期末大作业个人报告

侯士燊 17341048 计算机类

根据我们小组的分工，我需要实现该项目的镜头转动(camera.cpp 和 camera.h)，一个水族馆中的小动物的实现(crab.cpp)和水族馆植被的实现(plant.cpp)

实验思路及代码讲解：

镜头的实现(camera.h 及 camera.cpp)：

为了实现我需要的镜头转动及拉近拉远的功能，我首先在 camera.h 文件中定义了一个 Camera 类

```
class Camera
{
private:
    GLfloat verticalTilt;
    GLfloat horizontalAngle;
    GLfloat distance;

public:
    Camera();    //默认构造函数
    virtual ~Camera();  //默认析构函数

    void reset(void);
    void position(void);
    void dec(void);
    void inc(void);
    void clockwise(void);
    void anticlockwise(void);
    void tiltup(void);
    void tiltdown(void);
};

#endif
```

它具有实现上述功能的镜头基本属性：X 轴的倾斜角(verticalTilt)，Y 轴的倾斜角(horizontalAngle)及距离原点的距离(distance)，但仅仅初始化了镜头实例是不够的，为了满足我们观测水族馆的需

求，我还根据实际定义了下面的函数：复位函数(reset)，改位函数(position)，拉近函数(inc)，拉远函数(dec)以及其他 4 个用来调整镜头在 X 轴和 Y 轴倾斜角的函数(X、Y 轴正反向各两个)。接下来我将讲讲我是如何实现上述函数进而实现一个简陋的镜头的(camera.cpp)

reset()

```
void Camera::reset(void)
{
    this->distance = -50.0f;
    this->verticalTilt = -30.0f;
    this->horizontalAngle = 90.0f;
}
```

只需要将镜头的三个属性复位至特定值即可

position()

```
void Camera::position(void)
{
    glTranslatef(0.0f, 0.0f, this->distance);
    glRotatef(this->verticalTilt, 1.0f, 0.0f, 0.0f);
    glRotatef(this->horizontalAngle, 0.0f, 1.0f, 0.0f);
}
```

根据当前镜头的属性将其调整至相应位置

inc()和 dec()

```
void Camera::dec(void)
{
    this->distance--;
}

void Camera::inc(void)
{
    this->distance++;
}
```

更改 camera 中的 distance 属性进而实现镜头的拉近拉远

```
void Camera::clockwise(void)
{
    this->horizontalAngle++;
}

void Camera::anticlockwise(void)
{
    this->horizontalAngle--;
}
```

更改 camera 中的 horizontalAngle 属性，进而实现其在 Y 轴的旋转

```
void Camera::tiltup(void)
{
    if (this->verticalTilt < 0)
        this->verticalTilt++;
}

void Camera::tiltdown(void)
{
    if (this->verticalTilt > -90)
        this->verticalTilt--;
}
```

更改 camera 中的 verticalTilt 属性，进而实现其在 X 轴的旋转

实现了上述函数后，只需要构建一个 camera 实例，再根据输入调用相应的函数即可实现镜头的功能。

绘制一只小螃蟹(crab.cpp):

水族馆中在地表的生物可选择的有不少，但是螃蟹却是其中让人印象最为深刻的种类之一，同时实现难度适中，整体上来说和实现作业 2 的机器人比较类似，而且这部分我是根据组员写好的定义文件(renderable.h)中螃蟹的定义进行相应功能实现的，因此我还是比较顺畅地完成了这部分代码。

```
GLfloat Crab::material[4] = {0.5f, 0.5f, 0.5f, 1.f};
GLfloat Crab::shininess = 50.f;
```

首先根据定义给静态变量赋初始值。

```
Crab::Crab() : Renderable()
{
    cout << "-- Creating crab\n";
    sy = sx = sz = 2.f; //让螃蟹大小为之前的两倍
    build(dlist);
}
```

根据 renderable.cpp 中的定义进行实现，而且，螃蟹和镜头不同，它应该是一个能够进行自定义添加的对象，为了和我组员的代码进行相应的对接和符合我们组内的需求，因此我这里需要调用一次 build(dlist)函数。

```
void Crab::_draw(void)
{
    //小红蟹的颜色
    glColor3f(1.0f, 0.45f, 0.45f);

    //画出螃蟹的身子
    glPushMatrix();
    glScalef(1.0f, 0.5f, 1.0f);
    gluSphere(quadric, 0.3f, 16, 16);
    glPopMatrix();

    //画出螃蟹全部的腿
    glPushMatrix();
    drawLegs();
    glScalef(-1.f, 1.f, 1.f);
    glFrontFace(GL_CW);
    drawLegs();
    glFrontFace(GL_CCW);
    glPopMatrix();

    //将颜色设置为黑色
    glColor3f(0.0f, 0.0f, 0.0f);

    //螃蟹的左眼
    glTranslatef(-0.06f, 0.0f, 0.3f);
    glutSolidSphere(0.05f, 12, 8);

    //螃蟹的右眼
    glTranslatef(0.12f, 0.0f, 0.0f);
    glutSolidSphere(0.05f, 12, 8);
}
```

