Abertay University
Institute of Arts, Media & Computer Games

Applied Game Technology Report

Augmented Reality using Object-Transform Relationships

Craig Jeffrey - 1203086 *April 2016*

Contents

1. Introduction
2. Development
    2.1 Progression of Ideas
    2.2 Application Description
        2.2.1 Editing Stage
        2.2.2 Combat Stage
    2.3 Implementation
        2.3.1 Transform Parenting
        2.2.2 Procedural Meshes and Object Colourization
        2.3.3 Editing Stage Object-Transform Relationships
        2.3.4 Combat Stage Relative Transforms
    2.4 User Guide
        2.4.1 Editing Stage
        2.4.2 Combat Stage
        2.4.3 Difficulty Modes
3. Appraisal
    3.1 Potential Applications
4. Conclusion
5. References

1. Introduction

Three types of computer-mediated reality have recently made a resurgence in popularity. These are augmented reality (AR), virtual reality (VR) and hybrid or mixed reality (HR/MR). Augmented reality involves enhancing perception of the real world with additional information. Virtual reality involves replacing the real world entirely, completely immersing oneself in a virtual world. Hybrid or mixed reality is a combination of both where the real world and one or many virtual worlds are combined.

VR requires specialized hardware, a number of consumer grade VR devices exist such as the HTC Vive (2016) and the Oculus Rift (2016). HR also requires specialized hardware - the most prominent current hybrid reality device being Microsoft's HoloLens (2016). Augmented reality, on the other hand, can be done by most smartphones as it only requires a camera to begin overlaying augmentations over the camera feed.

This report focuses on the exploration of augmented reality using Sony's PlayStation Vita. The goal is to create a game application which exploits the potential of AR technology.

2. Development

In order to accomplish the goal of creating an AR game which exploits the strengths of AR, several mechanics and ideas were entertained.

2.1 Progression of Ideas

1. The initial idea came from an excellent puzzle game titled "*Antichamber*" (2013). In *Antichamber*, there are many puzzles which involve tricking the players perception of the game world with many not following the laws of physics. The specific example leading to the first idea is a door in the game which is closed when the player looks at it but open when not. It was first speculated that this trick is implemented by checking if the door is within the view frustum. This gave rise to the idea of changing changing the perception of objects depending on how they are viewed. Consideration was given to implementing a puzzle game where objects are viewed differently through several other "window" objects potentially using stencil buffers to change what is drawn behind these window objects.

2. One problem with the initial idea is that it was unknown how to use stencil buffers within the Abertay Framework for the PSVita. The idea of changing objects depending on how they were viewed was continued but with a different niche mechanic. The second incarnation of the idea was an alternative, simpler way to implement the door trick in *Antichamber.* Rather than expensively checking if something was within a view frustum the door could have simply been comparing the camera transform with the door transform. If the player is looking in the direction of the door at an angle that would put it in the view frustum, then close the door. The mechanic then changed from using window objects to simply changing the object viewed depending on the angle it is viewed from, e.g. viewing an object from the left makes it a cube whereas viewing it from the right makes it a sphere.

3. Following on from the second idea, it was realised that a generalization of altering objects by comparing different transforms could create potentially endless interesting interactions between objects. Many existing AR games detect features present in the real world and use their transforms to place things. An example of this is *PulzAR* (2012) where various objects are simply placed at marker transforms. This is effective for many games but it is rather superficial use of the potential of AR. This type of game could just as effectively be implemented with alternative controls without using augmented reality. Microsoft's HoloLens Minecraft Demo (For The Replay 2015) is an example of better use of the technology (HoloLens is technically mixed reality but the feature I'm referring to is borrowed from the AR side of mixed reality). The demo involved manipulating the view of the game world using hand gestures, where different gestures correlate to different ways of manipulating what is viewed. Pinching fingers and moving drags the view of the world, for example. Controlling the world this way would be impossible without augmented reality. Speculatively, this is implemented by replicating changes in the translation of the fingers transform when pinching with the translation in the global world transform for the game world. This is an example of an object-transform relationship: the game world is the object being controlled, through creating a relationship between the translation component of the world's transform and the translation component of the player's fingers.

