**海南大学计算机科学与技术学院**

**《计算机图形学》课内实验报告六**

****

**班 级：计算机科学与技术2021-3班**

**成 员： 李季鸿 20213002624**

**报告名称： 实验报告六**

**指导老师： 高新瑞**

**完成日期： 2023年11月02日**

**《计算机图形学》实验报告六**

**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：**2023.11.02  **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：**

1.学习如何使用Java 3D库创建和渲染三维图形。

2.掌握如何使用Java 3D库的不同组件，如`Canvas3D`、‘BranchGroup'、

'SimpleUniverse '、"Shape3D等，来构建三维场景。

3.熟悉如何实现鼠标交互功能，如旋转、平移和缩放，以便用户可以与三维场景进行交互。4.理解如何创建几何形状、设置外观（包括材质、颜色等)以及应用光照效果。

**实验二目的：**

1.学习如何使用Java 3D库创建和渲染三维图形，特别是球体的表面。

2.熟悉如何使用不同的几何数组类型(TriangleArray和QuadArray)来表示三维对象的不同部

分。

3.理解如何计算球体表面上的点的坐标，并为这些点设置法向量，以便在渲染时产生光照效

果。

4.学习如何实现鼠标交互功能，如旋转、平移和缩放，以便用户可以与三维对象互动。

**实验三目的：**

创建和展示三阶贝塞尔曲线的三维可视化。它包括了一个简单的三维场景，其中包含了贝塞尔曲线的可视化对象。

**实验四目的：**

实现了一个三阶贝塞尔曲面的可视化，包括控制顶点和曲面网格的绘制。

1. **实验过程**

**理论知识：**

1. **Bezier曲线的定义**

给定空间n+1个点P0、P1、P2…Pn,则可生成一条n次的Bezier曲线。



Bernstein基函数：



其中，P0、P1、P2…Pn称为n次的Bezier曲线的控制顶点；

u称为Bezier曲线的参数；

从公式中可看出，Bezier曲线的次数随着控制顶点个数的增加而增加。其中，最常用的是2次、3次Bezier曲线。通过调整控制顶点的位置，可改变曲线的形状。

**（2）Bernstein基函数的性质**

①非负性

②规范性

③对称性

④端点性

⑤可导性

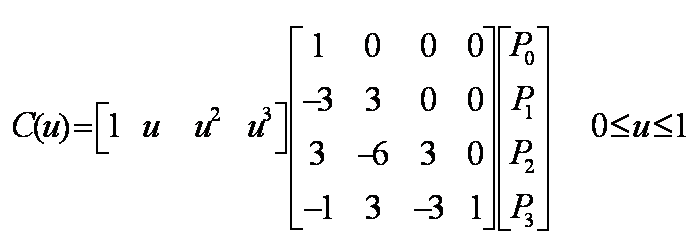
⑥基函数的最值

**（3）三次Bezier曲线的矩阵表示形式**

Bezier曲线的矩阵表示形式的优点就是便于计算机编程实现。如果给定4个控制顶点P0、P1、P2、P3，此时，n=3，则可生成一条三次Bezier曲线。三次Bezier曲线代数表示形式如下：



矩阵表示形式：



## 实验内容一：

（1）代码

package week10\_sixth.\_3\_17;  
  
/\*\*  
 \* \\* Created with IntelliJ IDEA.  
 \* \\* @ProjectName: Computer graphics  
 \* \\* @FileName: RotationQuadarray  
 \* \\* @author: li-jihong  
 \* \\* Date: 2023-11-02 10:45  
 \*/  
  
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.\*;  
  
public class RotationQuadarray extends Applet {  
 public RotationQuadarray() {  
 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 argv[]) {  
 new MainFrame(new RotationQuadarray(), 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);  
 TransformGroup transformgroup = new TransformGroup();  
 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);  
 Shape3D RotationSurface = new SurfaceDisplay();  
 transformgroup.addChild(RotationSurface);  
 BranchGroupRoot.compile();  
 return BranchGroupRoot;  
 }  
}  
  
class SurfaceDisplay extends Shape3D {  
 float[][][] SurfacePointsxyz = new float[5][51][3];  
  
 SurfaceDisplay() {  
 this.setGeometry(createGeometry0());  
 this.setAppearance(createAppearance0());  
 }  
  
