3D Printing of Computer Generated Celtic Knotwork Patterns

CSC2005Z: Research Project

Mitch Myburgh
UCT
Address
Cape Town, South Africa
mybmit001@myuct.ac.za

ABSTRACT
Celtic Knotwork is an ancient art form created by monks for illustrating illuminated manuscripts, it is made up of overlapping bands, to produce pleasing patterns. This report looks at the algorithmic construction of Celtic Knotwork and the creation of a user friendly interface for creating basic Celtic Knotwork, the user interface must find a balance between usability and complex customisability of the knot. The report also investigates the creation of customised 3D models of the knotwork suitable for 3D printing.

Categories and Subject Descriptors
H.5.2 [**User Interfaces**]: Graphical user interfaces (GUI); I.3.3 [**Picture/Image Generation**]: Display algorithms

General Terms
Design

Keywords
Celtic Knotwork, 3D Modelling

1. INTRODUCTION
Celtic knotwork's characteristic overlapping pattern of knot bands is a style of artwork that can be produced algorithmically. The Knotwork commonly adorns Manuscripts and can be found carved into stonework. This project focuses on the basic Celtic knotwork designs, made using a grid of squares, however by modifying this algorithm one can produce a variety of knots in almost any shape and design.
This project aims to produce 3D models from algorithmically produced Celtic Knots, that are suitable for printing with a 3D printer. A second goal is to build an intuitive user interface for constructing Celtic Knotwork, and a streamlined export of a file ready for 3D Printing. The software is built in Python and the Kivy display library for speed or programming and ease of use.

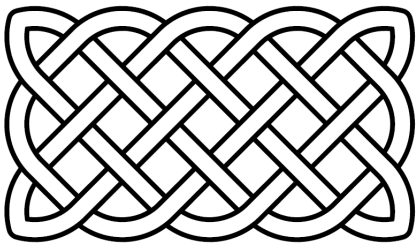

Figure 1: A basic knotwork pattern

1.1 Problem Statement
The current implementations of Celtic Knotwork are designed as an extension for quickly demonstrating the algorithm rather than focusing of the user experience (Figure 2 shows the interface made by Andrew Glassner [5] for producing Celtic Knotwork). The software produced for this report will be evaluated using Heuristic evaluation to determine its usability.

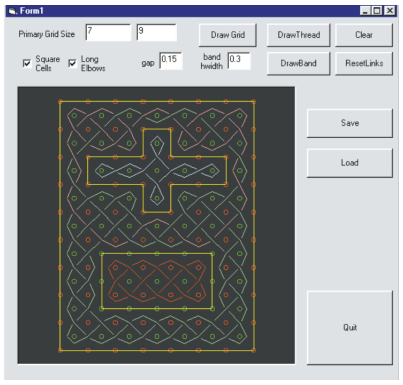

Figure 2: The interface created by Glassner

3D output has been achieved before by Glassner [7] (Figure 3) and Kaplan and Cohen [8] (Figure 4). The software created in this report is also able to output a 3D model of Knotwork as an STL file suitable for 3D printing.

1.2 Research Objectives

Figure 3: The 3D knotwork produced by Glassner

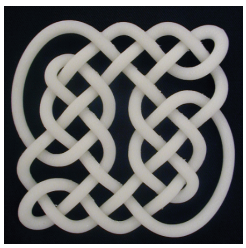

Figure 4: The 3D knotwork produced by Kaplan and Cohen, which has been 3D printed

Produce a 3D printable Celtic Knotwork Model

- The resulting knotwork should be a simple 3D model of the knotwork which is suitable for use in other applications or printing directly.
- This will be tested by running the resulting 3D knotwork through a program to test for manifold errors (gaps in the model and surfaces that don't connect) and by printing the 3D knot

Produce an improved, user-friendly, interactive interface for designing knotwork patterns

- The interface in most Celtic knotwork applications is utilitarian as the applications are mainly focused on the knotwork algorithm. A focus on the interface should make it easier for users to learn how to make Celtic Knotwork, making the user interface interactive will improve usability.
- This can be tested through user testing of the interface. A number of existing usability questionnaires can be administered as a part of this test.

