《计算机图形学基础》

习题课3

助教 曹耕晨

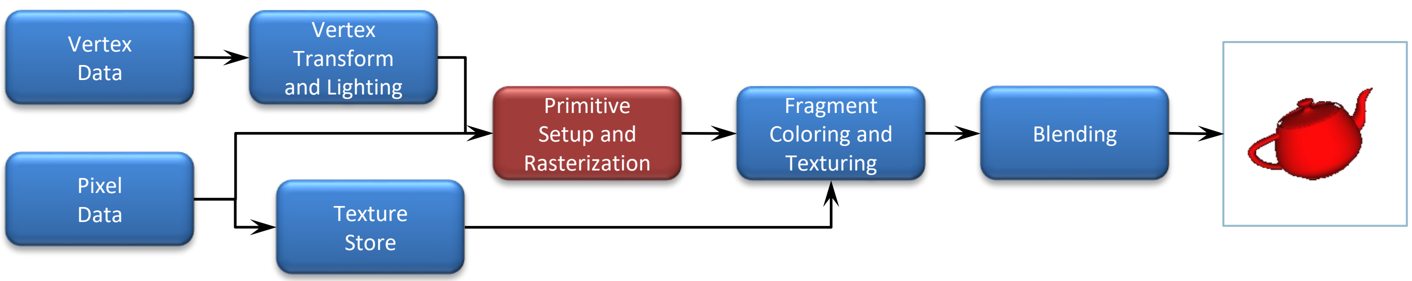
2022年4月17日

OpenGL是什么?

- OpenGL是一个图形学渲染API
  - 使用OpenGL渲染几何/图像元素，能够生成高质量照片
  - 是许多包含3D的交互应用程序的基础
  - 相同的代码可以在不同的操作系统共通

起源

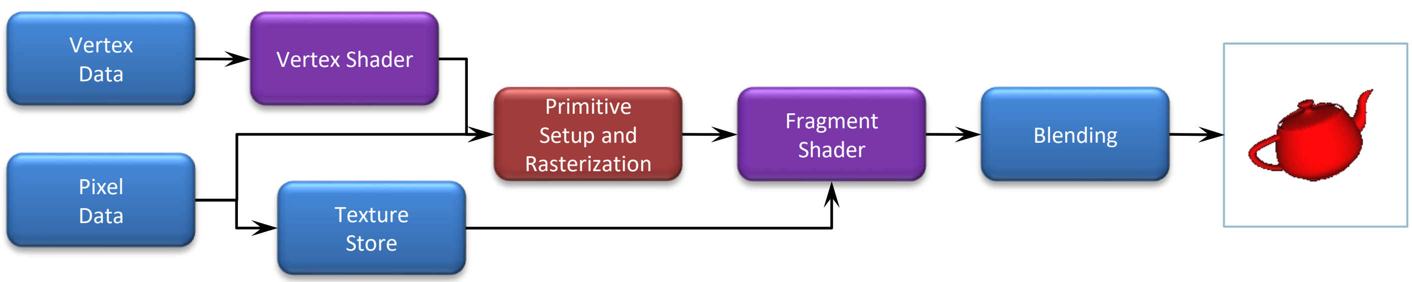
- OpenGL 1.0于1994年7月份发布
- 其渲染管线大部分都是固定的



- 这种固定管线模式一直从OpenGL版本1.1 持续到2.0 (2004年9月)

可编程渲染管线

- 从OpenGL 3.1开始，固定管线模式被移除
  - 自此版本之后，OpenGL渲染管线需要使用着色器（Shaders）编写



如何理解 OpenGL

OpenGL 自身是一个巨大的状态机，状态机中包含了一系列变量，这个状态通常被称为“上下文”（Context）。当调用了状态设置函数之后，所有之后执行的绘制指令都会依据当前的状态

