DUBLIN INSTITUTE of TECHNOLOGY

Institiúid Teicneolaíochta Bhaile Átha Cliath

Physical Realism in Augmented Reality

David B. Beaney

Degree in Computer Science
B.Sc. (Computer Science)
Dublin Institute of Technology

Supervisor: Dr. Brian Mac Namee
DT228 / 2009

Abstract

The purpose of this project is to create an augmented reality system that is believable. Augmented reality is a relatively new field of computer research which investigates the augmentation of video feeds of the real-world with computer-generated virtual objects that are integrated into the video feed in real-time. When (not if) augmented reality technology becomes common place, believability will be important so that the illusion is not broken. This project focuses on using physics to enhance realism, and utilises a physics engine for such ends. The end result of this project is an augmented reality system wherein both real and virtual objects are involved in believable physics based interactions. These interactions between real and virtual objects help to create a convincing augmented reality environment. Results gathered from psychometric experiments performed as part of the project show that users did find the system believable. The implications are that this is the right direction to take in creating believable augmented reality systems.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

David B. Beaney
2009

Acknowledgements

First of all, I would like to thank the staff in DIT Kevin Street, in particular Dr. Brian Mac Namee for introducing me to the exciting world of augmented reality, a subject which I will forever be obsessed about from now on.

I would also like to thank Dr. rer. nat. Daniel R. Berger of the Max Planck Institute for Biological Cybernetics, for allowing me the use of his NXT remote control software.

Finally, I want to thank my family and friends for their continued support during the course of the year.

Contents

List of Tables

List of Figures

1. Introduction

1.1. Augmented Reality

Augmented reality (AR) is a relatively new field of computer research which investigates the augmentation of video feeds of the real-world with computer-generated virtual objects that are integrated into the video feed in real-time. Common examples include the advertisements on rugby pitches and head-up displays in fighter jets. As AR systems are becoming increasingly popular, it is important that the virtual 3D objects displayed in such systems act in a realistic and expected manner.

1.2. Motivation

Much advanced research in AR is focused on using techniques from image processing to achieve robust and reliable *registration*. Registration is necessary so that the virtual augmentations are displayed correctly in the scene. A simple example: to display a virtual object on a table top (say a teapot), the distance from the camera and the orientation of the table must first be determined. This is quite obviously required, as otherwise the teapot might appear to be floating at a point in midair, or appear to be inside the table.

Although this focus on registration had led to AR applications becoming more impressive with each passing year, there are other aspects to creating successful AR applications that need similar attention. This project is focused on *believability* and *realism* in augmented reality systems. Real-time AR has not traditionally taken into account the physical effects of real objects on the the virtual objects that are superimposed. Such issues have recently become more important, and some progress has been made by such companies as Total Immersion.[1]

Physics engines have widespread use in computer games, but they are useful for any interactive media that requires realistic motion and collision of objects. This project aims to use a physics engine to enhance the AR, thus making it more believable.

1.3. Objectives

1.3.1. What is the project?

Figure 1 shows the initial idea for the end product of this project. The forklift would be a real object, while the boxes (crates) shown would be the virtual *augmentations*, which are rendered onto the camera feed. This illustrates what the user might actually see, comprising a mixture of real and virtual objects, interacting realistically together.

The forklift scenario was chosen simply to demonstrate the the realism and believability that AR can achieve. In some ways the project can be considered as a game, but it was not intended as such and should not be treated as such. The forklift scenario is a purely a contrivance, used to exhibit the real and virtual object interactions. The forklift represents a real object that can interact with the virtual boxes.

[1] http://www.t-immersion.com/

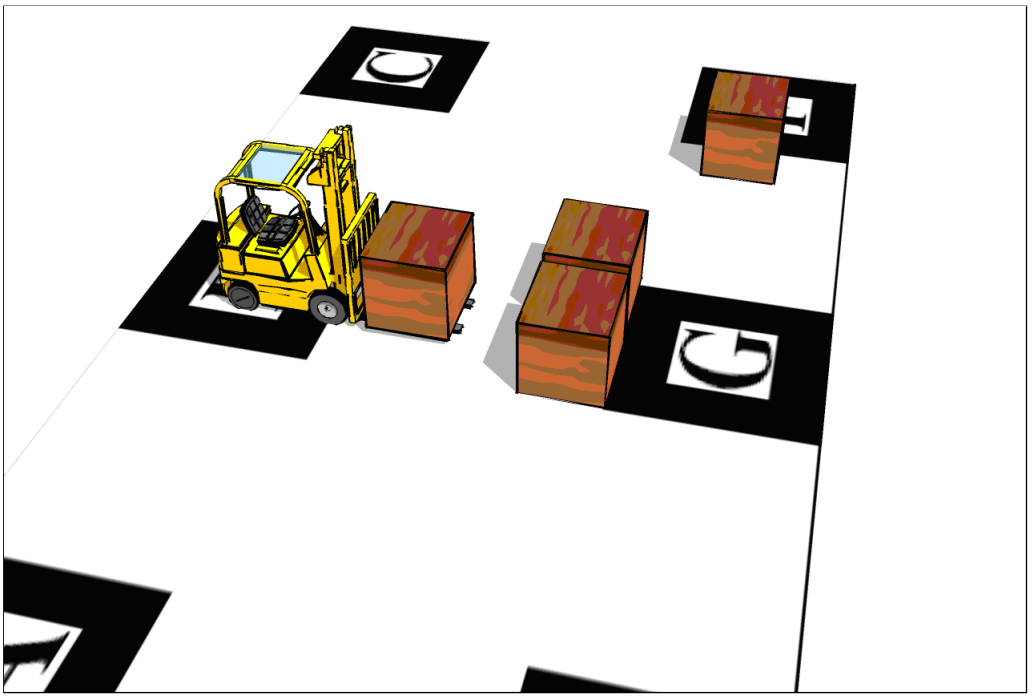


Figure 1: A mock up of the final system

1.3.2. High-Level Objectives

The original high-level objectives for this project were as follows:

- The project was to make use of an ARvision-3D augmented reality headset, and would therefore allow the wearer to walk around the virtual interactions and view them from different angles.
- The AR environment would be limited to, and conducted on a table that would feature suitable *fiduciary markers*. These markers will aid in the alignment of the virtual and real objects in respect to each other (known as registration).
- A toolkit such as ARToolkit or ARTag would assist in the recognition of these markers.
- The interactions of the real and virtual objects would utilise a physics engine.
- The real objects will visually *occlude* the virtual ones.
- Fiduciary markers will not be used to recognise the real objects that are to be interacted with, i.e., the forklift. Other techniques of pattern recognition and computer vision will be used instead.

The final objective was not achieved, for various reasons alluded to in section 1.2. It became clear early on in the project that this objective was overly ambitious; see section 4.9 for details.

1.4. Applications

Especially in entertainment, there is nothing worse than breaking the illusion. It is widely believed that in computer games, for example, realism is essential to keep a person immersed in the make-believe world [6]. Although it is not clear that progress towards ultra-realism is desirable (see Masahiro Mori's uncanny valley [7]), it is also clear that objects do not ordinarily go through walls (unless they are bullets). The perspicacious use of a physics engine, therefore, can help with realism.

A specific example of an ideal application is that of school physics experiments. Science equipment is usually expensive, and while currently AR technology may be even more expensive; the number of experiments that could be done with an AR headset is immense. For example experiments involving force, acceleration, momentum etc.

1.5. Dissertation Roadmap

The remainder of this document will proceed as follows.

- Section 2 will describe some of the background literature in the relevant fields of augmented reality, physics simulation and computer vision. More detailed topics covered include: mixed reality concepts, registration techniques, rendering, and user interfaces.
- Section 3 will discuss the methodology and design of the project. This section also features a discussion of the various technologies used.
- Section 4 describes the implementation of the project. Each of a series of prototypes developed is given a section.
- Section 5 will outline the testing procedures and the results of testing.
- Section 6 will discuss the way in which this project was evaluated and the outcomes of the evaluation.
- Section 7 will conclude this dissertation with closing remarks, reflecting on unforeseen consequences and possible future work.

2. Background

This section of the document will review the literature in the fields relevant to this project. Firstly, a short introduction to augmented reality is given and then topics such as rendering and registration are reviewed. Other pertinent topics such as physics engines are discussed. This chapter aims to provide a background to augmented reality to someone not previously familiar with it.

2.1. Introduction to Augmented Reality

Augmented reality (AR) is a relatively new field of computer research which investigates the augmentation of views of the real-world with computer-generated virtual objects, in real time.

Ivan Sutherland is often considered to be the first to have considered virtual reality in 1965, when he described what would be the ultimate display [8]:

> The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal. With appropriate programming, such a display could literally be the Wonderland into which Alice walked.

Although he was describing virtual reality, augmented reality shares many of the same problems that need to be solved.

Sutherland was the first to develop a functioning Head-Mounted Display (HMD) [9], through which the user saw both the real world and images displayed on cathode ray tubes. This was achieved using half-silvered mirrors. Hence this was the first HMD used for augmented reality.

Ronald Azuma is usually credited with a defining the field of augmented reality [10]. He defines AR systems as having the following characteristics:

1. Combine real and virtual
2. Interactive in real time
3. Registered in 3-D

Sutherland goes on to note that it is the second two aspects of AR systems that set them apart from such media as films:

> Films like “Jurassic Park” feature photorealistic virtual objects seamlessly blended with a real environment in 3-D, but they are not interactive media. 2-D virtual overlays on top of live video can be done at interactive rates, but the overlays are not combined with the real world in 3-D.

Current applications of AR are wide-ranging and varied, touching areas from military use [11] to entertainment. In industry, augmented reality systems can be used to aid manufacturers, for example see [12]. In entertainment AR games are being widely researched, for example, ARQuake [13].

As mobile phones become more powerful, mobile applications of AR are on the rise [14]. Mobile solutions are by definition suited to navigation applications. This area is likely to be the first one in which AR will become commonplace; most mobile phones possess impressive and increasing figures of instructions per second, and most new mobile phones possess a digital camera.

Augmented reality may have great potential in the area of ubiquitous computing in the future, where an augmented reality system would offer a very natural way of achieving human-computer interaction.

Augmented reality is a form of mixed reality. Augmented reality can be placed on a continuum between the real-world at one end, and a completely virtual world at the other. This was first described by Milgram [1] in 1994. Figure 2 shows Milgram's Reality-Virtuality Continuum which shows the spectrum of realities. AR sits close to the left hand side of the spectrum; AR experiences tend not to completely block out the real world.

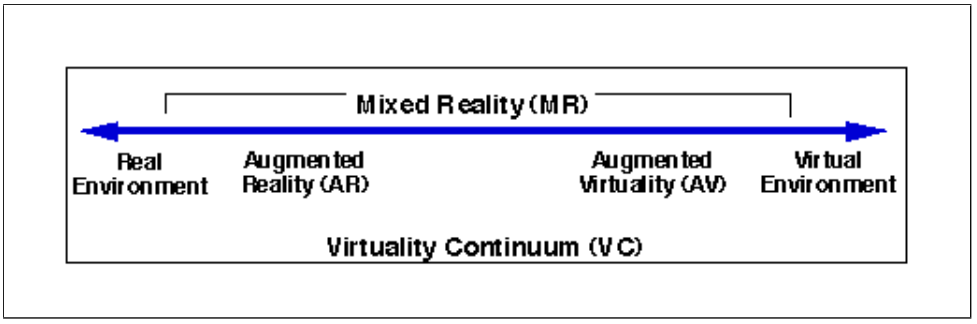


Figure 2: Milgram's Reality-Virtuality Continuum [1]

Milgram also suggested three axes for categorising mixed reality systems. They are: Reproduction Fidelity, Extent of Presence Metaphor and Extent of World Knowledge.

The first axis, Reproduction Fidelity, refers to the realism of the computer-generated images. Since augmented reality is always real-time, augmented reality systems to date tend not to be photorealistic, but rather simple. The Extent of Presence Metaphor relates to the level of immersion the user experiences while using the system. The last axis, Extent of World Knowledge, is important when it comes to registration, as augmented reality systems need to properly align the 3D augmentations with pre-existing real objects. This project sits low on the Reproduction Fidelity axis, as realistic computer-generated imagery is not used. The Extent of Presence for this project is also fairly low, as the virtual objects used are not particularly prominent. However, the Extend of World Knowledge is high, as a lot of information about the world is gathered.

2.2. How Augmented Reality Works

Figure 3, below, shows the generic structure of a typical augmented reality application. At stage one; any initialisation necessary is carried out. This could be such things as initialising the video capture and reading in fiduciary marker pattern files. Stage two is the main loop; this is continually repeated until the application quits. Other events may occur in this main loop, such as keyboard and mouse handling. Stage three is the shutdown stage; this is performed once at the end of the application.

The main loop is the most important part here; it can be broken into four main stages as in figure 3. These stages are generic and are applicable to any AR system, being independent of specific implementation details:

1. Capture a video frame(s) from the one or more camera sources.
2. Analyse the captured image data for certain criteria and features.
3. Use the image analysis to determine the camera position and orientation relative to a known starting point. Depending on the complexity of the system, this stage could be full 3D scene reconstruction [15].
4. Using the position and orientation data from above, the virtual objects can be rendered correctly in the scene.

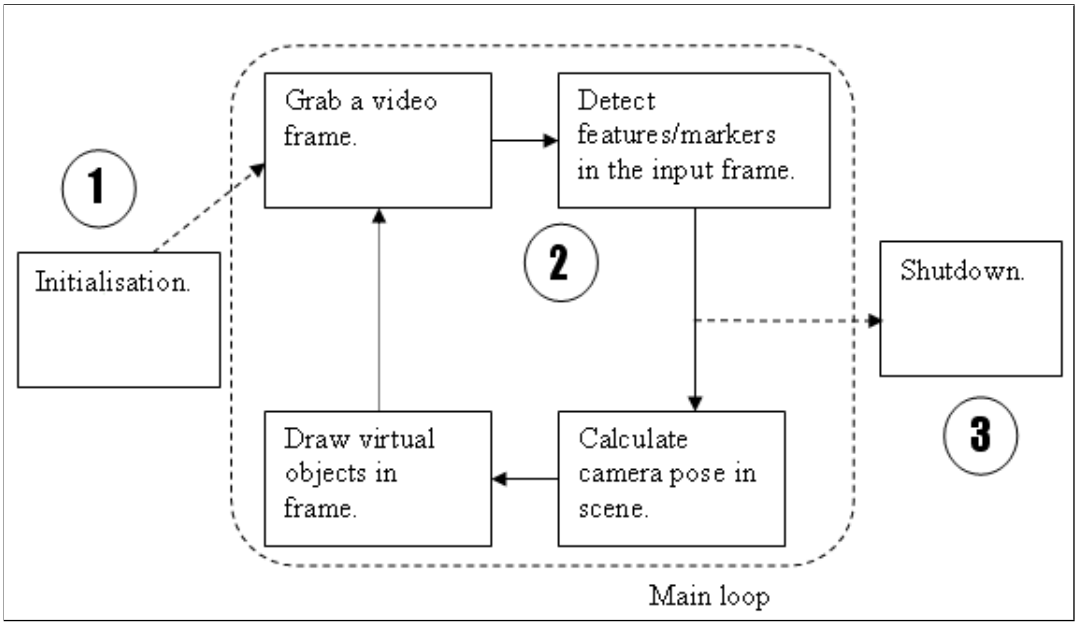


Figure 3: Structure of a generic augmented reality application

2.3. Registration

As alluded to above, virtual objects must be precisely placed into a scene if they are going to appear to be real. The process of *registration* consists of two parts: determining the coordinate system of scene as viewed by the camera; and placing the virtual objects correctly in the scene, given the coordinate system.

As Azuma suggested [16], even small errors in registration can easily be detected by humans in an augmented reality system. Errors in registration immediately reduce or eliminate any immersive qualities that the system may have had.

Registration is one of the fundamental problems to be solved in AR, especially in unprepared environments. The most cutting edge research into registration is being carried

out using advanced computer vision techniques such as object recognition and scene reconstruction [17, 18]. Because of its ease, much research relies on the solid basis of fiduciary marker technology, created by Hirokazu Kato [19].

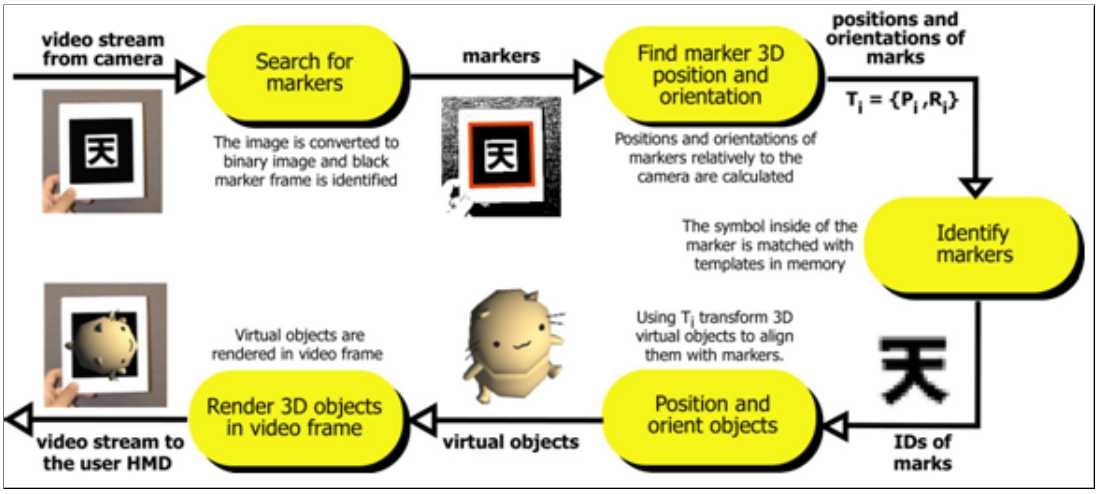


Figure 4: ARToolkit tracking [2]

Figure 4 shows the way in which AR registration is achieved. Once the markers are found, their positions and orientations can be determined relative to the camera. The markers have unique identifiers so that multiple markers can be used. With this position and orientation information, the virtual objects can be drawn in the correct way.

2.3.1. Computer Vision and registration

Fiduciary markers usually consist of a square, with a pattern inside. The square shape makes calculating the pose of the camera in the scene easier. The benefits of using such markers are that registration is quick in terms of processing power, and also that detection is quick and robust [20]. Figure 5 shows the default fiduciary marker used in the ARToolkit SDK, which is used in this project. The algorithm used searches for the black square, which is used in camera pose calculations. The pattern inside the square, in this case “Hiro”, is used as a unique identifier when many tags may be visible. Kato et al describes the way in which these fiduciary markers are tracked. An overview of the process is described by the following stages:

1. Threshold the input image.
2. Extract regions whose outlines can be fitted by four line segments.
3. Parameters of these four line segments are analysed and compared to a template (the template is a representation of the marker).

However, there are other much more sophisticated techniques, which attempt to eliminate the use of fiduciary markers and use natural features instead. A widespread technique is the use of optical flow. Optical flow is the apparent motion of objects in a visual scene cause by the movement of the viewer. Features of interest are picked out of a scene and

tracked continuously. Optical flow can be used to varying degrees of success in unprepared environments. Xu et al [3] is an example of progress in this area. Figure 6 shows optical flow in action.

Figure 5: Default ARToolkit tag

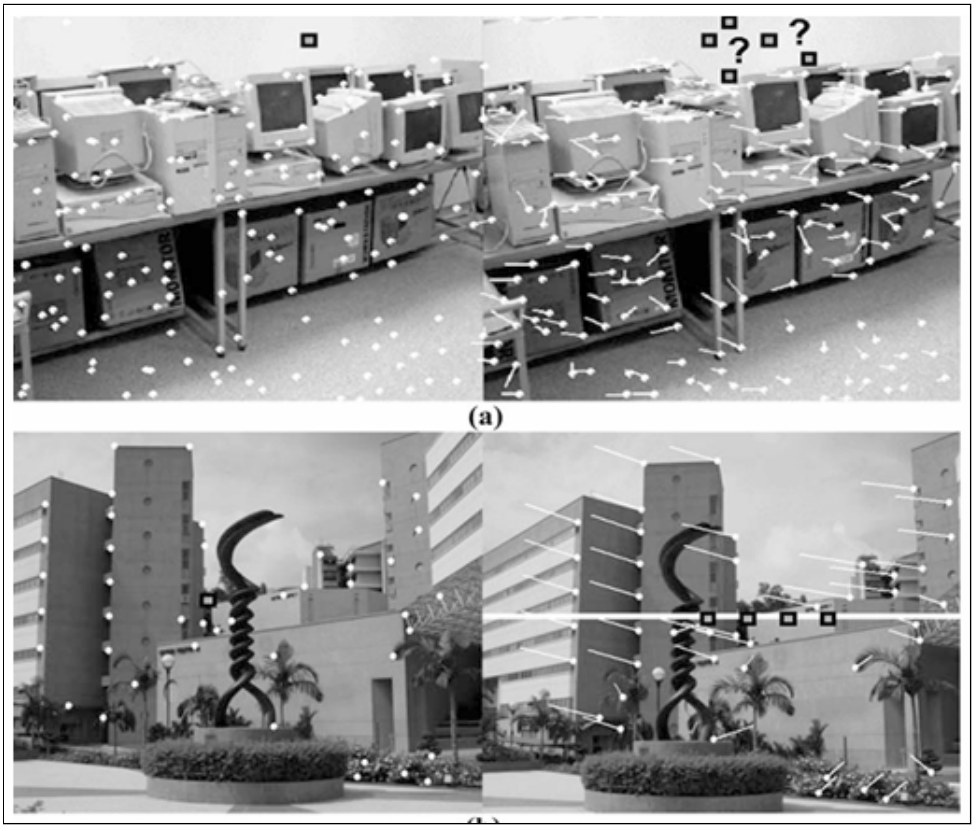


Figure 6: Optical flow, from [3]

2.3.2. Non-computer vision methods of registration

Other ways of achieving registration are usually based on some sort of sensor hardware. For example Neumann uses compasses [21], and Satoh et al use high precision gyroscopes [22]. These techniques rely on the calculation of the relative motion of the camera within the scene.

2.4. User Interfaces

There are five main approaches that are currently used to view 3D augmentations to the real world:

- Magic lens - a mobile device displays the scene from its camera with graphics overlaid. These were first described by Bier [23].
- Magic Mirror - the user stands in front of a screen and camera; and augmentations are applied to the user to change their appearance [24].
- Head mounted display (HMD) - the user wears a headset and augmented graphics are displayed directly onto what the user can see.
- A user looks at an augmented camera feed on a monitor. This is the most common way AR is achieved.
- Spatial augmented reality - the use of projectors to spatially align virtual projections in a real room or other space [25].

The magic lens approach is very popular. Mobile devices are powerful enough to run the software needed for object recognition, and most mobile devices now have cameras. Also, this is an attractive alternative to the currently expensive and unwieldy HMD option.

The magic mirror approach is very limited and there does not seem to be many useful applications.

The HMD display is the approach that will be used in this project. A HMD allows for the most immersive experience. There are limitations in the current hardware, such as the aforementioned bulkiness and price; also some HMDs block some of the user's peripheral vision. However, increasing popularity in AR means that this technology is expected to improve fast, and it is speculated that in the future this will be the dominant technology in AR.

Human interaction with augmented reality systems can take several forms. Conventional interfaces can be used, for example the Playstation 3 game "Eye of Judgement"[2].

More typically, AR systems do not lend themselves to interaction using standard interface devices. This is because augmented reality is connected to the real world. The ideal AR system would be mobile, and standard input devices such as mice and keyboards are not compact or particularly easy to carry around. Thus, other more sophisticated techniques have arisen for AR interaction. These techniques will be discussed in the below sections.

[2] http://www.us.playstation.com/PS3/Games/THE_EYE_OF_JUDGMENT

One technique is to use fiduciary markers to interact with an augmented reality system. Kato et al [26] have done substantial research into how intuitive it is to use these markers. Figure 7 shows an example of this interaction.