2. LITERATURE REVIEW

Celtic knotwork is a style of art characterised by interlacing patterns, often seen in illuminated manuscripts and in stone carvings. The technique was initially thought lost until it was rediscovered in 1951 by George Bain [1], when he published a book detailing the creation of the knotwork. Bain's work was later improved upon by his son Iain Bain [2], by simplifying his fathers method and presenting the 3 Grid system that would be the basis for the algorithm used by Andrew Glassner [5, 6, 7], when he produced the algorithm that the software in this report will be following. Glassner bases his knotwork on grids [5] and more complex tessellating shapes [6]. Kaplan and Cohen [8] expand on these ideas with an algorithm for creating more complex knotwork, by weaving it over any shape or 3D mesh, and also by introducing images into the knots. Drewes and Klempien-Hinrichs [3] describe a more detailed algorithm using a collage grammar, which uses tree evaluation and predefined images to produce the knots. Douglas Dunham [4] uses Glassner's algorithm to produce Celtic Knotwork that is combined with hyperbolic geometry, complementing Celtic knotwork on the plane and sphere. This illustrates the flexibility of the algorithm. 3D knotwork is formulated by Glassner [7] as weaving the knotwork on a deconstructed shape and then reconconstructing it to form a 3D object. Kaplan and Cohen [8] also touch on 3D knotwork by using their algorithm to produce 3D printed knots, but their approach is able to weave the pattern over an arbitrary mesh, allowing more variation in the designs.

- More papers
- Expand on explanation of current papers

3. IMPLEMENTATION

The software to algorithmically produce the Celtic Knotwork was built based on the work of Andrew Glassner [5][6][7]. This software is then able to take the Knotwork pattern produced by the user and create a 3D model of it suitable for input to meshing software that provides a set union of shapes and produces a printable STL file. This system will be provided by Professor Gain. This software will perform error checks to ensure that the file can be 3D printed properly, without manifold errors, but the output will be tested in other programs first. The user interface of the program will be built to be as user-friendly and intuitive as possible, using appropriate HCI heuristics, and a series of user tests will be performed to test this.

The software was built using Python and the Kivy interface library.

3.1 Major Software Artifacts

The Major software artifacts that will be produced will be two software divided into two parts:

1. The first to algorithmically produce a Celtic Knotwork pattern (Knotwork Program): **Key Features:** The ability to produce the knotwork pattern based on user specifications for the size, breaklines.
2. The second part of the software will be to convert this knotwork pattern into a 3D model which can be printed on a 3D printer (3D Generation Program): **Key Features:** The ability to convert the 2D pattern into a 3D model suitable for printing; **Major design challenges:** Producing over-under weaving without manifold errors.

3.2 Algorithm

The algorithm for this implementation of Celtic Knotwork is based on the algorithm put forth by Glassner [5] which in turn is based on the original algorithm by George Bain

[1] and the improvements made by his son Iain Bain [2], the algorithm is based on a grid of squares, with each cell containing a single piece of the knot.

3.2.1 Drawing the Grid

A x by y primary grid is specified with $\{x, y \geq 1 | x, y \in \mathbb{Z}\}$. From this primary grid a secondary grid is defined by splitting each cell into 4 squares resulting in a $2x$ by $2y$ grid.

3.2.2 Breaklines

Breaklines are the main method of producing knotwork patterns, these are lines which the knotwork cannot cross. Each cell in the secondary grid can have a maximum of 2 breaklines and the breaklines drawn along the primary grid lines cannot cross those drawn on the secondary grid lines. It is necessary to bound the outside of the grid in breaklines in order to properly define the knot. For the purpose of the software breaklines are numbered from 1 to 4 starting at the top and moving clockwise around the cell.

Breaklines can appear in the following positions:

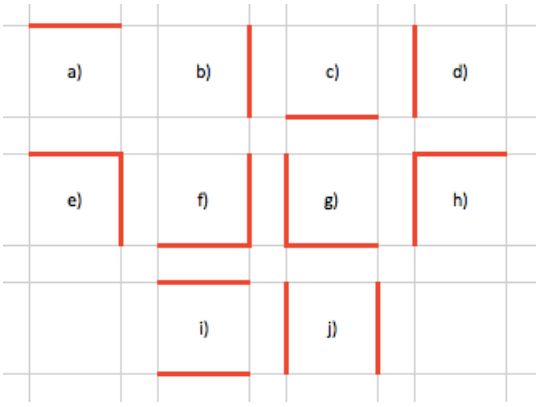

Figure 5: The possible positions of the breadlines in each secondary cell. a) 1, b) 2,c) 3,d) 4,e) 1 and 2,f) 2 and 3, g) 3 and 4, h) 4 and 1, i) 1 and 3, j) 2 and 4