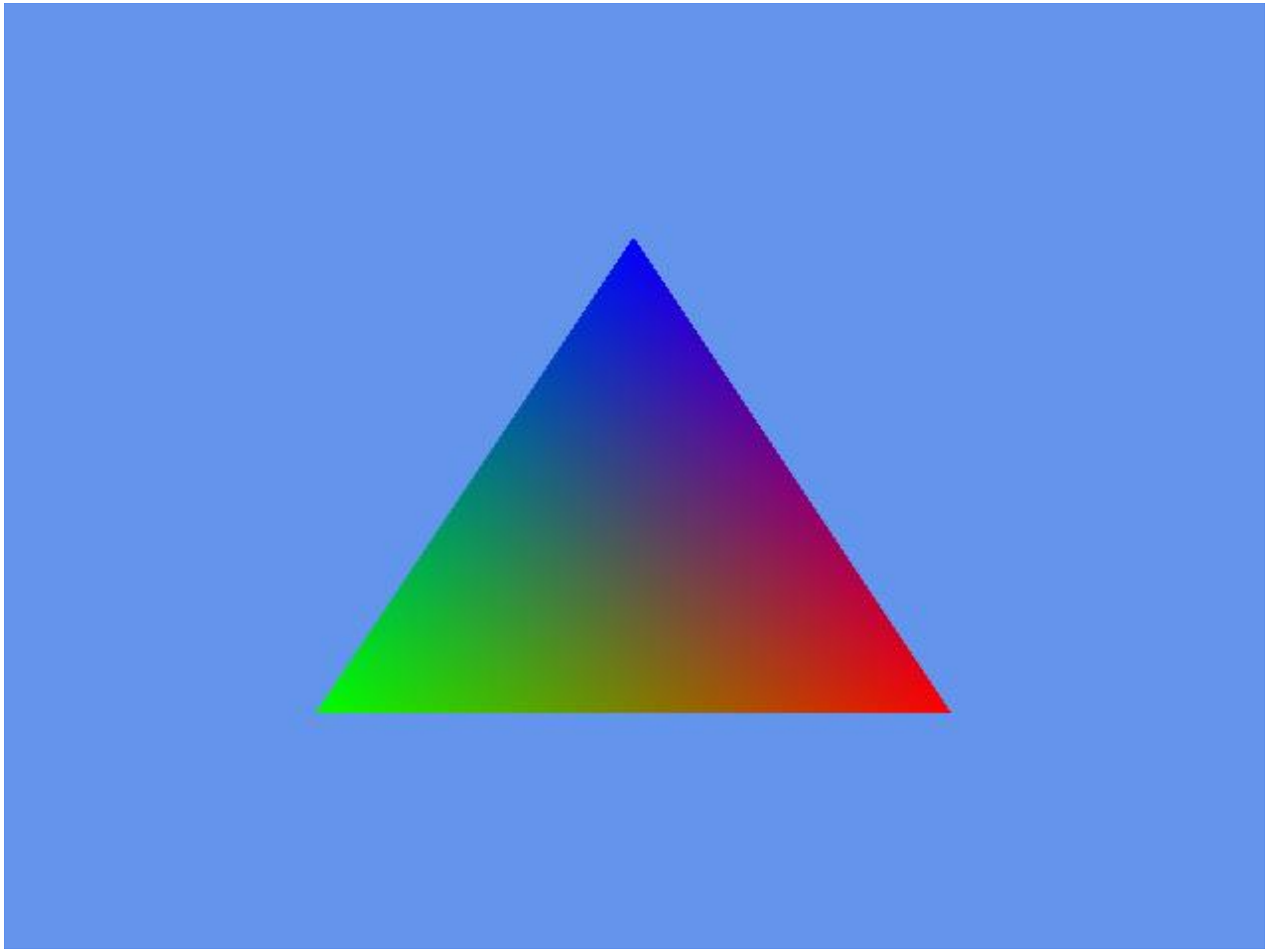
一个简单的OpenGL程序

```
int main(int argc, char** argv) {
    // 初始化GLUT，它负责创建OpenGL环境以及一个GUI窗口
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(60, 60);
    glutInitWindowSize(640, 480);
    glutCreateWindow("PA2 Immediate Mode");
    // 设置绘制函数为render()
    glutDisplayFunc(render);
    // 开始UI主循环
    glutMainLoop();
    return 0;
}
```

一个简单的OpenGL程序

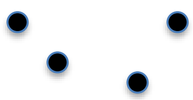
```
// 编译选项: g++ main.cpp -o main -lglut -lGL
#include <GL/glut.h>

void render() {
    // 设置背景色: 矢车菊蓝
    glClearColor(0.392, 0.584, 0.930, 1.0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    // 立即模式中的绘制以glBegin和glEnd包裹
    // GL_POINTS: 绘制点
    // GL_LINES: 绘制线段
    // GL_TRIANGLES: 绘制三角形
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0.0); glVertex2f(0.5, -0.5);     // 红
    glColor3f(0.0, 1.0, 0.0); glVertex2f(-0.5, -0.5);    // 绿
    glColor3f(0.0, 0.0, 1.0); glVertex2f(0.0, 0.5);      // 蓝
    glEnd();
    // 渲染图片
    glFlush();
}
```

一个简单的OpenGL程序

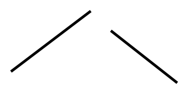


OpenGL's Geometric Primitives

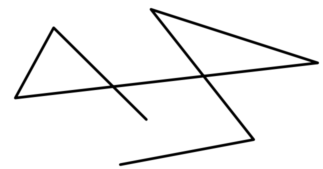
- All primitives are specified by vertices



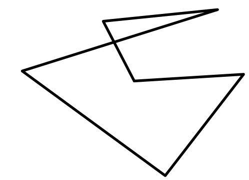
GL_POINTS



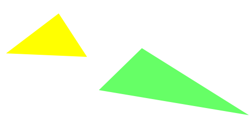
GL_LINES



GL_LINE_STRIP

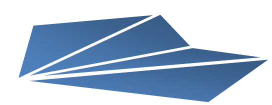


GL_LINE_LOOP



GL_TRIANGLES

GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

PA2代码框架 (main)

```
// Initialize GLUT
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowPosition(60, 60);
glutInitWindowSize(cam->getWidth(), cam->getHeight());
glutCreateWindow("PA2 OpenGL");

// Depth testing must be turned on
glEnable(GL_DEPTH_TEST);
// Enable lighting calculations
glEnable(GL_LIGHTING);
// In case for non-uniform transform.
glEnable(GL_NORMALIZE);

glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

// Set up callback functions for mouse
glutMouseFunc(mouseFunc);
glutMotionFunc(motionFunc);

// Set up the callback function for resizing windows
glutReshapeFunc(reshapeFunc);

// Call this whenever window needs redrawing
glutDisplayFunc(drawScene);

// Main UI Loop. This never returns.
glutMainLoop();

return 0;
```

