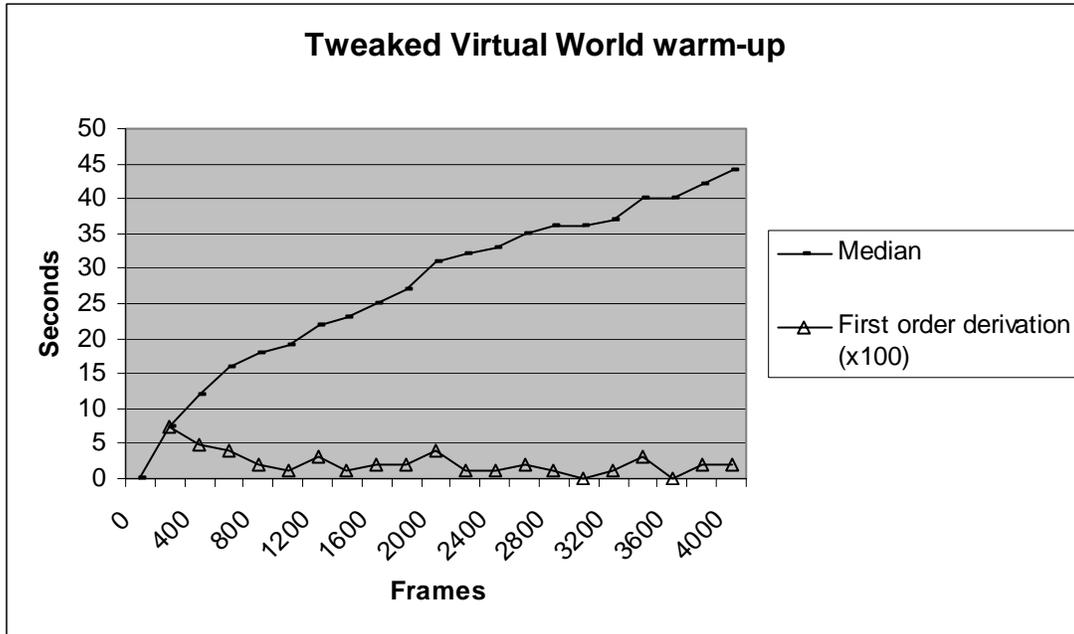
7.10.5 Tweaked version: Introduction

Again the main tweaking method I used was to eliminate all heap allocations in the Java program. The C++ had no heap allocations, so here was very little to improve.

In the GL4Java implementation, the tweaking of the Frustum class (`Frustum.java`) is a clear example of how tweaking Java code reduces readability. After tweaking, that class runs almost an order of magnitude faster, but it is almost impossible to read.

7.10.6 Tweaked version: Warm-up period

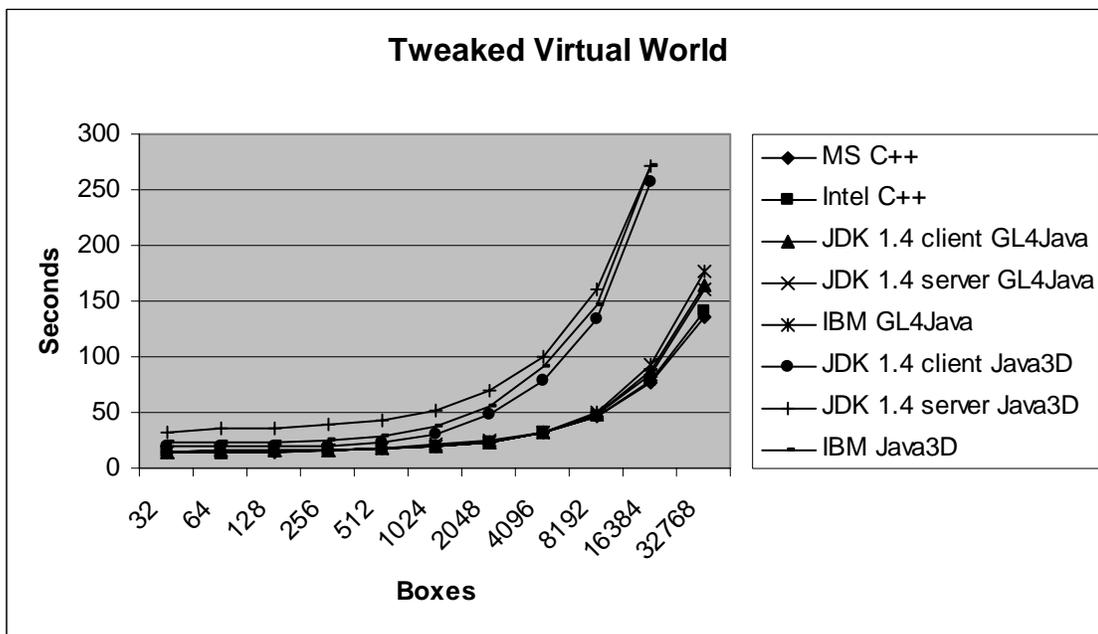
Since the Java3D implementation is the implementation with absolutely most code written in Java (most is within Java3D itself) I will use this version to measure the warm-up period.



After a clear warm-up period the graph becomes linear after about 15-20 seconds. Just to take care of any variation that may occur we will assume that the warm-up period is not fully over until after 25 seconds.

While making this test it was clear that system compiled the Java classes during the initial frames. This caused a lot of pauses during those initial 15 seconds after which the pauses disappeared.

7.10.7 Tweaked version: Results



In the chart above all the curves are clearly grouped into two distinct groups. Within each group the compilers/JVMs all perform almost identically, which make it hard to discern the individual curves in the chart.

The fastest versions are the two C++ versions. The GL4Java implementation is a little slower but they all stay within the same main group of curves. Much slower is the Java3D implementation of the demo.

The warm-up period is already over for the lowest number of boxes so the whole graph is valid. We can therefore use the absolute values in the graph above directly.

The Java3D version is about 2.5 times slower than the C++ and the GL4Java versions. This is surprising. I expected scene graph APIs to have an overhead, but a factor of 2.5 is *very* much. The overhead is especially severe since most of the execution time is supposed to be occurring in hardware. This means that the overhead of the Java3D API itself is many more times higher than the overhead of GL4Java.

I had expected that the overhead of Java3D was more or less a constant period of time, but according to the benchmark results Java3D does not scale any better than the other implementations. It remains 2.5 times slower than both GL4Java and C++ even when amount of geometry increases.

One should always take a benchmark with a grain of salt since it not completely represents a full application. In this case, however, the result is so clear that it cannot be ignored: I can only conclude that using Java3D is significantly slower than using GL4Java or C++ with OpenGL. This naturally assumes that the OpenGL application has been fully tweaked for performance, and this will become harder to do as the complexity of the application becomes higher.

7.11 Reviews of third party benchmarks

This report is not the first paper to contain benchmarks that compare Java and C++. Instead of just relying on my own benchmarks I will review some other benchmarks in this section. I will review their validity as fair benchmarks and summarize their results taking into account technology changes since they were originally published.

7.11.1 Evaluating benchmarks

When looking at Java benchmarks made by third parties one should always be skeptic since those benchmark might not always convey a result that is representative of real-world applications.

Here are some of the pitfalls that I have observed that third party benchmarks sometimes commit:

- The use of extremely short benchmarks, measuring the performance of single instructions, possible repeated many times to make time measuring possible. This is incorrect on any modern processor because of processor pipeline issues. The performance chapter (Appendix D) in the first edition of *Bruce Eckel: Thinking in*

Java [Eckel98] made this error. This chapter was, not surprisingly, removed in the second edition [Eckel00].

- Benchmarks that are run for very short periods of time – often just a few seconds. This way the warm-up period has not yet stopped and Java benchmark will look ridiculously poor. Running multiple Java benchmarks in the same VM session does not solve this problem but may give a bias towards runs that is run late in the session. Benchmarks should preferably measure the warm-up period explicitly and take it into account.
- Benchmarks sometimes give the impression that they use tweaked code, while the code could actually be tweaked more. So always check the code to see if it appears fair before believing in validity of a benchmark.
- Benchmarks that use random data but does not use repeated tests. These will often vary so much from run to run that single results cannot be trusted.
- Benchmarks using old versions of Java. These benchmarks were once relevant but are not any more. Since there have been significant advances in Java technology since the original release in 1995 it is vital to use the latest version of Java to make the comparison fair.
- Not using the best compiler options. This happens often when Java programmers want to compare C++ and Java. They neglect to turn on all optimization in C++. In Microsoft C++ this often happens because the standard setting for *maximize speed* does not aggressively inline code. The default setting will only inline functions and methods declared with the `inline` keyword. To make the inlining more aggressive one must include the `/Ob2` parameter.

7.11.2 Benchmarks by Ulrich Stern

Can be found at http://verify.stanford.edu/uli/java_cpp.html

This page contains a number of benchmarks in C++ and Java of:

- Various implementations of a generic bubble sort algorithm.
- The built-in in library sorting functions in Java and C++.

JDK 1.3 was used in the tests.

Although no clear conclusions are made in the paper it seems that the results say that C++ is about 4-8 times faster than Java for the bubble sort and about 30% faster for the library sort. The library sort is run for only a short while on the server VM and can therefore not be trusted.

For the sake of this review I downloaded the two programs *C++ pointer* and *Java array*. According to the benchmark results in the paper Java is 4 times slower than C++ for these two applications.

In the paper Microsoft C++ compares very poorly to gcc. This is not my experience. The author has probably used the wrong compiler options as discussed in section 7.11.1.

C++ pointer is a template version of a generic bubble sort and *Java array* is a generic bubble sort algorithm that works on an array. Due to the superiority of current C++ template implementations this causes specializations of the code to be made that are currently not done by the current Java compiler technology (although it might be added in the future). Furthermore, Java uses array bounds checking which C++ does not. It is therefore no surprise that the Java version is much slower than the C++ version.

If I had written the Java version to maximize for speed I had either written specialized versions for the sorting algorithm for each type of object to be sorted or I would have used a third party Java preprocessor to do the specialization.

For the sake of this review I have rewritten *Java array* in order to eliminate some unnecessary array accesses and have increased the number of elements to sort to 3000 in order to reduce the significance of the warm-up period.

Both *C++ pointer* and *Java array* was rewritten such that only the actual sorting, not the data generation, was timed. The results without explicit specialization were:

Microsoft C++	41.76 sec.
Intel C++	43.01 sec.
Sun JDK 1.4 server	104.88 sec.
IBM JDK 1.3.0 preview	113.78 sec.

If explicit specialization is added to the Java version so it now only can sort Range objects (this can be done generically by using parametric macros using a third party Java preprocessor) and reorganizing the code a bit to make it more Hotspot friendly the result is:

Sun JDK 1.4 server	61.48 sec.
--------------------	------------

This just shows us that current versions of Java is poor when it comes to generic algorithms, because it cannot generate specialized version of code like C++ templates can. However, if one is willing to use unorthodox means then much speed can be gained anyway.

So the conclusion that I derive from this benchmark is that Java is 1.5 – 2.5 slower than C++ depending on the amount of tweaking one is willing to perform on the Java code.

The edited source code can be found in Appendix H.

7.11.3 Benchmarks by Chris Rijk

Can be found at: http://www.aceshardware.com/Spades/read.php?article_id=153

This benchmark compares C and Java by running three applications:

- An array implementation of the Game of Life

- Calculation of Fibonacci numbers
- Fast Fourier Transforms.

When going over the used compiler options I noted that Microsoft C++ used the /O2 option but not the /Ob2 option. This causes inferior optimization and according to my experience usually a slow down of 10% or more. This should be taken into account when looking at the benchmark results. In the conclusion the author wonders why Microsoft C++ performs poorly in the tests. The missing optimization option is probably the reason.

The benchmarks are generally fair written, but run for only short periods of time and runs multiple benchmark in one session. This makes the right sides of the graphs in that paper more trustworthy than the left sides.

I have rerun some of the benchmarks provided with the paper and get results that are consistent with those given in the graphs; in fact in many cases my Java runs was relatively faster than those shown. This is probably due to the fact that I used Sun JDK 1.4 and the article uses Sun JDK 1.3. I also got better results for Microsoft C++ due to the added optimization option.

The general conclusion that one can derive from this benchmark suite is that Java is more or less the same speed as C. It is sometimes faster and sometimes slower.

7.11.4 Benchmarks by Osvaldo Pinali Doederlein

Is called the Java Performance report IV and can be found at:
<http://www.javalobby.org/fr/html/frm/javalobby/features/jpr/part4.html>

This report only contains limited discussion of Java vs. C benchmarks but mostly focuses on comparing Java regular virtual machines with Java static compilers. The C vs. Java comparisons that is present in the paper general state that Java and C are of the same speed. Microsoft C++ was used for the C code.

Of the static compilers Excelsior Jet generally seems to perform best, but in many cases there is little difference among them. This is among other things why Excelsior Jet was the static compiler of choice in this report.

7.11.5 Benchmarks by Dennis M. Sosnosky

Can be found at: <http://www.sosnoski.com/Java/Compare.html>

This set of benchmarks aims at measuring the performance of some very specific tasks.

The things measured are:

- Factoring of high numbers. Several factoring methods are used so this program uses many conditionals.
- File I/O.
- Memory exercising.

We will only review the first here in detail. Doing File I/O is mostly operating system dependant as is therefore not really interesting. The memory exercising benchmark is

heavily biased towards C++, because memory is only allocated once and never actually used. This causes Java to use much time to grow the heap from the initial 2 MB to size needed in the benchmark. I therefore do not consider this benchmark valid.

The first benchmark is, however, quite well written and should be trustworthy as a general Java vs. C++ benchmark.

The factoring benchmarks are run using only old versions of the Sun JDK and Microsoft Visual C++. I have recompiled both the Java and C++ and rerun the tests to see the results in a modern context.

`FactorsF` is the same program as `Factor` but uses `floats` instead of `ints` for the calculations.

Benchmark	Sun 1.4 server	Sun 1.4 client	IBM 1.3 preview	MS C++ 6	Intel C++ 5
<code>Factors</code>	139 sec.	485 sec.	121 sec.	118 sec.	112 sec.
<code>FactorsF</code>	76 sec.	342 sec.	72 sec.	69 sec.	63 sec.

Here the fastest Java is about 10% slower than the fastest C++. This is generally not surprising. However, it is surprising how good results the benchmark author got in his runs with JDK 1.1.8 and 1.2 compared to C++. I first suspected that the author did something wrong when compiling the C++ version, but when I tried the precompiled binaries that is provided with the benchmark I see that this is not the case. The only conclusion is - consistent with the author's own conclusion - that Hotspot compilation did not help at all with this benchmark.

7.11.6 Benchmark by Mads Pultz

Can be found at: <http://hjem.get2net.dk/mpultz/>

This benchmark compares the performance of matrix multiplication in C++ and Java. The application is available in multiple versions with varying degrees of tweaking. A beta version of JDK 1.4 was used in the test.

The benchmarks seem to well written. The only thing that I can criticize is that it, as so many other benchmarks, do not include the `/Ob2` parameter to Microsoft Visual C++.

The first versions of the benchmark shows Java to be approximately 40-50% slower than C++, but in later versions the C++ versions was improved more and the end result is that Java roughly 130% slower than C++.

7.12 Conclusions and thoughts

After running the benchmarks made for this report and reviewing the benchmarks made by other people we can conclude that:

- Code that is written in a straight forward way as intended in Java (as suggested by various programming books such as [Eckel00]) and C++ (as suggested in [Stroustrup]) will typically be significantly slower in Java than in C++. In our case it was factors varying from 2.5 to 4 (less in the case where most execution time occurred in 3D hardware). I expect this more or less represents the slow down that will be experienced in the major parts of any large application.
- Java code that is written to be tweaked for speed runs much faster. It ranges from being two times faster than C++ code to being two to three times slower. The tendency seems to be that Java is a little slower than C++, typically around 20-50% slower, but this varies much.
- When tweaking Java code the code loses much readability. Tweaking is therefore not something one would want to do for all of the code in a large application. If one did tweak everything, there would not be a productivity benefit by using Java.
- The IBM JVM and Excelsior Jet are more efficient at doing computations than Sun's Hotspot JVM. Sun's Hotspot JVM is much faster at heap allocation than the IBM JVM and Excelsior Jet. One should choose the one that suits the application in question best.
- Java in general has a lot of problems competing with C++ when it comes to generic code. C++ programs can use templates to generate specialized versions of the code. In principle Java could do the same, but the current virtual machines do not implement this optimization.
- Java programs run slowly in the beginning. This can be annoying for 3D applications, where this usually will result in a number of short pauses during the first frames to be rendered.

Given the big difference between the performance of untweaked vs. tweaked Java and because tweaking Java code tends to be harder than tweaking C++ code, it is important to avoid that the influence of the tweaked code spreads to the untweaked parts of the program thereby decreasing the ease of maintaining the rest of the application. To prevent this, it is important, during the design of the application, to try to determine which parts of the program that will have the highest throughput and then make the design such that these parts are isolated cleanly from the rest of program through well-defined interfaces. This will allow the programmer to tweak those parts without affecting the rest of the application.

8 Evaluating Java for a game project

In this chapter we will bring together the information from the previous chapters and discuss how and why Java can be used for games, and consider whether it is advantageous to do so.

8.1 Why use Java?

Before you begin to use a new system you should always ask, why this system is better than the one currently used. For Java the answer is that you will increase programmer productivity.

There is absolutely no doubt that programmers are more productive in Java than in C++. The reasons for this are many and they were discussed in chapter 5.

The improved ease of use and increased programmer productivity was an important goal in the design of Java, if not the main goal, from the beginning and has been an influencing factor ever since. [Gosling96]

Besides these reasons an added benefit that stems from the advantages presented in chapter 5 is that Java is easier to learn. This is of course no benefit to programmers who already know C++, but is a clear advantage when you need to hire new people or when you want the some of non-programmers on the team to do some of the programming. C++ is notoriously hard to learn for beginners, and is therefore not something game content providers usually are willing to learn. This is not so for Java.

You may think that the advantages of Java may seem very few, and if you are very productive in C++ you may think that you can live without them. I also thought so before I learnt how to use Java several years ago. I was surprised how much faster development in Java really is.

In May 1998 IDC published a large study: *Java Technology Pays Positively* by Evan Quinn & Chris Christiansen, IDC Bulletin #W16212. Unfortunately, I have not been able get access to this report first hand and have instead relied on second-hand accounts about it. [Byous] [Wells]

According the IDC study software projects written in 100% pure Java instead C++ yield a 25% overall time/cost saving cost! This corresponds to a productivity increase of 30%. This is quite a significant number.

The saving was 40% in the coding phase and 30% in the maintenance phase. This corresponds to productivity increases of approximately 67% and 42% respectively.

It is not clear how big the increase productiveness is for games, but it is a fact that Java increases your productiveness both by increasing code writing speed but also by reducing the number of bugs. This I can confirm from personal experience. The mere fact that so many companies are changing to Java for business development strengthens this argument.

The results in the IDC study assume that the project use 100% pure Java. This is often not the case with games. On the other hand, the Java has had some significant improvements since those project was made (They used JDK 1.0.2) so this should more or less compensate for this fact.

It should also be noted that the IDC report analyzed projects where the programmers were new to Java. When the programmers get more skilled the productivity should increase even more.

Since I do not have access to the actual report, I cannot know the exact context in which this numbers was calculated. I will, however, assume that the applications have been tweaked somewhat for performance. If they were not, the applications would have been extremely slow back then.

All in all I would say that the numbers from the IDC seems reasonable and can be applied to games too.

I have heard several Java developers claiming productivity increases from a factor of 2 to 10. To me these numbers seem exaggerated, except for certain isolated sections of the code, such as, say, network programming.

There are a number of other articles that compares C++ and Java developer productivity. They all find that Java gives higher developer productivity than C++. [Singal] [Tyma] [Phipps].

As time goes by, productivity tends to mean more and more. You can probably remember the time when nobody wanted to make Windows programs because DOS was in full control of the machine and that was the way to do it. I remember reading a magazine interview with a game developer in those times where it was stated that DOS would always prevail because games needed all the speed they could get. Even further back in time everybody used assembler and didn't like the idea of using C. It was too slow for games, they said. Then in 1993 ID Software released *Doom*. *Doom* was written almost solely in C, containing less than 5% assembler. That was a turning point for game developers who until then had written much code in assembler. Today no sane developer would use assembler for a full commercial game. In fact, at QuakeCon 2001, John Carmack (one of head programmers for ID software) spoke about the importance of high level tools and languages in development today. He said that Java may very well be the de facto language for game development in the future. That John Carmack does consider Java a good language is clear from this quote:

"We are still working with significant chunks of an existing code base. If I did want to go off and start fresh, I would likely try doing almost everything in Java." [Kreimeier]

8.2 Considering the risks

Before choosing to rely ones business on a new system, one must always be careful that the does not have some fatal flaw. Common flaws of apparent brilliant systems are that they are very buggy or that no proper documentation is present. These and other potential risks will be discussed in this section.

8.2.1 Reliability and stability

It is generally risky to use experimental systems or systems with only few users. Fortunately Java is no longer experimental or has few users.

There are extremely few bugs in Java – fewer than in most C++ compilers – and it has been used for half a decade by millions of people. In fact Java has a strong reputation for being stable and reliable.

This statement about high stability and reliability generally only applies to the Sun and IBM implementations of the Java compilers and virtual machines. Other implementations, including the Java static compilers are often less stable and reliable.

8.2.2 Documentation

It is often the case that experimental and open source systems are poorly documented making it hard to use them.

This is absolutely not the case with Java. Much very well written documentation exists for Java, including tutorials, actual manuals, specifications, articles and books. Most of documentation provided by Sun is freely available on-line so it is easy to get access to needed information.

8.2.3 Continued support

Another problem with new systems is that they are not properly supported and/or not actively developed.

Neither is true for Java.

The support for Java is very good:

- Both Sun and third party consulting company support is easy to get for the commercial customer.
- Large communities exist where Java programmers exchange information. These are quite good and Sun engineers are often involved in many of them.

Java is still very actively developed. Sun, IBM, and several other large corporations work very actively on Java and release dozens of new versions of Java related products every month. A new major release of the Java Development Kit is made approximately every 18 months.

Sun Microsystems state that it is goal for them to make Java as good as possible for game development. Mere words are of course not enough, but we have already seen a lot of game related improvements in Java over the last few years. This makes it seem that they are actually keeping their promises, although progress is slower than many had hoped from the outset.

At the Game Developers Conference 2001 it was announced in a Sun press release [Shuster] that Sony and Sega, among others, will collaborate to build a Java gaming API. Sony and Sega are very influential when it comes to games and their presence in the scene promises a bright future for Java gaming. This Java gaming API (called the *The*

Java Game Profile [JSR 134]) is being made primarily to allow games to be written for game consoles and other related game devices, at least partially, in Java. At the time of writing, the Java Game Profile is neither fully specified nor implemented.

In that same press release Sun also announced the launching of an official Java game development website: www.javagaming.org. Read [Melissinos] for more information about the Sun gaming initiative.

8.2.4 Migration costs

Some systems are very expensive to purchase or use.

This is certainly not the case with Java. Virtually all Java related products needed during the development of a game made in Java is available freely for download. This includes great development environments (such as Eclipse), the compilers, the virtual machines themselves and the rights to distribute the virtual machines with the game. None of these require any royalty payment.

Certain specialized tools may have to be bought separately such a profilers, automated testing suites and server application development environments. Free tools in these areas also exist, but one might benefit from the commercial ones.

In summary, the cost of per developer seat for Java developers is usually much lower than that for a C++ developer. In fact it is often completely free with regard to software.

8.2.5 Legacy code

If you already have some code written in C or C++ it is vital that this can be used.

This is possible, but the complexity of doing so heavily depends on the complexity of the interface of the C/C++ code. The more complex the interface is the more work is needed to interface it from Java.

If the interface to the C/C++ code is very complex, and that C/C++ code functionality is vital it may not pay to use Java. As mentioned in section 8.1 this is why ID software does not use Java. For more information about library wrapping see section 8.5.

8.2.6 Code security

One drawback of the Java byte code model is that it is quite easy to decompile. Byte code can often be decompiled to source code in a way that looks very much like the original including variable and method names. This is useful in many tools, but also provides a threat to source code security.

When Java source code is compiled to byte code, a large amount of information is included, including information about variable names, method names and code structure. This is needed, for instance, by the Java reflection mechanism.

This has discouraged some developers from using Java since they fear that any code they have written may be stolen.

Fortunately good solutions exist:

- You can use a **Java obfuscator**. The purpose of this tool is to mangle the byte code so it cannot be decompiled without reducing performance. It usually also reduces the size of the byte code files which is useful if the application is to be downloaded on the Internet.
- You can use a Java static compiler. These compile the application into machine code and are therefore much harder to decompile.

In other words, code security is not a practical problem.

8.2.7 Deployment

For some systems that use virtual machines or other platform specific libraries, deployment can be complicated. For instance, games on Microsoft Windows often need DirectX of a certain version number or higher to be present. The user will then have to install this first if needed.

Some new Java developers are often concerned that Java applications in most tutorials are started via the virtual machine (e.g. the application is an argument to `java .exe`). This makes it hard to start the application for end-users, who are used to that they just click on an icon or executable file.

However, these problems can easily be avoided in one of several ways:

- If the Java game is to be a regular game application as is usually known from retail it is highly recommended simply to bundle a Java virtual machine with the application, install it as part of the regular game installation (e.g. using InstallShield), and use a launcher program, such as the *Marner Java Launcher*, to provide an executable file to start it. This way the user will never know Java was used and will not have to worry about it. Furthermore, you are guaranteed that a virtual machine is present on the computer that it is 100% compatible with your application.
- Use a Java static compiler. These package the application into an exe file for the given platform making its use identical to other applications on the platform.
- If you want to avoid including a full virtual machine in the deployment (to make a single distribution more portable or just to reduce the size of the download) you can use *Java Webstart* or *Zero G InstallAnywhere*. These allow the same distribution to be used on Windows, Mac and Linux. However, using these approaches will be more complex for the end-user and can only be recommended if absolutely necessary.

I generally recommend the first option. This is available for all non-browser based games, is fully reliable and requires no user intervention. The only drawback is that the distribution becomes about 6 MB larger since a virtual machine must be included.

Using one of these options, deployment of Java applications is no longer a problem.

8.3 OpenGL, DirectX and other libraries

It is a common misconception about Java that it cannot use all sorts of 3rd party libraries such as DirectX, OpenGL or the Microsoft Windows SDK.

In fact using JNI any library written for use with C or C++ can be used from Java and DirectX, OpenGL, Windows or any other library is no exception. This means that there is *no limitation* about what system features you have access to from Java.

A good open source OpenGL wrapper for Java (usually called OpenGL Java bindings) is available and is called *GLJava* from *Jausoft* supporting OpenGL 1.3 and many of the latest vendor specific extensions.

Currently no full and proper DirectX or Windows wrappers are available, so the normal approach is simply to provide wrappers for those functions needed.

Microsoft has written a library they call the *Microsoft SDK for Java*. This allows the user to write full Windows application in Java. It even supports DirectX 3.0. Unfortunately DirectX 3.0 is old and the Microsoft SDK for Java only works on Microsoft's own Java virtual machine because it uses special features not in the official Java standard.

I have personally written an open source wrapper, called *JWindows*, for a subset of the *Microsoft Foundation Classes (MFC)*, itself an object-oriented wrapper around the win32 API. It allows the construction of full Microsoft Windows applications in Java, completely circumventing the Java Windows system. It works on any Java 2 compliant virtual machine. For an instruction to JWindows and more information about wrapping C++ class hierarchies in Java please refer to [Marner00] and [Marner01b].

I have also written an open source wrapper around the DirectX based 3D engine called *6DX* from *Eldermage Entertainment, Inc.* The wrapper, called *6DX for Java*, is somewhat out of date today, but it still proves that a DirectX based 3D engine can be used from Java.

Another popular library is *WildTangent*. WildTangent is a 3D engine for Java (and other languages) wrapping an improved version of the original Genesis3D engine. It runs very impressively. For a full review of WildTangent I recommend [Meigs]

8.4 Java3D

Java3D is an optional package to Java that provides the Java programmer with support for 3D graphics. It is therefore natural to ask if this is suitable for use with games.

Java3D is a Java library written using JNI to access OpenGL or DirectX implementations provided by the underlying operating system. Official ports currently exist for Windows and Sun Solaris. Ports exist for IBM AIX, HP-UX, Linux, and SGI IRIX. A Mac port is supposedly in development, but I have been unable to get this confirmed.

The OpenGL implementation of Java3D is generally more bug free and faster than the DirectX implementation.

Java3D is a scene graph API. This means that the programmer organizes the 3D scene in a tree structure and then gives the control of the actual rendering to library. Since most OpenGL and DirectX application usually contain some sort of scene graph anyway, such a scheme is inherently no slower than the underlying API.

There are several advantages of using Java3D over lower level APIs:

- Significantly reduced development costs. Since much code already has been written it becomes much easier to make a new 3D application and let the library handle the optimizations of the scene graph. This becomes more relevant as the scene grows.
- Built-in support of multiprocessing. Java3D is multithreaded internally in order to scale better to multiprocessor systems.
- Built-in support for VR-helmets and other VR related displays such as CAVES or stereo-vision.
- The code becomes 100% portable between DirectX and OpenGL on multiple platforms.

Unfortunately there are also several drawbacks of using Java3D:

- Significantly reduced performance compared to highly tuned OpenGL or DirectX applications. In the 3D Demo benchmark (see section 7.10) we saw that the Java3D version (Java3D 1.3 beta 1 for OpenGL was used) was about 2.5 times slower than the equivalent GL4Java version. This is a significant slow down since most of the processing time is supposed to be in 3D hardware. Any benchmark should always be taken with a grain of salt, but the result is so clear in this case that it cannot be ignored. The benchmark comparison assumes that the OpenGL application has been organized and tweaked fully for performance. This gets harder to do as the scene gets more complex so it is likely that Java3D compares better as this happens.
- Java3D has been made as a common denominator in 3D graphics. This means that many of the latest 3D features are not available to Java3D programmers. Currently only the features in OpenGL 1.1 and earlier are supported. Vertex and pixel shaders and other nVidia Geforce 3 features are supposedly planned for Java3D v1.4 but this is not scheduled to be released until the spring of 2004 at the earliest.
- It often happens that 3D hardware drivers for either DirectX or OpenGL contain bugs. When writing 3D applications it is common practice to work around these problems if they should arise. This ensures that it is possible to make the game work for as many users as possible. Java3D, on the other hand, is written assuming that the drivers are fully standard compliant. This means that user support has no option but tell the user to wait until a new driver is released for his 3D card - which may take a long time. Fortunately, nVidia and ATI both has helped to raise the quality of the hardware drivers available, so this is not the problem it once was.

If you evaluate Java3D yourself by running some of the demos available with the library please let each demo run for 20-30 seconds or so before judging its performance. This is the time it takes to warm-up the library on an AMD Athlon 700 MHz.

Roboforge from *Liquid Edge Games Ltd.* is an example of a commercial game using Java3D. You can get a good indication of the capabilities of the library by trying that game.

Given the poor performance of Java3D I can not recommend the use of Java3D for games with high requirements in performance. However, games without this requirement will benefit much from the reduced development costs of using Java3D. Such projects would probably not have the resources anyway to workaround driver bugs or to keep the performance up using one of the lower level APIs such as OpenGL.

8.5 Different ways Java can be used in games

When Java is to be used for games there is basically three approaches one can use (we ignore Java applets). These are:

- Pure Java application, possibly using Java2D and/or Java3D. These games are the most portable and the ones that benefit the most from the high Java programmer productivity, but are also limited when it comes to system access and performance.
- Mixed code Java applications (dirty Java). These mix C++ and Java as needed for the application. For instance, the 3D engine is written in C++ and all the control code is placed in Java. This approach is discussed in depth in [Kreimeier].
- Java as a scripting engine. This is basically games written in C++ that use Java as the general purpose scripting engine. Java is much faster than any custom scripting engine you could write yourself within the project period and it support dynamic class loading and (with a few tricks) dynamic class *reloading*. Dynamic class loading allows one to edit the scripts without having to recompile the whole system – this is useful for content control code. Dynamic class reloading allows content to be changed without even stopping the game engine from running.

Which one to choose and why is discussed in section 8.9.

8.6 Commercial games using Java

An important question when evaluating Java for use in a new project is whether it has been tried before and if so, how it turned out.

The best examples of Java in commercial games are games that use dirty Java, i.e. Java mixed with C++ or that use Java as a scripting engine. As time has passed, Java has improved, and as will be discussed in section 8.9 these two approaches are no longer the only viable way of using Java.

The list is not long; if it was there would not be much point in writing this report. I have not included any applets in the list on purpose. Java game applets exist in abundance so there is no doubt that Java can be used in that way.

8.6.1 Java as a scripting engine

Probably the best example of Java in games is the highly acclaimed game *Vampire - The Masquerade: Redemption* (2000) from Nihilistic software. This game uses Java JDK 1.1 as its scripting engine. The rest is written in C++. This game runs with very high performance, has excellent graphics and gotten fine reviews.

According to [Huebner], Nihilistic Software was very satisfied with the use of Java.

8.6.2 Mixed code Java (dirty java)

Worthy of note in this category is *Who wants to be a Millionaire* (2000) developed by Jellyvision and based on the popular TV-show of the same name. It topped the game sales charts for months. Here the game logic was written in Java and the rest in C++.

Of other commercial games using putting the game logic in Java and the rest in C++ can be mentioned *You don't know Jack* (1995) by Jellyvision and *Majestic* (2001) by Electronic Arts.

Red Storm Entertainment tried mixing some of the Java code into their product *Tom Clancy's Rainbow Six* (1999), but according to [Upton] this turned out not to be an advantage because the complexity of mixing Java and C++ in this case turned out to be too high.

8.6.3 Almost pure Java

The commercial full-price game *Roboforge* (2001) by In-house is made in Java using Java3D. Roboforge uses native code only to switch to full screen mode. This functionality has since then been added to the Java core API in JDK 1.4.

8.6.4 Pure Java applications

Commercial games written in pure Java include *Shadow Watch* (2000), *Tom Clancy's ruthless.com* (1998), and *Tom Clancy's Politika* (1997), all from Red Storm Entertainment; none of them got very good reviews.

8.6.5 Console games

As we will discuss in section 8.7 it can be problematic to use Java for console games. Nonetheless, according to Chris Melissinos (of Sun Microsystems) the Sega Dreamcast games *Skies of Arcadia* (2000) and *Daytona USA* (2001) both by Sega includes a Java virtual machine. Especially *Skies of Arcadia* got fine reviews.

8.6.6 Demo games

On QuakeCon 2001 *Fullsail Real World Entertainment* showed a Quake clone *Jamid* that they had made. January 2002 Fullsail also made available for viewing the *Java3D F1 Gran Prix Demo*. Both were written in Java using Java3D. Unfortunately these two games are not currently available for public viewing or use.

8.7 Portability

One of the arguments for using Java usually is that programs written in Java are fully portable. As discussed in section 5.3 this is truer for Java than for most languages, but as we will see in the following sections portability is not automatic.

8.7.1 Desktop and server computers

Automatic portability only holds true under the circumstance that the application is written in 100% pure Java using the Java core API only.

Furthermore, if one chooses to use new versions of Java, ports are only immediately available for Windows, Sun Solaris and Linux. On the other platforms they only become available when the respective vendors complete their ports. This often means that if you want to target as broad an audience as possible then you might be better off targeting a bit older version of Java.

If the Java program uses any native libraries, including any of Java optional packages such as Java3D, the portability of the program is limited to the platforms for which ports of those libraries exist. This must be so; native libraries are written in C or C++ and Java can provide no guarantees that these are fully portable.

That said, it is often easy to port various native libraries to multiple platforms if just they were well designed from the start, especially if the C/C++ code already was written to be somewhat portable. If, for instance, you write your own Java wrapper for OpenGL then the library code should easily be able to be ported to any platform that supports OpenGL.

In practice this means that if one plans a little ahead it is quite easily possible to write applications that with a minimal effort can be ported between Microsoft Windows, Mac, Linux and several other systems.

8.7.2 Dedicated game consoles

The only dedicated game console, for which a regular Java virtual machine is available, is the Sega Dreamcast. According to [Patrizio] this is a version of PersonalJava, a technology related to the Java micro edition for use on handheld computers and other mobile devices.

As mentioned in section 8.6.5 this has been used to make a few commercial games on that platform.

For all the major platforms, including the Microsoft Xbox, Sony Playstation and the Nintendo game consoles no virtual machine is available.

This does not, however, make it impossible to run Java applications on these machines. On most of them a port of gcc is available. This means that it is possible to use the Java gcc front-end gcj. Although it in theory should be possible to write applications for the consoles using gcj I have been unable to find anybody that has actually done or even attempted to do so.

gcj is an open source effort and in many areas it still appears to be experimental. I can therefore not recommend relying on it for production code.

So except if one develops only for the Sega Dreamcast (which is no longer in production), we must conclude that relying on Java for console development is a risky affair that cannot be recommended.

At the Game Developers Conference 2001, Rob Huebner of Nihilistic software (see section 8.6.1) said that they would not use Java for *Vampire: The Masquerade 2*. When asked why, he answered that they had no choice because the *Vampire: The Masquerade 2* was to be released on game consoles as well as on desktop computers.

It actually seems quite ironic that it is portability that causes the main problem for Java for use in game development.

It appears that both Sun and Sony are aware of this problem and this explains their effort with the Java Game Profile [JSR 134]. As mentioned in section 8.2.3 the Java Game Profile is a profile for the Java Micro Edition that is to be used to allow Java be used on game consoles. Since Sony is part of this effort we will very likely see this implemented on the Sony Playstation 2.

In fact, according to [Kayl] the Java platform will be integrated with the Sony Playstation 2 by the end of 2001. This has yet to come to pass but it is likely that it will happen at some point. The motivation mentioned in [Kayl] seems mainly to make it easier to write on-line games for consoles, but given the current proposal for The Game Profile, it seems that The Java Game Profile will have a broader appeal than just networks.

At the time of writing the Java Game Profile specification is not yet complete and the developers are very reluctant to give out details. So it will probably be at least a year before we will see the first implementation of it.

Microsoft is no longer actively supporting Java (Java is not even included with Internet Explorer in Windows XP) so it is very likely that implementations of the Java Game Profile never will appear on the Xbox.

Since The Java Game Profile has a separate API you should not expect that games written in Java for desktop computer easily can be ported to the consoles when the first implementations of The Game Profile appear.

To summarize; Java can currently not be used on consoles. The Java Game Profile is likely to fix that for the Sony Playstation 2 within a couple of years and probably later appear on the Nintendo consoles. The Game Profile will probably never be implemented on the Microsoft Xbox.

8.8 The performance of Java in games

Based on the discussions in chapter 6 I will here try to guess the performance consequence of using Java in games in various circumstances.

The time used in a game is the sum of three components: The time used in 3D hardware and the 3D API (for 3D games only), the time used in tweaked code, and the time used in untweaked code.

Tweaked Java code is usually 20-50% slower than tweaked C++ code but varies even more in practice. For the sake of the calculation in this section I will assume that it is 35%.

Untweaked Java code is usually 2.5 to 4 times slower than untweaked C++ code. In the calculations here I will assume it is 3.25.

I will assume that 40% of the time is used in 3D hardware in C++ and this time is increased with a factor of 2.5 if Java3D is used. Of the remaining time 10% of the time is used in untweaked code and 90% in tweaked code.

Scheme used	Calculation	Factor slower than C++ only.
C++ only (no 3D)	$0.9 * 1 + 0.1 * 1$	1.0 (0% slower)
Pure Java (no 3D)	$0.9 * 1.35 + 0.1 * 3.25$	1.54 (54% slower)
Mixed Java/C++ (no 3D)	$0.9 * 1 + 0.1 * 3.25$	1.225 (22.5% slower)
C++ only (with 3D)	$0.4 * 1 + 0.54 * 1 + 0.06 * 1$	1.0 (0% slower)
Pure Java (with Java3D)	$0.4 * 2.5 + 0.54 * 1.35 + 0.006 * 3.25$	1.924 (92.4% slower)
Mixed Java/C++ (with 3D)	$0.4 * 1 + 0.54 * 1.35 + 0.06 * 3.25$	1.32 (32% slower)

The mixed Java/C++ calculations assume that the most often used code is written in C++ and the rest in Java.

These numbers should not be taken too seriously, since the exact numbers will depend on a lot of issues and the many assumptions listed above. They should just be seen as guesses that apply to what seems to be a typical project. The purpose is to help you to choose the best solution of your project.

8.9 Should I use Java for my next game project?

This depends on the project. It is all about choosing the right tool for the right job. However, choosing C++ just because that is what game developers are used to do is no valid argument. But I fully understand that companies hesitate to try out new technologies. C++ has proven itself over the years and it therefore a safe bet.

As we saw in section 8.1 Java provides a clear benefit to productivity, so using Java will almost always be a benefit to a game project, unless some project specific issue

causes Java prevents it from being used. Also, as it was summarized in section 8.8 the use of Java is very likely to have a negative impact on performance.

The only issue that currently prevents the use of any Java at all is that the game needs to be ported to game consoles. The Java Game Profile may change this in the future, but currently, porting Java games to consoles is not feasible to do.

Issues that make *pure* Java (including Java3D) unsuitable for a project:

- The game relies of high performance to ensure sales. This is often the case with certain genres of games, for instance, first person shooters.
- You already have a large amount of legacy code written in C or C++ that you wish to reuse in the new project.

If this is the case you should consider whether it is feasible to mix C/C++ and Java code together in one program. Using Java as a scripting language is just a variant of this approach. This will allow you to combine the high productivity of Java with the high speed of C++.

The most common approach is to use C++ for the low level functionality of the game that is performance critical and which makes much system access (such as the 3D engine) and then write the rest in Java. This approach can be used even if the overall performance of the game is important since the overall performance impact of using Java in this case will be minimal (see section 8.8).

This way you can reuse any legacy code you may already have in C/C++. Assuming that the legacy code has a clear interface it will be easy to interface from Java. If your legacy code consists of C++ class hierarchies (not simple C-like calls) this interfacing gets much more complex and will probably not be worth the effort. In that case you are probably better of either ditching the legacy code or Java completely.

If you decide to use Java in your project you should what approach to take by considering your needs:

First consider building the game in pure Java (including any optional packages) since this will give the highest productivity gain. This is feasible for various strategy games or other games that do not rely on very high performance graphics. Using pure Java has the added benefit of being cross platform which is good if you later need to port it to, say, Mac OS or Linux.

If pure Java is not suitable for your project you should next consider mixing Java and C++. This can be done as simple as using the GL4Java library (which uses both Java and C++) and then writing all of your application code in Java. Or you can choose to write the performance critical sections of your code in C++ and the rest in Java. The game logic is usually those parts of the code that is most buggy so these will benefit much from being written in Java.

9 Conclusion

In contrary to popular belief, Java applications are in fact not much slower than C++ applications when they have been tuned for performance. A rough estimate based on the various benchmarks would say that tweaked Java code is a little slower than C++; typically 20-50% on the average, but this is hard to tell for certain because of the large variations in the speed seen in the benchmarks. The slowdown is less in 3D applications, where performance mostly depends on the performance of the 3D hardware and not on the programming language used.

For untweaked code, Java is much slower than C++, often a factor of three or four. This makes it vital to tweak the performance critical sections of Java code.

Java increases the overall productivity of a software project with about 30% and the productivity of the code phase with about 65%. This is quite a significant increase.

This productivity increase makes Java a good choice for low-profile games and for high profile games in genres that do not rely on maximum performance to ensure sales. It is especially good for low-profile games since the cost of Java tools and libraries are significantly lower than those for C++. For high profile games that do not need maximum performance, the use of Java will increase productivity and thereby allow a better game to be produced for the same money.

Because of the relatively lower performance of Java, I cannot recommend pure Java for highly performance critical games. It is, however, still recommended that Java is used for at least parts of the game, for instance, as a general purpose scripting language for the control code. Java runs much more efficient than the vast majority custom scripting languages available.

The biggest problem at the moment with using Java for game development is that proper ports do not exist for game consoles. I consider the current ones highly risky. If the game is to be released on consoles then the use of Java for any part of the game cannot be recommended.

With the development of the Java Game Profile, the rapidly improving development tools for Java, and the still improving speed of Java virtual machines, it is likely that the viability of Java for game development will improve in the future. However, as it stands now, Java can only be used for games that do not need to be ported to consoles and only partially for games that rely on maximum performance to ensure sales.

Appendix A: References

Note: The Internet is an evolving place, so many of the provided URLs will likely stop working within few months after this report has been published. If this happens, try finding the referred pages elsewhere by searching for them on Google.

[Assarsson] Assarsson, U. & Möller, T.: Optimized View Frustum Culling Algorithms for Bounding Boxes. Department of Computer Engineering. Chalmers University of Technology, Journal of graphics tools, 5(1):9-22, 2000. Available on-line at: http://www.ce.chalmers.se/staff/uffe/Vfc_Bbox.pdf

[Assert] The JDK 1.4 documentation: Assertion Facility. 2001-2002. Available only on-line at: <http://java.sun.com/j2se/1.4/docs/guide/lang/assert.html>

[Berg] Berg, M. de, Kreveld, M. van, Overmars, M. & Schwarzkopf, O.: Computational Geometry. Algorithms and Applications. Second Edition. 2000. Springer-Verlag.

[Byous] Byous, J.: A Jolt of Efficiency. 1998. Available only on-line at: <http://java.sun.com/features/1998/07/efficiency.html>

[Coulouris] Coulouris, G., Dollimore, J., Kindberg, T.: Distributed Systems – Concepts and Design. 3rd Edition. Addison-Wesley. 2001.

[DFC 1] DFC Intelligence: The state of the PC game market 1999. Available only on-line at: http://www.dfcint.com/game_article/statepcmarket99.html

[DFC 2] DFC Intelligence: Top Ten Myths of the Interactive Entertainment Industry. Excerpt from U.D. Market for Video Games & Interactive Electronic Entertainment. Available on-line at: http://www.dfcint.com/game_article/myths.html

[Eckel98] Eckel, B: Thinking in Java. 1st Edition. 1998. Prentice-Hall. Available on-line at <http://www.mindview.net/Books/TIJ/javabook.html>.

[Eckel00] Eckel, B: Thinking in Java. 2nd Edition. 2000. Prentice-Hall. Available on-line at <http://www.mindview.net/Books/TIJ/>

[Fowler] Fowler, M, Beck, K., Brant, J, Opdyke, W. & Roberts, D.: Refactoring – Improving the Design of Existing Code. 1999. Addison-Wesley.

[Galli] Ghalli, P.: Study - Java to overtake C/C++ in 2002. Available only on-line at: <http://zdnet.com.com/2100-1104-503983.html?legacy=zdn&chkpt=zdhnews01>

[Gamma] Gamma, E, Helm, R, Johnson, R. & Vlissides, J. : Design Patterns – Elements of Reusable Object-Oriented Software. 1994. Addison-Wesley.

[Gardner70] Gardner, M: Mathematical Games - The fantastic combinations of John Conway's new solitaire game "life". Scientific American. October 1970. p. 120-123.

[Gardner71] Gardner, M: Mathematical Games - On cellular automata, self-reproduction, the Garden of Eden and the game "life". Scientific American. February 1971. p. 112-117.

- [Gosling00] Gosling, J, Joy, B., Steele, G. & Bracha, G.: The Java Language Specification. 2nd Edition. 2000. Addison-Wesley. Available on-line at: <http://java.sun.com/docs/books/jls/index.html>
- [Gosling96] Gosling, J. : The Java language – An overview. Available on-line at <http://java.sun.com/docs/white/>
- [Hotspot] The Java Hotspot Performance Engine Architecture. White Paper. Available only on-line at: <http://java.sun.com/products/hotspot/whitepaper.html>
- [Huebner] Huebner, R.: Postmortem of Nihilistic Software’s Vampire: The Masquerade – Redemption. Game Developer Magazine. July 2000. Also available on-line at: http://www.gamasutra.com/features/20000802/huebner_01.htm
- [Hutcherson] Hutcherson, R. : The Evolution of Performance on the Java Platform “A Panel Discussion”. JavaOne 2001. Available on-line at: <http://java.sun.com/javaone/javaone2001/pdfs/2647.pdf>
- [JSR 134] Java Specification Requests (JSR) 134: The Java Game Profile. 2001. Available only on-line at: <http://jcp.org/jsr/detail/134.jsp>
- [JSR 14] Java Specification Request (JSR) 14: Add Generic Types to the Java programming language. 1999-2001. Available only on-line at: <http://jcp.org/jsr/detail/014.jsp>
- [Kayl] Kayl, K.: PLAYING TO MILLIONS - Sony Computer Entertainment Integrates Java™ Technology Capabilities into PlayStation 2. 2001. Available only on-line at: <http://java.sun.com/features/2001/06/sony.html?frontpage-headlinesfeatures>
- [Kernighan] Kernighan, B.W. & Ritchie D.M.: The C Programming Language. 2nd Edition. Prentice Hall. 1988.
- [Kreimeier] Kreimeier, B.: Dirty Java – Using the Java Native Interface within Games. Game Developer Magazine. July 1999. Available on-line at: http://www.gamasutra.com/features/19990611/java_01.htm
- [Lindholm] Lindholm, T. & Yellin, F.: The Java Virtual Machine Specification. 2nd Edition. 1999. Addison-Wesley. Available on-line at: <http://java.sun.com/docs/books/vmspec/>
- [Lippman] Lippman, S.B.: C++ Primer. 2nd Edition. 1991. Addison-Wesley.
- [Marner00] Marner, J.: Providing Java wrapper classes for C++ class hierarchies. 2000. Available on-line at <http://www.rolemaker.dk/articles/Wrappers/Report.zip>
- [Marner01] Marner, J.: Role Maker – A Computer Roleplaying Game Authoring System. 2001. Available only on-line at: <http://www.rolemaker.dk/>
- [Marner01b] Marner, J.: How to write a Java wrapper for a C++ class library. 2001. Available on-line at: <http://www.rolemaker.dk/articles/Wrappers/JavaWrapperHowTo.zip>
- [Meigs] Meigs, T.: Product Reviews: WildTangent’s Web Driver 2.0. Game Developer Magazine. June 2001.

[Melissinos] Melissinos, C.: Playing for Keeps. Sun Brings Enterprise Expertise to The Electronic Gaming Industry. 2002. Available only on-line at: <http://www.sun.com/software/cover/2002-0104/>

[Meurrens] Meurrens, M.W.F: Fundamental Design Patterns in Java – The “Enum” (Proto-)pattern. 1999. Available only on-line at: <http://www.meurrens.org/ip-Links/java/designPatterns/enum/index.html>

[Patrizio] Patrizio, A.: Dreamcast Gets Java Jolt. 2000. Available on-line at: <http://www.wired.com/news/culture/0,1284,36810,00.html>

[Phipps] Phipps, G.: Java Live Interview: C++ vs. Java. 1997. Available on-line at: <http://developer.java.sun.com/developer/community/chat/JavaLive/1997/jl0812.html>

[Photon] Photon: Low latency garbage collection via reference counting. 2000. Available only on-line at: <http://www.gamedev.net/reference/programming/features/llgc/>

[Shirazi] Shirazi, J.: Java Performance Tuning. O’Reilly & Associates. 2000. See also: <http://www.javaperformancetuning.com/>

[Shuster] Shuster, M. & Healy M. M.: Electronic Gaming Industry Chooses Java Technology as Open Software Development Platform for Next Generation Entertainment Services. Available on-line at: <http://www.sun.com/smi/Press/sunflash/2001-03/sunflash.20010322.1.html>

[Singal] Singal, S. & Nguyen, B.: The Java Factor. Communications of the ACM 41, 6. (Jun 1998). Pages 34 – 37.

[Stroustrup] Stroustrup, B.: The C++ Programming Language. 3rd Edition. Addison-Wesley. 1997.

[Sun] Sun Microsystems: The Java Platform: Five Years in Review. 2000. Available only on-line at: <http://java.sun.com/features/2000/06/time-line.html>

[Tyma] Tyma, P.: Why Are We Using Java Again? Communications of the ACM. 41, 6. (Jun 1998), Pages 38 – 42.