3.2.3 Drawing the Skeleton

The skeleton is the rough representation of the knotwork and is made by drawing straight lines in each cell joining algorithmically determined points. For ease of explanation the following points around each secondary cell are used throughout (Figure 6)

The knot is made by an alternating starting in the top left corner. For odd rows start with a line sloping to the right in the first cell, for even rows start with a line sloping to the left; then for each cell swap the direction of the line so that odd cells have a line sloping the same direction as the first cell. Because the secondary grid always has an even number of cells in each row you will always end the row (and column) with a line sloping the opposite direction to the line in the first cell. In the interior of the knot the lines connect opposite corners (1 and 2 or 3 and 7), breaklines affect the positioning of the knot lines, as displayed in figure 7

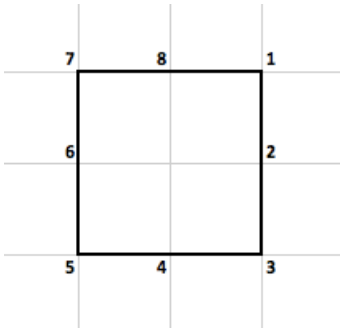

Figure 6: The grid points used to describe connecting lines in the skeleton

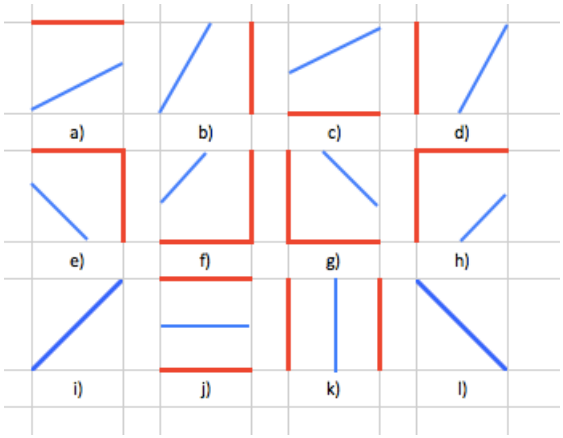

Figure 7: Diagram showing the affect of breaklines on the knot lines. i) and l) are the lines found in the interior of the knot; e), f), g) and h) are the breaklines in the corners; a), b), c) and d) show the knot lines in the borders and should be flipped in an alternating pattern (i.e. for a) the line can either join 5 and 2 or 3 and 6); j) and k) are special cases.

Figure 8 shows a complete knotwork skeleton, as well as the breaklines and grid lines.

3.2.4 Drawing the Knot

The final knot is based on the skeleton, but adds some features such as corners and curves. There are two types of corners available in the software: a right angle corner, which joins the midpoints of the sides with the middle of the cell (Figure 9); and a curved corner which is a quadratic Bézier curve (Figure 10).

The curves on the edges can be seen in figure 11 and are quadratic bezier curves, which appear when the cell is bordered by a breakline.

3.3 Interface

The knotwork interface is made using Python and the Kivy interface library, Kivy is designed for use with Android so its interface looks non-native while still being visually pleasing, and with the implementation of native keyboard shortcuts

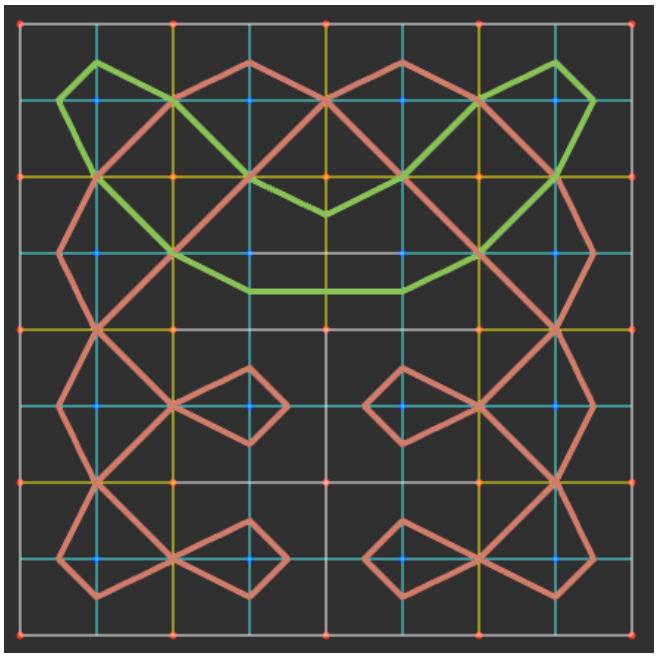

Figure 8: Image of the Celtic knotwork skeleton with breaklines in white, taken from the software.

