**详细设计报告**

1引言

1.1编写目的

详细地写出对图形学算法演示系统所作的设计，总结系统设计结构。

1.2背景

待开发软件系统的名称：Tiny Graphics

任务提出者：大型应用课程设计

开发者：shuz

用户：闲人

1.3参考资料

基于java的计算机图形学 〔日〕青野雅树 著 张文乐 译

ISBN 7-03-012815-X

Computer Graphics Principles and Practice （美）James D.Foley 等著

ISBN 7-111-13026-2

Procedural Elements for Computer Graphics (美) David F.Rogers著

ISBN 7-111-09595-2

深入浅出MFC 2nd ed. 侯俊杰 著

ISBN 7-900614-93-1

Programming Windows with MFC / Jeff Prosise. -- 2nd ed.

ISBN 1-57231-695-0

MSDN Library Copyright 1987-2003 Microsoft Corporation

2程序系统的结构

内部核心类：

1. World3D:

提供各种方法使得外界可以观察它，例如：

intersect方法取得外界于一条ray相交的最近物体，用于ray-trace和鼠标点选物体中

get\_object\_iterator方法使得外界可以遍历其中的对象

get\_light\_iterator方法可以遍历其中的光源

另外，还提供了各种方法使得外界可以修改里面的内容：

add/remove方法可以用于添加/删除物体或光源

1. Object3D:

提供了各种方法使外界可以观察、修改它（抽象类）。除去各种参数设置（比如位置、方向的set\_position方法）外，主要提供以下两个方法：

to\_triangle\_set方法返回一个TriangleSet，用于scan-convert算法绘制该图形

intersect方法取得交点

1. Light:

提供了一个方法：illuminate，参数为空间给定一点，返回光源对该点的光照强度以及光线的起始位置和方向

1. WorldViewer:

提供了一个draw方法，抽象接口，下有子类RaytraceWorldViewer, LineFrameWorldViewer, GouroudShadingWorldViewer

1. Camera:

提供了get\_normaliztion\_matrix使得世界坐标中的各种元素根据视角不同变化到标准坐标，方便处理和显示

这些接口均有一些子类实现了它们，并提供了各种各样的功能。所有核心算法均在此处实现

交界处：

Graphics2D提供了必要的绘图操作，在这里就是set\_pixel和draw\_line（仅用在LineFrame模式中)，而CDCGraphics2D用CDC实现了它

Texture 提供了纹理贴图所需的get\_color(u, v)，而CImageTexture借助CImage类实现了它

这样，整个核心实现完全与MFC隔离

MFC部分：

MFC部分负责了繁重的GUI以及Serialize工作，用了一系列Adaptor类代理了相应的核心类，并提供了EditObject（跳出窗口编辑自己），Serialize（序列化）的功能

MFC类中的一些列Dialog提供了对Dialog的访问，提供了对于EditObject的支持

CTinyGraphicsDoc存储了单个文档的所有信息，提供了AddObject，AddLight，RemoveObject，RemoveLight等函数，使得View可以轻易索取想要的adaptor（以及相应的核心类），完成了文件保存、Load的所有工作

CTinyGraphicsView负责了所有菜单、鼠标、键盘操作，主要调用其它部分相应的实现即可，虽然代码繁多，却是实现时比较顺利的一关。

其它类：

核心类中还有许多辅助类，比如Position为所有物体和Camera提供了旋转、平移功能，而更底层的Matrix4D和Vector4D等则在这些高端类地下默默工作，在上面相关的代码中却很少见。

3程序设计说明

3.1程序描述

1. 图形的任意添加/删除/旋转，场景浏览
2. 文件保存/读取
3. 软件ZBuffer，位于View -> Gouroud Shading
4. 纹理贴图，在物体属性窗口中，可以将bmp, png, jpg, gif等图片贴于所有支持的5种形体上
5. 高级光线跟踪效果，实现了半透明、反射、折射

