Yash Jalan
Computer Graphics w/ Prof. Panozzo

2D Tessellator!

Introduction

In this OpenGL application, a user can create and modify regular 2D polygonal tessellations. By pressing the appropriate keys, a user is presented with either a triangle, square or hexagon. Then the user can move around the control points to modify/distort the corresponding shape. Pressing X then fills the screen with repeated patterns of the distorted shape. The user is able to zoom and rotate the camera to get a different view of the tilings.

Technical/Implementation Details

I used both cubic B-splines and Lagrange interpolation to create and modify the edges of the shapes. Each edge has 5 control points, two of which are the vertices of the shape. So, a user can modify the 3 other points per edge. Each control point is connected to another corresponding control point (represented by same color), which means that if you move one, then the other moves to. The direction of the movement of the other point is such that the corresponding shape tries to emulate line, translational, rotational or glide symmetry. The hexagonal tessellations exhibit rotational and translational symmetry. The triangular tessellations exhibit translational, line and (I think) even some glide symmetry (glide symmetry within each triangle). The square tessellations exhibit translational and line symmetry.

I created a uniform clamped B-spline in parameter space [0,1]. The coefficients of the uniform clamped cubic B-spline basis function with 4 control points were precomputed (verified by hand). Using repeated first and last control points, the basis functions were evaluated on evenly spaced parameter space for each line segment (6 in total). This was different from the description of B-splines in the lecture and the source of B-splines in the lecture, since instead of knots being repeated, the control points were repeated and only 4 of the control points were used for each segment. However, I used reference (1) and it gave me satisfactory results.

For the 2D Lagrange interpolation also, I used evenly spaced parameter space [0,1]. I evaluated the interpolating polynomial for both x and y coordinates with the control points in parameter space hard-coded to 0, 0.25, 0.5, 0.75, 1. Since I was interpolating 5 points, the resulting polynomial was of degree 4.

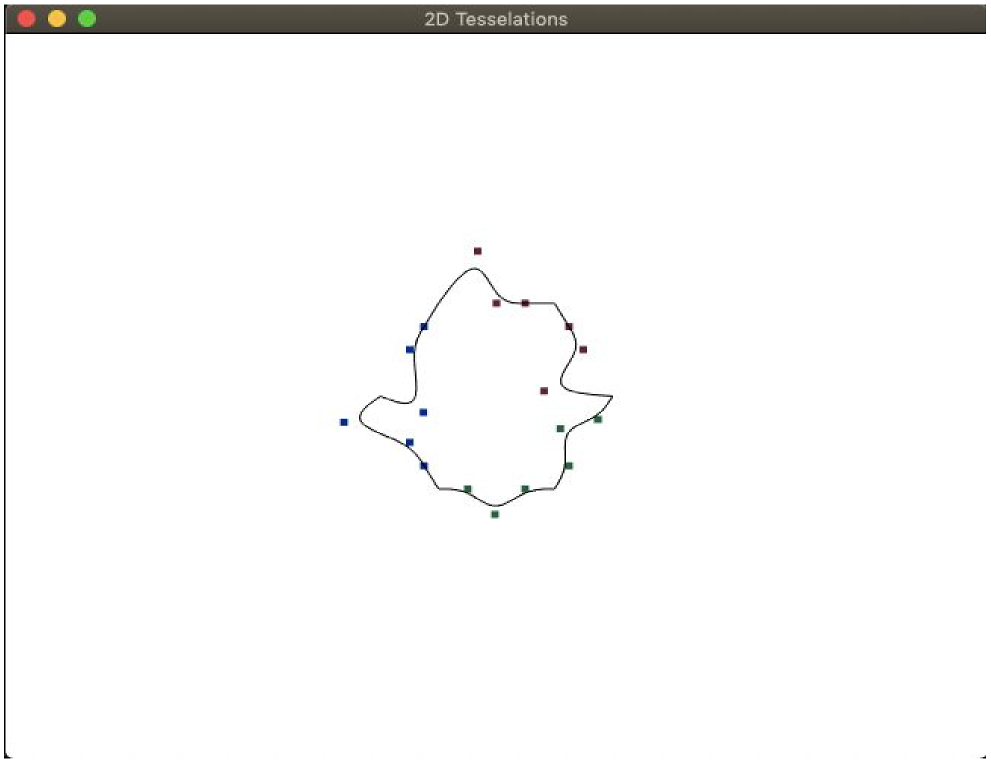
To color the distorted possibly concave shapes, I used the libigl's triangle library (2), which uses Delaunay's algorithm to find a triangulation. I included the relevant code in separate files to be able to easily call the appropriate function from it.

