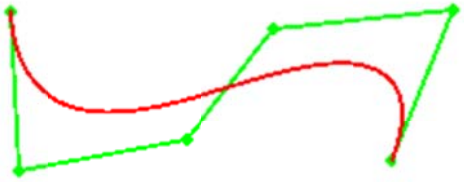
Programing Assignment #6 – Parametric Curves

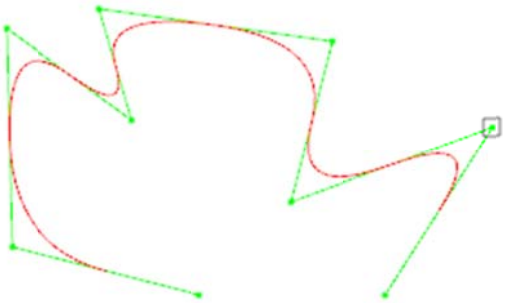
In this assignment you will implement several parametric curves:

- the **arbitrary order Bézier curve**,
- a **quadratic B-Spline** using the Cox-de Boor formulation with uniform knot vectors,
- the **Catmull-Rom** spline interpolator, and
- the **Bessel-Overhauser** spline interpolator.

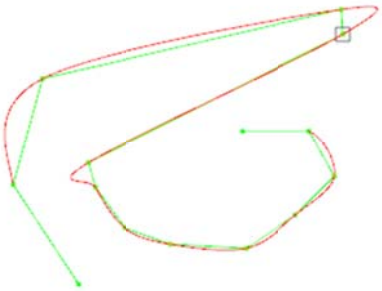
Their formulations are available in the lecture slides. See illustrations below.



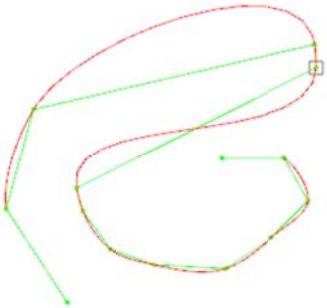
Bézier



Quadratic BSpline



Catmull-Rom



Bessel-Overhauser

Start this assignment from the new support code sigpolyed.7z available from the course webpage. Inside the package you will find a complete project where you can already create and edit a control polygon. Your first task is to implement functions to draw the curves above based on the control polygon being edited. Your tasks are summarized in the following items:

1) Curves: Implement the following functions to evaluate points in each spline. These functions have to be implemented as separate functions:

```
/*! Bezier of order n for given n=P.size()-1 control points, t in [0,1] */
GsPnt2 eval_bezier ( float t, const GsArray<GsPnt2>& P )

/*! B-Spline order k, n=P.size()-1.
    For order k=3 (degree 2, quadratic case): t in [2,n+1]  */
GsPnt2 eval_bspline ( float t, int k, const GsArray<GsPnt2>& P );

/*! Evaluates a Catmull-Rom cubic spline, n=P.size()-1, t in [0,n-2] */
GsPnt2 crspline ( float t, const GsArray<GsPnt2>& P );

/*! Evaluates a Bessel-Overhauser spline, n=P.size()-1, t in [0,n-2] */
GsPnt2 bospline ( float t, const GsArray<GsPnt2>& P );
```

In these functions, P is an array containing the control points. These functions must be implemented one for each curve equation. Feel free to update/adapt the names and parameter list of the functions as needed for your project. The support code already includes a placeholder for these functions with a simple code that just evaluates points between the first and last points of the control polygon.

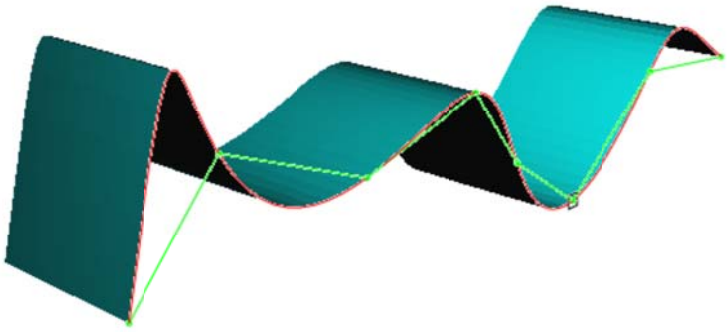
The above functions will allow you to evaluate several points in each curve in order to draw the curve as a polygonal line. Note that the *t* range is expected to be different for each type of curve, so be sure to pay attention to that and to scale/adapt the *t* and delta *t* parameters as/if needed.

(Note: you can of course choose to use std::vector instead of GsArray, but a conversion would be needed as the support code uses GsArray.)

2) User interface: the support code already includes a user interface 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.

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

3) Surface: Although your curves are in 2D, you will also generate a simple smooth 3D surface from your 2D XY curve. For that, whenever the spacebar 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 normal vectors of the two segments adjacent to the vertex. See the picture below for an example of how your results should look like. (The other option to compute the normal vectors is to compute them from the curve derivatives, which will be a specific procedure for each type of curve.)



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

You can accomplish the 3D surface by creating a SnModel with the needed vertices, faces and normal vectors - something that should be straightforward to assemble based on what you have learned in previous assignments.

You may choose to display the surface for all the currently active curves or for just one of the active ones, as long as a surface can be generated for any type of active curve.

4) Real-time Interaction: add the functionality of re-generating the 3D surface in real time as you edit the control points of the 2D curve. If the update becomes too slow then the delta t can be reduced or the 3D display can be turned off with another press of the spacebar key.

Spacebar key summary: every press alternates between 2D mode and 3D surface mode.

Grading

60% - Correct curves for all the 4 types are generated and implemented in 2D.
10% - Resolution (delta t) correctly controls the number of points in each curve.
15% - 3D Surface is correctly generated with correct smooth shading.
10% - real-time 3D surface generation is correct, and spacebar works as expected.
5% - the overall user interface works correctly and the project looks well prepared.

Submission:

Please present and submit your PA as usual according to parules.txt. (Do not forget to upload your project before the deadline!)