**海南大学计算机科学与技术学院**

**《计算机图形学》课内实验报告七**

****

**班 级：计算机科学与技术2021-3班**

**成 员： 李季鸿 20213002624**

**报告名称： 实验报告七**

**指导老师： 高新瑞**

**完成日期： 2023年11月09日**

**《计算机图形学》实验报告七**

**学生姓名：李季鸿 班级：21级计科3班 学号：20213002624**

**实验地点：9-202 指导教师：高新瑞**

**实验日期：**2023.11.09  **实验课时：2学时**

**实验环境：**Windows 10+JDK1.8+记事本+IntelliJ IDEA

**一、实验目的**

**实验一目的：**

学习如何使用两个双三次Bezier曲面拼接编程。

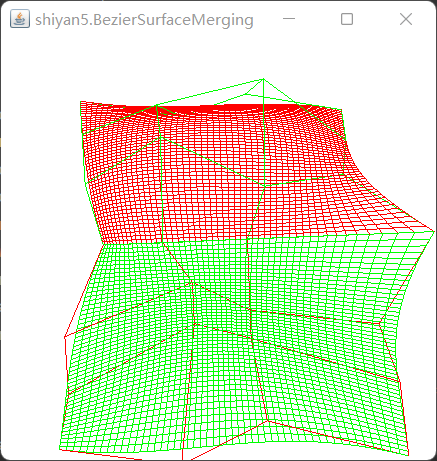
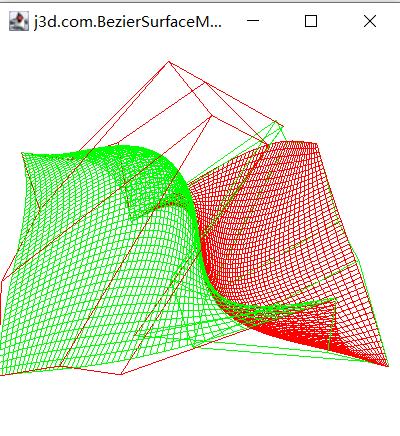
1. **实验过程**

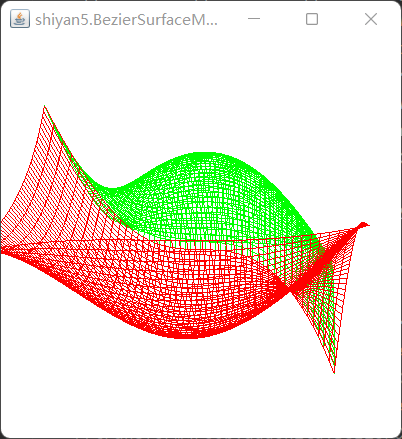
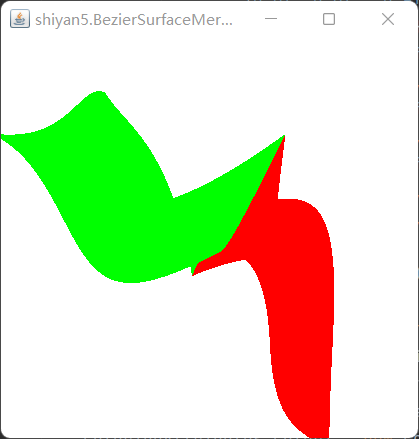
## 实验内容一：

