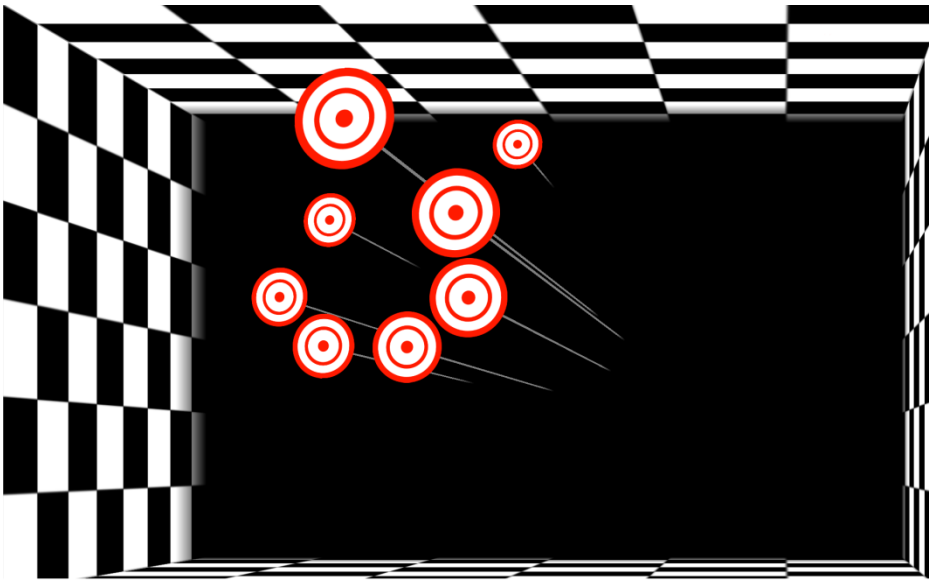
Virtual Reality

A study on perception

Authors

Martin Holst Erichsen <*fmw119@diku.dk*>

Peter Hillebrandt Poulsen <*hcp263@diku.dk*>



Supervisor

Jon Sporring

01-06-2012

Abstract

Ever since Johnny Lee published his work “Head Tracking for Desktop Virtual Reality Displays using the Wii Remote”, many similar desktop virtual reality projects have been made. In this project we create a similar 3D effect using the Microsoft Kinect and head tracking. We compare the resulting experience with other popular 3D technologies. We investigate whether it is possible to improve user experience and interaction with the virtual environment by conducting experiments targeted at different factors of perception, such as size and depth. Furthermore we investigate the player experience linked to the 3D experience when using head tracking. We find that this contribution to player experience is positive and portable to other applications. We also find that the users have an improved perception of depth when using head tracking.

We strongly recommend watching this video which is a demo of our test environment: http://www.youtube.com/watch?v=954jb2XZxD4

Preface

This project is the result of a 14 weeks bachelor thesis in computer science. It has been written by the bachelor students Martin Holst Erichsen and Peter Hillebrandt Poulsen at the University of Copenhagen under the supervision of associate professor Jon Sporring.

We expect the reader to have an understanding of fundamental computer science concepts, computer graphics and a very basic understanding of statistic modeling.

Documentation as well as source code to run the developed application can be downloaded from:
http://www.sendspace.com/file/hayfwm

Acknowledgements

We would like to thank our supervisor Jon Sporring for his counseling and feedback throughout the project. The weekly meetings provided a solid foundation for this piece of work and have kept the project on track. We would also like to thank Carnegie Mellon University for supplying the free game engine Panda3D which was used in development of this project. We also extend our thanks to Robert Kooima for making his paper "*Generalized Perspective Projection*" publicly available.

Finally we would like to thank Lykke Jensen and Kevin Le for proofreading parts of this report.

Contents

1 Introduction

In this bachelor project we will introduce a 3D experience to a virtual environment by using head tracking. We investigate the user's experience and compare user performance with and without head tracking.

The main goal of this thesis is expressed in the following question

> *Is it possible to improve the user's perception of and interaction with a virtual environment by creating and introducing a 3D experience based upon motion detection using the Microsoft Kinect and a prebuilt 3D engine?*

By using the Microsoft Kinect to track the user's movement, in particular head movement, we attempt to create a 3D effect by updating the scene on the screen based on the position of the user. By using a specialized perspective projection matrix, the screen will appear to be a window into the virtual environment. This will introduce a sense of 3D to the environment without the use of any inconvenient 3D equipment. We will then investigate the user's perception of the virtual environment by creating different scenarios for the user to complete with and without head tracking. We use a prebuilt 3D engine to create the virtual environment needed to perform the tests. To get the necessary readings of body movement, we will need to use a Kinect SDK. We will compare the achieved 3D experience to some of the current technologies on the market.

1.1 Motivation

3D technology as well as interaction using motion detection has in the recent years experienced tremendous progress. 3D technology is defined as the sort of 3D requiring a 3D compliant screen and glasses which creates an illusion of depth on a 2D screen. Even though the two technologies share a common purpose, which is to immerse the user into a 3D environment, there is a lack of coupling between them. Furthermore the availability of the equipment required to use motion detection has increased and the price has decreased. The current technology needed to experience 3D, such as a 3D screen and compatible glasses, makes it an inconvenience to the casual user and it provides a severe limitation to people already using glasses. By implementing head tracking we hope to eliminate the need of inconvenient 3D equipment.

1.2 Limitations

This project is primarily a research project and as such we won't implement a user interface. We won't focus on the quality of the graphical content. No user guide will be created as the users will be instructed in the use of the application hands on. As we will be using a SDK to read the output from the Kinect we are limited to the library's

capabilities. We will only be working with the Microsoft SDK for the Kinect as it provides sufficient performance and accuracy

1.3 Project Planning

In the early phases of this project we created a project plan. It reflected the tasks we needed to go through in order to complete the project and the time needed for each of these. The hours needed to complete the tasks were rough estimates. We didn't include the time needed to write the report, but assumed that it would double the time of each task. We have used an approach which bears a resemblance to a product breakdown structure and the result can be seen in Figure 1. It shows the 10 tasks in a network diagram with critical path activities highlighted (Cadle, et al.). The contents of each task and the contribution from it should be clear after reading the report.

Resources and finished products are inherited through dependencies.

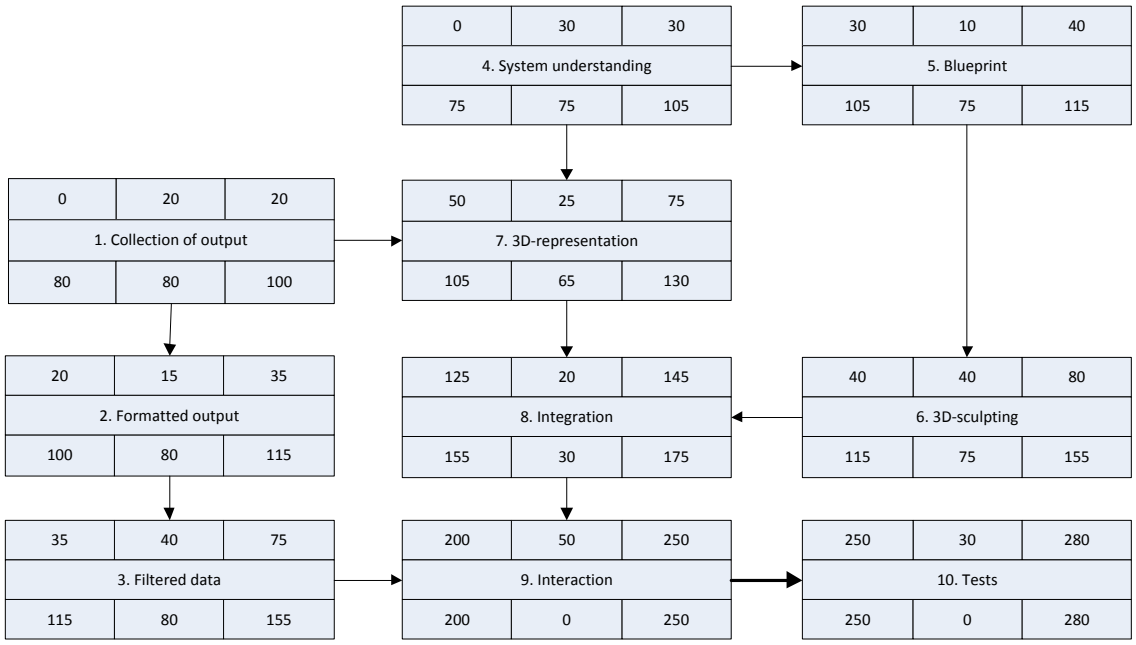


Figure 1: Networked diagram with the bold arrow highlighting the critical path

During the project decided not to use a 3D-representation of a human to navigate the environment. This was done because we wanted the screen to appear like a window into another reality. Furthermore we found that filtering of data was built into the Microsoft SDK.

The tasks remain rough estimations of the phases we have gone through in this project and the product from each is described in the following list.

1. Collection of output
Product: A library wrapper simplifying data retrieval from the Kinect
Resources: Kinect, Microsoft Kinect SDK, development environment compatible with the SDK
Dependencies: Not available
Expected workload: 20 hours
Actual workload: 35 hours

2. Formatted output
Product: Library wrapper capable of communicating with Python
Resources: Microsoft SDK documentation
Dependencies: #1
Expected workload: 15 hours
Actual workload: 10 hours

3. Filtered data
Product: Filtered and noiseless data from a Kinect
Resources: Not available
Dependencies: #2
Expected workload: 40 hours
Actual workload: 1 hour

4. System understanding
Product: Thorough understanding of 3D engine and physics being used
Resources: 3D engine, documentation, literature concerning 3D engine
Dependencies: None
Expected workload: 30 hours
Actual workload: 40 hours

5. Blueprint
Product: A blueprint of virtual environment with test scenarios
Resources: Pen and paper
Dependencies: #4
Expected workload: 10 hours
Actual workload: 10 hours

6. 3D sculpting
Product: Model of the virtual environment, scenarios and physics
Resources: None
Dependencies: #5
Expected workload: 40 hours
Actual workload: 120 hours

7. 3D-representation
Product: 3D-representation of human using Kinect data
Resources: None
Dependencies: #3 and #4
Expected workload: 25 hours
Actual workload: None

8. Integration
Product: 3D-representation integrated into the virtual environment
Resources: None
Dependencies: #6 and #7
Expected workload: 20 hours
Actual workload: None

9. Interaction
Product: Virtual environment taking user's position into account
Resources: None
Dependencies: #3 and #8
Expected workload: 50 hours
Actual workload: 50 hours

10. Tests
Product: Analysis of results from tests of users using the Virtual environment to complete tasks
Resources: 15 test subject
Dependencies: #8 and #9
Expected workload: 30 hours
Actual workload: 80 hours

2 Theory

This chapter will describe the fields of study we will use to draw our conclusions and forms a basis for our discussions. The theory will describe and explain sub components of the system.

2.1 The Microsoft Kinect

The Kinect was originally intended as Microsoft's answer to the highly popular Nintendo Wii. The Kinect is the first controller to be fully independent of any carried equipment. The incentive behind the Kinect was to give high accuracy at low cost. This has made the Kinect popular even in fields outside its original intent.

The head of the Kinect is motorized and it supports a range of 27 degrees. It can be moved from within applications. The Kinect has a RGB camera supporting a resolution of up to 1280x960 pixels and an infrared camera supporting a resolution of up to 640x480. Both cameras support a frame rate of 30Hz. Furthermore it has a built in infrared projector which projects a pattern of infrared light onto a surface in front of the Kinect. This infrared light is captured by the infrared camera and used to calculate a depth image. The Kinect is able to create a depth image of the environment in front of it. It works in the range of 0.8 – 4 meters, although the recommended distance to the player should be in the range of 0.9 – 3.6 meters. Just like other cameras, the Kinect's cameras are limited to a certain field of view. This is depicted in Figure 2. (Webb, et al.)

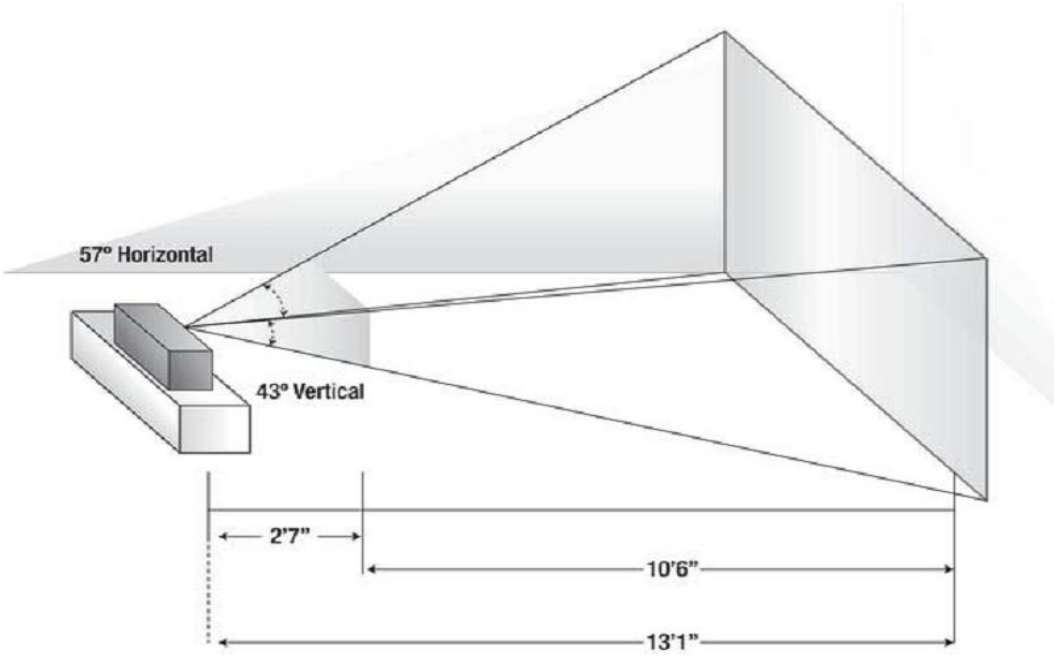


Figure 2: Field of view for the Kinect's cameras

2.2 3D Technologies

Most 3D technologies are based upon two cameras viewing the same scene from slightly different angles. How this creates the actual 3D effect depends on the technology used. This chapter describes some of the currently used technologies and will be divided into the technologies that use glasses and those that don't.

2.2.1 Glass technologies

The glass technologies can be divided into active and passive technologies. Passive technologies depend upon the images shown to the viewer, having been already manipulated in a way that allows the glasses to automatically filter the images correctly. Active technologies require electronics in the glasses to actively filter the images received from the screen.

2.2.1.1 Polarized lenses

The currently most popular technology uses polarized pictures and lenses. The 3D effect is created by first using circular or linear polarization on the images from each camera and then superimposing these images. The user wears low cost glasses with lenses that are polarized. This allows for each lens to filter out the images not meant for the specific eye, which in turn generates the 3D effect. Polarization works by filtering out all light that is differently polarized, leaving only the light that has the correct polarization. This is illustrated on Figure 3.

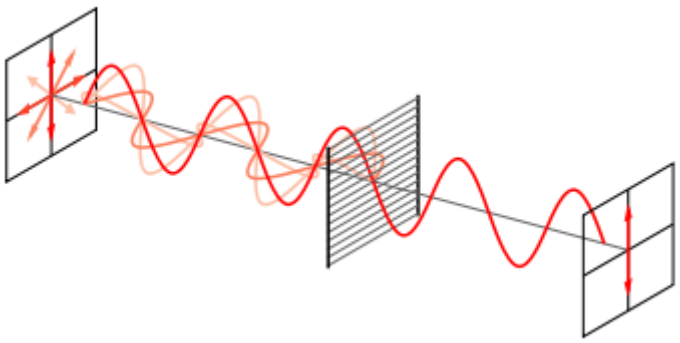


Figure 3: Explanation of polarization

By using circular polarization the viewer does not have to look directly into the screen to get the correct 3D experience but can in fact tilt their head in any direction without loss of quality. This technology has a number of advantages as well as some disadvantages. The advantages are the following

- Any number of users can use the same screen and get the same 3D effect
- The glasses are cheap and do not require any power
- People with one dominant eye can still get the full 3D effect

- The glasses are not colored which means no color is lost in the images
- It is not necessary to synchronize the glasses with the display

The disadvantages are

- An expensive screen which is capable of showing enough frames per second and polarizing the images is needed
- The user needs to wear glasses which can be uncomfortable. This can also reduce the image quality as the user is looking through an extra surface
- Some people complain about headaches when using this technique

2.2.1.2 Eclipse method

The eclipse method is an example of an active technology. It works by having the screen synchronize with the glasses. The screen then rapidly shows pictures alternating from both cameras. The lenses automatically shut out all light when an image is shown which is not supposed to be seen by that eye. This way, only one of the lenses is open at any given time and images meant for the right eye is only shown to the right eye and vice versa. This general technique is being used nVidia, XpanD 3D, earlier IMAX systems and others.

