Final Report

Abstract

*Enigma Machines were used in the main during World War II by the German military. It was a device which scrambled plain text into ciphered text. This project demonstrates both the enciphering and deciphering of the Enigma machine in a graphical simulation. The simulation demonstrates the movement of the rotors, within the machine, which presents a 3D graphical visualisation of the process of encrypting plain text into cipher text.*

Intro

With the outbreak of wireless communication in the early 1900s, there was a necessity for secure communication, particularly for military. With this came the invention of an Enigma machine in 1918, invented by a German engineer, Arthur Schebius, later the enigma machine patented in 1919. In the 1920s early models were used commercially, and later adopted by Nazi Germany before and during World War II. The Enigma machine was an electro-mechanical device which scrambled a plain text message into ciphered text using a letter substitution system. This enabled the military forces to communicate using coded messages.

In this project a graphical Enigma simulator was developed, which represented the inner working of the process of encryption, plain text to cipher text, as well as the process of decryption, ciphered text to plain text. The simulator will provide a greater detail of the processes in a 3-Dimensional graphical format. The simulation, developed by C++ language, also allowing users to encrypt their own text. In the background section, some historical information about the Enigma machine are presented, which enhance the understanding of Enigma machines.

Background

History

The Enigma machine was invented by a German engineer in 1918 and later adopted by Nazi Germany before and during World War II. The Enigma was a device used by the Germans to communicate with their allies using encrypted messages. The enigma consisted of a keyboard of 26 letters in the pattern of the normal German typewriter, but with no keys for numeric or punctuation characters. Behind the keyboard was a lamp board made up of 26 small circular windows, each bearing a letter in the same pattern as the keyboard, which could light up one at a time. Behind the lamp board was the scrambler unit consisting of a fixed wheel at each end, and a central space for three rotation wheels. Message were limited to a maximum of 250 letters to avoid the inner mechanism returning to the same position because the sequence would repeat itself after 17,576 (26x26x26) key rings. Had the messages not been limited then British code-breakers may have been able to break the encrypted messages. Thus potentially the number of cipher text alphabets was vast and this led German military authorities to believe in the absolute security of this cipher system [1]. The first Enigma machine was heavy and bulky.

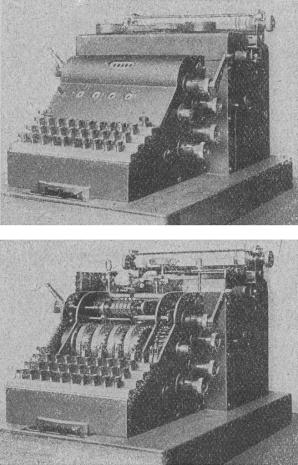


Figure 1 – Enigma A (Crypto Museum, 2008)

Various versions of the Enigma machine were developed, each with varying rotors. In 1926, a commercial version of the Enigma machine was purchased by the German Navy and adapted for military use. A special Enigma was developed by Chiffriermaschinen-AG, which had rotors that have the same contact alignment as the D rotors, but with teeth, multiple notches and are advanced cog wheels instead of pawls and ratchets. This model lead to the Enigma G. The Enigma G had different rotors with a zigzag pin placement and the counter on its right. Its rotors, which also had multiple notches, were moved by a system of gears.

In 1932 the Wehrmacht revised the commercial Enigma D and added the plugboard at the front of the machine. This version, known as Enigma I, became known as the Wehrmacht Enigma and was introduced on a large scale in the Army and public authorities. Initially this enigma came with three rotors, however from 1939 onwards they were equipped with five rotors. The Wehrmacht model was later adopted by the German Navy, with its securer plugboard and the extended set of rotors of eight. [2]

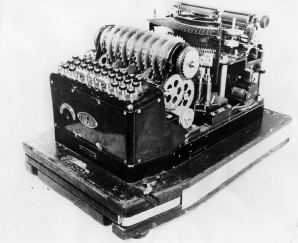


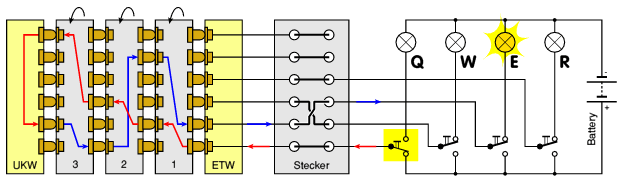
Figure 1-1 – The Wehrmacht Model (Crypto Museum, 2009)

