Thomas Stacey 3146

THE ROYAL LATIN SCHOOL  52131

H446 COMPONENT 3: VR Modelling Project

Contents

[Introduction 1](#_Toc499021043)

[Analysis 1](#_Toc499021044)

[Computational Methods 1](#_Toc499021045)

[Visualisation 1](#_Toc499021046)

[Abstraction 1](#_Toc499021047)

[Definite Inputs 1](#_Toc499021048)

[Stakeholders 1](#_Toc499021049)

[Existing Solutions 1](#_Toc499021050)

[Google Tilt Brush 2](#_Toc499021051)

[Mesh Maker VR 2](#_Toc499021052)

[My Approach 2](#_Toc499021053)

[Essential Features 3](#_Toc499021054)

[Limitations 3](#_Toc499021055)

[Requirements 3](#_Toc499021056)

[Success Criteria 4](#_Toc499021057)

[Design 4](#_Toc499021058)

[Breaking down the Problem 4](#_Toc499021059)

[VR 4](#_Toc499021060)

[Mesh 4](#_Toc499021061)

[Player 5](#_Toc499021062)

[Menu 5](#_Toc499021063)

[Grouping 5](#_Toc499021064)

[Development 6](#_Toc499021065)

[Initial Development 6](#_Toc499021066)

[Setting up the Workspace 6](#_Toc499021067)

[Creating the Offhand UI 6](#_Toc499021068)

[Drawing a Line for the Main Controller 7](#_Toc499021069)

[The Event System 8](#_Toc499021070)

[Spawning Primitives 8](#_Toc499021071)

[Changing the Button Script 9](#_Toc499021072)

[Adding a Preview 9](#_Toc499021073)

[Colours 10](#_Toc499021074)

# Introduction

For my H446 Component 3 project, I will be designing, making and evaluating a VR Modelling program, which will be able to create and edit models inside of a virtual reality environment, making it much easier for the user to visualise what they are creating. It will also be able to import and export these models to be compatible with Unity, allowing game creators to easily use them in their games. I will be creating this in Unity 3D, which will allow me to compile to multiple different operating systems.

# Analysis

## Computational Methods

This problem can be solved via computational methods, as it allows easy editing of models, which could not be done in the real world, as any mistakes would be final if the product were a physical object. This allows for quick revisions of models, making workflow easier. It also allows for the easy use of these models in games and other programs made with Unity, which has to be done computationally.

### Visualisation

This project heavily uses the concept of visualisation, as it allows for the user to see their models in the physical world, allowing them to better see them. This means that the model can be viewed easier, meaning that the visualisation is a success.

### Abstraction

This project will also make heavy use of abstraction, as it will allow the user to edit their models without having to know how exactly they are created. Whilst this ability will be available to users, it is not always enabled, allowing for the user to better create their models without having to worry about how they are rendered and the mesh layout.

### Definite Inputs

As there are only a certain number of actions the user can perform on their model, this makes it applicable to being solved via computational methods. This is because there is a limited way that the model can be changed by the user, which revolve around pressing buttons on the controllers and moving them around.

## Stakeholders

The stakeholders for my project are game creators such as I who want an easy way to create models for their games, and view them in the real world, and who also have access to a virtual reality headset. This is a gap in the market which is currently not filled by anything, and as such, my program will be usable by these people. Other potential stakeholders include 3D modellers for other purposes, and designers who wish to use VR for their projects, however they are not the primary focus of the project.

I will use focus groups to allow me to gain feedback on this project and make changes according to their needs. This will allow me to better refine my project to what is required by game designers.

## Existing Solutions

There are few existing solutions to my project, which I will be drawing inspiration from in order to make my software as user friendly as possible. The programs which I have chosen to analyse are Google Tilt Brush and Mesh Maker VR, as I feel that these are the closest in scope to what my project is aiming to be.

### https://i.ytimg.com/vi/TckqNdrdbgk/maxresdefault.jpgGoogle Tilt Brush

Google Tilt Brush is a VR art program, which allows the user to create drawings in 3D space. It has simple, user friendly controls, which I aim to emulate in my project as much as possible. For example, triggers are used to draw lines in the world, which follow the paths set by the user.