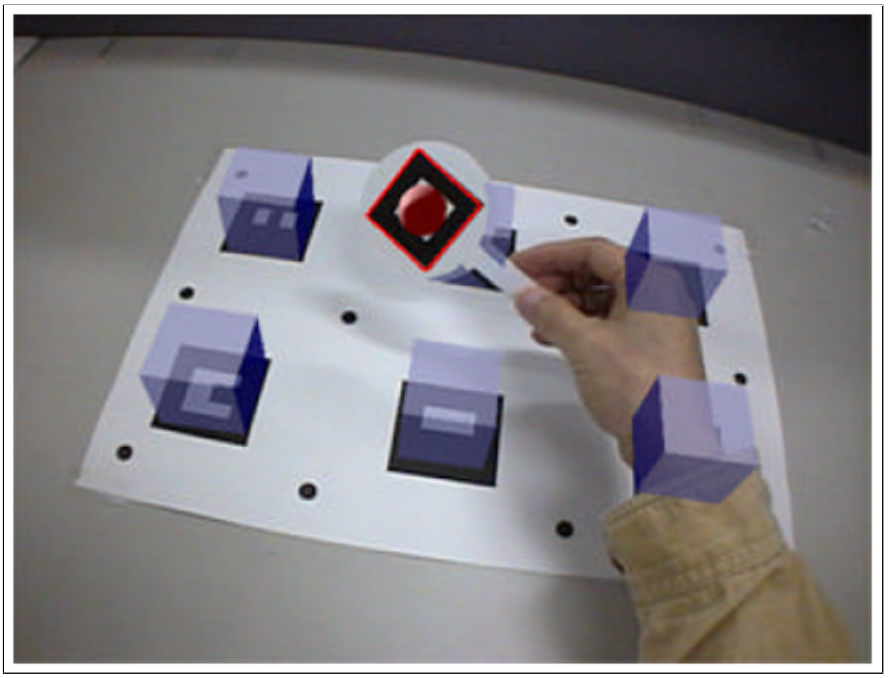


Figure 7: Fiduciary marker interaction

Other, more novel ways of interaction include the hand gestures shown in figure 8, from a system developed by Kojima et al [27]; and also system controlled by eye movements. Techniques such as these are likely to become the normal ways of interaction, given a future proliferation of AR technology. Another promising development for AR control lies in the area of brain-computer interfaces (see for example [28]). Some commercialisation of such interfaces has already begun in the video game market, for example OCZ's Neural Impulse Actuator[3].

2.5. Rendering

The rendering of the virtual objects is important to maintain realism. Although some applications purposely use a "cartoon" stylisation, for example Billinghurst et al [29], usually realism is strived for. Rendering realistically is challenging in augmented reality because it must be in real-time and latency is a problem. Such things as occlusion, realistic shadows, and accurate lighting are all open to a lot of improvement. The problem of rendering ties into the problem of registration; in order to draw virtual objects properly, the effects of the real scene must be taken into account.

Agusanto et al [30], have researched using the illumination of the surrounding environment to achieve photorealistic rendering without resorting to computationally expensive

[3] http://www.ocztechnology.com/products/ocz_peripherals/nia-neural_impulse_actuator

techniques such as ray tracing.

An even more advanced form of rendering is mediated reality. Mediated reality is when real objects are digitally *removed* from a scene. In 2001 Azuma et al, [31] suggested that both photorealism and mediated reality were not currently feasible in real-time. However, with the increasing power of computers and the increasingly advanced computer vision techniques, this will be a short-lived problem.

An important part of rendering is occlusion. In an AR scene it is not always the case that the desired position of the virtual objects is in front of all the real objects. If the desired position is behind real objects, occlusion must be handled, whether this be full or partial. (Occlusion is explained in section 4.8.1).

2.6. Physics and Believability

Real-time augmented reality has not traditionally taken into account the effects of real objects on the virtual objects that are superimposed. Such issues have recently become more important, and some progress has been made by such companies as Total Immersion[4]. The introduction of a physics engine is essential if the simulation is to attain even a modicum of realism. A prominent use of physics engines in augmented reality in games. Some examples of the use of physics engine in an augmented reality games are [32, 33, 34].

A perhaps more useful application is that of education. Kaufmann et al demonstrate the use of a physics engine and AR technology to teach physics through three-dimensional experiments [35].

A physics engine is software that simulates Newtonian physics, calculating the motion of objects, i.e. kinematics. They are widely used in computer games to improve believability and realism. Depending on how advanced the physics engine is, the following could also be modelled, on top of basic kinematics:

- Springs

[4] http://www.t-immersion.com/

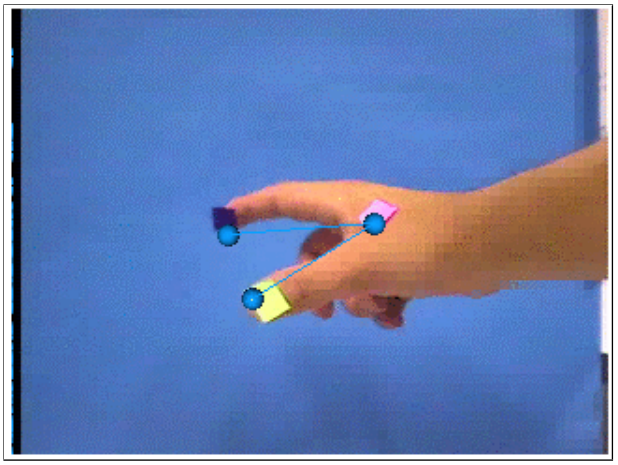


Figure 8: Hand gesture interface

- Hinges
- Fluid dynamics
- Cloth dynamics

A physics engine works by making predictions based on the laws of physics. The mathematical technique of making these predictions is called *integration*. A physics engine integrates the equations of motion for each time step, using numerical integration methods such as the Runge-Kutta method [36].

2.7. Conclusions

The hardware used in augmented reality continues to grow more compact all the time. An interesting new technology is virtual retinal displays, which use lasers to directly draw images directly onto the retina of the eye. Interested readers are directed to Pryor et al [37].

Thanks to the effects of Moore's law [38] and the advances made by researchers in the field, augmented reality software is making great strides. Advances in realism and registration occur all the time. A lot of current research focuses on computer vision and registration techniques that avoid fiduciary markers. However, not much progress has been made with realistic physics interactions in AR applications. There is therefore scope to improve AR applications by making them physically realistic, which is what this project focuses on.

The next section will describe the methodology and design of this project, and will also describe the technologies used in this project.

3. Methodology and Design

This section will outline the methodology and design involved in this project. The evolutionary prototyping model is discussed, and then the high-level architecture is illustrated. The technologies utilised in this project are also described.

3.1. Methodology

3.1.1. Evolutionary Prototyping

In this project evolutionary prototyping model was used as a software development methodology [39].

Evolutionary prototyping, otherwise known as breadboard prototyping, is the process whereby an initial model of a system is created and this model is refined iteratively, after very little requirements analysis. This model is the basis for further improvement. A better model is created after examination of the previous model. The new model is a refined or "evolved" version of the previous one. Each prototype is a more refined version of the first one created. Presenting at the Seventh International Python Conference, Roger Masse explained evolutionary prototyping as follows [40]:

> With evolutionary prototyping, the goal is to “evolve” or “grow” some or all of the programs functionality into the final product. During this process of evolution, a software design emerges from the prototype. Often the design is specific enough to include a final list of modules, classes, functions, and type definitions that the final design is intended to have. Either the final product will be re-implemented in a systems level language or the final product is simply the product of sufficient evolution of design in the prototyping language.

The major advantage of evolutionary prototyping is that it can be done very fast. It is ideal for situations where perhaps the requirements are unknown or unclear. It is also applicable to creating fundamentally new software. Figure 9 shows the four main stages involved in evolutionary prototyping.

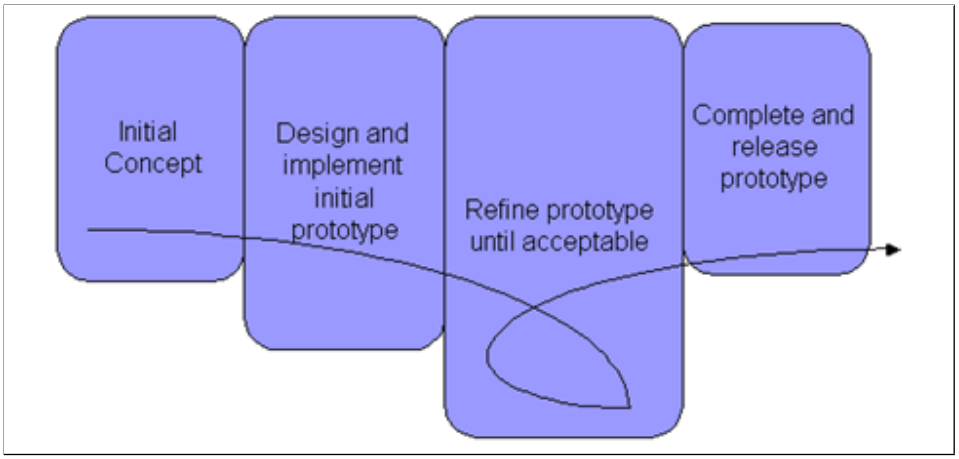


Figure 9: Evolutionary Prototyping, from [4]

These are:

- Definition of basic requirements
- Creation of first working prototype
- Evaluation of working prototype
- Change or elaboration of the prototype, after evaluation

There are problems with using prototyping, such as:

- The projects that are suitable for prototyping are usually high-risk by nature
- Prototyping is used as an excuse for "hacking" to avoid documenting the requirements
- Developing prototypes that are too large, hence spending too long on them. Prototypes are intended to be short-lived, not to be finished products
- Insufficient requirements analysis

UML diagrams are used at certain stages. In particular because of the object-oriented nature of this project, class diagrams are used, for example in section 4.4 of this document.

3.2. High-level Technical Architecture

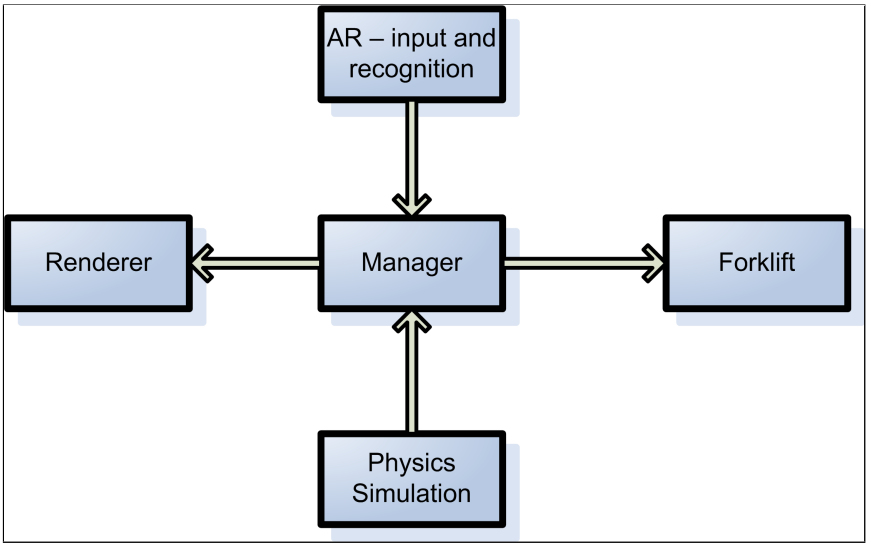


Figure 10: High-level technical architecture

Various technologies will be integrated to perform the functions outlined in figure 10. These technologies will be discussed below. OpenGL will be discussed in relation to rendering, Havok will handle the physics simulation, and the AR functionality will be

mostly handled by ARToolkit. The manager will be entirely novel code, that brings these libraries together harmoniously.

The rendering for this project includes two parts: rendering the images taken from the camera, and rendering the virtual objects that are superimposed on the scene. The rendering of the camera images is achieved using an ARToolkit function call (see section 4.3.2).

The physics module will simulate the forklift, and also the virtual objects in the world (mostly crates). The physics version of the forklift will be aligned with the real forklift.

The AR module is handled mostly by ARToolkit, which registers the fiduciary markers, the position and orientation of which the manager uses as input to the physics module.

The forklift is not a software module, but is the hardware robot. Only one way communication occurs between the manager and the forklift, the manager tells the forklift what to do. The forklift's fiduciary marker position and orientation are communicated to the manager via the AR module.

It should be noted that since evolutionary prototyping is being used, the low level design is done at the prototype stages.

3.3. Technologies

3.3.1. ARToolkit

Probably the most important technology that contributes to this project is ARToolkit. It was originally developed by Dr. Hirokazu Kato, and is being currently supported by the Human Interface Technology Laboratory at the University of Washington.

ARToolkit has been used in many AR projects, as version 2.X of the project is released under the GNU public license[5]. The interested reader can see for example [19, 29].

As suggested by the name, ARToolkit is a low-level toolkit; which provides programmers an interface that abstracts over details such as what camera is being used. ARToolkit is primarily based on fiduciary markers (see section 2.3.1). There are commercial alternatives such as Studierstube tracker [6] and open-source alternatives such as ARTag[7].

The primary reasons for choosing ARToolkit are:

- it is cross-platform
- it is open-source
- it has an active development community / forums
- it is the current leader in free augmented reality software

Zhang et al [41], does a comparative study of various toolkits, of which ARToolkit is one. This helped to evaluate ARToolkit. While ARToolkit is not the most accurate or stable fiduciary marker detection library, it is one of the fastest. This is a trade-off that is unavoidable in augmented reality. The system must perform in real-time, whilst being as accurate as possible.

3.3.2. Havok

Since this project involves physics simulation; a robust physics engine was required. Havok recently released part of their physics engine as a free download. Havok is an industry leader, being used by over 70 games companies around the world.

Alternatives include:

- Commercial
  - Havok (full version)
  - NVIDIA PhysX[8]
- Free
  - Open Dynamics Engine[9]
  - Bullet[10]

[5] http://www.gnu.org/copyleft/gpl.html
[6] http://studierstube.icg.tu-graz.ac.at/
[7] http://www.artag.net/
[8] http://www.nvidia.com/object/nvidia_physx.html
[9] http://www.ode.org/
[10] http://www.bulletphysics.com/Bullet/wordpress/

Havok was chosen for this project because of substantial previous experience with the engine. There was a limitation with using Havok however, in that the free version has only been released for Windows. Therefore it would not be possible to code this project in a cross-platform manner. It is possible, however, that the free version of Havok will be released for other platforms in the future. The superior performance of Havok was seen as being more important than the project being cross-platform. Havok can be run in a multi-threaded mode, which also helped the real-time performance of the system.

3.3.3. OpenGL

The rendering of the virtual objects is handled by OpenGL[11], the free, high performance, cross-platform graphics library. OpenGL is cross-platform. The major alternative to OpenGL is DirectX. OpenGL was chosen simply because of previous familiarity with it. As mentioned in section 4.3.1, previous DIT assignments relied on OpenGL use.

Glut[12], is an OpenGL toolkit that provides a system specific way of quickly writing OpenGL programs. It also provides handy macros for drawing 3-dimensional objects such as spheres and cubes.

3.3.4. Trivision ARVision 3D HMD and Cameras

ARvision-3D[13] is a stereoscopic video see-through binocular head-mounted display (HMD). This was purpose built for augmented reality use. It consists of two cameras and two screens, one for each eye. This HMD is being used because it is available within the DIT.

Two cameras were used throughout the duration of the project. These were a Logitech QuickCam Express, with a maximum resolution of 352 by 288 pixels, and a Logitech QuickCam S7500 with a resolution of 960 by 720 pixels. These are both off the shelf, and available to anyone. It was found that the increased resolution of the S7500 helped to make registration more robust and accurate.

3.3.5. Lejos and Lego Mindstorms NXT

Lejos[14] is a Java Virtual Machine that has been ported to the Lego NXT brick[15]. This was used at an early stage (see section 4.8.2). A model forklift was built using a Lego Mindstorms NXT, controlled by Bluetooth. A Lego Mindstorms NXT is a robot building kit. A programmable NXT brick can control a number of motors and receive input from sensors.

3.3.6. wxWidgets

wxWidgets[16] is a cross-platform toolkit written in C++ for creating graphical user interfaces on all major operating systems. This was used to create the interface for this project.

[11] http://www.opengl.org/
[12] http://www.opengl.org/resources/libraries/glut/
[13] http://www.trivisio.com/index.php/arvision3dhmd
[14] http://lejos.sourceforge.net/
[15] http://mindstorms.lego.com/eng/Stockholm_dest/default.aspx
[16] http://www.wxwidgets.org/

It was chosen initially because it is cross-platform; also because it is "light" in the sense that binaries compiled with it are small. It is also easy to use, which was a factor in deciding to use it over alternatives such as GTK+[17].

3.4. Programming Language Choice

As can be seen from the above sections, many libraries and APIs are being utilised in this project. All but one (Lejos), either are natively written in or have interfaces in C++. ARToolkit is written in C, and Havok is written mainly in C++. Since these are the two most important elements of the project, C++ was the natural choice for use.

3.5. Planned Prototypes and their Associated Risks

The following section details the original predicted prototypes, although of course with evolutionary prototyping these were subject to change in direction throughout the project. The section after the following one outlines the potential risks *originally* identified that could affect the project.

3.5.1. Predicted Prototypes

1. Achieve ARToolkit working and reading images from a USB camera
2. Get ARToolkit and OpenGL working together, drawing images from camera on screen, and drawing a simple OpenGL model on the position of a fiduciary marker
3. Setup environment (table) with fiduciary markers. Modularise code structure
4. Detect forklift (using fiduciary markers)
5. At this stage the AR HMD is integrated into the system to test for suitability early on in the project
6. Integrate Havok into the system. Implement a simple bounding box around the forklift. Virtual objects should behave somewhat realistically with forklift at this stage, although occlusion is not implemented, and the accuracy of collision is crude. At this stage the Lego forklift should have been built
7. This prototype is a refined version of prototype 6, but it will also include occlusion
8. The elimination of fiduciary markers on the forklift
9. A more accurate physics representation of forklift, rather than a simple bounding box
10. This is the final prototype, and will include any secondary features that time allows. Secondary features could include:
    - Other more advanced image processing techniques

[17] `http://www.gtk.org/`

- Depth perception to accurately locate objects in 3D space
- Training the system to recognise and include new objects into the physics simulation

3.5.2. Prototype Risks

This section details the risks that were identified at the beginning of the project.

It was decided that to make things "easy", the table would be covered in white paper, and fiduciary markers would be used liberally. To begin with, no occlusion and no secondary features would be implemented. The most obvious shortcoming at an early stage in the project was the use of a fiduciary tag for the recognition of the forklift.

Foreseeable problems stemmed from the lack of interoperability between the different APIs being utilised. ARToolkit and the head-mounted display (HMD) software, VRMagic, use different internal representations of images and it might have been that conversions were necessary. It was thought that VRMagic would be needed to utilise HMD. This turned out not to be the case (see section 4.6.1), but nevertheless this was one of the predicted risks.

A serious risk with this project was concerning registration. In the first prototypes, registration of the forklift was done using fiduciary markers and ARToolkit. The latest versions of the toolkit has much improved registration with markers, by remembering some of the history of the marker's position and speed etc. However, in the later prototypes it was thought that markers would not be used for registration with the forklift. This also turned out not to be the case (see section 4.9). The risk identified was that without using markers, accurate registration would not occur and the physics simulation would be unstable and chaotic.

It was anticipated that the "main loop" of this project was already going to be very crowded. Havok physics has excellent support for multi threading, but pre-design research had shown none of the current versions of the other libraries had inherent built in support. Since the original plan was to integrate OpenCV, a risk was that the program would not be able to run smoothly (Perhaps multi threading would help in this area.) This was a major risk and design consideration from the very start. Section 6.3.1 gives an account of how the program performed (CPU usage, etc.).

3.6. Conclusion

This section firstly described the methodology and design philosophy that would be utilised. It then defined the high-level system design of the project. Next it talked about the various technologies that were used throughout the project and discussed the risks that were originally identified surrounding the project.

The two most crucial design considerations were whether to:

- use Havok and sacrifice cross-platform support
- use multi threading

Evolutionary prototyping was chosen as an appropriate methodology for the project, mostly due to the fact that it is good for handling "unknown" elements that are innate to a project such as this.

The next section describes the implementation of the project; each prototype has its own section.

4. Implementation

This section of the dissertation describes the implementation of the project. Thus the section mostly describes coding, but also design, as evolutionary prototyping was the methodology that was used. The project was initially broken up into prototypes which would encapsulate the major features that were to be implemented. Each prototype is described here, along with how successful that prototype was, and the problems encountered along the way.

It is hoped that this document should also provide a useful guide for anyone who wishes to develop a similar system.

4.1. Development Environment

For the first few prototypes the hardware setup shown in figure 11 was used. It consists of a PC with the Windows XP operating system installed. Several cameras were used at various stages throughout the project; see section 3.3.4 for a description of these. The camera used at the beginning was a Logitech QuickCam Express, with a maximum resolution of 352 by 288 pixels. Microsoft Visual Studio 2005 was used as an integrated development environment. OpenGL, Havok and ARToolkit libraries were installed.

From prototype 8 onwards, a more sophisticated camera was used. It was a Logitech QuickCam S7500 with a resolution of 960 by 720 pixels. This project was coded using many different computers, but for the testing and evaluation, the computer used was an Apple MacBook Pro Rev 3. The evaluation also made use of a Trivision ARVision 3D head-mounted display (HMD), used to view the augmented scene.

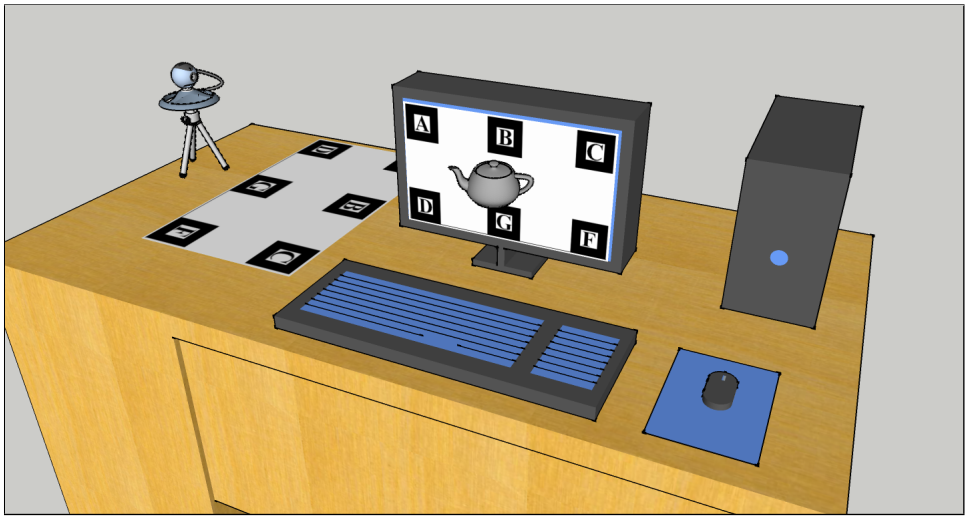


Figure 11: Hardware setup

4.2. Prototype 1 - Humble Beginnings

Prototype aim: To have ARToolkit working and reading images from a USB camera.

4.2.1. Beginning with ARToolkit

After the evaluation of ARTag, Bazar[18] and ARToolkit, it was decided that ARToolkit would be used as the base augmented reality technology. Bazar is an interesting alternative because of its fundamental difference; it is not based on tracking fiduciary markers, but uses OpenCV and feature tracking. See section 3.3.1 for a discussion of these various technologies and their various advantages.

This first prototype involved deeper research and investigation into the technologies that had been chosen previously. The core technology key to the project was of course the augmented reality software: ARToolkit.

A convenient starting point was the ARToolkit samples. The sample "simpleLite" simply draws a cube on a marker using Glut functions. The algorithm for finding a fiduciary marker in this sample is as follows:

Input: Marker pattern file, Image from camera
Output: Marker transformation matrix ($ResultMatrix$)
if $Image \neq$ *NULL* **then**
  $MarkerInfos =$ arDetectMarker($Image$, $MarkerPattern$);
  k = -1;
  foreach *Marker* j *in* $MarkerInfos$ **do**
    if $k =$ *-1* **then**
      $k = j$;
      // First marker detected.
    end
    else if *GetConfidenceValue(*j*)* $>$ *GetConfidenceValue(*k*)* **then**
      $k = j$;
      // Higher confidence marker detected.
    end
  end
  if $k \neq$ *-1* **then**
    $ResultMatrix$ = arGetTransformationMatrix($MarkerInfo[k]$);
  end
end

[18] http://cvlab.epfl.ch/software/bazar/

In the above algorithm, *arDetectMarker* is the ARToolkit function that searches an input image for markers. Any potential markers are put into the structure *MarkerInfos*. The function *arGetTransformationMatrix* converts a potential match from the *MarkerInfos* structure into the matrix *ResultMatrix* that can be manipulated. The matrix can be loaded as OpenGL's projection matrix [42], after which the 3D position (0, 0, 0) represents the tag position in the world:

```
double p[16];
// this gets the appropriate projection matrix
arglCameraFrustum(getARParam(), VIEW_DISTANCE_MIN, VIEW_DISTANCE_MAX, p);
glMatrixMode(GL_PROJECTION);
glLoadMatrixd(p);
```

So to draw a cube at the fiduciary marker position using Glut, a simple call to *glutSolidCube* is all that is required. The problem with this simplistic approach is that the position and orientation of the fiduciary marker in relation to the camera are not known. To acquire this information some knowledge of matrix and quaternion mathematics is required. Section 4.7.7 shows how to do this, but it was not required at this prototype.

