Using Kinect to interact with presentation software

Marcel Aarts

Supervisors:

Oguzhan Özcan (MDH)
Christian Sjöström (Imagination Studios)

Examiner:
Rikard Lindell (MDH)

Mälardalens högskola, Västerås

School of Innovation, Design and Engineering

Imagination Studios, Uppsala

May 2012

Abstract

Imagination Studios is a company specialized in motion capturing and animation. Part of their daily business is working at trade shows where they have a booth to keep close contact with existing customers and also to find new ones. However, usually only two to three people will be working at the booth, and frequently, these people will be in meetings with potential customers. During a time like this, nobody is free to attend to other people checking out the booth. This can result in a potential loss of a new customer. This project seeks a way to alleviate that problem.

The idea behind this project was to create an application that trade show visitors can interact with in a playful and innovative way while also giving them a feel of what Imagination Studios is all about while looking for information about the company. To do this it was decided to let users interact with the system by using a Microsoft Kinect. The Kinect allows for easy implementation of a user interface based on motion capturing while also being very cost effective. A new user interface was to be designed as well, without copying already existing solutions and without simply expanding a traditional UI with new elements. To achieve this several design sketches were made, and the most interesting ones were then turned into storyboards. These were then used to decide on the final design, which was then elaborated on by use of video sketches and a collage in Adobe Photoshop.

Several tools were used during the actual implementation. For the actual visualization and graphical design, the Unreal Engine 3 in combination with UDK was decided upon. To connect Kinect and Unreal Engine 3, a third party addon called NIUI which makes use of the open source SDK OpenNI was used. For ease of debugging and programming in Unrealscript, the programming language used by the Unreal Engine 3, an addon for Microsoft Visual Studio 2010 called nFringe (Pixel Mine, Inc., 2010) was used.

Table of Contents

Abstract I

Table of Figures IV

1. Introduction 1

2. Project Requirements 2

3. State of the Art Overview 2

4. Design Phase 3

4.1 Concept 3

4.2 Design process 4

4.3 Gestural Control 5

4.4 Description of Designed Features 6

4.5 Idle system 6

4.6 User detection 7

4.7 Data Access 7

4.8 Contact Information 9

4.9 Other Developed Ideas 10

4.9.1 A city of data 10

4.9.2 The Data Tree 10

5. Hardware / Used Tools 11

5.1 Kinect Hardware 11

5.2 Kinect SDK 12

5.2.1 Licensing 12

5.2.2 Features 12

5.2.3 Decision 13

5.3 Game Engine Comparison 13

5.3.1 Cryengine 3 13

5.3.2 Unreal Engine 3 14

5.3.3 Unity3D 15

5.3.4 Decision 16

5.4 Tools 16

5.4.1 nFringe 16

5.4.2 Visual Studio 16

5.4.3 Adobe Photoshop 17

5.5 Special techniques used during development 17

5.5.1 Gesture Recognition 17
5.5.2 Detecting if hand indicators are hovering over a bubble. 18
5.5.3 Using opacity maps for hover effects 22
5.5.4 Bubbles and Animations 24
6. Test and Evaluation..25
7. Conclusion and Outlook 25
8. References..26

Table of Figures

Figure 1: Screenshot 1 from a video sketch 4
Figure 2: Screenshot 2 from a video sketch 4
Figure 3: Idle screen showing different menu options 6
Figure 4: Idle screen in the actual software 7
Figure 5: Cells floating inside a life form 8
Figure 6: Information within a cell is being displayed 9
Figure 7: Cell containing QR-code....................10
Figure 8: Infrared grid projected by the Kinect 11
Figure 9: Kinect Camera 12
Figure 10: Comparison of GPU vs. CPU PhysX....................15
Figure 11: The problem with perspective 19
Figure 12: General hover detection algorithm....................21
Figure 13: Bubble without hover effect 22
Figure 14: Bubble with hover effect 22
Figure 15: Texture holding all text....................23
Figure 16: Opacity Map....................23
Figure 17: Shader program used for hover effects....................24

1. Introduction

Imagination Studios is a company specialized in motion capturing and character animation for the computer game industry. As this is a constantly changing business, it is very important to always be on top of the newest developments and to look for new customers, as well as keeping the old ones. One of the methods of doing this is having a booth on trade shows, where potential customers can have a first impression of the company, as well as booking meetings with the staff and leaving their contact information.

However, all staff will frequently be busy tending to clients, which means that potential customers may be ignored and even lost because nobody is around to help them and take their contact information.

To alleviate this problem, the idea was born to create an application which can interact with customers when staff is busy. This way, people can still find basic information on the company, and even leave their contact information without staff being around to help them. To make the application draw interest the idea came up that interaction should be done in a new and innovative way. As Imagination Studios are a company specialized in motion capturing, using this technique to interact with the system seemed the logical choice. Also, the whole interaction should be playful and encourage the user to explore.

A cheap and simple way to do motion capturing is using the Microsoft Kinect. It offers simple and reliable skeleton tracking as well as a choice of both proprietary and open source SDKs.

The goal of this project was to create a completely new kind of user interface that is designed from the ground up with the thought of gestural control in mind. It is very easy to expand a standard user interface designed to be controlled by keyboard and mouse with support for a device like the Kinect. However, doing this will often feel a bit clunky and completely ignores the strengths a natural user interface using gestures and motion can offer. Designing a completely new interface is no trivial task and is hampered by the simple fact that we have become so used to point and click interfaces used in operating systems like Microsoft Windows and Apple's OSX. An extensive design process was needed with many sketches discarded early on in the process. This was a very important part of the project as it helped with the process of "thinking out-of-the-box".

The focus of the project was laid on both the design phase and the implementation phase. The design phase was extensive to make sure that the final design was actually innovative but also to make sure that it was usable in an easy and logical way. During the implementation phase, the goal was to create a so called "working prototype", meaning it should be possible to show all the details of human-machine interaction without the requirement of having the final content included in the program. A short testing phase was also done at the end of the project, however, the focus of the project was not on extensive user testing.

Special thanks go to Daniel Kade, who has been a tremendous help during the course of this project.

2. Project Requirements

During the design process, several different ideas were developed. Several ideas were rejected very early on, often during the initial sketching, as they did not meet the requirements of the project. The requirements of the project were as follows:

- Use of the Microsoft Kinect
- Interface based on natural user interaction, using gestures
- Visually appealing design
- Easy to learn, as it will be used on trade shows where nobody has the time or patience to spend any time on trying to learn how to interact with a system.
- Should be able to present any type of information to the user. This may include, but is not limited to, images, sound, videos and even other software.

3. State of the Art Overview

When researching the Kinect on libraries like IEEE and ACM it becomes clear very quickly that an astonishing amount of research is being done with this piece of hardware which was originally only created as an innovative game controller. If then narrowing down the search towards gesture recognition and its possible uses it quickly becomes obvious that while there are many papers on the topic of how to technically detect gestures, there are surprisingly few papers on the actual implementation and usage of said gestures. Examples of these technical papers are Biswas and Basu (Biswas & Basu, 2011) and Lai, Konrad and Ishwar (Lai, et al., 2012).

Vera et al. (Vera, et al., 2011) describe a system where they combine several sensors into a augmented reality experience. They use two Kinect devices for capturing a user's location, a gyroscope in form of an Android smartphone to track head movements, a WiiMote for facial expressions and a microphone capturing the voice of the user which is then used to calculate lip movement on basis of an amplitude based algorithm. Using the data provided by these sensors, the user can interact with the virtual character on screen, for example they can walk around him or talk to him. This project requires a user to put certain equipment on specifically to interact with the system and is therefore not a pure natural interaction system. This system has been used successfully at a marketing presentation for touristic content (FITUR '11). This is especially interesting as the project described in the documentis also planned to be used at marketing presentations and trade shows

One of the most important websites that shows off actual projects done with Kinect is called Kinect Hacks (Kinect Hacks, 2012). It offers videos of countless different projects and even has a Top 10 of the current best projects.

One of these projects is currently in use at the World of Coca-Cola museum in Atlanta, GA (Kinect Hacks - Protecting The Secret, 2012). It allows tracking of several users at once who can then playfully use gestures to interact with all kinds of objects on screen to find new information about the company and their products. In one of the games the users can play they need to find the combination to a vault containing Coca-Cola's secret. This can be observed in the video on the website from about timestamp 1:18. At about 1:24, users can be observed making large gestures trying to turn the knobs of the combination lock. They instantly understand what to do without the need of any sort of explanation from the system. Understanding the system without the need for explanation is a very important concept and something that needs to be incorporated into the current project as well.

Another very impressive project that combines augmented reality and gestures into what is called a “god game” is called “Kinect-powered Augmented Reality Sandbox”. With this system, a user can play in a sandbox while the system projects contours onto the sand in real time. With a certain gesture, virtual water can then be poured into the sandbox which will react to the structure of the sand in the sandbox (Kinect Hacks - Augmented Reality Sandbox, 2012). Interesting about this project is the seamless connection between the real sandbox, the virtual contours and water, and the gestures to combine it all.

The project KinectJS (Kalogiros, 2012) attempts to combine JavaScript and Kinect to interact with a website, with the ultimate goal of bring motion controls into HTML5. The website shows a couple of videos of a user interacting with a website and playing games on it. The gestures are simple and often require large movements. Unfortunately the general user interface is not tailored to natural interaction but is in fact strongly inspired by Microsoft’s Metro UI which will be introduced in Windows 8. Instead of using natural gestures, the user still needs to click on items, which in this specific implementation seems to be done through hovering over a button for a certain amount of time. This interrupts work flow and can also potentially be tiring for the user due to the requirement of holding his arm still in an unusual position. For the current project, KinectJS essentially shows how not to do it, as they attempt to control a standard user interface with gestures instead of developing a user interface tailored to the needs of gestural controls.

A project of a different kind is “Cauldren – Gestural Performance Piece” (Kinect Hacks - Cauldren, 2012), where all gestures and movement captured by the Kinect are used to create an audio-visual experience. All movement data is converted into MIDI, which is then played as sound. This means that fast movement makes for different sounds than slow movement for example. This is accompanied by stylized fireworks that follow the user’s hand movements and change in intensity depending on speed amongst other things. Again, the video accompanying the description explains the usage very nicely. This system incorporates natural interaction very nicely, as any movement by the user is interpreted by the system and converted into a certain output. While this type of interaction does not seem suitable for interacting with specific objects on a screen, it uses the capabilities of both the Kinect and natural interaction to its full advantage.

4. Design Phase

4.1 Concept

Currently, whenever Imagination Studios have a booth at a trade show, they have a very basic setup with some chairs and a TV screen looping their show reel. When all personnel at the booth are in a meeting with a client, potentially interested customers have no way of leaving their contact information and will simply walk past the booth.

Initial design ideas for the project were a general content management system, but when being confronted with the aforementioned problem, the decision was made to adjust the first design to be used as presentation software instead. As Imagination Studios are a motion capturing and animation company, using Kinect to navigate the software seems a logical choice as this incorporates their main business area into the presentation software, immediately giving anybody using the system a feel for what the company is about. It also makes using the system feel more like a game, adding fun to it.

