OVERVIEW:

- This an iOS App developed using Swift and GLSL in Xcode.
- It draws Bézier Spline with each new touch input from on the screen. Based on the concept of Bézier Spline, as soon as there are 3n+1 control points (or touch inputs) on the screen it draws n splines using interpolation on those 3n+1 points. The control points can be grabbed and moved to any other position on screen. The other parts of the spline will change according to the translation without breaking the spline.
- This is a modification of a03. Here the interpolation functionality has been delegated to GPU. There is a function in the vertex shader which does the interpolation.

EXECUTION :

- Open the .xcodeproj file in Xcode.
- Run the project after setting the active scheme as iPhone6
- The can be directed to an actual handset by selecting the connected iPhone as the active scheme.

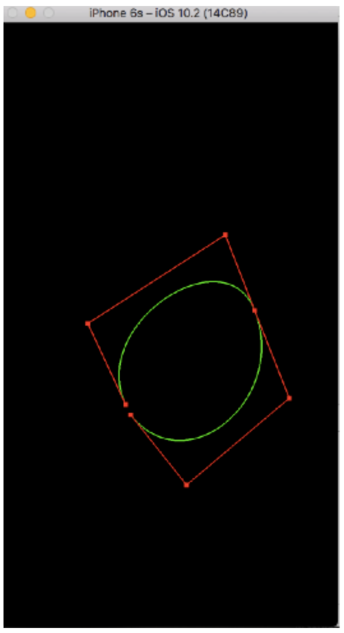
BEHAVIOR:

- When the app launches in simulator, any tap on the screen will add a control point (using GL_POINTS) at the tapped coordinates on screen. The control points are stored as uniform variables on CPU side and passed to the GPU with each shader call.
- As soon as there are 4(3*1+1) control points, first Bézier Spline will be drawn using the linear interpolation equation. Second spline segment will be drawn when there are 7(3*2+1) control points and so on.
- The interpolation is done in the vertex shader. A separate shader is called for each spline segment. Value of parameter ’t’ is passed as an attribute variable to shader.
- This Spline is flexible and any control point can be grabbed and moved/translated. The new set of points are again interpolated and the spline is redrawn without being broken.
- Derivatives are also drawn at each starting and ending control point as those are only points that are part of a Bèzeir Spline. Derivatives ensure the smoothness between two adjacent splines.

THE CODE

- In Xcode open the GameViewController.swift from the Project Navigator. This file has the touch-listeners and basic functions like setupGL(), glkView().
- After the required control points are entered, they are passed to the shader and the vertex shader does interpolations on them.
- The vertex shader (written in GLSL) also calculates the orthogonal projection matrix and multiplies it to the vertex coordinate calculated by its interpolation function and sends the result to the fragment shader.
- Fragment shader has been kept simple for this project and does not include any extra task.

SCREENSHOT:



DEPENDENCIES:

- The dependencies are: it might not draw the solid lines for some models other than iPhone6.

LIMITATIONS:

- In this project no matrix transformations have been used. To implement translation of control points, traditional insert and delete functions have been used on vertex arrays. Hence greater part of the work is being done by CPU which can be delegated to GPU using matrix transformations.

: