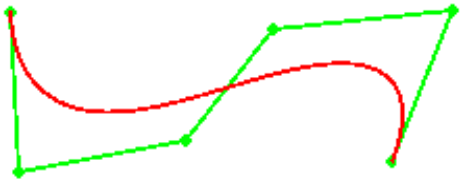
Programming Assignment #5 (Optional)

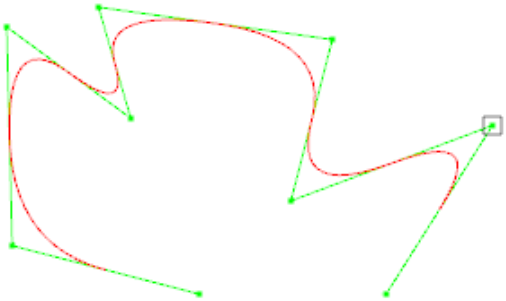
In this assignment you will implement the following parametric curves:

- the **arbitrary order Bézier curve**,
- a **quadratic B-Spline** using the Cox-de Boor formulation with uniform knot vectors,

Their formulations are available in the lecture slides. See illustrations below.



Bézier



Quadratic BSpline

Your task is to render the original control polygon (i.e., the green lines above), and then the interpolated curve above it (i.e., the red lines above) using either Bézier or B-Spline. Use what you have learned from your torus to render the curves as lines. You may use the existing axes in the support code as a starting point, and do not have to worry about lighting or texture.

Requirements:

Requirement 1 (50%): Curves.

Implement separate functions to evaluate points in each spline. For example:

```
/*! Bezier of order n for given n=P.size()-1 control points, t in [0,1] */
vec2 eval_bezier( float t, const vector<vec2>& P )

/*! B-Spline of order k, n=P.size()-1.
    For order k=3 (degree 2, quadratic case): t in [2,n+1]  */
vec2 eval_bspline( float t, int k, const vector<vec2>& P );
```

In these functions, P is the array containing the control points. Feel free to update/adapt the names and parameter list of the functions as needed for your project.

The above functions will allow you to evaluate several points in each curve in order to draw the curve as a polygonal line. Note that the *t* range is expected to be different for each type of curve, so be sure to pay attention to that and to scale/adapt the *t* and delta *t* parameters as/if needed.

The support code already includes controls to turn on or off the display of each curve, and to vary the delta *t* step to evaluate curve points. Keep this functionality and integrate it with your curves. You should achieve the functionality of displaying each curve independently or to draw all of them together. This will be useful for you to compare the curves and to study and verify their properties.

Requirement 2 (15%): Resolution (delta *t*).

Your implementation should allow for changing the delta *t* with the keyboard in order to evaluate more (smaller delta *t*) or less (higher delta *t*) curve points for the interpolated curve. The result should be updated in real-time.

Requirement 3 (30%): Editing control points.

Add functionality for moving the control points of the 2D curve, either via the mouse or with the keyboard. As the control points are edited, your curve should update in real-time. You should be able to pick a specific control point to move, and be able to move it along all 3 axes.

Requirement 4 (5%): Overall quality.

Everything counts here and, once again, there is no need to do anything complex. Just make sure your project works and looks good and you will get full points.

Grading

50% - Correct curves for both methods are generated and implemented in 2D
15% - Resolution (delta *t*) correctly controls the number of points in each curve.
30% - Ability to modify control points and regenerate the curve in real-time.
5% - The overall user interface works correctly and the project looks well prepared.

Submission:

Please follow the instructions in parules.txt (uploaded to CatCourses). In particular: please do not include any third-party support code and do not forget to Clean Solution before preparing your project for submission! Also, check for hidden folders (such as .vs) which can sometimes balloon to hundreds of megabytes!

Deadline lab for presentation and submission: your last lab.