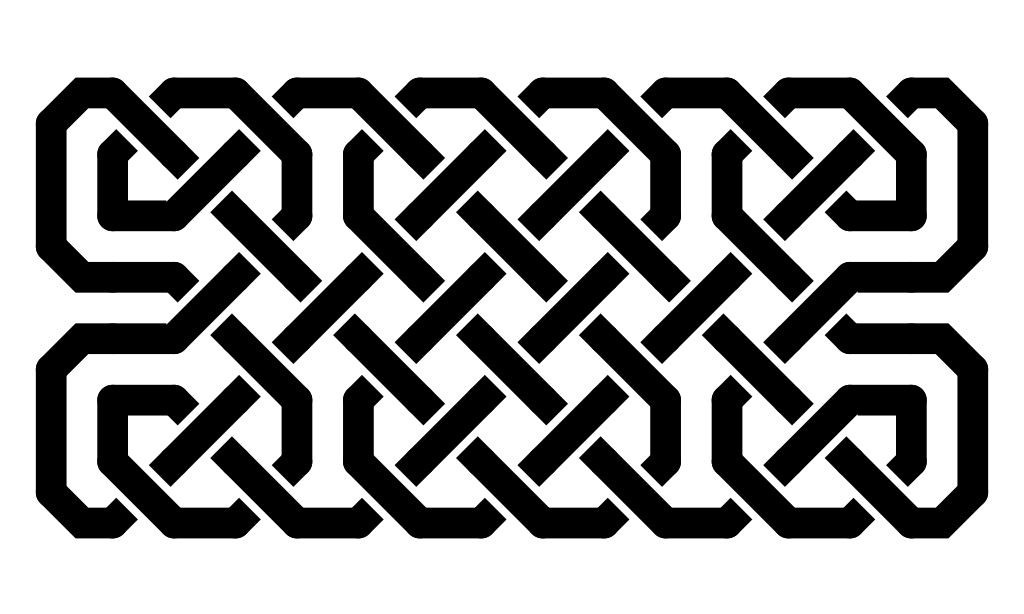
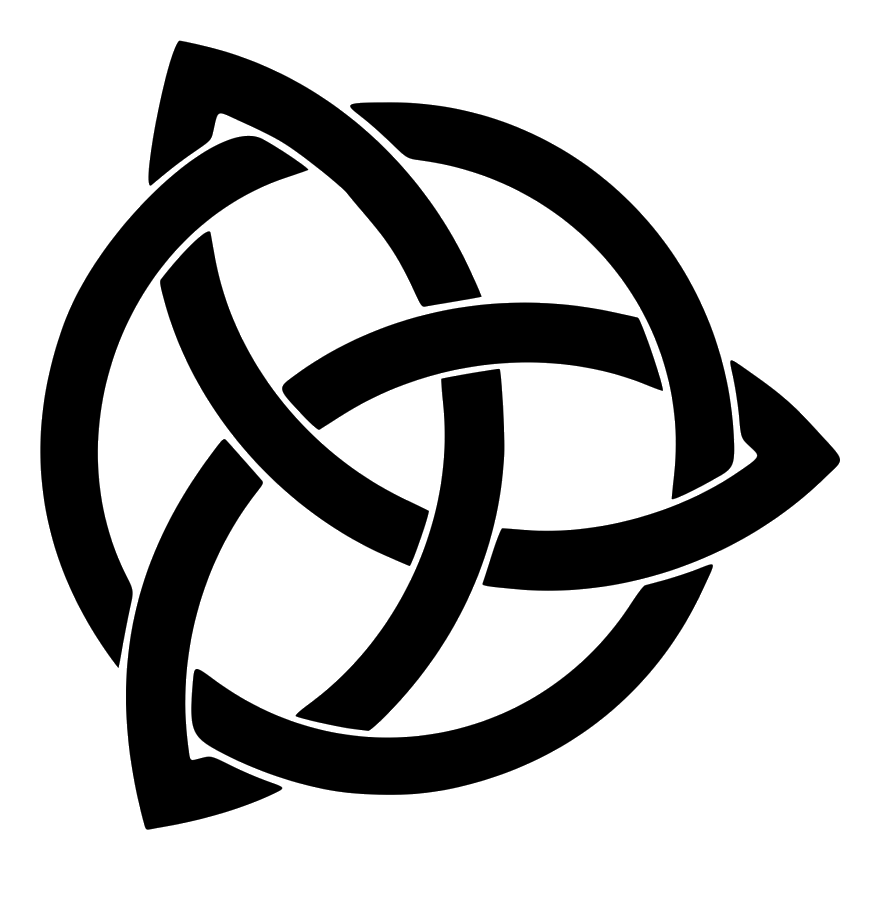
Project Proposal

