Lego Emulation in a Virtual 3D Environment

Richard Carp

Computer Science BSc(Hons)
(University of Bath)

May 8, 2006
LEGO Emulation in a Virtual 3D Environment

submitted by ...................................................

COPYRIGHT

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed: ..........................................................
Abstract

The use of LEGO as an entertainment device has spanned across many formats over the years. The advantage of virtual LEGO manipulations over physical ones is the lack of ‘real life’ constraints imposed upon virtual environments. There is a clear focus on simulation within 3D environments, involving expensive gadgets that many desktop PC owners do not want to or cannot use and afford. Therefore there is a clear need for enjoyable, LEGO manipulation based on a 2D environment using mouse and keyboard. The use of OpenGL has produced a usable, testable system that was shown to create a suitable environment for 2D LEGO manipulation, where movement and placing of bricks was simple to execute and therefore models could be created within a 3D format. Although the system in its current state is simple, this can be easily rectified through expanding knowledge of OpenGL, creating an improved system that extends beyond the simple manipulations achieved here.
Acknowledgements

I would like to deeply thank my project supervisor, Dr. Claire Willis, whose help, advice and patience have made this project possible. Many thanks also go to the group of test users who put up with my many requests.
# Contents

1 Introduction 2
  1.1 Introduction to the Problem .................................. 2
  1.2 Key Objectives .............................................. 3

2 Literature Survey 4
  2.1 Introduction ................................................. 4
  2.2 Designing a Usable Interface ............................... 4
  2.3 Display of 3D Environment Techniques .................... 6
    2.3.1 Visual Cues ........................................... 6
  2.4 Manipulation of Environment ............................... 8
    2.4.1 Selection of objects .................................. 8
  2.5 Choice of Language to Use ............................... 8
    2.5.1 Introduction .......................................... 8
    2.5.2 Introduction to C++ ................................... 9
    2.5.3 Introduction to Java .................................. 9
    2.5.4 C++ vs. Java ........................................ 9
  2.6 Similar Software .......................................... 10
    2.6.1 LEGO Digital Designer ............................... 10
    2.6.2 LDraw ................................................ 11
    2.6.3 LeoCAD ............................................. 11
  2.7 Conclusion ................................................ 12

3 Requirements 13
  3.1 Introduction ............................................... 13
    3.1.1 Requirements Gathering ............................. 13
    3.1.2 Gathering Techniques ................................ 13
    3.1.3 Requirement Specification ......................... 15
  3.2 Specific Requirements .................................... 15
  3.3 Conclusion ................................................. 16

4 Design 18
  4.1 Co-ordinate Systems ....................................... 18
    4.1.1 Transformations ....................................... 20
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.2</td>
<td>Design of LEGO bricks</td>
<td>20</td>
</tr>
<tr>
<td>4.3</td>
<td>Structure of Environment</td>
<td>21</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Object Selection</td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>Lighting</td>
<td>23</td>
</tr>
<tr>
<td>4.5</td>
<td>Viewports</td>
<td>23</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Introduction to Viewports</td>
<td>23</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Design Viewports</td>
<td>24</td>
</tr>
<tr>
<td>4.5.3</td>
<td>Viewing Viewports</td>
<td>25</td>
</tr>
<tr>
<td>4.5.4</td>
<td>Web of Influence</td>
<td>25</td>
</tr>
<tr>
<td>4.6</td>
<td>Manipulation of the Environment</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>Implementation</td>
<td>27</td>
</tr>
<tr>
<td>5.1</td>
<td>Choice of Language</td>
<td>27</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Implementation of brick</td>
<td>27</td>
</tr>
<tr>
<td>5.2</td>
<td>Storing Bricks in Our World</td>
<td>28</td>
</tr>
<tr>
<td>5.3</td>
<td>Selection of Bricks</td>
<td>29</td>
</tr>
<tr>
<td>5.4</td>
<td>Moving Bricks</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>Testing</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>Introduction</td>
<td>31</td>
</tr>
<tr>
<td>6.2</td>
<td>Conclusion</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>33</td>
</tr>
<tr>
<td>7.1</td>
<td>Authors Remarks</td>
<td>33</td>
</tr>
<tr>
<td>A</td>
<td>Feedback Data of Existing Systems</td>
<td>38</td>
</tr>
<tr>
<td>A.1</td>
<td>Introduction</td>
<td>39</td>
</tr>
<tr>
<td>A.2</td>
<td>Results</td>
<td>39</td>
</tr>
<tr>
<td>B</td>
<td>Software Requirement Specification Document</td>
<td>41</td>
</tr>
<tr>
<td>C</td>
<td>Introduction</td>
<td>42</td>
</tr>
<tr>
<td>C.1</td>
<td>Purpose</td>
<td>42</td>
</tr>
<tr>
<td>C.2</td>
<td>Scope</td>
<td>42</td>
</tr>
<tr>
<td>D</td>
<td>Overall Description</td>
<td>43</td>
</tr>
<tr>
<td>D.1</td>
<td>Product Perspective</td>
<td>43</td>
</tr>
<tr>
<td>D.2</td>
<td>Product Functions</td>
<td>43</td>
</tr>
<tr>
<td>D.3</td>
<td>User Characteristics</td>
<td>43</td>
</tr>
<tr>
<td>D.4</td>
<td>Constraints</td>
<td>43</td>
</tr>
<tr>
<td>D.5</td>
<td>Assumptions and Dependencies</td>
<td>44</td>
</tr>
<tr>
<td>D.6</td>
<td>Apportioning of requirements</td>
<td>44</td>
</tr>
</tbody>
</table>
### E Specific Requirements

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>E.1 External Interface Requirements</td>
<td>45</td>
</tr>
<tr>
<td>E.1.1 User Interfaces</td>
<td>45</td>
</tr>
<tr>
<td>E.1.2 Hardware Interface</td>
<td>45</td>
</tr>
<tr>
<td>E.1.3 Software Interfaces</td>
<td>45</td>
</tr>
<tr>
<td>E.2 System Features</td>
<td>46</td>
</tr>
<tr>
<td>E.2.1 Creation of LEGO Bricks</td>
<td>46</td>
</tr>
<tr>
<td>E.2.2 Editing and Manipulation of LEGO Bricks</td>
<td>46</td>
</tr>
<tr>
<td>E.2.3 Viewing the Environment</td>
<td>47</td>
</tr>
<tr>
<td>E.2.4 Data Saving and Retrieval</td>
<td>47</td>
</tr>
<tr>
<td>E.2.5 User Error Recovery</td>
<td>47</td>
</tr>
<tr>
<td>E.2.6 Application Error Recovery</td>
<td>47</td>
</tr>
<tr>
<td>E.3 Performance Requirements</td>
<td>47</td>
</tr>
</tbody>
</table>

### F Feedback Data of Produced System

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>F.1 Introduction</td>
<td>51</td>
</tr>
<tr>
<td>F.2 Results</td>
<td>51</td>
</tr>
</tbody>
</table>

### G Full Code Listing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
</table>

### H Sample Screenshots

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
</table>

1
Chapter 1

Introduction

1.1 Introduction to the Problem

Over the years, LEGO\textsuperscript{1} has established itself as a vastly popular toy enjoyed by many different audiences in a variety of formats. It’s versatility and simplicity results in endless possible creations, limited only by the user’s imagination.

This means that the number and variety of possible designs could almost be infinite. It is therefore a shame that these creations cannot always be shared with others due to real-world constraints and limitations; supply of bricks, gravity, physics to just name a few.

It is hoped that the end result of this project will produce an application that will allow users to interact with and manipulate building bricks similar to those of LEGO. This application should also remove as many of the limitations that conventional LEGO bricks come with whilst remaining easy for users to quickly and easily create and implement their designs.

Within 3D modelling software, there are many design issues that need to be considered in order for the outcome to be successful. A key issue is the interaction between a 2D and 3D space. We are trying to display to the user a 3D environment on a 2D monitor, which could potentially lead to the user perceiving something different to what is actually being displayed on the screen. This problem is also present when the user is interacting with a model. Most interaction with a graphical display is done via a mouse, which is also limited to two dimensions and can lead to problems when we want to translate this 2D motion from the user into a 3D action of the model.

With the ever increasing use of virtual environments, much research has gone into improving the interaction between these particular systems and users. However the majority of this research focuses on the use of innovative control devices. However, this project will be designed for use on a standard

\footnote{\textsuperscript{1}Information on the LEGO brand toys can be found at their website: \url{http://www.lego.com/eng/default.aspx}}
home PC, of which, the input methods are generally limited to a mouse and keyboard. The idea of introducing new hardware for users to operate the system is to be discounted as users often discount them themselves as being too complicated or foreign. For example, the DVORAC keyboard is designed for and proven to allow quicker typing speeds but still is only used by a small minority of computer users due to an unwillingness to relearn a basic input method.

Real-time modelling of 3D objects can be very laborious for some standard home PC’s. In order to maintain usability of the application, the structural design of the model must be capable of maintaining a balance between containing sufficient information and fast processing. The design must therefore address problems such as how to represent the physical shape of a brick, colouring information, positioning within the environment, so it will need to be compact and flexible.

1.2 Key Objectives

The key objective of this project is to produce a virtual environment that allows users to create and manipulate bricks with which they can create models. The bricks should be able to move to any point in the 3D space and be positioned at any angle desired. The bricks shall also not be subjected to properties such as gravity or physical collisions.
Chapter 2

Literature Survey

“We are like dwarfs standing upon the shoulders of giants, and so able to see more and see farther than the ancients.”