PA2代码框架 (main)

```
//  Called when mouse button is pressed.
void mouseFunc(int button, int state, int x, int y) {
    if (state == GLUT_DOWN) {

        switch (button) {
            case GLUT_LEFT_BUTTON:
                cameraController->mouseClick(CameraController::LEFT, x, y);
                break;
            case GLUT_MIDDLE_BUTTON:
                cameraController->mouseClick(CameraController::MIDDLE, x, y);
                break;
            case GLUT_RIGHT_BUTTON:
                cameraController->mouseClick(CameraController::RIGHT, x, y);
            default:
                break;
        }
    } else {
        cameraController->mouseRelease(x, y);
    }
    glutPostRedisplay();
}

// Called when mouse is moved while button pressed.
void motionFunc(int x, int y) {
    cameraController->mouseDrag(x, y);
    glutPostRedisplay();
}
```

PA2代码框架 (main)

```cpp
// This function is responsible for displaying the object.
void drawScene() {
    Vector3f backGround = sceneParser->getBackgroundColor();
    glClearColor(backGround.x(), backGround.y(), backGround.z(), 1.0);

    // Clear the rendering window
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    // Setup MODELVIEW Matrix
    sceneParser->getCamera()->setupGLMatrix();

    // Turn On all lights.
    for (int li = 0; li < sceneParser->getNumLights(); ++li) {
        Light *light = sceneParser->getLight(li);
        light->turnOn(li);
    }
    // Draw elements.
    Group *baseGroup = sceneParser->getGroup();
    baseGroup->drawGL();

    // Dump the image to the screen.
    glutSwapBuffers();

    // Save if not in interactive mode.
    if (!savePicturePath.empty()) {
        screenCapture();
        exit(0);
    }
}
```

PA2代码框架 (light)

```
class PointLight : public Light {
public:
    PointLight() = delete;

    PointLight(const Vector3f &p, const Vector3f &c) {
        position = p;
        color = c;
    }

    ~PointLight() override = default;

    void getIllumination(const Vector3f &p, Vector3f &dir, Vector3f &col) const override {
        // the direction to the light is the opposite of the
        // direction of the directional light source
        dir = (position - p);
        dir = dir / dir.length();
        col = color;
    }

    void turnOn(int idx) const override {
        glEnable(GL_LIGHT0 + idx);
        glLightfv(GL_LIGHT0 + idx, GL_DIFFUSE, Vector4f(color, 1.0));
        glLightfv(GL_LIGHT0 + idx, GL_SPECULAR, Vector4f(color, 1.0));
        glLightfv(GL_LIGHT0 + idx, GL_POSITION, Vector4f(position, 1.0));
    }
```

PA2代码框架 (material)

```cpp
// TODO (PA2): Copy from PA1.
class Material {
public:

    explicit Material(const Vector3f &d_color, const Vector3f &s_color = Vector3f::ZERO, float s = 0) :
            diffuseColor(d_color), specularColor(s_color), shininess(s) {

    }

    virtual ~Material() = default;

    virtual Vector3f getDiffuseColor() const {
        return diffuseColor;
    }


    Vector3f Shade(const Ray &ray, const Hit &hit,
                   const Vector3f &dirToLight, const Vector3f &lightColor) {
        Vector3f shaded = Vector3f::ZERO;
        //
        return shaded;
    }

    // For OpenGL, this is fully implemented
    void Use() {
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, Vector4f(diffuseColor, 1.0f));
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, Vector4f(specularColor, 1.0f));
        glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, Vector2f(shininess * 4.0, 1.0f));
    }
```

PA2代码框架 (Sphere)

```
class Sphere : public Object3D {
public:
    Sphere() {
        // unit ball at the center
    }

    Sphere(const Vector3f &center, float radius, Material *material) : Object3D(material) {
        //
    }

    ~Sphere() override = default;

    bool intersect(const Ray &r, Hit &h, float tmin) override {
        return false;
    }

    void drawGL() override {
        Object3D::drawGL();
        glMatrixMode(GL_MODELVIEW); glPushMatrix();
        glTranslatef(center.x(), center.y(), center.z());
        glutSolidSphere(radius, 80, 80);
        glPopMatrix();
    }

protected:
    Vector3f center;
    float radius;

};
```

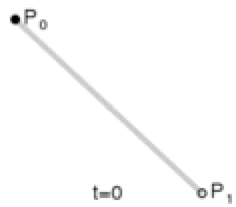
PA2代码框架 (plane)

```
class Plane : public Object3D {
public:
    Plane() {

    }

    Plane(const Vector3f &normal, float d, Material *m) : Object3D(m) {

    }

    ~Plane() override = default;

    bool intersect(const Ray &r, Hit &h, float tmin) override {
        return false;
    }

    void drawGL() override {
        Object3D::drawGL();
        Vector3f xAxis = Vector3f::RIGHT;
        Vector3f yAxis = Vector3f::cross(norm, xAxis);
        xAxis = Vector3f::cross(yAxis, norm);
        const float planeSize = 10.0;
        glBegin(GL_TRIANGLES);
        glNormal3fv(norm);
        glVertex3fv(d * norm + planeSize * xAxis + planeSize * yAxis);
        glVertex3fv(d * norm - planeSize * xAxis - planeSize * yAxis);
        glVertex3fv(d * norm + planeSize * xAxis - planeSize * yAxis);
        glNormal3fv(norm);
        glVertex3fv(d * norm + planeSize * xAxis + planeSize * yAxis);
        glVertex3fv(d * norm - planeSize * xAxis + planeSize * yAxis);
        glVertex3fv(d * norm - planeSize * xAxis - planeSize * yAxis);
        glEnd();
    }
```

