1634304 郑国林

Basic:

1. 用户能通过左键点击添加 Bezier 曲线的控制点，右键点击则对当前添加的最后一个控制点进行消除

2. 工具根据鼠标绘制的控制点实时更新 Bezier 曲线。

a. Bezier 曲线原理

本质上是由调和函数根据控制点插值生成.曲线的参数方程如下：其中 B 是基函数，P 是控制点

$$Q(t)=\sum_{i=0}^{n}P_iB_{i,n}(t)\,,\quad t\in[0,1]$$

伯恩斯坦基函数计算如下：

$$B_{i,n}(t)=\frac{n!}{i!(n-i)!}t^i(1-t)^{n-i}\,,\quad i=0,\ 1\ldots n$$

我们只需要根据控制点的坐标，通过上述的公式计算出点集，然后绘制出来就可以了。

b. 具体 openGL 中代码实现：

1. 通过回调函数监听鼠标点击事件，然后获取点击屏幕时控制点的坐标，并进行坐标转换，从屏幕坐标转换到窗口坐标。并且把控制点加入到数组中存储起来。如果是右键点击就从数组中弹出最后一个控制点。根据控制点集合我们就可以动态的增加和减少控制点来绘制贝塞尔曲线。

   鼠标点击事件回调函数如下：（其中 dynamicT 是在实现 bonus 中用到的）

```
1. void mouse_button_callback(GLFWwindow* window, int button, int action, int m
   ods) {
2.     if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
3.         controlPoints.pop_back();
```

```
4.
5.          dynamicT = 0;
6.      }
7.      else if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {
8.          double tempX = 0, tempY = 0;
9.          glfwGetCursorPos(window, &tempX, &tempY);
10.
11.         tempX = (-1) + (tempX / WIDTH) * 2;
12.         tempY = (-1)*((-1) + (tempY / HEIGHT) * 2);
13.         Point p;
14.         p.x = tempX;
15.         p.y = tempY;
16.         controlPoints.push_back(p);
17.
18.         dynamicT = 0;
19.     }
20. }
```

2. 根据控制点坐标以及参数方程的公式来计算出曲线的点集。其中由于 t 的范围是 0-1，我们可以自己设置 t 的步长。值得注意的是，在每个取值下都要用参数方程来计算得一个点，最后的点集就是构成贝塞尔曲线的点集。

   以下是计算贝塞尔曲线，伯恩斯坦基函数，以及阶乘的函数代码

```
1.  vector<Point> Bezier(vector<Point>& points)
2.  {
3.      vector<Point> resultPoints;
4.
5.      int n = points.size() - 1;
6.      for (double t = 0; t <= 1; t += 0.00005)
7.      {
8.          Point targetPoint;
9.          targetPoint.x = 0;
10.         targetPoint.y = 0;
11.
12.         for (int i = 0; i <= n; i++)
13.         {
14.             double B = Bernstein(i, n, t);
15.             Point curPoint = points[i];
16.
```

```
            targetPoint.x += curPoint.x * B;
            targetPoint.y += curPoint.y * B;
        }

        resultPoints.push_back(targetPoint);
    }

    return resultPoints;

}

double Bernstein(int i, int n, double t)
{
    double B = factorial(n) / (factorial(i) * factorial(n - i));
    B = B * pow(t, i) * pow((1 - t), (n - i));
    return B;
}

int factorial(int num)
{
    int ans = 1;
    for (int i = 1; i <= num; i++)
        ans *= i;
    return ans;
}
```

3. 计算出贝塞尔曲线的点集之后，我们就可以通过 openGL 来将这些点绘制出来得到贝塞尔曲线，并且可以通过控制点绘制出控制多边形。（shader 的编写在这里就不赘述了，只要最简单版本的 shader 就行）