The final idea is therefore to create relationships between objects by comparing different elements of their transforms. A prototype application was created as a proof of concept.

2.2 Application Description

The application developed is a two-stage game where the first stage is an editing stage and the second stage is a combat stage. Two markers are required to play.

2.2.1 Editing Stage

In the first stage, each marker has an object sitting atop it. The object on the first marker is the player's object, whereas the object on the second marker is the "enemy" object. Each object has three properties:

1. The type of object. The two available types are Cube and Sphere.
2. The scale of object. (Roughly between ~1x and ~3x default size)
3. The colour of object. The colour range uses a gradient rainbow texture and approximates the object colour as Red, Yellow, Green, Cyan, Blue or Magenta.

Properties of the enemy object are randomized whereas the player object's properties are based on three object-transform relationships:

1. The players object type is based on the angle the first marker is viewed from.
   Viewing the marker from the left (positive yaw) results in a Cube.
   Viewing the marker from the right (negative yaw) results in a Sphere.
2. The players object scale is based on the distance between the first and second markers. Higher distance between the markers creates a larger scale object.
3. The players object colour is based on the angle of the second marker relative to the first marker. This can be visualized as a colour wheel surrounding the first marker.

Once the player object matches the enemy object, the combat stage can be entered.

2.2.2 Combat Stage

The second stage involves enemy objects spawning from the area around the first marker towards the player, i.e. the camera/PSVita. The player must destroy enemy objects by firing their own objects at them. All player and enemy objects in this stage are parented to the first marker (the second marker is unused in the combat stage), causing object trajectories to remain constant even when the camera/PSVita is moved.

The player wins by destroying a count of enemy objects dependent on the difficulty selected. If an enemy object collides with the player (an object placed at the position of the camera), the player loses. After winning or losing the player can return to the editing stage where the enemy objects properties are randomized again.

2.3 Implementation

A number of problems/relevant features and how they were solved/implemented are discussed in this section. An overview of the classes in the application and the functionality they implement is shown in Figure 1 on the next page. Not everything is covered, only what is relevant to AR or the implementation of the prototypes main features.

2.3.1 Transform Parenting

GameObject implements a transform parenting system. Every game object stores a local transform and optionally a pointer to a parent transform. GameObjects have three possible “TransformState”s. These are Absolute, Relative and Deferred. Absolute causes the local transform to be used to build the objects final transform, ignoring the parent transform. Relative builds the transform using the local transform and the parent transform (if there is one set). Deferred ignores the local transform, using only the parent transform (falls back to using local transform if no parent is set). This system was developed to make it easier to create transform hierarchies, simplifying placing objects relative to marker transforms or potentially any other transform.

2.2.2 Procedural Meshes and Object Colourization

All of the objects for the game use procedurally generated meshes which are generated in the initialization of the application. Each mesh class is derived from a PolyhedronMesh class which defines a virtual function for generating the mesh given a scale and the platform, with optional parameters for the texture and initial texture u position. The available meshes are a simple 8-vertex Cube and a recursively generated Icosphere using 3 iterations.

C++11 had to be enabled to generate the Icosphere mesh as an std::unordered_map is used for vertex lookup to speed up and simplify generation, pairing vertex positions with indices. Not enabling C++11 gives a compilation error. Additionally, no std::hash function was implemented so a hacky solution which abuses undefined union behaviour was added to hash abfw::Vector3 components.

