Bachelor thesis

Independent degree project - first cycle

Datateknik
Computer Science

Simulations in 3D research
Can Unity3D be used to simulate a 3D display system?

Oskar Andersson

MID SWEDEN UNIVERSITY
Department of Information and Communication Systems

Examiner: Ulf Jennehag, ulf.jennehag@miun.se
Supervisor: Roger Olsson, roger.olsson@miun.se
Author: Oskar Andersson, osan1301@student.miun.se
Degree programme: Computer Engineering, 180 credits
Main field of study: Computer Science
Semester, year: VT, 2016

Abstract

Mid Sweden University is currently researching how to capture more of a scene with a camera and how to create 3D images that does not require extra equipment for the viewer. In the process of this research they have started looking into simulating some of the tests that they wish to conduct. The goal of this project is to research whether the 3D graphics engine Unity3D could be used to simulate these tests, and to what degree. To test this a simulation was designed and implemented. The simulation used a split display system where each camera is directly connected to a part of the screen and using the position of the viewer the correct part of the camera feed is shown. Some literary studies were also done into how current 3D technology works. The simulation was successfully implemented and shows that simple simulation can be done in Unity3D, however, some problems were encountered in the process. The conclusion of the project show that there is much work left before simulation is viable but that there is potential in the technology and that the research team should continue to investigate it.

Keywords: 3D, simulation, graphics engine, virtual reality, autostereoscopic.

Acknowledgements

I would like to thank my family for the support they have given me during these three years and my awesome classmates for keeping it fun.

Table of Contents

Terminology

Texture	An image or bitmap used to give details or color to a 3D object in computer 3D graphics.
UV mapping	The method used to determine what part of a texture to show on a 3D object.
Simulation	The recreation of a real world process or system over time. This is done by developing a model to represent the system and seeing how it works over time.
Parallax	"The apparent displacement of an observed object due to a change in the position of the observer."[1]
Stereo Parallax	Seeing a different image with each eye.
Movement Parallax	Seeing different images when we move our head.
Aperture	A hole or opening that light can pass through.
Pseudoscopic Image	An image with a reverse field of depth.
FPS	Frames Per Second, the number of frames shown each second.

1 Introduction

Chapter 1.1-1.7 gives an introduction to the project, its background, aim, scope and goals. The ethical implications are also discussed here.

1.1 Background and problem motivation

Since the invention of the Camera Obscura [2] the technology of image capturing has made great advances. The quality of the image has been improved, color has been added and in more recent years the 2D images has been transformed into 3D images.

However a large problem still remains, the images does not capture the entire scene. For example a picture of a cube will only capture at most three sides of the cube and if anything is behind the cube it will also not be captured.

To solve this the research team at Mid Sweden University is developing a new way of capturing 3D images that shows more of the scene. By displaying this on a custom made display the viewer can see other angles of the scene by moving around in the room, creating the feeling of looking through a window instead of looking at a TV screen.

The technology requires expensive equipment and resources that the team does not have. This has lead the team to look at other alternatives for testing and one of the alternatives is to create a simulation in a 3D engine. This would allow them to test different setups with minimal resources and work.

1.2 Overall aim

The aim of this project is to research whether a 3D engine can be used to test different setups for the research teams and whether this is worth investing in. This will be done by looking at different ways of simulating and implementing one of these in a 3D engine. The 3D technology to be simulated will be researched as well.

To implement the simulation first a test environment will have to be designed. Once the environment is designed a way of simulating the technology will be tested to see whether the simulation can implement

the rendering technique. Virtual Reality will also be implemented using Oculus Rift.

There may be other solutions for the implementations and a part of the project will be to look for alternative solutions as well and try to implement them.

1.3 Concrete and verifiable goals

The first goal will be to evaluate whether Unity3D can be used as a simulation framework. This will be done by implementing a simulation the research team has requested. This will be done by designing and implementing a test environment in Unity3D. The environment will use a display technique where the display is split into many parts and each part is directly connected to a camera and will change the image depending on the user's position. A virtual reality headset will also be implemented so the user can see the result in 3D, the headset used will be an Oculus Rift DK2

The second goal will be to do research into the technology the research team has done. Since this technology will be simulated this will be important for designing the simulations and evaluating if the technology can be used for further simulations.

The final step will be to evaluate the results of the research and the implemented simulation to determine if simulations is a worthwhile investment for the research team.

1.4 Scope

The project will use the Unity3D engine only. The simulations to be implemented is the one requested by the research team using a divided display that will use a split display where each camera is directly connected to a part of the display. The virtual reality headset used is an Oculus Rift DK2.

1.5 Ethical considerations

If the research team can switch to using computer simulations instead of having to build all the prototypes in real life they not only save money by not having to buy equipment but are also environment friendly by not having to use as many resources, less transport cost for transporting the parts and lower energy consumption.

This could also lower the need for transport costs and CO2 emissions if they ever need to collaborate with researchers from another city or country since they will not have to transport a prototype. It also opens new possibilities for the team to work with other researchers or amateurs to complete the research faster or get new input and ideas since the work can be easily shared through the internet.

There is also considerations that must be made if everyone can use this technology. Some people report feelings of nausea and headaches when watching 3D images meaning the technology is not usable by them. There is also a group of people that has lost sight in one eye that cannot use this technology. It may however help people that are cross-eyed to see 3D if the display system can be adapted to compensate for them so that the correct image reaches the correct eye.

1.6 Outline

The introduction chapter will provide an introduction to the background and reason this project is being done. It will also outline the scope and goals of the project. In the theory chapter theoretical information about different available 3D technologies and how they work will be presented. Information about virtual reality and the 3D engines to be compared are also given here. Methodology outlines the methods to be used in the project to achieve the goals set in Chapter 1.4. Implementation explains how the goals have been implemented and goes into further detail of how they were made. Results shows the results of the implemented simulations, the results for the research can be seen in Chapter 2. Finally the discussion chapter discusses the information gathered through research and the experience gained from implementation to judge if simulation is a good investment for the research team.

1.7 Contributions

The research done for the theoretical part of the project as well as the implementation of the simulations is the work of Oskar Andersson. Feedback, resources for the research and the Oculus Rift was provided by Roger Olsson from Mid Sweden University. Picture 4.1 and Picture 4.2 were created by Matilda Andersson.

2 Theory

Chapters 2.1-2.3 presents the theoretical details of the project.

2.1 Unity3D

Unity3D is a game engine made by Unity Technologies. It is one of the most used game engines in the world with a 45% market share [3]. The game engine can be used to create 2D and 3D games and supports the C# and JavaScript languages. It also has an asset store where 3D models, textures and other things can be purchased or downloaded for free. Unity supports plugins for different functions and virtual reality headsets.

2.2 Virtual reality

Virtual Reality is a computer generated environment that the user perceives as "real". The NASA Advanced Supercomputing definition of virtual reality is this "Virtual reality is the use of computer technology to create the effect of an interactive three-dimensional world in which the objects have a sense of spatial presence." [4]. This reality is usually seen through a Virtual Reality Headset [5] that uses two screens, one for each eye, to render different perspectives for each eye. Rendering a separate perspective for each eye gives the user stereo parallax and a 3D view of the scene they are looking at.

The Oculus Rift is a Virtual Reality headset created by Oculus VR. The headset started as a kickstarter on the first of august in 2012 [5]. The kickstarter created a new wave of interest in Virtual Reality and was hugely successful and when the kickstarter was over a total of $2,437,429 was pledged far surpassing the initial goal of $250,000 [6]. The first commercial version was released on the 28th of march 2016 [7].

2.3 3D display technology

Details and explanations for the most common techniques used to create 3D images are explained here.

2.3.1 Stereoscopic displays

Stereoscopic displays[8] are displays that can show images in 3D. This works by having some kind of extra equipment that enables the 3D

technology, this can be things like different types of glasses or other headgear. Three of the most common methods for stereoscopic displays are anaglyphic 3D, polarization multiplexing and time multiplexing.