 Geometry createGeometry0() {  
 int i, j, k, c;  
 int n0 = 50;  
 float theta;  
 //定义在xoy平面内的旋转线，旋转轴为y轴  
 float[] Xp0 = {.3f, .4f, .5f, .6f, .9f};  
 float[] Yp0 = {-.4f, -.2f, 0.f, .2f, .5f};  
 //计算对圆周n0等分后所得的旋转角  
 theta = 2.f \* (float) Math.PI / n0;  
 //计算旋转后旋转平面上点的x、y、z坐标值  
 for (i = 0; i < 5; i++)  
 for (j = 0; j < n0 + 1; j++) {  
 SurfacePointsxyz[i][j][0] = Xp0[i] \* (float) Math.cos(theta \* j);  
 SurfacePointsxyz[i][j][1] = Yp0[i];  
 SurfacePointsxyz[i][j][2] = Xp0[i] \* (float) Math.sin(theta \* j);  
 }  
 //按顺时针方向设置每个四边形点的坐标值，求法向量  
 QuadArray SurfaceQuadArray =  
 new QuadArray(5 \* n0 \* 4, GeometryArray.COORDINATES | GeometryArray.NORMALS);  
 c = 0;//该变量用来对顶点按顺序编号，该编号是连续的，不能重复  
 for (i = 0; i < 4; i++) {  
 for (j = 0; j < n0; j++) {  
 Point3f A = new Point3f(SurfacePointsxyz[i][j][0],  
 SurfacePointsxyz[i][j][1], SurfacePointsxyz[i][j][2]);  
 Point3f B = new Point3f(SurfacePointsxyz[i + 1][j][0],  
 SurfacePointsxyz[i + 1][j][1], SurfacePointsxyz[i + 1][j][2]);  
 Point3f C = new Point3f(SurfacePointsxyz[i + 1][j + 1][0],  
 SurfacePointsxyz[i + 1][j + 1][1], SurfacePointsxyz[i + 1][j + 1][2]);  
 Point3f D = new Point3f(SurfacePointsxyz[i][j + 1][0],  
 SurfacePointsxyz[i][j + 1][1], SurfacePointsxyz[i][j + 1][2]);  
 //计算四个点的法向量，使法向量指向体外  
 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  
 SurfaceQuadArray.setCoordinate(c, A);  
 SurfaceQuadArray.setCoordinate(c + 1, B);  
 SurfaceQuadArray.setCoordinate(c + 2, C);  
 SurfaceQuadArray.setCoordinate(c + 3, D);  
 //设置点的序号所对应的法向量  
 SurfaceQuadArray.setNormal(c, n);  
 SurfaceQuadArray.setNormal(c + 1, n);  
 SurfaceQuadArray.setNormal(c + 2, n);  
 SurfaceQuadArray.setNormal(c + 3, n);  
 c = c + 4;  
 }  
 }  
 return SurfaceQuadArray;  
 }  
  
 Appearance createAppearance0() {//指定外观，这样才有明暗效果  
 PolygonAttributes polygona = new PolygonAttributes();  
 //有了下面这两行语句，在有法向量的情况下，可使面的两侧都能显示  
 polygona.setBackFaceNormalFlip(true);  
 polygona.setCullFace(PolygonAttributes.CULL\_NONE);  
 //polygona.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  
 //polygona.setPolygonMode(PolygonAttributes.POLYGON\_POINT);  
 Appearance appearance = new Appearance();  
 appearance.setPolygonAttributes(polygona);  
 Material material = new Material();  
 Color3f white = new Color3f(1.0f, .0f, .0f);  
 Color3f red = new Color3f(.0f, .0f, 1.0f);  
 material.setDiffuseColor(white);  
 //material.setSpecularColor(red);  
 //material.setShininess(20.0f);  
 appearance.setMaterial(material);  
 return appearance;  
 }  
}

（2）结果截图（四边网格曲面模型）

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 面模型 | 线模型 | 点模型 |

## 实验内容二：

1. 代码

package week10\_sixth.\_3\_19;  
  