What keys to press?

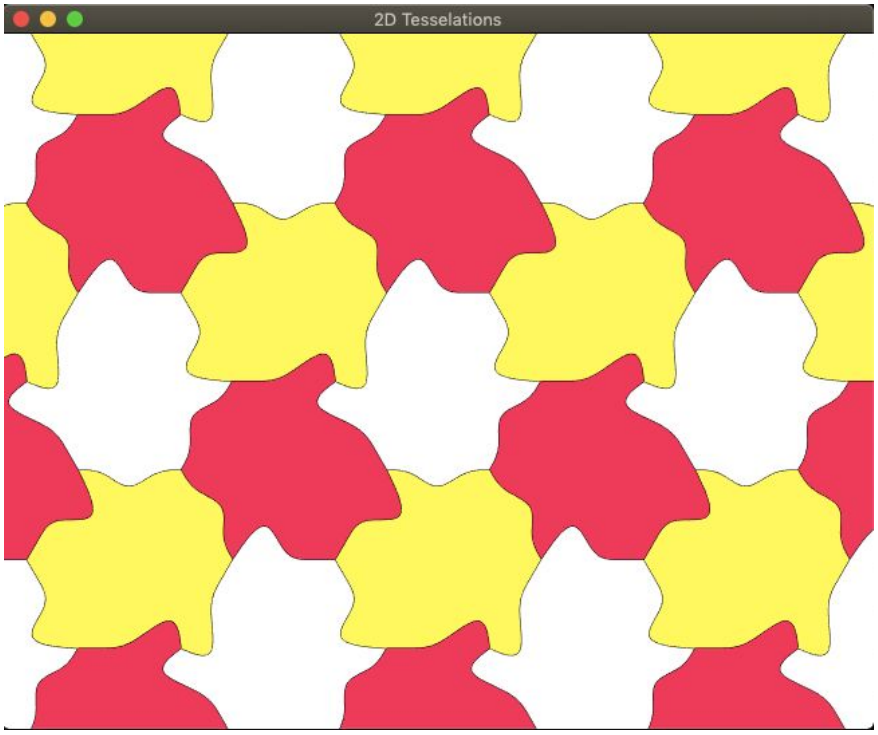
Key T: B-spline triangle
Key 1: interpolating triangle
Key S: B-spline square
Key 2: interpolating square
Key H: B-spline hexagon
Key 3: interpolating hexagon
Key X: tessellate the shape
Key Q/W: rotate camera clockwise/anti-clockwise
Key =/-: zoom in/out