[Upton] Upton, B.: Postmortem: Read Storm’s Rainbow Six. Game Developer Magazine. May 1999. Also available on-line at: http://www.gamasutra.com/features/20000121/upton_01.htm

[Veronis] Veronis Suhler Media Merchant Bank: Entertainment Industry Segment Performance. 2000. Available only on-line at: <http://www.veronissuhler.com/filmed/segment.html>

[Wardell00] Wardell, B.: PC gaming as an industry. 2000. Available only on-line at: <http://www.avault.com/developer/getarticle.asp?name=bwardell1&page=1>

[Wardell01] Wardell, B.: PC gaming as an industry Part IV – Destroying Myths. 2001. Available only on-line at: <http://www.avault.com/developer/getarticle.asp?name=bwardell4>

[Wells] Wells, R.: Java offers increased productivity. 1998-1999. Available only on-line at: <http://www.wellscs.com/robert/java/productivity.htm>

[Wilson] Wilson, S. & Kesselman, J.: Java Platform Performance – Strategies and Tactics. 2000. Addison-Wesley. Available on-line at:
http://java.sun.com/docs/books/performance/1st_edition/html/JPTitle.fm.html


```

// number gives more space.
const float PARTICLEDENSITY = 250000;

#endif

```

general.h

```

/////////////////////////////////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// A set of generic funtions that is useful for multiple purposes
//
/////////////////////////////////////////////////////////////////

#ifndef GENERAL_LIB
#define GENERAL_LIB

#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include <windows.h>

// Return the absolute value of a
template <class Type>
Type Abs(const Type a)
{
    return a<0 ? -a : a;
}

// Return a random int number between low and high, both inclusive.
inline int randomInteger(int low, int high)
{
    if (low>high)
    {
        int temp = low;
        low = high;
        high = temp;
    }
    int num = (int)((((double)rand())/((double)RAND_MAX)+1.0))
        * (high-low+1) + low);
    assert(num>=low);
    assert(num<=high);
    return num;
}

// Search a set of command line arguments for one beginning with label.
// If it exist return true.
inline bool getBooleanFlag(const char* label, int argc, char* argv[])
{
    for(int i=0;i<argc;i++)
    {
        if (!strncmp(label,argv[i],strlen(label)))
        {
            return true;
        }
    }
    return false;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as an integer and return
// the value. If label was not found return defaultvalue.
inline int getIntegerArgument(const char* label, int argc, char* argv[],
                             int defaultvalue)
{
    for(int i=0;i<argc;i++)
    {

```

```

        if (!strncmp(label,argv[i],strlen(label)))
        {
            return atoi(argv[i]+strlen(label));
        }
    }
    return defaultvalue;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as a float and return
// the value. If label was not found return defaultvalue.
inline float getFloatArgument(const char* label, int argc, char* argv[],
                             float defaultvalue)
{
    for(int i=0;i<argc;i++)
    {
        if (!strncmp(label,argv[i],strlen(label)))
        {
            char* s = argv[i]+strlen(label);
            float f;
            sscanf(s,"%f",&f);
            return f;
        }
    }
    return defaultvalue;
}

#endif

```

main.cpp

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// The Main class. This is where the timed calculation occurs.
//
// Takes these parameters:
// -showprogress : Prints a dot for each step computed.
// -steps : The number of steps to compute. Defaults to 100.
// -h: The amount of time that passes for each step. Use a negative
//     value to cause it to be autocomputed. Defaults to autocomputation.
// -particles: The number of particles in the world to begin with.
//     Particles that go out of bounds will be deleted.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "constants.h"
#include "Vector.h"
#include "general.h"
#include "world.h"

#include <list>
#include <stdio.h>
#include <float.h>
#include <iostream>

typedef list<Particle> ParticleList;

int main(int argc, char* argv[])
{
    // Read command line arguments
    bool showprogress = getBooleanFlag("-showprogress",argc,argv);
    int numsteps = getIntegerArgument("-steps:",argc,argv,100);
    if (numsteps<0)
        numsteps=0;
    float optionh = getFloatArgument("-h:",argc,argv,-1);

    int numparticles = getIntegerArgument("-particles:",argc,argv,100);

```

```

if (numparticles<0)
    numparticles=0;
float worldsize = getFloatArgument("-worldsize:",argc,argv,
                                   pow(PARTICLEDENSITY * max(1,numparticles),1.0/3.0));
World world(worldsize,worldsize,worldsize);
for (int i=0;i<numparticles;i++)
{
    world.addRandomParticle();
}

world.initializeFirstStep(optionh);
cout << "h: " << world.h << endl;

// Create particlelist from world
ParticleList particles;
world.resetTraversal();
float mass;
Vector x,v;
while (world.getNextParticle(&mass,&x,&v)!=0.0)
{
    particles.push_back(Particle(mass,x,v));
}

// Do the steps in the simulation
float twoh = world.h * 2;

clock_t starttime = clock();
for (int step = 0; step < numsteps; step++)
{
    // Compute one step in the leap-frog method
    ParticleList::iterator self;
    ParticleList::iterator other;
    for (self = particles.begin(); self!=particles.end();++self)
    {
        // For particles placed later in the list, than self.
        // (The once before has already been computed.)
        for (other = self; other!=particles.end(); ++other)
        {
            if (self != other)
            {
                Vector tmp = self->F(*other)*(twoh);
                self->v += tmp / self->getW();
                other->v += -tmp / other->getW();
            }
        }
    }

    for (self = particles.begin(); self!=particles.end();)
    {
        // Compute new x value
        self->x += self->v * (twoh);

        // Detect if particle is out of the world bounds. If so: delete.
        if (!world.isPointIn(self->x))
        {
            self = particles.erase(self);
        }
        else
            ++self;
    }

    // Do collision detection.
    for (self = particles.begin(); self!=particles.end(); ++self)
    {
        for (other = self; other!=particles.end();)
        {
            if (self != other)
            {
                float radiussum = self->getRadius() + other->getRadius();
                if ((self->x - other->x).getSquaredLength()
                    <= radiussum * radiussum)
            }
        }
    }
}

```

```

        { // Collision detected: Combine the two particles in one.
          float weightsum = self->getW() + other->getW();
          self->x = (self->x * self->getW() + other->x *
                   other->getW())/(weightsum);
          self->y = (self->y * self->getW() + other->y *
                   other->getW())/(weightsum);
          self->setW(weightsum);
          // Delete the particle pointed to by other.
          other = particles.erase(other);
        }
      }
      else
        ++other;
    }
    else
      ++other;
  }
}

if (showprogress)
  cout << "." << flush;
}

float timeelapsed = ((float)(clock() - starttime)) / CLOCKS_PER_SEC;
if (showprogress)
  cout << endl;
cout << "Elapsed time: " << timeelapsed << " sec." << endl;
cout << "Steps performed: " << numsteps << endl;
if (numsteps!=0)
  cout << "Time per step: " << timeelapsed / numsteps << " sec." << endl;

world.clear();
for (ParticleList::iterator it = particles.begin();
     it!=particles.end();++it)
{
  world.addParticle(it->getW(),it->x,it->y);
}

return 0;
}

```

Vector.h

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// A generic 3D vector class of floats
//
////////////////////////////////////

#ifndef VECTOR_CLASS
#define VECTOR_CLASS

#include "general.h"
#include <math.h>
#include "constants.h"
#include <stddef.h>

class Vector
{
public:
  float x;
  float y;
  float z;

  Vector(void) : x(0), y(0), z(0) {}
  Vector(const float x, const float y, const float z) : x(x), y(y), z(z) {}
  Vector(const Vector& v) : x(v.x), y(v.y), z(v.z) {}

  inline Vector& operator+=(const Vector& v)

```

```

{
    x += v.x;
    y += v.y;
    z += v.z;
    return *this;
}

inline Vector operator+(const Vector& v) const
{
    return Vector(x+v.x,y+v.y,z+v.z);
}

inline Vector operator-(const Vector& v) const
{
    return Vector(x-v.x,y-v.y,z-v.z);
}

inline Vector operator/(const float c) const
{
    return Vector(x/c,y/c,z/c);
}

inline Vector operator-(void) const
{
    return Vector(-x,-y,-z);
}

inline Vector operator*(const float c) const
{
    return Vector(x*c,y*c,z*c);
}

inline Vector& operator--(const Vector& v)
{
    x -= v.x;
    y -= v.y;
    z -= v.z;
    return *this;
}

inline Vector& operator*=(const float c)
{
    x *= c;
    y *= c;
    z *= c;
    return *this;
}

inline Vector& operator/=(const float c)
{
    x /= c;
    y /= c;
    z /= c;
    return *this;
}

// Copy vector
inline Vector& operator=(const Vector& v)
{
    x = v.x;
    y = v.y;
    z = v.z;
    return *this;
}

// Get 2-norm
inline float getLength(void) const
{
    return (float)sqrt(x*x+y*y+z*z);
}

```

```

// Get max of each element (not the same as infinity norm)
inline float getMax(void) const
{
    return max(max(x,y),z);
}

inline float getInfinityNorm(void) const
{
    return max(max(Abs(x),Abs(y)),Abs(z));
}

// Get 2-norm but before the square root is taken
inline float getSquaredLength(void) const
{
    return x*x+y*y+z*z;
}

// Return a normalized version (unit length) of the
// vector.
inline Vector normalized(void) const
{
    float l = getLength();
    return Vector(x / l, y / l, z / l);
}

inline Vector& normalize(void)
{
    float l = getLength();
    x /= l;
    y /= l;
    z /= l;
    return *this;
}

inline void invert(void)
{
    x = -x;
    y = -y;
    z = -z;
}

inline Vector inverted(void) const
{
    return Vector(-x,-y,-z);
}

inline Vector cross(const Vector& v) const
{
    return Vector(y*v.z-z*v.y,z*v.x-x*v.z,x*v.y-y*v.x);
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
inline bool isInRectangle(const Vector& lowerCorner,
                          const Vector& upperCorner) const
{
    return (x >= lowerCorner.x && y >= lowerCorner.y && z >= lowerCorner.z &&
           x < upperCorner.x && y < upperCorner.y && z < upperCorner.z);
}
};

#endif

```

World.cpp

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//

```

```

// The world within which the particles exist.
//
///////////////////////////////////////////////////////////////////

#include "world.h"
#include <stdio.h>
#include <assert.h>
#include <iostream>
#include <limits.h>
#include <float.h>

// Delete all particles in the world.
void World::clear(void)
{
    particleList.clear();
    srand(time(NULL));
}

// Add new particle to the world.
void World::addParticle(float mass, const Vector& x, const Vector& v)
{
    particleList.push_back(Particle(mass,x,v));
}

// Call this before enumerating using getNextParticle()
void World::resetTraversal(void)
{
    it = particleList.begin();
}

// Enumerate the set of particles/bodies. Call resetTraversal
// first.
float World::getNextParticle(float* mass, Vector* x, Vector* v)
{
    if (it == particleList.end())
    {
        *mass = 0;
        *x = Vector(0,0,0);
        *v = Vector(0,0,0);
        return 0;
    }
    else
    {
        Particle b = *it;
        *mass = b.getW();
        *x = b.x;
        *v = b.v;
        ++it;
        return *mass;
    }
}

// Generate a random number between 0 and maxValue.
float World::getRandomfloat(float maxValue)
{
    // Find float in interval 0 <= r <= maxValue
    float r = ((float)rand())/(((float)((float)RAND_MAX))+1.0);

    assert(r>=0.0);
    assert(r<=1.0);

    return r*maxValue;
}

// Generate a random particle/particle and add it to the world.
void World::addRandomParticle(void)
{
    Vector pos(getRandomfloat(maxx),getRandomfloat(maxy),getRandomfloat(maxz));
    Vector speed(getRandomfloat(MAXRANDOMSPPEED*2)-MAXRANDOMSPPEED,
                getRandomfloat(MAXRANDOMSPPEED*2)-MAXRANDOMSPPEED,
                getRandomfloat(MAXRANDOMSPPEED*2)-MAXRANDOMSPPEED);
}

```

```

    particleList.push_back(Particle(getRandomfloat(MAXRANDOMMASS),pos,speed));
}

// Print particle data to screen. Fopr each particle: Weight, pos, speed
void World::print(void)
{
    for (ParticleList::iterator it = particleList.begin();
        it!=particleList.end(); ++it)
    {
        Particle b = *it;
        cout << b.getW() << " "
            << b.x.x << " "
            << b.x.y << " "
            << b.x.z << " "
            << b.v.x << " "
            << b.v.y << " "
            << b.v.z << endl;
    }
}

// Call this to initialize h if needed. Pass the h option given
// on the command line. Giving a negative number will cause h to
// be approximated automatically.
void World::initializeFirstStep(float optionh)
{
    // Only initialize if it has not already been
    // done.
    if (h==-1.0)
    {
        // If the user did not set h explicitly or set it to
        // a negative value then approximate it.
        if (optionh<=0)
        {
            h = 1;
        }
        else
        {
            h = optionh;
        }
        initializeLeapFrog();
    }
}

// Compute v_1 to start leap frog method.
void World::initializeLeapFrog()
{
    // Note: There is no need to write v to temporary variable and then
    // commit changes since F() does not use x. We do not update x
    // in this first step since we can just use x0.
    ParticleList::iterator self;
    ParticleList::iterator other;
    for (self = particleList.begin(); self!=particleList.end(); ++self)
    {
        for (other = particleList.begin(); other!=particleList.end(); ++other)
        {
            if (self!=other)
                self->v += self->F(*other)*(h/self->getW());
        }
    }
}

// Add a particle with a given mass, position and direction that is
// orbiting around the other given mass and orbit.
void World::addOrbitalParticle(float orbitMass, const Vector& xn,
                               const Vector& vn,
                               float centerMass, const Vector& xo)
{
    Vector v;

    // Set direction of v by projecting v onto orbit plane.
    Vector normal = xn-xo;

```

```

Vector crossed = normal.cross(vn);
v = crossed.cross(normal);

// Set length of v
v.normalize();
v *= sqrt(GRAVITY * centerMass / normal.getLength());

addParticle(orbitMass, xn, v);
}

```

World.h

```

/////////////////////////////////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// The world within which the particles exist.
//
/////////////////////////////////////////////////////////////////

#ifndef WORLD_MODULE
#define WORLD_MODULE

#include "constants.h"
#include <list>
#include <ostream>
#include <stdio.h>
#include <string>
#include "time.h"
#include "Vector.h"

// Retrieve STL classes into default namespace.
using namespace std;

class Particle
{
    float w; // Mass

    inline void updateRadius()
    {
        radius = (float)pow(w/MASS_PER_VOLUME,1.0/3.0);
    }

    float radius;

public:
    Vector x; // Position
    Vector v; // Move vector

    inline float getW(void)
    {
        return w;
    }

    inline void setW(float mass)
    {
        w = mass;
        updateRadius();
    }

    inline float getRadius(void)
    {
        return radius;
    }

    Particle(float w, const Vector& x, const Vector& v)
    : w(w), x(x), v(v)
    {
        updateRadius();
    }
}

```

```

}

// Calculate and return force applied between otherx and otherw
inline Vector F(const Vector& otherx, float otherw)
{
    Vector diff(x - otherx);
    float slength = diff.getSquaredLength();
    return -(diff*(w*otherw*GRAVITY/(slength*sqrt((float)slength))));
}

inline Vector F(const Particle& other)
{
    return F(other.x,other.w);
}
};

typedef list<Particle> ParticleList;

class World
{
private:
    ParticleList particleList;

    // Iterator for traversal
    ParticleList::iterator it;

    // Return random float in range 0 < x <= maxValue.
    float getRandomfloat(float maxValue);

    // Performs the first step of the leap frog algorithm.
    void initializeLeapFrog(void);

public:
    // Stepsize
    float h;

    // Legal range are 0 <= max<a> < maxValue
    float maxx;
    float maxy;
    float maxz;

    // Create a new world with the given boundaries.
    World(float maxx, float maxy, float maxz) :
        h(-1), maxx(maxx), maxy(maxy), maxz(maxz)
    {
        srand(time(NULL));
        it = particleList.begin();
    }

    // Add a random Particle to the world.
    void addRandomParticle(void);

    // Add many random bodies to the world.
    void addRandomBodies(int count);

    // Add a Particle to this world with the given mass and speed.
    void addParticle(float mass, const Vector& x, const Vector& v);

    // Add a new Particle to the world such that it will orbit around a Particle
    // given by the second set of coordinates. The speed given will
    // be project down to orbital plane and the speed
    // may change.
    void addOrbitalParticle(float orbitMass, const Vector& x1, const Vector& v1,
        float centerMass, const Vector& x2);

    // Delete all bodies in this world.
    void clear(void);

    // Start a traversal of the bodies
    void resetTraversal(void);

```

```

// Get the next Particle. Returns 0 when done.
float getNextParticle(float* mass, Vector* x, Vector* v);

// Return the number of bodies in this world
inline int getSize()
{
    return particleList.size();
}

// Print the bodies to the standard output stream.
// Is printed in the form: "mass position speed" for
// each line.
void print(void);

// Returns true if the given vector falls within
// the world boundaries.
inline bool isPointIn(const Vector& p)
{
    return (p.x >= 0 && p.y >= 0 && p.z >= 0 &&
            p.x < maxx && p.y < maxy && p.z < maxz);
}

// Assuming that the initialization has not yet occurred
// for this world (if it has then nothing happens) then
// this will give h a value and then perform the first step
// of the leapfrog algorithm. If optionh is set to a positive
// value (i.e. h was set from the command line) use that h,
// otherwise calculate approximation.
void initializeFirstStep(float optionh);

// Return the size of the world.
Vector getWorldExtent(void)
{
    return Vector(maxx,maxy,maxz);
}
};

#endif

```

Part 2: Java source code

The source code consists of the following files:

Arguments.java	This file is used when parsing command line arguments
Constants.java	A set of constants defining the behavior of the particle simulator.
Iterator.java	An iterator for a singly linked list class-
List.java	A singly linked list class
Main.java	The Main class. This is where the timed calculation occurs.
Particle.java	Objects of this class represents a single particle
Vector.java	A generic 3D vector class of floats.
World.java	The world within which the particles exist.

Arguments.java

```
////////////////////////////////////  
// Untweaked particles  
// by Jacob Marner. Feb 2002.  
//  
// This file is used when parsing command line arguments  
//  
////////////////////////////////////  
  
public class Arguments  
{  
    String[] args;  
  
    public Arguments(String[] args)  
    {  
        this.args = args;  
    }  
  
    public boolean getBooleanFlag(String label)  
    {  
        for (int i = 0; i < args.length; i++)  
        {  
            if (args[i].equals(label))  
            {  
                return true;  
            }  
        }  
        return false;  
    }  
  
    public int getIntegerArgument(String label, int defaultvalue)  
    {  
        for (int i = 0; i < args.length; i++)  
        {
```

```

        if (args[i].startsWith(label))
        {
            String s = args[i].substring(label.length());
            try
            {
                return Integer.parseInt(s);
            }
            catch (NumberFormatException e)
            {
                return defaultvalue;
            }
        }
    }
    return defaultvalue;
}

public float getFloatArgument(String label, float defaultvalue)
{
    for (int i = 0; i < args.length; i++)
    {
        if (args[i].startsWith(label))
        {
            String s = args[i].substring(label.length());
            try
            {
                return Float.parseFloat(s);
            }
            catch (NumberFormatException e)
            {
                return defaultvalue;
            }
        }
    }
    return defaultvalue;
}
}
}

```

Constants.java

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// A set constants defining the behavior of
// particle simulator.
//
////////////////////////////////////

public class Constants
{
    public static final float GRAVITY = (float)(6.67390*Math.pow(10, -11));

    // The ratio between mass and volume of particles.
    // A high value makes the size of particles smaller
    // given the same mass.
    public static final float MASS_PER_VOLUME = 50000;

    // When particles are generated by random their speed
    // along each axis will be evenly distributed among
    // -MAX_RANDOM_SPEED and MAX_RANDOM_SPEED
    public static final float MAXRANDOMSPEED = 0;

    // When particles are generated by random their mass
    // is set to a random number between 0 and
    // MAXRANDOMMASS. 0 not included.
    public static final float MAXRANDOMMASS = 10000000;

    // When particles are generated the world is given a
    // size to achieve constant particle density. High

```

```

// number gives more space.
public static final float PARTICLEDENSITY = 250000;
}

```

Iterator.java

```

/////////////////////////////////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// An iterator for a singly linked list class.
//
/////////////////////////////////////////////////////////////////

import java.io.IOException;

public final class Iterator implements java.util.Enumeration
{
    List mySList;
    transient List.ListNode myNode;

    public Iterator()
    {
    }

    Iterator(List list, List.ListNode node)
    {
        mySList = list;
        myNode = node;
    }

    public Iterator(Iterator iterator)
    {
        mySList = iterator.mySList;
        myNode = iterator.myNode;
    }

    public Iterator assign(Iterator iterator)
    {
        mySList = iterator.mySList;
        myNode = iterator.myNode;
        return this;
    }

    public Object clone()
    {
        return new Iterator(this);
    }

    public boolean atBegin()
    {
        return myNode == mySList.head;
    }

    public boolean atEnd()
    {
        return myNode == null;
    }

    public boolean hasMoreElements()
    {
        return myNode != null;
    }

    public void advance()
    {
        myNode = myNode.next;
    }

    public Object nextElement()

```

```

    {
        try
        {
            Object object = myNode.object;
            myNode = myNode.next;
            return object;
        }
        catch (NullPointerException ex)
        {
            throw new java.util.NoSuchElementException("SListIterator");
        }
    }

    public Object get()
    {
        return myNode.object;
    }
}

```

List.java

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// A singly linked list class.
//
////////////////////////////////////

import java.util.Enumeration;
import java.io.IOException;

public class List
{
    ListNode head;
    ListNode tail;
    int length;

    public List()
    {
    }

    public Enumeration elements()
    {
        return new Iterator(this, head);
    }

    public Iterator begin()
    {
        return new Iterator(this, head);
    }

    public Iterator end()
    {
        return new Iterator(this, null);
    }

    public boolean isEmpty()
    {
        return length == 0;
    }

    public int size()
    {
        return length;
    }

    public void clear()
    {

```

```

    head = null;
    tail = null;
    length = 0;
}

public Object popBack()
{
    if (length == 0)
        throw new java.lang.Error("SList is empty");

    ListNode previous = null;

    for (ListNode node = head; node != tail; node = node.next)
        previous = node;

    Object r;
    if (previous == null)
    {
        r = head.object;
        head = null;
    }
    else
    {
        r = previous.next.object;
        previous.next = null;
    }

    tail = previous;
    --length;
    return r;
}

public void pushFront(Object object)
{
    ListNode node = new ListNode();
    node.object = object;
    node.next = head;
    head = node;

    if (++length == 1)
        tail = node;
}

public Object popFront()
{
    if (length == 0)
        throw new Error("Slist is empty");

    Object r = head.object;
    head = head.next;

    if (--length == 0)
        tail = null;

    return r;
}

public Object add(Object object)
{
    ListNode node = new ListNode();
    node.object = object;

    if (++length == 1)
        head = node;
    else
        tail.next = node;

    tail = node;
    return null;
}

```

```

public void pushBack(Object object)
{
    add(object);
}

public Object remove(Enumeration pos)
{
    if (!(pos instanceof Iterator))
        throw new IllegalArgumentException("Enumeration not a Iterator");

    if (((Iterator) pos).mySList != this)
        throw new IllegalArgumentException("Enumeration not for this List");

    Object retval = ((Iterator) pos).get();
    remove(((Iterator) pos).myNode);
    return retval;
}

private Object remove(ListNode target)
{
    ListNode previous = null;

    for (ListNode node = head; node != target; node = node.next)
        previous = node;

    if (previous == null)
        head = target.next;
    else
        previous.next = target.next;

    if (target == tail)
        tail = previous;

    --length;

    return target.object;
}

final static class ListNode
{
    public ListNode next = null;
    public Object object = null;
}
}

```

Main.java

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// The Main class. This is where the timed calculation occurs.
//
// Takes these parameters:
// -showprogress : Prints a dot for each step computed.
// -steps : The number of steps to compute. Defaults to 100.
// -h: The amount of time that passes for each step. Use a negative
// value to cause it to be autocomputed. Defaults to autocomputation.
// -particles: The number of particles in the world to begin with.
// Particles that go out of bounds will be deleted.
////////////////////////////////////

import java.util.*;

public class Main
{
    public static void main(String[] args)
    {
        Arguments arguments = new Arguments(args);
    }
}

```

```

boolean showprogress = arguments.getBooleanFlag("-showprogress");
int numsteps = arguments.getIntegerArgument("-steps:", 100);
if (numsteps < 0)
    numsteps = 0;
float optionh = arguments.getFloatArgument("-h:", -1);

int numparticles = arguments.getIntegerArgument("-particles:", 100);
if (numparticles < 0)
    numparticles = 0;
float worldsize =
    arguments.getFloatArgument(
        "-worldsize:",
        (float) Math.pow(
            Constants.PARTICLEDENSITY * Math.max(1, numparticles),
            1.0 / 3.0));

World world = new World(worldsize, worldsize, worldsize);
for (int i = 0; i < numparticles; i++)
{
    world.addRandomBody();
}

world.initializeFirstStep(optionh);
System.out.println("h: " + world.h);

// Create particlelist from world
List particles = new List();
world.resetTraversal();
Particle b;
while ((b = world.getNextBody()) != null)
{
    particles.pushBack(new Particle(b));
}

// Do the steps in the simulation
float twoh = /* world.h * 2.0f */
world.h + world.h;
Date starttime = new Date();
for (int step = 0; step < numsteps; step++)
{
    // Compute one step in the leap-frog method
    Iterator self;
    Iterator other;
    for (self = particles.begin(); !self.atEnd(); self.advance())
    {
        Particle bself = (Particle) self.get();

        // For particles placed later in the list, than self.
        // (The once before has already been computed.)
        for (other = (Iterator) self.clone(); !other.atEnd();
            other.advance())
        {
            Particle bother = (Particle) other.get();

            if (bself != bother)
            {
                Vector tmp = bself.F(bother).multiply(twoh);
                bself.v.added(tmp.divide(bself.getW()));
                bother.v.subtracted(tmp.divide(bother.getW()));
            }
        }
    }

    for (self = particles.begin(); !self.atEnd();)
    {
        Particle bself = (Particle) self.get();
        // Compute new x value
        bself.x.added(bself.v.multiply(twoh));

        // Detect if particle is out of the world bounds. If so: delete.
        if (!world.isPointIn(bself.x))

```

```

    {
        System.out.println(bself.x.x + " " + bself.x.y + " " + bself.x.z);
        Iterator tmp = (Iterator) self.clone();
        self.advance();
        particles.remove(tmp);
    }
    else
        self.advance();
}

// Do collision detection.
for (self = particles.begin(); !self.atEnd(); self.advance())
{
    Particle bself = (Particle) self.get();

    for (other = (Iterator) self.clone(); !other.atEnd();)
    {
        Particle bother = (Particle) other.get();

        if (bself != bother)
        {
            float radiussum = bself.getRadius() + bother.getRadius();
            if ((bself.x.subtract(bother.x)).getSquaredLength() <=
                radiussum * radiussum)
            { // Collision detected: Combine the two particles in one.
                float weightsum = bself.getW() + bother.getW();
                bself.x = bself.x.multiply(bself.getW())
                    .add(bother.x.multiply(bother.getW())).divide(weightsum);
                bself.v = bself.v.multiply(bself.getW())
                    .add(bother.v.multiply(bother.getW())).divide(weightsum);
                bself.setW(weightsum);
                // Delete the particle pointed to by other.
                Iterator tmp = (Iterator) other.clone();
                other.advance();
                particles.remove(tmp);
            }
            else
                other.advance();
        }
        else
            other.advance();
    }
}

if (showprogress)
{
    System.out.flush();
}

float timeelapsed =
    ((float) (new Date().getTime() - starttime.getTime())) / 1000.0f;
if (showprogress)
    System.out.println();
System.out.println("Elapsed time: " + timeelapsed + " sec.");
System.out.println("Steps performed: " + numsteps);
if (numsteps != 0)
    System.out.println("Time per step: " + timeelapsed / numsteps + " sec.");
}
}

```

Particle.java

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// Objects of this class represents a single particle
//
////////////////////////////////////

```

```

public class Particle
{
    private float w; // Mass

    public void updateRadius()
    {
        radius = (float) Math.pow(w / Constants.MASS_PER_VOLUME, 1.0 / 3.0);
    }

    private float radius; // Cache radius to speed up calculations.

    public Vector x; // Position
    public Vector v; // Move vector

    public float getW()
    {
        return w;
    }

    public void setW(float mass)
    {
        w = mass;
        updateRadius();
    }

    public float getRadius()
    {
        return radius;
    }

    Particle(float w, Vector x, Vector v)
    {
        this.w = w;
        this.x = new Vector(x);
        this.v = new Vector(v);
        updateRadius();
    }

    Particle(Particle b)
    {
        w = b.w;
        x = new Vector(b.x);
        v = new Vector(b.v);
        updateRadius();
    }

    // Calculate and return force applied between otherx and otherw
    public Vector F(Vector otherx, float otherw)
    {
        Vector diff = new Vector(x.subtract(otherx));
        float slength = diff.getSquaredLength();
        return diff.multiply(w
            * otherw * Constants.GRAVITY
            / (slength * (float) Math.sqrt((float) slength))).negate();
    }

    // Calculate force vector between this particle and the other
    // one given.
    public Vector F(Particle other)
    {
        return F(other.x, other.w);
    }
}

```

Vector.java

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.

```

```

//
// A generic 3D vector class of floats.
// Note that objects of this class are mutable.
//
////////////////////////////////////

public class Vector
{
    public float x;
    public float y;
    public float z;

    public Vector()
    {
        x = 0;
        y = 0;
        z = 0;
    }

    public Vector(float x, float y, float z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public Vector(Vector v)
    {
        x = v.x;
        y = v.y;
        z = v.z;
    }

    public Vector add(Vector v)
    {
        return new Vector(x + v.x, y + v.y, z + v.z);
    }

    public Vector subtract(Vector v)
    {
        return new Vector(x - v.x, y - v.y, z - v.z);
    }

    public Vector divide(float c)
    {
        return new Vector(x / c, y / c, z / c);
    }

    public Vector negate()
    {
        return new Vector(-x, -y, -z);
    }

    public Vector multiply(float c)
    {
        return new Vector(x * c, y * c, z * c);
    }

    public Vector added(Vector v)
    {
        x += v.x;
        y += v.y;
        z += v.z;
        return this;
    }

    public Vector subtracted(Vector v)
    {
        x -= v.x;
        y -= v.y;
        z -= v.z;
    }
}

```

```

    return this;
}

public Vector multiplied(float c)
{
    x *= c;
    y *= c;
    z *= c;
    return this;
}

public Vector divided(float c)
{
    x /= c;
    y /= c;
    z /= c;
    return this;
}

// Copy vector
public Vector assign(Vector v)
{
    x = v.x;
    y = v.y;
    z = v.z;
    return this;
}

// Get 2-norm
public float getLength()
{
    return (float) Math.sqrt(x * x + y * y + z * z);
}

// Get max of each element (not the same as infinity norm)
public float getMax()
{
    return Math.max(Math.max(x, y), z);
}

public float getInfinityNorm()
{
    return Math.max(Math.max(Math.abs(x), Math.abs(y)), Math.abs(z));
}

// Get 2-norm but before the square root is taken
public float getSquaredLength()
{
    return x * x + y * y + z * z;
}

// Return a normalized version (unit length) of the
// vector.
public Vector normalized()
{
    float l = getLength();
    return new Vector(x / l, y / l, z / l);
}

public Vector normalize()
{
    float l = getLength();
    x /= l;
    y /= l;
    z /= l;
    return this;
}

public void invert()
{
    x = -x;
}

```

```

    y = -y;
    z = -z;
}

public Vector inverted()
{
    return new Vector(-x, -y, -z);
}

public Vector cross(Vector v)
{
    return new Vector(y * v.z - z * v.y, z * v.x - x * v.z, x * v.y - y * v.x);
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
public boolean isInRectangle(Vector lowerCorner, Vector upperCorner)
{
    return (
        x >= lowerCorner.x
        && y >= lowerCorner.y
        && z >= lowerCorner.z
        && x < upperCorner.x
        && y < upperCorner.y
        && z < upperCorner.z);
}
};

```

World.java

```

////////////////////////////////////
// Untweaked particles
// by Jacob Marner. Feb 2002.
//
// The world within which the particles exist.
//
////////////////////////////////////

import java.util.*;

public class World
{
    private List bodyList = new List();

    private Random rand = new Random(new Date().getTime());

    public float h = -1;

    public float maxx;
    public float maxy;
    public float maxz;

    public World(float maxx, float maxy, float maxz)
    {
        this.maxx = maxx;
        this.maxy = maxy;
        this.maxz = maxz;
        resetTraversal();
    }

    // Delete all particles in the world.
    public void clear()
    {
        bodyList.clear();
    }

    // Add new particle to the world.
    public void addBody(float mass, Vector x, Vector v)
    {

```

```

    bodyList.pushBack(new Particle(mass, x, v));
}

Enumeration it;

// Call this before enumerating using getNextBody()
public void resetTraversal()
{
    it = bodyList.elements();
}

// Enumerate the set of particles/bodies. Call resetTraversal
// first.
public Particle getNextBody()
{
    if (!it.hasMoreElements())
    {
        return null;
    }
    else
    {
        return (Particle) it.nextElement();
    }
}

// Generate a random number between 0 and maxValue.
public float getRandomfloat(float maxValue)
{
    return rand.nextFloat() * maxValue;
}

// Generate a random body/particle and add it to the world.
public void addRandomBody()
{
    Vector pos =
        new Vector(getRandomfloat(maxx),
                   getRandomfloat(maxy),
                   getRandomfloat(maxz));
    Vector speed =
        new Vector(
            getRandomfloat(Constants.MAXRANDOMSPEED * 2) - Constants.MAXRANDOMSPEED,
            getRandomfloat(Constants.MAXRANDOMSPEED * 2) - Constants.MAXRANDOMSPEED,
            getRandomfloat(Constants.MAXRANDOMSPEED * 2) - Constants.MAXRANDOMSPEED);
    bodyList.pushBack(
        new Particle(getRandomfloat(Constants.MAXRANDOMMASS), pos, speed));
}

// Print particle data to screen. For each particle: Weight, pos, speed
public void print()
{
    for (Iterator it = bodyList.begin(); !it.atEnd(); it.advance())
    {
        Particle b = (Particle) it.get();
        System.out.println(
            b.getW() + " " + b.x.x + " " + b.x.y + " " + b.x.z
            + " " + b.v.x + " " + b.v.y + " " + b.v.z);
    }
}

// Call this to initialize h if needed. Pass the h option given
// on the command line. Giving a negative number will cause h to
// be approximated automatically.
public void initializeFirstStep(float optionh)
{
    // Only initialize if it has not already been
    // done.
    if (h == -1.0)
    {
        // If the user did not set h explicitly or set it to
        // a negative value then approximate it.
        if (optionh <= 0)
    }
}

```

```

    {
        h = 1;
    }
    else
    {
        h = optionh;
    }
    initializeLeapFrog();
}
}

// Compute v_1 to start leap frog method.
public void initializeLeapFrog()
{
    // Note: There is no need to write v to temporary variable and then
    // commit changes since F() does not use x. We do not update x
    // in this first step since we can just use x0.
    Iterator self;
    Iterator other;
    for (self = bodyList.begin(); !self.atEnd(); self.advance())
    {
        Particle bself = (Particle) self.get();
        for (other = bodyList.begin(); !other.atEnd(); other.advance())
        {
            Particle bother = (Particle) other.get();
            if (bself != bother)
                bself.v.added(bself.F(bother).multiply(h / bself.getW()));
        }
    }
}

// Add a particle with a given mass, position and direction that is
// orbiting around the other given mass and orbit.
public void addOrbitalBody(
    float orbitMass,
    Vector xn,
    Vector vn,
    float centerMass,
    Vector xo)
{
    Vector v;

    // Set direction of v by projecting v onto orbit plane.
    Vector normal = xn.subtract(xo);
    Vector crossed = normal.cross(vn);
    v = crossed.cross(normal);

    // Set length of v
    v.normalize();
    v.multiplied(
        (float) Math.sqrt(Constants.GRAVITY * centerMass / normal.getLength()));

    addBody(orbitMass, xn, v);
}

boolean isPointIn(Vector p)
{
    return (
        p.x >= 0 && p.y >= 0 && p.z >= 0 && p.x < maxx && p.y < maxy &&
        p.z < maxz);
}
}

```



```
const float PARTICLEDENSITY = 250000;
```

```
#endif
```

general.h

```
//////////////////////////////////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// A set of general purpose functions.
//
//////////////////////////////////////////////////////////////////

#ifndef GENERAL_LIB
#define GENERAL_LIB

#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#include <windows.h>

// Return the absolute value of a
template <class Type>
Type Abs(const Type a)
{
    return a<0 ? -a : a;
}

// Return a random int number between low and high, both inclusive.
inline int randomInteger(int low, int high)
{
    if (low>high)
    {
        int temp = low;
        low = high;
        high = temp;
    }
    int num = (int)((((double)rand())/((double)RAND_MAX)+1.0))
        * (high-low+1) + low;
    assert(num>=low);
    assert(num<=high);
    return num;
}

// Search a set of command line arguments for one beginning with label.
// If it exist return true.
inline bool getBooleanFlag(const char* label, int argc, char* argv[])
{
    for(int i=0;i<argc;i++)
    {
        if (!strcmp(label,argv[i],strlen(label)))
        {
            return true;
        }
    }
    return false;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as an integer and return
// the value. If label was not found return defaultvalue.
inline int getIntegerArgument(const char* label, int argc, char* argv[],
                             int defaultvalue)
{
    for(int i=0;i<argc;i++)
    {
```

```

        if (!strncmp(label,argv[i],strlen(label)))
        {
            return atoi(argv[i]+strlen(label));
        }
    }
    return defaultvalue;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as a float and return
// the value. If label was not found return defaultvalue.
inline float getFloatArgument(const char* label, int argc, char* argv[],
                             float defaultvalue)
{
    for(int i=0;i<argc;i++)
    {
        if (!strncmp(label,argv[i],strlen(label)))
        {
            char* s = argv[i]+strlen(label);
            float f;
            sscanf(s,"%f",&f);
            return f;
        }
    }
    return defaultvalue;
}

#endif

```

main.cpp

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked particles - C++ version
// by Jacob Marner. Feb 2002.
//
// Takes these optional parameters:
// -showprogress : Prints a dot for each step computed.
// -steps : The number of steps to compute. Defaults to 100.
// -h: The amount of time that passes for each step. Use a negative
//     value to cause it to be autocomputed. Defaults to autocomputation.
// -particles: The number of particles in the world to begin with.
//     Particles that go out of bounds will be deleted.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "constants.h"
#include "Vector.h"
#include "general.h"
#include "world.h"

#include <list>
#include <stdio.h>
#include <float.h>
#include <iostream>

typedef list<Particle> ParticleList;

int main(int argc, char* argv[])
{
    // Read command line arguments
    bool showprogress = getBooleanFlag("-showprogress",argc,argv);
    int numsteps = getIntegerArgument("-steps:",argc,argv,100);
    if (numsteps<0)
        numsteps=0;
    float optionh = getFloatArgument("-h:",argc,argv,-1);

    int numparticles = getIntegerArgument("-particles:",argc,argv,100);
    if (numparticles<0)
        numparticles=0;
    float worldsize = getFloatArgument("-worldsize:",argc,argv,
                                       pow(PARTICLEDENSITY * max(1,numparticles),1.0/3.0));
}

```

```

World world(worldsize,worldsize,worldsize);
for (int i=0;i<numparticles;i++)
{
    world.addRandomParticle();
}

world.initializeFirstStep(optionh);
cout << "h: " << world.h << endl;

// Create particlelist from world
ParticleList particles;
world.resetTraversal();
float mass;
Vector x,v;
while (world.getNextParticle(&mass,&x,&v)!=0.0)
{
    particles.push_back(Particle(mass,x,v));
}

// Do the steps in the simulation
float twoh = world.h * 2;
Vector tmp,tmp2;

clock_t starttime = clock();
for (int step = 0; step < numsteps; step++)
{
    // Compute one step in the leap-frog method
    ParticleList::iterator self;
    ParticleList::iterator other;
    for (self = particles.begin(); self!=particles.end();++self)
    {
        // For particles placed later in the list, than self.
        // (The once before has already been computed.)
        for (other = self; other!=particles.end(); ++other)
        {
            if (self != other)
            {
                self->F(*other,tmp);
                tmp *= twoh;
                tmp2 = tmp;
                tmp /= self->getW();
                self->v += tmp;
                tmp2 /= other->getW();
                other->v -= tmp2;
            }
        }
    }

    for (self = particles.begin(); self!=particles.end();)
    {
        // Compute new x value
        tmp = self->v;
        tmp *= twoh;
        self->x += tmp;

        // Detect if particle is out of the world bounds. If so: delete.
        if (!world.isPointIn(self->x))
        {
            self = particles.erase(self);
        }
        else
            ++self;
    }

    // Do collision detection.
    for (self = particles.begin(); self!=particles.end(); ++self)
    {
        for (other = self; other!=particles.end();)
        {
            if (self != other)
            {

```

```

        float radiussum = self->getRadius() + other->getRadius();
        if ((self->x - other->x).getSquaredLength()
            <= radiussum * radiussum)
        { // Collision detected: Combine the two particles in one.
            float selfw = self->getW();
            float otherw = other->getW();
            float weightsum = selfw + otherw;

            self->x *= selfw;
            self->x += other->x * otherw;
            self->x /= weightsum;
            self->v *= selfw;
            self->v += other->v * otherw;
            self->v /= weightsum;
            self->setW(weightsum);
            // Delete the particle pointed to by other.
            other = particles.erase(other);
        }
        else
            ++other;
    }
    else
        ++other;
}

if (showprogress)
    cout << "." << flush;
}

float timeelapsed = ((float)(clock() - starttime)) / CLOCKS_PER_SEC;
if (showprogress)
    cout << endl;
cout << "Elapsed time: " << timeelapsed << " sec." << endl;
cout << "Steps performed: " << numsteps << endl;
if (numsteps!=0)
    cout << "Time per step: " << timeelapsed / numsteps << " sec." << endl;

world.clear();
for (ParticleList::iterator it = particles.begin();
     it!=particles.end();++it)
{
    world.addParticle(it->getW(),it->x,it->v);
}

return 0;
}

```

Vector.h

```

////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// A generic 3D vector class of floats
//
////////////////////////////////////

#ifndef VECTOR_CLASS
#define VECTOR_CLASS

#include "general.h"
#include <math.h>
#include "constants.h"
#include <stddef.h>

class Vector
{
public:
    float x;

```

```

float y;
float z;

Vector(void) : x(0), y(0), z(0) {}
Vector(const float x, const float y, const float z) : x(x), y(y), z(z) {}
Vector(const Vector& v) : x(v.x), y(v.y), z(v.z) {}

inline Vector& operator+=(const Vector& v)
{
    x += v.x;
    y += v.y;
    z += v.z;
    return *this;
}

inline Vector operator+(const Vector& v) const
{
    return Vector(x+v.x,y+v.y,z+v.z);
}

inline Vector operator-(const Vector& v) const
{
    return Vector(x-v.x,y-v.y,z-v.z);
}

inline Vector operator/(const float c) const
{
    return Vector(x/c,y/c,z/c);
}

inline Vector operator-(void) const
{
    return Vector(-x,-y,-z);
}

inline Vector operator*(const float c) const
{
    return Vector(x*c,y*c,z*c);
}

inline Vector& operator-=(const Vector& v)
{
    x -= v.x;
    y -= v.y;
    z -= v.z;
    return *this;
}

inline Vector& operator*=(const float c)
{
    x *= c;
    y *= c;
    z *= c;
    return *this;
}

inline Vector& operator/=(const float c)
{
    x /= c;
    y /= c;
    z /= c;
    return *this;
}

// Copy vector
inline Vector& operator=(const Vector& v)
{
    x = v.x;
    y = v.y;
    z = v.z;
    return *this;
}

```

```

}

// Get 2-norm
inline float getLength(void) const
{
    return (float)sqrt(x*x+y*y+z*z);
}

// Get max of each element (not the same as infinity norm)
inline float getMax(void) const
{
    return max(max(x,y),z);
}

inline float getInfinityNorm(void) const
{
    return max(max(Abs(x),Abs(y)),Abs(z));
}

// Get 2-norm but before the square root is taken
inline float getSquaredLength(void) const
{
    return x*x+y*y+z*z;
}

// Return a normalized version (unit length) of the
// vector.
inline Vector normalized(void) const
{
    float l = getLength();
    return Vector(x / l, y / l, z / l);
}

inline Vector& normalize(void)
{
    float l = getLength();
    x /= l;
    y /= l;
    z /= l;
    return *this;
}

inline void invert(void)
{
    x = -x;
    y = -y;
    z = -z;
}

inline Vector inverted(void) const
{
    return Vector(-x,-y,-z);
}

inline Vector cross(const Vector& v) const
{
    return Vector(y*v.z-z*v.y,z*v.x-x*v.z,x*v.y-y*v.x);
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
inline bool isInRectangle(const Vector& lowerCorner,
                          const Vector& upperCorner) const
{
    return (x >= lowerCorner.x && y >= lowerCorner.y && z >= lowerCorner.z &&
           x < upperCorner.x && y < upperCorner.y && z < upperCorner.z);
}
};

#endif

```

world.cpp

```
////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// The world within which the particles exist.
//
////////////////////////////////////

#include "world.h"
#include <stdio.h>
#include <assert.h>
#include <iostream>
#include <limits.h>
#include <float.h>

// Delete all particles in the world.
void World::clear(void)
{
    particleList.clear();
    srand(time(NULL));
}

// Add new particle to the world.
void World::addParticle(float mass, const Vector& x, const Vector& v)
{
    particleList.push_back(Particle(mass,x,v));
}

// Call this before enumerating using getNextParticle()
void World::resetTraversal(void)
{
    it = particleList.begin();
}

// Enumerate the set of particles/bodies. Call resetTraversal
// first.
float World::getNextParticle(float* mass, Vector* x, Vector* v)
{
    if (it == particleList.end())
    {
        *mass = 0;
        *x = Vector(0,0,0);
        *v = Vector(0,0,0);
        return 0;
    }
    else
    {
        Particle b = *it;
        *mass = b.getW();
        *x = b.x;
        *v = b.v;
        ++it;
        return *mass;
    }
}

// Generate a random number between 0 and maxValue.
float World::getRandomfloat(float maxValue)
{
    // Find float in interval 0 <= r <= maxValue
    float r = ((float)rand())/(((float)((float)RAND_MAX))+1.0);

    assert(r>=0.0);
    assert(r<=1.0);

    return r*maxValue;
}
```

```

// Generate a random particle/particle and add it to the world.
void World::addRandomParticle(void)
{
    Vector pos(getRandomfloat(maxx),getRandomfloat(maxy),getRandomfloat(maxz));
    Vector speed(getRandomfloat(MAXRANDOMSPPEED*2)-MAXRANDOMSPPEED,
                 getRandomfloat(MAXRANDOMSPPEED*2)-MAXRANDOMSPPEED,
                 getRandomfloat(MAXRANDOMSPPEED*2)-MAXRANDOMSPPEED);
    particleList.push_back(Particle(getRandomfloat(MAXRANDOMMASS),pos,speed));
}

// Print particle data to screen. Fopr each particle: Weight, pos, speed
void World::print(void)
{
    for (ParticleList::iterator it = particleList.begin();
         it!=particleList.end(); ++it)
    {
        Particle b = *it;
        cout << b.getW() << " "
              << b.x.x << " "
              << b.x.y << " "
              << b.x.z << " "
              << b.v.x << " "
              << b.v.y << " "
              << b.v.z << endl;
    }
}

// Call this to initialize h if needed. Pass the h option given
// on the command line. Giving a negative number will cause h to
// be approximated automatically.
void World::initializeFirstStep(float optionh)
{
    // Only initialize if it has not already been
    // done.
    if (h==-1.0)
    {
        // If the user did not set h explicitly or set it to
        // a negative value then approximate it.
        if (optionh<=0)
        {
            h = 1;
        }
        else
        {
            h = optionh;
        }
        initializeLeapFrog();
    }
}

// Compute v_1 to start leap frog method.
void World::initializeLeapFrog()
{
    // Note: There is no need to write v to temporary variable and then
    // commit changes since F() does not use x. We do not update x
    // in this first step since we can just use x0.
    ParticleList::iterator self;
    ParticleList::iterator other;
    for (self = particleList.begin(); self!=particleList.end(); ++self)
    {
        for (other = particleList.begin(); other!=particleList.end(); ++other)
        {
            if (self!=other)
                self->v += self->F(*other)*(h/self->getW());
        }
    }
}

// Add a particle with a given mass, position and direction that is
// orbiting around the other given mass and orbit.

```

```

void World::addOrbitalParticle(float orbitMass, const Vector& xn,
                             const Vector& vn,
                             float centerMass, const Vector& xo)
{
    Vector v;

    // Set direction of v by projecting v onto orbit plane.
    Vector normal = xn-xo;
    Vector crossed = normal.cross(vn);
    v = crossed.cross(normal);

    // Set length of v
    v.normalize();
    v *= sqrt(GRAVITY * centerMass / normal.getLength());

    addParticle(orbitMass, xn, v);
}

```

world.h

```

/////////////////////////////////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// The world within which the particles exist.
//
/////////////////////////////////////////////////////////////////

#ifndef WORLD_MODULE
#define WORLD_MODULE

#include "constants.h"
#include <list>
#include <ostream>
#include <stdio.h>
#include <string>
#include "time.h"
#include "Vector.h"

// Retrieve STL classes into default namespace.
using namespace std;

class Particle
{
    float w; // Mass

    inline void updateRadius()
    {
        radius = (float)pow(w/MASS_PER_VOLUME,1.0/3.0);
    }

    float radius;

public:
    Vector x; // Position
    Vector v; // Move vector

    inline float getW(void)
    {
        return w;
    }

    inline void setW(float mass)
    {
        w = mass;
        updateRadius();
    }

    inline float getRadius(void)
    {

```

```

    return radius;
}

Particle(float w, const Vector& x, const Vector& v)
: w(w), x(x), v(v)
{
    updateRadius();
}

// Calculate and return force applied between otherx and otherw
inline Vector F(const Vector& otherx, float otherw)
{
    Vector diff(x - otherx);
    float slength = diff.getSquaredLength();
    return -(diff*(w*otherw*GRAVITY/(slength*sqrt((float)slength))));
}

inline void F(const Particle& other, Vector& result)
{
    result = x - other.x;
    float slength = result.getSquaredLength();
    result *= -w*other.w*GRAVITY/(slength*sqrt((float)slength));
}

inline Vector F(const Particle& other)
{
    return F(other.x,other.w);
}
};

typedef list<Particle> ParticleList;

class World
{
private:
    ParticleList particleList;

    // Iterator for traversal
    ParticleList::iterator it;

    // Return random float in range 0 < x <= maxValue.
    float getRandomfloat(float maxValue);

    // Performs the first step of the leap frog algorithm.
    void initializeLeapFrog(void);

public:
    // Stepsize
    float h;

    // Legal range are 0 <= max<a> < maxValue
    float maxx;
    float maxy;
    float maxz;

    // Create a new world with the given boundaries.
    World(float maxx, float maxy, float maxz) :
        h(-1), maxx(maxx), maxy(maxy), maxz(maxz)
    {
        srand(time(NULL));
        it = particleList.begin();
    }

    // Add a random Particle to the world.
    void addRandomParticle(void);

    // Add many random bodies to the world.
    void addRandomBodies(int count);

    // Add a Particle to this world with the given mass and speed.

```

```

void addParticle(float mass, const Vector& x, const Vector& v);

// Add a new Particle to the world such that it will orbit around a Particle
// given by the second set of coordinates. The speed given will
// be project down to orbital plane and the speed
// may change.
void addOrbitalParticle(float orbitMass, const Vector& x1, const Vector& v1,
                       float centerMass, const Vector& x2);

// Delete all bodies in this world.
void clear(void);

// Start a traversal of the bodies
void resetTraversal(void);

// Get the next Particle. Returns 0 when done.
float getNextParticle(float* mass, Vector* x, Vector* v);

// Return the number of bodies in this world
inline int getSize()
{
    return particleList.size();
}

// Print the bodies to the standard output stream.
// Is printed in the form: "mass position speed" for
// each line.
void print(void);

// Returns true if the given vector falls within
// the world boundaries.
inline bool isPointIn(const Vector& p)
{
    return (p.x >= 0 && p.y >= 0 && p.z >= 0 &&
            p.x < maxx && p.y < maxy && p.z < maxx);
}

// Assuming that the initialization has not yet occurred
// for this world (if it has then nothing happens) then
// this will give h a value and then perform the first step
// of the leapfrog algorithm. If optionh is set to a positive
// value (i.e. h was set from the command line) use that h,
// otherwise calculate approximation.
void initializeFirstStep(float optionh);

// Return the size of the world.
Vector getWorldExtent(void)
{
    return Vector(maxx,maxy,maxz);
}
};

#endif

```

Part 2: Java source code

The source code consists of the following files:

Arguments.java	This file is used when parsing command line arguments
Constants.java	A set of constants defining the behavior of the particle simulator.
Iterator.java	An iterator for a singly linked list class-
List.java	A singly linked list class
Main.java	The Main class. This is where the timed calculation occurs.
Particle.java	Objects of this class represents a single particle
Vector.java	A generic 3D vector class of floats.
World.java	The world within which the particles exist.

Arguments.java

```
////////////////////////////////////  
// Tweaked particles  
// by Jacob Marner. Feb 2002.  
//  
// This file is used when parsing command line arguments  
//  
////////////////////////////////////  
  
final public class Arguments  
{  
    String[] args;  
  
    public Arguments(String[] args)  
    {  
        this.args = args;  
    }  
  
    public boolean getBooleanFlag(String label)  
    {  
        for (int i = 0; i < args.length; i++)  
        {  
            if (args[i].equals(label))  
            {  
                return true;  
            }  
        }  
        return false;  
    }  
  
    public int getIntegerArgument(String label, int defaultvalue)  
    {  
        for (int i = 0; i < args.length; i++)  
        {
```

```

        if (args[i].startsWith(label))
        {
            String s = args[i].substring(label.length());
            try
            {
                return Integer.parseInt(s);
            }
            catch (NumberFormatException e)
            {
                return defaultvalue;
            }
        }
    }
    return defaultvalue;
}

public float getFloatArgument(String label, float defaultvalue)
{
    for (int i = 0; i < args.length; i++)
    {
        if (args[i].startsWith(label))
        {
            String s = args[i].substring(label.length());
            try
            {
                return Float.parseFloat(s);
            }
            catch (NumberFormatException e)
            {
                return defaultvalue;
            }
        }
    }
    return defaultvalue;
}
}
}

```

Constants.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// A set constants defining the behavior of
// particle simulator.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

final public class Constants
{
    public static final float GRAVITY = (float) (6.67390 * Math.pow(10, -11));

    // The ratio between mass and volume of particles.
    // A high value makes the size of particles smaller
    // given the same mass.
    public static final float MASS_PER_VOLUME = 50000;

    // When particles are generated by random their speed
    // along each axis will be evenly distributed among
    // -MAX_RANDOM_SPEED and MAX_RANDOM_SPEED
    public static final float MAXRANDOMSPEED = 0;

    // When particles are generated by random their mass
    // is set to a random number between 0 and
    // MAXRANDOMMASS. 0 not included.
    public static final float MAXRANDOMMASS = 10000000;

    // When particles are generated the world is given a
    // size to achieve constant particle density. High
    // number gives more space.

