# A6: Grid Acceleration & Solid Textures

## 一、概述

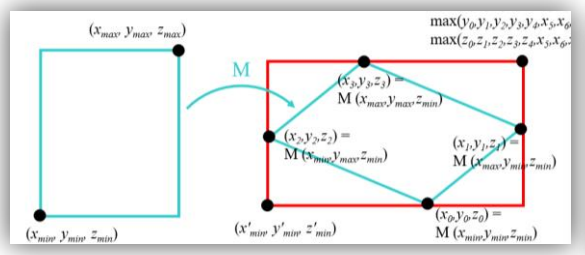
**1.目 标：**

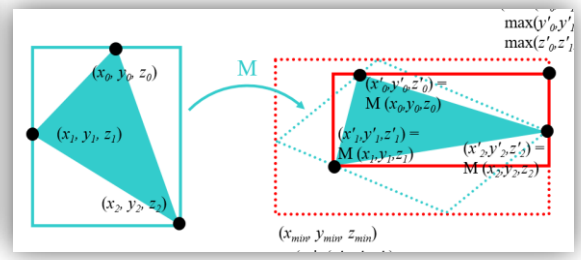
（1）用网格加速的方式来绘制物体，记录不同的统计数据
（2）实现网格加速中的 transform 变换
（3）实现固体纹理

**2.知识点介绍：**

（1）通过网格加速方式来绘制物体：

上次实验中，射线直接和 grid 中各个物体的包围盒进行交互，如果要和物体进行交互，就要遍历不为空的 cell 中的所有物体，找到射线和这些物体中最近的交点。

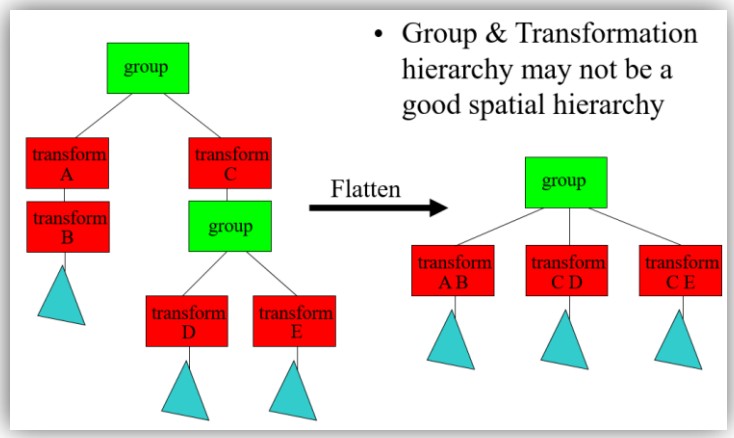



（2）transform 的包围盒有两种表达方式：

一种是直接对物体的包围盒进行变换后，取得新的最大值点和最小值点构成包围盒。另一种（对于三角形来说），先对三角形本身进行变换，直接根据变换过后的三角形形成包围盒。可以看到，第二种方式更加高效一点。

除此之外，还需要对 transfrom 的层级结构进行优化。grid 的 cell 中不存储 transform 的指针，每次遇到一个 transform 就把他们的矩阵累乘起来，最后插入 grid 的时候再根据累乘的矩阵对 obj 进行变换。不过这个地方我一直没有完全理解，最后虽然可以画出变换后的




物体，但是整个包围盒模拟却是错误的。

（3）Solid Texture 的实现：

棋盘、大理石等材质其实本身没有颜色的，它给出了混合两个带有颜色的材质的变换矩阵和噪声参数。由这些矩阵和噪声对物体上的某一点进行运算，得出该点混合两种材质颜

色的插值方式。

## 二、实现细节

1. 遍历 cell 中的所有物体，进行 ray 和物体的交互：

```
if (m_is_voxel_opaque[index]) {
                if (visualize) {
                    changeMaterial (index);
                    h.set(mi.tmin, material, mi.normal, r);
                    result = true; break;
                }else {
                    bool hasIntersect = false;
                    //在这里和cell中的每个primitive进行交互
                    for (int i = 0; i < objNum[index]; i++) {
                        if (object3ds[index][i]->intersect (r, h, tmin))
                            hasIntersect = true;
                    }
                    if (hasIntersect) { result = true; break; }
                }
            }
```

特别注意：对于 Plane，始终都要进行运算：

```
for (int i = 0; i < planes.size (); i++) {
        if (planes[i]->intersect (r, h, tmin))
            result = true;
    }
```