4.2.2. Results

To achieve this prototype, a USB camera was installed and tested with a simple Windows program that was based on the "simpleLite" sample program. Figure 12 shows the results of this prototype; a fiduciary marker is detected and a Glut cube is drawn at the position of it.

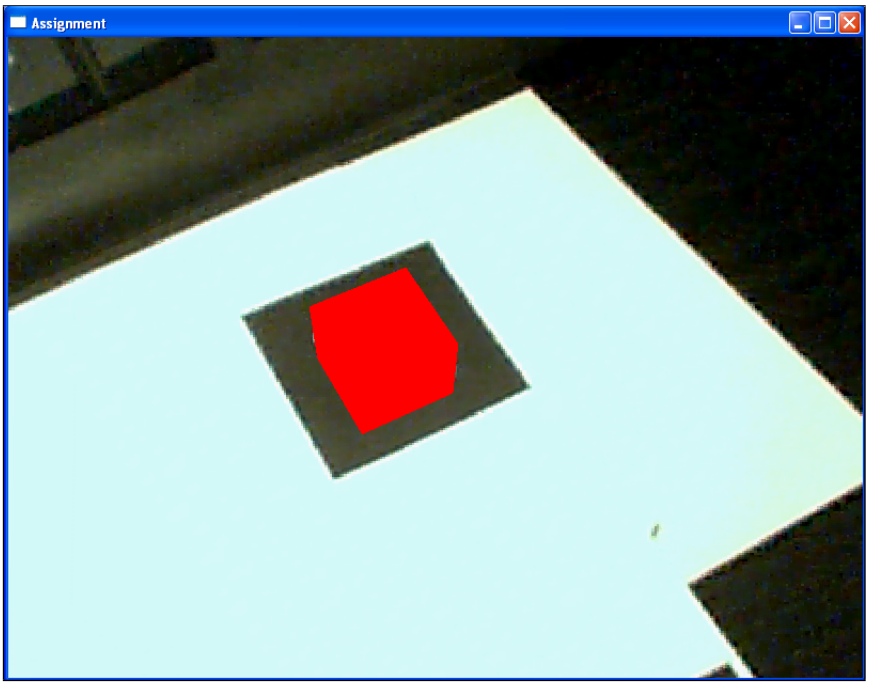


Figure 12: Results of prototype 1

4.3. Prototype 2 - Let There Be Cubes

Prototype aim: Get ARToolkit and OpenGL working together, drawing images from camera on screen, and drawing a simple OpenGL model on the position of a fiduciary marker.

4.3.1. Introduction of OpenGL

The first prototype simply used Glut to render the camera image, but this is much too limited for larger applications due to the fact that a Glut based program is required to call:

```
glutMainLoop()
```

This function is "blocking", i.e. it never returns. If a developer wishes to define and control their own main loop then using OpenGL without the glut interface is a better option, so that was the main aim of prototype 2.

The problem with not using Glut is that window creation also has to be defined by the developer. This means that on each operating system (e.g. Windows, Linux or Mac OS X), an application framework would need to be created (Although the Linux and Mac OS X frameworks would differ by only a few *#ifdef* directives.)

At this stage development continued only on Windows, simply because all the hardware was configured and working on a Windows system. However, the coding was continued in a cross-platform manner, and it is expected that with a little work the system could easily be made cross-platform compatible.

The base code for the Windows framework was a standard barebones Windows program which consisted of an implementation of *WinMain*, which needs to be implemented for any Windows program that is not a console program (there are ways to rename this function). A *WNDCLASSEX* class was created for the window. The function call *PeekMessage* was used to check for Windows messages. For messages that this simple program was not interested in, *DispatchMessage* was called. *wglCreateContext* was used to obtain an OpenGL rendering context.

The main update loop was in the WinMain function. Calls to *Draw* and *Update* drew and updated the program, respectively. The *Update* function called all the necessary ARToolkit functionality, including checking for fiduciary markers in the scene. The *Draw* function drew the captured camera images and the OpenGL objects that would be displayed.

Now that OpenGL and ARToolkit were working together, the next stage was to test drawing virtual objects. The results of this are shown in figure 13. The snooker table was used to test the drawing of objects; it was made up of spheres and OpenGL quads (Glut was still used to draw these, it is not necessary to use Glut's main loop to use it's helper functions).

4.3.2. ARToolkit Camera Drawing

To draw the camera images with OpenGL, ARToolkit provides the following function:

```
arglDispImage(m_arImage, &m_ARTCParam, 1.0, m_ArglSettings)
```

The variable *m_ArglSettings* is a reference to an OpenGL context. This context is set up by calling:

```
m_ArglSettings = arglSetupForCurrentContext()
```

Then the function:

```
arVideoCapNext()
```

is called to indicate that this frame had been finished with. To capture the next image from the camera a call is made to:

```
arVideoGetImage()
```

4.3.3. ARToolkit Settings

ARToolkit makes use of an XML file to store settings about what camera is being used. The most important line in this XML file is:

```
frame_width="960" frame_height="720"
friendly_name="Logitech QuickCam S7500"
```

This tells ARToolkit what resolution to use for the camera, and the friendly name of the camera to use. If these settings are not correct, ARToolkit will fail to function. This is necessary for Windows only; for other operating systems the configuration is slightly different[19]. The friendly name is Windows' way of referring to a specific camera. The friendly name of a camera can be determined on Windows using the program GraphEdit, which is part of the Microsoft DirectShow SDK.

A call to:

```
arVideoOpen(vconf)
```

where vconf is the path to the XML file, will setup ARToolkit to use a certain camera.

4.3.4. Results

Figure 14 shows a screenshot of this prototype, running without the use of a Glut main loop. The code for this project was relatively simple and was all contained in just one file. The next prototype describes the design of a more modular system.

[19] http://artoolkit.sourceforge.net/apidoc/video/

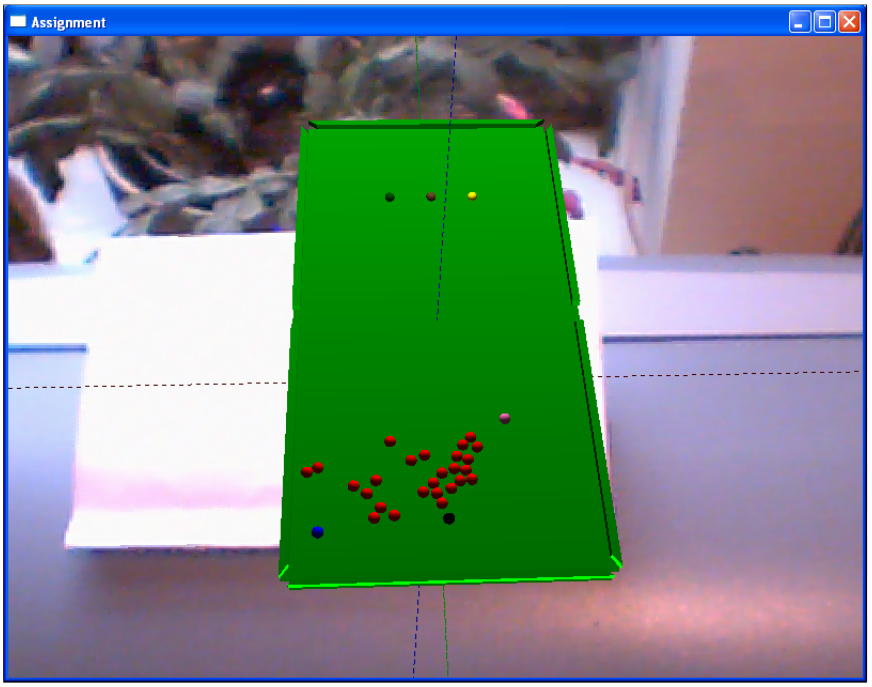


Figure 13: Testing OpenGL and ARToolkit integration

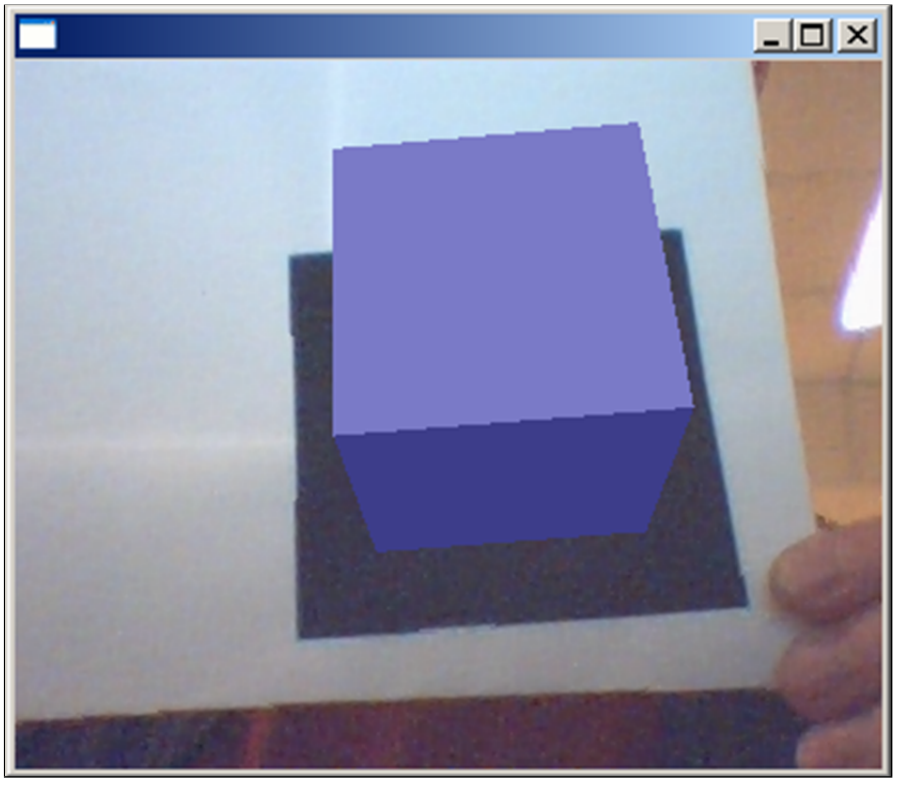


Figure 14: Screenshot of prototype 2

4.4. Prototype 3 - Tagged

Prototype aim: Setup environment (table) with fiduciary markers. Modularise code structure.

At this stage an OpenGL base had been created (on Windows) and ARToolkit was being used to draw cubes on augmented reality (AR) fiduciary markers. The next stage was to create the modular components that would form the basis of the system. The first iteration of the design of this modular system is shown in figure 15.

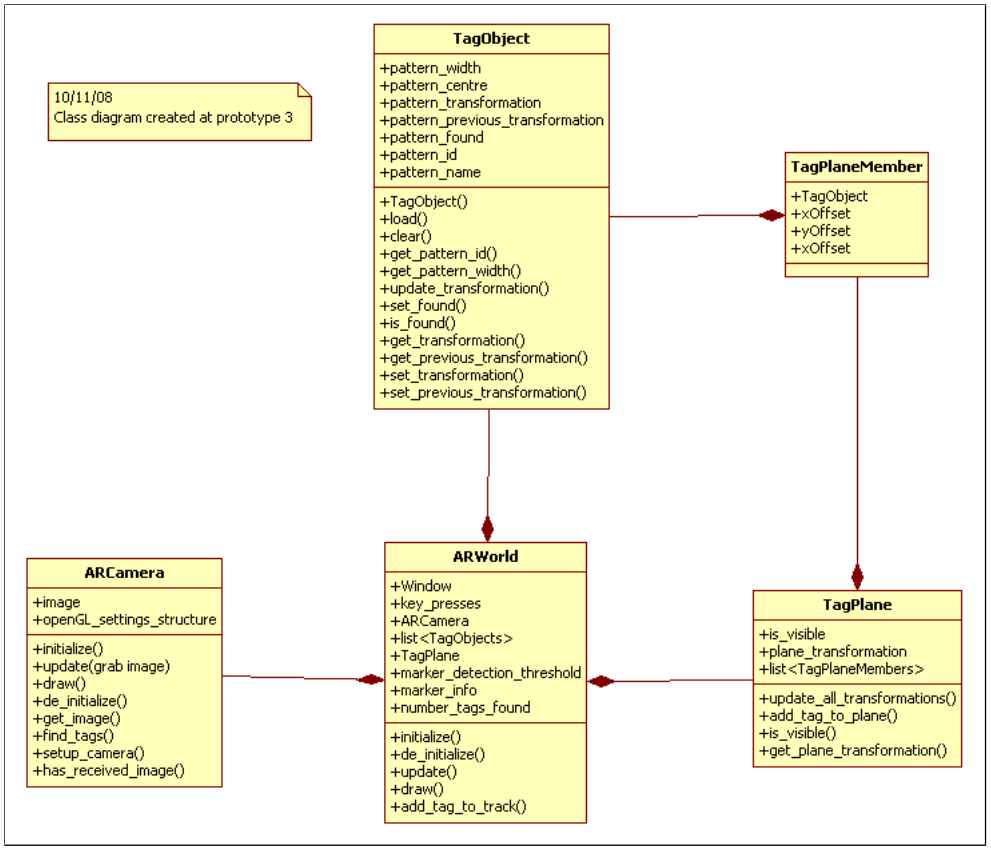


Figure 15: First design of modular system

4.4.1. Multiple Markers

ARToolkit provides the ability to track multiple patterns that define only one position. This is useful if there is a large area (such as a table), which needs only one reference in a coordinate frame. For this system, the table is defined as the "ground"; and multiple markers are necessary if the user is standing in different positions around the table, and also viewing from various angles. However, all these fiduciary markers only need to define one point, i.e., the center of the table as (0, 0, 0) in a normal 3D Cartesian coordinate system [43], and so fiduciary markers are defined in reference to another fiduciary marker, which itself defines the position (0, 0, 0).

This allows the use of a *multi-marker* configuration such as the one shown in figure 16, and if at least one of the markers in the multi pattern can be seen, the position of the

entire configuration can be determined. A multi-marker configuration is defined by a text file. For example, a simple configuration of only two member markers (patt.a and patt.b) is shown in snippet 1. In this case pattern B is 100 millimeters offset in the X-axis in relation to pattern A.

```
#the number of patterns to be recognized
2

#marker 1
Data/multi/patt.a
40.0
0.0 0.0
1.0000 0.0000 0.0000 0.0000
0.0000 1.0000 0.0000 0.0000
0.0000 0.0000 1.0000 0.0000

#marker 2
Data/multi/patt.b
40.0
0.0 0.0
1.0000 0.0000 0.0000 100.0000
0.0000 1.0000 0.0000 0.0000
0.0000 0.0000 1.0000 0.0000
```

Snippet 1: Multi marker configuration

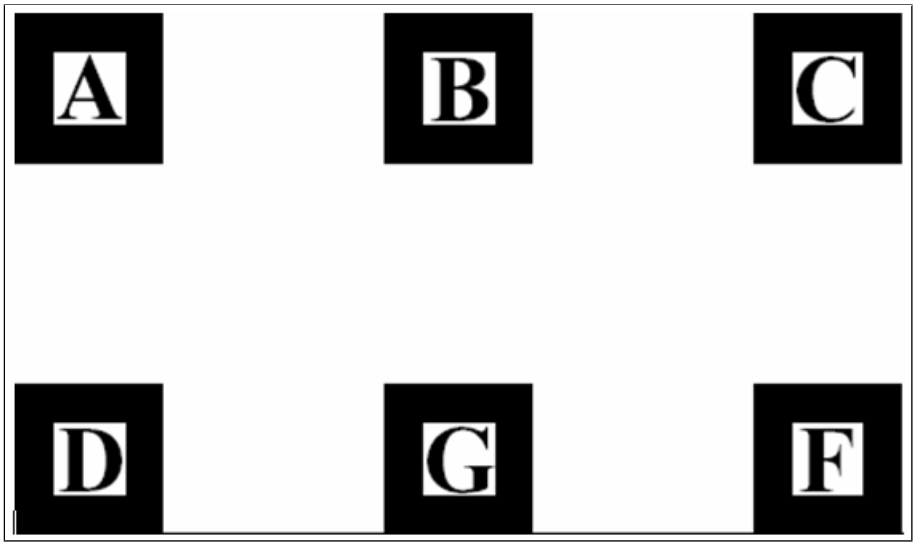


Figure 16: Sample multiple marker pattern included with ARToolkit

However, it was found that multi marker tracking was much less stable than single marker tracking. With single markers the ARToolkit function *arGetTransMatCont*, can be called instead of the function *arGetTransMat*.

This function uses a history function to reduce the area of the image to search for the markers, hence tracking is more stable. There is no corresponding history function for multi marker tracking. Thus the class TagPlane was created, in an attempt to better implement a similar system of multi-marker tracking by using the function *arGetTransMatCont* for the individual markers. This was a mistake, because although the TagPlane method worked, it was just as unstable as the ARToolkit implementation. It is thought that the relationships between the markers introduce inherent instabilities. Therefore, to keep the code simpler, the TagPlane class was eliminated and the ARToolkit multi marker tracking was used instead.

This does not mean that the entire system was completely unstable, only that is was *less* stable the registration of a single fiduciary marker is. It was still usable, it is just an inherent problem with the ARToolkit technology.

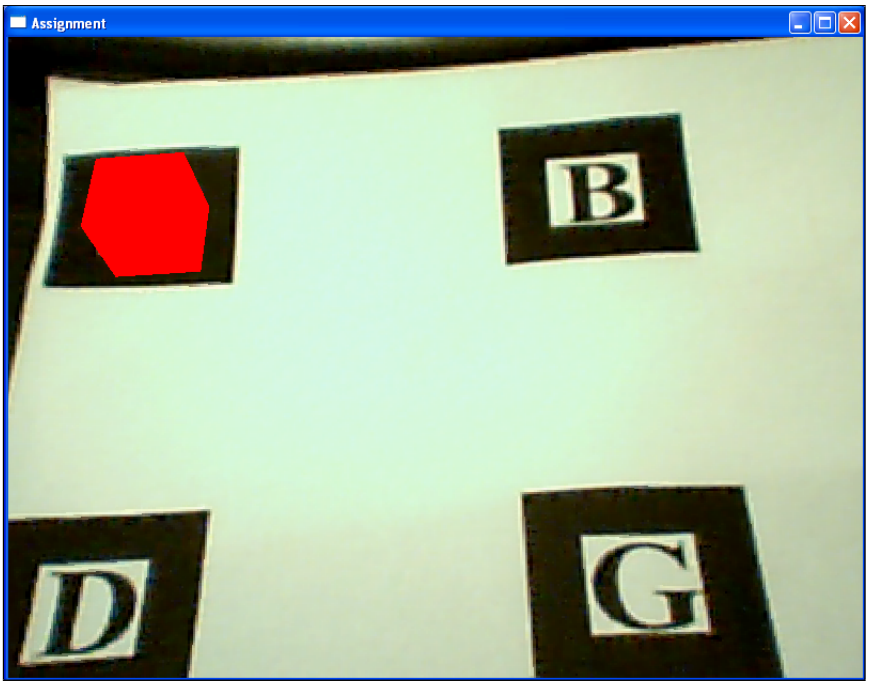


Figure 17: The printed out multi-marker pattern, with a virtual cube

4.4.2. Results

ARToolkit provides some basic marker patterns for detection. A sample multi-pattern uses the letters A, B, C, D, G, and F. These were used to define a flat plane which would form the "ground". They were however printed out at a larger size to make detection easier (see figure 17).

```
ARWorld* g_arWorld;

BOOL Init(Window* window, Keys* keys)
{
        g_arWorld = new ARWorld(window, keys);
        g_arWorld->initialize();

        return TRUE;
}

void DeInit(void)
{
        g_arWorld->deInitialize();
}

void Reset()
{
}

void Update (float ms)
{
        g_arWorld->update(ms);
}

void Draw (void)
{
        // draw image straight from camera
        g_arWorld->draw();
}
```

Snippet 2: Basic program structure

4.5. Prototype 4 - Forklift Introduced

Prototype aim: Detect forklift (using fiduciary markers).

At this stage a cardboard box was used as a stand in for the forklift, as a Lego Mindstorms Robot was not available for use. This prototype involved tracking a moving fiduciary marker, as well as the TagPlane discussed in prototype 3.

Snippet 2 shows that main control class, ARWorld (see figure 15.) Although fairly self-explanatory, this code can be visually understood from figure 3 in section 2.2; the three stages shown in the figure are present in the code.

4.5.1. Results

Figure 18 shows the forklift as represented by a cardboard box. The "world plane", or ground is defined by the multi-marker pattern as described in the previous prototype. The position of (0, 0, 0) in the world is at the center of the "A" pattern. This was the point of reference for all other objects in the world.

At the end of this prototype, single markers could now be detected and registered as well as the multi-marker pattern.

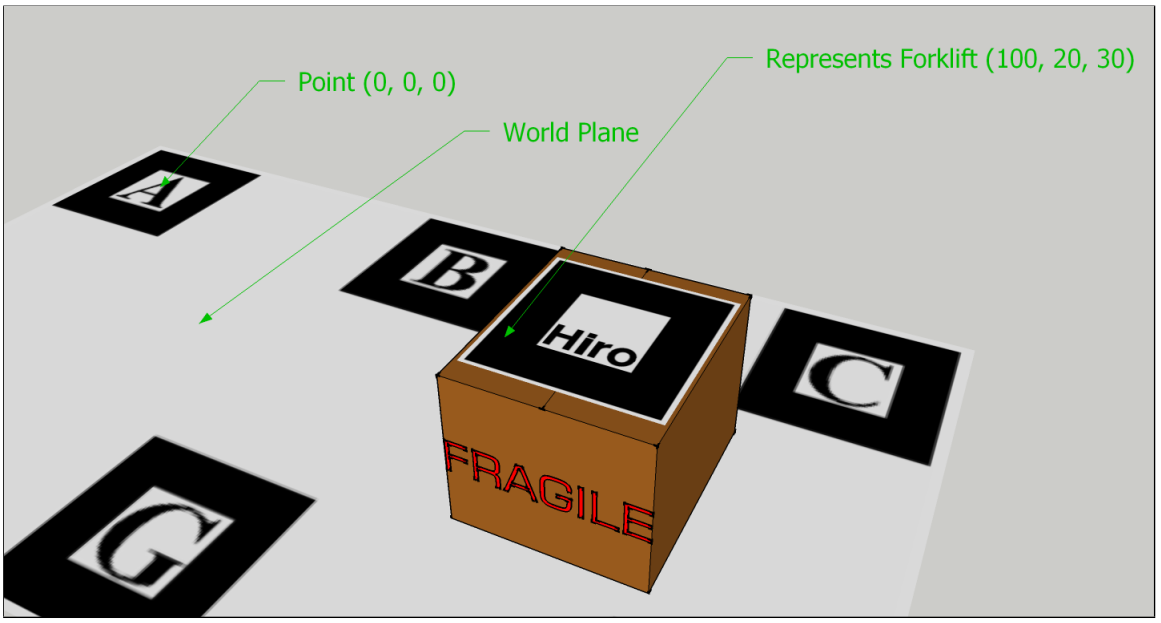


Figure 18: Forklift represented as a cardboard box

4.6. Prototype 5 - Headset

Prototype aim: At this stage the AR HMD is integrated into the system to test for suitability early on in the project.

4.6.1. Headset Installation

The AR HMD was installed on a Windows machine and tested with existing code. The HMD worked out of the box as expected. Figure 19 shows the HMD that was used in this prototype.