The basic design concept is inspired by life itself. Pieces of information are considered cells, and they can be grouped into topics, called life forms, which means that all life forms consist of 1 or more cells. A vital thought behind the design was to avoid simply adjusting a standard user interface to work with Kinect but instead design a user interface specifically geared towards functioning best with motion capturing.

4.2 Design process

Initially, several designs were thought out and sketched on paper using only a pencil. This first phase was important, as sometimes ideas that seemed good at first turned out to be poorly thought out or simply impossible to do when brought to paper. If an idea showed potential, a more detailed design was created, drawing a complete storyboard showing the full extent of the interaction. These storyboards were then used to discuss the design ideas with supervisors and test persons. Two of the later rejected storyboards are discussed in more detail later in this chapter.

The storyboards were shown to several test persons asking for their opinion on which design they considered to be best. Most of them preferred the design using life forms to hold information. During discussions with the project supervisors, the same design was also preferred.

After the decision was made to use the design based on life forms, several video sketches were created. While storyboards are very simple pencil drawings, video sketches are made up of short video clips, photos, both edited and unedited to show the interaction in a more realistic environment. A sequence of two screenshots from a video sketch are shown below to illustrate.

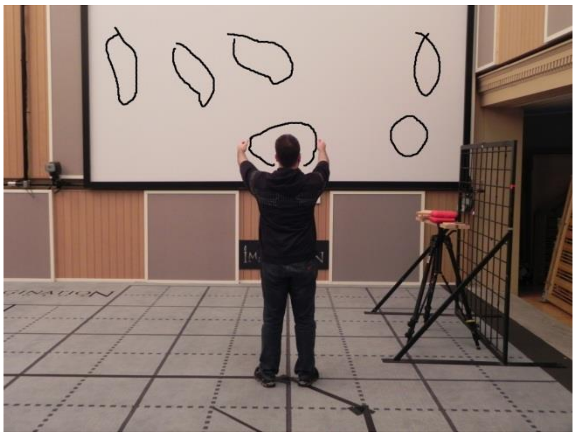
Figure 1: Screenshot 1 from a video sketch

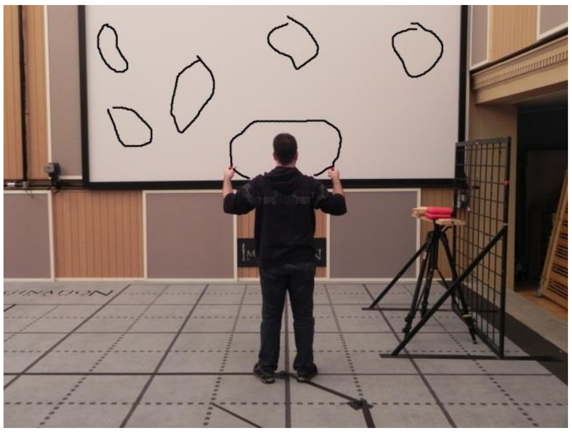
Figure 2: Screenshot 2 from a video sketch

The last step during the design phase was to create the actual visual design of the system. Storyboard and sketches would only give a very rough impression, and the video sketch was mainly used to explain the final interaction in great detail.

The visual design was done in Photoshop using several images that were then edited and put together to form the final design. Several of these images will be shown throughout this chapter when explaining different parts of the design in more detail.

4.3 Gestural Control

The system is designed to be controlled by capturing the body motions of the user by using a Kinect camera. The gestures are strongly inspired by gestures used for touch screens, mainly "Tap", "Pinch" and "Spread". This can be done according to Dan Saffer on page 45 (Saffer, 2009) where he writes:

"Patterns for touchscreens can also, in many cases, be applied to free-form interactive gestures, just slightly modified. For instance, the Spin to Scroll pattern can be performed with a finger on a device, but it could also be performed in the air with a finger"

During the design, it was also very important to keep the gestures as simple as possible, as the more complicated it gets, the fewer people will be able to perform these gestures. This is especially important for systems like for example a public kiosk that people rarely use as mentioned by Dan Saffer on page 38 (Saffer, 2009).

To access the cells, or information, within each life form, the user touches the cell and pulls it apart, increasing the size of the life form itself to fill the whole screen and with that making the cells within visible. This is a variant of Dan Saffer's *"Spread to Enlarge"* gesture found on page 64 (Saffer, 2009), where both arms are used instead of two fingers. Because the gestural system in place is three dimensional instead of two dimensional like a touch screen, an extra gesture to indicate that the user wishes to interact with the object is required, as failing to do so would randomly activate objects on screen. The gesture used here is a variant of Saffer's *"Tap to Select"* gesture found on page 48 (Saffer, 2009). To access the specific topic a certain cell covers, the user uses the same gesture as before. This creates consistency within the system and makes it easier to use. If the user lets go of the life form before opening it fully, for example when he suddenly decides he wants to look at something else instead, the life form will shrink back to its initial size on its own.

To make sure the user doesn't get confused when controlling the system, two markers on screen will indicate the position of each of the user's hands. The controls are kept very simple, as the design is explorative, meaning the user should be able to find out on his own how to use the system and enjoy himself while doing so.

To stop working with the information the user is currently accessing, it can be pushed together slightly, and it will automatically shrink again, in the same way it does when a user lets go of it before it is fully opened. This is a variant of the *"Pinch to Shrink"* gesture described by Saffer on page 64 (Saffer, 2009). Alternatively, the user can simply pull another cell visible on screen towards them, closing the current one automatically and opening up the new one. When having one cell open, the other cells in same life form will float next to it, allowing the user quick access to them. However, when a life form has been opened, the others will not show on screen to avoid information overload and confusing the user.

As these gestures are quite standard and are used in other software already, an alternative gesture system was developed as well. In the latest design, the life forms and cells are represented as soap bubbles. To access information within a soap bubble, the user would simply look at it and take a step towards it. When no longer interested in the information, they would then simply look away and take a step back, letting the bubble shrink back to its original size. Stepping forwards and backwards is a gesture described by Dan Saffer as well, found on page 74 and called *"Move Body to Activate"* (Saffer, 2009). Using the user's viewing direction as direct input is not a gesture that can be found yet, however it could be hypothesized that it is essentially a heavily modified version of Saffer's *"Point to Select/Activate"* gesture found on page 76 (Saffer, 2009).

4.4 Description of Designed Features

Following below is a description of all features designed during the design phase. During the implementation phase however, sometimes features were changed or discarded. This could happen for several reasons. For example, a better idea might have come up while working on the project, or certain ideas proved to be unsuitable when testing the prototype.

4.5 Idle system

Whenever no user is present, the system is in an idle state. While this is the case, it should try to capture the attention of anybody who happens to walk past the booth, get them to stop and watch. The life forms will slowly float around the screen, changing both their positions and the way they look. They may change both shape and color. The picture below shows the initial design for the idle state. The life forms all have different shapes that will keep changing. They also have a caption, clearly showing what information can be found inside them. The Imagination Studios logo is always present, on any screen to make sure the name of the company cannot be missed.

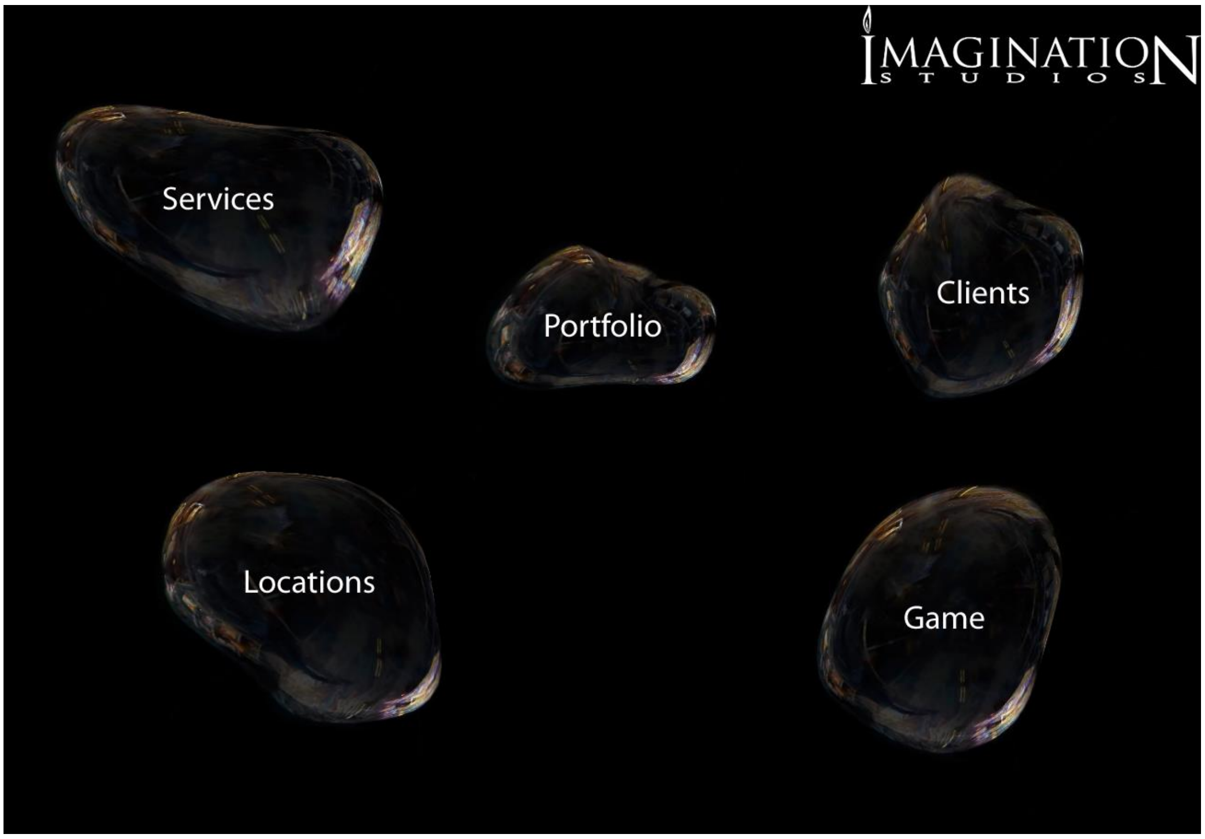

Figure 3: Idle screen showing different menu options

During implementation it was noticed that the design above was not very appealing and should be made much more interesting. Several changes were made, all of which can be seen in the screenshot below.