Tilt Brush also has a number of utility features I will be implementing in my project. These include the mirror tool, the ability to change the colour of lines without having to replace them, world scaling using the grips and a toolbar around the offhand controller. This means that the user can use one hand to paint, and the other as the palate, which allows them to easily change the colour of the paint or the tool they are using without having to bring up a menu.

However, Tilt Brush has features that can be improved upon in order to create a model editor. For example, it can only draw lines, not create 3d shapes, making it harder to create 3D models, as the user must use another program in order to turn the lines into shapes. Its eraser tool also removes the entire line, which may not be what the user wants in some cases, so I will create a tool which can remove certain pieces, as well as the full shape.

### https://i.ytimg.com/vi/JSs2y5CWhEE/maxresdefault.jpgMesh Maker VR

Mesh Maker VR is a program which allows the editing of meshes in VR. It is a powerful tool, which allows the user to dynamically change meshes inside a virtual environment, allowing them to make models easily.

Mesh Maker VR has a number of features that I am going to implement into my project. These include the ability to edit the triangles of the mesh, prefab objects which can be spawned into the world, and dynamic creation of objects using movable points.

However, the file handling system of Mesh Maker VR is unintuitive to use, and I will therefore be replacing it with my own version. It also does not have the ability to export models directly as Unity prefabs, meaning that the user has to import and re-implement any materials they may have applied to the object. It also does not make use of the default VR keyboard, making file names tedious to type out with a custom interface.

### My Approach

My approach to the problem will take inspiration from both of the existing solutions I have analysed, and I will take the best features from each to use in my own. The ease of editing models in Mesh Maker VR means that I will be using the ‘dot vertex’ system, to allow the user to manipulate each corner of the polygons with ease. However, I will be using the simple, intuitive controls of Google Tilt Brush, such as the toolbar around the offhand controller, and the simpler menus, as well as the teleportation mechanic to travel around the scene. These controls are common across many VR applications, meaning the user will not have to relearn basic controls when they open the application, allowing for a more seamless experience.

I am using an object oriented approach as each model can be modelled as a single object in the world, allowing for a single base class to control the behaviours of many objects. The objects also consist of many components, such as the mesh, each of which can be modelled as their own class and interact with the base object class. This means that an object oriented approach is suitable for this project.

C# is a valid language for this project due to the requirements for an object oriented approach and the time constraints as I already have knowledge of the language. For this reason, I am also pairing it with Unity as this provides the base libraries for rendering and the underlying structure, meaning that I do not need to implement features that have been done many times before, allowing me to focus more on the unique features of my project.

The iterative approach allows me to create systems which work and then incrementally improve on them at a later date. This means that I can test improvements in context of other systems, allowing for the solution to work better together, and it also allows me to test whether improvements to modules are required, meaning I can focus on adding new systems and updating systems that need it.

## Essential Features

|  |  |
| --- | --- |
| Feature | Justification |
| Able to spawn prefab meshes | Allows the user to more easily create complex objects, allowing for simple creations |
| User can edit existing objects | Allows the user to make changes to their existing work |
| Can load and save objects | Allows the user to work on a project over multiple different sessions, and also allows the user to use their work in different programs. |
| Can undo and redo | Allows the user to undo their mistakes |
| Can change the material of an object | Allows the user to texture their work |

## Limitations

Due requirements of Unity, there will be a minimum specification that computers will need to have in order to run the application at an acceptable framerate. This is especially true in VR, where the user will need a graphics card capable of handling rendering two screens at 90Hz refresh rate. I will lower these requirements as much as possible through efficiency savings, but the time available and my knowledge of the inner workings of C# and Unity may mean that the project will be unable to run on extremely low specification machines. The time constrains may also mean I am unable to implement all of my non-essential features, but these are not required, and so should not damage the final product.

A limited audience for testing may also mean that the final product may not appeal to mainstream audiences, but by selecting testers from a wide variety of people, I hope to reduce this limitation as much as possible.

## Requirements

The hardware requirements of this project include a computer of sufficient computing power to obtain an acceptable framerate running Windows, Mac OSX or a Linux distribution. The system will need adequate secondary storage space to hold the project data and a VR headset, controllers and base stations to allow them to run the application.

## Success Criteria

* TODO

The project will be successful when…

Table

Success criteria / how it will be measured

# Design

## Breaking down the Problem

This project can be broken down into different modules, each with a purpose, which will allow code to be implemented at different times, and allow the replacement of modules with improvements without breaking existing code. This also means that modules can be tested separately, removing the need for redundant tests if no code has changed.