- Bernard of Chartres

2.1 Introduction

In creating an interactive virtual 3D environment, there are many subgoals that will need to be developed effectively and in such a fashion that they cooperate well. These will cross borders between many topics including computer graphics, psychology, algorithm efficiency and human-computer interaction usability. The following sections will review texts on these topics and allow conclusions to be drawn on specific techniques for some of the problems faced in undertaking this project.

2.2 Designing a Usable Interface

Users of the system will no doubt have a varying competence with computers and with 3D modelling. Therefore, making an interface that is intuitive will allow it to be easily understood and used effectively. Far too little time is spent on gaining an understanding on the design of a sound GUI even though this aspect of a program can be as important as knowledge of control structures or object orientation [3]. As this is an interactive system, there will be much communication between the user and the environment and therefore the users understanding of the system is vital. Most modern popular desktop applications have a menu system and a taskbars with iconic buttons. If a user is presented with an interface that resembles something familiar to them, they will be able to get access a lot of the applications functions without having to learn any new techniques. The are some basic
ideas in desktop interface design that will aid the user in getting their desired tasks done [2].

**Provide an informative and consistent interface**

Menu labels should have informative labels as to what functions they provide. This will allow users to find the functions they want quicker. This will also apply to any buttons on a toolbar. Icons used should represent to the user what the button will do. This can be hard to do with such a small icon and many users don’t use toolbars as they do not know what the icons represent and don’t want to spend time reading manuals. One way to help the user learn the meaning of icons subtly is to include the icon on the corresponding drop down menu option. This soon leads to users recognising icons and being able to use the toolbar option which are generally quicker than browsing menus. Other useful features to include are tooltips which is a textual description of what a button will do if the user hovers their mouse over it for a short period of time.

**Provide Feedback**

Whenever possible, the user should be provided with feedback from the system. The purpose of this is simply to reassure the user that the system is doing as the user wants it to, or to check that the user knows what is about to happen [5]. Such things the user may need to be informed about are:

- **Showing progress** – if a part of the system is going to take some time, (e.g. saving a large document) then the user should be informed that this is what the system is doing and indicate how much time is left.

- **Provide warnings** – before performing an irreversible action, the system should check that this is exactly what the user intended.

- **Unsaved changes** – on exiting the program, the user should be made aware if the current model is not saved.

**Allow Recovery**

Recovery can be from many different situations. As mentioned above, users should be provided with warnings on all irreversible actions. The user should be provided with a ‘cancel’ option that will return them to the system with nothing altered. Recovery can take on the more drastic form of recovering from an unexpected exit of the system. If the program were to crash and the user had not saved their model recently, the system should allow the user to recover at least part of this work. This is done by regularly saving current models to a separate file and allowing the option to read back from it when an error has occurred.
Provide Help

Even the most intuitive interfaces will have some features that some users will be unable to understand without an explanation. This can be helped by something as simple as having ‘tooltips’ as described above. More complex features can provide a help menu / button that will give detailed accounts off all the interface components.

2.3 Display of 3D Environment Techniques

[15] describes the manipulation of objects in a virtual environment as being difficult due to the lack of haptic contact in which we rely on in the real world to orient ourselves with what we are manipulating. A lot of research ([7], [15], [18]) has been done on bettering users conception of their position and orientation within a 3D environment over the past few years. Much of this research involves advanced hardware such as headsets that recognise where a user is looking or data gloves. This Lego project is being developed for use on a home computer and therefore the input devices will be generally limited to a mouse and keyboard. So how can we employ effective understanding and interaction between user and virtual environment with these relatively primitive input devices?

2.3.1 Visual Cues

“One of the difficult decisions facing the designers of applications for the interactive viewing and manipulation of virtual spaces is determining the combination of rendering techniques to use for the generation of displays. Each rendering technique provides a subset of the perceptual cues used in determining spatial relations. The job of the designer is to maximise the spatial information perceived by the user, without exceeding the computational limitations of real-time image generation in the target computing environment.”

[19]

Occlusion

Occlusion (sometimes called interposition) is a simple depth cue where more distant objects are hidden behind nearer objects. This can cause problems in a 3D interaction, however, when distant objects are completely obscured by more proximal ones. This causes uncertainties about objects that cannot be seen or even lead the user to believe that there is no object behind. [11]

One method to overcome this limitation of total occlusion is to use partial occlusion, sometimes known as the silk effect. Partial occlusion uses
semi transparency of objects to alter the lighting on objects behind other objects. If a semi transparent object appears in front of another object, the overlapping part of the distant object will be appear lighter (as if looking at it through a silk stocking). This method enhances depth information to the user as the ‘darkness’ of an object is an indication to its distance.

Partial occlusion has been used to help users select objects more effectively in 3D environments and has been described in the Manipulating the Environment section of this paper.

**Perspective Projections**

Displaying objects to a user with a perspective projection helps the user to judge how far into the virtual environment an object is. Just as in the real world, in viewing an object, its size is scaled inversely proportional to the distance from the viewer.

**Lighting and Shadows**

A common practice in computer graphics is to light objects that are close more intensely than those that are further away. Generally this is inversely proportionate to the square of the distance between the object and light source. This can be utilised in such a fashion that if the light source were to follow the viewpoint, the user would always be able to judge how far away an object was by its lighting (and therefore colouring).

Shadows essentially add another dimension of realism to the user within a virtual 3D environment. Simply applying a shadowing feature can have adverse effects on how a user perceives his or her environment however.

[10] studies the effects of different depth cues on a users accuracy of manipulating an object in a 3D environment. When performing the same task with no depth cues and then again with shadows implemented, on average, users showed a decrease in accuracy. However, when users repeated the task with shadows implemented as well as interreflections, the average user increased accuracy by over 10%.

There have been many other experiments however (E.g. [16], [9]) that conclude the use of shadows is a useful cue for indicating an objects size and position in a 3D environment.

The conflict in these arguments and the fact there are many other experiments that argue for and against the use of shadows as a visual cue has lead the author to decide that the effectiveness of shadows can depend heavily on the application. Therefore, user testing and preferences with and without shadows will be carried out as early as possible in development to ascertain whether shadows will be required in this project.
2.4 Manipulation of Environment

2.4.1 Selection of objects

There have been efforts to use this partial occlusion to improve object selection within 3D environments. This method includes using a volumetric cursor that, as the authors describe it, has a ‘silk’ covering. This has the effect of when an object is within the volume cursor, it appears slightly darker than its original colour. This is due to the user viewing it through this one layer of virtual silk. Objects that are beyond the cursor appear even darker as they will have passed through two phases of partial occlusion [22], [21].

There are however methods for using a standard cursor to decide selection of objects in a 3D scene which are covered in chapter 4.3.1.

2.5 Choice of Language to Use

2.5.1 Introduction

Many programming languages were designed to fulfill specific problems when developed. There is therefore no one ‘best’ language to be used for any project. The nature of the problem should shape the choice of language to maximise the power the programmer has in creating the program. As in many fields of work, it is a case of using the right tool for the right job.

The fact that Lego bricks are all made with just a few basic shapes (a cuboid with a small cylinder on top duplicated many times), lends itself very nicely to be designed within an object oriented language.

The author claims that an object oriented language is the correct choice as these languages will offer the following:

- Objects – objects are self contained and hold properties similar to the objects being represented in real life.

- Encapsulation – separates the internal state of an object from the way in which the program or user interacts with it. This ensures that users of an object cannot change it properties in unexpected ways. E.g. changing the length of a brick after it’s been created.

- Abstraction – this allows objects to manipulate, reveal information or communicate with other objects without having to worry about how these objects are implemented.

- Polymorphism – using message sending, methods can react in different ways depending on which object it was sent to. E.g. if a brick is sent the message “rotate” it will behave differently to when the camera is sent the same message.
Inheritance – allows objects to be defined and created from already existing objects. They can inherit or extend behaviours without having to state them again. This is excellent as a simple piece of Lego (as stated above) is made of a cuboid and cylinder. All the other pieces are then made up of more and more of these.

There is much debate amongst computer programmers as to which language is the best or which one performs better but a lot of these arguments are based on bias personal opinions and preferences. The author will have to be careful in examining sources of comparison between languages to review their validity. The hardware that is used can also be a factor in performance of a computer language.

The most common object oriented languages used within modern times are Java and C++ and in these next sections I will try to form a reputable argument for choice of one over the other.

2.5.2 Introduction to C++

C++ is considered a general purpose language with support for procedural, object oriented and generic programming which has made it one of the most popular commercial programming languages since the 1990’s. It was developed in the 1980’s by Bjarne Stroustrup as an improvement to the C language.

2.5.3 Introduction to Java

Java was produced by Sun Microsystems with the intension to replace C++. It was released in 1994 but didn’t gain any real reputation until it was released in 1995 that the Netscape Navigator browser would include support for the language.

One of the main issues Java tried to cover was portability. By using the Java compiler to compile code into a bytecode, this can be run by a virtual machine that is written to coincide with the corresponding hardware. This allows programmers to write a program once and run it on any platform.

2.5.4 C++ vs. Java

Finding studies on the performance of one language compared to another from an independent body with no bias has proved difficult. The findings of studies I believe to have been reliable are summarised below.

One study, [17], considers not just the performance of languages but also takes into account the personal difference in programming methods. By using 38 different programmers to code a solution to the same problem, they were able to obtain a wide variety of coding methods and test a multitude of performances.
These tests showed that on average, Java programs used 2 to 3 times more memory than those written in C++. What it possibly more surprising and a more telling statistic is that even the most memory efficient Java programs were a little more memory intensive than the C++ programs with medium memory efficiency.