The Anaglyph 3D method is an old method of creating a 3D image dating back to the mid 1800s. This method uses a color coded image where each eye is assigned a color, usually red for the left eye and cyan for the right eye. The viewer then uses a pair of glasses where each lens is colored to only allow the eye to see the assigned color. This makes each eye see a different image causing a 3D effect. This method has two major drawbacks. The first is a loss of color, since the image has to be color coded. The second problem is cross talk where an eye might see parts of the other eye's image. There are however modern uses for this method and it is supported by popular graphics drivers like Nvidia and iZ3D. [9]

In polarization multiplexing two projectors are used, each projector shows the image to one eye and the image has a polarization state orthogonal to the other projectors image. This is then projected onto a specially made screen that is able to preserve the polarisation. The viewer wears polarised glasses where each lens is made to block the image from the projector not meant for it. This ensures that each eye only sees the image from the projector designated to it.[9]

Time-Multiplexing works by taking advantage of the human brain's persistence of vision to create a 3D effect. This is done by using a high frequency display to quickly alternate between the images for the left and right eye. This is coupled with a pair of glasses that the user wears that synchronises with the screen to only show the image to the eye that is supposed to see the image and blocking the other eye. [9]

2.3.2 Autostereoscopic two-view displays

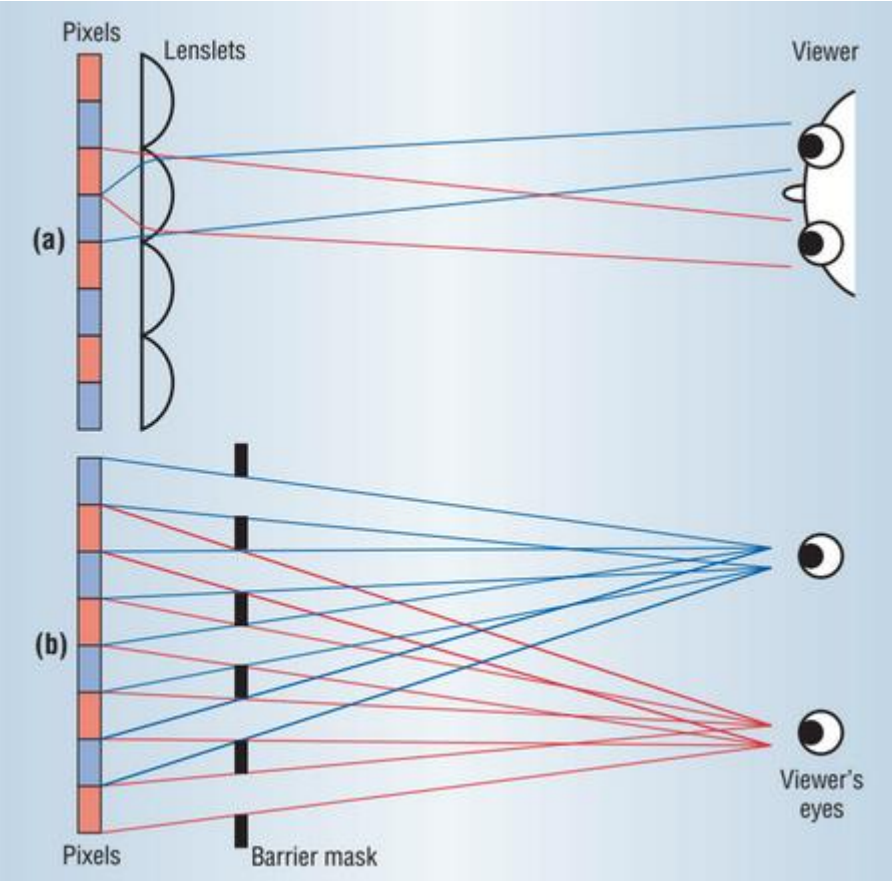
Two-view displays is a type of autostereoscopic display, meaning it does not use any extra equipment to show 3D images. Instead if works by showing two views, one for each eye, to a specific place and as long as the viewer is in the correct place they will see a 3D image. Two-view displays have been around for over a century using either parallax barrier or lenticular sheet technology. This works by having a display in the background that is divided into two sets, each set contains every other pixel and is assigned to one eye each. By putting a lenticular sheet

or a parallax barrier in front of the display the viewer can see a 3D image if they are in the right position and distance from the screen. [8]

Lenticular sheets are made of multiple lenses that are designed to bend the light to target a specific point in the viewing zone. This sheet is then placed on top of the display that has been divided into right and left eye sets so that the light from each set is targeted at the correct eye. This can be seen in Picture 1 example (a) [9]

The parallax barrier works by placing vertical apertures separated by black masks. The barrier is spaced out so the pixel sets light is made to pass to the viewing zone to the correct eye. This technology can also be used to repeat the viewing zone making a multiview display. The biggest problems with this technology is the loss of brightness and loss of spatial resolution since only half the pixels are used for each viewing zone. An example of this can be seen in Picture 1 example (b). [9]

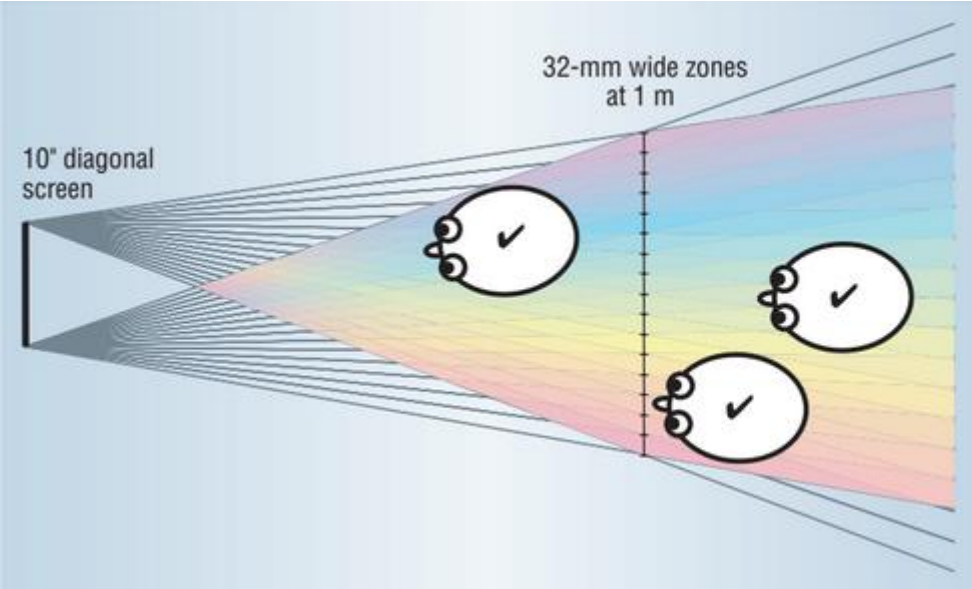
A problem with both methods is that the viewer has to be in the correct position in order to see the correct image. If the viewer is in the wrong position the image meant for the right eye goes to the left eye and vice versa causing a pseudoscopic image. The number of incorrect positions is 50% which is also a problem that needs to be addressed. [9]



Picture 1. (a) shows lenticular sheets placed over a display bending light to hit the correct eye. (b) shows a parallax barrier placed over a display blocking the view so each eye sees the correct image. [8]

2.3.3 Autostereoscopic multiview displays

A multiview display creates multiple stereo pairs that are projected over the entire viewing zone. This allows for multiple people to view images in 3D from the same display. It also allows for movement parallax where viewers can move sideways and see different angles of the scene, however, it normally has too few views to achieve continuous moving parallax but there are techniques to improve this. Picture 2 shows a 16 view autostereo multiview display that allows multiple users to see a 3D image if they are within the viewing zone. One way to achieve multiview displays is to combine time multiplex technology with parallax barriers. Another way is to combine lenticular arrays with pixelated emissive displays. [9]

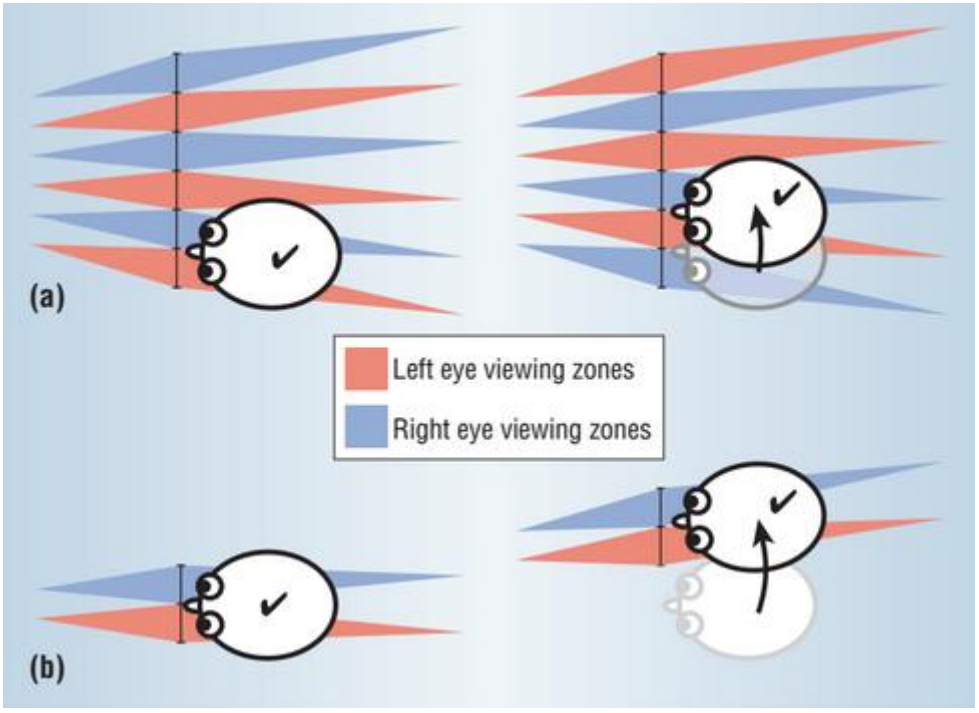


Picture 2. A 16 view autostereo display. As long as the viewers are in the viewing zone they will see a 3D image. [8]

2.3.4 Head-tracked displays

Head-Tracked displays keep track of where the viewer's head is and uses this to adapt the view. This is done by moving the viewing zones to match the viewer's position so that each eye always receives the correct image, this can be seen in Picture 3 (a). This prevents any pseudoscopic images from appearing. Another way is to only project two zones and physically move the zones or display to match the viewer as can be seen in Picture 3 (b). [8]

The problems with head-tracking displays is that they can not require the viewer to wear any extra equipment since this would defeat the purpose of replacing the glasses, the tracking also needs to be fast enough so the viewer does not perceive any delay in the tracking. Physically moving the zones can also become a problem if the solution is not fast enough to keep up with the viewer's movements, it also needs to be robust. The head-tracking display only works for a single user which may be a problem for certain applications. [8]



Picture 3. (a) shows the viewing zones being moved to match the user's head. (b) shows two viewing zones being moved to follow the user's head. [8]

3 Methodology

Chapters 3.1-3.5 shows the methods used to achieve the goals of the project.