Figure 9: A 2x1 knot with right angled corners

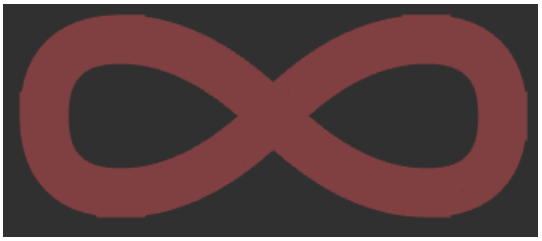

Figure 10: A 2x1 knot with curved corners

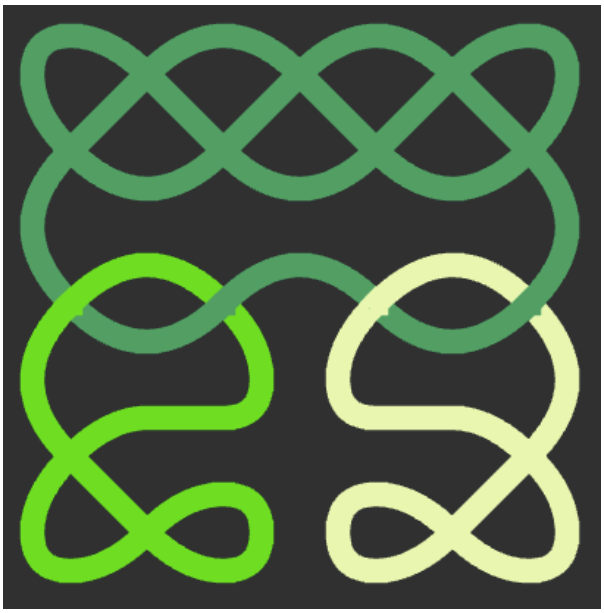

Figure 11: A 4x4 knot showing the curves and corners

this issue is minimised.

3.3.1 Knotwork Interface

The knotwork interface is a tabbed interface allowing the user to work on multiple knots at the same time. Each tab contains a main view, which prominently displays the knotwork with options and buttons down the left and right sides.

Down the right side are the main settings for generating the knot including setting its the size of the knot, the size of the grid cells, the width and the corner type. The option to output the 3D model is included in the bottom right.

Along the left column are the options to customise the knot including adding breadlines and showing or hiding various elements of the knot. The Save and load buttons are included in the top left. Keyboard shortcuts are provided for these options (the shotcut is included in brackets) as well as system shortcuts for save, load, undo, and redo.

Adding breaklines opens the interface in figure 13 allowing the user to produce breadlines by selecting dots of the same colour in a horizontal or vertical line, the system checks that the breaklines of the two subgrids (i.e. lines joining red and blue dots) do not cross.

The knotwork is drawn using the algorithm described in (TODO link to algorithm section), and the colours are randomised each time an action is performed.

3.3.2 Save/Load Interface

The program includes a save function that writes the knotwork and breaklines to the disk in JSON format. The user is given a save dialogue (provided by the Kivy library) as in figure 14. The user is also able to load a previously saved knot from the disk. The knotwork can be saved with any file extension.