```

```

    public static final float PARTICLEDENSITY = 250000;
}

```

Iterator.java

```

////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// An iterator for a singly linked list class.
//
////////////////////////////////////

import java.io.IOException;

public final class Iterator
{
    List mySList;
    transient List.Node myNode;

    public Iterator()
    {
    }

    Iterator(List list, List.Node node)
    {
        mySList = list;
        myNode = node;
    }

    public Iterator(Iterator iterator)
    {
        mySList = iterator.mySList;
        myNode = iterator.myNode;
    }

    public Iterator assign(Iterator iterator)
    {
        mySList = iterator.mySList;
        myNode = iterator.myNode;
        return this;
    }

    public Object clone()
    {
        return new Iterator(this);
    }

    public boolean atBegin()
    {
        return myNode == mySList.head;
    }

    public boolean atEnd()
    {
        return myNode == null;
    }

    public boolean hasMoreElements()
    {
        return myNode != null;
    }

    public void advance()
    {
        myNode = myNode.next;
    }

    public Particle nextElement()
    {

```

```

    try
    {
        Particle object = myNode.object;
        myNode = myNode.next;
        return object;
    }
    catch (NullPointerException ex)
    {
        throw new java.util.NoSuchElementException("SListIterator");
    }
}

public Particle get()
{
    return myNode.object;
}
}

```

List.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// A singly linked list class.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import java.util.Enumeration;
import java.io.IOException;

public final class List
{
    transient Node head;
    transient Node tail;
    transient int length;

    public List()
    {
    }

    public Iterator elements()
    {
        return new Iterator(this, head);
    }

    public Iterator begin()
    {
        return new Iterator(this, head);
    }

    public Iterator end()
    {
        return new Iterator(this, null);
    }

    public boolean isEmpty()
    {
        return length == 0;
    }

    public int size()
    {
        return length;
    }

    public void clear()
    {
        head = null;
    }
}

```

```

        tail = null;
        length = 0;
    }

    public Particle popBack()
    {
        if (length == 0)
            throw new java.lang.Error("SList is empty");

        Node previous = null;

        for (Node node = head; node != tail; node = node.next)
            previous = node;

        Particle r;
        if (previous == null)
        {
            r = head.object;
            head = null;
        }
        else
        {
            r = previous.next.object;
            previous.next = null;
        }

        tail = previous;
        --length;
        return r;
    }

    public void pushFront(Particle object)
    {
        Node node = new Node();
        node.object = object;
        node.next = head;
        head = node;

        if (++length == 1)
            tail = node;
    }

    public Particle popFront()
    {
        if (length == 0)
            throw new Error("Slist is empty");

        Particle r = head.object;
        head = head.next;

        if (--length == 0)
            tail = null;

        return r;
    }

    public void add(Particle object)
    {
        Node node = new Node();
        node.object = object;

        if (++length == 1)
            head = node;
        else
            tail.next = node;

        tail = node;
    }

    public void pushBack(Particle object)
    {

```

```

    add(object);
}

public void remove(Iterator pos)
{
    if (!(pos instanceof Iterator))
        throw new IllegalArgumentException("Enumeration not a SListIterator");

    if (((Iterator) pos).mySList != this)
        throw new IllegalArgumentException("Enumeration not for this SList");

    ((Iterator) pos).get();
    remove(((Iterator) pos).myNode);
}

private Particle remove(Node target)
{
    Node previous = null;

    for (Node node = head; node != target; node = node.next)
        previous = node;

    if (previous == null)
        head = target.next;
    else
        previous.next = target.next;

    if (target == tail)
        tail = previous;

    --length;

    return target.object;
}

final static class Node
{
    public Node next = null;
    public Particle object = null;
}
}

```

Main.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// The Main class. This is where the timed calculation occurs.
//
// Takes these parameters:
// -showprogress : Prints a dot for each step computed.
// -steps : The number of steps to compute. Defaults to 100.
// -h: The amount of time that passes for each step. Use a negative
//     value to cause it to be autocomputed. Defaults to autocomputation.
// -particles: The number of particles in the world to begin with.
//     Particles that go out of bounds will be deleted.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import java.util.*;

final public class Main
{
    public static void main(String[] args)
    {
        Arguments arguments = new Arguments(args);
        boolean showprogress = arguments.getBooleanFlag("-showprogress");
        int numsteps = arguments.getIntegerArgument("-steps:", 100);
        if (numsteps < 0)

```

```

    numsteps = 0;
float optionh = arguments.getFloatArgument("-h:", -1);

int numparticles = arguments.getIntegerArgument("-particles:", 100);
if (numparticles < 0)
    numparticles = 0;
float worldsize =
    arguments.getFloatArgument(
        "-worldsize:",
        (float) Math.pow(
            Constants.PARTICLEDENSITY * Math.max(1, numparticles),
            1.0 / 3.0));

World world = new World(worldsize, worldsize, worldsize);
for (int i = 0; i < numparticles; i++)
{
    world.addRandomBody();
}

world.initializeFirstStep(optionh);
System.out.println("h: " + world.h);

// Create particlelist from world
List particles = new List();
world.resetTraversal();
Particle b;
while ((b = world.getNextBody()) != null)
{
    particles.pushBack(new Particle(b));
}

// Do the steps in the simulation
float twoh = /* world.h * 2.0f */
world.h + world.h;
Vector tmp = new Vector(0, 0, 0);
Vector tmp2 = new Vector(0, 0, 0);
Particle bself;
Particle bother;
Iterator self;
Iterator other = (Iterator) particles.begin();
Iterator tmpit = (Iterator) particles.begin();

Date starttime = new Date();
for (int step = 0; step < numsteps; step++)
{
    // Compute one step in the leap-frog method
    for (self = particles.begin(); !self.atEnd(); self.advance())
    {
        bself = self.get();

        float selfw = bself.getW();

        // For particles placed later in the list, than self.
        // (The once before has already been computed.)
        for (other.assign(self); !other.atEnd(); other.advance())
        {
            bother = other.get();

            if (bself != bother)
            {
                bself.F(bother, tmp);
                tmp.multiplied(twoh);
                tmp2.assign(tmp);
                tmp2.divided(bself.getW());
                bself.v.added(tmp);
                tmp2.divided(bother.getW());
                bother.v.subtracted(tmp2);
            }
        }
    }
}

```

```

for (self = particles.begin(); !self.atEnd();)
{
    bself = self.get();
    // Compute new x value
    tmp.assign(bself.v);
    tmp.multiplied(twoh);
    bself.x.added(tmp);

    // Detect if particle is out of the world bounds. If so: delete.
    if (!world.isPointIn(bself.x))
    {
        tmpit.assign(self);
        self.advance();
        particles.remove(tmpit);
    }
    else
        self.advance();
}

// Do collision detection.
for (self = (Iterator) particles.begin(); !self.atEnd(); self.advance())
{
    bself = self.get();

    for (other.assign(self); !other.atEnd();)
    {
        bother = other.get();

        if (bself != bother)
        {
            float radiussum = bself.getRadius() + bother.getRadius();
            tmp.assign(bself.x);
            tmp.subtracted(bother.x);
            if (tmp.getSquaredLength() <= radiussum * radiussum)
            { // Collision detected: Combine the two particles in one.
                float weightsum = bself.getW() + bother.getW();
                bself.x.multiplied(bself.getW());
                bself.x.added(bother.x.multiply(bother.getW()));
                bself.x.divided(weightsum);
                bself.v.multiplied(bself.getW());
                bself.v.added(bother.v.multiply(bother.getW()));
                bself.v.divided(weightsum);
                bself.setW(weightsum);
                // Delete the particle pointed to by other.
                tmpit.assign(other);
                other.advance();
                particles.remove(tmpit);
            }
            else
                other.advance();
        }
        else
            other.advance();
    }
}

if (showprogress)
{
    System.out.print(".");
    System.out.flush();
}

}

float timeelapsed =
    ((float) (new Date().getTime() - starttime.getTime())) / 1000.0f;
if (showprogress)
    System.out.println();
System.out.println("Elapsed time: " + timeelapsed + " sec.");
System.out.println("Steps performed: " + numsteps);
if (numsteps != 0)

```

```

        System.out.println("Time per step: " + timeelapsed / numsteps + " sec.");
    }
}

```

Particles.java

```

////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// Objects of this class represents a single particle
//
////////////////////////////////////

final public class Particle
{
    private float w; // Mass

    public void updateRadius()
    {
        radius = (float) Math.pow(w / Constants.MASS_PER_VOLUME, 1.0 / 3.0);
    }

    private float radius;

    public Vector x; // Position
    public Vector v; // Move vector

    final public float getW()
    {
        return w;
    }

    final public void setW(float mass)
    {
        w = mass;
        updateRadius();
    }

    final public float getRadius()
    {
        return radius;
    }

    Particle(float w, final Vector x, final Vector v)
    {
        this.w = w;
        this.x = new Vector(x);
        this.v = new Vector(v);
        updateRadius();
    }

    Particle(final Particle b)
    {
        w = b.w;
        x = new Vector(b.x);
        v = new Vector(b.v);
        updateRadius();
    }

    // Calculate and return force applied between otherx and otherw
    final public Vector F(final Vector otherx, float otherw)
    {
        Vector diff = new Vector(x.subtract(otherx));
        float slength = diff.getSquaredLength();
        diff.multiplied(w * otherw * Constants.GRAVITY
            / (slength * (float) Math.sqrt((float) slength)));
        diff.negated();
        return diff;
    }
}

```

```

static Vector tmp = new Vector(0, 0, 0);

// Tweaked F
final public void F(final Particle other, Vector result)
{
    Vector otherx = other.x;
    float otherw = other.w;

    result.assign(x);
    result.subtracted(otherx);
    float slength = result.getSquaredLength();
    result.multiplied(w * otherw * Constants.GRAVITY
        / (slength * (float) Math.sqrt((float) slength)));
    result.negated();
}

final public Vector F(final Particle other)
{
    return F(other.x, other.w);
}
}

```

Vector.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// A generic 3D vector class of floats.
// Note that objects of this class are mutable.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

final public class Vector
{
    public float x;
    public float y;
    public float z;

    public Vector()
    {
        x = 0;
        y = 0;
        z = 0;
    }

    public Vector(float x, float y, float z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public Vector(Vector v)
    {
        x = v.x;
        y = v.y;
        z = v.z;
    }

    public Vector add(Vector v)
    {
        return new Vector(x + v.x, y + v.y, z + v.z);
    }

    public Vector subtract(Vector v)
    {
        return new Vector(x - v.x, y - v.y, z - v.z);
    }
}

```

```

public Vector divide(float c)
{
    return new Vector(x / c, y / c, z / c);
}

public Vector negate()
{
    return new Vector(-x, -y, -z);
}

public void negated()
{
    x = -x;
    y = -y;
    z = -z;
}

public Vector multiply(float c)
{
    return new Vector(x * c, y * c, z * c);
}

public void added(Vector v)
{
    x += v.x;
    y += v.y;
    z += v.z;
}

public void subtracted(Vector v)
{
    x -= v.x;
    y -= v.y;
    z -= v.z;
}

public void multiplied(float c)
{
    x *= c;
    y *= c;
    z *= c;
}

public void divided(float c)
{
    x /= c;
    y /= c;
    z /= c;
}

// Copy vector
public void assign(Vector v)
{
    x = v.x;
    y = v.y;
    z = v.z;
}

// Get 2-norm
public float getLength()
{
    return (float) Math.sqrt(x * x + y * y + z * z);
}

// Get max of each element (not the same as infinity norm)
public float getMax()
{
    return Math.max(Math.max(x, y), z);
}

```

```

public float getInfinityNorm()
{
    return Math.max(Math.max(Math.abs(x), Math.abs(y)), Math.abs(z));
}

// Get 2-norm but before the square root is taken
public float getSquaredLength()
{
    return x * x + y * y + z * z;
}

// Return a normalized version (unit length) of the
// vector.
public Vector normalized()
{
    float l = getLength();
    return new Vector(x / l, y / l, z / l);
}

public void normalize()
{
    float l = getLength();
    x /= l;
    y /= l;
    z /= l;
}

public void invert()
{
    x = -x;
    y = -y;
    z = -z;
}

public Vector inverted()
{
    return new Vector(-x, -y, -z);
}

public Vector cross(Vector v)
{
    return new Vector(y * v.z - z * v.y, z * v.x - x * v.z, x * v.y - y * v.x);
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
public boolean isInRectangle(Vector lowerCorner, Vector upperCorner)
{
    return (
        x >= lowerCorner.x
        && y >= lowerCorner.y
        && z >= lowerCorner.z
        && x < upperCorner.x
        && y < upperCorner.y
        && z < upperCorner.z);
}
};

```

World.java

```

////////////////////////////////////
// Tweaked particles
// by Jacob Marner. Feb 2002.
//
// The world within which the particles exist.
//
////////////////////////////////////

import java.util.*;

```

```

final public class World
{
    private List bodyList = new List();

    private Random rand = new Random(new Date().getTime());

    public float h = -1;

    public float maxx;
    public float maxy;
    public float maxz;

    public World(float maxx, float maxy, float maxz)
    {
        this.maxx = maxx;
        this.maxy = maxy;
        this.maxz = maxz;
        resetTraversal();
    }

    // Delete all particles in the world.
    public void clear()
    {
        bodyList.clear();
    }

    // Add new particle to the world.
    public void addBody(float mass, Vector x, Vector v)
    {
        bodyList.pushBack(new Particle(mass, x, v));
    }

    Iterator it;

    // Call this before enumerating using getNextBody()
    public void resetTraversal()
    {
        it = bodyList.elements();
    }

    // Enumerate the set of particles/bodies. Call resetTraversal
    // first.
    public Particle getNextBody()
    {
        if (!it.hasMoreElements())
        {
            return null;
        }
        else
        {
            return (Particle) it.nextElement();
        }
    }

    // Generate a random number between 0 and maxValue.
    public float getRandomfloat(float maxValue)
    {
        return rand.nextFloat() * maxValue;
    }

    // Generate a random body/particle and add it to the world.
    public void addRandomBody()
    {
        Vector pos = new Vector(getRandomfloat(maxx),
                                getRandomfloat(maxy),
                                getRandomfloat(maxz));
        Vector speed = new Vector(
            getRandomfloat(Constants.MAXRANDOMSPEED * 2) - Constants.MAXRANDOMSPEED,
            getRandomfloat(Constants.MAXRANDOMSPEED * 2) - Constants.MAXRANDOMSPEED,
            getRandomfloat(Constants.MAXRANDOMSPEED * 2) - Constants.MAXRANDOMSPEED);
    }
}

```

```

        bodyList.pushBack(
            new Particle(getRandomfloat(Constants.MAXRANDOMMASS), pos, speed));
    }

    // Print particle data to screen. For each particle: Weight, pos, speed
    public void print()
    {
        for (Iterator it = bodyList.begin(); !it.atEnd(); it.advance())
        {
            Particle b = (Particle) it.get();
            System.out.println(
                b.getW()
                + " "
                + b.x.x
                + " "
                + b.x.y
                + " "
                + b.x.z
                + " "
                + b.v.x
                + " "
                + b.v.y
                + " "
                + b.v.z);
        }
    }

    // Call this to initialize h if needed. Pass the h option given
    // on the command line. Giving a negative number will cause h to
    // be approximated automatically.
    public void initializeFirstStep(float optionh)
    {
        // Only initialize if it has not already been
        // done.
        if (h == -1.0)
        {
            // If the user did not set h explicitly or set it to
            // a negative value then approximate it.
            if (optionh <= 0)
            {
                h = 1;
            }
            else
            {
                h = optionh;
            }
            initializeLeapFrog();
        }
    }

    // Compute v_1 to start leap frog method.
    public void initializeLeapFrog()
    {
        // Note: There is no need to write v to temporary variable and then
        // commit changes since F() does not use x. We do not update x
        // in this first step since we can just use x0.
        Iterator self;
        Iterator other;
        for (self = bodyList.begin(); !self.atEnd(); self.advance())
        {
            Particle bself = (Particle) self.get();
            for (other = bodyList.begin(); !other.atEnd(); other.advance())
            {
                Particle bother = (Particle) other.get();
                if (bself != bother)
                    bself.v.added(bself.F(bother).multiply(h / bself.getW()));
            }
        }
    }

    // Add a particle with a given mass, position and direction that is

```

```

// orbiting around the other given mass and orbit.
public void addOrbitalBody(
    float orbitMass,
    Vector xn,
    Vector vn,
    float centerMass,
    Vector xo)
{
    Vector v;

    // Set direction of v by projecting v onto orbit plane.
    Vector normal = xn.subtract(xo);
    Vector crossed = normal.cross(vn);
    v = crossed.cross(normal);

    // Set length of v
    v.normalize();
    v.multiplied(
        (float) Math.sqrt(Constants.GRAVITY * centerMass / normal.getLength()));

    addBody(orbitMass, xn, v);
}

boolean isPointIn(Vector p)
{
    return (p.x >= 0 && p.y >= 0 &&
        p.z >= 0 && p.x < maxx &&
        p.y < maxy && p.z < maxz);
}
}

```

Appendix D: Untweaked Life Source code

Part 1: C++ source code

The source code consists of just one source file:

main.cpp	The application.
----------	------------------

main.cpp

```
////////////////////////////////////
// Untweaked Life - C++
// by Jacob Marner. Feb 2002.
//
// Takes two integer parameters.
// 1. The size of the world as given as the side length of a square world.
// 2. The number of time steps to run.
////////////////////////////////////

#include <queue>
#include <algorithm>
#include <iostream.h>
#include <vector>
#include <assert.h>
#include <time.h>

using namespace std;

struct Location
{
    Location(int x, int y) : x(x), y(y) {}

    Location& operator=(const Location& l)
    {
        x = l.x;
        y = l.y;
        return *this;
    }

    int x;
    int y;
};

class Society;

class Entity
{
public:
    Entity(void) :location(0,0) { };

    virtual void draw()=0;

    virtual void action(const Society& oldSociety, Society& newSociety) { };

    virtual Entity& operator=(Entity&);

protected:
    friend Society;
```

```

    Location location;
};

class Society
{
public:
    Society(const Location& size, int starttime);

    ~Society();

    void draw(void) const;

    void timetick(void);

    Entity* getEntity(const Location& pos) const;

    void setEntity(const Location& pos, Entity*);

    Location getSize() const;

    bool legalpos(const Location& pos) const;

    void registerDeath(Entity* e);

    void destroyCells(void);
private:
    Society& operator=(const Society&);

    typedef vector<vector<Entity* > > areatype;
    areatype area;

    Location size;

    int t; // time since start.

    queue<Entity* > deathrow;
};

////////////////////////////////////
// Implementation
////////////////////////////////////

Society::Society(const Location& size, int starttime)
    : size(size), t(starttime)
{
    area.resize(size.x);
    for (int i=0;i<size.x;i++)
        area[i].resize(size.y);
    for (int j=0;j<size.x;j++)
        for(int k=0;k<size.y;k++)
            area[j][k]=NULL;
}

Society::~Society()
{
    // Do not free regular cells here. They are not
    // owned by the society.

    // Free dead cells
    while (!deathrow.empty())
    {
        delete deathrow.front();
        deathrow.pop();
    }
}

void Society::destroyCells()
{
    for (int j=0;j<size.x;j++)
        for(int k=0;k<size.y;k++)
            if (area[j][k]!=NULL)

```

```

        delete area[j][k];
    }

void Society::draw(void) const
{
    int x,y;
    Entity* e;
    cout << "Time cycle: " << t << endl;
    cout << "+";
    for(x=0;x < size.x;x++)
        cout << "-";
    cout << "+" << endl;
    for(y=0;y < size.y;y++)
    {
        cout << "|";
        for(x=0;x < size.x;x++)
        {
            if((e=area[x][y])!=NULL)
                e->draw();
            else
                cout << " ";
        }
        cout << "|" << endl;
    }
    cout << "+";
    for(x=0;x < size.x;x++)
        cout << "-";
    cout << "+" << endl;
}

void Society::timetick(void)
{
    int x,y;
    Entity* e;

    Society* newsociety = new Society(size,t+1);

    for(y=0;y < size.y;y++)
        for(x=0;x < size.x;x++)
        {
            if((e=area[x][y])!=NULL)
                e->action(*this,*newsociety);
        }

    *this = *newsociety;

    delete newsociety;
}

// Copy the society. Note that the deathrow is not
// copied and that the cells are not copied either because
// they are not owned by the society.
Society& Society::operator=(const Society& ns)
{
    assert (size.x == ns.size.x);
    assert (size.y == ns.size.y);

    t = ns.t;
    area = ns.area;

    return *this;
}

void Society::setEntity(const Location& pos, Entity* e)
{
    assert(pos.x>=0);
    assert(pos.x<size.x);
    assert(pos.y>=0);
    assert(pos.y<size.y);
    area[pos.x][pos.y]=e;
    e->location = pos;
}

```

```

}

Entity* Society::getEntity(const Location& pos) const
{
    assert(pos.x>=0);
    assert(pos.x<size.x);
    assert(pos.y>=0);
    assert(pos.y<size.y);
    return area[pos.x][pos.y];
}

bool Society::legalpos(const Location& pos) const
{
    return pos.x>=0 && pos.x<size.x && pos.y>=0 && pos.y<size.y;
}

Location Society::getsize() const
{
    return size;
}

void Society::registerDeath(Entity* e)
{
    deathrow.push(e);
}

Entity& Entity::operator=(Entity& e)
{
    location = e.location;
    return *this;
}

////////////////////////////////////
// Cell definition
////////////////////////////////////
const int nx[] = {-1,-1,-1,0,0,1,1,1};
const int ny[] = {-1,0,1,1,-1,1,0,-1};

class Cell : public Entity
{
public:
    Cell() : Entity() { }

    int countNeighbours(const Society& s, const Location& pos)
    {
        int count=0;
        for (int i=0;i<8;i++)
        {
            Location l(pos.x+nx[i],pos.y+ny[i]);
            if(s.legalpos(l) &&
                s.getEntity(l)!=NULL)
                count++;
        }
        return count;
    }

    virtual void draw(void)
    {
        cout << "**";
    }

    virtual void action(const Society& oldSociety, Society& newSociety)
    {
        // If the cell has 2 or 3 neighbours it survives
        int n = countNeighbours(oldSociety,location);
        if ((n>1) && (n<4) && !newSociety.getEntity(location))
            newSociety.setEntity(location,this);
        else
            newSociety.registerDeath(this);

        // If a neighbour has 3 neighbours then a new is

```

```

        // born there.
        for (int i = 0; i<8; i++)
        {
            Location l(location.x+nx[i],location.y+ny[i]);
            if (oldSociety.legalpos(l) &&
                countNeighbours(oldSociety,l)==3 &&
                !newSociety.getEntity(l))
            {
                newSociety.setEntity(l,new Cell());
            }
        }
    };

inline int randomInteger(int high)
{
    int num = (int)((((double)rand())/((double)RAND_MAX)+1.0)
              * (high+1));
    assert(num>=0);
    assert(num<=high);
    return num;
}

////////////////////////////////////
// Main
////////////////////////////////////

int main(int argc, char* argv[])
{
    if (argc<3)
    {
        cout << "Usage: life <worldsize> <steps>" << endl;
        return 1;
    }

    int worldsize = atoi(argv[1]);

    Society mysociety(Location(worldsize,worldsize),0);

    for(int x=0;x<worldsize;x++)
        for (int y=0;y<worldsize;y++)
            if (randomInteger(10)<4)
                mysociety.setEntity(Location(x,y),new Cell());

    //mysociety.draw();
    int steps = atoi(argv[2]);
    srand(time(NULL));
    int starttime = clock();
    for (int j=0;j<steps;j++)
    {
        mysociety.timetick();
        //mysociety.draw();
    }

    mysociety.destroyCells();

    int endtime = clock();
    cout << "Time elapsed: " <<
        ((double)(endtime-starttime))/CLOCKS_PER_SEC << " sec." << endl;

    return 0;
}

```

Part 2: Java source code

The source code consists of just one source file:

Main.java	The application.
-----------	------------------

Main.java

```
////////////////////////////////////
// Untweaked Life
// by Jacob Marner. Feb 2002.
//
// Takes two integer parameters.
// 1. The size of the world as given as the side length of a square world.
// 2. The number of time steps to run.
////////////////////////////////////

import java.io.PrintStream;
import java.util.*;

class Location
{
    Location(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    Location add(Location lo)
    {
        return new Location(lo.x + x, lo.y + y);
    }

    int x;
    int y;
}

abstract class entity
{
    entity(society s, Location lo)
    {
        this.s = s;
        this.lo = lo;
    }

    char draw()
    {
        return 'x';
    }

    void action(society S)
    {
        S.setentity(getLocation(), this);
    }

    Location getLocation()
    {
        return lo;
    }

    protected Location lo;
    protected society s;
}
```

```

class society
{
    society(Location size)
    {
        this.size = size;
        e = new entity[size.x][size.y];
    }

    void setentity(Location lo, entity ent)
    {
        e[lo.x][lo.y] = ent;
    }

    entity getentity(Location lo)
    {
        return e[lo.x][lo.y];
    }

    void timetick()
    {
        society ns = new society(size);

        for (int y = 0; y < size.y; ++y)
            for (int x = 0; x < size.x; ++x)
                if (e[x][y] != null)
                    e[x][y].action(ns);

        e = ns.e;
        t = ns.t;
        size = ns.size;
    }

    void draw()
    {
        PrintStream ps = System.out;
        ps.println("Turn: " + t);
        ps.print('+');
        for (int x = 0; x < size.x; ++x)
            ps.print('-');
        ps.print('+');
        ps.println();
        for (int y = 0; y < size.y; ++y)
        {
            ps.print('|');
            for (int x = 0; x < size.x; ++x)
                if (e[x][y] != null)
                    ps.print(e[x][y].draw());
                else
                    ps.print(' ');
            ps.println('|');
        }
        ps.print('+');
        for (int x = 0; x < size.x; ++x)
            ps.print('-');
        ps.print('+');
        ps.println();
    }

    boolean legalpos(Location lo)
    {
        if (lo.x >= 0 && lo.y >= 0 && lo.x < size.x && lo.y < size.y)
            return true;
        return false;
    }

    private entity[][] e;

    private Location size;
    private int t;
}

```

```

class cell extends entity
{
    cell(society s, Location lo)
    {
        super(s, lo);
    }

    char draw()
    {
        return '*';
    }

    static final int neighbours = 8;
    static final Location[] neighbour_list =
    {
        new Location(-1, -1),
        new Location(0, -1),
        new Location(1, -1),
        new Location(-1, 0),
        new Location(1, 0),
        new Location(-1, 1),
        new Location(0, 1),
        new Location(1, 1)};

    private int count_neighbours(Location lo)
    {
        int count = 0;
        for (int i = 0; i < neighbours; ++i)
        {
            Location l = neighbour_list[i].add(lo);
            if (s.legalpos(l) && s.getentity(l) != null)
                ++count;
        }
        return count;
    }

    void action(society ns)
    {
        int n = count_neighbours(lo);
        if (n > 1 && n < 4 && ns.getentity(lo) == null)
            ns.setentity(lo, this);

        for (int i = 0; i < neighbours; ++i)
        {
            Location l = neighbour_list[i].add(lo);
            if (s.legalpos(l) && ns.getentity(l) == null && count_neighbours(l) == 3)
                ns.setentity(l, new cell(s, l));
        }
    }
}

public class Main
{
    public static void main(String[] args)
    {
        if (args.length!=2)
        {
            System.err.println("OO-Life. By Jacob Marner.");
            System.err.println("Usage: Main <worldsize> <steps>");
            System.err.println(" Will calculate the game of life for <steps> time steps");
            System.err.println(" with a sqaure world with room for <worldsize>*<worldsize>");
            System.err.println(" cells. The initial cell density is 40%.");
        }

        int worldsize = Integer.parseInt(args[0]);
        int steps = Integer.parseInt(args[1]);

        society s = new society(new Location(worldsize, worldsize));
    }
}

```

```

Random r = new Random(new Date().getTime());

for (int x = 0; x < worldsize; x++)
  for (int y = 0; y < worldsize; y++)
  {
    if ((r.nextFloat() * 10) < 4)
    {
      Location l = new Location(x, y);
      s.setentity(l, new cell(s, l));
    }
  }

Date starttime = new Date();
for (int i = 0; i < steps; i++)
{
  //s.draw(); // Uncomment this line to make
  s.timetick();
}
Date endtime = new Date();
System.out.println(
  "Time elapsed: "
  + (endtime.getTime() - starttime.getTime()) / 1000.0
  + " sec.");
}
}

```

Appendix E: Tweaked Life Source code

Part 1: C++ source code

The source code consists of just one source file:

main.cpp	The application.
----------	------------------

main.cpp

```
/*
*****
*      Life simulation      *
*      by Jacob Marner     *
*      5-26-99 edited 1-23-02 *
*****
*/

#include <queue>
#include <iostream.h>
#include <assert.h>
#include <time.h>

using namespace std;

struct Location
{
    Location(int x, int y) : x(x), y(y) {}

    Location& operator=(const Location& l)
    {
        x = l.x;
        y = l.y;
        return *this;
    }

    int x;
    int y;
};

class Society;

class Entity
{
public:
    Entity(void) :location(0,0) { };

    virtual void draw()=0;

    virtual void action(const Society& oldSociety, Society& newSociety) { };

    virtual Entity& operator=(Entity&);

protected:
    friend Society;

    Location location;
};

class Society
{
public:
```

```

Society(const Location& size, int starttime);

~Society();

void draw(void) const;

void timetick(void);

Entity* getEntity(const Location& pos) const;

void setEntity(const Location& pos, Entity*);

Location getsize() const;

bool legalpos(const Location& pos) const;

void registerDeath(Entity* e);

void destroyCells(void);
private:
void migrateTo(Society*);

typedef Entity* Entityptr;
Entityptr *area;

Location size;

int t; // time since start.

Entityptr *deathrow;
int deathpointer;
};

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Implementation
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

Society::Society(const Location& size, int starttime)
: size(size), t(starttime), deathpointer(0)
{
int sizex = size.x;
int sizey = size.y;
area = new Entityptr[sizex*sizey];

for (int i=0;i<sizex*sizey;i++)
area[i]=NULL;

deathrow = new Entityptr[sizex*sizey];
}

Society::~Society()
{
// Do not free regular cells here. They are not
// owned by the society.

// Free dead cells
for (int i=0;i<deathpointer;++i)
{
delete deathrow[i];
}
deathpointer=0;

delete[] deathrow;
}

void Society::destroyCells()
{
for (int j=0;j<size.x;j++)
for(int k=0;k<size.y;k++)
if (area[j+size.x*k]!=NULL)
delete area[j+size.x*k];
}

```

```

    delete[] area;
}

void Society::draw(void) const
{
    int x,y;
    Entity* e;
    cout << "Time cycle: " << t << endl;
    cout << "+";
    for(x=0;x < size.x;x++)
        cout << "-";
    cout << "+" << endl;
    for(y=0;y < size.y;y++)
    {
        cout << "|";
        for(x=0;x < size.x;x++)
        {
            if((e=area[x+size.x*y])!=NULL)
                e->draw();
            else
                cout << " ";
        }
        cout << "|" << endl;
    }
    cout << "+";
    for(x=0;x < size.x;x++)
        cout << "-";
    cout << "+" << endl;
}

void Society::timetick(void)
{
    Entity* e;

    Society* newsociety = new Society(size,t+1);

    int sizea = size.x*size.y;
    for(int index=0;index < sizea;index++)
        if((e=area[index])!=NULL)
            e->action(*this,*newsociety);

    migrateTo(new society);
}

// Copy the society. Note that the deathrow is not
// copied and that the cells are not copied either because
// they are not owned by the society.
void Society::migrateTo(Society* ns)
{
    delete[] area;

    assert (size.x == ns->size.x);
    assert (size.y == ns->size.y);

    area = ns->area;

    delete ns;
}

void Society::setEntity(const Location& pos, Entity* e)
{
    assert(pos.x>=0);
    assert(pos.x<size.x);
    assert(pos.y>=0);
    assert(pos.y<size.y);
    area[pos.x+size.x*pos.y]=e;
    e->location = pos;
}

Entity* Society::getEntity(const Location& pos) const

```

```

{
    assert(pos.x>=0);
    assert(pos.x<size.x);
    assert(pos.y>=0);
    assert(pos.y<size.y);
    return area[pos.x+size.x*pos.y];
}

bool Society::legalpos(const Location& pos) const
{
    return pos.x>=0 && pos.x<size.x && pos.y>=0 && pos.y<size.y;
}

Location Society::getsize() const
{
    return size;
}

void Society::registerDeath(Entity* e)
{
    deathrow[deathpointer++] = e;
}

Entity& Entity::operator=(Entity& e)
{
    location = e.location;
    return *this;
}

////////////////////////////////////
// Cell definition
////////////////////////////////////
const int nx[] = {-1,-1,-1,0,0,1,1,1};
const int ny[] = {-1,0,1,1,-1,1,0,-1};

class Cell : public Entity
{
public:
    Cell() : Entity() { }

    int countNeighbours(const Society& s, const Location& pos)
    {
        int count=0;
        for (int i=0;i<8;i++)
        {
            Location l(pos.x+nx[i],pos.y+ny[i]);
            if (s.legalpos(l) &&
                s.getEntity(l)!=NULL)
                count++;
        }
        return count;
    }

    virtual void draw(void)
    {
        cout << "**";
    }

    virtual void action(const Society& oldSociety, Society& newSociety)
    {
        // If the cell has 2 or 3 neighbours it survives
        int n = countNeighbours(oldSociety,location);
        if ((n>1) && (n<4) && !newSociety.getEntity(location))
            newSociety.setEntity(location,this);
        else
            newSociety.registerDeath(this);

        // If a neighbour has 3 neighbours then a new is
        // born there.
        for (int i = 0; i<8; i++)
        {

```

```

        Location l(location.x+nx[i],location.y+ny[i]);
        if (oldSociety.legalpos(l) &&
            countNeighbours(oldSociety,l)==3 &&
            !newSociety.getEntity(l))
        {
            newSociety.setEntity(l,new Cell());
        }
    }
};

inline int randomInteger(int high)
{
    int num = (int)((((double)rand())/((double)RAND_MAX)+1.0))
        * (high+1);
    assert(num>=0);
    assert(num<=high);
    return num;
}

////////////////////////////////////
// Main
////////////////////////////////////

int main(int argc, char* argv[])
{
    if (argc<3)
    {
        cout << "Usage: life <worldsize> <steps>" << endl;
        return 1;
    }

    int worldsize = atoi(argv[1]);

    Society mysociety(Location(worldsize,worldsize),0);

    for(int x=0;x<worldsize;x++)
        for (int y=0;y<worldsize;y++)
            if (randomInteger(10)<4)
                mysociety.setEntity(Location(x,y),new Cell());

    //mysociety.draw();
    int steps = atoi(argv[2]);
    srand(time(NULL));
    int starttime = clock();
    for (int j=0;j<steps;j++)
    {
        mysociety.timetick();
        //mysociety.draw();
    }

    mysociety.destroyCells();

    int endtime = clock();
    cout << "Time elapsed: " <<
        ((double)(endtime-starttime))/CLOCKS_PER_SEC << " sec." << endl;

    return 0;
}

```

Part 2: Java source code

The source code consists of just one source file:

Main.java	The application.
-----------	------------------

Main.java

```
////////////////////////////////////
// Tweaked Life
// by Jacob Marner. Feb 2002.
//
// Takes two integer parameters.
// 1. The size of the world as given as the side length of a square world.
// 2. The number of time steps to run.
////////////////////////////////////

import java.io.PrintStream;
import java.util.*;

class Location
{
    Location(int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    void added(Location lo)
    {
        x += lo.x;
        y += lo.y;
    }

    void assign(Location lo)
    {
        x = lo.x;
        y = lo.y;
    }

    int x;
    int y;
}

abstract class entity
{
    entity(society s, Location lo)
    {
        this.s = s;
        this.lo = lo;
    }

    char draw()
    {
        return 'x';
    }

    void action(society S)
    {
        S.setentity(getLocation(), this);
    }

    Location getLocation()
}
```

```

    {
        return lo;
    }

    protected Location lo;
    protected society s;
}

class society
{
    society(Location size)
    {
        this.size = size;
        e = new entity[size.x * size.y];
    }

    void setentity(Location lo, entity ent)
    {
        e[lo.x + size.x * lo.y] = ent;
    }

    entity getentity(Location lo)
    {
        return e[lo.x + size.x * lo.y];
    }

    void timetick()
    {
        society ns = new society(size);
        int sizea = size.x * size.y;

        for (int index = 0; index < sizea; ++index)
        {
            if (e[index] != null)
                e[index].action(ns);
        }

        e = ns.e;
        t = ns.t;
        size = ns.size;
    }

    void draw()
    {
        PrintStream ps = System.out;
        ps.println("Turn: " + t);
        ps.print('+');
        for (int x = 0; x < size.x; ++x)
            ps.print('-');
        ps.print('+');
        ps.println();
        for (int y = 0; y < size.y; ++y)
        {
            ps.print('|');
            for (int x = 0; x < size.x; ++x)
                if (e[x + size.x * y] != null)
                    ps.print(e[x + size.x * y].draw());
                else
                    ps.print(' ');
            ps.println('|');
        }
        ps.print('+');
        for (int x = 0; x < size.x; ++x)
            ps.print('-');
        ps.print('+');
        ps.println();
    }

    boolean legalpos(Location lo)
    {
        if (lo.x >= 0 && lo.y >= 0 && lo.x < size.x && lo.y < size.y)

```

```

        return true;
    }
    return false;
}

private entity[] e;

private Location size;
private int t;
}

class cell extends entity
{
    cell(society s, Location lo)
    {
        super(s, lo);
    }

    char draw()
    {
        return '*';
    }

    static final int neighbours = 8;
    static final Location[] neighbour_list =
    {
        new Location(-1, -1),
        new Location(0, -1),
        new Location(1, -1),
        new Location(-1, 0),
        new Location(1, 0),
        new Location(-1, 1),
        new Location(0, 1),
        new Location(1, 1)};

    Location l = new Location(0, 0);

    private int count_neighbours(Location lo)
    {
        int count = 0;
        for (int i = 0; i < neighbours; ++i)
        {
            l.assign(neighbour_list[i]);
            l.added(lo);
            if (s.legalpos(l) && s.getentity(l) != null)
                ++count;
        }
        return count;
    }

    Location m = new Location(0, 0);

    void action(society ns)
    {
        int n = count_neighbours(lo);
        if (n > 1 && n < 4 && ns.getentity(lo) == null)
            ns.setentity(lo, this);

        for (int i = 0; i < neighbours; ++i)
        {
            m.assign(neighbour_list[i]);
            m.added(lo);
            if (s.legalpos(l) && ns.getentity(l) == null && count_neighbours(l) == 3)
                ns.setentity(l, new cell(s, l));
        }
    }
}

public class Main
{
    public static void main(String[] args)

```

```

{
  if (args.length!=2)
  {
    System.err.println("OO-Life. By Jacob Marner.");
    System.err.println("Usage: Main <worldsize> <steps>");
    System.err.println("  Will calculate the game of life for <steps> time steps");
    System.err.println("  with a square world with room for <worldsize>*<worldsize>");
    System.err.println("  cells. The initial cell density is 40%.");
  }

  int worldsize = Integer.parseInt(args[0]);
  int steps = Integer.parseInt(args[1]);

  society s = new society(new Location(worldsize, worldsize));

  Random r = new Random(new Date().getTime());

  for (int x = 0; x < worldsize; x++)
    for (int y = 0; y < worldsize; y++)
      {
        if ((r.nextFloat() * 10) < 4)
          {
            Location l = new Location(x, y);
            s.setentity(l, new cell(s, l));
          }
      }

  Date starttime = new Date();
  for (int i = 0; i < steps; i++)
  {
    //s.draw(); // Uncomment this to print the world to screen.
    s.timetick();
  }
  Date endtime = new Date();
  System.out.println(
    "Time elapsed: "
    + (endtime.getTime() - starttime.getTime()) / 1000.0
    + " sec.");
}
}

```

Appendix F: Untweaked 3D Demo

The 3D demo exists in three versions, a C++ version and a Java version written in GL4Java and a Java version using Java3D. The Java3D demo only exists in one version so the source code will only be found in Appendix G.

Part 1: C++ source code

The source code consists of the following files:

BMPLoader.cpp	This file contains a BMP texture loader for OpenGL.
BMPLoader.h	Header file to BMPLoader.cpp
BoxTree.cpp	A Node in the kd-tree used for the cull scene graph.
BoxTree.h	Header file to BoxTree.cpp
Frustum.cpp	The view frustum.
Frustum.h	Header file to Frustum.cpp
general.h	A small set of general purpose functions.
main.cpp	Main application file.
Matrix.h	A 4x4 float matrix class
Vector.cpp	A generic float vector class.
Vector.h	Header to Vector.cpp. Contains most of the implementation of the vector.

BMPLoader.cpp

```
////////////////////////////////////  
// Untweaked virtual world (C++ version)  
// by Jacob Marner. Feb 2002.  
//  
// This file contains a BMP texture loader for OpenGL.  
//  
////////////////////////////////////  
// Copyright 2002 Jacob Marner. jacob@marner.dk. Released under LGPL.  
  
#include "BMPLoader.h"  
#include <stdio.h>  
#include <math.h>  
#include <memory.h>  
#include <string.h>  
  
// Note: This loader assumes that int is 32-bit, short is 16-bit and  
// char is 8-bit.
```

```

// Note: Data in the .BMP format is stored in little-endian format.

// The file that the BMP is stored in.
static FILE* file;

// The offset from the BITMAPFILEHEADER structure
// to the actual bitmap data in the file.
static int byteOffset;

// Image width in pixels
static int width;

// Image height in pixels
static int height;

// Number of bit per pixel. Is 1, 4, 8, 24. If it is 1,4 or 8 then
// images is paletted, otherwise not.
static int bitCount;

// Compression
enum CompressionType {BMP_BI_RGB=0, BMP_BI_RLE8, BMP_BI_RLE4 };
static CompressionType compression;

// Palette used for paletted images during load.
static unsigned char palette[3][256];

// The number of bytes read so far
static int bytesRead;

// Reads and returns a 32-bit value from the file.
static int read32BitValue()
{
    int c1 = fgetc(file);
    int c2 = fgetc(file);
    int c3 = fgetc(file);
    int c4 = fgetc(file);

    bytesRead+=4;

    return c1 + (c2<<8) + (c3<<16) + (c4<<24);
}

// Reads and returns a 16-bit value from the file
static short read16BitValue()
{
    int c1 = fgetc(file);
    int c2 = fgetc(file);

    bytesRead+=2;

    return c1 + (c2<<8);
}

// Reads and returns a 8-bit value from the file
static unsigned char read8BitValue()
{
    unsigned int c1 = fgetc(file);

    bytesRead+=1;

    return (unsigned char)c1;
}

// In Windows terms read this structure
// typedef struct tagBITMAPFILEHEADER {      /* bmfh */
//     UINT    bfType;
//     DWORD   bfSize;
//     UINT    bfReserved1;
//     UINT    bfReserved2;
//     DWORD   bfOffBits;

```

```

// } BITMAPFILEHEADER;
static LOAD_TEXTUREBMP_RESULT readFileHeader(void)
{
    // Read "BM" tag in ASCII.
    if (read8BitValue()!=66 || read8BitValue()!=77)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    // Read file size. We do not need this. Ignore.
    read32BitValue();

    // Skip the two reserved areas
    read16BitValue();
    read16BitValue();

    // Read the byte offset
    byteOffset = read32BitValue();

    return LOAD_TEXTUREBMP_SUCCESS;
}

// In windows terms read this struct:
//typedef struct tagBITMAPINFOHEADER {    /* bmih */
//    DWORD    biSize;
//    LONG     biWidth;
//    LONG     biHeight;
//    WORD     biPlanes;
//    WORD     biBitCount;
//    DWORD    biCompression;
//    DWORD    biSizeImage;
//    LONG     biXPelsPerMeter;
//    LONG     biYPelsPerMeter;
//    DWORD    biClrUsed;
//    DWORD    biClrImportant;
//} BITMAPINFOHEADER;
static LOAD_TEXTUREBMP_RESULT readInfoHeader(void)
{
    // We expect this to be at least 40. If it is larger we read
    // some more bytes in the end to be forward compatible.
    unsigned int sizeofInfoHeader = (unsigned int)read32BitValue();

    if (sizeofInfoHeader<40)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    width = read32BitValue();

    height = read32BitValue();

    // Read number of planes. According to the specification this
    // must be 1.
    if (read16BitValue()!=1)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    bitCount = read16BitValue();

    if (bitCount != 1 && bitCount != 4 && bitCount != 8 && bitCount != 24)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    compression = (CompressionType)read32BitValue();

    if (compression != BMP_BI_RGB && compression != BMP_BI_RLE8 &&
        compression != BMP_BI_RLE4)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    // Read image size. We do not need this since we have the
    // image size.
    read32BitValue();

    // Pixel to device mapping. This is irrelevant since we simply
    // want the bitmap.
    read32BitValue();
    read32BitValue();
}

```

```

// Read colors used. We do not need this, so it is ignored.
read32BitValue();

// Read the number of important colors. This will not be needed
// in OpenGL so we will ignore this.
read32BitValue();

// Apply padding in end of header to be forward compatible.
sizeofInfoHeader -= 40;
while (sizeofInfoHeader>0)
{
    read8BitValue();
    sizeofInfoHeader--;
}

return LOAD_TEXTUREBMP_SUCCESS;
}

// The palette follows directly after the
// info header
//
//typedef struct tagRGBQUAD {      /* rgbq */
// BYTE    rgbBlue;
// BYTE    rgbGreen;
// BYTE    rgbRed;
// BYTE    rgbReserved;
// } RGBQUAD;
static LOAD_TEXTUREBMP_RESULT readPalette(void)
{
    // 24-bit images are not paletted.
    if (bitCount==24)
        return LOAD_TEXTUREBMP_SUCCESS;

    int numColors = 1<<bitCount;
    for (int i=0;i<numColors;i++)
    {
        // Read RGB.
        for (int j=2;j>=0;j--)
            palette[j][i] = read8BitValue();

        // Skip reversed byte.
        read8BitValue();
    }

    return LOAD_TEXTUREBMP_SUCCESS;
}

static LOAD_TEXTUREBMP_RESULT readBitmapData1Bit(unsigned char* bitmapData)
{
    // 1-bit format cannot be compressed
    if (compression!=BMP_BI_RGB)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    unsigned char byteRead;
    for (int y=0;y<height;y++)
    {
        int index = y*width*3;
        for (int x=0;x<width;x++)
        {
            if (x%8==0)
            {
                byteRead = read8BitValue();
            }

            unsigned char color = (byteRead >> (7-(x%8))) & 1;

            bitmapData[index+x*3] = palette[0][color];
            bitmapData[index+x*3+1] = palette[1][color];
            bitmapData[index+x*3+2] = palette[2][color];
        }
    }
}

```

```

    // Pad to 32-bit boundary.
    while(bytesRead%4 != 0)
        read8BitValue();
}

return LOAD_TEXTUREBMP_SUCCESS;
}

// This is called after the first byte has been found to be 0
// and the second to be 0-2. This is only used in RLE-4 and RLE-8
// encoding.
static bool handleEscapeCode(int secondByte, int* x, int* y,
                             LOAD_TEXTUREBMP_RESULT* res)
{
    if (secondByte==0x00)
    {
        // End of line
        (*x)=0;
        if ((*y)>=height)
        {
            *res = LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
            return true;
        }
        (*y)++;
    }
    else if (secondByte==0x01)
    {
        // End of bitmap
        *res = LOAD_TEXTUREBMP_SUCCESS;
        return true;
    }
    else // secondByte=0x02
    {
        // Delta. Move drawing cursor.
        *x += read8BitValue();
        *y += read8BitValue();
        if (*x>=width || *x<0 || *y>=height || *y<0)
        {
            *res = LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
            return true;
        }
    }
}

return false;
}

// Draw a 4-bit image. Can be uncompressed or RLE-4 compressed.
static LOAD_TEXTUREBMP_RESULT readBitmapData4Bit(unsigned char* bitmapData)
{
    if (compression!=BMP_BI_RGB && compression!=BMP_BI_RLE4)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    // Uncompressed 4 bit encoding
    if (compression==BMP_BI_RGB)
    {
        for (int j=0;j<height;j++)
        {
            int index = j*width*3;
            unsigned char byteValue;
            unsigned char color;
            for (int i=0;i<width;i++)
            {
                if (i%2==0)
                {
                    byteValue = read8BitValue();
                    color = byteValue>>4;
                }
                else
                {
                    color = byteValue & 0x0F;
                }
            }
        }
    }
}

```

```

    }

    bitmapData[index+i*3] = palette[0][color];
    bitmapData[index+i*3+1] = palette[1][color];
    bitmapData[index+i*3+2] = palette[2][color];
}

// Pad to 32-bit boundary.
for (int k=0; k<(((width+1)/2)%4);k++)
    read8BitValue();
}
}

// RLE encoded 4-bit compression
if (compression==BMP_BI_RLE4)
{
    // Drawing cursor pos
    int x=0;
    int y=0;

    // Clear bitmap data since it is legal not to
    // fill it all out.
    memset(bitmapData,0,sizeof(unsigned char)*width*height*3);

    bytesRead=0;

    while(true)
    {
        unsigned char firstByte = read8BitValue();
        unsigned char secondByte = read8BitValue();

        // Is this an escape code or absolute encoding?
        if (firstByte==0)
        {
            // Is this an escape code
            if (secondByte<=0x02)
            {
                LOAD_TEXTUREBMP_RESULT res;
                if (handleEscapeCode(secondByte,&x,&y,&res))
                    return res;
            }
            else
            {
                // Absolute encoding
                int index = y*width*3;
                int color;
                unsigned char databyte;
                for (int i=0; i<secondByte; i++)
                {
                    if (x>=width)
                        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

                    if (i%2==0)
                    {
                        databyte = read8BitValue();
                        color = databyte >> 4;
                    }
                    else
                    {
                        color = databyte & 0x0F;
                    }

                    bitmapData[index+x*3] = palette[0][color];
                    bitmapData[index+x*3+1] = palette[1][color];
                    bitmapData[index+x*3+2] = palette[2][color];
                    x++;
                }

                // Pad to 16-bit word boundary
                while (bytesRead%2 != 0)
                    read8BitValue();
            }
        }
    }
}

```

```

    }
  }
  else
  {
    // If not absolute or escape code, perform data decode for next
    // length of colors
    int color1 = secondByte >> 4;
    int color2 = secondByte & 0x0F;

    for (int i=0;i<firstByte;i++)
    {
      if (x>=width)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

      int color;
      if (i%2==0)
        color = color1;
      else
        color = color2;

      int index = y*width*3+x*3;
      bitmapData[index] = palette[0][color];
      bitmapData[index+1] = palette[1][color];
      bitmapData[index+2] = palette[2][color];
      x++;
    }
  }
}

return LOAD_TEXTUREBMP_SUCCESS;
}

// Read 8-bit image. Can be uncompressed or RLE-8 compressed.
static LOAD_TEXTUREBMP_RESULT readBitmapData8Bit(unsigned char* bitmapData)
{
  if (compression!=BMP_BI_RGB && compression!=BMP_BI_RLE8)
    return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

  if (compression==BMP_BI_RGB)
  {
    // For each scan line
    for (int i=0;i<height;i++)
    {
      int index = i*width*3;
      for (int j=0;j<width;j++)
      {
        int color = read8BitValue();
        bitmapData[index+j*3] = palette[0][color];
        bitmapData[index+j*3+1] = palette[1][color];
        bitmapData[index+j*3+2] = palette[2][color];
      }

      // go to next alignment of 4 bytes.
      for (int k=0; k<(width%4);k++)
        read8BitValue();
    }
  }

  if (compression==BMP_BI_RLE8)
  {
    // Drawing cursor pos
    int x=0;
    int y=0;

    bytesRead=0;

    // Clear bitmap data since it is legal not to
    // fill it all out.
    memset(bitmapData,0,sizeof(unsigned char)*width*height*3);
  }
}

```

```

while(true)
{
    unsigned char firstByte = read8BitValue();
    unsigned char secondByte = read8BitValue();

    // Is this an escape code or absolute encoding?
    if (firstByte==0)
    {
        // Is this an escape code
        if (secondByte<=0x02)
        {
            LOAD_TEXTUREBMP_RESULT res;
            if (handleEscapeCode(secondByte,&x,&y,&res))
                return res;
        }
        else
        {
            // Absolute encoding
            int index = y*width*3;
            for (int i=0; i<secondByte; i++)
            {
                if (x>=width)
                    return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
                int color = read8BitValue();
                bitmapData[index+x*3] = palette[0][color];
                bitmapData[index+x*3+1] = palette[1][color];
                bitmapData[index+x*3+2] = palette[2][color];
                x++;
            }

            // Pad to 16-bit word boundary
            if (secondByte%2 == 1)
                read8BitValue();
        }
    }
    else
    {
        // If not, perform data decode for next length of colors
        for (int i=0;i<firstByte;i++)
        {
            if (x>=width)
                return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
            int index = y*width*3+x*3;
            bitmapData[index] = palette[0][secondByte];
            bitmapData[index+1] = palette[1][secondByte];
            bitmapData[index+2] = palette[2][secondByte];
            x++;
        }
    }
}

return LOAD_TEXTUREBMP_SUCCESS;
}

// Load a 24-bit image. Cannot be encoded.
static LOAD_TEXTUREBMP_RESULT readBitmapData24Bit(unsigned char* bitmapData)
{
    // 24-bit bitmaps cannot be encoded. Verify this.
    if (compression!=BMP_BI_RGB)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    for (int i=0;i<height;i++)
    {
        int index = i*width*3;
        for (int j=0;j<width;j++)
        {
            bitmapData[index+j*3+2] = read8BitValue();
            bitmapData[index+j*3+1] = read8BitValue();
            bitmapData[index+j*3] = read8BitValue();
        }
    }
}

```

```

        // go to next alignment of 4 bytes.
        for (int k=0; k<((width*3)%4);k++)
            read8BitValue();
    }

    return LOAD_TEXTUREBMP_SUCCESS;
}

static LOAD_TEXTUREBMP_RESULT readBitmapData(unsigned char* bitmapData)
{
    // Pad until byteoffset. Most images will need no padding
    // since they already are made to use as little space as
    // possible.
    while(bytesRead<byteOffset)
        read8BitValue();

    // The data reading procedure depends on the bit depth.
    switch(bitCount)
    {
    case 1:
        return readBitmapData1Bit(bitmapData);
    case 4:
        return readBitmapData4Bit(bitmapData);
    case 8:
        return readBitmapData8Bit(bitmapData);
    default: // 24 bit.
        return readBitmapData24Bit(bitmapData);
    }
}

// Load the BMP. Assumes that no extension is appended to the filename.
// If successful *bitmapData will contain a pointer to the data.
LOAD_TEXTUREBMP_RESULT loadBMP(const char* filename,
                               unsigned char** bitmapData)
{
    *bitmapData = NULL;

    // Append .bmp extension
    char* str = new char[strlen(filename)+5];
    if (!str)
        return LOAD_TEXTUREBMP_OUT_OF_MEMORY;

    strcpy(str,filename);
    strcat(str, ".bmp");

    // Open file for buffered read.
    file = fopen(str,"rb");

    char readBuffer[BUFSIZ];

    LOAD_TEXTUREBMP_RESULT res = LOAD_TEXTUREBMP_SUCCESS;

    if (!file)
        res = LOAD_TEXTUREBMP_COULD_NOT_FIND_OR_READ_FILE;

    if (!res)
    {
        setbuf(file, readBuffer);

        bytesRead=0;
    }

    // Read File Header
    if (!res)
        res=readFileHeader();

    // Read Info Header
    if (!res)
        res=readInfoHeader();
}

```

```

// Read Palette
if (!res)
    res=readPalette();

if (!res)
{
    // The bitmap data we are going to hand to OpenGL
    *bitmapData = new unsigned char[width*height*3];

    if (!bitmapData)
        res = LOAD_TEXTUREBMP_OUT_OF_MEMORY;
}

if (!res)
{
    // Read Data
    res=readBitmapData(*bitmapData);
}

// Clean up
delete[] str;

// Only clean up bitmapData if there was an error.
if (*bitmapData && res)
    delete[] *bitmapData;

if (file)
    fclose(file);

return res;
}

// Removes the .bmp extension of the filename if it exists.
static void removeBMPextension(const char* withext, char* result)
{
    for(int i=strlen(withext)-1;i>=0;i--)
    {
        if (!strcmp(withext+i, ".bmp") || !strcmp(withext+i, ".BMP"))
        {
            for (int j=0;j<i;j++)
            {
                result[j] = withext[j];
            }
            result[i]='\0';
            return;
        }
    }
}

// The main function. This is the only one that is exported.
LOAD_TEXTUREBMP_RESULT loadOpenGL2DTextureBMP(const char* filename,
                                              GLuint *textureName,
                                              GLint internalformat)
{
    LOAD_TEXTUREBMP_RESULT res;

    char* str = new char[strlen(filename)];

    if (!str)
        return LOAD_TEXTUREBMP_OUT_OF_MEMORY;

    removeBMPextension(filename, str);

    glPushClientAttrib(GL_CLIENT_PIXEL_STORE_BIT);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

    glGenTextures(1, textureName);

    glBindTexture(GL_TEXTURE_2D, *textureName);

```

```

// load bmp
unsigned char* bitmapData;
res = loadBMP(str,&bitmapData);

// load texture into OpenGL with mip maps and scale it if needed.
if (!res)
{
    gluBuild2DMipmaps(GL_TEXTURE_2D, internalformat, width, height, GL_RGB,
                     GL_UNSIGNED_BYTE,bitmapData);
}

if (glGetError()!=GL_NO_ERROR)
{
    res = LOAD_TEXTUREBMP_OPENGL_ERROR;
}

// Clean up.
if (bitmapData)
    delete[] bitmapData;
glPopClientAttrib();
delete[] str;
if (res)
    glDeleteTextures(1,textureName);
return res;
}

```

BMPLoader.h

```

/////////////////////////////////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A OpenGL BMP loader
//
/////////////////////////////////////////////////////////////////
// Copyright 2002 Jacob Marner. jacob@marner.dk. Released under LGPL.