Collin Avidano, Claudia Chu, Evan Strat, Jack Wolfard

# Problem

Celtic knots are decorative knots consisting of “strands” that intertwine with each other in particular ways. They are often found in religious texts, manuscripts, and monuments. These knots have interesting properties and configurations that one can analyze, such as the path of the strands, intersections, and turns. Given their traditionally regular shape and relatively strict rules, Celtic knots present an interesting opportunity to explore how a computer might be able to detect the strands and map the underlying structure of these knots. Typical celtic knots consist of one or more strands of some fixed width that, in a series of straight paths and curves, fold over each other (or itself) in an interlocking “knot” or “weaving” pattern. In the final image, there are sections of the strands that are covered, or “occluded,” by another strand that goes over it. A human can easily perceive that these strands are continuous. This leads us to consider how a computer may be able to detect these strands and identify parts of the image that constitute a continuous strand, and the underlying structure of the knot, such as the number of strands, and create a “spline” representation of the general path of the strands.



We propose creating an algorithm to detect the path of each string, intersections, and other relevant structures in a given celtic knots, provided that this celtic knot (1) is fixed width, (2) has standard “solid” style with spaces at intersections to indicate an overlap (example below), and (3) follows standard celtic knot style conventions.

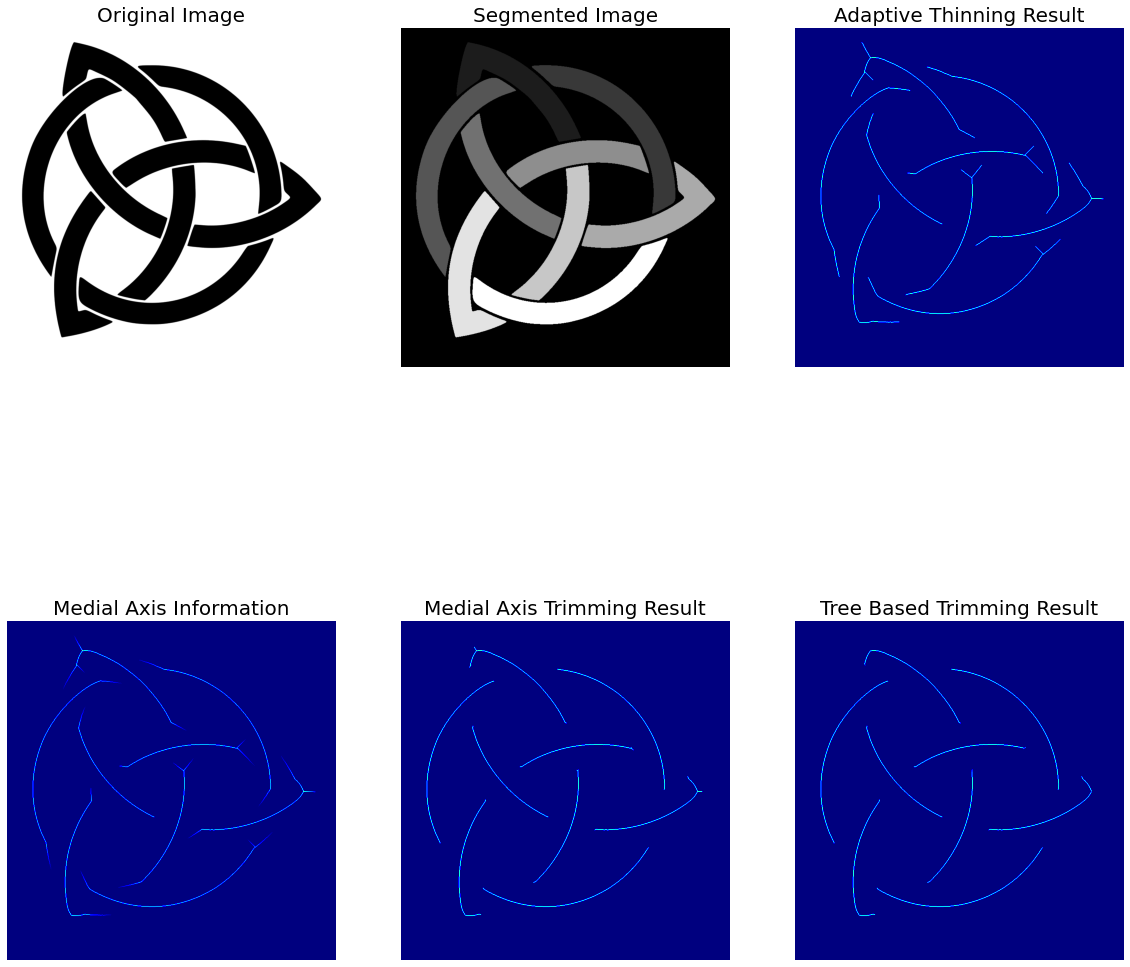
Furthering on this project, we can take this underlying structure and manipulate it. For example, imagine “tightening” or “loosening” the knots. Additionally, the method may be refined to be more generalizable to not only apply to variations in celtic knot styles but also other natural, irregular knot-like patterns found physically, such as highway systems and plant stems and vines.

# Approach

We will explore 2 different methods to detect the underlying structure of standard celtic knots.