/\*\*  
 \* \\* Created with IntelliJ IDEA.  
 \* \\* @ProjectName: Computer graphics  
 \* \\* @FileName: SphereTriangleQuadSurfacenew  
 \* \\* @author: li-jihong  
 \* \\* Date: 2023-11-02 10:48  
 \*/  
  
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.\*;  
  
public class SphereTriangleQuadSurfacenew extends Applet {  
 public SphereTriangleQuadSurfacenew() {  
 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 argv[]) {  
 new MainFrame(new SphereTriangleQuadSurfacenew(), 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, 1.f, 1.f);  
 Vector3f vec = new Vector3f(0.f, 0.f, -1.0f);  
 DirectionalLight directionalLight = new DirectionalLight(directionalColor, vec);  
 directionalLight.setInfluencingBounds(bounds);  
 BranchGroupRoot.addChild(directionalLight);  
 TransformGroup transformgroup = new TransformGroup();  
 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);  
 transformgroup.addChild(new SphereTriangleArrayDisplay());  
 transformgroup.addChild(new SphereQuadArrayDisplay());  
 BranchGroupRoot.compile();  
 return BranchGroupRoot;  
 }  
}  
  
class SphereTriangleArrayDisplay extends Shape3D {  
 SphereTriangleArrayDisplay() {//定义两个TriangleArray数组  
 float theta1, theta2;//等分角  
 float R = 0.8f;//球体半径  
 int i, j, k;  
 int nn1 = 20, nn2 = 50;//对球体表面的纵向与横向等分点数  
 TriangleArray Trianglesurface1 = new TriangleArray(nn2 \* 3,  
 TriangleArray.COORDINATES | TriangleArray.NORMALS);  
 TriangleArray Trianglesurface2 = new TriangleArray(nn2 \* 3,  
 TriangleArray.COORDINATES | TriangleArray.NORMALS);  
 //定义存放球体体表数据点的数组  
 float[][][] spherexyz = new float[100][200][3];  
 theta1 = (float) Math.PI / nn1;//纵向分角  
 theta2 = 2.0f \* (float) Math.PI / nn2;//横向分角  
 //球体表坐标点计算  
 for (i = 0; i < nn1 + 1; i++)  
 for (j = 0; j < nn2 + 1; j++) {  
 spherexyz[i][j][0] = R \* (float) Math.sin(i \* theta1) \* (float) Math.cos(j \* theta2);  
 spherexyz[i][j][1] = R \* (float) Math.cos(i \* theta1);  
 spherexyz[i][j][2] = R \* (float) Math.sin(i \* theta1) \* (float) Math.sin(j \* theta2);  
 }  
 int c = 0;//以顶点数累加的方式设置点的序号  
 Point3f A01 = new  
 Point3f(spherexyz[0][0][0], spherexyz[0][0][1], spherexyz[0][0][2]);  
 for (j = 0; j < nn2; j++) {  
 Point3f A1 = new Point3f(spherexyz[1][j][0],  
 spherexyz[1][j][1], spherexyz[1][j][2]);  
 Point3f A2 = new Point3f(spherexyz[1][j + 1][0],  
 spherexyz[1][j + 1][1], spherexyz[1][j + 1][2]);  
 Vector3f a = new Vector3f(A1.x - A01.x, A1.y - A01.y, A1.z - A01.z);  
 Vector3f b = new Vector3f(A2.x - A01.x, A2.y - A01.y, A2.z - A01.z);  
 Vector3f n = new Vector3f();  
 n.cross(a, b);  
 n.normalize();  
 //设置点序号及坐标  
 Trianglesurface1.setCoordinate(c, A01);  
 Trianglesurface1.setCoordinate(c + 1, A1);  
 Trianglesurface1.setCoordinate(c + 2, A2);  
 //设置点法向量  
 Trianglesurface1.setNormal(c, n);  
 Trianglesurface1.setNormal(c + 1, n);  
 Trianglesurface1.setNormal(c + 2, n);  
 c = c + 3;  
 }  
 c = 0;  
 Point3f A02 = new Point3f(spherexyz[nn1][0][0],  
 spherexyz[nn1][0][1], spherexyz[nn1][0][2]);  
 for (j = 0; j < nn2; j++) {  
 Point3f A1 = new Point3f(spherexyz[nn1 - 1][j][0],  
 spherexyz[nn1 - 1][j][1], spherexyz[nn1 - 1][j][2]);  
 Point3f A2 = new Point3f(spherexyz[nn1 - 1][j + 1][0],  
 spherexyz[nn1 - 1][j + 1][1], spherexyz[nn1 - 1][j + 1][2]);  
 Vector3f a = new Vector3f(A1.x - A02.x, A1.y - A02.y, A1.z - A02.z);  
 Vector3f b = new Vector3f(A2.x - A02.x, A2.y - A02.y, A2.z - A02.z);  
 Vector3f n = new Vector3f();  
 n.cross(a, b);  
 n.normalize();  
 //设置点序号及坐标  
 Trianglesurface2.setCoordinate(c, A02);  
 Trianglesurface2.setCoordinate(c + 1, A1);  
 Trianglesurface2.setCoordinate(c + 2, A2);  
 //设置点法向量  
 Trianglesurface2.setNormal(c, n);  
 Trianglesurface2.setNormal(c + 1, n);  
 Trianglesurface2.setNormal(c + 2, n);  
 c = c + 3;  
 }  
 this.addGeometry(Trianglesurface1);  
 this.addGeometry(Trianglesurface2);  
 this.setAppearance(createAppearance0());  
 }  
  
