Basic:

先实现 mouse 的监听：

```
glfwSetMouseButtonCallback(window, mouse_button_callback);
```

```
void mouse_callback(GLFWwindow* window, double xpos, double ypos)
{
    x_pos = xpos;
    y_pos = ypos;
}
```

```
void mouse_button_callback(GLFWwindow* window, int button, int action, int mods)
{
    if (action == GLFW_PRESS) switch (button)
    {
    case GLFW_MOUSE_BUTTON_LEFT:
        points.push_back(Point((x_pos - SCR_WIDTH / 2) / (SCR_WIDTH / 2),
            (SCR_HEIGHT / 2 - y_pos) / (SCR_HEIGHT / 2)));
        change = true;
        break;
    case GLFW_MOUSE_BUTTON_MIDDLE:
        break;
    case GLFW_MOUSE_BUTTON_RIGHT:
        points.pop_back();
        change = true;
        break;
    default:
        return;
    }
    return;
}
```

当点击鼠标左键时，将当前鼠标的 x，y 坐标对屏幕大小进行处理，将其化为[-1,1]区间范围内，然后存储到一个类 Point 中，将 Point 放到一个 vector<Point> points 里，points 这个容器用以存储控制点。而点击鼠标右键时，将最后一个 point 弹出。

类 Point 的实现：

```
class Point {
public:
    float x;
    float y;

    Point() {
        x = 0.0f;
        y = 0.0f;
    }
    Point(float x_, float y_) {
        x = x_;
        y = y_;
    }
```

```
Point operator*(const float scalar) { ... }
void operator*=(const float scalar) { ... }
Point operator+(const Point& point) { ... }
Point operator-(const Point& point) { ... }
void operator+=(const Point& point) { ... }
```

Point 存储点的坐标，并且重载了一些会用到的运算符。

Bezier Curve 的实现：

又控制点生成的曲线公式如下：

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

其中，B 的公式为（左部份为组合，即 n 中取 i）：

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

代码实现：

```
float B(float i, float n, float t) {
    return C(n, i)* pow(t, i)* pow(1 - t, n - i);
}
float C(int n, int i) {
    // select i from n
    int border = n - i;
    if (n - i < i) {
        border = i;
        i = n - border;
    }
    border += 1;
    int numerator = 1, denominator = 1;
    for (int j = n; j >= border; j--) {
        numerator *= j;
    }
    for (i; i >= 1; i--) {
        denominator *= i;
    }
    return numerator / denominator;
}
```

其中，C 函数即公式左半部分，排列组合 n 中取 i。而 B 函数需要参数 t。

当要生成 Bezier 曲线时，需要 t 以一定步长从 0 移动到 1，从而不断生成多个插值点（步长越小，插值点越多），用一个函数 getCurve() 来获得描绘曲线所需的插值点：

```
vector<Point> getCurve() {
    vector<Point> curve;
    for (float t = 0.0f; t <= 1.0f; t += 0.0001f) {
        curve.push_back(getBezierPoints(t));
    }
    return curve;
}
```

对于每一个 t，都调用一个函数来获得插值点 Q(t)，根据上面的公式实现：

```
Point getBezierPoints(float t) {
    int n = points.size() - 1;
    Point point(0.0f, 0.0f);
    for (int i = 0; i <= n; i++) {
        point += points[i] * B(i, n, t);
    }
    return point;
}
```

最后将获得的点放入 VAO 中进行渲染即可，当控制点数量发生变化（鼠标 click）时将 change 变量置为 true：

```
case GLFW_MOUSE_BUTTON_LEFT:
    points.push_back(Point((x
        (SCR_HEIGHT / 2 - y_p
    change = true;
```

```
case GLFW_MOUSE_BUTTON_RIGHT:
    points.pop_back();
    change = true;
    break;
```

当 change 为 true 时，就重新调用 getCurve 获得新的曲线，将其转化为数组并进行渲染：

```
if (change) {
    if (curveVertices != NULL) {
        delete[] curveVertices;
    }
    vector<Point> curve = getCurve();
    curveSize = curve.size() * 2;
    curveVertices = new float[curveSize];
    index = 0;
    for (auto it = curve.begin(); it != curve.end(); it++) {
        curveVertices[index++] = it->x;
        curveVertices[index++] = it->y;
    }
    change = false;
}
shader.setBool("colorful", true);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * curveSize, curveVer
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float)
glEnableVertexAttribArray(0);
glBindVertexArray(curveVAO);
glDrawArrays(GL_POINTS, 0, curveSize / 2);
glBindVertexArray(0);
```

以上即完成了 Bezier 曲线的实时更新。

Bonus:

要求动态呈现曲线的生成过程，就需要用到时间相关的变量，用一个变量 time 来控制 t 从 0 到 1 的速度：

```
currentFrame = glfwGetTime();
deltaTime = currentFrame - lastFrame;
lastFrame = currentFrame;
time += deltaTime / 10;
if (time > 1.0f) {
    time = 0.0f;
}
```

用一个 getLines(float t) 函数来获得每个时间点 t 的需要渲染的辅助线段：

```
float* getLines(float t) {
    vector<Point> lines(points);
    int size = 0;
    for (int i = points.size(); i > 1; i--) {
        size += (i - 1) * 2;
    }
    float* vertices = new float[size * 2];
    int index = 0;
    while (lines.size() > 1) {
        for (int i = 0; i < lines.size() - 1; i++) {
            vertices[index++] = lines[i].x;
            vertices[index++] = lines[i].y;
            vertices[index++] = lines[i + 1].x;
            vertices[index++] = lines[i + 1].y;
            lines[i] += (lines[i + 1] - lines[i]) * t;
        }
        lines.pop_back();
    }
    return vertices;
}
```

原理很简单，就是每一步都将控制点更新，新的点为：

P0' = (P1 – P0) * t + P0

因为渲染线段需要起点和终点两个点，因此每次更新前都将 P[i] 和 P[i+1]放到数组中，然后才对 P[i]进行更新；最后将 vector 中的最后一个点去掉。

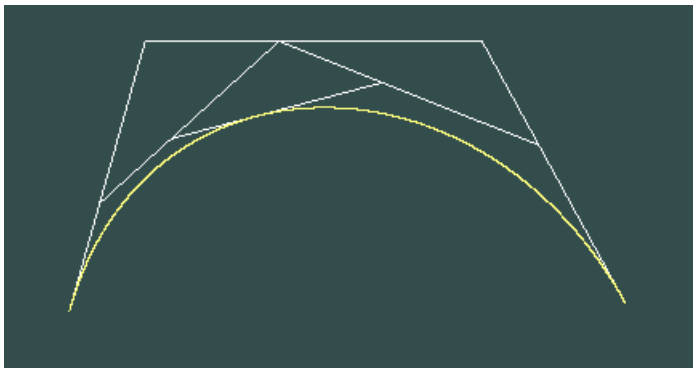
当 vector 中只剩下一个点时，更新结束，返回得到的数组。

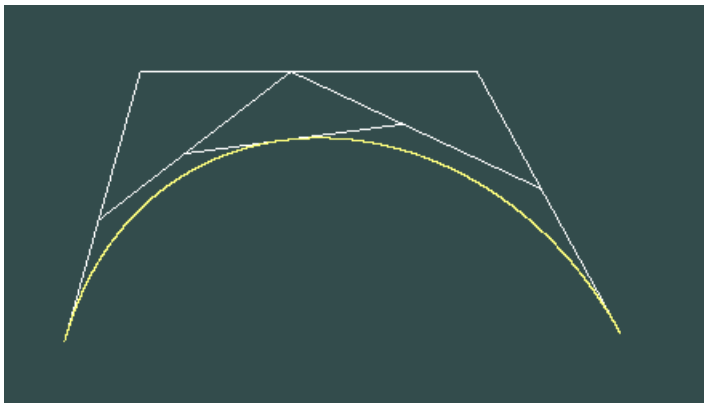
进行渲染：

```
// render points and lines
if (vertices != NULL) {
    delete[] vertices;
}
size = 0;
for (int i = points.size(); i > 1; i--) {
    size += (i - 1) * 2;
}
size *= 2;
vertices = getLines(time);
shader.setBool("colorful", false);
glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(float) * size, vertices, GL_STATIC_DRAW);
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)0);
glEnableVertexAttribArray(0);
glBindVertexArray(VAO);
glDrawArrays(GL_LINES, 0, size / 2);
```

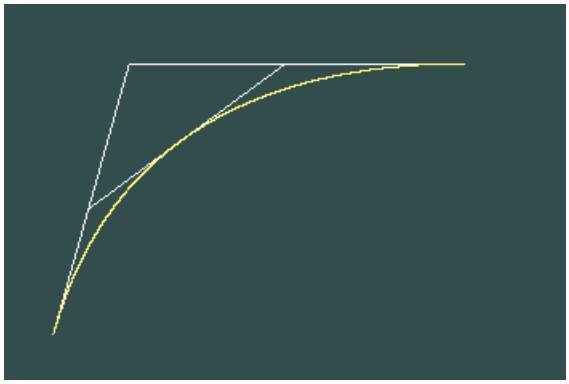
结果展示:

鼠标左键点四个控制点，黄色线即为 Bezier Curve，会展示曲线的生成过程:





点右键，去掉最后一个点，更新 Bezier 曲线:



更多展示请看 mp4。