**CMP404: Applied Games Technology Report**

**Callum Henderson – 1302869**

# Project Aim

The aim of the application proposed is to make use of the AR marker system used with the Sony PlayStation Vita (Vita) to create a playable game.

The intended game will make use of the AR markers as a sort of waypoint system that a game object will then follow to a destination that will be marked by AR marker 6. The aim of the game will be for the player to fire projectiles out of the vita screen at the objects moving between the makers with the goal being to destroy these objects before the final waypoint is reached.

# Background

Augmented Reality (AR) is a technology that has been around for quite some time now, but has only really began to appear in everyday applications. With the success of Pokémon GO (Niantic, 2016) and its huge player base of 45 million at its height, (Javornik, 2016) A case can be made that it is viable for AR to be a large commercial success. However augmented reality has been around well before the release of Pokémon Go.

In 1968 the first prototype of what would be considered AR was created by Ivan Sutherland. It consisted of a head mounted display that would display virtual information over objects in the physical world. In the following decades AR was developed further on, eventually leading to prototypes being developed for military purposes. These prototypes would be able to identify terrain and overlay information relevant to what the user was looking at in the physical world. (Javornick, 2016)

It would be in 2008 however that the first commercial AR application would be released to the public, where BMW used an advertisement to provide a marker for a user with a camera equipped computer to view a 3D model of the car (Fig 1).

The advertisement was revolutionary in that it gave the user the ability to rotate and view the model as if it existed in physical form by simply rotating the paper with the advertisement in front of their computer camera.

With phones becoming more and more powerful, the viability of AR because significantly improved as the common smartphone became powerful enough to run the image recognition software required for AR to function. Widespread use of AR would begin to appear in 2011 with snapchat (Snap Inc., 2011) and its face recognition software. That provided a marker less solution for users to impose various “Filters” over images of themselves in real time.

The first example of a portable game making use of AR would be in 2000 when Bruce Thomas demonstrated the first portable augmented reality game (Augmented Reality Games, 2017) This setup required a backpack mounted computer as well as a gyroscope, however this allowed the user to play a game based on Quake (id Software, 1997) called ARQuake. (Wearable Computers Lab, 2000) Later AR would be adopted console developers such as Nintendo including AR capabilities into the 3DS (Nintendo, 2011) and Sony with the PlayStation Vita. (Sony, 2011)

Figure 1 - Image of the BMW Mini

AR advertisement (Strauss, 2008)

However augmented reality failed to gain traction with handheld consoles due to lack of support of third party game developers. This meant that there were no titles to draw consumers to the consoles leading to AR falling into obscurity within the portable console gaming market. But with the startling success of Pokémon GO on mobile phone as well as the majority of games making use of AR being found on smartphones, it seems AR has found its place on the mobile gaming scene.

**User Guide**

The game requires that 6 PS Vita AR markers are used. Each marker should be placed in such a way that the marker preceding and proceeding should be visible, however both do not have to be visible at the same time. Scan the markers in 1-2, 2-3, 3-4, 4-5 and 5-6.

After these markers have been scanned in, a prompt should confirm that the co-ordinates between marks have been calculated and a prompt should appear to begin the game looking at marker 1.

When the game is started, the tank will begin moving to the next waypoint. The goal is to use the projectiles fired by using the square button to destroy the target before it is able to reach the final waypoint.

Should no markers be visible, the game shall pause and prompt the user to re-acquire an AR maker. Once the vita has re-acquired a marker the user will then be able to unpause the game. The game should ideally be played at such a height that a marker is always visible to the Vita. After the target has been destroyed the player will be prompted to restart. This will be an opportunity to move the markers to form a new rout should the user desire. The user will then be prompted to once more scan the markers.

# Methodology

The system will dynamically change the marker used as the reference for the world based on the visibility of each marker. The purpose behind this is to ensure that as long as a single marker remains visible to the Vita, gameplay may continue and transformations and gameplay logic can still accurately progress. The active marker, which shall be named the parent as all game transformations will be done local to this marker so the game objects will move independently to the world transformation that is set to the Vita’s perspective. The game will also make use of marker local positions between each marker in order to calculate a markers position relative to every other marker in the world. This will ensure that if a marker is not visible or is off screen, the local position between the target marker and the current parent will be used to calculate the position of the target marker relative to the position of the current parent. This will mean that at the start of the application each marker in the waypoint will need to be captured with the marker before it and with the marker after it.