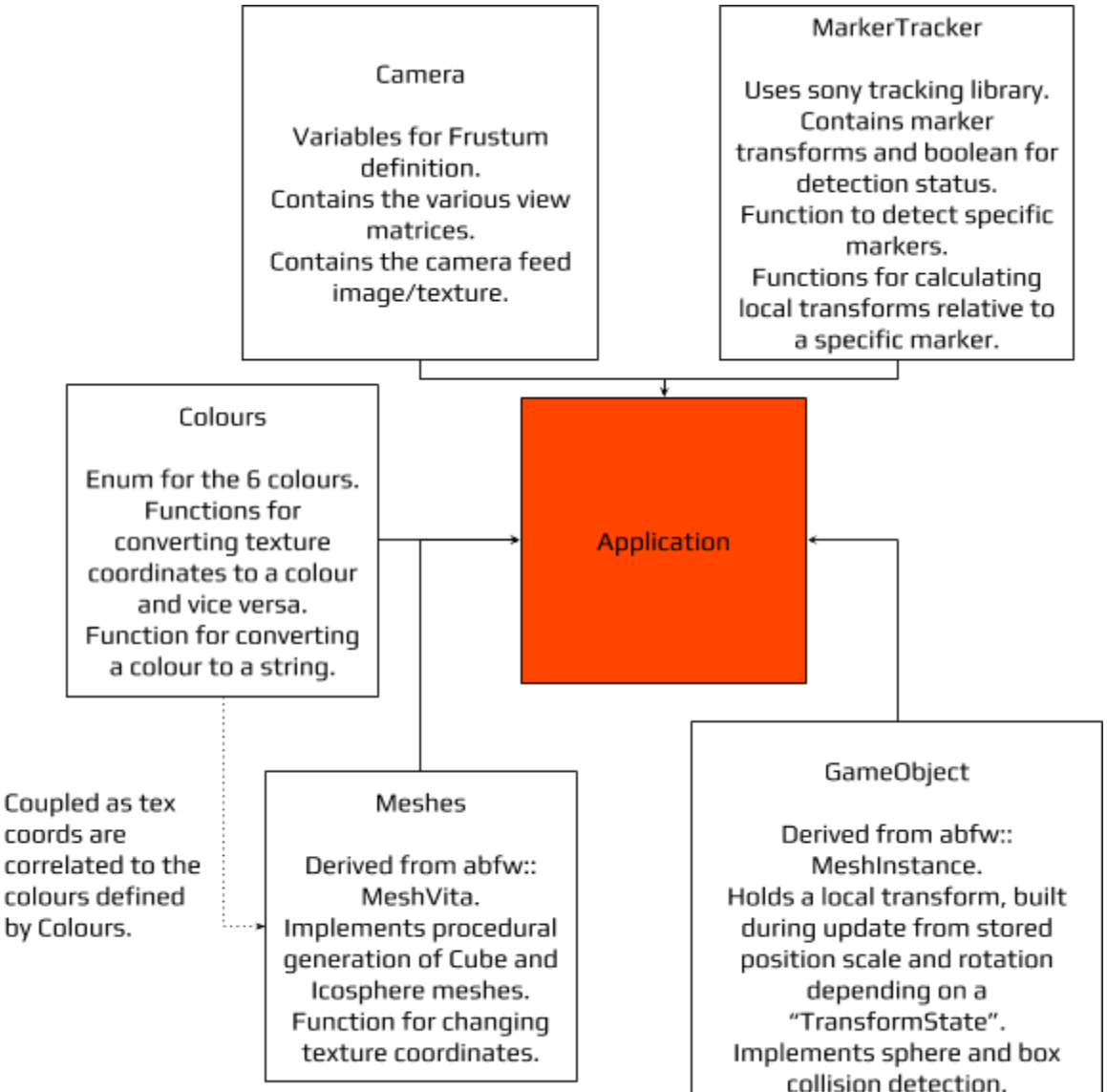

Figure 1: Application Overview

Every mesh uses 1D texture coordinates since they are only colourized and not fully textured. To avoid the potentially time consuming process of creating a shader for colourizing meshes and since there was no real benefit to implementing a scalable solution, a hacky solution was developed to colourize meshes using a gradient rainbow texture. To change colour, the texture coordinates for the mesh vertices is moved along the x axis of the gradient rainbow texture. Initially, changing vertex texture coordinates was done by calling InitVertexBuffer, but it was quickly discovered that this caused a memory leak as it simply allocated more memory every call. This required extending the abfw::MeshVita class with a function for updating vertices dynamically - since the available one had no implementation. The Colours class is used to approximately represent a texture coordinates u value as an enumerated colour, since using all 128 colours available from the 128 pixel wide gradient rainbow texture is too precise for game logic. The approximate labels given to these enumerated colours were picked empirically from the gradient rainbow texture.