Breaking the code

The Germans believed that the messages being sent to their allies were not breakable. However, the code breakers based at Bletchley Park cracked the secret messages being broadcasted, which played a crucial role in the defeat of Germany. The Polish were the first people to come close to cracking the Enigma code. Marian Rejewski, Henryk Zygalski and Jerzy Rozicki were three mathematicians who successfully cracked the Enigma. They also developed an electro-mechanical machine, called the Bomba, to speed up the code breaking processing. [2] With the invasion of Poland looming, the Poles shared their information with the British, who in turn established the Government Code and Cipher School at Bletchley Park, however it was only in 1941 where their work began to pay off meaningfully when they were able to gather evidence of the planned invasion of Greece. [3]

Bletchley Park

Bletchley Park is the home of the Government Code and Cipher School (GC&CS) based in Milton Keynes, UK, where the Enigma was initially broken. This location was chosen as it is 45 miles north of London with direct railway connections to here, as well as to Cambridge and Oxford, which allowed scientists and army personnel to travel inconspicuously. Alan Turing developed the Bombe, not to been confused by the Bomba which it was in fact based on. He developed a more universal method based on cribs, pieces of guessed plain text, due to the Polish method of exploiting the German vulnerability of the double-enciphered message indicator which could no longer be used. The value of this codebreaking machinery was recognised by the British Prime Minister, Winston Churchill, who introduced a new level of secrecy to supersede all other levels, known as ULTRA. Three additional rotors used exclusively by the Navy and not shared with any other parts of the army. In 1941 Turing discovered the procedure of the additional wheels. [4]

Figure 1-2 Circuit Diagram (Crypto Museum, 2009)

Rotor Wiring

Each rotor had 26 positions, one for each letter of the alphabet. After a key has been pressed the rotor rotates so the next letter is visible. If the letter ‘A’ was active on the key press then on the next key press the letter ‘B’ will be active. Once a full cycle is complete the next rotor will rotate one notch. Once the first rotor reaches the letter ‘Z’ it will ensure the letter ‘A’ is active for the next key press but also if the next rotor was on the letter ‘Q’ then the letter ‘R’ would be active during the next key press. The same would occur for the third rotor once the second rotor has completed a cycle. This results in 17,576 (26 x 26 x 26) possibilities. In addition to the rotors there was a reflector (Umkehrwalze) added on the end and a plugboard (Steckerbrett or Stecker) was introduced to the first Wehrmacht version of the Enigma machine. The reflector redirected the current back to the rotors by a different route. With the exception of the beta and gamma reflectors, each letter was pair with one another. For example ‘E’ and ‘Q’ were paired together on reflector B, so when the rotor passed the current to ‘E’, ‘Q’ would be passed back to that rotor. The plugboard added an extra layer of complexity to the Enigma machine. It was situated at the front of the machine and enabled the key press to map to a different letter on the rotor.

Related Work

We are not aware of any simulators which represent the inner works of an Enigma machine in a 3D graphical representation. However various simulator which encipher and decipher text are available widely on the internet. One simulator in particular caught the attention. [5] It demonstrates the current path when a key is pressed through three rotors, in a simple form. This simulator provided a deeper understanding on how a simulator for this project could be developed.

Specification

*Project Management*

Project Specification

The purpose of this project is to develop a Graphical Enigma Simulator which will demonstrate the process of encryption and decryption. A particular aim is to visually demonstrate the principle of polyalphabetic substitution in operation in the rotors.

Project Plan

A plan of how much time to allocate to each aspect of the project was essential to facilitate good time management and to provide an understanding of how the project is progressing. The aspects of the project are as follow:

Background research

Requirement analysis

Project design

Ethical Approval

Code Implementation

Testing

Interim report

Evaluation  
Final report and Portfolio

A Gantt chart, which is shown in Appendix 1, was created with task list and timescales to ensure the developer could complete tasks on time.

Methodology

Throughout the development of the project the iterative development cycle was utilised. Waterfall and agile methodologies were considered however neither were suitable for this project because the waterfall approach did not allow revision to the project and as for the agile approach not enough information on the structure of the simulator was known at that stage. The iterative approach allowed revision of other parts of the project in stages.