Figure 2 demonstrates the logic behind storing the local co-ordinates between the markers proceeding and preceding in order to accurately determine their positions while they are off camera. The game will also feature the ability for the player to fire projectiles from the vita screen. Here the projectiles will be created behind the vita camera and will be initially given a velocity to move down the Z-Axis of the Vita.

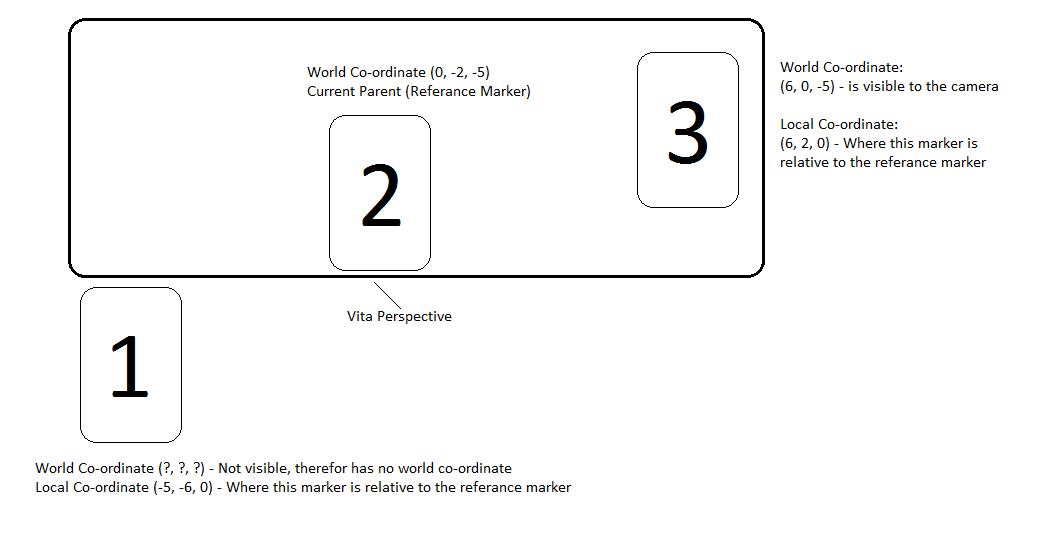


Figure 2 – Representation of how world and local co-ordinates can be used to determine the location of a marker that is not visible

The problem with this however is that the projectiles will not move if the vita’s perspective is moved due to the projectiles being moved down the Z axis of the world and as the world will always use the Vita’s perspective as the origin, the projectiles will not move as intended. In order to resolve this, the projectiles will be assigned local co-ordinates to the current parent marker. Through calculating the relative co-ordinates between the Vita (which will always be (0, 0, 0) and the world co-ordinates of the parent marker, we can then use this as the local transform of the projectiles. It will then be a case of simply multiplying the matrix of the projectile by a matrix containing the desired velocity the bullet should travel in.

Finally, An object will be created that will move sequentially between the waypoints at a set speed. The object will use the co-ordinates calculated from the relative positions between markers to move in local space to the marker it is parented to. A position to each marker will be recorded as needed so as if the parent is lost, the local co-ordinates from the new parent can be applied to the object so as it will not change its gameplay position every time the parent is changed. Another advantage to using local co-ordinate system is that if the waypoint object, the marker or even both are off screen, the object will still move towards an accurate position of the target marker relative to any markers that are still visible.

The entire system is reliant on each marker having seen the other markers in order for the relative positions to be known. Another problem would be if a marker is moved during gameplay, then the game would require that marker be viewed along with the other markers in order to recalculate the local positions. Failing to do this will result in the game object using the relative co-ordinates of the old position should the marker not be on screen.

# Program Design

1. Game Object Class

The game object class takes inheritance from the mesh instance class as all objects in the game will make use of all the data contained within the class. Game Object is then given additional variable and functionality that will be relevant to all game objects in the future, for instance, Local Transformation manipulation and functionality to assign reference (referred to as parent) markers.