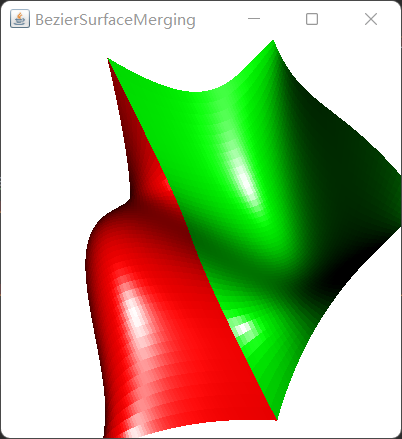
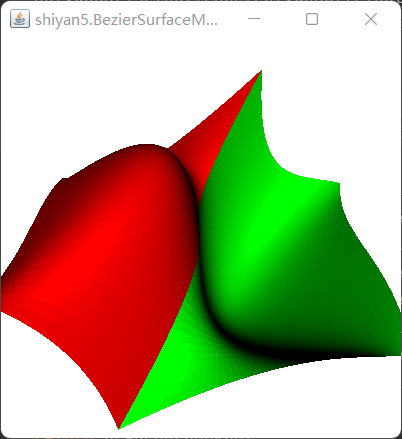
（1）代码

package week11\_seventh.\_4\_3;  
  
import com.sun.j3d.utils.applet.MainFrame;  
import com.sun.j3d.utils.behaviors.mouse.MouseRotate;  
import com.sun.j3d.utils.behaviors.mouse.MouseTranslate;  
import com.sun.j3d.utils.behaviors.mouse.MouseZoom;  
import com.sun.j3d.utils.universe.SimpleUniverse;  
  
import javax.media.j3d.\*;  
import javax.vecmath.Color3f;  
import javax.vecmath.Point3d;  
import javax.vecmath.Point3f;  
import javax.vecmath.Vector3f;  
import java.applet.Applet;  
import java.awt.\*;  
  
/\*\*  
 \* \\* Created with IntelliJ IDEA.  
 \* \\* @ProjectName: Computer graphics  
 \* \\* @FileName: BezierSurfaceMerging  
 \* \\* @author: li-jihong  
 \* \\* Date: 2023-11-09 19:24  
 \*/  
public class BezierSurfaceMerging extends Applet {  
 public BezierSurfaceMerging() {  
 setLayout(new BorderLayout());  
 GraphicsConfiguration gc = SimpleUniverse.getPreferredConfiguration();  
 Canvas3D c = new Canvas3D(gc);  
 add("Center", c);  
 BranchGroup BranchGroupScene = createBranchGroupSceneGraph();  
 SimpleUniverse u = new SimpleUniverse(c);  
 u.getViewingPlatform().setNominalViewingTransform();  
 u.addBranchGraph(BranchGroupScene);  
 }  
  
 public static void main(String[] args) {  
 new MainFrame(new BezierSurfaceMerging(), 400, 400);  
 }  
  