 Appearance createAppearance0() {//指定外观  
 PolygonAttributes polygona = new PolygonAttributes();  
 polygona.setCullFace(PolygonAttributes.CULL\_NONE);  
 polygona.setBackFaceNormalFlip(true);  
 //polygona.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  
 Appearance appearance = new Appearance();  
 appearance.setPolygonAttributes(polygona);  
 Material material = new Material();  
 Color3f red = new Color3f(.0f, 1.0f, 0.0f);  
 material.setDiffuseColor(red);  
 appearance.setMaterial(material);  
 return appearance;  
 }  
}  
  
class SphereQuadArrayDisplay extends Shape3D {  
 SphereQuadArrayDisplay() {//计算球面上点的x、y、z坐标  
 int nn1 = 20, nn2 = 50;  
 float theta1, theta2;  
 float R = 0.8f;  
 int i, j, k;  
 float[][][] spherexyz = new float[100][200][3];  
 theta1 = (float) Math.PI / nn1;  
 theta2 = 2.0f \* (float) Math.PI / nn2;  
 for (i = 0; i < nn1 + 1; i++)  
 for (j = 0; j < nn2 + 1; j++) {  
 spherexyz[i][j][0] = R \* (float) Math.sin(i \* theta1) \* (float) Math.cos(j \* theta2);  
 spherexyz[i][j][1] = R \* (float) Math.cos(i \* theta1);  
 spherexyz[i][j][2] = R \* (float) Math.sin(i \* theta1) \* (float) Math.sin(j \* theta2);  
 }  
 QuadArray Quadsurface = new QuadArray((nn1 - 2) \* nn2 \* 4,  
 GeometryArray.COORDINATES | GeometryArray.NORMALS);  
 int c = 0;//以顶点数累加的方式设置数组中顶点的序号  
 for (i = 1; i < nn1 - 1; i++) {  
 for (j = 0; j < nn2; j++) {//设置一个四边形上的4个点坐标值  
 Point3f A = new Point3f(spherexyz[i][j][0], spherexyz[i][j][1], spherexyz[i][j][2]);  
 Point3f B = new Point3f(spherexyz[i][j + 1][0],  
 spherexyz[i][j + 1][1], spherexyz[i][j + 1][2]);  
 Point3f C = new Point3f(spherexyz[i + 1][j + 1][0],  
 spherexyz[i + 1][j + 1][1], spherexyz[i + 1][j + 1][2]);  
 Point3f D = new Point3f(spherexyz[i + 1][j][0],  
 spherexyz[i + 1][j][1], spherexyz[i + 1][j][2]);  
 //计算四边形的法向量  
 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  
 Quadsurface.setCoordinate(c, A);  
 Quadsurface.setCoordinate(c + 1, B);  
 Quadsurface.setCoordinate(c + 2, C);  
 Quadsurface.setCoordinate(c + 3, D);  
 //按序号设置点法向量  
 Quadsurface.setNormal(c, n);  
 Quadsurface.setNormal(c + 1, n);  
 Quadsurface.setNormal(c + 2, n);  
 Quadsurface.setNormal(c + 3, n);  
 c = c + 4;  
 }  
 this.addGeometry(Quadsurface);  
 this.setAppearance(createAppearance0());  
 }  
 }  
  