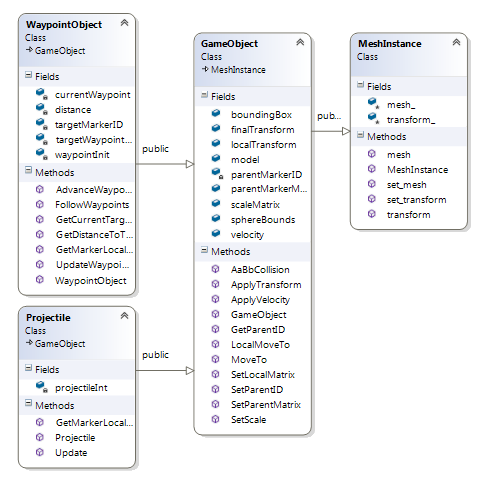
Game object is then further inherited by the Waypoint Object class and the Projectile class which contain functionality and variable relevant to those classes themselves. Overall this helps with memory since if the functionality within waypoint object were to be added to game object then all game objects will contain a large amount of unnecessary data.

Figure 3. Code diagram of the GameObject class and its parent and child classes

1. MarkerInformation

The marker information class is designed to hold all of the information relevant to the AR markers in a centralised class. The class contains getter and setter functions for access to data as well as holding important information such as local transforms, marker visibility and world positions of markers. The marker container is created within the ArApp class and is passed to functions within the GameObject class as well as its child classes via a pointer. This ensures that the data being used by these classes is always the most recent and saves on memory as all of the data structures within the markercontainer are not being passed unless they are called by its getter.

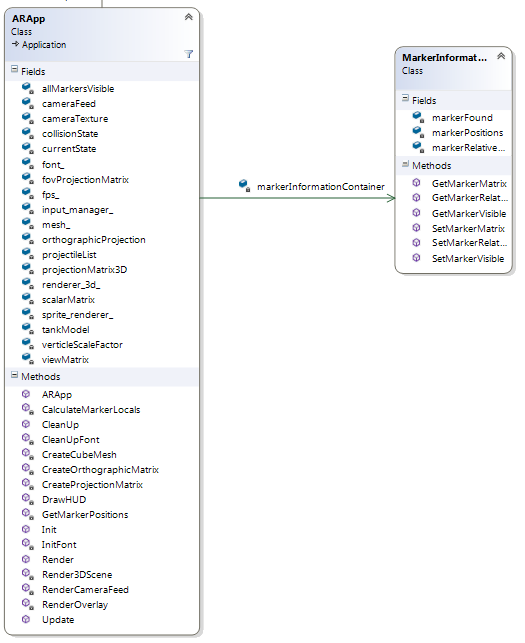


Figure 4. ArApp Class and the Marker Information Class.

# Evaluation

1. Marker Information Container

Stepping off from the initial lab work where marker reading was implemented a class header was created as a container for information relating to the makers. This marker container contains various information including; an array containing the matrices of all visible markers, an array containing the visibility status of each marker and a 2D array containing the local matrix of each marker to every other marker. The container also contains various utility functions, primarily getter and setter functions for retrieving and updating data.

Though for the purpose of the application created, each marker only requires the local matrix of the markers preceding and proceeding it. However, the system is capable of storing the local position of every other marker that has been visible. This sort of information can then be used to calculate the position of markers that are not visible thus allowing for the single marker system to work even if waypoints are not visible.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | IDENTITY | 2-1 Local | 3-1 Local | 4-1 Local | 5-1 Local | 6-1 Local |
| 2 | 1-2 Local | IDENTITY | 3-2 Local | 4-2 Local | 5-2 Local | 6-2 Local |
| 3 | 1-3 Local | 2-3 Local | IDENTITY | 4-3 Local | 5-3 Local | 6-3 Local |
| 4 | 1-4 Local | 2-4 Local | 3-4 Local | IDENTITY | 5-4 Local | 6-4 Local |
| 5 | 1-5 Local | 2-5 Local | 3-5 Local | 4-5 Local | IDENTITY | 6-5 Local |
| 6 | 1-6 Local | 2-6 Local | 3-6 Local | 4-6 Local | 5-6 Local | IDENTITY |

Though this system works well in that to pass information to a game object, a pointer is given to the marker information container thus drastically reducing the memory usage for multiple objects requiring access to the marker information. However an aspect that makes this inefficient is that the data stored within the array and be effectively halved by simply using the inverse of the opposing matrix. For instance, the relative position of markers 1 – 2 is the inverse of 2 – 1. Not only would the array be half the size, it would also save on operations to calculate the local position of those co-ordinates.

