SCHOOL OF COMPUTING

UNIVERSITY OF TEESSIDE

MIDDLESBROUGH

TS1 3BA

Human Computer Interaction within Industry Tools

Bsc. Computer Games Programming

Sam Oates

14 – 05 – 2013

Supervisor: Tyrone Davison

Second Reader: Suiping Zhou

SCHOOL OF COMPUTING

UNIVERSITY OF TEESSIDE

MIDDLESBROUGH

TS1 3BA

Human Computer Interaction within Industry Tools

Bsc. Computer Games Programming

Sam Oates

14 – 05 – 2013

Supervisor: Tyrone Davison

Second Reader: Suiping Zhou

ABSTRACT

This paper follows the development of a 3D computer games tool powered by a human computer interaction based device, the Microsoft Kinect.
Research was based around three fundamental areas required for the project; human computer interaction (HCI), real-time image recognition and the deformation of terrain within 3D graphics.
Using previously gained industry knowledge and details gained from my areas of research, an initial design prototype was created, followed by a small amount of user testing. Testing for ease of use, productivity and comparing against gestures natural within the real world.

ACKNOWLEDGEMENTS

I would like to thank my two managers whilst working at Blitz Games Studios. Neil Holmes and Tom Gaulton, both of which encouraged my passion towards computer games tool systems. I would especially like to thank Neil Holmes for giving thorough feedback on the project in its later stages of development. Along with this Terry Greer, a designer a met whilst working at Blitz Games Studios helped out when drafting up initial concepts and design ideas. Finally I would like to thank every person whom tested the project, both at the designated times and at random points during the development cycle.

CONTENTS

1 INTRODUCTION

1.1 Research Question

In the modern day games studio, artists and designers are often found using keyboard and mouse input to create scenes, art assets and such; for games. However, creative people have a tendency to work better with their hands. The keyboard and mouse input may limit their ability to do this. Posing the question, is current computer hardware limiting usability with its non-natural interfaces?

I aim to create a simple tool (in the form of a terrain editing system), where the input is based upon the user within their 3D environment (via the use of the Microsoft Kinect device) as well as using other inputs such as the users' voice. This creates an interface more in tune with its users' tendencies resulting in the exploration of the users' potential productivity gain and higher quality of work. Where by the main complication in implementation will be finger tracking and hand gesture recognition, due to variations in hand size and shape of different users as well as different mentalities of how the user believes the gestures should work.

1.2 Rationale for Project Choice

I have had a life-long passion for tools within computer games, trying to make interfaces and systems as simple as possible for the user to interact with. My inspiration for this project was founded whilst on work placement at 'Blitz Games Studios'. Whilst there I spent time working on their tool system, 'Blitz Tech'. Working closely with game teams and at points the Microsoft Kinect. Whilst working I noticed how the artists, designers and animators used real-life models and scenarios to compare with their plans or creations. Using pen and paper as well as other input devices such as tablets to draft work before creating the asset within a 2D or 3D graphics computer tool.

With this, I have first-hand experience of how an artist works and how a programmer creates software. However the two do not necessarily correlate due to the differences in rationale between artists and programmers. To expand on this, I have experience with user interfaces, tools, graphics/rendering and the Microsoft Kinect.

1.3 The Current State of Human Computer Interaction

Human computer interaction (HCI) is an astronomical field of ongoing research. However the majority of such research is specific towards the general user and or non-computer user, attempting to allow non-technical people to interact with computer hardware. The problem lies in extracting data from the user in a manor most natural to them and evaluating the data for use with a device, as doing such is hard to generalize. This results in software that feels natural to some and not to others.

This problem is reduced when looking into to HCI within the games industry, as we can make the assumption that the user is somewhat technically minded. Already the user should have an understanding of current HCI making use of the standard keyboard and mouse, as well as other artist specific input devices.

1.4 System Requirements

Given the problem of non-natural HCI interfaces for creative peoples, specifications for a natural HCI interface can be formed.
An artist should be able to move a gizmo (a replacement for the mouse cursor in three dimensional space) about the terrain environment using nothing but there hand. Once positioned to the users requirements, the user need only use their other hand to apply the selected brush to the terrain in an area about the gizmo.

The user should be able to change the selected brush via a graphical user interface (GUI) based menu system, via the use of voice commands and hand

gestures. The GUI menu system should also allow access to other mandatory tasks associated with a terrain based tool system. This includes creating a new default terrain, opening existing terrains and saving the currently active terrain. Along with this editing settings about the size and strength of the currently active brush should be performed via voice and gesture based commands. This does not require the user to traverse a menu system and instead should be performed at any point during runtime.

To achieve the above outcomes the following steps need to be fulfilled.

- Implement a simple C++ terrain rendering system using Microsoft's Direct3D 11.
- Allow the terrain system to be deformed via the use of the traditional keyboard and mouse.
- Consult potential users on gestures for different operations.
- Implement the first draft of gesture based commands.
- Test the first implementation with potential users.
- Improve the first draft based on feedback from initial testing
- Consult potential users on voice commands for performing different operations within the system.
- Implement the first draft of voice based commands
- Test improved gesture commands and draft voice commands with potential users.
- Finalize gestures and voice commands based upon user feedback collected during testing.

Overall, the goals for the project are as follows.

- Create a simple terrain editing system powered by a natural human based input device.
- Attempt to prove that both productivity and quality can be improved by reducing the barrier that exists between creative people and the tools existent within industry.

2 METHODOLOGY

2.1 The Methodology behind the Implementation

As the product requires feedback based upon user experience, the product has to go through multiple repeated steps of development until all discovered issues are resolved. This means should there be an unseen problem within the initial plans; it can be refactored out at a later stage of development. The Kinect device runs at a low resolution meaning sampling hand data has potential issues. The recursive development cycle can help resolve the issues should initial implementations be unsuccessful. Three public testing points are to be set, where artists, designers and other creative people will be invited to try out the project in its current development state. After each test session feedback will be collected and development tasks reassessed.

Due to the rapid changes that will take place based upon user feedback; a form of source code versioning control software is required. Whilst working in the industry we used subversion control (SVN). SVN is based around the principle of one main repository (the server) and multiple local copies (the client). A client need simply 'checkout' the latest revision from the server to create a local copy. Changes are then made to the local copy and 'committed' to the server. Each revision stores additions, deletions and changes to source files. This allows the client to revert back to a previous version of the code base, if required.

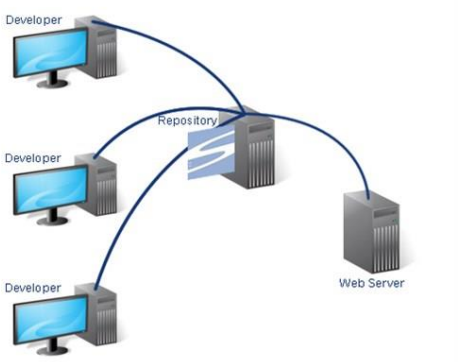


Figure 1: The server client subversion network setup

Along with this, SVN has the ability to branch and tag revisions. A branch allows a developer to work in parallel to the main repository. Commits made by the developer are made into their branch of code rather than the main repository allowing large scale features to be implemented without breaking the main repository. Once the feature has been completed the branch is merged back into the main repository. Tagging can be used to flag a given

revision. For example the version of the project used for each test process will be tagged as such. Meaning in the future returning to specific versions can be performed to compare and contrast both the source code and the features.

With each commit a comment can be entered detailing important changes and additions that occur within the commit, making finding older revisions easier to pin point. The ability to revert to previous versions of the project will help in finding and resolving many issues that may occur during development.