2.3.3 Editing Stage Object-Transform Relationships

Three object-transform relationships in the editing stage for calculating the player object type, scale and colour were implemented:

- Type is based on the z rotation, or yaw, of the first marker. In order to extract yaw from a transform, a function was added to the abfw::Matrix44 class which first sets a quaternion from the matrix using the abfw::Quaternion class. The function then calculates the individual euler angles yaw, pitch and roll independently of each other. Since there is only two types they could be distinguished simply using the sign of the yaw value.
- Scale is based on the distance between the first and second markers. The relative position vector between both markers is calculated using the MarkerTracking GetRelativeLocalTransform function which uses the abfw::Matrix44 AffineInverse function to calculate the local transform of any other transform or a marker relative to the input reference marker. The scale is based only on the length of the relative position vector so the direction of the vector is irrelevant as is the order of calculating the relative local transform. Scale is simply set to the length of the relative position vector multiplied by 10.
- Colour is based on the relative angle between the first and second markers. This can be calculated by obtaining a direction vector using $pos = m2_{pos} - m1_{pos}$ and then converting to an angle using $angle = atan2(pos.y, pos.x)$. The angle is converted to degrees out of preference. The texture coordinate u position, tex_u is then set to $angle/360$ to scale the value within the range of $0 \leq tex_u \leq 1$.

2.3.4 Combat Stage Relative Transforms

In the combat stage objects must continue heading in the direction they are launched at. This is done by parenting all objects to the first marker. When the player launches a projectile or an enemy spawns, it's velocity is calculated using transforms in global space converted to local transforms relative to the first marker. This is done using the MarkerTracking GetRelativeLocalTransform function. Since the objects final transform will then continue to be calculated relative to the first marker due to parenting, this is the only step necessary to avoid transforming these objects when moving the PSVita.

2.4 User Guide

In both modes, select may be pressed to reinitialize the camera feed.

2.4.1 Editing Stage

The application begins in object editing mode. This mode is controlled mostly by moving the two markers. Figure 2 shows a diagram of how the properties of the object on marker 1 changes depending on the implemented object-transform relationships. The inner circle shows how the object type is dependent on the yaw of marker 1. The outer circle shows how the player object will change scale or colour based on the position of marker 2 relative to marker 1. The enemy object can be randomized by pressing ○ (circle). To attempt to move to the combat stage, ✕ (cross), □ (square) or △(triangle) can be pressed. If the objects match (equal type and colour and within ±0.2 scale) combat begins. Holding left shoulder skips checking if the objects match. This was originally for debugging but left in for convenience.