3.1 Unity3D as a simulation framework

Unity3D was used as the simulation framework mainly at the request of the research team but also because it supports many of the features needed to implement the simulations. For example being able to connect a camera to a texture, this is needed so that a camera feed can be shown in the engine. It also supports the use of multiple cameras which is needed since the project requires configurations with multiple cameras at the same time. It also support technology that could be used in future work like custom shaders and normal textures.

3.2 Researching 3D technology

To understand the technology that is to be simulated research was needed. This was done by reading articles that was recommended by the research team, source 10 and 11, and writing a compilation of this information. This compilation can be found in chapter 2.3.1-2.5.4.

3.3 Designing test environment

The simulation works by using a split screen where each segment of the screen is directly connected to a camera, a script then changes what part of the camera feed is shown on the segment depending on the position of the viewer. This is the base for the test environment design.

The test environment needs to contain four things: A camera system that contains all the cameras, a display system to show the camera systems feed, a viewer avatar that can move around in the 3D space and virtual rooms that contains all these things. With these things in mind and the time limit of the project a simple virtual room will be created. The scene to be viewed will be a room that contains a cube and an orb. These are placed in a line with the cube closest to the wall where the cameras are placed, this is done so that the user will be able to move and change the perspective of the display so they can look behind the cube and see the orb. This room is hereafter referred to as the Scene Room and also contains the camera system. The viewer avatar will be placed in a

second room, hereafter referred to as the Viewer Room, that also contains the display system.

The placement of the cameras will be done in a straight line with even distance between them. The difference in each room will be how many cameras there are in each room and this will also determine the distance between each camera. This is to test different camera setups and the impact the number of cameras have on the quality of the final product.

The first room will have one camera, this is simply to show how the scene looks using a single camera. The second room will have two cameras to show that the display system can show the view from two cameras at once. The third room will use 10 cameras to show what an increase in cameras will do to the quality of the final image. 27 cameras will be used because this allows for placing cameras over the entire wall with no space between them, eliminating "dead zones" that no camera can see. 50 cameras will be used to show the result of the cameras overlapping each other. 351 cameras will be used to show the result of having both vertical and horizontal movement parallax, to achieve this the cameras are placed in a grid that is 27 camera's wide and 13 cameras high.

3.4 Alternative solution

An alternative solution to eliminate "dead zones" is to change the placement of the cameras. If they are placed in a semi-circle facing inwards they should cover more of the scene with less cameras being used.

3.5 Adding virtual reality support

Virtual reality will be added so that the user can view the 3D effect. With only one screen to view the result will be in 2D but with the dual screens in the oculus rift the display gains a 3D effect. It also gives the user a deeper immersion and a better sense of how the real product would look.

3.6 Performance requirements

For the simulation to be viable certain performance requirements must be met. The first is that it needs to perform above a certain frames per second, for this project a minimum of 24 frames per second will be

required. This was chosen as the minimum because it is the standard frame rate for movie theaters [10].

The quality of the final image must also be considered but is harder to measure objectively. However a few basics can be aimed for:

- The orb and cube must be visible.
- The seams between display parts must be as unnoticeable as possible.
- The movement parallax should be as smooth as possible.

These are "soft" requirements since they are not measurable but provides a guideline for judging the final image and the quality of the image.

3.7 Simulation hardware

The simulation was implemented on a computer with the following hardware:

Processor: Intel Core i5-4690K, 3.50GHz

Graphics Card: Nvidia GeForce GTX 970

HDD: Samsung SSD 840 EVO 250GB

Motherboard: Asus Z97 Pro Gamer

Memory: Corsair Vengeance Pro DDR3 2400MHz 16GB

4 Implementation

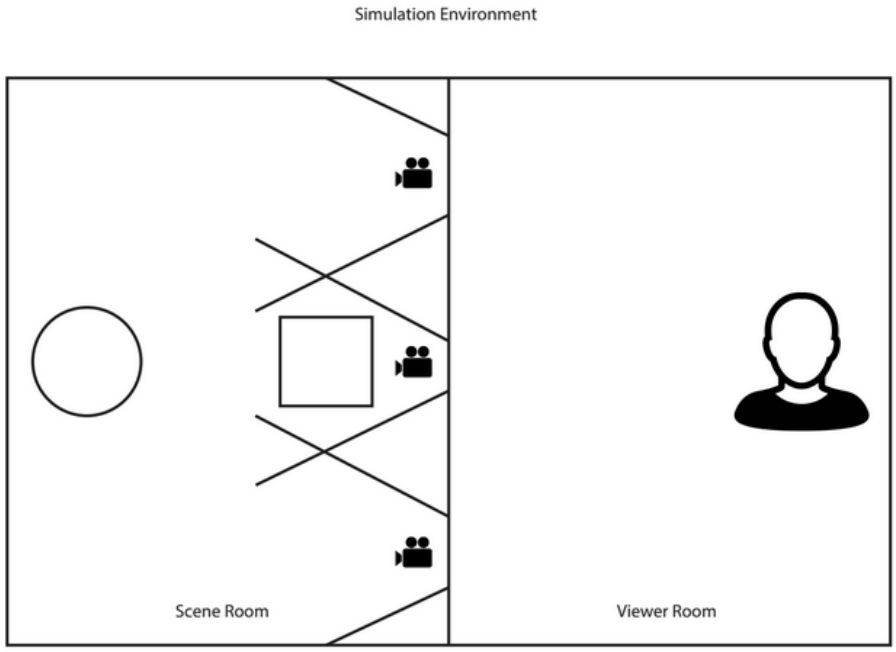
Chapters 4.1-4.5 shows how the project was implemented.

4.1 Making the rooms

To create the rooms in Unity3D five 3D objects called “Planes” were used as walls and floor, planes are flat square surfaces that can have a texture attached. The planes are aligned to create a room and textures are added to distinguish what is floor and what is a wall. Another five planes are added to create the second room next to the first.

The viewer avatar will be a simple cylinder with a camera placed on top. The cylinder will have a script attached to it to enable movement. The avatar will be placed in in the viewing room. The Scene room will contain the cube and the orb, these will be placed in a straight line facing the camera system. A sketch of the design can be seen in Picture 4.

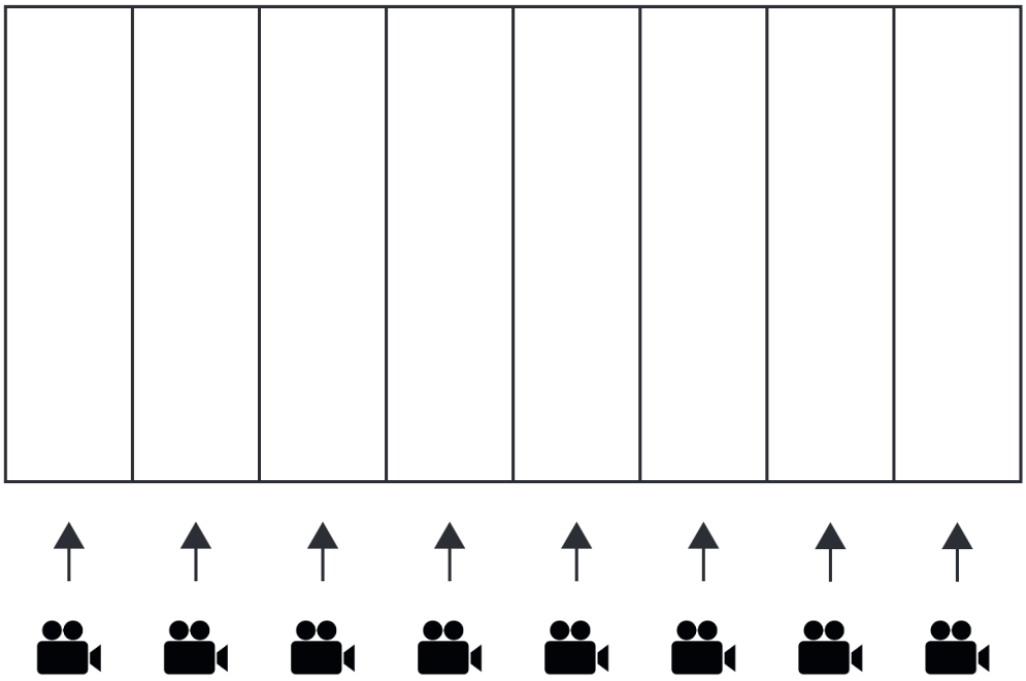
Once the rooms are completed the cameras are added. They are placed in a horizontal line at center height of the wall that connects the two rooms. Each camera system will have a different number of cameras as discussed in chapter 3.3.