Figure 19: Augmented Reality Head-mounted display (HMD)

4.6.2. Textures and Attaching Cameras

To make the virtual boxes more realistic, it was decided to texture them. In OpenGL to create a texture requires several function calls:

```
// create storage space for a texture
static GLuint texture;

// create storage space for a texture
AUX_RGBImageRec *TextureImage;

// set the pointer to null
memset(TextureImage,0,sizeof(void *));

// load a bitmap, might not be found
if ( (TextureImage=LoadBMP("res/crate.bmp")) != 0)
{
        // create the texture
        glGenTextures(1, &texture);

        // Typical Texture Generation Using Data From The Bitmap
        glBindTexture(GL_TEXTURE_2D, texture);
        glTexImage2D(GL_TEXTURE_2D, 0, 3,
                TextureImage->sizeX, TextureImage->sizeY,
                 0, GL_RGB, GL_UNSIGNED_BYTE, TextureImage->data);
        glTexParameteri(GL_TEXTURE_2D,
                        GL_TEXTURE_MIN_FILTER,
                        GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D,
                        GL_TEXTURE_MAG_FILTER,
                        GL_LINEAR);
}
```

After a texture has been created, it has to be bound before it is used:

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texture[0]);
```

When using this system, a user might have forgotten to plug in a camera, or may have not installed the camera properly. It would not be good if the program crashed because of this. Therefore, the camera is *attached* after the program is started by pressing a keyboard shortcut. In this context attached means that the relevant ARToolkit functions for setting up a camera are called.

Also, a keyboard press could *detach* a camera, so that another or the same camera could be attached.

4.7. Prototype 6 - Physics

Prototype aim: Integrate Havok into the system. Implement a simple bounding box around the forklift. Virtual objects should behave somewhat realistically with the forklift at this stage, although occlusion is not implemented, and the accuracy of collision is crude.

4.7.1. New Classes

This prototype added a several new classes to the system: *HavokWorld*, *PhysicsTagObject* and *Forklift*. The *HavokWorld* class is responsible for handling all of the physics simulation. The *PhysicsTagObject* class is a specialisation of the TagObject class that includes a Havok *hkRigidBody* as an attribute. This allows a fiduciary marker to be associated with a particular physics object. The *Forklift* class is a specialisation of the *PhysicsTagObject* class.

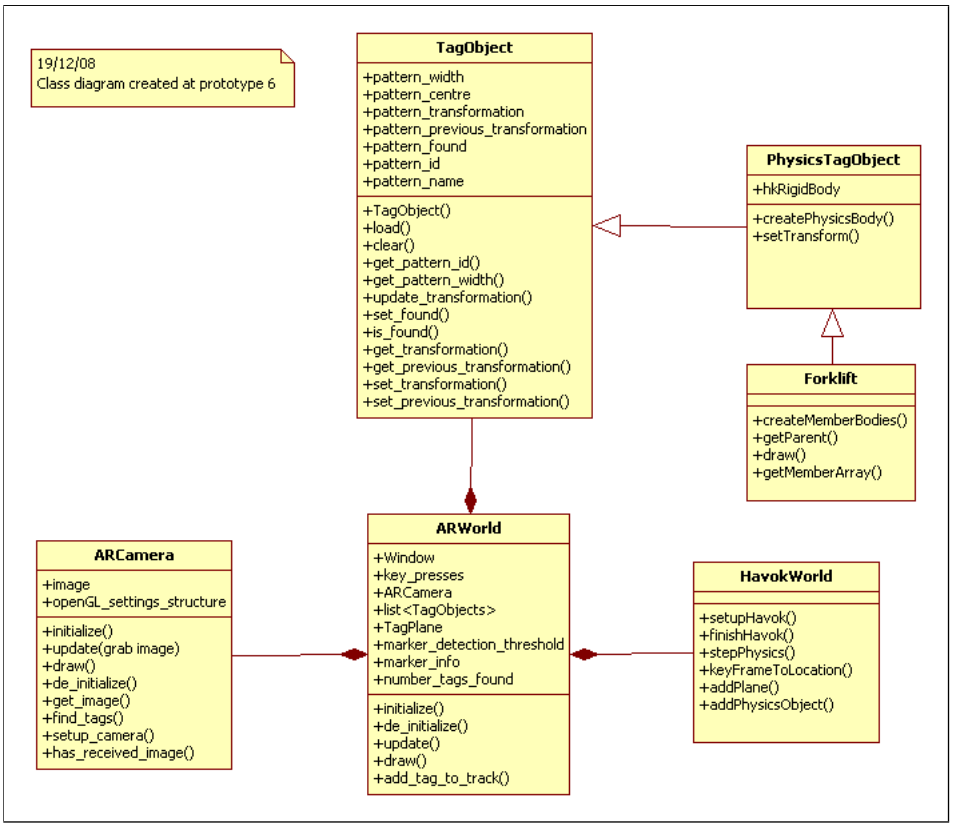


Figure 20: Class diagram of improved design of modular system

Figure 20 shows these three new classes. It may also be noticed that two other classes are no longer present. These are *TagPlaneMember* and *TagPlane*. These classes became deprecated; see section 4.4.1 for details.

4.7.2. Initialising Havok

To initialise the Havok engine several things need to be done. Firstly, the base system has to be initialised, including the memory system:

```
m_memoryManager = new hkPoolMemory();

m_threadMemory = new hkThreadMemory(m_memoryManager);

hkBaseSystem::init( m_memoryManager, m_threadMemory, errorReport );

m_memoryManager->removeReference();
```

The next step is initialise the stack area for fast temporary memory for the engine:

```
int stackSize = 0x100000;

m_stackBuffer = hkAllocate<char>( stackSize, HK_MEMORY_CLASS_BASE);

hkThreadMemory::getInstance().setStackArea( m_stackBuffer, stackSize);
```

Havok needs to know about the system hardware so that the number of threads can be set:

```
// Get the number of physical threads available on the system

hkHardwareInfo hwInfo;

hkGetHardwareInfo(hwInfo);

totalNumThreadsUsed = 2;
```

In this case the number of threads was set to two. Although it is usual to set the number of threads to *hwInfo.m_numThreads*, it was found that quad core systems when this variable took the value of four that Havok was performed poorly. Forcing quad cores systems to use two threads solved this. Considerable time was spent determining the cause for this poor performance.

Havok uses a thread pool to complete various tasks, so this is the next thing to be set up:

```
// We use one less than this for our thread pool,
//because we must also use this thread for our simulation

hkCpuJobThreadPoolCinfo threadPoolCinfo;

threadPoolCinfo.m_numThreads = totalNumThreadsUsed - 1;


// This line enables timers collection,
// by allocating 200 Kb per thread.
// If you leave this at its default (0)
// timer collection will not be enabled.

threadPoolCinfo.m_timerBufferPerThreadAllocation = 200000;

m_threadPool = new hkCpuJobThreadPool( threadPoolCinfo );
```

```
// We also need to create a Job queue.
// This job queue will be used by all
// Havok modules to run multithreaded work.

// Here we only use it for physics.

hkJobQueueCinfo info;

info.m_jobQueueHwSetup.m_numCpuThreads = totalNumThreadsUsed;

m_jobQueue = new hkJobQueue(info);
```

The next step in configuring Havok is to set up the global simulation parameters, including gravity, solver settings etc. It is in this section that gravity is set to an abnormal level (see section 4.10.6).

The simulation type is set to SIMULATION_TYPE_MULTITHREADED, to make use of computers with multiple cores. The broadphase border limit is essentially the size of the world. Setting such a limit is a performance issue; when objects reach this border they are usually deactivated and not updated, or sometimes they might be deleted. For debugging purposes, objects in this system are not even deactivated, but are still considered in collision detection, etc.

A new *hkpWorld* object is created using these parameters:

```
// The world cinfo contains global simulation parameters,
// including gravity, solver settings etc.


// Set the simulation type of the world to multi-threaded.

m_worldInfo.m_simulationType
                = hkpWorldCinfo::SIMULATION_TYPE_MULTITHREADED;

m_worldInfo.setupSolverInfo(hkpWorldCinfo::SOLVER_TYPE_4ITERS_MEDIUM);

m_worldInfo.m_gravity = hkVector4(0, -3500, 0);


// Flag objects that fall "out of the world"
/// to not be deactivated - not necessary for this scene
m_worldInfo.m_broadPhaseBorderBehaviour
                = hkpWorldCinfo::BROADPHASE_BORDER_DO_NOTHING;


m_worldInfo.setBroadPhaseWorldSize(2200);


m_physicsWorld = new hkpWorld(m_worldInfo);
```

This concludes the main steps that are necessary to initialise Havok. Next it will briefly be shown how the physics is updated.

4.7.3. Updating Havok

This following code snippet shows how to update the physics simulation. Since Havok is being run in multithreaded mode a call is made to *stepMultithreaded*. *m_timestep* is set to $1/60$ so this function is called sixty times every second. This value was chosen simply as a value recommended by the Havok documentation (provided with the free download).

```
m_physicsWorld->stepMultithreaded(m_jobQueue, m_threadPool, m_timestep);

hkMonitorStream::getInstance().reset();

m_threadPool->clearTimerData();


// Pause until the actual time has passed

while (m_stopWatch.getElapsedSeconds() < m_lastTime + m_timestep);

m_lastTime += m_timestep;
```

4.7.4. Finishing the Simulation

To shutdown Havok properly the thread memory, stack area and job queue have to be released and deallocated.

4.7.5. Resetting Havok

To reset Havok, it was found that it was easiest for this project just to call the shutdown and initialise functions that encapsulate the above functionality.

4.7.6. Adding Physics Objects

To add a physics object to the world, the world must first be *marked for writing*:

```
m_physicsWorld->markForWrite();
```

This is because it must be ensured that only one thread is modifying the world at any one time. After the object has been added a call is made to:

```
m_physicsWorld->unmarkForWrite();
```

In Havok the most common type of object that is added to the world is a rigid body. This is a body that does not change its shape. The Havok class for rigid bodies is the class *hkRigidBody*.

The following example shows how to add a simple box shaped *hkRigidBody* to the simulation; in this case a plane. The *hkpRigidBodyCinfo* contains information about the rigid body that is to be created, including its shape and the quality of the collision detection for the rigid body. Also, the position of the rigid body must be set.

```
// this will be returned

hkpRigidBody* rigidBody;


hkVector4 planeRadii( width/2, 2.0f, height/2 );

hkpConvexShape* shape = new hkpBoxShape( planeRadii, 0 );


hkpRigidBodyCinfo ci;


ci.m_shape = shape;

ci.m_motionType = hkpMotion::MOTION_FIXED;

ci.m_position = hkVector4(xPos, yPos, zPos);

ci.m_qualityType = HK_COLLIDABLE_QUALITY_FIXED;

rigidBody = new hkpRigidBody( ci );

m_physicsWorld->addEntity(rigidBody)->removeReference();

shape->removeReference();
```

4.7.7. Integration of Havok and ARToolkit

To integrate Havok and ARToolkit, we need to obtain a position and orientation of the forklift that can be passed to the Havok simulation.

To do this, the first task is to get the matrix that defines the relationship between the multi-marker pattern and the forklift fiduciary marker. We start off with two known matrices (obtained by the correct ARToolkit function calls):

1. The matrix that defines the position and orientation of the multi-marker pattern in relation to the camera position and orientation.

2. The matrix that defines the position and orientation of the forklift fiduciary marker in relation to the camera position and orientation.

The only common reference point between these two matrices is that of the position of the camera; thus the matrix that defines the camera position and orientation must be found. *m_multi_config->trans* in the below snippet holds the matrix that defines the orientation and position of the multi-marker pattern, in relation to the camera. Thus, to get the position and orientation of the camera, we get the inverse of this matrix:

$$M^{-1}_{multi-marker} \Rightarrow M_{camera}$$

Once we have this, the next step is to multiply this camera matrix by the matrix that defines the position and orientation of the forklift fiduciary marker:

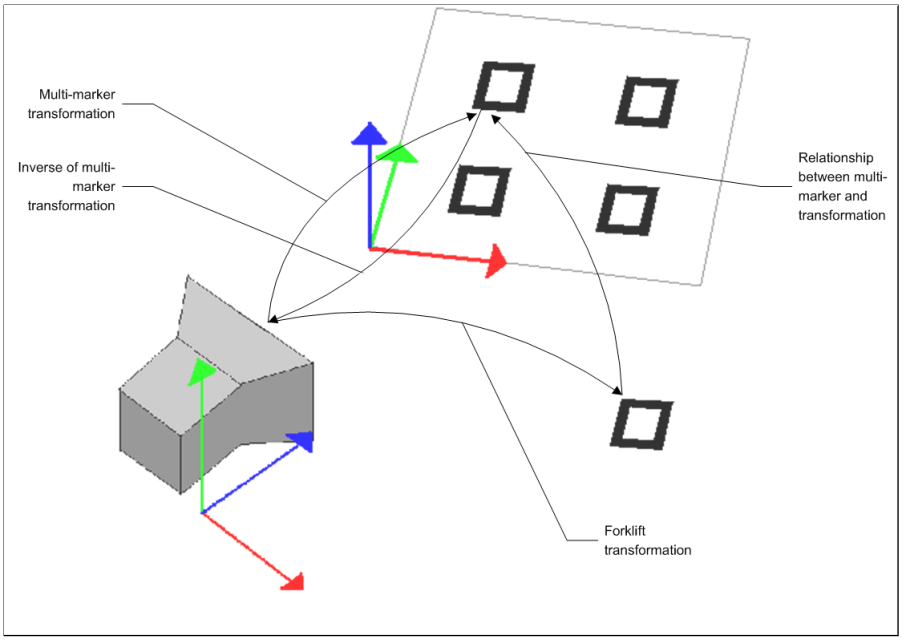


Figure 21: Relationship of forklift to multi-marker pattern

$$M_{camera} \times M_{forklift} \Rightarrow M_{result}$$

This will result in a matrix which defines the relationship between the forklift and the multi-marker pattern. This works because of the fact that multiplying two transformation matrices together will result in a matrix that is the combination of the two. From this resultant matrix we can extract the position and orientation of the forklift, relative to the multi-marker pattern. Combining the above two steps into one:

$$M^{-1}_{multi-marker} \times M_{forklift} \Rightarrow M_{result}$$

If this seems confusing, try reading the above again whilst referring to figure 21.

The following code achieves this, storing the result in wmat2:

```
double forklift_trans[3][4];
double gl_para[16];
double  wmat1[3][4], wmat2[3][4];

arUtilMatInv(m_multi_config->trans, wmat1);

// forklift tag
m_forklift.updateTrans(m_marker_info, m_num_tags);
m_forklift.getPattTrans(forklift_trans);

arUtilMatMul(wmat1, forklift_trans, wmat2);

argConvGlpara(wmat2, gl_para);
```

The position and orientation can be extracted from wmat2(or any matrix) using the ARToolkit utility function *arUtilMat2QuatPos*:

```
double quat[4], pos[3];
int error = arUtilMat2QuatPos(wmat2, quat, pos);
```

This position and orientation information can be used to synchronise the motion of a Havok physics entity and a fiduciary marker. However, it was found that it did not work to just simply set the position of the forklift. Chaos ensued in the simulation, for when the forklift fiduciary markers was momentarily lost and then subsequently found again, the forklift would be instantly transported across great distances; and often to a position *inside* other entities.

The Havok physics engine offered a suitable solution, which was the notion of keyframed entities. The idea of these entities is similar to the idea of keyframing in animation [44]. A keyframed physics entity in Havok is entirely under the control of the programmer, and has an effectively infinite mass. A keyframed physics entity only goes where it is told to go, and is not affected by collisions with other entities and gravity, etc. Even better, keyframing in Havok controls the amount of time that an object takes to reach a new position. This ability was used to prevent the forklift instantly jumping from one location in the simulation to another; instead it moved smoothly to the correct position over a specified period of time.

Snippet 3 demonstrates the use of keyframing. It may be noticed that the X, Y, and Z coordinates obtained from a tag are not directly passed to the *applyHardKeyFrame* function. This is because ARToolkit uses a different coordinate system than Havok. Havok uses a right-handed coordinate system, with the camera facing in the direction of -Z in the same way that OpenGL does.

In ARToolkit the X and Y axes are parallel to the tag and the Z axis is orthogonal. See figure 22 for a graphical representation, from the ARToolkit official documentation [5].

4.7.8. Results

At this point, the forklift is represented in physics only by a simple box. In prototype 9, the forklift was created out of several boxes, creating a more accurate physics representation

```
void HavokWorld::keyFrameToLocation(hkpRigidBody* body,
                                    hkReal x,
                                    hkReal y,
                                    hkReal z,
                                    hkQuaternion* rot)
{
        m_physicsWorld->lock();

        hkVector4 pos;

        pos.set(hkMath::clamp<hkReal>(x, -1000, 1000),
                hkMath::clamp<hkReal>(z, -1000, 1000),
                hkMath::clamp<hkReal>(-y, -1000, 1000));
        rot->setInverse((*rot));

        hkpKeyFrameUtility::applyHardKeyFrame(pos, (*rot), 7.0f, body);

        m_physicsWorld->unlock();
}
```

Snippet 3: Havok Keyframing

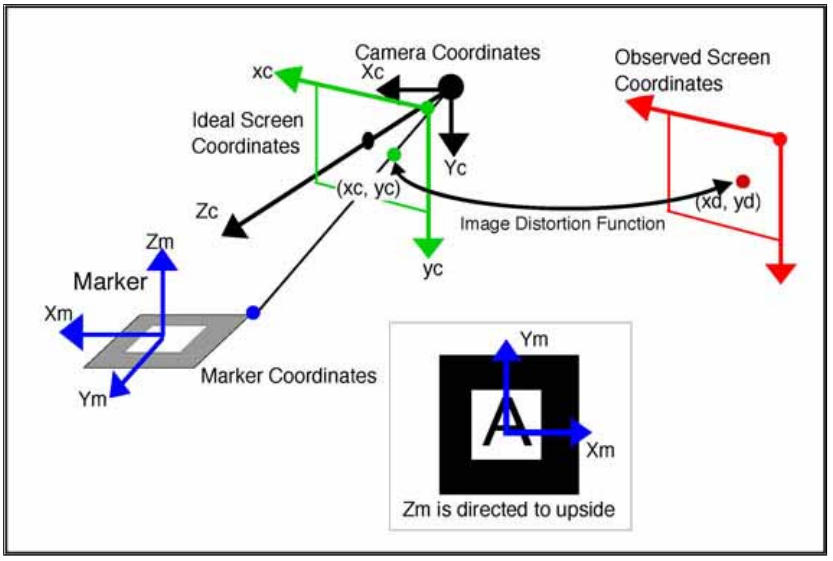


Figure 22: ARToolkit coordinate systems, from [5]

of the forklift.

4.8. Prototype 7 - Occlusion

Prototype aim: This prototype is a refined version of prototype 6, but it will also include occlusion.

4.8.1. OpenGL Depth Buffer Method

How to accomplish occlusion was considered to be one of the more involved components of the project. It turned out however to be fairly simple, thanks to the use of OpenGL. Although a popular technique for similar situations [45], the method was discovered thanks to a DIT final year project from 2008 by Qingqing Dong [32].

OpenGL implements a depth buffer, whereby objects that are behind others (from the camera viewpoint), are *occluded*. Occlusion *culling* is used to remove hidden surfaces to reduce the amount that must be rendered each frame. A depth buffer, otherwise known as a Z-buffer, is one way to keep track of which parts of a rendered scene are visible, and which are hidden. A depth buffer is usually a two-dimensional array that keeps track of the depth at any particular pixel. In this way, only visible objects are rendered.

The technique used to achieve occlusion can be thought as a three step process:

1. Draw transparent objects that are representative of the forklift. This will place the forklift in the correct place in the depth-buffer model of the world.

2. Draw camera image. This will overwrite the colour buffer values, so the objects drawn previously will not be seen.

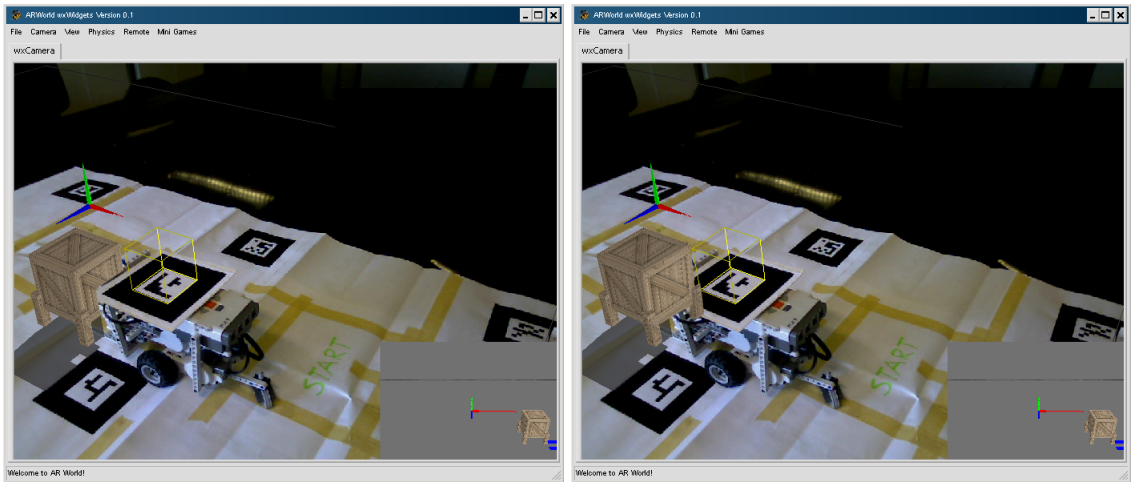
3. Draw the virtual objects in the scene. Only those in front of the forklift will be added to the colour buffer thus are seen.

For a while it seemed that the method described above was flawed, as objects in the scene popped in and out of their correct positions. However, the OpenGL documentation provided the fix, (to be specific in section 12.050 of the documentation [46]). The near clipping plane must be above a certain minimum, otherwise the depth calculations performed by OpenGL would be dividing by a very small number, resulting in poor accuracy. Figure 23a shows a box being occluded by the forklift (This photo was *not* taken at this prototype stage.) Figure 23b shows the same scene, but without occlusion; this demonstrates the difference that occlusion makes to the believability of a scene.

4.8.2. Building the Forklift

The forklift was constructed using a Lego Mindstorms NXT kit. These NXT kits have programmable "bricks" and also come with motors and sensors (sensors were not needed for this project). Figure 24 shows one of these programmable "bricks".

Figure 26 shows the finished forklift. Three motors were used: two for driving and one for controlling the lift. One of the motors drove a *worm drive* mechanism [47]; this lifted the forks. A worm drive is a gear arrangement consisting of a worm and a worm gear. The worm is a gear in the form of a screw. Standard Lego components were used for these gears. Figure 25 shows the mechanism and how it was used to lift the forks.



(a) Scene with depth buffer occlusion (b) Scene without depth buffer occlusion

Figure 23: Depth buffer occlusion

Figure 24: NXT Lego programmable “brick”

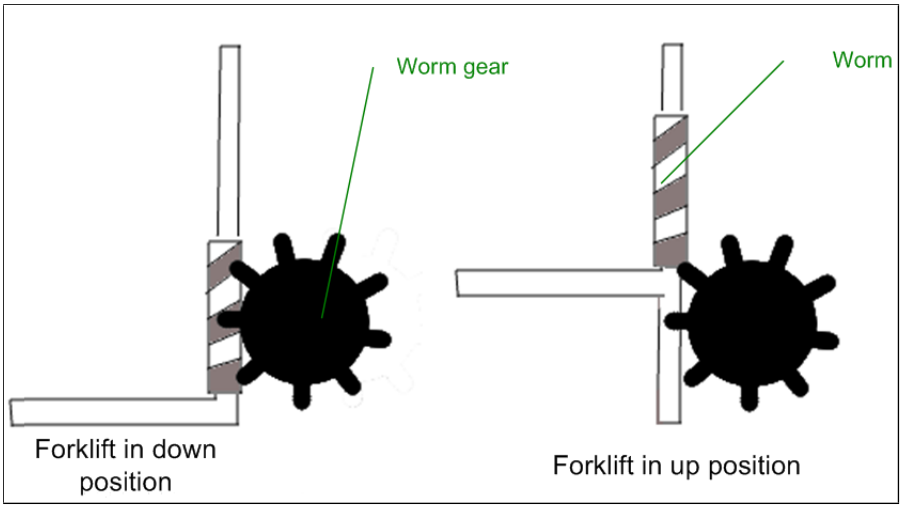


Figure 25: Forklift worm gear mechanism

The Lego remote control system was programmed in Java using Lejos[20] at this proto-

[20]http://lejos.sourceforge.net/

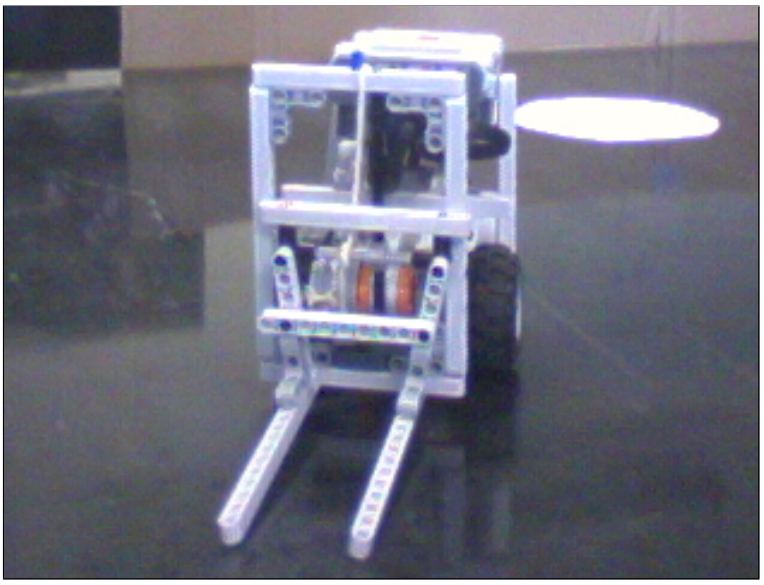


Figure 26: First attempt Lego Forklift

type. Lejos has built in support for Bluetooth communication and advanced motor control functionality. The NXT "brick" had to be flashed with the Lejos firmware before it could be controlled. A simple client-server] paradigm was used to implement a remote control system. The PC acted as a server and initiated the connection to the NXT. The NXT received commands which it obeyed and acknowledged. Using Lejos for this task was somewhat limited, in that the remote control program was a separate entity to the main augmented reality program. At this stage several methods of communication between the two programs were considered:

- The use of a JNI
- Network communication (Localhost)

In prototype 8 Lejos was abandoned, as a more elegant solution was found: direct control of the robot. See section 4.9 for details on this. At this stage, Lejos was the best known choice.

The forklift remote control system is very simple and can be most easily described by the state machine diagram in figure 27. A single key press can switch from any state to any other state. The starting state should find the forklift in an idle position with the lift in the down position.

At this stage the forklift was controlled using keyboard input. The keys for controlling the forklift are shown in table 1.

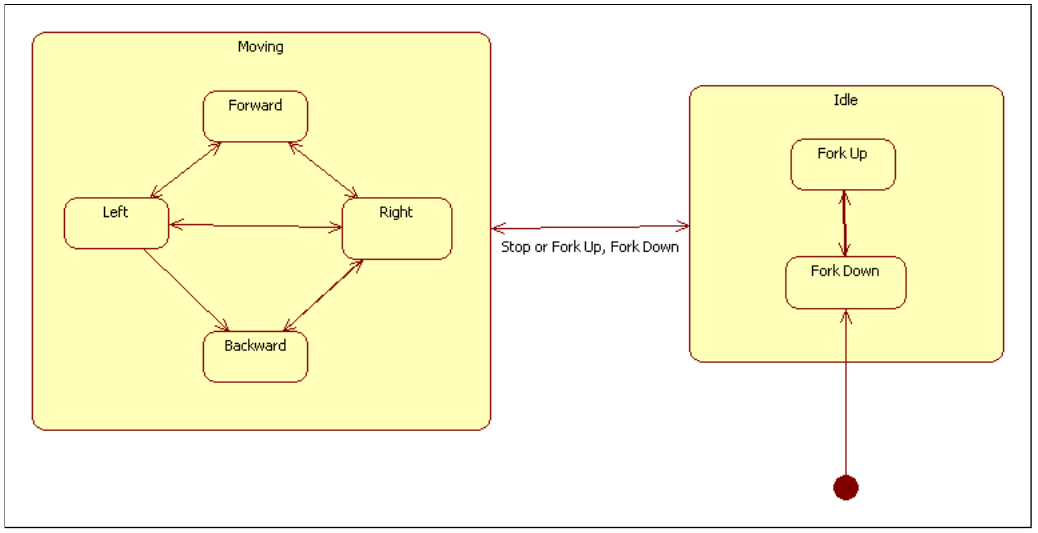


Figure 27: State machine diagram for the forklift

Action	Keyboard Input
Forward	W
Backward	S
Left	A
Right	D
Stop	Space
Lift Up	R
Lift Down	F

Table 1: Keys for controlling the forklift