#ifndef BMP_LOADER_HEADER
#define BMP_LOADER_HEADER

// Does your program not use GLUT? If not, remove the following
// include line and include the standard OpenGL headers instead,
// <GL/gl.h> and <GL/glu.h>. If you use Windows but not GLUT
// remember to include <windows.h> also.
#include <GL/glut.h>

// The following is the function return type. Use this to
// get information about how the loading operation went.

enum LOAD_TEXTUREBMP_RESULT {
    // The texture loading operation succeeded.
    LOAD_TEXTUREBMP_SUCCESS=0,
    // The file could not be found or it could not be opened for reading.
    LOAD_TEXTUREBMP_COULD_NOT_FIND_OR_READ_FILE,
    // The file does not comply with the specification.
    LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT,
    // OpenGL could not accept the texture. You probably used an internal
    // format not accepted by your OpenGL implementation or you may have run
    // out of available texture names.
    LOAD_TEXTUREBMP_OPENGL_ERROR,
    // The system ran out of heap memory during the texture load.
    LOAD_TEXTUREBMP_OUT_OF_MEMORY};

// Loads the specified BMP image and stores it into an OpenGL 2D texture, whose
// name will be returned in textureName. If an error occurred, textureName is
// undefined. The status of the operation will be returned as the
// return value.

// The bitmap will be loaded without borders and all mip maps will be
// generated automatically. If you need the bitmap for anything else than

```

```

// textures, or you need borders you will have to edit the source code.

// Parameters:
// [in] filename: The name of the texture file. Must be Windows .BMP format.
// The actual file *must* use the .bmp extension. For the filename string
// given as parameter the extension is optional - if it is not there
// it is appended automatically.
// [in/out] textureName: Pass a pointer to an already allocated GLuint and if
// the operation success then this data will point to the texture name
// of the texture object loaded. This can immediately be used with
// glBindTexture() to use the texture during drawing. If the return value
// is not LOAD_TEXTUREBMP_SUCCESS then this value is undefined. If you
// later want to delete the loaded texture call glDeleteTextures() on the
// returned texture name.
// [in] internalformat: The internal format of the loaded texture. Is passed
// directly to glTexImage2D(). Defaults to GL_RGB.
// Return value:
// [out] Status of operation.

LOAD_TEXTUREBMP_RESULT loadOpenGL2DTextureBMP(const char* filename,
                                              GLuint* textureName,
                                              GLint internalformat = GL_RGB);

#endif

```

BoxTree.cpp

```

////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A Node in the kd-tree used for the cull scene graph.
//
////////////////////////////////////

#include <iostream.h>
#include "BoxTree.h"
#include "Matrix.h"

// Reorder the box data in place according to position.x such that all with
// values below split value comes first.
static int reorderx(BoxData boxData[], int n, float splitvalue);
static int reordery(BoxData boxData[], int n, float splitvalue);
static int reorderz(BoxData boxData[], int n, float splitvalue);

static inline bool isOutside(float x, float y, float z, const Vector& point,
                            const Vector& normal)
{
    return normal.dot(Vector(x,y,z)-point)>=0;
}

static bool isInside;

bool Node::basicCullTest(int plane, const Frustum& frustum)
{
    // Find corner furthest from plane in normals direction.
    int nx = frustum.normals[plane].x >= 0 ? 1 : 0;
    int ny = frustum.normals[plane].y >= 0 ? 1 : 0;
    int nz = frustum.normals[plane].z >= 0 ? 1 : 0;

    // Check if corner furthest from plane in normals direction is outside.
    // If not it is fully inside.
    if (isOutside(boundingBox[nx].x, boundingBox[ny].y, boundingBox[nz].z,
                 frustum.points[plane],
                 frustum.normals[plane]))
    {
        // Check if corner closest to plane in normal's direction is outside.
        // If yes, the box is fully outside.
        // If no, the box is intersecting the frustum border.
        if (isOutside(boundingBox[1-nx].x, boundingBox[1-ny].y,

```

```

        boundingBox[1-nz].z,
        frustum.points[plane],
        frustum.normals[plane]))
    {
        return true;
    }
    else
        isInside=false;
}

return false;
}

// We do a frustum culling algorithm, consisting of
// 1. Bounding spheres for the frustum and bounding box
// is precalculated. These are used to do a bounding sphere
// culling.
// 2. Plane to point comparison to see if the box
// is entirely outside the plane. We only test to points
// per bounding box for each plane.
// 3. Temporal coherence culling. We remember what plane
// was used to cull this the last time and we start with
// that.
// Note: We do not use the octant test to reduce the plane
// comparisons needed to 3 since that requires a symmetric
// frustum.
void Node::render(const Frustum& frustum)
{
    isInside = true;

    int plane;

    if ((boxCenter - frustum.center).getLengthSquared() >
        frustum.radiusSquared + boxRadiusSquared)
        return;

    if (basicCullTest(lastPlaneCulled, frustum))
        return;

    for (plane=0; plane<6; plane++)
    {
        if (plane==lastPlaneCulled)
            continue;

        if (basicCullTest(plane, frustum))
        {
            lastPlaneCulled = plane;
            return;
        }
    }

    // Is the box fully within the frustum? If yes, simply render the
    // display list.
    if (isInside)
    {
        glCallList(displayList);
        return;
    }

    // We are now in doubt whether the box should be drawn.
    // If it is a leaf draw it all, if not go to children nodes.
    if (leftChild==NULL)
    {
        glCallList(displayList);
    }
    else
    {
        leftChild->render(frustum);
        rightChild->render(frustum);
    }
}
}

```

```

void Node::transformAndDrawVertex(const Matrix& m, float texx, float texy,
                                float posx, float posy, float posz)
{
    glTexCoord2f(texx, texy);
    Vector w(posx, posy, posz);
    w.MultLeftMatrix(m);
    glVertex3f(w.x, w.y, w.z);
}

// The given normal is assumed to be normalized already.
void Node::transformAndSetNormal(const Matrix& t, float x, float y, float z)
{
    Vector v(x, y, z);
    v.MultLeftMatrixIgnoreW(t);
    v.normalize();
    glNormal3f(v.x, v.y, v.z);
}

Node::Node(BoxData boxData[], int n) :
    leftChild(NULL),
    rightChild(NULL),
    displayList(0),
    lastPlaneCulled(0)
{
    boundingBox[0] = Vector(WORLD_MAX_X, WORLD_MAX_Y, WORLD_MAX_Z);
    boundingBox[1] = Vector(0, 0, 0);
    // Find and set bounding box by iterating through boxes.
    // (Boxes is centered about position and has a distance
    // from center of max sqrt(2*pow(scale,2)) )

    for (int c=0; c<n; ++c)
    {
        float boxMaxDistFromCenter = sqrt(2*pow(boxData[c].scale,2));
        float candidate;

        candidate = boxData[c].position.x - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].x)
            boundingBox[0].x = candidate;

        candidate = boxData[c].position.y - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].y)
            boundingBox[0].y = candidate;

        candidate = boxData[c].position.z - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].z)
            boundingBox[0].z = candidate;

        candidate = boxData[c].position.x + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].x)
            boundingBox[1].x = candidate;

        candidate = boxData[c].position.y + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].y)
            boundingBox[1].y = candidate;

        candidate = boxData[c].position.z + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].z)
            boundingBox[1].z = candidate;
    }

    boxCenter = (boundingBox[1]-boundingBox[0])/2 + boundingBox[0];
    boxRadiusSquared = (boxCenter - boundingBox[0]).getLengthSquared();

    // If n is more than max_boxes
    if (n>MAX_BOXES_PER_LEAF)
    {
        float xspan = boundingBox[1].x - boundingBox[0].x;
        float yspan = boundingBox[1].y - boundingBox[0].y;
        float zspan = boundingBox[1].z - boundingBox[0].z;
    }
}

```

```

int splitn;

if (xspan >= yspan && xspan >= zspan)
{
    // Find center along max axis
    float center = boundingBox[0].x + xspan/2.0f;

    // Reorder box data according to center on the max axis.
    splitn = reorderx(boxData,n,center);
}
else
{
    if (yspan >= zspan)
    {
        // Find center along max axis
        float center = boundingBox[0].y + yspan/2.0f;

        // Reorder box data according to center on the max axis.
        splitn = reordery(boxData,n,center);
    }
    else
    {
        // Find center along max axis
        float center = boundingBox[0].z + zspan/2.0f;

        // Reorder box data according to center on the max axis.
        splitn = reorderz(boxData,n,center);
    }
}

// Create nodes for each child
leftChild = new Node(boxData, splitn);
rightChild = new Node(&(boxData[splitn]), n-splitn);

// Initialize display list that calls children display list
displayList = glGenLists(1);
ASSERT(displayList);
glNewList(displayList, GL_COMPILE);
    glCallList(leftChild->displayList);
    glCallList(rightChild->displayList);
glEndList();
}
else
{
    // Initialize display list to draw the boxes. This display list
    // must be executed within a begin() end() block that was started
    // with GL_QUAD_STRIP

    displayList = glGenLists(1);
    ASSERT(displayList);
    glNewList(displayList, GL_COMPILE);
        for (int i=0;i<n;++i)
        {
            Matrix m;
            m.loadIdentity();
            m.translate(boxData[i].position.x,boxData[i].position.y,
                       boxData[i].position.z);

            m.rotate(boxData[i].rotation.x,1,0,0);
            m.rotate(boxData[i].rotation.y,0,1,0);
            m.rotate(boxData[i].rotation.z,0,0,1);
            float scale = boxData[i].scale;
            m.scale(scale,scale,scale);

            Matrix t(m);
            VERIFY(t.invert());
            t.transpose();

            // Left side
            transformAndSetNormal(t,-1,0,0);
            transformAndDrawVertex(m,1,0,-1,-1,1);
            transformAndDrawVertex(m,1,1,-1,1,1);
        }
}
}

```

```

transformAndDrawVertex(m,0,1,-1,1,-1);
transformAndDrawVertex(m,0,0,-1,-1,-1);

// Front side
transformAndSetNormal(t,0,0,1);
transformAndDrawVertex(m,1,0,1,-1,1);
transformAndDrawVertex(m,1,1,1,1,1);
transformAndDrawVertex(m,0,1,-1,1,1);
transformAndDrawVertex(m,0,0,-1,-1,1);

// Right side
transformAndSetNormal(t,1,0,0);
transformAndDrawVertex(m,1,0,1,-1,-1);
transformAndDrawVertex(m,1,1,1,1,-1);
transformAndDrawVertex(m,0,1,1,1,1);
transformAndDrawVertex(m,0,0,1,-1,1);

// Top side
transformAndSetNormal(t,0,1,0);
transformAndDrawVertex(m,1,0,1,1,1);
transformAndDrawVertex(m,1,1,1,1,-1);
transformAndDrawVertex(m,0,1,-1,1,-1);
transformAndDrawVertex(m,0,0,-1,1,1);

// Back side
transformAndSetNormal(t,0,0,-1);
transformAndDrawVertex(m,1,0,-1,-1,-1);
transformAndDrawVertex(m,1,1,-1,1,-1);
transformAndDrawVertex(m,0,1,1,1,-1);
transformAndDrawVertex(m,0,0,1,-1,-1);

// Under side
transformAndSetNormal(t,0,-1,0);
transformAndDrawVertex(m,1,0,1,-1,-1);
transformAndDrawVertex(m,1,1,1,-1,1);
transformAndDrawVertex(m,0,1,-1,-1,1);
transformAndDrawVertex(m,0,0,-1,-1,-1);
}
glEndList();
}
}

Node::~Node(void)
{
    if (leftChild)
    {
        delete leftChild;
        ASSERT(rightChild);
        delete rightChild;
    }

    glDeleteLists(displayList,1);
}

#define REORDER(X) \
static int reorder ## X(BoxData boxData[], int n, float splitvalue) \
{ \
    if (n<1) \
        return 0; \
 \
    int r = n-1; \
    int p = 0; \
 \
    int i = p-1; \
    int j = r+1; \
    while (true) \
    { \
        i++; \
        j--; \
 \
        while (j>=p && boxData[j].position. ## X>splitvalue) \

```

```

        j--;

while(i<=r && boxData[i].position. ## X<splitvalue)
    i++;

if (i<j)
{
    BoxData temp = boxData[i];
    boxData[i] = boxData[j];
    boxData[j] = temp;
}
else
{
    while (j<r && boxData[j].position. ## X<splitvalue)
        j++;
    if (j<=p)
        j=p;

    return j;
}
}
}

REORDER(x)
REORDER(y)
REORDER(z)

```

BoxTree.h

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A Node in the kd-tree used for culling.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#ifndef BOXTREE_LIBRARY
#define BOXTREE_LIBRARY

#include "Vector.h"
#include "Matrix.h"
#include "Frustum.h"
#include <GL/glut.h>

#define WORLD_MAX_X 100
#define WORLD_MAX_Y 100
#define WORLD_MAX_Z 100

#define MAX_BOXES_PER_LEAF 10

struct BoxData
{
    float scale;
    Vector position;
    Vector rotation;
};

class Node
{
public:
    // This method may reorder the ordering of the
    // box data array. n is the size of the array.
    Node(BoxData boxData[], int n);

    ~Node();

    // First min box then max box
    Vector boundingBox[2];

```

```

void render(const Frustum& frustum);

GLuint displayList;

Node* leftChild;
Node* rightChild;

int lastPlaneCulled;

private:
void transformAndDrawVertex(const Matrix& m, float texx, float texy,
                           float posx, float posy, float posz);
void transformAndSetNormal(const Matrix& m, float x, float y, float z);
bool basicCullTest(int plane, const Frustum& frustum);

Vector boxCenter;
float boxRadiusSquared;
};

#endif

```

Frustum.cpp

```

/////////////////////////////////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// The view frustum.
//
/////////////////////////////////////////////////////////////////

#include "Frustum.h"

Frustum::Frustum(Vector cameraPos, Vector cameraDirection, Vector cameraUp,
                GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
                GLdouble pnear, GLdouble pfar)
{
// Make sure that all the direction vectors are unit vectors.
cameraDirection.normalize();
cameraUp.normalize();

// Make sure that the up vector is perpendicular to the
// cameraDirection. It is now paralell to the near plane.
cameraUp = cameraDirection.cross(cameraUp.cross(cameraDirection));

int i;

// The camera is in the plane of the sides.
for (i=0;i<4;i++)
    points[i] = cameraPos;

Vector centerOfNearPlane = cameraPos + cameraDirection * pnear;

// Points for the near anf far planes are found by intersection
// viewing direction line with plane.
points[4] = centerOfNearPlane;
points[5] = cameraPos + cameraDirection * pfar;

// Set normals for near and far planes.
normals[4] = -cameraDirection; // Normal points towards camera
normals[5] = cameraDirection; // Normal points away from camera

// Calculate normals for sides. The general scheme is:
// 1. We have one point on the plane as the cameraPos.
// 2. By moving along the near plane with the given amount
//    we get another point in the plane.
// 3. These two points can, crossed with the upvector (left,right)
//    or the perpendicular of the upvector (top,bottom) give us
//    the normal of the plane.

```

```

// Calculate normal for left side
Vector leftNearPoint = cameraDirection.cross(cameraUp) * left +
    centerOfNearPlane;
normals[0] = - (leftNearPoint - cameraPos).cross(cameraUp);

// Calculate normal for right side
Vector rightNearPoint = cameraDirection.cross(cameraUp) * right +
    centerOfNearPlane;
normals[1] = (rightNearPoint - cameraPos).cross(cameraUp);

// Calculate normal for bottom side
Vector bottomNearPoint = centerOfNearPlane + cameraUp * bottom;
normals[2] = - cameraDirection.cross(cameraUp)
    .cross(bottomNearPoint - cameraPos);

// Calculate normal for top side
Vector topNearPoint = centerOfNearPlane + cameraUp * top;
normals[3] = cameraDirection.cross(cameraUp)
    .cross(topNearPoint - cameraPos);

// Calculate the center of the frustum
center = (points[5] - points[4])/2 + points[4];

Vector extremeNearCorner = centerOfNearPlane + cameraUp * max(top,-bottom) +
    cameraDirection.cross(cameraUp) * max(right,-left);

Vector extremeFarCorner = ((extremeNearCorner - cameraPos) / pnear) * pfar
    + cameraPos;

radiusSquared = (extremeFarCorner - center).getSquaredLength();
}

```

Frustum.h

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// The view fustrum
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#ifndef FRUSTUM_LIBRARY
#define FRUSTUM_LIBRARY

#include <GL/glut.h>
#include "Vector.h"

class Frustum
{
public:
    // points and normals for planes in this order:
    // left,right,bottom,top,near,far.
    Vector points[6];
    Vector normals[6];

    Vector center;
    float radiusSquared;

    // The first three args is information about the camera in the world.
    // The last 6 is the parameters given to glFrustum().
    Frustum(Vector cameraPos, Vector cameraDirection, Vector cameraUp,
        GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
        GLdouble pnear, GLdouble pfar);
};

#endif

```

general.h

```
/////////////////////////////////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A small set of general purpose functions.
//
/////////////////////////////////////////////////////////////////
#ifndef GENERAL_LIB
#define GENERAL_LIB

#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#ifdef _DEBUG
#define ASSERT(X) assert(X)
#else
#define ASSERT(X)
#endif

#ifdef _DEBUG
#define VERIFY(X) assert(X)
#else
#define VERIFY(X) X
#endif

#ifndef max
template <class Type>
Type max(const Type a, const Type b)
{
    return a < b ? a : b;
}
#endif

#ifndef min
template <class Type>
Type min(const Type a, const Type b)
{
    return a > b ? b : a;
}
#endif

// Return the absolute value of a
template <class Type>
Type Abs(const Type a)
{
    return a < 0 ? -a : a;
}

// Return a random int number between low and high, both inclusive.
inline int randomInteger(int low, int high)
{
    if (low > high)
    {
        int temp = low;
        low = high;
        high = temp;
    }
    int num = (int)((double)rand()/((double)RAND_MAX+1.0))
        * (high-low+1) + low;
    ASSERT(num >= low);
    ASSERT(num <= high);
    return num;
}

// Return a random floating point number between low and high
// both inclusive
inline float randomFloat(float low, float high)
```

```

{
    if (low>high)
    {
        float temp = low;
        low = high;
        high = temp;
    }
    float num = (float)((((double)rand())/((double)RAND_MAX)+1.0))
                * (high-low) + low);
    ASSERT(num>=low);
    ASSERT(num<=high);
    return num;
}

// Search a set of command line arguments for one beginning with label.
// If it exist return true.
inline bool getBooleanFlag(const char* label, int argc, char* argv[])
{
    for(int i=0;i<argc;i++)
    {
        if (!strncmp(label,argv[i],strlen(label)))
        {
            return true;
        }
    }
    return false;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as an integer and return
// the value. If label was not found return defaultvalue.
inline int getIntegerArgument(const char* label, int argc, char* argv[],
                             int defaultvalue)
{
    for(int i=0;i<argc;i++)
    {
        if (!strncmp(label,argv[i],strlen(label)))
        {
            return atoi(argv[i]+strlen(label));
        }
    }
    return defaultvalue;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as a float and return
// the value. If label was not found return defaultvalue.
inline float getFloatArgument(const char* label, int argc, char* argv[],
                              float defaultvalue)
{
    for(int i=0;i<argc;i++)
    {
        if (!strncmp(label,argv[i],strlen(label)))
        {
            char* s = argv[i]+strlen(label);
            float f;
            sscanf(s,"%f",&f);
            return f;
        }
    }
    return defaultvalue;
}

#endif

```

main.cpp

```

////////////////////////////////////
// Virtual World (C++ version)
// by Jacob Marner. Feb 2002.

```

```

//
// The application takes two optional parameters:
// -frames:<int>   : The number of frames to run the animation. Default: 3000.
// -boxes:<int>    : The number of boxes to generate. Default: 1000.
//
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
#include <GL/glut.h>

#include "general.h"
#include "Vector.h"
#include <time.h>
#include <iostream.h>
#include "BoxTree.h"
#include "BMPLoader.h"
#include "Frustum.h"

#ifdef _WIN32
#include <windows.h>
#endif

Node* root;

clock_t starttime;

GLuint tex;

int frame = 0;
int numFrames;

void startTimer(void)
{
    starttime = clock();
}

void stopTimer(void)
{
    float timeelapsed = ((float)(clock() - starttime)) / CLOCKS_PER_SEC;
    cout << "Elapsed time: " << timeelapsed << " sec." << endl;
    cout << "Average frame rate: " << ((float)numFrames)/timeelapsed <<
        " fps." << endl;
}

GLdouble frustumLeft;
GLdouble frustumRight;
GLdouble frustumBottom;
GLdouble frustumTop;
GLdouble frustumNear;
GLdouble frustumFar = 40;

void reshape(int width, int height)
{
    glViewport(0, 0, width, height);

    glMatrixMode(GL_PROJECTION);

    frustumLeft = -0.5;
    frustumRight = 0.5;
    frustumBottom = -0.5*((double)height/(double)width);
    frustumTop = 0.5*((double)height/(double)width);
    frustumNear = 0.5;

    glLoadIdentity();

    glFrustum(frustumLeft, frustumRight, frustumBottom,
              frustumTop, frustumNear, frustumFar);
    glMatrixMode(GL_MODELVIEW);
}

void display(void)

```

```

{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glLoadIdentity();

    // Move camera in circle around center of world.
    float angle = (float)frame / 600.0f;
    float radius = 20;
    Vector cameraPos(WORLD_MAX_X/2.0f+radius*cos(angle),
                    WORLD_MAX_Y/2.0f,
                    WORLD_MAX_Z/2.0f+radius*sin(angle));
    Vector cameraDirection(cos(angle+3.14f/2.0f),0,sin(angle+3.14f/2.0f));
    Vector upVector(0,1,0);

    gluLookAt(cameraPos.x,cameraPos.y,cameraPos.z,
              cameraPos.x+cameraDirection.x,
              cameraPos.y+cameraDirection.y,
              cameraPos.z+cameraDirection.z,upVector.x,
              upVector.y, upVector.z);

    Frustum frustum(cameraPos, cameraDirection, upVector, frustumLeft,
                    frustumRight, frustumBottom, frustumTop, frustumNear,
                    frustumFar);

    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat light_position[] = {0.0, 1.0, -1.0, 0.0 };
    GLfloat diffuse_light[] = {1.0, 1.0, 1.0, 1.0};
    GLfloat specular_light[] = {0.0, 0.0, 0.0, 0.0};
    GLfloat ambient_light[] = {1.0, 1.0, 1.0, 1.0};
    glMaterialfv(GL_FRONT,GL_SPECULAR,mat_specular);
    glMaterialfv(GL_FRONT,GL_SHININESS,mat_shininess);
    glLightfv(GL_LIGHT0,GL_POSITION,light_position);
    glLightfv(GL_LIGHT0,GL_DIFFUSE,diffuse_light);
    glLightfv(GL_LIGHT0,GL_SPECULAR,specular_light);
    glLightfv(GL_LIGHT0,GL_AMBIENT,ambient_light);

    glBindTexture(GL_TEXTURE_2D, tex);

    glBegin(GL_QUADS);
        root->render(frustum);
        //glCallList(root->displayList);
    glEnd();

    glutSwapBuffers();

    // If final frame has been drawn stop timer, print results and exit.
    frame++;
    if (frame>=numFrames)
    {
        stopTimer();
        exit(0);
    }
}

void idle(void)
{
    glutPostRedisplay();
}

void init(int numBoxes)
{
    int i;

    // Generate numBoxes box data put them in array. For each one
    // store scale(1), pos(3), rotation(3). Generate each number
    // randomly.
    srand(time(NULL));

    BoxData* boxData = new BoxData[numBoxes];

```

```

for (i=0;i<numBoxes;++i)
{
    boxData[i].scale= randomFloat(0.1,1.5);
    boxData[i].position.x= randomFloat(0,WORLD_MAX_X);
    boxData[i].position.y= randomFloat(0,WORLD_MAX_Y);
    boxData[i].position.z= randomFloat(0,WORLD_MAX_Z);
    boxData[i].rotation.x= randomFloat(0,360);
    boxData[i].rotation.y= randomFloat(0,360);
    boxData[i].rotation.z= randomFloat(0,360);
}

// Create a tree structure. Each node has bounding box information
// and zero or two children. Each node contains are display list
// printing all contents of that node as one diplaylist. Inner nodes
// use call list to sub node display lists. The leaf nodes push
// viewmodel matrix, scale, pos, rotation (as one matrix!) draw the
// stripped box and then pop matrix again.

root = new Node(boxData, numBoxes);

delete[] boxData;
}

void __cdecl exitfunc( void )
{
    delete root;
}

int main(int argc, char *argv[])
{
    ////////////////////////////////////////
    // Initialize OpenGL
    ////////////////////////////////////////

    glutInit(&argc, argv);

    atexit(exitfunc);

    int numBoxes = getIntegerArgument("-boxes:", argc, argv, 100);
    numFrames = getIntegerArgument("-frames:", argc,argv, 1000);

    // Tell OpenGL to use double buffering and
    // 16/32 bit colors (whatever is the default)
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);

    // Create a 640x480 window
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("C++ OpenGL benchmark");

    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    // Turn on z-buffer for hidden surface removal
    glEnable(GL_DEPTH_TEST);

    // Enable back face culling.
    glCullFace(GL_BACK);
    glEnable(GL_CULL_FACE);

    glClearColor(0, 0, 0, 0);

    // Tell OpenGL to use flat shading
    glShadeModel(GL_FLAT);

    // Enable Phong lighting model and one global
    // directional light.
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);

    // Load texture
    if (loadOpenGL2DTextureBMP("glow.bmp", &tex, GL_RGB))

```

```

    {
        cerr << "Error loading texture." << endl;
        exit(1);
    }

    // Enable texture mapping
    glEnable(GL_TEXTURE_2D);
    // Blend lighting a texture effects.
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    // Enable trilinear filtering.
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
    glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    // Enable fogging
    glEnable(GL_FOG);
    GLfloat fogColor[4] = {0.0f, 0.0f, 0.0f, 0.0f};
    glFogi(GL_FOG_MODE, GL_LINEAR);
    glFogfv(GL_FOG_COLOR, fogColor);
    glFogf(GL_FOG_DENSITY, 0.35);
    glHint(GL_FOG_HINT, GL_DONT_CARE);
    glFogf(GL_FOG_START, frustumFar * 0.85);
    glFogf(GL_FOG_END, frustumFar);

    // Register call back functions
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutIdleFunc(idle);

#ifdef _WIN32
    // Turn off vertical screen sync under Windows.
    // (I.e. it uses the WGL_EXT_swap_control extension)
    // On under systems we just live with the default.
    typedef BOOL (WINAPI *PFNWGLSWAPINTERVALEXTPROC)(int interval);
    PFNWGLSWAPINTERVALEXTPROC wglSwapIntervalEXT = NULL;
    wglSwapIntervalEXT =
        (PFNWGLSWAPINTERVALEXTPROC)wglGetProcAddress("wglSwapIntervalEXT");
    ASSERT(wglSwapIntervalEXT);
    wglSwapIntervalEXT(0);
#endif

    // Create data and display lists
    init(numBoxes);

    startTimer();

    glutMainLoop();
    return 0;
}

```

Matrix.h

```

////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A 4x4 float matrix class
//
////////////////////////////////////

#ifndef MATRIX_CLASS
#define MATRIX_CLASS

#include <math.h>

template<class Type>
inline float toRadians(Type deg)
{
    return deg*3.1415926535f/180.0f;
}

```

```

class Matrix
{
public:
    float m[16];

    Matrix(void)
    {
    }

    void loadIdentity()
    {
        m[0]=m[5]=m[10]=m[15]=1;
        m[1]=m[2]=m[3]=m[4]=m[6]=m[7]=m[8]=m[9]=m[11]=m[12]=m[13]=m[14]=0;
    }

    Matrix(float m0,float m1,float m2,float m3,float m4,
           float m5,float m6,float m7,float m8,float m9,
           float m10,float m11,float m12,float m13,float m14,
           float m15)
    {
        m[0] = m0;
        m[1] = m1;
        m[2] = m2;
        m[3] = m3;
        m[4] = m4;
        m[5] = m5;
        m[6] = m6;
        m[7] = m7;
        m[8] = m8;
        m[9] = m9;
        m[10] = m10;
        m[11] = m11;
        m[12] = m12;
        m[13] = m13;
        m[14] = m14;
        m[15] = m15;
    }

    Matrix(const Matrix& ma)
    {
        memcpy(m,ma.m,16*sizeof(float));
    }

    Matrix& operator=(const Matrix& ma)
    {
        memcpy(m,ma.m,16*sizeof(float));
        return *this;
    }

    void mult(Matrix &ma)
    {
        float *n = ma.m;
        float t[12];
        t[0] = m[0] * n[0] + m[4] * n[1] + m[8] * n[2] + m[12] * n[3];
        t[1] = m[1] * n[0] + m[5] * n[1] + m[9] * n[2] + m[13] * n[3];
        t[2] = m[2] * n[0] + m[6] * n[1] + m[10] * n[2] + m[14] * n[3];
        t[3] = m[3] * n[0] + m[7] * n[1] + m[11] * n[2] + m[15] * n[3];
        t[4] = m[0] * n[4] + m[4] * n[5] + m[8] * n[6] + m[12] * n[7];
        t[5] = m[1] * n[4] + m[5] * n[5] + m[9] * n[6] + m[13] * n[7];
        t[6] = m[2] * n[4] + m[6] * n[5] + m[10] * n[6] + m[14] * n[7];
        t[7] = m[3] * n[4] + m[7] * n[5] + m[11] * n[6] + m[15] * n[7];
        t[8] = m[0] * n[8] + m[4] * n[9] + m[8] * n[10] + m[12] * n[11];
        t[9] = m[1] * n[8] + m[5] * n[9] + m[9] * n[10] + m[13] * n[11];
        t[10] = m[2] * n[8] + m[6] * n[9] + m[10] * n[10] + m[14] * n[11];
        t[11] = m[3] * n[8] + m[7] * n[9] + m[11] * n[10] + m[15] * n[11];
        m[12] = m[0] * n[12] + m[4] * n[13] + m[8] * n[14] + m[12] * n[15];
        m[13] = m[1] * n[12] + m[5] * n[13] + m[9] * n[14] + m[13] * n[15];
        m[14] = m[2] * n[12] + m[6] * n[13] + m[10] * n[14] + m[14] * n[15];
        m[15] = m[3] * n[12] + m[7] * n[13] + m[11] * n[14] + m[15] * n[15];
        memcpy(m,t,12*sizeof(float));
    }
}

```

```

}

void rotate(float a, float x, float y, float z)
{
    // Normalize axis vector
    float slength = x*x+y*y+z*z;
    if (slength!=1.0f)
    {
        float length = (float)sqrt(slength);
        x/=length;
        y/=length;
        z/=length;
    }

    float rad = toRadians(a);

    float c = (float)cos(rad);
    float s = (float)sin(rad);
    float oneminusc = 1.0f - c;
    float xy = x*y;
    float yz = y*z;
    float xz = x*z;
    float xs = x*s;
    float ys = y*s;
    float zs = z*s;

    float n[11];
    n[0] = x*x*oneminusc+c;
    n[1] = xy*oneminusc+zs;
    n[2] = xz*oneminusc-ys;
    // n[3] not used
    n[4] = xy*oneminusc-zs;
    n[5] = y*y*oneminusc+c;
    n[6] = yz*oneminusc+xs;
    // n[7] not used
    n[8] = xz*oneminusc+ys;
    n[9] = yz*oneminusc-xs;
    n[10] = z*z*oneminusc+c;

    /* m[12] to m[15] are not changed by a rotate */
    float t[8];
    t[0]= m[0] * n[0] + m[4] * n[1] + m[8] * n[2];
    t[1]= m[1] * n[0] + m[5] * n[1] + m[9] * n[2];
    t[2]= m[2] * n[0] + m[6] * n[1] + m[10] * n[2];
    t[3]= m[3] * n[0] + m[7] * n[1] + m[11] * n[2];
    t[4]= m[0] * n[4] + m[4] * n[5] + m[8] * n[6];
    t[5]= m[1] * n[4] + m[5] * n[5] + m[9] * n[6];
    t[6]= m[2] * n[4] + m[6] * n[5] + m[10] * n[6];
    t[7]= m[3] * n[4] + m[7] * n[5] + m[11] * n[6];
    m[8]= m[0] * n[8] + m[4] * n[9] + m[8] * n[10];
    m[9]= m[1] * n[8] + m[5] * n[9] + m[9] * n[10];
    m[10]=m[2] * n[8] + m[6] * n[9] + m[10] * n[10];
    m[11]=m[3] * n[8] + m[7] * n[9] + m[11] * n[10];
    memcpy(m,t,8*sizeof(float));
}

void translate(float x, float y, float z)
{
    /* m[0] to m[11] are not changed by a translate */
    m[12] = m[0] * x + m[4] * y + m[8] * z + m[12];
    m[13] = m[1] * x + m[5] * y + m[9] * z + m[13];
    m[14] = m[2] * x + m[6] * y + m[10] * z + m[14];
    m[15] = m[3] * x + m[7] * y + m[11] * z + m[15];
}

void scale(float x, float y, float z)
{
    /* m[12] to m[15] are not changed by a scale */
    m[0] *= x;
    m[1] *= x;
    m[2] *= x;

```

```

m[3] *= x;
m[4] *= y;
m[5] *= y;
m[6] *= y;
m[7] *= y;
m[8] *= z;
m[9] *= z;
m[10] *= z;
m[11] *= z;
}

// Returns true at success or false if the matrix was singular
bool invert(void)
{
    const int n = 4;
    const int n2 = 2*n;
    const int nn = n*n;

    float alpha;
    float beta;
    int i;
    int j;
    int k;

    float D[n2*n];
    for (i=0;i<nn;++i)
        D[i]=m[i];

    // init the reduction matrix
    for( i = 0; i < n; i++ )
    {
        for( j = 0; j < n; j++ )
        {
            D[i+n*j+nn] = 0.0;
        }
        D[i+n*i+nn] = 1.0;
    }

    // perform the reductions
    for( i = 0; i < n; i++ ) // For each row
    {
        alpha = D[i*n+i]; // Get diagonal value

        // Make sure it is not 0. If so the matrix is singular and we will not
        // invert it.
        // A non-singular matrix is one where inv(inv(A)) = A
        if(!alpha)
            return false;

        // For each column in this row divide though with the diagonal value so
        // so alpha becomes 1.
        for( j = 0; j < n2; j++ )
        {
            D[i+n*j] /= alpha;
        }

        // Update all other rows.
        for( k = 0; k < n; k++ )
        {
            if( (k-i) != 0 )
            {
                beta = D[k+n*i];
                for( j = i; j < n2; j++ )
                {
                    D[k+n*j] -= beta*D[i+n*j];
                }
            }
        }
    }

    // Copy result from D to m

```

```

        memcpy(m,&D[nn],sizeof(float)*nn);

        return true;
    }

void transpose()
{
    float temp;
    temp = m[1]; m[1] = m[4]; m[4] = temp;
    temp = m[2]; m[2] = m[8]; m[8] = temp;
    temp = m[3]; m[3] = m[12]; m[12] = temp;
    temp = m[6]; m[6] = m[9]; m[9] = temp;
    temp = m[7]; m[7] = m[13]; m[13] = temp;
    temp = m[11]; m[11] = m[14]; m[14] = temp;
}
};

#endif

```

Vector.cpp

```

/////////////////////////////////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A generic float vector class.
//
/////////////////////////////////////////////////////////////////

#include "Vector.h"

ostream& operator<<(ostream& os, Vector& v)
{
    os << "(" << v.x << ", " << v.y << ", " << v.z << ")";
    return os;
}

```

Vector.h

```

/////////////////////////////////////////////////////////////////
// Untweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A 3d vector float class.
//
/////////////////////////////////////////////////////////////////

#ifndef VECTOR_CLASS
#define VECTOR_CLASS

#include "general.h"
#include "Matrix.h"
#include <math.h>
#include <stddef.h>
#include <ostream.h>

class Vector
{
public:
    float x;
    float y;
    float z;

    Vector(void) : x(0), y(0), z(0) {}
    Vector(const float x, const float y, const float z) : x(x), y(y), z(z) {}
    Vector(const Vector& v) : x(v.x), y(v.y), z(v.z) {}

    inline Vector& operator+=(const Vector& v)
    {

```

```

    x += v.x;
    y += v.y;
    z += v.z;
    return *this;
}

inline Vector operator+(const Vector& v) const
{
    return Vector(x+v.x,y+v.y,z+v.z);
}

inline Vector operator-(const Vector& v) const
{
    return Vector(x-v.x,y-v.y,z-v.z);
}

inline Vector operator/(const float c) const
{
    return Vector(x/c,y/c,z/c);
}

inline Vector operator-(void) const
{
    return Vector(-x,-y,-z);
}

inline Vector operator*(const float c) const
{
    return Vector(x*c,y*c,z*c);
}

inline Vector& operator--(const Vector& v)
{
    x -= v.x;
    y -= v.y;
    z -= v.z;
    return *this;
}

inline Vector& operator*=(const float c)
{
    x *= c;
    y *= c;
    z *= c;
    return *this;
}

inline Vector& operator/=(const float c)
{
    x /= c;
    y /= c;
    z /= c;
    return *this;
}

// Copy vector
inline Vector& operator=(const Vector& v)
{
    x = v.x;
    y = v.y;
    z = v.z;
    return *this;
}

// Get 2-norm
inline float getLength(void) const
{
    return (float)sqrt(x*x+y*y+z*z);
}

inline float getLengthSquared(void) const

```

```

{
    return x*x+y*y+z*z;
}

// Get max of each element (not the same as infinity norm)
inline float getMax(void) const
{
    return max(max(x,y),z);
}

inline float getInfinityNorm(void) const
{
    return max(max(Abs(x),Abs(y)),Abs(z));
}

// Get 2-norm but before the square root is taken
inline float getSquaredLength(void) const
{
    return x*x+y*y+z*z;
}

// Return a normalized version (unit length) of the
// vector.
inline Vector normalized(void) const
{
    float l = getLength();
    return Vector(x / l, y / l, z / l);
}

inline Vector& normalize(void)
{
    float l = getLength();
    x /= l;
    y /= l;
    z /= l;
    return *this;
}

inline void invert(void)
{
    x = -x;
    y = -y;
    z = -z;
}

inline Vector inverted(void) const
{
    return Vector(-x,-y,-z);
}

inline Vector cross(const Vector& v) const
{
    return Vector(y*v.z-z*v.y,z*v.x-x*v.z,x*v.y-y*v.x);
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
inline bool isInRectangle(const Vector& lowerCorner,
                          const Vector& upperCorner) const
{
    return (x >= lowerCorner.x && y >= lowerCorner.y && z >= lowerCorner.z &&
           x < upperCorner.x && y < upperCorner.y && z < upperCorner.z);
}

inline void MultLeftMatrix(const Matrix& ma)
{
    const float* m = ma.m;
    float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
}

```

```

float w1 = m[3] * x + m[7] * y + m[11] * z + m[15];
x = x1;
y = y1;
z = z1;
if (w1!=1)
{
    x/=w1;
    y/=w1;
    z/=w1;
}
}

// Do matrix result, but do not divide through with w
// afterwards. Just throw w away (it is in fact never
// calculated. The last row of ma is thus not used.
inline void MultLeftMatrixIgnoreW(const Matrix& ma)
{
    const float* m = ma.m;
    float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
    x = x1;
    y = y1;
    z = z1;
}

inline float dot(const Vector& v) const
{
    return x * v.x + y * v.y + z * v.z;
}

friend ostream& operator<<(ostream&, Vector& v);
};

#endif

```

Part 2: Java source code

The source code consists of the following files:

Arguments.java	This file is used when parsing command line arguments
BoxData.java	This class represents the data generated to represent a box.
EventHandling.java	The class that handles all the call backs. This is similar to the call back functions in GLUT. We also do the scene generation here.
Frustum.java	This class represents a view frustum. It stores the 6 planes in world coordinates.
Main.java	The main class. Start this to run the application.
Matrix.java	A generic 4x4 matrix float class.
Node.java	A node in the scene graph / kd-tree.
Vector.java	A generic 3D vector class of floats.
gl4java/utils/textures/ BmpTextureLoader.java	This is a BMP loader for gl4Java. It has been submitted for inclusion in later versions of GL4Java.

Arguments.java

```
////////////////////////////////////  
// Untweaked virtual world (GL4Java version)  
// by Jacob Marner. Feb 2002.  
//  
// This file is used when parsing command line arguments  
//  
////////////////////////////////////  
  
public class Arguments  
{  
    String[] args;  
  
    public Arguments(String[] args)  
    {  
        this.args = args;  
    }  
  
    public boolean getBooleanFlag(String label)  
    {  
        for (int i = 0; i < args.length; i++)  
        {  
            if (args[i].equals(label))  
            {  
                return true;  
            }  
        }  
    }  
}
```

```

    }
  }
  return false;
}

public int getIntegerArgument(String label, int defaultvalue)
{
  for (int i = 0; i < args.length; i++)
  {
    if (args[i].startsWith(label))
    {
      String s = args[i].substring(label.length());
      try
      {
        return Integer.parseInt(s);
      }
      catch (NumberFormatException e)
      {
        return defaultvalue;
      }
    }
  }
  return defaultvalue;
}

public float getFloatArgument(String label, float defaultvalue)
{
  for (int i = 0; i < args.length; i++)
  {
    if (args[i].startsWith(label))
    {
      String s = args[i].substring(label.length());
      try
      {
        return Float.parseFloat(s);
      }
      catch (NumberFormatException e)
      {
        return defaultvalue;
      }
    }
  }
  return defaultvalue;
}
}
}

```

BoxData.java

```

////////////////////////////////////
// Untweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// This class represent the data generated to represent
// a box.
// These are converted to 3D geometry when the kd-tree
// (scene graph) is built.
////////////////////////////////////

public class BoxData
{
  public float scale;
  public Vector position = new Vector();
  public Vector rotation = new Vector();

  public void assign(BoxData b)
  {
    scale = b.scale;
    position.assign(b.position);
    rotation.assign(b.rotation);
  }
}

```



```

public void init(GLDrawable drawable)
{
    gl = drawable.getGL();
    glu = drawable.getGLU();
    glj = drawable.getGLContext();
    Node.gl = gl;

    // This Will Clear The Background Color To Black
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

    gl.glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    // Turn on z-buffer for hidden surface removal
    gl.glEnable(GL_DEPTH_TEST);

    // Enable back face culling.
    gl.glCullFace(GL_BACK);
    gl.glEnable(GL_CULL_FACE);

    // Tell OpenGL to use flat shading
    gl.glShadeModel(GL_FLAT);

    // Enable Phong lighting model and one global
    // directional light.
    gl.glEnable(GL_LIGHTING);
    gl.glEnable(GL_LIGHT0);

    BmpTextureLoader texLoader = new BmpTextureLoader(gl, glu);
    texLoader.readTexture("glow.bmp");

    if (!texLoader.isOk())
    {
        System.err.println("Error loading texture.");
        System.exit(1);
    }

    //Create Texture
    gl.glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    gl.glGenTextures(1, tex);
    gl.glBindTexture(GL_TEXTURE_2D, tex[0]);

    glu.gluBuild2DMipmaps(
        GL_TEXTURE_2D,
        GL_RGB,
        texLoader.getImageWidth(),
        texLoader.getImageHeight(),
        texLoader.getGLFormat(),
        texLoader.getGLComponentFormat(),
        texLoader.getTexture());

    // Enable texture mapping
    gl.glEnable(GL_TEXTURE_2D);
    // Blend lighting a texture effects.
    gl.glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    // Enable trilinear filtering.
    gl.glTexParameteri(
        GL_TEXTURE_2D,
        GL_TEXTURE_MIN_FILTER,
        GL_LINEAR_MIPMAP_LINEAR);
    gl.glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    // Enable fogging
    gl.glEnable(GL_FOG);
    float[] fogColor = { 0.0f, 0.0f, 0.0f, 0.0f };
    gl.glFogi(GL_FOG_MODE, GL_LINEAR);
    gl.glFogfv(GL_FOG_COLOR, fogColor);
    gl.glFogf(GL_FOG_DENSITY, 0.35f);
    gl.glHint(GL_FOG_HINT, GL_DONT_CARE);
    gl.glFogf(GL_FOG_START, (float) (frustumFar * 0.85f));
    gl.glFogf(GL_FOG_END, (float) frustumFar);

```

```

// Generate numBoxes box data put them in array. For each one
// store scale(1), pos(3), rotation(3). Generate each number
// randomly.
rand = new Random(new Date().getTime());

BoxData[] boxData = new BoxData[numBoxes];

int i;
for (i = 0; i < numBoxes; ++i)
{
    boxData[i] = new BoxData();
    boxData[i].scale = getRandomFloat(0.1f, 1.5f);
    boxData[i].position.x = getRandomFloat(0.0f, WORLD_MAX_X);
    boxData[i].position.y = getRandomFloat(0.0f, WORLD_MAX_Y);
    boxData[i].position.z = getRandomFloat(0.0f, WORLD_MAX_Z);
    boxData[i].rotation.x = getRandomFloat(0.0f, 360.0f);
    boxData[i].rotation.y = getRandomFloat(0.0f, 360.0f);
    boxData[i].rotation.z = getRandomFloat(0.0f, 360.0f);
}

root = new Node(boxData, 0, numBoxes);

startTimer();

reshape(drawable, 100, 100);
}

public void reshape(GLDrawable d, int width, int height)
{
    gl.glViewport(0, 0, width, height);

    gl.glMatrixMode(GL_PROJECTION);

    frustumLeft = -0.5;
    frustumRight = 0.5;
    frustumBottom = -0.5 * ((double) height / (double) width);
    frustumTop = 0.5 * ((double) height / (double) width);
    frustumNear = 0.5;

    gl.glLoadIdentity();

    gl.glFrustum(
        frustumLeft,
        frustumRight,
        frustumBottom,
        frustumTop,
        frustumNear,
        frustumFar);
    gl.glMatrixMode(GL_MODELVIEW);
}

float c = 1.0f;
int frame = 0;
public static int numFrames;
public static int numBoxes;

public void display(GLDrawable drawable)
{
    //Clear The Screen And The Depth Buffer
    gl.glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //Reset The View
    gl.glLoadIdentity();

    // Move camera in circle around center of world.
    float angle = (float) frame / 600.0f;
    float radius = 20;
    Vector cameraPos =
        new Vector(
            (float) (WORLD_MAX_X / 2.0f + radius * Math.cos(angle)),
            WORLD_MAX_Y / 2.0f,
            (float) (WORLD_MAX_Z / 2.0f + radius * Math.sin(angle)));
}

```

```

Vector cameraDirection =
    new Vector(
        (float) (Math.cos(angle + 3.14f / 2.0f)),
        0.0f,
        (float) (Math.sin(angle + 3.14f / 2.0f)));
Vector upVector = new Vector(0, 1, 0);

glu.gluLookAt(
    cameraPos.x,
    cameraPos.y,
    cameraPos.z,
    cameraPos.x + cameraDirection.x,
    cameraPos.y + cameraDirection.y,
    cameraPos.z + cameraDirection.z,
    upVector.x,
    upVector.y,
    upVector.z);

Frustum frustum =
    new Frustum(
        cameraPos,
        cameraDirection,
        upVector,
        (float) frustumLeft,
        (float) frustumRight,
        (float) frustumBottom,
        (float) frustumTop,
        (float) frustumNear,
        (float) frustumFar);

float[] mat_specular = { 1.0f, 1.0f, 1.0f, 1.0f };
float[] mat_shininess = { 50.0f };
float[] light_position = { 0.0f, 1.0f, -1.0f, 0.0f };
float[] diffuse_light = { 1.0f, 1.0f, 1.0f, 1.0f };
float[] specular_light = { 0.0f, 0.0f, 0.0f, 0.0f };
float[] ambient_light = { 1.0f, 1.0f, 1.0f, 1.0f };
gl.glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
gl.glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
gl.glLightfv(GL_LIGHT0, GL_POSITION, light_position);
gl.glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_light);
gl.glLightfv(GL_LIGHT0, GL_SPECULAR, specular_light);
gl.glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light);

gl.glBindTexture(GL_TEXTURE_2D, tex[0]);

gl.glBegin(GL_QUADS);
    root.render(frustum);
gl.glEnd();

frame++;
if (frame >= numFrames)
{
    stopTimer();
    System.exit(0);
}
}

public void cleanup(GLDrawable drawable)
{
}

public void preDisplay(GLDrawable drawable)
{
}

public void postDisplay(GLDrawable drawable)
{
}
}

```



```

// Calculate normal for right side
Vector rightNearPoint =
    cameraDirection.cross(cameraUp).multiply(right).add(centerOfNearPlane);
normals[1] = (rightNearPoint.subtract(cameraPos)).cross(cameraUp);

// Calculate normal for bottom side
Vector bottomNearPoint = centerOfNearPlane.add(cameraUp.multiply(bottom));
normals[2] =
    cameraDirection
        .cross(cameraUp)
        .cross(bottomNearPoint.subtract(cameraPos))
        .negate();

// Calculate normal for top side
Vector topNearPoint = centerOfNearPlane.add(cameraUp.multiply(top));
normals[3] =
    cameraDirection.cross(cameraUp).cross(topNearPoint.subtract(cameraPos));

// Calculate the center of the frustum
center = points[5].subtract(points[4]).divide(2).add(points[4]);

Vector extremeNearCorner =
    centerOfNearPlane.add(cameraUp.multiply(Math.max(top, -bottom))).add(
        cameraDirection.cross(cameraUp).multiply(Math.max(right, -left)));

Vector extremeFarCorner =
    extremeNearCorner.subtract(cameraPos).divide(pnear).multiply(pfar).add(
        cameraPos);

radiusSquared = extremeFarCorner.subtract(center).getSquaredLength();
}
}

```

Main.java

```

////////////////////////////////////
// Untweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// The main class. Start this to run the application.
//
// The application takes two optional parameters:
// -frames:<int> : The number of frames to run the animation. Default: 3000.
// -boxes:<int> : The number of boxes to generate. Default: 1000.
//
////////////////////////////////////

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

import gl4java.*;
import gl4java.awt.*;
import gl4java.drawable.*;

public class Main extends JFrame
{
    //Our rendering canvas
    //We are using GLAnimCanvas because we want the canvas
    //to be constantly redrawn
    GLAnimCanvas canvas = null;

    public Main(String s)
    {
        super(s);
    }

    public static void main(String args[])
    {

```

```

Main f = new Main("Virtual World");

Arguments arg = new Arguments(args);

EventHandling.numBoxes = arg.getIntegerArgument("-boxes:", 1000);
EventHandling.numFrames = arg.getIntegerArgument("-frames:", 3000);

GLContext.gljNativeDebug = false;
GLContext.gljThreadDebug = false;
GLContext.gljClassDebug = false;

f.addWindowListener(new WindowAdapter()
{
    public void windowClosed(WindowEvent e)
    {
        System.exit(0);
    }
    public void windowClosing(WindowEvent e)
    {
        windowClosed(e);
    }
});

f.getContentPane().setLayout(new BorderLayout());