 public BranchGroup createBranchGroupSceneGraph() {  
 BranchGroup BranchGroupRoot = new BranchGroup();  
 BoundingSphere bounds = new BoundingSphere(new Point3d(0.0, 0.0, 0.0), 100.0);  
 Color3f bgColor = new Color3f(1.0f, 1.0f, 1.0f);  
 Background bg = new Background(bgColor);  
 bg.setApplicationBounds(bounds);  
 BranchGroupRoot.addChild(bg);  
 Color3f directionalColor = new Color3f(1.f, 0.f, 0.f);  
 Vector3f vec = new Vector3f(0.f, 0.f, -1.0f);  
 DirectionalLight directionalLight = new DirectionalLight(directionalColor, vec);  
 directionalLight.setInfluencingBounds(bounds);  
 BranchGroupRoot.addChild(directionalLight);  
 Transform3D tr = new Transform3D();  
 tr.setScale(0.85);  
 TransformGroup transformgroup = new TransformGroup(tr);  
 transformgroup.setCapability(TransformGroup.ALLOW\_TRANSFORM\_WRITE);  
 transformgroup.setCapability(TransformGroup.ALLOW\_TRANSFORM\_READ);  
 BranchGroupRoot.addChild(transformgroup);  
 MouseRotate mouserotate = new MouseRotate();  
 mouserotate.setTransformGroup(transformgroup);  
 BranchGroupRoot.addChild(mouserotate);  
 mouserotate.setSchedulingBounds(bounds);  
 MouseZoom mousezoom = new MouseZoom();  
 mousezoom.setTransformGroup(transformgroup);  
 BranchGroupRoot.addChild(mousezoom);  
 mousezoom.setSchedulingBounds(bounds);  
 MouseTranslate mousetranslate = new MouseTranslate();  
 mousetranslate.setTransformGroup(transformgroup);  
 BranchGroupRoot.addChild(mousetranslate);  
 mousetranslate.setSchedulingBounds(bounds);  
//定义第一个Bezier曲面的16个控制顶点  
 float[][][] P1 = {{{-0.8f, 0.9f, -0.4f, 1.f},  
 {-0.2f, 0.8f, -0.5f, 1.f},  
 {0.2f, 0.9f, -0.4f, 1.f},  
 {0.8f, 0.8f, -0.5f, 1.f}},  
 {{-0.8f, 0.7f, -0.4f, 1.f},  
 {-0.2f, 0.6f, 0.9f, 1.f},  
 {0.2f, 0.7f, 0.8f, 1.f},  
 {0.8f, 0.6f, -0.4f, 1.f}},  
 {{-0.8f, 0.4f, -0.4f, 1.f},  
 {-0.2f, 0.5f, 0.8f, 1.f},  
 {0.2f, 0.3f, 0.7f, 1.f},  
 {0.8f, 0.4f, -0.5f, 1.f}},  
 {{-0.8f, 0.f, -0.8f, 1.f},  
 {-0.2f, 0.1f, 0.9f, 1.f},  
 {0.2f, 0.f, -0.8f, 1.f},  
 {0.8f, 0.1f, 0.9f, 1.f}}};  
//定义第一个Bezier曲面外观属性  
 Appearance app1 = new Appearance();  
 PolygonAttributes polygona1 = new PolygonAttributes();  
 polygona1.setBackFaceNormalFlip(true);  
 polygona1.setCullFace(PolygonAttributes.CULL\_NONE);  
 polygona1.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  
 app1.setPolygonAttributes(polygona1);  
 ColoringAttributes color1 = new ColoringAttributes();  
 color1.setColor(1.f, 0.f, 0.f);  
 app1.setColoringAttributes(color1);  
//定义第二个Bezier曲面的16个控制顶点  
 float[][][] P2 = {{{-0.8f, 0.f, -0.8f, 1.f},  
 {-0.2f, 0.1f, 0.9f, 1.f},  
 {0.2f, 0.f, -0.8f, 1.f},  
 {0.8f, 0.1f, 0.9f, 1.f}},  
 {{-0.8f, -0.4f, 0.2f, 1.f},  
 {-0.2f, -0.3f, -0.9f, 1.f},  
 {0.2f, -0.5f, -0.8f, 1.f},  
 {0.8f, -0.4f, 0.2f, 1.f}},  
 {{-0.8f, -0.7f, 0.2f, 1.f},  
 {-0.2f, -0.6f, -0.9f, 1.f},  
 {0.2f, -0.7f, -0.8f, 1.f},  
 {0.8f, -0.6f, 0.5f, 1.f}},  
 {{-0.8f, -0.9f, 0.4f, 1.f},  
 {-0.2f, -0.9f, 0.6f, 1.f},  
 {0.2f, -0.8f, 0.4f, 1.f},  
 {0.8f, -0.9f, 0.6f, 1.f}}};  
//定义第二个Bezier曲面外观属性  
 Appearance app2 = new Appearance();  
 PolygonAttributes polygona2 = new PolygonAttributes();  
 polygona2.setBackFaceNormalFlip(true);  
 polygona2.setCullFace(PolygonAttributes.CULL\_NONE);  
 polygona2.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  
 app2.setPolygonAttributes(polygona2);  
 ColoringAttributes color2 = new ColoringAttributes();  
 color2.setColor(0.f, 1.f, 0.f);  
 app2.setColoringAttributes(color2);  
 Shape3D BezierSurfaceface1 = new BezierThreeOrderSurfaceface(P1, app1);  
 transformgroup.addChild(BezierSurfaceface1);  
 Shape3D BezierSurfaceface2 = new BezierThreeOrderSurfaceface(P2, app2);  
 transformgroup.addChild(BezierSurfaceface2);  
 Shape3D BezierControlPoints1 = new BezierSurfaceControlPoints(P1, app2);  
 transformgroup.addChild(BezierControlPoints1);  
 Shape3D BezierControlPoints2 = new BezierSurfaceControlPoints(P2, app1);  
 transformgroup.addChild(BezierControlPoints2);  
 BranchGroupRoot.compile();  
 return BranchGroupRoot;  
 }  
}  
  