样条曲线、曲面

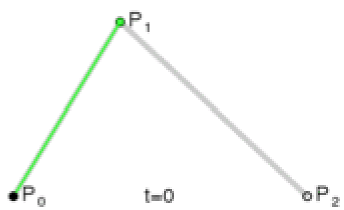
Bézier曲线/曲面

Linear Bézier



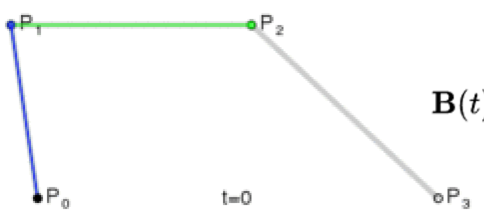
$$\mathbf{B}(t) = \mathbf{P}_0 + t(\mathbf{P}_1 - \mathbf{P}_0) = (1-t)\mathbf{P}_0 + t\mathbf{P}_1 \,, 0 \le t \le 1$$

Quadratic Bézier



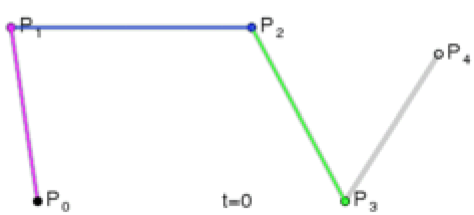
$$\begin{aligned}\mathbf{B}(t) &= (1-t)[(1-t)\mathbf{P}_0 + t\mathbf{P}_1] + t[(1-t)\mathbf{P}_1 + t\mathbf{P}_2] \,, 0 \le t \le 1, \\ &= (1-t)^2\mathbf{P}_0 + 2(1-t)t\mathbf{P}_1 + t^2\mathbf{P}_2 \,, 0 \le t \le 1.\end{aligned}$$

Cubic Bézier



$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3 \,, 0 \le t \le 1.$$

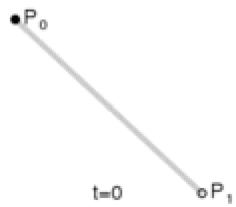
Quartic Bézier



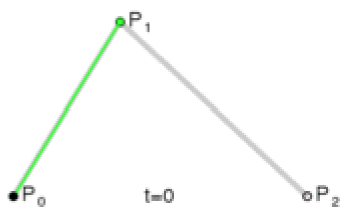
……

Bézier曲线/曲面

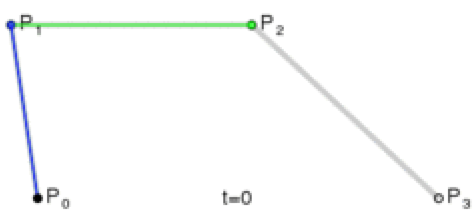
Linear Bézier



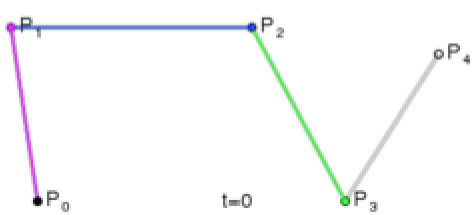
Quadratic Bézier



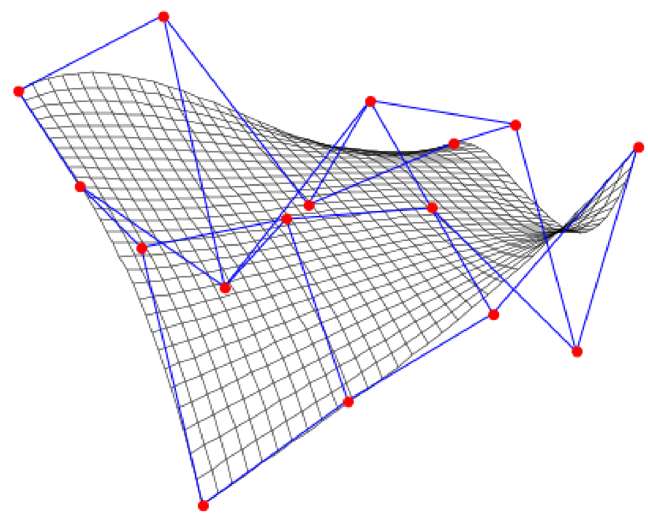
Cubic Bézier



Quartic Bézier



Bi-cubic Bézier

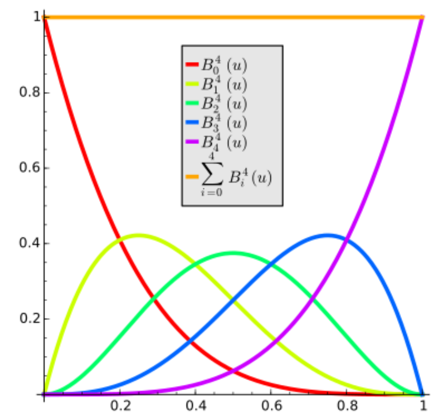


Bézier曲线造型的表示

- 参数曲线方程

$$\boldsymbol{P}(t)=\sum_{i=0}^{n}P_i\boldsymbol{B}_{\boldsymbol{i},\boldsymbol{n}}(t)$$

$$\boldsymbol{B}_{\boldsymbol{i},\boldsymbol{n}}(t)=C_n^i t^i(1-t)^{n-i}$$



其中C_n^i是大家高中熟知的排列组合常数，代表了n次多项式的第i个系数

$$C_n^i=\frac{n!}{i!\,(n-i)!}$$

Bézier曲线求导

- 参数曲线方程

$$\boldsymbol{P}(t)=\sum_{i=0}^{n}P_i\boldsymbol{B}_{\boldsymbol{i},\boldsymbol{n}}(t)$$

$$\boldsymbol{P}'(t)=n\sum_{i=0}^{n-1}\boldsymbol{B}_{\boldsymbol{i},\boldsymbol{n-1}}(t)(P_{i+1}-P_i)$$