// We pack the frame - otherwise the insets are not
// initialized.
f.pack();

// Set the size of the Window
Insets i = f.getInsets();
f.setBounds(0, 0, 640 + i.right + i.left, 480 + i.bottom + i.top);

// Create the 3D canvas
Dimension d = f.getSize();
GLCapabilities caps = new GLCapabilities();
GLAnimCanvas canvas = f.canvas =
    GLDrawableFactory.getFactory()
        .createGLAnimCanvas(caps, d.width, d.height);

// Create and add the class with the callback methods
EventHandling demo = new EventHandling();
canvas.addGLEventListener(demo);
// Turn off vertical sync and any other pauses.
canvas.setUseRepaint(false);
canvas.setUseFpsSleep(false);
canvas.setUseYield(false);

canvas.requestFocus();
f.getContentPane().add("Center", canvas);
canvas.start();

f.setVisible(true);
}
}

```

Matrix.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Untweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// A generic 4x4 matrix float class.
//
// All operations are applied such that the
// Matrix itself is the Left operand.
//
// The internal matrix representation is directly
// compatible with that of OpenGL, so you can use
// glLoadMatrix() on m to import the matrix into

```

```

// OpenGL.
//
////////////////////////////////////

class Matrix
{
    public float[] m = new float[16];

    public Matrix()
    {
    }

    public void loadIdentity()
    {
        m[0] = m[5] = m[10] = m[15] = 1;
        m[1] = m[2] = m[3] = m[4] = m[6] = m[7]
            = m[8] = m[9] = m[11] = m[12] = m[13] = m[14] = 0;
    }

    public Matrix(float m0, float m1, float m2, float m3, float m4,
        float m5, float m6, float m7, float m8, float m9, float m10,
        float m11, float m12, float m13, float m14, float m15)
    {
        m[0] = m0;
        m[1] = m1;
        m[2] = m2;
        m[3] = m3;
        m[4] = m4;
        m[5] = m5;
        m[6] = m6;
        m[7] = m7;
        m[8] = m8;
        m[9] = m9;
        m[10] = m10;
        m[11] = m11;
        m[12] = m12;
        m[13] = m13;
        m[14] = m14;
        m[15] = m15;
    }

    public Matrix(Matrix ma)
    {
        for (int i = 0; i < 16; i++)
            m[i] = ma.m[i];
    }

    public void assign(Matrix ma)
    {
        for (int i = 0; i < 16; i++)
            m[i] = ma.m[i];
    }

    public void mult(Matrix ma)
    {
        float[] n = ma.m;
        float[] t = new float[12];
        t[0] = m[0] * n[0] + m[4] * n[1] + m[8] * n[2] + m[12] * n[3];
        t[1] = m[1] * n[0] + m[5] * n[1] + m[9] * n[2] + m[13] * n[3];
        t[2] = m[2] * n[0] + m[6] * n[1] + m[10] * n[2] + m[14] * n[3];
        t[3] = m[3] * n[0] + m[7] * n[1] + m[11] * n[2] + m[15] * n[3];
        t[4] = m[0] * n[4] + m[4] * n[5] + m[8] * n[6] + m[12] * n[7];
        t[5] = m[1] * n[4] + m[5] * n[5] + m[9] * n[6] + m[13] * n[7];
        t[6] = m[2] * n[4] + m[6] * n[5] + m[10] * n[6] + m[14] * n[7];
        t[7] = m[3] * n[4] + m[7] * n[5] + m[11] * n[6] + m[15] * n[7];
        t[8] = m[0] * n[8] + m[4] * n[9] + m[8] * n[10] + m[12] * n[11];
        t[9] = m[1] * n[8] + m[5] * n[9] + m[9] * n[10] + m[13] * n[11];
        t[10] = m[2] * n[8] + m[6] * n[9] + m[10] * n[10] + m[14] * n[11];
        t[11] = m[3] * n[8] + m[7] * n[9] + m[11] * n[10] + m[15] * n[11];
        m[12] = m[0] * n[12] + m[4] * n[13] + m[8] * n[14] + m[12] * n[15];
        m[13] = m[1] * n[12] + m[5] * n[13] + m[9] * n[14] + m[13] * n[15];
    }
}

```

```

    m[14] = m[2] * n[12] + m[6] * n[13] + m[10] * n[14] + m[14] * n[15];
    m[15] = m[3] * n[12] + m[7] * n[13] + m[11] * n[14] + m[15] * n[15];
    for (int i = 0; i < 12; i++)
        m[i] = t[i];
}

public void rotate(float a, float x, float y, float z)
{
    // Normalize axis vector
    float slength = x * x + y * y + z * z;
    if (slength != 1.0f)
    {
        float length = (float) Math.sqrt(slength);
        x /= length;
        y /= length;
        z /= length;
    }

    float rad = (float) Math.toRadians(a);

    float c = (float) Math.cos(rad);
    float s = (float) Math.sin(rad);
    float oneminusc = 1.0f - c;
    float xy = x * y;
    float yz = y * z;
    float xz = x * z;
    float xs = x * s;
    float ys = y * s;
    float zs = z * s;

    float[] n = new float[11];
    n[0] = x * x * oneminusc + c;
    n[1] = xy * oneminusc + zs;
    n[2] = xz * oneminusc - ys;
    // n[3] not used
    n[4] = xy * oneminusc - zs;
    n[5] = y * y * oneminusc + c;
    n[6] = yz * oneminusc + xs;
    // n[7] not used
    n[8] = xz * oneminusc + ys;
    n[9] = yz * oneminusc - xs;
    n[10] = z * z * oneminusc + c;

    /* m[12] to m[15] are not changed by a rotate */
    float[] t = new float[8];
    t[0] = m[0] * n[0] + m[4] * n[1] + m[8] * n[2];
    t[1] = m[1] * n[0] + m[5] * n[1] + m[9] * n[2];
    t[2] = m[2] * n[0] + m[6] * n[1] + m[10] * n[2];
    t[3] = m[3] * n[0] + m[7] * n[1] + m[11] * n[2];
    t[4] = m[0] * n[4] + m[4] * n[5] + m[8] * n[6];
    t[5] = m[1] * n[4] + m[5] * n[5] + m[9] * n[6];
    t[6] = m[2] * n[4] + m[6] * n[5] + m[10] * n[6];
    t[7] = m[3] * n[4] + m[7] * n[5] + m[11] * n[6];
    m[8] = m[0] * n[8] + m[4] * n[9] + m[8] * n[10];
    m[9] = m[1] * n[8] + m[5] * n[9] + m[9] * n[10];
    m[10] = m[2] * n[8] + m[6] * n[9] + m[10] * n[10];
    m[11] = m[3] * n[8] + m[7] * n[9] + m[11] * n[10];
    for (int i = 0; i < 8; i++)
        m[i] = t[i];
}

public void translate(float x, float y, float z)
{
    /* m[0] to m[11] are not changed by a translate */
    m[12] = m[0] * x + m[4] * y + m[8] * z + m[12];
    m[13] = m[1] * x + m[5] * y + m[9] * z + m[13];
    m[14] = m[2] * x + m[6] * y + m[10] * z + m[14];
    m[15] = m[3] * x + m[7] * y + m[11] * z + m[15];
}

public void scale(float x, float y, float z)

```

```

{
  /* m[12] to m[15] are not changed by a scale */
  m[0] *= x;
  m[1] *= x;
  m[2] *= x;
  m[3] *= x;
  m[4] *= y;
  m[5] *= y;
  m[6] *= y;
  m[7] *= y;
  m[8] *= z;
  m[9] *= z;
  m[10] *= z;
  m[11] *= z;
}

// Returns true at success or false if the matrix was singular
public boolean invert()
{
  final int n = 4;
  final int n2 = 2 * n;
  final int nn = n * n;

  float alpha;
  float beta;
  int i;
  int j;
  int k;

  float[] D = new float[n2 * n];
  for (i = 0; i < nn; ++i)
    D[i] = m[i];

  // init the reduction matrix
  for (i = 0; i < n; i++)
  {
    for (j = 0; j < n; j++)
    {
      D[i + n * j + nn] = 0.0f;
    }
    D[i + n * i + nn] = 1.0f;
  }

  // perform the reductions
  for (i = 0; i < n; i++) // For each row
  {
    alpha = D[i * n + i]; // Get diagonal value

    // Make sure it is not 0. If so the matrix is singular and we will not
    // invert it.
    // A non-singular matrix is one where  $\text{inv}(\text{inv}(A)) = A$ 
    if (alpha == 0.0f)
      return false;

    // For each column in this row divide though with the diagonal value so
    // so alpha becomes 1.
    for (j = 0; j < n2; j++)
    {
      D[i + n * j] /= alpha;
    }

    // Update all other rows.
    for (k = 0; k < n; k++)
    {
      if ((k - i) != 0)
      {
        beta = D[k + n * i];
        for (j = i; j < n2; j++)
        {
          D[k + n * j] -= beta * D[i + n * j];
        }
      }
    }
  }
}

```

```

    }
}

// Copy result from D to m
for (i = 0; i < nn; i++)
    m[i] = D[i + nn];

return true;
}

public void transpose()
{
    float temp;
    temp = m[1]; m[1] = m[4]; m[4] = temp;
    temp = m[2]; m[2] = m[8]; m[8] = temp;
    temp = m[3]; m[3] = m[12]; m[12] = temp;
    temp = m[6]; m[6] = m[9]; m[9] = temp;
    temp = m[7]; m[7] = m[13]; m[13] = temp;
    temp = m[11]; m[11] = m[14]; m[14] = temp;
}
};

```

Node.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Untweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// A node in the scene graph / kd-tree.
//
// Every node has a bounding box and zero or two
// children. Each node contains a display list.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import java.util.*;
import gl4java.GLContext;
import gl4java.awt.*;
import gl4java.*;

class Node implements GLEnum
{
    public static GLFunc gl;

    static final int MAX_BOXES_PER_LEAF = 10;

    // This method may reorder the ordering of the
    // box data array. n is the size of the array.
    public Node(BoxData boxData[], int startindex, int numindex)
    {
        boundingBox[0] = new Vector(EventHandling.WORLD_MAX_X,
                                   EventHandling.WORLD_MAX_Y,
                                   EventHandling.WORLD_MAX_Z);
        boundingBox[1] = new Vector(0, 0, 0);
        // Find and set bounding box by iterating through boxes.
        // (Boxes is centered about position and has a distance
        // from center of max sqrt(2*pow(scale,2)) )

        for (int c = startindex; c < startindex + numindex; ++c)
        {
            float boxMaxDistFromCenter =
                (float) Math.sqrt(2.0f * (float) Math.pow(boxData[c].scale, 2));
            float candidate;

            candidate = boxData[c].position.x - boxMaxDistFromCenter;
            if (candidate < boundingBox[0].x)
                boundingBox[0].x = candidate;

            candidate = boxData[c].position.y - boxMaxDistFromCenter;

```

```

    if (candidate < boundingBox[0].y)
        boundingBox[0].y = candidate;

    candidate = boxData[c].position.z - boxMaxDistFromCenter;
    if (candidate < boundingBox[0].z)
        boundingBox[0].z = candidate;

    candidate = boxData[c].position.x + boxMaxDistFromCenter;
    if (candidate > boundingBox[1].x)
        boundingBox[1].x = candidate;

    candidate = boxData[c].position.y + boxMaxDistFromCenter;
    if (candidate > boundingBox[1].y)
        boundingBox[1].y = candidate;

    candidate = boxData[c].position.z + boxMaxDistFromCenter;
    if (candidate > boundingBox[1].z)
        boundingBox[1].z = candidate;
}

boxCenter =
    boundingBox[1].subtract(boundingBox[0]).divide(2).add(boundingBox[0]);
boxRadiusSquared = boxCenter.subtract(boundingBox[0]).getSquaredLength();

// If n is more than max_boxes
if (numindex > MAX_BOXES_PER_LEAF)
{
    float xspan = boundingBox[1].x - boundingBox[0].x;
    float yspan = boundingBox[1].y - boundingBox[0].y;
    float zspan = boundingBox[1].z - boundingBox[0].z;

    int splitn;

    if (xspan >= yspan && xspan >= zspan)
    {
        // Find center along max axis
        float center = boundingBox[0].x + xspan / 2.0f;

        // Reorder box data according to center on the max axis.
        splitn = reorder(boxData, startindex, numindex, center, 0);
    }
    else
    {
        if (yspan >= zspan)
        {
            // Find center along max axis
            float center = boundingBox[0].y + yspan / 2.0f;

            // Reorder box data according to center on the max axis.
            splitn = reorder(boxData, startindex, numindex, center, 1);
        }
        else
        {
            // Find center along max axis
            float center = boundingBox[0].z + zspan / 2.0f;

            // Reorder box data according to center on the max axis.
            splitn = reorder(boxData, startindex, numindex, center, 2);
        }
    }

    // Create nodes for each child
    leftChild = new Node(boxData, startindex, splitn - startindex);
    rightChild = new Node(boxData, splitn, startindex + numindex - splitn);

    // Initialize display list that calls children display list
    displayList = gl.glGenLists(1);

    if (displayList == 0)
    {
        System.err.println("Internal error. Display used before set.");
    }
}

```

```

        System.exit(1);
    }

    gl.glNewList(displayList, GL_COMPILE);
    gl.glCallList(leftChild.displayList);
    gl.glCallList(rightChild.displayList);
    gl.glEndList();
}
else
{
    // Initialize display list to draw the boxes. This display list
    // must be executed within a begin() end() block that was started
    // with GL_QUAD

    displayList = gl.glGenLists(1);

    if (displayList == 0)
    {
        System.err.println("Internal error. Could not allocate display list.");
        System.exit(1);
    }

    gl.glNewList(displayList, GL_COMPILE);
    for (int i = startindex; i < startindex + numindex; ++i)
    {
        Matrix m = new Matrix();
        m.loadIdentity();
        m.translate(
            boxData[i].position.x,
            boxData[i].position.y,
            boxData[i].position.z);
        m.rotate(boxData[i].rotation.x, 1, 0, 0);
        m.rotate(boxData[i].rotation.y, 0, 1, 0);
        m.rotate(boxData[i].rotation.z, 0, 0, 1);
        float scale = boxData[i].scale;
        m.scale(scale, scale, scale);

        Matrix t = new Matrix(m);
        if (!t.invert())
        {
            System.err.println("Internal error. Matrix invert failed.");
            System.exit(1);
        }
        t.transpose();

        // Left side
        transformAndSetNormal(t, -1, 0, 0);
        transformAndDrawVertex(m, 1, 0, -1, -1, 1);
        transformAndDrawVertex(m, 1, 1, -1, 1, 1);
        transformAndDrawVertex(m, 0, 1, -1, 1, -1);
        transformAndDrawVertex(m, 0, 0, -1, -1, -1);

        // Front side
        transformAndSetNormal(t, 0, 0, 1);
        transformAndDrawVertex(m, 1, 0, 1, -1, 1);
        transformAndDrawVertex(m, 1, 1, 1, 1, 1);
        transformAndDrawVertex(m, 0, 1, -1, 1, 1);
        transformAndDrawVertex(m, 0, 0, -1, -1, 1);

        // Right side
        transformAndSetNormal(t, 1, 0, 0);
        transformAndDrawVertex(m, 1, 0, 1, -1, -1);
        transformAndDrawVertex(m, 1, 1, 1, 1, -1);
        transformAndDrawVertex(m, 0, 1, 1, 1, 1);
        transformAndDrawVertex(m, 0, 0, 1, -1, 1);

        // Top side
        transformAndSetNormal(t, 0, 1, 0);
        transformAndDrawVertex(m, 1, 0, 1, 1, 1);
        transformAndDrawVertex(m, 1, 1, 1, 1, -1);
        transformAndDrawVertex(m, 0, 1, -1, 1, -1);
    }
}

```

```

        transformAndDrawVertex(m, 0, 0, -1, 1, 1);

        // Back side
        transformAndSetNormal(t, 0, 0, -1);
        transformAndDrawVertex(m, 1, 0, -1, -1, -1);
        transformAndDrawVertex(m, 1, 1, -1, 1, -1);
        transformAndDrawVertex(m, 0, 1, 1, 1, -1);
        transformAndDrawVertex(m, 0, 0, 1, -1, -1);

        // Under side
        transformAndSetNormal(t, 0, -1, 0);
        transformAndDrawVertex(m, 1, 0, 1, -1, -1);
        transformAndDrawVertex(m, 1, 1, 1, -1, 1);
        transformAndDrawVertex(m, 0, 1, -1, -1, 1);
        transformAndDrawVertex(m, 0, 0, -1, -1, -1);
    }
    gl.glEndList();
}
}

protected void finalize()
{
    gl.glDeleteLists(displayList, 1);
}

// First min box then max box
Vector[] boundingBox = new Vector[2];

boolean isInside;

// We do a frustum culling algorithm, consisting of
// 1. Bounding spheres for the frustum and bounding box
// is precalculated. These are used to do a bounding sphere
// culling.
// 2. Plane to point comparison to see if the box
// is entirely outside the plane. We only test to points
// per bounding box for each plane.
// 3. Temporal coherence culling. We remember what plane
// was used to cull this the last time and we start with
// that.
// Note: We do not use the octant test to reduce the plane
// comparisons needed to 3 since that requires a symmetric
// frustum.
public void render(Frustum frustum)
{
    isInside = true;

    int plane;

    if ((boxCenter.subtract(frustum.center)).getSquaredLength()
        > frustum.radiusSquared + boxRadiusSquared)
        return;

    if (basicCullTest(lastPlaneCulled, frustum))
        return;

    for (plane = 0; plane < 6; plane++)
    {
        if (plane == lastPlaneCulled)
            continue;

        if (basicCullTest(plane, frustum))
        {
            lastPlaneCulled = plane;
            return;
        }
    }

    // Is the box fully within the frustum? If yes, simply render the
    // display list.
    if (isInside)

```

```

    {
        gl.glCallList(displayList);
        return;
    }

    // We are now in doubt whether the box should be drawn.
    // If it is a leaf draw it all, if not go to children nodes.
    if (leftChild == null)
    {
        gl.glCallList(displayList);
    }
    else
    {
        leftChild.render(frustum);
        rightChild.render(frustum);
    }
}

int displayList;

Node leftChild = null;
Node rightChild = null;

int lastPlaneCulled = 0;

private void transformAndDrawVertex(
    Matrix m,
    float texx,
    float texy,
    float posx,
    float posy,
    float posz)
{
    gl.glTexCoord2f(texx, texy);
    Vector w = new Vector(posx, posy, posz);
    w.MultLeftMatrix(m);
    gl.glVertex3f(w.x, w.y, w.z);
}

private void transformAndSetNormal(Matrix t, float x, float y, float z)
{
    Vector v = new Vector(x, y, z);
    v.MultLeftMatrixIgnoreW(t);
    v.normalize();
    gl.glNormal3f(v.x, v.y, v.z);
}

static private boolean isOutside(
    float x,
    float y,
    float z,
    Vector point,
    Vector normal)
{
    return normal.dot(new Vector(x, y, z).subtract(point)) >= 0;
}

private boolean basicCullTest(int plane, Frustum frustum)
{
    // Find corner furthest from plane in normals direction.
    int nx = frustum.normals[plane].x >= 0 ? 1 : 0;
    int ny = frustum.normals[plane].y >= 0 ? 1 : 0;
    int nz = frustum.normals[plane].z >= 0 ? 1 : 0;

    // Check if corner furthest from plane in normals direction is outside.
    // If not it is fully inside.
    if (isOutside(boundingBox[nx].x,
        boundingBox[ny].y,
        boundingBox[nz].z,
        frustum.points[plane],
        frustum.normals[plane]))

```

```

    {
        // Check if corner closest to plane in normal's direction is outside.
        // If yes, the box is fully outside.
        // If no, the box is intersecting the frustum border.
        if (isOutside(boundingBox[1 - nx].x,
            boundingBox[1 - ny].y,
            boundingBox[1 - nz].z,
            frustum.points[plane],
            frustum.normals[plane]))
        {
            return true;
        }
        else
            isInside = false;
    }

    return false;
}

Vector boxCenter;
float boxRadiusSquared;

// Reorder boxData array according to value at given offset. If offset is 0 then
// x is used. 1 gives y. 2 gives z.
static int reorder(
    BoxData boxData[],
    int startindex,
    int numindex,
    float splitvalue,
    int offset)
{
    BoxData temp = new BoxData();

    if (numindex < 1)
        return 0;

    int r = startindex + numindex - 1;
    int p = startindex;

    int i = p - 1;
    int j = r + 1;
    while (true)
    {
        i++;
        j--;

        while (j >= p && boxData[j].position.index(offset) > splitvalue)
            j--;

        while (i <= r && boxData[i].position.index(offset) < splitvalue)
            i++;

        if (i < j)
        {
            temp.assign(boxData[i]);
            boxData[i].assign(boxData[j]);
            boxData[j].assign(temp);
        }
        else
        {
            while (j < r && boxData[j].position.index(offset) < splitvalue)
                j++;
            if (j <= p)
                j = p;

            return j;
        }
    }
};

```

Vector.java

```
////////////////////////////////////  
// Untweaked virtual world (GL4Java version)  
// by Jacob Marner. Feb 2002.  
//  
// A generic 3D vector class of floats.  
// Note that objects of this class are mutable.  
//  
////////////////////////////////////  
  
final public class Vector  
{  
    public float x;  
    public float y;  
    public float z;  
  
    public Vector()  
    {  
        x = 0;  
        y = 0;  
        z = 0;  
    }  
  
    public Vector(float x, float y, float z)  
    {  
        this.x = x;  
        this.y = y;  
        this.z = z;  
    }  
  
    public Vector(Vector v)  
    {  
        x = v.x;  
        y = v.y;  
        z = v.z;  
    }  
  
    public Vector add(Vector v)  
    {  
        return new Vector(x + v.x, y + v.y, z + v.z);  
    }  
  
    public Vector subtract(Vector v)  
    {  
        return new Vector(x - v.x, y - v.y, z - v.z);  
    }  
  
    public Vector divide(float c)  
    {  
        return new Vector(x / c, y / c, z / c);  
    }  
  
    public Vector negate()  
    {  
        return new Vector(-x, -y, -z);  
    }  
  
    public void negated()  
    {  
        x = -x;  
        y = -y;  
        z = -z;  
    }  
  
    public Vector multiply(float c)  
    {  
        return new Vector(x * c, y * c, z * c);  
    }  
  
    public void added(Vector v)
```

```

    {
        x += v.x;
        y += v.y;
        z += v.z;
    }

    public void subtracted(Vector v)
    {
        x -= v.x;
        y -= v.y;
        z -= v.z;
    }

    public void multiplied(float c)
    {
        x *= c;
        y *= c;
        z *= c;
    }

    public void divided(float c)
    {
        x /= c;
        y /= c;
        z /= c;
    }

    // Copy vector
    public void assign(Vector v)
    {
        x = v.x;
        y = v.y;
        z = v.z;
    }

    // Get 2-norm
    public float getLength()
    {
        return (float) Math.sqrt(x * x + y * y + z * z);
    }

    // Get max of each element (not the same as infinity norm)
    public float getMax()
    {
        return Math.max(Math.max(x, y), z);
    }

    public float getInfinityNorm()
    {
        return Math.max(Math.max(Math.abs(x), Math.abs(y)), Math.abs(z));
    }

    // Get 2-norm but before the square root is taken
    public float getSquaredLength()
    {
        return x * x + y * y + z * z;
    }

    // Return a normalized version (unit length) of the
    // vector.
    public Vector normalized()
    {
        float l = getLength();
        return new Vector(x / l, y / l, z / l);
    }

    public void normalize()
    {
        float l = getLength();
        x /= l;
        y /= l;
    }

```

```

    z /= 1;
}

public void invert()
{
    x = -x;
    y = -y;
    z = -z;
}

public Vector inverted()
{
    return new Vector(-x, -y, -z);
}

public Vector cross(Vector v)
{
    return new Vector(y * v.z - z * v.y, z * v.x - x * v.z, x * v.y - y * v.x);
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
public boolean isInRectangle(Vector lowerCorner, Vector upperCorner)
{
    return (
        x >= lowerCorner.x
        && y >= lowerCorner.y
        && z >= lowerCorner.z
        && x < upperCorner.x
        && y < upperCorner.y
        && z < upperCorner.z);
}

// Return x on 0, y on 1 and z on 2.
public float index(int n)
{
    switch (n)
    {
        case 0 :
            return x;
        case 1 :
            return y;
        case 2 :
            return z;
        default :
            throw new ArrayIndexOutOfBoundsException();
    }
}

// Do matrix result, but do not divide through with w
// afterwards. Just throw w away (it is in fact never
// calculated. The last row of ma is thus not used.
public void MultLeftMatrixIgnoreW(Matrix ma)
{
    float[] m = ma.m;
    float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
    x = x1;
    y = y1;
    z = z1;
}

public void MultLeftMatrix(Matrix ma)
{
    float[] m = ma.m;
    float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
    float w1 = m[3] * x + m[7] * y + m[11] * z + m[15];
}

```

```

    x = x1;
    y = y1;
    z = z1;
    if (w1 != 1)
    {
        x /= w1;
        y /= w1;
        z /= w1;
    }
}

public float dot(Vector v)
{
    return x * v.x + y * v.y + z * v.z;
}
};

```

gl4java/utls/textures/BmpTextureLoader.java

```

/////////////////////////////////////////////////////////////////
// Untweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// This is a BMP loader for gl4Java.
//
// It was not included with gl4Java 2.8.2 but I have
// submitted it for inclusion in later versions so
// other people might benefit from it.
//
// It loads Windows .BMP images according to the
// specification into textures.
//
/////////////////////////////////////////////////////////////////

package gl4java.utls.textures;

import gl4java.*;

import java.awt.*;
import java.awt.image.*;
import java.awt.Color.*;
import java.awt.event.*;
import java.applet.*;
import java.io.*;
import java.net.*;

/**
 * This class implements a Window .BMP
 * texture loader. This loader implements the
 * full .BMP specification.
 *
 * Before using the loaded texture data you
 * should call
 * glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
 * The rest of the PixelStore
 * arguments should stay at the default.
 *
 * The Windows BMP format does not support
 * multiple layers or transparency. It supports
 * 1-bit, 4-bit, 8-bit and 24-bit colors.
 * The 4-bit and 8-bit mode can be RLE encoded.
 * This loader supports all the variants.
 *
 * @author Jacob Marner jacob@marner.dk
 * @see IOTextureLoader
 * @see TextureLoader
 */
public class BmpTextureLoader extends IOTextureLoader
{
    public BmpTextureLoader(GLFunc gl, GLUFunc glu)

```

```

    {
        super(gl, glu);
    }

protected boolean readTexture(InputStream is)
{
    byte[][] bitmapData = new byte[1][];
    int res = loadBMP(is, bitmapData);

    if (res != SUCCESS)
    {
        error = true;
        return false;
    }

    imageWidth = width;
    imageHeight = height;

    glFormat = GL_RGB;

    pixel = bitmapData[0];

    setTextureSize();
    return true;
}

// The operation ended successfully
private final static int SUCCESS = 0;

// The file could not be found or it could not be opened for reading.
private final static int COULD_NOT_FIND_OR_READ_FILE = 1;

// The file does not comply with the specification.
private final static int ILLEGAL_FILE_FORMAT = 2;

// OpenGL could not accept the texture. You probably used a internal
// format not accepted by your OpenGL implementation or you may have run
// out of available texture names.
private final static int OPENGL_ERROR = 3;

// The system ran out of heap memory during the texture load.
private final static int OUT_OF_MEMORY = 4;

private InputStream file;

// The offset from the BITMAPFILEHEADER structure
// to the actual bitmap data in the file.
private int byteOffset;

// Image width in pixels
private int width;

// Image height in pixels
private int height;

// Number of bit per pixel. Is 1, 4, 8, 24. If it is 1,4 or 8 then
// images is paletted, othewise not.
private int bitCount;

// Compression
private final static int BI_RGB = 0;
private final static int BI_RLE8 = 1;
private final static int BI_RLE4 = 2;
private static int compression;

// Palette used for paletted images during load.
private byte[][] palette = new byte[3][256];

// The number of bytes read so far
private int bytesRead;

```

```

// Reads and returns a 32-bit value from the file.
private int read32BitValue()
{
    bytesRead += 4;

    try
    {
        int c1 = file.read();
        int c2 = file.read();
        int c3 = file.read();
        int c4 = file.read();

        return c1 + (c2 << 8) + (c3 << 16) + (c4 << 24);
    }
    catch (IOException e)
    {
        return 0;
    }
}

// Reads and returns a 16-bit value from the file
private short read16BitValue()
{
    bytesRead += 2;

    try
    {
        int c1 = file.read();
        int c2 = file.read();

        return (short) (c1 + (c2 << 8));
    }
    catch (IOException e)
    {
        return 0;
    }
}

// Reads and returns a 8-bit value from the file (0-255)
private int read8BitValue()
{
    bytesRead += 1;

    try
    {
        int c1 = file.read();

        return c1;
    }
    catch (IOException e)
    {
        return 0;
    }
}

// In Windows terms read this structure
// typedef struct tagBITMAPFILEHEADER { /* bmfh */
//     UINT    bfType;
//     DWORD   bfSize;
//     UINT    bfReserved1;
//     UINT    bfReserved2;
//     DWORD   bfOffBits;
// } BITMAPFILEHEADER;
private int readFileHeader()
{
    // Read "BM" tag in ASCII.
    if (read8BitValue() != 66 || read8BitValue() != 77)
        return ILLEGAL_FILE_FORMAT;

    // Read file size. We do not need this. Ignore.
    read32BitValue();
}

```

```

// Skip the two reserved areas
read16BitValue();
read16BitValue();

// Read the byte offset
byteOffset = read32BitValue();

return SUCCESS;
}

// In windows terms read this struct:
//typedef struct tagBITMAPINFOHEADER {    /* bmih */
//  DWORD   biSize;
//  LONG    biWidth;
//  LONG    biHeight;
//  WORD    biPlanes;
//  WORD    biBitCount;
//  DWORD   biCompression;
//  DWORD   biSizeImage;
//  LONG    biXPelsPerMeter;
//  LONG    biYPelsPerMeter;
//  DWORD   biClrUsed;
//  DWORD   biClrImportant;
//} BITMAPINFOHEADER;
private int readInfoHeader()
{
// We expect this to be at least 40. If it is larger we read
// some more bytes in the end to be forward compatible.
int sizeofInfoHeader = (int) read32BitValue();

if (sizeofInfoHeader < 40)
return ILLEGAL_FILE_FORMAT;

width = read32BitValue();

height = read32BitValue();

// Read number of planes. According to the specification this
// must be 1.
if (read16BitValue() != 1)
return ILLEGAL_FILE_FORMAT;

bitCount = read16BitValue();

if (bitCount != 1 && bitCount != 4 && bitCount != 8 && bitCount != 24)
return ILLEGAL_FILE_FORMAT;

compression = read32BitValue();

if (compression != BI_RGB && compression != BI_RLE8 && compression != BI_RLE4)
return ILLEGAL_FILE_FORMAT;

// Read image size. We do not need this since we have the
// image size.
read32BitValue();

// Pixel to device mapping. This is irrelevant since we simply
// want the bitmap.
read32BitValue();
read32BitValue();

// Read colors used. We do not need this, so it is ignored.
read32BitValue();

// Read the number of important colors. This will not be needed
// in OpenGL so we will ignore this.
read32BitValue();

// Apply padding in end of header to be forward compatible.
sizeofInfoHeader -= 40;
}

```

```

while (sizeofInfoHeader > 0)
{
    read8BitValue();
    sizeofInfoHeader--;
}

return SUCCESS;
}

// The palette follows directly after the
// info header
//
//typedef struct tagRGBQUAD {      /* rgbq */
// BYTE    rgbBlue;
// BYTE    rgbGreen;
// BYTE    rgbRed;
// BYTE    rgbReserved;
// } RGBQUAD;
private int readPalette()
{
    // 24-bit images are not paletted.
    if (bitCount == 24)
        return SUCCESS;

    int numColors = 1 << bitCount;
    for (int i = 0; i < numColors; i++)
    {
        // Read RGB.
        for (int j = 2; j >= 0; j--)
            palette[j][i] = (byte) read8BitValue();

        // Skip reversed byte.
        read8BitValue();
    }

    return SUCCESS;
}

private int readBitmapData1Bit(byte[] bitmapData)
{
    // 1-bit format cannot be compressed
    if (compression != BI_RGB)
        return ILLEGAL_FILE_FORMAT;

    int byteRead = 0;
    for (int y = 0; y < height; y++)
    {
        int index = y * width * 3;
        for (int x = 0; x < width; x++)
        {
            if (x % 8 == 0)
            {
                byteRead = read8BitValue();
            }

            char color = (char) ((byteRead >> (7 - ((char) (x % 8)))) & 1);

            bitmapData[index + x * 3] = palette[0][color];
            bitmapData[index + x * 3 + 1] = palette[1][color];
            bitmapData[index + x * 3 + 2] = palette[2][color];
        }

        // Pad to 32-bit boundary.
        while (bytesRead % 4 != 0)
            read8BitValue();
    }

    return SUCCESS;
}

// This is called after the first byte has been found to be 0

```

```

// and the second to be 0-2. This is only used in RLE-4 and RLE-8
// encoding.
private boolean handleEscapeCode(int secondByte, int[] x, int[] y, int[] res)
{
    if (secondByte == 0x00)
    {
        // End of line
        x[0] = 0;
        if (y[0] >= height)
        {
            res[0] = ILLEGAL_FILE_FORMAT;
            return true;
        }
        (y[0]) ++;
    }
    else if (secondByte == 0x01)
    {
        // End of bitmap
        res[0] = SUCCESS;
        return true;
    }
    else // secondByte=0x02
    {
        // Delta. Move drawing cursor.
        x[0] += read8BitValue();
        y[0] += read8BitValue();
        if (x[0] >= width || x[0] < 0 || y[0] >= height || y[0] < 0)
        {
            res[0] = ILLEGAL_FILE_FORMAT;
            return true;
        }
    }

    return false;
}

// Draw a 4-bit image. Can be uncompressed or RLE-4 compressed.
private int readBitmapData4Bit(byte[] bitmapData)
{
    if (compression != BI_RGB && compression != BI_RLE4)
        return ILLEGAL_FILE_FORMAT;

    // Uncompressed 4 bit encoding
    if (compression == BI_RGB)
    {
        for (int j = 0; j < height; j++)
        {
            int index = j * width * 3;
            int byteValue = 0;
            int color;
            for (int i = 0; i < width; i++)
            {
                if (i % 2 == 0)
                {
                    byteValue = read8BitValue();
                    color = (byte) (byteValue >> 4);
                }
                else
                {
                    color = (byte) (byteValue & 0x0F);
                }

                bitmapData[index + i * 3] = palette[0][color];
                bitmapData[index + i * 3 + 1] = palette[1][color];
                bitmapData[index + i * 3 + 2] = palette[2][color];
            }

            // Pad to 32-bit boundary.
            for (int k = 0; k < (((width + 1) / 2) % 4); k++)
                read8BitValue();
        }
    }
}

```

```

}

// RLE encoded 4-bit compression
if (compression == BI_RLE4)
{
    // Drawing cursor pos
    int[] x = new int[1];
    int[] y = new int[1];
    x[0] = 0;
    y[0] = 0;

    // Clear bitmap data since it is legal not to
    // fill it all out.
    for (int i = 0; i < width * height * 3; i++)
        bitmapData[i] = 0;

    bytesRead = 0;

    while (true)
    {
        int firstByte = read8BitValue();
        int secondByte = read8BitValue();

        // Is this an escape code or absolute encoding?
        if (firstByte == 0)
        {
            // Is this an escape code
            if (secondByte <= 0x02)
            {
                int[] res = new int[1];
                if (handleEscapeCode(secondByte, x, y, res))
                    return res[0];
            }
            else
            {
                // Absolute encoding
                int index = y[0] * width * 3;
                int color;
                int databyte = 0;
                for (int i = 0; i < secondByte; i++)
                {
                    if (x[0] >= width)
                        return ILLEGAL_FILE_FORMAT;

                    if (i % 2 == 0)
                    {
                        databyte = read8BitValue();
                        color = databyte >> 4;
                    }
                    else
                    {
                        color = databyte & 0x0F;
                    }

                    bitmapData[index + x[0] * 3] = palette[0][color];
                    bitmapData[index + x[0] * 3 + 1] = palette[1][color];
                    bitmapData[index + x[0] * 3 + 2] = palette[2][color];
                    (x[0]) ++;
                }

                // Pad to 16-bit word boundary
                while (bytesRead % 2 != 0)
                    read8BitValue();
            }
        }
        else
        {
            // If not absolute or escape code, perform data decode for next
            // length of colors
            int color1 = secondByte >> 4;
            int color2 = secondByte & 0x0F;

```

```

        for (int i = 0; i < firstByte; i++)
        {
            if (x[0] >= width)
                return ILLEGAL_FILE_FORMAT;

            int color;
            if (i % 2 == 0)
                color = color1;
            else
                color = color2;

            int index = y[0] * width * 3 + x[0] * 3;
            bitmapData[index] = palette[0][color];
            bitmapData[index + 1] = palette[1][color];
            bitmapData[index + 2] = palette[2][color];
            (x[0]) ++;
        }
    }
}

return SUCCESS;
}

// Read 8-bit image. Can be uncompressed or RLE-8 compressed.
private int readBitmapData8Bit(byte[] bitmapData)
{
    if (compression != BI_RGB && compression != BI_RLE8)
        return ILLEGAL_FILE_FORMAT;

    if (compression == BI_RGB)
    {
        // For each scan line
        for (int i = 0; i < height; i++)
        {
            int index = i * width * 3;
            for (int j = 0; j < width; j++)
            {
                int color = read8BitValue();
                bitmapData[index + j * 3] = palette[0][color];
                bitmapData[index + j * 3 + 1] = palette[1][color];
                bitmapData[index + j * 3 + 2] = palette[2][color];
            }

            // go to next alignment of 4 bytes.
            for (int k = 0; k < (width % 4); k++)
                read8BitValue();
        }
    }

    if (compression == BI_RLE8)
    {
        // Drawing cursor pos
        int[] x = new int[1];
        int[] y = new int[1];
        x[0] = 0;
        y[0] = 0;

        bytesRead = 0;

        // Clear bitmap data since it is legal not to
        // fill it all out.
        for (int i = 0; i < width * height * 3; i++)
            bitmapData[i] = 0;

        while (true)
        {
            int firstByte = read8BitValue();
            int secondByte = read8BitValue();

```

```

// Is this an escape code or absolute encoding?
if (firstByte == 0)
{
    // Is this an escape code
    if (secondByte <= 0x02)
    {
        int[] res = new int[1];
        if (handleEscapeCode(secondByte, x, y, res))
            return res[0];
    }
    else
    {
        // Absolute encoding
        int index = y[0] * width * 3;
        for (int i = 0; i < secondByte; i++)
        {
            if (x[0] >= width)
                return ILLEGAL_FILE_FORMAT;
            int color = read8BitValue();
            bitmapData[index + x[0] * 3] = palette[0][color];
            bitmapData[index + x[0] * 3 + 1] = palette[1][color];
            bitmapData[index + x[0] * 3 + 2] = palette[2][color];
            (x[0]) ++;
        }

        // Pad to 16-bit word boundary
        if (secondByte % 2 == 1)
            read8BitValue();
    }
}
else
{
    // If not, perform data decode for next length of colors
    for (int i = 0; i < firstByte; i++)
    {
        if (x[0] >= width)
            return ILLEGAL_FILE_FORMAT;
        int index = y[0] * width * 3 + x[0] * 3;
        bitmapData[index] = palette[0][secondByte];
        bitmapData[index + 1] = palette[1][secondByte];
        bitmapData[index + 2] = palette[2][secondByte];
        (x[0]) ++;
    }
}
}

return SUCCESS;
}

// Load a 24-bit image. Cannot be encoded.
private int readBitmapData24Bit(byte[] bitmapData)
{
    // 24-bit bitmaps cannot be encoded. Verify this.
    if (compression != BI_RGB)
        return ILLEGAL_FILE_FORMAT;

    for (int i = 0; i < height; i++)
    {
        int index = i * width * 3;
        for (int j = 0; j < width; j++)
        {
            bitmapData[index + j * 3 + 2] = (byte) read8BitValue();
            bitmapData[index + j * 3 + 1] = (byte) read8BitValue();
            bitmapData[index + j * 3] = (byte) read8BitValue();
        }

        // go to next alignment of 4 bytes.
        for (int k = 0; k < ((width * 3) % 4); k++)
            read8BitValue();
    }
}

```

```

    return SUCCESS;
}

private int readBitmapData(byte[] bitmapData)
{
    // Pad until byteoffset. Most images will need no padding
    // since they already are made to use as little space as
    // possible.
    while (bytesRead < byteOffset)
        read8BitValue();

    // The data reading procedure depends on the bit depth.
    switch (bitCount)
    {
        case 1 :
            return readBitmapData1Bit(bitmapData);
        case 4 :
            return readBitmapData4Bit(bitmapData);
        case 8 :
            return readBitmapData8Bit(bitmapData);
        default : // 24 bit.
            return readBitmapData24Bit(bitmapData);
    }
}

// Load the BMP. Assumes that no extension is appended to the filename.
// If successful *bitmapData will contain a pointer to the data.
private int loadBMP(InputStream filep, byte[][] bitmapData)
{
    file = filep;

    bitmapData[0] = null;

    // Open file for buffered read.
    int res = SUCCESS;

    bytesRead = 0;

    // Read File Header
    if (res == 0)
        res = readFileHeader();

    // Read Info Header
    if (res == 0)
        res = readInfoHeader();

    // Read Palette
    if (res == 0)
        res = readPalette();

    if (res == 0)
    {
        // The bitmap data we are going to hand to OpenGL
        bitmapData[0] = new byte[width * height * 3];

        if (bitmapData[0] == null)
            res = OUT_OF_MEMORY;
    }

    if (res == 0)
    {
        // Read Data
        res = readBitmapData(bitmapData[0]);
    }

    return res;
}
}

```



```

// Note: Data in the .BMP format is stored in little-endian format.

// The file that the BMP is stored in.
static FILE* file;

// The offset from the BITMAPFILEHEADER structure
// to the actual bitmap data in the file.
static int byteOffset;

// Image width in pixels
static int width;

// Image height in pixels
static int height;

// Number of bit per pixel. Is 1, 4, 8, 24. If it is 1,4 or 8 then
// images is paletted, otherwise not.
static int bitCount;

// Compression
enum CompressionType {BI_RGB=0, BI_RLE8, BI_RLE4 };
static CompressionType compression;

// Palette used for paletted images during load.
static unsigned char palette[3][256];

// The number of bytes read so far
static int bytesRead;

// Reads and returns a 32-bit value from the file.
static int read32BitValue()
{
    int c1 = fgetc(file);
    int c2 = fgetc(file);
    int c3 = fgetc(file);
    int c4 = fgetc(file);

    bytesRead+=4;

    return c1 + (c2<<8) + (c3<<16) + (c4<<24);
}

// Reads and returns a 16-bit value from the file
static short read16BitValue()
{
    int c1 = fgetc(file);
    int c2 = fgetc(file);

    bytesRead+=2;

    return c1 + (c2<<8);
}

// Reads and returns a 8-bit value from the file
static unsigned char read8BitValue()
{
    unsigned int c1 = fgetc(file);

    bytesRead+=1;

    return (unsigned char)c1;
}

// In Windows terms read this structure
// typedef struct tagBITMAPFILEHEADER { /* bmfh */
//     UINT    bfType;
//     DWORD   bfSize;
//     UINT    bfReserved1;
//     UINT    bfReserved2;
//     DWORD   bfOffBits;
// } BITMAPFILEHEADER;

```

```

static LOAD_TEXTUREBMP_RESULT readFileHeader(void)
{
    // Read "BM" tag in ASCII.
    if (read8BitValue()!=66 || read8BitValue()!=77)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    // Read file size. We do not need this. Ignore.
    read32BitValue();

    // Skip the two reserved areas
    read16BitValue();
    read16BitValue();

    // Read the byte offset
    byteOffset = read32BitValue();

    return LOAD_TEXTUREBMP_SUCCESS;
}

// In windows terms read this struct:
//typedef struct tagBITMAPINFOHEADER {    /* bmih */
//    DWORD    biSize;
//    LONG     biWidth;
//    LONG     biHeight;
//    WORD     biPlanes;
//    WORD     biBitCount;
//    DWORD    biCompression;
//    DWORD    biSizeImage;
//    LONG     biXPelsPerMeter;
//    LONG     biYPelsPerMeter;
//    DWORD    biClrUsed;
//    DWORD    biClrImportant;
//} BITMAPINFOHEADER;
static LOAD_TEXTUREBMP_RESULT readInfoHeader(void)
{
    // We expect this to be at least 40. If it is larger we read
    // some more bytes in the end to be forward compatible.
    unsigned int sizeofInfoHeader = (unsigned int)read32BitValue();

    if (sizeofInfoHeader<40)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    width = read32BitValue();

    height = read32BitValue();

    // Read number of planes. According to the specification this
    // must be 1.
    if (read16BitValue()!=1)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    bitCount = read16BitValue();

    if (bitCount != 1 && bitCount != 4 && bitCount != 8 && bitCount != 24)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    compression = (CompressionType)read32BitValue();

    if (compression != BI_RGB && compression != BI_RLE8 &&
        compression != BI_RLE4)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    // Read image size. We do not need this since we have the
    // image size.
    read32BitValue();

    // Pixel to device mapping. This is irrelevant since we simply
    // want the bitmap.
    read32BitValue();
    read32BitValue();
}

```

```

// Read colors used. We do not need this, so it is ignored.
read32BitValue();

// Read the number of important colors. This will not be needed
// in OpenGL so we will ignore this.
read32BitValue();

// Apply padding in end of header to be forward compatible.
sizeofInfoHeader -= 40;
while (sizeofInfoHeader>0)
{
    read8BitValue();
    sizeofInfoHeader--;
}

return LOAD_TEXTUREBMP_SUCCESS;
}

// The palette follows directly after the
// info header
//
//typedef struct tagRGBQUAD {      /* rgbq */
// BYTE    rgbBlue;
// BYTE    rgbGreen;
// BYTE    rgbRed;
// BYTE    rgbReserved;
// } RGBQUAD;
static LOAD_TEXTUREBMP_RESULT readPalette(void)
{
    // 24-bit images are not paletted.
    if (bitCount==24)
        return LOAD_TEXTUREBMP_SUCCESS;

    int numColors = 1<<bitCount;
    for (int i=0;i<numColors;i++)
    {
        // Read RGB.
        for (int j=2;j>=0;j--)
            palette[j][i] = read8BitValue();

        // Skip reversed byte.
        read8BitValue();
    }

    return LOAD_TEXTUREBMP_SUCCESS;
}

static LOAD_TEXTUREBMP_RESULT readBitmapData1Bit(unsigned char* bitmapData)
{
    // 1-bit format cannot be compressed
    if (compression!=BI_RGB)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    unsigned char byteRead;
    for (int y=0;y<height;y++)
    {
        int index = y*width*3;
        for (int x=0;x<width;x++)
        {
            if (x%8==0)
            {
                byteRead = read8BitValue();
            }

            unsigned char color = (byteRead >> (7-(x%8))) & 1;

            bitmapData[index+x*3] = palette[0][color];
            bitmapData[index+x*3+1] = palette[1][color];
            bitmapData[index+x*3+2] = palette[2][color];
        }
    }
}

```

```

        // Pad to 32-bit boundary.
        while(bytesRead%4 != 0)
            read8BitValue();
    }

    return LOAD_TEXTUREBMP_SUCCESS;
}

// This is called after the first byte has been found to be 0
// and the second to be 0-2. This is only used in RLE-4 and RLE-8
// encoding.
static bool handleEscapeCode(int secondByte, int* x, int* y,
                             LOAD_TEXTUREBMP_RESULT* res)
{
    if (secondByte==0x00)
    {
        // End of line
        (*x)=0;
        if ((*y)>=height)
        {
            *res = LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
            return true;
        }
        (*y)++;
    }
    else if (secondByte==0x01)
    {
        // End of bitmap
        *res = LOAD_TEXTUREBMP_SUCCESS;
        return true;
    }
    else // secondByte=0x02
    {
        // Delta. Move drawing cursor.
        *x += read8BitValue();
        *y += read8BitValue();
        if (*x>=width || *x<0 || *y>=height || *y<0)
        {
            *res = LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
            return true;
        }
    }
}

return false;
}

// Draw a 4-bit image. Can be uncompressed or RLE-4 compressed.
static LOAD_TEXTUREBMP_RESULT readBitmapData4Bit(unsigned char* bitmapData)
{
    if (compression!=BI_RGB && compression!=BI_RLE4)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    // Uncompressed 4 bit encoding
    if (compression==BI_RGB)
    {
        for (int j=0;j<height;j++)
        {
            int index = j*width*3;
            unsigned char byteValue;
            unsigned char color;
            for (int i=0;i<width;i++)
            {
                if (i%2==0)
                {
                    byteValue = read8BitValue();
                    color = byteValue>>4;
                }
                else
                {
                    color = byteValue & 0x0F;
                }
            }
        }
    }
}

```

```

        bitmapData[index+i*3] = palette[0][color];
        bitmapData[index+i*3+1] = palette[1][color];
        bitmapData[index+i*3+2] = palette[2][color];
    }

    // Pad to 32-bit boundary.
    for (int k=0; k<(((width+1)/2)%4);k++)
        read8BitValue();
}
}

// RLE encoded 4-bit compression
if (compression==BI_RLE4)
{
    // Drawing cursor pos
    int x=0;
    int y=0;

    // Clear bitmap data since it is legal not to
    // fill it all out.
    memset(bitmapData,0,sizeof(unsigned char)*width*height*3);

    bytesRead=0;

    while(true)
    {
        unsigned char firstByte = read8BitValue();
        unsigned char secondByte = read8BitValue();

        // Is this an escape code or absolute encoding?
        if (firstByte==0)
        {
            // Is this an escape code
            if (secondByte<=0x02)
            {
                LOAD_TEXTUREBMP_RESULT res;
                if (handleEscapeCode(secondByte,&x,&y,&res))
                    return res;
            }
            else
            {
                // Absolute encoding
                int index = y*width*3;
                int color;
                unsigned char databyte;
                for (int i=0; i<secondByte; i++)
                {
                    if (x>=width)
                        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

                    if (i%2==0)
                    {
                        databyte = read8BitValue();
                        color = databyte >> 4;
                    }
                    else
                    {
                        color = databyte & 0x0F;
                    }

                    bitmapData[index+x*3] = palette[0][color];
                    bitmapData[index+x*3+1] = palette[1][color];
                    bitmapData[index+x*3+2] = palette[2][color];
                    x++;
                }

                // Pad to 16-bit word boundary
                while (bytesRead%2 != 0)
                    read8BitValue();
            }
        }
    }
}

```

```

    }
    else
    {
        // If not absolute or escape code, perform data decode for next
        // length of colors
        int color1 = secondByte >> 4;
        int color2 = secondByte & 0x0F;

        for (int i=0;i<firstByte;i++)
        {
            if (x>=width)
                return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

            int color;
            if (i%2==0)
                color = color1;
            else
                color = color2;

            int index = y*width*3+x*3;
            bitmapData[index] = palette[0][color];
            bitmapData[index+1] = palette[1][color];
            bitmapData[index+2] = palette[2][color];
            x++;
        }
    }
}

return LOAD_TEXTUREBMP_SUCCESS;
}

// Read 8-bit image. Can be uncompressed or RLE-8 compressed.
static LOAD_TEXTUREBMP_RESULT readBitmapData8Bit(unsigned char* bitmapData)
{
    if (compression!=BI_RGB && compression!=BI_RLE8)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    if (compression==BI_RGB)
    {
        // For each scan line
        for (int i=0;i<height;i++)
        {
            int index = i*width*3;
            for (int j=0;j<width;j++)
            {
                int color = read8BitValue();
                bitmapData[index+j*3] = palette[0][color];
                bitmapData[index+j*3+1] = palette[1][color];
                bitmapData[index+j*3+2] = palette[2][color];
            }

            // go to next alignment of 4 bytes.
            for (int k=0; k<(width%4);k++)
                read8BitValue();
        }
    }

    if (compression==BI_RLE8)
    {
        // Drawing cursor pos
        int x=0;
        int y=0;

        bytesRead=0;

        // Clear bitmap data since it is legal not to
        // fill it all out.
        memset(bitmapData,0,sizeof(unsigned char)*width*height*3);

        while(true)
    }
}

```

```

{
    unsigned char firstByte = read8BitValue();
    unsigned char secondByte = read8BitValue();

    // Is this an escape code or absolute encoding?
    if (firstByte==0)
    {
        // Is this an escape code
        if (secondByte<=0x02)
        {
            LOAD_TEXTUREBMP_RESULT res;
            if (handleEscapeCode(secondByte,&x,&y,&res))
                return res;
        }
        else
        {
            // Absolute encoding
            int index = y*width*3;
            for (int i=0; i<secondByte; i++)
            {
                if (x>=width)
                    return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
                int color = read8BitValue();
                bitmapData[index+x*3] = palette[0][color];
                bitmapData[index+x*3+1] = palette[1][color];
                bitmapData[index+x*3+2] = palette[2][color];
                x++;
            }

            // Pad to 16-bit word boundary
            if (secondByte%2 == 1)
                read8BitValue();
        }
    }
    else
    {
        // If not, perform data decode for next length of colors
        for (int i=0;i<firstByte;i++)
        {
            if (x>=width)
                return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;
            int index = y*width*3+x*3;
            bitmapData[index] = palette[0][secondByte];
            bitmapData[index+1] = palette[1][secondByte];
            bitmapData[index+2] = palette[2][secondByte];
            x++;
        }
    }
}

return LOAD_TEXTUREBMP_SUCCESS;
}

// Load a 24-bit image. Cannot be encoded.
static LOAD_TEXTUREBMP_RESULT readBitmapData24Bit(unsigned char* bitmapData)
{
    // 24-bit bitmaps cannot be encoded. Verify this.
    if (compression!=BI_RGB)
        return LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT;

    for (int i=0;i<height;i++)
    {
        int index = i*width*3;
        for (int j=0;j<width;j++)
        {
            bitmapData[index+j*3+2] = read8BitValue();
            bitmapData[index+j*3+1] = read8BitValue();
            bitmapData[index+j*3] = read8BitValue();
        }
    }
}

```

```

        // go to next alignment of 4 bytes.
        for (int k=0; k<((width*3)%4);k++)
            read8BitValue();
    }

    return LOAD_TEXTUREBMP_SUCCESS;
}

static LOAD_TEXTUREBMP_RESULT readBitmapData(unsigned char* bitmapData)
{
    // Pad until byteoffset. Most images will need no padding
    // since they already are made to use as little space as
    // possible.
    while(bytesRead<byteOffset)
        read8BitValue();

    // The data reading procedure depends on the bit depth.
    switch(bitCount)
    {
    case 1:
        return readBitmapData1Bit(bitmapData);
    case 4:
        return readBitmapData4Bit(bitmapData);
    case 8:
        return readBitmapData8Bit(bitmapData);
    default: // 24 bit.
        return readBitmapData24Bit(bitmapData);
    }
}

// Load the BMP. Assumes that no extension is appended to the filename.
// If successful *bitmapData will contain a pointer to the data.
LOAD_TEXTUREBMP_RESULT loadBMP(const char* filename,
                               unsigned char** bitmapData)
{
    *bitmapData = NULL;

    // Append .bmp extension
    char* str = new char[strlen(filename)+5];
    if (!str)
        return LOAD_TEXTUREBMP_OUT_OF_MEMORY;

    strcpy(str,filename);
    strcat(str, ".bmp");

    // Open file for buffered read.
    file = fopen(str,"rb");

    char readBuffer[BUFSIZ];

    LOAD_TEXTUREBMP_RESULT res = LOAD_TEXTUREBMP_SUCCESS;

    if (!file)
        res = LOAD_TEXTUREBMP_COULD_NOT_FIND_OR_READ_FILE;

    if (!res)
    {
        setbuf(file, readBuffer);

        bytesRead=0;
    }

    // Read File Header
    if (!res)
        res=readFileHeader();

    // Read Info Header
    if (!res)
        res=readInfoHeader();

    // Read Palette

```

```

    if (!res)
        res=readPalette();

    if (!res)
    {
        // The bitmap data we are going to hand to OpenGL
        *bitmapData = new unsigned char[width*height*3];

        if (!bitmapData)
            res = LOAD_TEXTUREBMP_OUT_OF_MEMORY;
    }

    if (!res)
    {
        // Read Data
        res=readBitmapData(*bitmapData);
    }

    // Clean up
    delete[] str;

    // Only clean up bitmapData if there was an error.
    if (*bitmapData && res)
        delete[] *bitmapData;

    if (file)
        fclose(file);

    return res;
}

// Removes the .bmp extension of the filename if it exists.
static void removeBMPextension(const char* withext, char* result)
{
    for(int i=strlen(withext)-1;i>=0;i--)
    {
        if (!strcmp(withext+i, ".bmp") || !strcmp(withext+i, ".BMP"))
        {
            for (int j=0;j<i;j++)
            {
                result[j] = withext[j];
            }
            result[i]='\0';
            return;
        }
    }
}

// The main function. This is the only one that is exported.
LOAD_TEXTUREBMP_RESULT loadOpenGL2DTextureBMP(const char* filename,
                                             GLuint *textureName,
                                             GLint internalformat)
{
    LOAD_TEXTUREBMP_RESULT res;

    char* str = new char[strlen(filename)];

    if (!str)
        return LOAD_TEXTUREBMP_OUT_OF_MEMORY;

    removeBMPextension(filename, str);

    glPushClientAttrib(GL_CLIENT_PIXEL_STORE_BIT);

    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

    glGenTextures(1, textureName);

    glBindTexture(GL_TEXTURE_2D, *textureName);

    // load bmp

```

```

unsigned char* bitmapData;
res = loadBMP(str,&bitmapData);

// load texture into OpenGL with mip maps and scale it if needed.
if (!res)
{
    gluBuild2DMipmaps(GL_TEXTURE_2D, internalformat, width, height, GL_RGB,
        GL_UNSIGNED_BYTE,bitmapData);
}

if (glGetError()!=GL_NO_ERROR)
{
    res = LOAD_TEXTUREBMP_OPENGL_ERROR;
}

// Clean up.
if (bitmapData)
    delete[] bitmapData;
glPopClientAttrib();
delete[] str;
if (res)
    glDeleteTextures(1,textureName);
return res;
}

```

BMPLoader.h

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// An opengl .BMP texture loader.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Copyright 2002 Jacob Marner. jacob@marner.dk. Released under LGPL.

#ifndef BMP_LOADER_HEADER
#define BMP_LOADER_HEADER

// Does your program not use GLUT? If not, remove the following
// include line and include the standard OpenGL headers instead,
// <GL/gl.h> and <GL/glu.h>.
// If you use Windows remember to include <windows.h> also.
#include <GL/glut.h>

// The following is the function return type. Use this to
// get information about how the loading operation went.

enum LOAD_TEXTUREBMP_RESULT {
    // The texture loading operation succeeded.
    LOAD_TEXTUREBMP_SUCCESS=0,
    // The file could not be found or it could not be opened for reading.
    LOAD_TEXTUREBMP_COULD_NOT_FIND_OR_READ_FILE,
    // The file does not comply with the specification.
    LOAD_TEXTUREBMP_ILLEGAL_FILE_FORMAT,
    // OpenGL could not accept the texture. You probably used an internal
    // format not accepted by your OpenGL implementation or you may have run
    // out of available texture names.
    LOAD_TEXTUREBMP_OPENGL_ERROR,
    // The system ran out of heap memory during the texture load.
    LOAD_TEXTUREBMP_OUT_OF_MEMORY};

// Loads the specified BMP image and stores it into an OpenGL 2D texture, whose
// name will be returned in textureName. If an error occurred, textureName is
// undefined. The status of the operation will be returned as the
// return value.

// The bitmap will be loaded without borders and all mip maps will be
// generated automatically. If you need the bitmap for anything else than
// textures, or you need borders you will have to edit the source code.

