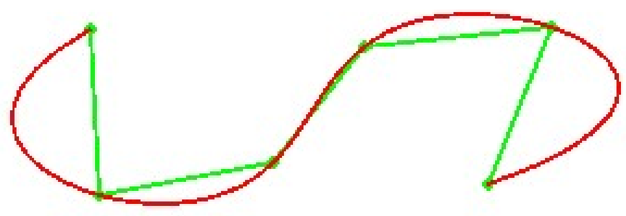
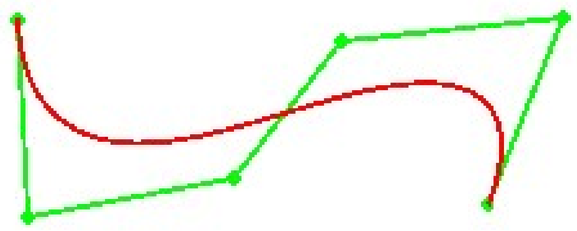
Programing Assignment #7

In this assignment you will implement the **Lagrange Interpolator Polynomial** and the **arbitrary order Bézier curve**, i.e. the Bézier curve of order *n* for given *n*+1 control points. See pictures below for examples of these curves.


Lagrange


Bézier

Download the new support code glutapp3dPolyEditor.7z from the course webpage. Inside the package you will find a complete project where you can create and edit a control polygon. Your task is to implement functions to draw the Lagrange and Bézier curves for the control polygon being edited. Your tasks are summarized in the following items:

1) Implement the following two functions to evaluate points in the Lagrange and Bézier curves. These functions will not rely on OpenGL and they must be implemented in their own files, for example you may implement them in files curve_eval.cpp/curve_eval.h :

```
GsVec eval_bezier ( float t, const GsArray<GsVec>& ctrlpnts )

GsVec eval_lagrange ( float t, const GsArray<GsVec>& ctrlpnts )
```

(small variations in the function prototypes are fine, for example, you can choose to use “GsArray” or std::vector. The support code however uses GsArray.)

2) You will then evaluate several points in each curve in order to build the OpenGL arrays to draw the curves. The recommended approach is that you create a specific class for drawing the curves, either as a single SoCurve class which can be initialized to evaluate different types of curves, or as specific classes, one for each type of curve. Include in your program the following keys to choose which curve to draw:

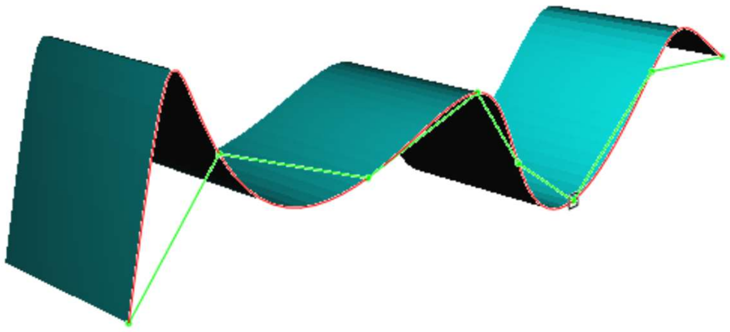
- **0** - no curve
- **1** - Lagrange
- **2** - Bézier

3) You will also need to be able to change the number of segments used to approximate each curve, and for that use the following keys:

- **q** - increase the number of segments
- **a** - decrease the number of segments

Next you will experiment with a simple way to generate a 3D surface from your 2D curves.

4) Although your curves are in 2D, you will also generate a simple smooth 3D surface from your 2D XY curve. For that, whenever the ‘**z**’ key is pressed, you are going to traverse all the vertices on the current curve approximation being displayed, and generate planar faces along the Z direction. To achieve smooth shading you may compute the normal vectors from the polygonal lines approximating your curves: given a curve vertex, take the average of the normals of the two segments adjacent to the vertex. See the picture below for an example.



Example of a 3D surface generated from a Lagrange 2D curve

5) Real-time interaction: add the option of re-generating the 3D surface in real time as you drag/edit points of the 2D curve. If the update becomes too slow then the real time behavior can be turned off and an update of the 3D curve can be called every time z is pressed. Use the following keys:

- **z** – every time ‘z’ is pressed, re-generate the entire 3D surface.
- **x** – every time ‘x’ is pressed turn on and off the real-time 3D surface update.

Grading

50% - Correct Lagrange and Bezier curves are generated and implemented.
15% - Resolution is correctly controlled.
15% - 3D Surface is correctly generated with correct smooth shading.
10% - real-time 3D surface generation is correct, and z and x keys work as expected.
10% - all controls are correct, the results look good, and the application is well developed.