计算机图形学期末大作业
个人报告

学号	姓名
17341045	郭俊楠

1. 实现思路

在本项目中，我负责的是主函数以及辅助函数的编写，包括与用户的交互等内容。
总的来说，我们基于场景内容对整个项目进行了划分，要求每个人实现一些函数或接口，方便分工合作，而最后我们需要使用这些函数和接口控制整个应用程序的运行过程，提供交互功能。为了做到这一点，我们不妨将实现内容分为交互和绘制两部分，并为它们实现相应的辅助函数，之后在主函数中调用。这样，整个主函数就显得简单明了，各部分内容也能得到比较好的分离。
而关于交互的功能，我们也能将其分为两类：场景内容与观察方式。其中，场景内容主要包括物体的增添，而观察方式包括灯的开关和观察的角度等。为了切合Opengl的"状态机"设计，我们也用许多状态变量来控制我们的绘制过程，而通过控制这些变量，我们可以比较简单地实现交互功能。

2. 实现过程

下面，我们自顶向下地展示实现过程以及一些关键的代码。
关于主函数，我们需要实现整个程序的初始化和设置，包括回调函数注册、纹理生成、场景创建、开启渲染循环等。而在场景创建后，我们需要对其进行初始化，包括地板的创建和状态的初始化。

```
/*################################################
##   主函数，控制整个运行过程
##   参数：
##   返回值：
################################################*/
int main(int argc, char *argv[])
{
        cout << "-- Program starting\n";

        srand(time(NULL));
        init(argc, argv);

        // 注册回调函数
        cout << "-- Registering callback functions\n";
        glutDisplayFunc(drawScene);
```

```
        glutReshapeFunc(resizeWindow);
        glutKeyboardFunc(keyboardInput);
        glutSpecialFunc(keyboardInput);

        // 生成纹理
        cout << "-- Generating/Loading Textures\n";
        getTextures();

        // 创建场景
        scene = new Scene();
        scene->perspectiveMode = true;

        // 添加地板
        Quad *quad;
        for (GLfloat i = -9.5; i <= 9.5; i++)
        {
                for (GLfloat j = -9.5; j <= 9.5; j++)
                {
                        quad = new Quad();
                        quad->ry = 0.0f;
                        quad->rx = 90.0f;
                        quad->x = 3.5f * i;
                        quad->z = 3.5f * j;
                        quad->scale(3.5f, 3.5f, 1.0f);
                        scene->add(quad);
                }
        }

        // 初始化设置
        keyboardInput((unsigned char)'L', 0, 0);
        keyboardInput((unsigned char)'0', 0, 0);
        keyboardInput((unsigned char)'1', 0, 0);
        keyboardInput((unsigned char)'F', 0, 0);

        // 开启计时器
        glutTimerFunc(50, animator, 0);
        glutMainLoop();

        return 0;
}
```

在大致实现了主函数后，我们需要对辅助函数进行具体实现。 首先是窗口的缩放，这里要注意在缩放窗口的同时更新一些变量，比如场景类的宽度和高度。

```
/*########################################################
##   窗口缩放
##   参数：
##       w, h： 宽度，高度
##   返回值：
########################################################*/
```

```
void resizeWindow(int w, int h)
{
        glViewport(0, 0, w, h);

        // 更新场景的宽度和高度
        scene->width = w;
        scene->height = h;

        setupViewVolume();
}
```

关于交互部分，我们通过按键功能来实现，而实现方式主要是根据按键设置相应的变量。处理正常按键的函数代码如下，处理特殊按键的代码与此类似。

```
/*########################################################
##  正常按键处理函数，根据按键做出反应
##  参数：
##      key： 按键
##  返回值：
########################################################*/
void keyboardInput(unsigned char key, int x, int y)
{
    // 根据按键做出处理
        switch(key) {
        case 27:        // ESC，退出
            exit(0);
            break;

        case ' ':       // SPACE，Toggle flat/smooth shading
            flatShading = !flatShading;
            if (flatShading)
                glShadeModel(GL_FLAT);
            else
                glShadeModel(GL_SMOOTH);
            break;

        case 'A':
        case 'a':
            scene->camera.tiltdown();
            break;

        case 'Z':
        case 'z':
            scene->camera.tiltup();
            break;

        case 'W':
        case 'w':
            wireMode = !wireMode;
            if (!wireMode) {
```

```
                glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
                glDisable(GL_BLEND);
                glDisable(GL_LINE_SMOOTH);
            } else {
                glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
                glEnable(GL_BLEND);
                glEnable(GL_LINE_SMOOTH);
            }
            break;

        case 'P':
        case 'p':
            scene->perspectiveMode = !scene->perspectiveMode;
            setupViewVolume();
            break;

        case 'f':
        case 'F':
            scene->fogMode = !scene->fogMode;
            if (scene->fogMode) glEnable(GL_FOG);
            else glDisable(GL_FOG);
            break;

        case 'l':
        case 'L':
            scene->lightMode = !scene->lightMode;
            if (scene->lightMode) glEnable(GL_LIGHTING);
            else glDisable(GL_LIGHTING);
            break;

        case '0':
            scene->light0On = ! scene->light0On;
            if (scene->light0On) glEnable(GL_LIGHT0);
            else glDisable(GL_LIGHT0);
            break;

        case '1':
            scene->light1On = ! scene->light1On;
            if (scene->light1On) glEnable(GL_LIGHT1);
            else glDisable(GL_LIGHT1);
            break;
        }
}
```