Some of the major drawbacks of this technique include the price of the equipment, glasses are required and the fact that the glasses are heavier than when using the passive technologies. Imperfections in the shuttering of the glasses can create a phenomenon known as ghosting or crosstalk. Ghosting means that some images are shown to the wrong or both eyes (Woods, et al., 2002).

2.2.2 Glassless technologies

Technologies exist that attempt to create a 3D effect without the use of glasses. This section will go through two of these technologies.

2.2.2.1 Parallax barrier

The parallax barrier screen emits rows of partial images meant for the left and right eye, one after the other. By placing rows of light-blocking barriers in front of the screen, zones are created in which only the eye that is meant to see each image actually can. This effect is illustrated on Figure 4.

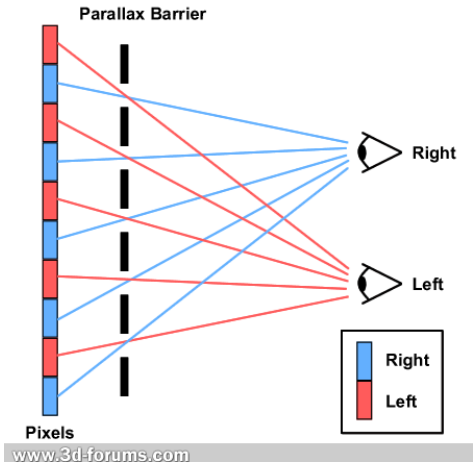


Figure 4: Pixels observed with two eyes through a parallax barrier

This technique comes with some major disadvantages. In particular there are multiple zones in which the image might be blurry or the 3D effect can be inverted. This does not allow for any viewer movement as this could potentially bring them into one of these zones.

2.2.2.2 Lenticular lens

The lenticular lens attempts to create a 3D effect by interlacing video from two cameras. Lenses break the light in such a way that a viewer who is positioned correctly will get a 3D impression. The general principle is shown on Figure 5.

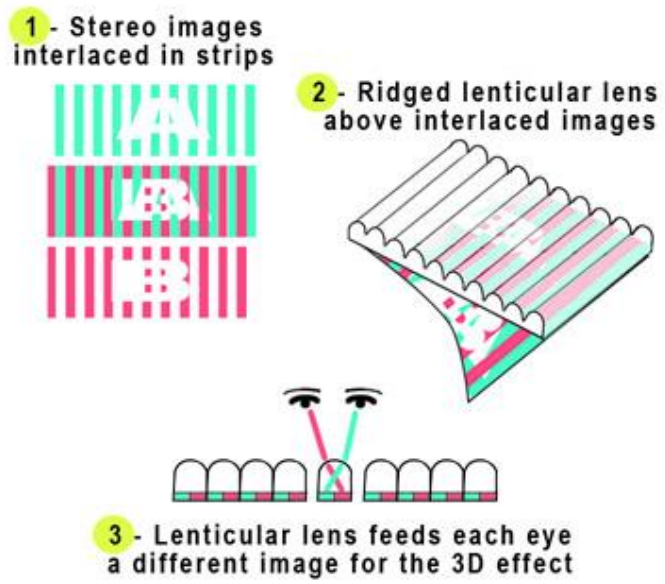


Figure 5: Illustration of a lenticular lens in effect

The lenticular lens is quite uncommon due to it being very expensive to produce. Currently only one lenticular lens 3D television model exists and the price of it is 20.000$. The

lenticular lens technology also exhibits zones in which no 3D is experienced and only allows a small viewing angle. Unlike the parallax barrier method the lenticular lens method lets more backlight shine through, which again allows for better contrast and brightness. A trade-off for this technology is that it halves the resolution but does not require more than usual frames per second to be shown in order to send pictures from different camera angles.

2.3 Choice of 3D-Engine

In the early phases of development it was realized that collision detection would be time consuming to implement and the process of implementing it wouldn't contribute to the findings. We therefore made the choice to use a game-engine with a built in physics motor and the basic capability of rendering 3D.

Before development could begin we had to decide which game-engine we wished to use. As the market for game-engines is broad we decided to single out a few of them and limit our choices to these. The selected game-engines were Unity, Ogre3D, Panda3D and Crytek. Instead of spending time at getting a good look under the hood of each of these game engines, we realized that it would save us some time if we created a list of criteria and based our choice upon these. The list and reasons for the separate points were as follows

1. Good and fast prototyping language
    - Due to the tight timeframe for this entire project, quick prototyping was a priority. This allowed us to iterate quickly through the programming phases
2. We should already know the language
    - We could not afford to use time needed to learn a new language
3. It should be able to work together with C# or C++
    - Since the Microsoft SDK is written in C# or C++, being able to communicate with them was essential
4. Good and thorough documentation for the engine
    - In learning to use the engine, documentation would prove invaluable and without it time would be wasted
5. The game engine should not provide any obstacles when dealing with team development
    - We both wanted to be able to work on different parts of the application at the same time and have a proper versioning tool

Crytek was eliminated based on a lack of documentation and Unity was eliminated since team development is restricted in the free version. This narrowed our choices down to Ogre3D and Panda3D. Ogre3D has the advantage that it is open source and written in C++ which means it would be easier to interface with the Kinect SDK. Panda3D is also open source and is written in Python. As Python is a better language for prototyping and pushing quick iterations of code we decided to use Panda3D. To interface with the Kinect SDK we decided to create a NUI library wrapper written in C++ but compiled to python.

Like most of the other current free game-engines, Panda3D contains features such as collision detection and other physics simulations. The engine itself is written in C++ but it interfaces with Python which is the intended language of use. This makes the game engine as fast as the other game engines that are written in C++ but with a bit more overhead.

Panda3D is known as a scene graph engine which means its scene is initially empty and then you add elements to it. Each element serves as a root to which you can apply further elements. At the top of the tree the node "Render" is found. This system is smart because any movement of a node will cause all below nodes to be moved equally which means you can build structures by use of substructures.

The Panda3D scene graph is built on OpenGL and DirectX and exposes some of their functionalities more or less directly. For example to add fog you simply add some fog parameters on a node. These parameters match exactly the parameters in OpenGL and as such Panda3D simply becomes a wrapper to the lower level API.

2.4 Kinect Integration

This section will briefly review the architecture used to utilize the Kinect's skeleton tracking. A brief overview of the three different abstraction layers being used is shown, followed by a deeper explanation of all the layers.

2.4.1 Architecture

To utilize the Kinect in Panda3D it was necessary to retrieve the data from the Kinect using an API. This implied use of a multilayered architecture. In this architectural model each layer provides a well-defined interface to the layer immediately below it. Each layer has its own functionalities and responsibilities to take care of.

The Microsoft Kinect SDK is written in C++ and C# from which we chose C++. Either way it was incompatible with Panda3D since that is written in Python. To solve this problem, a C++ library wrapper was written using the Microsoft Kinect API and afterwards compiled to make it compatible with Python.

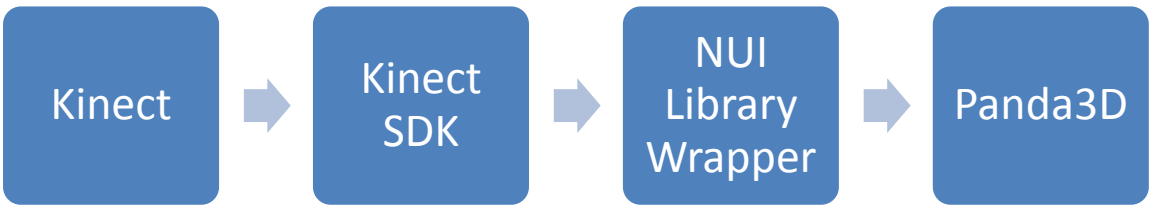


Figure 6: Flow diagram illustrating data originating from the Kinect and then proceeding through the three different layers

The different layers provide a clear distinction in regards to functionality and aids in the understanding of the dataflow from the Kinect to Panda3D. This is illustrated in the flow diagram seen in Figure 6.

The advantages of using this architecture include amongst others, ease of testability, increased abstraction, and strong cohesion for the separate layers. Using a layered architecture has the advantage that if a rewrite of a specific part is necessary, only that specific layer is rewritten rather than rewriting the whole application. In total this provides a basis for fast and effective development, which is a priority.

2.4.2 Kinect SDK

A Kinect must go through certain steps to work properly. These are detection of hardware, initialization, producing data and finally uninitialization. The Kinect SDK provides an interface for all these steps.

The detection of hardware ensures that a Kinect is connected to the computer executing the program. Initialization starts up the cameras in the Kinect and opens the different data streams being used. After this, the data from the Kinect simply flows through these streams providing data such as skeletal joint positions. When the program terminates the Kinect is uninitialized and released so another application can make use of it.

The Kinect sensor does not have a public constructor and the object *KinectSensor* is therefore created by iterating over a collection of Kinects connected to the computer. A *KinectSensor* has a status which indicates the state of the Kinect. It is possible to detect whether the Kinect is ready to be used by checking the state of the *KinectSensor* object.

When the Kinect is ready to be used three different streams can be opened providing different kinds of data. The three different streams are the depth stream, color image stream and skeleton stream. Each of these streams has an event which is fired when new data is provided by the Kinect. The streams are initialized by starting the different cameras and the Kinect will begin feeding data into the streams when the function *start()* is run from the *KinectSensor* object.

The simplest of the three data streams is the color image stream. This stream uses the RGB camera and it provides the same pictures as a webcam or any other camera would. The data is offered in the form of raw pixel data, which means that to process the data it simply needs to be mapped to a bitmap image.

It is necessary to go through three steps regardless of the stream being used. Enabling the stream is necessary to tell the Kinect which streams are being used. After this an image frame is extracted from the Kinect and as the last step the application must process the data. The two last steps are executed repeatedly until the application terminates. The steps are illustrated in Figure 7.

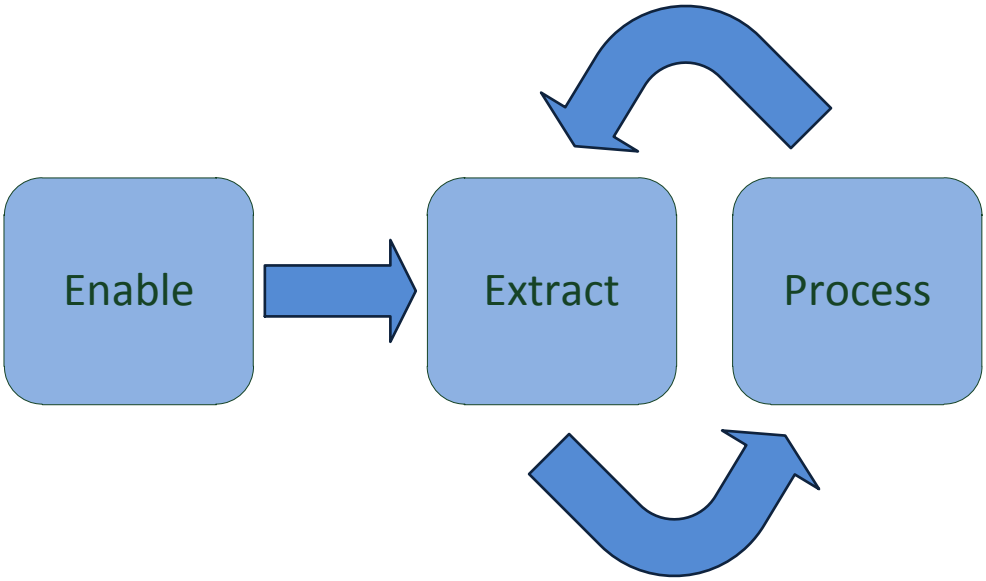


Figure 7: Steps performed on any stream

The Kinect uses a lot of computing power when all the streams are enabled. By only enabling the necessary streams overall performance can be increased. As previously mentioned the Kinect generally offers frames at an approximate rate of 30 frames per second.

The primary function of the Kinect is its ability to detect depth. This differentiates it from a normal camera which does not provide a reliable way of detecting the depth of objects in a picture. Instead of only working with two dimensions in an application, this functionality adds another dimension. The functionality can be utilized by opening up the depth stream. Working with this stream is reminiscent to working with the color image stream, although with a few deviations. The stream must go through the same three steps previously described and illustrated on Figure 7. The difference between this stream and the image stream lies in the data provided. Whereas each pixel was represented by 32 bits in the image stream, the pixels in the depth stream is represented by using only 16 bits. The contents of the bits for each stream can be seen on Figure 8.

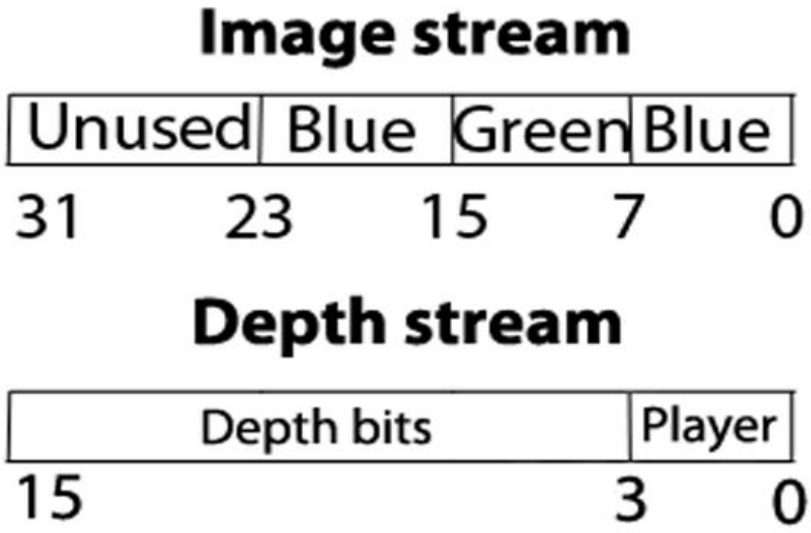


Figure 8: Bits used in streams

The bits 0-2 in the depth stream describe the player index. These bits identify the player the depth data belongs to. Skeleton tracking must be enabled before using this feature. The last 12 bits are used to define the depth of the image. Using these values it is possible to create a depth image as the one seen on Figure 9. It can be seen that the lighter the pixel is the closer the object is to the camera. Furthermore some of the areas around the player and various other pixels are purple. This is the values the Kinect has deemed undeterminable due to noise or the object being too far away.

Figure 9: Depth image produced by the Microsoft Kinect SDK

The depth stream opens up for a whole realm of possibilities, one of them being skeleton tracking. Skeleton tracking can only be enabled when the depth stream is enabled and the resulting data is provided through the skeleton stream.

The skeleton stream uses the data received from the depth stream to allow tracking of up to six users in front of the Kinect. Utilizing the skeleton stream requires a lot of performance since it calculates the skeletons from the data provided by depth stream at runtime. The skeletons are represented with a *NUI_SKELETON_DATA* structure in C++. This data structure contains various data such an array of skeleton joint positions, tracking state indicating whether the current skeleton joints are tracked or if they are predicted, and an overall position of the skeleton.

