CSE 167 (FA 2021) Homework 4

In this homework, we will explore the 2D vector graphics with spline curves. You will use the de Casteljau Algorithm, and variations of it, to draw Bézier curves, B-Splines, and subdivision curves.

There is also an extra credit portion for this homework. The extra credit is to build a surface of revolution using the 2D spline curve you have developed.

4.1 Assignment Overview

Before developing the code, make sure you can compile the skeleton code. As you run the program, you should get a white canvas on which you can add control points (left click), drag control points, and delete control points (right-click). Pressing 0,1,2,3 will (eventually) show

- `0`: just the control polygon,
- `1`: Bézier spline,
- `2`: B-spline,
- `3`: Subdivided control polygon that converge to the Bézier spline.

Keys `+`, `-` increase/decrease the resolution for the Bézier spline and B-spline. Keys `a`, `z` increase/decrease the depth of recursive subdivisions for the subdivided curve.

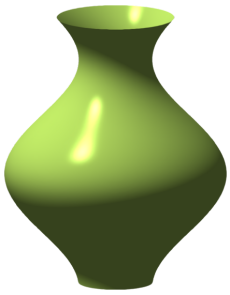
You won't see any additional curve for mode 1,2,3. Your job is to fill in the sections of `src/Spline.cpp`. Specifically `Spline::Bezier`, `Spline::BSpline` and `Spline::Subdiv`.

Submission

Submit the `Spline.cpp` file.

Extra Credit (5 pts)

After you have developed `Spline.cpp` (whose dependencies are `Spline.h`, `Curve.h` and `ControlCurve.h`) you can use this mini-library to further build some smooth surfaces. The most straightforward surface you can build is a surface of revolution. Using a smooth planar curve $\mathbf{f}: [0,1] \to \mathbb{R}^2$, with components denoted by $\mathbf{f}(t) = (f_x(t), f_y(t))$, the surface of revolution is a parametrized surface:



$$\begin{cases} x(s,t) = f_x(t)\sin(s) \\ y(s,t) = f_y(t) \\ z(s,t) = f_x(t)\cos(s) \end{cases} \qquad s \in [0, 2\pi], t \in [0,1]. \tag{1}$$

Build a triangle mesh that realizes this surface. You can use your scene builder and phong shading from HW3 to render it. To submit extra credit, upload a screenshot and submit a zipped up source code.

4.2 Background

In the program, there are two curves: the `control` (control polygon), and the `curve` (the spline/-subdivided curve). The user controls the `control` curve.

Let `vec2s` denote `std::vector<glm::vec2>` for short (array of $\mathbb{R}^2$ positions).

Each of these curves is a (pointer to an) instance of the class `Curve` (defined in `Curve.h`). Each curve has a member called

```
vec2s P;
```

which is the list of point positions that constitute the curve. There are also three member functions that are convenient:

```
void Curve::addPoint( glm::vec2 position ); // add another point to the curve
void Curve::clear();  // clear all points
int Curve::size();    // returns the number of points in the curve
```

For example, you can query the size by calling

```
int num_of_control_pts = control -> size();
```

The zeroth point position is

```
glm::vec2 P0 = control -> P[0];
```

and so on. See the header files for Curve (and its inherited class ControlCurve). Read `Scene.cpp` to see how each relevant function will be called in the program.

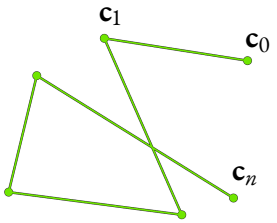
Now, in the part where we are coding, such as `Spline::Bezier(ControlCurve* control, Curve* curve, int resolution)`, we assume the user has already provided `control`. Our goal is to modify the state of `curve` so that it becomes a curve with (`resolution` + 1) many points following the Bezier curve. You can also assume that `curve` has been cleared (no points) to start with. Similar idea applies to the B-spline and subdivision: Read data from `control`, and construct content for `curve`.