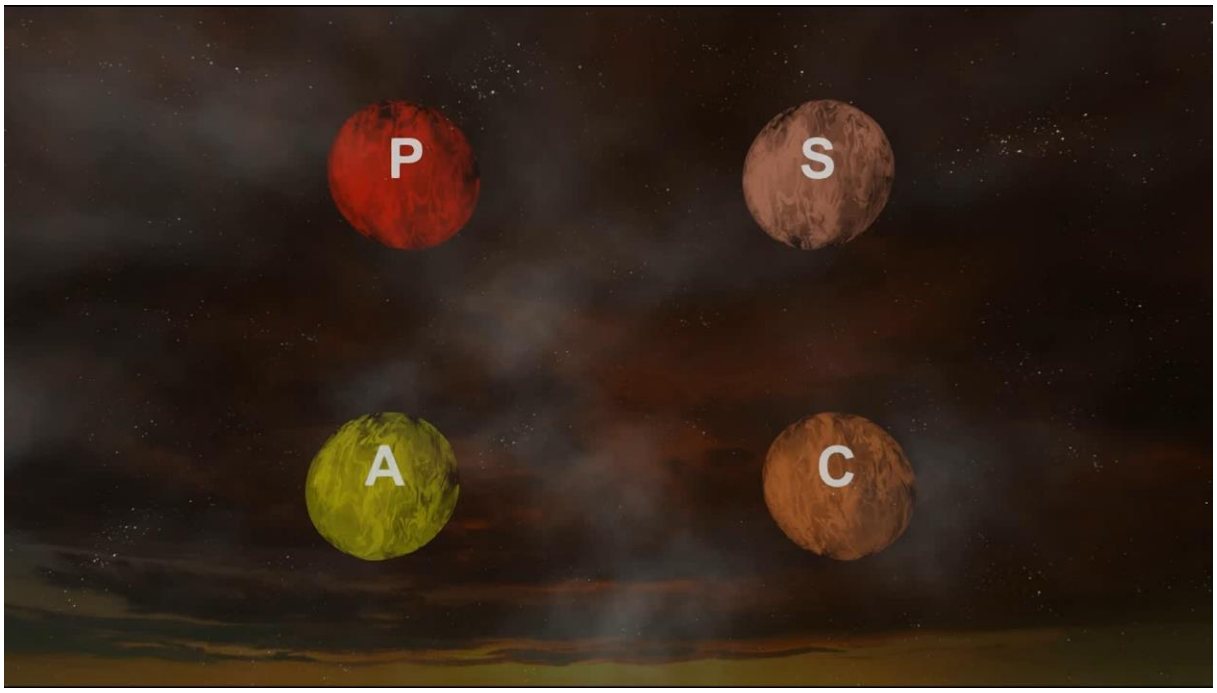

Figure 4: Idle screen in the actual software

Instead of always showing the full text of the content, only the first letter is displayed. The full text is then only shown when a user moves a hand indicator to move over one of the bubbles. A black background does not seem inviting, and therefore an animated night sky with moving stars and clouds was implemented. This was a suggestion made by an Imagination Studios employee who was testing the system.

4.6 User detection

When a user steps into the Kinect control zone, the system greets the user. This is used as an acknowledgement for the user so he knows that the system sees him now and is ready to get input. The life forms will stop moving now to facilitate interaction with the system and to prevent the user having to chase after the information he is interested in.

4.7 Data Access

When the user starts to access the information he is interested in, the system will keep track of what he has accessed. This data will later be used if the user requests more detailed information or wishes to be contacted.

The life form the user accessed will grow to fill up most of the screen, showing the cells it contains. It will not change its shape from round to rectangular however. This is to keep the design consistent and by that making sure that the user always knows where he is and won't feel lost. The cells inside float freely, also changing their form and color randomly. Each cell has a caption, showing what specific topic it contains information about.

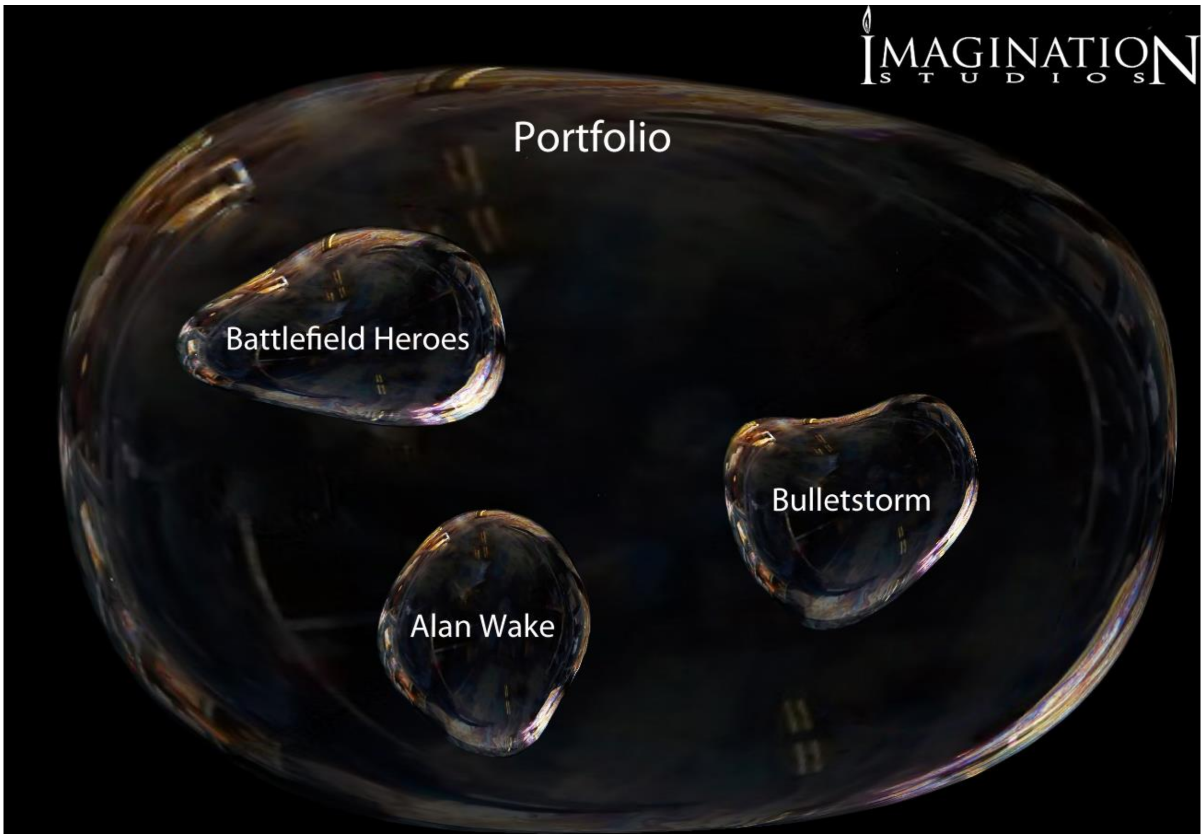

Figure 5: Cells floating inside a life form

Accessing a specific cell makes it grow larger. The other cells will still be visible next to it however, to make accessing them easy. A new cell labeled “Detailed Information” will appear as well, enabling the user to request more detailed information, or leave his contact details. Cells will only show basic information, a picture and a headline is all that is needed as nobody is interested in reading much on a screen at a trade show.

During implementation, several changes were made here. The user’s moves are no longer tracked as that information is not actually needed. Also, whenever a cell is being accessed, the others cells are faded out instead, allowing the user to focus on the item he is currently interested in. The other cells fade back in when the user indicates he is done with the current cell.

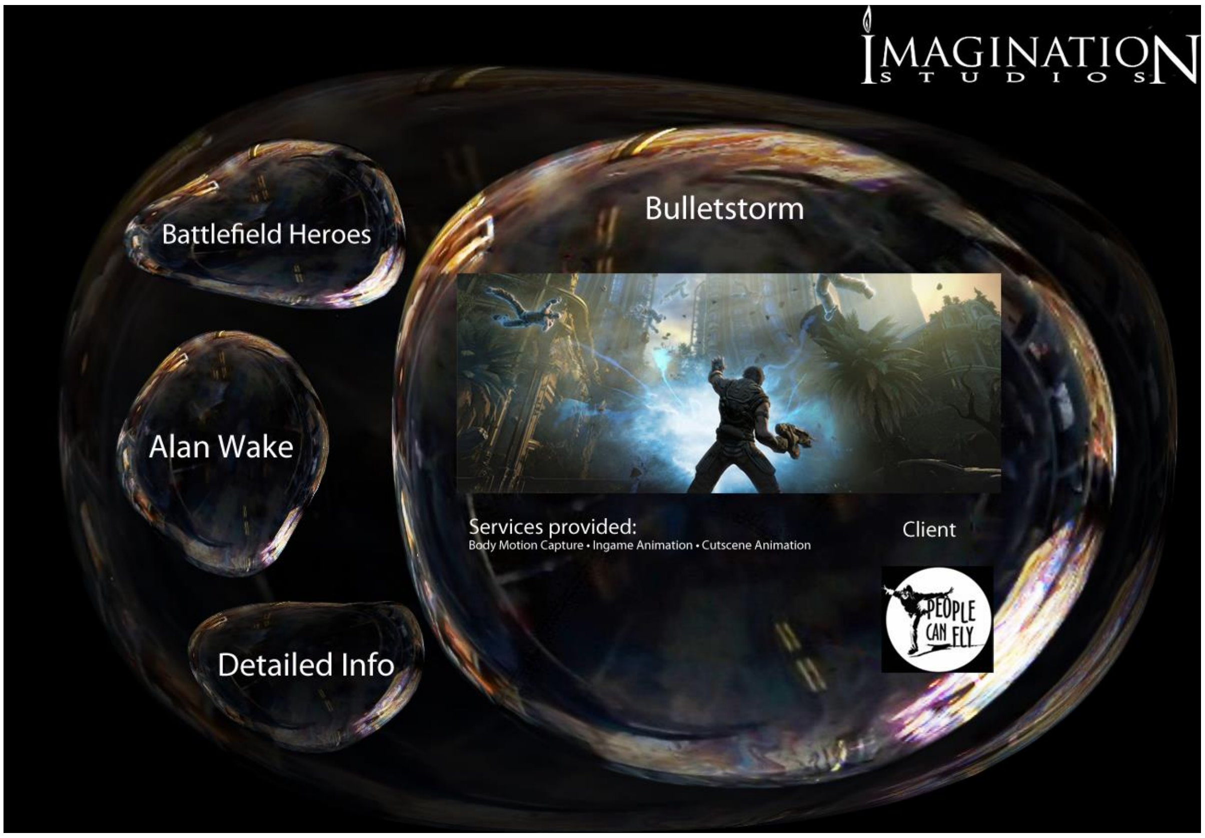

Figure 6: Information within a cell is being displayed

4.8 Contact Information

If a user wants to know more about the topics he has visited so far, he can access the detailed information cell, which will show him a QR-code giving him a link showing more information on everything he has accessed so far. It is also possible to scan a business card, which will then allow IMS to contact the person later. When a business card is scanned, the history of the current user is saved with it to make it easier for the sales department to contact that customer with specific information.

Another way for the user to leave his contact details is to give him an email address using a QR-code as well, to which he could then simply send his vCard. This would not save the history of the current user however.

During implementation the idea for the business card scanner and the personalized link were scrapped in the “Detailed Info” cell were scrapped. Instead, a cell called “Contact” was introduced on the main page, allowing the user to scan the QR-code inside and send his vCard to that email address. This was done to keep everything as simply as possible.