```

```

// Parameters:
// [in] filename: The name of the texture file. Must be Windows .BMP format.
// The actual file *must* use the .bmp extension. For the filename string
// given as parameter the extension is optional - if it is not there
// it is appended automatically.
// [in/out] textureName: Pass a pointer to an already allocated GLuint and if
// the operation success then this data will point to the texture name
// of the texture object loaded. This can immediately be used with
// glBindTexture() to use the texture during drawing. If the return value
// is not LOAD_TEXTUREBMP_SUCCESS then this value is undefined. If you
// later want to delete the loaded texture call glDeleteTextures() on the
// returned texture name.
// [in] internalformat: The internal format of the loaded texture. Is passed
// directly to glTexImage2D(). Defaults to GL_RGB.
// Return value:
// [out] Status of operation.

LOAD_TEXTUREBMP_RESULT loadOpenGL2DTextureBMP(const char* filename,
                                              GLuint* textureName,
                                              GLint internalformat = GL_RGB);

#endif

```

BoxTree.cpp

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A node in the kd-tree used for culling.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include <iostream.h>
#include "BoxTree.h"
#include "Matrix.h"

// Reorder the box data in place according to position.x such that all with
// values below split value comes first.
static int reorderx(BoxData boxData[], int n, float splitvalue);
static int reordery(BoxData boxData[], int n, float splitvalue);
static int reorderz(BoxData boxData[], int n, float splitvalue);

static inline bool isOutside(float x, float y, float z, const Vector& point,
                            const Vector& normal)
{
    Vector v(x,y,z);
    v-=point;
    return normal.dot(v)>=0;
}

static bool isInside;

bool Node::basicCullTest(int plane, const Frustum& frustum)
{
    // Find corner furthest from plane in normals direction.
    const Vector& normal = frustum.normals[plane];

    int nx = normal.x >= 0 ? 1 : 0;
    int ny = normal.y >= 0 ? 1 : 0;
    int nz = normal.z >= 0 ? 1 : 0;

    // Check if corner furthest from plane in normals direction is outside.
    // If not it is fully inside.
    if (isOutside(boundingBox[nx].x, boundingBox[ny].y, boundingBox[nz].z,
                frustum.points[plane],
                normal))
    {
        // Check if corner closest to plane in normal's direction is outside.

```

```

    // If yes, the box is fully outside.
    // If no, the box is intersecting the frustum border.
    if (isOutside(boundingBox[1-nx].x, boundingBox[1-ny].y,
        boundingBox[1-nz].z,
        frustum.points[plane],
        normal))
    {
        return true;
    }
    else
        isInside=false;
}

return false;
}

// We do a frustum culling algorithm, consisting of
// 1. Bounding spheres for the frustum and bounding box
// is precalculated. These are used to do a bounding sphere
// culling.
// 2. Plane to point comparison to see if the box
// is entirely outside the plane. We only test to points
// per bounding box for each plane.
// 3. Temporal coherence culling. We remember what plane
// was used to cull this the last time and we start with
// that.
// Note: We do not use the octant test to reduce the plane
// comparisons needed to 3 since that requires a symmetric
// frustum.
void Node::render(const Frustum& frustum)
{
    isInside = true;

    int plane;

    if ((boxCenter - frustum.center).getLengthSquared() >
        frustum.radiusSquared + boxRadiusSquared)
        return;

    if (basicCullTest(lastPlaneCulled, frustum))
        return;

    for (plane=0; plane<6; plane++)
    {
        if (plane==lastPlaneCulled)
            continue;

        if (basicCullTest(plane, frustum))
        {
            lastPlaneCulled = plane;
            return;
        }
    }

    // Is the box fully within the frustum? If yes, simply render the
    // display list.
    if (isInside)
    {
        glCallList(displayList);
        return;
    }

    // We are now in doubt whether the box should be drawn.
    // If it is a leaf draw it all, if not go to children nodes.
    if (leftChild==NULL)
    {
        glCallList(displayList);
    }
    else
    {
        leftChild->render(frustum);
    }
}

```

```

        rightChild->render(frustum);
    }
}

void Node::transformAndDrawVertex(const Matrix& m, float texx, float texy,
                                  float posx, float posy, float posz)
{
    glVertexCoord2f(texx, texy);
    Vector w(posx, posy, posz);
    w.MultLeftMatrix(m);
    glVertex3f(w.x, w.y, w.z);
}

// The given normal is assumed to be normalized already.
void Node::transformAndSetNormal(const Matrix& t, float x, float y, float z)
{
    Vector v(x, y, z);
    v.MultLeftMatrixIgnoreW(t);
    v.normalize();
    glNormal3f(v.x, v.y, v.z);
}

Node::Node(BoxData boxData[], int n) :
    leftChild(NULL),
    rightChild(NULL),
    displayList(0),
    lastPlaneCulled(0)
{
    boundingBox[0] = Vector(WORLD_MAX_X, WORLD_MAX_Y, WORLD_MAX_Z);
    boundingBox[1] = Vector(0, 0, 0);
    // Find and set bounding box by iterating through boxes.
    // (Boxes is centered about position and has a distance
    // from center of max sqrt(2*pow(scale,2)) )

    for (int c=0; c<n; ++c)
    {
        float boxMaxDistFromCenter = sqrt(2*pow(boxData[c].scale,2));
        float candidate;

        candidate = boxData[c].position.x - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].x)
            boundingBox[0].x = candidate;

        candidate = boxData[c].position.y - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].y)
            boundingBox[0].y = candidate;

        candidate = boxData[c].position.z - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].z)
            boundingBox[0].z = candidate;

        candidate = boxData[c].position.x + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].x)
            boundingBox[1].x = candidate;

        candidate = boxData[c].position.y + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].y)
            boundingBox[1].y = candidate;

        candidate = boxData[c].position.z + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].z)
            boundingBox[1].z = candidate;
    }

    boxCenter = (boundingBox[1]-boundingBox[0])/2 + boundingBox[0];
    boxRadiusSquared = (boxCenter - boundingBox[0]).getLengthSquared();

    // If n is more than max_boxes
    if (n>MAX_BOXES_PER_LEAF)
    {
        int splitn;

```

```

if (n>2)
{
    float xspan = boundingBox[1].x - boundingBox[0].x;
    float yspan = boundingBox[1].y - boundingBox[0].y;
    float zspan = boundingBox[1].z - boundingBox[0].z;

    if (xspan >= yspan && xspan >= zspan)
    {
        // Find center along max axis
        float center = boundingBox[0].x + xspan/2.0f;

        // Reorder box data according to center on the max axis.
        splitn = reorderx(boxData,n,center);
    }
    else
    {
        if (yspan >= zspan)
        {
            // Find center along max axis
            float center = boundingBox[0].y + yspan/2.0f;

            // Reorder box data according to center on the max axis.
            splitn = reordery(boxData,n,center);
        }
        else
        {
            // Find center along max axis
            float center = boundingBox[0].z + zspan/2.0f;

            // Reorder box data according to center on the max axis.
            splitn = reorderz(boxData,n,center);
        }
    }
}
else
{
    splitn=1;
}

if (splitn==0)
    splitn = 1;

if (splitn==n)
    splitn = n-1;

// Create nodes for each child
leftChild = new Node(boxData, splitn);
rightChild = new Node(&(boxData[splitn]), n-splitn);

// Initialize display list that calls children display list
displayList = glGenLists(1);
ASSERT(displayList);
glNewList(displayList, GL_COMPILE);
    glCallList(leftChild->displayList);
    glCallList(rightChild->displayList);
glEndList();
}
else
{
    // Initialize display list to draw the boxes. This display list
    // must be executed within a begin() end() block that was started
    // with GL_QUAD_STRIP

    displayList = glGenLists(1);
    ASSERT(displayList);
    glNewList(displayList, GL_COMPILE);
        for (int i=0;i<n;++i)
        {
            Matrix m;
            m.loadIdentity();
            m.translate(boxData[i].position.x,boxData[i].position.y,

```

```

        boxData[i].position.z);
    m.rotate(boxData[i].rotation.x,1,0,0);
    m.rotate(boxData[i].rotation.y,0,1,0);
    m.rotate(boxData[i].rotation.z,0,0,1);
    float scale = boxData[i].scale;
    m.scale(scale,scale,scale);

    Matrix t(m);
    VERIFY(t.invert());
    t.transpose();

    // Left side
    transformAndSetNormal(t,-1,0,0);
    transformAndDrawVertex(m,1,0,-1,-1,1);
    transformAndDrawVertex(m,1,1,-1,1,1);
    transformAndDrawVertex(m,0,1,-1,1,-1);
    transformAndDrawVertex(m,0,0,-1,-1,-1);

    // Front side
    transformAndSetNormal(t,0,0,1);
    transformAndDrawVertex(m,1,0,1,-1,1);
    transformAndDrawVertex(m,1,1,1,1,1);
    transformAndDrawVertex(m,0,1,-1,1,1);
    transformAndDrawVertex(m,0,0,-1,-1,1);

    // Right side
    transformAndSetNormal(t,1,0,0);
    transformAndDrawVertex(m,1,0,1,-1,-1);
    transformAndDrawVertex(m,1,1,1,1,-1);
    transformAndDrawVertex(m,0,1,1,1,1);
    transformAndDrawVertex(m,0,0,1,-1,1);

    // Top side
    transformAndSetNormal(t,0,1,0);
    transformAndDrawVertex(m,1,0,1,1,1);
    transformAndDrawVertex(m,1,1,1,1,-1);
    transformAndDrawVertex(m,0,1,-1,1,-1);
    transformAndDrawVertex(m,0,0,-1,1,1);

    // Back side
    transformAndSetNormal(t,0,0,-1);
    transformAndDrawVertex(m,1,0,-1,-1,-1);
    transformAndDrawVertex(m,1,1,-1,1,-1);
    transformAndDrawVertex(m,0,1,1,1,-1);
    transformAndDrawVertex(m,0,0,1,-1,-1);

    // Under side
    transformAndSetNormal(t,0,-1,0);
    transformAndDrawVertex(m,1,0,1,-1,-1);
    transformAndDrawVertex(m,1,1,1,-1,1);
    transformAndDrawVertex(m,0,1,-1,-1,1);
    transformAndDrawVertex(m,0,0,-1,-1,-1);
    }
    glEndList();
}
}

Node::~Node(void)
{
    if (leftChild)
    {
        delete leftChild;
        ASSERT(rightChild);
        delete rightChild;
    }

    glDeleteLists(displayList,1);
}

#define REORDER(X) \
static int reorder ## X(BoxData boxData[], int n, float splitvalue) \

```



```

// viewing direction line with plane.
points[4] = centerOfNearPlane;
points[5] = cameraPos + cameraDirection * pfar;

// Set normals for near and far planes.
normals[4] = -cameraDirection; // Normal points towards camera
normals[5] = cameraDirection; // Normal points away from camera

// Calculate normals for sides. The general scheme is:
// 1. We have one point on the plane as the cameraPos.
// 2. By moving along the near plane with the given amount
//    we get another point in the plane.
// 3. These two points can, crossed with the upvector (left,right)
//    or the perpendicular of the upvector (top,bottom) give us
//    the normal of the plane.

// Calculate normal for left side
Vector leftNearPoint = cameraDirection.cross(cameraUp) * left +
    centerOfNearPlane;
normals[0] = - (leftNearPoint - cameraPos).cross(cameraUp);

// Calculate normal for right side
Vector rightNearPoint = cameraDirection.cross(cameraUp) * right +
    centerOfNearPlane;
normals[1] = (rightNearPoint - cameraPos).cross(cameraUp);

// Calculate normal for bottom side
Vector bottomNearPoint = centerOfNearPlane + cameraUp * bottom;
normals[2] = - cameraDirection.cross(cameraUp)
    .cross(bottomNearPoint - cameraPos);

// Calculate normal for top side
Vector topNearPoint = centerOfNearPlane + cameraUp * top;
normals[3] = cameraDirection.cross(cameraUp)
    .cross(topNearPoint - cameraPos);

// Calculate the center of the frustum
center = (points[5] - points[4])/2 + points[4];

Vector extremeNearCorner = centerOfNearPlane + cameraUp * max(top,-bottom) +
    cameraDirection.cross(cameraUp) * max(right,-left);

Vector extremeFarCorner = ((extremeNearCorner - cameraPos) / pnear) * pfar
    + cameraPos;

radiusSquared = (extremeFarCorner - center).getSquaredLength();
}

```

Frustum.h

```

/////////////////////////////////////////////////////////////////
// Tweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// The view fustrum.
//
/////////////////////////////////////////////////////////////////

#ifndef FRUSTUM_LIBRARY
#define FRUSTUM_LIBRARY

#include <GL/glut.h>
#include "Vector.h"

class Frustum
{
public:
    // points and normals for planes in this order:
    // left,right,bottom,top,near,far.
    Vector points[6];

```

```

Vector normals[6];

Vector center;
float radiusSquared;

// The first three args is information about the camera in the world.
// The last 6 is the parameters given to glFrustum().
Frustum(Vector cameraPos, Vector cameraDirection, Vector cameraUp,
        GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
        GLdouble pnear, GLdouble pfar);
};

#endif

```

general.h

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A set of general purpose functions.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#ifndef GENERAL_LIB
#define GENERAL_LIB

#include <assert.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

#ifdef _DEBUG
#define ASSERT(X) assert(X)
#else
#define ASSERT(X)
#endif

#ifdef _DEBUG
#define VERIFY(X) assert(X)
#else
#define VERIFY(X) X
#endif

#ifndef max
template <class Type>
Type max(const Type a, const Type b)
{
    return a < b ? a : b;
}
#endif

#ifndef min
template <class Type>
Type min(const Type a, const Type b)
{
    return a > b ? b : a;
}
#endif

// Return the absolute value of a
template <class Type>
Type Abs(const Type a)
{
    return a < 0 ? -a : a;
}

// Return a random int number between low and high, both inclusive.

```

```

inline int randomInteger(int low, int high)
{
    if (low>high)
    {
        int temp = low;
        low = high;
        high = temp;
    }
    int num = (int)((((double)rand())/((double)RAND_MAX)+1.0))
              * (high-low+1) + low);
    ASSERT(num>=low);
    ASSERT(num<=high);
    return num;
}

// Return a random floating point number between low and high
// both inclusive
inline float randomFloat(float low, float high)
{
    if (low>high)
    {
        float temp = low;
        low = high;
        high = temp;
    }
    float num = (float)((((double)rand())/((double)RAND_MAX)+1.0))
               * (high-low) + low);
    ASSERT(num>=low);
    ASSERT(num<=high);
    return num;
}

// Search a set of command line arguments for one beginning with label.
// If it exist return true.
inline bool getBooleanFlag(const char* label, int argc, char* argv[])
{
    for(int i=0;i<argc;i++)
    {
        if (!strcmp(label,argv[i],strlen(label)))
        {
            return true;
        }
    }
    return false;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as an integer and return
// the value. If label was not found return defaultvalue.
inline int getIntegerArgument(const char* label, int argc, char* argv[],
                             int defaultvalue)
{
    for(int i=0;i<argc;i++)
    {
        if (!strcmp(label,argv[i],strlen(label)))
        {
            return atoi(argv[i]+strlen(label));
        }
    }
    return defaultvalue;
}

// Search a set of command line arguments for one beginning with label.
// If found parse the remainder of the string as a float and return
// the value. If label was not found return defaultvalue.
inline float getFloatArgument(const char* label, int argc, char* argv[],
                              float defaultvalue)
{
    for(int i=0;i<argc;i++)
    {
        if (!strcmp(label,argv[i],strlen(label)))

```

```

    {
        char* s = argv[i]+strlen(label);
        float f;
        sscanf(s,"%f",&f);
        return f;
    }
}
return defaultvalue;
}

#endif

```

main.cpp

```

/////////////////////////////////////////////////////////////////
// Tweaked Virtual World (C++ version)
// by Jacob Marner. Feb 2002.
//
// The main class. Start this to run the application.
//
// The application takes two optional parameters:
// -frames:<int> : The number of frames to run the animation. Default: 3000.
// -boxes:<int> : The number of boxes to generate. Default: 1000.
//
/////////////////////////////////////////////////////////////////

#include <GL/glut.h>

#include "general.h"
#include "Vector.h"
#include <time.h>
#include <iostream.h>
#include "BoxTree.h"
#include "BMPLoader.h"
#include "Frustum.h"

#ifdef _WIN32
#include <windows.h>
#endif

Node* root;

clock_t starttime;

GLuint tex;

int frame = 0;
int numFrames;

void startTimer(void)
{
    starttime = clock();
}

void stopTimer(void)
{
    float timeelapsed = ((float)(clock() - starttime)) / CLOCKS_PER_SEC;
    cout << "Elapsed time: " << timeelapsed << " sec." << endl;
    cout << "Average frame rate: " << ((float)numFrames)/timeelapsed <<
        " fps." << endl;
}

GLdouble frustumLeft;
GLdouble frustumRight;
GLdouble frustumBottom;
GLdouble frustumTop;
GLdouble frustumNear;
GLdouble frustumFar = 40;

void reshape(int width, int height)

```

```

{
    glViewport(0, 0, width, height);

    glMatrixMode(GL_PROJECTION);

    frustumLeft = -0.5;
    frustumRight = 0.5;
    frustumBottom = -0.5*((double)height/(double)width);
    frustumTop = 0.5*((double)height/(double)width);
    frustumNear = 0.5;

    glLoadIdentity();

    glFrustum(frustumLeft, frustumRight, frustumBottom,
        frustumTop, frustumNear, frustumFar);
    glMatrixMode(GL_MODELVIEW);

    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 50.0 };
    GLfloat diffuse_light[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat specular_light[] = { 0.0, 0.0, 0.0, 0.0 };
    GLfloat ambient_light[] = { 1.0, 1.0, 1.0, 1.0 };
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_light);
    glLightfv(GL_LIGHT0, GL_SPECULAR, specular_light);
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light);
}

Vector upVector(0,1,0);
GLfloat light_position[] = {0.0, 1.0, -1.0, 0.0 };

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Calculate camera position and direction
    float angle = (float)frame / 600.0f;
    float viewAngle = angle+3.14f/2.0f;
    float radius = 20;
    Vector cameraPos(WORLD_MAX_X/2.0f+radius*cos(angle),
        WORLD_MAX_Y/2.0f,
        WORLD_MAX_Z/2.0f+radius*sin(angle));
    Vector cameraDirection(cos(viewAngle),0,sin(viewAngle));

    // Set view matrix
    glLoadIdentity();
    gluLookAt(cameraPos.x, cameraPos.y, cameraPos.z,
        cameraPos.x+cameraDirection.x,
        cameraPos.y+cameraDirection.y,
        cameraPos.z+cameraDirection.z, upVector.x,
        upVector.y, upVector.z);

    Frustum frustum(cameraPos, cameraDirection, upVector, frustumLeft,
        frustumRight, frustumBottom, frustumTop, frustumNear,
        frustumFar);

    glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    glBegin(GL_QUADS);
        root->render(frustum);
    glEnd();

    glutSwapBuffers();

    // If final frame has been drawn stop timer, print results and exit.
    frame++;
    if (frame>=numFrames)
    {
        stopTimer();
    }
}

```

```

        exit(0);
    }
}

void idle(void)
{
    glutPostRedisplay();
}

void init(int numBoxes)
{
    int i;

    // Generate numBoxes box data put them in array. For each one
    // store scale(1), pos(3), rotation(3). Generate each number
    // randomly.
    srand(time(NULL));

    BoxData* boxData = new BoxData[numBoxes];

    for (i=0;i<numBoxes;++i)
    {
        boxData[i].scale= randomFloat(0.1,1.5);
        boxData[i].position.x= randomFloat(0,WORLD_MAX_X);
        boxData[i].position.y= randomFloat(0,WORLD_MAX_Y);
        boxData[i].position.z= randomFloat(0,WORLD_MAX_Z);
        boxData[i].rotation.x= randomFloat(0,360);
        boxData[i].rotation.y= randomFloat(0,360);
        boxData[i].rotation.z= randomFloat(0,360);
    }

    // Create a tree structure. Each node has bounding box information
    // and zero or two children. Each node contains are display list
    // printing all contents of that node as one diplaylist. Inner nodes
    // use call list to sub node display lists. The leaf nodes push
    // viewmodel matrix, scale, pos, rotation (as one matrix!) draw the
    // stripped box and then pop matrix again.

    root = new Node(boxData, numBoxes);

    delete[] boxData;
}

void __cdecl exitfunc( void )
{
    delete root;
}

int main(int argc, char *argv[])
{
    ////////////////////////////////////////
    // Initialize OpenGL
    ////////////////////////////////////////

    glutInit(&argc, argv);

    atexit(exitfunc);

    int numBoxes = getIntegerArgument("-boxes:", argc, argv, 100);
    numFrames = getIntegerArgument("-frames:", argc,argv, 1000);

    // Tell OpenGL to use double buffering and
    // 16/32 bit colors (whatever is the default)
    glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);

    // Create a 640x480 window
    glutInitWindowSize(640, 480);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("C++ OpenGL benchmark");

    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
}

```

```

// Turn on z-buffer for hidden surface removal
glEnable(GL_DEPTH_TEST);

// Enable back face culling.
glCullFace(GL_BACK);
glEnable(GL_CULL_FACE);

glClearColor(0, 0, 0, 0);

// Tell OpenGL to use flat shading
glShadeModel(GL_FLAT);

// Enable Phong lighting model and one global
// directional light.
glEnable(GL_LIGHTING);
glEnable(GL_LIGHT0);

// Load texture
if (loadOpenGL2DTextureBMP("glow.bmp",&tex, GL_RGB))
{
    cerr << "Error loading texture." << endl;
    exit(1);
}

// Enable texture mapping
glEnable(GL_TEXTURE_2D);
// Blend lighting a texture effects.
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
// Enable trilinear filtering.
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// Enable fogging
glEnable(GL_FOG);
GLfloat fogColor[4] = {0.0f, 0.0f, 0.0f, 0.0f};
glFogf(GL_FOG_MODE, GL_LINEAR);
glFogfv(GL_FOG_COLOR, fogColor);
glFogf(GL_FOG_DENSITY, 0.35);
glHint(GL_FOG_HINT, GL_DONT_CARE);
glFogf(GL_FOG_START, frustumFar * 0.85);
glFogf(GL_FOG_END, frustumFar);

// Register call back functions
glutReshapeFunc(reshape);
glutDisplayFunc(display);
glutIdleFunc(idle);

#ifdef _WIN32
// Turn off vertical screen sync under Windows.
// (I.e. it uses the WGL_EXT_swap_control extension)
// On under systems we just live with the default.
typedef BOOL (WINAPI *PFNWGLSWAPINTERVALEXTPROC)(int interval);
PFNWGLSWAPINTERVALEXTPROC wglSwapIntervaleXT = NULL;
wglSwapIntervaleXT =
(PFNWGLSWAPINTERVALEXTPROC)wglGetProcAddress("wglSwapIntervaleXT");
ASSERT(wglSwapIntervaleXT);
wglSwapIntervaleXT(0);
#endif

// Create data and display lists
init(numBoxes);

startTimer();

glutMainLoop();
return 0;
}

```

Matrix.h

```
/////////////////////////////////////////////////////////////////
// Tweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A 4x4 float matrix class.
//
// All operations are applied such that the
// Matrix itself is the Left operator.
//
// The internal matrix representation is directly
// compatible with that of OpenGL, so you can use
// glLoadMatrix() on m to import the matrix into
// OpenGL.
//
/////////////////////////////////////////////////////////////////

#ifndef MATRIX_CLASS
#define MATRIX_CLASS

#include <math.h>

template<class Type>
inline float toRadians(Type deg)
{
    return deg*3.1415926535f/180.0f;
}

class Matrix
{
public:
    float m[16];

    Matrix(void)
    {
    }

    void loadIdentity()
    {
        m[0]=m[5]=m[10]=m[15]=1;
        m[1]=m[2]=m[3]=m[4]=m[6]=m[7]=m[8]=m[9]=m[11]=m[12]=m[13]=m[14]=0;
    }

    Matrix(float m0,float m1,float m2,float m3,float m4,
           float m5,float m6,float m7,float m8,float m9,
           float m10,float m11,float m12,float m13,float m14,
           float m15)
    {
        m[0] = m0;
        m[1] = m1;
        m[2] = m2;
        m[3] = m3;
        m[4] = m4;
        m[5] = m5;
        m[6] = m6;
        m[7] = m7;
        m[8] = m8;
        m[9] = m9;
        m[10] = m10;
        m[11] = m11;
        m[12] = m12;
        m[13] = m13;
        m[14] = m14;
        m[15] = m15;
    }

    Matrix(const Matrix& ma)
    {
        memcpy(m,ma.m,16*sizeof(float));
    }
};
```

```

}

Matrix& operator=(const Matrix& ma)
{
    memcpy(m,ma.m,16*sizeof(float));
    return *this;
}

void mult(Matrix &ma)
{
    float *n = ma.m;
    float t[12];
    t[0] = m[0] * n[0] + m[4] * n[1] + m[8] * n[2] + m[12] * n[3];
    t[1] = m[1] * n[0] + m[5] * n[1] + m[9] * n[2] + m[13] * n[3];
    t[2] = m[2] * n[0] + m[6] * n[1] + m[10] * n[2] + m[14] * n[3];
    t[3] = m[3] * n[0] + m[7] * n[1] + m[11] * n[2] + m[15] * n[3];
    t[4] = m[0] * n[4] + m[4] * n[5] + m[8] * n[6] + m[12] * n[7];
    t[5] = m[1] * n[4] + m[5] * n[5] + m[9] * n[6] + m[13] * n[7];
    t[6] = m[2] * n[4] + m[6] * n[5] + m[10] * n[6] + m[14] * n[7];
    t[7] = m[3] * n[4] + m[7] * n[5] + m[11] * n[6] + m[15] * n[7];
    t[8] = m[0] * n[8] + m[4] * n[9] + m[8] * n[10] + m[12] * n[11];
    t[9] = m[1] * n[8] + m[5] * n[9] + m[9] * n[10] + m[13] * n[11];
    t[10] = m[2] * n[8] + m[6] * n[9] + m[10] * n[10] + m[14] * n[11];
    t[11] = m[3] * n[8] + m[7] * n[9] + m[11] * n[10] + m[15] * n[11];
    m[12] = m[0] * n[12] + m[4] * n[13] + m[8] * n[14] + m[12] * n[15];
    m[13] = m[1] * n[12] + m[5] * n[13] + m[9] * n[14] + m[13] * n[15];
    m[14] = m[2] * n[12] + m[6] * n[13] + m[10] * n[14] + m[14] * n[15];
    m[15] = m[3] * n[12] + m[7] * n[13] + m[11] * n[14] + m[15] * n[15];
    memcpy(m,t,12*sizeof(float));
}

void rotate(float a, float x, float y, float z)
{
    // Normalize axis vector
    float slength = x*x+y*y+z*z;
    if (slength!=1.0f)
    {
        float length = (float)sqrt(slength);
        x/=length;
        y/=length;
        z/=length;
    }

    float rad = toRadians(a);

    float c = (float)cos(rad);
    float s = (float)sin(rad);
    float oneminusc = 1.0f - c;
    float xy = x*y;
    float yz = y*z;
    float xz = x*z;
    float xs = x*s;
    float ys = y*s;
    float zs = z*s;

    float n[11];
    n[0] = x*x*oneminusc+c;
    n[1] = xy*oneminusc+zs;
    n[2] = xz*oneminusc-ys;
    // n[3] not used
    n[4] = xy*oneminusc-zs;
    n[5] = y*y*oneminusc+c;
    n[6] = yz*oneminusc+xs;
    // n[7] not used
    n[8] = xz*oneminusc+ys;
    n[9] = yz*oneminusc-xs;
    n[10] = z*z*oneminusc+c;

    /* m[12] to m[15] are not changed by a rotate */
    float t[8];
    t[0]= m[0] * n[0] + m[4] * n[1] + m[8] * n[2];

```

```

t[1]= m[1] * n[0] + m[5] * n[1] + m[9] * n[2];
t[2]= m[2] * n[0] + m[6] * n[1] + m[10] * n[2];
t[3]= m[3] * n[0] + m[7] * n[1] + m[11] * n[2];
t[4]= m[0] * n[4] + m[4] * n[5] + m[8] * n[6];
t[5]= m[1] * n[4] + m[5] * n[5] + m[9] * n[6];
t[6]= m[2] * n[4] + m[6] * n[5] + m[10] * n[6];
t[7]= m[3] * n[4] + m[7] * n[5] + m[11] * n[6];
m[8]= m[0] * n[8] + m[4] * n[9] + m[8] * n[10];
m[9]= m[1] * n[8] + m[5] * n[9] + m[9] * n[10];
m[10]=m[2] * n[8] + m[6] * n[9] + m[10] * n[10];
m[11]=m[3] * n[8] + m[7] * n[9] + m[11] * n[10];
memcpy(m,t,8*sizeof(float));
}

void translate(float x, float y, float z)
{
    /* m[0] to m[11] are not changed by a translate */
    m[12] = m[0] * x + m[4] * y + m[8] * z + m[12];
    m[13] = m[1] * x + m[5] * y + m[9] * z + m[13];
    m[14] = m[2] * x + m[6] * y + m[10] * z + m[14];
    m[15] = m[3] * x + m[7] * y + m[11] * z + m[15];
}

void scale(float x, float y, float z)
{
    /* m[12] to m[15] are not changed by a scale */
    m[0] *= x;
    m[1] *= x;
    m[2] *= x;
    m[3] *= x;
    m[4] *= y;
    m[5] *= y;
    m[6] *= y;
    m[7] *= y;
    m[8] *= z;
    m[9] *= z;
    m[10] *= z;
    m[11] *= z;
}

// Returns true at success or false if the matrix was singular
bool invert(void)
{
    const int n = 4;
    const int n2 = 2*n;
    const int nn = n*n;

    float alpha;
    float beta;
    int i;
    int j;
    int k;

    float D[n2*n];
    for (i=0;i<nn;++i)
        D[i]=m[i];

    // init the reduction matrix
    for( i = 0; i < n; i++ )
    {
        for( j = 0; j < n; j++ )
        {
            D[i+n*j+nn] = 0.0;
        }
        D[i+n*i+nn] = 1.0;
    }

    // perform the reductions
    for( i = 0; i < n; i++ ) // For each row
    {
        alpha = D[i*n+i]; // Get diagonal value

```

```

// Make sure it is not 0. If so the matrix is singular and we will not
// invert it.
// A non-singular matrix is one where inv(inv(A)) = A
if(!alpha)
    return false;

// For each column in this row divide though with the diagonal value so
// so alpha becomes 1.
for( j = 0; j < n2; j++ )
    {
        D[i+n*j] /= alpha;
    }

// Update all other rows.
for( k = 0; k < n; k++ )
    {
        if( (k-i) != 0 )
            {
                beta = D[k+n*i];
                for( j = i; j < n2; j++ )
                    {
                        D[k+n*j] -= beta*D[i+n*j];
                    }
            }
    }

// Copy result from D to m
memcpy(m,&D[nn],sizeof(float)*nn);

return true;
}

void transpose()
{
    float temp;
    temp = m[1]; m[1] = m[4]; m[4] = temp;
    temp = m[2]; m[2] = m[8]; m[8] = temp;
    temp = m[3]; m[3] = m[12]; m[12] = temp;
    temp = m[6]; m[6] = m[9]; m[9] = temp;
    temp = m[7]; m[7] = m[13]; m[13] = temp;
    temp = m[11]; m[11] = m[14]; m[14] = temp;
}
};

#endif

```

Vector.cpp

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A 3d float vector class.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

#include "Vector.h"

ostream& operator<<(ostream& os, Vector& v)
{
    os << "(" << v.x << ", " << v.y << ", " << v.z << ")";
    return os;
}

```

Vector.h

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

// Tweaked virtual world (C++ version)
// by Jacob Marner. Feb 2002.
//
// A 3d float vector class.
//
/////////////////////////////////////////////////////////////////

#ifndef VECTOR_CLASS
#define VECTOR_CLASS

#include "general.h"
#include "Matrix.h"
#include <math.h>
#include <stddef.h>
#include <ostream.h>

class Vector
{
public:
    float x;
    float y;
    float z;

    Vector(void) : x(0), y(0), z(0) {}
    Vector(const float x, const float y, const float z) : x(x), y(y), z(z) {}
    Vector(const Vector& v) : x(v.x), y(v.y), z(v.z) {}

    inline Vector& operator+=(const Vector& v)
    {
        x += v.x;
        y += v.y;
        z += v.z;
        return *this;
    }

    inline Vector operator+(const Vector& v) const
    {
        return Vector(x+v.x,y+v.y,z+v.z);
    }

    inline Vector operator-(const Vector& v) const
    {
        return Vector(x-v.x,y-v.y,z-v.z);
    }

    inline Vector operator/(const float c) const
    {
        return Vector(x/c,y/c,z/c);
    }

    inline Vector operator-(void) const
    {
        return Vector(-x,-y,-z);
    }

    inline Vector operator*(const float c) const
    {
        return Vector(x*c,y*c,z*c);
    }

    inline void operator-=(const Vector& v)
    {
        x -= v.x;
        y -= v.y;
        z -= v.z;
    }

    inline void operator*=(const float c)
    {
        x *= c;
        y *= c;
    }
}

```

```

    z *= c;
}

inline void operator/=(const float c)
{
    x /= c;
    y /= c;
    z /= c;
}

// Copy vector
inline Vector& operator=(const Vector& v)
{
    x = v.x;
    y = v.y;
    z = v.z;
    return *this;
}

// Get 2-norm
inline float getLength(void) const
{
    return (float)sqrt(x*x+y*y+z*z);
}

inline float getLengthSquared(void) const
{
    return x*x+y*y+z*z;
}

// Get max of each element (not the same as infinity norm)
inline float getMax(void) const
{
    return max(max(x,y),z);
}

inline float getInfinityNorm(void) const
{
    return max(max(Abs(x),Abs(y)),Abs(z));
}

// Get 2-norm but before the square root is taken
inline float getSquaredLength(void) const
{
    return x*x+y*y+z*z;
}

// Return a normalized version (unit length) of the
// vector.
inline Vector normalized(void) const
{
    float l = getLength();
    return Vector(x / l, y / l, z / l);
}

inline void normalize(void)
{
    float l = getLength();
    x /= l;
    y /= l;
    z /= l;
}

inline void invert(void)
{
    x = -x;
    y = -y;
    z = -z;
}

inline Vector inverted(void) const

```

```

{
    return Vector(-x,-y,-z);
}

inline Vector cross(const Vector& v) const
{
    return Vector(y*v.z-z*v.y,z*v.x-x*v.z,x*v.y-y*v.x);
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
inline bool isInRectangle(const Vector& lowerCorner,
                        const Vector& upperCorner) const
{
    return (x >= lowerCorner.x && y >= lowerCorner.y && z >= lowerCorner.z &&
           x < upperCorner.x && y < upperCorner.y && z < upperCorner.z);
}

inline void MultLeftMatrix(const Matrix& ma)
{
    const float* m = ma.m;
    float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
    float w1 = m[3] * x + m[7] * y + m[11] * z + m[15];
    x = x1;
    y = y1;
    z = z1;
    if (w1!=1)
    {
        x/=w1;
        y/=w1;
        z/=w1;
    }
}

// Do matrix result, but do not divide through with w
// afterwards. Just throw w away (it is in fact never
// calculated. The last row of ma is thus not used.
inline void MultLeftMatrixIgnoreW(const Matrix& ma)
{
    const float* m = ma.m;
    float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
    x = x1;
    y = y1;
    z = z1;
}

inline float dot(const Vector& v) const
{
    return x * v.x + y * v.y + z * v.z;
}

friend ostream& operator<<(ostream&, Vector& v);
};

#endif

```

Part 2: Java GL4Java source code

The source code consists of the following files:

Arguments.java	This file is used when parsing command line arguments
BoxData.java	This class represents the data generated to represent a box.
EventHandling.java	The class that handles all the call backs. This is similar to the call back functions in GLUT. We also do the scene generation here.
Frustum.java	This class represents a view frustum. It stores the 6 planes in world coordinates.
Main.java	The main class. Start this to run the application.
Matrix.java	A generic 4x4 matrix float class.
Node.java	A node in the scene graph / kd-tree.
Vector.java	A generic 3D vector class of floats.
gl4java/utils/textures/ BmpTextureLoader.java	This is a BMP loader for gl4Java. It has been submitted for inclusion in later versions of GL4Java.

Arguments.java

```
////////////////////////////////////  
// Tweaked virtual world (GL4Java version)  
// by Jacob Marner. Feb 2002.  
//  
// This file is used when parsing command line arguments  
//  
////////////////////////////////////  
  
public class Arguments  
{  
    String[] args;  
  
    public Arguments(String[] args)  
    {  
        this.args = args;  
    }  
  
    public boolean getBooleanFlag(String label)  
    {  
        for (int i = 0; i < args.length; i++)  
        {  
            if (args[i].equals(label))  
            {  
                return true;  
            }  
        }  
    }  
}
```

```

    }
  }
  return false;
}

public int getIntegerArgument(String label, int defaultvalue)
{
  for (int i = 0; i < args.length; i++)
  {
    if (args[i].startsWith(label))
    {
      String s = args[i].substring(label.length());
      try
      {
        return Integer.parseInt(s);
      }
      catch (NumberFormatException e)
      {
        return defaultvalue;
      }
    }
  }
  return defaultvalue;
}

public float getFloatArgument(String label, float defaultvalue)
{
  for (int i = 0; i < args.length; i++)
  {
    if (args[i].startsWith(label))
    {
      String s = args[i].substring(label.length());
      try
      {
        return Float.parseFloat(s);
      }
      catch (NumberFormatException e)
      {
        return defaultvalue;
      }
    }
  }
  return defaultvalue;
}
}
}

```

BoxData.java

```

////////////////////////////////////
// Tweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// This class represent the data generated to represent
// a box.
// These are converted to 3D geometry when the kd-tree
// (scene graph) is built.
////////////////////////////////////

public class BoxData
{
  public float scale;
  public Vector position = new Vector();
  public Vector rotation = new Vector();

  public void assign(BoxData b)
  {
    scale = b.scale;
    position.assign(b.position);
    rotation.assign(b.rotation);
  }
}

```

```
};
```

EventHandling.java

```
////////////////////////////////////  
// Tweaked virtual world (GL4Java version)  
// by Jacob Marner. Feb 2002.  
//  
// The class that handles all the call backs.  
// This is similar to the call back functions in  
// GLUT. We also do the scene generation here.  
//  
////////////////////////////////////  
  
import java.util.*;  
import gl4java.*;  
import gl4java.utils.textures.*;  
import gl4java.drawable.*;  
  
class EventHandling implements GLEventListener  
{  
    public final static int WORLD_MAX_X = 100;  
    public final static int WORLD_MAX_Y = 100;  
    public final static int WORLD_MAX_Z = 100;  
  
    Node root;  
  
    int[] tex = new int[1];  
  
    double frustumLeft;  
    double frustumRight;  
    double frustumBottom;  
    double frustumTop;  
    double frustumNear;  
    double frustumFar = 40;  
  
    public EventHandling()  
    {  
    }  
  
    Date starttime;  
  
    private void startTimer()  
    {  
        starttime = new Date();  
    }  
  
    private void stopTimer()  
    {  
        Date endTime = new Date();  
        float timeelapsed = (endTime.getTime() - starttime.getTime()) / 1000.0f;  
        System.out.println("Elapsed time: " + timeelapsed + " sec.");  
        System.out.println(  
            "Average frame rate: " + ((float) numFrames) / timeelapsed + " fps.");  
    }  
  
    Random rand;  
    private GLFunc gl;  
    private GLUFunc glu;  
    private GLContext glj;  
  
    // Generate a random number between 0 and maxValue.  
    public float getRandomFloat(float minValue, float maxValue)  
    {  
        float interval = maxValue - minValue;  
        float offset = rand.nextFloat() * interval;  
        return minValue + offset;  
    }  
  
    public void init(GLDrawable drawable)
```

```

{
    gl = drawable.getGL();
    glu = drawable.getGLU();
    glj = drawable.getGLContext();
    Node.gl = gl;

    // This Will Clear The Background Color To Black
    gl.glClearColor(0.0f, 0.0f, 0.0f, 0.0f);

    gl.glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);

    // Turn on z-buffer for hidden surface removal
    gl.glEnable(GL_DEPTH_TEST);

    // Enable back face culling.
    gl.glCullFace(GL_BACK);
    gl.glEnable(GL_CULL_FACE);

    gl.glClearColor(0, 0, 0, 0);

    // Tell OpenGL to use flat shading
    gl.glShadeModel(GL_FLAT);

    // Enable Phong lighting model and one global
    // directional light.
    gl.glEnable(GL_LIGHTING);
    gl.glEnable(GL_LIGHT0);

    BmpTextureLoader texLoader = new BmpTextureLoader(gl, glu);
    texLoader.readTexture("glow.bmp");

    if (!texLoader.isOk())
    {
        System.err.println("Error loading texture.");
        System.exit(1);
    }

    //Create Texture
    gl.glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
    gl.glGenTextures(1, tex);
    gl.glBindTexture(GL_TEXTURE_2D, tex[0]);

    glu.gluBuild2DMipmaps(
        GL_TEXTURE_2D,
        GL_RGB,
        texLoader.getImageWidth(),
        texLoader.getImageHeight(),
        texLoader.getGLFormat(),
        texLoader.getGLComponentFormat(),
        texLoader.getTexture());

    // Enable texture mapping
    gl.glEnable(GL_TEXTURE_2D);
    // Blend lighting a texture effects.
    gl.glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
    // Enable trilinear filtering.
    gl.glTexParameterf(
        GL_TEXTURE_2D,
        GL_TEXTURE_MIN_FILTER,
        GL_LINEAR_MIPMAP_LINEAR);
    gl.glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

    // Enable fogging
    gl.glEnable(GL_FOG);
    float[] fogColor = { 0.0f, 0.0f, 0.0f, 0.0f };
    gl.glFogi(GL_FOG_MODE, GL_LINEAR);
    gl.glFogfv(GL_FOG_COLOR, fogColor);
    gl.glFogf(GL_FOG_DENSITY, 0.35f);
    gl.glHint(GL_FOG_HINT, GL_DONT_CARE);
    gl.glFogf(GL_FOG_START, (float) (frustumFar * 0.85f));
    gl.glFogf(GL_FOG_END, (float) frustumFar);

```

```

float[] mat_specular = { 1.0f, 1.0f, 1.0f, 1.0f };
float[] mat_shininess = { 50.0f };

float[] diffuse_light = { 1.0f, 1.0f, 1.0f, 1.0f };
float[] specular_light = { 0.0f, 0.0f, 0.0f, 0.0f };
float[] ambient_light = { 1.0f, 1.0f, 1.0f, 1.0f };
gl.glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
gl.glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

gl.glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse_light);
gl.glLightfv(GL_LIGHT0, GL_SPECULAR, specular_light);
gl.glLightfv(GL_LIGHT0, GL_AMBIENT, ambient_light);

// Generate numBoxes box data put them in array. For each one
// store scale(1), pos(3), rotation(3). Generate each number
// randomly.
rand = new Random(new Date().getTime());

BoxData[] boxData = new BoxData[numBoxes];

int i;
for (i = 0; i < numBoxes; ++i)
{
    boxData[i] = new BoxData();
    boxData[i].scale = getRandomFloat(0.1f, 1.5f);
    boxData[i].position.x = getRandomFloat(0.0f, WORLD_MAX_X);
    boxData[i].position.y = getRandomFloat(0.0f, WORLD_MAX_Y);
    boxData[i].position.z = getRandomFloat(0.0f, WORLD_MAX_Z);
    boxData[i].rotation.x = getRandomFloat(0.0f, 360.0f);
    boxData[i].rotation.y = getRandomFloat(0.0f, 360.0f);
    boxData[i].rotation.z = getRandomFloat(0.0f, 360.0f);
}

root = new Node(boxData, 0, numBoxes);

startTimer();

reshape(drawable, 100, 100);
}

public void reshape(GLDrawable d, int width, int height)
{
    gl.glViewport(0, 0, width, height);

    gl.glMatrixMode(GL_PROJECTION);

    frustumLeft = -0.5;
    frustumRight = 0.5;
    frustumBottom = -0.5 * ((double) height / (double) width);
    frustumTop = 0.5 * ((double) height / (double) width);
    frustumNear = 0.5;

    frustum =
        new Frustum(
            (float) frustumLeft,
            (float) frustumRight,
            (float) frustumBottom,
            (float) frustumTop,
            (float) frustumNear,
            (float) frustumFar);

    gl.glLoadIdentity();

    gl.glFrustum(
        frustumLeft,
        frustumRight,
        frustumBottom,
        frustumTop,
        frustumNear,
        frustumFar);
}

```

```

    gl.glMatrixMode(GL_MODELVIEW);
}

float c = 1.0f;
int frame = 0;
public static int numFrames;
public static int numBoxes;

float radius = 20;

Vector cameraPos = new Vector();
Vector cameraDirection = new Vector();
Vector upVector = new Vector(0, 1, 0);

Frustum frustum;

float[] light_position = { 0.0f, 1.0f, -1.0f, 0.0f };

public void display(GLDrawable drawable)
{
    //Clear The Screen And The Depth Buffer
    gl.glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    //Reset The View
    gl.glLoadIdentity();

    // Move camera in circle around center of world.
    float angle = (float) frame / 600.0f;

    cameraPos.assign(
        (float) (WORLD_MAX_X / 2.0f + radius * Math.cos(angle)),
        WORLD_MAX_Y / 2.0f,
        (float) (WORLD_MAX_Z / 2.0f + radius * Math.sin(angle)));
    float eangle = angle + 3.14f / 2.0f;
    cameraDirection.assign(
        (float) (Math.cos(eangle)),
        0.0f,
        (float) (Math.sin(eangle)));

    glu.gluLookAt(
        cameraPos.x,
        cameraPos.y,
        cameraPos.z,
        cameraPos.x + cameraDirection.x,
        cameraPos.y + cameraDirection.y,
        cameraPos.z + cameraDirection.z,
        upVector.x,
        upVector.y,
        upVector.z);

    frustum.set(cameraPos, cameraDirection, upVector);

    gl.glLightfv(GL_LIGHT0, GL_POSITION, light_position);

    gl.glBegin(GL_QUADS);
        root.render(frustum);
    gl.glEnd();

    frame++;
    if (frame >= numFrames)
    {
        stopTimer();
        System.exit(0);
    }
}

public void cleanup(GLDrawable drawable)
{
}

public void preDisplay(GLDrawable drawable)
{
}

```

```

    }

    public void postDisplay(GLDrawable drawable)
    {
    }
}

```

Frustum.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// This class represents a view frustum. It stores the
// 6 planes in world coordinates.
//
// This class a clear example of how tweaking can reduce the
// readability of Java code. Compare this with the untweaked
// version. But tweaked did in fact increase the speed of this
// quite slow class with an order or so.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

class Frustum
{
    // points and normals for planes in this order:
    // left,right,bottom,top,near,far.
    public Vector[] points = new Vector[6];
    public Vector[] normals = new Vector[6];

    public Vector center = new Vector();
    float radiusSquared;

    float left;
    float right;
    float bottom;
    float top;
    float pnear;
    float pfar;

    // The first three args is information about the camera in the world.
    // The last 6 is the parameters given to glFrustum().
    public Frustum(float left, float right, float bottom, float top,
                   float pnear, float pfar)
    {
        this.left = left;
        this.right = right;
        this.bottom = bottom;
        this.top = top;
        this.pnear = pnear;
        this.pfar = pfar;

        for (int i=0;i<6;i++)
        {
            points[i] = new Vector();
            normals[i] = new Vector();
        }
    }

    Vector temp = new Vector();
    Vector temp2 = new Vector();

    Vector centerOfNearPlane = new Vector();
    Vector cameraDirectionCrossedWithUp = new Vector();

    public void set(Vector cameraPos, Vector cameraDirection, Vector cameraUp)
    {
        // Make sure that all the direction vectors are unit vectors.
        cameraDirection.normalize();
        cameraUp.normalize();
    }
}

```

```

// Make sure that the up vector is perpendicular to the
// cameraDirection. It is now paralell to the near plane.
// cameraUp = cameraDirection.cross(cameraUp.cross(cameraDirection));
temp.assign(cameraUp);
temp.crossedWith(cameraDirection);
cameraUp.assign(cameraDirection);
cameraUp.crossedWith(temp);

int i;

// The camera is in the plane of the sides.
for (i=0;i<4;i++)
    points[i] = cameraPos;

// Calculate center of near plane
temp.assign(cameraDirection);
temp.multiplied(pnear);
centerOfNearPlane.assign(cameraPos);
centerOfNearPlane.added(temp);

// Points for the near anf far planes are found by intersection
// viewing direction line with plane.
points[4] = centerOfNearPlane;
// The following line is a faster version of:
// points[5] = cameraPos.add(cameraDirection.multiply(pfar));
temp.assign(cameraDirection);
temp.multiplied(pfar);
points[5].assign(cameraPos);
points[5].added(temp);

// Set normals for near and far planes.
normals[4].assign(cameraDirection);
normals[4].negated(); // Normal points towards camera
normals[5] = cameraDirection; // Normal points away from camera

// Calculate normals for sides. The general scheme is:
// 1. We have one point on the plane as the cameraPos.
// 2. By moving along the near plane with the given amount
//    we get another point in the plane.
// 3. These two points can, crossed with the upvector (left,right)
//    or the perpendicular of the upvector (top,bottom) give us
//    the normal of the plane.