Figure 2 – Iterative model (Voltreach, 2011)

Source Control

To reduce the risk of file corruption and deletion from the developer’s local machine, GitHub was used. GitHub is a repository hosting service, which also offers revision control. This allows regular backups to be made and what aspects of the project has been done.

Ethical Approval

User participation was required to carry evaluation therefor ethical approval from the School of Computing Ethics Committee was required. The ethics form and approval letter can be found in Appendix 2 and Appendix 3.

Meeting with Supervisor

Weekly meetings were arranged with the project supervisor. During meetings, the tasks that had been carried out the previous week was discussed. The supervisor provided suggestions and improvements to the project to ensure the requirements could be fulfilled. Insight was also provided on difficulties encountered by the developer and the suggested next steps for the following week.

Requirement Elicitation

The first step in the development lifecycle was requirements. The main source for gathering requirements was from meetings with the supervisor.

Functional Requirements

A set of functional requirements were established detailing each of the functions the simulator should facilitate.

Main Menu

The simulator shall have a main menu.

Main Menu – options

The main menu shall contain three options: Encrypt, Decrypt and Exit.

Encrypt Option

The simulator shall allow for encryption process to be simulated after selecting Encrypt from the Main Menu.

Decrypt Option

The simulator shall allow for decryption process to be simulated after selecting Decrypt from the Main Menu.

Exit Option

This option shall allow the simulator to close once after selecting Exit from the Main Menu.

Encryption

The simulator shall scramble plain text into cipher text.

One Rotor – Encryption

The simulator shall demonstrate the operation of encryption in one rotor.

Three Rotors – Encryption

The simulator may demonstrate the operation of encryption in three rotors.

Decryption

The simulator shall unscramble cipher text into plain text.

One Rotor – Decryption

The simulator shall demonstrate the operation of decryption in one rotor.

Three Rotors – Decryption

The simulator may demonstrate the operation of decryption in three rotors.

Visual Representation

The simulator shall visually demonstrate the principle of poly-alphabetic substitution in operation in the scrambling unit of an enigma machine.

Animation

The simulation shall be demonstrated using animation.

Attack Method

The simulator may include an attack method, which could become a game, where the user would guess the encrypted plain text.

Non-Functional Requirements

A set of non-functional requirements were established detailing requirements of the implementation.

Graphical User Interface

The interface shall be presented in a graphical format.

Operating System – Windows

The simulator shall be compatible on Windows Operating Systems.

Operating System – Mac/Linux

The simulator may be compatible on Mac/Linux Operation Systems.

Development

The simulator should be developed using C++ and Visual Studio IDE.

Design

Design Prototypes

Several user interface prototypes were developed during the initial stages of development. The main focus was to ensure a high level of usability could be achieved in terms of ease of use for the users. As part of the iterative approach, throughout development prototypes were revised.

<PICTURE OF IT HERE!>

Figure 3 – Hand drawn prototype.

After the first iteration of hand drawn prototyping, high fidelity prototypes were designed.

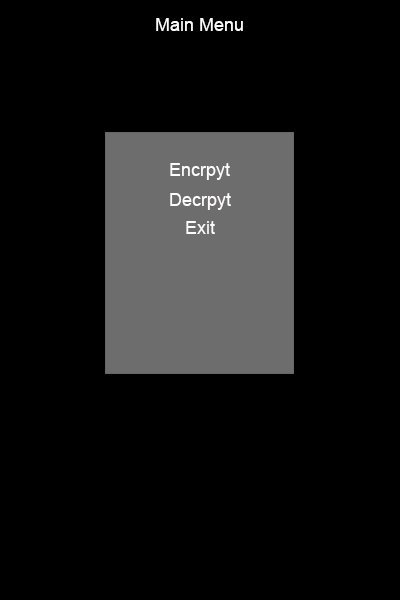


Figure 3.1 – Main Menu prototype design (Portrait orientation)