2.2 Design Methodology

Given that hardware tessellation is implemented within the Direct3D 11 SDK this will most likely be chosen to be the graphical API, as tessellation can be used to smooth the terrain with little overhead in performance. Due to this, implementation shall be done using the program using the C++ programming language. Not only for the easier implementation of hardware tessellation but also for the speed and performance gain which are present with the C++ programming language. The Kinect SDK (which I have previously used) also has a C++ implementation which again has performance gains over other language implementations. The speed benefits from using a low level language such as C++ allows for fast processing of the vast amounts of data that will be gathered by the Kinect camera and its microphones.

The project will be initially drafted via the use of unified modelling language (UML) diagrams. The object-orientated design of the C++ programming language allows UML to easily layout and design classes and interfaces required for the project.

2.3 Methodology of Testing

2.3.1 Static Analysis

Throughout the project not only will I test usability and features, but static analysis will be performed on the code before every commit.

Static analysis is the term used for the process of programmatically scanning source code for potential issues. To perform this task an SVN commit hook can be created to perform static analysis via a program called Cppcheck. This means, that prior to any SVN commit static analysis will be run on the added and or changed code. Cppcheck performs the following checks;

- Out of bounds checking.
- Check the code for each class.
- Checking exception safety.
- Memory leaks checking.
- Warn if obsolete functions are used.
- Check for invalid usage of STL.
- Check for uninitialized variables and unused functions.

Should there be an issue with any code a report is presented to the user and the commit is cancelled until all static analysis tests pass successfully. This will help vastly in improving the stability of the project as well as pointing out potential mistakes in logic which would previously go unseen.

2.3.2 User Based Testing

User based testing will be performed in two parts; three main testing phases and continuous testing with non-project specific users. The continuous testing will be performed whilst developing the project in the universities computer laboratories. The basic principle will be based around people's interest in the project. Given the interactivity and abstract nature of the project due to the use of the Microsoft Kinect device, other people about the computer lab will be willing to test new in-development features at arbitrary points. This will help fine tune features as well as spot issues within the design at an earlier stage of development.

Finally the three designated user test points will be used to test the current state of the project, upon the users the tool is designed for. Each test phase is to be designed to test a specific feature of the tool system. The phases are as follows;

- Basic hand gesture detection and the basics of the terrain system itself.
- Finalizing hand gesture detection, introducing voice based commands.
- Final testing of fully implemented hand gesture and voice recognition commands.

After each session feedback will be taken both verbally and in the form of a short questionnaire. The questionnaire will pinpoint areas which are new to the current test state of the project. The interesting point with the test is that, in every session the user can not only test the Kinect based input, but also all features will be implemented via the keyboard and mouse. This allows the user to properly compare the two interfaces upon the same terrain editing system.

Along with this, in the later stages of the project I shall contact my manager from Blitz Games Studios, Neil Holmes to ask for his professional opinion about the product. With the aim that the project may be passed around the office to some other professionals whom not only work on computer game tools but also artists whom use the tools themselves. Giving a real insight to whether the industry actually believes natural HCI is a possibility within industry level tool systems.

3 RESEARCH ANALYSIS

3.1 Real-Time Hand Detection

The project lies around the principle of using the Microsoft Kinect as an input device, using nothing more than the users' hand(s). Although I have previous experience with the Kinect device, hand detection lies out of the bounds of the normal Kinect and its SDK. The Kinect SDK implements full body (skeletal) tracking, but nothing for individual limbs of the human body. It also provides no built in image/object detection methods.

The Kinect device offers two types of data; a depth image and a colour image. Mathew Tang proposes the use of both sets of data to best estimate the hands existence and gesture (or shape) [1]. His technique involves cleaning up the RGB (red, green and blue colour) image, using standard image processing techniques like colour balancing and dilation/erosion, then incorporating the depth data using a simple probabilistic model. This was followed by normalizing the rotation of the estimated results. Finally three sets of features are then extracted from the data; the raw pixel estimates, a radial histogram and a modified version of the SURF (speeded up robust features [2]) descriptors. Tang had some success with his approach, though accuracy lied around the 90% threshold. Tangs technique also had the limitation of only detecting a single hand, where as I aim to support multiple hands.

Du and To [3] suggest using a different Kinect SDK from the official Microsoft SDK. They suggest using OpenNI (Open Natural Interaction), an open source alternative. However their implementation follows the same principle as Tang; however they only filter down the hand data based upon the depth data the Kinect produces. This is preceded by six steps, which are used to calculate the contour of the hand.

Firstly the image is filtered, via the means of a median filter with a sample area of 15x15 pixels. This smooth's the raw extract depth image. The second stage is to trace the edges of the filtered image, giving the rough contours of

the hand. This simplifies the complexity of the rest of the stages. Given the outline of the hand, an estimated polygon can be formed, estimating a low poly shape of the hand. Using an image library such as OpenCV (Open Source Computer Vision Library) this can be done using an algorithm such as the Douglas-Peucker graph algorithm. Given the low poly estimate of the hand and the contours of the hand, stage four is to detect the concavities and the convex points on the hand. Again this task can be performed via the use of OpenCV commands. Once points have been generated upon the contours of the hand, the convex and concave points are filtered. The filter is performed in two passes. Firstly clusters of points were merged into a single point, should the distance between the points fall below a given threshold. Secondly any convex points which fell below the palm of the hand where removed. Finally the resultant points can be used to estimate the number of digits visible on the hand. Du and To's technique is again limited to the use of only one hand, however has a 94% accuracy rate and a low performance impact.

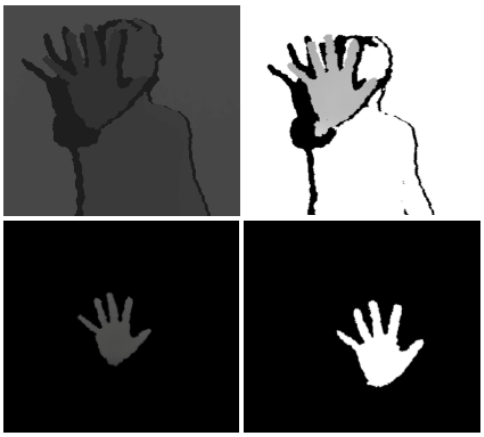


Figure 2: Starting from the top left; Original depth map, Rescaled image, Background elimination, Extraction

3.2 Voice Recognition and Synthesis

The Kinect hardware contains four microphones in an array. Along with this the Microsoft Kinect SDK has support and example projects showing how to use and implement voice recognition. However, although the Kinects array of directional microphones work well, I predict that in the hectic work place background noise may be an issue whilst performing voice commands. A solution to this would be for the user to wear a headset (with a microphone). This would mean that background noise would be brought to a minimum, resulting in less false positives of keyword recognition.

Microsoft offers the Speech API (SAPI), which supports speech recognition and speech synthesis (via the Microsoft text-to-speech (TTS) engine). However research suggests that the official Kinect SDK implements part of SAPI internally, making the two API's incompatible.

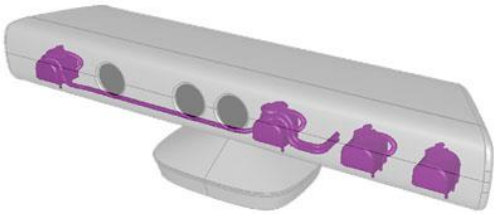


Figure 3: The Kinects Microphone Array highlighted in Purple. http://social.microsoft.com/Forums/getfile/18629/

As a bare minimum, support for voice recognition via the Kinect microphone array should be supported as well as speech synthesis. Research into alternatives to SAPI mainly looked into open source alternatives, due to the unique nature of the product. Viable solutions are the eSpeak and Voce projects. Voce is a Java based project; however it has C++ bindings whereas eSpeak is a standard C++ library.