 Appearance createAppearance0() {//指定外观  
 PolygonAttributes polygona = new PolygonAttributes();  
 polygona.setCullFace(PolygonAttributes.CULL\_NONE);  
 polygona.setBackFaceNormalFlip(true);  
 //polygona.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  
 Appearance appearance = new Appearance();  
 appearance.setPolygonAttributes(polygona);  
 Material material = new Material();  
 Color3f red = new Color3f(1.0f, .0f, .0f);  
 //material.setAmbientColor(red);  
 //material.setSpecularColor(red);  
 material.setDiffuseColor(red);  
 //material.setEmissiveColor(red);  
 //material.setShininess(50.f);  
 appearance.setMaterial(material);  
 return appearance;  
 }  
}

（2）结果截图（球面模型）

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

由三角扇网格、四边网格组成的球面模型

## 实验内容三：

（1）代码

package week10\_sixth.\_4\_1;  
  
/\*\*  
 \* \\* Created with IntelliJ IDEA.  
 \* \\* @ProjectName: Computer graphics  
 \* \\* @FileName: BezierThreeOrderCurve  
 \* \\* @author: li-jihong  
 \* \\* Date: 2023-11-02 10:50  
 \*/  
  
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.\*;  
  
public class BezierThreeOrderCurve extends Applet {  
 public BezierThreeOrderCurve() {  
 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 BezierThreeOrderCurve(), 300, 300);  
 }  
  
 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);  
 TransformGroup transformgroup = new TransformGroup();  
 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);  
 Shape3D Bezier = new BezierThreeOrderCurve0();  
 transformgroup.addChild(Bezier);  
 BranchGroupRoot.compile();  
 return BranchGroupRoot;  
 }  
}  
  
class BezierThreeOrderCurve0 extends Shape3D {  
 public BezierThreeOrderCurve0() {  
 int i, j, k;  
 float[] u = new float[100];  
 //下面定义存放[1 u u2 u3]参数矩阵的数组  
 float[][] UU = new float[1][4];  
 //下面定义存放Bezier曲线上点的坐标的数组  
 float[][] curvepoints = new float[100][4];  
 //存放控制顶点坐标的数组  
 float[][] P = {{-0.9f, -0.6f, 0.f, 1.f},  
 {-0.2f, 0.7f, 0.f, 1.f},  
 {0.2f, 0.9f, 0.f, 1.f},  
 {0.9f, -0.7f, 0.f, 1.f}};  
 //存放系数矩阵M的数组  
 float[][] M = {{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}};  
 int n;//对参数u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  
 n = 50;  
 division = 1.f / n;  
 for (i = 0; i < n + 1; i++) {  
 u[i] = i \* division;  
 }  
 for (i = 0; i < n + 1; i++) {  
 UU[0][0] = 1.f;  
 UU[0][1] = u[i];  
 UU[0][2] = u[i] \* u[i];  
 UU[0][3] = u[i] \* u[i] \* u[i];  
 matrixm g0 = new matrixm(1, 4, 4, UU, M);  
 matrixm g1 = new matrixm(1, 4, 4, g0.CC, P);  
 curvepoints[i][0] = g1.CC[0][0];  
 curvepoints[i][1] = g1.CC[0][1];  
 curvepoints[i][2] = g1.CC[0][2];  
 curvepoints[i][3] = g1.CC[0][3];  
 //在齐次坐标表示中，前三维的的坐标要除第四维的坐标，才能得到三维坐标系下的坐标值。  
 //在该程序中，第四维的值为1,也可以不除  
 curvepoints[i][0] = g1.CC[0][0] / curvepoints[i][3];  
 curvepoints[i][1] = g1.CC[0][1] / curvepoints[i][3];  
 curvepoints[i][2] = g1.CC[0][2] / curvepoints[i][3];  
 }  
 //定义曲线上点的数组  
 Point3f[] curvepoints0 = new Point3f[100];  
 for (i = 0; i < 100; i++) curvepoints0[i] = new Point3f();  
 for (k = 0; k < n + 1; k++) { //将曲线上点的二维数组转换为Point3f类型的一维数组  
 curvepoints0[k].x = curvepoints[k][0];  
 curvepoints0[k].y = curvepoints[k][1];  
 curvepoints0[k].z = curvepoints[k][2];  
 }  
 //将控制顶点的二维数组转换为Point3f类型的一维数组  
 Point3f[] contralpoints = new Point3f[4];  
 for (i = 0; i < 4; i++) contralpoints[i] = new Point3f();  
 for (i = 0; i < 4; i++) {  
 contralpoints[i].x = P[i][0];  
 contralpoints[i].y = P[i][1];  
 contralpoints[i].z = P[i][2];  
 }  
 int[] contralpointscount = new int[1];  
 int[] curvepointscount = new int[1];  
 contralpointscount[0] = 4;  
 curvepointscount[0] = n + 1;  
 LineStripArray contralpointslines = new  
 LineStripArray(4, LineArray.COORDINATES, contralpointscount);  
 contralpointslines.setCoordinates(0, contralpoints);  
 LineStripArray curvepointslines = new  
 LineStripArray(100, LineArray.COORDINATES, curvepointscount);  
 curvepointslines.setCoordinates(0, curvepoints0);  
 //设置线的属性  
 LineAttributes lineattributes = new LineAttributes();  
 lineattributes.setLineWidth(4.0f);  
 lineattributes.setLineAntialiasingEnable(true);  
 lineattributes.setLinePattern(0);  
 Appearance app = new Appearance();  
 app.setLineAttributes(lineattributes);  
 //针对整个曲线定义颜色，而不是针对顶点定义颜色  
 ColoringAttributes color = new ColoringAttributes();  
 color.setColor(1.f, 0.f, 0.f);  
 app.setColoringAttributes(color);  
 this.addGeometry(curvepointslines);  
 this.addGeometry(contralpointslines);  
 this.setAppearance(app);  
 }  
}  
  