```
#include "wx/wx.h"

class MyApp: public wxApp
{
    virtual bool OnInit();
};

class MyFrame: public wxFrame
{
public:

    MyFrame(const wxString& title,
            const wxPoint& pos,
            const wxSize& size);
};

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    MyFrame *frame = new MyFrame( _T("Hello World"),
                    wxPoint(50,50), wxSize(450,340) );
    frame->Show(TRUE);
    SetTopWindow(frame);
    return TRUE;
}

MyFrame::MyFrame(const wxString& title,
            const wxPoint& pos, const wxSize& size)
: wxFrame((wxFrame *)NULL, -1, title, pos, size)
{
}
```

Snippet 4: wxWidgets Hello World

4.9. Prototype 8 - A User Interface in Born

Prototype aim: The creation of a graphical user interface and the improvement of the physics simulation.

4.9.1. Creating the Graphical User Interface

The first attempt at an interface was developed using the Microsoft Foundation Classes (MFC) library. However, after research a simpler and easier to use alternative was found; this was wxWidgets. wxWidgets was chosen because it is open-source, free and easily extensible. Program listing 4 shows a wxWidgets "'Hello World" program, demonstrating the simplicity of wxWidgets.

The graphics for this project are based on OpenGL. wxWidgets includes a *wxGLCanvas*, which is an embeddable GUI item which OpenGL can render to. The wxGLCanvas is embedded in a wxPanel, which is in turn embedded in a wxFrame (see figure 29). Figure 28 shows the initial design of the GUI.

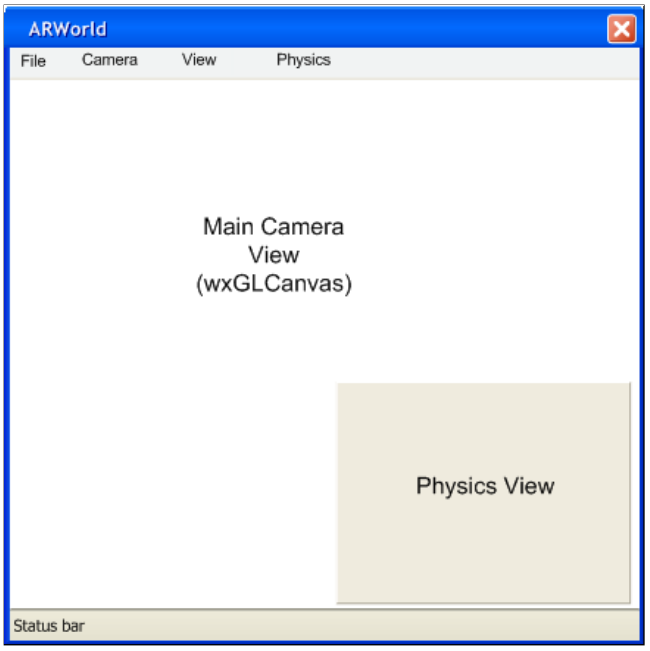


Figure 28: Interface design

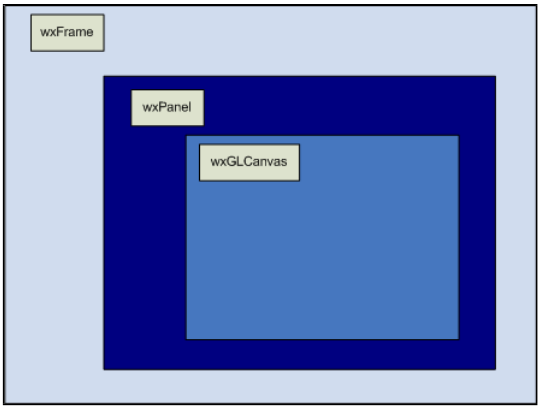


Figure 29: GUI structure

The interface was designed to have to two *view ports*. A main view port that filled the entire window, and a smaller view port in the bottom right corner. These would initially display the captured camera images and the physics representation of the scene, respectively. However, these could be alternated between and also the smaller view port could be turned off.

The menu bar of the application was initially designed to have the following menus:

- File - for closing the application and general options
- Camera - for attaching and detaching cameras
- View - for changing the view ports as described above - also toggling occlusion and solid ground
- Physics - for adding physics objects and resetting the physics engine

4.9.2. Physics Rigid Bodies and Dialogs

At this stage, various physics rigid bodies could be added to the simulation. These were:

- A box (crate)
- A wall - a series of boxes
- A ball

To insert physics objects into the simulation, wxWidgets dialogs were used. These dialogs have three text boxes, one each for the X, Y and Z coordinates of the position to add the Havok rigid body or group of rigid bodies.

4.9.3. wxWidgets Events

wxWidgets uses a powerful event system, similar but more extensive than MFC's message maps. Events that a program throws can be directed to specific member functions. Event tables are used to tell wxWidgets to link events with certain functions. Here is an example event table, the first two entries showing menu events:

```
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU      (wxID_EXIT, MyFrame::OnExit)
    EVT_MENU      (DO_TEST,   MyFrame::DoTest)
    EVT_SIZE      (           MyFrame::OnSize)
    EVT_BUTTON    (BUTTON1,   MyFrame::OnButton1)
END_EVENT_TABLE()
```

Events are generated for all kinds of things, such as menu events, window resize events and even idle events. It was these three that were used predominantly. Menu events were used to link the menu described above with certain functions. The resize event was handled so that when the window was resized, the layout of widgets could be recalculated. The most important event handled was the idle event. It is in this way that the physics was updated and camera images captured; the idle event handling function acted like the main update loop. Whenever wxWidgets has finished doing its own processing, it checks to see if any widgets have requested an idle event. If there is one, the function linked to the idle event is called. However, for a processor intensive application like this one, when the idle event was handled, a call was made to *event.RequestMore()*. This tells wxWidgets to keep giving idle events to make sure that the physics and the camera images are updated regularly.

The program also handled the paint event, so that the window was repainted. It was in the paint event handling function that the OpenGL graphics was updated. Therefore, the updating of the program and the painting of the program where completely separate. The advantage of this separation is that the physics simulation and the rendering of the scene could have different frame rates. Also, input events were handled to capture mouse events and keyboard events.

4.10. Prototype 9 - Refinement and Presentation

Prototype aim: A more accurate physics representation of forklift, rather than a simple bounding box.

4.10.1. A Better Forklift with hkpListShapes

To create a more realistic physics representation of the forklift, it would have to be modelled by more than just a single box. The forklift main body was created out of several shapes, which were grouped together in a compound body using a Havok *hkpListShape*.

The biggest section of the main body was a normal *hkpConvexShape* and the other parts of the forklift were created as *hkpConvexTranslateShapes*. Snippet 5 illustrates.

```
// Create the shapes
hkVector4 boxRadii( (hkReal)FORKLIFT_WIDTH,
                    (hkReal)FORKLIFT_WIDTH,
                    (hkReal)FORKLIFT_WIDTH );
hkpConvexShape* shape = new hkpBoxShape( boxRadii , 0 );

hkVector4 boxRadiiSmall((hkReal)FRONT_WIDTH,
                        (hkReal)FRONT_HEIGHT,
                        (hkReal)FRONT_DEPTH );
hkpConvexShape* shapeSmall = new hkpBoxShape( boxRadiiSmall , 0 );

hkVector4 liftRadii((hkReal)LIFT_WIDTH,
                    (hkReal)LIFT_HEIGHT,
                    (hkReal)LIFT_DEPTH );
hkpConvexShape* liftShape = new hkpBoxShape( liftRadii , 0 );

// We'll basically have to create a 'List' Shape
// (ie. a hkpListShape) in order to have many
// shapes in the same body. Each element of the list
// will be a (translated) hkpShape, ie.
// a hkpConvexTranslateShape (which basically is a
// (geometry,translation) pair).
// The hkpListShape constructor needs a pointer to an
// array of hkShapes, so we create an array here, and push
// back the hkTransformShapes as we create them.
hkInplaceArray<hkpShape*,2> shapeArray;

// Create main body
{
        shape->addReference();
        shapeArray.pushBack(shape);
}

// Create main body
{
        hkVector4 v(0.0f,0.0f,(hkReal)FRONT_Z_TRANSLATE);
        hkpConvexTranslateShape* cubeTrans
                = new hkpConvexTranslateShape( shapeSmall, v );
        shapeArray.pushBack(cubeTrans);
}

// Now we can create the compound body as a hkpListShape
hkpListShape* listShape;
{
        listShape = new hkpListShape(shapeArray.begin(),
                                     shapeArray.getSize());
        shape->removeReference();
        shapeSmall->removeReference();
        for (int i = 0; i < shapeArray.getSize(); ++i)
        {
                shapeArray[i]->removeReference();
        }
}
```

Snippet 5: Havok Listshape

4.10.2. The Forks

The forks of the forklift were modelled as two thin, long, cuboids, which approximated the real Lego forks. Originally it was attempted to attach these to the main body using physics hinges, and various other types of joint. After much trial and error, it was decided to keyframe these also, as this was more stable. The forks are keyframed to the correct position and orientation simply by taking the position and orientation of the main body (already calculated from fiduciary markers), and offsetting the position using the look vector derived from the main body. A look vector is a vector that represents the direction in which an object is pointing. Obtaining this look vector was not a simple process, and involves some complex mathematics. In short, the original look vector of the forklift is rotated by a matrix that is constructed from a quaternion (a quaternion is a way of representing a rotation); this quaternion is taken directly from the physics engine. In the end, the code looks fairly simple, but it was the difference in the Havok and ARToolkit coordinate systems that caused the hardship. The following code achieves the desired result; a look vector from a quaternion that is derived from an ARToolkit transformation matrix:

```
// get the look vector from a quaternion
void Forklift::getMemberLookFromQuat(hkpRigidBody* body,
                                     hkVector4 &input,
                                     hkQuaternion &quat)
{
        if (body == NULL)
        {
                body = m_forkliftBodies[0];
        }

        hkVector4 look;
        look.set(0, 0, -1);

        hkQuaternion quatLocal;
        quatLocal.set(quat(0), quat(2), -quat(1), quat(3));

        quatLocal.setInverse(quatLocal);
        quatLocal.normalize();
        hkRotation matrix;
        hkQsTransform trans;

        matrix.set(quatLocal);
        trans.setIdentity();
        trans.setRotation(matrix);

        look.setTransformedInversePos(trans, look);

        look.normalize4();