// Calculate normal for left side
//Vector leftNearPoint = cameraDirection.cross(cameraUp).multiply(left)
// .add(centerOfNearPlane);
cameraDirectionCrossedWithUp.assign(cameraDirection);
cameraDirectionCrossedWithUp.crossedWith(cameraUp);
temp.assign(cameraDirectionCrossedWithUp);
temp.multiplied(left);
temp.added(centerOfNearPlane);
//normals[0] = (leftNearPoint.subtract(cameraPos)).cross(cameraUp).negate();
temp.subtracted(cameraPos);
temp.crossedWith(cameraUp);
temp.negated();
normals[0].assign(temp);

// Calculate normal for right side
//Vector rightNearPoint = cameraDirection.cross(cameraUp).multiply(right)
// .add(centerOfNearPlane);
temp.assign(cameraDirectionCrossedWithUp);
temp.multiplied(right);
temp.added(centerOfNearPlane);
//normals[1] = (rightNearPoint.subtract(cameraPos)).cross(cameraUp);
temp.subtracted(cameraPos);
temp.crossedWith(cameraUp);
normals[1].assign(temp);

// Calculate normal for bottom side
//Vector bottomNearPoint = centerOfNearPlane.add(cameraUp.multiply(bottom));

```

```

temp.assign(centerOfNearPlane);
temp2.assign(cameraUp);
temp2.multiplied(bottom);
temp.added(temp2);
/*normals[2] = cameraDirection.cross(cameraUp)
   .cross(bottomNearPoint.subtract(cameraPos)).negate();*/
normals[2].assign(cameraDirectionCrossedWithUp);
temp.subtracted(cameraPos);
normals[2].crossedWith(temp);
normals[2].negated();

// Calculate normal for top side
//Vector topNearPoint = centerOfNearPlane.add(cameraUp.multiply(top));
temp.assign(cameraUp);
temp.multiplied(top);
temp.added(centerOfNearPlane);
//normals[3] = cameraDirection.cross(cameraUp)
//   .cross(topNearPoint.subtract(cameraPos));
temp.subtracted(cameraPos);
normals[3].assign(cameraDirectionCrossedWithUp);
normals[3].crossedWith(temp);

// Calculate the center of the frustum
//center = points[5].subtract(points[4]).divide(2).add(points[4]);
center.assign(points[5]);
center.subtracted(points[4]);
center.divided(2);
center.added(points[4]);

//Vector extremeNearCorner =
//centerOfNearPlane.add(cameraUp.multiply(Math.max(top,-bottom)))
//.add(cameraDirection.cross(cameraUp).multiply(Math.max(right,-left)));
temp.assign(cameraUp);
temp.multiplied(Math.max(top,-bottom));
temp.added(centerOfNearPlane);
temp2.assign(cameraDirectionCrossedWithUp);
temp2.multiplied(Math.max(right,-left));
temp.added(temp2);

//Vector extremeFarCorner =
//extremeNearCorner.subtract(cameraPos).divide(pnear).multiply(pfar)
temp.subtracted(cameraPos);
temp.divided(pnear);
temp.multiplied(pfar);
temp.added(cameraPos);

//radiusSquared = extremeFarCorner.subtract(center).getSquaredLength();
temp.subtracted(center);
radiusSquared = temp.getSquaredLength();
}
}

```

Main.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Tweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// The main class. Start this to run the application.
//
// The application takes two optional parameters:
// -frames:<int> : The number of frames to run the animation. Default: 3000.
// -boxes:<int> : The number of boxes to generate. Default: 1000.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

```

```

import gl4java.*;
import gl4java.awt.*;
import gl4java.drawable.*;

public class Main extends JFrame
{
    //Our rendering canvas
    //We are using GLAnimCanvas because we want the canvas
    //to be constantly redrawn
    GLAnimCanvas canvas = null;

    public Main(String s)
    {
        super(s);
    }

    public static void main(String args[])
    {
        Main f = new Main("Virtual World");

        Arguments arg = new Arguments(args);

        EventHandling.numBoxes = arg.getIntegerArgument("-boxes:", 1000);
        EventHandling.numFrames = arg.getIntegerArgument("-frames:", 3000);

        GLContext.gljNativeDebug = false;
        GLContext.gljThreadDebug = false;
        GLContext.gljClassDebug = false;

        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosed(WindowEvent e)
            {
                System.exit(0);
            }
            public void windowClosing(WindowEvent e)
            {
                windowClosed(e);
            }
        });

        f.getContentPane().setLayout(new BorderLayout());

        // We pack the frame - otherwise the insets are not
        // initialized.
        f.pack();

        // Set the size of the Window
        Insets i = f.getInsets();
        f.setBounds(0, 0, 640 + i.right + i.left, 480 + i.bottom + i.top);

        // Create the 3D canvas
        Dimension d = f.getSize();
        GLCapabilities caps = new GLCapabilities();
        GLAnimCanvas canvas = f.canvas =
            GLDrawableFactory.getFactory()
                .createGLAnimCanvas(caps, d.width, d.height);

        // Create and add the class with the callback methods
        EventHandling demo = new EventHandling();
        canvas.addGLEventListener(demo);
        // Turn of vertical sync and any other pauses.
        canvas.setUseRepaint(false);
        canvas.setUseFpsSleep(false);
        canvas.setUseYield(false);

        canvas.requestFocus();
        f.getContentPane().add("Center", canvas);
        canvas.start();

        f.setVisible(true);
    }
}

```

```
}  
}
```

Matrix.java

```
////////////////////////////////////  
// Tweaked virtual world (GL4Java version)  
// by Jacob Marner. Feb 2002.  
//  
// A generic 4x4 matrix float class.  
//  
// All operations are applied such that the  
// Matrix itself is the Left operand.  
//  
// The internal matrix representation is directly  
// compatible with that of OpenGL, so you can use  
// glLoadMatrix() on m to import the matrix into  
// OpenGL.  
//  
////////////////////////////////////  
  
final public class Matrix  
{  
    public float[] m = new float[16];  
  
    public Matrix()  
    {  
    }  
  
    public void loadIdentity()  
    {  
        m[0]=m[5]=m[10]=m[15]=1;  
        m[1]=m[2]=m[3]=m[4]=m[6]=m[7]=m[8]=m[9]=m[11]=m[12]=m[13]=m[14]=0;  
    }  
  
    public Matrix(float m0,float m1,float m2,float m3,float m4,  
                  float m5,float m6,float m7,float m8,float m9,  
                  float m10,float m11,float m12,float m13,float m14,  
                  float m15)  
    {  
        m[0] = m0;  
        m[1] = m1;  
        m[2] = m2;  
        m[3] = m3;  
        m[4] = m4;  
        m[5] = m5;  
        m[6] = m6;  
        m[7] = m7;  
        m[8] = m8;  
        m[9] = m9;  
        m[10] = m10;  
        m[11] = m11;  
        m[12] = m12;  
        m[13] = m13;  
        m[14] = m14;  
        m[15] = m15;  
    }  
  
    public Matrix(Matrix ma)  
    {  
        for (int i=0;i<16;i++)  
            m[i] = ma.m[i];  
    }  
  
    public void assign(Matrix ma)  
    {  
        for (int i=0;i<16;i++)  
            m[i] = ma.m[i];  
    }  
}
```

```

public void mult(Matrix ma)
{
    float[] n = ma.m;
    float[] t = new float[12];
    t[0] = m[0] * n[0] + m[4] * n[1] + m[8] * n[2] + m[12] * n[3];
    t[1] = m[1] * n[0] + m[5] * n[1] + m[9] * n[2] + m[13] * n[3];
    t[2] = m[2] * n[0] + m[6] * n[1] + m[10] * n[2] + m[14] * n[3];
    t[3] = m[3] * n[0] + m[7] * n[1] + m[11] * n[2] + m[15] * n[3];
    t[4] = m[0] * n[4] + m[4] * n[5] + m[8] * n[6] + m[12] * n[7];
    t[5] = m[1] * n[4] + m[5] * n[5] + m[9] * n[6] + m[13] * n[7];
    t[6] = m[2] * n[4] + m[6] * n[5] + m[10] * n[6] + m[14] * n[7];
    t[7] = m[3] * n[4] + m[7] * n[5] + m[11] * n[6] + m[15] * n[7];
    t[8] = m[0] * n[8] + m[4] * n[9] + m[8] * n[10] + m[12] * n[11];
    t[9] = m[1] * n[8] + m[5] * n[9] + m[9] * n[10] + m[13] * n[11];
    t[10] = m[2] * n[8] + m[6] * n[9] + m[10] * n[10] + m[14] * n[11];
    t[11] = m[3] * n[8] + m[7] * n[9] + m[11] * n[10] + m[15] * n[11];
    m[12] = m[0] * n[12] + m[4] * n[13] + m[8] * n[14] + m[12] * n[15];
    m[13] = m[1] * n[12] + m[5] * n[13] + m[9] * n[14] + m[13] * n[15];
    m[14] = m[2] * n[12] + m[6] * n[13] + m[10] * n[14] + m[14] * n[15];
    m[15] = m[3] * n[12] + m[7] * n[13] + m[11] * n[14] + m[15] * n[15];
    for (int i=0;i<12;i++)
        m[i]=t[i];
}

public void rotate(float a, float x, float y, float z)
{
    // Normalize axis vector
    float slength = x*x+y*y+z*z;
    if (slength!=1.0f)
    {
        float length = (float)Math.sqrt(slength);
        x/=length;
        y/=length;
        z/=length;
    }

    float rad = (float)Math.toRadians(a);

    float c = (float)Math.cos(rad);
    float s = (float)Math.sin(rad);
    float oneminusc = 1.0f - c;
    float xy = x*y;
    float yz = y*z;
    float xz = x*z;
    float xs = x*s;
    float ys = y*s;
    float zs = z*s;

    float[] n = new float[11];
    n[0] = x*x*oneminusc+c;
    n[1] = xy*oneminusc+zs;
    n[2] = xz*oneminusc-ys;
    // n[3] not used
    n[4] = xy*oneminusc-zs;
    n[5] = y*y*oneminusc+c;
    n[6] = yz*oneminusc+xs;
    // n[7] not used
    n[8] = xz*oneminusc+ys;
    n[9] = yz*oneminusc-xs;
    n[10] = z*z*oneminusc+c;

    /* m[12] to m[15] are not changed by a rotate */
    float[] t = new float[8];
    t[0]= m[0] * n[0] + m[4] * n[1] + m[8] * n[2];
    t[1]= m[1] * n[0] + m[5] * n[1] + m[9] * n[2];
    t[2]= m[2] * n[0] + m[6] * n[1] + m[10] * n[2];
    t[3]= m[3] * n[0] + m[7] * n[1] + m[11] * n[2];
    t[4]= m[0] * n[4] + m[4] * n[5] + m[8] * n[6];
    t[5]= m[1] * n[4] + m[5] * n[5] + m[9] * n[6];
    t[6]= m[2] * n[4] + m[6] * n[5] + m[10] * n[6];
    t[7]= m[3] * n[4] + m[7] * n[5] + m[11] * n[6];
}

```

```

m[8]= m[0] * n[8] + m[4] * n[9] + m[8] * n[10];
m[9]= m[1] * n[8] + m[5] * n[9] + m[9] * n[10];
m[10]=m[2] * n[8] + m[6] * n[9] + m[10] * n[10];
m[11]=m[3] * n[8] + m[7] * n[9] + m[11] * n[10];
for (int i=0;i<8;i++)
    m[i]=t[i];
}

public void translate(float x, float y, float z)
{
    /* m[0] to m[11] are not changed by a translate */
    m[12] = m[0] * x + m[4] * y + m[8] * z + m[12];
    m[13] = m[1] * x + m[5] * y + m[9] * z + m[13];
    m[14] = m[2] * x + m[6] * y + m[10] * z + m[14];
    m[15] = m[3] * x + m[7] * y + m[11] * z + m[15];
}

public void scale(float x, float y, float z)
{
    /* m[12] to m[15] are not changed by a scale */
    m[0] *= x;
    m[1] *= x;
    m[2] *= x;
    m[3] *= x;
    m[4] *= y;
    m[5] *= y;
    m[6] *= y;
    m[7] *= y;
    m[8] *= z;
    m[9] *= z;
    m[10] *= z;
    m[11] *= z;
}

// Returns true at success or false if the matrix was singular
public boolean invert()
{
    final int n = 4;
    final int n2 = 2*n;
    final int nn = n*n;

    float alpha;
    float beta;
    int i;
    int j;
    int k;

    float[] D = new float[n2*n];
    for (i=0;i<nn;++i)
        D[i]=m[i];

    // init the reduction matrix
    for( i = 0; i < n; i++ )
    {
        for( j = 0; j < n; j++ )
        {
            D[i+n*j+nn] = 0.0f;
        }
        D[i+n*i+nn] = 1.0f;
    }

    // perform the reductions
    for( i = 0; i < n; i++ ) // For each row
    {
        alpha = D[i*n+i]; // Get diagonal value

        // Make sure it is not 0. If so the matrix is singular and we will not
        // invert it.
        // A non-singular matrix is one where inv(inv(A)) = A
        if(alpha==0.0f)
            return false;
    }
}

```

```

    // For each column in this row divide though with the diagonal value so
    // so alpha becomes 1.
    for( j = 0; j < n2; j++ )
        {
            D[i+n*j] /= alpha;
        }

    // Update all other rows.
    for( k = 0; k < n; k++ )
        {
            if( (k-i) != 0 )
                {
                    beta = D[k+n*i];
                    for( j = i; j < n2; j++ )
                        {
                            D[k+n*j] -= beta*D[i+n*j];
                        }
                }
        }

    // Copy result from D to m
    for (i=0;i<nn;i++)
        m[i] = D[i+nn];

    return true;
}

public void transpose()
{
    float temp;
    temp = m[1]; m[1] = m[4]; m[4] = temp;
    temp = m[2]; m[2] = m[8]; m[8] = temp;
    temp = m[3]; m[3] = m[12]; m[12] = temp;
    temp = m[6]; m[6] = m[9]; m[9] = temp;
    temp = m[7]; m[7] = m[13]; m[13] = temp;
    temp = m[11]; m[11] = m[14]; m[14] = temp;
}
};

```

Node.java

```

/////////////////////////////////////////////////////////////////
// Tweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// A node in the scene graph / kd-tree.
//
// Every node has a bounding box and zero or two
// children. Each node contains a display list.
//
/////////////////////////////////////////////////////////////////

import java.util.*;
import gl4java.GLContext;
import gl4java.awt.*;
import gl4java.*;

class Node implements GLEnum
{
    public static GLFunc gl;

    static final int MAX_BOXES_PER_LEAF = 1;

    // This method may reorder the ordering of the
    // box data array. n is the size of the array.
    public Node(BoxData boxData[], int startindex, int numindex)
    {
        boundingBox[0] =

```

```

    new Vector(
        EventHandling.WORLD_MAX_X,
        EventHandling.WORLD_MAX_Y,
        EventHandling.WORLD_MAX_Z);
boundingBox[1] = new Vector(0, 0, 0);
// Find and set bounding box by iterating through boxes.
// (Boxes is centered about position and has a distance
// from center of max sqrt(2*pow(scale,2)) )

for (int c = startindex; c < startindex + numindex; ++c)
{
    float boxMaxDistFromCenter =
        (float) Math.sqrt(2.0f * (float) Math.pow(boxData[c].scale, 2));
    float candidate;

    candidate = boxData[c].position.x - boxMaxDistFromCenter;
    if (candidate < boundingBox[0].x)
        boundingBox[0].x = candidate;

    candidate = boxData[c].position.y - boxMaxDistFromCenter;
    if (candidate < boundingBox[0].y)
        boundingBox[0].y = candidate;

    candidate = boxData[c].position.z - boxMaxDistFromCenter;
    if (candidate < boundingBox[0].z)
        boundingBox[0].z = candidate;

    candidate = boxData[c].position.x + boxMaxDistFromCenter;
    if (candidate > boundingBox[1].x)
        boundingBox[1].x = candidate;

    candidate = boxData[c].position.y + boxMaxDistFromCenter;
    if (candidate > boundingBox[1].y)
        boundingBox[1].y = candidate;

    candidate = boxData[c].position.z + boxMaxDistFromCenter;
    if (candidate > boundingBox[1].z)
        boundingBox[1].z = candidate;
}

boxCenter =
    boundingBox[1].subtract(boundingBox[0]).divide(2).add(boundingBox[0]);
boxRadiusSquared = boxCenter.subtract(boundingBox[0]).getSquaredLength();

// If n is more than max_boxes
if (numindex > MAX_BOXES_PER_LEAF)
{
    int splitn;
    if (numindex > 2)
    {
        float xspan = boundingBox[1].x - boundingBox[0].x;
        float yspan = boundingBox[1].y - boundingBox[0].y;
        float zspan = boundingBox[1].z - boundingBox[0].z;

        if (xspan >= yspan && xspan >= zspan)
        {
            // Find center along max axis
            float center = boundingBox[0].x + xspan / 2.0f;

            // Reorder box data according to center on the max axis.
            splitn = reorder(boxData, startindex, numindex, center, 0);
        }
        else
        {
            if (yspan >= zspan)
            {
                // Find center along max axis
                float center = boundingBox[0].y + yspan / 2.0f;

                // Reorder box data according to center on the max axis.
                splitn = reorder(boxData, startindex, numindex, center, 1);
            }
        }
    }
}

```

```

    }
    else
    {
        // Find center along max axis
        float center = boundingBox[0].z + zspan / 2.0f;

        // Reorder box data according to center on the max axis.
        splitn = reorder(boxData, startindex, numindex, center, 2);
    }
}
else
{
    splitn = startindex + 1;
}

if (splitn == startindex)
    splitn = startindex + 1;

if (splitn == startindex + numindex)
    splitn = startindex + numindex - 1;

// Create nodes for each child
leftChild = new Node(boxData, startindex, splitn - startindex);
rightChild = new Node(boxData, splitn, startindex + numindex - splitn);

// Initialize display list that calls children display list
displayList = gl.glGenLists(1);

if (displayList == 0)
{
    System.err.println("Internal error. Display used before set.");
    System.exit(1);
}

gl.glNewList(displayList, GL_COMPILE);
gl.glCallList(leftChild.displayList);
gl.glCallList(rightChild.displayList);
gl.glEndList();
}
else
{
    // Initialize display list to draw the boxes. This display list
    // must be executed within a begin() end() block that was started
    // with GL_QUAD_STRIP

    displayList = gl.glGenLists(1);

    if (displayList == 0)
    {
        System.err.println("Internal error. Display used before set.");
        System.exit(1);
    }

    gl.glNewList(displayList, GL_COMPILE);
    for (int i = startindex; i < startindex + numindex; ++i)
    {
        Matrix m = new Matrix();
        m.loadIdentity();
        m.translate(
            boxData[i].position.x,
            boxData[i].position.y,
            boxData[i].position.z);
        m.rotate(boxData[i].rotation.x, 1, 0, 0);
        m.rotate(boxData[i].rotation.y, 0, 1, 0);
        m.rotate(boxData[i].rotation.z, 0, 0, 1);
        float scale = boxData[i].scale;
        m.scale(scale, scale, scale);

        Matrix t = new Matrix(m);
        if (!t.invert())

```

```

    {
        System.err.println("Internal error. Matrix invert failed.");
        System.exit(1);
    }
    t.transpose();

    // Left side
    transformAndSetNormal(t, -1, 0, 0);
    transformAndDrawVertex(m, 1, 0, -1, -1, 1);
    transformAndDrawVertex(m, 1, 1, -1, 1, 1);
    transformAndDrawVertex(m, 0, 1, -1, 1, -1);
    transformAndDrawVertex(m, 0, 0, -1, -1, -1);

    // Front side
    transformAndSetNormal(t, 0, 0, 1);
    transformAndDrawVertex(m, 1, 0, 1, -1, 1);
    transformAndDrawVertex(m, 1, 1, 1, 1, 1);
    transformAndDrawVertex(m, 0, 1, -1, 1, 1);
    transformAndDrawVertex(m, 0, 0, -1, -1, 1);

    // Right side
    transformAndSetNormal(t, 1, 0, 0);
    transformAndDrawVertex(m, 1, 0, 1, -1, -1);
    transformAndDrawVertex(m, 1, 1, 1, 1, -1);
    transformAndDrawVertex(m, 0, 1, 1, 1, 1);
    transformAndDrawVertex(m, 0, 0, 1, -1, 1);

    // Top side
    transformAndSetNormal(t, 0, 1, 0);
    transformAndDrawVertex(m, 1, 0, 1, 1, 1);
    transformAndDrawVertex(m, 1, 1, 1, 1, -1);
    transformAndDrawVertex(m, 0, 1, -1, 1, -1);
    transformAndDrawVertex(m, 0, 0, -1, 1, 1);

    // Back side
    transformAndSetNormal(t, 0, 0, -1);
    transformAndDrawVertex(m, 1, 0, -1, -1, -1);
    transformAndDrawVertex(m, 1, 1, -1, 1, -1);
    transformAndDrawVertex(m, 0, 1, 1, 1, -1);
    transformAndDrawVertex(m, 0, 0, 1, -1, -1);

    // Under side
    transformAndSetNormal(t, 0, -1, 0);
    transformAndDrawVertex(m, 1, 0, 1, -1, -1);
    transformAndDrawVertex(m, 1, 1, 1, -1, 1);
    transformAndDrawVertex(m, 0, 1, -1, -1, 1);
    transformAndDrawVertex(m, 0, 0, -1, -1, -1);
}
gl.glEndList();
}
}

protected void finalize()
{
    gl.glDeleteLists(displayList, 1);
}

// First min box then max box
Vector[] boundingBox = new Vector[2];

boolean isInside;

static Vector temp = new Vector();

// We do a frustum culling algorithm, consisting of
// 1. Bounding spheres for the frustum and bounding box
// is precalculated. These are used to do a bounding sphere
// culling.
// 2. Plane to point comparison to see if the box
// is entirely outside the plane. We only test to points
// per bounding box for each plane.

```

```

// 3. Temporal coherence culling. We remember what plane
// was used to cull this the last time and we start with
// that.
// Note: We do not use the octant test to reduce the plane
// comparisons needed to 3 since that requires a symmetric
// frustum.
public void render(Frustum frustum)
{
    isInside = true;

    int plane;

    temp.assign(boxCenter);
    temp.subtracted(frustum.center);
    if (temp.getSquaredLength() > frustum.radiusSquared + boxRadiusSquared)
        return;

    if (basicCullTest(lastPlaneCulled, frustum))
        return;

    for (plane = 0; plane < 6; plane++)
    {
        if (plane == lastPlaneCulled)
            continue;

        if (basicCullTest(plane, frustum))
        {
            lastPlaneCulled = plane;
            return;
        }
    }

    // Is the box fully within the frustum? If yes, simply render the
    // display list.
    if (isInside)
    {
        gl.glCallList(displayList);
        return;
    }

    // We are now in doubt whether the box should be drawn.
    // If it is a leaf draw it all, if not go to children nodes.
    if (leftChild == null)
    {
        gl.glCallList(displayList);
    }
    else
    {
        leftChild.render(frustum);
        rightChild.render(frustum);
    }
}

int displayList;

Node leftChild = null;
Node rightChild = null;

int lastPlaneCulled = 0;

private void transformAndDrawVertex(
    Matrix m,
    float texx,
    float texy,
    float posx,
    float posy,
    float posz)
{
    gl.glTexCoord2f(texx, texy);
    Vector w = new Vector(posx, posy, posz);
    w.MultLeftMatrix(m);
}

```

```

    gl.glVertex3f(w.x, w.y, w.z);
}

private void transformAndSetNormal(Matrix t, float x, float y, float z)
{
    Vector v = new Vector(x, y, z);
    v.MultLeftMatrixIgnoreW(t);
    v.normalize();
    gl.glNormal3f(v.x, v.y, v.z);
}

static private boolean isOutside(
    float x,
    float y,
    float z,
    Vector point,
    Vector normal)
{
    temp.assign(x, y, z);
    temp.subtracted(point);
    return normal.dot(temp) >= 0;
}

private boolean basicCullTest(int plane, Frustum frustum)
{
    // Find corner furthest from plane in normals direction.
    Vector n = frustum.normals[plane];
    int nx = n.x >= 0 ? 1 : 0;
    int ny = n.y >= 0 ? 1 : 0;
    int nz = n.z >= 0 ? 1 : 0;

    // Check if corner furthest from plane in normals direction is outside.
    // If not it is fully inside.
    if (isOutside(boundingBox[nx].x,
        boundingBox[ny].y,
        boundingBox[nz].z,
        frustum.points[plane],
        n))
    {
        // Check if corner closest to plane in normal's direction is outside.
        // If yes, the box is fully outside.
        // If no, the box is intersecting the frustum border.
        if (isOutside(boundingBox[1 - nx].x,
            boundingBox[1 - ny].y,
            boundingBox[1 - nz].z,
            frustum.points[plane],
            n))
        {
            return true;
        }
        else
            isInside = false;
    }

    return false;
}

Vector boxCenter;
float boxRadiusSquared;

// Reorder boxData array according to value at given offset. If offset is 0 then
// x is used. 1 gives y. 2 gives z.
static int reorder(
    BoxData boxData[],
    int startindex,
    int numindex,
    float splitvalue,
    int offset)
{
    BoxData temp = new BoxData();

```

```

    if (numindex < 1)
        return 0;

    int r = startindex + numindex - 1;
    int p = startindex;

    int i = p - 1;
    int j = r + 1;
    while (true)
    {
        i++;
        j--;

        while (j >= p && boxData[j].position.index(offset) > splitvalue)
            j--;

        while (i <= r && boxData[i].position.index(offset) < splitvalue)
            i++;

        if (i < j)
        {
            temp.assign(boxData[i]);
            boxData[i].assign(boxData[j]);
            boxData[j].assign(temp);
        }
        else
        {
            while (j < r && boxData[j].position.index(offset) < splitvalue)
                j++;
            if (j <= p)
                j = p;

            return j;
        }
    }
};

```

Vector.java

```

//////////////////////////////////////////////////////////////////
// Tweaked virtual world (GL4Java version)
// by Jacob Marner. Feb 2002.
//
// A generic 3D vector class of floats.
// Note that objects of this class are mutable.
//
//////////////////////////////////////////////////////////////////

final public class Vector
{
    public float x;
    public float y;
    public float z;

    public Vector()
    {
        x = 0;
        y = 0;
        z = 0;
    }

    public Vector(final float x, final float y, final float z)
    {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    public Vector(final Vector v)

```

```

{
    x = v.x;
    y = v.y;
    z = v.z;
}

final public Vector add(final Vector v)
{
    return new Vector(x + v.x, y + v.y, z + v.z);
}

final public Vector subtract(final Vector v)
{
    return new Vector(x - v.x, y - v.y, z - v.z);
}

final public Vector divide(float c)
{
    return new Vector(x / c, y / c, z / c);
}

final public Vector negate()
{
    return new Vector(-x, -y, -z);
}

final public void negated()
{
    x = -x;
    y = -y;
    z = -z;
}

final public Vector multiply(final float c)
{
    return new Vector(x * c, y * c, z * c);
}

final public void added(final Vector v)
{
    x += v.x;
    y += v.y;
    z += v.z;
}

final public void subtracted(final Vector v)
{
    x -= v.x;
    y -= v.y;
    z -= v.z;
}

final public void multiplied(final float c)
{
    x *= c;
    y *= c;
    z *= c;
}

final public void divided(final float c)
{
    x /= c;
    y /= c;
    z /= c;
}

// Copy vector
final public void assign(final Vector v)
{
    x = v.x;
    y = v.y;
}

```

```

    z = v.z;
}

final public void assign(final float x, final float y, final float z)
{
    this.x = x;
    this.y = y;
    this.z = z;
}

// Get 2-norm
final public float getLength()
{
    return (float) Math.sqrt(x * x + y * y + z * z);
}

// Get max of each element (not the same as infinity norm)
final public float getMax()
{
    return Math.max(Math.max(x, y), z);
}

final public float getInfinityNorm()
{
    return Math.max(Math.max(Math.abs(x), Math.abs(y)), Math.abs(z));
}

// Get 2-norm but before the square root is taken
final public float getSquaredLength()
{
    return x * x + y * y + z * z;
}

// Return a normalized version (unit length) of the
// vector.
final public Vector normalized()
{
    final float l = getLength();
    return new Vector(x / l, y / l, z / l);
}

final public void normalize()
{
    final float l = getLength();
    x /= l;
    y /= l;
    z /= l;
}

final public void invert()
{
    x = -x;
    y = -y;
    z = -z;
}

final public Vector inverted()
{
    return new Vector(-x, -y, -z);
}

final public Vector cross(final Vector v)
{
    return new Vector(y * v.z - z * v.y, z * v.x - x * v.z, x * v.y - y * v.x);
}

final public void crossedWith(final Vector v)
{
    final float tx = y * v.z - z * v.y;
    final float ty = z * v.x - x * v.z;
    z = x * v.y - y * v.x;
}

```

```

    x = tx;
    y = ty;
}

// Return true if this vector is a position between the lower and upper
// corner given of a rectangle. Each coordinate of lower must be lower
// than the same of that of upper.
final public boolean isInRectangle(
    final Vector lowerCorner,
    final Vector upperCorner)
{
    return (
        x >= lowerCorner.x
        && y >= lowerCorner.y
        && z >= lowerCorner.z
        && x < upperCorner.x
        && y < upperCorner.y
        && z < upperCorner.z);
}

// Return x on 0, y on 1 and z on 2.
final public float index(final int n)
{
    switch (n)
    {
        case 0 :
            return x;
        case 1 :
            return y;
        case 2 :
            return z;
        default :
            throw new ArrayIndexOutOfBoundsException();
    }
}

// Do matrix result, but do not divide through with w
// afterwards. Just throw w away (it is in fact never
// calculated. The last row of ma is thus not used.
final public void MultLeftMatrixIgnoreW(final Matrix ma)
{
    final float[] m = ma.m;
    final float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    final float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    final float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
    x = x1;
    y = y1;
    z = z1;
}

final public void MultLeftMatrix(final Matrix ma)
{
    final float[] m = ma.m;
    final float x1 = m[0] * x + m[4] * y + m[8] * z + m[12];
    final float y1 = m[1] * x + m[5] * y + m[9] * z + m[13];
    final float z1 = m[2] * x + m[6] * y + m[10] * z + m[14];
    final float w1 = m[3] * x + m[7] * y + m[11] * z + m[15];
    x = x1;
    y = y1;
    z = z1;
    if (w1 != 1)
    {
        x /= w1;
        y /= w1;
        z /= w1;
    }
}

final public float dot(final Vector v)
{
    return x * v.x + y * v.y + z * v.z;
}

```

```
}  
};
```

gl4java/utils/textures

```
package gl4java.utils.textures;  
  
import gl4java.*;  
  
import java.awt.*;  
import java.awt.image.*;  
import java.awt.Color.*;  
import java.awt.event.*;  
import java.applet.*;  
import java.io.*;  
import java.net.*;  
  
/**  
 * This class implements a Window .BMP  
 * texture loader. This loader implements the  
 * full .BMP specification.  
 *  
 * Before using the loaded texture data you  
 * should call  
 * glPixelStorei(GL_UNPACK_ALIGNMENT, 1);  
 * The rest of the PixelStore  
 * arguments should stay at the default.  
 *  
 * The Windows BMP format does not support  
 * multiple layers or transparency. It supports  
 * 1-bit, 4-bit, 8-bit and 24-bit colors.  
 * The 4-bit and 8-bit mode can be RLE encoded.  
 * This loader supports all the variants.  
 *  
 * @author Jacob Marner jacob@marner.dk  
 * @see IOTextureLoader  
 * @see TextureLoader  
 */  
public class BmpTextureLoader extends IOTextureLoader  
{  
    public BmpTextureLoader(GLFunc gl, GLUFunc glu)  
    {  
        super(gl, glu);  
    }  
  
    protected boolean readTexture(InputStream is)  
    {  
        byte[][] bitmapData = new byte[1][];  
        int res = loadBMP(is, bitmapData);  
  
        if (res!=SUCCESS)  
        {  
            error=true;  
            return false;  
        }  
  
        imageWidth = width;  
        imageHeight = height;  
  
        glFormat = GL_RGB;  
  
        pixel = bitmapData[0];  
  
        setTextureSize();  
        return true;  
    }  
  
    // The operation ended successfully  
    private final static int SUCCESS = 0;
```

```

// The file could not be found or it could not be opened for reading.
private final static int COULD_NOT_FIND_OR_READ_FILE = 1;

// The file does not comply with the specification.
private final static int ILLEGAL_FILE_FORMAT = 2;

// OpenGL could not accept the texture. You probably used a internal
// format not accepted by your OpenGL implementation or you may have run
// out of available texture names.
private final static int OPENGL_ERROR = 3;

// The system ran out of heap memory during the texture load.
private final static int OUT_OF_MEMORY = 4;

// Loads the specified BMP image and stores it into a OpenGL 2D texture, whose
// name will be returned in textureName. If an error occurred, textureName is
// undefined. The status of the operation will be returned as the
// return value.

// The bitmap will be loaded without borders and all mip maps will be
// generated automatically. If you need the bitmap for anything else than
// textures, or you need borders you will have to edit the source code.

// Parameters:
// [in] filename: The name of the texture file. Must be Windows .BMP format.
// The actual file *must* use the .bmp extension. For the filename string
// given as parameter the extension is optional - if it is not there
// it is appended automatically.
// [in/out] textureName: Pass a pointer to an already allocated GLuint and if
// the operation success then this data will point to the texture name
// of the texture object loaded. This can immediately be used with
// glBindTexture() to use the texture during drawing. If the return value
// is not LOAD_TEXTUREBMP_SUCCESS then this value is undefined. If you
// later want to delete the loaded texture call glDeleteTextures() on the
// returned texture name.
// [in] internalformat: The internal format of the loaded texture. Is passed
// directly to glTexImage2D(). Defaults to GL_RGB.
// Return value:
// [out] Status of operation.

private InputStream file;

// The offset from the BITMAPFILEHEADER structure
// to the actual bitmap data in the file.
private int byteOffset;

// Image width in pixels
private int width;

// Image height in pixels
private int height;

// Number of bit per pixel. Is 1, 4, 8, 24. If it is 1,4 or 8 then
// images is paletted, otherwise not.
private int bitCount;

// Compression
private final static int BI_RGB = 0;
private final static int BI_RLE8 = 1;
private final static int BI_RLE4 = 2;
private static int compression;

// Palette used for paletted images during load.
private byte[][] palette = new byte[3][256];

// The number of bytes read so far
private int bytesRead;

// Reads and returns a 32-bit value from the file.
private int read32BitValue()
{

```

```

bytesRead += 4;

try
{
    int c1 = file.read();
    int c2 = file.read();
    int c3 = file.read();
    int c4 = file.read();

    return c1 + (c2 << 8) + (c3 << 16) + (c4 << 24);
}
catch (IOException e)
{
    return 0;
}
}

// Reads and returns a 16-bit value from the file
private short read16BitValue()
{
    bytesRead += 2;

    try
    {
        int c1 = file.read();
        int c2 = file.read();

        return (short)(c1 + (c2 << 8));
    }
    catch(IOException e)
    {
        return 0;
    }
}

// Reads and returns a 8-bit value from the file (0-255)
private int read8BitValue()
{
    bytesRead += 1;

    try
    {
        int c1 = file.read();

        return c1;
    }
    catch(IOException e)
    {
        return 0;
    }
}

// In Windows terms read this structure
// typedef struct tagBITMAPFILEHEADER { /* bmfh */
//     UINT    bfType;
//     DWORD   bfSize;
//     UINT    bfReserved1;
//     UINT    bfReserved2;
//     DWORD   bfOffBits;
// } BITMAPFILEHEADER;
private int readFileHeader()
{
    // Read "BM" tag in ASCII.
    if (read8BitValue() != 66 || read8BitValue() != 77)
        return ILLEGAL_FILE_FORMAT;

    // Read file size. We do not need this. Ignore.
    read32BitValue();

    // Skip the two reserved areas
    read16BitValue();
}

```

```

    read16BitValue();

    // Read the byte offset
    byteOffset = read32BitValue();

    return SUCCESS;
}

// In windows terms read this struct:
//typedef struct tagBITMAPINFOHEADER {    /* bmih */
//    DWORD    biSize;
//    LONG     biWidth;
//    LONG     biHeight;
//    WORD     biPlanes;
//    WORD     biBitCount;
//    DWORD    biCompression;
//    DWORD    biSizeImage;
//    LONG     biXPelsPerMeter;
//    LONG     biYPelsPerMeter;
//    DWORD    biClrUsed;
//    DWORD    biClrImportant;
//} BITMAPINFOHEADER;
private int readInfoHeader()
{
    // We expect this to be at least 40. If it is larger we read
    // some more bytes in the end to be forward compatible.
    int sizeofInfoHeader = (int) read32BitValue();

    if (sizeofInfoHeader < 40)
        return ILLEGAL_FILE_FORMAT;

    width = read32BitValue();

    height = read32BitValue();

    // Read number of planes. According to the specification this
    // must be 1.
    if (read16BitValue() != 1)
        return ILLEGAL_FILE_FORMAT;

    bitCount = read16BitValue();

    if (bitCount != 1 && bitCount != 4 && bitCount != 8 && bitCount != 24)
        return ILLEGAL_FILE_FORMAT;

    compression = read32BitValue();

    if (compression != BI_RGB && compression != BI_RLE8 && compression != BI_RLE4)
        return ILLEGAL_FILE_FORMAT;

    // Read image size. We do not need this since we have the
    // image size.
    read32BitValue();

    // Pixel to device mapping. This is irrelevant since we simply
    // want the bitmap.
    read32BitValue();
    read32BitValue();

    // Read colors used. We do not need this, so it is ignored.
    read32BitValue();

    // Read the number of important colors. This will not be needed
    // in OpenGL so we will ignore this.
    read32BitValue();

    // Apply padding in end of header to be forward compatible.
    sizeofInfoHeader -= 40;
    while (sizeofInfoHeader > 0)
    {
        read8BitValue();
    }
}

```

```

        sizeofInfoHeader--;
    }

    return SUCCESS;
}

// The palette follows directly after the
// info header
//
//typedef struct tagRGBQUAD {      /* rgbq */
// BYTE    rgbBlue;
// BYTE    rgbGreen;
// BYTE    rgbRed;
// BYTE    rgbReserved;
// } RGBQUAD;
private int readPalette()
{
    // 24-bit images are not paletted.
    if (bitCount == 24)
        return SUCCESS;

    int numColors = 1 << bitCount;
    for (int i = 0; i < numColors; i++)
    {
        // Read RGB.
        for (int j = 2; j >= 0; j--)
            palette[j][i] = (byte)read8BitValue();

        // Skip reversed byte.
        read8BitValue();
    }

    return SUCCESS;
}

private int readBitmapData1Bit(byte[] bitmapData)
{
    // 1-bit format cannot be compressed
    if (compression != BI_RGB)
        return ILLEGAL_FILE_FORMAT;

    int byteRead = 0;
    for (int y = 0; y < height; y++)
    {
        int index = y * width * 3;
        for (int x = 0; x < width; x++)
        {
            if (x % 8 == 0)
            {
                byteRead = read8BitValue();

                char color = (char)((byteRead >> (7 - ((char)(x % 8)))) & 1);

                bitmapData[index + x * 3] = palette[0][color];
                bitmapData[index + x * 3 + 1] = palette[1][color];
                bitmapData[index + x * 3 + 2] = palette[2][color];
            }

            // Pad to 32-bit boundary.
            while (bytesRead % 4 != 0)
                read8BitValue();
        }

        return SUCCESS;
    }

    // This is called after the first byte has been found to be 0
    // and the second to be 0-2. This is only used in RLE-4 and RLE-8
    // encoding.
    private boolean handleEscapeCode(

```

```

int secondByte,
int[] x,
int[] y,
int[] res)
{
    if (secondByte == 0x00)
    {
        // End of line
        x[0] = 0;
        if (y[0] >= height)
        {
            res[0] = ILLEGAL_FILE_FORMAT;
            return true;
        }
        (y[0]) ++;
    }
    else if (secondByte == 0x01)
    {
        // End of bitmap
        res[0] = SUCCESS;
        return true;
    }
    else // secondByte=0x02
    {
        // Delta. Move drawing cursor.
        x[0] += read8BitValue();
        y[0] += read8BitValue();
        if (x[0] >= width || x[0] < 0 || y[0] >= height || y[0] < 0)
        {
            res[0] = ILLEGAL_FILE_FORMAT;
            return true;
        }
    }

    return false;
}

// Draw a 4-bit image. Can be uncompressed or RLE-4 compressed.
private int readBitmapData4Bit(byte[] bitmapData)
{
    if (compression != BI_RGB && compression != BI_RLE4)
        return ILLEGAL_FILE_FORMAT;

    // Uncompressed 4 bit encoding
    if (compression == BI_RGB)
    {
        for (int j = 0; j < height; j++)
        {
            int index = j * width * 3;
            int byteValue = 0;
            int color;
            for (int i = 0; i < width; i++)
            {
                if (i % 2 == 0)
                {
                    byteValue = read8BitValue();
                    color = (byte)(byteValue >> 4);
                }
                else
                {
                    color = (byte)(byteValue & 0x0F);
                }

                bitmapData[index + i * 3] = palette[0][color];
                bitmapData[index + i * 3 + 1] = palette[1][color];
                bitmapData[index + i * 3 + 2] = palette[2][color];
            }

            // Pad to 32-bit boundary.
            for (int k = 0; k < (((width + 1) / 2) % 4); k++)
                read8BitValue();
        }
    }
}

```

```

    }
}

// RLE encoded 4-bit compression
if (compression == BI_RLE4)
{
    // Drawing cursor pos
    int[] x = new int[1];
    int[] y = new int[1];
    x[0] = 0;
    y[0] = 0;

    // Clear bitmap data since it is legal not to
    // fill it all out.
    for (int i=0;i<width*height*3;i++)
        bitmapData[i] = 0;

    bytesRead = 0;

    while (true)
    {
        int firstByte = read8BitValue();
        int secondByte = read8BitValue();

        // Is this an escape code or absolute encoding?
        if (firstByte == 0)
        {
            // Is this an escape code
            if (secondByte <= 0x02)
            {
                int[] res = new int[1];
                if (handleEscapeCode(secondByte, x, y, res))
                    return res[0];
            }
            else
            {
                // Absolute encoding
                int index = y[0] * width * 3;
                int color;
                int databyte = 0;
                for (int i = 0; i < secondByte; i++)
                {
                    if (x[0] >= width)
                        return ILLEGAL_FILE_FORMAT;

                    if (i % 2 == 0)
                    {
                        databyte = read8BitValue();
                        color = databyte >> 4;
                    }
                    else
                    {
                        color = databyte & 0x0F;
                    }

                    bitmapData[index + x[0] * 3] = palette[0][color];
                    bitmapData[index + x[0] * 3 + 1] = palette[1][color];
                    bitmapData[index + x[0] * 3 + 2] = palette[2][color];
                    (x[0])++;
                }

                // Pad to 16-bit word boundary
                while (bytesRead % 2 != 0)
                    read8BitValue();
            }
        }
        else
        {
            // If not absolute or escape code, perform data decode for next
            // length of colors
            int color1 = secondByte >> 4;

```

```

        int color2 = secondByte & 0x0F;

        for (int i = 0; i < firstByte; i++)
        {
            if (x[0] >= width)
                return ILLEGAL_FILE_FORMAT;

            int color;
            if (i % 2 == 0)
                color = color1;
            else
                color = color2;

            int index = y[0] * width * 3 + x[0] * 3;
            bitmapData[index] = palette[0][color];
            bitmapData[index + 1] = palette[1][color];
            bitmapData[index + 2] = palette[2][color];
            (x[0])++;
        }
    }
}

return SUCCESS;
}

// Read 8-bit image. Can be uncompressed or RLE-8 compressed.
private int readBitmapData8Bit(byte[] bitmapData)
{
    if (compression != BI_RGB && compression != BI_RLE8)
        return ILLEGAL_FILE_FORMAT;

    if (compression == BI_RGB)
    {
        // For each scan line
        for (int i = 0; i < height; i++)
        {
            int index = i * width * 3;
            for (int j = 0; j < width; j++)
            {
                int color = read8BitValue();
                bitmapData[index + j * 3] = palette[0][color];
                bitmapData[index + j * 3 + 1] = palette[1][color];
                bitmapData[index + j * 3 + 2] = palette[2][color];
            }

            // go to next alignment of 4 bytes.
            for (int k = 0; k < (width % 4); k++)
                read8BitValue();
        }
    }

    if (compression == BI_RLE8)
    {
        // Drawing cursor pos
        int[] x = new int[1];
        int[] y = new int[1];
        x[0] = 0;
        y[0] = 0;

        bytesRead = 0;

        // Clear bitmap data since it is legal not to
        // fill it all out.
        for (int i=0;i<width*height*3;i++)
            bitmapData[i] = 0;

        while (true)
        {
            int firstByte = read8BitValue();
            int secondByte = read8BitValue();

```

```

// Is this an escape code or absolute encoding?
if (firstByte == 0)
{
    // Is this an escape code
    if (secondByte <= 0x02)
    {
        int[] res = new int[1];
        if (handleEscapeCode(secondByte, x, y, res))
            return res[0];
    }
    else
    {
        // Absolute encoding
        int index = y[0] * width * 3;
        for (int i = 0; i < secondByte; i++)
        {
            if (x[0] >= width)
                return ILLEGAL_FILE_FORMAT;
            int color = read8BitValue();
            bitmapData[index + x[0] * 3] = palette[0][color];
            bitmapData[index + x[0] * 3 + 1] = palette[1][color];
            bitmapData[index + x[0] * 3 + 2] = palette[2][color];
            (x[0])++;
        }

        // Pad to 16-bit word boundary
        if (secondByte % 2 == 1)
            read8BitValue();
    }
}
else
{
    // If not, perform data decode for next length of colors
    for (int i = 0; i < firstByte; i++)
    {
        if (x[0] >= width)
            return ILLEGAL_FILE_FORMAT;
        int index = y[0] * width * 3 + x[0] * 3;
        bitmapData[index] = palette[0][secondByte];
        bitmapData[index + 1] = palette[1][secondByte];
        bitmapData[index + 2] = palette[2][secondByte];
        (x[0])++;
    }
}
}

return SUCCESS;
}

// Load a 24-bit image. Cannot be encoded.
private int readBitmapData24Bit(byte[] bitmapData)
{
    // 24-bit bitmaps cannot be encoded. Verify this.
    if (compression != BI_RGB)
        return ILLEGAL_FILE_FORMAT;

    for (int i = 0; i < height; i++)
    {
        int index = i * width * 3;
        for (int j = 0; j < width; j++)
        {
            bitmapData[index + j * 3 + 2] = (byte)read8BitValue();
            bitmapData[index + j * 3 + 1] = (byte)read8BitValue();
            bitmapData[index + j * 3] = (byte)read8BitValue();
        }

        // go to next alignment of 4 bytes.
        for (int k = 0; k < ((width * 3) % 4); k++)
            read8BitValue();
    }
}

```

```

    }

    return SUCCESS;
}

private int readBitmapData(byte[] bitmapData)
{
    // Pad until byteoffset. Most images will need no padding
    // since they already are made to use as little space as
    // possible.
    while (bytesRead < byteOffset)
        read8BitValue();

    // The data reading procedure depends on the bit depth.
    switch (bitCount)
    {
        case 1 :
            return readBitmapData1Bit(bitmapData);
        case 4 :
            return readBitmapData4Bit(bitmapData);
        case 8 :
            return readBitmapData8Bit(bitmapData);
        default : // 24 bit.
            return readBitmapData24Bit(bitmapData);
    }
}

// Load the BMP. Assumes that no extension is appended to the filename.
// If successful *bitmapData will contain a pointer to the data.
private int loadBMP(
    InputStream filep,
    byte[][] bitmapData)
{
    file = filep;

    bitmapData[0] = null;

    // Open file for buffered read.
    int res = SUCCESS;

    bytesRead = 0;

    // Read File Header
    if (res==0)
        res = readFileHeader();

    // Read Info Header
    if (res==0)
        res = readInfoHeader();

    // Read Palette
    if (res==0)
        res = readPalette();

    if (res==0)
    {
        // The bitmap data we are going to hand to OpenGL
        bitmapData[0] = new byte[width * height * 3];

        if (bitmapData[0]==null)
            res = OUT_OF_MEMORY;
    }

    if (res==0)
    {
        // Read Data
        res = readBitmapData(bitmapData[0]);
    }

    return res;
}

```

}

Part 3: Java Java3D source code

The source code consists of the following files:

Arguments.java	This file is used when parsing command line arguments
BoxData.java	This class represents the data generated to represent a box.
Main.java	The main class. Start this to run the application.
MyCanvas3D.java	The class derived from Canvas3D.
VisualBox.java	A Shape3D used to draw a box.

