计算机图形学实验报告

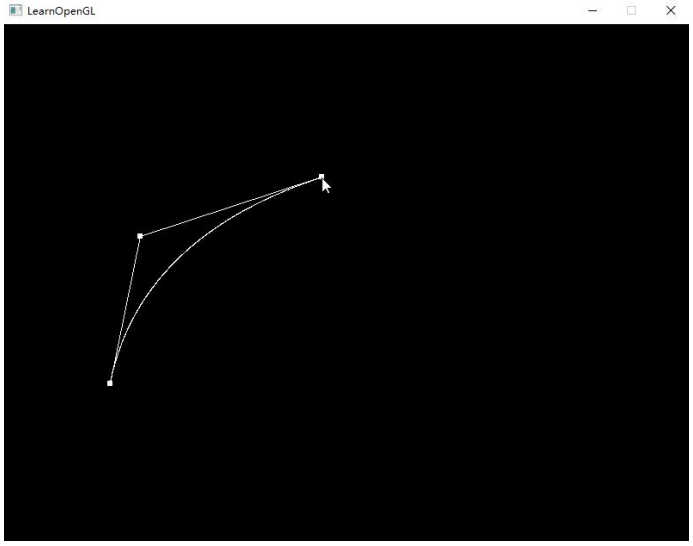
第一部分：实现思路和代码分析

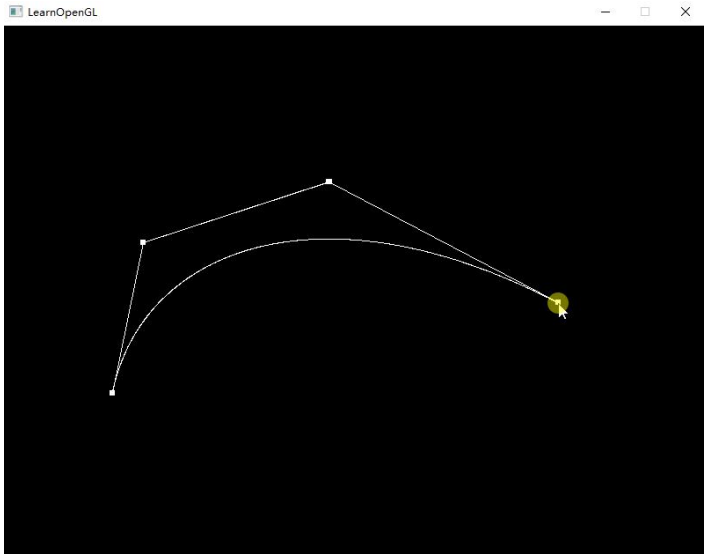
代码截图	代码说明
`GLfloat currentPosX = 0, currentPosY = 0;` `std::deque<Point> pointList;` `float t = 0;`	全局变量，用于记录当前鼠标位置、已选择的控制点位置以及绘图计时。
`void processMouseClick(GLFWwindow* window, int button, int action, int mods) {` `  if (button == GLFW_MOUSE_BUTTON_LEFT) {` `  if (action == GLFW_PRESS) {` `  pointList.push_back(Point(currentPosX, currentPosY));` `  t = 0;` `  }` `  }` `  else if(button == GLFW_MOUSE_BUTTON_RIGHT){` `  if (action == GLFW_PRESS) {` `  pointList.pop_back();` `  t = 0;` `  }` `  }` `}`	用于处理鼠标左右键的点击函数。当点击鼠标左键时，将当前鼠标所在的位置记为新的控制点并保存；当点击鼠标右键时，将最后添加的控制点删除。任何鼠标的点击操作都会重置绘图过程的计时。
`void processMousePosition(GLFWwindow* window, double xpos, double ypos) {` `  currentPosX = GLint(xpos);` `  currentPosY = GLint(ypos);` `}`	用于获取鼠标当前位置的函数。鼠标位置一般以像素为单位，所以可以使用整型数进行存储。
`struct Point {` `  GLint x, y;` `  Point(GLint a, GLint b) {` `  x = a;` `  y = b;` `  }` `};`	存储一个控制点的数据结构，只记录控制点的屏幕 xy 坐标（以像素为基本单位）。
`class Bezier {` `public:` `private:` `  const int WINDOW_WIDTH;` `  const int WINDOW_HEIGHT;` `public:` `  Bezier(int w, int h) :WINDOW_WIDTH(w), WINDOW_HEIGHT(h) {};` `  // 保存直线上的点` `  void savePoint(float temp_X, float temp_Y, std::vector<GLfloat>& out) {` `  void drawBezier(std::deque<Point>& points, std::vector<GLfloat>& out) {` `  void show(std::deque<Point>& points, float t) {` `  int C(int n, int m) {` `};`	用于绘制 Bezier 曲线的类。共有四个函数。savePoint 用于把计算出来的 Bezier 曲线上的点的像素坐标转化为 OpenGL 的屏幕坐标；drawBezier 用于计算 Bezier 曲线上的点的坐标；show 用于显示绘制过程；C 是组合数公式，用于计算组合数。
`void savePoint(float temp_X, float temp_Y, std::vector<GLfloat>& out) {` `  GLfloat color = 1.0f;` `  out.push_back(GLfloat(float(temp_X) / float(WINDOW_WIDTH / 2)) - 1.0f);` `  out.push_back(1.0f - GLfloat(float(temp_Y) / float(WINDOW_HEIGHT / 2)));` `  out.push_back(0.0f);` `  for (int i = 0; i < 3; i++) {` `  out.push_back(color);` `  }` `}`	savePoint 函数。用于将计算出来的 Bezier 曲线上的点转为屏幕坐标中的点。
`void drawBezier(std::deque<Point>& points, std::vector<GLfloat>& out) {` `  if (points.size() <= 2) {` `  return;` `  }` `  for (int t = 0; t <= 1000; t++) {` `  double x = 0, y = 0;` `  for (int i = 0; i < points.size(); i++) {` `  x += C(points.size()-1, i) * pow(0.001*t, i) *` `  pow((1.0 - 0.001*t), points.size()-1 - i) * points[i].x;` `  y += C(points.size()-1, i) * pow(0.001*t, i) *` `  pow((1.0 - 0.001*t), points.size()-1 - i) * points[i].y;` `  }` `  savePoint(x, y, out);` `  }` `}`	drawBezier 函数。Bezier 曲线公式 $B(t)=\sum_{i=0}^{n} C_n^i P_i (1-t)^{n-i} t^i$ 的具体实现。其中参数 t 在范围[0,1]内进行了 1000 次抽样，使得 Bezier 曲线在视觉上保持连续。
`void show(std::deque<Point>& points, float t) {` `  if (points.size() <= 1) {` `  return;` `  }` `  std::deque<Point> v;` `  for (int i = 0; i < points.size()-1; i++) {` `  Point temp(int(t * points[i+1].x + (1 - t) * (points[i].x) + 0.5),` `  int(t * points[i+1].y + (1 - t) * (points[i].y) + 0.5));` `  v.push_back(temp);` `  }` `  points = v;` `}`	show 函数。用于计算在绘制 Bezier 曲线的过程中所有插值点的位置。