//下面是实现两矩阵相乘的类  
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];  
 }  
 }  
 }  
}

（2）结果截图

|  |  |
| --- | --- |
|  |  |
| 图4.1三次Bezier曲线及控制多边形 | 图4.2对u∈[0，1]区间5等分的三次Bezier曲线 |

## 实验内容四：

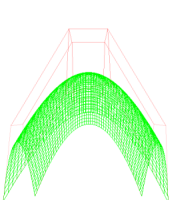
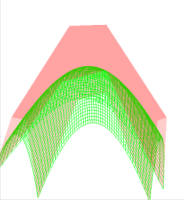
1. 代码

package week10\_sixth.\_4\_2;  
  
/\*\*  
 \* \\* Created with IntelliJ IDEA.  
 \* \\* @ProjectName: Computer graphics  
 \* \\* @FileName: BezierThreeOrderSurface  
 \* \\* @author: li-jihong  
 \* \\* Date: 2023-11-02 10:52  
 \*/  
  
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.\*;  
  
public class BezierThreeOrderSurface extends Applet {  
 public BezierThreeOrderSurface() {  
 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 BezierThreeOrderSurface(), 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);  
 TransformGroup transformgroup = new TransformGroup();  
 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Bezier曲面16个控制顶点的坐标  
 float[][][] P1 = {{{-0.8f, -1.2f, -0.8f, 1.f},  
 {-0.2f, 0.2f, -0.5f, 1.f},  
 {0.2f, 0.3f, -0.5f, 1.f},  
 {0.8f, -1.2f, -0.8f, 1.f}},  
 {{-0.8f, -0.1f, -0.2f, 1.f},  
 {-0.2f, 0.9f, -0.2f, 1.f},  
 {0.2f, 0.9f, -0.2f, 1.f},  
 {0.8f, -0.1f, -0.2f, 1.f}},  
 {{-0.8f, -0.1f, 0.2f, 1.f},  
 {-0.2f, 0.9f, 0.2f, 1.f},  
 {0.2f, 0.9f, 0.2f, 1.f},  
 {0.8f, -0.1f, 0.2f, 1.f}},  
 {{-0.6f, -0.6f, 0.9f, 1.f},  
 {-0.2f, 0.2f, 0.5f, 1.f},  
 {0.2f, 0.3f, 0.5f, 1.f},  
 {0.6f, -0.65f, 0.8f, 1.f}}};  
 //指定Bezier曲面外观属性  
 Appearance app1 = new Appearance();  
 PolygonAttributes polygona1 = new PolygonAttributes();  
 polygona1.setCullFace(PolygonAttributes.CULL\_NONE);  
 //polygona1.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  
 app1.setPolygonAttributes(polygona1);  
 ColoringAttributes color1 = new ColoringAttributes();  
 color1.setColor(0.f, 1.f, 0.f);  
 app1.setColoringAttributes(color1);  
 //指定Bezier曲面控制顶点网格的外观属性  
 Appearance app2 = new Appearance();  
 PolygonAttributes polygona2 = new PolygonAttributes();  
 polygona2.setCullFace(PolygonAttributes.CULL\_NONE);  
 //polygona2.setPolygonMode(PolygonAttributes.POLYGON\_LINE);  
 app2.setPolygonAttributes(polygona2);  
 ColoringAttributes color2 = new ColoringAttributes();  
