CS130 - LAB - Bézier curves

Name: ______________________________ 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:

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

- degree: n

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

```
float factorial(int n)
{
    if (n==0) return 1;
    return n * factorial(n-1);
}

float combination(int n, int k)
{
  return factorial(n)/(factorial(k)*factorial(n-k));

}

float polynomial(int n, int k, float t)
{
  return combination(n,k)* pow(t,k)* pow(1-t, n-k);
```

}

■

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 = 1 and $r_i = (\frac{n-i+1}{i} t)r_{i-1}$. Similarly, s_i can be easily computed right to left, since s_n = 1 and $s_i = (1-t)s_{i+1}$. Note that each of these expressions should be $O(1)$ and use only basic arithmetic (+,-,*,/).

■

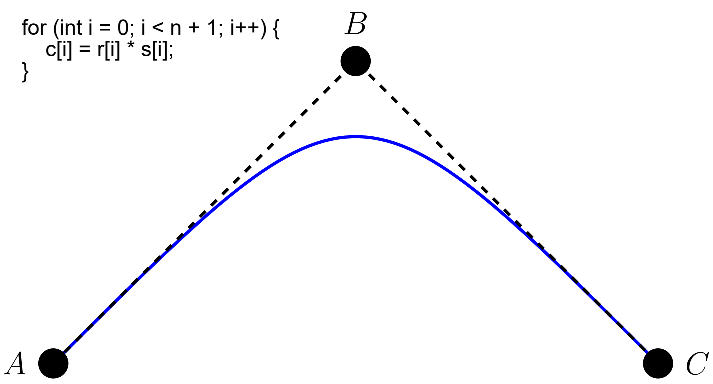
4. Next, write code for an $O(n)$ algorithm that computes all of the coefficients `c[0]`, ..., `c[n]` at once. Use only basic arithmetic (+,-,*,/).

```
void coefficients(float* c, int n, float t)
{   float* r = new float[n + 1], s = new float[n + 1];
    r[0] = 1;
    s[n] = 1;

    for (int i = 1; i < n + 1; i++) {
        r[i] = (t * ((float)(n - i + 1) / i)) * r[i - 1];
}   }

■   for (int i = n-1; i >= 0; i--) {
        s[i] = (1 - t) * s[i + 1];
    }

    for (int i = 0; i < n + 1; i++) {
        c[i] = r[i] * s[i];
    }
```



We can construct the quadratic Bézier curve by assuming that it takes the general form $P(t) = (a_2 t^2 + a_1 t + a_0)A + (b_2 t^2 + b_1 t + b_0)B + (c_2 t^2 + c_1 t + c_0)C$. We can use the properties below to solve for the coefficients.

5. Assumption: $P(0) = A$. Use this to solve for a_0 = 1, b_0 = 0, and c_0 = 0.

■

6. Assumption: $P(1) = C$. Use this to solve for a_1 = $-a_2-1$, b_1 = $-b_2$, and c_1 = $1-c_2$.

$a_2 + a_1 + 1 = 0$ $\quad$ $b_2 + b_1 + 0 = 0$ $\quad$ $c_2 + c_1 + 0 = 1$

$a_1 = -a_2 - 1$ $\quad$ $b_1 = -b_2$ $\quad$ $c_1 = 1 - c_2$

■

7. Assumption: If $A = B = C$, then $P(t) = A$ for all t. Use this to solve for b_2 = $-c_2 - a_2$.

■

8. Assumption: $P'(0)$ depends on A and B, but it does not depend on C. Use this to solve for c_2 = 1.

■

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

■

10. Substituting in all of the coefficients and *factoring the resulting polynomials* produces $P(t) = ((t-1)^2)A + (-2(t^2-t))B + (t^2)C$.

■

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

■ $\alpha = 2$ $\quad$ $\beta = -2$

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`.