1. Game Objects

For the creation of game objects a core class with common components intended for use with all objects in the game was created. This class takes a simple constructor consisting of a spawn location and a mesh to apply to the created game object. Thanks to game objects being created as pointers to game object, memory usage is reduced as well as providing a pointer to any meshes intended for use.

The waypoint class contains all of the necessary variable and functionality required for game objects intended to use the waypoint system. This class requires an update function be ran that will take a pointer for the marker container in order to provide up to date matrices for currently visible markers. The class also contains functions to calculate relative positions between game objects and markers which is used with the parenting system.

The projectile class is a separate class inheriting from game object that contains the required functions and logic for the projectiles. This class is significantly smaller compared to waypoint objects due to their single purpose of moving towards the point of the world origin at the time of firing. The logic inside this class is similar to that used in the waypoint object class in that it parents a projectile to a visible marker and uses this marker to apply transformation to the projectiles. Without this system, the projectiles would simply either move in world space, thus only ever moving down the Z axis of the Vita’s perspective, or it will move down the Z axis of the parent marker.

Though these classes attempt to contain information that is only relevant to objects of those types, some functions such as calculating local position from object to marker is reused in some classes. This can be resolved by moving any common functions or variable up into the Game Object class where it can instead be inherited by the waypoint and projectile classes.

1. Summery

Overall the application was designed to be as efficient with memory usage as possible. By passing a pointer to a central information container, this has drastically reduced the memory usage compared to simply passing matrices to functions that require them. This can also be seen with the modular nature of the Game Objects class that has been designed to make heavy use of inheritance for its child classes to provide a solid basis for all game objects while leaving specific functionality to the child class responsible for it. This means that there is little to no unnecessary being stored in game objects. A point I would like to improve on would be to restructure the application to make use of a state machine. A state machine would allow greater control and optimisations to be made, such as removing data that is not being used from memory and only storing data for functions that are currently being used in that state. The current system makes use of an enum system to run various components of the game and while this does only run functions that are required in that current state, the data for the objects in use are still in memory.

# Problems and Solutions

1. Co-ordinates system

The initial prototype made use of a world based co-ordinate system in order to manipulate the waypoint object. This worked for its intended purpose as the game object did successfully move towards the designated waypoint. The initial plan was to use addition and subtraction of world co-ordinates and store it as a local co-ordinate for the marker. However, this presented the problem that if the vita lost track of a marker, then the game objects position would be permanently changed. This also presented the problem of the vita’s movement also changing the world position of the game object

Solution

The solution to this problem was to have the entire game work within the local co-ordinates of a reference, often referred to as the “Parent” marker. Thanks to the object now working in the local space of a marker, any movement of the vita did not affect the objects position and it would always maintain its correct position relative to the parent marker. This also made calculating the local move matrix far easier as the result was simply the relative transform between two markers that would have the X Y Z co-ordinates placed within a vector, normalised and multiplied by a desired speed multiplier and then placed back within the local transform matrix.

1. Projectile System

The projectile was initially planned to move down the Z axis of the vita screen and simply parent to whatever marker was visible at the time. The problem with this is that the projectile would spawn, and then move to the markers position and then begin moving down the marker’s negative Z axis.

Solution

The solution to this problem was to take the local co-ordinates between the projectiles initial spawn point, which will always be 0,0,0 as it spawns in the Vita’s origin, so the local transform was simply the parent marker’s world co-ordinates. This is then assigned as the projectiles local transform relative to the marker. No rotational data is provided to the projectile as this would cause the projectile to face in the direction of the marker, therefor the projectile retains its initial starting rotation. The projectiles local transform is then multiplied by a matrix containing only the Z velocity desired for the projectile to move in. This results in the projectile moving down its own Z axis as well as the projectile also being affected by the rotation of its parent marker.