接下来就是绘制螃蟹实例，先设定好螃蟹的颜色，然后绘制螃蟹的身子，调用函数绘制好后，调用提前定义并实现好的画腿函数(后续会提到的 drawLegs())绘制螃蟹的腿部，绘制好主体后，将颜色设置

为黑色，并接着绘制螃蟹的眼睛，这样，螃蟹实体就完成了。

```
void Crab::_draw_dlist(void)
{
    //设置材质属性（只需设置前视图）
    glMaterialfv(GL_FRONT, GL_SPECULAR, material);
    glMaterialf(GL_FRONT, GL_SHININESS, shininess);
    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
    glEnable(GL_COLOR_MATERIAL);

    glCallList(this->dlist);

    //跟踪色料
    glDisable(GL_COLOR_MATERIAL);
}
```

这个函数是设置螃蟹的材质属性，并绘制螃蟹对象属性的列表，因为我们希望实现的是可以自主添加实例的水族馆，因此需要写一个函数来实现该功能。

```
void Crab::drawLeg(GLfloat jointAngle, GLfloat jointOffset)
{
    //画出腿的第一个关节
    glPushMatrix();
    glTranslatef(-0.38f, 0.0f, 0.0f);
    glScalef(3.0f, 1.0f, 1.0f);
    glutSolidCube(0.06f);
    glPopMatrix();

    //画出腿的第二个关节
    glPushMatrix();
    glTranslatef(-0.53f, jointOffset, 0.0f);
    glRotatef(jointAngle, 0.0f, 0.0f, 1.0f);
    glScalef(4.0f, 1.0f, 1.0f);
    glutSolidCube(0.06f);
    glPopMatrix();
}
```

画腿函数，根据输入的参数，绘制螃蟹的一条腿

```
void Crab::drawLeg()
{
    drawLeg(-45.0f, 0.075f);
}
```

设置该函数的目的是为了无需设置参数就能默认地调用一次画腿函数，方便最后一个函数的调用

```
void Crab::drawLegs()
{
    //设置腿部的颜色（较深的粉红色）
    glColor3f(1.0f, 0.55f, 0.55f);

    //画出三条侧腿
    for (GLfloat i = -15.0f; i <= 15.0f; i += 15.0f)
    {
        glPushMatrix();
        glTranslatef(0.0f, 0.0f, -0.025f);
        glRotatef(i, 0.0f, 1.0f, 0.0f);
        drawLeg();
        glPopMatrix();
    }

    //画第四条腿（直的和弯曲的向下的腿）
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, -0.00f);
    glRotatef(-65.0f, -0.2f, 1.0f, 0.0f);
    drawLeg(0.0f, 0.0f);
    glPopMatrix();

    //设置前腿的颜色（浅粉色）
    glColor3f(1.0f, 0.65f, 0.65f);

    //绘制前足（前臂）
    glPushMatrix();
    glTranslatef(0.0f, 0.0f, 0.0f);
    glRotatef(55.0f, 0.0f, 1.0f, 0.0f);
    glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    drawLeg();
    glPopMatrix();

    //绘制前足（前臂）上的钳子
    glPushMatrix();
    glTranslatef(0.24f, 0.0f, 0.725f);
    glRotatef(-15.0f, 0.0f, 1.0f, 0.0f);
    glRotatef(90.0f, 1.0f, 0.0f, 0.0f);
    drawLeg(-60.0f, 0.1f);
    glPopMatrix();
}
```

画腿函数(画出全部的腿)，调用之前实现好的画单条腿函数并设置相关参数和调用相应变换，进而达到绘制出全部的腿的目的。

绘制植被(plant.cpp)

水族馆的地表除了沙子之外当然还得要有植被，可要是单纯地绘制一个植被实例再重复添加的话未免又过于单调，考虑到植被和其他实例不一样，它的每一层哪怕重复度略高也不会显得很奇怪，