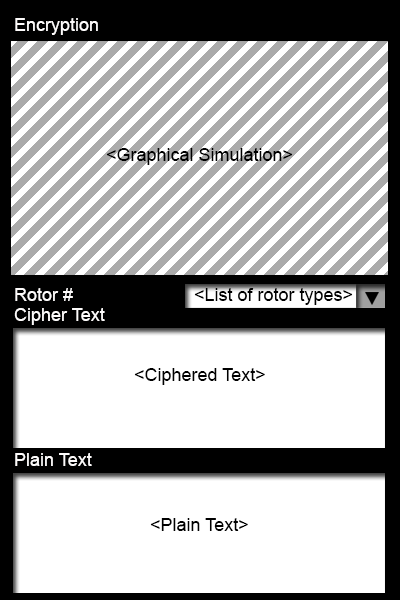


Figure 3.2 – Encryption screen prototype design (Portrait orientation)

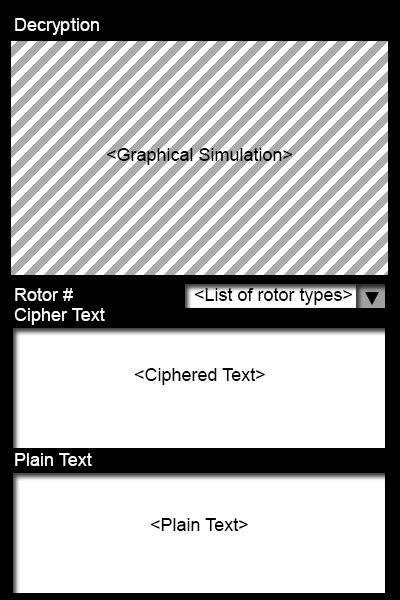


Figure 3.3 – Decryption screen prototype design (Portrait orientation)

Landscape versions of the prototypes were also designed. These can be found in Appendix[INSERT INTERFACE DSEIGN NUMBER].

Final Design

During development the screen felt cluttered with portrait orientation while the simulation was running. At that stage a design decision was made to set the orientation on the simulation screens (Figure 3.5 & Figure 3.6) to landscape. The main menu is not affected by this.

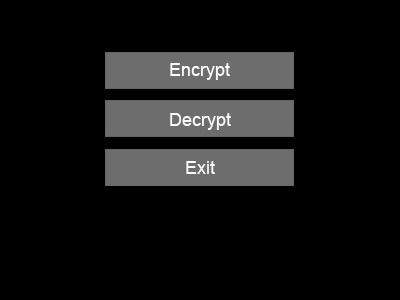


Figure 3.4 – Main Menu (Final prototype)

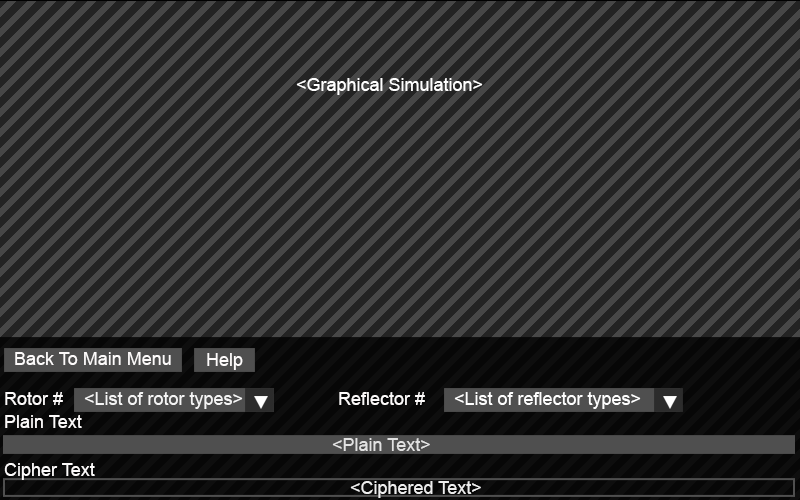


Figure 3.5 – Encryption Screen (Final Prototype)