- 大家可以自行尝试推导，或参考维基百科

Bézier曲线造型的表示

- 参数曲面方程（绕Z轴旋转）

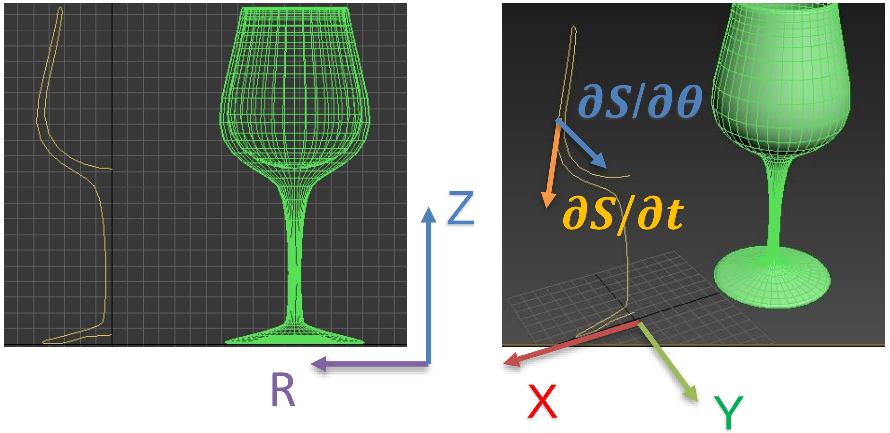
$$\boldsymbol{S}(\boldsymbol{t},\boldsymbol{\theta}) = \Phi(P(t),\theta) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \cdot P(t)$$

$$\boldsymbol{\Phi}\big((\boldsymbol{r},\boldsymbol{z}),\boldsymbol{\theta}\big) = [r\cos\theta \quad r\sin\theta \quad z]^{\mathrm{T}}$$

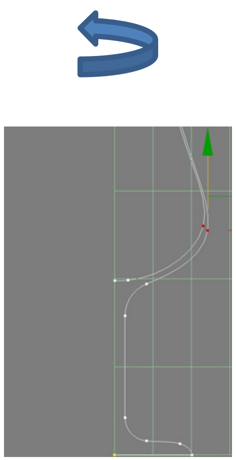
- 一阶雅各比：

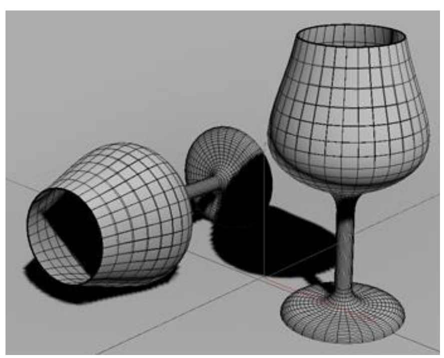
$$\frac{\boldsymbol{\partial S}}{\boldsymbol{\partial t}} = \frac{\partial\Phi(P(t),\theta)}{\partial P(t)} \cdot \frac{\partial P(t)}{\partial t}$$

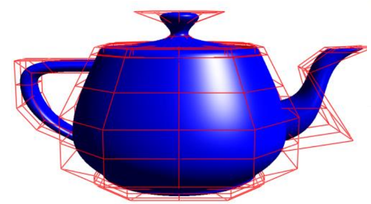
$$\frac{\boldsymbol{\partial S}}{\boldsymbol{\partial \theta}} = \frac{\partial\Phi(P(t),\theta)}{\partial\theta}$$

