期末大作业个人报告

郭梓煜 17341046 计算机类

目录

小组分工

- 根据我们的小组分工，我负责的部分是水族馆中的多种多样的 *海洋生物的构思创造，设计以及部分实现*。具体来说就是，对水族馆中可渲染物体的基类Renderable的定义及实现（***Renderable.h，Renderable.cpp***），还有继承Renderable的子类存在于水族馆中的物体 Fish，Starfish，Crab，Octopus，Quad，Plant等子类的定义，为其他人的子类实现设计好接口。另外，我还实现了海洋生物章鱼（***Octopus.cpp***）的渲染与绘制实现。
  - Renderable.h，Renderable.cpp：可渲染基类定义与实现，海洋生物子类的定义
  - Octopus.cpp：海洋生物章鱼的绘制
- *水族馆的设计*，*整合以及试跑程序*，*查找并修改**bug***
- 最终的pre上我还负责 *小组报告* 的编写。

算法介绍

- Renderable基类的设计
  - 为了避免在实现诸多海洋生物时重复代码过多的情况，我定义了这个Renderable虚基类以便于子类的实现。其中关键之处在于定义物体的各种坐标以及虚函数_draw, _draw_dlist的定义，便于子类对物体的移动，旋转，以及缩放。最重要的是，子类可以通过对_draw等虚函数的重定义来实现不一样的绘制效果。
  - 下面是部分主要元素的解析（代码其实也有详细注释）
    - x,y,z为物体的三维坐标
    - rx,ry,rz为物体的旋转角度
    - sx,sz,sz为物体的缩放尺度
    - isList：是否在显示列中
    - _draw, _draw_dlist:绘制
  - 代码如下：

```
class Renderable
{
public:
    GLfloat x; /// x position of object
    GLfloat y; /// y position of object
    GLfloat z; /// z position of object

    GLfloat rx; /// x rotation angle of object
    GLfloat ry; /// y rotation angle of object
    GLfloat rz; /// z rotation angle of object

    GLfloat sx; /// x scale of object
    GLfloat sy; /// y scale of object
    GLfloat sz; /// z scale of object

    bool isList; /// is this a display list object?

    static unsigned int textures[2]; /// texture id array
    static GLUquadricObj *quadric;   /// quadric object for all renderables
public:
    Renderable();          /// default constructor
    virtual ~Renderable(); /// default destructor

    void build(GLuint &dlist);                    /// builds a display list
of this object
    void move(GLfloat x, GLfloat y, GLfloat z);   /// moves the object
    void rotate(GLfloat x, GLfloat y, GLfloat z); /// rotates the object
    void scale(GLfloat x, GLfloat y, GLfloat z);  /// scales the object
    void draw(void);

    static GLfloat getRand(GLfloat minimum, GLfloat range); /// generates a
random value in max range
protected:
    virtual void _draw(void) = 0;     /// draws the object
    virtual void _draw_dlist(void) {} /// draws the display list for object
};
#endif
```

- 基类实现
  - build
    - 为该物体创建显示列表

```
void Renderable::build(GLuint &dlist)
{
    dlist = glGenLists(1);
    if (!glIsList(dlist))
    {
        isList = false;
        return;
    }
```

```
    isList = true;

    glPushMatrix();
    glNewList(dlist, GL_COMPILE);
    _draw();
    glEndList();
    glPopMatrix();
}
```

- move
  - 移动物体

```
void Renderable::move(GLfloat x, GLfloat y, GLfloat z)
{
    this->x = x;
    this->y = y;
    this->z = z;
}
```

- rotate
  - 旋转物体

```
void Renderable::rotate(GLfloat x, GLfloat y, GLfloat z)
{
    this->rx = x;
    this->ry = y;
    this->rz = z;
}
```

- scale
  - 缩放物体

```
void Renderable::scale(GLfloat x, GLfloat y, GLfloat z)
{
    this->sx = x;
    this->sy = y;
    this->sz = z;
}
```

- draw

```
void Renderable::draw(void)
{
```

```
    glPushMatrix();

    glTranslatef(this->x, this->y, this->z);

    glRotatef(this->rx, 1.0f, 0.0f, 0.0f);
    glRotatef(this->ry, 0.0f, 1.0f, 0.0f);
    glRotatef(this->rz, 0.0f, 0.0f, 1.0f);

    glScalef(sx, sy, sz);

    // if the object is flagged as a display list object, then call the
    // display list drawing function of the object, otherwise just call
    // the normal draw function of the object
    if (this->isList)
        _draw_dlist();
    else
        _draw();

    glPopMatrix();
}
```

- getRand

```
GLfloat Renderable::getRand(GLfloat minimum, GLfloat range)
{
    return (((GLfloat)rand() / (GLfloat)RAND_MAX) * range) + minimum;
}
```

- 子类定义
  - 子类的定义以及接口只需继承基类，然后重构绘制相关的函数_draw,_draw_dlist即可。
  - StarFish

```
class StarFish : public Renderable
{
private:
    static GLfloat vertex[];  /// vertex array data
    static GLfloat normal[];  /// normals for each vertex
    static GLfloat colours[]; /// colour array data
    static GLfloat material[4];
    static GLfloat shininess;

public:
    StarFish();          /// default constructor
    virtual ~StarFish(); /// default destructor

protected:
```

```
    void _draw(void); /// draws the StarFish
};
```