class BezierThreeOrderSurfaceface extends Shape3D {  
 public BezierThreeOrderSurfaceface(float[][][] P, Appearance app) {  
 int i, j, k;  
 int n0;//定义对参数u、v在[0，1]区间的等分点数  
 float division;//参数u在[0，1]区间的等分线段长度  
 n0 = 50;  
 division = 1.f / n0;  
//分别定义存放控制顶点x、y、z坐标与第四维坐标的数组  
 float[][] PX = new float[4][4];  
 float[][] PY = new float[4][4];  
 float[][] PZ = new float[4][4];  
 float[][] P4 = new float[4][4];  
//定义系数矩阵及其转置矩阵  
 float[][] M1 = {{1.f, 0.f, 0.f, 0.f},  
 {-3.f, 3.f, 0.f, 0.f},  
 {3.f, -6.f, 3.f, 0.f},  
 {-1.f, 3.f, -3.f, 1.f}};  
 float[][] M2 = {{1.f, -3.f, 3.f, -1.f},  
 {0.f, 3.f, -6.f, 3.f},  
 {0.f, 0.f, 3.f, -3.f},  
 {0.f, 0.f, 0.f, 1.f}};  
//定义Bezier曲面的u、v参数分割点坐标数组  
 float[][][] UV = new float[n0 + 1][n0 + 1][2];  
//定义U、V矩阵数组  
 float[][] UU = new float[1][4];  
 float[][] VV = new float[4][1];  
//定义存放曲面上点的坐标的数组  
 float[][][] SurfaceXYZ = new float[n0 + 1][n0 + 1][4];  
 for (i = 0; i < n0 + 1; i++)  
 for (j = 0; j < n0 + 1; j++) {  
 UV[i][j][0] = i \* division;  
 UV[i][j][1] = j \* division;  
 }  
 for (i = 0; i < 4; i++)  
 for (j = 0; j < 4; j++) {  
 PX[i][j] = P[i][j][0];  
 PY[i][j] = P[i][j][1];  
 PZ[i][j] = P[i][j][2];  
 P4[i][j] = P[i][j][3];  
 }  
//计算曲面上点的坐标值  
 for (i = 0; i < n0 + 1; i++)  
 for (j = 0; j < n0 + 1; j++) {  
 UU[0][0] = 1.f;  
 UU[0][1] = UV[i][j][0];  
 UU[0][2] = UV[i][j][0] \* UV[i][j][0];  
 UU[0][3] = UV[i][j][0] \* UV[i][j][0] \* UV[i][j][0];  
 VV[0][0] = 1.f;  
 VV[1][0] = UV[i][j][1];  
 VV[2][0] = UV[i][j][1] \* UV[i][j][1];  
 VV[3][0] = UV[i][j][1] \* UV[i][j][1] \* UV[i][j][1];  
 //计算一点的x坐标  
 matrixm g0 = new matrixm(1, 4, 4, UU, M1);  
 matrixm g1 = new matrixm(1, 4, 4, g0.CC, PX);  
 matrixm g2 = new matrixm(1, 4, 4, g1.CC, M2);  
 matrixm g3 = new matrixm(1, 4, 1, g2.CC, VV);  
 SurfaceXYZ[i][j][0] = g3.CC[0][0];  
 //计算一点的y坐标  
 matrixm g4 = new matrixm(1, 4, 4, UU, M1);  
 matrixm g5 = new matrixm(1, 4, 4, g4.CC, PY);  
 matrixm g6 = new matrixm(1, 4, 4, g5.CC, M2);  
 matrixm g7 = new matrixm(1, 4, 1, g6.CC, VV);  
 SurfaceXYZ[i][j][1] = g7.CC[0][0];  
 //计算一点的z坐标  
 matrixm g8 = new matrixm(1, 4, 4, UU, M1);  
 matrixm g9 = new matrixm(1, 4, 4, g8.CC, PZ);  
 matrixm g10 = new matrixm(1, 4, 4, g9.CC, M2);  
 matrixm g11 = new matrixm(1, 4, 1, g10.CC, VV);  
 SurfaceXYZ[i][j][2] = g11.CC[0][0];  
 //计算一点的第4维坐标  
 matrixm g12 = new matrixm(1, 4, 4, UU, M1);  
 matrixm g13 = new matrixm(1, 4, 4, g12.CC, P4);  
 matrixm g14 = new matrixm(1, 4, 4, g13.CC, M2);  
 matrixm g15 = new matrixm(1, 4, 1, g14.CC, VV);  
 SurfaceXYZ[i][j][3] = g15.CC[0][0];  
 //将齐次坐标转换为三维坐标系坐标，如果第4维为1，则不用除第4维  
 SurfaceXYZ[i][j][0] = SurfaceXYZ[i][j][0] / SurfaceXYZ[i][j][3];  
 SurfaceXYZ[i][j][1] = SurfaceXYZ[i][j][1] / SurfaceXYZ[i][j][3];  
 SurfaceXYZ[i][j][2] = SurfaceXYZ[i][j][2] / SurfaceXYZ[i][j][3];  
 }  
 QuadArray BezierQuadsurfaceface = new QuadArray(n0 \* n0 \* 4,  
 GeometryArray.COORDINATES | GeometryArray.NORMALS);  
 int c = 0;//以顶点数累加的方式设置顶点的序号  
 for (i = 0; i < n0; i++) {  
 for (j = 0; j < n0; j++) {//设置一个平面上的4个点  
 Point3f A = new Point3f(SurfaceXYZ[i][j][0], SurfaceXYZ[i][j][1],  
 SurfaceXYZ[i][j][2]);  
 Point3f B = new Point3f(SurfaceXYZ[i][j + 1][0], SurfaceXYZ[i][j + 1][1],  
 SurfaceXYZ[i][j + 1][2]);  
 Point3f C = new Point3f(SurfaceXYZ[i + 1][j + 1][0], SurfaceXYZ[i + 1][j + 1][1],  
 SurfaceXYZ[i + 1][j + 1][2]);  
 Point3f D = new Point3f(SurfaceXYZ[i + 1][j][0], SurfaceXYZ[i + 1][j][1],  
 SurfaceXYZ[i + 1][j][2]);  
//计算由四个点形成的平面的法向量  
 Vector3f a = new Vector3f(A.x - B.x, A.y - B.y, A.z - B.z);  
 Vector3f b = new Vector3f(C.x - B.x, C.y - B.y, C.z - B.z);  
 Vector3f n = new Vector3f();  
 n.cross(b, a);  
 n.normalize();  
//设置点的序号  
 BezierQuadsurfaceface.setCoordinate(c, A);  
 BezierQuadsurfaceface.setCoordinate(c + 1, B);  
 BezierQuadsurfaceface.setCoordinate(c + 2, C);  
 BezierQuadsurfaceface.setCoordinate(c + 3, D);  
//设置点的法向量  
 BezierQuadsurfaceface.setNormal(c, n);  
 BezierQuadsurfaceface.setNormal(c + 1, n);  
 BezierQuadsurfaceface.setNormal(c + 2, n);  
 BezierQuadsurfaceface.setNormal(c + 3, n);  
 c = c + 4;  
 }  
 }  
 this.addGeometry(BezierQuadsurfaceface);  
 this.setAppearance(app);  
 }  
}  
  