        input = look;
}
```

4.10.3. Legged Boxes

In real life, crates designed to be picked up by a forklift usually have some sort of legs or raised platform so that the forks can be inserted underneath the main body of the crate. A similar design was used for the virtual boxes. The previously mentioned hkpListShapes were used to create such boxes. Figure 30 shows the profile of the design for one of these legged boxes.

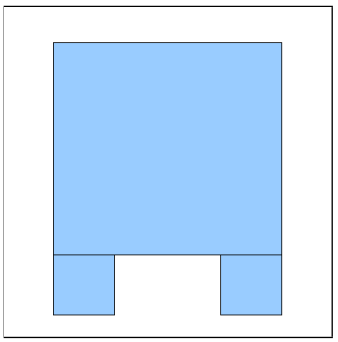


Figure 30: Legged box design

4.10.4. Camera Problems

At this stage it was attempted to fine tune the registration to make it as accurate as possible. It seemed that after a certain time of the day, ARToolkit simply could not register the markers. Therefore coding could only be done during the day time. For some reason natural light was better than artificial light; possibly the 50 Hertz of the artificial light was interfering in some way. This was mostly solved by the acquisition of a new camera that had a higher refresh rate. However, the majority of the project had been constructed during daylight hours because of this issue.

To combat this limitation so that coding on some sections of the program could continue; a flag in the code could be set that faked a camera. Instead of ARToolkit providing the matrix that sets up the OpenGL camera (see section 4.2.1), a standard projection matrix was created so that the physics etc. could be viewed without a camera even being attached. However, obviously this only applies to a small part of the coding of the project, most of which required both a camera and decent lighting conditions.

4.10.5. Improved Remote Control Program

At this juncture, the forklift remote control program was rewritten in C++. Code for directly controlling the forklift over a Bluetooth serial connection was used (with permission) thanks to code developed by Daniel Berger of the Max Planck Institute for Biological Cybernetics [48].

This method of directly accessing the Lego hardware has some severe limitations, such as that the motors can essentially only be turned on and off. With Lejos the wheels can be controlled to within a degree of rotation. With this new method a crude system of timing was used; the times originating from trial and error.

However, this was still preferable to using Lejos, because the remote control code could be integrated into the main program; thus the actions of the real forklift could be more easily reflected in the virtual one.

The remote control functionality was embedded into a remote control dialog, with clickable buttons. The buttons had icons that indicated the command that would be sent to the forklift (The movement keys had arrow icons). The button was highlighted when clicked, i.e., the icon changed so that the user knew it was depressed. The depressed icons were the same as the unpressed ones, except that they were made brighter. When a user depressed one button, all the others went into the unpressed state.

4.10.6. Virtual Fidelity

In their 1999 paper "Real Time Virtual Humans" [49], Badler et al coined the term *virtual fidelity*. This term describes the accuracy of a representation or a simulation when compared to the real world. Badler et al posit that in virtual environments, fidelity in different areas is important, depending on the application. Also, and more importantly, an application may be more successful if certain aspects of a simulation are *not* entirely realistic. For example, in 3D graphics, the attenuation of light is not usually simulated to be the exactly the same as the attenuation in real life.

It was found that setting gravity to the approximate of real life gravity (approximately $10ms^{-2}$) did not produce an entirely convincing result. Although this is the correct value for the simulation, it seemed that the virtual boxes moved somewhat sluggishly. Instead gravity was set to $3500ms^{-2}$, producing a more realistic result. This is an example of virtual fidelity. However, there is another reason that gravity was set to such a high value. When the keyframed forklift intercepted the virtual boxes they would sometimes interpenetrate, causing the boxes to fly off into the scene at a great velocity. This interpenetration was caused by the fact that the forklift was keyframed, and it was forced inside the virtual boxes. To combat this effect, the linear damping value for the simulation was set to a high value. The default is value is 0.05, but it was set to 0.75 for this project. The official Havok documentation (included with the free download) explains linear damping:

> Linear damping reduces the movement of the object over time, and angular damping performs the same function but with the rotation of the object e.g. if you set the linear damping to 0.1f, then every second about 10% of the objects linear velocity will be removed. (Or to be precise: newVelocity = oldVelocity * exp(-damping*deltaTime))

It can be seen from this that 0.75 is a very high value indeed, since every second 75% of the objects linear velocity will removed. This helps to explain the almost absurd value of gravity that is being used. These values for the world parameters gave the most convincing result of a believable simulation of real physics.

4.10.7. Results

Figure 31 shows the results of the keyframed compound forklift. The forklift shown is made of two boxes that represent the main body. The forks can be seen to the left of the main body.

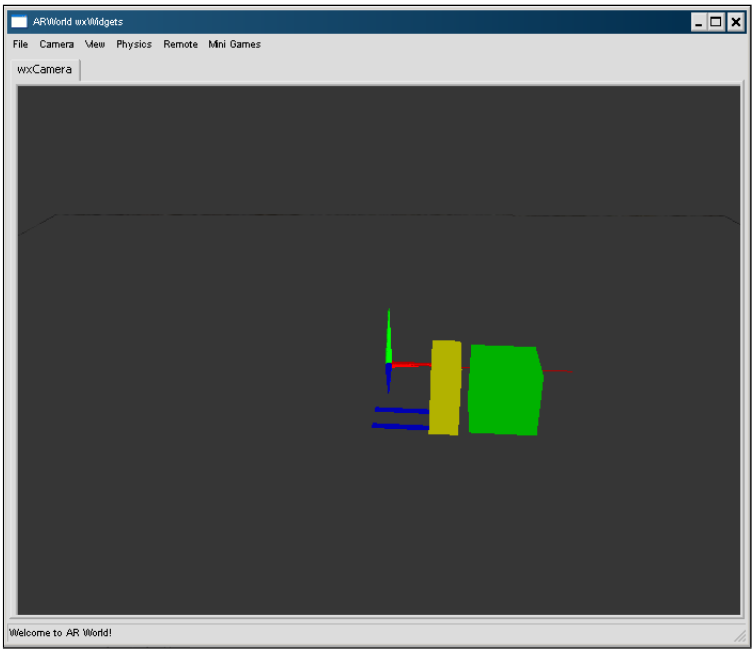


Figure 31: Compound Forklift

4.11. Prototype 10 - The End

Prototype aim: This is the final prototype, and will include any secondary features that time allows. Secondary features could include:

- **Other more advanced image processing techniques**
- **Depth perception to accurately locate objects in 3D space**
- **Training the system to recognise and include new objects into the physics simulation**

None of these secondary features were implemented. The major focus at this stage was to get the system working well with the current functionality.

At this stage the mini-games were created for evaluation purposes (See section 6.1.1.) These were essentially dialogs that have a timer that shows the time taken for a task (see figure 32).

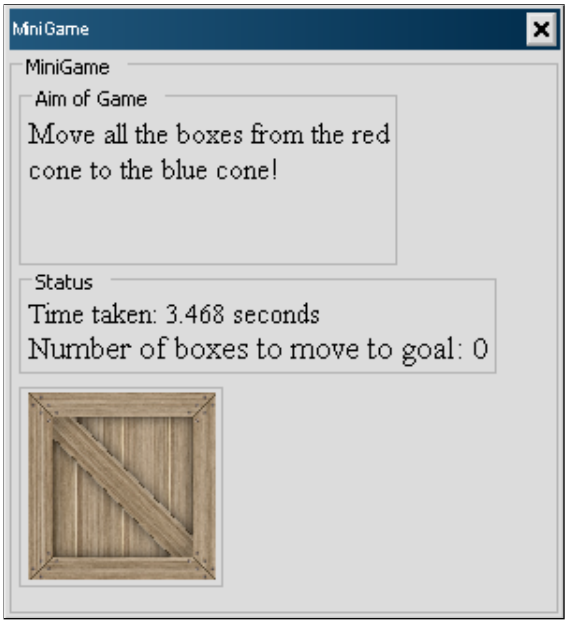


Figure 32: A mini-game

Figure 33 shows the final class diagram for the project, including the main wxWidgets classes.

During this prototype the remote control code was modified to also accept input from an XBOX controller. Microsoft's XInput library was used to achieve this. It was easier for people to control the forklift using an XBOX controller rather than using a keyboard; especially whilst wearing a HMD.

At this stage, improved patterns were designed for marker detection. As mentioned in section 4.4.1, the built in patterns supplied with ARToolkit were being utilised. These were the letters A, B, C, D, G , and F. Some of these patterns are quite similar to each other (especially C and G), so they were sometimes confused by ARToolkit. Using the

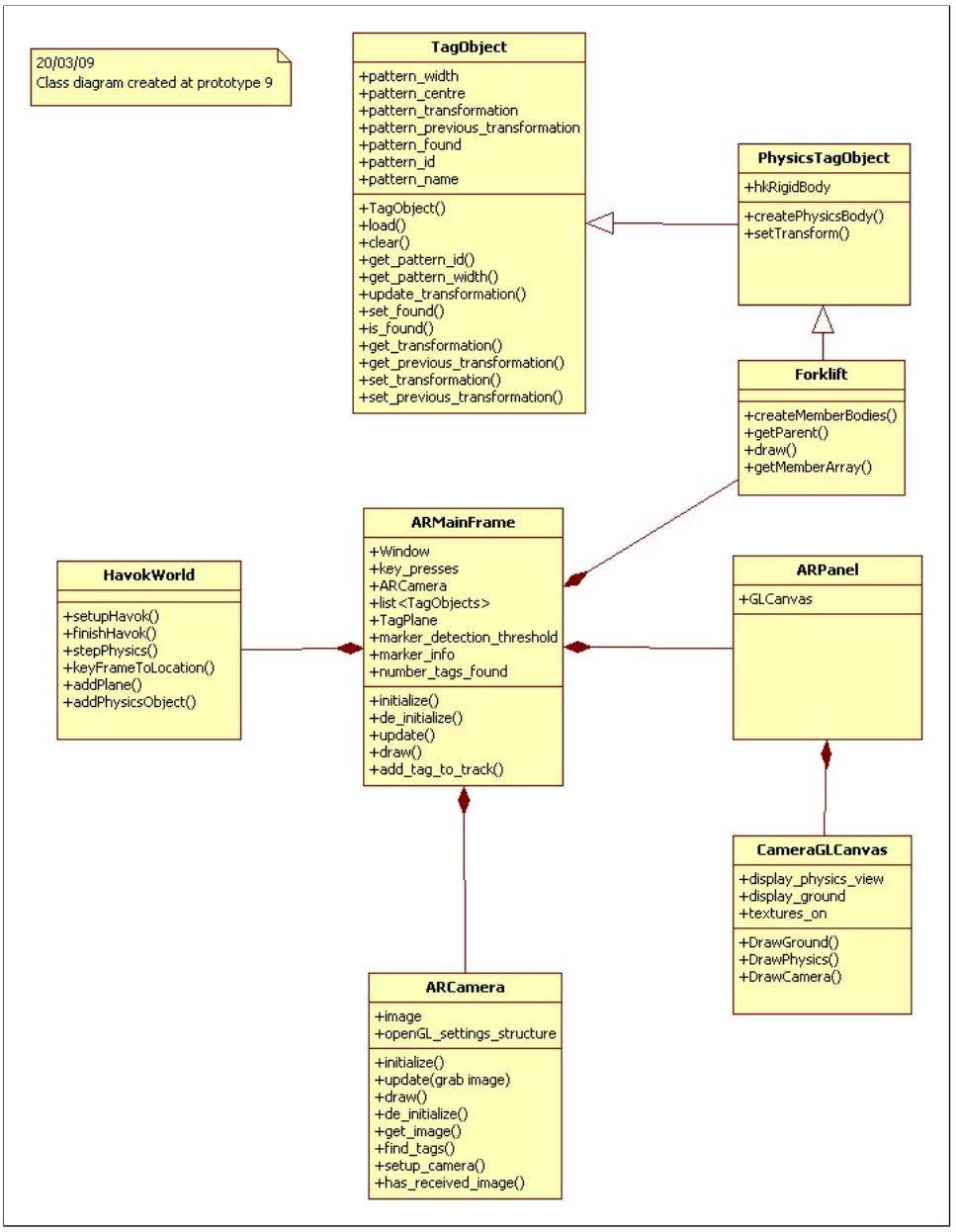


Figure 33: Final design of the system

GNU Image Manipulation Program (GIMP)[21], new patterns were constructed; patterns that were completely dissimilar from each other (see figure 34).

4.11.1. Improved Forklift and Program Parameters

The first version of the Lego forklift created in prototype 7 used a simple cog system to lift the forklift. Although this worked, the forks did not lift very high off the ground. Before the evaluation proceeded, a new improved version of the forklift was constructed from scratch. This version used a pulley system to lift the forks; thus they lifted much

[21] http://www.gimp.org/

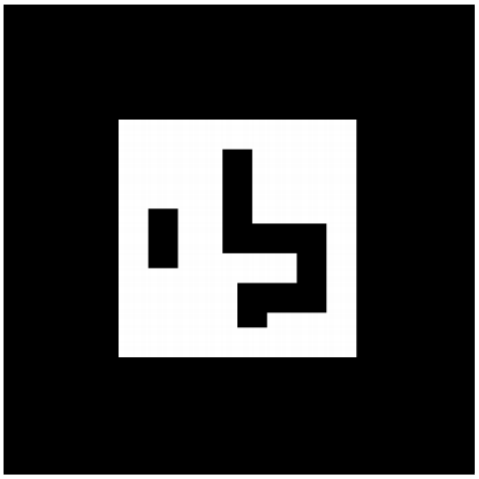


Figure 34: An improved pattern

higher. The pulleys were made of string. The system had to recalibrated so that the physics forklift was aligned with this new forklift.

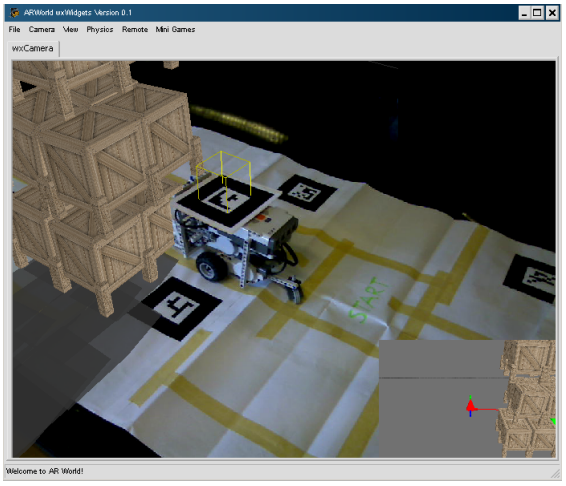
The source of the project depends on a “settings.h” file. This file contains parameters for controlling the entire system; for example the size of the virtual boxes, the size of the ground, and the length of the forklift can all be controlled by changing numbers in this file. Currently the project has to be compiled when changes are made to this file, but for future work, this file could be created in XML, so that settings can be read in without a recompile.

4.11.2. Making Use of the Environment

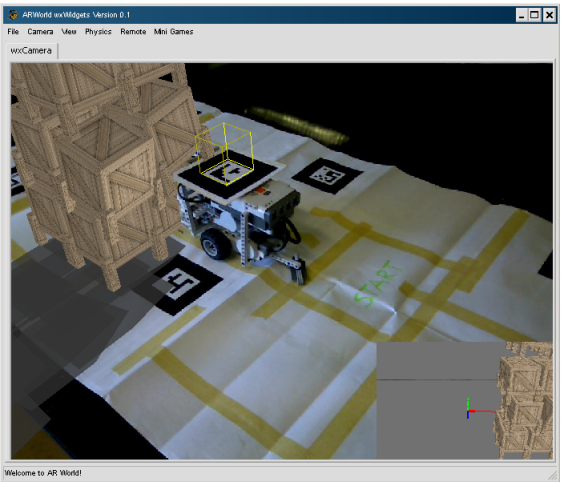
Several other things were done in this prototype to make the virtual boxes seem more realistic. Firstly, shadows were introduced for the boxes. To keep things simple the compound shape of the box was not taken into account, but it was assumed that the shadows would be based on the boxes being square. Secondly, the edges of the multi-pattern were aligned with the edges of the physics plane that was used to represent the ground. In this way, if the boxes were pushed past the edges of the table, they would appear to fall off to the ground. This was a simple adjustment, but it added to the realism significantly.

4.12. Conclusion

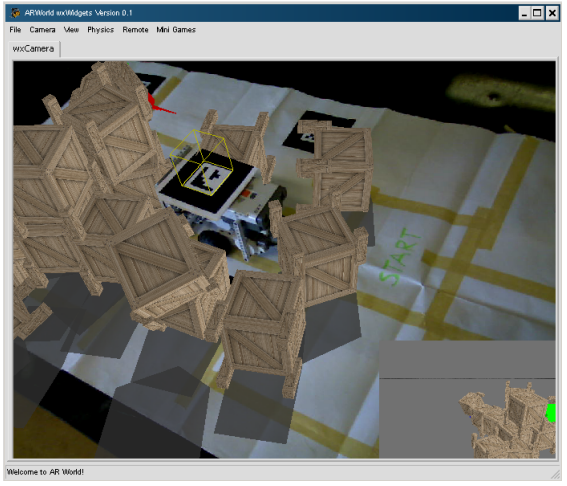
Figure 35 shows some of the final results of the project. A remote controlled forklift is in (a) driving towards and intact wall; in (b) pushing the wall over, and in (c) has destroyed the completely.



(a) Intact wall



(b) Falling over



(c) Destroyed wall

Figure 35: Destruction of a wall

The system works better than was expected given the short duration of this project. The biggest challenge with the implementation was integrating the two different coordinate systems of ARToolkit and Havok. The primary goal of real and virtual objects interacting realistically was achieved, and users of the system found picking up virtual boxes to be comparable in believability to picking up real boxes (see section 6.2.2).

The previous sections of the dissertation described the steps taken in the construction of the project. The next section of the dissertation describes the testing of the system after the final prototype had been completed.

5. Testing

Testing is the procedure used to determine the correctness, completeness and quality of a piece of software. Testing can never completely ascertain correctness under all possible scenarios.

In general, testing objectives are:

1. The process of executing a program with the objective of finding errors.
2. A good test case is one with a high probability of discovering an error.
3. A successful test case is one that finds a formerly unidentified error.

Testing cannot show the absence of errors, it can only show that they are present. With this in mind, the test cases were designed with the intent of breaking the system.

5.1. Black Box Testing

As the name suggests, black box testing is a testing strategy whereby no knowledge of internal structure of the program is required [50].

Black box tests can be functional or non-functional; in the case of this project they will be functional. Functional testing is essentially verifying that the functions of a system are present and working properly. Non-functional testing is testing the program for non-functional attributes, for example performance testing or localisation testing.

Black box testing strategies can be broken into two main groups:

- Testing in which the user plays a role of tester
- User is not required

For the first group of tests, the tester should be knowledgeable about the system so that as a tester they are able to identify if correct responses are given by the system. It is traditional for someone other than the programmer to handle the testing, as the programmer would know too much about the inner workings of the system. For this project, this is not possible because only one person is involved, so the programmer and the tester will be the same person.

5.1.1. Advantages and Disadvantages of Black Box Testing

The advantages of black box testing are:

- The tester doesn't need to know the implementation, or even the programming language
- The tester and programmer are independent
- Tests are done from a user perspective
- Ambiguities in system specification will be exposed

- Test cases can be designed immediately after the specification is complete

The disadvantages of black box testing are:

- Only a subsection of the range of inputs can be tested
- Without clear specifications, test cases are hard to design
- Some program paths may be untested
- Cannot be directed as specific areas of code that are complex and therefore more error prone

5.2. Test Plan

The objectives of this test plan are to find any major errors or bugs. Bugs will be classified as in table 2.

Table 2: Bug classification

Class	Severity of bug
A	Radically effects the functionality of the system, major data loss or program crash
B	Serious cosmetic error or minor affection of functionality, but program continues
C	Minor cosmetic issues such as spelling mistakes

The levels of "acceptable" bugs for this project are: no class A bugs, at most one class B bug and five class C bugs. Level A bugs would not be acceptable as these would cause a critical failure. The levels for B and C bugs were chosen simply because they seemed acceptable. A few level C cosmetic bugs would not matter much for this project, and since the program is not being commercially sold a single level B would not matter either. If more bugs are found in each of these classes, attempts will be made to fix them, and then the test plan will be executed again.

5.2.1. Assumptions and Testing Scope

As described in section 3, many APIs and libraries have been integrated into this project:

- Havok
- OpenGL
- wxWidgets
- ARToolkit
- XInput
- DirectShow
- Glut

It is assumed that these pieces of software have been tested by their respective manufacturers, and so only the novel code that integrates this software is to be tested.

It should be noted at this point that because cameras are involved, lighting conditions have a big impact on the results of some of these tests. It is assumed that *ideal* lighting conditions are present during testing. Ideal lighting conditions are defined simply as normal daylight conditions. Many of the tests performed could be made to "fail" by changing the lighting conditions.

5.2.2. Testing Environment

The camera used during testing was the Logitech QuickCam S7500 which has a resolution of 960 by 720 pixels. The computer used was an Apple MacBook Pro Rev 3.

5.3. Results

Table 3: Camera Test Cases

Functionality	Initial state	Input	Expected output	Result
Camera connection	Camera installed and connected	Launch Application	Application started	Pass
Camera connection	Application started	Click Attach Camera	Displaying camera images	Pass
Camera disconnection	Camera installed and connected	Launch Application	Application started	Pass
Camera disconnection	Application started	Click Attach Camera	Displaying camera images	Pass
Camera disconnection	Displaying camera images	Click Dettach Camera	Not displaying camera images	Pass
Camera reconnection	Camera installed and connected	Launch Application	Application started	Pass
Camera reconnection	Application started	Click Attach Camera	Displaying camera images	Pass
Camera reconnection	Displaying camera images	Click Dettach Camera	Not displaying camera images	Pass
Camera reconnection	Application started	Click Attach Camera	Displaying camera images	Pass
Camera error handling	Camera installed but not connected	Launch Application	Application started	Pass
Camera error handling	Application started	Click Attach Camera	No camera error message	Pass
Camera error handling	Camera installed and connected	Launch Application	Application started	Pass
Camera error handling	Application started	Click Attach Camera	Displaying camera images	Pass
Camera error handling	Displaying camera images	Click Attach Camera again	Camera already connected warning message	Pass
Camera settings	Camera installed and connected	Launch Application	Application started	Pass
Camera settings	Application started	Click Attach Camera	Displaying camera images	Pass
Camera settings	Displaying camera images	Click Camera Settings	Setting dialog is shown	Pass
Camera settings	Setting dialog shown	Adjust Threshold bar to lower limit	Augmentations disappear	Pass
Camera settings	Setting dialog shown	Adjust Threshold bar to previous position	Augmentations return	Pass
Camera settings	Setting dialog shown	Adjust Threshold bar to upper limit	Augmentations disappear	Pass
Camera settings	Setting dialog shown	Adjust Threshold bar to previous position	Augmentations return	Pass

Table 4: View Test Cases

Functionality	Initial state	Input	Expected output	Result
View Changes	Camera functioning	Launch Application	Application started	Pass
View Changes	Application started	Click Attach Camera	Displaying camera images	Pass
View Changes	Displaying camera images	Toggle physics view	Only camera view seen	Pass
View Changes	Camera functioning	Launch Application	Application started	Pass
View Changes	Application started	Click Attach Camera	Displaying camera images	Pass
View Changes	Displaying camera images	Toggle camera/-physics view	Only physics view seen	Pass
View Changes	Only physics view seen	Toggle camera/-physics view again	Both views seen	Pass
View Changes	Camera functioning	Launch Application	Application started	Pass
View Changes	Application started	Click Attach Camera	Displaying camera images	Pass
View Changes	Displaying camera images	Toggle physics view	Only camera view seen	Pass
View Changes	Only camera view seen	Toggle camera/-physics view	Only camera view seen	Pass
View Changes	Camera functioning	Launch Application	Application started	Pass
View Changes	Application started	Click Attach Camera	Displaying camera images	Pass
View Changes	Displaying camera images	Toggle camera/-physics view	Only physics view seen	Pass
View Changes	Only physics view seen	Toggle physics view	Only physics view seen	Pass
View Changes	Camera functioning	Launch Application	Application started	Pass
View Changes	Application started	Click Attach Camera	Displaying camera images	Pass
View Changes	Displaying camera images	Toggle occlusion	Solid forklift seen	Pass
View Changes	Displaying camera images, Solid forklift seen	Toggle occlusion again	Forklift is “transparent”	Pass
View Changes	Camera functioning	Launch Application	Application started	Pass
View Changes	Application started	Click Attach Camera	Displaying camera images	Pass
View Changes	Displaying camera images,	Toggle solid ground	Solid ground seen	Pass
View Changes	Displaying camera images, solid ground seen	Toggle solid ground again	Wireframe ground seen	Pass

Table 5: View Test Cases

Functionality	Initial state	Input	Expected output	Result
Physics insertions	Camera installed and connected	Launch Application	Application started	Pass
Physics insertions	Application started	Click Attach Camera	Displaying camera images	Pass
Physics insertions	Displaying camera images	Add Crate Dialog	Add Physics Item Dialog Appears	Pass
Physics insertions	Add physics item dialog active	Hit Ok	Crate is added to world, dialog disappears	Pass
Physics insertions	Displaying camera images	Add Wall Dialog	Add Physics Item Dialog Appears	Pass
Physics insertions	Add physics item dialog active	Hit Ok	Wall is added to world, dialog disappears	Pass
Physics insertions	Displaying camera images	Add Ball Dialog	Add Physics Item Dialog Appears	Pass
Physics insertions	Add physics item dialog active	Hit Ok	Ball is added to world, dialog disappears	Pass
Physics insertions	Displaying camera images	Reset Physics	Physics Objects disappear	Pass
Physics insertions	Displaying camera images	Add Crate Dialog	Add Physics Item Dialog Appears	Pass
Physics insertions	Add physics item dialog active	Hit Ok	Crate is added to world, dialog disappears	Pass
Physics reset	Camera installed and connected	Launch Application	Application started	Pass
Physics reset	Application started	Click Attach Camera	Displaying camera images	Pass
Physics reset	Displaying camera images, no objects in world	Reset Physics	Still no objects in the world	Pass
Physics reset	Displaying camera images	Add Crate Dialog	Add Physics Item Dialog Appears	Pass
Physics reset	Add physics item dialog active	Hit Ok	Crate is added to world, dialog disappears	Pass
Physics reset	Displaying camera images, objects in world	Reset Physics	Objects disappear from the world	Pass

Table 6: Remote Control Test Cases

Functionality	Initial state	Input	Expected output	Result
Remote control	Camera installed and connected, bluetooth on, robot on	Launch Application	Application started	Pass
Remote control	Application started	Click Attach Camera	Displaying camera images	Pass
Remote control	Displaying camera images	Click Remote Control...	Bluetooth connects, control dialog appears	Pass
Remote control	Remote control dialog active	Click Remote Dialog	Bluetooth disconnects, control dialog disappears	Pass
Remote control	Camera installed and connected, bluetooth on, robot OFF	Launch Application	Application started	Pass
Remote control	Application started	Click Attach Camera	Displaying camera images	Pass
Remote control	Displaying camera images	Click Remote Control...	Connection times out, control dialog does not appear	Fail
Remote control	Camera installed and connected, bluetooth on, robot on	Launch Application	Application started	Pass
Remote control	Application started	Click Attach Camera	Displaying camera images	Pass
Remote control	Displaying camera images	Click Remote Control...	Bluetooth connects, control dialog appears	Pass
Remote control	Remote control dialog active	Click Left Button	Forklift goes left	Pass
Remote control	Remote control dialog active	Click Right Button	Forklift goes right	Pass
Remote control	Remote control dialog active	Click Stop Button	Forklift stops	Pass
Remote control	Remote control dialog active	Click Forward Button	Forklift goes forward	Pass
Remote control	Remote control dialog active	Click Backward Button	Forklift goes backward	Pass
Remote control	Remote control dialog active	Click Fork up Button	Forklift stops and Forks go up	Pass
Remote control	Remote control dialog active	Click Fork down Button	Forklift stops and Forks go down	Pass

Table 7: More Remote Control Test Cases

Functionality	Initial state	Input	Expected output	Result