### VR

The VR module will be responsible for the basic interface with SteamVR, which allows for the game to be run on Oculus Rift and HTC Vive headsets, among others. This will mostly be done through the SteamVR API, but the VR module will interface with this code in order to create the experience.

### Mesh

The Mesh module will control the rendering of the models as meshes. It will need to separate each polygon into triangles, and then render them for the user to be able to see.

#### Triangulation Algorithm

FUNCTION Triangulate(vertices[])

IF vertices.Length == 3 THEN

triangles = vertices

ELSE IF vertices.Length == 4 THEN

triangles.Add(vertices[0],vertices[1],vertices[2])

triangles.Add(vertices[2],vertices[3],vertices[0])

ELSE

centre = Average(vertices[])

FOR i=0 TO vertices.Length – 2 DO

triangles.Add(centre, vertices[i], vertices[i+1])

ENDFOR

triangles.Add(vertices[vertices.Length-2], vertices[verticies.Length-1], vertices[0])

ENDIF

RETURN triangles

ENDFUNCTION

This algorithm takes an array of vertices, which represents a 2D shape and turns each one into a set of triangles. With the exception of triangles (which should just return the triangle) and squares (which can be split into 2 triangles instead of 4), it gets the midpoint of the shape and takes triangles from that point to the vertices. Whilst this is not the most efficient algorithm, and only works for convex shapes (shapes with all angles less than 180 degrees), it is the simplest, requiring no matrix algebra to work.

### Player

The player module will be responsible for handling the players actions in the world and passing these onto the respective classes which handle said actions. It will listen for events such as the player pulling the trigger on their controller, determine the appropriate action, and execute it from the respective class.

### Menu

The menu module will be responsible for handling the menus in the game, both around the player’s offhand, and any menus that can be placed in the world. Each menu item should perform an action, and at a minimum, it should have controls for saving, loading, changing the colour of spawned polygons, and spawning shapes.

### Grouping

The grouping module will handle the grouping of objects into groups. This will then allow the user to interact with multiple objects as if they were just one, allowing them to more easily manipulate them. All objects will also be in a ‘base’ group, which will allow moving and scaling the entire project around the user.

#### Grouping Algorithms

Dictionary<Group, List<Transform>> groups

FUNCTION AddToGroup(Group group, Transform trans)

trans.parent = this.transform

groups[group].Add(trans)

ENDFUNCTION

FUNCTION RemoveFromGroup(Transform trans)

trans.parent = defaultParent

groups[group].Remove(trans)

IF groups[group].Length == 0 THEN

groups.Remove(group)

ENDIF

ENDFUNCTION

FUNCTION CreateGroup(string groupName)

GameObject go = new GameObject

go.name = groupName

go.AddComponent<Group>

groups.Add(group, new List<Transform>)

ENDFUNCTION

These three algorithms describe how to create groups, and add and remove items to and from them. In order to keep track of groups, I will use a dictionary of lists, as this allows me to reference the groups as lists, as each key corresponds to the object which is the parent of the group, and the individual items in the group, which are obtained by finding it in the list

# Development

My development will be split into phases, with each phase adding more features as time progresses. After each phase I will test the entire project with my target audience in order to gain feedback so that I can make changes to the design to tailor the game to them.

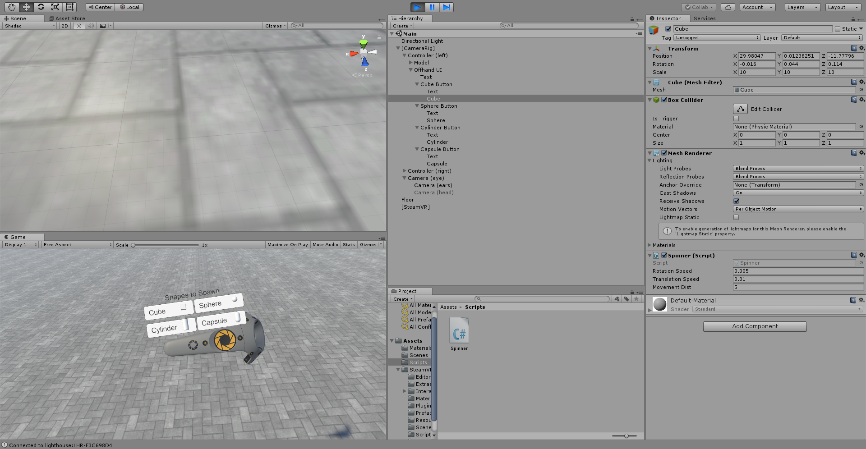
## Initial Development

My initial development phase will add the base features required to create a working solution. This will allow for the project to be able to work, and be semi usable, if a bit simplistic by the end of this phase.