Studies using voice based commands for performing simple tasks in a word processing application saw a 12%-30% improvement of performance [4], as the user did not need to move their hands away from the keyboard to the mouse to perform the same task. The same principles apply within this project, as some tasks (such as changing the current editing tool) can be performed in parallel to moving about the three dimensional environment via hand gestures. Ben Shneiderman [5] explains the limitations of voice based input in comparison to hand based input devices. Pointing out that since speaking commands consumes cognitive resources in a human, it makes it difficult to speak whilst solving problems at the same time. Meaning any voice based commands should be simple and intuitive for the user.

4 DESIGN AND IMPLEMENTATION

4.1 Programming Languages and Software Development Kits

4.1.1 C++11

Due to the need of the Direct3D 11 SDK and the computational speed required for processing the terrain data and Kinect image/voice data, the optimal programming language for development is C++, more specifically C++ 11. To use C++11 I will need to compile with the latest version of a compiler, due to this I will be developing the project in Microsoft Visual Studio 2012 (VS2012). Another advantage of using the VS2012 IDE is the new graphical debugging options that exist within the IDE itself. Prior to VS2012 direct3D debugging was performed via an external tool, named PIX. The built in graphical debugging should help fix any problems whilst implementing the terrain renderer, as I have minimal experience in Direct3D 11 (my experience lies in Direct3D 9).

Version 11 of the C++ language adds some new features which will potentially streamline parts of the development process. Both Kinect input and voice commands will need to be on different threads as to not stall the main process. C++11 implements a new <thread> class which will allow easy implementation of this, rather than the traditional Win32 threading layout. Along with this C++11 offers improvements to features such as Lambda expressions. Lambdas can be used to improve development time, by saving us from creating function objects whilst doing simple searches within data (such as gesture recognition).

4.1.2 Direct3D 11

The Direct3D 11 runtime supports three stages that implement real-time tessellation, something I believe the project will benefit from drastically visually. Although the project itself is based around user input, rather than the tool itself I believe that the creative people whom test the project will

undoubtedly point out if the rendering of the terrain is below par with an acceptable level. This may cause feedback to be weighted due to the lower rendering quality, not truly reflecting the aim of the project, the input method. However, although the project will be utilizing the Direct3D API the project rendering will be designed in such a way that a different rendering API could easily take its place; such as OpenGL. To help with the implementation of the Direct3D 11 renderer, I shall be loosely following the Raster Tek tutorials.

4.1.3 The Microsoft Kinect SDK and OpenNI

Before the start of the project I wanted to test both OpenNI and Kinect SDK with my home PC setup and Xbox Kinect. There are two types of Kinect, the Xbox Kinect and the PC Kinect. The main difference being that the PC Kinect has a much closer range (~3 feet) compared with the Xbox Kinect (~6 feet). Unfortunately I could not get OpenNI to work with my Xbox Kinect (the drivers simply failed to install), and although I could have used the university laboratories for development, I wanted to avoid this as I have a tendency to work into the early hours of the morning (when the laboratories are closed). This forced me to use the official Microsoft Kinect SDK.

The Kinect SDK comes with samples for multiple programming languages including C++. The API also suggests that it exposes the same amount of information as OpenNI and has similar methods, so previous research into implementations should still be valid. Using the Kinect SDK means that I will most likely run into an issue whilst trying to synthesize voice to the user, due to the conflicts in SAPI and the Kinect SDK.

4.2 Terrain System Design

The terrain system itself can be represented by a one dimensional array of points. The points can be expanded to contain useful information about the terrain. Position, normal and the texture coordinate will suffice for this project. The reason behind using a one dimensional array rather than a two dimensional array is to help with writing and reading terrain data to and from

files. This does have some complications when accessing points about the terrain based upon a three dimensional world position. This will be covered shortly.

To create and initialize the terrain data into a state where we can pass it to the renderer to be drawn to screen, we need to firstly populate our height map array. We store the desired size of the terrain in a two dimensional vector. Given the size of the terrain we allocate a buffer of the size; width * depth. Then initialize the array spinning through all elements setting the position of the point, leaving the y-coordinate to zero.

Now we have the x, y, z coordinates of each point in the height map correctly initialized we need to calculate the normal and texture coordinate of each point. Every time the user deforms a part of the terrain, not only the position will need updating. Both the normal and texture coordinate will need updating as well. Creating a public method for both properties will help greatly.

Calculating the normals of the terrain data is performed in two passes. Firstly we need to go through all the faces of the terrain and calculate their normals. Secondly we need to go through every point (vertex) in the terrain and average each face normal to get the averaged normal for that vertex. The first pass takes the three vertices of a face then takes the cross product of the two vectors created from these vertices. The second pass calculates each vertex normal. This is calculated by taking the normalized value of the sum of their four surrounding face normals, calculated in the first pass.

To calculate a texture-coordinate of a vertex within the height map we need simply spin through each vertex in the terrain setting the u and v coordinates to a stepped value, wrapped between 0 and 1. The stepped value is based upon a repeat value, which represents how many times the texture will be tiled upon the terrain. The texture repeat value should take into account the size of the terrain to make sure the textures seamlessly wrap the edges of the terrain mesh.

Given a method on the terrain class which can return a height map vertex at a given coordinate, we can adjust any given point based upon different requirements. This can then be accessed via the input class (mouse or keyboard). To get a vertex on the terrain we simply find the nearest value to a given point. We can ignore the y-coordinate of the vertex and simply compare the x and z coordinates. A simple solution would be to spin through the vertex array storing the distance from the requested point and the vertex. This however may be performance intensive should we be transforming multiple vertices per update. Knowing the size of the terrain we could estimate the nearest vertex point which would have performance benefits, but suffer from potential loss of accuracy.

The generated vertices can now be passed to the renderer. The renderer can then create the relevant resources (such as vertex and index buffers for the Direct3D 11 implementation of the renderer). Mapping and un-mapping of the vertex buffers can be used to update the rendered buffers to the latest terrain buffers. Ready for deformation via the relevant input device. A UML diagram describing the terrain and rendering system can be seen at appendix A.

4.3 Kinect Input System

The Kinect input system can be split into two uniquely implemented parts. The hand gesture controls and the voice based commands. Firstly the hand detection and gesture needs to be implemented as depending on what features are successfully implemented directly effects what voice commands will be required.

I believe Du and To offer the best solution for hand detection using the Kinect hardware, so the implementation will be based around theirs. However Du and To use the OpenNI API as well as OpenCV. I will be using the Kinect SDK. Although both have access to the same information generated by the Kinect hardware, it is presented in different formats.

Given the depth data from the Kinect, I will first sample down to a range between a minimum and maximum clip point. The depth data is present in a one dimensional array or RGB data, however the data is actually grayscale. The value of a pixel represents the depth value, or the distance from the point to the camera. This allows us to remove any information from the background and ignore anything too close. The problem with items which are too close is that the Kinect has a limit on how close an object can be to the camera. If an object is too close the data is inaccurate and distorted.

Next I sampled the depth data, looking for information that might represent the bounds of the hand. The depth data is presented by 640x480 pixels. Performing hand detection on that area would be slow. By making the assumption of a hand being a certain size we can sample down to a much smaller area. An area of 120x120 pixels for a fully open hand was achieved. To achieve this I simply spin through the data finding a point that represents the upper, lower, left most and right most points, from the previously sampled depth data.

Using this I implemented what was planned to be the first test iteration of simple hand detection. Given the width of the sampled hand bounds relative to a minimum and maximum hand span bound, I could estimate two different hand states; open (fingers fully extended) and closed (a closed fist). The basic idea is that if the distance to the minimum hand span bound is less than the distance to the maximum hand span bound, we presume the hand is closed; else presume the hand is open. This rudimental approximation was enough for the first testing phase. At this point closing the hand simply raised the terrain vertex at the center of the terrain mesh.