class BezierSurfaceControlPoints extends Shape3D {  
 public BezierSurfaceControlPoints(float[][][] P, Appearance app) {  
 int i, j, k;  
 QuadArray BeziersurfacecontrolPointsNet = new QuadArray(3 \* 3 \* 4,  
 GeometryArray.COORDINATES | GeometryArray.NORMALS);  
 int c = 0;  
 for (i = 0; i < 3; i++) {  
 for (j = 0; j < 3; j++) {  
 Point3f A = new Point3f(P[i][j][0], P[i][j][1], P[i][j][2]);  
 Point3f B = new Point3f(P[i][j + 1][0], P[i][j + 1][1], P[i][j + 1][2]);  
 Point3f C = new Point3f(P[i + 1][j + 1][0], P[i + 1][j + 1][1], P[i + 1][j + 1][2]);  
 Point3f D = new Point3f(P[i + 1][j][0], P[i + 1][j][1], P[i + 1][j][2]);  
 Vector3f a = new Vector3f(A.x - B.x, A.y - B.y, A.z - B.z);  
 Vector3f b = new Vector3f(C.x - B.x, C.y - B.y, C.z - B.z);  
 Vector3f n = new Vector3f();  
 n.cross(b, a);  
 n.normalize();  
 BeziersurfacecontrolPointsNet.setCoordinate(c, A);  
 BeziersurfacecontrolPointsNet.setCoordinate(c + 1, B);  
 BeziersurfacecontrolPointsNet.setCoordinate(c + 2, C);  
 BeziersurfacecontrolPointsNet.setCoordinate(c + 3, D);  
 BeziersurfacecontrolPointsNet.setNormal(c, n);  
 BeziersurfacecontrolPointsNet.setNormal(c + 1, n);  
 BeziersurfacecontrolPointsNet.setNormal(c + 2, n);  
 BeziersurfacecontrolPointsNet.setNormal(c + 3, n);  
 c = c + 4;  
 }  
 }  
 this.addGeometry(BeziersurfacecontrolPointsNet);  
 this.setAppearance(app);  
 }  
}  
  