The run time of programs had some extremely conclusive results in the favour of C++. The mean of the run times of Java programs were 18 times slower than the C++ one. This was because of a few exceptionally slow running Java programs. This statistic isn’t helped much even when these extreme values are removed as the median of the best Java run times was still 10 times slower than the fastest C++ programs.

The other measurement of these programs was on the processing time which will highly depend on the design considerations from the programmer and so does not exactly reflect the performance of the language. The interesting statistic that was shown from this test was that processing times varied much more in the Java programs compared to the C++. This would suggest that the design of Java programs require more careful considerations.

2.6 Similar Software

Into sentence needs to define whether i knew these already existed or whether these changed the aims of the project. Should also only be breif overviews as the requirements will properly test them,

There are some applications available for inspection that are similar to the LEGO software intended by the author. These software will provide a valuable insight to some of the aspects that the author will have to undertake in his own project. By questioning users of their opinions on features within the systems, the author can discover what features would be good to incorporate into his own project. Feedback was gathered from a number of test users on some of the applications available and their findings are briefly mentioned in the corresponding sub-sections but are further detailed in chapter 6.

2.6.1 LEGO Digital Designer

Introduction

LEGO Digital Designer (LDD) is the LEGO brands official software for designing LEGO models. Users can download models of a vast selection of the official LEGO bricks with which to design with. Users can then upload their models to share with others and, likewise, download others models. This idea turned out to be extremely beneficial to LEGO as it offered LEGO the chance to scout new employees with very little effort and there are some people who have gained employment through their personal models. The
other major advantage to LEGO this system offers is that once a user has created a model, or downloaded one they like, they can order all the bricks in that model and create the model for real themselves.

**Key Points**
- Wide range of official LEGO bricks.
- Official bricks means purpose of most bricks will be familiar to most users with any LEGO experience.
- Same window for creating and viewing models.
- Camera rotation lacks precision to view models at certain angles.
- Lengthy operation to choose a desired brick.

### 2.6.2 LDraw

**Introduction**

LDraw was originally created by James Jessiman and has a very large fan base across the internet. Unfortunately, James died in 1997 and since then, LDraw has had very little modification except to expand the number of bricks available to users.

**Key Points**
- Offers users 3 windows with different views to manipulate model
- Separate window to view model
- Massive choice of bricks (and continually growing)
- Animation feature

### 2.6.3 LeoCAD

**Introduction**

LeoCAD is a LEGO CAD program that was designed by someone going by the name of ‘Leo’. Information is very sparse on LeoCAD as the creator refuses to leave any contact information about them self and also has not completed the content of the official website for LeoCAD. None the less, the program offers a basic interface that provides some of the features aiming to be achieved by this project.
Key Points

- Provides a list of recently used bricks.
- Design and display done in same window.
- Erratic movement of bricks.
- Cannot access items that are obscured by others without rotating the view.
- Erratic movement of camera.
- Lengthy process between switching operations (rotation, moving, selecting).

2.7 Conclusion

From studying much material, it is clear that there are many possible routes to the same goal. Consideration of each method’s strengths and weaknesses in the domain of our system design will need to be considered to maximise the potential benefits.

It is also noticeable that our desired system has been attempted by others and, in the opinion of the author, have only partially been successful. To remove the possibly bias opinions of the author, it is thought that further opinions of the existing software are required.
Chapter 3

Requirements

“Software requirements express the needs and constraints that are placed upon a software product that contribute to the satisfaction of some real world application.”

3.1 Introduction

The main requirements for the system are easily derivable from the objectives set out in this chapter. To obtain some further (and possibly more specific) requirements, ‘requirements gathering’ was conducted with several test users. The following section details the procedure used for this and the conclusions drawn.

3.1.1 Requirements Gathering

Research has shown that 40% to 60% of errors in a system stem from errors within the requirements stage ([6], [13]). There are many sources that detail the importance of correctly gathering requirements and techniques which can be employed to do so effectively. The author’s main concerns about requirements are detailed below but readers are directed to [12] for an in depth discussion of requirements engineering.

3.1.2 Gathering Techniques

There are many ways to collect requirements and no one way is perfect. It is suggested that a combination of multiple techniques are used to to ensure best results. To collect requirements for this project, the author decided the collaboration of 3 different techniques would be most beneficial. These were:

- Questionnaires
Brainstorming Sessions

Requirement Workshops

The decision to use these methods was based on their simplicity and ease of execution. A brief overview of how these three methods were incorporated into the project are detailed below.

Questionnaires

During the research stage of this project, it was discovered that there are some applications in existence that are close to what we are trying to achieve (chapter 2.6). It was decided that testing these could offer insights into solving known problems and highlight previously unseen problems that may need consideration.

9 test users were asked to spend 20 minutes using LEGO Digital Designer, LeoCAD and LDraw and asked to fill out a feedback questionnaire after each session. Using questionnaires allowed for a very quick method to gather a large amount of information with the least amount of inconvenience to the participants. Each user was also asked to reproduce the LEGO structure in figure 3.1 with all 3 pieces of software as well as with real LEGO bricks. The purpose of this is to allow a loose form of comparison between the developed system and of those already in existence. It will also give an indication as to whether the ease of creating models has transferred from conventional LEGO into the developed system. A selection of these questionnaires are included in appendix ?? that highlight the general thoughts of the test users. The conclusions from the feedback forms are available in appendix A.

Brainstorming Session

All 9 users were asked to attend a brainstorming session of whom 5 attended and lasted for 80 minutes. Using the comments from the questionnaires, ideas were put to the group and discussed to discover what requirements should be included in the system. It also gave the test users a chance to put
forward any thoughts or ideas they had of their own on how such a system should be designed.

Requirement Workshop

Of the 9 test users, 6 attended a requirement workshop that was held as an open discussion in a similar way to that of the brainstorming session. The critical difference between the two sessions was that instead of generating ideas, the group were confirming an understanding of the ideas put forward. From this session came the finalised requirements.

3.1.3 Requirement Specification

During the requirement gathering process, requirements were detailed in a Software Requirement Specification (SRS) document. A SRS acts as a central declaration of the requirements of a system which is used by anybody connected to that system (e.g. developers, managers, users) It is therefore vital that the requirements are documented in a readily understandable manner and with a unified meaning so that no expectation gaps appear, where the system produced is not that what the potential users envisioned [1].

There are many examples of how a SRS document can be structured and the final decision may come down to domain compatibility and/or personal preference. As a template for an SRS model, the author chose an example from the IEEE recommendations [4]1 for its credibility and accessibility.

A copy of the SRS generated for this project can be viewed in Appendix B. The requirements detailed in the SRS document cover those of a complete system and therefore exceed that of what is possible within the timescale allocated for this project. What we are focusing on are the key ideas to making a successful system to achieve our aims.

The author has picked the key requirements from the SRS document and project proposal, which shall be documented in the following section.

3.2 Specific Requirements

The aim of this project has been stated to produce a piece of software that enabled a virtual emulation of building with LEGO bricks that were not bound by the restrictions imposed upon model making by the real world.

The most important features that the system should incorporate can be derived from the specification made in chapter 1. These are:

- Provide the user with a 3D environment in which to design models.

1The example chosen is A5 but slightly modified to fit better with the project.
• Allow users to create a brick that represents that of a LEGO brick.
• Allow users to manipulate created bricks in the following ways:
  – Position along all three axis.
  – Rotation of a brick.
  – Colour of a brick.
• All control should be possible via a mouse or keyboard.
• Provide the user with a realistic 3D view of their model.
• The 3D view allows users to view their model freely from any angle.

Other requirements can be drawn from the requirement gathering sessions held, detailed in the SRS document (appendix B)

• Provide users with a ‘zoom’ feature so they can view models more closely or distantly.
• Design should be carried out separately from 3D viewing.
• All design should be carried out in 2D views to avoid complication.
• There must be enough 2D views provided to enable users to view bricks positions along all 3 axes.
• When moving bricks, their position should be shown in real time.

The main point test users stressed was that designing a model within a 3D view, such as LeoCAD and LEGO Digital Designer, was far from satisfactory. It was described as ‘confusing’, ‘frustrating’ and behaved ‘randomly’. The other method for designing tested by users, was that of LDraw which only allows manipulation of bricks along 2 axes at a time. To enable full 3D editing, the view could be changed to make a different combination of 2 axes active. The differences in times for creating the test model (figure 3.1) certainly favour the use of LDraw (appendix A) and it is this fact that resulted in the following key decision; that the design of models should take place separately from the 3D modeling.

3.3 Conclusion
The main aims for this project were clearly set in chapter 1, and with some user testing, we have established a few more low level requirements. Also, with the discovery and testing of some existing software, it is apparent that many of the required features for the system have already been designed to
an acceptable standard. This allows the author to attempt to amalgamate all the best features of the existing systems to produce a complete system to cover our needs. This is preferable to attempting to create an entirely new system that would encompass the same ideals as what is already available.
Chapter 4

Design

“Think simple, as my old master used to say - meaning reduce the whole of its parts into the simplest terms, getting back to first principles.”

- Frank Lloyd Wright

4.1 Co-ordinate Systems

Everything within the LEGO environment will require a position and a size, both of which can be achieved using the 3D Cartesian co-ordinate system. This system allows for representation of height, width and length which can be used to define the dimensions of a LEGO brick. It also allows for the unique description of any point within a volume through 3 numbers (usually called the x, y and z co-ordinates) which is demonstrated in figure 4.1. Keeping these two properties detailed in the same system enables the user to easily move a brick to its correct position and draw its correct dimensions. Therefore, it is proposed that the system shall use the 3D Cartesian co-ordinate system where the position and size of a brick can be stored as values of x, y and z in the corresponding axis.