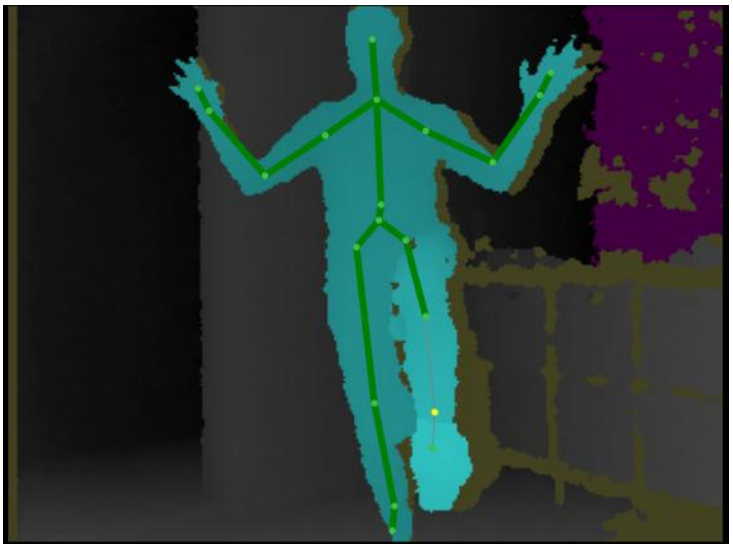


Figure 10: Skeleton joints overlay on the depth image

There are 20 skeleton joints and their positions are calculated using an algorithm made by Microsoft Research Group. The algorithm is run for each depth frame to calculate and provide the skeleton joint positions. The joints are represented by a Vector4 structure containing X-, Y- and Z-coordinates and a variable which is set to unity at all times. An image of the mapped skeleton joints can be seen on Figure 10. The lines which aren't marked green are the joints which are predicted.

To use the skeleton stream it is necessary to enable the stream. As the only stream it is not triggered when a frame from the camera is ready, but instead when the depth stream sends an event trigger. When the event is triggered, a skeleton frame is ready from which the skeleton joint positions can be extracted.

The joint data received from the stream suffers from the fact that this is a new technology and an early version of the SDK. This results in lag when a user moves and occasionally lost tracking of joints. This is especially noticeable when the user moves quickly or when the side of the user is turned towards the camera. The joints can jitter from time to time, even if the user isn't moving. The jittering can however be reduced by using a smoothing function provided by the SDK. The function is implemented using a method used for statistical analysis of economic data called Hold Double Exponential Smoothing. It is

possible to specify parameters such as maximum jitter radius and the number of predicted frames. Increasing the values of the different parameters results in decreased performance and tweaking them might increase responsiveness and performance of the application. This has however not been done for the application developed for this report, but could be a possible future improvement. (Webb, et al.)

2.4.3 NUI library wrapper

To allow a connection between the Kinect SDK and Python a library wrapper was written allowing Python to call the functions from the Kinect SDK. Architecturally the library wrapper fits in between the application layer and the SDK layer. The library wrapper qualifies as a separate layer as it can operate with any Python application.

The library wrapper is written using two parts – a C++ part which implements the actual functionality and the setup script making the library wrapper compatible with Python. This section will describe how each of the two parts is implemented and briefly go through the functionality offered by the library wrapper.

The C++ part of the library wrapper is responsible for providing functions to initialize the Kinect and retrieve data from it. The Microsoft Kinect SDK requires the user to jump through a lot of hoops to retrieve data from the Kinect. One of the aims the library wrapper is to reduce the complexity of data retrieval. When writing the library wrapper this was kept in mind so that the task of retrieving the positions of a tracked skeleton's head would be as simple as calling an initialize function followed by a call to a retrieval function.

The initialize function is responsible for detecting a Kinect connected to the computer and start it up. After that the depth stream is opened and the events, described in the previous section, handling skeleton and depth frames are created. It then starts the process thread which handles the events and specifies what happens when the events are triggered. Allocating all this functionality to the initialize function fulfills the requirement that only one function should be called before retrieval of data is possible. When the events are triggered, a function is called depending on the event being triggered. When the depth event is triggered the function simply releases the frame received allowing the skeleton event to be triggered. The function handling skeleton events finds the first skeleton being tracked and saves the player index in a variable. It then proceeds to smooth the skeleton data of the found skeleton using the built in smoothing function described in the previous chapter. The smoothing is done with the default parameters. After smoothing, the positions for the joints are retrieved and saved into variables. The retriever functions in the library wrapper simply return the data contained in these variables as Python tuples. The variables are automatically updated from the process thread so the latest data will always be returned when calling the retrieval functions.

A method table is also specified defining which functions are callable from Python and what the functions in Python should be named. This method table is used by the setup script, the second part of the library wrapper.

The setup script is necessary as the library wrapper is compiled as an extension module to Python using the *Distutils*. When writing an extension module in Python it is necessary to use a setup script to compile the module. This script makes the module code useable for other Python applications and specifies libraries to be included, source files, and package names among others. A setup script for extensions should contain a call to the functions *Setup* and *Extension*. The *Setup* function is the entry point of the setup script and can be used to specify metadata, packages and other details. It includes a list of extension modules which is made using a call to the function *Extension*. Each entry in the extension list specifies a module and information regarding this module. The library wrapper only consists of one module, so the list simply contains one entry called "*nui*". The entry contains a reference to the C++ part, all the include directories, library directories and libraries used to compile and run the C++ code.

Running the setup script compiles the library wrapper into a .pyd file which can be used in Python applications. Currently five functions are provided by the library wrapper, and they can be seen in the method table taken from the C++ part in Figure 11.

```
static PyMethodDef nui_methods[] = {
    {"init", Nui_Init, METH_VARARGS, "init() doc string"},
    {"gethead", GetHead, METH_VARARGS, "gethead() doc string"},
    {"getlhand", GetLHand, METH_VARARGS, "getlhand() doc string"},
    {"getrhand", GetRHand, METH_VARARGS, "getrhand() doc string"},
    {"uninit", Nui_UnInit, METH_VARARGS, "uninit() doc string"},
    {NULL, NULL}
};
```

Figure 11: Method table indicating functions callable from Python

There are three getter functions, a function to initialize the Kinect and one to uninitialize it. The method table satisfies the need for the library wrapper to provide a simple interface for Python applications to interact with.

When new joint data is received from the Kinect, it is stored in a variable until new data is provided. In the meantime the getter functions for the joint data might be called any number of times between the updating of new data. This might result in the data not being updated between the calls if the Python application queries for new data faster than the Kinect can update it. This performance loss could be avoided by the use of events, but it wasn't found necessary in the development of the application.

2.4.4 Panda3D layer

Panda3D was the last layer in the architecture and several things should be done in this layer. The first thing we had to do was create and populate the 3D scene. This is described in the section "*2.6 Creating the Virtual Environment*". To get realistic behavior of moving entities, we added physics to the created environment. The scene also had to be updated in correspondence with the data, received from the layer above. A part of this was altering the used perspective projection matrix to get the wanted 3D effect. Refer to the chapter "*2.5.3 Generalized off-axis perspective projection*" for a description of the details behind this customization. Different scenarios for performing various experiments were then implemented. We then implemented input options to record data from the users. Positioning of the Kinect and specifications of the screen was made customizable. This was done to make the application easily deployable in multiple locations. A simplified class diagram for the finished code is illustrated on Figure 12. Notice that scenario is denoted as a metaclass as it describes the behavior of all the scenarios.

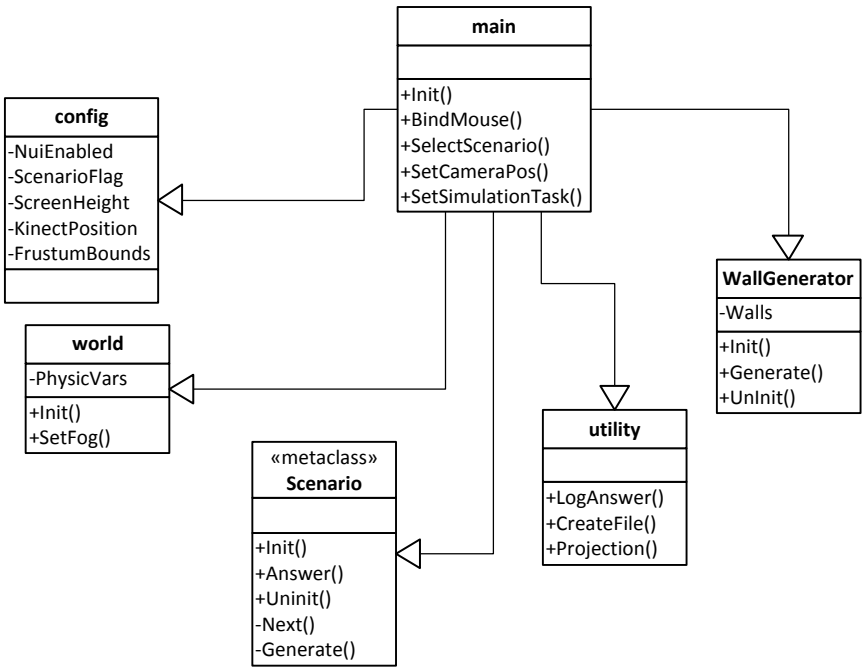


Figure 12: Simplified class diagram for the Python code written with Panda3D

The file *main.py* is the starting point of the application. It sets up the world using various classes and functions and furthermore sets up handling of runtime input and output. The entry point of the application is shown in the piece of code on Figure 13. Panda3D has tasks which are run for each frame through a task manager's task list. Among other things the task manager handles user input and physics.

```
def main():

    global wall
    global element
    # Running Nui or mouse? No problem. We can handle it.
    if(NUI):
        nui.init()
        taskMgr.add(setNuiCameraPos, "cameraPosition")
    else:
        taskMgr.add(setMouseCameraPos, "cameraPosition")

    # Initialize walls depending on screen specifications
    # Save the walls to an array
    wall = Wall(world.space,WIDTH,HEIGHT,CENTER)

    # Select the correct start scenario
    element = selectScenario(world)

    # Wait a split second, then start the simulation
    taskMgr.add(0.5, simulationTask, "Physics Simulation")

    # Run the panda task manager
    run()
```

Figure 13: Entry point of the application

Figure 13 shows that we add several tasks to the task manager, such as the function for updating the scene depending on whether the user wishes to use a Kinect or mouse. It can be seen that the function *nui.init()* is called. This function refers to the layer above and initializes the Kinect. In reference to Figure 7 this satisfies the first step the Kinect must go through. Until the program terminates, the layer above and the function handling the updating of the scene cycles through the last two steps repeatedly. In the last line the function *run()* is called. This starts the task manager and runs the program.

2.5 Perspective Projection

This section gives a shallow description of how perspective projection works. A lot of details are intentionally left out and a purely algorithmic approach will be described. It will also include a more detailed description of how we manipulated the standard projection matrix.

To be able to perceive a 3D world on a 2D screen it is necessary to project the environment onto a projection plane. There are two general projections which are parallel projection and perspective projection. Parallel projection is quite simple as the environment is simply mapped onto the projection plane without a specific focus point. Because the distance to the object is irrelevant the X and Y coordinates are mapped directly onto the projection plane. Perspective projection is harder to do as the correct X and Y values are calculated based upon the depth information. The perspective projection aims to project what the eye sees onto the projection plane. This also automatically means that objects that are closer to the projection plane will appear larger because they take up more of the plane as Figure 14 shows. Figure 14 shows how the two different projections work.

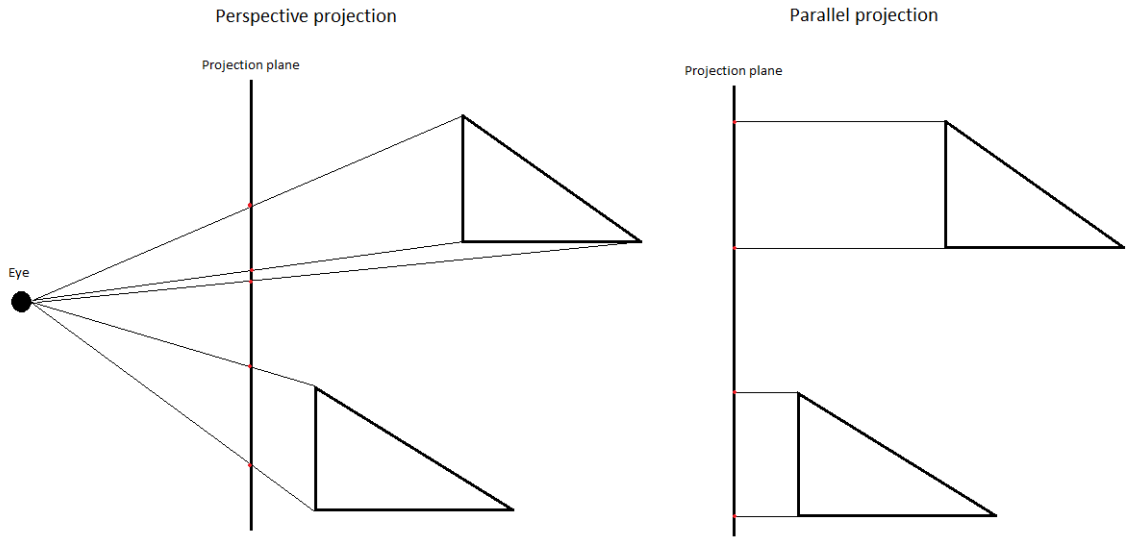


Figure 14: Perspective projection and parallel projection simplified

It is necessary to specify the frustum. The frustum is also sometimes defined as a view volume and it defines what is and is not rendered. As such the frustum should be thought of as what is looked at. Different variables are used to define the frustum but we will not go further into these. Examples of the perspective projection frustum and parallel projection frustum can be seen on Figure 15

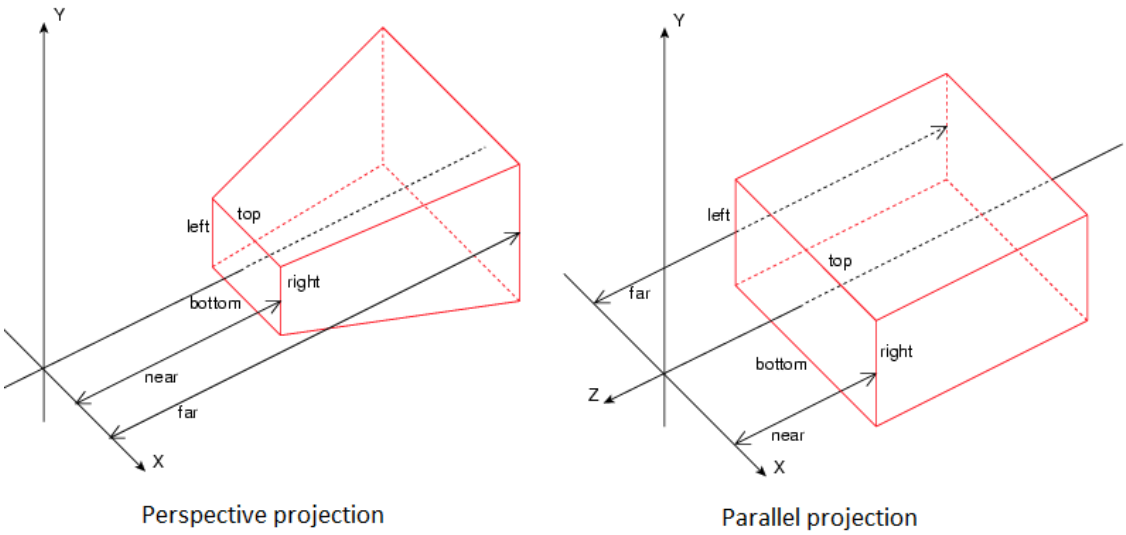


Figure 15: Perspective projection and parallel projection frustum extends

In future figures we will be watching the frustum directly from the side through the X axis and the frustum will be marked green. In the following sections it is explained how to create a perspective projection by first modifying the model and then using parallel projection on this new model.

2.5.1 Perspective projection algorithm

The goal of perspective projection is to display the 3D world on a 2D image as is illustrated on Figure 16. The axes drawn are the camera space representation of the ordinary world space basis vectors. As mentioned earlier this is a perspective projection seen directly from the side. This means the triangles are 3D objects simply shown from the side. The green area is the frustum and the triangles are objects that are projected onto the projection plane.

