* 1. Topics/Purpose
* Approximation of curves and surfaces
* Lagrange interpolation polynomials, Bézier curves, b-splines, and Cubic Hermite splines (csplines)
* Modeling of objects using patches of approximated surfaces
  1. Statement

I will create a program for modeling curves and surfaces, demonstrating the different techniques available. Options for both 2D and 3D modes will be available, for creating curves and surfaces respectively.

2D curves will be modeled using Lagrange interpolation polynomials, Bézier curves, B-splines, Cubic Hermite splines (csplines), and NURBS. 3D surfaces will also be modeled using these approximation techniques. This 3D surface mode will also include functionality to draw an object using a number of patches.

A checkbox menu will be included to turn on visibility of the control points, control polygon, and curve/surface.

The parameters for the curve/surface or object will be taken from an input file. This file will define vertices. A second file will define the individual patches for object models. A few patch models of various complexities will be included to demonstrate the functionality of this patch surface modeling technique.

Basic lighting/shading will be implemented, but the bulk of the work will be in regards to the various mathematical approximations for curves/surfaces.

* 1. Technical Outline

Lagrange interpolation polynomials are calculated to a polynomial with degree one higher than the number of input points. These polynomials will be computed brute-force, then displayed using OpenGL. The algorithm I will use to interpolate the curve is as follows:

* First, get the function value for each control point
* Then, calculate the basis polynomials for each control point
* Finally, we solve the matrix of the system of equations from the prior step to get our interpolation function

Bézier curves will be modeled using a simple “brute-force” calculation of the blending functions. The degree of the curve will be limited to cubic as computational complexity greatly increases past this point. This algorithm will simply draw a bunch of little lines (representing the level of detail), and is defined as follows:

For each little line segment (incrementally), do the following:

* Calculate the parametric time value 0 to 1
* Calculate the blending functions for each of the control points
* Calculate the x, y, z values of the curve point by summing the blending functions and the control vertices
* Specify/draw the point

B-splines are simply a generalization of Bézier curves. However, they avoid the problem of the addition of a single point affecting the look of the rest of the curve. They use only those points nearest the current parameter to calculate parameters. As with Bézier curves, this algorithm will simply draw a bunch of little lines (representing the level of detail) and is defined as follows:

For each little line segment (incrementally), do the following:

* Calculate the parametric time value 0 to 1
* Calculate the blending functions for each of the control points
* Calculate the x, y, z values of the curve point by summing the control points multiplied by their respective blending functions
* Specify/draw the point

Cubic Hermite splines (csplines) are third-degree splines with each polynomial of the spline in Hermite form. They can be converted into the form of bézier curves and then drawn using the same methodology. Again, this algorithm will simply draw a bunch of little lines (representing the level of detail) and is defined as follows:

For each little line segment (incrementally), do the following:

* Calculate the parametric time value 0 to 1
* Calculate the blending functions for each of the control points
* Calculate the x, y, z values of the curve point
* Specify/draw the point

NURBS (non-uniform rational basis splines) are simply b-splines with a weight assigned to every control point. They are defined using control points as well as a knot vector. These will be defined in the input file. The algorithm for drawing a NURBS spline is a little more complicated than the prior methods, and is defined as follows:

* Calculate the parametric time value using a combination of the knot function
* Sum the effect of all the CoxDeBoor values on the curve at this control point
* To calculate the CoxDeBoor values, use the knot function and the points around the control points
* This sum is the point at the time value
* Specify/draw the point

3D surfaces are similar in logic to the above cases. Each surface is simply defined by a net of curves in two parametric directions (u and v). We must define a grid of control points over the surface in order to calculate the approximation to the surface. Then for each parametric time value, we run the following algorithm:

* Calculate the parametric u value
* For each point along the u direction, calculate the parametric v value
* Using these two points, we calculate the point on the surface (using extensions on the above algorithms)
* Finally, specify/draw the point

Patches are essentially small pieces of a 3D surface, defined to simplify the calculation of the entire surface. Pieced together, we can define an object using a number of these “primitives,” reducing computation time. If our input is defined correctly, the drawing of multiple patches should ensure the seamlessness of our object.

Input is defined using input files. The first line will describe which rendering method is to be used, the second line will describe whether the scene should be rendered in 2D or 3D, and the rest of the file will describe a number of vertices. NURBS also requires a knot vector to be defined in this input file. For patches, a second file is included which defines each of the patches defining the object using the input vertices.

* 1. Bibliography

1. Fundamentals of Computer Graphics, Third Edition. Marschner and Shirley, A K Peters, 2009, pp. 339-383 (Curves). - *This chapter of the excellent Shirley book defines the basics of curve approximation and describes the algorithms for most of the used approximation techniques.*
2. Implementing Surfaces in OpenGL. Zhao, University of Waterloo, 2006. *– This Masters these by Hui Zhao describes a number of pitfalls occurring when implementing 3D surfaces, as well as the basics of 3D surface approximation.*
3. OpenGL Programming Guide, Seventh Edition (“The Red Book”). Addison-Weslet Professional, 2009, Chapter 12 (Evaluators and NURBS). *– Instructs how to draw basic curves and surfaces using OpenGL evaluator commands, as well as GLU’s higher-level NURBS facility to draw more complex curves and surfaces.*
   1. Objectives

* Implement the drawing of 2D curves using Lagrange interpolation polynomials.
* Implement the drawing of 2D curves using Bézier curves.
* Implement the drawing of 2D curves using b-splines.
* Implement the drawing of 2D curves using Cubic Hermite splines (csplines).
* Implement the drawing of 2D curves using NURBS.
* Extend the drawing of curves drawing to work on 3D surfaces.
* Use the 3D surface modeling to implement the drawing of a single input patch.
* Extend this 3D surface modeling method to draw an arbitrary number of input patches.
* Implement seamless patch boundaries, ensuring seams between patches are smooth and filled.
* Create input files defining an object requiring at least twenty patches.