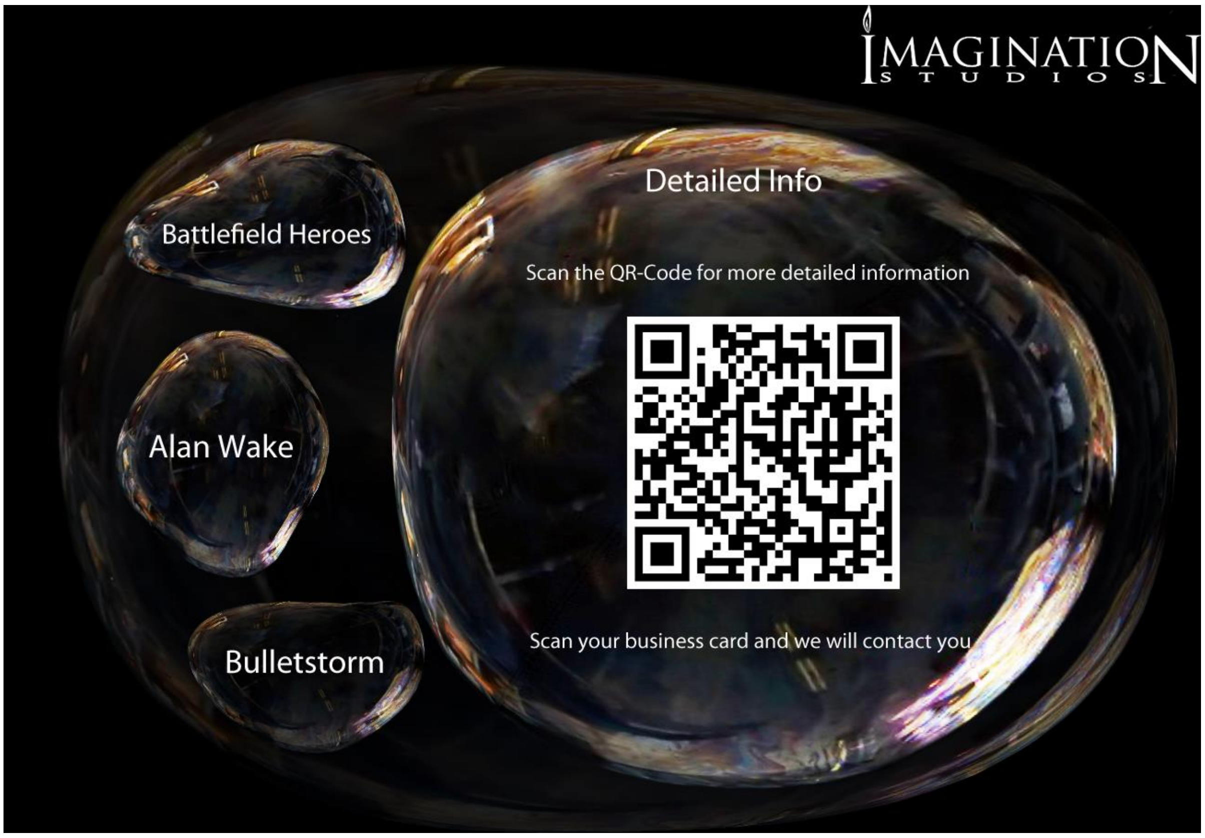
IMAGINATION STUDIOS
Battlefield Heroes
Alan Wake
Bulletstorm
Detailed Info
Scan the QR-Code for more detailed information
Scan your business card and we will contact you

Figure 7: Cell containing QR-code

4.9 Other Developed Ideas

4.9.1 A city of data

The first idea was to organize all the data as a city, where the buildings would represent organizational units, and the inhabitants specific data. The user would be able to walk through the city and interact with any inhabitant he encounters and in that way accessing whatever data this inhabitant was representing. This idea was rejected because while it would be a nice interface when simply wanting to browse random data, it would actually be very hard to implement a quick and easy direct-access system.

4.9.2 The Data Tree

Another idea that was developed during the design process was based on a tree carrying fruit. The tree would have several branches, each branch representing an organizational unit. Each of these branches would carry fruits, which would represent the specific data. A user would be able to both add data to and remove data from the tree. If data was added that did not fit into any of the currently available organizational units, the tree would then grow another branch, allowing the data to be put on it. This type of interface would work very well with a gesture based system and was generally liked when shown to testers. However, the design is more suited towards a general content managing system whereas the project itself was set up as presentation software, which does not require users to be able to add data to the tree.

5. Hardware / Used Tools

5.1 Kinect Hardware

The Kinect Camera was released on November 4, 2010 in North America and November 10, 2010 in Europe. It is a device developed by Microsoft for their Xbox 360 console to allow user input without the need of a controller. It was a huge success, selling more than 8 million units in the first 60 days, giving it the Guinness World Record of being the "fastest selling consumer electronics device". It can both track gestures and recognize spoken commands through the use of a microphone.

The ranged camera technology was developed by the Israeli company PrimeSense. It uses an infrared projector and camera to track movement in 3 dimensions. The infrared laser projects a grid onto the environment which is then used to calculate distances. The image below shows an infrared image of the laser grid. This technology was invented in 2005 by Zeev Zalevsky, Alexander Shpunt, Aviad Maizels and Javier Garcia.

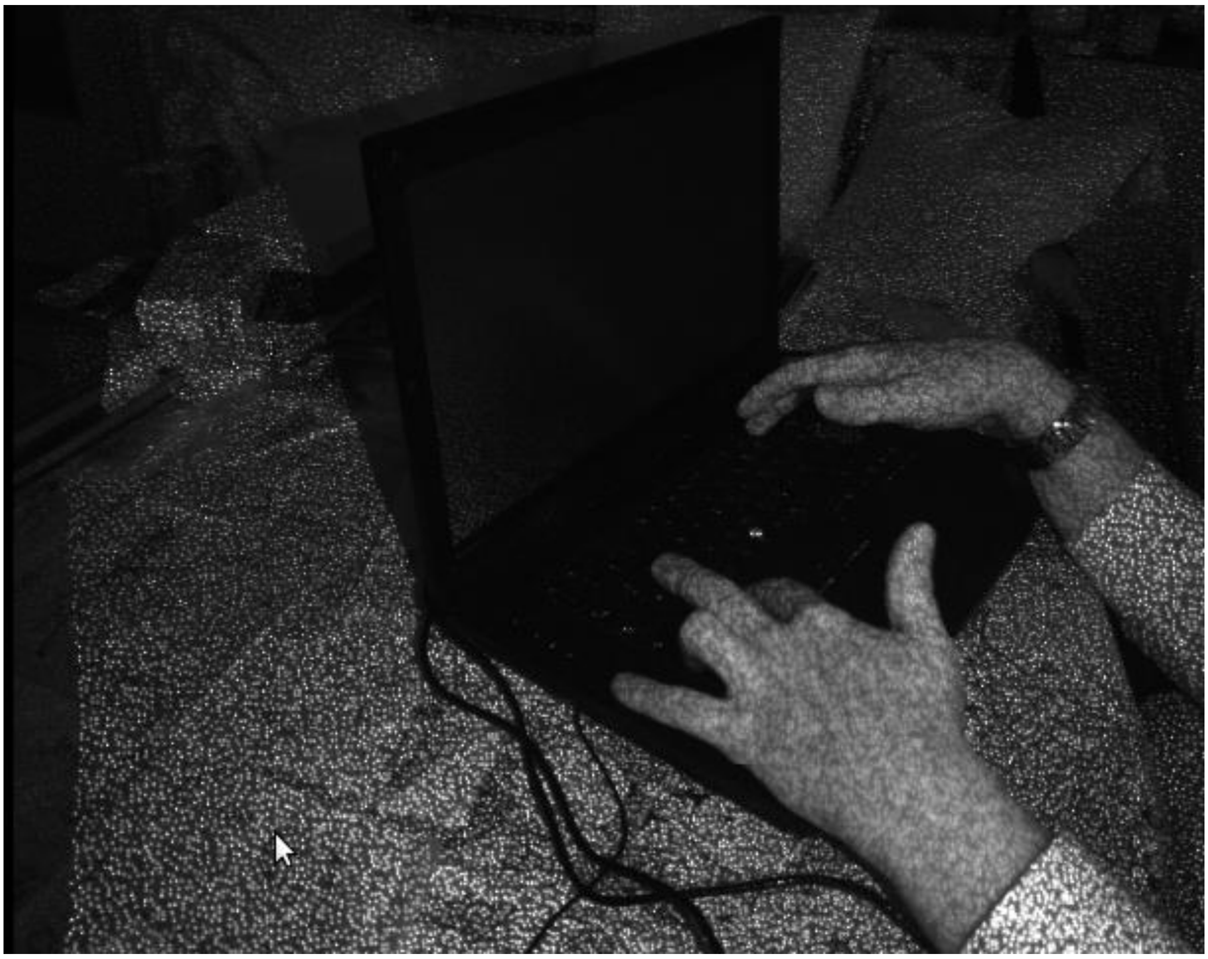

Figure 8: Infrared grid projected by the Kinect

The hardware itself captures videos at a rate of 30 frames per second at a resolution of 640 x 480 pixels. The RGB video stream uses 8 bit, whereas the monochrome depth sensor uses 11 bit depth, allowing for a total of 2048 levels of sensitivity. The picture below explains in detail where which part of the Kinect is located.

