CSE 167 (WI 2025) Homework 4 – Due 3/7

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.

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 B-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.

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`.

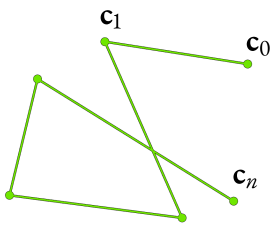
In the following, we explain the definitions for each of the 3 functions. For simplicity of exposition, we use the following notations. In the program, the user selects $(n+1)$ points in the plane. Let us call the positions of the points $\mathbf{c}_0, \ldots, \mathbf{c}_n$, where $\mathbf{c}_i = (x_i, y_i)$. 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 curve (that would refine to B-spline)

Note

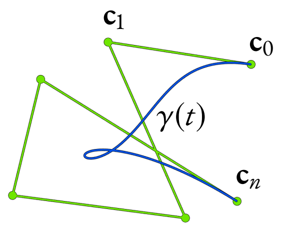
The n in this document equals to the number of control points (`control -> size()`) *minus one*.

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 $\gamma_i\colon [0,1] \to \mathbb{R}^2$, $i = 0, \ldots, n-1$, 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)

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

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

$$\gamma(t) = \sum_{k=0}^{n} \binom{n}{k} t^k (1-t)^{n-k} \mathbf{c}_k \tag{2}$$

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 (2). Let $\mathbf{c}_{0,k} = \mathbf{c}_k$,

$k = 0, \ldots, n$. Construct

$$\begin{tikzcd}
\mathbf{c}_{0,0} \arrow[dr] & & \mathbf{c}_{0,1} \arrow[dl] \arrow[dr] & & \mathbf{c}_{0,2} \arrow[dl] & & \cdots \arrow[dr] & & \mathbf{c}_{0,n} \arrow[dl] \\
& \mathbf{c}_{1,0} \arrow[dr] & & \mathbf{c}_{1,1} \arrow[dl] & & \cdots \arrow[dr] & & \mathbf{c}_{1,n-1} \arrow[dl] & \\
& & \mathbf{c}_{2,0} \arrow[dr] & & \cdots & & \mathbf{c}_{2,n-2} \arrow[dl] & & \\
& & & \ddots \arrow[dr] & & \iddots \arrow[dl] & & & \\
& & & & \mathbf{c}_{n,0} & & & &
\end{tikzcd} \tag{3}$$

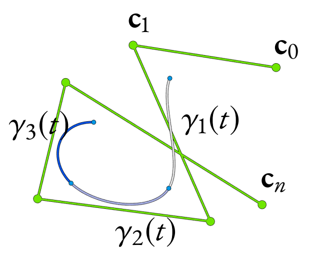
where each "funnel" is a linear (affine) interpolation

$$\mathbf{c}_{j,k} = (1-t)\mathbf{c}_{j-1,k} + t\mathbf{c}_{j-1,k+1}. \tag{4}$$

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

$$\gamma(t) = \begin{cases} \gamma_1(t) & 1 \le t \le 2 \\ \gamma_2(t) & 2 \le t \le 3 \\ \vdots & \\ \gamma_{n-1}(t) & n-1 \le t \le n. \end{cases} = \sum_{k=0}^{n} \mathbf{c}_k B_{k,4}(t-k) \tag{5}$$

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

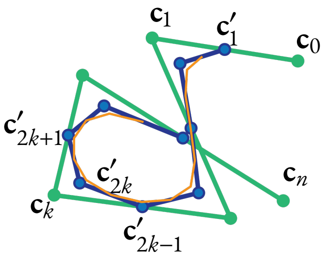
- F is symmetric: $F(t_1, t_2, t_3) = F(t_2, t_1, t_3) = F(t_1, t_3, t_2)$.
- F is tri-affine: Fixing any pair of the 3 variables, the remaining 1-variate function is affine $F(ax + by, t_2, t_3) = aF(x, t_2, t_3) + bF(y, t_2, t_3)$ $(a + b = 1)$.