3.2功能

菜单功能：

File：

1. New…

新建文件

1. Open…

打开文件

1. Save

保存文件（注：Texture仅保存绝对路径）

View：

1. Line Frame

线框图显示

2. Gouroud Shading

以Gouroud Shading方式显示

3. Render…

用Raytrace方式绘制图形

4. Camera Setup…

设置Camera类型以及相关参数

Object：

添加、删除、修改图形

Light：

添加、删除、修改光源

鼠标漫游：

在场景中，按住鼠标左键拖动即可旋转视角或物体，按住鼠标右键拖动为平移视角或物体，滚动滚轮进行视角缩放。

具体如下：（以下坐标系均相对于视角）

1 左键/右键单击物体即选中该物体，在空白处单击取消选择。

2 选中物体的情况下左键拖动，镜头以物体为中心旋转，绕X和Y轴

3 选中物体的情况下右键拖动，镜头平移，在X-Y 平面

4 选中物体的情况下按住Ctrl + 左/右键拖动，变换轴相应变成X和Z

5 选中物体的情况下按住Shift + 左键拖动，则旋转该物体而非镜头（可与Ctrl结合使用，以切换旋转轴）

6 没有选中物体的情况下按左键拖动，旋转镜头，绕镜头位置(Location)旋转，Ctrl键意义不变

7 没有选中物体的情况下按右键拖动，平移镜头，Ctrl键意义不变

8 修改物体：双击物体弹出上述窗口可供修改

9 删除物体：选中物体，按DEL键删除

主要功能：

* 1. 软件ZBuffer，位于View -> Gouroud Shading
  2. 纹理贴图，在物体属性窗口中，可以将bmp, png, jpg, gif等图片贴于所有支持的5种形体上
  3. 高级光线跟踪效果，实现了半透明、反射、折射，但没有实现散射

3.3性能

物体较多时Gouroud Shading实时性不好，或者当物体离镜头很近时速度非常慢

3.5算法

3.6.1 光线跟踪算法

void RayTraceWorldViewer::draw(Graphics2D \*graphics) const {

graphics->draw\_background(pimpl->world->get\_background\_color());

int nu, nv;

pimpl->camera->get\_vpw\_resolution(nu, nv);

Impl::ObjectStack stack;

stack.reserve(pimpl->transmit\_depth + 1);

stack.push\_back(0);

for (int y = 0; y <=nv; ++y) {

for (int x = 0; x <= nu; ++x) {

Ray ray(pimpl->camera->get\_ray(x, y));

ColorRGB color = pimpl->ray\_trace(ray, pimpl->specular\_depth,

pimpl->transmit\_depth, stack);

graphics->set\_pixel(IntPoint2D(x, y), color);

}

}

}

ColorRGB RayTraceWorldViewer::Impl::ray\_trace(const Ray& ray, int specular\_depth, int transmit\_depth,

const ObjectStack& object\_stack) const {

real min\_dist = 0;

Vector3D normal;

int plane\_idx;

Object3D \*hit\_object = world->intersect(ray, min\_dist, normal, plane\_idx);

if (hit\_object == 0) // nothing hit, return background color

return world->get\_background\_color();

Point3D hit\_point = ray.point\_at(min\_dist);

const Material& material = hit\_object->get\_material();

ColorRGB texture\_color;

if (material.texture == 0)

texture\_color = material.color;

else

texture\_color = material.texture->get\_color(hit\_object->get\_plane\_coord(plane\_idx, hit\_point));

Vector3D sight = -ray.direction;

bool shoot\_into = true;

if (normal.dot\_product(sight) < 0) {

normal = -normal;

shoot\_into = false;

}

ColorRGB color = shade(hit\_object, texture\_color, hit\_point, plane\_idx, normal, sight);

if (specular\_depth > 0 && material.k\_specular != 0) { // specular ray

Ray reflect\_ray;

reflect\_ray.origin = hit\_point;

reflect\_ray.direction = reflect(sight, normal);

color += material.k\_specular \*

ray\_trace(reflect\_ray, specular\_depth-1,

transmit\_depth, object\_stack);

}

if (transmit\_depth > 0 && material.k\_transparency != 0) { // transmitted ray

Ray refract\_ray;

ObjectStack copy\_stack(object\_stack);

if (refract(hit\_object, hit\_point, sight, normal, shoot\_into,

copy\_stack, refract\_ray)) {

color += material.k\_transparency \* texture\_color \*

ray\_trace(refract\_ray, specular\_depth,

transmit\_depth-1, copy\_stack);

}

}

color.clamp();

return color;

}

3.6.2 z-buffer

// set the pixel if point (x, y) in the screen and

// zbuffer(x, y) <= z < 0

void set\_pixel(int x, int y, real z, const ColorRGB& color) {

if (0 <= x && x < x\_ext && 0 <= y && y < y\_ext) {

int pos = get\_pos(x, y);

if (zbuffer[pos] <= z && z < 0) {

graphics\_2d->set\_pixel(IntPoint2D(x, y), color);

zbuffer[pos] = z;

}

}

}