 color2.setColor(2.f, 0.f, 0.f);  
 app2.setColoringAttributes(color2);  
 TransparencyAttributes transparence = new TransparencyAttributes(1, .8f);  
 app2.setTransparencyAttributes(transparence);  
 Shape3D BezierSurfaceface1 = new BezierThreeOrderSurfaceface(P1, app1);  
 transformgroup.addChild(BezierSurfaceface1);  
 Shape3D BezierControlPoints1 = new BezierSurfaceControlPoints(P1, app2);  
 transformgroup.addChild(BezierControlPoints1);  
 BranchGroupRoot.compile();  
 return BranchGroupRoot;  
 }  
}  
  
//计算Bezier曲面上的点，并生成Bezier曲面  
class BezierThreeOrderSurfaceface extends Shape3D {  
 public BezierThreeOrderSurfaceface(float[][][] P, Appearance app) {  
 int i, j, k;  
 int n0;//定义参数u、v在[0，1]区间的等分点数  
 float division;//定义参数u、v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}};  
 //定义存放Bezier曲面u、v参数分割点的坐标数组  
 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];  
 }  
 //计算曲面上所有点的坐标  
 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，在该程序中，第4维坐标全为1，可不计算  
 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，如果第四维为1，可不除该项  
 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 BeziersurfacecontrolPointsNet = new QuadArray(n0 \* n0 \* 4,  
 GeometryArray.COORDINATES | GeometryArray.NORMALS);  
 int c = 0;//以顶点数累加的方式设置数组中顶点的序号  
 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]);  
 //计算四个点的法向量  
 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  
 BeziersurfacecontrolPointsNet.setCoordinate(c, A);  
 BeziersurfacecontrolPointsNet.setCoordinate(c + 1, B);  
 BeziersurfacecontrolPointsNet.setCoordinate(c + 2, C);  
 BeziersurfacecontrolPointsNet.setCoordinate(c + 3, D);  
 //按序号设置点的法向量  
 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);  
 }  
}  
  
//生成控制顶点网格  
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];  
 }  
 }  
 }  
}

（2）结果截图

|  |
| --- |
|  |

修改代码查看线模型：

**三、实验总结**

**实验一总结：**

1.创建一个简单的三维场景，其中包括一个旋转的表面，这个表面由多个四边形组成。

2.设置了背景颜色，以及一个方向光源用于照明。

3.实现了鼠标交互功能，包括旋转、平移和缩放，以便用户可以通过鼠标与三维对象互动。

4.创建了一个自定义的'SurfaceDisplay'类，用于定义表面的几何形状和外观，包括设置多边

形属性和材质颜色。

**实验二总结：**

在本次实验中，实践了一个旋转体的计算和显示程序和实践了一个球面三角网格与四边网格程序显示实例，进一步加深了旋转体设计的原理的理解，自己同时通过局部代码的修改，来查看两个实例的各种模型，逐步了解模型建立的步骤。当然也遇到一些小问题，比如代码打错，容易忘记设置高亮，但在后续都能通过自己的检查修改成功。