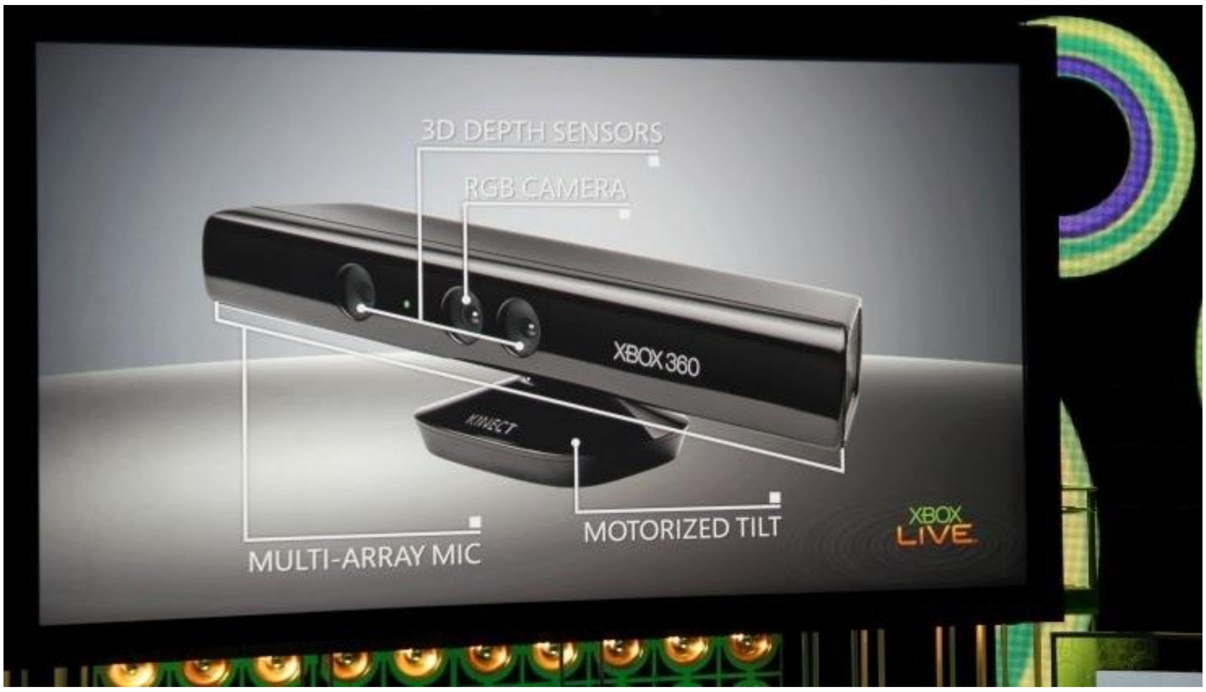

Figure 9: Kinect Camera

5.2 Kinect SDK

The Kinect SDKs make it possible to access various functions of the Kinect hardware within a programming project. Often they also include pre-made functions, allowing the users to use functions that would otherwise potentially require extensive development time. There are several SDKs available for Kinect and during the course of the project a decision had to be made on which one to use. OpenNI, openKinect and the official Microsoft Kinect SDK were considered during this process.

5.2.1 Licensing

As the result of this project could potentially be used as a basis for a commercial program, the type of license each SDK has needed to be considered.

OpenNI comes with the LGPL, or GNU Lesser General Public License. This means that the source code for this SDK is freely available but unlike with the standard GPL, or GNU General Public License, it is not required that applications developed with the SDK need to be open source as well.

The Microsoft Kinect SDK was still in beta stage when the research was conducted. The beta license prohibited any kind of commercial usage and reserved the right for Microsoft to use any application developed with the SDK for advertising etc. free of charge. The cost for a commercial license after release of the SDK was still unknown during the decision phase.

openKinect is released under the Apache20 or GPL license, requiring any application developed with it to also be open source.

5.2.2 Features

OpenNI and Microsoft SDK have very similar features, however OpenNI requires a short calibration phase before a user can use the system, whereas the Microsoft SDK is capable of tracking a user without the need for calibration.

OpenKinect does not have a skeleton tracking feature included.

5.2.3 Decision

OpenNI was finally chosen for this project. The Microsoft SDK was rejected due to uncertainty with future licensing and costs. OpenKinect was rejected for several reasons. Licensing was an important reason, as well as its lack of a skeleton tracking feature. Another reason for rejecting openKinect was its focus on the Linux platform which means that it exhibits instability and performance issues on Windows platforms, which were to be the development platform for this project.

5.3 Game Engine Comparison

There are a great many options to use to actually draw the content on screen. DirectX or OpenGL could be used, requiring a graphics engine to be written from scratch. This in itself would be very time consuming and would more than fill a bachelor thesis all on its own.

Another option is to use one of the many game engines available. Possibilities here include Unreal Development Kit (UDK), Cryengine 3, Unity3D, Ogre3D etc. The first three mentioned game engines are looked at in more detail here. Ogre3D does not have content editors like the other three, so it was omitted from the decision process early.

5.3.1 Cryengine 3

The Cryengine 3 SDK builds upon the Cryengine 3 by Crytek and has in previous versions been used for games such as Far Cry, Crysis and Crysis 2. It is mainly geared towards being used in first-person shooter video games. Crytek offer a large community with special forums geared towards game development with their engine. This offers a large knowledge base and contact with many experienced users to help with any problems that may arise during development.

5.3.1.1 Main use

This engine is specifically geared towards making landscapes; even very large outdoor maps can be created using the editor. This means that whenever a new map is created, the editor assumes that every new level requires terrain, water and sky, neither of which can actually be removed, only edited. This limits the usage possibilities of this engine for other types of games. For example, games set in a space environment or indoors are less suited to this engine.

Any programming done when creating a project with this engine is done in C++; however Lua and XML are also available. A physics engine is included as well, allowing the user to create physical effects within their maps. It is proprietary however, meaning it does not utilize NVidia’s PhysX and therefor does not support hardware accelerated physics.

5.3.1.2 Always online requirement

To use the SDK, an account with Crymod is needed. The user must be logged in at all times, which means that an Internet outage at the user’s end or a service interruption at Crymod’s servers mean that the user cannot work with the SDK. Even though permanent internet access is common nowadays, interruptions can still occur resulting in loss of work time.

5.3.1.3 Licensing

The SDK is free to download and use, giving everybody a chance to evaluate it properly without the risk of potential financial loss if the engine ends up being the wrong choice for the project in question. For educational use and non-commercial projects no additional license is required and it is absolutely free to use for this purpose.

Independent developers that wish to sell their products need to apply for a special Independent Developers License. This is a royalty-only license, allowing these developers to work on their projects without needing initial funding to pay for the license. When the game is on sale, Crytek then requires 20% of the developer's revenues from it.

For normal commercial applications Crytek needs to be contacted for standard licensing terms. These terms are not publically available.

5.3.1.4 Kinect

There is no support for Kinect at the time of writing, and there are no third-party plugins available either, meaning a connection between Kinect and Cryengine 3 would have to be developed from scratch.

5.3.2 Unreal Engine 3

Unreal Engine 3 is developed by Epic Games, and the full game engine package is offered to users in form of the so-called Unreal Development Kit, or UDK with updates being released once a month. It is mainly developed for first-person shooter games, but has been used for other game types as well, including for example MMORPGs. UDK offers support for many platforms including DirectX 9, DirectX 11, Android etc. UDK was first released in November 2009 and has since then built up a massive community. There are many websites available offering almost any kind of thinkable tutorial and there are professionally made video tutorial series on a wide range of topics available on YouTube as well. An extensive wiki explaining all functions is offered too. UDK presumably has the most extensive collection of freely available knowledge available at the time of writing.

Even though UDK gets updated once a month, there are often incompatibilities between different versions when Epic change features or fix certain bugs. Because of this it is highly discouraged to update to a newer version of UDK within the same project. Epic Games is aware of this and all UDK versions since release in November 2009 are still available for download.

UDK is designed to only have one project per installation, as all user-generated content goes into one folder. However, it is perfectly possible to have several UDK installations on one machine to work on several projects simultaneously. The engine comes with a good amount of samples, allowing new users to examine them to learn some basic techniques. Unlike Cryengine 3, it is possible to have empty space in a map to generate a location for space ships to battle, for example.

With the standard license, the user does not have access to the actual source code of the engine written in C++. UDK does offer a special script language called UnrealScript however, which is fairly powerful. Included with the language come a large collection of pre-made classes a user can inherit from to make their own classes.

5.3.2.1 Licensing

UDK is free to use for educational use. When used commercially, a 99 US$ fee needs to be paid, with extra royalties required after the first 50000 US$ in revenue. If used internally in a company, a flat fee of 2500 US$ per year is to be paid.

5.3.2.2 Kinect

Unreal Engine 3 does not have inbuilt support for Kinect, there is however a third-party add-on called NIUI which gives access to the coordinates of all joints of the currently tracked user. It is still beta software and therefore only offers basic functionality.

5.3.3 Unity3D

Unity3D is developed by Unity Technologies. As with the other two Game Engines, the SDK is free to download. It is not aimed at a specific type of game, but can be used for anything from browser-based MMOGs to Roleplaying games. It has a very strong community but does not have quite the same development power behind as the other two game engines have. Often, third-party plugins are required to achieve certain results however, and these are quite often still in beta status and need to be purchased. Programming can be done in JavaScript, C# or Boo. The whole SDK is based on Mono, which is an open-source implementation of the .Net framework developed by Microsoft. Physics are implemented using the PhysX libraries developed by nVidia, allowing the required calculations to be done in hardware on GPUs developed by the same company and by that significantly improving performance over calculations done on a CPU instead. As an example, the website benchmarkreviews.com has run several tests with the game Mafia II where they compare performance using both GPU and CPU calculated PhysX. It becomes very clear that GPU PhysX is vastly superior (Coles, 2010).

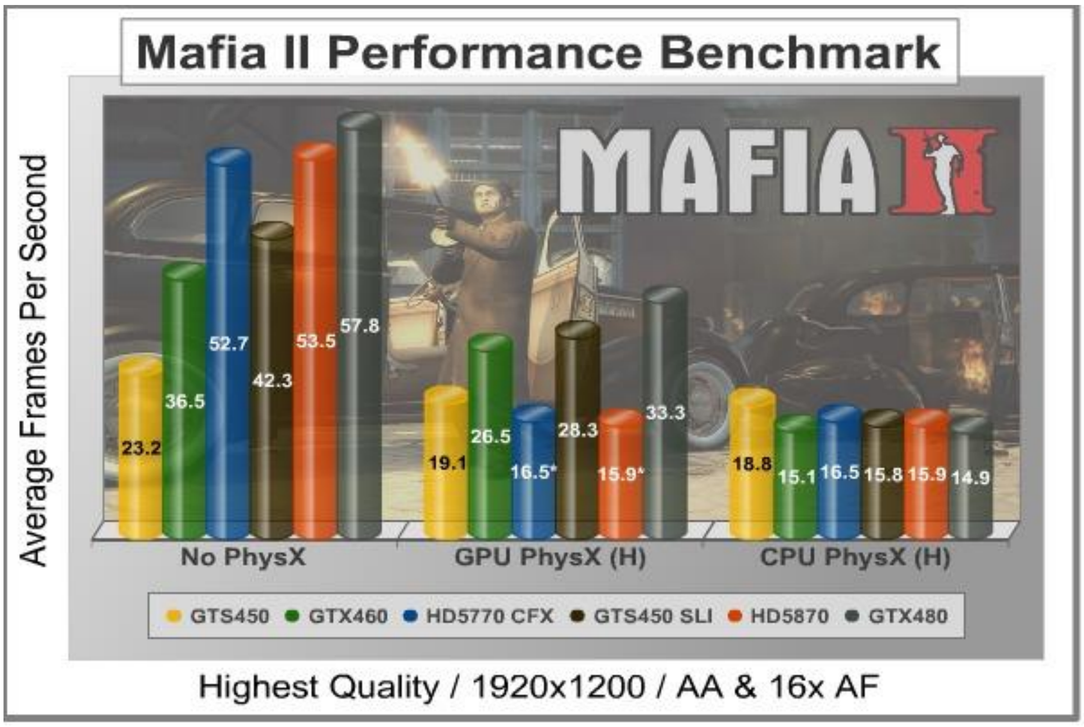

Figure 10: Comparison of GPU vs. CPU PhysX

5.3.3.1 Licensing
Unity offers different licenses where their standard license, simply called Unity, even allows development of games to be sold without any kind of royalties to be paid. However, even this license has limitations, mainly within functionality, but it may also not be used by companies that have had a turnover in excess of US$100000 in their last fiscal year. There are special licenses for Android and iOS as well. Their commercial license is called Unity Pro and offers all possible functionality, apart from the tools for mobile operating systems which need to be purchased extra. Unity Pro comes at a flat fee of US$1500 without any royalties.