   根据曲线点集绘制贝塞尔曲线的代码：

```
void drawCurve(unsigned int shader)
{
    unsigned int VBO, VAO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);

    vector<float> points;
    vector<Point> bezierPoints = Bezier(controlPoints);
    glm::vec4 color(1.0f, 0.5f, 0.2f, 1);
```

```
10.
11.     glUniform4fv(glGetUniformLocation(shader, "drawColor"), 1, &color[0]);
12.     for (int i = 0; i < bezierPoints.size(); i++)
13.     {
14.         Point P = bezierPoints[i];
15.         points.push_back(P.x);
16.         points.push_back(P.y);
17.     }
18.     int point_num = points.size() / 2;
19.     float* vertices = convert2vertice(points);
20.
21.     glBindVertexArray(VAO);
22.
23.     glBindBuffer(GL_ARRAY_BUFFER, VBO);
24.
25.     glBufferData(GL_ARRAY_BUFFER, point_num * 3 * sizeof(float), vertices, G
    L_STREAM_DRAW);
26.
27.     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void
    *)0);
28.     glEnableVertexAttribArray(0);
29.
30.     //glDrawArrays(GL_LINE_STRIP, 0, 2);
31.     glDrawArrays(GL_POINTS, 0, point_num);
32.     glBindVertexArray(0); // no need to unbind it every time
33. }
```

根据控制点集绘制控制多边形的时候我们只需要将控制点前后组合绘制直线就可以了。

```
1.  for (int i = 0; i < controlPoints.size() - 1; i++)
2.  {
3.      drawLine(controlPoints[i], controlPoints[i + 1], shaderProgram);
4.  }
```

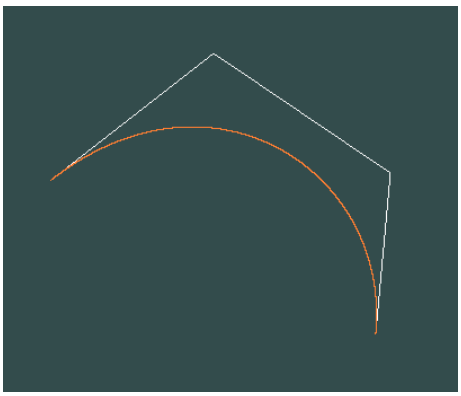
绘制直线代码

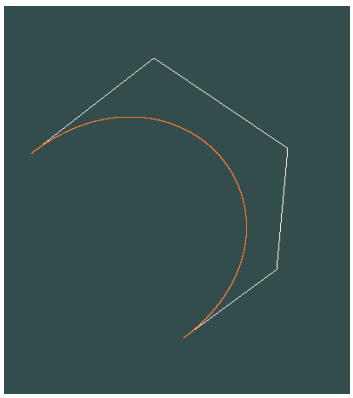
```
1.  void drawLine(Point fir, Point sec, unsigned int shader)
2.  {
3.      unsigned int VBO, VAO;
```

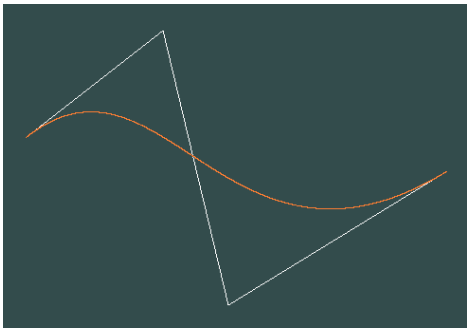
```
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glm::vec4 color(1, 1, 1, 1);
    glUniform4fv(glGetUniformLocation(shader, "drawColor"), 1, &color[0]);

    float vertices[] = { fir.x, fir.y, 0.0, sec.x, sec.y, 0.0 };
    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);

    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STREAM_DRAW
);

    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void
*)0);
    glEnableVertexAttribArray(0);

    glDrawArrays(GL_LINE_STRIP, 0, 2);
    glBindVertexArray(0); // no need to unbind it every time
}
```