1.创建一个简单的三维场景，其中包括一个球体的表面，该表面由不同类型的几何数组表示。

2.设置了背景颜色，以及一个方向光源用于照明。

3.实现了鼠标交互功能，包括旋转、平移和缩放，以便用户可以通过鼠标与三维对象互动。

4.创建了两个自定义的'SphereTriangleArrayDisplay'和`SphereQuadArrayDisplay'类，分别表示球体表面的三角面和四边形面，并设置了外观属性，如颜色和光照。

**实验三总结：**

1. `BezierThreeOrderCurve` 类继承自 `Applet` 类，用于创建和展示三阶贝塞尔曲线的三维可视化。它包括了一个简单的三维场景，其中包含了贝塞尔曲线的可视化对象。

2. `createBranchGroupSceneGraph` 方法创建了场景图（`BranchGroup`），包括了背景、光照、鼠标交互等元素。它还创建了贝塞尔曲线的可视化对象，并将其添加到场景图中。

3. `BezierThreeOrderCurve0` 类继承自 `Shape3D` 类，表示了三阶贝塞尔曲线。它通过计算贝塞尔曲线上的点的坐标，以及绘制控制顶点的线条来创建曲线的可视化。这个类包括以下关键步骤：

- 计算参数 `u` 在 `[0, 1]` 区间的等分点数和线段长度。

- 使用贝塞尔曲线的控制点坐标，计算贝塞尔曲线上每个参数 `u` 对应的坐标。

- 将计算得到的曲线上的点的坐标以及控制点的坐标，转换为 `Point3f` 类型的数组，并创建线条数组表示控制点和曲线。

- 设置线条的属性，包括线宽、抗锯齿和颜色。

- 最后，将这些几何对象添加到 `BezierThreeOrderCurve0` 的外观中。

4. `matrixm` 类是一个用于两矩阵相乘的辅助类，用于计算矩阵相乘的结果。

总体来说，这段代码实现了一个三阶贝塞尔曲线的可视化，并包括了控制点和曲线的绘制。通过 Java 3D 库，你可以在三维空间中创建和呈现这样的曲线，了解如何实现鼠标交互，以及如何使用外观属性来设置线条的样式和颜色。这是一个用于学习三维图形可视化的基本示例。

**实验四总结：**

这段Java代码实现了一个三阶贝塞尔曲面（Bezier Three-Order Surface）的可视化。以下是代码的主要结构和功能：

1. `BezierThreeOrderSurface` 类继承自 `Applet` 类，用于创建和展示三阶贝塞尔曲面的三维可视化。它包括了一个简单的三维场景，其中包含了贝塞尔曲面的可视化对象。

2. `createBranchGroupSceneGraph` 方法创建了场景图（`BranchGroup`），包括了背景、光照、鼠标交互等元素。它还创建了贝塞尔曲面的可视化对象和控制顶点的网格，并将它们添加到场景图中。

3. `BezierThreeOrderSurfaceface` 类继承自 `Shape3D` 类，表示了三阶贝塞尔曲面。它通过计算曲面上的点的坐标以及点的法向量来创建曲面的可视化。这个类包括以下关键步骤：

- 计算参数 `u` 和 `v` 在 `[0, 1]` 区间的等分点数和线段长度。

- 使用贝塞尔曲面的控制点坐标，计算曲面上每个参数 `u` 和 `v` 对应的坐标。

- 计算曲面上点的法向量，以便渲染阴影和光照效果。

- 创建 `QuadArray` 以表示曲面的网格，并将其中的坐标和法向量设置为曲面上点的值。

- 设置曲面的外观属性，包括颜色和抗锯齿。

4. `BezierSurfaceControlPoints` 类用于创建控制顶点的网格，类似于 `BezierThreeOrderSurfaceface` 类，但它只绘制控制顶点而不计算法向量。

5. `matrixm` 类是一个用于两矩阵相乘的辅助类，用于计算矩阵相乘的结果。

总体来说，这段代码实现了一个三阶贝塞尔曲面的可视化，包括控制顶点和曲面网格的绘制。这是一个用于学习三维图形可视化的高级示例，涵盖了曲面计算、鼠标交互、光照、法向量计算等方面。通过 Java 3D 库，你可以在三维空间中创建和呈现这样的曲面，用于学习计算机图形学的基本概念和技术。