Arguments.java

```
////////////////////////////////////  
// Virtual World (Java3D version)  
// by Jacob Marner. Feb 2002.  
//  
// This file is used when parsing command line arguments  
//  
////////////////////////////////////  
  
public class Arguments  
{  
    String[] args;  
  
    public Arguments(String[] args)  
    {  
        this.args=args;  
    }  
  
    public boolean getBooleanFlag(String label)  
    {  
        for(int i=0;i<args.length;i++)  
        {  
            if (args[i].equals(label))  
            {  
                return true;  
            }  
        }  
        return false;  
    }  
  
    public int getIntegerArgument(String label, int defaultvalue)  
    {  
        for(int i=0;i<args.length;i++)  
        {  
            if (args[i].startsWith(label))  
            {  
                String s = args[i].substring(label.length());  
                try  
                {  
                    return Integer.parseInt(s);  
                }  
                catch(NumberFormatException e)  
                {  

```

```

        return defaultvalue;
    }
}
return defaultvalue;
}

public float getFloatArgument(String label, float defaultvalue)
{
    for(int i=0;i<args.length;i++)
    {
        if (args[i].startsWith(label))
        {
            String s = args[i].substring(label.length());
            try
            {
                return Float.parseFloat(s);
            }
            catch(NumberFormatException e)
            {
                return defaultvalue;
            }
        }
    }
    return defaultvalue;
}
}
}

```

BoxData.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Virtual World (Java3D version)
// by Jacob Marner. Feb 2002.
//
// This class represent the data generated to represent
// a box.
// These are converted to 3D geometry when the kd-tree
// (scene graph) is built.
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

import javax.vecmath.*;

public class BoxData
{
    public float scale;
    public Vector3f position = new Vector3f();
    public Vector3f rotation = new Vector3f();

    public void assign(BoxData b)
    {
        scale = b.scale;
        position.set(b.position);
        rotation.set(b.rotation);
    }
};

```

Main.java

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Virtual World (Java3D version)
// by Jacob Marner. Feb 2002.
//
// The main class. Start this to run the application.
//
// The application takes two optional parameters:
// -frames:<int> : The number of frames to run the animation. Default: 3000.
// -boxes:<int> : The number of boxes to generate. Default: 1000.
//

```

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
import java.awt.*;
import com.sun.j3d.utils.universe.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import javax.swing.*;

public class Main extends JFrame
{
    static Main frame;
    MyCanvas3D canvas3D;

    static TransformGroup viewTransformGroup;

    public Main(String s)
    {
        super(s);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        int width = 640;
        int height = 480;

        // Set the size of the Window
        getContentPane().setLayout(new BorderLayout());
        pack();
        Insets i = getInsets();
        setBounds(0, 0, width + i.right + i.left, height + i.bottom + i.top);

        // Initialize
        GraphicsConfiguration config = SimpleUniverse.getPreferredConfiguration();
        canvas3D = new MyCanvas3D(config);
        getContentPane().add("Center", canvas3D);

        BranchGroup scene = canvas3D.createSceneGraph();

        createUniverse(scene, canvas3D);

        setVisible(true);
    }

    public static void main(String[] args)
    {
        Arguments arg = new Arguments(args);

        MyCanvas3D.numBoxes = arg.getIntegerArgument("-boxes:", 1000);
        MyCanvas3D.numFrames = arg.getIntegerArgument("-frames:", 3000);

        // Create Window
        frame = new Main("Virtual World Java3D");
    }

    public void createUniverse(BranchGroup scene, Canvas3D canvas3D)
    {
        VirtualUniverse universe = new VirtualUniverse();

        Locale locale = new Locale(universe);

        // Create viewing branch
        BranchGroup viewBranch = new BranchGroup();
        viewTransformGroup = new TransformGroup();
        viewTransformGroup.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
        viewBranch.addChild(viewTransformGroup);

        ViewPlatform viewPlatform = new ViewPlatform();
        viewTransformGroup.addChild(viewPlatform);

        View view = new View();
        view.attachViewPlatform(viewPlatform);
        view.addCanvas3D(canvas3D);
    }
}

```

```

PhysicalBody physicalBody = new PhysicalBody();
view.setPhysicalBody(physicalBody);

PhysicalEnvironment physicalEnvironment = new PhysicalEnvironment();
view.setPhysicalEnvironment(physicalEnvironment);

// Set viewing parameters to ensure correct view frustum
// I have avoided compatibility mode
double frustumfar = 40.0;

view.setFieldOfView(Math.PI / 2.0);
view.setFrontClipPolicy(View.VIRTUAL_EYE);
view.setBackClipPolicy(View.VIRTUAL_EYE);
view.setFrontClipDistance(0.5);
view.setBackClipDistance(frustumfar);
view.setScreenScalePolicy(View.SCALE_EXPLICIT);

// Disable all behaviors for speed. This makes the
// application somewhat faster and is only possible
// because we do not use behaviors.
view.stopBehaviorScheduler();

// Add fog
BranchGroup bg = new BranchGroup();
LinearFog fog =
    new LinearFog(new Color3f(0.0f, 0.0f, 0.0f), frustumfar * 0.85, frustumfar);
fog.setInfluencingBounds(new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 1000));
bg.addChild(fog);

// Add global directional light
DirectionalLight light = new DirectionalLight();
light.setDirection(0.0f, -1.0f, 1.0f);
light.setColor(new Color3f(1.0f, 1.0f, 1.0f));
light.setEnabled(true);
light.setInfluencingBounds(
    new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 1000));
bg.addChild(light);

// Add vague global ambient light (this is enabled
// by default in OpenGL versions)
AmbientLight alight = new AmbientLight(true, new Color3f(0.4f, 0.4f, 0.4f));
alight.setInfluencingBounds(
    new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 1000));
bg.addChild(alight);

bg.compile();
locale.addBranchGraph(bg);

// Activate scene graph - making it live
locale.addBranchGraph(scene);
viewBranch.compile();
locale.addBranchGraph(viewBranch);
}
}

```

MyCanvas3D.java

```

////////////////////////////////////
// Virtual World (Java3D version)
// by Jacob Marner. Feb 2002.
//
// The canvas on which the drawing occurs. We here
// perform the drawing and timing. We also do the scene
// graph building.
//
////////////////////////////////////

import java.awt.*;
import com.sun.j3d.utils.universe.*;

```

```

import com.sun.j3d.utils.geometry.*;
import javax.media.j3d.*;
import javax.vecmath.*;
import java.util.*;
import com.sun.j3d.utils.image.*;
import java.io.*;

public class MyCanvas3D extends Canvas3D
{
    int frame = 0;
    Date starttime;
    static int numBoxes;
    static int numFrames;

    public final static int WORLD_MAX_X = 100;
    public final static int WORLD_MAX_Y = 100;
    public final static int WORLD_MAX_Z = 100;
    static final int MAX_BOXES_PER_LEAF = 10;

    MyCanvas3D(GraphicsConfiguration graphicsConfiguration)
    {
        super(graphicsConfiguration);
    }

    private void startTimer()
    {
        starttime = new Date();
    }

    private void stopTimer()
    {
        Date endTime = new Date();
        float timeelapsed = (endTime.getTime() - starttime.getTime()) / 1000.0f;
        System.out.println("Elapsed time: " + timeelapsed + " sec.");
        System.out.println(
            "Average frame rate: " + ((float) numFrames) / timeelapsed + " fps.");
    }

    public void preRender()
    {
        display();

        super.preRender();
    }

    public void postRender()
    {
        super.postRender();

        if (frame == 0)
            startTimer();

        frame++;
        if (frame >= numFrames)
        {
            stopTimer();
            System.exit(0);
        }
    }

    Random rand;

    // Generate a random number between 0 and maxValue.
    private float getRandomFloat(float minValue, float maxValue)
    {
        float interval = maxValue - minValue;
        float offset = rand.nextFloat() * interval;
        return minValue + offset;
    }

    static public Texture2D texture;

```

```

public BranchGroup createSceneGraph()
{
    // Generate numBoxes box data put them in array. For each one
    // store scale(1), pos(3), rotation(3). Generate each number
    // randomly.
    rand = new Random(new Date().getTime());

    BoxData[] boxData = new BoxData[numBoxes];

    int i;
    for (i = 0; i < numBoxes; ++i)
    {
        boxData[i] = new BoxData();
        boxData[i].scale = getRandomFloat(0.1f, 1.5f);
        boxData[i].position.x = getRandomFloat(0.0f, WORLD_MAX_X);
        boxData[i].position.y = getRandomFloat(0.0f, WORLD_MAX_Y);
        boxData[i].position.z = getRandomFloat(0.0f, WORLD_MAX_Z);
        boxData[i].rotation.x = getRandomFloat(0.0f, 360.0f);
        boxData[i].rotation.y = getRandomFloat(0.0f, 360.0f);
        boxData[i].rotation.z = getRandomFloat(0.0f, 360.0f);
    }

    // Load texture
    String filename = "glow.jpg";
    TextureLoader loader =
        new TextureLoader(filename, TextureLoader.GENERATE_MIPMAP, this);

    ImageComponent2D image = null;
    try
    {
        image = loader.getImage();
    }
    catch (NullPointerException e)
    {
        System.exit(0);
    }

    int imageWidth = image.getWidth();
    int imageHeight = image.getHeight();

    texture =
        new Texture2D(Texture.MULTI_LEVEL_MIPMAP, Texture.RGB,
            imageWidth, imageHeight);

    // Generate MipMaps
    int imageLevel = 0;
    texture.setImage(imageLevel, image);

    while (imageWidth > 1 || imageHeight > 1)
    { // loop until size: 1x1
        imageLevel++; // compute this level

        if (imageWidth > 1)
            imageWidth /= 2; // adjust width as necessary
        if (imageHeight > 1)
            imageHeight /= 2; // adjust height as necessary

        image = loader.getScaledImage(imageWidth, imageHeight);
        texture.setImage(imageLevel, image);
    }

    // Trilinear filtering
    texture.setMagFilter(Texture.BASE_LEVEL_LINEAR);
    texture.setMinFilter(Texture.MULTI_LEVEL_LINEAR);

    // Create scene
    BranchGroup root = new BranchGroup();
    populateScene(root, boxData, 0, numBoxes);

    root.compile();
}

```

```

    return root;
}

private void populateScene(
    BranchGroup bg,
    BoxData boxData[],
    int startindex,
    int numindex)
{
    // First min box then max box
    Point3d[] boundingBox = new Point3d[2];

    boundingBox[0] = new Point3d(WORLD_MAX_X, WORLD_MAX_Y, WORLD_MAX_Z);
    boundingBox[1] = new Point3d(0, 0, 0);
    // Find and set bounding box by iterating through boxes.
    // (Boxes is centered about position and has a distance
    // from center of max sqrt(2*pow(scale,2)) )

    for (int c = startindex; c < startindex + numindex; ++c)
    {
        float boxMaxDistFromCenter =
            (float) Math.sqrt(2.0f * (float) Math.pow(boxData[c].scale, 2));
        float candidate;

        candidate = boxData[c].position.x - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].x)
            boundingBox[0].x = candidate;

        candidate = boxData[c].position.y - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].y)
            boundingBox[0].y = candidate;

        candidate = boxData[c].position.z - boxMaxDistFromCenter;
        if (candidate < boundingBox[0].z)
            boundingBox[0].z = candidate;

        candidate = boxData[c].position.x + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].x)
            boundingBox[1].x = candidate;

        candidate = boxData[c].position.y + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].y)
            boundingBox[1].y = candidate;

        candidate = boxData[c].position.z + boxMaxDistFromCenter;
        if (candidate > boundingBox[1].z)
            boundingBox[1].z = candidate;
    }

    // Does using a bounding sphere instead of bounding box make
    // it faster. No! It makes it slower but only a little. I tried.

    /*
    Point3d bcenter = new Point3d(boundingBox[1]);
    bcenter.sub(boundingBox[0]);
    bcenter.scale(0.5f);
    double radius = Math.sqrt(bcenter.x*bcenter.x+bcenter.y*bcenter.y+
        bcenter.z*bcenter.z);
    bg.setBounds(new BoundingSphere(bcenter, radius));
    */

    bg.setBounds(new BoundingBox(boundingBox[0], boundingBox[1]));

    // If n is more than max_boxes
    if (numindex > MAX_BOXES_PER_LEAF)
    {
        int splitn;
        if (numindex > 2)
        {
            double xspan = boundingBox[1].x - boundingBox[0].x;
            double yspan = boundingBox[1].y - boundingBox[0].y;

```



```

        rotZ.rotZ(boxData[i].rotation.z);
        transform.mul(rotZ);

        scale.setScale(boxData[i].scale);
        transform.mul(scale);

        TransformGroup tg = new TransformGroup();
        tg.setTransform(transform);
        bg.addChild(tg);

        // Add some geometry under the transform group
        //tg.addChild(new ColorCube(0.4));
        tg.addChild(new VisualBox(boxData[i]));
    }
}

static float vector3fIndex(Vector3f v, int n)
{
    switch (n)
    {
        case 0 :
            return v.x;
        case 1 :
            return v.y;
        case 2 :
            return v.z;
        default :
            throw new ArrayIndexOutOfBoundsException();
    }
}

// Reorder boxData array according to value at given offset. If offset is 0 then
// x is used. 1 gives y. 2 gives z.
static int reorder(
    BoxData boxData[],
    int startindex,
    int numindex,
    double splitvalue,
    int offset)
{
    BoxData temp = new BoxData();

    if (numindex < 1)
        return 0;

    int r = startindex + numindex - 1;
    int p = startindex;

    int i = p - 1;
    int j = r + 1;
    while (true)
    {
        i++;
        j--;

        while (j >= p && vector3fIndex(boxData[j].position, offset) > splitvalue)
            j--;

        while (i <= r && vector3fIndex(boxData[i].position, offset) < splitvalue)
            i++;

        if (i < j)
        {
            temp.assign(boxData[i]);
            boxData[i].assign(boxData[j]);
            boxData[j].assign(temp);
        }
        else
        {
            while (j < r && vector3fIndex(boxData[j].position, offset) < splitvalue)

```

```

        j++;
        if (j <= p)
            j = p;

        return j;
    }
}

Point3d cameraPos = new Point3d();
Point3d cameraDirection = new Point3d();
Point3d endPoint = new Point3d();
Vector3d upVector = new Vector3d(0, 1, 0);
Transform3D v = new Transform3D();

private void display()
{
    // Move camera in circle around center of world.
    float angle = (float) frame / 600.0f;
    float radius = 20;
    cameraPos.set(
        (WORLD_MAX_X / 2.0f + radius * Math.cos(angle)),
        WORLD_MAX_Y / 2.0f,
        (WORLD_MAX_Z / 2.0f + radius * Math.sin(angle)));
    float newAngle = angle + 3.14f / 2.0f;
    cameraDirection.set((Math.cos(newAngle)), 0.0, (Math.sin(newAngle)));

    endPoint.add(cameraPos, cameraDirection);

    v.lookAt(cameraPos, endPoint, upVector);
    v.invert(); // We need to invert the matrix.
    // See the Java3D 1.3 spec beta 1 p. 294-295.

    Main.viewTransformGroup.setTransform(v);
}
}

```

VisualBox.java

```

////////////////////////////////////
// Virtual World (Java3D version)
// by Jacob Marner. Feb 2002.
//
// The generation of a single box in the scene graph.
//
////////////////////////////////////

import javax.media.j3d.*;
import javax.vecmath.*;

public class VisualBox extends Shape3D
{
    private Geometry geometry;
    static private Appearance appearance = null;

    public VisualBox(BoxData b)
    {
        geometry = createGeometry();
        appearance = createAppearance();
        setGeometry(geometry);
        setAppearance(appearance);
    }

    private void drawVertex(QuadArray g, int index, float t1, float t2,
                           float c1, float c2, float c3,
                           float n1, float n2, float n3)
    {
        g.setCoordinate(index, new Point3f(c1, c2, c3));
        g.setTextureCoordinate(0, index, new TexCoord2f(t1, t2));
        g.setNormal(index, new Vector3f(n1, n2, n3));
    }
}

```

```

}

private Geometry createGeometry()
{
    // We do not refer geometry by reference since
    // that would prevent the elimination of
    // the transform node above the shape3d during
    // compilation of the scene graph (this optimization
    // is only applied in Java3D 1.3 and later)

    QuadArray g = new QuadArray(4 * 6, GeometryArray.COORDINATES |
        GeometryArray.NORMALS |
        GeometryArray.TEXTURE_COORDINATE_2);

    int i=0;

    // left side
    drawVertex(g,i++,1,0,-1,-1,1,-1,0,0);
    drawVertex(g,i++,1,1,-1,1,1,-1,0,0);
    drawVertex(g,i++,0,1,-1,1,-1,-1,0,0);
    drawVertex(g,i++,0,0,-1,-1,-1,-1,0,0);

    // Front side
    drawVertex(g,i++,1,0,1,-1,1,0,0,1);
    drawVertex(g,i++,1,1,1,1,1,0,0,1);
    drawVertex(g,i++,0,1,-1,1,1,0,0,1);
    drawVertex(g,i++,0,0,-1,-1,1,0,0,1);

    // Right side
    drawVertex(g,i++,1,0,1,-1,-1,1,0,0);
    drawVertex(g,i++,1,1,1,1,-1,1,0,0);
    drawVertex(g,i++,0,1,1,1,1,1,0,0);
    drawVertex(g,i++,0,0,1,-1,1,1,0,0);

    // Top side
    drawVertex(g,i++,1,0,1,1,1,0,1,0);
    drawVertex(g,i++,1,1,1,1,-1,0,1,0);
    drawVertex(g,i++,0,1,-1,1,-1,0,1,0);
    drawVertex(g,i++,0,0,-1,1,1,0,1,0);

    // Back side
    drawVertex(g,i++,1,0,-1,-1,-1,0,0,-1);
    drawVertex(g,i++,1,1,-1,1,-1,0,0,-1);
    drawVertex(g,i++,0,1,1,1,-1,0,0,-1);
    drawVertex(g,i++,0,0,1,-1,-1,0,0,-1);

    // Under side
    drawVertex(g,i++,1,0,1,-1,-1,0,-1,0);
    drawVertex(g,i++,1,1,1,-1,1,0,-1,0);
    drawVertex(g,i++,0,1,-1,-1,1,0,-1,0);
    drawVertex(g,i++,0,0,-1,-1,-1,0,-1,0);

    return g;
}

private Appearance createAppearance()
{
    // We refer Appearance by reference since is
    // required in order for the scene graph
    // compiler to merge Shape3D objects.

    if (appearance==null)
    {
        appearance = new Appearance();

        // Set material properties
        Material material = new Material(
            new Color3f(0.3f,0.3f,0.3f), // Ambient
            new Color3f(0.0f,0.0f,0.0f), // Emissive
            new Color3f(1.0f,1.0f,1.0f), // Diffuse
            new Color3f(0.0f,0.0f,0.0f), // Specular

```

```

        50.0f);
    appearance.setMaterial(material);

    // Turn on flat shading
    appearance.setColoringAttributes(
        new ColoringAttributes(1.0f, 1.0f, 1.0f, ColoringAttributes.SHADE_FLAT));

    // Add texture
    appearance.setTexture(MyCanvas3D.texture);

    TextureAttributes texattrib = new TextureAttributes();

    texattrib.setPerspectiveCorrectionMode(TextureAttributes.NICEST);
    texattrib.setTextureMode(TextureAttributes.MODULATE);

    appearance.setTextureAttributes(texattrib);
}

return appearance;
}
}

```

Edited “Java array” without specialization

```

// generic bubble sort in Java
// array version written Dec 99 by Ulrich Stern

// Edited Feb 2002 by Jacob Marner

import java.util.Date;

interface Comparator {
    boolean compare(Object o1, Object o2);
}

final class Lib {
    static void sort(Object[] a, Comparator cmp) {
        int al = a.length-1;
        for (int i=0; i<al; i++)
            for (int j=al; j>i; j--)
                {
                    Object aj1 = a[j-1];
                    Object aj = a[j];
                    if (! cmp.compare(aj1, aj)) {
                        a[j-1] = aj;
                        a[j] = aj1;
                    }
                }
    }
}

final public class sort {

    static final class Range {
        int lb;
        int ub;
        Range(int l, int u) {
            lb = l;  ub = u;
        }
        public String toString() {
            return "[" + lb + "," + ub + "];"
        }
    }

    // compare lower bound
    static final class LBComparator implements Comparator {
        public boolean compare(Object o1, Object o2) {
            return (((Range)o1).lb < ((Range)o2).lb);
        }
    }
}

```

```

    }
}

// compare upper bound
static final class UBComparator implements Comparator {
    public boolean compare(Object o1, Object o2) {
        return (((Range)o1).ub < ((Range)o2).ub);
    }
}

public static void main(String[] args) {
    System.out.println("sorting...");

    final int N = 30000;
    Range[] a = new Range[N];
    for (int i=0; i<a.length; i++)
        a[i] = new Range(i*7%1000, i*13%1000);

    final LBComparator lb = new LBComparator();
    final UBComparator ub = new UBComparator();

    Date starttime = new Date();

    Lib.sort(a, lb);
    Lib.sort(a, ub);

    float timeelapsed = ((float)(new Date().getTime() - starttime.getTime())) / 1000.0f;
    System.out.println("Elapsed time: " + timeelapsed + " sec.");
}
}

```

Appendix H: Edited third party benchmark

In section 7.11.2 we changed some benchmarks by Ulrich Stern. The modified benchmarks can be found here.

Edited “C++ pointer”

```
// generic bubble sort in C++
// pointer version written Jan 00 by Ulrich Stern

// Edited Feb 02 by Jacob Marner

#include <iostream.h>

#include <time.h>

template<class Object, class Comparator>
void mysort(Object* a[], int n, Comparator cmp) {
    for(int i=0; i<n-1; i++)
        for(int j=n-1; j>i; j--)
            if(!cmp(a[j-1], a[j])) {
                Object* tmp = a[j-1];
                a[j-1] = a[j];
                a[j] = tmp;
            }
}

struct Range {
    int lb, ub;
    Range() {}
    Range(int l, int u): lb(l), ub(u) {}
};

ostream& operator << (ostream& out, const Range* r) {
    return out << '[' << r->lb << ',' << r->ub << ']'<
}

struct LBComparator {
    bool operator() (const Range* a, const Range* b) const {
        return a->lb < b->lb;
    }
};

struct UBComparator {
    bool operator() (const Range* a, const Range* b) const {
        return a->ub < b->ub;
    }
};

int main() {
    const int N = 30000;
    Range* a[N];

    for(int i=0; i<N; i++)
        a[i] = new Range(i*7%1000, i*13%1000);

    cout << "sorting..." << endl;

    clock_t starttime = clock();

    mysort(a, N, LBComparator());
    mysort(a, N, UBComparator());

    float timeelapsed = ((float)(clock() - starttime)) / CLOCKS_PER_SEC;
    cout << "Elapsed time: " << timeelapsed << " sec." << endl;
}
```

```

    for(int j=0; j<N; j++)
        delete a[j];

    return 0;
}

```

Edited “Java array” with specialization

```

// generic bubble sort in Java
// array version written Dec 99 by Ulrich Stern

// Edited Feb 2002 by Jacob Marner

import java.util.Date;

interface Comparator {
    boolean compare(Range o1, Range o2);
}

final class Lib {

    static Range[] a;
    static Comparator cmp;

    static int al;
    static int i;

    static void inner()
    {
        for (int j=al; j>i; j--)
        {
            Range aj1 = a[j-1];
            Range aj = a[j];
            if (! cmp.compare(aj1, aj)) {
                a[j-1] = aj;
                a[j] = aj1;
            }
        }
    }

    static void sort(Range[] a, Comparator cmp) {
        Lib.a = a;
        Lib.cmp = cmp;
        al = a.length-1;
        for (i=0; i<al; i++)
            inner();
    }
}

final class Range {
    int lb;
    int ub;
    Range(int l, int u) {
        lb = l; ub = u;
    }
    public String toString() {
        return "[" + lb + "," + ub + "]";
    }
}

final public class sort {

    // compare lower bound
    static final class LBComparator implements Comparator {
        public boolean compare(Range o1, Range o2) {
            return o1.lb < o2.lb;
        }
    }
}

```

```

}

// compare upper bound
static final class UBComparator implements Comparator {
    public boolean compare(Range o1, Range o2) {
        return o1.ub < o2.ub;
    }
}

public static void main(String[] args) {
    System.out.println("sorting...");

    final int N = 30000;
    Range[] a = new Range[N];
    for (int i=0; i<a.length; i++)
        a[i] = new Range(i*7%1000, i*13%1000);

    final LBComparator lb = new LBComparator();
    final UBComparator ub = new UBComparator();

    Date starttime = new Date();

    Lib.sort(a, lb);
    Lib.sort(a, ub);

    float timeelapsed = ((float)(new Date().getTime() - starttime.getTime())) / 1000.0f;
    System.out.println("Elapsed time: " + timeelapsed + " sec.");
}
}

```

Appendix I: Benchmark results

This section contains the results of the benchmarks in tabular form. This data was used to generate the graphs used in chapter 7. The original Excel spreadsheet can be downloaded at the home page of this report.

Untweaked Particles

Warm-up:

		300 Test is done on Java 1.4 server java -server																				
Particles	Steps	0	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
	1	0	8.8	16	24	30	39	45	53	60	67	74	82	91	93	102	111	119	118	136	141	147
	2	0	9	15	23	31	42	46	53	61	68	77	80	88	102	98	117	117	123	135	142	143
	3	0	8.7	16	23	37	38	45	50	58	67	74	81	91	98	104	107	116	128	13	141	150
	4	0	8.8	16	22	33	37	45	53	61	68	72	84	89	98	100	109	118	125	135	141	144
	5	0	8.9	17	23	31	37	46	52	60	61	74	80	90	97	103	106	120	126	135	137	149
	Median	0	8.8	16	23	31	38	45	53	60	67	74	81	90	98	102	109	118	125	135	141	147
	First order derivation (x100)		8.8	7.2	7	8	7	7	8	7	7	7	7	9	8	4	7	9	7	10	6	6

Results:

Run using 100 time steps of h=0.001. Results are given in seconds with 2 significant digits to the nearest number of seconds.

		MS C++														
Num Particles		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
1	0.23	0.89	2.1	3.6	5.7	8.2	11	14	18	22	28	32	39	44	52	
2	0.22	0.89	2.2	3.7	5.6	8.1	11	14	18	22	28	33	38	45	51	
3	0.22	0.93	2.3	3.6	5.6	8	11	14	18	23	27	32	38	44	51	
4	0.23	0.9	2	3.6	5.6	8	11	14	18	22	6	32	39	50	51	
5	0.22	0.9	2.1	3.6	5.7	8.2	11	14	19	23	27	32	38	45	51	
Median	0.22	0.9	2.1	3.6	5.6	8.1	11	14	18	22	27	32	38	45	51	
First order derivation	0.0022	0.0068	0.012	0.015	0.02	0.025	0.029	0.03	0.04	0.04	0.04	0.05	0.05	0.06	0.07	0.08

		Intel C++														
Num Particles		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
1	0.26	1.1	2.4	4.4	6.6	9.4	13	17	21	27	32	38	45	52	60	
2	0.31	1.1	2.4	4.3	6.6	9.6	13	17	21	24	32	38	45	52	60	
3	0.26	1	2.4	3.2	6.6	9.4	13	17	21	26	32	38	44	53	60	
4	0.26	1.1	2.5	4.2	6.6	9.4	13	17	21	26	32	38	45	52	60	
5	0.26	1.1	2.4	4.1	6.6	9.6	13	17	22	26	32	38	45	52	63	
Median	0.26	1.1	2.4	4.2	6.6	9.4	13	17	21	26	32	38	45	52	60	
First order derivation	0.0026	0.0084	0.013	0.018	0.024	0.028	0.036	0.04	0.04	0.05	0.06	0.06	0.07	0.07	0.08	

		JDK 1.4 client										java	
Num Particles		100	200	300	400	500	600	700	800	900	1000	1100	1200
1	0.86	3.2	7.2	13	20	28	41	55	70	90	103	121	
2	0.85	3.2	7.1	13	20	29	40	53	71	87	105	121	
3	0.85	3.2	7.1	13	20	29	40	54	71	89	103	123	
4	0.86	3.3	7.1	12	20	29	41	55	70	85	103	123	
5	0.86	3.3	7.2	13	20	29	41	54	71	88	105	123	
Median	0.86	3.2	7.1	13	20	29	41	54	71	88	103	123	
First order derivation	0.0086	0.0234	0.039	0.059	0.07	0.09	0.12	0.13	0.17	0.17	0.15	0.2	

		JDK 1.4 server										java -server	
Num Particles		100	200	300	400	500	600	700	800	900	1000	1100	1200
1	2	4	8.8	15	22	31	43	57	73	90	105	129	
2	2.1	4.4	8.9	14	22	31	43	56	73	90	106	129	
3	1.8	4.6	8.9	14	22	31	44	57	73	91	106	129	
4	2.2	4.6	8.9	14	22	31	42	57	73	90	106	126	
5	2.2	4.1	8.8	14	22	31	44	57	71	90	106	129	
Median	2.1	4.4	8.9	14	22	31	43	57	73	90	106	129	
First order derivation	0.021	0.023	0.045	0.051	0.08	0.09	0.12	0.14	0.16	0.17	0.16	0.23	

		IBM										java	
Num Particles		100	200	300	400	500	600	700	800	900			
1	1.7	6	14	25	40	56	78	105	134				
2	1.7	6.1	14	25	39	57	79	104	135				
3	1.7	6.1	14	25	39	58	79	104	135				
4	1.8	6.3	14	25	39	58	79	104	133				
5	1.7	6.1	14	26	39	57	78	104	134				
Median	1.7	6.1	14	25	39	57	79	104	134				
First order derivation	0.017	0.044	0.079	0.11	0.14	0.18	0.22	0.25	0.3				

		Jet									
Num Particles		100	200	300	400	500	600	700	800	900	1000
1	1.3	5.2	12	21	32	47	66	86	111	135	
2	1.3	5.2	12	21	33	48	66	86	112	135	
3	1.3	5.1	13	21	32	48	65	86	113	138	
4	1.3	5.3	12	21	33	47	66	87	111	137	
5	1.3	5.1	12	21	33	48	64	88	109	135	
Median	1.3	5.2	12	21	33	48	66	86	111	135	
First order derivation	0.013	0.039	0.068	0.09	0.12	0.15	0.18	0.2	0.25	0.24	

Tweaked Particles

Warm-up:

Particles	400 Test is done on Java 1.4 server										java -server										
Steps	0	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000
1	0	7.4	14	20	27	33	40	48	52	58	60	70	78	84	93	90	104	111	118	123	128
2	0	7.3	15	20	28	33	41	47	52	59	65	72	79	85	92	99	103	112	116	125	131
3	0	7.3	14	21	27	34	40	46	54	54	66	71	72	86	93	99	104	111	118	123	132
4	0	7.2	14	21	28	33	40	46	52	58	65	71	80	86	92	97	108	110	117	124	131
5	0	7.3	14	21	27	33	39	46	54	59	65	73	78	86	93	91	110	111	118	124	130
Median	0	7.3	14	21	27	33	40	46	52	58	65	71	78	86	93	97	104	111	118	124	131
First order derivation (x100)		7.3	6.7	7	6	6	7	6	6	6	7	6	7	8	7	4	7	7	7	6	7

Results:

Run using 100 time steps of h=0.001. Results are given in seconds with 2 significant digits to the nearest number of seconds.

MS C++																	
Num Particles	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700
1	0.2	0.77	1.8	3.1	4.9	7.1	9.6	12	16	19	24	28	33	38	45	51	64
2	0.2	0.77	1.8	3	4.9	7	9.5	12	16	19	2	28	36	38	44	51	57
3	0.19	0.78	1.7	3	4.9	7	9.6	12	17	20	23	28	35	39	50	50	64
4	0.19	0.79	1.7	3.1	4.9	6.9	9.7	12	16	19	23	28	25	38	44	51	57
5	0.19	0.79	1.8	3.3	4.8	7.1	9.5	12	16	19	24	28	19	38	47	51	58
Median	0.19	0.78	1.8	3.1	4.9	7	9.6	12	16	19	23	28	33	38	45	51	58
First order derivation	0.0019	0.0059	0.0102	0.013	0.018	0.021	0.026	0.024	0.04	0.03	0.04	0.05	0.05	0.05	0.07	0.06	0.07

Intel C++																	
Num Particles	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700
1	0.24	0.99	2.3	4	6.3	9	12	16	21	26	34	40	43	50	59	67	74
2	0.25	0.99	2.3	4	6.2	9.1	12	16	21	29	36	38	43	49	58	25	74
3	0.26	1	2.3	4	6.3	9.1	12	17	21	27	35	36	42	50	59	65	74
4	0.26	0.99	2.3	4.1	6.2	9.2	12	16	21	28	34	36	42	50	58	66	75
5	0.25	1	2.2	4	6.2	9	12	17	21	26	35	40	42	50	57	66	74
Median	0.25	0.99	2.3	4	6.2	9.1	12	16	21	27	35	38	42	50	58	66	74
First order derivation	0.0025	0.0074	0.0131	0.017	0.022	0.029	0.029	0.04	0.05	0.06	0.08	0.03	0.04	0.08	0.08	0.08	0.08

JDK 1.4 client																	
Num Particles	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500		
1	0.39	1.4	3	5.1	8.2	12	16	21	27	33	40	49	58	67	78		
2	0.41	1.4	3	5.2	8.3	12	16	21	27	33	40	53	59	68	78		
3	0.4	1.4	3	5.3	8.2	12	16	21	27	33	40	50	58	67	79		
4	0.4	1.4	3	5.2	8.1	12	16	21	27	33	40	52	60	67	78		
5	0.4	1.4	3	5.2	8.2	12	16	21	26	33	40	55	58	68	79		
Median	0.4	1.4	3	5.2	8.2	12	16	21	27	33	40	52	58	67	78		
First order derivation	0.004	0.01	0.016	0.022	0.03	0.038	0.04	0.05	0.06	0.06	0.07	0.12	0.06	0.09	0.11		

JDK 1.4 server																	
Num Particles	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500		
1	1.4	2.7	4.4	7.3	11	15	21	27	34	42	52	62	73	86	98		
2	1	2.5	4.4	7.3	11	15	21	27	34	43	51	62	73	85	102		
3	1.1	2.6	4.3	7.2	11	15	21	27	34	42	53	61	73	85	102		
4	1.1	2.7	4.4	7.5	11	16	21	27	34	42	51	61	73	85	101		
5	1.5	2.7	4.4	7.3	11	15	21	27	34	42	51	62	73	84	102		
Median	1.1	2.7	4.4	7.3	11	15	21	27	34	42	51	62	73	85	102		
First order derivation	0.011	0.016	0.017	0.029	0.037	0.04	0.06	0.06	0.07	0.08	0.09	0.11	0.11	0.12	0.17		

IBM																	
Num Particles	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700
1	0.41	1.2	2.6	4.7	7.1	10	14	19	24	30	36	44	52	61	70	80	90
2	0.4	1.1	2.6	4.8	7	10	14	18	24	30	36	43	52	61	70	80	91
3	0.4	1.2	2.6	4.7	7.3	10	14	18	24	30	36	44	53	60	70	80	91
4	0.45	1.2	2.5	4.7	7	10	14	18	24	30	36	44	52	61	69	83	95
5	0.41	1.2	2.6	4.5	7	10	14	18	24	30	36	44	52	61	70	80	90
Median	0.41	1.2	2.6	4.7	7	10	14	18	24	30	36	44	52	61	70	80	91
First order derivation	0.0041	0.0079	0.014	0.021	0.023	0.03	0.04	0.04	0.06	0.06	0.06	0.08	0.08	0.09	0.09	0.1	0.11

Jet																	
Num Particles	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700
1	0.26	1.1	2.3	4.4	6.8	10	13	18	22	29	34	41	48	56	66	74	83
2	0.26	1.1	2.4	4.3	6.7	9.8	13	18	26	30	34	41	48	56	64	74	83
3	0.27	1	2.4	4.4	6.7	9.6	13	18	23	28	34	41	49	57	65	74	84
4	0.29	1.1	2.4	4.2	6.8	9.7	13	18	23	28	34	41	48	56	65	74	83
5	0.27	1	2.4	4.3	6.8	9.6	13	18	26	32	34	41	48	56	65	74	83
Median	0.27	1.1	2.4	4.3	6.8	9.7	13	18	23	29	34	41	48	56	65	74	83
First order derivation	0.0027	0.0083	0.013	0.019	0.025	0.029	0.033	0.05	0.05	0.06	0.05	0.07	0.07	0.08	0.09	0.09	0.09

Untweaked Life

Warm-up:

World size	200 Test is done on Java 1.4 server												
Steps	0	500	1000	1500	2000	2500	3000	3500	4000	4500	5000		
1	0	33	55	68	86	99	114	137	137	151	188		
2	0	34	54.6	74	88	103	103	112	153	166	157		
3	0	35	60	71	84	101	113	134	144	165	174		
4	0	32	54.1	70	84	106	111	124	134	173	176		
5	0	35	57	74	86	102	121	115	137	162	178		
Median	0	34	55	71	86	102	113	124	137	165	176		
First order derivation (x500)		34	21	16	15	16	11	11	13	28	11		

Results:

World size	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
1	0.13	0.59	1.4	2.7	4.3	6.4	8.9	12	15	20	23	28	34	39	46
2	0.12	0.58	1.4	2.7	4.2	6.3	9	12	15	20	23	28	34	38	46
3	0.12	0.59	1.4	2.7	4.3	6.3	8.7	12	15	20	23	28	34	39	46
4	0.13	0.58	1.4	2.7	4.3	6.6	8.7	12	15	20	23	28	34	38	45
5	0.13	0.58	1.4	2.7	4.2	6.3	8.7	12	15	20	23	28	34	38	46
Median	0.13	0.58	1.4	2.7	4.3	6.3	8.7	12	15	20	23	28	34	38	46
First order derivation	0.0013	0.0045	0.0082	0.013	0.016	0.02	0.024	0.033	0.03	0.05	0.03	0.05	0.06	0.04	0.08
Intel C++															
World size	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
1	0.1	0.49	1.2	2.3	3.6	5.4	7.7	10	13	17	20	24	30	34	40
2	0.09	0.49	1.2	2.3	3.6	5.4	7.7	10	13	17	20	24	30	34	44
3	0.1	0.5	1.2	2.3	3.7	5.4	7.7	10	13	17	20	24	30	34	41
4	0.11	0.49	1.2	2.2	3.7	5.5	7.7	10	13	17	20	24	30	36	40
5	0.1	0.49	1.2	2.3	3.6	5.6	7.6	10	13	17	20	25	30	34	40
Median	0.1	0.49	1.2	2.3	3.6	5.4	7.7	10	13	17	20	24	30	34	40
First order derivation	0.001	0.0039	0.0071	0.011	0.013	0.018	0.023	0.023	0.03	0.04	0.03	0.04	0.06	0.04	0.06
JDK 1.4 client															
Worldsize	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
1	0.53	2.7	6.3	12	18	25	35	46	58	71	87	106	121	141	165
2	0.53	2.6	6.3	12	18	25	35	46	58	71	87	106	121	141	165
3	0.52	2.7	6.4	12	18	25	35	46	58	71	87	106	121	141	165
4	0.51	2.6	6.3	12	18	26	35	46	59	71	87	106	120	141	165
5	0.52	2.6	6.2	12	18	25	35	45	58	71	87	107	122	141	165
Median	0.52	2.6	6.3	12	18	25	35	46	58	71	87	106	121	141	165
First order derivation	0.0052	0.0208	0.037	0.057	0.06	0.07	0.1	0.11	0.12	0.13	0.16	0.19	0.15	0.2	0.24
JDK 1.4 server															
World size	100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500
1	0.6	2.3	6.1	11	17	24	28	42	53	59	79	88	98	118	134
2	0.61	2.6	6.2	11	17	23	29	41	49	65	75	90	116	137	140
3	0.59	2.6	6.2	11	17	21	29	38	53	68	72	98	103	119	140
4	0.52	2.7	5.5	11	17	23	34	41	49	57	73	87	101	118	134
5	0.59	2.6	5.4	11	17	21	29	41	47	65	74	88	102	116	151
Median	0.59	2.6	6.1	11	17	23	29	41	49	65	74	88	102	118	140
First order derivation	0.0059	0.0201	0.035	0.049	0.06	0.06	0.12	0.08	0.16	0.09	0.14	0.14	0.14	0.16	0.22
IBM															
World size	100	200	300	400	500	600	700	800	900	1000	1100	1200			
1	0.93	4.2	9.5	19	28	43	59	76	98	123	153	180			
2	0.92	4.1	9.5	17	28	40	59	76	99	124	150	183			
3	0.95	4.3	9.7	17	28	43	59	76	97	123	151	190			
4	0.97	4.3	9.4	19	28	40	58	76	97	124	151	186			
5	0.93	4.2	9.4	19	28	43	59	76	100	121	150	188			
Median	0.93	4.2	9.5	19	28	43	59	76	98	123	151	186			
First order derivation	0.0093	0.0327	0.053	0.095	0.09	0.15	0.16	0.17	0.22	0.25	0.28	0.35			
Jet															
World size	100	200	300	400	500	600	700	800	900	1000	1100				
1	0.85	3.4	8	14	24	43	59	85	124	157	202				
2	0.83	3.4	8	14	24	43	59	86	124	149	202				
3	0.84	3.4	8	14	24	43	60	86	125	150	204				
4	0.86	3.5	8	14	24	43	61	86	124	154	204				
5	0.88	3.4	8	14	24	46	57	85	124	154	207				
Median	0.85	3.4	8	14	24	43	59	86	124	154	204				
First order derivation	0.0085	0.0255	0.046	0.06	0.1	0.19	0.16	0.27	0.38	0.3	0.5				

Tweaked Life

Warm-up:

World size	300 Test is done on Java 1.4 server					-server -Xmx200MB						
Steps	0	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	
1	0	14	28	41	55	71	82	97	110	126	138	
2	0	14	28	30	55	67	81	98	111	125	99	
3	0	14	28	40	55	70	84	96	109	126	140	
4	0	14	29	37	54	51	84	98	110	124	138	
5	0	14	28	41	55	69	83	97	108	124	139	
Median	0	14	28	40	55	69	83	97	110	125	138	
First order derivation (x500)		14	14	12	15	14	14	14	14	13	13	

Results:

Run using 10 steps. Results are given in seconds with 2 significant digits to the nearest number of seconds.

MS C++		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000	2100
World size	1	0.13	0.57	1.3	2.5	4	5.8	7.9	10	13	16	20	24	28	33	38	43	49				
	2	0.13	0.57	1.3	2.5	4	5.7	7.9	10	13	16	20	24	28	33	38	43	49				
	3	0.13	0.57	1.4	2.5	4	5.8	7.9	10	13	16	20	24	28	33	38	43	49				
	4	0.13	0.57	1.3	2.5	4	5.7	7.9	10	13	16	20	24	28	33	38	43	49				
	5	0.13	0.57	1.4	2.5	4	5.7	7.9	10	13	16	20	24	28	33	38	43	49				
Median		0.13	0.57	1.3	2.5	4	5.7	7.9	10	13	16	20	24	28	33	38	43	49				
First order derivation		0.0013	0.0044	0.0073	0.012	0.015	0.017	0.022	0.021	0.03	0.03	0.04	0.04	0.04	0.05	0.05	0.05	0.06				
Intel C++		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700				
World size	1	0.09	0.44	1.1	1.9	3.1	4.6	6.2	8.2	11	13	16	19	22	26	30	35	39				
	2	0.09	0.44	1	2	3.1	4.5	6.2	8.2	10	13	16	19	23	26	30	35	39				
	3	0.1	0.44	1.1	1.9	3.1	4.5	6.2	8.2	11	13	16	19	23	26	30	36	39				
	4	0.09	0.43	1	2	3.1	4.5	6.3	8.2	11	13	16	19	23	26	30	35	39				
	5	0.1	0.44	1	1.9	3.1	4.5	6.2	8.2	11	13	16	19	23	26	30	35	39				
Median		0.09	0.44	1	1.9	3.1	4.5	6.2	8.2	11	13	16	19	23	26	30	35	39				
First order derivation		0.0009	0.0035	0.0056	0.009	0.012	0.014	0.017	0.02	0.028	0.02	0.03	0.03	0.04	0.03	0.04	0.05	0.04				
JDK 1.4 client		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000	
Worldsize	1	0.1	0.39	1	1.5	2.6	3.3	4.9	4.7	7.5	8.9	11	13	11	20	16	25	28	32	36	39	
	2	0.1	0.33	1	1.5	2.5	2.6	4.9	5.4	7.4	9	11	13	15	20	22	26	28	32	35	39	
	3	0.1	0.34	1	1.5	2.6	2.3	4.9	6.6	7.4	8.9	11	10	15	19	23	25	28	22	19	39	
	4	0.1	0.4	1	1	2.5	3.3	4.9	5.5	7.5	8.9	11	13	15	19	18	15	28	26	36	34	
	5	0.08	0.4	0.83	1.5	2.1	3.3	3.7	6.9	7.6	9	7.6	6.2	11	19	22	26	19	32	36	38	
Median		0.1	0.39	1	1.5	2.5	3.3	4.9	5.5	7.5	8.9	11	13	15	19	22	25	28	32	36	39	
First order derivation		0.001	0.0029	0.0061	0.005	0.01	0.008	0.016	0.006	0.02	0.014	0.021	0.02	0.02	0.04	0.03	0.03	0.03	0.04	0.04	0.03	0.03
JDK 1.4 server		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000	2100
World size	1	0.15	0.33	0.79	0.97	1.6	2.1	2.9	11	5	6	7	8	9	10	12	11	16	19	21	22	24
	2	0.14	0.42	0.73	0.89	1.6	7.2	2.8	4	5	6	7	8	9	11	12	14	16	19	21	22	23
	3	0.14	0.35	0.75	1	5.2	7	2.5	4	5	6	8	8	9	11	12	14	16	20	21	22	24
	4	0.15	0.34	0.73	2.7	1.6	2.1	2.8	3	3	5	7	27	9	11	12	14	16	20	13	26	24
	5	0.15	0.34	0.72	1	1.6	2.1	2.8	4	14	6	7	8	9	11	12	14	16	19	21	21	24
Median		0.15	0.34	0.73	1	1.6	2.1	2.8	4	5	6	7	8	9	11	12	14	16	19	21	22	24
First order derivation		0.0015	0.0019	0.0039	0.0027	0.006	0.005	0.007	0.012	0.01	0.01	0.01	0.01	0.01	0.02	0.01	0.02	0.02	0.03	0.02	0.01	0.02
IBM		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800	1900	2000	
World size	1	0.1	0.34	0.66	1.1	1.8	2.1	4.9	5.2	6.2	7.8	9	10	12	14	16	19	22	25	22	32	
	2	0.1	0.36	0.63	1.2	2	1.7	4.9	5.3	6	7.7	9	10	8.8	14	16	18	21	25	28	67	
	3	0.09	0.35	0.59	1.3	1.8	2.7	4.9	5.2	6.3	7.6	9	10	6.7	13	17	18	21	27	29	32	
	4	0.1	0.32	0.56	1	2	2.7	4.9	5.2	4.8	7.8	9	10	12	14	16	19	22	15	28	33	
	5	0.1	0.35	0.67	1.3	1.7	2.3	4.9	5.3	6.2	7.8	9	10	12	14	16	19	21	25	28	33	
Median		0.1	0.35	0.63	1.2	1.8	2.3	4.9	5.2	6.2	7.8	9	10	12	14	16	19	21	25	28	33	
First order derivation		0.001	0.0025	0.0028	0.0057	0.006	0.005	0.026	0.003	0.01	0.016	0.012	0.01	0.02	0.02	0.02	0.03	0.02	0.04	0.03	0.03	0.05
Jet		100	200	300	400	500	600	700	800	900	1000	1100	1200	1300	1400	1500	1600	1700	1800			
World size	1	0.1	0.45	1	1.8	2.2	4.2	5.8	7.6	9.7	12	11	17	20	23	31	36	40				
	2	0.1	0.18	1	1.9	3	4.2	5.7	7.6	9.6	12	14	17	20	10	28	29	36	32			
	3	0.1	0.45	1.1	1.9	2.9	4.3	4.5	7.5	9.7	12	16	18	20	24	27	32	35	40			
	4	0.11	0.44	1.1	1.8	2.9	4.3	5.8	4.6	9.6	12	14	17	20	24	27	31	36	39			
	5	0.11	0.45	1	0.8	1.2	4.3	5.8	7.6	9.7	12	14	12	8.5	24	27	18	15	25			
Median		0.1	0.45	1	1.8	2.9	4.3	5.8	7.6	9.7	12	14	17	20	24	27	31	36	39			
First order derivation		0.001	0.0035	0.0055	0.008	0.011	0.014	0.015	0.018	0.021	0.023	0.02	0.03	0.03	0.04	0.03	0.04	0.05	0.03			

Untweaked 3D Demo

Warm-up:

Boxes	1000	Test is done on Java 1.4 server										java -server									
Frames	0	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000
1	0	2.7	4.8	7.2	9.6	11	14	16	18	20	23	25	30	29	33	34	35	38	40	42	47
2	0	2.8	4.7	7.1	8.9	12	14	16	19	21	22	24	28	29	31	33	37	37	40	42	43
3	0	2.5	4.9	7.3	9.2	11	14	16	18	21	22	24	27	31	33	33	35	37	41	41	44
4	0	2.7	4.8	7.2	9.4	12	14	16	18	21	23	24	27	32	32	33	37	37	41	42	43
5	0	2.6	5	6.9	9.6	12	14	16	18	20	22	24	29	31	32	34	37	40	42	44	44
Median	0	2.7	4.8	7.2	9.4	12	14	16	18	21	22	24	28	31	32	33	37	37	41	42	44
First order derivation (x500)		2.7	2.1	2.4	2.2	2.6	2	2	2	3	1	2	4	3	1	1	4	0	4	1	2

Results:

		MS C++										
Num Boxes		32	64	128	256	512	1024	2048	4096	8192	16384	32768
	1	14	16	17	17	19	22	29	38	59	94	162
	2	14	16	16	17	19	23	28	39	60	94	159
	3	16	16	16	17	19	21	30	39	58	94	162
	4	16	16	16	17	19	22	28	39	57	99	164
	5	15	16	16	17	20	23	27	38	59	94	160
Median		15	16	16	17	19	22	28	39	59	94	162
First order derivation		0.15	0.01	0	0.01	0.02	0.03	0.06	0.11	0.2	0.35	0.68

		Intel C++										
Num Boxes		32	64	128	256	512	1024	2048	4096	8192	16384	32768
	1	15	16	16	17	19	22	27	38	60	95	163
	2	16	16	16	18	19	22	28	38	58	96	163
	3	16	16	16	17	19	22	28	38	56	94	162
	4	15	16	16	17	19	22	27	38	59	93	163
	5	15	16	16	17	19	22	28	38	58	94	165
Median		15	16	16	17	19	22	28	38	58	94	163
First order derivation		0.15	0.01	0	0.01	0.02	0.03	0.06	0.1	0.2	0.36	0.69

GL4Java

		JDK 1.4 client			java							
Num Boxes		32	64	128	256	512	1024	2048	4096	8192	16384	32768
	1	16	16	16	18	19	23	30	39	61	100	173
	2	15	16	16	17	19	22	28	40	62	98	174
	3	11	16	16	17	19	22	28	39	60	99	175
	4	12	16	17	18	20	22	28	38	62	99	172
	5	16	16	17	18	19	22	29	39	61	100	175
Median		15	16	16	18	19	22	28	39	61	99	174
First order derivation		0.15	0.16	0.16	0.18	0.19	0.22	0.28	0.39	0.61	0.99	1.74

		JDK 1.4 server			java -server							
Num Boxes		32	64	128	256	512	1024	2048	4096	8192	16384	32768
	1	16	16	17	18	19	23	30	39	61	99	170
	2	16	16	17	18	19	24	30	37	58	99	171
	3	16	16	17	17	19	23	29	39	58	99	171
	4	16	16	17	17	19	22	30	40	59	99	173
	5	16	16	17	18	20	22	30	39	60	98	169
Median		16	16	17	18	19	23	30	39	59	99	171
First order derivation		0.16	0	0.01	0.01	0.01	0.04	0.07	0.09	0.2	0.4	0.72

		IBM			java							
Num Boxes		32	64	128	256	512	1024	2048	4096	8192	16384	32768
	1	16	16	17	18	20	24	30	42	65	112	193
	2	16	16	17	18	20	23	29	40	65	112	194
	3	16	16	17	18	20	24	30	41	64	114	194
	4	16	16	17	18	20	24	29	42	65	113	195
	5	16	16	17	18	20	23	30	42	65	112	194
Median		16	16	17	18	20	24	30	42	65	112	194
First order derivation		0.16	0	0.01	0.01	0.02	0.04	0.06	0.12	0.23	0.47	0.82

Tweaked 3D Demo

Warm-up:

		1000 Test is done on Java 1.4 server java -server																				
Boxes	Frames	0	200	400	600	800	1000	1200	1400	1600	1800	2000	2200	2400	2600	2800	3000	3200	3400	3600	3800	4000
	1	0	8.3	13	19	18	19	21	25	24	27	31	32	32	35	37	35	38	41	39	42	44
	2	0	7.3	12	16	12	18	23	22	28	26	31	33	31	32	35	38	37	40	42	44	42
	3	0	7.7	12	18	18	20	20	23	25	24	31	29	33	34	36	36	37	42	40	41	46
	4	0	4.9	13	16	17	18	26	23	25	30	32	31	35	37	36	37	40	40	41	42	44
	5	0	4.5	7.1	15	20	19	22	24	26	30	30	33	34	35	35	35	36	40	40	41	42
Median		0	7.3	12	16	18	19	22	23	25	27	31	32	33	35	36	36	37	40	40	42	44
First order derivation (x100)			7.3	4.7	4	2	1	3	1	2	2	4	1	1	2	1	0	1	3	0	2	2

Results:

Run using 5000 time steps (i.e. frames of animation) Results are given in seconds with 2 significant digits to the nearest number of seconds.

MS C++											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1	15	15	16	16	17	19	24	32	48	77	136
2	15	15	15	16	17	20	23	32	46	74	136
3	15	15	15	16	17	20	22	32	46	75	133
4	15	15	15	16	17	19	24	32	49	79	136
5	15	15	15	16	17	19	23	33	47	78	135
Median	15	15	15	16	17	19	23	32	47	77	136
First order derivation	0.15	0	0	0.01	0.01	0.02	0.04	0.09	0.15	0.3	0.59

Intel C++											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1	15	15	15	9	17	19	24	32	46	78	142
2	15	15	15	15	17	20	24	30	47	80	141
3	15	15	15	16	17	20	24	32	46	78	141
4	15	15	15	16	17	20	22	34	47	79	143
5	15	15	12	16	17	19	23	33	45	80	140
Median	15	15	15	16	17	20	24	32	46	79	141
First order derivation	0.15	0	0	0.01	0.01	0.03	0.04	0.08	0.14	0.33	0.62

GL4Java

JDK 1.4 client GL4Java											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1	15	15	16	16	17	19	25	32	49	88	166
2	15	15	16	16	17	20	22	32	47	88	162
3	15	15	16	16	17	20	24	33	49	87	165
4	15	16	16	16	17	19	23	32	50	87	165
5	14	15	16	16	17	20	23	30	48	87	166
Median	15	15	16	16	17	20	23	32	49	87	165
First order derivation	0.15	0	0.01	0	0.01	0.03	0.03	0.09	0.17	0.38	0.78

JDK 1.4 server GL4Java											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1	15	16	16	16	18	20	24	33	47	85	160
2	16	16	16	16	18	22	22	32	48	88	163
3	15	16	16	16	17	23	30	32	49	84	158
4	16	16	18	16	21	20	30	37	54	83	157
5	14	16	16	16	21	21	25	33	50	82	164
Median	15	16	16	16	18	21	25	33	49	84	160
First order derivation	0.15	0.01	0	0	0.02	0.03	0.04	0.08	0.16	0.35	0.76

IBM GL4Java											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1	15	16	16	17	18	20	23	33	50	95	174
2	15	15	15	16	17	19	24	31	50	93	175
3	15	16	16	16	18	19	24	32	50	93	176
4	15	16	16	16	17	20	24	33	51	94	178
5	15	15	16	16	18	20	25	32	49	93	178
Median	15	16	16	16	18	20	24	32	50	93	176
First order derivation	0.15	0.01	0	0	0.02	0.02	0.04	0.08	0.18	0.43	0.83

Jet											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1											
2											
3											
4											
5											
Median	#NUM!										
First order derivation	#NUM!										

I was unable to get Jet to work with gl4java for unknown reasons

Java3D

JDK 1.4 client Java3D											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	
1	19	19	19	20	23	32	51	79	138	260	
2	19	19	20	20	23	31	49	79	135	257	
3	19	19	19	20	22	32	49	78	134	260	
4	19	19	20	21	23	31	47	77	134	258	
5	19	19	20	20	23	31	48	79	134	257	
Median	19	19	20	20	23	31	49	79	134	258	
First order derivation	0.19	0	0.01	0	0.03	0.08	0.18	0.3	0.55	1.24	

JDK 1.4 server Java3D											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	
1	32	38	36	39	42	52	69	99	161	272	
2	35	40	35	36	41	51	72	100	164	271	
3	35	36	35	37	42	51	70	105	160	273	
4	32	36	36	39	42	50	69	98	160	268	
5	33	34	34	39	44	50	69	102	164	272	
Median	33	36	35	39	42	51	69	100	161	272	
First order derivation	0.33	0.03	-0.01	0.04	0.03	0.09	0.18	0.31	0.61	1.11	

IBM Java3D											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	
1	23	21	24	25	29	39	56	91	145	165	
2	24	21	24	25	30	39	55	91	147	173	
3	29	24	24	26	27	38	57	91	151	266	
4	23	24	24	26	28	38	56	89	145	272	
5	23	24	24	25	28	38	57	93	150	282	
Median	23	24	24	25	28	38	56	91	147	272	
First order derivation	0.23	0.01	0	0.01	0.03	0.1	0.18	0.35	0.56	1.25	

Jet											
Num Boxes	32	64	128	256	512	1024	2048	4096	8192	16384	32768
1											
2											
3											
4											
5											
Median	#NUM!										
First order derivation	#NUM!										

I was unable to get Jet 2.1 to work with Java3D 1.3 because uses the JDK 1.4 NIO library. Jet 2.1 only supports Ja