图形学实验 2：OpenGL 绘制

指导教师：胡事民　助教：方晓楠

2021 年 3 月 30 日

1　实验综述

本次作业将通过实验的方式向大家介绍图形学研究和应用中必不可少的一项技能:OpenGL 编程。你将学会调用 OpenGL 的库函数，实现基于硬件加速的三维场景绘制，渲染出和 PA1 中相同的图片。本次的框架代码和上次的代码在结构上保持基本不变，我们将主要在 7 个测例上评测你的代码。

2　细节说明

2.1　OpenGL 简介

OpenGL（Open Graphics Library）是一个跨编程语言、跨平台的编程图形程序接口（API），它将计算机的资源抽象称为一个个 OpenGL 的对象，对这些资源的操作抽象为一个个的 GL 指令。它相当于一个“桥梁”，沟通了图形软件和显示硬件（例如 GPU）：暴雪/育碧等厂商的游戏大作、广泛应用于工程领域的 CAD 软件、时下最新的 AR/VR 引擎等各类软件在绘制三维场景的时候都需要调用 OpenGL 定义的接口；而 Nvidia、AMD、Intel 等硬件生产厂商则需要生产能够适配这些接口标准的独立或集成显卡。

除了 OpenGL 之外，还有许多其他的图形绘制标准接口，例如微软的 DirectX 系列、Vulkan 等等，相比之下 OpenGL 适配平台更加广泛（支持几乎所有操作系统，包括安卓）、支持社区更完善（这意味着更完善的资料和技术支持），是图形学爱好者和开发者的首选。

2.2　状态机模型

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。例如：

```
// 绘制一条 (0,1) 到 (1,0) 的线段
void drawLine() {
  glBegin(GL_LINES); glVertex2f(0.0, 1.0); glVertex2f(1.0, 0.0); glEnd();
}
glColor3f(1.0, 0.0, 0.0);     // 设置绘制颜色为红色
drawLine();                   // 绘制红色线段
glColor3f(0.0, 1.0, 0.0);     // 设置绘制颜色为绿色
drawLine();                   // 同样的函数，此时绘制的线段为绿色
```

只要你记住 OpenGL 本质上是一个大状态机，就更容易理解他的大部分特性。有时候明明调用了绘制函数，屏幕上却显示一片黑，此时很有可能是上下文设置出现了问题（例如灯光错误、材质错误、相机参数错误等）。

2.3 简单三维绘制

使用 OpenGL 进行绘制可以非常简洁，你只需要通过程序告诉 OpenGL 需要绘制的三维物体形状，OpenGL 就能够自动处理消隐（深度测试）、染色、光栅化等操作。一个简单的 OpenGL 例程如下所示：

```
// 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();
}

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;
}
```

这段例程绘制的效果如图1所示。

为了绘制更加复杂的物体，我们使用 GLUT（全称为 OpenGL Utility Toolkit）库，该库调用 OpenGL 以及某些操作系统的底层函数，能够方便地实现复杂图形的绘制。

```
// 绘制一个半径为1.0，纵向切片为60，横向切片为80的实心球体
glutSolidSphere(1.0, 60, 80);
// 绘制一个边长为1.0的实心正方体
glutSolidCube(1.0);
// 绘制一个大小为1.0的犹他茶壶 (Wikipedia: Utah Teapot)
glutSolidTeapot(1.0);
// 绘制一个二十面体
glutSolidIcosahedron();
// 使用24号Times New Roman字体绘制字符S
glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, 'S');
```