class matrixm {  
 public float CC[][] = new float[4][4];  
 int ll, mm, kk;  
  
 public matrixm(int mmm, int kkk, int nnn, float a[][], float b[][]) {  
 for (ll = 0; ll < mmm; ll++)  
 for (mm = 0; mm < nnn; mm++) {  
 CC[ll][mm] = 0.f;  
 }  
 for (ll = 0; ll < mmm; ll++)  
 for (mm = 0; mm < nnn; mm++) {  
 for (kk = 0; kk < kkk; kk++) CC[ll][mm] = CC[ll][mm] + a[ll][kk] \* b[kk][mm];  
 }  
 }  
}

**三、实验总结**

结果截图（四边网格曲面模型）

两张双三次Bezier曲面的拼接及不同光照效果

**三、实验总结**

通过本次实验，进一步巩固理解了双三次Bezier曲面的定义，以及如何将双三次Bezier曲面的矩阵表示应用到java3D编程中，关于理论知识个人将其总结在第二部分。本次实验不仅需要理解和读懂代码，更要自己修改代码。要将Bezier曲线连接成曲面，然后设置高亮颜色，这是最关键的部分。连接成曲面只需要注释掉以下代码：

//polygona1.setPolygonMode(PolygonAttributes.POLYGON\_LINE);

//polygona2.setPolygonMode(PolygonAttributes.POLYGON\_LINE);

其中设置高亮最为关键的代码如下：

Material ma1 = new Material();  
ma1.setSpecularColor(1f, 0f, 0f);  
ma1.setDiffuseColor(1f, 0f, 0f);  
app1.setMaterial(ma1);