CS130 - LAB - Bézier curves

Name: Jianeng Yang SID:

In this lab, we will render an approximation of a parametric curve known as the Bézier.

Consider the parametric equation of a segment between two control points P_0 and P_1

$$B(t) = (1-t)P_0 + tP_1 \tag{1}$$

For n control points, we can recursively apply Eq. (1) to consecutive control points until we are left with only $P(t)$. For three control points, $B(t) = (1-t)[(1-t)P_0+tP_1]+t[(1-t)P_1+tP_2]$.

1. Given n control points, what is the degree of the polynomial equation for the Bézier curve? In general, $B(t)$ for $n+1$ points is given by:

$$B(t) = \sum_{i=0}^{n} \binom{n}{i} t^i (1-t)^{n-i} P_i$$

■**The degree of the polynomial equation for the Bézier curve of n control points is n-1.**

Since, in the n+1 control points, for each basis function, the exponent is i + (n - i) = n. Thus, the degree of curves is n for n +1 control points. So, for Bézier curves with x controls points, the degree of curves is controls points - 1. Hence, the degree of the polynomial = x - 1.

2. Since we may need the factorial $n!$, combination $\binom{n}{i}$, and polynomial of $B(t)$ in this lab, complete the code to for these functions below.

```
float factorial(int n)
{
    // 0! = 1, 1! = 1, so return 1
    if (n <= 1) {
        return 1;
    }
    // recursive call
    // then the final will return
    // n! = n*(n-1)*(n-2)*...3*2*1
```

```
    return n * factorial(n-1);
}

float combination(int n, int i)
{
    // combination formula of nCi is n!/((n-i)!*i!)
    // The numerator and denominator are all factorial, then we can use the f
    int c = 0;
    c = factorial(n)/( factorial(n-i)*factorial(i) );
    return c;

}

float polynomial(int n, int i, float t)
{
    //float bc = 0.0; // for store //the bezier curve
    //int c= 0; // for store the combination value
    //c = combination(n,i);
    //for (int index = 0; index < n; index++) {
        // Bezier curve = nCi * t^i *(1-t)^(n-i)* P_i
    //    bc += c*pow(t, index)*pow((1-t), n-index) * points[index];
    //}
    //return bc;
    // Bezier curve = nCi * t^i *(1-t)^(n-i)* P_i
     return combination(n, i) * pow(t, i) * pow((1 - t), (n -i));

}
```

■

The code is an $O(n^2)$ algorithm for computing the $n+1$ coefficients

$$c_i = \binom{n}{i} t^i (1-t)^{n-i}.$$

Next, lets improve upon this. Let

$$r_i = \binom{n}{i} t^i \qquad s_i = (1-t)^{n-i} \qquad c_i = r_i s_i$$

3. The advantage of dividing c_i into two parts is that r_i can be easily computed left to right, since $r_0 = \binom{n}{0} t^0 = \frac{n!}{0! \cdot n!} \cdot 1 = \mathbf{1}$ and $r_i = (t(n-i+1))r_{i-1}$. Similarly, s_i can be easily computed right to left, since $s_n = (1-t)^{n-n} = \mathbf{1}$ and $s_i = (1-t)s_{i+1}$. Note that each of these expressions should be $O(1)$ and use only basic arithmetic (+,-,*,/).

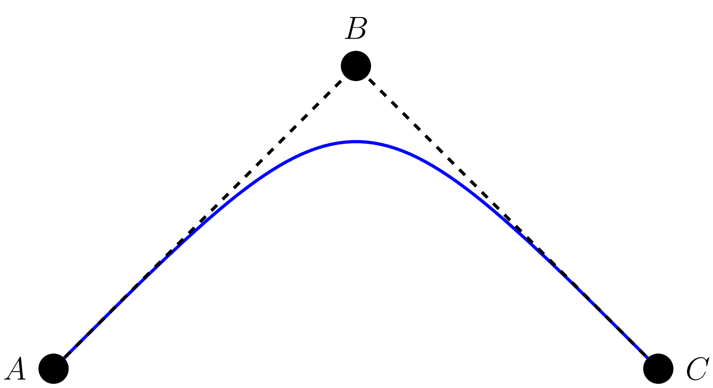
■$c_i = (t(n-i+1))r_{i-1}(1-t)s_{i+1}$

4. Next, write code for an $O(n)$ algorithm that computes all of the coefficients `c[0]`, ..., `c[n]` at once. Use only basic arithmetic (`+`,`-`,`*`,`/`).

```
void coefficients(float* c, int n, float t)
{
    float r[n];   // for the store the left part of c
    float s[n];   // for store the right part of c
    r[0] = 1.0;   // initial our r
    for (int i = 1; i < n; i++) {
        r[i] = (t * (n-i+1)) * r[i-1];
    }
    s[n-1] = 1.0; // initial our s
    for (int i = n-2; i <= 0; i--) {
        s[i] = (1-t) * s[i+1]

    }
    // calculate c
    for (int i = 0; i < n; i++) {
        c[i] = r[i] * s[i];
    }
}
```

■



We can construct the quadratic Bézier curve by assuming that it takes the general form $P(t) = (a_2t^2 + a_1t + a_0)A + (b_2t^2 + b_1t + b_0)B + (c_2t^2 + c_1t + c_0)C$. We can use the properties below to solve for the coefficients.