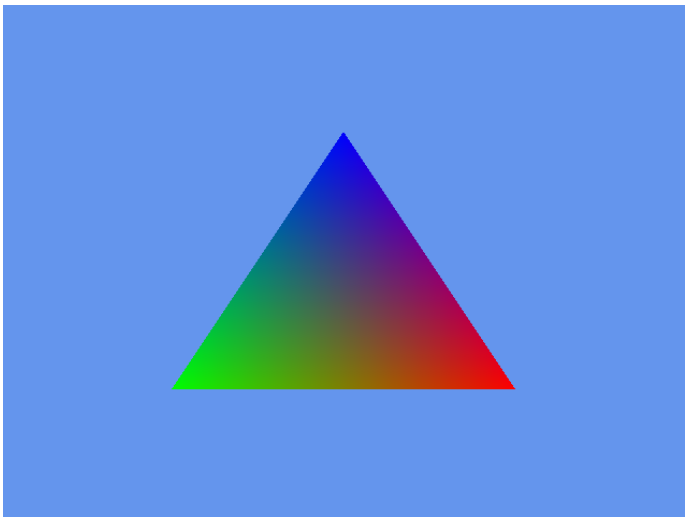

图 1: 立即模式绘制效果

除了绘制复杂物体之外，GLUT 还支持图形界面的创建和管理，例如创建窗口、处理鼠标键盘事件等，具体的使用示例请参见框架代码。

2.4 渲染模式和 OpenGL 版本

OpenGL 接口规范由Khronos组织维护，截至目前最新的版本为 4.6。早期的 OpenGL 使用立即渲染模式（Immediate mode），这种模式使用固定渲染管线：坐标变换、着色、光栅化等一系列操作均由标准定义好，使用起来非常方便。实际上，*2.3*节的简单绘制代码就是使用的这种模式，本次 *PA* 的所有代码也均以这种模式进行实现。立即渲染模式的缺点在于：开发者无法对渲染管线进行自定义的修改，大场景下的绘制的效率也比较低。[1]

随着时间的推移，从 OpenGL3.2 版本开始，立即渲染模式被废弃，开发者被鼓励于核心模式（Core-profile）下进行开发，这种模式是目前现代图形软件（包括手机等终端）所使用的主流绘制模式，具有效率高、灵活性强的特点。但是出于教学目的，本次的代码将不使用核心模式，如果同学们对现代 OpenGL 绘制感兴趣，可以参考LearnOpenGL以及其中文翻译版本。其中每一个教程都提供了代码和示例可供参考，是学习现代 OpenGL 的较好入门资料。

3 框架代码说明

3.1 环境配置与编译

我们推荐使用带有 `CMake` 套件的 `Ubuntu` 系统进行编程,`Windows 10` 下可以使用 `Ubuntu Subsystem`。如果你使用 `Ubuntu Subsystem`，你还需要安装 `Xming` 以显示图形界面[2]。在 `Windows` 系统下按照默认选项运行 `XLaunch`，然后在 `Ubuntu Subsystem` 下执行

```
export DISPLAY=:0
```

我们的框架代码依赖于 `GL` 和 `GLUT`，前者一般在安装显卡驱动的时候会自动配置好，如果 `CMake` 配置出错可以尝试使用 `apt` 安装 `mesa-common-dev`;后者请使用 `apt` 安装 `freeglut3-dev`。安装完成之后，请在包含有 `run_all.sh` 的文件夹下打开终端，并执行：

[1] 比你想象中的“低效”还是有所差距的，著名的《雷神之锤 II》就使用立即模式编写。

[2] https://xming.en.softonic.com/

```
bash ./run_all.sh
```

这段脚本会自动设置编译，并在 7 个测例上运行你的程序。你的程序最终会被编译到 bin/文件夹中。

程序的输入参数为场景文件名，该场景文件的定义和 PA1 是一模一样的。程序运行后会弹出窗口显示图像并进入交互模式，交互模式的操作说明如下（参考 CameraController 类）：

- 左键拖动：旋转
- 右键拖动：缩放
- 中间拖动：平移

关闭窗口即可结束程序。

对于使用 Windows 系统的同学，我们在 deps/文件夹中提供了 GLUT 库，你需要在 VC++ 项目中加入相应的包含目录、库目录和附加依赖项。

3.2 推荐实现步骤

本次作业的框架与 PA1 大体相同。Object3D 类中使用 drawGL 方法调用 OpenGL 进行绘制。Material 类中新增了 Use 方法用于设置材质。本次作业所有需要填写的地方都被标记了 TODO (PA2)，请全文查找该字样，每实现完成一个功能，你就可以去掉一个 TODO。大部分的 TODO 都是从 PA1 中拷贝代码，只有少部分 main 中的逻辑以及 mesh 中的函数需要自己实现。即便如此，我们也鼓励你通读代码并比较和 PA1 的异同，这对之后的编程作业会有帮助。

