# Computer Graphics Homework 8

## Basic

### 用户增减Bezier曲线控制点

实现这一功能我分了三个部分：

1. 鼠标左键在屏幕上生成点，鼠标右键屏幕上点消失
2. Beizer曲线的计算
3. 使用vector数组存放控制点，并在处理鼠标操作的函数中更新控制点vector

#### 鼠标点击操作控制点

查阅资料，glfw中处理鼠标操作的函数如下：

```
// 在main中根据mouse_button_callback的定义监听鼠标点击事件
glfwSetMouseButtonCallback(window, mouse_button_callback);
// 定义回调的操作
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
```

在`mouse_button_callback`调用封装到`Homework8`类中的，我将它封装到`Homework8.mouseButtonCallBack`：

```
void Homework8::mouseButtonCallBack(int button, int action) {
    // get the position being clicked, from viewport coordinate to window coordinate
    lockX = ((float)currentX - windowWidth / 2) / (windowWidth / 2);
    lockY = (-1)*((float)currentY - windowHeight / 2) / (windowHeight / 2);
    // add
    if (button == GLFW_MOUSE_BUTTON_LEFT && action == GLFW_PRESS) {
        addControlPoints();
    }
    // delete
    if (button == GLFW_MOUSE_BUTTON_RIGHT && action == GLFW_PRESS) {
        deleteControlPoints();
    }
    n = controlPointsVertices.size() / 6 - 1;
    calculateBeizer();
}
```

根据API，这里使用`button == GLFW_MOUSE_BUTTON_LEFT`判断是鼠标左键的操作（右键同理），`action == GLFW_PRESS`判断为按下的操作。调用对应的函数实现添加或删除控制点。此外，每次添加删除后还需要更新Beizer曲线上的点集（后文会说`calculateBeizer()`）.

#### Bezier曲线的计算

根据公式直接套用即可，这里将曲线的计算分为了Bezier曲线、Bernstein基函数和组合数三个模块分别实现：

**Bezier曲线整体**

```
void Homework8::calculateBeizer() {
    beizerCurveVertices.clear();
    if (n == 0)
        return;
    for (double t = 0; t <= 1; t += deltaT) {
        float x = 0, y = 0;
        for (int i = 0; i <= n; i++) {
            x += controlPointsVertices[6 * i] * bernstain(t, i);
            y += controlPointsVertices[6 * i + 1] * bernstain(t, i);
        }
        vector<float> tmp = {
            x, y, 0, 1, 1, 1
        };
        beizerCurveVertices.insert(beizerCurveVertices.end(), tmp.begin(), tmp.end());
    }
}
```

**Bernstein基函数**

```
double Homework8::bernstain(double t, int i) {
    double ans = combineNumber(i);
    ans *= pow(t, i) * pow(1 - t, n - i);
    return ans;
}
```

**组合数**

```
int Homework8::combineNumber(int i) {
    int ans = 1;
    for (int j = 0; j < i; j++) {
        ans *= (n - j);
    }
    for (int j = 0; j < i; j++) {
        ans /= (i - j);
    }
    return ans;
}
```

## 使用vector数组存放控制点，并在处理鼠标操作的函数中更新控制点vector

控制点的增删主要是对于`controlPointsVertices`这个vector的添加/删除元素的操作，代码如下：

**增加控制点**

```
void Homework8::addControlPoints() {
    vector<float> tmp = {
        (float)lockX, (float)lockY, 0, 1, 1, 1
    };
    controlPointsVertices.insert(controlPointsVertices.end(), tmp.begin(), tmp.end());
}
```

**删除控制点**

这里根据鼠标点击的位置以及设定的控制顶点半径大小`radius`判断鼠标右键点击的位置是否为控制点，并遍历控制点数组删除对应的点。

```
void Homework8::deleteControlPoints() {
    for (int i = 0; i < controlPointsVertices.size(); ) {
        if (abs(controlPointsVertices[i] - lockX) < radius / windowWidth * 2 &&
abs(controlPointsVertices[i + 1] - lockY) < radius / windowHeight * 2) {
            for (int j = 0; j < 6; j++) {
                controlPointsVertices.erase(controlPointsVertices.begin() + i);
            }
        } else {
            i += 6;
        }
    }
}
```