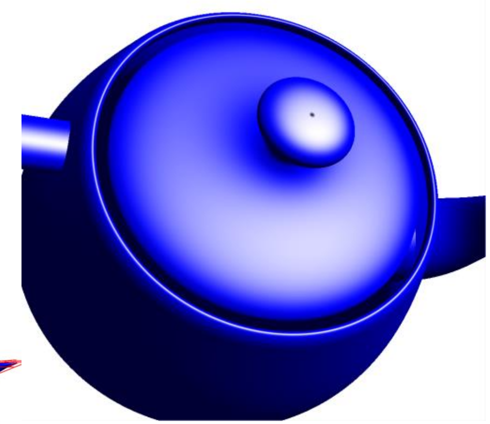


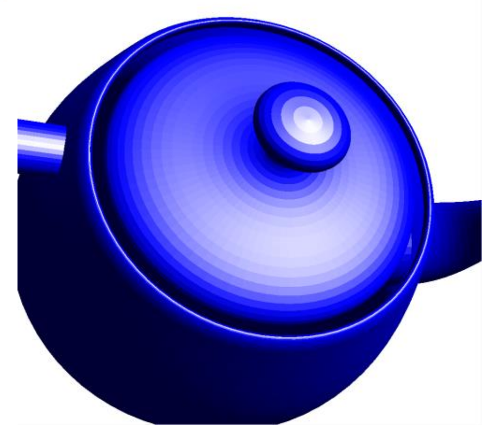
样条曲线/曲面造型举例



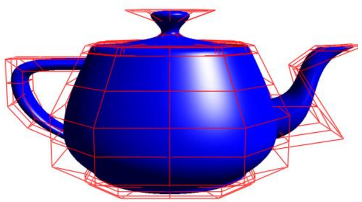
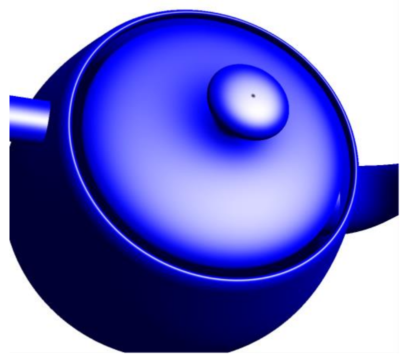
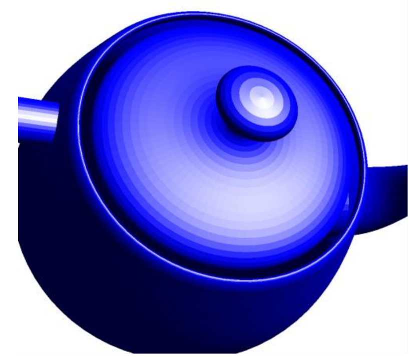
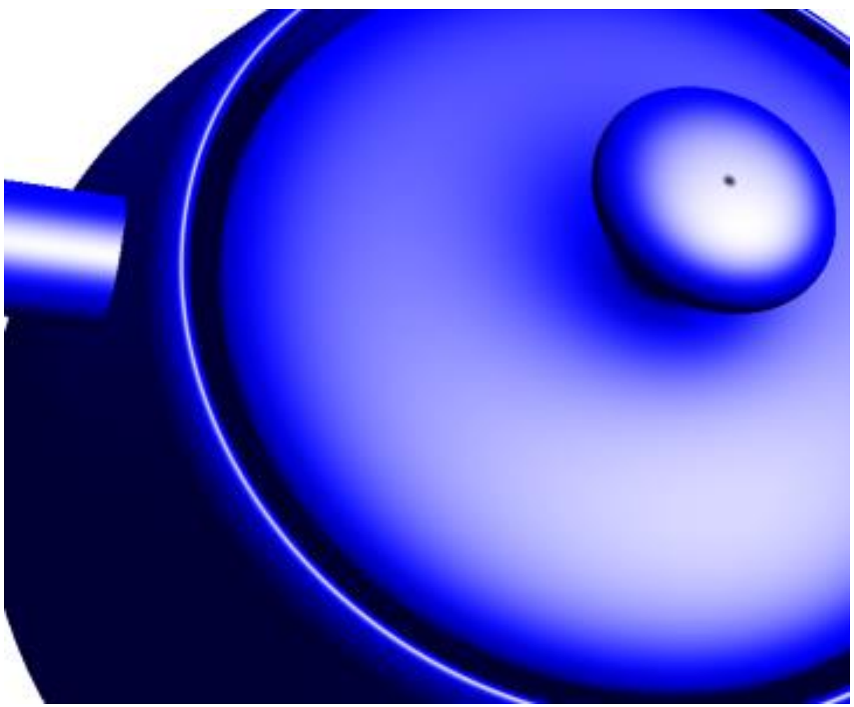
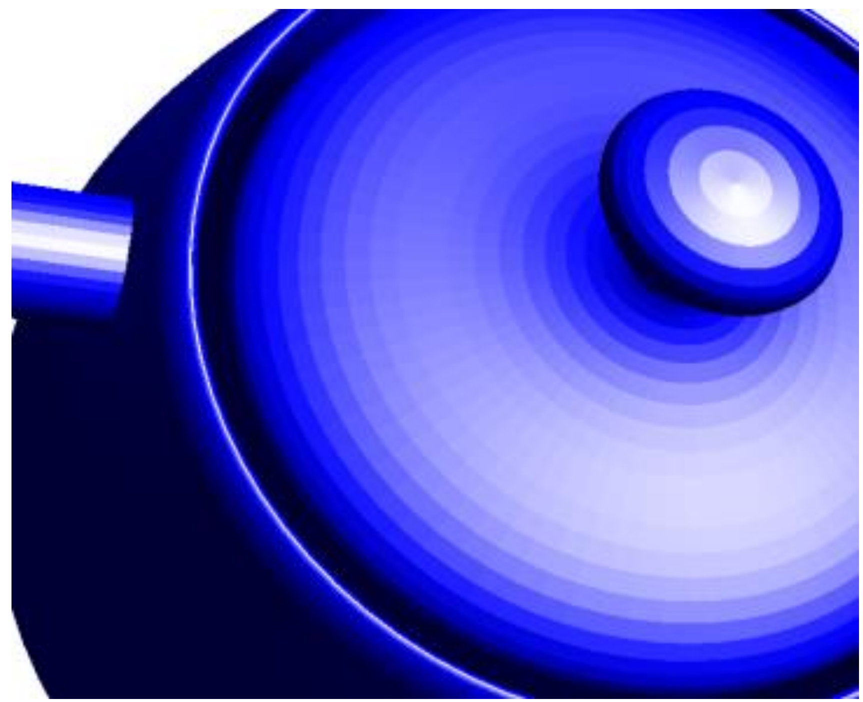