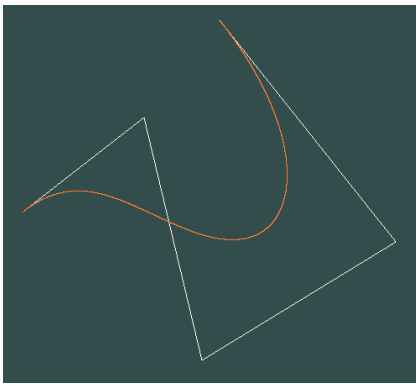
c. **贝塞尔曲线截图**

白色是控制多边形，橙色是贝塞尔曲线







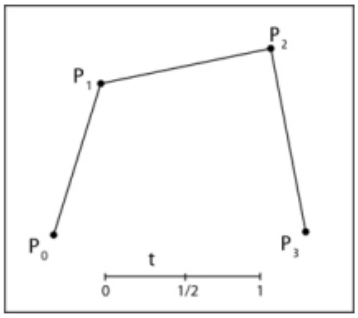


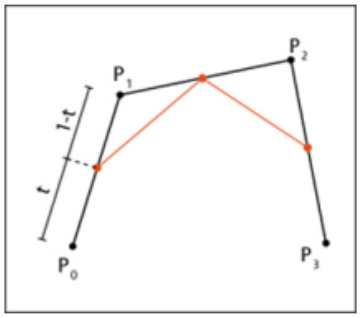
Bonus:

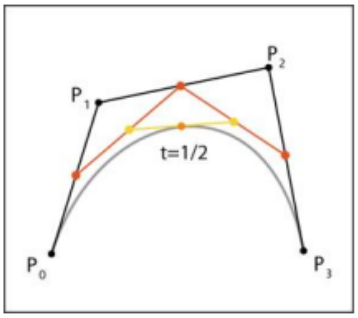
1. 可以动态地呈现 Bezier 曲线的生成过程。

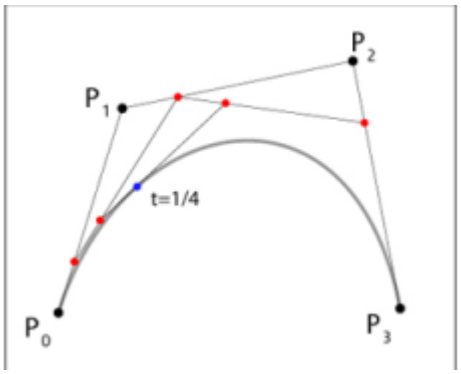
   a. 曲线生成原理

我们需要更加了解贝塞尔曲线的生成过程，对于其参数方程，针对每个 t 值，对控制多边形的每一条边使用 t 值进行插值得到一个新的点，将得到的这些相邻的新的点连接成线，就得到了一个新的“控制多边形”。重复上述步骤 n 次最后只得到一个新的点，该点就是这个 t 值对应的曲线上的点。









b. 具体代码实现

为了达到一个动态的效果，我们不能在一个渲染循环中把所有 t 的生成过程绘制出来，因此在全局维持一个 dynamicT 值，每绘制一次这个过程就把 dynamicT 加上步长，直到达到 1 为止。并且当鼠标点击增加或减少控制点的时候，将 dynamicT 重置为 0。

然后就是将控制多边形的边进行迭代线性插值的过程，将每个迭代得到的新的“控制多边形”绘制出来。

代码：

```
1.  void drawDynamicProcedure(unsigned int shader)
```

```
{
    double t = dynamicT;
    vector<Point> prePoints(controlPoints);
    while (prePoints.size() != 0)
    {
        vector<Point> temp(prePoints);
        prePoints.clear();

        for (int j = 0; j < temp.size() - 1; j++)
        {
            Point p;
            Point fir = temp[j];
            Point sec = temp[j + 1];
            p.x = t * sec.x + (1 - t)*fir.x;
            p.y = t * sec.y + (1 - t)*fir.y;
            prePoints.push_back(p);
        }
        //draw line
        for (int j = 0; prePoints.size() >= 1 &&  j < prePoints.size() - 1; j++)
        {
            drawLine(prePoints[j], prePoints[j + 1], shader);
        }
    }
    if (dynamicT + 0.005 > 1)
        dynamicT = 1;
    else
        dynamicT += 0.005;
}
```

c． 实现截图