A gizmo exists within the tools, which is used as a replacement for a mouse cursor within three dimensional space. Given the sampled hand bounds, we can estimate the center of the palms position. Treating the hand like an analogue stick about the center of the depth data, we can control the gizmo by applying relative movements. Taking the vector between the center of the

depth data and the hands palm we can simply translate the gizmo by the same vector. The gizmo uses the terrain height as its y-coordinate. Doing this means the speed the gizmo travels is based upon the hands distance from the center of the depth data. Adding a small dead zone about the center stops any potential noise whilst trying to keep the gizmo still. Once again this enforces the principle of an analogue stick on a game controller. Finally I set the colour buffer of the Kinect to save an image every ten seconds to further monitor users and their interactions.

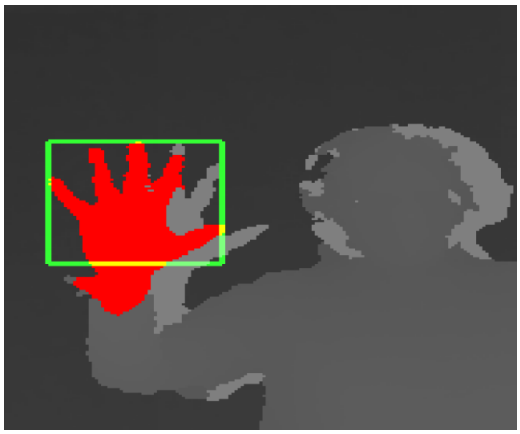


Figure 4: Basic hand detection implementation preview

5 TESTING: ROUND ONE

The first open testing was performed in a university lab containing artists and designers. The current project had rudimentary gesture detection and the ability to move a gizmo about the terrain. Using their hand a user could raise the terrain under the gizmo.

Users used the product one at a time, editing the terrain using the Kinect and their hands. The same user then used the traditional input of the keyboard and mouse to perform the same task. Following this a short questionnaire was completed.

5.1 Intuitiveness of Hand Gestures

The first section of the questionnaire examined the intuitiveness of the gestures required to interact with the product.

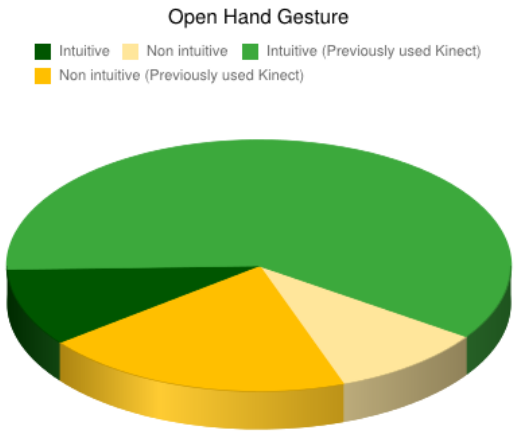


Figure 5: How intuitive users found the open hand gesture, taking into account past experience with the Kinect device

Whilst watching users interact with the product they used the open hand gesture without hesitation. This however may be due to users having previous exposure to a Kinect based interface. Whilst navigating menus within Kinect powered games, an open hand is often used to move a curser about the screen. This is reflected in the feedback with 60% of users whom had previously used the Kinect finding the gesture intuitive. It is interesting to note that 30% of people found the gesture non-intuitive of which two thirds of them had used a Kinect before; potentially indicating that the gesture is only intuitive to some due to it being the norm.

The closed hand gesture showed around the same amount of intuitiveness as the open hand, however less people whom had previously used a Kinect found this. Most likely due to the fact that the Kinect does not natively support hand detection; meaning that current exposure to users doesn't include changes in their hand state. However, with 70% of users finding it intuitive initial feedback appeared positive, but has room for improvement.

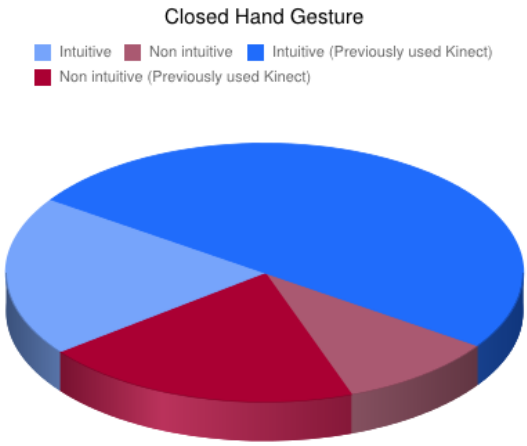


Figure 6: How intuitive users found the closed hand gesture, taking into account past experience with the Kinect device

The testing also allowed users to compare and contrast between the Kinect input and the keyboard and mouse. Instead of using the hand, the user can use the mouse to move the gizmo about the terrain, using right click to raise the point under the gizmo.

When asked what the users preferred method of input was, more users said the Kinect. This however is most likely down to the difference of input and the 'hype' around the device. Given a test period of longer than 5 – 10 minutes per user I would presume this value would drop due to fatigue. This is something that would be tested later in development.

The more noticeable result occurred when asked which device they felt was more productive. 80% of users found the keyboard and mouse more productive. This most likely reflects the fact that the users are more used to the traditional input of the keyboard and mouse. Aside from this fact the mouse is a highly accurate input device, allowing the user to make small and detailed movements. Alternatively, the Kinect offers data in a low quality image, meaning smaller movements are extremely noisy causing issues whilst trying to add detail to the terrain.

Whilst observing the users it was blatantly obvious that the traditional keyboard and mouse input was easier to use for the user. Allowing the user to create faster and more detailed terrain systems with the keyboard and mouse compared with the Kinect.

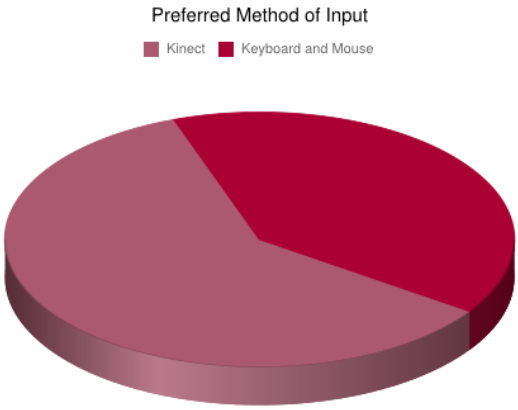


Figure 7: Users preferred the Kinect based input device.

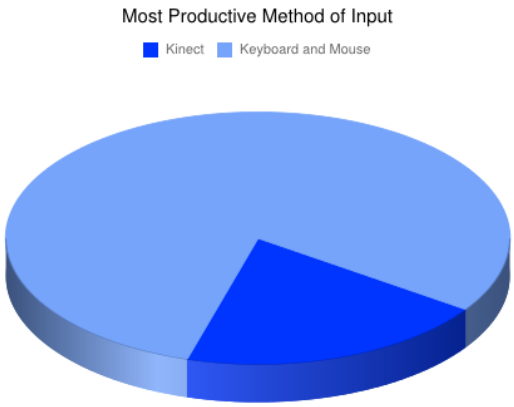


Figure 8: Users found the keyboard and mouse input more productive

5.2 Suggested Gestures

Finally, the questionnaire asked what other types of hand gestures could be used for traversing and deforming the terrain. The sensible and more popular suggestions can be seen in the chart to the side [Figure 9]. It was suggested that the open hand should pan the camera, then to transform the gizmo a single finger should be user. This in theory would allow easier traversal of the full terrain system.

The other notable suggestion is the use of two hands. This was something originally proposed by Terry Greer at Blitz Games Studios whilst talking about the concept. However, all of my research into hand detection using the Kinect had little too no success in accurately capturing both hands. For this reason I did not attempt to implement multi-hand detection.