3.3.3 Output 3D Interface

The software provides an interface for outputting the knotwork as an STL file for 3D printing. The interface for producing the 3D knotwork (Figure 15) shows two cross sections of the knot to illustrate how they will interact and the affect the various options have on their shape. The user can set the width, height and radius of the corners of the rounded

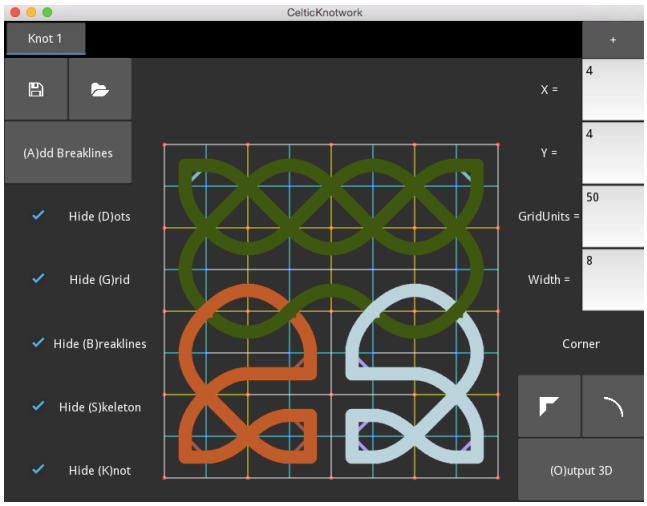

Figure 12: The main program interface

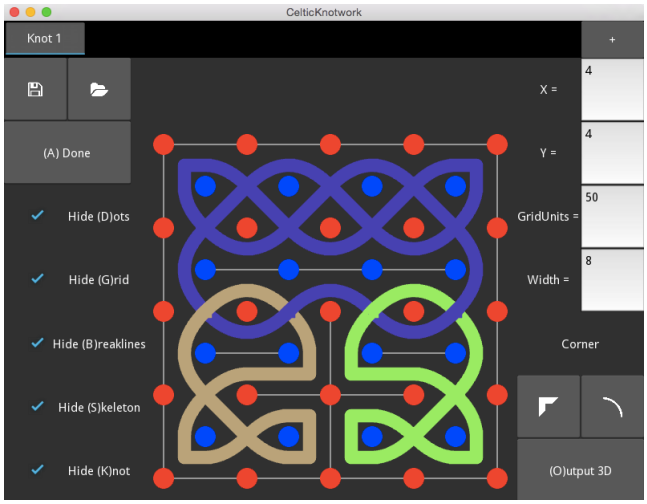

Figure 13: The interface for adding breaklines

rectangle allowing them to specify the cross section as either a rectangle, rounded rectangle or circle. They can indicate how much overlap the knot should have, which sets the height of the one band above the other at the overlap point. They can also specify the smoothness of the corners in the cross sections (the number of triangles the corner is divided into) and the smoothness of the 3D model (the number of cylinders each cell is divided into). The final 3D output is shown in figure 16

4. EXPERIMENT AND RESULTS

Four types of evaluation were carried out on the software. The first evaluation looks at knots produced using the software and knots from other sources, a visual examination was conducted to evaluate the programs strength at producing accurate celtic knotwork. The second evaluation is a Time evaluation to evaluate the speed and responsiveness of the program when generating the knots and the 3D output. The third evaluation is a heuristic evaluation using Neilsen Heuristics and (Gerhardt Powals?) heuristic evaluation techniques, which will evaluate the suitability of the interface. The fourth evaluation is completed by running the 3D output through netfabb and checking for errors

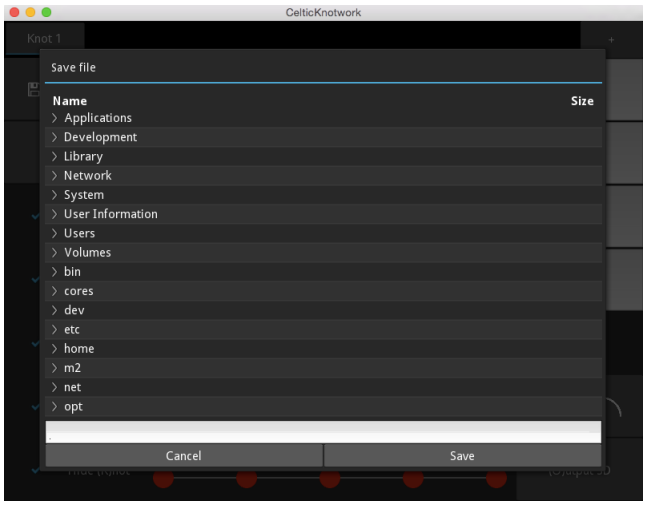

Figure 14: The interface for saving the knotwork in JSON format

4.1 Comparison with Other Knotwork

Comparisons of a knot with the equivalent produced using the software are made in Figures 19, 22 and 25, in most cases it is possible to produce a structurally identical knot, but the lack of customised curves prevents it from being visually identical (TODO more complex knots - Should these go in the appendix?)