5.3.3.2 Kinect
Unity3D already has a good integration with OpenNI allowing the user to start developing Kinect applications with Unity3D quickly and easily.

5.3.4 Decision
Taking everything into account, using Unity3D would have been the preferred choice to develop this prototype. Especially the most advanced implementation of a Kinect interface would have been extremely beneficial to the project. However, as Imagination Studios has good contacts with the developers of UDK the decision was made to use the Unreal Development Kit instead of Unity3D.

5.4 Tools

5.4.1 nFringe
NFringe is an add-on for Visual Studio 2010 developed by a company called Pixel Mine Games. It integrates seamlessly with Visual Studio and allows the user to develop applications in Unrealscript within this development environment.

NFringe offers basic Intellisense support, giving the user suggestions on what he might need next while he is typing. This support is useful, but unfortunately far from perfect. Not all possible options are always detected which can be confusing especially when the actually needed option is not showing up in the list of suggestions. Certain mistakes made during typing may also generate an error message which is disruptive, but does not hinder the function of the addon itself.

NFringe also offers syntax highlighting, making it much easier to read source code and detect obvious mistakes.

The most important fuction of nFringe however is the possibility to use the powerful debugger included in Visual Studio for running Unrealscript projects. While Intellisense and syntax highlighting can also be configured to work with a simple text editor like Notepad++ (Ho, 2011) even without the need for nFringe, actual runtime debugging is not possible with the tools UDK offers. Being able to set breakpoints and watch variables change values during runtime is often invaluable while trying to fix slightly more complicated bugs and errors.

5.4.2 Visual Studio
Visual Studio is a development suite produced by Microsoft. It offers support for many languages and allows for easy management of very large development projects. It also offers a powerful debugger

allowing easy real-time debugging with watch lists and breakpoints. One of the key features of Visual Studio is its Intellisense which aids the programmer while writing source code by offering suggestions on possible keywords and also showing what parameters and datatypes are used when calling a specific function or method. These features greatly increase productivity.

For this project, Visual Studio was chosen due to the possibility to write UnrealScript code with full Intellisense support with it with help of the aforementioned nFringe add-on.

5.4.3 Adobe Photoshop

Photoshop by Adobe is considered to be the standard program for image manipulation and editing. It has many very powerful features making possible to edit images in such a way that it is not possible to notice any editing. During this project, Photoshop was used to create the collages that show the final design.

5.5 Special techniques used during development

5.5.1 Gesture Recognition

Recognizing gestures is a very complicated process. OpenNI, which is the Kinect SDK that is being used, has inbuilt gesture recognition. However, NIUI, the beta software used to connect UDK and Kinect does not offer access to these functions, which means that a gesture recognition system had to be implemented from scratch. Gestures have many different components that need to be considered.

- Spatial component: Each gesture has several key locations within a defined space. With Kinect, each of these key locations has 3 dimensions.
- Temporal component: A gesture is not instant, but rather happens over a certain period of time.
- Absolute vs. relative movement: It is not enough to simply compare the coordinates of a certain joint at different intervals to detect a gesture. For example, and hand can be moved in different ways. One way would be to use an arm to push the hand forward and pull it backward. Another way would be to keep the arm steady, but move the torso forward, or even keep the torso straight and simply take a step forward. All these movements would result in the same change of coordinates for the hands, but they might mean completely different things.

NIUI implemented a function that allows to read the current coordinates of each joint at any time. It also automatically corrects the aforementioned problem of absolute vs. relative movement. This is done by assuming that only movements that are in relation to the rest of the body are important and by ignoring movement of the whole body. If applied to the example above, when taking a step forward the coordinates for the hand do not change, when moving just the hand however, coordinates do change. This behavior, while convenient for normal gestures, prevented implementation of a gesture that required the user to take a step towards the screen, as this simply would not be recognized.

There are a couple of key points that are important for the implementation of gestures.

- A certain threshold for the movement amount needs to be defined, as it is impossible to hold a hand completely still. Kinect is very accurate at detecting movements, so even slight shifts would immediately trigger a gesture, which is not a desired effect.
- This movement also needs to happen within a certain timeframe because otherwise random movement could trigger a gesture after several seconds, potentially activating a function that the user had no intention of using.
- If movement stops, the gesture must be reset after a certain period of time as the system would never go back to a state where no gesture is recognized.

Biswas and Basu (Biswas & Basu, 2011) describe a very robust and easily adaptable system for gesture recognition using a multi class Support Vector Machine and histograms to determine the differences between two frames and with this method detect any gesture that the system has been trained for. However, due to limitations in both UDK and the NIUI SDK, it was not possible to implement this method. Lai, Konrad and Ishwar also confirm that a method based on machine learning would be more flexible and robust. They write:

“We believe that gesture recognition based on machine learning algorithms with suitably selected features and representations are likely to be more flexible and robust” (Lai, et al., 2012)

The currently implemented method uses hard coded thresholds and functions to determine gestures. While this method is robust and functional, it is also very difficult to add new gestures, as each new gesture would need to be coded and then tested to find the correct threshold, making this a time consuming task. This is also confirmed by Lai, Konrad and Ishwar when they write:

“…, using a fixed set of parameters (thresholds) makes the insertion of a new action/gesture difficult” (Lai, et al., 2012)

5.5.1.1 Push and Pull Gestures

These two gestures were the first to be implemented in the system. They both work in essentially the same way and are distinguished by their opposite movement direction, so only the push implementation will be discussed.

Push is essentially movement of the hand towards the screen. To detect if the user is actually performing a gesture as opposed to simply moving randomly, a history of the last hand coordinates is kept. This history is checked at every tick of the game engine, and if the history shows a large enough difference in distance, it is assumed that the user wants to perform the push gesture.

5.5.1.2 Pinch and Spread Gestures

These gestures are implemented in a very simple way. During every tick of the game engine, the distance between both hand indicators is calculated. As with the push and pull gestures, a history of previous distances is kept. If the distance between both hand indicators is growing, a spread gesture is assumed, whereas a shrinking distance is assumed to be a pinch gesture.

5.5.2 Detecting if hand indicators are hovering over a bubble.

One of the most important functions of the interface is detecting if the hand indicators representing the user’s hands are hovering over one of the bubbles currently showing on screen. This will often indicate that the user wants to interact with that bubble in some way, meaning this detection needs to work quick and very exact. Any misdetection happening here would result in unexpected behavior and confuse the user.

The first problem presenting itself is the fact that the whole system is based in a three-dimensional room. This means the bubbles are spheres placed at certain coordinates, and the hand indicators are planes that are positioned in front of all other objects within this room to prevent them from ever being hidden behind an object, as that would be confusing for the user. What is seen on screen is in fact the image captured by a virtual camera also positioned in the world at a specific set of coordinates. An issue concerning perspective comes into play here however. If a bubble is not exactly centered on screen, a hand indicator may appear to not hover over a bubble while a comparison of their coordinates in fact indicates that it is hovering.

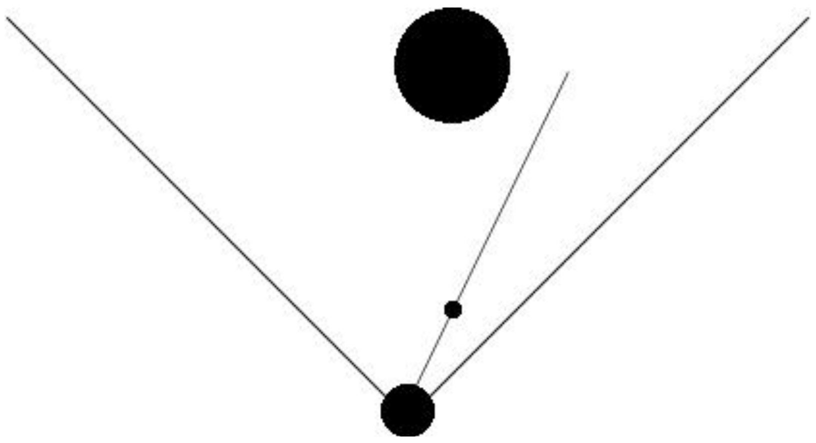
Figure 11: The problem with perspective

This problem is illustrated in the picture above. The medium sized dot at the bottom represents the camera, the large dot at the top is one of the bubbles on screen and the small dot in the middle is one of the hand indicators. The long angled lines to the left and right represent the Field-of-View of the camera. The line in between represents the view vector from the camera towards the hand indicator. As can be seen, the vector misses the actual bubble, creating the illusion that the hand indicator is to the right of the bubble, even though the top-down view reveals that it is actually right in front of it.

To alleviate this problem, the 3D coordinates of both the bubble and the hand indicator need to be projected to two-dimensional screen coordinates. Because perspective can make objects appear smaller than they actually are, the radius of the projected bubble also needs to be calculated. When all these calculations have been done, the distance of the projected center of the bubble and the projected location of the hand indicator can be calculated. This distance is then compared to the size of the projected radius. If the distance is smaller this means the hand indicator is in fact hovering over this specific bubble. If the distance is larger, the hand indicator is not hovering.

This basic algorithm needs to be extended however, as the nature of the system means that this algorithm may sometimes return ambiguous and unpredictable results. A bubble can contain other bubbles, and when hovering over a bubble that is within another bubble this algorithm would return a positive result for both of them. So a function that decides which of these results is the required one is needed.

The flowchart below explains the implemented algorithm in more detail. First, the coordinates of both hand indicators are projected to 2D screen coordinates. Then, a list containing all current bubbles is checked to determine if a hover test needs to be done on them. If there are, the algorithm described above is applied to check if one or both of the hand indicators are currently hovering over this bubble. If a hover is detected, another check is then applied to determine if this bubble contains other bubbles. If so, these bubbles are then added to the end of the list of bubbles to check for hovering. As these internal bubbles are not always on screen, they are only temporarily added to the list and this only when they belong to a bubble that is currently at maximum zoom level and actually showing its contents. If it is then decided that no hand indicator is actually hovering over these bubbles, they are removed from the list again.