Picture 4. Sketch of a room with 3 cameras.

4.2 Creating the display system

To create the divided display planes are once again used. Each rooms has as many planes as it has cameras and the planes are equally sized so that the combined size is the same as one wall sized plane, this is the display system itself. They are then placed back-to-back with the wall the cameras are connected to and a render texture is applied to each part. Using the "Target Texture" option on each camera a render texture can be selected to display the camera feed, this is used to connect each part of the display to a camera. This means camera one is connected to plane one, camera two to plane two and so on. Since each part of the display shows a different view this creates a stereo parallax giving the viewer a 3D experience. A sketch of this can be seen in Picture 5.



Picture 5. Sketch showing the basics of a divided display. Each camera is connected to a part of the display.

4.3 Alternative camera system

Instead of placing the cameras in a straight line along the wall they are instead placed in a semi-circle around the scene angled inwards. This allows less cameras to cover the same area. This is connected to the same display system as the other camera system.

4.4 Viewer movement

To enable the viewer to move in the Viewer Room a premade C# script from the Unity3D documentation was used[11] that uses the WASD control scheme to move the avatar and the spacebar to jump.

4.5 Texture movement

Once the render texture was created and connected to a camera it was added to a plane that was used as part of the display. When added the texture was resized to fit the plane which distorted the image. To fix this the tiling setting was changed to show 1/X of the render texture where X is the number of cameras used in the room. For instance room 5 has 50 cameras so the tiling setting for the render textures are 1/50.

This makes it so only a part of the render texture is shown at any time and by manipulating what piece is shown the effect of being able to look behind objects is created. To achieve the movement of the texture a script was created that takes the current position of the viewer and the current offset of the texture and from this calculates what the new offset should be.

4.6 Virtual reality

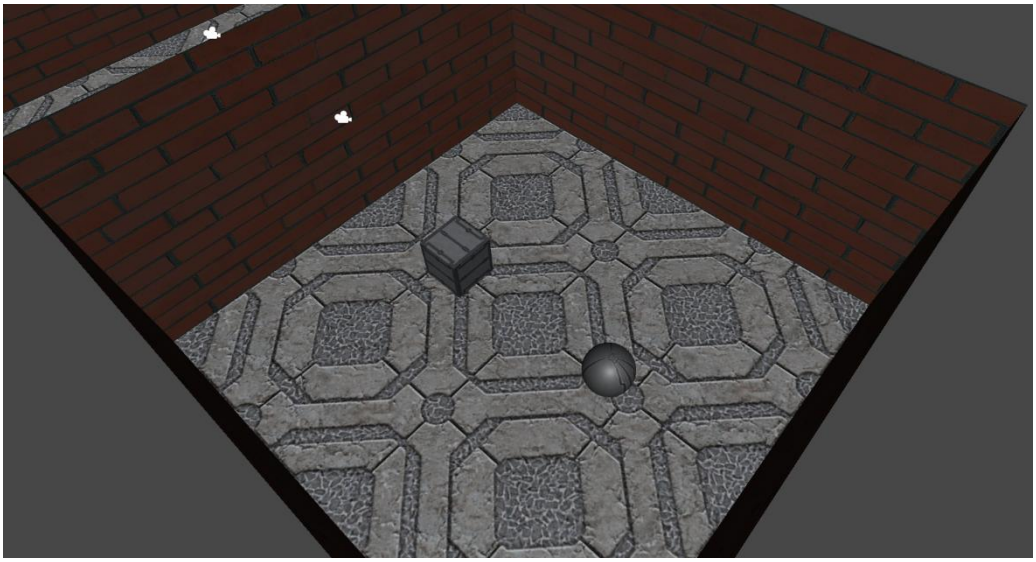
The addition of virtual reality was done by first installing the needed components, Oculus Runtime on Windows 0.8.0.0-beta[12] and the Oculus hardware. Once these were installed virtual reality was enabled in the project in the Edit->Project Settings -> Player -> Other Settings -> Virtual Reality Supported.

5 Results

Chapter 5.1-5.7 show the results of the implemented simulation.

5.1 Scene room

The scene room contains a cube and an orb with a metal texture placed in a line. There are four walls with a brick texture and a floor with a stone pattern texture. The cameras are placed at the center of the wall connected to the viewer room. The room can be seen in Picture 6.



Picture 6. Overview of the scene room

5.2 Viewer room

The viewer room contains one cylinder acting as the viewer's avatar, this has a grey texture and has a camera connected to the top of the cylinder. The movement script is connected to the cylinder. The floor has the same stone pattern texture as the scene room. Three of the walls has brick texture while the fourth wall is the display system. This can be seen in Picture 7.

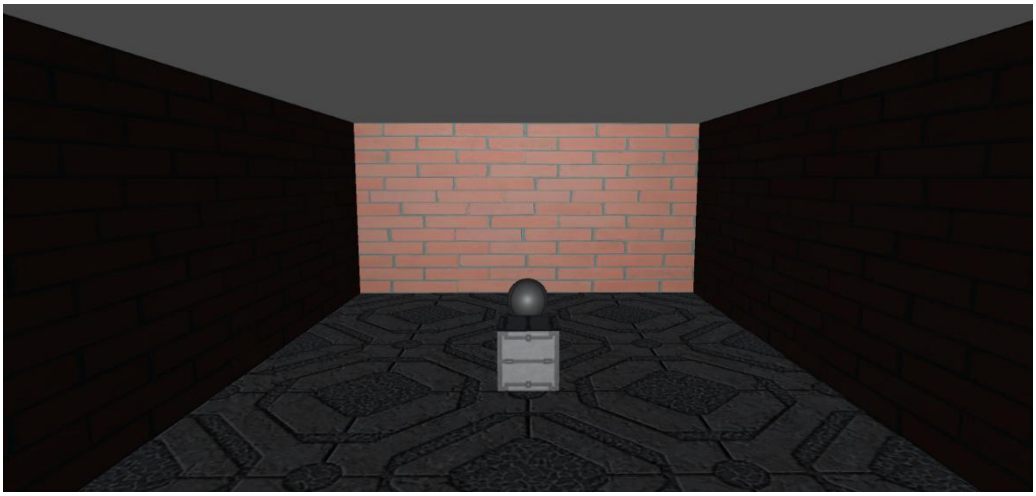
Picture 7. Overview of the viewer room.

5.3 Display systems

Shows the results for the different displays systems.

5.3.1 Original view

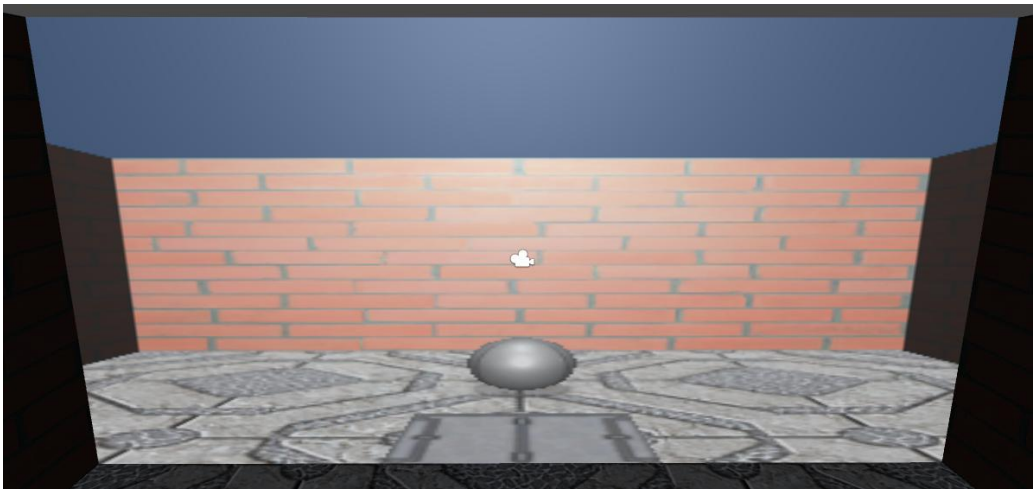
In Picture 8 the original view without a display system can be seen. This is used as a reference for what the display system tries to show.



Picture 8. Original view that the project tries to simulate.

