School of Computing

University of Teesside

Middlesbrough

TS1 3BA

Human Computer Interaction within Industry Tools

Bsc. Computer Games Programming

Sam Oates

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**

Neil Holmes, Tom Gaulton, Terry Greer, Test Subjects.

**Contents**

Abstract

Acknowledgements

1. Introduction
   1. Research Question
   2. Rationale for Project Choice
   3. The Current State of Human Computer Interaction
   4. System Requirements
2. Methodology
   1. The Methodology behind the Implementation
   2. Design Methodology
   3. Methodology of Testing
3. Research Analysis
   1. A
   2. B
4. Design and Implementation
   1. A
   2. B
   3. C
5. Testing: Round One
   1. A
   2. B
   3. C
6. Reevaluating the Design and Implementation
   1. A
   2. B
7. Testing: Round Two
   1. A
   2. B
8. Evaluation
   1. A
   2. B
9. References
10. Bibliography
11. Appendices

## Introduction

### 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 interface?

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 a potential 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 they believe the gestures should work.

### 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 found whilst on work placement at ‘Blitz Games Studios’. Whilst there I spent time working on their tool system (‘Blitz Tech’) as well as 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 rational between artists and programmers. To expand on this, I have experience with user interfaces, tools graphics/rendering and the Microsoft Kinect.

### 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.

### 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 however 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 Microsofts 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 first implementation with potential users.
* Improve first draft based on feedback from initial testing
* Consult potential users on voice commands for performing different operations within the system.
* Implement 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.
* Attempting to prove that both productivity and quality can be improved by reducing the barrier that exists between creative people and the tools extant within industry.

## Methodology

### The Methodology behind the Implementation

As the product required feedback based upon user experience, the product had to go through multiple repeated steps of development until all discovered issues were resolved. This means should there be an unseen problem within the initial plans; it can be refactored out at a later stage. 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 plans be unsuccessful. Three public testing points were set, where I would invite artists, designers and other creative to try out the project in its current state. After each test session feedback was collected and tasks reassessed.

Due to the rapid changes that would take place based upon user feedback I opted to use a form of source code versioning control software. Whilst working in the industry we used subversion control (SVN) [1]. SVN is based around the principle of one main repository (the server) and multiple local copies (the clients). A client simply checks out 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.

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 merges 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 was tagged as such. Meaning in the future I could easily return to that version to compare and contrast both the source code and the features.

Figure 1: http://www.hosting.com/media/387108/svndiagram\_500x416.jpg

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 helped find and resolve many issues in the later stages of the project.

### Design Methodology

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

Given that hardware tessellation is implemented within the Direct3D 11 SDK I choose this to be the graphical API I would use, as tessellation could be used to smooth the terrain with little overhead in performance. Due to this I also implemented 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 programming language. The Kinect SDK 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 microphones.

### Methodology of Testing

Throughout the project not only did I test usability and features, but static analysis was 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 I used an SVN commit hook with a program called Cppcheck [2]. This meant, that prior to any SVN commit static analysis was 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 helps in vastly improving the stability of the project as well as pointing out potential mistakes in logic which would previously go unseen.