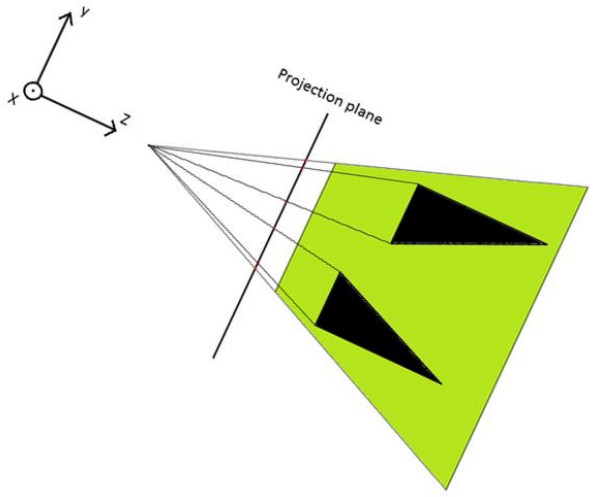


Figure 16: Scene projected onto projection plane from arbitrary viewing angle

To make everything simpler the camera is moved from its arbitrary position to the origin. This can be achieved by going from world space coordinates to camera space coordinates which is done by a simple matrix multiplication. The resulting situation can be seen on Figure 17.

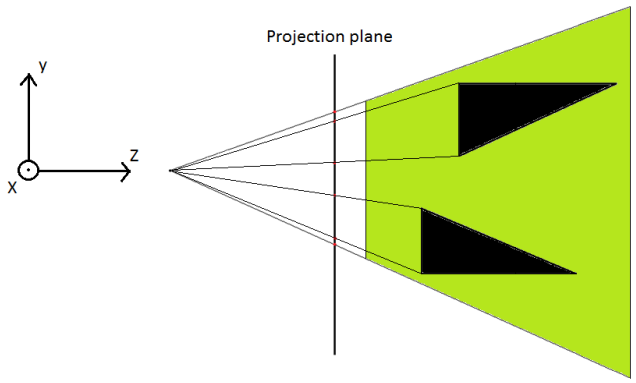


Figure 17: Perspective view volume

It is simpler to do calculations from this position and there is the added benefit that the near and far end of the frustum can now simply be defined by a scalar which is their Z values. From this position it may appear like it isn't easier to perform the perspective projection, but by applying a transformation on the view frustum it is converted to the parallel view frustum as seen on Figure 18. This is done with one matrix multiplication. Doing this matrix transformation does not change the image shown on the projection plane as size ratios are kept constant on the projection plane.

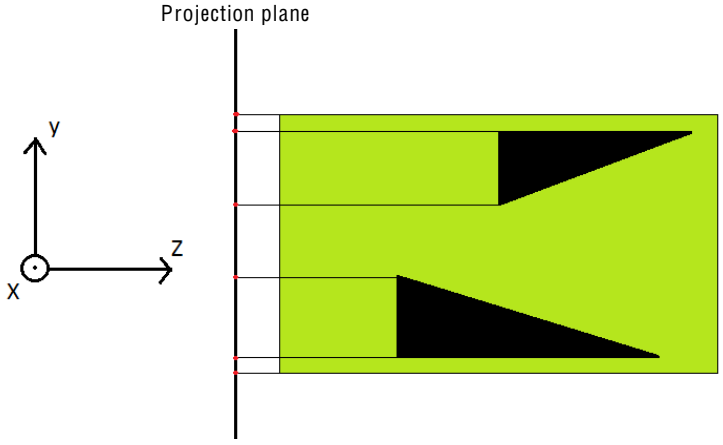


Figure 18: Orthographic/parallel view volume

To further simplify the task of projecting the view volume it is practical to transform the view volume into canonical form. This is done by scaling the view volume and moving it so that it is a 3-dimensional square cube which has its center in origin and has a side length of two. This new situation is shown in Figure 19a in which the projection plane is merged with the left side of the frustum. The objects are now projected parallel to the X and Y plane and

the figure shows that the Z values are used to determine which points should be drawn on the projection plane. Pixels that are behind other drawn pixels are culled leaving only the front of the objects remaining on the projection plane. We now no longer need the Z values and they are removed. What is left is a 2D frustum which is called the *screen space* and is shown on Figure 19b.

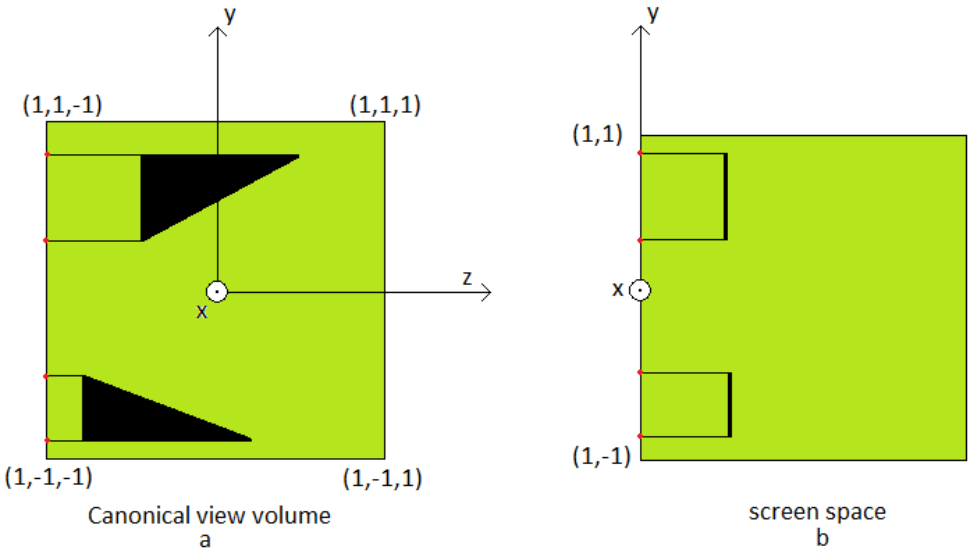


Figure 19: Canonical form of view volume and screen space

From the screen space all that is left to do is to scale the image from the current canonical form and to the wanted image size. This is a simple scaling transformation made even simpler by the fact that the current size ranges from -1 to 1 in both the X and Y plane. The reason behind doing the overall perspective projection like this is that all transformations can be done by matrix multiplication. This means that the result is a single perspective projection matrix P which will do all of the above steps in one single transformation. (Hürst, 2011)

2.5.2 Creation of a 3D-environment

When attempting to create a 3D effect it is important to create the illusion that some objects are outside the screen, meaning between the viewer and the actual screen. To get this effect we decided to use an effect similar to portal culling seen on Figure 20. By making the edges of the screen fit to the edges of an open ended room created in Panda3D, and making this the edges of the portal we achieve this effect. This means that no matter the position from which you view the room, the entrance will always be glued to the edges of the screen.

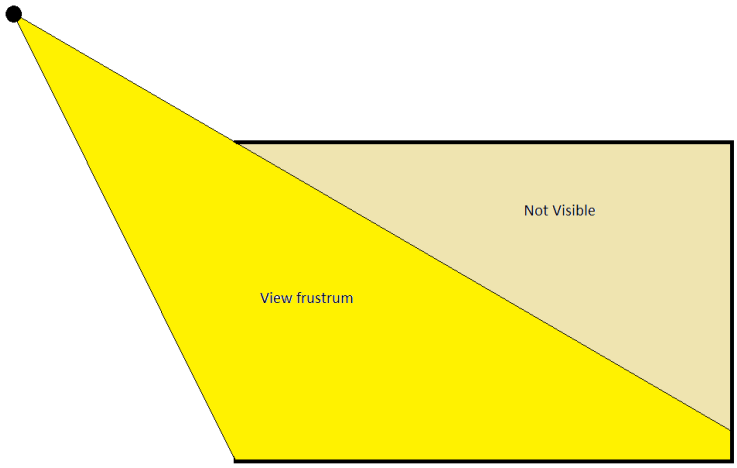


Figure 20: Portal culling effect describing our frustum

To create this effect the normal perspective projection built into most 3D engines will not suffice as they require the user to be stationary. In our project the view position is not centered upon the screen as is assumed for the built in perspective projection.

2.5.3 Generalized off-axis perspective projection

To achieve the wanted effect our perspective projection needs to be determined by the position of the viewer's head for every frame. To do this we need to first define the screen. Figure 21 shows a screen with the edges defined by three of their endpoints which together contain enough information to determine the size, position and aspect ratio of the screen. These points are p_a at the lower left, p_b at the lower right and p_c at the upper left.

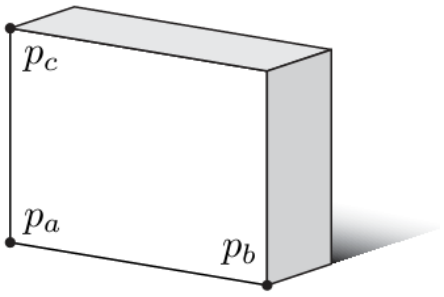


Figure 21: Screen position

From these three points we can create an orthonormal basis of the screen's local coordinate system. To do this we need three normalized vectors each of which is perpendicular to the other two. These will be called v_r, v_u and v_n which are vector right, vector up and vector normal respectively. One can calculate these by using the points p_a, p_b and p_c in the following way

$$v_r = \frac{p_b - p_a}{\|p_b - p_a\|} \quad v_u = \frac{p_c - p_a}{\|p_c - p_d\|} \quad v_n = \frac{v_r \times v_u}{\|v_r \times v_u\|}$$

Just like X, Y and Z can be used to describe the position of a point in space relative to the origin; these vectors v_r, v_u and v_n can be used to define the position of a point relative to the screen origin. This is depicted on Figure 22.

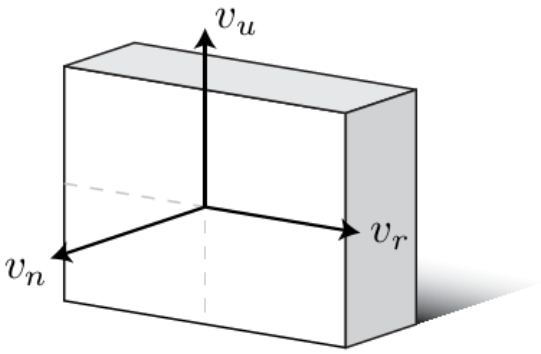


Figure 22: Screen space basis vectors

We now need to introduce the position of the viewer p_e. This is done on Figure 23a.

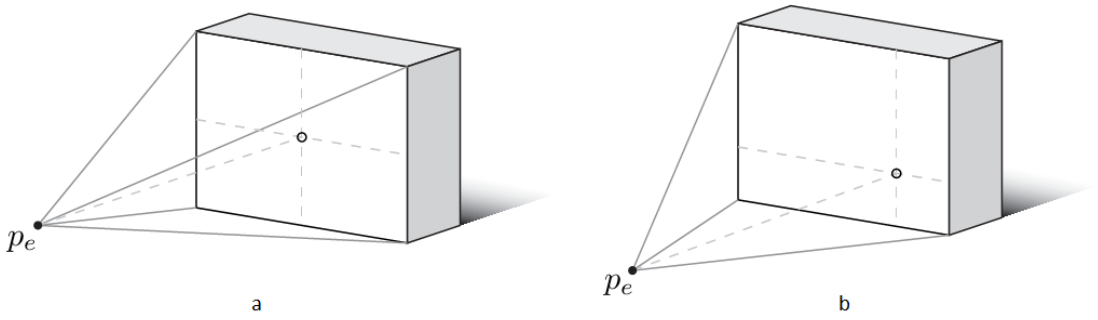


Figure 23: Player position in front of the screen

The position where the line drawn from p_e along v_n intersects the screen is known as the *screen-space origin*. When p_e is positioned centered in front of the screen a perfectly symmetrical cone frustum is the result. This is what is normally used for perspective projection but is not enough for us as we need to take the position of the users head into account. By continuing to call intersection between the screen and the line drawn from p_e along v_n the *screen-space origin*, we get a reference point to describe the extent of the view frustum. To define this extent we introduce six more variables which are *l, r, b, t, n and f* and they define the distance from the *screen-space origin* to the left, the right, the bottom

and the top of the screen as well as the near and far plane respectively as shown by Figure 23b and Figure 24.

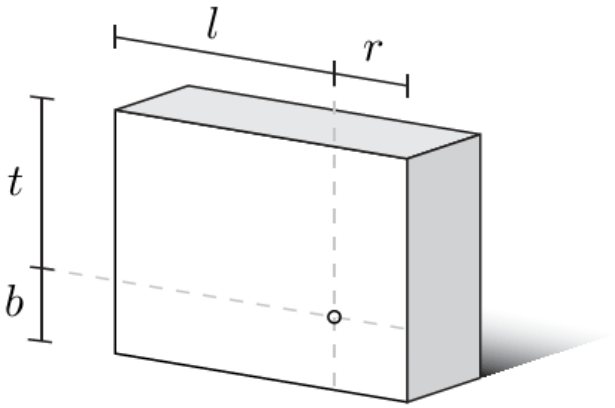


Figure 24: Frustum extends

To calculate the distances *l, r, t* and *b* we need to first calculate the vectors v_a, v_b and v_c. These are the vectors from the position of the viewer's head p_e to the screen corners p_a, p_b and p_c. The vectors are shown on Figure 25 and are computed as follows

$$v_a = p_a - p_e \quad v_b = p_b - p_e \quad v_c = p_c - p_e$$

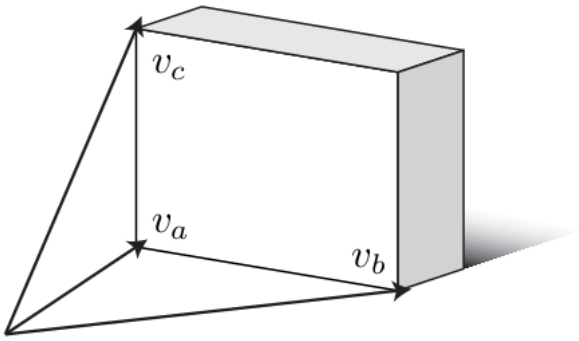


Figure 25: Vectors to screen corners

By taking the dot product of one of these vectors and the corresponding unit vector v_r, v_u or v_n you end up with a scalar which is the distance from the *screen-space origin* to the corresponding side of the screen. Overall the distances are calculated like this

$$l = v_r \cdot v_a \quad r = v_r \cdot v_b \quad b = v_u \cdot v_a \quad t = v_u \cdot v_c$$

Since we are defining the frustum extends at the near plane and the positions p_a, p_b and p_c are defined at the screen, we need to scale the values from the position on the screen onto the near plane. To do this we need to know the distance *d* which is defined as the distance

from p_e to the *screen- space origin.* To calculate this distance one simply takes the dot product of any one of the vectors v_a, v_b or v_c and the unit vector v_n. Because these vectors point in opposite directions one must negate the result. Overall d can be calculated as follows

$$d = -\left(v_n \cdot v_a\right)$$

As the distances to the screen and to the near plane are now known we simply need to divide the distance to the near plane with the distance to the screen plane. The result is the ratio which should be used to calculate the values of l, r, t and b on the near plane which are then redefined as follows

$$l = \left(v_r \cdot v_a\right)\frac{n}{d} \quad r = \left(v_r \cdot v_b\right)\frac{n}{d} \quad b = \left(v_u \cdot v_a\right)\frac{n}{d} \quad t = \left(v_u \cdot v_c\right)\frac{n}{d}$$

By inserting these values into the standard perspective projection matrix P which is defined as shown below, we now have the ability to define a frustum based upon an arbitrary screen and an arbitrary head position

$$P = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

This is not enough though as the frustum is still not aligned correctly. To create the proper portal effect we need to align the frustum based upon the position of the viewer's head.

2.5.3.1 Frustum alignment and composition

The perspective projection matrix we just defined creates the correct frustum based upon the head position of the viewer but this frustum has its apex in the origin. This is depicted in Figure 26 in which camera is denoted as cam and is placed in origin.

