**Tools Programming Report**

**CMP405 : Tools Programming**

**1302869 – Callum Henderson**

# Introduction

For the implementation of tools for this module, the features added includes:

* The ability to pick select an object on screen
* manipulate the object position in the X, Y and Z axis using the mouse cursor
* Having an object move to the terrain position under the mouse cursor
* change the scale of an object using the mouse cursor
* User interface for altering the manipulation mode.
* Toggles wireframe mode for the terrain

The picking allows the user to select any object that is on screen by clicking on its model. This is a single object selection in that multiple objects cannot be selected and modified at once. The picking will be disabled should the user switch to a mode of manipulation via the toolbar interface.

Another feature implemented is the ability to manipulate an objects position using the mouse cursor. The X and Z movement will be discussed in the next paragraph however the Y position is manipulated using the difference in mouse position (delta mouse).

The X and Z plane of an object is manipulated by hovering the mouse cursor over a position on the terrain and then clicking. This will then move the selected object to that selected grid. Sometimes it may be necessary to shake the mouse on the terrain before the object will translate.

The scale of an object may be manipulated by moving the mouse to the left and right. Moving the mouse to the right will increase the object’s scale while moving to the left will decrease the object’s scale.

The current selection and current tool may be cleared using the CL button. This will allow the user to select a new object using the picking interface.

The terrain may also be toggled between opaque and wireframe mode using the toggle button. This will present a useful reference for the grid based movement system used by the X, Z Translations.

The user is able to change the mode of manipulation by selecting the mode on the toolbar at the top of the screen. The first four buttons relate to manipulating the translation of a selected object. The functions are as follows.



 Locks the translation to the X axis. Selecting a positon on the terrain will move the selected object to the positions X coordinate.

 Locks the translation to the Y axis. Moving the mouse left and right will alter the selected objects Y transform.

 Locks the transform to the Z axis. Selecting a position on the terrain will move the selected object to the positions Z coordinate.

 Locks the transform to the X AND Z axis. Selecting a position on the terrain will move the selected object to that position’s X, Z coordinate. The height is not translated.

 Locks the transformation and enables the scaling of the selected object. The scaling will be dictated by mouse movement. Left movement will scale down while right will scale up.

Clears the selected tool and selection. Allows the user to pick a new object for selection.



 Toggles wireframe mode for the terrain. Useful for placing objects within a specific grid.

# Methodology

The core design of the applications was intended to mimic the functionality of a typical mission editor such as the world editor from Command and Conquer Generals (Electronic Arts, 2003) or game engine object manipulation such as Unity (Unity Technologies, 2005). This includes the ability to manipulate an object’s location as well as its scale using the tools provided by the application.

## Picking

The first aspect to be discussed is the object translation system. In order to begin with this, first it was required that we were able to calculate the world position of where the mouse cursor is relative to the terrain. All object positions will be placed relative to the terrain so it would make sense that the mouse world positions would be calculated relative to it. With the use of the tutorial from the Microsoft MSDN (Microsoft, 2011) detailing the methodology of implementing selection of objects in 3D space using the mouse. This tutorial meant that it would make more sense to begin by implementing this system for picking objects first before attempting to reuse it for calculating positions on the terrain.

Due to the tutorial being made for XNA (Microsoft, 2006), some alterations had to be made to this tutorial in order to have it work with DirectX 11. First the screen mouse position was taken and used to calculate a near and far plane. Next a matrix is created for each of the display list objects and is used with the world matrix in the unproject function with DirectX. Using our near and far sources calculated using the mouse co-ordinates, a near and far point is created between screen space and object. A direction is then calculated by obtaining the vector between far and near points. Each object’s bounding box is then used to check for intersections using the newly calculated ray. Should the check return a positive, the object is then marked as selected and its object ID is then returned as the picked object, which in turn is made the selected object. The distance is then updated to be the maximum value allowed to prevent any further objects that may be directly behind the first object to replace it as the picked object.

## Mouse World Coordinates

With the algorithm already created for selecting objects, the ray cast system was further used in the calculation of the world position of the mouse cursor. First the terrain data is retrieved from the m\_displaychunk object in which a getter was implemented to retrieve the terrain positional data from the DisplayChunk class. Once the terrain positions have been retrieved, it is then given into the ray cast function. Next, each tile is iterated through and temporarily assigned a bounding box with its position and size set to the same as the terrain tile. The bounding box is check for intersections with the ray cast and if so, returns the tile that the ray cast has hit. The co-ordinates are then taken from the hit tile and passed into the world mouse position variables for use with the manipulation system.

## Object Manipulation

Object manipulation is carried out via a state based system. When the corresponding icon is selected in the toolbar, the state is activated and the object may now be manipulated in that manner. Currently there are 5 modes of manipulation; X axis, Y axis, Z axis, X/Z axis and Scaling. The position manipulation apart from the Y manipulation all make use of the world mouse position discussed previously. Once one of the modes previously mentioned has been selected, the user may then click on a point on the terrain and the object will move to that position. The user may even drag the mouse over the terrain and the object will follow the mouse cursor across the terrain. Should the user move over terrain that has elevation applied to it, the object will inherit the height value of the terrain, thus matching the elevation. If the user wishes to specify a custom Y value for an object, the user may use the Y manipulation functionality which uses the mouse screen position as input for the scaling. Y position is dictated by the amount of movement the user makes using the mouse. If the user moves the mouse right with the left mouse button held down, the object will move up the Y axis while moving to the right will cause the object to move down the Y axis. The scaling tool uses the same functionality in that scaling will be applied as a factor of the mouse delta position on the screen. For example, if the mouse is moved to the right with the left mouse button held down, the object will scale up. If the mouse is moved to the left, the object will then scale down. The purpose of the X, Z and X/Z buttons is to lock the manipulation of those axis to the corresponding axis. This allows the user to adjust and objects X position without moving it from the Z position and vice versa. The X / Z button will allow the user to move the object free form around the terrain.