In the following, we explain the definitions for each of the 3 functions. For simplicity of exposition, we use the following notations. In the program, the user selects $(n+1)$ points in the plane. Let us call the positions of the points $\mathbf{c}_0, \ldots, \mathbf{c}_n$, where $\mathbf{c}_i = (x_i, y_i)$. These points are called the **control points**. The program can display 4 types of curves:

0. Polygonal curve
1. Bézier curve
2. Cubic B-spline
3. Subdivision Bézier curve

4.2.0 Polygonal Curve

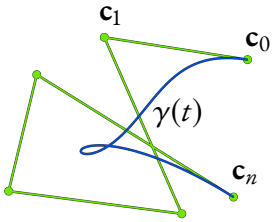
The polygonal curve connects the $(n+1)$ points $\mathbf{c}_0, \ldots, \mathbf{c}_n$ by n straight line segments. Mathematically, the curve is assembled by n parametric curves $\gamma_i : [0,1] \to \mathbb{R}^2$, $i = 0, \ldots, n-1$, where $\gamma_i(t) = (1-t)\mathbf{c}_i + t\mathbf{c}_{i+1}$. That is, $\gamma_i(t)$ linearly interpolates the consecutive points $\mathbf{c}_i$ and $\mathbf{c}_{i+1}$.



4.2.1 Bézier Curve

The Bézier curve controlled by $\mathbf{c}_0, \ldots, \mathbf{c}_n$ is a single curve (instead of a piecewise-defined curve)

$$\gamma : [0,1] \to \mathbb{R}^2, \quad \gamma(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix}. \tag{2}$$



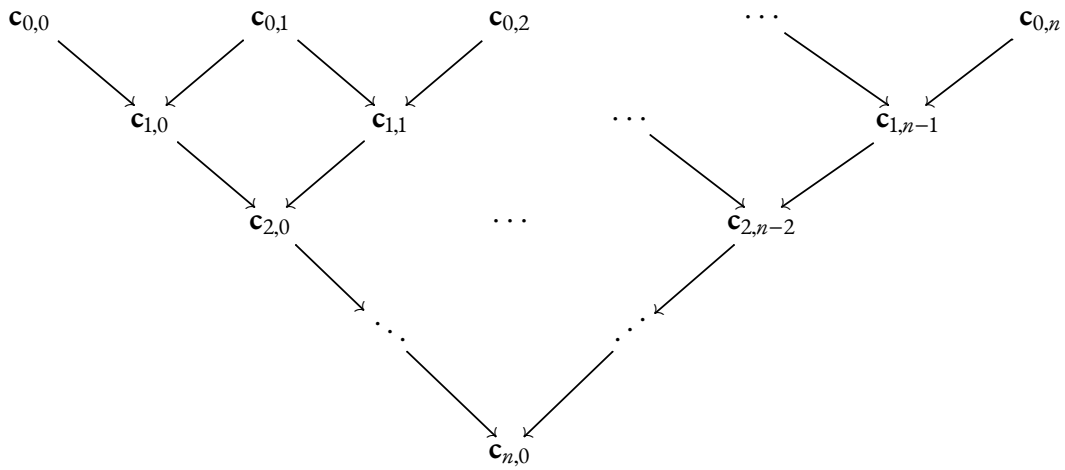
It has the following features:

- Both $x(t)$ and $y(t)$ are n-th order polynomials in t.
- $\gamma(0) = \mathbf{c}_0$ and $\gamma(1) = \mathbf{c}_n$.
- $\gamma'(0)$ is parallel to $\mathbf{c}_1 - \mathbf{c}_0$ and $\gamma'(1)$ is parallel to $\mathbf{c}_n - \mathbf{c}_{n-1}$.

More precisely, $\gamma(t)$ is given in terms of the Bernstein polynomials by

$$\gamma(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} \mathbf{c}_k \tag{3}$$

However, numerically evaluating the multiplications in this formula is expensive and unstable. The de Casteljau algorithm is a numerically stable and efficient way to evaluate (3). Let $\mathbf{c}_{0,k} = \mathbf{c}_k$, $k = 0, \ldots, n$. Construct



(4)

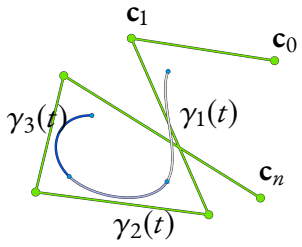
where each "funnel" is a linear (affine) interpolation

$$\mathbf{c}_{j,k} = (1-t)\mathbf{c}_{j-1,k} + t\mathbf{c}_{j-1,k+1}. \tag{5}$$

Then the final aggregated value $\mathbf{c}_{n,0}$ equals to the desired $\gamma(t)$. Note that one does not need to store all the $O(n^2)$ values of $\mathbf{c}_{j,k}$'s. One only need to use $O(n)$ memory for this calculation.