Remote control	Camera installed and connected, bluetooth on, robot on	Launch Application	Application started	Pass
Remote control	Application started	Click Attach Camera	Displaying camera images	Pass
Remote control	Displaying camera images	Click Remote Control...	Bluetooth connects, control dialog appears	Pass
Remote control	Remote control dialog active	Press A Key	Forklift goes left	Pass
Remote control	Remote control dialog active	Press D Key	Forklift goes right	Pass
Remote control	Remote control dialog active	Press Space Key	Forklift stops	Pass
Remote control	Remote control dialog active	Press W Key	Forklift goes forward	Pass
Remote control	Remote control dialog active	Press S Key	Forklift goes backward	Pass
Remote control	Remote control dialog active	Press R Key	Forklift stops and Forks go up	Pass
Remote control	Remote control dialog active	Press F Key	Forklift stops and Forks go down	Pass

Table 8: Mini Game Test Cases

Functionality	Initial state	Input	Expected output	Result
Mini games	Application started, Hardware Ready	Click Attach Camera	Displaying camera images	Pass
Mini games	Displaying camera images	Click Pick Up One Mini Game	Mini game dialog appears, physics reset, one block appears	Pass
Mini games	Mini game dialog active	Click Wall Bash Mini Game	Mini game dialog is replaced, physics reset, wall appears	Pass
Mini games	Mini game dialog active	Click Football One Mini Game	Mini game dialog is replaced, physics reset, one block appears and two cones	Pass
Mini games	Mini game dialog active	Close Mini Game Dialog	Mini game dialog disappears	Pass
Mini games	Displaying camera images	Click Move Blocks Mini Game	Mini game dialog appears, physics reset, wall appears, and two cones	Pass
Mini games	Application started, Hardware ready	Click Attach Camera	Displaying camera images	Pass
Mini games	Displaying camera images	Click Pick Up One Mini Game	Mini game dialog appears, physics reset, one block appears	Pass
Mini games	Mini game dialog active	Complete mini game objective	Mini game clock stops	Pass
Mini games	Mini game dialog active	Click Wall Bash Mini Game	Mini game dialog is replaced, physics reset, wall appears	Pass
Mini games	Mini game dialog active	Complete mini game objective	Mini game clock stops	Fail
Mini games	Mini game dialog active	Click Football One Mini Game	Mini game dialog is replaced, physics reset, one block appears and two cones	Pass
Mini games	Mini game dialog active	Complete mini game objective	Mini game clock stops	Pass
Mini games	Displaying camera images	Click Move Blocks Mini Game	Mini game dialog is replaced, physics reset, wall appears, and two cones	Pass
Mini games	Mini game dialog active	Complete mini game objective	Mini game clock stops	Pass

Table 9: Using Markers Test Cases

Functionality	Initial state	Input	Expected output	Result
Marker movement	Camera installed and connected, bluetooth on, robot on	Launch Application	Application started	Pass
Marker movement	Application started	Click Attach Camera	Displaying camera images	Pass
Marker movement	Displaying camera images	Remove Forklift Marker	Forklift lost from scene	Pass
Marker movement	Displaying camera images	Insert Forklift Marker	Forklift reappears	Pass
Marker movement	Camera installed and connected, bluetooth on, robot on	Launch Application	Application started	Pass
Marker movement	Application started	Click Attach Camera	Displaying camera images	Pass
Marker movement	Displaying camera images	Cover one plane marker	Scene still displays	Pass
Marker movement	Displaying camera images	Cover two plane marker	Scene still displays	Pass
Marker movement	Displaying camera images	Cover three plane marker	Scene still displays	Pass
Marker movement	Displaying camera images	Cover four plane marker	Scene still displays	Pass
Marker movement	Displaying camera images	Cover five plane marker	Scene still displays	Pass
Marker movement	Displaying camera images	Cover six plane marker	Scene does not display	Pass

5.3.1. Analysis of Results

Out of the above tests, two failed. These were:

1. When the robot is OFF and a remote control connection is attempted, the remote dialog should not appear. It does however appear momentarily before disappearing.

2. When the "wall bash" or "Knock Them Down" mini-game is completed, the timer should stop. It does not; this is most likely because the game end conditions have not been met.

Of these failures, the first is considered a class C bug, the second a class B. These failures are deemed tolerable, as they have not exceeded the limits set in section 5.2.

Overall the system is stable. The final prototype that was tested has never been seen to fatally crash.

6. Evaluation

6.1. Evaluation Approach

Evaluation consisted of two sections. Firstly, two experiments were run. Each experiment consisted of two parts; it was run once in the "virtual" world, and once in the real world as a control. Although simplistic, it is hoped that these mini-games will highlight the areas in which the system is not realistic and fails to be convincing. These areas could form the basis for future work and improvement. The goal of this part of the evaluation is to discover whether the virtual system is *comparable* in believability to the real world. I.e., can the participants be convinced that they are manipulating real boxes.

Secondly, a short survey was used help to evaluate the system. The survey aims to discover how aware people are of the technology that they are using and of the fact that they are interacting with objects that do not actually physically exist.

6.1.1. Experiments

The two experiments took the form of "mini-games" that users attempted. The time taken and number of attempts for each mini-game was recorded. People tend to be somewhat distracted by the "wow factor" of augmented reality when they first experience it. This factor was offset by allowing participants to experiment with the system before starting any mini-games.

The mini-games were:

1. Pick up a single box (the box must simply be elevated in the air using the forks)
2. Forklift football, a box must be moved into a marked goal

At the end of each task any comments that the participants made were recorded, in conjunction with the time taken and number of attempts. The real world experiments were close replications of the virtual mini-games. The real boxes were constructed of cardboard, and were approximately to the same relative size and had the same center of gravity as the virtual boxes. The boxes were put into approximately the same relative starting positions. Whether or not the end conditions of the mini-games had been met was determined by the experimenter, and all times were measured using a stopwatch.

6.1.2. Survey

The questions in the survey were psychometric in nature, therefore it was proposed that a typical five-level Likert *item*[51] be used for the question format.

The questions were as follows:

- How conscious of the technology were you?
- How would you rate the believability of the virtual tasks as compared to the non-virtual tasks?
- Did you find the field of view of the HMD to be adequate?

- Rate the level of discomfort experienced as a result of using the system for the duration of the experiments:
  - fingers
  - eyes
  - neck
  - other (please state area and discomfort level)

The full survey can be found in Appendix A.

6.2. Evaluation Results

Eleven students from the DIT School of Computing took part in this evaluation. Most of the participants had little or no experience with augmented reality (AR). However, the entire group was very computer literate as would be generally expected of Computer Science students. Each participant was given an identical evaluation "script" so that they each had the same information for the evaluation. (See appendix A).

6.2.1. Evaluation Environment

The camera used during testing was the Logitech QuickCam S7500 which has a resolution of 960 by 720 pixels. The computer used was an Apple MacBook Pro Rev 3. The Trivision ARVision 3D HMD was used to view the augmentations, while the Logitech camera was used to view them. This was because the cameras built into the HMD are of a poor quality. Duct tape was used to secure the Logitech camera to the ARVision HMD (see figure 43).

6.2.2. Mini-Game Results

Each mini-game experiment consisted of two parts, the real task and the corresponding virtual task. The tasks were labeled as T1-T4; these are as shown in table 10.

Table 10: Mini-games legend

Mini-game section	Description
T1	Real version of the Pick Up One game
T2	Virtual version of the Pick Up One game
T3	Real version of the Forklift Football game
T4	Virtual version of the Forklift Football game

For the analysis of the results, it should be noted that what is being looked for is evidence that the virtual tasks were *comparable* to the real ones, in terms of believability. Obviously it is not expected that the virtual tasks be "more real" than the actual real tasks.

On average it took participants 16.5 seconds to complete the real pick up one task (T1), 25.2 seconds to complete the virtual pick up one task (T2), 19.7 seconds to complete the real forklift football task (T3) and 25.9 seconds to complete the virtual forklift football task (T4)(see Figure 36).

From these figures it can be determined that the virtual tasks (T1 and T3) took on average 8.7 and 6.2 seconds longer than the real tasks (T2 and T4) for the two mini-games. The standard deviations for the tasks T1 to T4 are shown in table 11. The deviations are smaller for the real tasks, at 4.3 and 3.5; for the virtual tasks they are 20.4 and 11.5. This variation is put down to the fact that the physics simulation is not entirely accurate and anomalies can occur that would not happen in real life. Some participants commented that they thought the actions that they had performed should have had different consequences.

However, an important point to note is that people *could complete* the virtual tasks, and they did this in a reasonable time. People found the virtual task harder then the real task

for two reasons. Firstly, the rendering of the boxes makes it hard to tell exactly where the boxes sit on the ground. There was not much height between the box and the table. Better rendering would help in this area; it was for this reason that shadows were implemented for the virtual boxes. Secondly, people did not feel free to move around whilst wearing the HMD. People found it much easier to complete the virtual tasks once this fact was made clear, as they could view the boxes from different angles. People found the HMD equipment to be both bulky and uncomfortable; this may have been a factor in why people thought they should not move around. People will no doubt find it easier to move around with improved hardware.

A paired two-tailed Student's t-test [52] was performed on the time data. This test focuses on determining the probability that the actual mean difference is consistent with zero. I.e. if the result of the t-test returns a small number, then the probability that the difference between the real and virtual times is due to chance is also small. It is paired because of the relationship between the two input data sets, and it is two-tailed because no predictions are being made about the direction of the difference. The hypothesis is that although the virtual tasks took consistently longer than the real ones, the virtual tasks are comparable to the real tasks in terms of the time taken.

The t-test was performed for the pick up one mini-game and returned a p-value of 0.14 (to 2 decimal places). This supports the hypothesis at the 90% confidence level. The t-test was performed on the forklift football mini-game and returned a p-value of 0.06 (to 2 decimal places). This was a better result than for the pick up one mini-game, and supports the hypothesis at just under the 95% confidence level. These results show that the difference in average times between the real and virtual tasks are likely to be statistically significant and not the result of random chance events.

As discussed above in section 6.1.1, the boxes and forklift started in the same positions for both the real and virtual tasks. Also, the real and virtual boxes were of approximately the same size. It must therefore be concluded that the reason for these discrepancies in the average times is likely that the participants did not find the virtual boxes to be convincing facsimiles of the real ones. The real and virtual boxes are the only significant difference between the two tasks.

Table 11: Standard deviations and average times for tasks

Task	**Standard deviation**	**Average time**
T1	4.321195121	16.45454545
T2	20.40043573	25.2
T3	3.55220802	19.72727273
T4	11.51046954	25.90909091

Considering the forklift football mini-game; this game was especially straightforward in both a metaphorical and literal sense. The participants simply had to drive forward in a straight line to complete the task; and yet on average the virtual task took 6.2 seconds longer. This is considered a clear and unequivocal result; people were not as comfortable with the virtual tasks.

The number of attempts was also recorded for each of the tasks (T1 to T4), however only three times out of forty-four was more than one attempt necessary. This means that

approximately 93% of people took were able to complete both the virtual and real tasks in one attempt. Of the three cases where more than one attempt was required, two occurred during virtual tasks and one occured during a real task. These results show that although the participants did take longer over the virtual tasks, they did not find them particularly hard; most participants successfully completing the task at the first attempt.

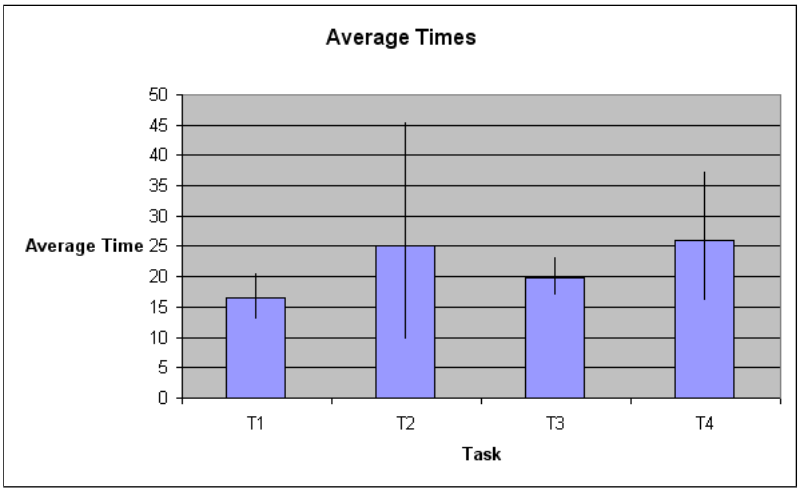


Figure 36: Mini-game average times

Figure 37 shows the participants ratings for the ease of the pick up one tasks (T1 and T2). For the real task (T1), most of the responses were “easy” with 6 participants scoring the ease of the task at this level. For the virtual task (T2), most of the responses occurred at the rating of “moderately easy”, with 5 participants rating the ease of the task at this level. There was equal numbers of ratings of an ease of “very easy”, with 4 ratings each for T1 and T2. These statistics show that participants routinely rated the real task as being *easier* than the corresponding virtual tasks. This is to be expected, considering the fact that the participants also took longer on the virtual tasks than the real tasks.

Similar results are seen in the forklift football tasks (T3 and T4). Figure 38 shows the participants responses for these tasks. For the real task (T3), 7 participants rated the ease of the task as “easy”, and 4 as “very easy”. For the virtual task (T4), the corresponding number of participant ratings for these levels are 6 and 3. Figure 38 also shows that the participants rated the real task “easy“ more often that they did for the virtual task, and the real and virtual tasks have equal ratings of “very easy”. Once again, these statistics are in agreement with the times that were recorded for these tasks; the real task (T3) being performed quicker on average than the virtual task (T4).

Comments made about these mini-games by participants were that they were very easy. This is reflected in the fact that most of the time only one attempt was required to complete the tasks, both real and virtual. The tasks were indeed simplistic and trivial; but they were chosen out of a set that contained more complex mini-games for the two following reasons.

Firstly, it was thought that a higher number of participants would be obtained if the evaluation took a reasonably short amount of time. As it was, the evaluation took a minimum of about ten minutes to complete for a single participant (including the five minute

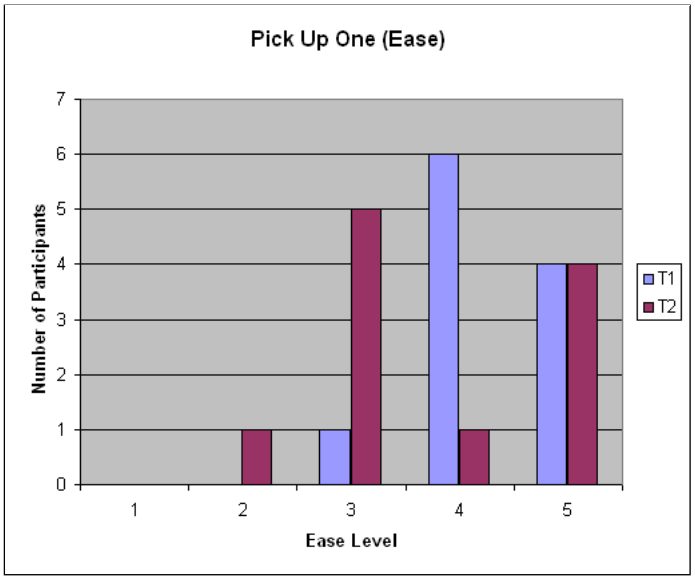


Figure 37: Pick up one ease

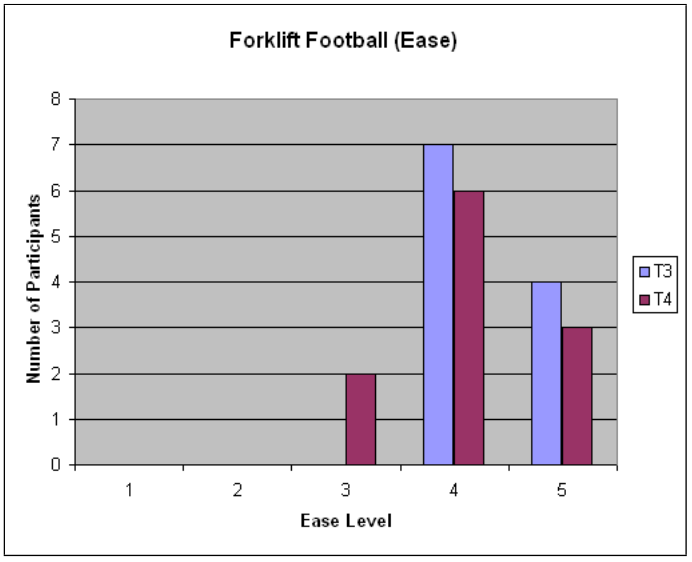


Figure 38: Forklift football ease

"play" time at the beginning).

Secondly, more complex tasks may have introduced other considerations; such as whether the participant fully understood the task, and the skill level in controlling the forklift would become an unnecessary factor.

6.2.3. Survey Results

The average response to the question "How conscious of the technology were you?" was 4.8 (see Figure 39), where 1 was "not at all" and 5 were "very". This question was designed to discover how aware people were of the fact that they were using a head-mounted display (HMD). Although the response indicates obvious consciousness of this fact, it is thought that in the future, improved mixed reality hardware will reduce the awareness of the technology. This will be useful for future comparisons.

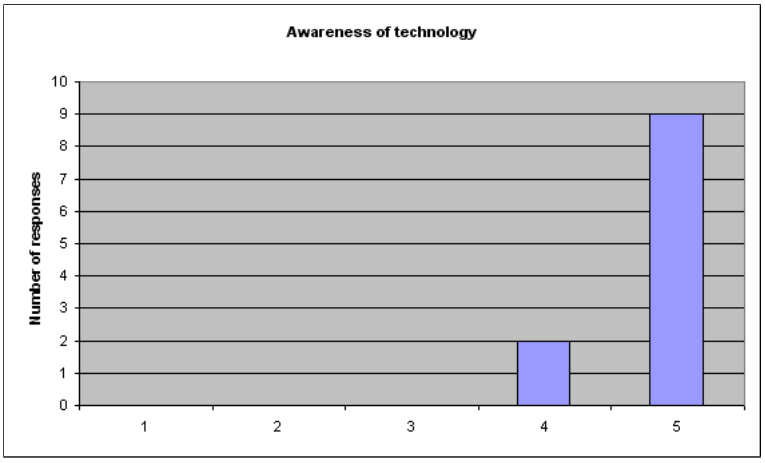


Figure 39: Awareness of technology

The average response to the question "How would you rate the believability of the virtual tasks as compared to the non-virtual tasks?" was 3.7 (see Figure 40), where 1 was "not at all" and 5 was "believable". This was a gratifying result; surpassing what was anticipated. It indicates that the participants found that the virtual tasks acted in the expected manner, Ie. they believed they could pick up virtual boxes with a real forklift.

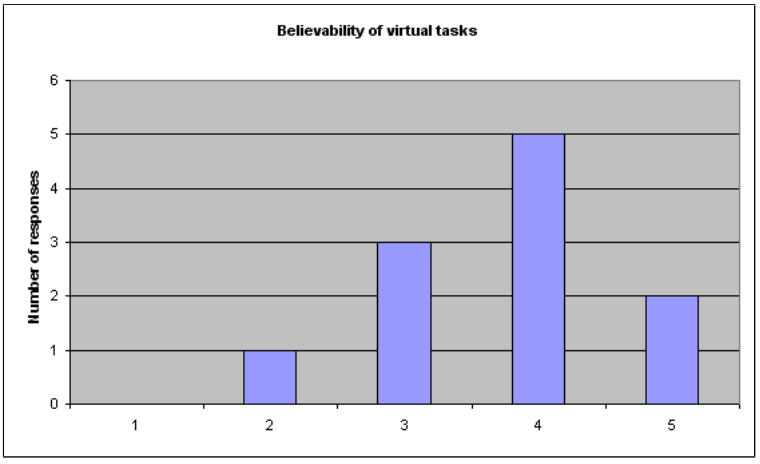


Figure 40: Believability of the virtual tasks

The average response to the question "Did you find the field of view of the HMD to be adequate?" was 3.3 (see Figure 40), where 1 was "not at all" and 5 was "adequate". This

was a higher figure than was expected, suggesting that participants found the HMD field of view to be adequate.

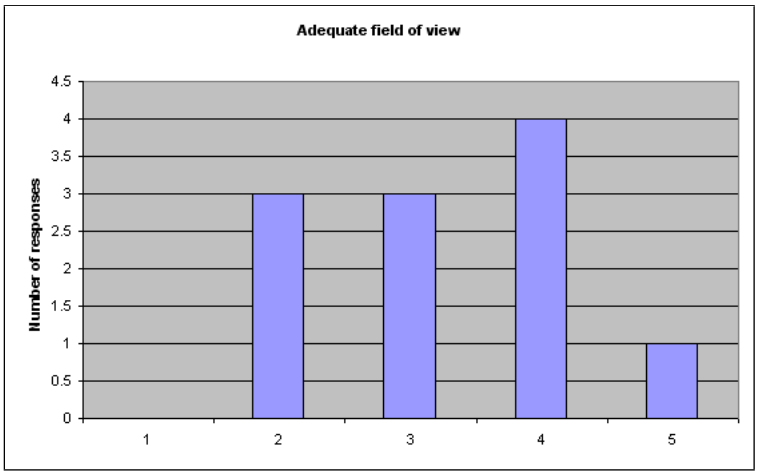


Figure 41: Adequate field of view

The final question was "Rate the level of discomfort experienced as a result of using the system for the duration of the experiments" for the areas of fingers, eyes and neck. This question also included an "other" field so that participants could comment on other discomfort they experienced in other areas of the body. The average discomfort for the areas is as follows: the average felt for the fingers was 1.2, the average felt for the eyes was 3.2, and the average felt for the next was 1.9 (see Figure 42). From these statistics it is concluded that little or no discomfort was felt in the fingers; from controlling the forklift with the XBOX controller[22].

Moderate to large amounts of discomfort was felt in the eyes for participants, due to the HMD. People noticed also that wearing glasses was an impediment. Other comments were that lateral light coming in from the sides of the HMD disturbed the view of the augmented scene. This is undoubtedly one of the more substantial problems with using the HMD.

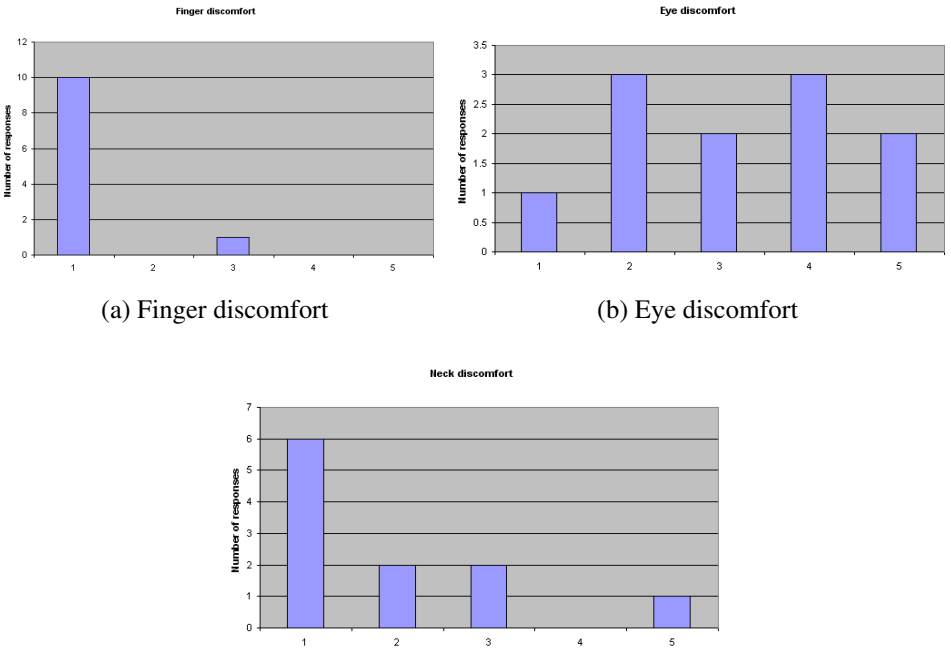
People found that a small amount of discomfort was caused in the neck area.

For the other section, only three people filled this in; all comments related to discomfort for their noses.

The largest amount of discomfort felt was unquestionably in the eyes of the participants. The large amount of discomfort felt in people's eyes was probably a combination of factors. Firstly, the physical displays in the HMD are essentially lights that shine directly into their eyes. This could cause discomfort in itself; however, the displays do appear to flicker which could be a further cause of discomfort.

The headset surely does allow a higher amount of immersion than viewing the augmented scene on a conventional PC monitor. However, the current technology is bulky and in some respects (especially the eyes), uncomfortable to use. However, this technology is young, and it is anticipated that when the technology matures, discomfort will be

[22]http://www.xbox.com/en-US/



(a) Finger discomfort

(b) Eye discomfort

(c) Neck discomfort

Figure 42: Discomfort levels

less of an issue. Hopefully the statistics in this section would prove useful in future study. Figure 43 shows an evaluation participant.

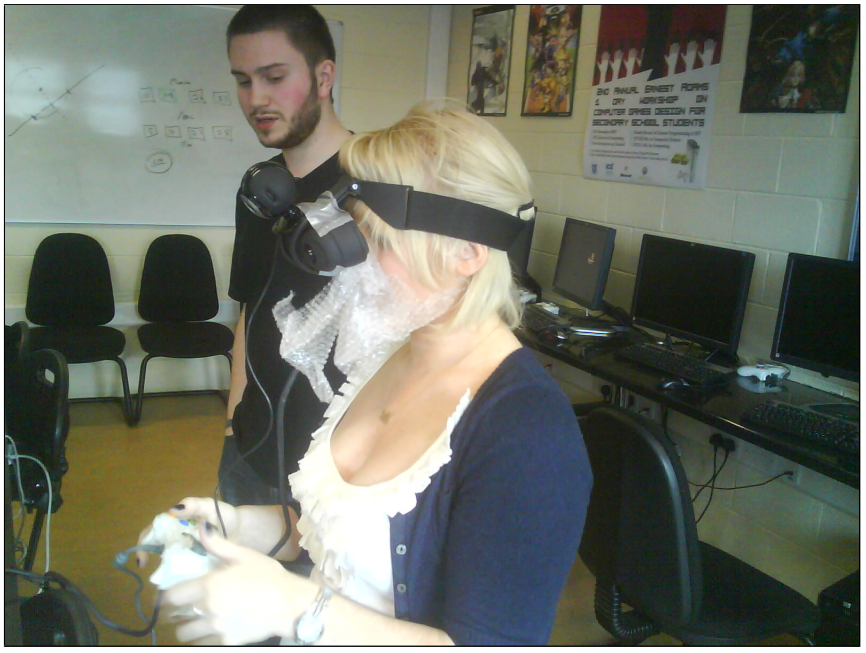


Figure 43: An evaluation participant

6.3. Program Statistics

This section describes some other ways that the system was evaluated, which did not rely on subjective user opinion.

6.3.1. CPU Usage

The program was evaluated in terms of how much CPU it utilised. Since the program has to run in real-time, performance is important. Performance was tested by adding more and more physics boxes into the simulation, and recording the CPU usage. Table 12 shows the results; the setup is the same as the one described in section 6.2.1, the computer used was an Apple MacBook Pro Rev 3. The increase in CPU usage is approximately linear with the increase in number of boxes (see figure 44). Increasing the boxes puts the system under stress in two ways. Firstly, the physics engine load is increased, and secondly more objects have to be rendered. Three hundred and twenty boxes is considered a high value; the fact that an average off the shelf system was able to manage this number of boxes is a positive result.

Table 12: CPU usage for different number of physics objects

Number of boxes	Percent of CPU usage
0	20%
64	25
128	31
192	33
256	36
320	40

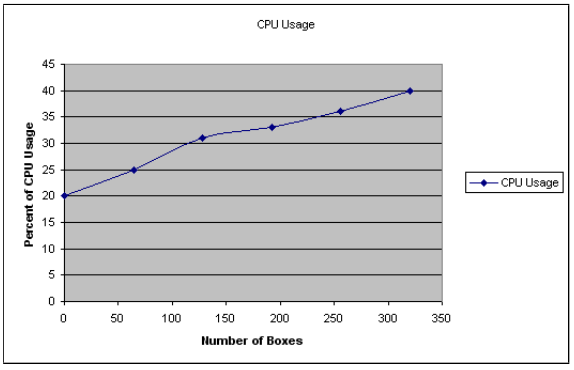


Figure 44: Percent of CPU usage for different box numbers

6.3.2. Marker Detection

To test the robustness of the marker detection, tests were done to see at what distances marker detection of both the table markers and the forklift marker could be successfully

achieved. Figure 45 shows the "zones" that were used for distance evaluation. The concentric circles indicate the distances at which marker detection was tested. Detection was also tested at two different heights for each of these distances. The two heights were half the height of an average person, and the height of an average person (0.8m and 1.75m respectively). Table 13 shows the results of these tests.

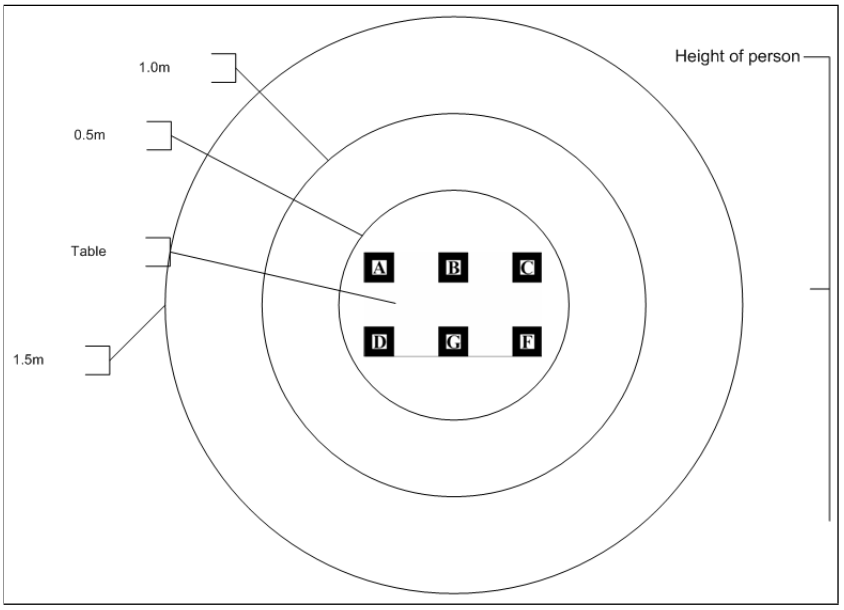


Figure 45: Distance of marker detection

Table 13: Marker detection at distances

Height	Distance	Markers detected
0.8m	0.5m	Yes
	1.0m	Yes
	1.5m	No
1.75m	0.5m	Yes
	1.0m	Yes
	1.5m	Yes