Sample Images

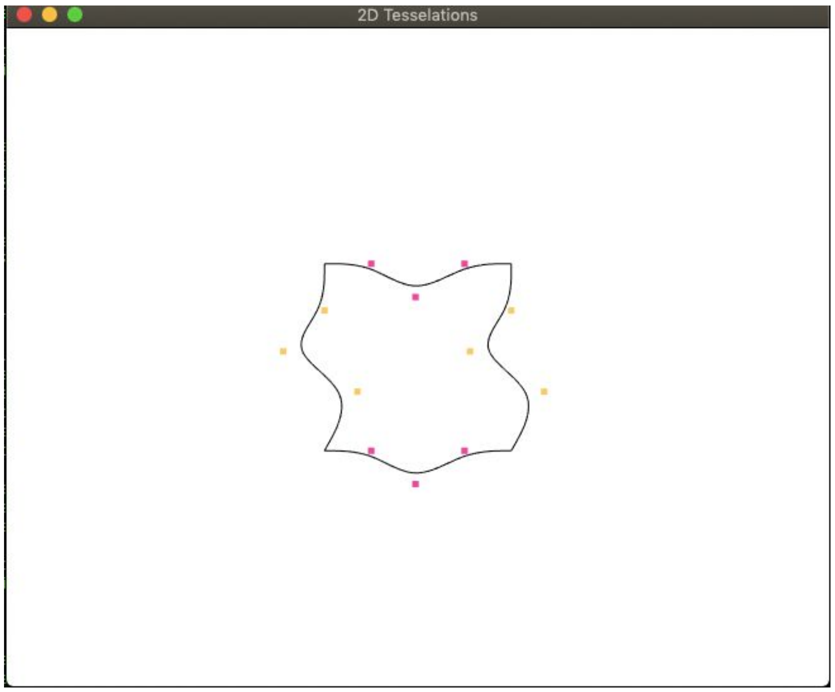
HEXAGON (before tessellate):



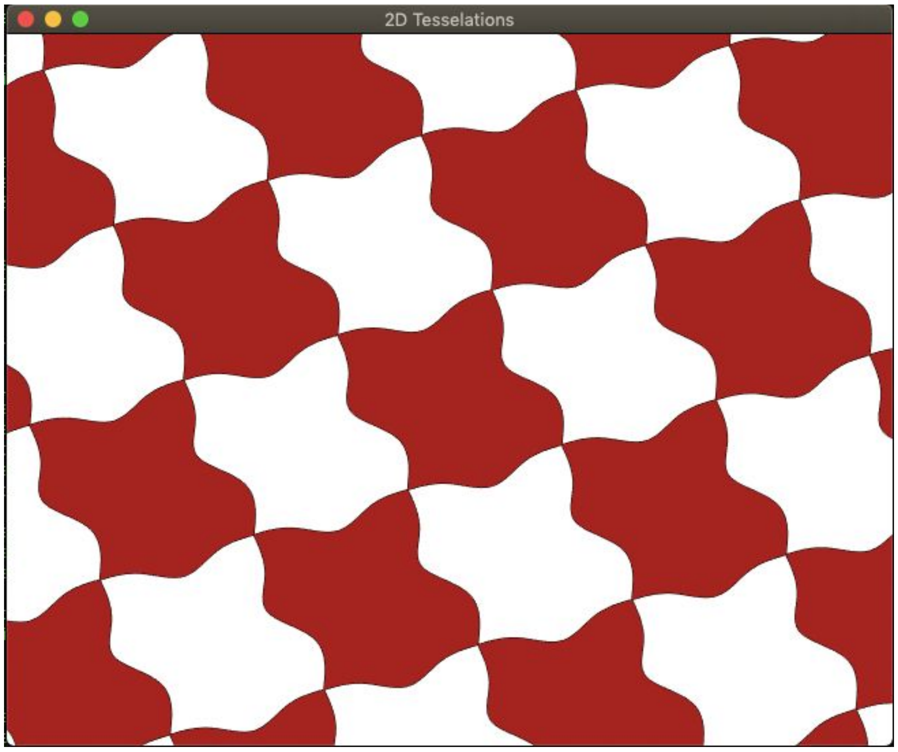
After tessellate:



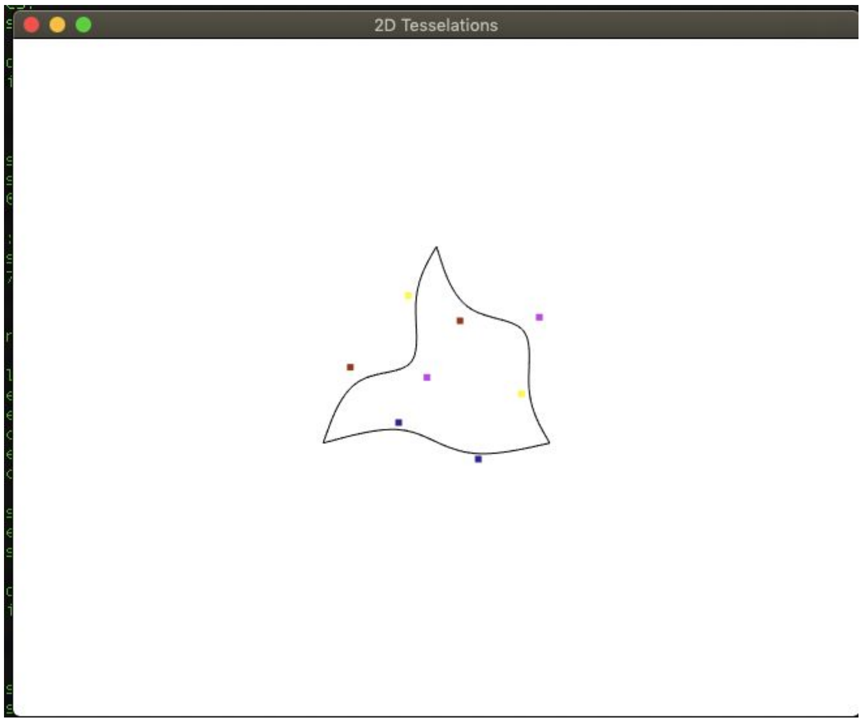
SQUARE (before tessellate):



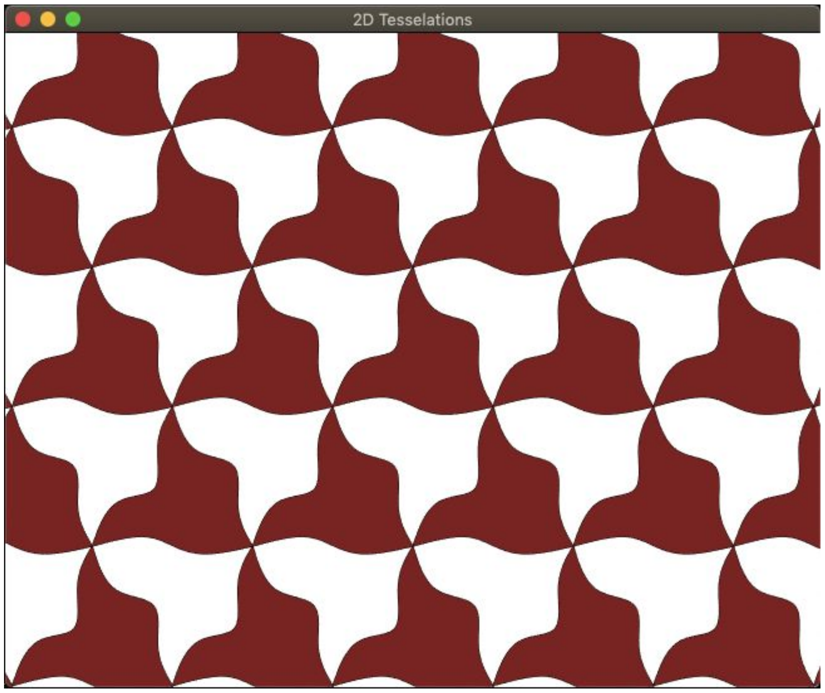
After tessellate:



TRIANGLE (before tessellate):



After tessellate:



Libraries Used/References

(1) https://nccastaff.bournemouth.ac.uk/jmacey/RobTheBloke/www/opengl_programming.html
(2) https://github.com/libigl/triangle

P.S. I couldn't successfully use .gitmodules to upload eigen, glew, glfw libraries, which is possibly why travis-ci fails. I used my Assignment 2's repository to modify and run the code.