

**Applied Game Technologies – Report**

**Michael Little - 1103677**

**Contents**

1. Introduction
2. Description
3. Guide
4. Software Outline
   1. Main
   2. S3D
      1. Update
      2. Render/DrawHUD
      3. CreateCubeMesh
   3. Camera
      1. Update
      2. Render
   4. GameObject
      1. Update
   5. System
      1. SphereCollision
      2. Update
5. Problems and Issues
   1. Camera Lookat
   2. Collision Detection
   3. Object Scaling
6. Reflection
7. Discussion
8. References
9. **Introduction**

The purpose of this report is to describe the application for Applied Game Technologies coursework submitted with this document by Michael Little.

For this coursework we had to create an application using one of two chosen technologies: Stereoscopic 3D or augmented reality, and design an application that utilises the technology to its advantage. Stereoscopic 3D utilises special rendering techniques and coloured glasses to trick the users eyes into perceiving depth in 2D images, and augmented reality utilises a camera and real world markers to make it appear as if computer generated assets are in the real world.

In this document we will give a brief description of the final application and its features as well as breakdown of its various mechanics, software design and any problems encountered along the way. At the end we will reflect on the development and discuss any possible opportunities for innovation in technology relative to the submitted application.

1. **Description**

The application submitted for this assessment is a simple yet fun game created for the PlayStation Vita (SONY. 2011) utilising the Abertay framework (ABFW). For the coursework we had to select one of two different technologies and create an application that utilised the technology in an innovative or interesting way that benefitted the game as a whole. The choices were between augmented reality and Stereoscopic 3D, for this submission it was decided that stereoscopic 3D would be chosen as it would allow for easy development of game mechanics without need of the devices camera or a marker for the device to track. The 3D effect is done by taking an image of the 3D environment from two separate positions, colouring them red and blue and displaying them both at a predetermined distance apart, when viewed through specific colour lensed glasses it should give the player a perceived sense of depth.

The design for the application was thought over and over, but in the end it was decided that a design that would utilise the 3D’s added sense of depth and the added immersion of a first person camera view would work best. The original design was a game where the user had to move around a 3d world collecting objects to build up as high a score possible before the timer ran out, this design felt rather dull and didn’t take the best advantage of the 3D effect as the objects all moved at the same speed throughout the entire game and there wasn’t much sense of immersion. To fix this, other games of a similar genre were looked at and it was discovered what was missing from the design, namely speed and an increasing sense of danger, these would allow the 3D effect to actually make a difference as the added sense of depth would greatly help out the users, and as the game would progressively get harder and harder the player would rely on it more and should also feel an added sense of immersion as they try to avoid all the obstacles.

The final application is a falling game where the player has to collect as many targets possible to build up their score and replenish the always decreasing timer, all before the game ends by either running out of time or they hit an obstacle. This design is similar to games like “AaaaaAAaaaAAAaaAAAAaAAAAA!!! – A Reckless Disregard for Gravity” (Dejobaan Games. 2009) and “Race the Sun” (Flippfly. 2013) which are games that rely on a user’s sense of depth and quick reflexes to achieve a high score.

This design was chosen as it was felt it would complement the 3D effect since it features objects flying towards the player at varying distances and speeds, and the added sense of depth perception from 3D would allow the user to easier navigate and judge how close / far away the objects were from the player and there for be more likely to achieve a higher score.

As the game plays out the player will eventually keep gaining a higher score, so to help add an element of increasing difficulty and excitement, every 10 points the player collects, the speed of the objects will increase and occasionally additional obstacles will spawn for the player to avoid. Thanks to the 3D and the ever increasing excitement from the moving objects the game gains a good sense of immersion where you can feel the speed of the objects flying past you, as you try desperately to avoid them.

1. **Guide**

When the game starts the player begins falling towards hundreds of different spherical targets, the player can then use the arrow keys to move left, right, up and down in order to collect as many as possible.

Each target gives the player a point and adds 3 seconds to the timer, the more they collect the higher the score and the more time they have. While collecting the targets large cube shaped obstacles will begin spawning and head towards the player, if they collide with the player or the timer reaches 0 the game will end.