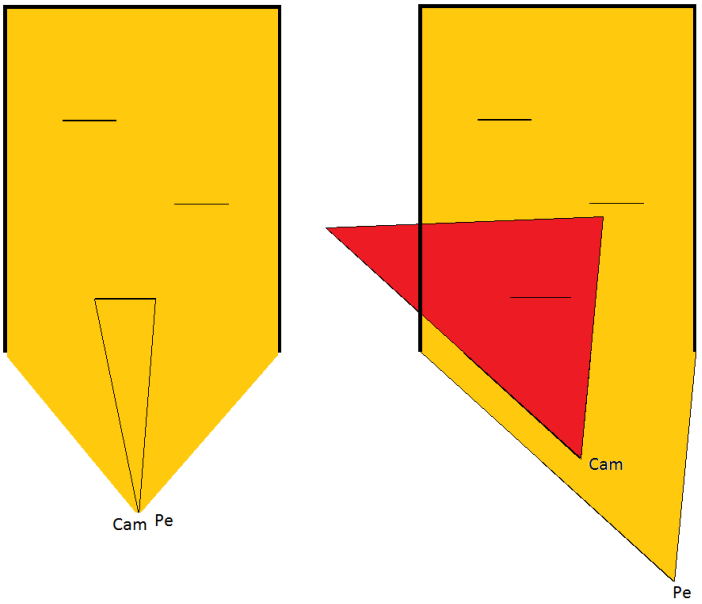


Figure 26: Frustum alignment in generalized off axis projection

Now we need to move the apex of the frustum to the position of the tracked head of the viewer. We would like to simply move the camera but the nature of the perspective projection matrix P disallows this. This means that we instead have to move the entire world the opposite way to get the same effect. We must translate our tracked head position to the apex of our frustum. The apex of the frustum is at zero which means that we have to translate the entire world along the vector from the head p_e. This is done by using the transformation matrix T.

$$T = \begin{bmatrix} 1 & 0 & 0 & -p_{ex} \\ 0 & 1 & 0 & -p_{ey} \\ 0 & 0 & 1 & -p_{ez} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

By then combining P and T we get our final projection matrix P' as

$$P' = P \cdot T = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & -p_{ex} \\ 0 & 1 & 0 & -p_{ey} \\ 0 & 0 & 1 & -p_{ez} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

P' will create a frustum based upon the position of the viewer's head and defined corners of the screen and align it correctly to make the screen appear like a doorway into a room. Because P and T are multiplied, we end up with a single new matrix which will handle all

projection, but contrary to the ordinary perspective projection matrix it will need to be calculated for every frame as the position of the head is a variable. (Kooima, 2009)

2.6 Creating the Virtual Environment

This section will describe the environment we have created which consists of an open ended room attached to the screen edges. This makes the screen appear like an extension of reality. As we will need to test player perception of a virtual environment, we decided that the more the environment mimics actual space the better the perception might be. Coordinates in Panda3D are relatable to real world coordinates by a factor decided by the screen size. Due to this, movement in the real world corresponds directly to movement in the virtual environment. This means that moving to edge of the screen also aligns the room correspondingly.

It should be mentioned that while our environment is a room, the graphical projection matrix can be used on any environment, making the screen appear like a window you can look through.

In our environment we have not taken advantage of all the possible depth cues such as shadows, lighting and familiar objects.

2.6.1 Collision detection

In order to make our environment interactive and natural we decided to implement physics such as collision detection and gravity. Since Panda3D is a game engine this was quite simple as Panda3D boasts several different APIs to do this. We decided to use an API called OdeWorld which is normally used to model environments attempting to mimic the actual world. We will not be going into details of how OdeWorld works, but only describe the very basics of it. First of all we created a simple space in which collision elements were added. Then we defined how the elements should behave. This was done by defining settings such as gravity, bounciness and dampening. Finally we set a collision bitmask which determined whether two elements react upon collision.

2.6.2 Walls

The walls of the virtual room have the role that they limit the space in which we have our model and enhance the illusion of depth. As the tasks we need to perform in the virtual room will vary we need to be able to quickly setup a room to fit our needs. To do this we created a function that will take a center position and build the room around this position. Each wall is created by using a class in Panda3D called *cardmaker*. The *cardmaker* takes a position and an offset in each direction and creates a plane using the inputs. After the plane is created it is rotated into position and settings such as texture and collision detection are applied.

In most virtual environments and in particular gaming environments you have some recognizable structures such as buildings or cars. These structures serve as a mental guideline and help the user estimate depth. While there are no recognizable structures in our environment, we decided upon a chess-pattern texture for the walls. The squares of the chessboard might serve as cues for the user determining depth.

Because the field of view is highly dependent on the position of the viewer and unlike the natural human field of view is limited by the screen size, distortions of the back wall occurs. This distortion makes the back wall seem bigger the closer the viewer gets, which is an unnatural reaction to movement. Because of this fog has been implemented to cover the back wall for most scenarios.

2.6.3 Targets

To measure the general perception of the environment we need to know how size, position and even speed is perceived by the user. To do this we insert some elements into the scene. These elements can be manipulated in size, position and speed, allowing us to set up scenarios from which we can extract specific data. We decided it would be sufficient to use the built in *cardmaker* to create 2D elements instead of using more complex 3D structures.

We also needed to decide on a proper texture as it is substantially harder to tell the location of the objects if they blend into the environment. As a result we decided to use the texture shown on Figure 27.

Figure 27: Texture of Targets

This texture is well defined around the edges making it easily noticeable and it has a pattern which makes it easier to estimate the size or position of the object it is applied to.

2.6.4 Angles

A logical outcome of our 3D perspective projection method is that all objects which originally face the viewer will always do so. This is because a frustum which fits the opening of the room is calculated each frame, regardless of the viewer's position. Since the corners of the room entrance will always be fitted to the corners of the screen, the width and height at the entrance of the room will remain constant. This means that any surface that

initially faces the viewer will have constant width/height dimensions. This phenomenon is seen on Figure 28 in which the angles have not been corrected.

To fix the above mentioned issue we had to apply a fix to the objects in the scene. We decided to rotate all objects in the room for each frame based upon the viewer's position. To do this we needed to find the angle between the user and the target, shown as A on Figure 29. We found this angle by using the tangent function

$$\tan(A) = \frac{opposite}{adjacent}$$

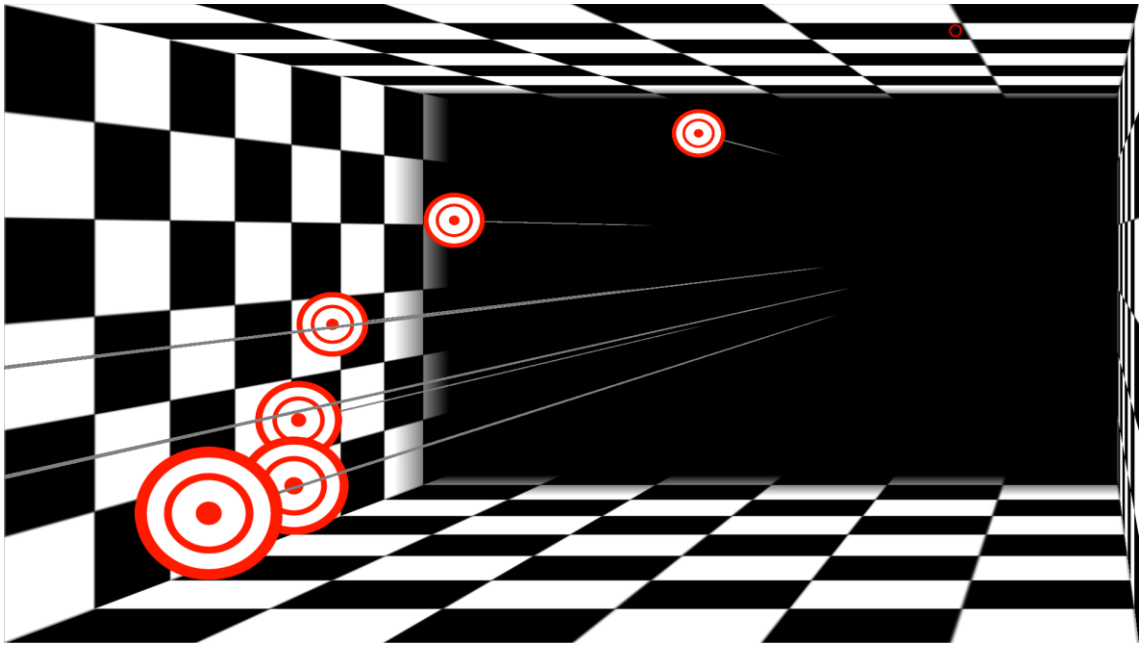


Figure 28: Illustration of targets looking at the player

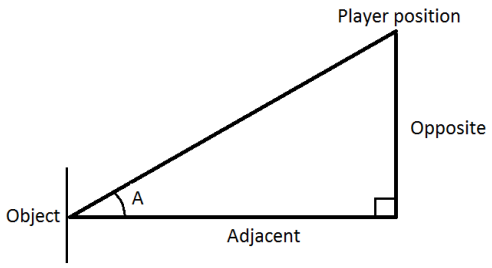


Figure 29: Finding angle depending on player position

To get the length of the adjacent we added the contributing factors such as camera position and player position. To get the length of the opposite we simply took the X or Y position of the user depending on which angle we were calculating.

Knowing the angle of the target compared to the position of the viewer, the object was rotated by A degrees away from the viewer resulting in the correct orientation.

2.6.5 Further development

The way we calculate angles is intuitive though quite outdated and does require more processing power than necessary. Another method which is quite popular in the field of graphics is the use of quaternions to rotate objects. Quaternions are to the imaginary numbers what the imaginary numbers are to the real numbers, namely an extension. Just like imaginary numbers can be used to calculate angles in 2D, quaternions can be used to effectively calculate angles in 3D. Since we calculate and set angles on all objects in the scene for every frame this would be a significantly better solution which should increase the number of frames per second as well as reduce lag in our application.

2.7 Lag and its Impact on 3D Immersion

This section dissects and explains the term lag and briefly describes the effect it has in Virtual Reality systems. The effect of lag on our project is investigated and the results are reviewed.

2.7.1 Introduction

Lag is defined as the delay to a response in a system when it is provided with input. Several research papers have investigated the nature of lag and its impact on human performance (Shneiderman). They all state that to maximize the user's performance a maximum response time should be maintained by the system. It is extremely important to keep the lag at a low rate, because humans are sensitive to it. Furthermore it has been documented that in Virtual Reality systems relying on head- and/or hand-tracking, lag is one of the determining factors of the user's immersion and involvement (Wloka). To enhance the feeling of the Virtual Reality system being a natural extension of the real world, changes in the real world must be reflected in the virtual system with minimal delay. Unfortunately due to the heavy computations often connected with Virtual Reality, lag is present in a lot of these systems and it is hard to circumvent this. As an example, shutter glasses require two images per frame, which increases the amount of computation required by the system and thus introduces lag. Research has shown that virtual reality demands that the lag does not exceed 300ms. If this limit is exceeded the users begin to disassociate themselves from the virtual world, which results in a lost sense of immersion (Wloka). The importance of lag should not be underestimated and in the tradeoff between reducing lag and a higher quality of graphics, lag takes precedence. (MacKenzie, et al., 93)

2.7.2 Sources of lag

For this project four sources of lag were identified and they are illustrated in Figure 30. The first source of lag originates from the Kinect and is called device specific lag. This lag is created because the data from the Kinect can't be delivered instantaneously due to an upper

bound of throughput in the hardware. Using the depth camera furthermore increases the amount of lag by an undetermined amount because of the mapping of IR reflections. The second source of lag comes from the applications using the data from the Kinect and is called application specific lag. The delay is created in the gap of time from when the application receives the data from the Kinect, until it has been processed and is sent to rendering. The main part of the overall lag originates from this source as the data requires a lot of processing.

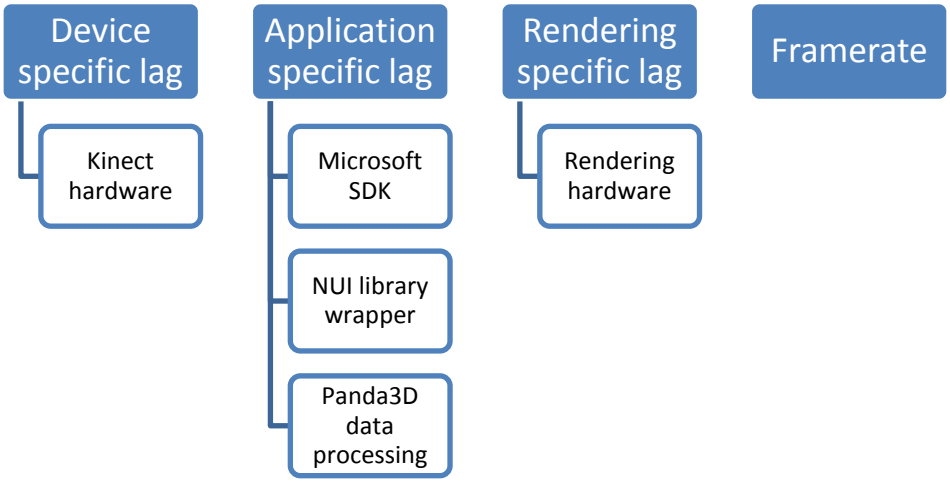


Figure 30: Sources of lag in the system

The rendering specific lag is the third source of lag and is defined by the time it takes from when the data is sent from the application until it is shown on the monitor. The fourth and last source of lag is frame-rate induced lag and is only a problem if the frame-rate of the system is too low. The reason it occurs is because every time a new frame is shown on the monitor it is considered to be up to date, but as the next frame is prepared the current frame grows older. When the frame-rate is low, each frame appears for longer periods of time on the monitor and this results in visual lag.

2.7.3 Testing of lag

To determine whether lag had an effect on this project, tests were conducted to determine the total amount of lag in the system. The reasoning behind this was to determine if the lag in the system could have an effect on the user's sense of immersion. Furthermore a comparison could be made between the developed application and the built in Microsoft sample programs. This was done to allow an estimation of the contribution to lag made by the application.

2.7.3.1 Setup and procedure

The test environment consisted of a Kinect connected to a computer, a TV and a camera for recording the interaction with the system. The TV was by the manufacturer LG and had a display delay of 8-10ms. The camera used for recording was the webcam in the HP Envy

17-3D 2090eo, which produces the video with 29 frames per second. A recording of interaction using the skeleton tracking sample included in the Microsoft Kinect SDK and the application was made.

A test subject was asked to move his stretched arm from a position over his head to his thighs in a downwards motion as quickly as possible. The user's movement in front of the Kinect and the reaction on the TV was captured with the camera. Replaying the video afterwards, the delay between the real movement and the reaction in the virtual environment could be seen by analyzing the frames in Windows Movie Maker.

2.7.3.2 Results

The products of the tests were two videos which were analyzed. There were two samples from each video making for a total of four samples.

Developed application								
Sample nr.	User start time	User end time	Kinect start time	Kinect end time	User delta	Kinect delta	Start time lag	End time lag
1	6,26	6,59	6,55	6,86	0,33	0,31	0,29	0,27
2	10,60	10,88	10,90	10,14	0,28	0,24	0,30	0,26
Microsoft Kinect SDK								
Sample #	User start time	User end time	Kinect start time	Kinect end time	User delta	Kinect delta	Start time lag	End time lag
3	13,37	13,90	13,67	14,17	0,53	0,5	0,30	0,27
4	14,94	15,34	15,20	15,55	0,4	0,35	0,26	0,21

Table 1: Results from lag tests shown in seconds

All times in the above table is in seconds with a precision of two decimals. The application was capped at 60 FPS and maintained this for the entire test session.

User delta and Kinect delta defines the duration of the corresponding motion and is calculated by subtracting the start time from the end time of the respective motion. Start time lag and end time lag defines the difference between the user's movement and the reaction on the screen.

2.7.4 Remarks and discussion

As mentioned earlier in the report an upper boundary of lag in virtual reality is 300ms. If the lag exceeds this the users begin to disassociate themselves from the virtual environment. It has however not been researched if this disassociation results in a loss of depth perception.

The approximate lag of our system is 300ms. The difference in the User delta and the Kinect delta can be explained by varying lag from the Kinect or a single missed frame at either the start or end of motion. The precision of the lag in our application is 40ms seen in sample two. This is seen as the highest measured difference in Start time lag and End time lag. This puts the range of the lag in our system at 260ms to 340ms.

Compared to the sample program provided by Microsoft we see a similar amount of lag. From this we conclude that the primary source of lag in our system is not in the application but somewhere in the Microsoft Kinect SDK or in the Kinect itself.

The camera used to record this may have had an impact on the results and the low number of samples may also be inadequate to draw any definite conclusions.

2.7.5 Further development

By introducing prediction supported by multicore processing, we could create a number of predictions for the next frame based upon the current frame and the previous frames. Then if the data retrieved from the Kinect matches any of these predictions, we can simply take the result straight away. This would reduce the lag contribution from our application significantly.