2.transform 的 BoundingBox 对于三角形的特殊处理：

```
BoundingBox* Transform::TransformBoundingBox () {
    Matrix m_matrix = transform_matrix;
    if (object3d->triangle) {
        Vec3f a = object3d->a;    Vec3f b = object3d->b;    Vec3f c = object3d->c;
        m_matrix.Transform (a);
        m_matrix.Transform (b);
        m_matrix.Transform (c);
        boundingBox = new BoundingBox (Vec3f (MIN (a.x (), b.x (), c.x ()), MIN (a.y
(), b.y (), c.y ()), MIN (a.z (), b.z (), c.z ())), Vec3f (MAX (a.x (), b.x (), c.x
()), MAX (a.y (), b.y (), c.y ()), MAX (a.z (), b.z (), c.z ())));
        return boundingBox;
    }
```

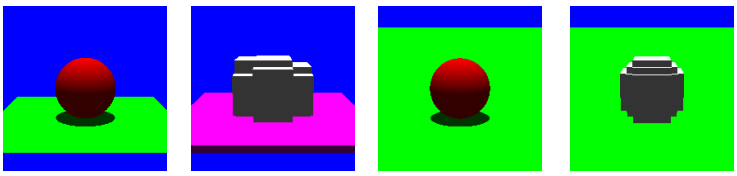
3.大理石纹理的判断（根据柏林噪声）：

```
Vec3f Marble::shade (const Ray &ray, const Hit &hit, const Vec3f &dirToLight, const
Vec3f &lightColor, bool shade_back)const {
    Vec3f hitpos = hit.getIntersectionPoint ();
    double answer = 0;
    for (int i = 0; i < octaves; i++){
        float tmp = pow (2.0f, i);
answer+= PerlinNoise::noise(tmp*hitpos[0],tmp*hitpos[1],tmp*hitpos[2])/ float (tmp);
```

```
    }
    double m = 0.5 + 0.5 * sin (frequency * hitpos[0] + amplitude * answer);
  // M(x,y,z) = sin (frequency * x + amplitude * N(x,y,z))
    Hit tempHit;
    tempHit.set (hit.getT (), mat1, hit.getNormal (), ray);
    Vec3f color0 = hit.getMaterial ()->shade (ray, tempHit, dirToLight,
lightColor,shade_back);
    tempHit.set (hit.getT (), mat2, hit.getNormal (), ray);
    Vec3f color1 = hit.getMaterial ()->shade (ray, tempHit, dirToLight, lightColor,
shade_back);
    Vec3f color2 = m * color0 + (1 - m) * color1;
    return  color2;
}
```

## 三、结果展示

1.阴影和平面测试:



2.三角形模型测试：数据差异明显。200 面不加阴影不优化需要 6s，1k 面加阴影优化后只需要 4s。5k 面开阴影优化后需要 15s,40k 高精度模型开阴影需要 13min。

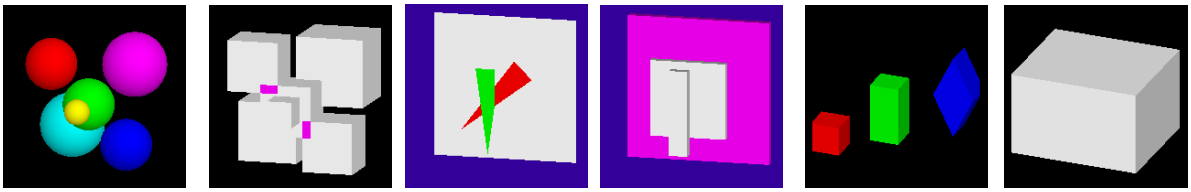
（1）200 面不开阴影不加速

```
-input scene6_04_bunny_mesh_200.txt -output output6_04a.tga -size 200 200 -stats
Rendering....done.
*********************************************
RAY TRACING STATISTICS
  total time                0:00:06
  num pixels                40000 (200x200)
  scene bounds              NULL
  num grid cells            NULL
  num non-shadow rays       40000
  num shadow rays           0
  total intersections       8040000
  total cells traversed     0
  rays per second           6666.7
  rays per pixel            1.0
  intersections per ray     201.0
  cells traversed per ray   0.0
*********************************************
```