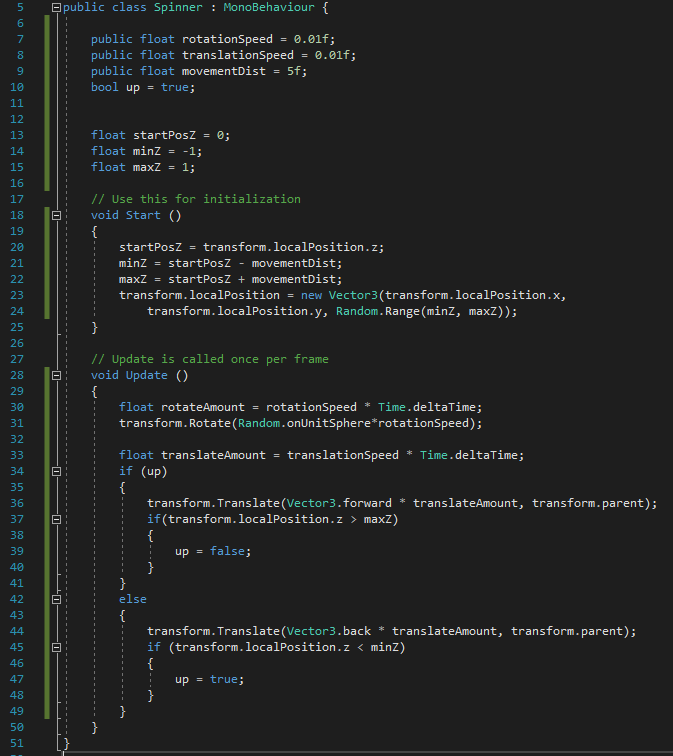
### Setting up the Workspace

To begin this project, I needed to setup the workspace and import the Steam VR Package, allowing for my game to work in a VR World. I then loaded the base VR Camera Rig, which allows for the headset and controllers to appear within the game world. Once I had done this, I created a plane for the floor, so the user was more comfortable, and added a texture to make it visible.

### Creating the Offhand UI

The next step was to create a UI around the offhand. This would be parented to the left controller, and would serve as the main way to control how the user is interacting with the scene. To begin, I created a canvas with some text on it, and parented it to the left controller game object. This presented me with some scaling issues, as the text was extremely large, requiring me to reduce the scale to 0.001 to have a normal size. I then rotated the canvas 90 degrees about the X axis and -90 degrees about the Y, and transformed it to it rested above the controller, in line with the edge, so that the player could hold their hand to look at it as if it were a watch.

Next I created 4 Buttons, one each for a cube, a sphere, a cylinder and a capsule, and laid them out below the text.

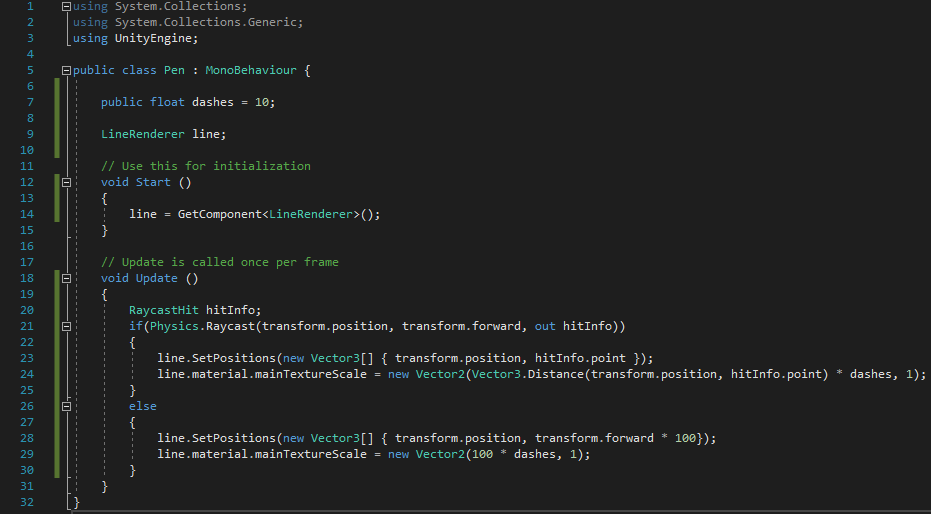
I created a spinner script and attached it to each of the previews shapes so that they would move up and down and rotate, allowing them to be more dynamic to the user, and therefore more interesting.