`` int C(int n, int m) { if (m == 0 || n == m) return 1; int sb = std::min(m, n - m); int f = 1, f1 = 0; for (int i = 1; i <= sb; i++){ f1 = f * (n - i + 1) / (i); f = f1; } return f1; } ```	C 函数。使用递推法计算 C_n^i 的数值。
`` // 作图 // 点 for (int ptr = 0; ptr < pointList.size(); ptr++) { for (int i = -3; i < 3; i++) { for (int j = -3; j < 3; j++) { bresenham.saveLinePoint(pointList[ptr].x+i, pointList[ptr].y+j, p); } } } // 线 if (pointList.size() >= 2) { for (int ptr = 0; ptr < pointList.size() - 1; ptr++) { bresenham.drawLine(pointList[ptr].x, pointList[ptr].y, pointList[ptr + 1].x, pointList[ptr + 1].y, p); } } // bezier曲线 bezier.drawBezier(pointList, p); ```	主函数中的绘图部分。首先将控制点绘制成大小为 6*6 的白色方块；然后使用 Bresenham 方式绘制直线将它们连起来；最后根据控制点列表将 Bezier 曲线绘制出来。
`` // 画直线 void drawLine(GLint Xa, GLint Ya, GLint Xb, GLint Yb, std::vector<GLfloat>& res) { drawLineWithBresenham(Xa, Ya, Xb, Yb, res); } ```	在作业 3（Bresenham 画三角形）的基础上，增添绘制单一直线功能。控制点间的连线用此方法进行绘制。
`` // 过程 t += deltaTime * 0.05; if (pointList.size() >= 3) { std::deque<Point> tp = pointList; for (int count = 0; count < pointList.size() - 1; count++) { bezier.show(tp, t < 1.0 ? t : 1.0); for (int ptr = 0; ptr < tp.size(); ptr++) { for (int i = -3; i < 3; i++) { for (int j = -3; j < 3; j++) { bresenham.saveLinePoint(tp[ptr].x + i, tp[ptr].y + j, p); } } } for (int ptr = 0; ptr < tp.size() - 1; ptr++) { bresenham.drawLine(tp[ptr].x, tp[ptr].y, tp[ptr + 1].x, tp[ptr + 1].y, p); } } } if (t >= 1.0f) { t = 0.0f; } ```	Bonus 部分：将 Bezier 曲线公式 $B(t)=\sum_{i=0}^{n} C_n^i P_i (1-t)^{n-i} t^i$ 中的参数 t 设置为动态值，随时间的增长而增大。t 范围为[0,1]。假如有 n 个控制点，那么一共会进行 n-1 轮插值过程。依次计算每轮插值过程中得到的插值点及其之间的连线，保存起来。n-1 轮插值计算完成后将插值点和插值点间的连线渲染到屏幕上。

第二部分：实验截图分析

（1）左键添加控制点，右键删除最后添加的控制点





（2）Bonus：动态地呈现 Bezier 曲线的生成过程