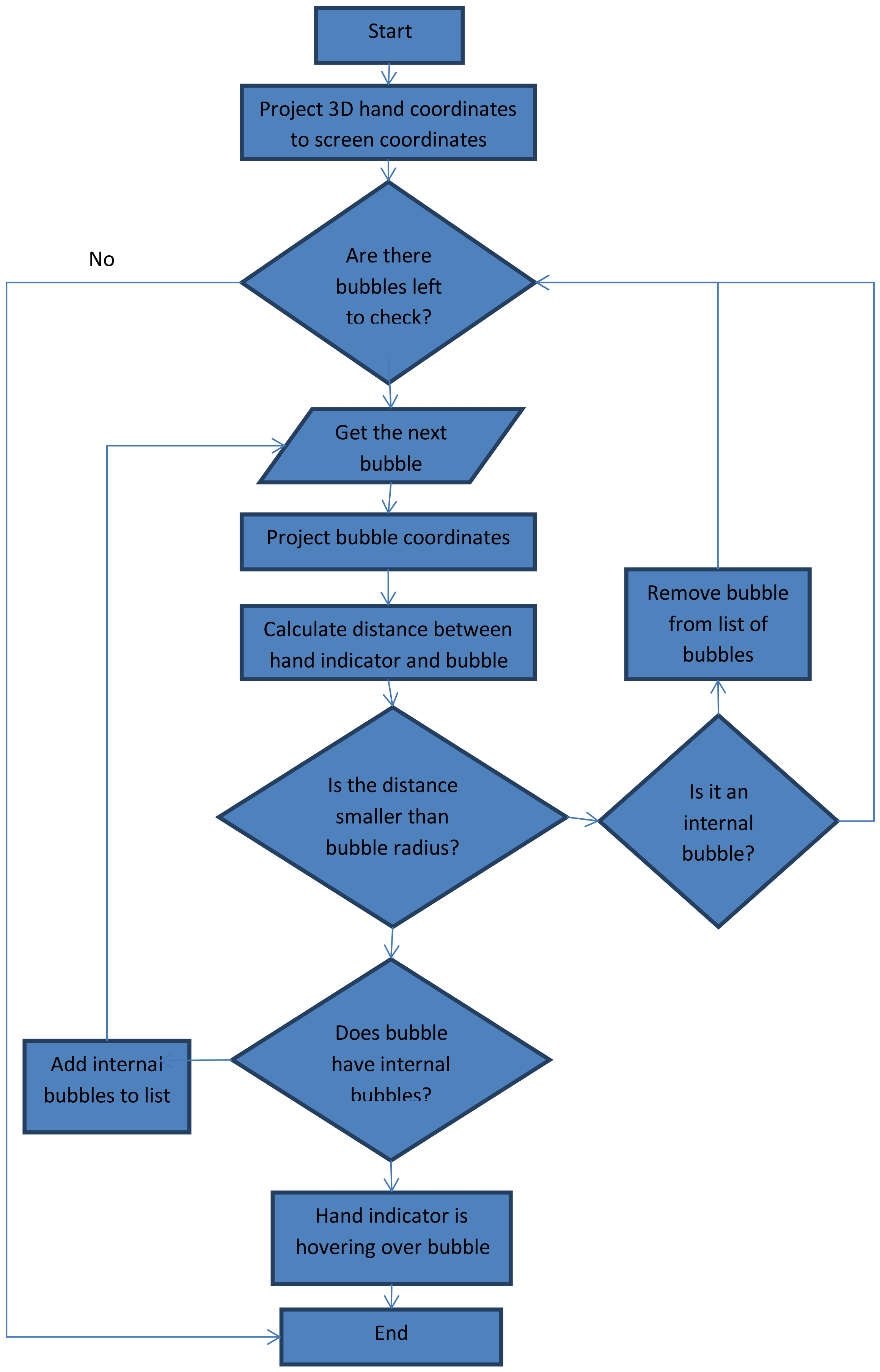
Start
Project 3D hand coordinates to screen coordinates
Are there bubbles left to check?
No
Get the next bubble
Project bubble coordinates
Calculate distance between hand indicator and bubble
Is the distance smaller than bubble radius?
Is it an internal bubble?
Remove bubble from list of bubbles
Does bubble have internal bubbles?
Add internal bubbles to list
Hand indicator is hovering over bubble
End

Figure 12: General hover detection algorithm

5.5.3 Using opacity maps for hover effects

Whenever a bubble is not used, it only shows a single letter to indicate what type of content it contains, as shown in the picture. To prevent a user from having to guess what it means, a function has been implemented that adds the full title of that bubble whenever a hand indicator hovers over this bubble, as shown in the second picture.

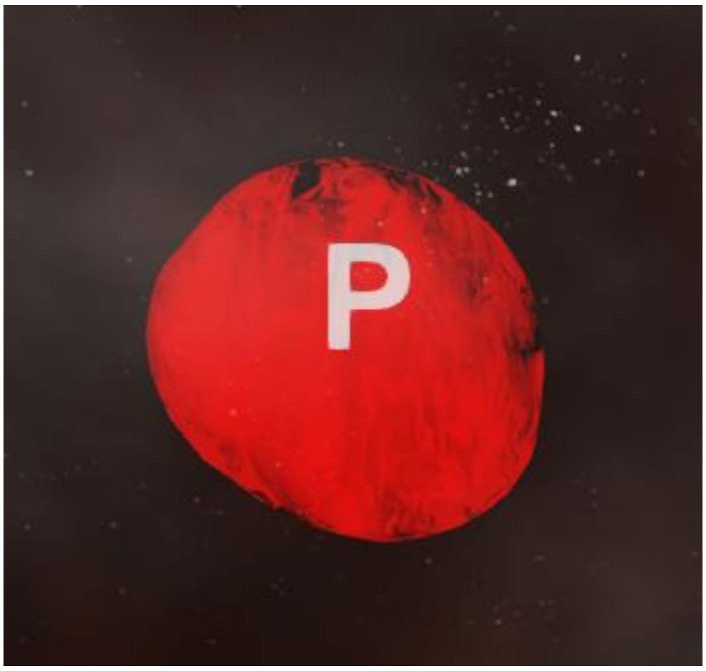

Figure 13: Bubble without hover effect

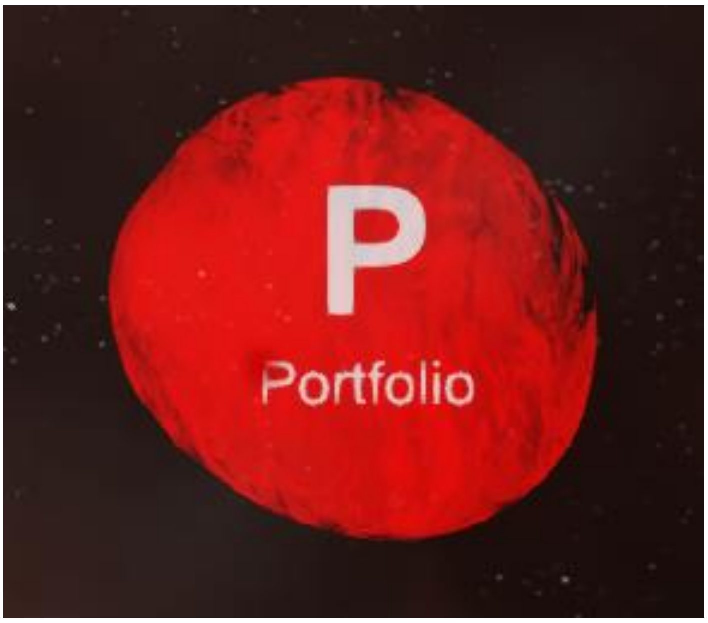

Figure 14: Bubble with hover effect

To achieve this effect, an opacity map is used which defines which parts of a texture are transparent, and which parts are not. Both the opacity map and the texture used to display the text are shown here. The black background of the picture is not part of the original texture, it is usually transparent. The background has been added to allow presentation within this document.

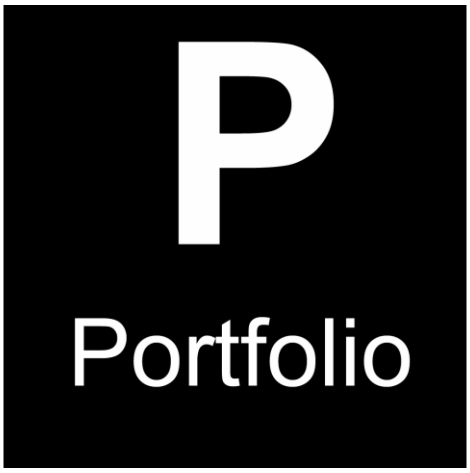

Figure 15: Texture holding all text

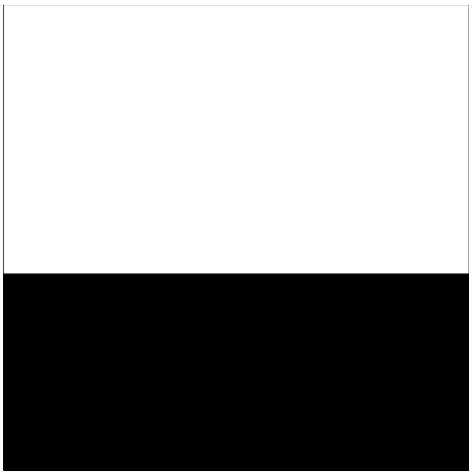

Figure 16: Opacity Map

Opacity maps define what parts of a texture should be opaque, usually represented by the color white, and which parts should be transparent, usually represented by black. Looking at both textures above it can easily be seen that adding the opacity map to the text texture would result in making the lower part containing the word “Portfolio” transparent. So by dynamically adding and removing the opacity map, the desired hover effect can be achieved.

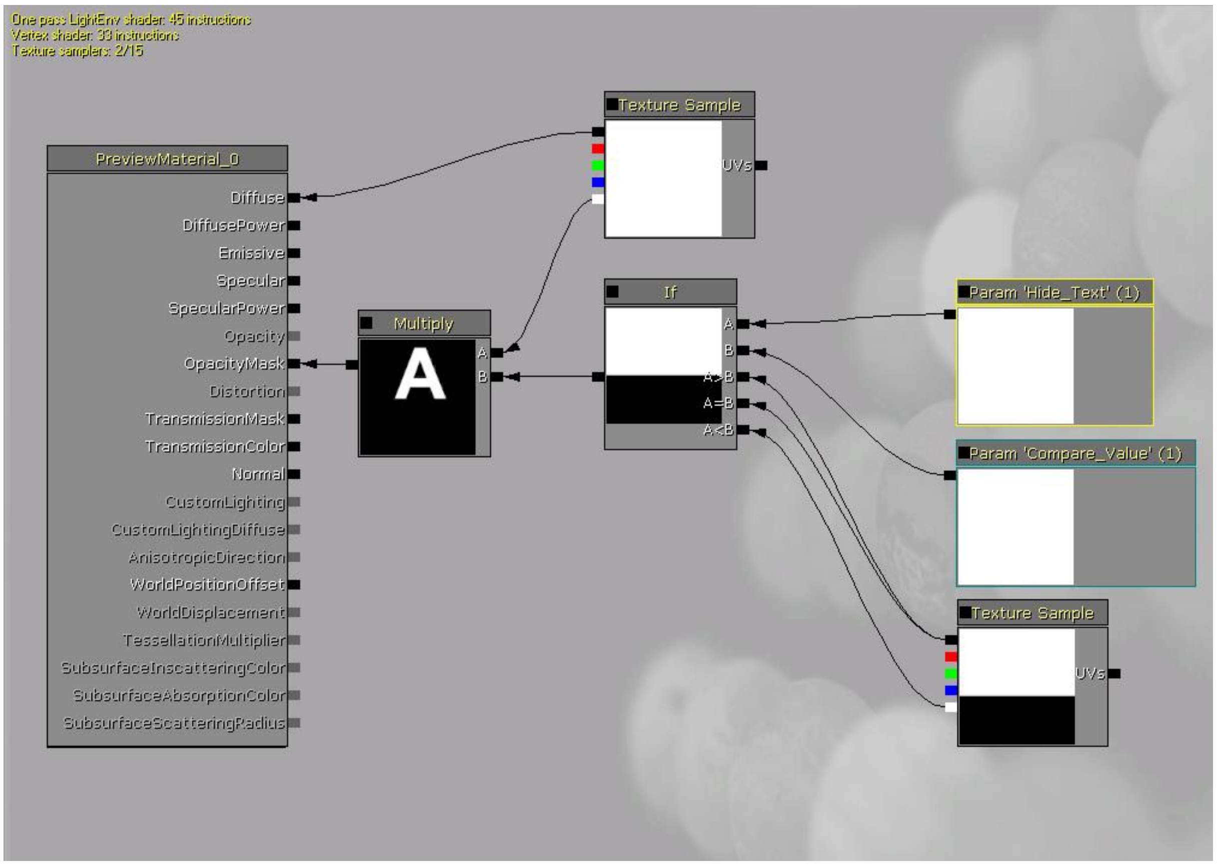
One pass LightEnv shader: 45 instructions
Vertex shader: 33 instructions
Texture samplers: 2/16