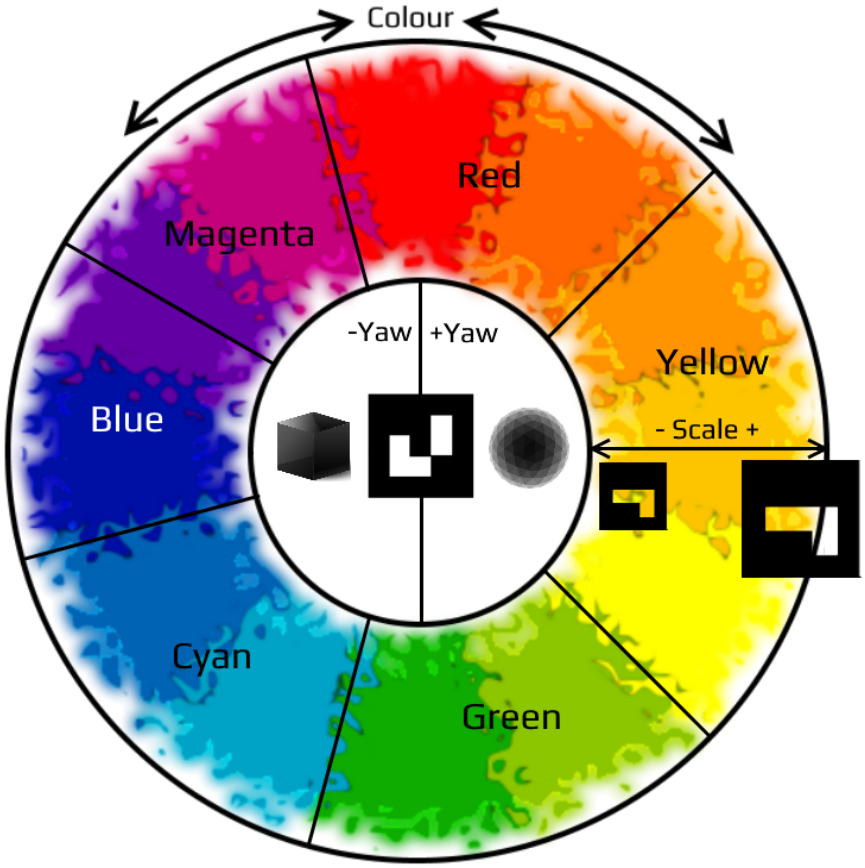

Figure 2: Diagram of Object-Transform Relationships

2.4.2 Combat Stage

In the combat stage the objects are all a constant scale of $\frac{1}{2}$ the default scale. Normal sized objects felt too big. Enemy objects are launched towards the camera in waves at constant intervals depending on various difficulty values.

- The player may fire projectiles from their left or right towards where they are aiming by pressing L (Left shoulder) or R (Right shoulder) buttons respectively. If a player object collides with an enemy object, the enemy count decreases by 1 and both objects are destroyed.
- If the player collides with an enemy object, the player loses and may return to editing by pressing □ (cross).
- If the enemy count reaches 0, the player wins and may return to editing by pressing □ (cross).
- Player objects are automatically destroyed when they move past the position of the first marker relative to the camera.
- Enemy objects are automatically destroyed when they move past the camera. (Does not reduce enemy count)

2.4.3 Difficulty Modes

The combat stage has three difficulty modes. The difficulty depends on which button was pressed to move to the stage:

Mode	Button	Enemies per wave	Enemy velocity	Enemy spawn offset	Enemy aim offset (accuracy)	Enemy spawn rate	Enemy count
Normal	▯	1-4	0.3m/s	±0.15 x & y	±0.08 x, y & z	0.5s	20
Hard	□	1-5	0.6m/s	±0.25 x & y	±0.04 x, y & z	0.33s	30
Transcendent	△	1-15	1.0m/s	±0.3 x & y	±0.12 x, y & z	0.2s	50

- Enemies per wave - amount of potential enemies that spawn every time the spawn interval passes.
- Enemy velocity - the speed enemies travel at.
- Enemy spawn offset - the randomized x and y offset that enemies spawn at relative to the marker position.
- Enemy aim offset - the randomized x, y and z offset that enemies aim towards relative to the camera when spawning. Note that the table values are correct. Transcendent is exceptionally difficult as the sheer number of enemies overwhelms the player, the inaccuracy is intended.
- Enemy spawn rate - how often waves of enemies spawn.
- Enemy count - how many enemies the player must kill to win.

3. Appraisal

The prototype application produced shows use of the potential for innovation using augmented reality technology. The game makes use of object-transform relationships to allow the creation of natural interactions between the player and the game which would be inconvenient without the use of augmented reality. A touchscreen could certainly be used to drag around the objects or scale them, just how many smartphones and tablet devices work currently, however touchscreens do not naturally allow for interactions in 3D. Similar interactions in 3D could be reproduced with special controllers such as the Oculus Touch (2016) but one could argue that interpreting hands using a camera is a more natural form of interaction than using an additional device.

The prototype developed is somewhat constrained by the use of markers. Markerless AR would be far more desirable than using markers for the developed game. If hands or fingers could be detected, all sorts of gestures could be used to manipulate objects using object-transform relationships. For example scaling can be done by moving both hands or two fingers apart. Moving objects could be done by pinching fingers and moving a hand such as showcased in the Microsoft HoloLens Minecraft Demo (2015). Changing colour could be done in any number of ways, e.g. rotating a hand in a claw shape as if to grab and rotate a colour wheel.