2.8 Player Experience

By introducing head tracked 3D to an environment we wish to increase depth perception and thus immersion and potentially success rate of the challenges presented to a player. By targeting some of the factors known to be parts of the player experience we hope to increase the overall experience and performance of the player.

2.8.1 Definition

The concept of player experience is hard to define as it consists of several different terms such as fun, pleasure, immersion and flow just to name a few. As these terms are often defined nearly the same way and are used interchangeably it makes it very hard to measure any of these components of player experience. To further complicate matters there is definite agreement upon which of the attributes that are the most important and no empirical data to support the claims being made. It is however agreed that player experience plays an important role in the player's interaction with the game as well as the longevity of the game.

Attempts have been made to use HCI as an evaluation tool for measuring player experience but it was found inadequate as it turns out that player experience is more complicated. It differs from HCI in the core aspect that HCI is applied to applications whose purpose are to increase productivity and i.e. achieve a goal. Player experience is focused on increasing the user's experience during the process. Furthermore it is now known that a long list of factors such as company branding, previous experiences and indeed the current mood of the player

have impact on the player experience. On top of this there is a temporal aspect to measuring player experience as it is a product of both previous experiences and the current setting in which the application takes place. From all this we can conclude that games are indeed complex and requires their own field of study. To this day player experience remains an elusive term which is currently only possible to measure by breaking it down into its lesser components.

There are many examples of games with poor usability being successful and it is a generally accepted fact that when dealing with game software, player experience should always be prioritized, even if it means this might have a negative impact upon usability or functionality. The game can be complex but it should be possible for the user to learn how to handle this complexity. It is important that the learning curve of the game matches the skill of the target audience. If the game turns out to be too easy or too hard it will discourage the player and might even make them quit playing the game altogether. (Nacke, et al.)

2.8.2 Contribution to player experience from NUI

This section will discuss the effect interaction with a virtual environment using a NUI has on player experience. Investigations from various studies show that physical activity is related directly to mental well- being. Furthermore increased beneficial effects on self-image and even food intake have been measured on players as they improve in movement based games in comparison with non-movement based games. As previously mentioned personal context such as state of mind and image on self, have direct effect on player experience. From this it can be derived that the more the players move, the more they are immersed into the game, and the more they will want to move (Bianchi-Berthouze).

According to the model proposed by (*Ermi and Mäyrä (2005)* cited by (Bianchi-Berthouze)) immersion can be divided into three subcategories namely challenge-based immersion, imaginative immersion and sensory immersion. While challenge-based and imaginative immersion is well understood, the movement-based aspect of sensory immersion is less so. Research has shown that sense of presence in a virtual environment is enhanced by movement when correct sensory feedback is given, so the user accepts the virtual environment as a natural extension of the real world (Bianchi-Berthouze).

Studies show that the relationship between an enjoyable situation or experience and player experience differs depending on the realism of the situation and the motivation of the player. In general it seems that the player experience in a role-related environment is increased by any factor contributing to the illusion of the role (Bianchi-Berthouze). An example of this is the game Guitar Hero released in 2005 by RedOctane where the player uses a guitar shaped controller

3 Experiments

This section will describe the experiments made with our system to investigate the user's perception of the scene. The test subjects will be briefly described. Following this, the environment and the technical setup of the experiments will be described. Each of the experiments has a specific purpose and because of this there is a dedicated subsection for each experiment. They each contain a description of the setup, goals, expected results, actual results and a review of the results. As all of the experiments have some commonalities we have included a section describing the generally expected results.

3.1 Setup

The experiments were conducted over the course of a single day. Each experiment took between 30 and 45 minutes. It has been shown that fatigue is more present in virtual environments than it is in the real world (Yongwan, et al., 2011). To prevent physical and mental exhaustion from affecting the test subjects, short breaks were occasionally taken between the experiments. The same setup was kept throughout all the tests and the strength of the light in the room was kept constant.

The setup consisted of a computer, a Kinect, a wireless mouse and a projector. The projector was stationed on a table approximately 2 meters from a white wall. The aspect ratio of the image was 4:3 and the dimensions were 65cm in height and 49cm in width. The resolution of the projected image was 1600x1200. The Kinect was placed behind the projector on the table so it faced towards the room. This allowed the subjects to utilize the full range of the Kinect view volume. There were no noticeable disturbances in the depth image from the Kinect. The mouse was a Logitech m705 with 6 buttons.

3.2 Subjects

Seven different test subjects participated in the conducted experiments. All test subjects were male ranging from 20 and 27 years of age and they were all students at DIKU. None of them had any prior experience in using the system, nor did any of them have any previous experience with a system reminiscent of this one. The height of the subjects ranged between 180cm and 195cm. None of them had any problems with their sight or required any form of visual aids such as glasses or contact lenses. The test subjects were not offered any compensation for the time used on the experiments.

3.3 General Procedure

For the duration of all the experiments, both authors of this report were present as supervisors. Each subject was told that they would be participating in an experiment which consisted of a series of tests. They were then introduced to the system as a form of Virtual Reality based on head tracking. They were informed of the expected duration of the experiments prior to the start. Before a new part of the experiment was started the subject

was introduced to the task at hand. Only one test subject was present in the room at any given time. Manuscripts were prepared so everyone would receive the same information.

Each experiment was divided into two parts and each part was done with and without head tracking. In the experiments without head tracking, the camera was locked in center position. First a high deviation experiment in which the parameters such as apparent size or depth were allowed a wide range. We define apparent size as the visual size of the object in the user's field of view. When the user had finished the high deviation experiment they had to complete a low deviation version of the experiment in which the parameters were allowed a smaller range. The 20 first answers were logged for each version of all the experiments.

We opted against using feedback as it has been shown that feedback encourages a learning curve in virtual environments (Waller). A learning curve was undesirable as we wanted the user's skill level to be as constant as possible throughout the experiments.

Each time an experiment had finished, all data recorded was written to a text file. To plot the data from the experiments we created a parser in Python which read all the text files and inserted their data into a dictionary. This made the data readily available for plotting and provided the option of easily increasing the amount of test subjects. To create the charts we used a library for Python called matplotlib. To process the data we used the Python libraries SciPy and NumPy.

To create the charts for correctness we sorted by indices and took the data in chunks of 7 giving us 20 measurement points. We chose this because we had 7 test subjects, each of which did 20 measurements for each experiment. The number of correct answers was divided by the number of answers in the dataset and multiplied by 100, resulting in a number describing the percentage of correct answers.

The charts for answer times were generated by sorting the indices and taking the median of the corresponding data in chunks of 7. We used the median of the data to eliminate potential outliers and we used the average on the indices because we knew there were no outliers. The median absolute deviation is also plotted along with the median in the charts for answer times and has been calculated by the corresponding chunks of data. It is shown both above and below the median in dotted thin lines. We picked the median absolute deviation on the basis that it provides a way of eliminating outliers from the dataset. We did not use the standard deviation since it is not as resilient to outliers.

3.4 General Expected Results

It was expected that when the user had no head tracking, they would base their answer on the apparent size of the objects. This method of determining depth and size is insufficient as large objects far away could appear to be small objects close to the viewer and vice versa.

The deviation in apparent size of the object was intended to aid the user in determining the depth, but could also have a negative effect if the closest object was moved further away. This would make its apparent size smaller than the other target. The deviation did in some cases make the answer obvious as the repositioning of a target made it visible against one of the walls, providing an essential depth cue.

We were also interested in determining the amount of extra time needed to take advantage of the increased depth perception and the parallax cue resulting from the use of head tracking. When a user moves, the relative visual velocity of stationary objects depends on their distance from the user. This is known as motion parallax, also referred to as the parallax cue and it is known to be a strong cue to depth perception (Rolland, et al.). Our concern was that the user might need too much time in order to produce the correct result. This would make the head tracking technique impractical for real world applications.

Some variations are expected in the graphs, but these could very well be caused by too few measurements and as such the main focus will be on overall trends.

3.5 Comparative depth perception

3.5.1 Rationale and setup

One way the measure the user's perception of an environment is to test his or her ability to judge exocentric (interobject) depth. To measure this we inserted two targets into each side of the scene. The targets could vary in depth and size and these were chosen randomly, though with constraints. The assignment for the test subjects was to judge which of the elements was the closest to them. To answer the user had to left or right click on the wireless mouse depending on which of the targets they deemed to be the correct answer. When a click was registered from the mouse, our application logged the correctness of the answer, the position of the targets and the time it took to answer.

When randomly generating the targets and thus allowing for all outcomes, too many obvious situations appeared. To avoid this we decided to only test the situations where the users might have a difficult time determining the correct answer. These are the situations where the targets may appear to be side by side, but in fact the bigger object is simply further away which creates the illusion of equal depth. To achieve this effect we calculated the apparent size of the left target in the user's field of view and positioned the right target to match this size in the user's field of view. The achieved effect can be seen on Figure 31. To introduce some variance in the apparent sizes we allowed the target's position to deviate from the calculated position by a margin depending on the scenario. This made them look bigger or smaller depending on the direction they were moved. In the case of the high deviation we allowed a variance of 15 to 20 units in Panda3D. In the low deviation of the experiment we allowed for a variance of 0 to 10 units.

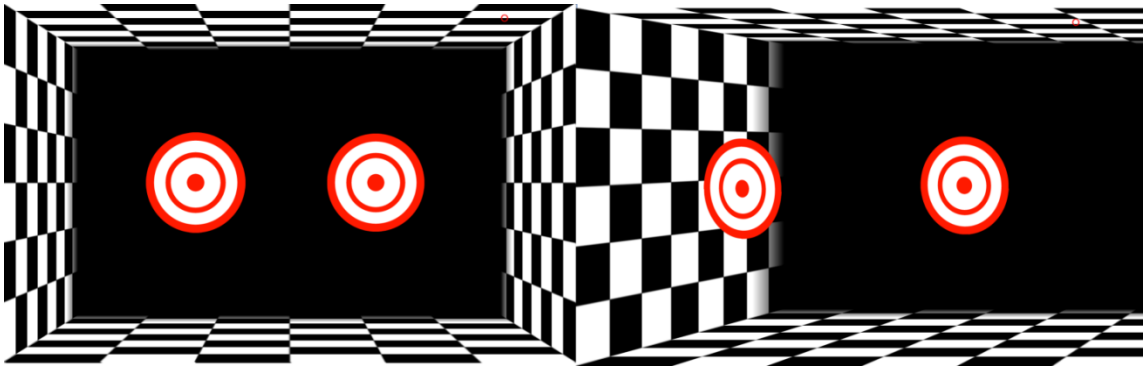


Figure 31: Same scene seen from two different angles demonstration illusion of depth

3.5.2 Expected results

When the user had head tracking to aid them we expected the parallax cue to help them get close to perfect accuracy for most exocentric distances. This stands in contradiction to the same situation with no head tracking, in which we expected the average of the correct answer percentage to be well above 50. This would be helped along by the deviation in apparent size as well as the environment. In the case of very low exocentric distances we expected the parallax cue to be less efficient and this should lead to a lower overall correctness. Judging very low exocentric distances would correspond to telling which target is five to 10 centimeters closer at an egocentric(between viewer and target) distance of 4 meters.

It was expected that when the user had no head tracking, a constant response time no matter the position difference of the targets would be observed. In general we expected that some time was needed to take advantage of the parallax cue. As a result the time needed to answer while using head tracking should always be somewhat higher than without head tracking. With the use of head tracking, we expected that for larger distances between targets the response time would be close to the one without head tracking. This would be due to the user being able to use the same visual cues as for no head tracking, or the parallax cue being explicit. At very low exocentric distances the user would need to utilize the parallax cue to its full extend to get the correct answer. Therefore we expected the time needed to choose the correct target to be higher in these situations.

3.5.3 Results and discussion

The results from this experiment are shown on Figure 32.

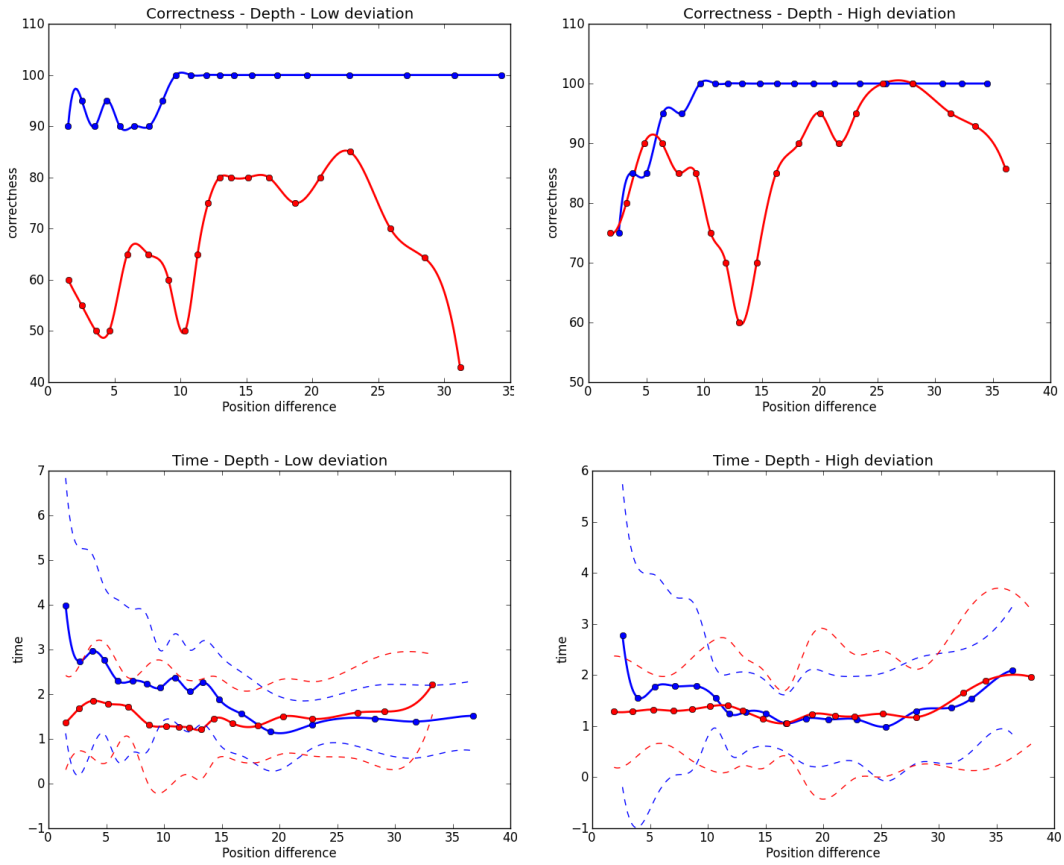


Figure 32: Top charts show the correctness and the position difference of the targets. The bottom two charts show the answer time in seconds and the position difference of the targets. The blue line is with 3D and the red line is without. Dotted lines are standard deviation of the correspondingly colored line.

As expected the users picked the correct target more often while using head tracking. Head tracking resulted in a near perfect estimation of comparative depth. By using head tracking the users got 96.6 percent of the answers correct, while not using head tracking only resulted in 68.5 percent. This is an improvement of 28 percent.

In the top two charts we see that the results without head tracking seem to be random. This can be explained by the observation that the naïve way of judging depth by size, would work in at least 50 percent of the cases. The deviation and the environment seem to help the test subjects pick the correct answer. This can be seen by comparing the red lines of the correctness charts for high and low deviation.

The answer times without head tracking on the bottom two charts in Figure 32 are more or less constant as expected. This suggests that the test subjects only utilized the apparent size of the objects in their judgment of the depth. The answer times vary more with head tracking, but they do however follow a declining curve as the position difference of the targets increases. This seems to indicate a definite connection between answer time and position difference in depth perception. The declining slope of the blue lines suggest that

the test subjects were less secure in their choice of target for low exocentric distances and therefore required extra time to utilize the parallax cue. This is also evident from the standard deviation for head tracking which starts high and proceeds to drop as the position difference increases. Both answer time charts have an intersection point between the red and the blue line, and after this intersection the lines follow each other. This seems to indicate that there is a point at which the parallax cue becomes just as fast as or faster than using the environment and apparent size for depth perception.

It was observed that when using head tracking, the test subjects used different methods to determine the depth of the targets. In most cases the users utilized the parallax cue by moving from side to side or head bopping constantly. A few test subjects decided to observe the scene from an extreme angle and thereby judge which target was closer. However when the exocentric distance was low everybody decided to use the parallax cue. We see that for low exocentric distances the users had a higher error rate which was expected.