5.3.2 One display

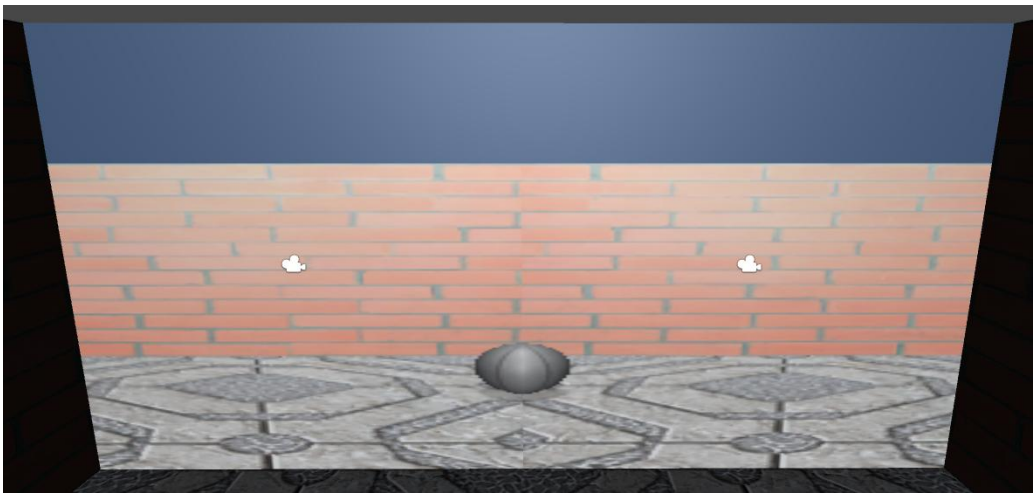
The first display is made of one plane and one texture. The texture does not move since everything that is visible from the camera is already shown. This can be seen in Picture 9. The FPS performance for the simulation remains around 60-70 frames per second.



Picture 9. Display made up of one part.

5.3.3 Two displays

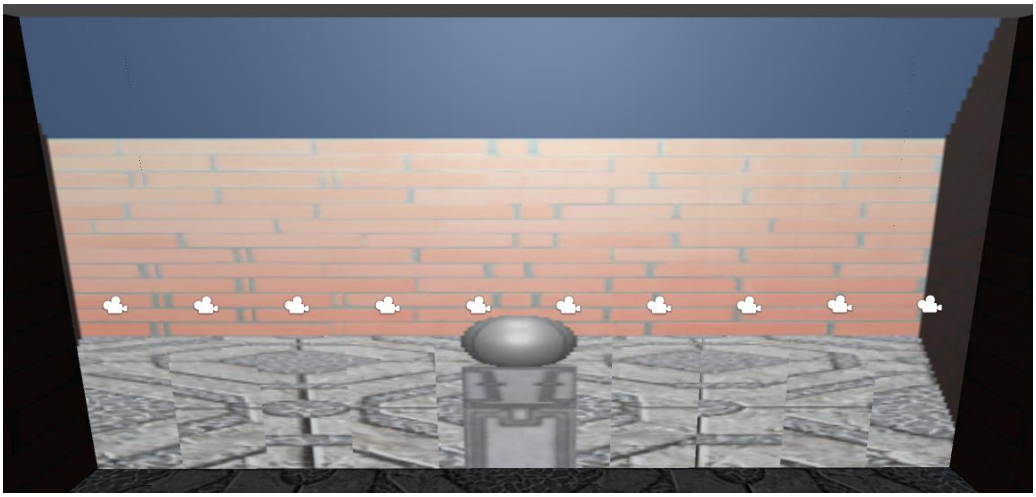
Picture 10 shows a display made up of two parts and two textures. The textures will move to match the movements of the viewer and show the relevant part of the camera feed. This provides limited movement parallax. The FPS performance remains between 60-70 frames per second.



Picture 10. Display made up of two parts.

5.3.4 Ten displays

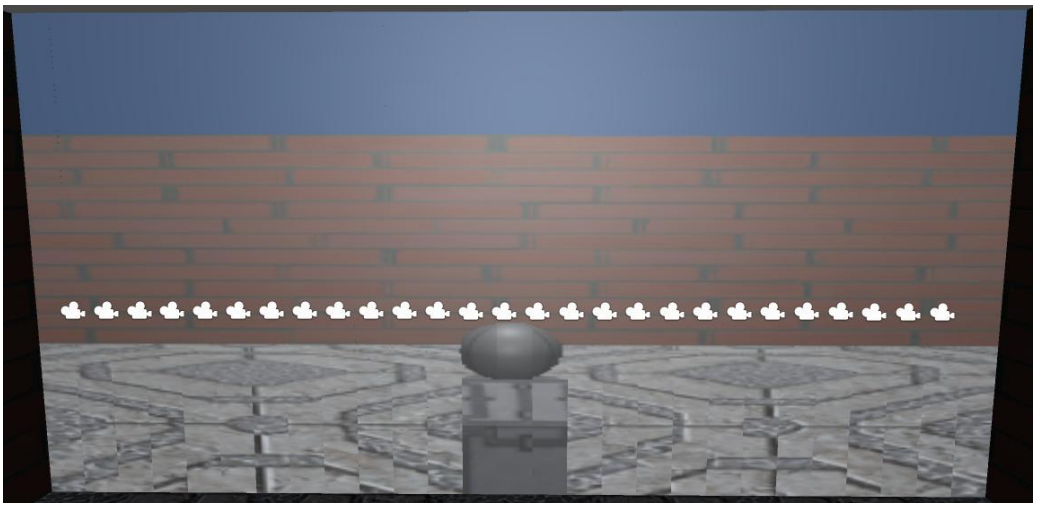
Picture 11 shows a display made up of ten parts and ten textures. The textures will move to match the movements of the viewer and show the relevant part of the camera feed. This provides improved movement parallax compared to the two part display in chapter 5.3.3. The seams between the different parts are visible causing disruption in the movement parallax. The FPS performance remains between 60-70 frames per second but could occasionally drop below 60.



Picture 11. Display made up of ten parts.

5.3.5 27 displays

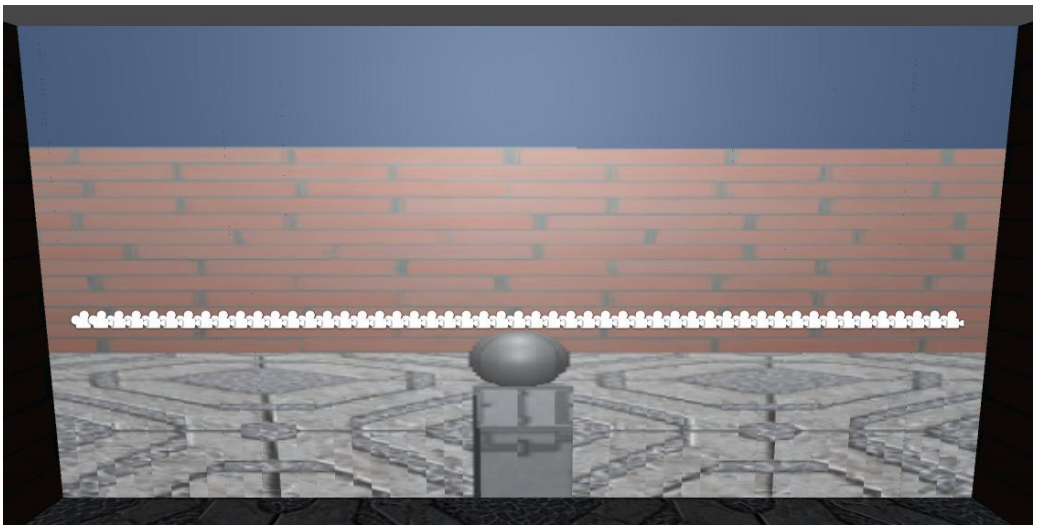
Picture 12 shows a display made up of 27 parts and 27 textures. The textures will move to match the movements of the viewer and show the relevant part of the camera feed. This provides improved movement parallax compared to the previously shown displays. The seams between the different parts are visible causing disruption in the movement parallax but is an improvement compared to the ten part display in chapter 5.3.4. The FPS performance has dropped compared to previous displays and remains between 30-40 frames per second.



Picture 12. Display made up of 27 parts.

5.3.6 50 displays

Picture 13 shows a display made up of 50 parts and 50 textures. The textures will move to match the movements of the viewer and show the relevant part of the camera feed. This provides improved movement parallax compared to the previously shown displays. The seams between the different parts are slightly visible causing minimal disruption in the movement parallax and is an improvement compared to the 27 part display in chapter 5.3.5. The FPS performance is between 15-25 frames per second.



Picture 13. Display made up of 50 parts.

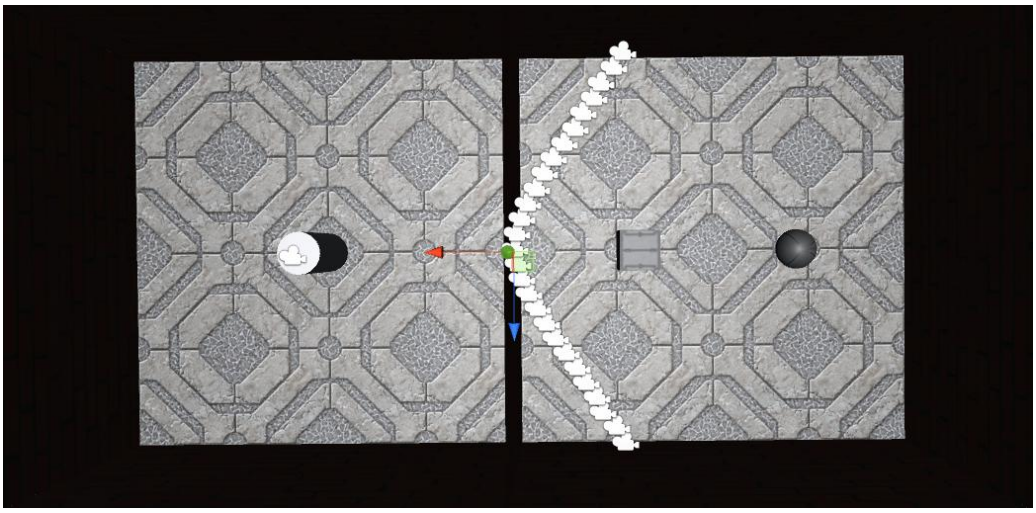
5.3.7 351 displays

Picture 14 shows a display made up of 351 parts and 351 textures. The textures will move to match both the horizontal and vertical movements of the viewer and show the relevant part of the camera feed. This provides movement parallax for both vertical and horizontal movements and makes it possible to not only look behind objects but also over or under them. The seams between the different parts are visible causing disruption in the movement parallax. The FPS performance is 2-4 frames per second.