1. Marker Parenting

After the conversion to use local co-ordinates, a solution needed to be created in order to have game objects change to another reference marker if the initial reference marker is lost. This was a simple case of checking if the reference marker is visible and if not, changing it to a marker that is visible. A problem emerged where the local transform of the object would be retained and after the new reference marker is applied, the game object would use its old relative position on the new reference marker (Fig 3)

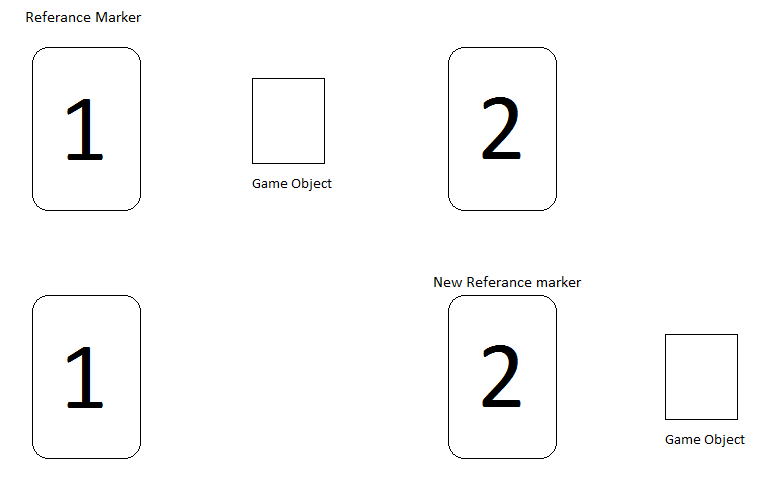


Figure 5.

Representation of the game object retaining the old local position after changing reference marker.

Solution

To resolve this problem, when the reference marker is changed the game object calculates its local position relative to its new parent marker. When the parent matrix is then applied to the game object, the local position is also changed to the relative position between the game object and the new reference marker. This ensures that the game object will maintain its old position regardless of the marker that is then chosen to be the reference.

# Further Innovation

After the great success of Pokémon Go (Niantic, 2016) in 2016 its legacy fell to the wayside very quickly, and a year later it is remembered as a short lived viral success. Since then no other games that make use of AR has been able to grader the success that Pokémon go had accomplished and AR technology as fallen to the wayside for games development.

However, with the current rise in popularity of Virtual Reality (VR) there lies an opportunity for AR to still be used in mainstream games development in the form of Mixed Reality (MR). MR makes use of the user immersion and control that VR provides but also provides the windows into the physical world that AR provides. As VR continues to grow more and more in popularity, technology such as the Microsoft HoloLens (Microsoft, 2016) or the Asus HC102 (Asus, 2017) that introduce both AR and VR open up the possibility of AR to be used feasible component.

In relation to the application created, the introduction of VR could provide a realistic and immersive environment for the player. For example, in the context of the application the player can be placed in the cockpit or the operator’s terminal of a military gunship and the AR technology of the headset can be used to detect the markers and set up a gameplay environment using the markers. Much like with the Vita, this will allow the player to set up a rout in anyway desired.

Another aspect of the application that can prove to be very useful is the dynamic reference markers system that will use whatever marker visible to expand the play area for a game using AR. This can provide a game with a theoretical limitless play area depending on the amount of markers supported by the device and with the introduction of accelerometer or infrared sensor bars, more accurate calculations can be made in regards to tracking the players head location and recalculating game geometry accordingly.

# References

Niantic (2016) *Pokemon Go – Android, IOs* [Video game]. Niantic.

Snap Inc. (2011) *SnapChat* – *Android, IOs* [Software]. Snap Inc.

idSoftware (1996) *Quake – PC* [Video game]. idSoftware

Wearable Computer Lab (2000) *ARQuake – Portable Computer* [Video game]. Wearable Computer Lab

Nintendo (2011) *Nintendo 3DS*. Nintendo Entertainment

Sony (2011) *Playstation Vita*. Sony Corporation

Microsoft (2016) *Microsoft Hololens.* Microsoft Corporation

Asus (2017) *Asus HC102.* AsusTek Computer Inc.

Javornik, A. (2016) *The Mainstreaming of Augmented Reality: A Brief History.* Available at: https://hbr.org/2016/10/the-mainstreaming-of-augmented-reality-a-brief-history (Accessed: 18 November 2017)

Strauss, P. (2008) *Mini Augmented Reality Ads Hit Newstands*. Available at: https://technabob.com/blog/2008/12/17/mini-augmented-reality-ads-hit-newstands/ (Accessed: 18 November 2017)

Augmented Reality Games. (2017) *Augmented Reality History*. Available at: http://www.augmented-reality-games.com/history.php (Accessed: 19 November 2017)