- Fish

```
class Fish : public Renderable
{
private:
    GLfloat tailAngle;
    GLfloat tailAngleCutOff;
    GLfloat tailAngleInc;
    static GLfloat vertex[];  /// vertex array data
    static GLfloat normal[];  /// normals for each vertex
    static GLfloat texels[];  /// texture coords for each vertex
    static GLfloat colours[]; /// colour array data
    static GLfloat material[4];
    static GLfloat shininess;

private:
    void drawSide(void); /// draws a side of the fish
public:
    Fish();          /// default constructor
    virtual ~Fish(); /// default destructor

protected:
    void _draw(void); /// draws the Fish
};
```

- Crab

```
class Crab : public Renderable
{
private:
    GLuint dlist; /// display list
    static GLfloat material[4];
    static GLfloat shininess;

private:
    static void drawLeg(void);                                  /// draws
one leg
    static void drawLeg(GLfloat jointAngle, GLfloat jointOffset); /// draw
leg with an angle specified
    static void drawLegs(void);                                 /// draws
complete set of legs
public:
    Crab();          /// default constructor
    virtual ~Crab(); /// default destructor
protected:
    void _draw(void);                    /// draws the crab
```

```
    void _draw_dlist(void);               /// draws the crab's display list
    void generate(int level, int number); /// generates the branches

    friend class Octopus; /// so the octopus can use the drawLeg method
};
```

- Octopus

```
class Octopus : public Renderable
{
private:
    GLfloat legAngle;       /// angle to spin the legs at
    GLfloat legAngleCutOff; /// cut of angle for spinning
    GLfloat legAngleInc;    /// angle spin increment
    static GLfloat material[4];
    static GLfloat shininess;

public:
    Octopus();          /// default constructor
    virtual ~Octopus(); /// default destructor
protected:
    void _draw(void); /// draws the Octopus
};
```

- Quad

```
class Quad : public Renderable
{
private:
    static GLfloat material[4];
    static GLfloat shininess;

public:
    Quad();          /// default constructor
    virtual ~Quad(); /// default destructor
protected:
    void _draw(void); /// draws the quad
};
```

- Plant

```
class Plant : public Renderable
{
private:
    GLuint dlist; /// display list
    static GLfloat material1[4];
```

```
    static GLfloat material2[4];
    static GLfloat shininess;

public:
    Plant();          /// default constructor
    virtual ~Plant(); /// default destructor
protected:
    void _draw(void);                    /// draws the plant
    void generate(int level, int number); /// generates the branches
    void _draw_dlist(void);              /// draws the display list of
this object
};
```

- Octopus
  - 构造函数

```
Octopus::Octopus()
{
    cout << "-- Creating octopus\n";

    // leg rotation angles
    legAngle = 0.0f;
    legAngleCutOff = 30.0f;
    legAngleInc = 1.0f;
}
```

  - 重定义虚函数_draw
    - 章鱼的实现并不难，其运动方式与实验二的机器人有类似之处，不难实现。
    - 由于章鱼是后来新增的生物（感觉水族馆上空的生物较少所以新增），我整合代码时看到了队友的螃蟹实现，就直接调用了他螃蟹画腿的函数来实现我的章鱼腿。

```
void Octopus::_draw(void)
{
    // select our colour
    glColor3f(1.0f, 1.0f, 0.0f);

    // set up the material properties (only front needs to be set)
    glMaterialfv(GL_FRONT, GL_SPECULAR, material);
    glMaterialf(GL_FRONT, GL_SHININESS, shininess);
    glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
    glEnable(GL_COLOR_MATERIAL);

    // draw octopus body (stretched along Y axis)
    // up and down
    glPushMatrix();
    glScalef(1.0f, 3.0f, 1.0f);
    glutSolidSphere(0.3f, 16, 8);
```

```
    glPopMatrix();

    legAngle += legAngleInc;
    if (legAngle < -legAngleCutOff || legAngle > legAngleCutOff)
legAngleInc *= -1;

    glRotatef(legAngle, 0.0f, 1.0f, 0.0f);// rotate

    GLfloat step = 360.0f / 8;
    for (int i = 0; i < 8;  i++)// 8 legs
    {
        glPushMatrix();

        glRotatef(i * step, 0.0f, 1.0f, 0.0f);
        glTranslatef(0.1f, 0.5f + (legAngle / legAngleCutOff) / 7.0f,
0.0f);
        Crab::drawLeg(); //call the drawleg function of crab

        glTranslatef(0.2f, 0.725f, 0.0f);
        glRotatef(120.0f, 0.0f, 0.0f, 1.0f);

        Crab::drawLeg();//call the drawleg function of crab

        glPopMatrix();
    }

    // turn of colour material tracking
    glDisable(GL_COLOR_MATERIAL);
}
```

遇到问题及解决方法

主要出现的问题呢，其实还是在对接的过程中，比如函数的接口之类的，一旦涉及到多个人实现的项目，对接就是一个难题。好在我们每个人都较为活跃，这样在整合代码，运行程序的时候，问题就解决的快了很多。

至于我在代码遇到的问题，基本不大。Renderable基类不难实现，至于章鱼也是和实验二的机器人差不多，甚至还可以调用队友实现过的函数（由于我是整合代码的），所以在实现过程中并没遇到太大问题。

总结与感想

这一次的大作业组队实验，我负责了水族馆的构思设计，物体的虚基类的定义实现，以及最后的代码整合，bug修改以及最后组队报告的编写，收获良多。由于是组队完成，所以分工显得特别关键。基于我们商讨过后决定实现的项目，我们对整个场景都进行了划分。由我定义的基类开始，每个人可以对基类进行继承，分别实现一些接口，进而实现不同的海洋生物的构建。这样一来分工就显得很明确，大家合作起来也是井井有条。

总而言之，这一次大作业组队实验让我收获良多，不仅有代码方面的提升，对OpenGL的进一步理解，还有小组合作的分工工作，这对我以后的工作都有帮助。