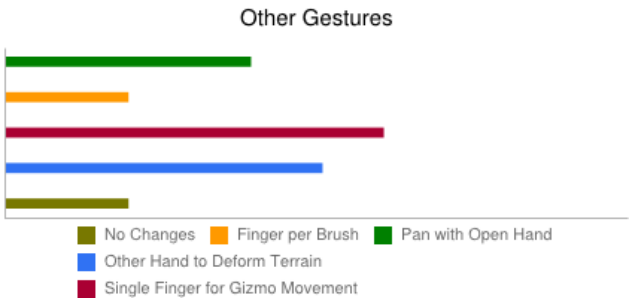


Figure 9: Suggested hand based gestures

6 REEVALUATING THE DESIGN AND IMPLEMENTING VOICE COMMANDS

The first test stage suggested that the users wanted more hand based gestures to perform different actions. As the initial implementation didn't properly detect the users' hand(s) this is what will be implemented next.

6.1 Edge Detection

Loosely following Du and To's method I plan to filter the sampled area to find the counters of the hand. To do such I will use Sobel's edge detection filter. The Sobel filter performs a two dimensional spatial gradient measurement on an image. The Sobel edge detection filter uses a pair of 3x3 convolution masks, one which estimates the gradient in the x-direction and another which estimates the gradient in the y-direction. The masks can be seen in figure 5. The magnitude of the gradient can then be calculated using the following equation;

$|G| = \sqrt{(Gx^2+Gy^2)}$.

-1	0	+1
-2	0	+2
-1	0	+1

Gx

+1	+2	+1
0	0	0
-1	-2	-1

Gy

Figure 10: Sobel edge detection masks

6.2 Dynamic Deformable Template Models

To find the gestures themselves I will use dynamic deformable template models [6] (DDTMs). DDTMs are a subset of DTMs (deformable template models). A DTM is a collection of points and colours which can be used to identify shapes and objects. The basic principle is that given a source (center) point and n points of data points, a normalized shape can be found. A source point in a DTM contains three values; position (vector2), colour (vector3) and colour tolerance (integer). The bounds of the

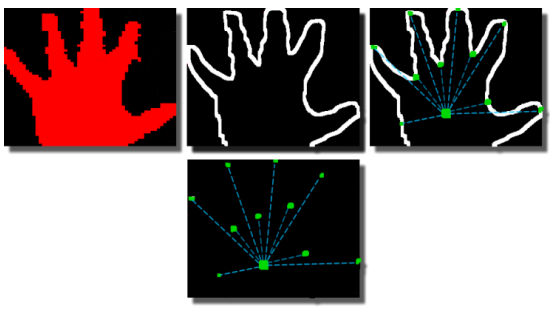


Figure 11: Open hand DDTM mask

data points are also stored. The DTM mask then scans over an image, failing if the source point fails a colour comparison and also fails if anyone of the data points fails a colour comparison. A DDTM stores and additional value per data point, a position tolerance. Along with this a DDTM can be rotated meaning the image does not need to be normalized prior to being processed. The positional variance allows the same mask to be use even when variations in the sample image exist, making DDTMs a good solution for detecting hands, as consistently generating the same hand shape for a user is unlikely and also this allows for differences in hand shape and size per user.

6.3 Point, Pan and Apply

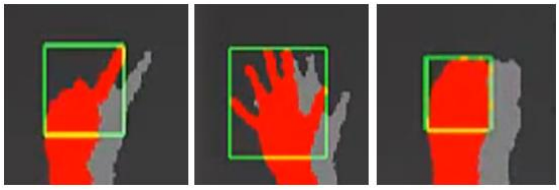


Figure 12: Point (left), Pan (center), Apply (right)

With the DDTM system in place the following three gestures can be implemented. Point, Pan and Apply. The point gesture is used to move the gizmo about the terrain, without moving the camera. The pan gesture is used to move the camera in a two dimensional panning motion. The apply gesture is used to apply the currently active brush. The system worked extremely accurately, however computational time was astronomical. The sampling of the 640x480 depth image, Sobel edge detection and DDTM gesture detection is being performed on the CPU, crippling the Kinect processing thread.

It is possible to have an interactive computation time, but only by dropping the tolerance, in doing so drops the gesture detection accuracy. This isn't an issue whilst detecting my own hand as the DDTM is modelled about the shape and size of my hand. The problems arise when supporting other people's hands sizes. Without high enough tolerance the accuracy of gesture detection drops vastly.

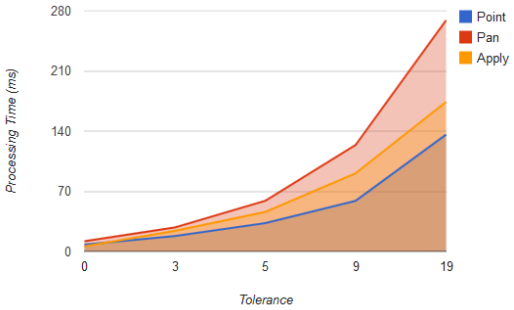


Figure 13: Processing computation time relative to position tolerance

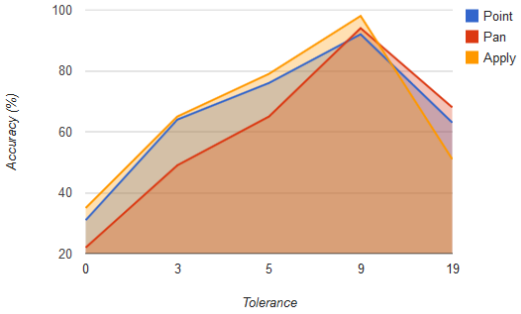


Figure 14: Gesture detection accuracy relative to position tolerance

6.4 Brushes

It was suggested in the first test stage it should be possible to deform the terrain in different ways. User feedback suggested that gestures should be used to change how the terrain was deformed, however due to the computational limitations discussed previously I opted for a brush based system. A brush is the term used for a different operation that the gizmo will apply to the terrain. A brush has size and strength. The size is the radius about the center of the gizmo the brush should affect. The strength is the power of the brush, affecting the rate of change of the terrain when the brush is applied to the terrain. Six different brushes will be implemented.

- Raise – Increases the height of the terrain
- Lower – Decreases the height of the terrain
- Deform – Changes the height of the terrain based upon the users hand position
- Level – Sets the height of the surrounding points to the same height as the center point
- Noise – Randomly offsets the terrain up and down
- Smooth – Averages out the terrain under the gizmo, smoothing out the terrain

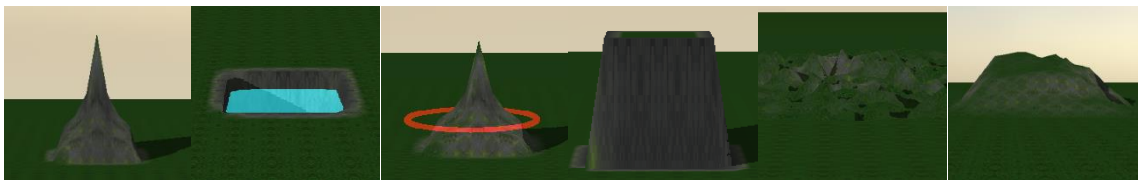


Figure 15: From Left to Right. Raise, Lower, Deform, Level, Noise, Smooth

6.5 Voice Commands

With the new brush system implemented, the logical method of changing brush is via brush commands. To support voice recognition a separate thread will be required to process audio without stalling the application or Kinect image processing thread. The Kinect SDK supports voice recognition, expanding on Microsoft's SAPI.

There are five steps to enabling voice recognition within the application.

1. Initialize the audio stream.
2. Create the speech recognizer.
3. Load and parse the grammar file.
4. Start the speech recognition.
5. Enable the audio command processor.