5. Assumption: $P(0) = A$. Use this to solve for a_0 = _____, b_0 = _____, and c_0 = _____.

■ $P(0) = (a_20^2 + a_10 + a_0)A + (b_20^2 + b_10 + b_0)B + (c_20^2 + c_10 + c_0)C = a_0A + b_0B + c_0C$
Since,$P(0) = A$, then $a_0A + b_0B + c_0C = (1)A + (0)B + (0)C = A$.
So, $a_0 = 1, b_0 = 0, c_0 = 0$.

6. Assumption: $P(1) = C$. Use this to solve for a_1 = _____, b_1 = _____, and c_1 = _____.

■ $P(1) = (a_2 1^2 + a_1 1 + a_0)A + (b_2 1^2 + b_1 1 + b_0)B + (c_2 1^2 + c_1 1 + c_0)C$
$= (a_2 + a_1 + a_0)A + (b_2 + b_1 + b_0)B + (c_2 + c_1 + c_0)C$
Since,$P(1) = C$, then $(a_2 + a_1 + a_0)A + (b_2 + b_1 + b_0)B + (c_2 + c_1 + c_0)C = (0)A + (0)B + (1)C = C$.
To solve a_1: $a_2 + a_1 + a_0 = 0$, then $a_1 = -a_2 - a_0$,from properties above we know $a_0 = 1$. Hence, $a_1 = -a_2 - 1$.
To solve b_1: $b_2 + b_1 + b_0 = 0$, then $b_1 = -b_2 - b_0$,from properties above we know $b_0 = 0$. Hence, $b_1 = -b_2 - 0 = -b_2$.
To solve c_1: $c_2 + c_1 + c_0 = 1$, then $c_1 = 1 - c_2 - c_0$,from properties above we know $c_0 = 0$. Hence, $c_1 = 1 - c_2 - 0 = 1 - c_2$.
So, $a_1 = -a_2 - 1, b_1 = -b_2, c_1 = 1 - c_2$.

7. Assumption: If $A = B = C$, then $P(t) = A$ for all t. Use this to solve for $b_2 =$ _____.

■ Given:$P(t) = (a_2t^2 + a_1t + a_0)A + (b_2t^2 + b_1t + b_0)B + (c_2t^2 + c_1t + c_0)C$
Since $A = B = C$, we can rewrite the equation:
$P(t) = (a_2t^2 + a_1t + a_0)A + (b_2t^2 + b_1t + b_0)A + (c_2t^2 + c_1t + c_0)A$
$P(t) = [(a_2t^2 + a_1t + a_0) + (b_2t^2 + b_1t + b_0) + (c_2t^2 + c_1t + c_0)]A$.
Plug in a_0, b_0, c_0, a_1, b_1, and c_1 from previous properties:
$P(t) = [(a_2t^2 + (-a_2 - 1)t + 1) + (b_2t^2 + (-b_2)t + 0) + (c_2t^2 + (1 - c_2)t + 0)]A$
$P(t) = [a_2t^2 + -a_2t + -1t + 1 + b_2t^2 - b_2t + c_2t^2 - c_2t + 1t]A$
$P(t) = [(a_2(t^2 - t) + b_2(t^2 - t) + c_2(t^2 - t) + (-1 + 1)t + 1]A$
$P(t) = [a_2(t^2 - t) + b_2(t^2 - t) + c_2(t^2 - t) + 1]A$
Since, P(t) = A:
$A = [a_2(t^2 - t) + b_2(t^2 - t) + c_2(t^2 - t) + 1]A$
$1 = [a_2(t^2 - t) + b_2(t^2 - t) + c_2(t^2 - t) + 1]1$
$0 = a_2(t^2 - t) + b_2(t^2 - t) + c_2(t^2 - t)$
$0 = a_2 + b_2 + c_2$
$b_2 = -a_2 - c_2$
So, answer is $b_2 = -a_2 - c_2$.