For the environment, three separate spatial systems will be used. This will help break down all the transformations required on a model into more manageable sub sections.

World Space - Describes any transformations that are to be applied to the whole system. These would be things such as if the user wanted to rotate the world to view the model from a different angle.

1order of sections here needs considering
Figure 4.1: 3 Dimension Cartesian Co-ordinate System.

Figure 4.2: Left handed co-ordinate system (a) and a right handed co-ordinate system (b)

**Model Space** - Each brick in the model will have details about its position, and any rotations upon it.

**Screen Space** - The positions of bricks calculated by the previous two points, will then need to be put into the co-ordinate system of the screen rendering software.

To avoid unexpected results from these co-ordinate systems, orientation must be taken into account. In 3D co-ordinate definition, most models are considered to be ‘left handed’ where as most rendering software uses a co-ordinate system that is ‘right handed’. The difference between these two systems is demonstrated in figure 4.2. Altering the orientation is simply done by flipping the y-axis, which can be achieved at the point before rendering.
4.1.1 Transformations

When applying all the transformations from the mentioned co-ordinate systems, if we pay attention to the order in which we apply them, we can save ourselves from repeating the same transformations needlessly. The sequence of transformations that each LEGO brick will undergo is as follows.

- Translate to the origin
- Apply the world space matrix
- Translate the brick to its respective position in the environment
- Rotate the brick accordingly
- Apply the viewport orientation
- Translate to screen space

Notice that the last step we take before rendering the model to the screen is we adapt the model to the viewport. Viewports are discussed in section 4.5 (page , viewpoint section23) and how each one affects the model. The reason the viewport matrix is applied so late is that it saves us from having to maintain several separate instances of the model to have multiple views of it, thus, lessening memory load.

4.2 Design of LEGO bricks

Before designing a virtual LEGO brick for the system, we should first consider the design of a real LEGO brick. For the first version of software produced, which is only a prototype to test the fundamental ideals upon which this project is based, the LEGO system shall only consider the ‘classic’ LEGO brick, figure 4.3.

In the world of LEGO, this ‘classic’ brick is described as a ‘4 x 2’ brick; noticeably because it has 4 rows and 2 columns of nobs on the top. We
can see that this brick, and many more of the common LEGO bricks can be made up of a standard single brick like the one displayed in 4.4\textsuperscript{2}.

LEGO bricks have a design on the underside described as anti-nobs, which are predominantly hollow. This is included to allow bricks to be connected via nobs and anti-nobs. This design is solely for physical purposes and coupled with the fact it is visually complex to model, this design shall be ignored in the produced system. This shall save time in modeling and rendering the bricks.

With the above points considered, the proposed design for a virtual LEGO brick shall be a cuboid with a cylinder placed on top (like that of figure 4.4). Using dimension information about a certain brick will indicate how many times the standard brick is to be drawn.

### 4.3 Structure of Environment

Now that we have a method of modeling a single LEGO brick, we need a way to store all those bricks active within the environment. The system will be using a linked-list structure to store a link to each of the instances of bricks created. Traversing the list and rendering each object will create a scene with all our active bricks. A linked-list will make it easy to dynamically add and remove bricks from anywhere in the model at any time.

The only other obvious solution to the storage of all our bricks is an array. The linked-list design was chosen over this approach though for many factors. Once declared, arrays have a fixed length. This leads to a dilemma as to what length to make the array. Making it too short will mean the user cannot create any more bricks but making it too big will mean memory is being wasted.

The simplicity of arrays will have to be considered against the flexibility of linked lists when implementing this section.

\textsuperscript{2}Numbers in the diagram are for ratios only, not measurements.
4.3.1 Object Selection

Being an interactive system, we need a way for our users to select a brick to manipulate. The best method for this would be to simply click on the desired brick and then perform whatever actions are required. By the time a model is rendered on the screen, all the bricks have gone under many translations and rotations and therefore would make deciding which brick the user has clicked on very difficult. The use of OpenGL solves this problem. OpenGL has built in features that are commonly used for the selection of objects. The following subsections will provide a brief overview of both methods that the author will then use to determine which is more suitable.

‘Picking’ Algorithm

Each primitive object in the system must first be given a unique ID. By enabling OpenGL’s selection mode, it will return a ‘hit’ whenever a primitive lies within a certain specified area (the mouse co-ordinates in our case). Since we are specifying a point and not an area, we will only ever get one (or none if nothing is there) hit and so we can return this ID as the primitive that has been selected [14].

The main advantage for this technique is that all the features are already built into OpenGL and therefore minimises the need for any complex code to be written. This method is also very flexible and can be used with grouped or hierarchical items which may be useful for further implementations of the system but also has limitations on the complexity of hierarchical models it can handle due to the size of the name stack.

Back Buffer Colouring

Again, this technique requires each primitive to be uniquely identifiable, but this time each object is given a unique colour. OpenGL then renders the scene into the back buffer (which doesn’t get rendered to screen) with each object drawn in their unique colour. Determining the selected object is then simply a case of retrieving the colour at the position of the mouse and matching it back to your objects.

This method requires some care as the colour buffer of the hardware will not always be the same which can cause some colours chosen to not ‘display’ correctly. This can cause the algorithm to perform incorrectly as it does not recognise the colour it has found under the cursor and cannot match it to any of the ones you have specified. This algorithm is extremely fast however, since you are simply recolouring a scene that you have already created and will also work with extremely complex models.
Conclusion

As there is little difference between the outcomes of both methods, there is no right or wrong method to use. In anticipation of greater ease of use, the picking algorithm will be implemented in this project.

4.4 Lighting

During the requirement gathering stage, it was discovered that users were not too concerned about any kind of lighting on the model. One of the testers stated “Lighting could be seen as a physical limitation, like gravity and the others, which your program is trying to remove”. The argument being that if realistic lighting were to be implemented then our models would end up with shadows in areas which can reduce the visual appeal or inhibit design in some manner.

This somewhat ties up with the discoveries made in the literature review concerning shadows (section 2.3.1) where there were studies showed improvements and lowering of performance with shadows. One of the main benefits of including shadows in a virtual environment was that is gave a good indication as to an objects position within the environment. Since we have already discussed viewports as a method of describing where an object is in space, the use of shadows shall be ignored from the initial implementation of the system and reconsidered with feedback from users.

Within OpenGL, there are many lighting functions and features, but if none these are specified, then models will be rendered with pure ambient lighting, where all object are coloured with their specified colour and not modified in any way. This will create a bright, clear picture of our models.

4.5 Viewports

4.5.1 Introduction to Viewports

A viewport is defined as a 2D rectangle (on the screen) that a 3D scene is projected onto and in turn viewed [8]. These viewports can contain different projections of the same model to provide different angles of viewing.

As it was seen in the user evaluation of similar software, viewports will play an important role in the interaction between the user and their virtual models. When users were trying to manipulate the model in a 3D view, they found it very difficult to position bricks where they wanted or to even know if bricks were where they thought they were. To overcome this problem, it is suggested that we only allow users to manipulate a model in two dimensions, which is easily controllable as a mouse works in these dimensions also. This will mean that any movements using three axis that a user wants to perform on bricks will require two phases; one in one 2D view and then another in
Figure 4.5: Demonstration of where three design viewports will view a model from.

Figure 4.6: Demonstration of how the three design viewports will display a model.

a different view. Using two operations to complete one task isn’t an ideal solution, but based on the feedback from the test users it would appear that this is still a quicker method than direct manipulation of the 3D model.

4.5.2 Design Viewports

By offering 3 orthogonal views of the model, we can ensure the user has a clear representation of where a brick is along each of the three axis in our virtual space. These three viewports will be considered as the ‘design’ views as only in these views will the user be able to manipulate their model. The choice of orthogonal viewing for the design viewports is because we don’t want any form of perspective in these views. If we did have perspective active, a brick close to the ‘camera’ would appear bigger and obstruct the user from viewing any other bricks.

If we were to place the camera for each of the design views in positions that looked straight down an axis towards the origin (demonstrate by A, B & C in figure 4.5) we will obtain the desired representation of location in a 3D space. The views that this will create are shown in figure 4.5. Simply put, we are viewing the model from the top, front and side.
4.5.3 Viewing Viewports

Being able to create a model that cannot be viewed appropriately seems fairly illogical, and so we need to ensure we give the users a method with which they can view models such as they would with a physical LEGO model. The author proposes this should be achieved by allowing a camera to rotate on circles around the x and y axes, and always pointing towards the origin.

4.5.4 Web of Influence

Many of the features mentioned above rely on and directly affect one another. To try and indicate these dependencies, figure 4.7 maps out this ‘Web of Influence’.

In the diagram, an arrow flows from the affector to the influenced section. It shows how the displayed model is influenced directly by the viewports and by the objects being rendered. Both of these factors are further influenced by other details, all of which build up the model.

4.6 Manipulation of the Environment

When moving bricks, it is standard amongst the existing LEGO software that bricks are moved by dragging them with the mouse. This clearly is a sensible option for moving objects are the dragging of the mouse whilst holding a button down, somewhat represents the way in what a human would move a real LEGO brick; grasp the brick and move their hand.