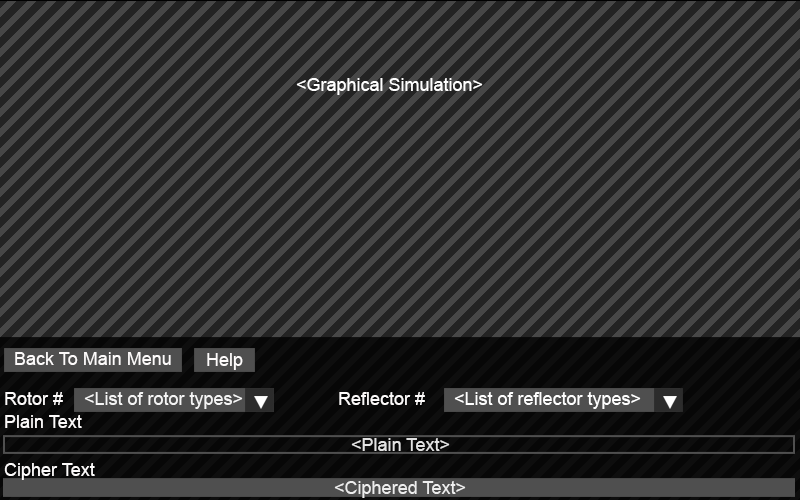


Figure 3.6 – Decryption Screen (Final Prototype)

During the final iteration of design a number of features to the user interface was added. One feature added was the selection of reflector types as well as rotor types. A help button was also added to give additional guidance to the user. Initially the idea was to have a menu bar, but it was decided rather to be a button which would take the user back to the main menu. In order to allow the users to understand the process of encrypting and decrypting, the processes are done live. For example during the encrypting process, when a used enters a letter into the plain text field, the encrypted letter would be process and outputted instantaneously to the cipher text field. Further details can be found in Appendix [DESIGN INTERFACE].

Rotor Design

While the purpose of this project is to show a detailed view of the encryption and decryption process, each part of the rotor was modelled using Blender [6]. Each component was modelled as closely as possible to the real life counterpart.



Figure 4 – Exploded View of Rotor (Picture by Jerry Proc [7])

Once modelled using Blender, the model was exported as a Wavefront file format (.obj). Colours or textures were not modelled in Blender but instead using OpenGL.

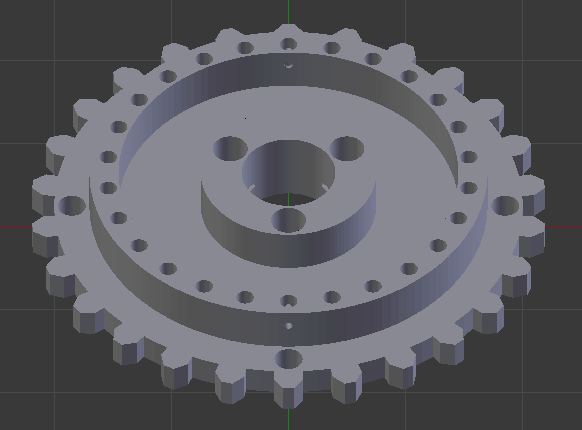


Figure 4.1 – Ratchet Wheel Model

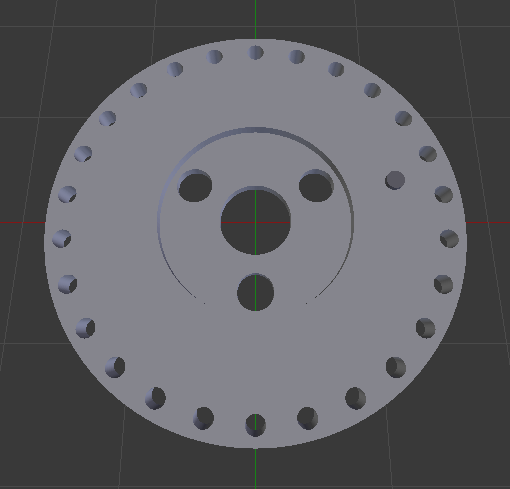


Figure 4.2 – Contact Wheel Model

The scale of these models were not an important factor because they could be modified during coding therefore the developer decided model them without too much importance given to this factor. However it was important that all the components were proportional to each other while modelling to save time during coding to align them correctly. With this factor taken into account, the diameter of the holes in each component was consistent throughout.

Implementation and Testing

Considerations

A variety of technologies were researched before coding commenced. The key areas researched were a programming language, APIs (application programming interfaces) used to render the graphics and a graphical user interface library.

APIs

A choice of programming language and graphical user interface library hinged on the choice of this. The two choice research were Direct3D and modern OpenGL.

Direct3D

Direct3D is a graphics API which can be used to create 2D and 3D graphics but it is proprietary software. Also it is not cross-platform compatible and is only available on Microsoft Windows operation systems.