4.2.2 B-Spline

The B-spline is composed of $(n-1)$ curve segments $\gamma_1, \ldots, \gamma_{n-1}$, each of which is a cubic polynomial. The junction between γ_k and γ_{k+1} is G^2; that is, the curve is continuous with continuous tangent and continuous curvature. The curve segment γ_k is only a function of the four control points $\mathbf{c}_{k-1}, \mathbf{c}_k, \mathbf{c}_{k+1}, \mathbf{c}_{k+2}$.



For a *uniform B-spline*, the joint curve is parameterized as $\gamma\colon [1, n-1] \to \mathbb{R}^2$ with

$$\gamma(t) = \begin{cases} \gamma_1(t) & 1 \le t \le 2 \\ \gamma_2(t) & 2 \le t \le 3 \\ \vdots & \\ \gamma_{n-1}(t) & n-1 \le t \le n. \end{cases} = \sum_{k=0}^{n} \mathbf{c}_k B_{k,4}(t-k) \tag{6}$$

where $B_{k,4}$ is the basis piecewise cubic polynomial whose definition can be found here (https://en.wikipedia.org/wiki/B-spline#Definition). Instead of explicitly computing the rather complicated formula for B, we will use the de Casteljau algorithm and the cubic blossom. Consider $F(t_1, t_2, t_3)$ being a function on 3 scalar variables t_1, t_2, t_3 such that

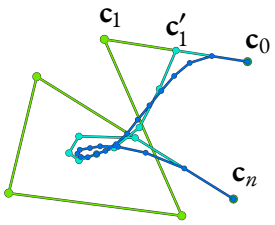
- F is symmetric: $F(t_1, t_2, t_3) = F(t_2, t_1, t_3) = F(t_1, t_3, t_2)$.
- F is tri-affine: Fixing any pair of the 3 variables, the remaining 1-variate function is affine $F(ax + by, t_2, t_3) = aF(x, t_2, t_3) + bF(y, t_2, t_3)$ $(a + b = 1)$.

Now, assign

$$F(-1,0,1) = \mathbf{c}_0, \quad F(0,1,2) = \mathbf{c}_1, \quad F(1,2,3) = \mathbf{c}_2, \quad F(2,3,4) = \mathbf{c}_3, \quad \text{etc.} \tag{7}$$

Our final desired $\gamma(t)$ is given by $F(t,t,t)$. Using the symmetry and tri-affinity of F, one can uniquely compute $\gamma(t) = F(t,t,t)$, say for $1 \le t \le 2$, using the values of $F(-1,0,1)$, $F(0,1,2)$, $F(1,2,3)$, $F(2,3,4)$. Concretely, from $F(-1,0,1) = F(0,1,-1)$ and $F(0,1,2)$ one computes $F(0,1,t)$ through affine combination. Similarly, one obtains $F(1,2,t)$ and $F(2,3,t)$ from other pairs. Then, from $F(0,1,t) = F(1,t,0)$ and $F(1,2,t) = F(1,t,2)$ one computes $F(1,t,t)$. Similarly, one gets $F(2,t,t)$. Finally, one computes $F(t,t,t)$.

4.2.3 Subdivision Bézier Curve



A spline, such as the Bézier curve, is determined from a set of control points. For many spline schemes (including Bézier) we observe that a spline from a coarse set of control points $\mathbf{c}_0, \ldots, \mathbf{c}_n$ happens to be the same as the spline coming from another finer set of control points $\mathbf{c}'_0, \ldots, \mathbf{c}'_n$ and $\mathbf{c}'_n, \ldots, \mathbf{c}'_{2n}$. The process of given $\mathbf{c}_0, \ldots, \mathbf{c}_n$ and compute $\mathbf{c}'_0, \ldots, \mathbf{c}'_{2n}$ is called subdivision. The idea of subdivision is that, instead of drawing the spline, we will just draw the control points after a few iteration of subdivisions. The subdivision points converge to the Bézier spline under multiple recursion of subdivisions.