Picture 14. Display made up of 351 parts.

5.3.8 Alternative camera placement

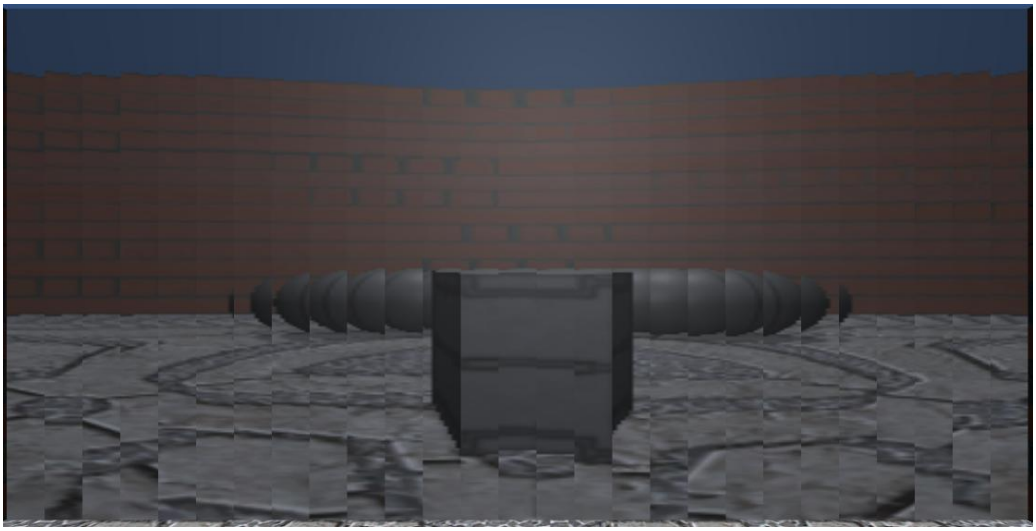
Picture 15 shows the alternative placement of cameras where all the cameras are placed in a semi-circle with a slight angle inwards to cover blind spots.



Picture 15. Placement of cameras in a semi-circle.

5.3.9 Alternative camera system displayed

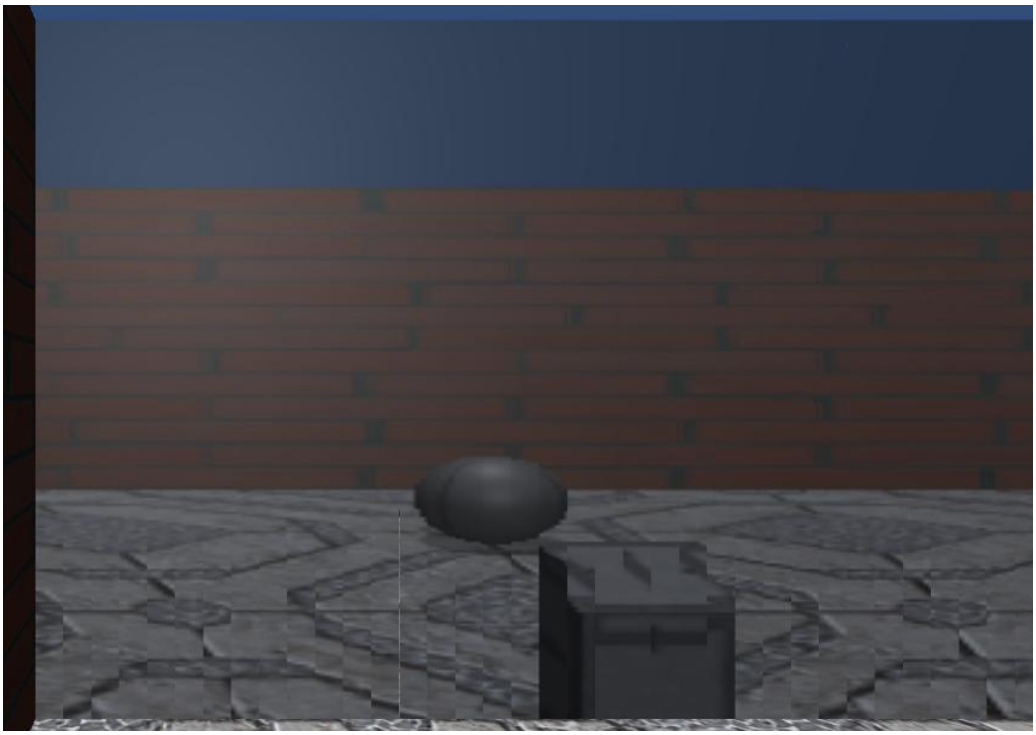
Picture 16 shows how the alternative camera placement displayed on the display system. The alternative solution produces a warped view of the image.



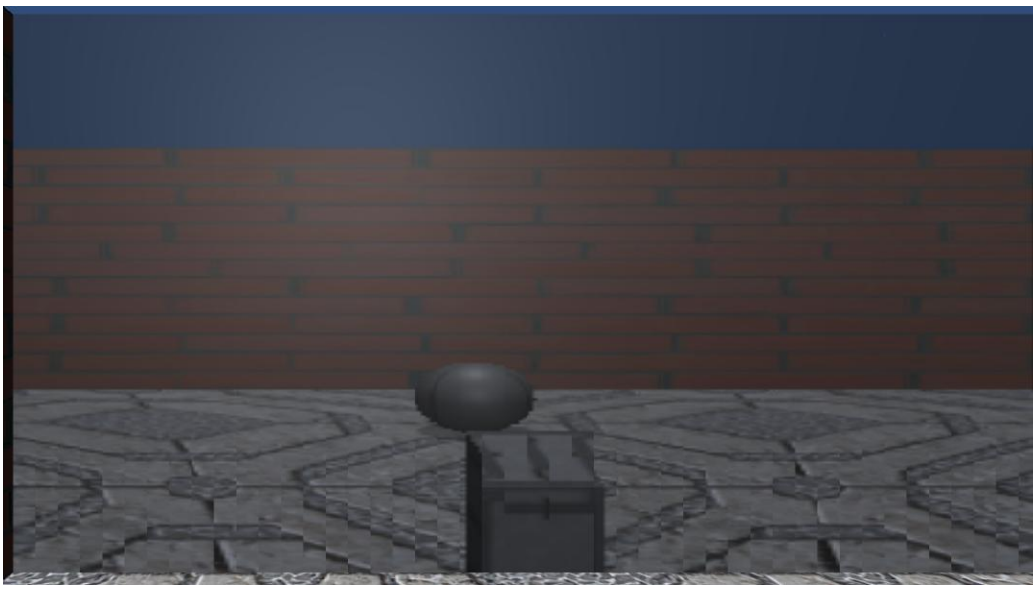
Picture 16. Distortion in the image caused by semi-circle placement of cameras.

5.4 Movement parallax

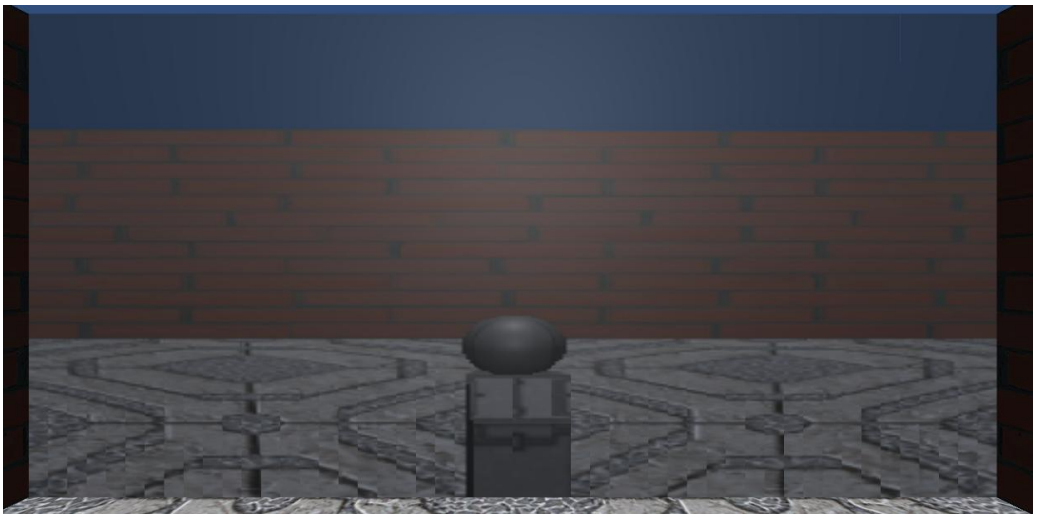
Pictures 17-21 shows the 50 camera display system viewed from different angles to show the movement parallax and how the image changes depending on the viewer's position. The viewer is starting from the far left and moves to the right.