The spinner script is fairly simple – when the game is started, it calculates the minimum and maximum positions it should reach, then each frame it moves towards one of them at a constant speed. When it reaches the minimum or maximum value, it toggles the ‘up’ Boolean, and so reverses its direction for the next frame.

It also rotates a random rotation, which is a point on the unit sphere. This means that the speed of rotation should be constant.

### Drawing a Line for the Main Controller

In order to see what the main controller is pointing at, I added a line renderer and a corresponding script to cause the line to stop when it collides with an object.

I also added a dashed material, and added to the script so that the length of the dashes remains constant as the length of the line changes.

The script casts a ray along the forward axis of the controller, and returns information about the object it hits (if it does so). I then used this point, and drew a line inbetween the two with SetPositions. In the event the line does not hit anything, I instead draw the line for 100 units in the direction the controller is facing.

In order to keep the lengths of the dashes the same, I modify the mainTextureScale, which is the number of times it tiles, and set it to tile the same amount as the length of the line, multiplied by a constant.

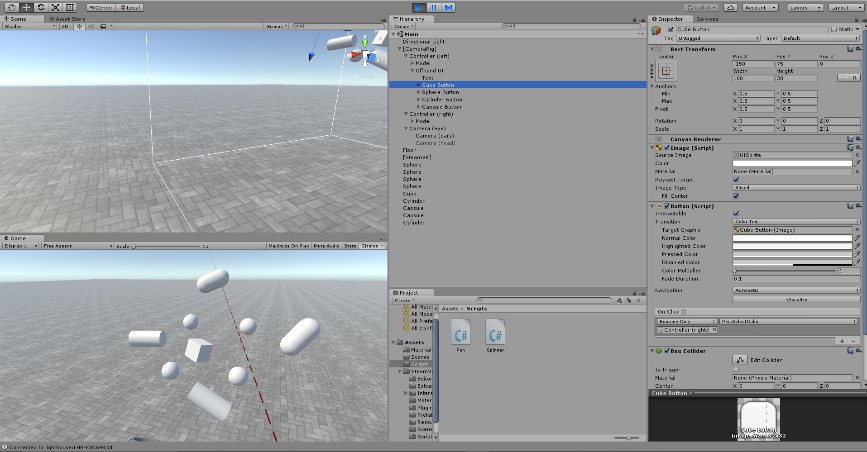
### The Event System

In order to tell whether the controller buttons are pressed down, I need to use the SteamVR event system. To do this, I first added TrackedController scripts to both objects, which allows them to create events, which I can subscribe to through other scripts. In order to do this, I simply add my function to the correct event on the controller script, and SteamVR will trigger that function whenever the trigger is clicked. I also need to unsubscribe from these events when the controller gets disabled, so that it does not keep on influencing the world.

### Spawning Primitives

To test that everything is working up to this point, I am going to test by spawning the selected primitive using the touchpad. A primitive is one of the base unity objects, and since my four default objects are primitives, it makes sense to test spawning them.

In order to spawn the primitives, I need to keep track of which primitive I am spawning. As Unity has an enum of these, I can simply create an instance of that enum to define the current primitive.

In order to press the buttons, I am using button.OnClick.Invoke, as it allows me to specify the actions of the buttons in the editor, and it is the simplest way to activate the button. In the editor, I assigned each button’s onClick event to the function corresponding to selecting the correct primitive in the Pen class.

When the touchpad is pressed, it spawns the selected primitive, meaning that all the code so far works, and has been validated.