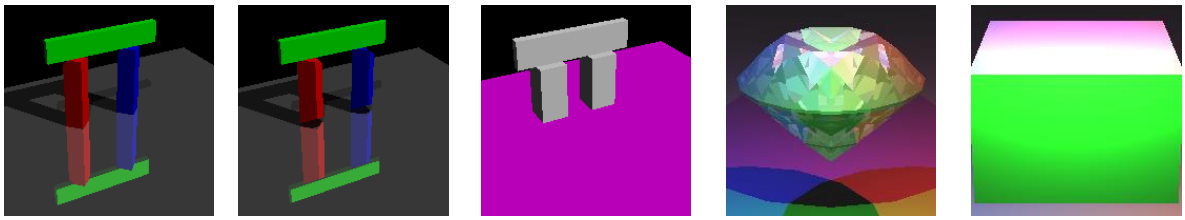
（2）1k 面开阴影加速

```
E:\MyStudy\Assignments\Assignments\Debug\A6>E:\MyStudy\Assignments\Assignments\Debug\A6\A6.exe -
sh_1k.txt  -output output6_05.tga -size 200 200 -grid 15 15 12 -stats -shadows
Rendering....done.
平面的数量是 : 1
*********************************************
RAY TRACING STATISTICS
  total time                0:00:04
  num pixels                40000 (200x200)
  scene bounds              NULL
  num grid cells            NULL
  num non-shadow rays       40000
  num shadow rays           66782
  total intersections       1218549
  total cells traversed     397287
  rays per second           26695.5
  rays per pixel            2.7
  intersections per ray     11.4
  cells traversed per ray   3.7
```

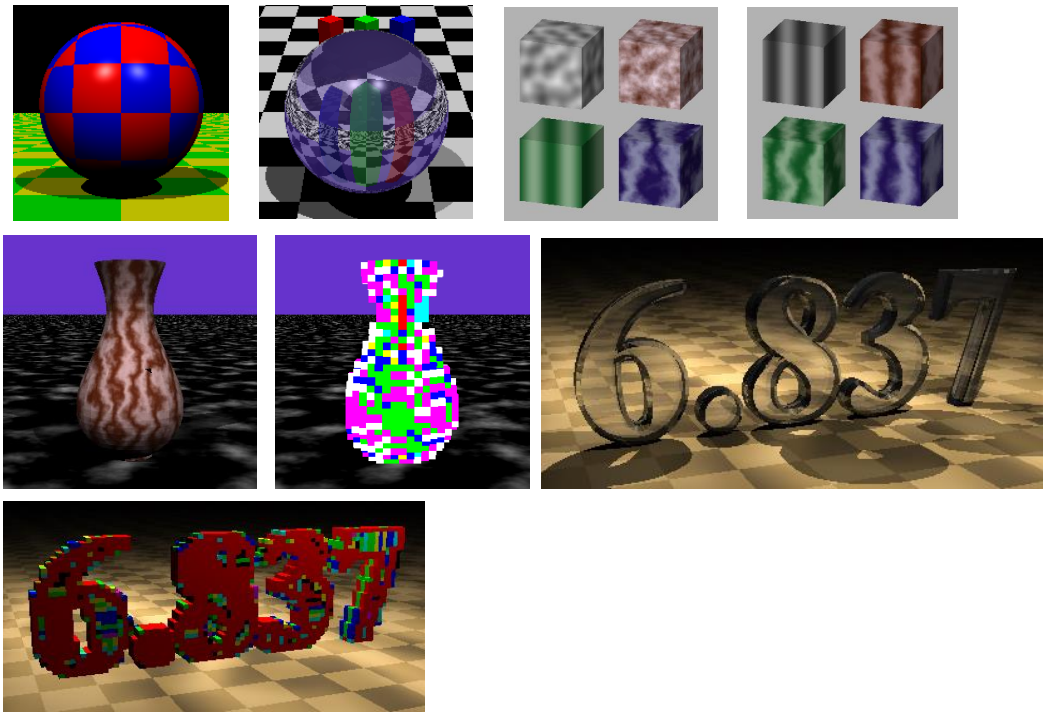
3.transform 测试：前两个正常，最后一个包围盒出现问题



4.反射和折射测试：transform 有些问题，最后的图和原图有些差别



5.纹理测试：



## 四、心得体会

通过网格加速的方式很大地提高了渲染的效率，使得 ray 避免了很多无谓的计算量。transform 涉及到坐标空间的转换和包围盒的更新，处理过程中有些棘手。通过程序化的方式生成的 solid texture 可以在已有材质的基础上生成千变万化的纹理，可以说是技术对艺术创作的一份启发了。