OpenGL

Modern OpenGL can run across multiple platforms and is an open standard rather than proprietary software.

Graphical User Interface

Various graphical user interface libraries were researched. This would be the component allowing the user to interact with the simulation.

CEGUI [8]

Crazy Eddie’s GUI system. A free library providing windowing and widgets for graphics API and engines where such functionality is not natively available or is severely lacking. The library is written in C++, object orientated and is primarily targeted at game developers. It is licensed under the MIT license.

libRocket [9]

LibRocket is a C++ interface middleware package designed for game applications. At its core it is based on the popular HTML and CSS specifications. It implements the Model-View-Controller (MVC) design pattern. It is licensed under the MIT license.

LibUFO [10]

LibUFO is a C++ core library for forms respectively graphical user interfaces. It is mainly used as an OpenGL GUI toolkit. It is based upon an abstract layer which must be implemented by a native backend. It is licensed under the GNU LGPL version 2.1 license.

AntTweakBar [11]

AntTweakBar is a small and easy to use C/C++ library that allows programmers to quickly add a light and intuitive graphical user interface into graphical applications based on graphical APIs such as OpenGL and DirectX to interactively tweak parameters on-screen. It is design to be fast, clean and intuitive while minimizing the programmers work. It is released under the zlib/libpng license.

ImGUI [12]

Immediate Mode Graphical User Interface (ImGUI) is a bloat-free graphical user interface library for C++. It outputs vertex buffers that you can render in your 3D-pipline enabled application. It favours simplicity and productivity rather than certain features normally found in more high-level libraries. It is licensed under the MIT license.

Qt [13]

Qt is a cross-platform application and UI framework for developers using C++, a CSS and Javascript like language. It is licensed under a commercial and open source license.

FLTK [14]

Pronounced “full tick” FLTK is a cross-platform C++ GUI toolkit. It provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL and its built-in GLUT emulation. It is licensed under the GNU Library General Public License.

Programming Language

An important factor which impacted the selection of language was cross-platform compatibility. C#, C++ and Java were taken into consideration given the developers previous experience with these.

C#

C# is an object oriented programming language. It allows development applications that run on the .NET framework. The syntax is simple and easy to learn so it was an easy choice for consideration. Also the developer had vast experience using this language.

C++

An extension of C language, C++ is an object oriented programming language. It encapsulates both high and low level language features. It is seen by many as the best language for creating large-scale applications.

Java

Designed to have the look and feel of C++, Java is simpler and easier to learn. It is object orientated and robust. Also it is platform independent.

Technologies Used

Decisions about which technology to use were finalised early on during development since they were based on the requirements.

Blender [6]

The individual components of the rotor had to be modelled in 3D. Blender was the obvious choice since it was free to use, compared to most popular modelling tools which required paid licensing. A major disadvantage of this however was the steep learning curve and the fact that the developer had not worked with this software before. With this factor taken into account the components were not modelled to complete precision but the important details which would be required to enable the user to view the rotor were successfully modelled. A feature of Blender is that it can export to a variety of file formats. The file format used to export the objects was a Wavefront (.obj) format. This was because the vertex positions of the object could easily be parsed into any program.

OpenGL

The main reason behind choosing OpenGL was because the Graphics module was well underway therefore it made sense to choose OpenGL instead of having to learn a brand new language. In addition OpenGL is an open standard not proprietary software and OpenGL is cross-platform which means it is not restricted to only being run on Microsoft Windows operating system. In addition this would meet one of the requirements.

GLSL [15]

Once the objects were imported into our OpenGL environment, lighting was required to give a more realistic effect as well as being aesthetically pleasing. GLSL (OpenGL Shading Language) was used to achieve this effect. It was also taught during the Graphics module alongside OpenGL. It is used to so code can be run on the GPU. It is made up of four main shaders, vertex shaders, fragment shaders, geometry shaders and tessellation shaders. GLSL is a similar to C programming language with C++ mixed in.

Vertex Shaders

The vertex shader processes each vertex separately so they run once per vertex passed to the graphics processor. This is required for all OpenGL programs and must have a shader to pass to, usually the fragment shader. The purpose of this shader is to define vertex positions and other vertex attributes such as position, colour and texture coordinates.