接着是添加物体的辅助函数，实现的关键思路是根据物体类型创建物体并更新场景。值得注意的是，物体的初始化位置是随机的，而由于物体的高度不同，所以要注意对物体所处位置的高度做特殊的随机处理。

```
/*##########################################################
##  往场景里添加物体
##  参数：
```

```
##       type： 物体类型
##   返回值：
##########################################################*/
void addObject(int type)
{
        // 随机选择位置
        GLfloat x = Renderable::getRand(-25.0f, 50.0f);
        GLfloat z = Renderable::getRand(-25.0f, 50.0f);

        // 高度因物体而异
        GLfloat y;

        Renderable *object;

        switch (type)
        {
        case STARFISH:
            y = -0.3f;
            object = new StarFish();
            break;
        case CRAB:
            y = -0.4f;
            object = new Crab();
            break;
        case FISH:
            y = Renderable::getRand(-26.0f, 25.0f);
            object = new Fish();
            break;
        case OCTOPUS:
            y = Renderable::getRand(-27.0f, 25.0f);
            object = new Octopus();
            break;
        case PLANT:
            y = 0.0f;
            object = new Plant();
            object->ry = 0.0f;
            break;
        }

    // 更新位置
        object->move(x, y, z);
    // 加入场景
        scene->add(object);
    // 更新计数
        scene->objects[type]++;
}
```

下面是设置视景体的代码，正如前面提到的那般，我们使用许多状态变量来控制绘制流程，所以这里的视景体设置过程中需要实时计算一些参数，以切合当前的状态。

```
/*#####################################################
##  View Volume设置
##  参数：
##  返回值：
#####################################################*/
void setupViewVolume(void)
{
        // 计算比例
        GLfloat aspect = (GLfloat)scene->width / (GLfloat)scene->height;
        GLfloat iaspect = (GLfloat)scene->height / (GLfloat)scene->width;

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();

        // 根据当前状态进行设置
        if (scene->perspectiveMode == true)
                gluPerspective(-45.0f, aspect, 1.0f, 250.0f);
        else {
                if (aspect >= 1.0f)
                        glOrtho(-40.0f * aspect, 40.0f * aspect, -40.0f, 4(
                else
                        glOrtho(-40.0f, 40.0f, -40.0f * iaspect, 40.0f * ia
        }

        glMatrixMode(GL_MODELVIEW);
}
```

下面代码对Opengl进行了初始化设置。值得注意的是，这里需要设置的关键变量是不包含在用户交互功能的设置范围内的，所以需要进行初始化设置。

```
/*#####################################################
##  Opengl 初始化
##  参数：
##  返回值：
#####################################################*/
void setupGL(void)
{
        cout << "-- Setting up OpenGL state\n";

        // 蓝绿色背景
        glClearColor(0.0, 0.5, 0.55, 1.0);

        glShadeModel(GL_SMOOTH);
        glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);

        // 开启深度测试
        glDepthFunc(GL_LESS);
        glEnable(GL_DEPTH_TEST);
```

```
        // 设置雾
        glDisable(GL_FOG);
        glFogi(GL_FOG_MODE, GL_EXP);
        GLfloat fogColor[4] = {0.0f, 0.5f, 0.55f, 1.0f};
        glFogfv(GL_FOG_COLOR, fogColor);
        glFogf(GL_FOG_DENSITY, 0.0075);
        glHint(GL_FOG_HINT, GL_NICEST);

        glEnable(GL_NORMALIZE);

        // 设置灯光
        glDisable(GL_LIGHTING);
        GLfloat ambient[] = {0.1f, 0.1f, 0.1f, 1.0};
        glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambient);
        glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);

        // 混合设置
        glHint(GL_LINE_SMOOTH_HINT, GL_NICEST);
        glLineWidth(1.0f);
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

        // 面剔除
        glCullFace(GL_BACK);
        glEnable(GL_CULL_FACE);

        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST);
    }
```

3. 总结

本次项目中，我负责程序的主体运行部分，即主函数以及相应的辅助函数的编写。关于我所做的工作，从具体实现上来讲是比较容易的，因为经过前面几次编程作业，我对Opengl有了一定的了解，同时也比较熟悉一个交互式软件运行的流程；但从实现思路和规划来讲，难度还是比较大的，因为我不仅需要规划整个程序的运行逻辑，同时也要与队友统一接口，防止造成内容脱节，而遇到的大部分问题也是出自这块。
而从本次项目中，我了解了一个复杂场景的绘制流程，同时也学习了如何去添加交互式功能。在我看来，设计复杂场景时可以先考虑对场景里的物体进行抽象，规划好绘制流程；如果与他人进行合作实现，那么需要与队友讨论好接口，这样就不容易在交接的时候出错。