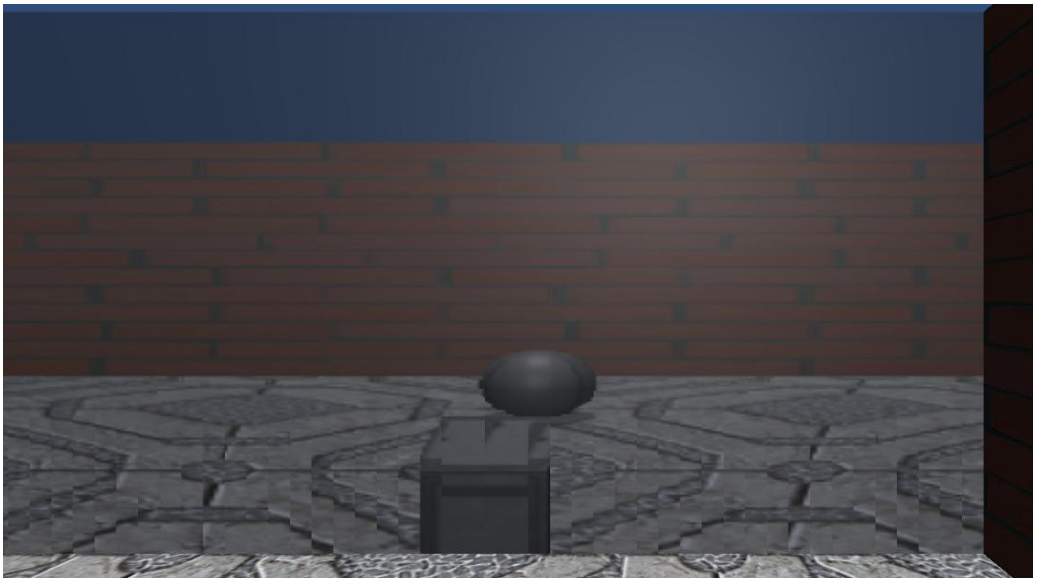
Picture 17. View from the far left.



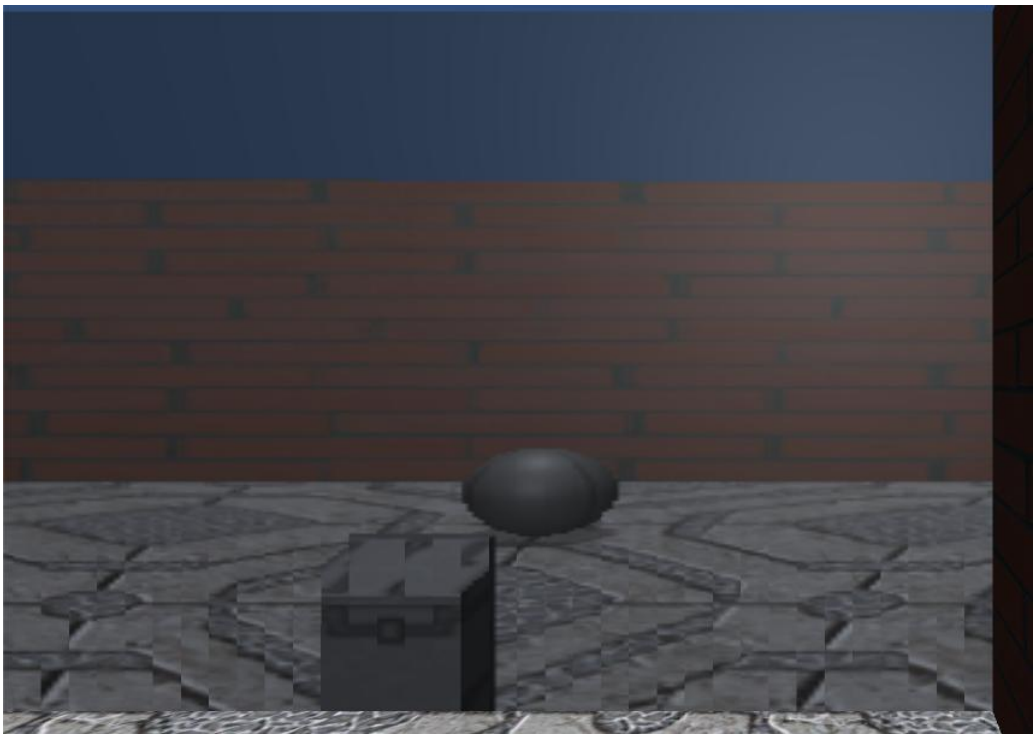
Picture 18. View from the left.



Picture 19. View from the center.



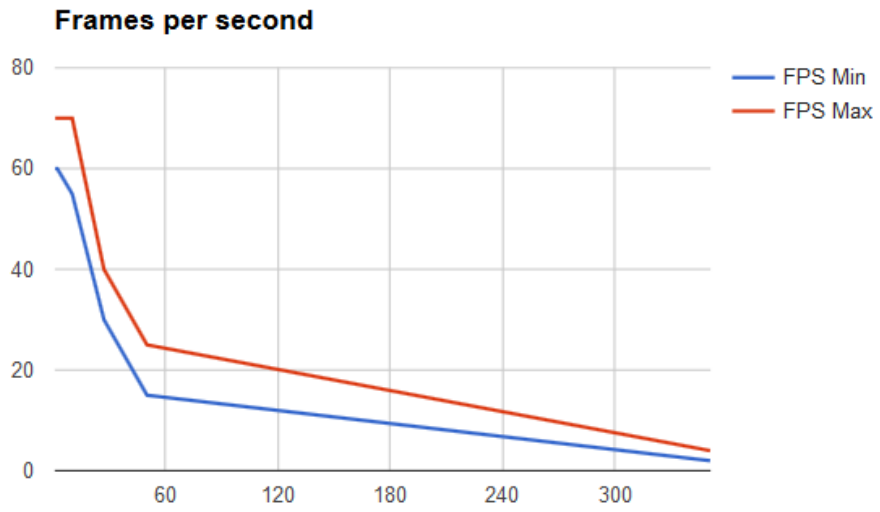
Picture 20. View from the right.



Picture 21. View from the far right.

5.5 Performance

During runtime the frames per second for the different camera system varied and as the number of cameras increased the frames per second dropped and could vary between 2 and 70 frames per second. A graph of the results can be seen in Picture 22.



Picture 22. The number of display parts can be seen on the X axis and the frames per second can be seen on the Y axis.

5.6 Texture movement script

The script that moves the texture based on the viewer's position is written in C# and can be seen in Picture 23. The script has three functions, Start, Update and OnDisable. Start is executed when the script is first called and is only called once. The Start function saves the textures current offset to a variable so it can be used later to reset the values.

The Update function is executed once for each frame and is usually where most of the work is done in a script. Here the parent object's position is saved so it can be subtracted from the player object's position so it starts at zero. This is calculated and saved in the variable z. The speed variable can be set to different values from inside Unity3D and determines how much the texture is offset compared to the viewer's movements, the value used during the tests is 0.05. A new Vector2 is made with the updated position the texture will use and the Vector2 is finally applied to the texture offset.

When the script completes the OnDisable functions is executed to restore the original offset values to the texture to avoid having to reset it manually.

```
using UnityEngine;
using System.Collections;

0 references
public class MovingTexture : MonoBehaviour
{
    private Vector2 savedOffset;
    public float speed = 0;
    public GameObject player;

    // Use this for initialization
    0 references
    void Start()
    {
        savedOffset = GetComponent<Renderer>().sharedMaterial.GetTextureOffset("_MainTex");
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        GameObject parent = this.transform.parent.gameObject;
        Vector3 parentZ = parent.transform.position;
        Vector3 position = player.transform.position;
        float z = (position.z - parentZ.z) * speed * -1;
        Debug.Log(z);
        Vector2 offset = new Vector2(z + savedOffset.x, savedOffset.y);
        GetComponent<Renderer>().sharedMaterial.SetTextureOffset("_MainTex", offset);
    }
    0 references
    void OnDisable()
    {
        GetComponent<Renderer>().sharedMaterial.SetTextureOffset("_MainTex", savedOffset);
    }
}
```

Picture 23. The script managing texture movement.