Now, assign

$$F(-1,0,1) = \mathbf{c}_0, \quad F(0,1,2) = \mathbf{c}_1, \quad F(1,2,3) = \mathbf{c}_2, \quad F(2,3,4) = \mathbf{c}_3, \quad \text{etc.} \tag{6}$$

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 Curve



A spline, such as the B-spline curve, is determined from a set of control points. We observe that a cubic B-spline from a coarse set of control points $\mathbf{c}_0, \ldots, \mathbf{c}_n$ happens to be the same as the cubic B-spline spline coming from another finer set of control points $\mathbf{c}'_1, \ldots, \mathbf{c}'_{2n-1}$. Concretely,

1. $\mathbf{c}'_{2k+1} = \frac{1}{2}(\mathbf{c}_k + \mathbf{c}_{k+1})$, for $k = 0, \ldots, n-1$;
2. $\mathbf{c}'_{2k} = \frac{1}{4}\mathbf{c}'_{2k-1} + \frac{1}{2}\mathbf{c}_k + \frac{1}{4}\mathbf{c}'_{2k+1}$, for $k = 1, \ldots, n-1$.

The second step is equivalent to $\mathbf{c}'_{2k} = \frac{1}{8}\mathbf{c}_{k-1} + \frac{3}{4}\mathbf{c}_k + \frac{1}{8}\mathbf{c}_{k+1}$ by combining step 1. The process of given $\mathbf{c}_0, \ldots, \mathbf{c}_n$ and compute $\mathbf{c}'_1, \ldots, \mathbf{c}'_{2n-1}$ 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. Note that the number of points grow exponentially under the repeated subdivisions. The subdivision points converge to the cubic B-spline under multiple recursion of subdivisions.

A way to understand the subdivision schemes is to look at the blossom. As explained in Section 4.2.2, we have $F(-1,0,1) = \mathbf{c}_0$, $F(0,1,2) = \mathbf{c}_1$, ..., $F(n-1,n,n+1) = \mathbf{c}_n$, with arguments of the tri-affine function F successively increasing. The subdivided control points are in fact the evaluation of F also on successively increasing indices but with half the step size: $\mathbf{c}'_1 = F(0,0.5,1)$, $\mathbf{c}'_2 = F(0.5,1,1.5)$, $\mathbf{c}'_3 = F(1,1.5,2)$, ..., $\mathbf{c}'_{2n-1} = F(n-1,n-0.5,n)$. In particular, they represent the same B-spline.

Another common way to describe the subdivision scheme is to summarize the averaging process in terms of a subdivision matrix

$$\begin{bmatrix} \mathbf{c}'_1 \\ \mathbf{c}'_2 \\ \mathbf{c}'_3 \\ \vdots \\ \mathbf{c}'_{2n-1} \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{1}{2} & \frac{1}{2} & & & & \\ \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & & & \\ & \frac{1}{2} & \frac{1}{2} & & & \\ & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{1}{8} & \frac{3}{4} & \frac{1}{8} \\ & & & & \frac{1}{2} & \frac{1}{2} \end{bmatrix}}_{\text{subdivision matrix}} \begin{bmatrix} \mathbf{c}_0 \\ \vdots \\ \mathbf{c}_n \end{bmatrix} \tag{7}$$

4.3 Specifications

Call $r =$ `resolution` and $\ell =$ `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).

Subdiv This is the subdivision curve. The level of detail ℓ represents the level of repeated subdivisions. So, note that the number of points to draw grow exponentially in ℓ. Each subdivision turns a curve with n edges into a curve with $2n - 2$ edges. That is, it turns a curve with m points into a curve with $2m - 3$ points. With $\ell \approx 6$ we should see that the subdivision curve looks the same as the the BSpline.

Programming 4.1 Upload `scr/Spline.cpp`. ▪

4.4 How to know if the result is correct

There is no reference result for you to compare the result. We will judge the correctness of the result qualitatively. With a small number of control points, the following quality of the curves are good indications:

- Bézier curve passes through the two endpoints, and is tangent to the first and last edge. The whole curve is smooth.
- B-spline doesn't pass through the two endpoints. The whole curve is smooth despite its piecewise definition. The junction between pieces has continuous tangent and continuous curvature (C^2 continuity). When the control points are symmetric (say symmetric left-right), then the B-spline is also symmetric.
- Subdivision should converge to the B-spline. The first level of subdivision should have the endpoints and every other points agreeing with the control edge midpoints.

These are good indication and should give you confidence if your curve agrees with these behavior. Any bug will easily destroy these smoothness and continuity.