It is therefore proposed that movement of bricks shall be carried out by a
user clicking on a brick and dragging it to their desired location. The brick’s location should be rendered in real time to provide the user with sufficient feedback to where they are moving the brick.
Chapter 5

Implementation

5.1 Choice of Language

During the literature review (chapter 2), we discussed how C++ would make an ideal choice for use within the project. Reasons for this were that object orientation would provide an easy way to set a base brick design that would allow the creation of many common LEGO bricks via inheritance. In reality though, this design idea was not used. During the implementation, the author had many problems in improving knowledge to a sufficient level to cope with the designed method of object selection. With this setting the whole project behind schedule, it was decided to switch the algorithm (described in chapter 4.3.1). In doing so, the author took the further step to switch from C++ to ANSI C as it felt by the author that the use of object orientation with OpenGL was not very well suitable. The switch was also an effort to gain back time lost on the project due to the authors familiarity with C more so than C++.

5.1.1 Implementation of brick

We represent a brick as a simple record, defined in ANSI C standard, declared as follows:

```c
struct brick{
    Bool rotated;
    int size;
    int x;
    int y;
    int z;
};
```

The x, y and z elements contain our details of where in the environment the centre of this particular brick is. Placing the brick in the correct position is then simply a case of:
The element size was included to allow the creation of different length bricks, a feature that was never implemented.

Bool rotated defines whether a brick is rotated or not. If a brick is set as rotated then it is turned 90 about the y-axis. Setting this back to false then sets the brick back to original rotation which gives the impression that the brick is spinning constantly in 90 increments. In looking back at this implementation choice, it bears no advantages over us storing the angle of rotation around an axis. The choice to do it this way was that it was considered more compatible with the selection algorithm, which was initially written when bricks could not obtain any rotation.

5.2 Storing Bricks in Our World

Our choice of data structure which will contain all active bricks in the environment is of high importance as it will highly influence the the structure of our rendering code. The popular choice here would probably be simply to store our brick records in a C array. However introducing arbitrary boundary conditions, and possible costly array resizes is not considered good engineering practice. Indeed, it is the case here that we would not care about the index of the array in the program logic, and as such a linked list would be a better choice. Defined as:

typedef struct ns {
    struct brick *data;
    struct ns *next;
} node;

We then introduce standard functions for adding and removing elements:

node *list_add(node **p, struct brick *i) {
    node *n = malloc(sizeof(node));
    n->next = *p;
    *p = n;
    n->data = i;
    return n;
}

void list_remove(node **p) {
    if (*p != NULL) {
    
28
We also introduce a higher order function, ‘map’ *, in the functional programming tradition, for our linked list of bricks:

```c
void map(node *n, int (*fp)(struct brick *i)) {
    while (n != NULL) {
        fp ((n->data));
        n = n->next;
    }
}
```

This function allows for rather elegant and simple rendering of a scene:

```c
map(lList, drawGLBrick);
```

Where `lList` is our global list of bricks, and `drawGLBrick` is our function to render a single brick. The function `list_search` , whose full definition is omitted for brevity, is utilised by our picking code described elsewhere.

```c
struct brick *list_search(node *n, int x, int y, int window)
```

### 5.3 Selection of Bricks

To allow maximum ease of use, it was decided that bricks should be ‘dragable’ in the design viewports, to mimic those actions of picking up a physical brick and moving it with your hand. Concentration on this feature turned out to be the demise of the project as the designed method of performing this action was not achievable by the designer and so some improvisation was required. The following algorithm is how the selection of a brick was implemented. Source code of this can be found in appendix G.

- On detection of the mouse button is pressed
- Calculate which view the cursor is in
- Iterate through the list of active bricks and determine if any of them are under the cursor.
- Upon detection of a brick that is under the cursor, store the depth of that brick, in the chosen view, as the closest found
• If more bricks are found under the cursor, the depth is compared against that of the closest one found yet

• On reaching the end of the list, the closest brick is returned, or null if no bricks were under the cursor

5.4 Moving Bricks

Using the selection algorithm described above, we also set a boolean variable movingBrickNow which indicates we are about to move a brick. With this set, the system monitors for movements in the mouse and if detected, adjusts the position values (x, y and z) for the corresponding axis of the view the cursor is in.
Chapter 6

Testing

6.1 Introduction

The time available to test the system was seriously limited due to development overrunning schedule and so, testing had to be limited and targeted as best as possible.

No formal code inspections were carried out because (as detailed in chapter 7.1) the produced code was deemed to not be of sufficient quality for extending to further use. The author felt that it was more important to test whether any of the system designs that were implemented were successful in their tasks.

Testing shall be done in exactly the same way in which existing LEGO software was previously tested in the project. The layout of this test is already designed and requires the author to begin straight away and allow for direct comparison between the sets of data for the produced system and existing ones.

6.2 Conclusion

The user testing produced some mixed results of which shall be summarised here. Full details of the results can be found in appendix F.

The main achievement of the system was the ease with which users were able to create models. This was shown by a high proportion of users rating the ease to create models as “Good” and “Very Good” in some cases. These statements were backed up by the performance of users in the test to reproduce the test model (figure 3.1). The produced system allowed users, on average, to create the model quicker than when using LeoCAD and marginally faster than LEGO Digital Designer.

There is a significant jump in average score for the produced system between the ability to create and view models (3.4 and 1.8 respectively). This is due to the 3D viewpoint never got fully implemented and therefore
only offered one static view.

In regards to the main requirements set for the project, the system has managed to achieve the creation of a 3D environment that allows the creation and manipulation of bricks via their position and rotation. Most of the physical limits imposed on real LEGO bricks were also removed, such as gravity and intersection. The produced system is still however very basic and only covers some of these requirements to a minimal standard. The system needs to allow users a greater control of accuracy and freedom, for example, in rotating bricks, where users are currently limited to large rotations.
Chapter 7

Conclusion

The objective of this project was to implement a working prototype of a virtual LEGO experience that allowed the manipulation of bricks in a variety of manners and removed some of the physical limitations imposed on standard LEGO. The produced system meets these essential demands and many of the requirements detailed in chapter 3.2. The author feels that despite meeting most of the essential requirements, there are many problems with the system, which contains many limitations. The following sections will detail the authors opinion of the project.

7.1 Authors Remarks

The key point to be commented on is that despite covering the majority of set requirements and some positive remarks from test users of the system, the author feels that the produced system, as a whole, is of poor quality.

A fatal error made was in underestimating the learning time required for all the desired OpenGL features to be used. As development time was restricted, the implementation had to deviate away from the design in order to produce a system capable of testing some if not all of the design features. All problems stemmed from being unable to implement the designed method of object selection. This proved to be far more complicated than much of the documentation claims it to be. This put the development behind schedule. Also object selection is a vital part of the system because many other features rely upon it, and so problems associated with object selection hindered the development of much of the project. For example, the selection algorithm implemented which, although works correctly, limited further development in other areas due to its simplicity. The main example of this being that bricks can only be rotated by 90 degrees around the y-axis. If bricks are rotated by angles other than multiples of 90, then the selection algorithm can behave incorrectly. This limitation to the system could be overcome with the use of the selection algorithm detailed in chapter 4.3.1 which would simply
require more experience in OpenGL.

With regards to the original problem; the need to create a virtual environment where LEGO bricks could be manipulated without the constraints of real life physics, a solution has been reached. The programme does indeed enable the user to do this, albeit in a overly simplified manner due to the issues with the code. The user testing has shown that a given model can be recreated using the system within a reasonable time frame. The manipulation of bricks accurately modelled the actions performed when picking up and placing physical bricks, thereby aiding the user considerably as he or she could replicate actions that were familiar to them. This can be viewed as a considerable success of the system. Obviously the system was let down by the limited knowledge of the OpenGL libraries, and so could readily be improved in numerous ways once this knowledge was gained.

One area to concentrate on further developments or work within this field would be the display and viewing of models. Each of the four tested systems scored rather poorly in their presentation of the 3D model. Interestingly, each system got bad remarks from users for different reasons. The four systems each scored low on the ability to display the 3D models. Their scores are listed below with common reasons that users gave for rating them so low.

**LDraw** - 2.8 - Did not behave as expected.

**LeoCAD** - 2.6 - Far too hard to control.

**LEGO Digital Designer** - 2.8 - Lack of precision in moving the camera

**Produced system** - 1.8 - Static view of model

It is therefore a shame that the intended method for viewing models in 3D was not implemented (detailed in chapter 4.5.3) as this could possibly have solved a problem that is common to all current LEGO systems. It would be highly advisable that any further development in this area concentrate on this problem as it is letting each system down and would heighten the experience greatly if a reasonable solution was found.
Bibliography


Appendix A

Feedback Data of Existing Systems
A.1 Introduction

The following tables show the data collected from the tests carried out by users on the three existing LEGO systems, LeoCAD, LDraw and LEGO Digital Designer.

The letters in the table represent the following:

A - Users response to the question “How easy was it to make a model with the system?”

B - Users response to the question “What is your opinion of the method used to view your model?”

C - Users response to the question “What was the standard of the rest of the features offered?”

D - The time it took a user to complete the test model, figure 3.1.

The conditions, A, B and C are scored from 1 - 5 with 1 being the lowest score and 5 being the highest. Condition D is measured in seconds.

A.2 Results

This table clearly shows that LDraw out performs the other software in every statistic.
Figure A.1: The N/A result for column D in LeoCAD’s data is due to the fact that this user refused to carry on after 7 minutes with little progress.