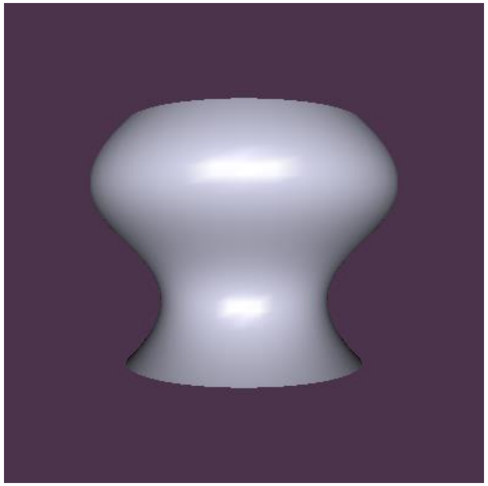


样条曲线/曲面造型举例

大作业附加要求

- 场景中包含参数曲面，可以为
  - 一维参数曲线加一个旋转参数
  - 二维参数曲面
- 曲面与射线求交采用数值解法



线面求交（基本）

- 光线：参数直线，最终交点必须满足 $t > 0$
- $L(t) = P + w \cdot t$

- 若L是最初产生的光线， P为视点， w 为视点发出的光线方向；
- 否则，P为交点， w根据反射，折射公式求得。

- 求交关键要求两个东西：
  - 交点
  - 交点处面的法向

线面求交（参数曲面，P107）

- 求交判断的加速：根据Bézier曲面凸包性，可以先判断与其构造点组成的凸包是否有交。
- 低次曲面的求交，可以选择直接解方程：

$$F(x)=F(t,u,v)\triangleq L(t)-P(u,v)=0$$

- $F(x)=\begin{bmatrix}F_1(x)\\F_2(x)\\F_3(x)\end{bmatrix}=0$
- 涉及到三元非线性方程组的求解问题

线面求交（参数曲面）

- 高次曲面的求交，可以参考P107，也可以按照本页来求：

- 根据上式写成迭代方程，进行牛顿迭代：

$$x_{i+1} = x_i - [F'(x_i)]^{-1} \cdot F(x_i)$$

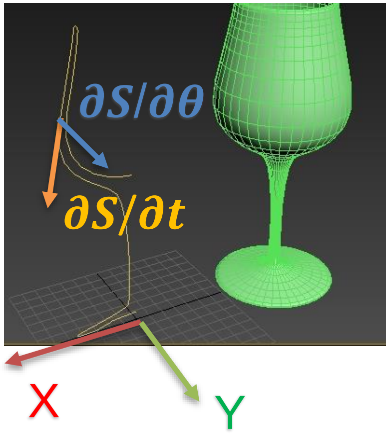
- 写出自己选用曲线的Jacobian（建议不要用数值Jacobian）：

$$F'(x) = \begin{bmatrix} \frac{\partial F_1}{\partial t} & \frac{\partial F_1}{\partial u} & \frac{\partial F_1}{\partial v} \\ \frac{\partial F_2}{\partial t} & \frac{\partial F_2}{\partial u} & \frac{\partial F_2}{\partial v} \\ \frac{\partial F_3}{\partial t} & \frac{\partial F_3}{\partial u} & \frac{\partial F_3}{\partial v} \end{bmatrix}$$

线面求交（参数曲面，P107）

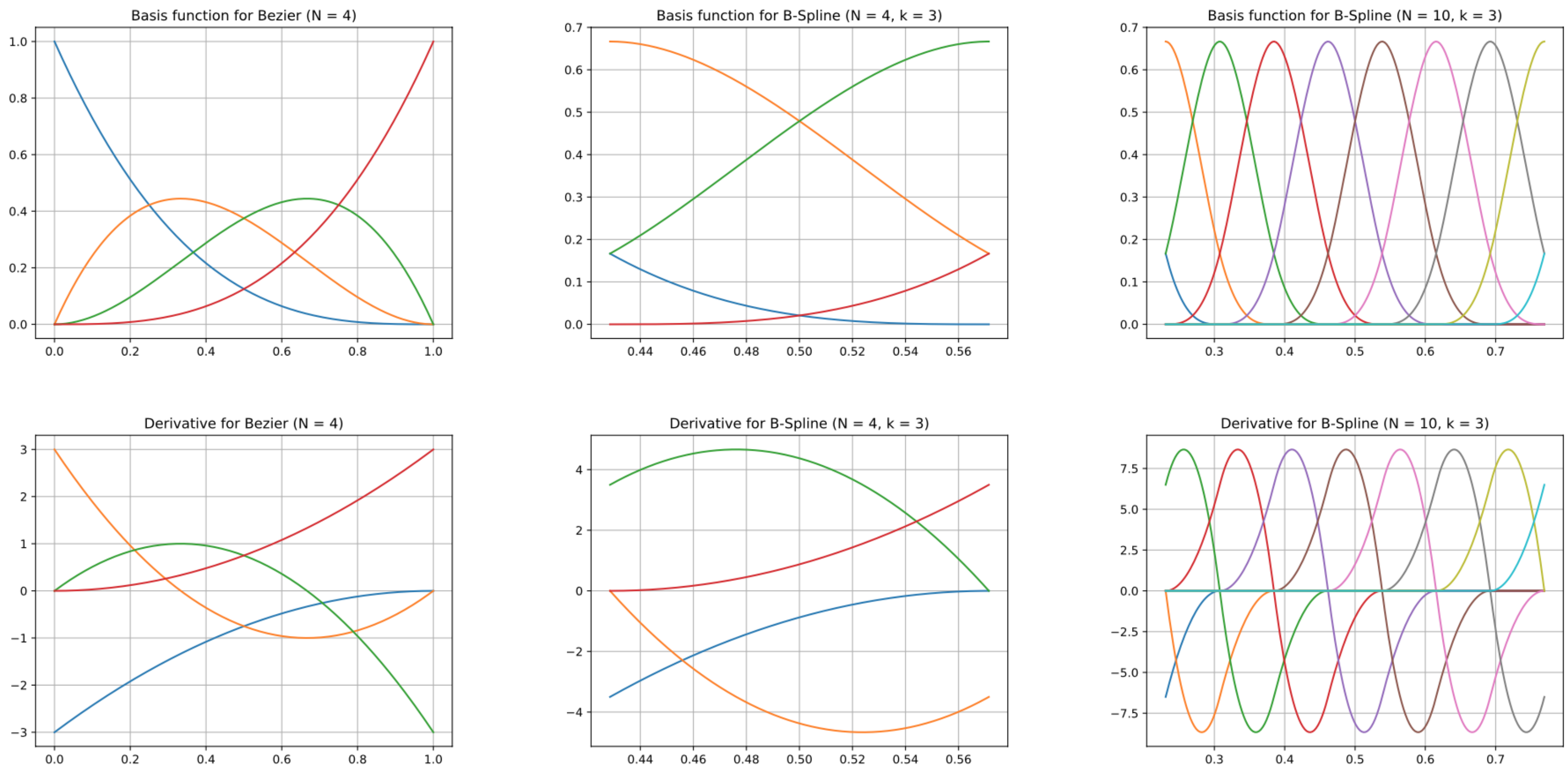
- 迭代的初值：光线和参数曲面四叉树叶子节点的包围盒求交。
- 迭代终止：步长过小或者超过最大次数。
  - 发现t, u, v越界，不收敛，作差不为0的情况，表示：求了半天，其实没交点。