4.2 Time Evaluation

The knotwork program was tested in order to gauge the time the user would be required to wait when producing large and complex knots. The size of the knots was tested in particular, both when generating the knot and when generating the 3D Model of the knotwork.
The time taken to produce the Knot data structure was recorded over 100 trials and resulted in:

1. **4x4 Knot:** $0.00056537 \pm 2.15991 \times 10^{-05}$
2. **100x100 Knot:** $0,44124815 \pm 0,00317555$

The time is acceptable as even a 100x100 knot is produced in less than 1 second

The time taken to generate the 3D model was recorded over 100 trials and resulted in

1. **4x4 Knot:** $1,14852401 \pm 0,009837714$
2. **10x10 Knot:** $7,64070195 \pm 0,024236157$

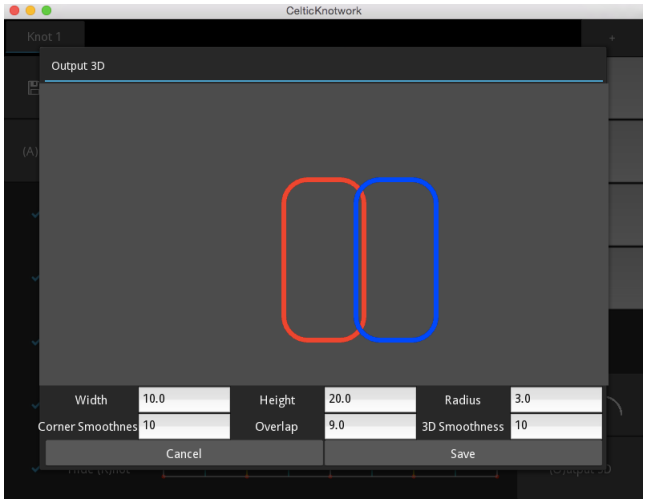

Figure 15: The interface for outputting a 3D model of the knotwork

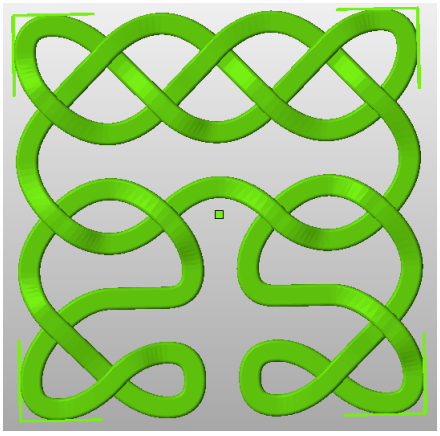

Figure 16: The 3D model of the knotwork

(TODO: bigger knots?) The time taken to generate the 3D knot is substantial due to the large number of calculations, and for larger knots this can cause the program to hang until the computation is complete, which can lead the user to pressing the "Output 3D" button multiple times (which leads to even longer calculations ad the calculation is repeated on each click).

4.3 Heuristic Evaluation

The Heuristic evaluation was completed by 3 people of which 2 had previous experience with Celtic Knots. They were asked to complete a series of tasks relating to the interface and rating their experience. Comments were also recorded. Those with previous experience found the system easier to use.

The basic evaluation will be done using Nielsen's heuristics:

1. **Visibility of system status:** When asked during the tasks whether the system gave appropriate feedback

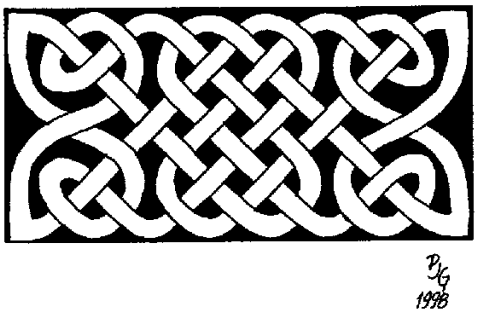

Figure 17: a source (possibly for lit review): http://www.mi.sanu.ac.rs/vismath/fisher/