Figure 17: Shader program used for hover effects

This picture above shows the complete shader program created within UDKs material editor. Important to know is that RGBA values in UDK have a range between (0.0, 0.0, 0.0, 0.0) and (1.0, 1.0, 1.0, 1.0). Multiplying a texture with a black texture will therefore always generate a black texture, and multiplying a texture with a white texture will not change the initial texture at all.

The top texture sample that is appearing white contains the texture that holds all the text. Its RGB channels, represented by the black box, are connected to the "Diffuse" channel of the material. Its transparency channel is multiplied with the opacity map which then generates the final opacity map that is connected to the "OpacityMask" channel of the material. The original opacity mask is altered depending on the value of the parameter "Hide_Text". Whenever its value is the same or larger than the parameter "Compare_Value", the normal RGB channels of the texture are multiplied with the transparency channel, making the lower part black, or transparent, in the process and hiding that part of the text. When the parameter value is lower however, the transparency channel of the opacity map is used instead. Because this texture is completely opaque, this means that its transparency channel is actually completely white. If this texture is then multiplied with the transparency channel of the other texture, no change is made at all and the whole texture including the previously hidden text is shown.

5.5.4 Bubbles and Animations

During the initial implementation phase, the bubbles on screen were created by using a standard static mesh provided by UDK. This made for some problems later on, as static meshes cannot be animated properly because they are, as their name suggests, static. As zoom levels of static meshes can be altered in all three dimensions, animating the static bubbles in this way was attempted first, but this proved to be an unreliable and ugly implementation. Animation was desired however, as the bubbles are inspired by soap bubbles and having them animated would look much more lifelike and interesting.

Instead, Imagination Studios supplied an animated bubble combined with a skeleton modeled in Autodesk Maya 2011. Epic games provide plugins for both Autodesk Maya and Autodesk 3D Studio MAX to help export models and animations into a file format that can then be imported into UDK. This plugin is called ActorX and can be downloaded free of charge. The exported animation and skeletal mesh were imported into UDK and combined there, making the bubbles much more interesting to look at.

6. Test and Evaluation

This section will be relatively short. This is due to the fact that the main focus of this project was to create a new type of user interface and the development of a conceptual prototype afterwards. Testing was therefore a low priority during this project.

During development and testing it was quickly noted that a gesture system like the one implemented, where a user needs to hold his arms and hands up is tiring very quickly and is only suitable for applications that are only used during a short period of time. To alleviate this problem, the gestural system was adjusted to allow the user to keep his hands and arms closer to their body, thus reducing the fatigue experienced by using gestures.

A second issue with the gestural recognition system was an unwanted detection of gestures when people would let their arms and hands drop to their sides while looking at the content of a cell. The system would often mistake the movement for a tap gesture, making everything on screen behave erratically. A temporary solution involving the tweaking of gestural parameters was implemented, but the end result was not satisfactory. Further development would need to look into this problem in more detail to find a proper solution.

Another issue was that it was not immediately obvious to people that they needed to tap a cell if they wanted to activate it. Even though this is essentially the same gesture required as on a touch screen, the different way of interacting nonetheless seemed to make this gesture less obvious. A solution for this problem has not been found and should be addressed during further development of the system.

7. Conclusion and Outlook

This chapter will contain a conclusion and an outlook on how the project could be improved upon.

In general, the project turned out satisfactory. The whole process from the first design sketches with pencil in paper up to seeing the finished product on screen and being able to interact with it was very fun and educational. Doing a proper design process has really helped the project along, allowing for a much clearer goal. This has always been helpful to keep focus during the actual development phase, and also making it more effective. Without the design process the design might have been changed or completely redone in the middle of the development, which would have slowed down the process and potentially prevented the project from reaching the status it has now.

The research on both technical and legal issues of the project was very interesting as well, allowing a much deeper insight into the whole world of software, or more specifically game, development. Legal issues especially were never a concern before, so learning to read and understand license agreements was a completely new experience. The research into all technical issues offered a lot of new insight especially into tools and game engines.

Towards the end it became clear however that the simple approach to gesture recognition used in the project only offered basic functionality. User testing often exhibited unintended activation of certain gestures, confusing the user in the process. Only at this point did it become obvious that more research time for proper gestural recognition should have been allocated at the beginning of the project.

Further development should focus on two key issues, gestural recognition and easier content management.

The gestural recognition system would probably need to be re-implemented in a different way. A possible solution would be to implement a system using a support vector machine as done by Biswas and Basu (Biswas & Basu, 2011). Using statistics and pattern recognition, this would not only allow for a much more robust gesture recognition, but also for an easy way to add new gestures, as the current system requires extensive programming work to add new gestures.

Another issue with the project is adding new content. This problem results from using UDK, as it requires content to be added before runtime using the included editors. This can make fixing even small mistakes like typos into a larger process. A function that would allow for content to be loaded dynamically at runtime without the need for using the editor would make the software a lot more usable in day to day operations.

8. References

Biswas, K. K. & Basu, S. K., 2011. *Gesture Recognition using Microsoft Kinect.* Wellington, IEEE.

Coles, O., 2010. *benchmarkreviews.com.* [Online]
Available at:
http://benchmarkreviews.com/index.php?option=com_content&task=view&id=600&Itemid=38&limit=1&limitstart=3
[Accessed August 2012].

Crytek GmbH, 2012. [Online]
Available at: http://mycryengine.com
[Accessed 2012].

Crytek GmbH, 2012. *Game Development License.* [Online]
Available at: http://mycryengine.com/index.php?conid=43
[Accessed 2012].

Epic Games Inc., 2012. *Mesh Export Tools.* [Online]
Available at: http://udn.epicgames.com/Two/ActorX.html
[Accessed April 2012].

Epic Games, Inc., 2010. [Online]
Available at: http://www.udk.com
[Accessed February 2012].

Epic Games, Inc., 2012. *Showcase.* [Online]
Available at: http://www.unrealengine.com/showcase
[Accessed May 2012].

Free Software Foundation, Inc., 2007. *GNU Lesser General Public License.* [Online]
Available at: http://www.gnu.org/licenses/lgpl.html
[Accessed November 2011].

Guinness World Records, 2012. *Kinect Confirmed As Fastest-Selling Consumer Electronics Device.* [Online]
Available at: http://community.guinnessworldrecords.com/_Kinect-Confirmed-As-Fastest-Selling-Consumer-Electronics-Device/blog/3376939/7691.html
[Accessed May 2012].

Ho, D., 2011. *Notepad++.* [Online]
Available at: http://notepad-plus-plus.org/
[Accessed November 2011].

Kalogiros, P., 2012. *Kinect JS, Kinect plus Javascript.* [Online]
Available at: http://kinect.childnodes.com/
[Accessed 29 May 2012].

Kinect Hacks, 2012. *Cauldren.* [Online]
Available at: http://www.kinecthacks.com/cauldren-gestural-performance-piece/
[Accessed 29 May 2012].

Kinect Hacks, 2012. *Kinect Hacks.* [Online]
Available at: http://www.kinecthacks.com

Kinect Hacks, 2012. *Kinect-powered Augmented Reality Sandbox.* [Online]
Available at: http://www.kinecthacks.com/kinect-powered-augmented-reality-sandbox/
[Accessed 29 May 2012].

Kinect Hacks, 2012. *Protecting The Secret – An Innovative Kinect-Powered Coke Tour.* [Online]
Available at: http://www.kinecthacks.com/protecting-the-secret-an-innovative-kinect-powered-coke-tour/
[Accessed 29 May 2012].

Lai, K., Konrad, J. & Ishwar, P., 2012. *A gesture-driven computer interface using Kinect.* Boston, IEEE.

Microsoft, 2012. *Kinect for Windows.* [Online]
[Accessed 2012].

Microsoft, 2012. *Microsoft News Center.* [Online]
Available at: http://www.microsoft.com/en-us/news/press/2010/mar10/03-31PrimeSensePR.aspx
[Accessed May 2012].

onethought, 2011. *NIUI: OpenNI / Kinect API for UDK [Beta].* [Online]
Available at: http://forums.epicgames.com/threads/765636-NIUI-OpenNI-Kinect-API-for-UDK-Beta
[Accessed November 2011].

Pixel Mine, Inc., 2010. *nFringe.* [Online]
Available at: http://pixelminegames.com/nfringe/
[Accessed 2012].

Saffer, D., 2009. *Designing Gestural Interfaces.* 1st ed. Sebastopol: O'Reilly Media, Inc..

The OpenNI Organization, 2011. [Online]
Available at: http://www.openni.org
[Accessed November 2011].

Torus Knot Software Ltd, 2009. [Online]
Available at: http://www.ogre3d.org/
[Accessed November 2011].

Unity Technologies, 2012. [Online]
Available at: http://unity3d.com/
[Accessed 2012].

Vera, L., Gimeno, J., Coma, I. & Fernández, M., 2011. *Augmented Mirror: Interactive Augmented Reality System Based on Kinect,* València: Springer Verlag.

Wikimedia Foundation, Inc., 2012. *Kinect.* [Online]
Available at: http://en.wikipedia.org/wiki/Kinect
[Accessed May 2012].

World Intellectual Property Organization, 2007. *1. (WO2007043036) METHOD AND SYSTEM FOR OBJECT RECONSTRUCTION.* [Online]
Available at: http://patentscope.wipo.int/search/en/WO2007043036
[Accessed May 2012].