### Changing the Button Script

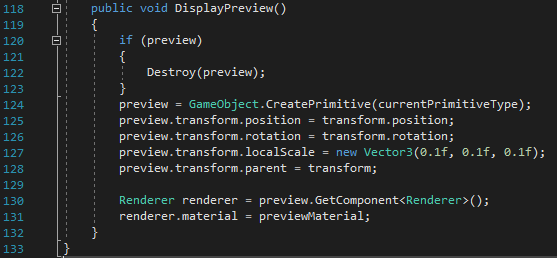
The current Unity button script, while useful for most desktop applications, does not work in VR. In order to remedy this, I created a new script to replace it called VRButton, and extended it with a class called PrimitiveButton. This will allow me to later create buttons that have different actions to spawning primitives that exhibit the same behaviour of highlighting when selected.

Creating the VR button is as simple as creating a script with two methods, one for selection, which highlights the button, and one for deselection, which resets the colour back to the default. I also defined two colours, which default to white and red, but can be changed in the editor at a later time.

PrimitiveButton is a simple extension of the VRButton class, which adds one new method – SelectType, which returns the type of the button, as selected in the editor. This then allows me to call that method in the main script and change the preview and spawned objects accordingly.

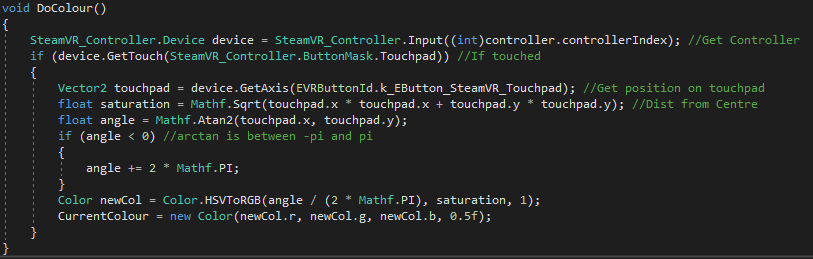
### Adding a Preview

To allow the user to see what they are doing, I added a preview of the object before they spawn. This is done by creating a copy of the object attached to the main hand of the player, and using a separate material to make it appear translucent.

I also need to apply the same transformations that I apply when I spawn the transforms, so that the preview will look the same as the spawned objects. Finally, I need to destroy any previous instances of the preview, so that they do not stack up, causing multiple objects to appear on the hand at the same time.

### Colours

The next major feature I added to the program is the ability to colour the primitives they place in the world. To select the colours, I used input from the touchpad on the right hand controller to make it a colour wheel.

As the touchpad is a circle, using the HSV colour space with constant value allows for each point inside the circle can be mapped to a specific colour, with the distance from the centre representing saturations (with the centre being white) and the angle around representing hue, with the top being red. Distance can be calcuated with Pythagorus’s theorum, .

The angle is slightly more complicated, returns a value in the range , but an angle of 0 radians needs to map to a hue of 0. To solve this, if the angle is below 0, I add , to loop it round, so that will map to , allowing the circle to complete. This means that colours with hues greater tha 0.5 (blues and purples) will be selectable on the wheel. Finally, I need to divide this angle by , so that it will map into the range , which is the range which hues map to. I then need to convert the angle to RGB so it can be handled by Unity, which I use the inbuilt function HSVToRGB to do, and then return that colour. When I then change the colour of objects and the preview, I only modify the R, G and B values, not the A, allowing for transparency to be preserved.

To constantly update the X and Y positions of the player’s thumb on the touchpad, I tried two methods before settling on a third. The first method, InvokeRepeating, did not take arguments for colour, and also used strings for the function name, so at first I discounted it. However, when I implemented a Coroutine, I realised that the X and Y positions would not be updated unless the touch event was triggered again, and so I would have to use another method to get these values. I therefore decided to revert to InvokeRepeating, as the Coroutine unnecessarily limited my function, and then used the SteamVR API to select the controller and therefore the position the touchpad was touched. I also needed to check that the touchpad was, infact, being touched before changing the colour, otherwise the coordinates would revert to (0,0), causing the colour to change back to white when the user stopped touching the pad. Since I was now checking if the touchpad was touched anyway, using events to check this too was pointless, and as InvokeRepeating required a specific time that I could not sync to the framerate, I decided to run the function during the Update cycle, which would allow for the colour to change as soon as the player touched the pad.