Initializing the audio stream sets up the Kinects microphones. Setting how many microphones to use as the Kinect supports four. Next the audio stream type needs to be set too initialize the audio stream. The format was set to the WAV format, with a single channel sampling at 32kbs.

The speech recognizer utilized the Microsoft's Speech API (SAPI). The API simply queries the hardware for the best fitting audio in device. The query takes a flag specifying the language to initialize for and a flag specifying if it should prioritize the Kinect device over other input devices. Setting the Kinect flag to true and the language to 409, enables the speech recognizer for English and initializes the Kinect audio device. The query fails if the Kinect device could not be found.

The grammar file is an xml based file that specifies keywords and similar sounding words. The audio processing thread, via the use of SAPI methods passes successful semantic checks to a single method, which I use to handle the keywords. Each keyword

```
<item>
  <tag>VISCRAFT</tag>
  <one-of>
    <item> viscraft </item>
    <item> vis </item>
    <item> craft </item>
  </one-of>
</item>
```

Figure 16: The semantic used for the VisCraft keyword

(or item) is created from one or more semantic (or words). The file location is simply passed to a SAPI method which parses and prepares the file for use, creating a grammar object.

Starting the speech recognition can now be performed as SAPI has been setup correctly. We need to set the grammar state created to active, as it is possible to have multiple grammar objects. The speech recognizer object also needs its state set to active, to start the recognition process. Finally we store the speech recognition event. This will be called when a keyword is recognized by SAPI, allowing us to catch the event and process it our separate audio processing thread.

Finally, we start our audio processing thread, to which we pass notifications when the SAPI event fires. A static map is created in the method which contains all the semantics described in the grammar file, along with a corresponding identifier (for which an enumeration value can be used). The map is then spun through comparing the notification semantic tag with our list. If a match is found the relevant operation is processed. It is worth noting that the speech recognizer takes an accuracy value between 0 and 1, where 0 is no accuracy and 1 is 100% accuracy. I had the best success with 0.2 whilst testing with people about the laboratories.

A keyword ‘VisCraft’ is used to start main processing of the audio commands. This stops false positives accruing; for example a stander by suggesting using a specific brush would instantly change the brush. Instead, any additional command will not be processed until the VisCraft keyword is recognized. This also removes additional computational overhead.

7 TESTING: ROUND TWO

The second testing phase aimed to discover the robustness of the newly implemented voice commands; as well as testing the performance and accuracy of the new hand gesture detection algorithm. Similar to the first round of testing, users where given time to use the project firstly via the Kinect input device, followed by the use of the keyboard and mouse.

7.1 Robustness of Voice Commands

The user was informed of the VisCraft keyword, however they were not told about any other keywords. The reason being that, when the VisCraft keyword is detected an on screen user interface is displayed to the user. Not informing the users of other keywords meant the intuitiveness of the commands that corresponds to options upon the user interface could be examined. The users adapted to the voice commands quickly and seemed to find them useful.

The number of times each user said a keyword was collated, along with this the number of times the keyword was recognized by the project and the number of times a keyword was recognized without the user saying the keyword. The information gathered [Figure 17] shows that any keyword which did not require the VisCraft keyword to be spoken first worked well, however this is expected as all instances of said keywords will be ignored if the VisCraft keyword is not firstly detected, reducing the potential for false positive detections drastically.

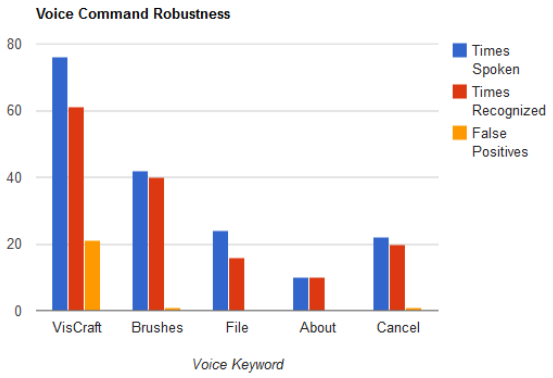


Figure 17: Comparing the number of time a keyword was spoken with the number of times the keyword was detected along with all detections of keywords which were not spoken

The VisCraft keyword itself had an 80.26% successful detection rate, which given there was background noise and different accents is promising. The issue lies in the false positives the VisCraft keyword also generated. When

the keyword is detected and the user interface is displayed all interactions with the terrain gizmo are disabled, making the false positives a major annoyance as well as a hindrance to productivity. The VisCraft keyword listens for three semantics; VisCraft, Vis and Craft. To reduce the number of false positives there are two options. Firstly the number of semantics could be reduced to just the word 'VisCrart'. Alternatively, the accuracy value used by the speech API could be increased to a larger value. The problem with the later is that it could reduce the number of successful recognitions for all keywords. Due to this reason, the number of semantics will be reduced first.

When it came to using the mouse and keyboard to navigate the user interface and traverse the terrain system, most users still used the voice commands for initially opening up the user interface and in some cases used the voice commands to then navigate the user interface. This is probably down to having the ability to speak in parallel whilst performing a mouse and or keyboard based gesture, meaning the user did not have to move their hands away from their current position to access the menu interface.

7.2 Performance and Accuracy of Hand Gestures

To start with the tolerance of the DDTM for each gesture was set to zero. This failed to detect even my own hand consistently and completely failed to detect any other user's hand. Based upon testing on my own hand whilst implementing the DDTM gesture recognition I set the positional tolerance to nine.

With the tolerance set to nine, the gestures where recognized whilst testing against the user's hand. However, the performance was so poor that traversing the terrain with the gizmo became impossible due to the performance issues. This in affect made the tool system unusable. Due to the performance vs. accuracy issues with the new DDTM gesture recognition, the new gestures themselves could not be correctly evaluated by the users.

8 CHANGES BASED UPON USER FEEDBACK AND ISSUES WITHIN THE IMPLEMENTATION

Based on public testing of the hand based DDTM gesture detection performance issues and time constraints the hand gesture detection will need to be vastly simplified, as well as user interaction being more based around voice recognition.

Given that users found the first iteration of hand detection usable (although lacking features) I will fall back to that implementation. This is where SVN can be used, as I can simply revert the hand detection based files back to the previously required state.

8.1 Improvements to Hand Detection

Since the hand detection is being vastly simplified, it requires some improvement. The first change is to make it so that the pivot point (previously this was the center of the depth render window) is set to where the hand is first detected. What this means is all movements are relative to the position of the users hand rather than absolute to screen space. To further improve on relative vs. absolute movement issues, when the user closes their hand to apply the currently selected brush the pivot point is moved to the position of the hand when the closed gesture was first recognized. This also fixes issues were users would start applying a brush and continue moving the gizmo at the same time. Moving the pivot point means that the application of the brush will only effect the point where the user wants, unless they move their hand.

8.2 Finalizing Voice Commands

With the voice commands seeming to be working well and receiving high praise from users, all brush based and render variants will also be exposed. Commands for changing the brush size and brush strength will be implemented; however these are variants that are likely to be changed often. Taking this into account, rather than navigating the menu interface to change the size and or strength the commands will be available at any point (similar

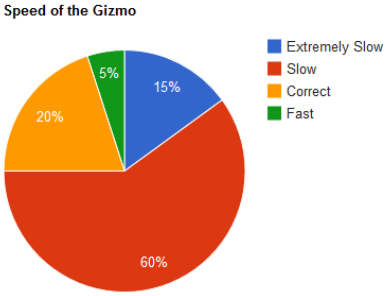
to the VisCraft keyword). Finally a voice command to toggle the render mode of the terrain will be added. One command for toggling wireframe based rendering and another to toggle smooth render vs. texture rendering.