Note that the Bézier curve can also be viewed in terms of the blossom. For $(n+1)$ control points, the function $F(t_1, \ldots, t_n)$ is a symmetric n-affine function. Letting $F(0,0,\ldots,0) = \mathbf{c}_0$, $F(0,0,\ldots,0,1) = \mathbf{c}_1$, $F(0,\ldots,0,1,1) = \mathbf{c}_2$,.., $F(1,\ldots,1) = \mathbf{c}_n$, we have the Bézier curve given by $\gamma(t) = F(t,t,\ldots,t)$ through repeated affine combinations (unfolding the de Castejau algorithm (4)). In this formalism, the subdivided points are

$$\mathbf{c}'_0 = F(0,\ldots,0,0), \quad \mathbf{c}'_1 = F(0,\ldots,0,\tfrac{1}{2}), \quad \mathbf{c}'_2 = F(0,\ldots,\tfrac{1}{2},\tfrac{1}{2}), \quad \ldots, \quad \mathbf{c}'_n = F(\tfrac{1}{2},\ldots,\tfrac{1}{2}),$$
$$\mathbf{c}'_{n+1} = F(\tfrac{1}{2},\ldots,\tfrac{1}{2},1), \quad \mathbf{c}'_{n+2} = F(\tfrac{1}{2},\ldots,1,1), \quad \ldots, \quad \mathbf{c}'_{2n} = F(1,\ldots,1).$$

Each of these $\frac{1}{2}$ points can be evaluated by a de Casteljau type averaging.

It might also be useful to compute the linear combination coefficients

$$\begin{bmatrix} \mathbf{c}'_0 \\ \mathbf{c}'_1 \\ \vdots \\ \mathbf{c}'_n \\ \vdots \\ \mathbf{c}'_{2n} \end{bmatrix} = \underbrace{\begin{bmatrix} s_{00} & \cdots & s_{0n} \\ s_{10} & \cdots & s_{1n} \\ \vdots & & \vdots \\ s_{n0} & \cdots & s_{nn} \\ \vdots & & \vdots \\ s_{2n,n} & \cdots & s_{2n,2n} \end{bmatrix}}_{\text{subdivision matrix}} \begin{bmatrix} \mathbf{c}_0 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} \tag{8}$$

4.3 Specifications

Call r = `resolution` and ℓ = `subdivLevel` for short.

Bezier Devide the curve into r segments. Hence, you need to evaluate the curve at $r+1$ points, connecting these with line segments. The evaluation can be done by the deCasteljau algorithm. You can also evaluate the curve by the explicit Bernstein polynomials.

Bspline This is the cubic B-spline. The curve is again divided into r straight line segments with parameter t ranging from 1 to $n-1$ (if there are $n+1$ control points $\mathbf{c}_0, \ldots, \mathbf{c}_n$). Each integer interval $k \le t < k+1$ corresponds to a piece of the curve γ_k computed using only the 4 points $\mathbf{c}_{k-1}, \mathbf{c}_k, \mathbf{c}_{k+1}, \mathbf{c}_{k+2}$. You can either use the deCasteljau-type algorithm, or the explicit B-spline basis (e.g. in terms of the spline matrix and geometry matrix).

Bezier2 This is the subdivision Bézier curve. The level of detail ℓ represents the level of recursion. So, note that the number of points to draw grow exponentially in ℓ. You can draw the curve by nested for loops, or by recursion. Each subdivision splits the curve at its midpoint each time. You can use the deCasteljau algorithm for the subdivision.

4.3.1 Extra credit

Use `Spline` methods you developed to generate a smooth curve, and use it to build a smooth surface of revolution. You can assign the control points for the smooth curve just in the code (without making a full-blown GUI). You can use the code skeleton from HW3. Specifically, your surface of revolution may be a inherited class of `Geometry` so that you can put it in the scene similar to how you build a scene in HW3. For example, the surface of revolution can be a vase that can be put on top of the table next to a teapot.

Take a screenshot of the result. Upload the zip file of all the source code.