3.6 Comparative size perception

3.6.1 Rationale and setup

To get a better idea of exactly how good the user's perception of the virtual environment is, we needed to test how well they could tell the size of objects, in particular which of two objects was the largest. To do this the user would need to first judge not only which object was the closest, but also how much closer it was, to make a correct decision of which object was the largest.

To test this we setup the scene exactly the same way as we did to measure comparative depth perception and thus this will not be described again.

3.6.2 Expected results

In this scenario the deviation in position was expected to have a negative impact on the user's ability to judge size when using head tracking. This is because two objects of close to equal size will be easier to tell apart when they are side by side, which they will almost be without the deviation. The deviation might move the right target either further away or closer to the user. The resulting exocentric distance would make it harder to tell which object was larger. The effect was not expected to have an impact on the overall results of experiment without head tracking.

Because it is a more complicated task to determine the size of objects in a virtual environment, we expected the error rate to be somewhat higher than what was seen when measuring comparative depth perception. At very low radius differences we expected approximately 50 percent of the answers to be correct with or without the use of head tracking. The reason for this is that if the two targets have nearly the same radius, they will also be positioned side by side with only the position deviation to move them away from

each other. The deviation can move the targets either away or towards the user with an equal probability and as such it shouldn't alter the overall precision of the user's correct answer ratio. As the radius difference increased we expected the results with head tracking to improve. This is because the users could estimate depth and use this information to decide which target was larger. We expect the results for no head tracking to be somewhat random but with a correct answer ratio above 50 percent as some targets may be positioned in a way that gives the user some unintended depth information.

3.6.3 Results and discussion

The results of the comparative size perception experiment are shown on Figure 33.

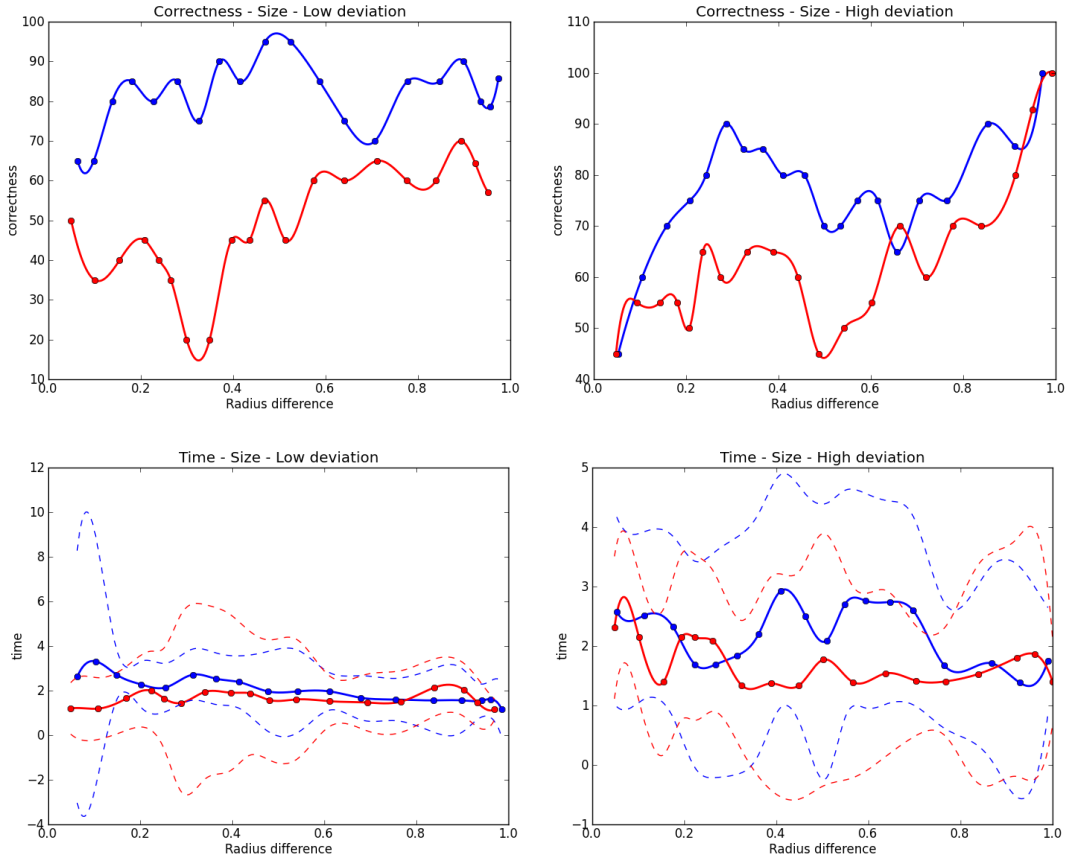


Figure 33: The top two charts show the correctness and the radius difference of the targets. The bottom two charts show the answer time in seconds and the radius difference of the targets. The blue line is with 3D and the red line is without. Dotted lines are standard deviation of the correspondingly colored line.

The test subjects generally had a success rate of 80 percent with the use of head tracking in comparison to 49.3 without the use of head tracking. This is an improvement of 30.7 percent. As expected the general tendency is that the larger the radius difference, the higher the amount of correct answers are. We see that for very low radius differences the success rates with and without head tracking are similar which was expected. The measurements at the end of the high deviation graph show a perfect success rate both with and without head

tracking. This is because the algorithm that positions the targets will position one target at the far back or front and the other at the opposite position in order to achieve an equal apparent size of the objects. Because the small target will have to be at the front it will be easy to spot its actual size and position because it will be visible against one of the walls as demonstrated on Figure 34.

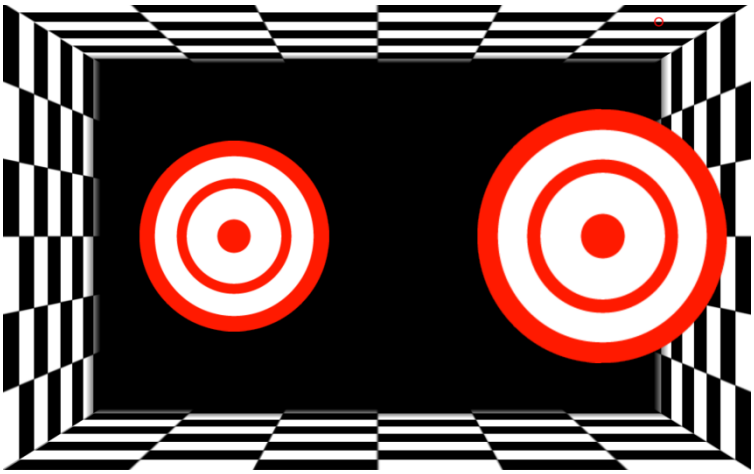


Figure 34: High radius difference results in depth cue

From the lower two charts in Figure 33 we see that the overall tendency is quite similar to the tendency for time needed when estimating comparative depth (bottom two charts in Figure 32). This was expected as the task of estimating size is highly dependent on first estimating distance to target. The time from which the depth is known and until the user is able to make a judgment on size seems to be very short as the answer times measured in comparative size perception are only slightly higher than the times measured in the comparative depth perception experiment.

3.7 Comparative speed perception

3.7.1 Rationale and setup

The goal of this experiment is to gain an understanding of the user's perception of movement in the virtual environment, as it builds upon the relative depth perception in a temporal dimension.

The experiment with low deviation was made by having two targets move horizontally across the screen at different speeds. Given the two targets the test subject was to pick the one moving faster. In the test with high deviation both targets started in the opposite end of the room and moved towards the player. Again the test subject was to pick the one moving faster.

3.7.2 Expected Results

With and without head tracking the test subject was expected to reach a near perfect correctness regardless of the deviation. It was expected that perception of speed would be aided by introducing head tracking. Using the parallax cue, the test subject was expected to have a slightly faster response time when using head tracking. Without head tracking the user would be limited to judging speed by the increasing apparent sizes of the targets. While this method should provide a good indication of speed it was expected to be less efficient. For both experiments the answer time should be declining as the speed difference increases. When the speed difference was large the exocentric distance would increase faster, thus making the choice easier.

3.7.3 Results and discussion

The results from the experiments are illustrated in the four graphs on Figure 35.

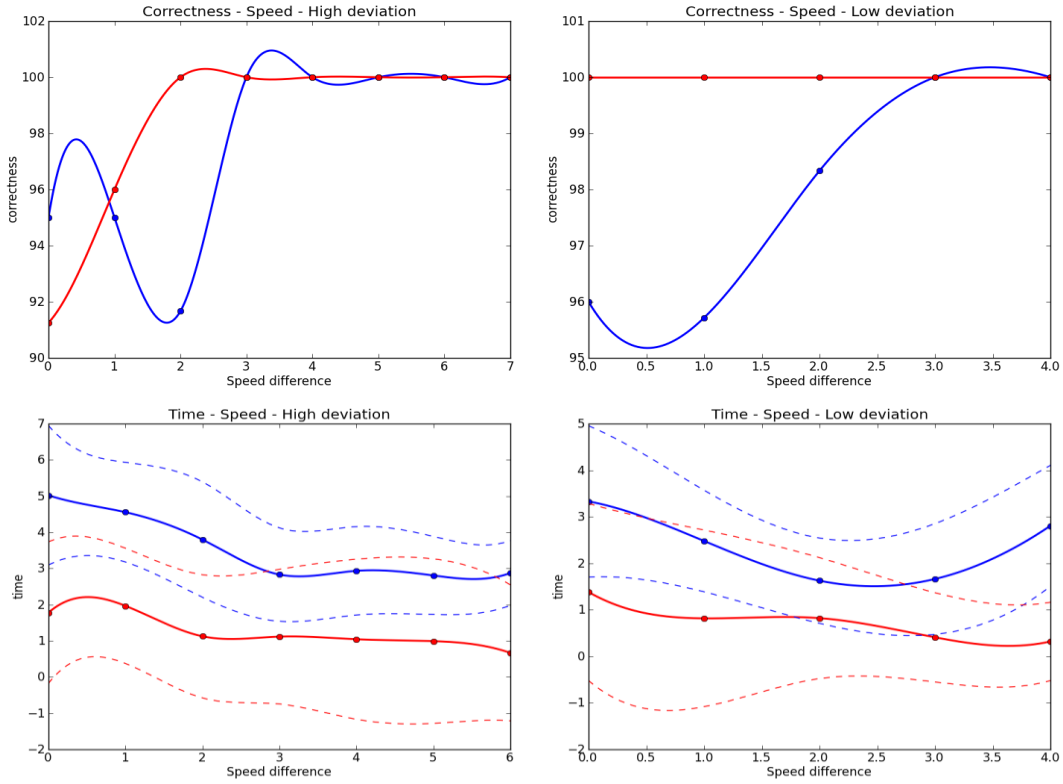


Figure 35: The two top charts show the correctness and the speed difference of the targets. The bottom two charts show the answer time in seconds and the speed difference of the targets. The blue line is with 3D and the red line is without. Dotted lines are standard deviation of the correspondingly colored line.

After conducting the experiments it was realized that only five different outcomes for the low deviation and eight different outcomes for the high deviation were possible. The impact of this design flaw in the experiment is visible in the charts, where it can be seen that only a few indicator points are present.

As expected the correctness was near perfect both with and without head tracking. All samples lie in the range between 90 and 100 percent, which can be seen in the top two charts of Figure 35. It was observed during the experiments that the test subjects employed different strategies to judge speed. Many of the subjects tended to start out by utilizing the horizontal parallax cue, but quickly realized that this was an ineffective strategy. In alternating from side to side the object on the opposite side would seem to approach the user faster even if this was not the case. Upon realizing their failure to determine the relative speed with this approach, the users either stood still and used the size cue to determine speed, or utilized the vertical parallax cue by moving up and down. Both tactics proved vastly superior to the initial strategy.

In the bottom two charts of Figure 35 it can be seen that the answer times for head tracking is generally higher than without. This can be explained by the test subjects' inefficient use of strategies to determine the depth of the objects. The answer times are generally declining in both graphs, which was expected. There does however seem to be a lower limit to the answer times in both cases. It should be noted that the decreased time to make an answer did not seem to have any direct effect on the subject's ability to answer correctly.

3.8 Distance to Target

3.8.1 Rationale and setup

To gain an understanding of the user's perception of depth, we conducted an experiment to determine how well the user estimated exocentric distances. The point of the experiment was to see if the user generally over or underestimated distances in the environment.

In the scene of the experiment there are two targets present. One of them is henceforth denoted as the *reference-target*, and the other as the *distance-target*. The scene is shown on Figure 36. It was up to the user to select the correct distance to the distance-target, given the distance to the reference-target. The reference-target would always be placed closer to the user than the distance-target. There was text floating above the reference-target expressing the distance from the user to the target. Located above the distance-target were three numbers of increasing value. Of the three values, one of them expressed the correct distance from the distance-target to the user. The difference between any two consecutive values was constant; each rising with one fifth of the exocentric distance of the targets. The correct value could be any of the three. The user was to select the correct value using one of three buttons on the mouse.

Unlike the other experiments the size of the targets was equal and their position was random, only limited by the scene.

The experiment was to be conducted both with and without 3D.

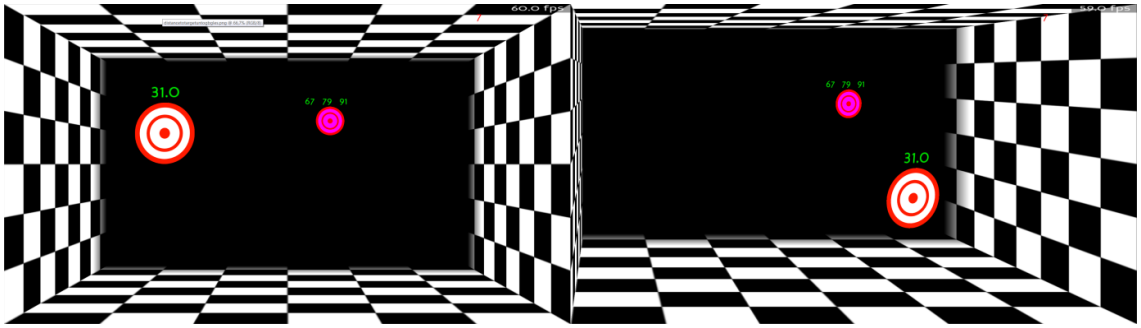


Figure 36: Distance to target setup, observed with and without head tracking

3.8.2 Expected results

Previous research papers investigating egocentric distances have found test subjects generally tend to underestimate the distance to targets in virtual environments while overestimating exocentric distances. (Waller) Based on this, the expectation was that the test subjects would overestimate the distance to the targets in general.

It was expected that the user would be able to utilize the parallax cue when head tracking was enabled. This should provide a clearer tendency in results than without head tracking. Without head tracking it was expected that the results would be random, due to the test subject's incapability of guessing exact depth based on the size of an object.

3.8.3 Results and discussion

The results from the experiments can be seen on Figure 37 and Table 2.

	Correct %	Overestimation %	Underestimation %
With head tracking	41.5	38.89	19.60
Without head tracking	32.25	34.28	33.46

Table 2: Showing the overall distribution of answers

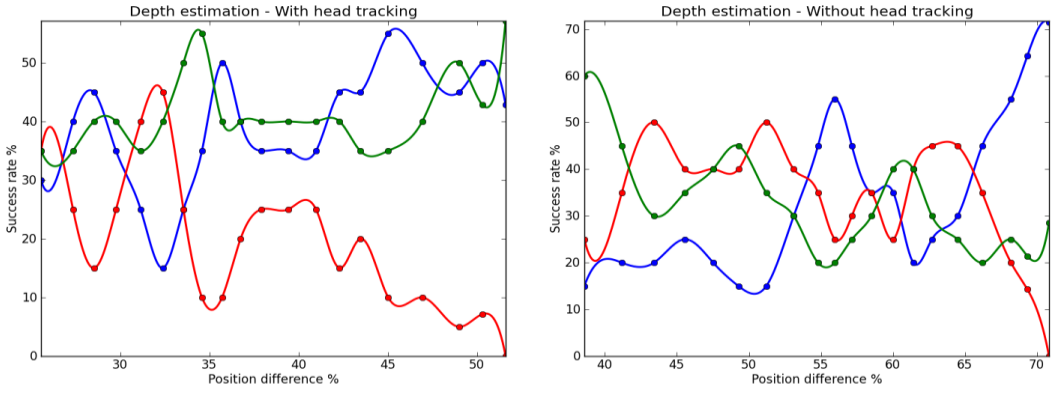


Figure 37: **The test subjects' success rate in the exp**eriment depth estimation. Green bar shows correct answer percentage, red bar is underestimation percentage and the blue bar shows overestimation percentage

During testing a bug in the experiment was discovered. While not using head tracking the reference-target could be positioned outside the field of view making it extremely difficult to judge the correct distance to the distance-target. The bug could only occur when the reference-target was positioned very close to the user. We have no way to exclude the data from the bugged situations. The chance of the bug occurring is proportional to the position difference, as higher position differences means the reference target is more likely to be close to the user. As the bug was discovered we instructed the users to simply click any button when the bug was present. Due to the design of the wireless mouse it is more likely that they either right or left clicked. This could explain the phenomenon seen on the right graph in Figure 37.