<table>
<thead>
<tr>
<th>Test User</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>254</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>222</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>399</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>N/A</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
<td>3</td>
<td>410</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>231</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>254</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>380</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>288</td>
</tr>
<tr>
<td><strong>Mean:</strong></td>
<td><strong>1.8</strong></td>
<td><strong>2.6</strong></td>
<td><strong>1.9</strong></td>
<td><strong>305</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test User</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>184</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>172</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>4</td>
<td>270</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>222</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>2</td>
<td>3</td>
<td>170</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>194</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>183</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>215</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>130</td>
</tr>
<tr>
<td><strong>Mean:</strong></td>
<td><strong>4.1</strong></td>
<td><strong>2.8</strong></td>
<td><strong>2.7</strong></td>
<td><strong>193</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test User</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>254</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>222</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>212</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>193</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>410</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>231</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>254</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>301</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>288</td>
</tr>
<tr>
<td><strong>Mean:</strong></td>
<td><strong>2</strong></td>
<td><strong>2.8</strong></td>
<td><strong>2</strong></td>
<td><strong>263</strong></td>
</tr>
</tbody>
</table>
Appendix B

Software Requirement Specification Document
Appendix C

Introduction

C.1 Purpose

This Software Requirement Specification document acts as the official statement regarding all requirements of the LEGO software to be produced. For the test users, this document states precisely what features would be desired in the outcome and as to what importance they are. As a developer, the author will use these stated requirements as strict guidelines during development.

C.2 Scope

This document covers the Virtual LEGO Environment being produced by the author as part of BSc(Hons) course in Computer Science at the University of Bath. The software produced is intended to run on a standard home PC with mouse and keyboard as input allowing for the viewing and manipulation of LEGO bricks.
Appendix D

Overall Description

D.1 Product Perspective

The Virtual LEGO Environment is intended as a stand alone application. It is an attempt to recreate the current LEGO software available with the users considerations in design.

D.2 Product Functions

The major functions that the LEGO Virtual Environment will incorporate are listed below:

- Allow the user to view a virtual 3D environment.
- Allow the user to create LEGO bricks and place them into the environment.
- Allow for the manipulation of created bricks (colour, position, rotation).

D.3 User Characteristics

Users of the system have no necessary level of education or experience. The more experience a user has though will indeed speed up their training time to use the more advanced features.

D.4 Constraints

- Whilst the system is running, other processes should be allowed to continue on the computer running it.
D.5 Assumptions and Dependencies

- The system will run on the Microsoft Windows XP.
- There will be input device for control of a pointer.

D.6 Apportioning of requirements

For the first version of the system, only those requirements listed as Essential shall be implemented. After those have been approved by testing, with time permitting, the development of those requirements listed as Conditional may begin.
Appendix E

Specific Requirements

E.1 External Interface Requirements

E.1.1 User Interfaces

• The system interface shall have a look and feel recognisable to 80% of users.

• The system interface shall adjust to work on all monitor resolutions perceivably testable.

• Messages to the user shall be judged correctly by the user as warning, error, information, etc in all cases.

• After 30 minutes of use, users shall be able to recall without remembering how to perform 75% of functions within the system.

E.1.2 Hardware Interface

• The system shall work on 90% of all perceivable computers that fall within the constraints of section (Assumptions and Dependencies)

• The system shall handle cursor control with all perceivably testable devices.

• The system shall work with all perceivable text input devices.

E.1.3 Software Interfaces

• Communication between the system and operating system shall be via the .NET framework.
E.2 System Features

E.2.1 Creation of LEGO Bricks

- All available bricks must be listed to the user.
- The user must be able to filter the bricks to smaller categories (e.g. tiles, bricks, etc).
- Pictorial previews of the bricks must be presented to the user.
- Created bricks will be of the colour last selected.

E.2.2 Editing and Manipulation of LEGO Bricks

Colour

- Users shall be able to change the colour of bricks after creation.
- Users shall have a list of common colours always available to them.
- Users shall be able to use a colour picking chart to access and colour bricks in over 32,000 colours.

Position

- Bricks shall be moveable in 2D views after creation.

Transparency

- The transparency of bricks shall be changeable after creation.

Removal

- Users will be able to remove any created bricks.

Rotation

- It shall be possible to rotate bricks around all three axis.
- Rotation shall be possible to an accuracy of 1 degree.
- There shall be shortcuts to allow rotation along any axis by 90 degrees.
E.2.3 Viewing the Environment

Camera Control
• User should be able to have multiple views of the model at one time.
• Camera shall maintain view of centre of model.
• Camera shall be able to zoom in and out.
• Camera shall be able to rotate on all three axes.

E.2.4 Data Saving and Retrieval

Saving a Model
• Users shall be able to save a model to an external file.
• Saved files shall be compatible with other computers other than those originally saved with.

Loading a Model
• Users should be able to edit loaded models.

E.2.5 User Error Recovery
• An undo function shall be provided to one level of recovery.

E.2.6 Application Error Recovery
• Upon an unexpected application exit (e.g. power failure, system crash) the system should offer users the chance to recover their model to a position of no more than 5 steps previous to the crash.

E.3 Performance Requirements
• 90% of all rotation operations on bricks shall take less than 1 second.
• The time between creating a brick and seeing that created brick display shall be less than 1 second for 90% of bricks created.
• When moving a brick within the environment, the system shall take less than 1 second to update the on screen position of the brick 90% of the time.
Appendix F

Feedback Data of Produced System
F.1 Introduction

The following table shows the data collected from the test carried out by users on the produced system.

The letters in the table represent the following:

A - Users response to the question “How easy was it to make a model with the system?”

B - Users response to the question “What is your opinion of the method used to view your model?”

C - Users response to the question “What was the standard of the rest of the features offered?”

D - The time it took a user to complete the test model, figure 3.1.

The conditions, A, B and C are scored from 1 - 5 with 1 being the lowest score and 5 being the highest. Condition D is measured in seconds.

F.2 Results
<table>
<thead>
<tr>
<th>Test User</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>254</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>254</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>280</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>1</td>
<td>189</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>259</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>231</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>220</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>367</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>257</td>
</tr>
<tr>
<td><strong>Mean:</strong></td>
<td><strong>3.4</strong></td>
<td><strong>1.8</strong></td>
<td><strong>1.6</strong></td>
<td><strong>257</strong></td>
</tr>
</tbody>
</table>