To cancel the current manipulation state, the user must use the CL button to clear the current manipulation state. This will allow the user to select a new object for manipulation.

# Discussion

## 3.1 Camera system

The camera system implemented is a simple first person floating camera that makes use of the right mouse button to enable camera controls. When the camera was first being implemented, an issue was encountered in that mouse input was only calculated when there was input. The issue with this was the method that was being used to calculate the mouse delta position which was an integral part of manipulating the camera. The mouse delta was calculated inside of the ToolMain class, more specifically alongside the code that fetched the mouse co-ordinates from the MFC framework. The issue with this was that the mouse delta value was updated only when the mouse was moving, which while sound like a non-issue, caused the mouse delta to permanently return the last value after the mouse had stopped moving.

This could be triggered by moving the mouse vigorously and abruptly stopping would cause the class to stop updating as there was no mouse input, however it could not access this same class to update the previous mouse position to the same as the current. This would cause the camera to continuously rotate in the direction the mouse was last moved in. This was resolved by having the mouse delta calculation moved to the game update function inside the game class file. Here the mouse coordinates were passed into the game class update and the mouse delta then calculated.

## Picking

The picking system itself worked as intended after some trial and error with the ray casting code, this was to be expected as this was the first time attempting to implement ray casting in any capacity. The first real problem occurred when attempting to have the selection system work in tandem with the older menu based selection system. The original menu based selection system assigned each array entry of the display objects to a new menu entry. This however had caused the problem in that MFC does not start a list at value 0 and thus, object 0 on the objects array could not be selected using the menu. This also cause a far greater problem in that if value 15, which was the size of the object list, was selected on the menu, it would attempt to access out of scope of the array. This had to be resolved by ensuring that the object list being given to the menu system was shifted up by one place. This was done by altering the selection code of the list to have the list number select an object that was one place below the current value of the list. This for instance caused value 1 on the list to select object 0 on the display objects array and thus solved the out of range array error when list value 15 was selected as this would now select object 14 in the display objects array.

## World Mouse Coordinates

The problem encountered when attempting to retrieve the world coordinates of the mouse was that the value returned was magnified by two places. For instance, if the mouse was over terrain grid (0,0) the correct grid would be selected, however if the mouse was over grid (10, 10) the grid triggering the ray cast would be grid (5, 5). This was originally believed to be an issue with the collision box used to check for intersections with the ray cast however, after a large amount of testing and altering of values this did not turn out to be the case. The next path of investigation lay within the near and far point projections as this lead to the suspect that the terrain was projected to be twice the size than what it actually was. This however, was also not the issue however did yield some close fixes. After re consulting the XNA tutorial it was discovered than the tutorial had made use of a world matrix while the system that was implemented made use of a world multiplied by local system.

This was because the ray casting code was copied from the picking function in which this matrix calculation was actually necessary. A blank identity matrix was provided to the near and far point calculation and this resulted in the ray cast returning accurate his on the terrain. It is speculated that since the local matrix being used to multiply the world matrix contained the transform of the target cell, this would offset each tiles position by its own position. This made sense, as was discussed in the example, attempting to select tile (20, 20) for instance would have the tile (10, 10) be selected instead leading to the speculation that tile (10, 10) had moved by its own coordinate to position (20, 20) on the terrain and tile (20, 20) would by extension be located at (40, 40) which turned out to be the case.

# Conclusion

The tool created delivers a great deal more usability in the form of being able to position an object by simply clicking and dragging or clicking and pointing to a desired position in 3D space. Any fine manipulation may be performed by using the locking functionality of the toolbar as well as allowing the user to alter the scale of an object on the spot with a single tool. The user may also click on any object to on the scene and quickly begin manipulating this object with a single button press. This in contrast to the menu based selection is faster and more intuitive to the user. Methods of improving upon this would be to implement a handlebar system similar to that used in game engines such as Unity as well as to have keys assigned to contest actions for tools. This would reduce the tools on the toolbar to a single button as scaling, manipulation and axis locking can all be handled by keyboard context buttons and UI elements added to the object being manipulated. Another aspect that could, and very much in need of improving is the ray casting method used to calculate the world coordinates of the mouse cursor. The problem with the current system is that it requires to check every tile in the terrain in order to locate the tile that the ray cast is hitting. This causes a large frame rate hit as 128 x 128 tiles are present in the terrain and each must be individually checked for intersection. A method of reducing this overhead would be the implementation of a quad tree algorithm, which would assign quadrants to each of the tiles thus reducing the amount of terrain tiles that the system must iterate through in order to find the tile being hit by the raycast. Another UI improvement would be to have a status bar displaying the currently selected tool. As of the current build, the currently selected tool is not displayed and could cause confusion if the user is attempting to perform a manipulation with the wrong tool selected or move and object will attempting to pick a new object to manipulate.

**Referances**

Electronic Arts (2003) *Command and Conquer : Generals – PC* [Video game]. Electronic Arts

Microsoft (2011) *Selecting an Object with a Mouse*. Available at: <https://docs.microsoft.com/en-us/previous-versions/windows/xna/bb203905(v=xnagamestudio.41)> (Accessed: 24 April 2017)

Unity Systems (2005) *Unity Game Engine* [Game Engine]. Unity Systems

Microsoft (2011) *Microsoft XNA Framework*. Microsoft Corporation.