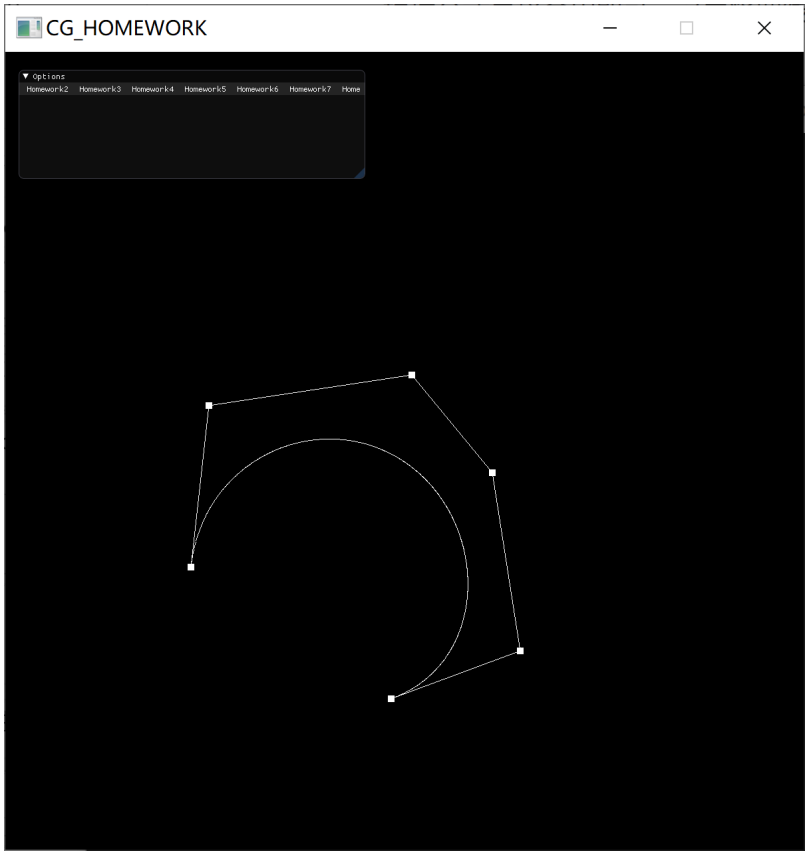
在显示控制中，将`controlPointsVertices`传入，就可以绘制其中的点了。

```
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * pointsVec.size(), &pointsVec[0],
GL_STATIC_DRAW);
```

## 实时更新Bezier曲线

上个部分将模块的基础功能基本实现，实时更新的实现是在`Homework8::mouseButtonCallBack`函数的最后调用了`calculateBeizer`实现的，即每次控制顶点数组产生变化后，曲线需要重新绘制，会在这个函数中将原来的曲线上的点数组清空，重新计算。（我的程序中用1000个点连接成曲线）。

效果如下：




# Bonus

## 动态呈现生成Bezier曲线的过程

动态生成曲线我主要分了两个步骤实现：

1. 静态绘制出辅助线（也就是除了连接控制顶点的直线之外的有层次的直线）
2. 加入对于参数T的更新

### 加入辅助线

根据Bezier曲线的原理，每条辅助线是将上一层的控制多边形两个相邻线段上分别取t位置进行连接形成的。

因而控制多边形的下一层辅助线为遍历控制顶点数组，并计算t位置的点，加入`lines` vector并将相邻的顶点两两连成线段：

```
void Homework8::calculateAssLines() {
    lines.clear();
    if (controlPointsVertices.size() > 6) {
        // 处理第一层
        for (int i = 0; i < controlPointsVertices.size() / 6 - 1; i++) {
            float x = (1 - currentT) * controlPointsVertices[i * 6] + currentT *
controlPointsVertices[(i + 1) * 6];
            float y = (1 - currentT) * controlPointsVertices[i * 6 + 1] + currentT *
controlPointsVertices[(i + 1) * 6 + 1];
            vector<float> tmp = {
                x,y,0,1,1,1
            };
```

```
            lines.insert(lines.end(), tmp.begin(), tmp.end());
        }
        recursive(lines.size() / 6);
    }
}
```

此后的步骤就可以如此递归进行，并将它们都加入到`lines` vector的末尾：

```
void Homework8::recursive(int count) {
    int tmpSize = lines.size();
    if (count < 2) {
        return;
    }
    for (int i = tmpSize / 6 - count; i < tmpSize / 6 - 1; i++) {
        float x = (1 - currentT) * lines[i * 6] + currentT * lines[(i + 1) * 6];
        float y = (1 - currentT) * lines[i * 6 + 1] + currentT * lines[(i + 1) * 6 +
1];
        vector<float> tmp = {
            x,y,0,1,1,1
        };
        lines.insert(lines.end(), tmp.begin(), tmp.end());
    }
    recursive(count - 1);
}
```

这里选择上一层的数目`count` < 2作为递归截止条件的原因是如果上一层有2个控制点，即一条线段，仍然需要计算这条线段的t处，绘制最后一层直线和Bezier曲线的交点。

---

对于辅助直线的绘制，如果直接将`lines`数组传入，则上一层的最后一个线段的末端点会和下一层的第一个线段的起点相连。因此绘制时用一个for循环控制：

```
void Homework8::drawAssLines() {
    int count = 0;
    for (int i = controlPointsVertices.size() / 6 - 1; i > 1; i--) {
        vector<float> tmp(lines.begin() + count * 6, lines.begin() + count*6 + i*6);
        count += i;
        drawLine(tmp);
    }
    drawPoints(lines);
}
```

这里`drawPoints(lines)`可以将所有辅助线的端点，以及最后一层辅助线和Bezier曲线相交的点直接绘制。

### 加入对于参数T的更新

这个步骤需要引入一个`currentT`变量作为参数，随着while循环，每次将`currentT`增加一点点。

```
void Homework8::displayController() {
    // deal with animation
    currentT += speed;
    calculateAssLines();
    drawAssLines();
    if (currentT > 1) {
        currentT = 0;
    }
    // basic
    displayControlPoints();
    drawLine(controlPointsVertices);
    displayBeizerCurve();
}
```

由于需要实现动画过程，因此每一帧都要更新辅助线，`calculateAssLines()`在while循环中被调用。

效果如下：