8. Assumption: $P'(0)$ depends on A and B, but it does not depend on C. Use this to solve for $c_2 =$ _____.

■ $P(t) = (a_2t^2 + a_1t + a_0)A + (b_2t^2 + b_1t + b_0)B + (c_2t^2 + c_1t + c_0)C$
$P'(t) = (2a_2t + a_1)A + (2b_2t + b_1)B + (2c_2t + c_1)C$
Not depend on C, so C = 0.
$P'(t) = (2a_2t + a_1)A + (2b_2t + b_1)B + (2c_2t + c_1)(0)$
$P'(t) = (2a_2t + a_1)A + (2b_2t + b_1)B$
$P'(0) = (2a_20 + a_1)A + (2b_20 + b_1)B$
$P'(0) = (a_1)A + (b_1)B$
$P'(0) = (-a_2 - 1)A + (-b_2)B$
$P'(0) = (-a_2 - 1)A + (-(-a_2 - c_2))B$
$P'(0) = (-a_2 - 1)A + (a_2 + c_2)B$
$0 = (-a_2 - 1)A + (a_2 + c_2)B$
$(a_2 + c_2)B = -(-a_2 - 1)A$

$(a_2 + c_2) = -(-a_2 - 1)A/B$
$c_2 = \frac{(a_2+1)A}{B} - a_2$
So, answer is $c_2 = \frac{(a_2+1)A}{B} - a_2$

9. Assumption: $P'(1)$ depends on B and C, but it does not depend on A. Use this to solve for $a_2 =$ _____.

■ $P'(t) = (2a_2t + a_1)A + (2b_2t + b_1)B + (2c_2t + c_1)C$
Not depend on A, so A = 0.
$P'(t) = (2a_2t + a_1)0 + (2b_2t + b_1)B + (2c_2t + c_1)C$
$P'(t) = (2b_2t + b_1)B + (2c_2t + c_1)C$
$P'(1) = (2b_2(1) + b_1)B + (2c_2(1) + c_1)C$
$P'(1) = (2b_2 + b_1)B + (2c_2 + c_1)C$
$P'(1) = (2b_2 + -b_2)B + (2c_2 + 1 - c_2))C$
$P'(1) = (b_2)B + (c_2 + 1)C$
$P'(1) = (-a_2 - c_2)B + (c_2 + 1)C$
$0 = (-a_2 - c_2)B + (c_2 + 1)C$
$-(-a_2 - c_2)B = (c_2 + 1)C$
$(a_2 + c_2)B = (c_2 + 1)C$
$(a_2 + c_2) = (c_2 + 1)C/B$
$a_2 = (c_2 + 1)C/B - c_2$
$a_2 = \frac{(c_2+1)C}{B} - c_2$
So, answer is $a_2 = \frac{(c_2+1)C}{B} - c_2$

10. Substituting in all of the coefficients and *factoring the resulting polynomials* produces $P(t) = ($___________________$)A + ($___________________$)B + ($___________________$)C$.

■ $P(t) = (a_2t^2 + a_1t + a_0)A + (b_2t^2 + b_1t + b_0)B + (c_2t^2 + c_1t + c_0)C$
$P(t) = (a_2t^2 + (-a_2 - 1)t + 1)A + (b_2t^2 + (-b_2)t + 0)B + (c_2t^2 + (1 - c_2)t + 0)C$
$P(t) = (a_2t^2 + (-a_2 - 1)t + 1)A + (b_2t^2 + (-b_2)t)B + (c_2t^2 + (1 - c_2)t)C$
$P(t) = (a_2t^2 + (-a_2 - 1)t + 1)A + ((-a_2 - c_2)t^2 + (-(-a_2 - c_2))t)B + (c_2t^2 + (1 - c_2)t)C$
$P(t) = (a_2t^2 - (a_2 + 1)t + 1)A + (-(a_2 + c_2)t^2 + (a_2 + c_2)t)B + (c_2t^2 + (1 - c_2)t)C$

11. One can show that $P'(0) = \alpha(B - A)$ and $P'(1) = \beta(B - C)$. Find α and β.

■ $P'(0) = (-a_2 - 1)A + (a_2 + c_2)B$
$P'(0) == \alpha(B - A) = (-a_2 - 1)A + (a_2 + c_2)B$
$P'(0) == \alpha(B - A) = (a_2 + 1)(B - A)$ if $c_2 = 1$
$\alpha = (a_2 - 1), if\ c_2 = 1$
$P'(1) = (-a_2 - c_2)B + (c_2 + 1)C$
$P'(1) = \beta(B - C) = (-a_2 - c_2)B + (c_2 + 1)C$
$P'(1) = \beta(B - C) = -(c_2 + 1)(B - C)$
$\beta = -(c_2 + 1), if\ a_2 = 1$

Part 2: Coding

Download the skeleton code and modify `main.cpp` as follows:

- Define a global vector to store the control points.
- Push back the mouse click coordinates into the vector in the function `GL_mouse`.
- Write the code for the `factorial`, `combination` and `binomial`.
- Draw line segments between points along the Bézier curve in `GL_render()`.
- You can use `GL_LINE_STRIP` to draw line segments between consecutive points.
- You can iterate t between 0 and 1 in steps of 0.01.

Optional: Rather than using the general equation for the Bézier curve to write your program, you can write a program where you recursively apply Eq. (1) to consecutive points to get $B(t)$. Alternatively, you can use the more efficient algorithm `coefficients`.