Figure F.1: Data from the user tests on the produced system
Appendix G

Full Code Listing
The following is a full listing of the code produced. The file is based on that of Lesson 5 from the Neon Helium OpenGL tutorial[]. Details of important algorithms or sections of code can be found in chapter[].

```
#include <stdio.h>
#include <GL/glx.h>
#include <GL/gl.h>
#include <GL/glu.h>
#include <X11/extensions/xf86vmode.h>
#include<X11/keysym.h>

/* stuff about our window grouped together */
typedef struct {
    Display *dpy;
    int screen;
    Window win;
    GLXContext ctx;
    XSetWindowAttributes attr;
    Bool fs;
    Bool doubleBuffered;
    XF86VidModeModeInfo deskMode;
    int x, y;
    unsigned int width, height;
    unsigned int depth;
} GLWindow;

GLUquadricObj *quadratic; /* pointer to our quadratic objects */

/* attributes for a single buffered visual in RGBA format with at least
 * 4 bits per color and a 16 bit depth buffer */
static int attrListSgl[] = {GLX_RGBA, GLX_RED_SIZE, 4,
    GLX_GREEN_SIZE, 4,
    GLX_BLUE_SIZE, 4,
    GLX_DEPTH_SIZE, 16,
    None};

/* attributes for a double buffered visual in RGBA format with at least
 * 4 bits per color and a 16 bit depth buffer */
static int attrListDbl[] = { GLX_RGBA, GLX_DOUBLEBUFFER,
    GLX_RED_SIZE, 4,
    GLX_GREEN_SIZE, 4,

54
GLX_BLUE_SIZE, 4,
GLX_DEPTH_SIZE, 16,
None );

GLWindow GLWin; GLfloat rotTri, rotQuad;

/* function called when our window is resized (should only happen in
window mode) */ void resizeGLScene(unsigned int width, unsigned int
height) {
    if (height == 0) /* Prevent A Divide By Zero If The Window Is Too Small */
        height = 1;
    glViewport(0, 0, width, height); /* Reset The Current Viewport And Perspective */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, (GLfloat)width / (GLfloat)height, 0.1f, 100.0f);
    glMatrixMode(GL_MODELVIEW);
}

/* general OpenGL initialization function */ int initGL(GLvoid) {
    glShadeModel(GL_SMOOTH);
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glClearDepth(1.0f);
    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LEQUAL);
    glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    /* we use resizeGLScene once to set up our initial perspective */
    resizeGLScene(GLWin.width, GLWin.height);
    /* Reset the rotation angles of our objects */
    rotTri = 0;
    rotQuad = 0;
    quadratic = gluNewQuadric();
    gluQuadricNormals(quadratic, GLU_SMOOTH);
    gluQuadricTexture(quadratic, GL_TRUE);
    gluQuadricDrawStyle(quadratic, GLU_FILL);

    glFlush();
    return True;
}

/* function to release/destroy our resources and restoring the old
desktop */ GLvoid killGLWindow(GLvoid) {
    if (GLWin.ctx)
    {
        if (!glXMakeCurrent(GLWin.dpy, None, NULL))

55
Could not release drawing context.

glXDestroyContext(GLWin.dpy, GLWin.ctx);
GLWin.ctx = NULL;

/* switch back to original desktop resolution if we were in fs */
if (GLWin.fs)
{
    XF86VidModeSwitchToMode(GLWin.dpy, GLWin.screen, &GLWin.deskMode);
    XF86VidModeSetViewPort(GLWin.dpy, GLWin.screen, 0, 0);
}
XCloseDisplay(GLWin.dpy);

/* this function creates our window and sets it up properly */
/*
FIXME: bits is currently unused */
Bool createGLWindow(char* title,
int width, int height, int bits,
    Bool fullscreenflag)
{
    XVisualInfo *vi;
    Colormap cmap;
    int dpyWidth, dpyHeight;
    int i;
    int glxMajorVersion, glxMinorVersion;
    int vidModeMajorVersion, vidModeMinorVersion;
    XF86VidModeModeInfo **modes;
    int modeNum;
    int bestMode;
    Atom wmDelete;
    Window winDummy;
    unsigned int borderDummy;

    GLWin.fs = fullscreenflag;
    /* set best mode to current */
    bestMode = 0;
    /* get a connection */
    GLWin.dpy = XOpenDisplay(0);
    GLWin.screen = DefaultScreen(GLWin.dpy);
    XF86VidModeQueryVersion(GLWin.dpy, &vidModeMajorVersion,
        &vidModeMinorVersion);
    printf("XF86VidModeExtension-Version %d.%d\n", vidModeMajorVersion,
        vidModeMinorVersion);
    XF86VidModeGetAllModeLines(GLWin.dpy, GLWin.screen, &modeNum, &modes);
/* save desktop-resolution before switching modes */
GLWin.deskMode = *modes[0];

/* look for mode with requested resolution */
for (i = 0; i < modeNum; i++)
{
    if ((modes[i]->hdisplay == width) && (modes[i]->vdisplay == height))
    {
        bestMode = i;
    }
}

/* get an appropriate visual */
vi = glXChooseVisual(GLWin.dpy, GLWin.screen, attrListDbl);
if (vi == NULL)
{
    vi = glXChooseVisual(GLWin.dpy, GLWin.screen, attrListSgl);
    GLWin.doubleBuffered = False;
    printf("Only Singlebuffered Visual!\n");
}
else
{
    GLWin.doubleBuffered = True;
    printf("Got Doublebuffered Visual!\n");
}

/* create a GLX context */
GLWin.ctx = glXCreateContext(GLWin.dpy, vi, 0, GL_TRUE);

/* create a color map */
cmap = XCreateColormap(GLWin.dpy, RootWindow(GLWin.dpy, vi->screen),
                       vi->visual, AllocNone);
GLWin.attr.colormap = cmap;
GLWin.attr.border_pixel = 0;

if (GLWin.fs)
{
    XF86VidModeSwitchToMode(GLWin.dpy, GLWin.screen, modes[bestMode]);
    XF86VidModeSetViewPort(GLWin.dpy, GLWin.screen, 0, 0);
    dpyWidth = modes[bestMode]->hdisplay;
    dpyHeight = modes[bestMode]->vdisplay;
    printf("Resolution %dx%d\n", dpyWidth, dpyHeight);
    XFree(modes);

    /* create a fullscreen window */
    GLWin.attr.override_redirect = True;
GLWin.attr.event_mask = ExposureMask | KeyPressMask | ButtonPressMask |
  StructureNotifyMask | ButtonMotionMask | KeyReleaseMask | ButtonReleaseMask | PointerMotionMask;
GLWin.win = XCreateWindow(GLWin.dpy, RootWindow(GLWin.dpy, vi->screen),
  0, 0, dpyWidth, dpyHeight, 0, vi->depth, InputOutput, vi->visual,
  CWBorderPixel | CWColormap | CWEventMask | CWOverrideRedirect,
  &GLWin.attr);
XWarpPointer(GLWin.dpy, None, GLWin.win, 0, 0, 0, 0, 0);
XMapRaised(GLWin.dpy, GLWin.win);
XGrabKeyboard(GLWin.dpy, GLWin.win, True, GrabModeAsync,
  GrabModeAsync, CurrentTime);
XGrabPointer(GLWin.dpy, GLWin.win, True, ButtonPressMask,
  GrabModeAsync, GrabModeAsync, GLWin.win, None, CurrentTime);
}
else
{
  /* create a window in window mode*/
  GLWin.attr.event_mask = ExposureMask | KeyPressMask | ButtonPressMask |
    StructureNotifyMask | ButtonMotionMask | KeyReleaseMask | ButtonReleaseMask | PointerMotionMask;
  GLWin.win = XCreateWindow(GLWin.dpy, RootWindow(GLWin.dpy, vi->screen),
    0, 0, width, height, 0, vi->depth, InputOutput, vi->visual,
    CWBorderPixel | CWColormap | CWEventMask, &GLWin.attr);
  /* only set window title and handle wm_delete_events if in windowed mode */
  wmDelete = XInternAtom(GLWin.dpy, "WM_DELETE_WINDOW", True);
  XSetWMProtocols(GLWin.dpy, GLWin.win, &wmDelete, 1);
  XSetStandardProperties(GLWin.dpy, GLWin.win, title,
    title, None, NULL, NULL, NULL);
  XMapRaised(GLWin.dpy, GLWin.win);
}
/* connect the glx-context to the window */
glxMakeCurrent(GLWin.dpy, GLWin.win, GLWin.ctx);
XGetGeometry(GLWin.dpy, GLWin.win, &winDummy, &GLWin.x, &GLWin.y,
    &GLWin.width, &GLWin.height, &borderDummy, &GLWin.depth);
printf("Depth %d\n", GLWin.depth);
if (glXIsDirect(GLWin.dpy, GLWin.ctx))
  printf("Congrats, you have Direct Rendering!\n");
else
  printf("Sorry, no Direct Rendering possible!\n");
initGL();
return True;

/*********************************************************/

58
float zoomOut = 80.0;

/* Firstly our brick structure/record type */

struct brick{
    bool rotated;
    int size;
    int x;
    int y;
    int z;
};

/* Then a linked list data structure for keeping
our bricks in. We could use an array, but it's
nicer to be able to have arbitrarily many bricks
without horid array resizes. Note we declare a
global instance 'lList' here, this will hold all
our bricks. */

typedef struct ns {
    struct brick *data;
    struct ns *next;
} node;

node *lList = NULL;

node *list_add(node **p, struct brick *i) {
    node *n = malloc(sizeof(node));
    n->next = *p;
    *p = n;
    n->data = i;
    return n;
}

void list_remove(node **p) {
    if (*p != NULL) {
        node *n = *p;
        *p = (*p)->next;
        free(n);
    }
}


/* A functional programming style map function*/
void map(node *n, int (*fp)(struct brick *i)) {
    while (n != NULL) {
        fp ((n->data));
        n = n->next;
    }
}

Bool inXY(int x, int y, struct brick *i, int window) {
    /*here we adjust our x and y positions to account for how far
     * zoomed out we are. */

    if (window == 0) {
        x = x*(320/(zoomOut-i->z));
        y = y*(320/(zoomOut-i->z));
        if (i->rotated) {
            return (
                ((i->x - 1) < x) &&
                ((i->x + 1) > x) &&
                ((i->y - 2) < y) &&
                ((i->y + 2) > y)
            );
        } else {
            return (
                ((i->x - 2) < x) &&
                ((i->x + 2) > x) &&
                ((i->y - 1) < y) &&
                ((i->y + 1) > y)
            );
        }
    }
    else {
        if (window == 1) {
            x = x*(320/(zoomOut+i->y));
            y = y*(320/(zoomOut+i->y));
            if (i->rotated) {
                return (
                    ((i->x - 1) < x) &&
                    ((i->x + 1) > x) &&
                    ((i->z - 1) < y) &&
                    ((i->z + 1) > y)
                );
            }
        }
    }
}
else{
    return (  
          ((i->x - 2) < x) &&  
          ((i->x + 2) > x) &&  
          ((i->z - 1) < y) &&  
          ((i->z + 1) > y)  
    );
}

if (window == 2){
    x = x*320/(zoomOut+i->x);  
    y = y*320/(zoomOut+i->x);  
    if (i->rotated){