因此我植被实现思路是先实现一个绘制一层植被的函数，然后经过随机函数获得随机的转向角度，并按照植物生长规律大致缩小下一层的大小，再在上一层基础上画出下一层植物，再辅以随机分叉数来实现大小不一，形状不一的水族馆底部植被。

```
//设置静态变量
GLfloat Plant::material1[4] = {0.1f, 0.3f, 0.15f, 1.f};
GLfloat Plant::material2[4] = {0.6f, 1.f, 0.8f, 1.f};
GLfloat Plant::shininess = 100.f;
```

还是先设置静态变量

```
Plant::Plant() : Renderable()
{
    cout << "-- Creating Plant\n";
    build(dlist);
}
Plant::~Plant()
{
    cout << "++ Destructing Plant\n";
}
```

默认的构造和析构函数

```
void Plant::_draw(void)
{
    //在一开始我们就生成一个至少有一个分支的植被，这样我们不会得到没有分支的植被
    generate(0, Renderable::getRand(1, 6));
}
```

绘制函数，用来生成随机分叉数的植被

```
void Plant::_draw_dlist(void)
{
    //设置材质属性（只需设置前视图）
    glMaterialfv(GL_FRONT, GL_AMBIENT, material1);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, material1);
    glMaterialfv(GL_FRONT, GL_SPECULAR, material2);
    glMaterialf(GL_FRONT, GL_SHININESS, shininess);
    glColor4fv(material1);

    glCallList(this->dlist);
}
```

绘制植被对象的显示列表

```
void Plant::generate(int level, int number)
{
    //到了第五层递归则不再递归
    if (level == 5) return;

    //每个分支数量都是上一个分支的3/4
    GLfloat height = 3.0f / (0.75 * (level + 1.f));
    GLfloat bottom = 0.75f / pow(2, level);
    GLfloat top = 0.75f / pow(2, (GLfloat)level + 1.f);

    for (int i = 0; i < number; i++)
    {

        GLfloat horzAngle = Renderable::getRand(0, 180);
        GLfloat vertAngle = Renderable::getRand(0, 180);
        int numChildren = Renderable::getRand(0, 6);

        glPushMatrix();
        glRotatef(horzAngle, 0.0f, 1.0f, 0.0f);
        glRotatef(vertAngle, 1.0f, 0.0f, 0.0f);

        //封底
        glFrontFace(GL_CW);
        gluDisk(quadric, 0.0f, bottom, 8 - level, 1);
        glFrontFace(GL_CCW);

        //画出分支
        gluCylinder(quadric, bottom, top, height, 8 - level, 1);

        //封顶
        glTranslatef(0.0f, 0.0f, height);
        gluDisk(quadric, 0.0f, top, 8 - level, 1);

        //调整角度回原来位置后再次调用生成函数产生分支
        glRotatef(-vertAngle, 1.0f, 0.0f, 0.0f);
        generate(level + 1, numChildren);
        glPopMatrix();
    }
}
```

生成一节植被的函数，每层植被的绘制都要先经历随机转向，接着封底，然后根据已经随机好的分叉绘制该层植被，最后封顶的过程。接着，因为需要画出好几层，因此需要递归调用该函数，为了防止画的层数太多或太少导致不真实，经过我多次实际调用后，发现层数为 5 时画出来的平均效果是最好的。

个人心得体会：

这次的实验中，因为是我牵头把人拉到小组来的，因此就顺理成章地成为了小组长，在负责部分代码的同时还顺便负责最后 ppt 的制作以及上台做报告。当时一开始报选题的时候是选择的小动画，但后来发现如果想要做的短一些就要把效果做的好一点，渲染的复杂些，做的长一些虽然可以在这方面放低要求，但又可能使工作量变得略大，甚至我们一度考虑过要不要更换选题，但最后有组员提供了这个类似以前 windows 待机页面的水族馆的想法，大家都觉得很不错，就沿着一直做下来了。不过有了思路还不够，把思路变为一个能够运行的项目对于我们这个不够成熟的小组来说显然还是较为困难的，这其中代码的部分因为有之前项目的经验，而且也能从网上查找相应的实现方法，所以还不算太过困难，但是如何把组员间负责的内容安排分工好且有序对接我们做出来的内容对我们来说无疑显得十分困难，但是好在大家都比较活跃，在群组上定时分享自己的进度，方便他人进行对接，并且讨论也多，最后我们才能够成功地完成这个项目。做完这个项目之后，我觉得最大的收获还不是代码能力上的提升，而是体验了一次项目任务分配以及各自实现自己部分导致不容易整合的困难，这让我对以后工作上的合作有了一次大概的认识，让我收获良多。