To increase the difficulty every 10 points the player scores will increase the speed and occasionally spawn another obstacle for the player to avoid, soon the game becomes less about trying not to running out of time and more about avoiding the obstacles at ever increasing speeds.

A pause function was also included to help players out if they need to look away for a moment, simply pressing the Space bar will bring up the pause screen and pressing the Space bar again will remove it. While paused the user can use the right Shift and Control keys to adjust the convergence distance on the 3D to suit the user’s preference.

**Controls**

W = Move Up

S = Move Down

A = Move Left

D = Move Right

SPACE = Pause

(While Paused)

SPACE = Unpause

SHIFT = Increase Convergence distance

CONTROL = Decrease Convergence distance

1. **Software Outline**

As stated earlier the application takes advantage of the Abertay Framework for PS Vita, this allows for much easier use of DirectX and the various features of the PlayStation Vita. The application can be broken down into various different classes that create and handle the various elements that make up the game.

* 1. **Main**

Main.cpp is the base of the application, when called it creates the instance of the game and runs the application.

* 1. **S3D**

S3D is where all the other components come together; it inherits from Camera, GameObject and System and brings their various functions together to create the final application. The application also features the code that creates the two separate views of the game environment for the 3D effect.

**4.2.1** **Update**

The Update function fulfils several important functions for the application. The First function it serves is updating the Camera and System by calling their respective update functions.

The update function isn’t just used to call the update functions though, it also calls the System class’s SphereCollision function by passing through the Camera and the various GameObjects and depending on the result returned it takes the appropriate action. If a collision was detected between the Camera and one of the various targets, it will delete said target, award the player one point and spawn another target to take its place. If the collision was between the Camera and an obstacle it will delete the obstacle and set GameOver to true.

It also checks for any player input and then calls the appropriate move functions in the Camera class.

**4.2.2 Render / DrawHud**

The Render function’s main purpose is to prepare everything for the RenderScene function; it does this by calling the Camera’s render function which updates the Camera’s variables. It then uses the updated variables to create the different projection matrixes for the left and right eye. It then calls the RenderScene function twice and passes through the different variables for the left and right eye (e.g. target, view matrix), this creates the two seperate images that when viewed through the S3D glasses creates the 3D effect.

The DrawHUD function is where all the text drawn on the screen is called, it takes the font that was loaded in the InitFont function, uses this and the abfw::Font class’s RenderText to display all the information the player needs to play the game, and depending on different conditions (e.g. paused, gameover) it will also display specific messages.

**4.2.3 CreateCubeMesh**

The CreateCubeMesh function is used to create the various cube GameObjects that the player has to avoid. The function uses abfw::Mesh’s CreateMesh function to create a cube of a predetermined scale whenever called, it also assigns it a bounding sphere used for collision detection.

* 1. **Camera**

The Camera class utilises various matrices to create a camera that can freely move around the game environment and view it in stereoscopic 3D.

**4.3.1 Update**

The Update function is used to keep the camera from leaving a designated area by checking the cameras position and moving it back when it moves too far horizontally or vertically. It also features functions that will eventually return the camera to the centre of the screen if the player were to stop moving.

**4.3.2 Render**

The Render function is used to setup and update the cameras variables. Utilising several different equations it calculates the cameras forward, up and left vectors this allows the camera to move smoothly around the world without being affected by the cameras rotation.

The camera also feature code to generate a bounding sphere around the cameras position this is used in collision detection

* 1. **GameObject**

The GameObject class inherits from abfw::MeshInstance is used to control the various objects in the game.

The class takes in a mesh and has several functions to allow it to be freely moved around the world, it also features the bounding sphere code with allows the various objects to be used in collision detection.

**4.4.1 Update**

The Update function is used to check whether or not a specific GameObjects position has moved behind the camera, if so it moves the object to a random location far in front of the camera, if not it continues to move the GameObject towards the camera until it does.

* 1. **System**

The System class is used to hold/control the logic that makes up the game.