From Table 2 it is evident that a better success rate is achieved when using head tracking than without. The choices made by the users are equally distributed among the three options and this indicates that their perception of depth was not good enough to judge exact distances.

The general trend while using head tracking is that the user either picks the correct target or overestimates the distance. In the real world egocentric distances are usually underestimated while exocentric distances are often overestimated by a factor of 20 to 40 percent. It has also been shown that there is slight overestimation of exocentric distances in virtual environments (Waller). When analyzing our results we get to a similar conclusion. At small differences in distance it seems to be too hard for the user to make the correct choice. This may be because the parallax cue is less explicit in these situations or because the answer choices were too similar. However as the exocentric distance increases so does the user's tendency to overestimate the distance. This indicates that the user takes advantage of some of the same cues as in the real world for perception of depth.

3.9 3D Perception

3.9.1 Rationale and setup

While the other experiments have been aimed at measuring hard data about how users are able to use head tracked 3D to estimate depth, size and speed, we still had no way to compare the 3D experience we have created to other products on the current market. To do this we setup a scene containing a series of targets in a spiral pattern. Having been inspired by (Lee) we added rays to the targets that continue into the fog. These rays serve as reference points and add a feeling of depth which improves the 3D experience. The setup of the scene can be seen on Figure 38.

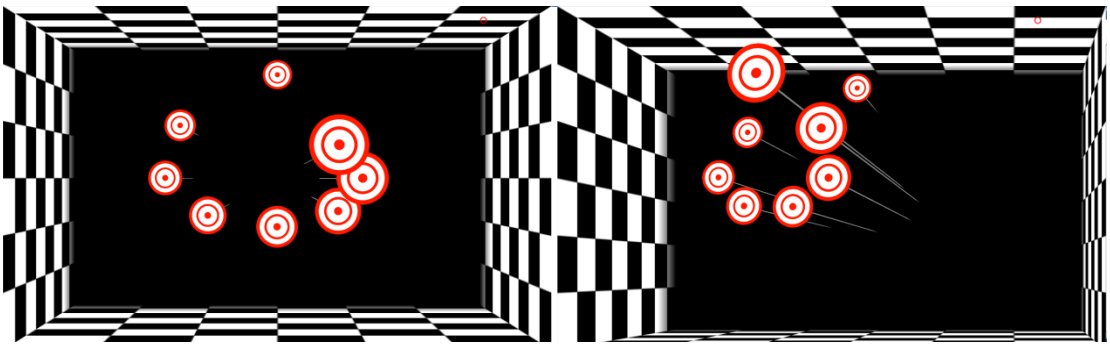


Figure 38: Perception of depth experiment seen with and without head tracking

In the actual test we simply ask the user to tell us subjectively if they experience the wanted 3D effect or if it shines through that it is a 3D scene rendered on a 2D screen. We then asked the user to close one eye and tell us if this had changed their perception of the scene. This was to check for the influence of stereoscopic vision.

3.9.2 Expected results

We know that humans have stereoscopic vision and we expected this would influence the perception of the scene. As such we expected that the users would generally not experience the full 3D experience with both eyes open. By asking the user to close one eye we expected the user to experience the wanted 3D effect, since stereoscopic vision would no longer be available to them.

3.9.3 Results and discussion

As expected the users all agreed that with both eyes open the 3D experience was not evident. When asked to close one eye, some but not all of the users, said it was better. This was unexpected because we had video of the scene and in the video the 3D effect was very clear and impressive. After having received feedback from the users we showed them this video and they all agreed that the 3D effect is much better on video than it is in reality.

Research into this area showed that many people are not able to experience objects that are outside the screen because they perceive the screen as a window they look through and as such perceive all objects as on the other side of the window. Another reason for the confusion in depth perception is that objects are shown at the screen surface even though they are supposed to be between the user and the screen. This causes unrealistic left and right images for the user. It is possible to fix this issue by use of 3D glasses and a technique known as cadre viewing (Mulder, et al.).

3.10 Racket

3.10.1 Rationale and setup

To get an understanding of the player experience introduced by head tracking and the resulting 3D experience, we have setup an experiment which involves the use of head tracking as well as hand tracking. To view the scene the user moves around and the scene is updated as usual. To interact with the scene the position of the user's hands are used and in their place a pair of transparent yellow balls are positioned. These balls have been made transparent because they otherwise obstruct what the user is trying to reach for. In the experiment tennis balls are sent flying towards the user, one at a time. The tennis balls can collide with the walls of the room and the balls representing the user's hands. The tennis balls are also affected by gravity to create a realistic feel to the scene. The challenge set for the user is to stop the tennis balls before they move past them as illustrated on Figure 39. After 20 balls, data about hit rate was logged and the user was asked for a subjective opinion of the experience. This setup allowed us to get some data of success rate, learning curves as well as general player experience in a system controlled by head tracking as opposed to the same system with no head tracking. To illustrate the results we have plotted the accumulated success rate of each player with a different color as well as an accumulated average hit success.

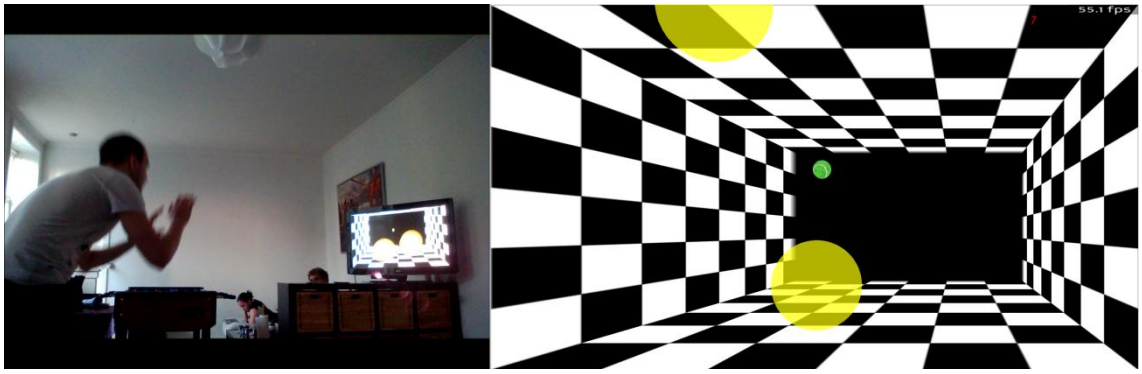


Figure 39: Left picture shows user using racket and right image shows how it appears on screen

3.10.2 Expected results

Because head tracking is unfamiliar to the user's on which the experiment was tested, we expect to have a lower success rate while using head tracking than with no head tracking. We expect to see a learning curve in both experiments but this would be most significant while using head tracking. A learning curve would be visible as a curve which is initially rather flat but gets a steeper angle the further the experiment goes. Another factor we will likely see is the difference in skill level and motor coordination. Furthermore different levels of motivation will have an impact on the results of the test which will further cloud the results. As such we will not be looking into each player's performance but instead look at the average user's performance.

3.10.3 Results and discussion

The results of the experiment are shown on Figure 40

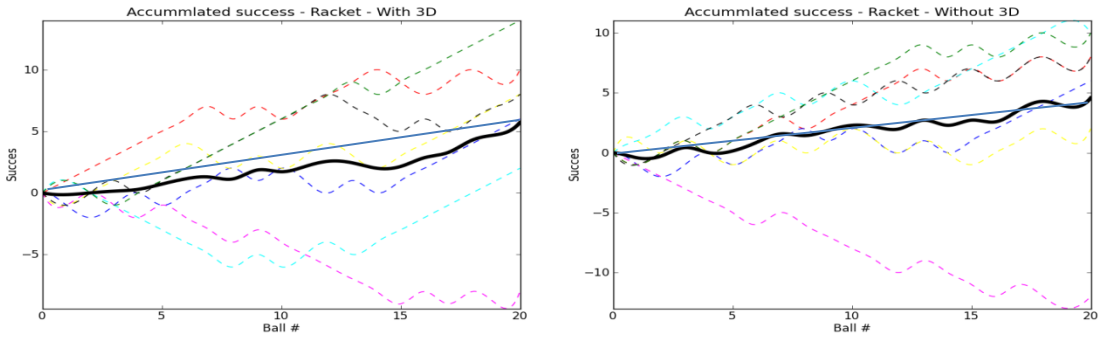


Figure 40: Charts showing accumulated hit rate of the racket experiments with and without 3D for all the subjects. Black line shows the average.

From the results we see that there is a wide variation in the skill level of the users who participated in the experiment. Some of the users were able to hit most of the tennis balls and some of the users barely hit any. By comparing the two charts we see what might be the beginning of a learning curve for the experiment done while using head tracking. This is apparent as the blue line drawn from the start of the average to the end point of the average is above all of the average points in between. This is not the case in the chart showing success rate for the experiment with no head tracking. It would be interesting to get some data from a longer session as this could determine if there was in fact a steeper learning curve while using head tracking.

The subjective feedback on the test was very positive and in some cases the users even wanted to keep on going. While most of the users seemed to enjoy the general setup of the challenge, the positive feedback was primarily weighted towards the experiment while using head tracking. This feedback was surprising as we had expected some frustration to follow from the increased difficulty introduced by having to control the camera by moving the head.

4 Conclusion

Our experiments show that the users have a generally better perception of depth while using head tracking. While using head tracking in our environment users showed an improvement of up to 31 percent when judging relative sizes compared to when not using head tracking. When comparing relative depths of objects, the same setup provided a 28 percent higher success rate which resulted in a 96.5 percent score. When judging the speed of objects while using head tracking the users showed a slight decline in success. Up to 2.4 percent more answers were incorrect which due to the lack of test subjects is well within the margin of error. Our results also show that the increased number of correct answers came with little to

no extra time needed to answer. In some cases the test subjects were even able to answer quicker than with no head tracking. We suspect that this is because the parallax cue was so explicit that it provided an instant depth cue. When the distance between objects was small the users needed more time to answer when using head tracking. This would be a natural response to challenging depth estimation in the real world as well.

While performing the experiments using head tracking, test subjects applied depth perception strategies that would also work in the real world. These included viewing the scene from sharp angles, as well as head bobbing and general movement to achieve a parallax cue. When asked to judge exocentric distances in the virtual environment the users showed similar results as would be expected in the real world which in general is an overestimation. In comparison, when not using head tracking the users produced random results.

By researching player experience we reached a conclusion that there currently is no exact framework for measuring player experience. Instead the player experience is defined by all contributing benefactors. This infers that by adding a positive factor to the list of contributing factors, the overall experience is improved. Users seemed to enjoy interacting with the virtual environment. Especially in the experiment where the users deflected incoming balls using their hands, they expressed positive emotion and gave encouraging feedback on the experience of using head tracking. This indicates that the introduced head tracked 3D has a positive impact on player experience.

While the users generally seemed satisfied with the reflected movement of the head, several of the users expressed dissatisfaction with the lag and low precision when using their hands. This could potentially lead to frustration and thus have a negative impact on player experience. By testing the system we have measured the amount of lag to be approximately 300ms. This is also the limit where the users begin to disassociate themselves from the virtual environment. As it has been shown that the perception of the scene has been improved, we can conclude that while the technology is not yet mature enough to be applied to reaction dependent applications or games, it does provide a satisfying way of viewing a virtual environment. The choice of using a layered software architecture makes the technology easily applicable to other applications.

In comparison to other current 3D technologies, head tracked 3D has a number of advantages. The technology does not require any expensive specialized equipment and can be used on any ordinary screen. Furthermore the 3D effect does not require two simultaneous images to be shown which means it uses less resources. This allows even low spec PCs to keep a stable frame rate at high resolutions. On top of this the technology does not require any carried equipment and allows for a broad viewing angle. The application we created requires a lot of movement to utilize the full viewing angle. This can however be circumvented by using the near mode in the Microsoft SDK. Because humans have stereoscopic vision the full 3D effect is not as visible when observing the environment with

both eyes open. The 3D effect is very explicit when the scene is observed through a camera or in a video. We thought the same experience would be achieved by closing one eye, but the feedback from the users show that only some of them experience the 3D effect. This indicates that the brain is not easily fooled by the illusion of depth we created. A way to fix this issue would be to implement stereoscopic viewing through glasses and by introducing cadre viewing.

5 References

Bianchi-Berthouze, Nadia. *Understanding the role of body movement in player engangement.* London : University College London.

Cadle, James and Yeates, Donald. *Project Management for Information Systems.* 4.

Hürst, Wolfgang. 2011. Youtube. *Computer Graphics 2011, Lect. 7(1) - Perspective projection.* [Online] 2011. http://www.youtube.com/watch?v=Xt-AkLPC3Iw.

Kooima, Robert. 2009. *Generalized Perspective Projection.* 2009.

Lee, Johnny Chung. Youtube. *Head Tracking for Desktop VR Displays using the WiiRemote .* [Online] http://www.youtube.com/watch?v=Jd3-eiid-Uw.

MacKenzie, Scott I and Ware, Colin. 93. Laq as a Determinant of Human Performance. *InterChi.* 93.

Mulder, Jurriaan D and van Liere, Robert. *Enhancing Fish T ank VR.* Amsterdam : Center for Mathematics and Computer Science CWI.

Nacke, Lennart and Drachen, Anders. *Towards a Framework of Player Experience Research.* s.l. : University of Saskatchewan, Aalborg University.

Rolland, Jannick P, Gibson, William and Ariefy, Dan. *Towards Quantifying Depth and Size Perception in Virtual Environments.*

Shneiderman, Ben. *Response Time and Display Rate in Human Performance with Computers .* s.l. : Department of Computer Science, University of Maryland.

Steinbach, Eckehard, et al. 2011. *ADVANCES IN MEDIA TECHNOLOGY.* s.l. : Technische Universität München, 2011.

Waller, David. *Factors Affecting the Perception of Interobject Distances in Virtual Environments.* s.l. : University of Washington.

Webb, Jarrett and Ashley, James. *Beginning Kinect Programmg with the Microsoft Kinect SDK.* s.l. : Microsoft.

Wloka, Matthias M. *Lag in Multiprocessor Virtual Reality.* s.l. : Brown University.

Woods, Andrew J and Stanley, S. L. Tan. 2002. Characterising Sources of Ghostingin Time-Sequential Stereoscopic Video Displays. *Stereoscopic Displays and Virtual Reality Systems IX.* 2002.

Yongwan, Kim, et al. 2011. Analysis on Virtual Interaction-induced Fatigue and Difficulty in Manipulation for Interactive 3D Gaming Console. *IEEE.* 2011.

5.1 Figure references

This shows references to the source of figures not created by the authors of this report.

Figure 2 - (Waller)

Figure 3 - http://www.arborsci.com/cool/polarization

Figure 4 - http://www.thereformedbroker.com/2011/01/30/the-next-must-have-gadget-is-from-nintendo-not-apple/

Figure 5 - http://neilnathanson.com/3D02/lenticular.html

Figure 15 - http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=linux&db=bks&fname=/SGI_EndUser/MPK_UG/ch03.html

Figure 21 - (Kooima, 2009)

Figure 22 - (Kooima, 2009)

Figure 23 - (Kooima, 2009)

Figure 24 - (Kooima, 2009)

Figure 25 - (Kooima, 2009)

Figure 26 - (Kooima, 2009)

Figure 27 - http://johnnylee.net/projects/wii/