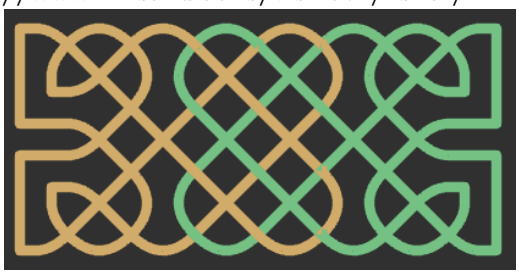

Figure 18: b

Figure 19: Knots

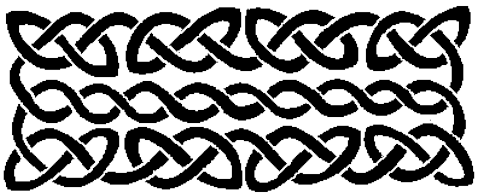

Figure 20: a source http://www.ceolas.org/clipart.html

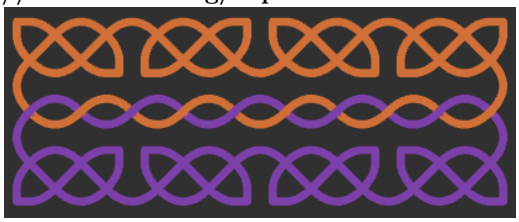

Figure 21: b

Figure 22: Knot

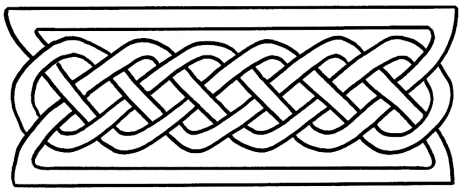

Figure 23: a source http://imgarcade.org/1/celtic-knot-border/

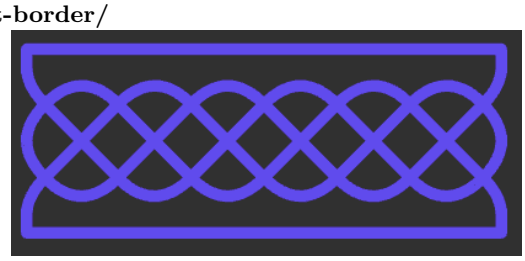

Figure 24: b

Figure 25: Knot

within a reasonable amount of time 95.24% of people responded "Yes", the only case where the interface did not give appropriate feedback was during the saving of the 3D knot where the time taken to save the knot (approx 1 seccond) left the users thinking they had not clicked the button. When asked in general their agreement with the statement "The system kept me informed about status within a reasonable amount of time" the response had an average of 1.67 (where 1 is strongly agree and 5 is strongly disagree). This shows that the system generally responds quickly to events, a time evaluation is included in the above section which shows the time taken to produce the knotwork.

2. **Match between system and the real world:** When asked how much they agreed with the statement "The system was easy to understand and the terms used were clear" the users gave it an average score of 1.67 (with 1 being strongly agree and 5 being strongly disagree). The system uses plain English terms and pictographic representations as much as possible, but specific options require a small amount of knowledge of Celtic Knotwork (for example Breaklines).

3. **User control and freedom:** The system supports undo (ctrl+z) and redo (shift+ctrl+z), dialogues for save and load have cancel buttons and most actions are reversible through the use of undo. Changing the size of a knot erases breaklines and is not reversible but users are presented with a dialogue warning them of this fact. When asked how much they agree with the following statement "I was able to escape from any scenario or action with undo/redo cancel etc" users gave an average rating of 1.67 (where 1 is strongly agree and 5 is strongly disagree)

4. **Consistency and standards:** When asked how much they agree with the statement "The system language and design was consistent with my expectations" user responded with an average rating of 1.33 (where 1 is strongly agree and 5 is strongly disagree). Platform conventions for undo, redo, save and load shortcuts were maintained.

5. **Error prevention:** All inputs are carefully checked for validity and only one error was found by the users, which was saving the data to a restricted disk location.