The class features two variables that are used throughout the game, score and timer. These two simple variables are the base on which the game is built and have several functions allowing the other classes to retrieve, set and change these variables.

**4.5.1 SphereCollision**

Another important function this class features is the SphereCollision function. This function takes in a GameObject and a Camera takes the radius of their bounding spheres, adds them together and checks if it’s smaller than the distance between the two objects and returns true or false appropriately.

**4.5.2 Update**

The class also features an Update function that checks if the timer has reached 0, if so it will set the GameOver Boolean to true. It also features a collection of IF statements that check the user’s current score and increases the objects speed and amount of obstacles spawned after an appropriate amount of points have been reached.

1. **Problems and issues**

During the applications development several different problems developed along the way, this is a look at some of these problems and how they were solved.

**5.1 Camera lookat**

During the integration of the camera class an annoying issue cropped up that didn’t want to go away, every time the camera moved vertically or horizontally it would also rotate towards the direction it was moving away from. This problem persisted for the next few weeks of development, luckily there were other areas that could be worked on but eventually the problem had to be addressed. The code was looked over top to bottom and cleaned up/optimised where possible, eventually the problem was narrowed down to the cameras lookat. The lookat was being updated whenever the camera was moved, so it seemed strange that the issue was still there, but after a closer look at the cameras variables it was noticed that the 3D’s left and right lookat weren’t updating. This was then fixed and the camera now moves around smoothly without unintentionally rotating.

**5.2 Collision Detection**

After the integration of the collision detection an annoying issue was cropping up, every so often when the program was run the collision detection would not work, or every object on the screen would register a collision and be deleted. It was uncertain as to why this was occurring as there were two Boolean variables that stopped the collision detection going off or not at all, and occasionally it would run smoothly. After a closer look at the objects variables while the application was running, an interesting discovery was made. Some of the variables were not what they were initialised as, this was confusing as they were all initialised in their individual class constructors, yet every so often the variables would change. To be safe the variables were initialised a second time, this time in the S3D classes Init function using setter functions, this fixed the problem yet it still seemed strange.

**5.3 Object Scaling**

The last major problem was that objects weren’t scaling when the transform.Scale() function was being called, this was especially annoying for the Ball object as without scaling it was very large and there was no way to change this. Eventually it was discovered that the problem was the result of two different issues; first was an over use of the SetIdentity() function, this was resetting the objects transform which was reverted any scaling that was occurring. The second was the order in which these transform functions was called, the Scale function had a SetIdentity() call within it so it had to be called first before any other transformation could occur on the objects. After this was working the Ball objects were scaled down to be the targets and the cubes were scaled up to be the obstacles.

1. **Reflection**

The purpose of this section is to look back at the development of the entire module and critically evaluate the approach taken and the effectiveness said solution. For this module we had to select one of two chosen technologies, either Stereoscopic 3D or augmented reality and create an application that utilises this technology in an effective way that emphasises the games features and creates an experience that essentially would be worse off without the chosen technology.

For this submission Stereoscopic 3D was chosen as it was felt that this would allow for more freedom with the game design aspect of the project, as it wouldn’t have to revolve around real world markers that interact with each other in some way. Before any progress could be made on the application a design for the gameplay had to be made, this design had to make good use of the 3D in a way that would benefit the gameplay as a whole. Since 3D can help users notice the difference in depths better as well as help the player feel more immersive, It was felt that a design that a first person game involving the player having to collect / avoid objects would work well in this case. The first person view in conjunction with the 3D helps emphasise the players sense of immersion as it will help appear as if the player was in the game and had to personally avoid / collect the various object flying at them which thanks to the added sense of depth from the 3D is much easier to accomplish.

If the application had to be done over again unless the other technology choice was chosen, there wouldn’t be a lot done differently, the main change would be implementing ways to make the 3D effect and gameplay better. This could be done by having the gameplay take place in a giant textured tube this would help the user gain an even better sense of depth as it would give the player a point of reference of how far away the objects are. Another thing that would be implemented is different visual effects that work with the 3D, for example maybe having the text in the game be affected by the 3D and have a depth factor, rather than be left out of the 3D rendering all together, this might help the 3D effect as a whole. Another thought that might help the players sense of immersion, is having the camera slightly rotate in the opposite direction when moving, this could give the impression that the player is turning to move, similar to real skydiving, Though its unsure how this would work with the 3D effect, it might help the players sense of immersion.