4 测试用例

为了测试代码是否正确无误，我们构建了以下 7 个测试用例（如图2-8所示），你也可以根据场景的文件格式构造样例进行自我测试。但我们在检查作业的时候将主要检查这 7 个样例的输出结果。

5 提交说明

请将你的代码放入 code 文件夹（无须包含可执行文件和输出结果），和一个 REPORT.pdf 文档一起打包成 zip 文件提交至网络学堂。具体文件树结构如下所示：[3]

[3]由于助教会使用自动化测试脚本来运行大家的程序，因此请务必遵循该文件树。文件夹包含 MacOS 生成的.DS_STORE 文件不影响评分。

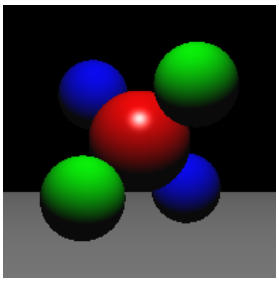

图 2: 五个球体

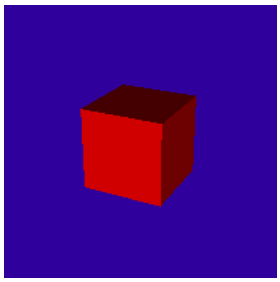

图 3: 正方体

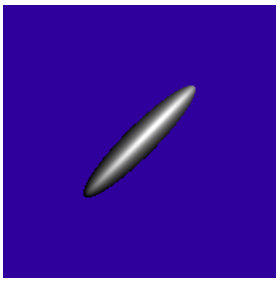

图 4: 变形球体

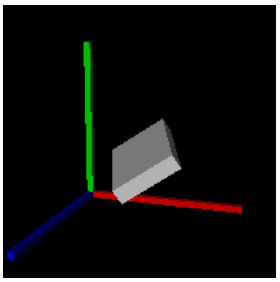

图 5: 坐标系

图 6: 简单兔子

图 7: 复杂兔子

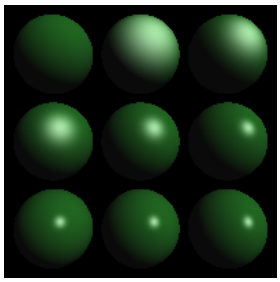

图 8: 镜面反射

```
姓名-学号-PA2.zip
├─ REPORT.pdf
├─ code
│   ├─ run_all.sh
│   ├─ CMakeLists.txt
│   ├─ src/
│   ├─ include/
│   └─ ...
└─ ... (其他你想放的文件)
```

在 `REPORT.pdf` 文档中主要回答以下问题：

- 你在 OpenGL 的环境配置中遇到了哪些问题？是怎么解决的？
- 结合核心代码分析使用 OpenGL 的绘制逻辑和光线投射的绘制逻辑有什么不同？
- 你在完成作业的时候和哪些同学进行了怎样的讨论？是否借鉴了网上/别的同学的代码？
- (可选) 你对本次编程作业有什么建议？文档或代码中有哪些需要我们改进的地方？

本次作业的 Deadline 以网络学堂为准。迟交的同学将得到一定的惩罚：晚交 3 天内分数将降低为 80%，3 天以上 1 周以内分数降为 50%，迟交一周以上的同学分数为 0。

在检查你的作业时，助教的自动化脚本会在 `code` 目录下运行 `run_all.sh` 文件，不能成功编译者可能会被酌情扣分。最后，欢迎同学们在压缩包中加入自己设计的场景 `txt` 文件，你的场景可能会被用于下学期的课堂作业教学中哦。

6 致谢

本实验文档和代码部分借鉴于MIT Open Courseware，按照其发布协议，本文档原则上允许同学们以CC BY-NC-SA 4.0协议共享引用，但是由于教学需要请同学们尽量不要将本文档或框架代码随意传播，感谢同学们的支持。