6. **Recognition rather than recall:** The system does not require the user to remember anything between dialogues and all relevant information is displayed on the screen, when asked how much they agree with the statement "The system was easy to use and I did not have to remember anything between dialogues" the users gave it an average rating of 1.33 (where 1 is Strongly Agree and 5 is Strongly Disagree)

7. **Flexibility and efficiency of use:** The system provides a set of keyboard shortcuts, which are listed in the app (e.g. (A)dd Breaklines uses the keyboard shortcut A) as well as system standard undo, redo, save and load shortcuts. Users made use of keyboard shortcuts 66.67% of the time, and when asked how much they agree with the statements "Keyboard shortcuts were helpful" and "Keyboard shortcuts were easy to figure out" they gave average ratings of 2 and 1 respectively (where 1 is Strongly Agree and 5 is Strongly Disagree)

8. **Aesthetic and minimalist design:** When asked how much they agree with the statements "The design was visually pleasing" and "All information presented was relevant" users gave a average responses of 1.33 and 2.33 respectively(with 1 being strongly agree and 5 being strongly disagree). The interface uses the Kivy library and although it provides a non-native look and feel it is still visually pleasing.

9. **Help users recognise, diagnose, and recover from errors:** One error was encountered by the users, which involved saving the file to a location on the disk the user did not have access to (this error occurred because the standard Kivy save dialogue presents the user with the root of the hard drive as the main directory). Users were unable to recover from this error as it crashed the software. No other errors were discovered by the users.

10. **Help and documentation:** When asked whether they used the documentation for each task, users responded "Yes" 19.05% of the time. Those that had previous experience with Celtic Knots did not use the documentation, while those who had no previous experience used the documentation in 57.14% of the tasks.

A second Heuristic evaluation will be made using Gerhardt-Powals? cognitive engineering principles or Weinschenk and Barker classification. (TODO) - seems like it might just be a rewording of the above evaluation?

4.4 3D Evaluation

The 3D produced knotwork does not show any errors in netfabb, so it can be concluded that it is suitable for 3D printing.

- The output doesn't show any errors
- Try different designs
- show 3d models against knots

5. CONCLUSION

This report has show that it is possible to create a user friendly interface for producing Celtic Knotwork, as well as producing 3D models which do not show errors.

6. FUTURE RESEARCH

Future research could be conducted to improve the implentation of the knotwork algorithm. The knot could be expanded to work with other types of grids (e.g. Triangular grids) or deforming and wrapping the knotwork around other shapes, such as circles or within text as is seen in Illuminated manuscripts (Figure 26)

Figure 26: A Knotwork pattern taken from an illuminated manuscript (source: http://static1.1.sqspcdn.com/static/f/1422346/22201177/1363492527663/16834306-14692883-thumbnail.jpg?token=eKhHi1IsND2asAe6mhucrzlW90s%3D)

The software could be improved by providing more granular control to the user (although this will have to be balanced with usability concerns) such as styling the knot with colours and custom designs. The user could also be given options to set each corner style individually.

7. ACKNOWLEDGEMENTS

The author would like to thank those that participated in this study as well as Professor J Gain for his input.

8. REFERENCES

[1] G. Bain. *Celtic Art: The Methods of Construction.* William MacLellan and Co., Glasgow, 1951.

[2] I. Bain. *Celtic Knotwork.* Sterling Publishing, New York, 1986.

[3] F. Drewes and R. Klempien-Hinrichs. *Theory and Application of Diagrams*, volume 1889 of *Lecture Notes in Computer Science*, chapter Picking Knots from Trees, pages 89–104. Springer Berlin Heidelberg, 2000.

[4] D. Dunham. Hyperbolic celtic knot patterns. In R. Sarhangi, editor, *Bridges: Mathematical Connections in Art, Music, and Science*, pages 13–22, Southwestern College, Winfield, Kansas, 2000. Bridges Conference. Available online at http://archive.bridgesmathart.org/2000/bridges2000-13.html.

[5] A. Glassner. Andrew glassner's notebook: Celtic knotwork, part 1. *Computer Graphics and Applications, IEEE*, 19(5):78 – 84, Sept/Oct 1999.

[6] A. Glassner. Andrew glassner's notebook: Celtic knotwork, part 2. *Computer Graphics and Applications, IEEE*, 19(6):82–86, Nov/Dec 1999.

[7] A. Glassner. Andrew glassner's notebook: Celtic knotwork, part 3. *Computer Graphics and Applications, IEEE*, 20(1):70–75, Jan/Feb 2000.

[8] M. Kaplan and E. Cohen. Computer generated celtic design. In P. Christensen and D. Cohen-Or, editors, *Eurographics Symposium on Rendering 2003*, pages 9–19, 2003.