So in reflection its felt that the application fulfils the coursework requirements as it’s a fun simple application that takes good advantage of the technology in a way that emphasises not only the technologies features but the games features, in a way that creates a fun and immersive experience that the player can enjoy. There would be a few things done differently if given the chance but they would be mostly visual and only to emphasise the gameplay and the 3D effect.

1. **Discussion**

The purpose of this section is to discuss the various opportunities for innovation within the chosen technological field that the application takes advantage of.

3D has come a long way since the days of the original Stereoscope, 3D has now become ingrained into our culture, from movies to games 3D allows us to create a more immersive and breath-taking experience that can emphasise the excitement of an action scene or the beauty of an expansive landscape.

All 3D effects are tricks, each uses different techniques to trick our eyes into believing that the 2D images we are seeing are actually one three dimensional scene. This is done in various ways but they all share one common thing, each eye sees a different image and our brain combines them to create a 3D effect. How this accomplished has changed a lot thanks to various advances in technology, original 3D films accomplished this by utilising the stereoscopic 3D that was used in this application, this was done through having two separate different coloured images on screen that when viewed through coloured lenses would allow each eye to see a different image and produce the perception of depth. These days there are much more effective ways of creating this effect without ruining the visual quality of the image by colouring it red and green; one way of doing this is by utilising either Active or Passive Glasses. Active Glasses utilise an advanced shutter system that’s synced with the display, this system swaps the left and right images at high frequency on the display and the shutters in the glasses makes sure that the correct eye sees the correct image. Passive glasses are more similar to the coloured Stereoscopic 3D system but instead of having the images be different colours the system uses special lenses and polarised light to filter which image each eye sees.

Another way 3D is accomplished is with advanced Auto-Stereo displays, these displays negate the need for glasses though they do require that the user has to view the 3D image at a specific angle otherwise the effect doesn’t work perfectly. This method can be found in the New Nintendo 3DS (Nintendo. 2014) but to help combat the problems with this method it also utilises the systems front camera to track the position of the users head; this negates the need for the user to view the image at a specific angle, as it will adjust itself based on the users position.

For this project if the application was to utilise any of these new technologies it would help the project as a whole, and allow for the application to take better advantage of the vita’s technology and visually look much better. The biggest problem with this version of Stereoscopic 3D is that it uses 2 colours to create this effect; as such other colours don’t show well, and some aren’t allowed to be used in the display. If one of the more advanced 3D techniques were used the first thing to change would be that the different mesh’s would be coloured / textured differently and the background would be given another colour other than black.

So in conclusion 3D has definitely advanced since its inception and has become much more advanced and readily available without having to sacrifice the overall visual quality of film/game. And if this was to be implemented into the submitted application it would definitely look better and be able to take advantage of different colours and textures, but like 3D itself it would only really be a visual improvement.

1. **References**

**Consoles**

Sony. 2011. *Playstation Vita*. [online] Available From: <https://www.playstation.com/en-gb/explore/ps-vita/> [Accessed 27 April 2015]

Nintendo. 2014. *New Nintendo 3DS*. [online] Available From: <https://www.nintendo.com/3ds/new-nintendo-3ds> [Accessed 27 April 2015]

**Games**

Dejobaan Games. 2009. *AaaaaAAaaaAAAaaAAAAaAAAAA!!! – A Reckless Disregard for Gravity.* [Download]. PC, Mac, Android. Available From: <http://www.dejobaan.com/aaaaa/> [Accessed 27 April 2015]

Flippfly. 2013. *Race the Sun*. [Download]. PC, Mac, Linux, PS3, PS4, PS Vita. Available From: <http://flippfly.com/racethesun/> [Accessed 27 April 2015]