|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **计算机图形学实验报告（五）** | | | | | | |
| 学 号 | 姓 名 | | 班 级 | | 报告日期 | |
| 180400715 | 杨卓宸 | | 1804102 | | 2020/11/18 | |
| 实验内容 | 建立一个MFC单文档程序，完成以下平面曲线的设计功能:   1. 实现Bezier曲线的交互绘制，即用鼠标点击给出控制点的位置， 定义3次Bezier曲线，并绘制出控制点、控制多边形和Bezier曲线。 2. 能够绘制多段 Bezier 曲线并进行拼接，使得拼接达到 1 阶几何连续(G1连续)。 3. 利用实现的上述功能，设计出一个有点复杂度的平面图形，例如手的轮廓、一个动物的轮廓等。 | | | | | |
| 实验目的 | 要求:   1. Bezier 曲线上点的计算要求使用de Casteljau算法。 2. 不同段的 Bezier 曲线用不同的颜色显示。 3. 为了达到两段曲线的G1拼接，要求后一段曲线的第一个控制点和前一段曲线的最后一个控制点重合，并且后一段曲线的第2个控制点、前一段曲线的倒数第2个控制点、前一段曲线的最后一个控制点 (即后一段曲线的第一个控制点)在同一条直线上。   可以在前一段曲线设计完成后，画出曲线上最后一个点处的切线 L，用户绘制后一段曲线的第2个控制点时需绘制在直线L附近，程序自动将用户绘制的控制点投影到切线 L 上。 | | | | | |
| 实验预备知识 | De Casteljau算法的基本观点是选择在AB中的一个点C，C将AB分为u:1-u(A到C的距离与AB之间的距离之比是u)，让我们找到决定C在哪里的方法。  从A到B的向量是B-A。因为u是在0和1之间的比率，点C位于u(B-A)。将A的位置加以考虑，点C为A+u(B-A)=(1-u)A+uB。因此，对于给定的u，(1-u)A+uB是在A和B之间的点C，将AB分为u:1-u的两段  De Casteljau算法的想法如下。假设我们想要找到C(u)，u在[0,1]中。由第一个多段线00-01-02-03...-0n开始，利用上面的法则找到在线段上的点1i，1i在0i到0(i+1)的连线上并且将这段线分为u:1-u的两部分。依次地，我们可以得到n个点10,11,12,...,1(n-1)，他们定义了一个新的多段线（polyline），一共有n-1段。  新点由1i进行标记，再次利用上面的规则我们可以得到第二个多段线，具有n-1个点（20,21,...,2(n-2)）和n-2条边。从这个多段线开始，进行第三次，得到新的多段线，由n-2个点30,31,...,3(n-3)和n-3条边组成。重复这个过程n次得到一个点n0，Casteljau已经证明在曲线上的点C(u)对应u。  这是de Casteljau算法的几何解释，是在曲线设计中最漂亮结果之一。  de Casteljau 算法伪代码：  Input: array P[0:n] of n+1 points and real number u in [0,1]  Output: point on curve, C(u)  Working: point array Q[0:n]  for i := 0 to n do  Q[i] := P[i]; // save input  for k := 1 to n do  for i := 0 to n - k do  Q[i] := (1 - u)Q[i] + u Q[i + 1];  return Q[0]; | | | | | |
| 实验过程描述 | // 重绘所有直线和贝塞尔曲线  int n = 0;  for (auto thisPoint = startPoint.begin(); thisPoint < startPoint.end(); ++thisPoint){  n++;  // 重绘直线  if (thisPoint == startPoint.begin())  {  pDC->MoveTo(\*thisPoint);  continue;  }  pDC->LineTo(\*thisPoint);  pDC->MoveTo(\*thisPoint);  // 重绘曲线  if (n % 3 == 1 && n != 1)  {  std::vector<CPoint> p(thisPoint - 3, thisPoint + 1);  CPen penCil;  // 第奇数条曲线，画红线  if (n / 3 % 2)  {  penCil.CreatePen(PS\_SOLID, 5, RGB(0, 255, 0));  }  // 第偶数条曲线，画蓝线  else  {  penCil.CreatePen(PS\_SOLID, 5, RGB(0, 0, 255));  }  DrawBezierCurve(p, penCil);  }  }  // 绘制贝塞尔曲线  void CBezierCurveView::DrawBezierCurve(std::vector<CPoint>& p, CPen& thisPen)  {  CClientDC dc(this);  dc.SelectObject(&thisPen);  dc.MoveTo(p.front());  for (double i = 0.0; i < 1.0; i += 0.01)  {  double points[4] = { (1 - i) \* (1 - i) \* (1 - i), 3 \* i \* (1 - i) \* (1 - i), 3 \* i \* i \* (1 - i), i \* i \* i };  double p\_x = points[0] \* p[0].x + points[1] \* p[1].x + points[2] \* p[2].x + points[3] \* p[3].x;  double p\_y = points[0] \* p[0].y + points[1] \* p[1].y + points[2] \* p[2].y + points[3] \* p[3].y;  dc.LineTo(p\_x, p\_y);  dc.MoveTo(p\_x, p\_y);  }  dc.LineTo(p.back());  } | | | | | |
| 实验结果 | 1.可实现左键点击三次绘出曲线，并且奇数次曲线为绿色，偶数次曲线为蓝色。  2  2.窗口变动后，曲线仍可以重绘。  3  3.使用Bezier曲线绘制出的灯笼形对称图案。  1 | | | | | |
| 实验当中问题  及解决方法 | 1、Point点的存储对我造成了一定的困难。一开始有些无从下手。最终选用了vector的CPoint类数组存储，方便很多，并且可以随时调用。   1. 鼠标左键绘图和右键暂停，我采用了实验一的解决办法，添加相应函数进行处理，也算是对之前学过的知识的总结。为了应用程序更完美的实现，我亦添加了窗口重绘功能，也采用实验一相同的处理办法。   3、Bezier曲线的一阶连续，我使用了点集进行存储关键点，并通过De Casteljau算法计算得到。 | | | | | |
| 成绩（教师打分） | 优秀 | 良好 | | 及格 | | 不及格 |