5.7 Viewer movement script

The script used to control player movements uses predefined values to calculate the new position of the viewer avatar. The script can be seen in Picture 24.

```
using UnityEngine;
using System.Collections;

0 references
public class Moving : MonoBehaviour
{
    public float speed = 6.0F;
    public float jumpSpeed = 8.0F;
    public float gravity = 20.0F;
    private Vector3 moveDirection = Vector3.zero;

    0 references
    void Start() {
        double s = -4.9;
        for (int i = 0; i < 50; i++)
        {
            s += 0.196;
            Debug.Log(s);
        }
    }
    0 references
    void Update() {
        CharacterController controller = GetComponent<CharacterController>();
        if (controller.isGrounded)
        {
            moveDirection = new Vector3(Input.GetAxis("Horizontal"), 0, Input.GetAxis("Vertical"));
            moveDirection = transform.TransformDirection(moveDirection);
            moveDirection *= speed;
            if (Input.GetButton("Jump"))
                moveDirection.y = jumpSpeed;

        }
        moveDirection.y -= gravity * Time.deltaTime;
        controller.Move(moveDirection * Time.deltaTime);
    }
}
```

Picture 24. The script managing viewer movement.

6 Discussion

In chapter 6.1-6.6 the results and research of the project will be discussed and evaluated to reach a conclusion if simulation is useful enough for the research team to invest in.

6.1 Performance

The performance of the implemented simulation varies,as can be seen in Chapter 5.3. It is however apparent from these results that the amount of cameras affect the performance and when the amount of cameras exceed 50 the performance drops below the required 24 frames per second and in the simulation using 351 cameras the performance drops down as low as two frames per second. This is however most likely tied to the hardware used to run the simulation and more powerful hardware could most likely run even more cameras at the same time. It does however point to this method of simulating being inefficient and may need to be replaced with a more effective method that does not require more powerful hardware to run.

6.2 Image quality

The quality of the image being displayed is hard to measure objectively but using the original view in Chapter 5.3.1 and the single display in Chapter 5.3.2 as well as the requirements in Chapter 3.6 a discussion can be had about the quality of the image. For starters there is a large difference in what can be seen in the single display and original view, the camera's field of view does not encompass the entire scene but this can be fixed by simply moving the camera further back. Comparing the single display to the displays using more cameras the first thing that becomes obvious is the seams between each part in the display. In the ten part display shown in Chapter 5.3.4 the seams are quite visible and creates jumps when moving around and the orb and cube sometimes disappears or are cut off. The seams are however less noticeable the more cameras and screens are added. In the display using 27 cameras shown in Chapter 5.3.5 and the display using 50 cameras in Chapter 5.3.6 the seams are much less noticeable and the movements are much less jumpy with less of the orb and cube disappearing.

With all this considered I would say that the display using 50 cameras is the one closest to passing the requirements in Chapter 3.6 however there is still improvements that could be made. The seams are still noticeable in all the displays and the movement parallax is still not perfect. The orb and cube are visible in all displays but the displays with 10 parts does cut it off making parts of it disappear.

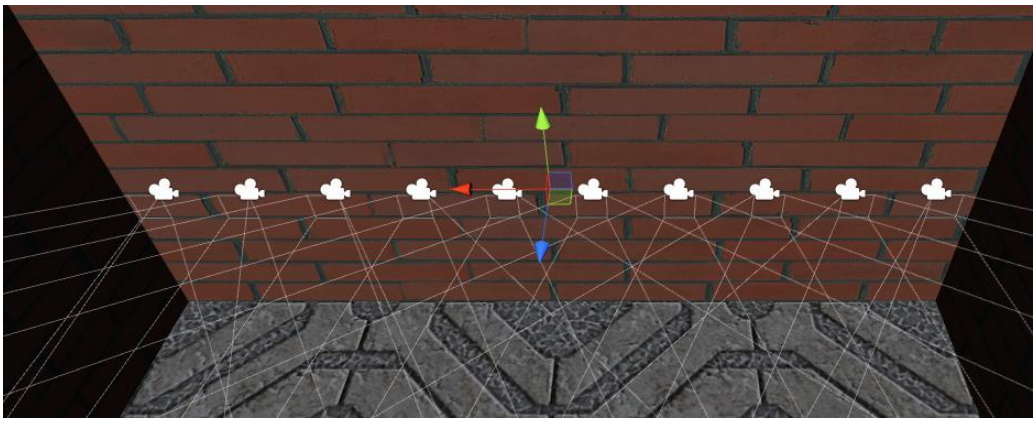
6.3 Implementation

The divided display was relatively simple to complete but there were a few challenges. The first challenge was how to move the texture based on the position of the player. This was made more complicated by the fact that the viewer avatars position was affected by the parent object but once I figured that out I could fetch the position of the parent object and subtract it from the viewer avatars position. The second problem was camera placement. I started by putting ten cameras in a straight line with even spacing and noticed that this created a large "gap" between the cameras creating dead zones that no camera covered, this can be seen in Picture 25. This caused the movement parallax to become jumpy and very noticeable at the seams between the display segments.

My first attempt at solving this was to increase the number of cameras to a point where there were no dead zones and the entire area was covered, this resulted in the 27 camera setup. This was effective at reducing the jumpy feeling in the movement parallax but the seams between the display segments were still very noticeable so I created another simulation with 50 cameras and 50 segments. With this I noticed another clear jump in quality but performance started to suffer and the lag was very noticeable making further increases in cameras pointless.

The alternative solution to this problem, placing the cameras in a semi-circle angled inward, covered all the dead zones better with less cameras being used. However the resulting image was distorted, as can be seen in Picture 16, making the system unusable.

The display using 351 cameras was an attempt at creating a display that could show both horizontal and vertical movement parallax and to see where the upper limit of the hardware was. The movement parallax does indeed work but since the performance was so low the simulation is practically useless unless much more powerful hardware is used.



Picture 25. Showing the gaps between cameras creating dead zones.

6.4 Future Work

The simulation implemented in this project is one of many that could be made by a more knowledgable team. For two-view display technology both parallax barriers and lenticular display might be doable in simulations. Using a virtual reality headset time multiplexing displays could also potentially be made since each eye has a separate screen that could be turned on and off like the time multiplexing glasses normally used. Time-multiplexing is however limited at the moment due to the hardware in modern virtual reality headsets like the Oculus Rift since they do not support the 120 Hz prefered.

In the future more simulation could be done, the research team had two simulations in mind at the start of the project but only one was implemented. The second simulation would collect the feed from all cameras and combine it into a single render texture. This was looked at during the project and it looks like normal render textures do not support multiple cameras to the same texture or fusing different render textures. In order to complete this custom shaders could instead be used, another alternative is to use normal textures that are replaced during every frame. Unity3D also supports plugins which may be usable to create the desired effect.

The alternative camera placement could also be further researched. It would be a good solution if the angles can be offset to avoid the warping, it could be a good way to reduce cameras while also capturing more of the scene.

There are also other 3D engines available on the market but since the scope of this project was to only use Unity3D these have not been looked at. In the future other 3D engines could also be compared to see if there is an engine that better suites the project.

6.5 Conclusion

There is still much work to be done before simulating the team's research is a viable option. First other 3D engines must be investigated to see if they offer better performance or have functions that are better suited for the task. More complex simulations must also be attempted since the simulation performed in this project was quite narrow and simple compared to the final product. Performance must also be taken into account to see if more effective ways are available.

The project does however show that simulation is worthwhile to keep looking into since the simulation attempted was successfully implemented. Most of the problems encountered during the project are also most likely solvable with more time and more qualified people looking into it.

This leads me to conclude that there is potential in the technology and that the research team should continue looking into simulations as a potential tool for their research.

References

[1] Dictionary.com “Parallax”
http://www.dictionary.com/browse/parallax
Retrieved: 2016-05-10

[2] Internet Archive “The Camera Obscura : Aristotle to Zahn”
https://web.archive.org/web/20080420165232/http://www.acmi.net.au/AIC/CAMERA_OBSCURA.html
Retrieved: 2016-04-01

[3] Unity “Fast Facts”
https://unity3d.com/public-relations
Retrieved: 2016-03-30

[4] NAS “Virtual Reality: Definition and Requirements”
http://www.nas.nasa.gov/Software/VWT/vr.html
Retrieved: 2016-03-30

[5] Virtual Reality Site “Head Mounted Displays”
http://www.vrs.org.uk/virtual-reality-gear/head-mounted-displays/
Retrieved: 2016-03-30

[6] Kickstarter “Oculus Rift: Step Into the Game”
https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game
Retrieved: 2016-03-30

[7] Oculus “Oculus Rift is Shipping”
https://www.oculus.com/en-us/blog/oculus-rift-is-shipping/
Retrieved: 2016-03-30

[8] Neil A. Dodgson,
“Autostereoscopic 3D Displays”,
IEEE Computer Society, August 2005, s. 31-36

[9] Hakan Urey, Kishore V. Chellappan, Erdem Erden and Phil Surman,

"State of the Art in Stereoscopic and Autostereoscopic Displays", Proceedings of the IEEE | Vol. 99, No. 4, April 2011, s. 540-552

[10] Apple Inc "Final Cut Pro 7 User Manual: How Many Frames Per Second Is Best?"
https://documentation.apple.com/en/finalcutpro/usermanual/index.html#chapter=D%26section=3%26tasks=true
Retrieved: 2016-05-27

[11] Unity "Scripting API: CharacterController.Move"
https://docs.unity3d.com/ScriptReference/CharacterController.Move.html
Retrieved: 2016-05-12

[12] Oculus "Oculus Runtime for Windows"
https://developer.oculus.com/downloads/pc/0.8.0.0-beta/Oculus_Runtime_for_Windows/
Retrieved: 2016-05-06