9 TESTING: THE FINAL ROUND

With hand detection reverted and improved slightly based on user observations, along with the final voice commands being implemented I opened up testing to all, rather than the previous selection of users.
So far the lacking part of the project during testing has been the hand gesture recognition. Similar to the previous two testing phases, one by one users firstly tested the project using only Kinect based upon, followed by performing the same tasks using the traditional keyboard and mouse input devices.

9.1 Improved Hand Gesture Recognition

Whilst observing users, they seemed to be able to traverse the terrain much easier now that the pivot-point was local. This also meant that the Kinect no longer needed to be adjust per user (though in extreme cases the user may benefit from slight adjustments). When questioned however, the majority of users complained about the speed that the gizmo moves about the terrain. 60% stating that they believed it to be slow. This is probably a side effect of the relative based movement, as previously the users hand was often a greater distance about the pivot-point, however now the user moves relatively, distances between the pivot-point and hand position are minimal.



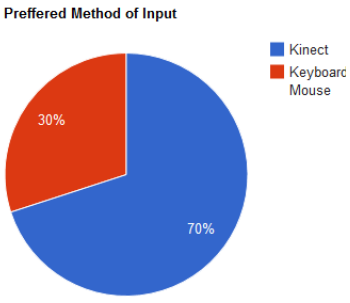


Figure 19: (Left) the users' Preferred Method of Input for the Terrain System.

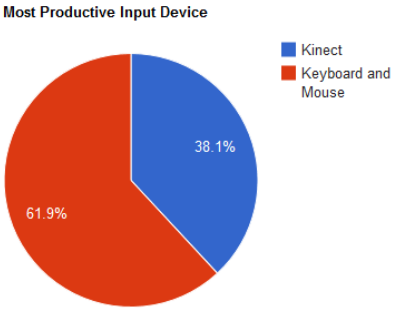


Figure 20: (Right) The Method of Input Users' found more Productive.

Given that 25% of users found the speed correct or too fast, the speed magnitude should be an exposed variable, allowing the user to set the speed to their liking, rather than that value being constant.

Users whom had not been present at the first or second testing pointed out that more hand based gestures should be added, similar to the results of the first testing phase. This indicates that although the product is functional, improvements still need to be made to the hand gesture recognition.
Once again users (for the majority) preferred the Kinect based input over the traditional keyboard and mouse input, reinforcing the principle of natural interfaces for program input. However, the more productive method of input is still the keyboard and mouse. This is most likely due to the previously discussed speed issues.

9.2 Voice Based Commands

The users were informed of the additional brush and render based voice command keywords. The ability to change the size and strength of brushes (on a per brush basis) was complimented by the majority. However, many commented on the lack of a prompt when the command had been recognized, suggesting something similar to how a television shows volume change via a bar on screen would suffice.

Again, given the chance to use the voice commands alongside of the keyboard and mouse all users utilized the feature. Similar to the second phase of testing, mainly the level-one keywords were used (the main VisCraft keyword, Brush size and strength changes and the render mode toggles).

In the final testing phase, it was also discovered that turning down the gain on the Kinect microphones improved the voice recognition accuracy, as it was possible to remove the majority of background noise from the input device, whilst still hearing the user and their commands.

9.3 Industry Feedback

Given the final state of the project, I prepared a video and a release candidate to send to my manager from Blitz Games Studio's, Neil Holmes. Given that Neil was involved when initial ideas were being drafted, he mainly commented on the missing features; such as duel hand support, along with this requesting more gestures, again something that was tested during the project but failed unfortunately. Neil also pointed out what most other users had said, in that the movement seemed to slow. See appendix B for the full feedback email.

10 EVALUATION

The majority of users preferred the interactivity that the Kinect device brought to the project. However there is a long way to go before this project becomes a viable solution for HCI within the games industry and professional level tool systems. Also given that the users only used the tool for 10 – 15 minutes and in industry a professional could potentially be using the tool for 8 or more hours, I would make the assumption that the benefits the interactivity brings would soon dwindle due to fatigue, however without full testing this may turn out not to be the case. The reason why this could not be tested was due to finding people whom can spare an entire day was not feasible.

The project requires more interactivity and fluidity from the user and their gesture and motions. A fully interactive menu system would most definitely be something the user and system would benefit from. Testing itself needs to occur over longer periods to truly evaluate the prospect of natural based input devices within industry tools.

Ethics were another area of issue within the project. Before starting the project I had not planned to use the colour output from the Kinect as a means of testing and evaluating users. However, whilst implementing the system, I added the ability to store images from the Kinect for evaluation post use. This has ethical issues, due to the potential miss use of imagery; the images are also stored without the users' knowledge. This functionality can be disabled by removing the path to the save folder. It is also worth mentioning images were only used in the short term, due to file size constraints. After every test session the locally stored images were deleted, once all relevant information had been gathered. Given the chance to produce this product again, I would first seek ethical clearance.

The project itself did struggle in fundamental areas. This most probably being caused by a lack of in depth research into hand detection using the specific SDK I ended up using. All research papers implemented their versions of

hand detection using OpenNI rather than the Kinect SDK. On hind sight the project would most likely been more successful if I had of used OpenNI (meaning more time would have been spent in the labs allowing for more continuous testing of the project). Similar to this, I opted to implement edge detection and shape detections myself. The OpenCV image library already offered these and many more advanced features which would most definitely of saved time and improved performance.

That being said, the terrain system created was stable and robust. This reiterates the importance of static analysis and proper source control, something that I believe is lacking from the curriculum of my university course. The pre-commit static analysis often detected potential memory leaks and uninitialized variables which could have resulted in crashes or bugs nested deeply within the code base.

Finally, implementing my own terrain rendering system was a huge mistake. Large proportions of time were spent improving the rendering of a system which could have been implemented using an external library or program. This meant less time was left for implementing the main area of the project.

10.1 An Alternative to the Kinect

The project itself questions the integrity of the Kinect hardware. Due to its limitations in the official SDK and low resolution cameras, extracting accurate and detailed information implementing efficient and accurate hand detection was not possible. Since the project began other hardware has emerged.

The LeapMotiontm is a small USB device which is designed to be placed on a table, facing upward. Using two cameras and three infrared LEDs, the device observes a roughly hemispherical area, to a distance of 1 meter. It is designed to track fingers (or similar items such as a pen) which cross into the observed area. This would have been perfect for the system implemented, allowing hand based interaction in a confined area. However the device is not

available publically and developer versions were only made available after the project had been planned and implementation was underway.

10.2 Future Improvements

The project contains many potential areas of improvements. The following should be implemented to fully explore human computer interaction within industry tools;

- **Multiple hand detection** - The ability to detect both of the users' hands opens up the product to many changes which would potentially improve the human computer interaction between the user and tool.
- **Advanced gesture detection** – Most users wanted to use different hand gestures and movements to perform different tasks, such as pulling up and down terrain and moving left and right to smooth the terrain.
- **Configurable movement speeds** – The speed of the gizmo about the terrain should be configurable allowing the user to traverse the terrain at a speed most suited to them.
- **Voice recognition via any microphone device** – The voice recognition was a massive success, with most people utilizing the feature to the maximum and often favored when an alternative method existed for performing the same task. The only issue was background noise causing false positive results. Adding the ability to use a headset with a microphone much closer to the users' mouth would reduce this issue greatly.
- **A configuration stage, allowing features to be tailored per user** – This would configure such things as hand size whilst open as well as hand size whilst closed, allowing for much greater accuracy for a larger variety of hand types. Along with this, should the Kinect microphone be used, asking the user to speak certain phrases would allow the directional microphones to better pinpoint the users location, along with such the gain of the microphone could be varied based upon the detection of the specific phrases.

11 CONCLUSION

The project set out to see if creative professionals would benefit from tool systems which tended towards their natural means of creation. The general feedback from users was positive, and strongly suggested that a more natural form of input would benefit both their creativity and productivity. However, the project itself did not fulfil all the needs the user required.

