CSE 333/533
Assignment 01:Bezier Curves

Abhishek Pratap Singh

2018211 - abhishek18211@iiitd.ac.in

1 TASK

Implement the drawing of piecewise Cubic Bezier curves in OpenGL based on vertex points at mouse clicks that are taken progressively.

2 IMPLEMENTATION

Given is a template code to capture mouse coordinates on the OpenGL application canvas. The code then draws a piecewise linear curve connecting the points.
Seeing through the code, we see that all points coordinates are stored in a vector as 3 float values (**x,y,z=0**) This point is then passed to the *calculatePiecewiseLinearBezier()* method that computes the linear bezier between points. Finally OpenGL renders the points and lines as primitives.

To implement piecewise Bezier Curves, the strategy I used is as follows.

1. Store the number of points clicked by the user in a variable (**int** *points*). This will help in implementing the cubic bezier curve which takes 4 points as input. The user gives 4 mouse clicks to generate one bezier curve.

2. A cubic bezier curve has 4 control points (P0,P1,P2,P3). These points are used to compute the curve using the formula:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \ , 0 \leq t \leq 1.$$

Figure 1: Explicit Formula for Cubic Bezier Curve

3. The above formula is implemented in the *calculatePiecewiseCubicBezier()* method and the calculated curve points are stored in as 3 float values (**x,y,z=0**) in a vector. The implementation for this method is similar to that of linear bezier curve We get the following result:

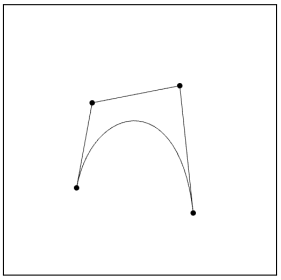

Figure 2: Simple cubic bezier curve.

4. However if we add more points, we get piecewise Bezier Curve that is not C1 continuous. To maintain C1 continuity, I first keep a check on the **points** variable. If the number

of points is equal to 4, then I draw the first bezier curve.

5. I then calculate the next equidistant collinear point P5 using the midpoint formula. This now ensures that the next piecewise bezier curve will be C1 continuous to the previous curve, since the points are equidistant and collinear.

$$(x_m, y_m) = (\frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2})$$

(x_m, y_m) = coordinates of the midpoint
(x_1, y_1) = coordinates of the first point
(x_2, y_2) = coordinates of the second point

Figure 3: Midpoint Formula

The following output is now obtained.

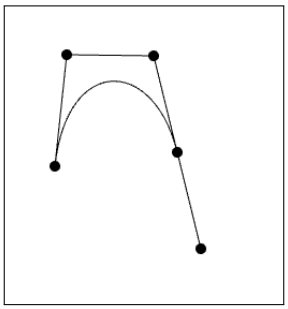

Figure 4: Continuous Bezier Curve

6. Since a new point is now automatically generated, I account for this change and modify by drawing the curve if the number of points(***int*** *points*) equals the below constraints.

```
if((points % 3 == 1 && points > 4) || points == 4)
{
    // Draw Curve
}
```

7. Hence with the constraint, when the user now clicks 4 points, the first bezier curve is drawn and the next control point is plotted automatically. Now, the user clicks 3 new points and the next bezier curve is drawn.

The following output is obtained.

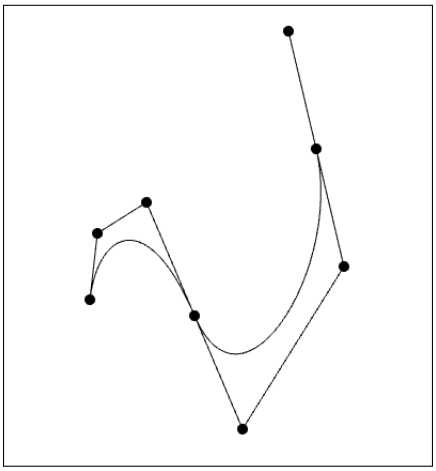

Figure 5: Continuous Bezier Curve

Hence we have created a piecewise cubic Bezier curve that has C1 Continuity.

3 C1 CONTINUOUS PROOF

The above bezier curve was drawn such that two constraints were imposed on the points.

1. The points **(Pn-1, Pn, Pn+1)** where **Pn** is the endpoint for the first curve; are collinear to each other. This is done by making then tangent to the Pn in the above implementation.

2. The points **(Pn-1, Pn, Pn+1)** are equidistant from each other. This is done by using the midpoint formula to draw point **Pn+1** using **Pn-1** and **Pn**.

Using above constraints we can now prove that the obtained curve is C1 continuous.

To prove: C0 continuity, G1 continuity and first derivatives at endpoints are equal.

Let **(P0,P1,P2,P3)** be the control points for the first curve **A(u)** and **(Q0,Q1,Q2,Q3)** be the control points for the second curve **B(u)**.

We know that the two curves are piecewise hence P3 = Q0. Hence, we can say that currently the curves are C0 continuous since the last (P3) and first(Q0) control points are the same.

Now, we implemented the first constraint of collinearity. Hence, the vectors P3-P2 and Q1-Q0 are tangent vectors in opposite directions. **This property satisfies the G1 continuity.**

Suppose we take the first derivatives of the curves to obtain the tangent vectors. To achieve C1 continuity, the derivatives must be equal.

```
A'(1) = a(P3 - P2) ,        a is the length of the vector P3-P2.
B'(0) = b(Q1 - Q0) ,        b is the length of the vector Q1-Q0.

Hence, we need to prove
A'(1) = B'(0)

Using the second constraint,
we set the length of the tangent vectors equal.

=> a = b
=> (P3 - P2) = (Q1 - Q0)
=> A'(1) = B'(0)

Hence proved.
```

Hence, we have the first derivatives at the endpoints equal.
Hence, the curves are C1 continuous.

4 BONUS IMPLEMENTATION

The bonus section requires us to implement the modification of control points using mouse pick and drag. This behaviour should result in a modified piecewise Bezier displayed on screen.

For this, I used the following strategy.

1. Whenever a user creates a point, I store the original mouse coordinates in a vector (***vector*** *placePoints*).

2. Since the vector stores the coordinates of a single pixel, it is really hard to re select the same point. To solve this issue, I set a threshold variable that specifies the range around the clicked point. This range would allow more flexible selection of points.

3. When a user tries to re select a previously created point, I iterate through the vector of stored coordinates and compare the difference of the stored and new coordinates. If this difference is within the threshold, the point is selected. Else a new point is created.

PseudoCode:

```
int threshold = 10
bool pointSelected = false
for stored in placePoints{
    new = io.mousepoint

    if((stored - new) <= threshold){
        pointSelected = true
        // Point is selected.
    }
    else{
        // Point not selected. Create new point.
    }
}
```

4. If a new point is selected, a boolean variable (***bool*** *pointSelected*) is set to true. This is to ensure that the next click removes the old point and a new point is drawn.

PseudoCode:

```
if(pointSelected){
    \\ Remove old point.
    \\ Plot new point.
    \\ Calculate curve.
}

else{
    if(new point within threshold){
        pointSelected=true
    }
    else{
        \\ Draw new point.
    }
}
```

\tab 5. Hence points can be changed and the curve changes accordingly.