The results shown here in table 13 show that detection is fairly robust even at long distances, but that at acute angles and long distances detection begins to fail. This is a limitation of ARToolkit more than anything else, but because of the large size of the fiduciary markers, detection is acceptable and nobody complained about excessive loss of fiduciary markers during the evaluation. Also, it is unlikely that a user would be situated in the location where detection failed.

6.4. Conclusion

The previous sections described evaluation procedure for this project, and the statistics and results that were gathered. Overall this project is considered to have been a success; participants were able to use the system as intended.

The evaluation of this project shows that the virtual tasks were comparable in ease and time taken to the real ones. From this it can be assumed that the system obtained an acceptable level of believability, which was the primary goal of the project. The CPU usage tests showed that the system can run acceptably on off the shelf hardware, and the marker detection tests showed that marker detection was stable and robust.

The next section will conclude the dissertation with a description of future work, and final observations about this project.

7. Summary and Conclusions

This section will complete the dissertation with ideas as to possible future directions that the project could take. This future work includes the features that were not completed in this project due to lack of time, etc., and also improvements that could be made to make a better augmented reality (AR) system.

7.1. Summary

During this project much was learnt about various areas of Computer Science, especially in the areas of physics simulation and augmented reality technology. Lack of experience with augmented reality and an overabundance of enthusiasm meant that the project had to be scaled back somewhat. Much research was carried out and many research papers were read, and the project became more realistic.

Many different APIs and toolkits were used throughout the implementation, and most if not all of them have alternatives. Section 3.3 details why the specific technologies were chosen. All the technologies chosen have two things in common: firstly, they are all *free*; secondly, they can all run on an off the shelf PC. These two facts mean that *anyone* can run the program developed as part of this project, assuming they have a cheap web cam. This is typical of AR programs that use ARToolkit, and is one of the main benefits of fiduciary marker technology.

The evaluation showed that the virtual tasks performed were comparable in believability to similar real tasks. This is a gratifying result; achieving what was originally planned.

7.2. Project Changes

The initial idea for this project was somewhat different to the final result. It was first thought that the main focus of the project would be the reduction or removal of fiduciary markers. Fiduciary markers are used predominantly throughout AR research. This is because they are easier (relatively) than other more advanced image processing techniques. Whilst fiduciary markers are a good first step towards more sophisticated AR systems, it is unlikely that fiduciary markers are the future of AR. Section 2.3.1 briefly mentions optical flow; one more sophisticated technique that is being researched.

As computer processing power increases, and computer vision algorithms improve, fiduciary markers will be entirely obsolete. The area of computer vision was to be the primary focus of this project. However, as more research was done into the technologies used in this project (such as ARToolkit), it became clear that this goal of fiduciary marker elimination was overly ambitious and considerably too large for the timescale of this project. Therefore, the project idea evolved and changed; whilst still being about AR, it focused on believability whilst still using the stable base of fiduciary marker technology.

As the previous section described, the evaluation showed that the final prototype achieved the original goal; people who took the survey and performed the psychometric experiments found the system to be believable.

7.3. Future Work

Whilst it is considered that the project was successful in achieving what it set out to do, there are some ways in which it could be improved. These improvements were not implemented for various reasons such as lack of time, etc. Some of these improvements could form the basis of future work.

The biggest inadequacy of this project is the basic rendering. Although in the final prototype basic shadows were added to increase the realism, the rendering of the virtual objects was not very realistic. Realistic rendering is the other side of the coin from physics in terms of believable AR. Unless the virtual objects are stylised in some way (to look cartoonish) for example, photorealistic rendering will be key to realistic AR. Some interesting future work would be based on the use of stereo vision, i.e. using two cameras and two viewing screens (one for each eye). There are two uses for stereo vision:

1. Using stereo camera information to gather depth information from a scene, and using this information to achieve better registration.

2. Using stereo camera information to create a 3D viewing experience for the user, via the two viewing screens.

There is plenty of future work around the original idea for this project that was mentioned in the section above: elimination of fiduciary markers. Perhaps fiduciary markers could continue to be used for the registration of the table, but other more sophisticated techniques might be used for the registration of the forklift. Also, instead of a forklift being the only real physics object in a scene, a system might be able to "learn" new objects, and create a physics representation based on the size and shape of the object that is being learned.

In this project, real and virtual objects interact, but only really in one direction: the forklift moves, pushes and lifts the virtual objects. An interesting improvement would be to allow the virtual objects to move the forklift. I.e., if a virtual box bounces off the forklift, it might "shiver" or move a little to simulate the effect the box would have on the forklift were it real. However, this improvement relies on the real objects being able to move themselves; in the case of the forklift, motors could be used for this purpose, but a more generic object may not have the means to move itself.

7.4. Final Observations

Picking a suitable and original final year project is not an easy task. In the end, this project idea was mostly adapted from an idea by the project supervisor.

It was anticipated that the project would be a challenge, and it did turn out to be such. It involved some difficult and somewhat labourious mathematics, and it was not clear that a working prototype would ever be achieved. However, it pulled together in the end and overall the project was thoroughly enjoyable. The project impressed many of its users. AR is indeed an impressive technology; although AR technology is still very immature, the technology is constantly improving. It is expected that in the near future AR will become an integral part of most people's lives in one form or another. The hardware is getting very impressive[23], and active research into better software continues.

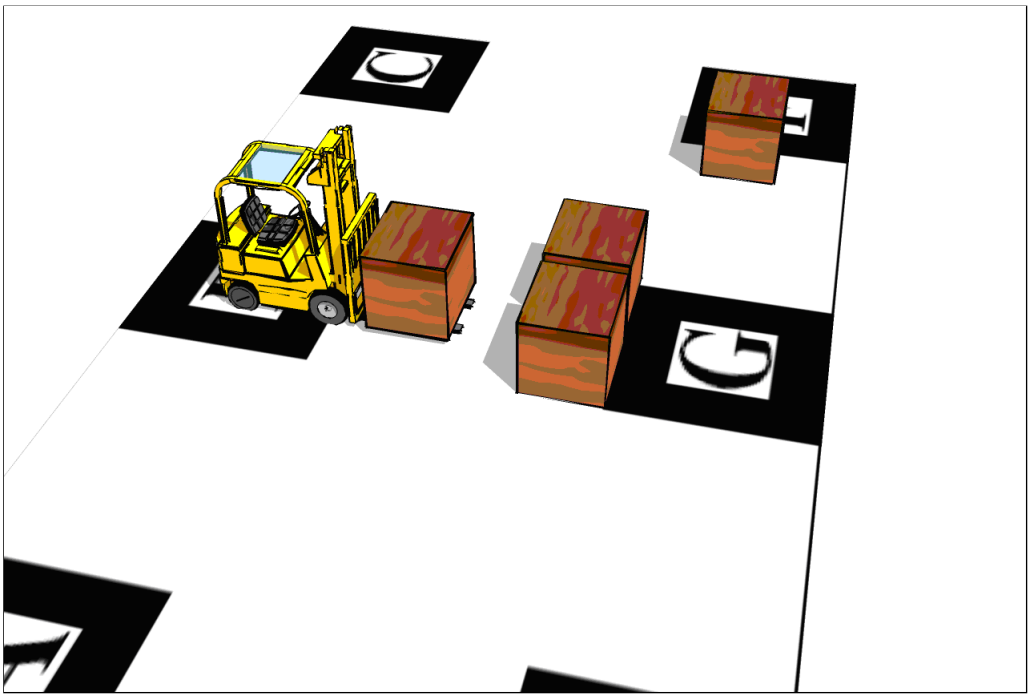
[23] `http://www.vuzix.com/iwear/products_wrap920av.html`

A. Appendix A - Evaluation Script

Realism in Augmented Reality
Evaluation Script

March 2009

Introduction



Welcome and thank you for taking the time to evaluate this project. Today you will be asked to play several games and answer some questions concerning them. The games will involve picking up and moving boxes with a remote controlled forklift.

The games will be of two types:

1. Picking up of virtual boxes using augmented reality technology
2. Picking up of real cardboard boxes that simulate the virtual ones.

Instructions will be provided as you proceed with the questions.

Orientation

You will now have a 5 minute period in which to "play" with the system and become used to the controls. You may drive the forklift around and pick up boxes to become familiar with the controls. You may terminate this orientation period before the 5 minutes is up if you are confident that you can control the system.

The forklift may be driven forward or backward. It may also turn left and right. The lift may be in the up position or the down position.

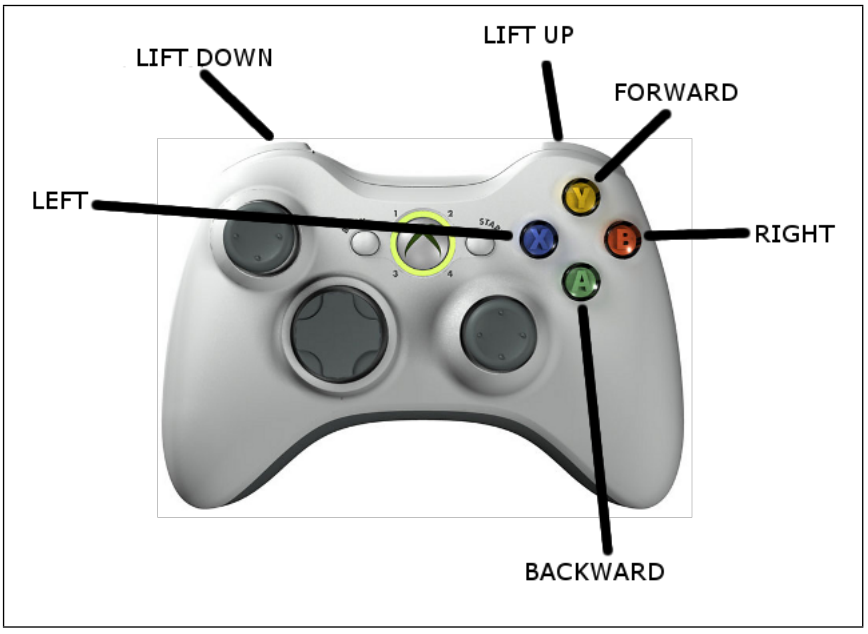
The forklift may most easily be controlled with the XBOX 360 controller. The following diagram shows the buttons for the controller.

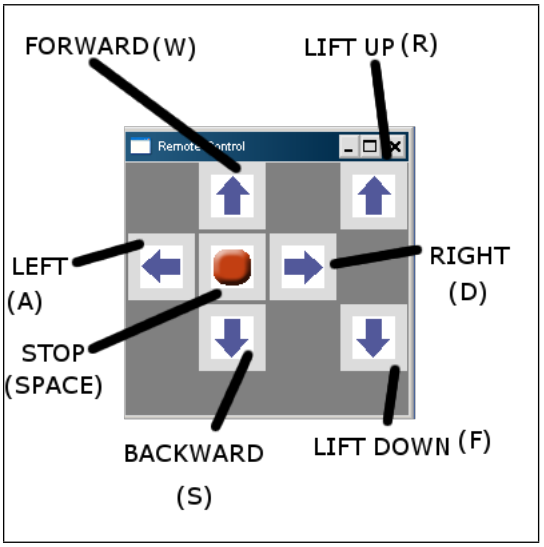
The forklift may also be controlled with the keyboard. The keys for controlling the forklift are indicated in the following diagram.

You may also use the mouse if you find it easier. Please note the remote control window must have focus for the keys to work.

Please take careful note of the fact that the “space” key stops the forklift completely; both the wheel and lift motors stop. If at any time you feel you are losing control of the forklift, just hit the space key.

If you are ready you may now go ahead and put on the headset. After the 5 minutes is up you will be ready to play the first game.





Pick Up One Box

T1

For this task, the forklift will be moved into the starting position and the system will be reset. Your task is to pickup a single cardboard box with the forklift. Please complete the task as quickly as possible, but do not rush. The experimenter will indicate when you have sufficiently completed the task.

Please note that you may restart the tasks at any time you wish. You may take as many attempts as necessary to complete the task.

On a Likert scale from 0-5 (0=worst, 5=best), how would you rate the overall ease of performing this task? (Please circle)

1.

2.

3.

4.

5.

T2

As before, the forklift will be placed in the starting position and the system will be reset. Now you will complete the same task as the previous one, except this time you will be lifting a *virtual* box with the forklift. These locations will be indicated.

On a scale from 0-5 (0=worst, 5=best), how would you rate the overall ease of performing this task? (Please circle)

1.

2.

3.

4.

5.

Forklift Football

T3

For this task, the forklift will be moved into the starting position and the system will be reset. Your task is to move a single cardboard box between two goal posts that are indicated as red and blue cones. Please, do not forget that you may restart this task at any

time. Please complete the task as quickly as possible, but do not rush. The experimenter will indicate when you have sufficiently completed the task.

On a Likert scale from 0-5 (0=worst, 5=best), how would you rate the overall ease of performing this task? (Please circle)

1.

2.

3.

4.

5.

T4

As before, the forklift will be placed in the starting position and the system will be reset. Now you will complete the same task as the previous one, except this time you will be trying to score a goal with a virtual box. Note that the goal posts are also virtual.

On a scale from 0-5 (0=worst, 5=best), how would you rate the overall ease of performing this task? (Please circle)

1.

2.

3.

4.

5.

Final Questions

You have completed all the tasks. Please answer the following short section of questions.

On a scale from 0-5 (0=not at all, 5=very), how would you rate how conscious you were of the technology during the experiments? (Please circle)

1.

2.

3.

4.

5.

On a scale from 0-5 (0=not at all, 5=very), how would you rate the believability of the virtual tasks as compared to the non-virtual tasks? (Please circle)

1.

2.

3.

4.

5.

On a scale from 0-5 (0=not at all, 5=very), how would you rate the adequacy of the field of view of the HMD? (Please circle)

1.

2.

3.

4.

5.

On a scale from 0-5 (0=not at all, 5=very), how would you rate the level of discomfort you experienced as a result of using the system for the duration of the experiments? (Please circle for each part of body)

- fingers
  1.
  2.
  3.
  4.
  5.
- eyes
  1.
  2.
  3.
  4.
  5.
- neck
  1.
  2.
  3.
  4.
  5.
- other (please state area and discomfort level)

Conclusion

Thank you for taking the time to help with the evaluation of this project.

References

[1] Milgram, P. and Kishino, F. (1994). "A Taxonomy of Mixed Reality Visual Displays". *IEICE Transactions on Information Systems*, E77-D(12).

[2] (-). "How does ARToolKit work?" Date Accessed: Nov 2008. `http://www.hitl.washington.edu/artoolkit/documentation/userarwork.htm`.

[3] Xu, K., Prince, S. J. D., Cheok, A. D., Qiu, Y., and Kumar, K. G. (2003). "Visual registration for unprepared augmented reality environments". *Personal Ubiquitous Comput.*, 7(5):pages 287–298. ISSN 1617-4909.

[4] Ramalingam, S. (2008). "Software Life Cycle Planning and Modeling". Date Accessed: Nov 2008. `http://microsoft.apress.com/asptodayarchive/73782/software-life-cycle-planning-and-modeling`.

[5] (-). "Coordinate Systems". Date Accessed: Feb 2008. `http://www.hitl.washington.edu/artoolkit/documentation/cs.htm`.

[6] Whitson, J., Eaket, C., Greenspan, B., Tran, M. Q., and King, N. (2008). "Neo-immersion: awareness and engagement in gameplay". In *Future Play '08: Proceedings of the 2008 Conference on Future Play*, pages 220–223. ACM, New York, NY, USA. ISBN 978-1-60558-218-4.

[7] Mori, M. (1970). "The Uncanny Valley". Date Accessed: March 2009. `http://www.androidscience.com/theuncannyvalley/proceedings2005/uncannyvalley.html`.

[8] Sutherland, I. E. (1965). "The Ultimate Display". In *Paper presented at the Proceedings of the IFIP Congress*, pages 506–508.

[9] Sutherland, I. E. (1998). "A head-mounted three dimensional display". *Seminal Graphics : Pioneering Efforts that Shaped the Field ACM*, pages 295–302.

[10] Azuma, R. (1997). "A Survey of Augmented Reality. Presence: Teleoperators and Virtual Environments". pages 355–385.

[11] Livingston, M. A., Rosenblum, L. J., Julier, S. J., Brown, D., Baillot, Y., Edward, J., Ii, S., Gabbard, J. L., and Hix, D. (2002). "An Augmented Reality System for Military . . ." In *In Interservice/Industry Training, Simulation, and Education Conference*, page 89.

[12] Curtis, D., Mizell, D., Gruenbaum, P., and Janin, A. (1999). "Several devils in the details: making an AR application work in the airplane factory". In *Proceedings of the international Workshop on Augmented Reality : Placing Artificial Objects in Real Scenes (Bellevue, Washington, United States)*.

[13] Thomas, B., Close, B., Donoghue, J., Squires, J., Bondi, P. D., and Piekarski, W. (2002). "First Person Indoor/Outdoor Augmented Reality Application: ARQuake". *Personal Ubiquitous Comput.*, 6(1):pages 75–86. ISSN 1617-4909.

[14] Reitmayr, G. and Schmalstieg, D. (2003). "Location based applications for mobile augmented reality". In *Proceedings of the Fourth Australasian User interface Conference on User interfaces*, volume 36, pages 65–73. Australian Computer Society, Darlinghurst, Australia.

[15] Klein, G. and Murray, D. (2007). "Parallel Tracking and Mapping for Small AR Workspaces". In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*. Nara, Japan.

[16] Azuma, R. (1993). "Tracking requirements for augmented reality". *Commun. ACM*, 36(7):pages 50–51. ISSN 0001-0782.

[17] Lowe, D. G. (2004). "Distinctive Image Features from Scale-Invariant Keypoints". *Int. J. Comput. Vision*, 60(2):pages 91–110. ISSN 0920-5691.

[18] Ong, S. K., Yuan, M. L., and Nee, A. Y. C. (2005). "Tracking points using projective reconstruction for augmented reality". In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 421–424. ACM, New York, NY, USA. ISBN 1-59593-201-1.

[19] Kato, H. and Billinghurst, M. (1999). "Marker Tracking and HMD Calibration for a Video-Based Augmented Reality Conferencing System". In *IWAR '99: Proceedings of the 2nd IEEE and ACM International Workshop on Augmented Reality*, page 85. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-0359-4.

[20] Abawi, D. F., Bienwald, J., and Dorner, R. (2004). "Accuracy in Optical Tracking with Fiducial Markers: An Accuracy Function for ARToolKit". In *ISMAR '04: Proceedings of the 3rd IEEE/ACM International Symposium on Mixed and Augmented Reality*, pages 260–261. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2191-6.

[21] You, S., Neumann, U., and Azuma, R. (1999). "Hybrid Inertial and Vision Tracking for Augmented Reality Registration". In *VR '99: Proceedings of the IEEE Virtual Reality*, page 260. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-0093-5.

[22] Satoh, K., Anabuki, M., Yamamoto, H., and Tamura, H. (2001). "A hybrid registration method for outdoor augmented reality". *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, pages 67–76.

[23] Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. (1993). "Toolglass and magic lenses: the see-through interface". In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 73–80. ACM, New York, NY, USA. ISBN 0-89791-601-8.

[24] Grosjean, J. and Coquillart, S. (1999). "The magic mirror: A metaphor for assisting the exploration of virtual worlds".

[25] Bimber, O. and Raskar, R. (2005). *Spatial Augmented Reality: Merging Real and Virtual Worlds*. A K Peters, Ltd. ISBN 1568812302.

[26] Kato, H., Billinghurst, M., Poupyrev, I., Imamoto, K., and Tachibana, K. (2000). "Virtual object manipulation on a table-top AR environment". pages 111–119.

[27] Kojima, Y., Yasumuro, Y., Sasaki, H., Kanaya, I., Oshiro, O., Kuroda, T., Manabe, S., and Chihara, K. (2001). "Hand manipulation of virtual objects in wearable augmented reality". pages 463–469.

[28] Cabrera, A., do Nascimento, O., Farina, D., and Dremstrup, K. (2008). "Brain-Computer Interfacing: How to control computers with thoughts". pages 1–4.

[29] Billinghurst, M., Kato, H., and I., P. (2001). "The magicbook: A transitional ar interface". *Elsevier Computers and Graphics*.

[30] Agusanto, K., Li, L., Chuangui, Z., and Sing, N. W. (2003). "Photorealistic rendering for augmented reality using environment illumination". In *ISMAR '03: Proceedings of the 2nd IEEE/ACM International Symposium on Mixed and Augmented Reality*, page 208. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-2006-5.

[31] Azuma, R., Baillot, Y., Behringer, R., Feiner, S., Julier, S., and MacIntyre, B. (2001). "Recent advances in augmented reality". *Computer Graphics and Applications, IEEE*, 21(6):pages 34–47.

[32] Dong, Q., Sun, Z., and Mac Namee, B. (2008). "Physics-Based Table-Top Mixed Reality Games". In *Proceedings of the 39th Conference of the International Simulation and Gaming Association*.

[33] Oda, O., Lister, L. J., White, S., , and Feiner, S. (2007). "Developing an augmented reality racing game". In *In Proceedings of the 2nd international Conference on intelligent Technologies For interactive Entertainment (Cancun, Mexico, January 08 - 10, 2008)*, pages 1–8. ICST, Brussels, Belguim.

[34] Leitner, J., Haller, M., Yun, K., Woo, W., Sugimoto, M., and Inami, M. (2008). "IncreTable, a mixed reality tabletop game experience". In *ACE '08: Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology*, pages 9–16. ACM, New York, NY, USA. ISBN 978-1-60558-393-8.

[35] Kaufmann, H. and Meyer, B. (2008). "Simulating educational physical experiments in augmented reality". In *SIGGRAPH Asia '08: ACM SIGGRAPH ASIA 2008 educators programme*, pages 1–8. ACM, New York, NY, USA. ISBN 978-1-60558-388-4.

[36] Weisstein, E. W. (-). "Runge-Kutta Method". Date Accessed: April 2009. `http://mathworld.wolfram.com/Runge-KuttaMethod.html`.

[37] Pryor, H., Furness, T., and Viirre, E. (1998). "Developing an augmented reality racing game". In *Proc. 42nd Human Factors Ergonomics Soc*, pages 1570–1574. Human Factors Ergonomics Soc, Santa Monica, Calif.

[38] Moore, G. E. (2006). "Cramming more components onto integrated circuits, Reprinted from Electronics, volume 38, number 8, April 19, 1965, pp.114 ff." *Solid-State Circuits Newsletter, IEEE*, 20(3):pages 33–35.

[39] Davis, A. (1992). "Operational prototyping: a new development approach". *Software, IEEE*, 9(5):pages 70–78.

[40] Masse, R. E. (2008). "Evolutionary Prototyping: "Add Later" Static Types for Python". Date Accessed: Nov 2008. http://www.python.org/workshops/1998-11/proceedings/papers/masse/masse.html.

[41] Zhang, X., Fronz, S., and Navab, N. (2002). "Visual Marker Detection and Decoding in AR Systems: A Comparative Study". In *ISMAR '02: Proceedings of the 1st International Symposium on Mixed and Augmented Reality*, page 97. IEEE Computer Society, Washington, DC, USA. ISBN 0-7695-1781-1.

[42] Ahn, S. H. (-). "OpenGL Projection Matrix". Date Accessed: April 2009. http://www.songho.ca/opengl/gl_projectionmatrix.html.

[43] Ralph, W. J. (2008). "Coordinate System (mathematics)". Date Accessed: April 2009. http://encarta.msn.com/encyclopedia_761579532/Coordinate_System_(mathematics).html.

[44] (-). "Introduction to 3D Animation". Date Accessed: April 2009. http://www.animationarena.com/introduction-to-3d-animation.html.

[45] Klinker, G. (2001). "Praktikum augmented reality - occlusion handling between real and virtual objects". Date Accessed: April 2009. http://wwwbruegge.in.tum.de/teaching/ss01/AR-Praktikum01/presentation/occlusions.pdf.

[46] (-). "The Depth Buffer". Date Accessed: Dec 2008. http://www.opengl.org/resources/faq/technical/depthbuffer.htm.

[47] Beardmore, R. (2008). "Worm Gears". Date Accessed: April 2009. http://www.roymech.co.uk/Useful_Tables/Drive/Worm_Gears.html.

[48] Berger, D. (2007). "Remote-controlling Lego Mindstorms NXT by using Visual C++". Date Accessed: Jan 2009. http://www.kyb.tuebingen.mpg.de/bu/people/berger/nxt/nxt.html.

[49] Badler, N. I., Bindiganavale, R., Bourne, J., Allbeck, J., Shi, J., and Palmer, M. (1999). "Real Time Virtual Humans". Date Accessed: March 2009. http://www.cis.upenn.edu/~badler/bcs/Paper.htm.

[50] Pressman, R. S. (2001). *Software Engineering: A Practitioner's Approach*. McGraw-Hill Higher Education. ISBN 0072496681.

[51] Likert, R. (1932). "A technique for the measurement of attitudes." *Archives of Psychology*, 22(140):pages 1–55.

[52] (2009). "Students t-test". Date Accessed: April 2009. http://www.britannica.com/EBchecked/topic/569907/Students-t-test.