Fragment Shaders

This is actually the last part of the shading pipeline. It processes the individual fragments generated by OpenGL’s rasterizer and their main purpose is to compute the colour and depth of pixel fragments. It must also have a shader bound to it, normally the vertex shader.

Tessellation Shaders

This shader is optional. It receives its inputs from the vertex shader. It processes patches, a type of geometric primitive specifically for tessellation shaders. Its purpose is to tessellate, spilt mesh into smaller geometric primitives such as triangles, the mesh patches.

Geometry Shaders

This stage is also optional. Its inputs are either from the vertex or tessellation shader. It processes each geometric primitive and defines geometric primitives. It can modify the type and number of geometric primitives by emitting altered or new primitives, or discarding them.

For this project only the vertex and fragments shaders were required because greater detail was not required will rendering the graphics therefor these could be sufficient enough. Also geometry and tessellation shaders were not taught during the Graphics module.

C++

Given that a user interface was a necessity C# was an obvious choice for consideration due to the fact that Visual Studio allows drag and drop of user interface items using this programming language. One major advantage of this would be that it would save time on implementing a graphical user interface library on top as it would not be required. However modern OpenGL implementation did not seem feasible given that experience using OpenGL was with C++.

ImGUI

A graphics library which was compatible with modern OpenGL and GLFW [16] windowing system was essential. ImGUI, Immediate Mode Graphical User Interface, was the chosen library as it was compatible with both. It is maintained on GitHub, meaning updates for this library and made regularly. Integrating a graphical user interface library with OpenGL was found to be a more difficult task than initially thought, however ImGUI was the only library which the developer was able to integrate successfully with OpenGL and GLFW. While a lack of support exists on the internet, it provides code which is easy to understand which made it the correct decision to use it as a graphical user interface library.

Classes

Testing

White Box Testing

Black Box Testing

User Testing

Evaluation

Usability

Sus stuff

Nielson’s Heuristics

Other Criteria

Summary and Conclusions

Acknowledgment

References

[1] Royal Naval Museum. ‘The Enigma Machine’ [Online]

Available at:

http://www.royalnavalmuseum.org/info\_sheets\_enigma.htm

(Last Accessed at: 23/03/2015)

[2] Rijmenants D. ‘The German Enigma Cipher Machine’ [Online]

Available at:

http://users.telenet.be/d.rijmenants/en/enigma.htm

(Last Accessed at: 23/03/2015)

[3] History ‘Code Breaking’ [Online]

Available at:

http://www.history.co.uk/study-topics/history-of-ww2/code-breaking

(Last Accessed at: 23/03/2015)

[4] Enigma History [Online]

Available at:

http://www.cryptomuseum.com/crypto/enigma/hist.htm

(Last Accessed at: 23/03/2015)

[5] EnigmaCo.de [Online]

Available at:

http://www.Enigmaco.de

(Last Accessed at: 23/03/2015)

[6] Blender [Online]

Available at:

http://www.Enigmaco.de

(Last Accessed at: 23/03/2015)

[7] Jerry Proc [Online]

Available at:

http://www.proc.ca/crypto/enigma.html

(Last Accessed at: 23/03/2015)

[8] CEGUI [Online]

Available at:

http://cegui.org.uk

(Last Accessed at: 23/03/2015)

[9] LibRocket [Online]

Available at:

http://www.librocket.com

(Last Accessed at: 23/03/2015)

[10] LibUFO [Online]

Available at:

http://libufo.sourceforge.net

(Last Accessed at: 23/03/2015)

[11] AntTweakBar [Online]

Available at:

http://anttweakbar.sourceforge.net

(Last Accessed at: 23/03/2015)

[12] ImGUI [Online]

Available at:

http://github.com/ocornut/imgui

(Last Accessed at: 23/03/2015)

[13] Qt [Online]

Available at:

http://qt-project.org

(Last Accessed at: 23/03/2015)

[14] FLTK [Online]

Available at:

http://fltk.org

(Last Accessed at: 23/03/2015)

[15] D. Shreiner, G. Sellers, J. Kessenich and B. Licea-Kane (2013), *‘OpenGL Programming Guide 8th Edition’* Version 4.3, pp. 34-36.

[16] GLFW [Online]

Available at:

http://www.glfw.org

(Last Accessed at: 23/03/2015)