        return (  
          ((i->y - 1) < x) &&  
          ((i->y + 1) > x) &&  
          ((i->z - 2) < y) &&  
          ((i->z + 2) > y)  
    );
    }else{
        return (  
          ((i->y - 1) < x) &&  
          ((i->y + 1) > x) &&  
          ((i->z - 1) < y) &&  
          ((i->z + 1) > y)  
    );
    }
}

int whichWindow(int x,int y){
    if (x>320)  
        if (y<240) return 0;  
        else return 1;  
    if ((x<320)&&(y>240)) return 2;  
    else return 3;
}

struct brick *list_search(node *n, int x, int y, int window) {  
  /* currentMaxZ is only 'z' in window 0 (because the others are rotated).*/  
  struct brick *currentMaxZ = NULL;  
  // Code here...
switch(window){
case 0:
    while (n != NULL) {
        if (inXY (x,y,(n->data),window)){
            if((!currentMaxZ) || ((n->data)->z > currentMaxZ->z)) {
                currentMaxZ = (n->data);
            }
        }
        n = n->next;
    }
    return (currentMaxZ);

case 1:
    while (n != NULL) {
        if (inXY (x,y,(n->data),window)){
            if((!currentMaxZ) || ((n->data)->y > currentMaxZ->y) ){
                currentMaxZ = (n->data);
            }
        }
        n = n->next;
    }
    return (currentMaxZ);

case 2:
    while (n != NULL) {
        if (inXY (x,y,(n->data),window)){
            if((!currentMaxZ) || ((n->data)->x > currentMaxZ->x) ){
                currentMaxZ = (n->data);
            }
        }
        n = n->next;
    }
    return (currentMaxZ);
}

return NULL;
}

int drawGLBrick(struct brick *lb);

int drawGLScene(GLvoid){
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glTranslatef(0.0f, 0.0f, -zoomOut);
    62
/* glRotatef(rotQuad, 1.0f, 0.0f, 0.0f); */

glViewport(0,0,320,240);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -zoomOut);
glRotatef(-90,1,0,0);
glRotatef(90,0,0,1);
map(lList,drawGLBrick);

glViewport(320,240,320,240);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -zoomOut);
map(lList,drawGLBrick);

glViewport(320,0,320,240);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -zoomOut);
glRotatef(-90,1,0,0);
map(lList,drawGLBrick);

glViewport(0,240,320,240);
glLoadIdentity();
glTranslatef(0.0f, 0.0f, -zoomOut-20);
glRotatef(-60,1,0,0);
glRotatef(-60,0,0,1);
map(lList,drawGLBrick);

/* rotTri += 0.2f;
rotQuad -= 0.15f; */
if (GLWin.doubleBuffered)
{
    glXSwapBuffers(GLWin.dpy, GLWin.win);
}
return True;
}

/* This function can draw an arbitrary brick, of given dimensions*/

int drawGLBrick(struct brick *lb) {
    int xt,yt,zt;
    if (lb->rotated){
        glRotatef(90,0,0,1);
        /* We’ve rotated 90 degrees about z, so as not to duplicate
           * the code fo rendering a brick we adjust it’s position with
           */
    }
}
glBegin(GL_QUADS);
    /* top of cube */
    glColor3f(0.0f, 1.0f, 0.0f);
    glVertex3f(lb->x+2.0f, lb->y+1.0f, lb->z-1.0f);
    glVertex3f(lb->x-2.0f, lb->y+1.0f, lb->z-1.0f);
    glVertex3f(lb->x-2.0f, lb->y+1.0f, lb->z+1.0f);
    glVertex3f(lb->x+2.0f, lb->y+1.0f, lb->z+1.0f);
    /* bottom of cube */
    glColor3f(1.0f, 0.5f, 0.0f);
    glVertex3f(lb->x+2.0f, lb->y+-1.0f, lb->z+1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+-1.0f, lb->z+1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+-1.0f, lb->z+-1.0f);
    glVertex3f(lb->x+2.0f, lb->y+-1.0f, lb->z+-1.0f);
    /* front of cube */
    glColor3f(1.0f, 0.0f, 0.0f);
    glVertex3f(lb->x+2.0f, lb->y+1.0f, lb->z+1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+ 1.0f, lb->z+1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+-1.0f, lb->z+1.0f);
    glVertex3f(lb->x+2.0f, lb->y+-1.0f, lb->z+1.0f);
    /* back of cube */
    glColor3f(1.0f, 1.0f, 0.0f);
    glVertex3f(lb->x+-2.0f, lb->y+1.0f, lb->z+-1.0f);
    glVertex3f(lb->x+2.0f, lb->y+1.0f, lb->z+-1.0f);
    glVertex3f(lb->x+2.0f, lb->y+-1.0f, lb->z- -1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+-1.0f, lb->z+-1.0f);
    /* right side of cube */
    glColor3f(1.0f, 0.0f, 1.0f);
    glVertex3f(lb->x+2.0f, lb->y+1.0f, lb->z+-1.0f);
    glVertex3f(lb->x+2.0f, lb->y+1.0f, lb->z+1.0f);
    glVertex3f(lb->x+2.0f, lb->y+-1.0f, lb->z+1.0f);
    glVertex3f(lb->x+2.0f, lb->y+-1.0f, lb->z+-1.0f);
    /* left side of cube */
    glColor3f(0.0f, 1.0f, 1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+1.0f, lb->z+1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+1.0f, lb->z+-1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+-1.0f, lb->z+1.0f);
    glVertex3f(lb->x+-2.0f, lb->y+-1.0f, lb->z+-1.0f);
glVertex3f(lb->x+-2.0f, lb->y+-1.0f, lb->z+-1.0f);
glVertex3f(lb->x+-2.0f, lb->y+ -1.0f, lb->z+1.0f);
glEnd();
glTranslatef(lb->x+1.0f, lb->y+0.5f, lb->z+1.0f);
gluCylinder(quadratic, 0.2f, 0.2f, 0.2f, 16, 16);
gluDisk(quadratic, 0.0f, 0.2f,16,16);
glTranslatef(0.0f, 0.0f, -0.2f);
glTranslatef(-1.0f, 0.0f, 0.0f);
gluCylinder(quadratic, 0.2f, 0.2f, 0.2f, 16, 16);
gluDisk(quadratic, 0.0f, 0.2f,16,16);
glTranslatef(0.0f, 0.0f, -0.2f);
glTranslatef(-1.0f, 0.0f, 0.0f);
gluCylinder(quadratic, 0.2f, 0.2f, 0.2f, 16, 16);
gluDisk(quadratic, 0.0f, 0.2f,16,16);
glTranslatef(0.0f, 0.0f, -0.2f);
65
glTranslatef(0.0f, 0.0f, -0.2f);

glTranslatef(-(lb->x)+1.0f, -(lb->y)+0.5f, -(lb->z)-1.0f);

if (lb->rotated){
    glRotatef(-90,0,0,1);
    lb->y=yt;
    lb->x=xt;
    lb->z=zt;
}

    glEnd();
}

void addBrick(int x, int y, int z, Bool rotated){
    struct brick *brickPtr;
    brickPtr = malloc(sizeof(struct brick));
    brickPtr->x=x;
    brickPtr->y=y;
    brickPtr->z=z;
    brickPtr->rotated=rotated;
    list_add(&lList,brickPtr);
}

int main(int argc, char **argv) {

    XEvent event;
    Bool done;
    Bool movingBrickNow;
    struct brick *brickPtr;
    int w;

    brickPtr = NULL;
    movingBrickNow=False;
    done = False;
    /* default to fullscreen */
    GLWin.fs = False;
    createGLWindow("Lego simulator.", 640, 480, 24, GLWin.fs);

    /* wait for events*/
while (!done)
{
    /* handle the events in the queue */
    while (XPending(GLWin.dpy) > 0)
    {
        XNextEvent(GLWin.dpy, &event);
        switch (event.type)
        {
            case Expose:
                if (event.xexpose.count != 0)
                    break;
                drawGLScene();
                break;
            case ConfigureNotify:
                /* call resizeGLScene only if our window-size changed */
                if ((event.xconfigure.width != GLWin.width) ||
                    (event.xconfigure.height != GLWin.height))
                {
                    GLWin.width = event.xconfigure.width;
                    GLWin.height = event.xconfigure.height;
                    printf("Resize event\n");
                    resizeGLScene(event.xconfigure.width,
                                  event.xconfigure.height);
                }
                break;
            case ButtonPress:
                movingBrickNow = True;
                break;
            case ButtonRelease:
                movingBrickNow = False;
                brickPtr = NULL;
                break;
            case KeyPress:
                if (XLookupKeysym(&event.xkey, 0) == XK_b )
                {
                    brickPtr = malloc(sizeof(struct brick));
                    brickPtr->x=19;
                    brickPtr->y=14;
                    brickPtr->z=-10;
                    brickPtr->rotated=False;
                    list_add(&lList,brickPtr);
                }
                if (XLookupKeysym(&event.xkey, 0) == XK_r && brickPtr)
{ 
  brickPtr->rotated=!(brickPtr->rotated);
}

if (XLookupKeysym(&event.xkey, 0) == XK_Escape) {
  done = True;
}

if (XLookupKeysym(&event.xkey, 0) == XK_minus) {
  zoomOut = zoomOut+1;
}

if (XLookupKeysym(&event.xkey, 0) == XK_plus) {
  zoomOut = zoomOut-1;
}

if (XLookupKeysym(&event.xkey,0) == XK_F1) {
  killGLWindow();
  GLWin.fs = !GLWin.fs;
  createGLWindow("NeHe’s Solid Objects Tutorial",
                  640, 480, 24, GLWin.fs);
}

break;

switch (w){
  case 0:
    brickPtr = list_search(
      lList,-((event.xmotion.x-480)),
      ((120-event.xmotion.y)), w
68
break;
    case 1:
    brickPtr = list_search(lList, ((event.xmotion.x-480)),
                          ((360-event.xmotion.y)), w);
    break;
    case 2:
    brickPtr = list_search(lList, ((event.xmotion.x-160)),
                          ((360-event.xmotion.y)), w);
    break;
}
}
if (movingBrickNow && brickPtr)
    w = (whichWindow(event.xmotion.x, event.xmotion.y));
    switch (w){
    case 0:
        brickPtr->x = (event.xmotion.x-480)/(320*1/(zoomOut-brickPtr->z));
        brickPtr->y = -(event.xmotion.y-120)/(320*1/(zoomOut-brickPtr->z));
        break;
    case 1:
        brickPtr->x = (event.xmotion.x-480)/(320*1/(zoomOut+brickPtr->z));
        brickPtr->z = -(event.xmotion.y-360)/(320*1/(zoomOut+brickPtr->z));
        break;
    case 2:
        brickPtr->y = -(event.xmotion.x-160)/(320*1/(zoomOut+brickPtr->z));
        brickPtr->z = -(event.xmotion.y-360)/(320*1/(zoomOut+brickPtr->z));
        break;
    }
    break;
    default:
        break;
    }
}
drawGLScene();
}
Appendix H

Sample Screenshots
This section contains some sample screen shots of the produced system that demonstrate some of the features that were implemented.
Figure H.1: Demonstrating two bricks created in the environment
Figure H.2: Demonstrating a rotated brick
Figure H.3: Demonstrating that bricks are not affected by collisions
Figure H.4: Demonstration of bricks not being affected by gravity
Figure H.5: Demonstrating bricks can be moved in all directions of the environment