- (t_i, u_i, v_i)处的法向：$n = \left(\frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v}\right)\Big|_{(t_i, u_i, v_i)}$



PA2 编程作业

PA2：De Boor Cox实现



PA2

- 需要实现curve.hpp里的两个discretize函数
- 即在有效参数区间内采样若干个t值并计算对应的曲线坐标P(t)以及该点的切向量P’(t)

- PA3采用OpenGL绘制，曲面根据离散化的采样点被划分为三角网格

PA2扩展

- 在revsurface.hpp中的实现intersect方法，在光线投射模式下进行数值求交

针对旋转面的特殊处理

- 曲面$\boldsymbol{S}(\boldsymbol{t},\boldsymbol{\theta}) = \Phi(P(t),\theta) = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \cdot P(t)$
- 样条曲线的二维坐标$P(t) = \big(x(t), y(t)\big)$
- 旋转面的三维坐标$\boldsymbol{S}(\boldsymbol{t},\boldsymbol{\theta}) = \big(x(t)\cos\theta, x(t)\sin\theta, y(t)\big)$
- 点$\boldsymbol{S}$在以(o_x, o_y, o_z)为起点，(d_x, d_y, d_z)为方向的射线上
- 故射线参数

$$t_r = \frac{x(t)\cos\theta - o_x}{d_x} = \frac{x(t)\sin\theta - o_y}{d_y} = \frac{y(t) - o_z}{d_z}$$

针对旋转面的特殊处理

- 射线参数

$$t_r = \frac{x(t)\cos\theta - o_x}{d_x} = \frac{x(t)\sin\theta - o_y}{d_y} = \frac{y(t) - o_z}{d_z}$$

- 整理得到

$$x(t)\cos\theta = \frac{(y(t) - o_z)d_x}{d_z} + o_x$$

$$x(t)\sin\theta = \frac{(y(t) - o_z)d_y}{d_z} + o_y$$

- 两式平方相加消去变量θ，最终得到一个只含t的方程

大作业附加要求

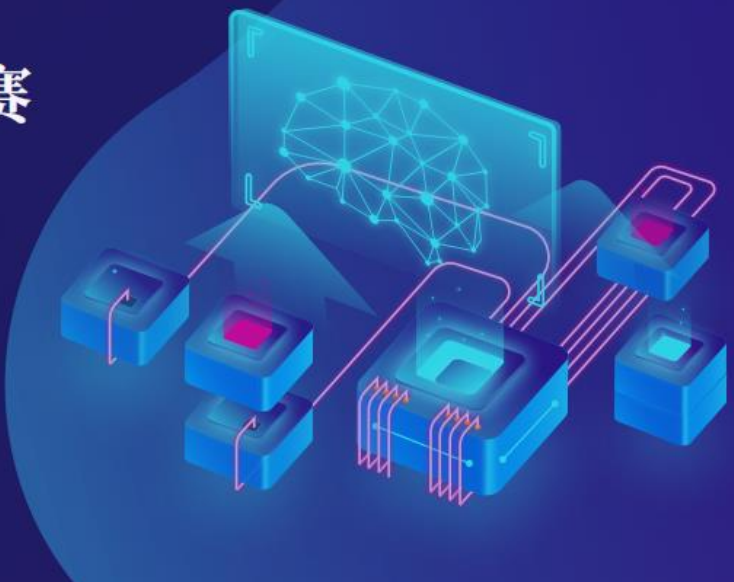
Jittor 计图

第五届中国软件开源创新大赛
第二届「计图 Jittor」人工智能挑战赛

指导机构 国家自然科学基金委信息科学部

主办单位 北京信息科学与技术国家研究中心

清华-腾讯互联网创新技术联合实验室



比赛介绍

- 比赛可替代光线追踪大作业。
- 可组队参数，1~3人一队，其中1人为队长。
- 评分标准
  - 队伍得分根据按照A榜截止时（6.30）比赛中的排名和成绩计算。
  - 队内得分按照贡献度略有偏差。

比赛链接

奖项设置

- 计图定制小礼品
- 两个正式赛题分别产生8个获奖团队，其中一、二、三等奖的数量如下。

名称	数量	奖金（税前）
一等奖	1项	5万元人民币+荣誉证书
二等奖	2项	2万元人民币+荣誉证书
三等奖	5项	1万元人民币+荣誉证书

- 腾讯招聘绿色通道
- **论文推荐资格**
- **专家委员会推荐信（留学/保研）**

Thank You！
Any Questions?