Our first method is derived from observations on how humans deal with occlusion of objects via prior knowledge of the objects properties and contextual cues. In particular we will attempt to leverage that the direction of a section of string headed towards an intersection will not rapidly change on the other side of the intersection. Using this knowledge we propose the following approach:

1. Assign each continuous segment to a cluster membership of points
2. Erode the fixed width celtic knot until every segment is only 1-pixel-wide
3. Use the scipy interpolate library to fit a spline to the remaining 1-pixel-wide segments by their original membership
4. For each individual spline segment, extend one end by some small amount and run an intersection test with another spline. If the angle of intersection is below an arbitrary threshold, take its closest endpoint and yours and make one continuous spline
5. Repeat this process at the furthest endpoint until every original single segment has either been merged or evaluated once without merging.





*Incomplete visualization of the steps of method one. Considering image one to be the image on the far left and five on the far right:*

*Image one is the original knot input image.*

*Image two is a visualization of the knot segmented by connectivity.*

*Images three and four will be where the curve fitting and extrapolation process takes place to figure out how many disparate strings exist in the given image.*

*Image five is the idealized output now having only two segments which is how many strings there are in the original image*

Our second method is derived from observing how many celtic knot generators use standard tiles to create aesthetically pleasing celtic knots. Tiles consist of a set of squares, and each square has one or more lines depicting straight lines, curves, and intersections in varying orientations. Assuming that our data set consists of knots derived from the same set of tiles, we can:

1. Create a set of templates, each representing one of the tiles
2. Apply template matching on the knot
3. Align the matched tiles to generate the spline segment

Since we know the structural properties of the tiles, we can stitch together adjacent tiles to create continuous splines, and more easily label points of intersection.