The combat stage of the game developed game shows how the PSVita and its camera be used as an object in the game world itself. Simply moving the PSVita is a way to control the player in the game. The PSVita also acts as the player's weapon since the player can shoot projectiles from its current position. Rather than just being a controller to allow playing the game, the controller becomes a part of the game world as a necessary tool to participate rather than an obstacle, just like many activities in the real world, e.g. a sword for sword fighting.

3.1 Potential Applications

The developed application only shows a small glimpse of potential applications for both object-transform relationships and augmented reality in general. By definition, augmented reality allows experiencing the real world with enhancements. Many current AR applications simply add to a display on top of a camera feed at positions determined by detected features like markers or QR codes. Using object-transform relationships can improve the clarity or convenience of the objects displayed in an augmented reality application. Imagine a board of advertisements where the advertisements scale up the closer you point the camera at them or change entirely depending on the viewing angle.

More specialized applications could exist as well. When markerless augmented reality becomes effective enough to accurately discern finger transforms it would be reasonable to propose ideas such as 3D model editing software where the models are projected onto a workstation in front of a user and interacted with as if the model was really there.

Object-transform relationships can potentially be applied in any situation where an interaction between two objects can be expressed by comparing their transforms. The simplest of these relationships are where one object controls another however this is likely only the beginning of what is possible. Altering object perception based on transforms such as what was done for changing object type in the game developed is another plausible use.

In terms of games, augmented reality itself appears to be well suited to being used to enhance the immersion of genres such as card games, tabletop games and strategy games where objects can be projected onto a tabletop surface. The combat stage of the developed application can be considered an FPS, which works particularly well since the camera can be used as the players perspective. AR is also suited to other games which involve adding to the player's perception of reality, e.g. alien invasion games, or player versus player networked games such as online FPS games are another candidate for AR with something like smartphone laser tag coming to mind.

However, AR is not applicable to all genres of games. Games which fundamentally replace reality or want to detract from it somehow are likely best left to VR implementations. This includes full immersion games such as space simulators and MMORPGs. It’s also hard to imagine that horror games or other games which rely on seamless immersion could be effective on current AR technology although this will likely change in the future as AR devices become more effective and realistic.

4. Conclusion

The project is considered a success as an application was developed which utilized the potential of augmented reality in a way that enhances the application. Object-transform relationships were used to create what could be considered - subjectively - natural feeling controls. Several potential applications of using these relationships with augmented reality have been discussed in addition to discussion of augmented reality as a technology itself.

If one major thing has been learned it is that innovation is difficult. Original ideas seem to be an impossibility stumbled on only by a select few. That said, there is always room for taking a different perspective on existing ideas and applications which can lead to interesting new ideas or simply new variations of old ideas. A famous quote by Sir Isaac Newton goes “if I have seen further it is by standing on the shoulders of giants”.

5. References

Antichamber. 2013. [computer game]. Microsoft Windows, Linux, Mac OS. Bruce, A.

For The Replay. 2015. *Hololens Minecraft Live demo E3 2015 Microsoft Xbox Conference*. [online] Available from: https://www.youtube.com/watch?v=mMHyh6DumaQ [Accessed 25th April 2016].

HTC Vive. 2016. [online]. Available from: https://www.htcvive.com/uk/ [Accessed 25th April 2016].

Microsoft HoloLens. 2016. [online]. Available from: https://www.microsoft.com/microsoft-hololens/en-us [Accessed 25th April 2016].

Oculus Rift. 2016. [online]. Available from: https://www.oculus.com/en-us/rift/ [Accessed 25th April 2016].

Oculus Touch. 2016. [online]. Available from: https://www.oculus.com/en-us/touch/ [Accessed 25th April 2016].

PulzAR. 2012. [computer game]. PlayStation Vita. XDev Studio Europe.