The current generation of the Kinect does not handle hand and finger detection well. The lack of resolution with the Kinect cameras and limiting features of the official Microsoft Kinect SDK make finger detection computationally improbable. Other hardware devices are emerging which tend towards the area of professional level human computer interaction, rather than the traditional general user.

The voice commands recognition and detection is by far the most successful area of the project. The technology for voice detection in both hardware and APIs exists for the majority of users. Users like to be able to open menus using their voice rather than moving their hand to the other side of the screen, or move their entire body position altogether. The ability to parallelize some commands via voice could help productivity in the majority of tools in the industry.

REFERENCES

[1] Matthew Tang. (2011). Recognizing Hand Gestures with Microsoft's Kinect. Department of Electrical Engineering. Stanford University.

[2] H. Bay, T. Tuytelaars, L. Van Gool. (2008). SURF: Speeded Up Robust Features. Lecture Notes in Computer Science.

[3] Du, Heng. To, TszHang. (2011). Hand Gesture Recognition Using Kinect. *Department of Electrical and Computer Engineering*. Boston University.

[4] Karl, L. Pettey, M. Shneiderman, B. (1993). Speech versus mouse commands for word processing applications: An empirical evaluation. 667-687.

[5] Shneiderman, B. (2000). The limits of speech recognition. Communications of the ACM. 43 (9), 63-65.

[6] Jain, A.K. Zhong, Yu. Lakshmanan, S. (1996). Object matching using deformable templates. *Pattern Analysis and Machine Intelligence*. 18 (3), 267-278.

BIBLIOGRAPHY

http://www.blitzgamesstudios.com/

http://www.blitztech.com/

http://subversion.apache.org/

http://cppcheck.sourceforge.net/

http://www.cs.sunysb.edu/~algorith/implement/DPsimp/implement.shtml

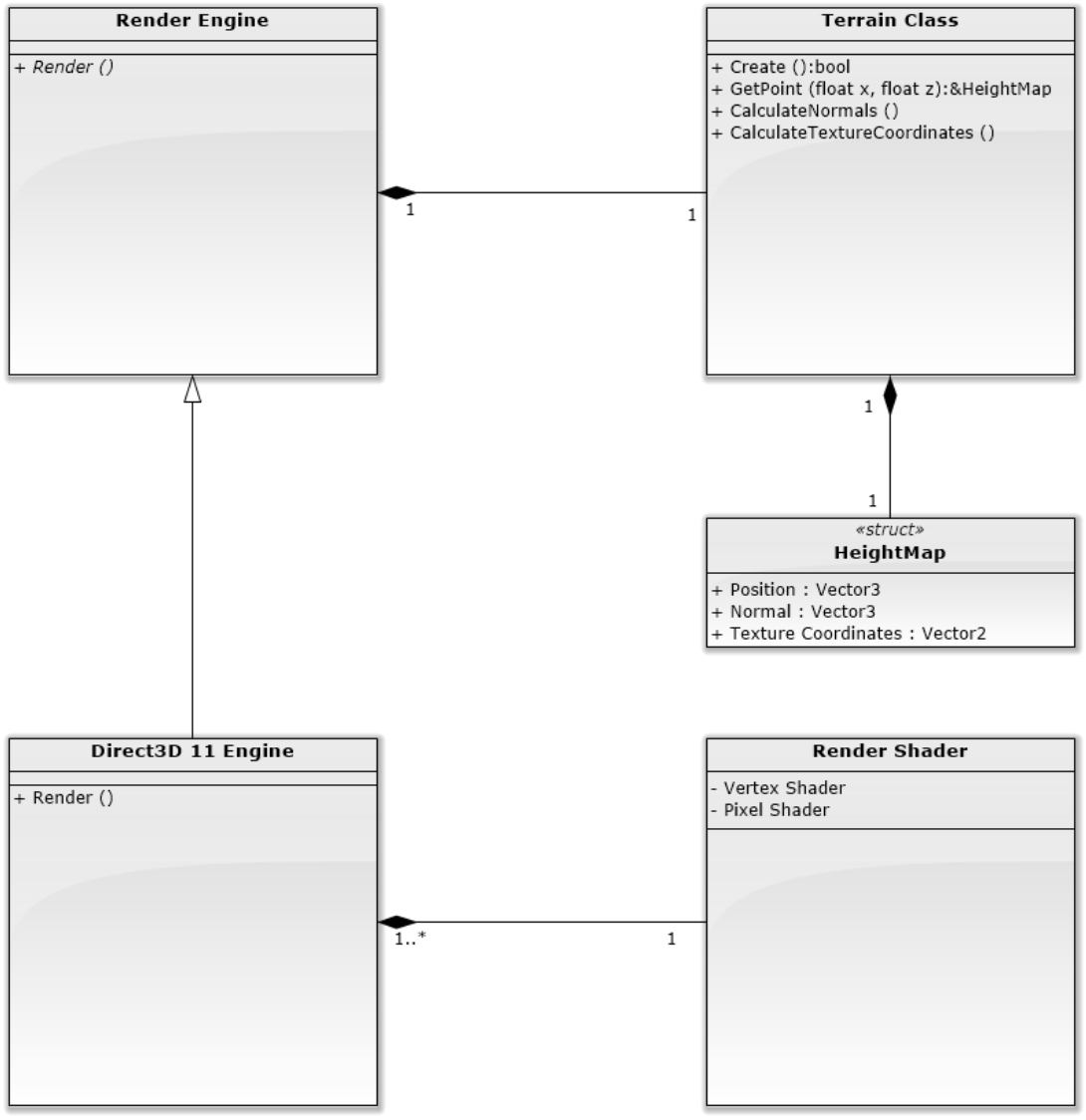
http://www.openni.org/

http://opencv.org/

http://espeak.sourceforge.net/

http://voce.sourceforge.net/

APPENDIX A


Render Engine
+ Render ()
Terrain Class
+ Create ():bool
+ GetPoint (float x, float z):&HeightMap
+ CalculateNormals ()
+ CalculateTextureCoordinates ()
1
1
1
1
«struct»
HeightMap
+ Position : Vector3
+ Normal : Vector3
+ Texture Coordinates : Vector2
Direct3D 11 Engine
+ Render ()
Render Shader
- Vertex Shader
- Pixel Shader
1..*
1

APPENDIX B

General stuff about the project:

Ok, big one first - It's kinda a shame that the UI doesn't really make use of gesturing to edit the terrain, I imagine you've had a nightmare with getting gestures to work, and the voice stuff is pretty cool, but it would have been so much more awesome to be able to raise, lower, add noise and smooth all with gestures rather than a clunky menu. I realise it's way too late to make big changes now but some stuff you might be able to do?

- What's ya other hand for? Could you not track that and have it raise/lower the land with up and down gestures,
- smooth the land with side to side gesture
- add noise by rapidly shaking your hand up/down?
- You could use the same open/closed fist gesture to know when to start tracking and when to stop and it would be much cooler looking
- Without really knowing what you intend to say in the paperwork it's hard to know how important this is. I'm guessing it's more about the journey than the final product, and you might have already tried that stuff? Perhaps if that is the case it might be interesting to cover some of that in the video? I dunno – just thinking out loud

Generally movement looks really slow, it makes the whole thing feel a bit ponderous – I reckon you could double it and it would still be controllable? For your final round of testing can you set up a few x2 x4 x8 speed key shortcuts and see just how fast you can have things moving before it becomes silly?

The floating pivot point is really cool – very nice usability, is movement speed based on how far you are from the pivot, like with an analog stick? That would really add some nice controllability – small movements for slow speed, large movements to whiz about. If that's already in there it doesn't come across in the video