If time permits, we have considered a third method, which consists of using Hough Transforms to detect the structure of knots. Standard celtic knots consist of parallel lines and curves. There may be a method using a combination of line detection and curve detection to identify these features, and infer the splines to connect the strands together.

# Experiments and results

In order to test our methods we will set up a set of knots where we will label the segmentation of the strings to be each perceptually continuous string as well as label each intersection or crossing to the best of our ability as people. Then we will have both method one and method two generate similar maps and compare the chamfer distance of the segmentation to our hand-labeled set. Then we will compare the SSD of our detected intersection crossings to our hand labeled crossings.

**Dataset**

To create our test dataset, we will utilize existing celtic knot generators found online. Currently, we have two possible data sources in mind. To manually create celtic knots, we will use the tile-based generator found online at [http://hypatiastudio.com/celticknots](http://hypatiastudio.com/celticknots/). If we need to generate celtic knots consistently, quickly, or programmatically, we have created a functioning fork of a GitHub repository ([rspencer01/celtic](https://github.com/rspencer01/celtic), released under the MIT License) containing a Python script for programmatically generating celtic knots.

**Code**

We will utilize some existing Python libraries, including scipy and skimage, to perform the fundamental steps (such as template matching or interpolation) of our overall algorithms. For method one, we plan on using scipy interpolation to fit splines to preprocessed segments of the celtic knots. For method two, we will use skimage template matching to identify the location of our tile templates in our image.

Both methods involve detecting parts of the celtic knot. Our main contribution in terms of code will be the pre and post-processing of our data as well as putting those pieces together to form a continuous representation of the celtic knot. With regards to method one, that is the code to connect the individual splines. With regards to method two, that is the code to find the structure of each tile and assemble a final representation of the structure after identifying the location of various templates in the image.

**Experiment**

We will test the performance (or viability) of each method to do the following:

1. Detect the width of the strands - We compare the truth width of the lines to what the method detects
2. Width-invariant spline detection - The method detects the same spline regardless of the width of the strands
3. Orientation and scaling-invariant performance
4. Counting number of strand components

To compare the methods’ performance to each other, we will consider:

1. Efficiency (run-time and space complexity)
2. Similarity to pre-generated information - For example, we can generate knots based on tiles. Since we already know the set of tiles used, we can compare the splines from that set of tiles, and compare with the spline calculated by our methods
3. Similarity to what a human would produce - a human will use a drawing tool to approximate the splines in a celtic knot. We compare that spline with the one calculated by our methods.

**Possible Results**

We anticipate that each method will have certain strengths and weaknesses in both viability and robustness. We predict the tile-based method will be the most brittle to variations in celtic knots due to its finite library of templates. However, this method will make it easier to connect the detected tiles together since each tile already comes with metadata we can use to connect the components.

Interpolation has much more flexibility in finding more irregular or complicated features in knots. Nevertheless, this is dependent on the ability to extrapolate the interpolated splines accurately in order to find points of intersection or continuation of a spline. This may be very sensitive to noise or sharp edges in the original image.

Timeline

* Proposal due next week (10/1)
* 11/1 - Mid-term
  + Week 1: Now - 10/01
    - Finish proposal, start initial experiments
    - Generating data
      * Find a standard set of tiles, OR use numpy, imsave for each window
    - Initial experiments with interpolation and tile based approach
  + Week 2: 10/02 - 10/08
    - Interpolation - Collin, Evan
    - Tile - Claudia, Jack
  + Week 3: 10/09 - 10/22
    - Semi-working code, initial experiments
  + Week 4: 10/23 - 10/29
    - Web page write up
  + Web page: current experiments
* 11/19 - Final update
  + 10/30 - 11/5
    - Experimenting
  + 11/6 - 11/13
    - Experimenting
  + 11/14 - 19: Web Page
  + Timeline TBD
  + Web page - Completed experiments?
* 11/23 - Video
  + 1 minute video (every step of the process to illustrate the algorithm)
  + Made the webpage by a jupyter notebook