一、实验步骤：

1、增加 RayTracingStats 内的函数来检测性能，并实现加速的 RayCast。

在 RayTracer 中选择 intersect 的方法：

```
       if (isgrid) {
            if (isvisgrid) intsec = grid->intersect(ray, hit, tmin);
            else intsec = grid->intersect_2(ray, hit, tmin);
        }
            else intsec = gro->intersect(ray, hit, tmin);
```

关于 intersect_2 函数：

```
    virtual bool intersect_2(const Ray &r, Hit &h, float tmin) {
        int i, j, k;
        Material *m;
        Vec3f diffuseColor;
        diffuseColor.Set(1, 1, 1);
        Vec3f specularColor(0, 0, 0);
        float exponent = 1;
        Vec3f reflectiveColor(0, 0, 0);
        Vec3f transparentColor(0, 0, 0);
        float indexOfRefraction = 1;
        m = new PhongMaterial(diffuseColor, specularColor, exponent, reflectiveColor,
transparentColor, indexOfRefraction);

        MarchingInfo mi;
        initializeRayMarch(mi, r, tmin);
        if (mi.GetTmin() == INFINITY) {
            int obnum = inf_obj.getNumObjects();
            int state = 0;
            for (k = 0; k < obnum; k++) {
                if (inf_obj.getObject(k)->intersect(r, h, tmin)) state = 1;
            }
            return state;
        }
        int indx, indy, indz;
        mi.GetIndice(indx, indy, indz);
        while (indx >= 0 && indx < nx && indy >= 0 && indy < ny && indz >= 0 && indz < nz)
{
            //cout << indx << "  " << indy << "  " << indz << endl;
            Vec3f v0, v1, v2, v3, v4, v5, v6, v7;
            bb->Get(v0, v6);
            Vec3f x(lx, 0, 0), y(0, ly, 0), z(0, 0, lz);
            v0 += x * indx + y * indy + z * indz;
            v1 = v0 + y;
            v2 = v0 + y + z;
            v3 = v0 + z;
            v4 = v0 + x;
            v5 = v0 + x + y;
            v6 = v0 + x + y + z;
            v7 = v0 + x + z;
            Vec3f n;
            n.Set(-1, 0, 0);
            RayTree::AddHitCellFace(v3, v2, v1, v0, n, m);
            if (n == mi.GetNormal()) RayTree::AddEnteredFace(v3, v2, v1, v0, n, m);
            n.Set(1, 0, 0);
            RayTree::AddHitCellFace(v4, v5, v6, v7, n, m);
            if (n == mi.GetNormal()) RayTree::AddEnteredFace(v4, v5, v6, v7, n, m);
```

```
            n.Set(0, -1, 0);
            RayTree::AddHitCellFace(v4, v7, v3, v0, n, m);
            if (n == mi.GetNormal()) RayTree::AddEnteredFace(v4, v7, v3, v0, n, m);
            n.Set(0, 1, 0);
            RayTree::AddHitCellFace(v1, v2, v6, v5, n, m);
            if (n == mi.GetNormal()) RayTree::AddEnteredFace(v1, v2, v6, v5, n, m);
            n.Set(0, 0, -1);
            RayTree::AddHitCellFace(v0, v1, v4, v5, n, m);
            if (n == mi.GetNormal()) RayTree::AddEnteredFace(v0, v1, v4, v5, n, m);
            n.Set(0, 0, 1);
            RayTree::AddHitCellFace(v2, v3, v7, v6, n, m);
            if (n == mi.GetNormal()) RayTree::AddEnteredFace(v2, v3, v7, v6, n, m);

            if (occ[indx][indy][indz] == 1) {
                int obnum = objs[indx][indy][indz].getNumObjects();
                int state = 0;
                for (k = 0; k < obnum; k++) {
                    if (objs[indx][indy][indz].getObject(k) == NULL) continue;
                    if(objs[indx][indy][indz].getObject(k)->intersect(r, h, tmin))
state = 1;
                }
                if (state) return 1;
            }
            mi.nextCell();
            mi.GetIndice(indx, indy, indz);

            RayTracingStats::IncrementNumGridCellsTraversed();
        }
        int obnum = inf_obj.getNumObjects();
        int state = 0;
        for (k = 0; k < obnum; k++) {
            if (inf_obj.getObject(k)->intersect(r, h, tmin)) state = 1;
        }
        return state;
    }
```

在遇到物体占用的体素时，选择与里面所有的物体做 intersect，如果有交点，则返回，如果没有，则继续；在循环结束后还应该记得与场景里的所有平面作相交，平面的指针放在 inf_obj 里。

RayTracingStats 函数需要按照实验指导里的要求放在不同的位置，需要注意不仅要使得加速渲染时的测量正确，还要使最开始的测量是正确的，所以要谨慎考虑函数所放位置的合理性。

2、构造 Material 的子类 CheckBoard 类，用于两种物质的格状显示：

```
class CheckerBoard :public Material
{
public:
    CheckerBoard(Matrix *_m, Material *_mat1, Material *_mat2) {
        m = _m;
        mat1 = _mat1;
        mat2 = _mat2;
    }
    void glSetMaterial(void) const{
        mat1->glSetMaterial();
```

```
        return;
    }
    Vec3f Shade(const Ray &ray, const Hit &hit, const Vec3f &dirToLight, const Vec3f
&lightColor) const {
        Vec3f p = hit.getIntersectionPoint();
        m->Transform(p);
        bool b1 = odd(floor(p.x()));
        bool b2 = odd(floor(p.y()));
        bool b3 = odd(floor(p.z()));
        if (odd(b1 + b2 + b3)) {
            return mat2->Shade(ray, hit, dirToLight, lightColor);
        }
        else {
            return mat1->Shade(ray, hit, dirToLight, lightColor);
        }
    }
private:
    Matrix *m;
    Material *mat1;
    Material *mat2;
};
```

使用数的奇偶性来分割三维空间。

3、构造 Material 的子类 Noise 类：

```
class Noise :public Material {
public:
    Noise(){}
    Noise(Matrix *_m, Material *_mat1, Material *_mat2, int _octaves) {
        m = _m;
        mat1 = _mat1;
        mat2 = _mat2;
        octaves = _octaves;
    }
    ~Noise() {}
    Vec3f Shade(const Ray &ray, const Hit &hit, const Vec3f &dirToLight,const Vec3f
&lightColor) const {
        Vec3f p = hit.getIntersectionPoint();
        m->Transform(p);
        double noi = N(p.x(),p.y(),p.z());
        noi = noi * 0.6 + 0.4;
        Vec3f col1 = mat1->Shade(ray, hit, dirToLight, lightColor);
        Vec3f col2 = mat2->Shade(ray, hit, dirToLight, lightColor);
        return col1 * noi + col2 * (1 - noi);
    }
    void glSetMaterial(void) const {
        mat1->glSetMaterial();
        return;
    }
    double N(double x, double y, double z) const{
        PerlinNoise pn;
        double result = 0;
        int count = 0;
        while (count < octaves) {
            double a = pow(2, count);
            result += pn.noise(a*x, a*y, a*z) / a;
            count++;
```

```
        }
        return result;
    }
private:
    Matrix *m;
    Material *mat1;
    Material *mat2;
    int octaves;
};
```

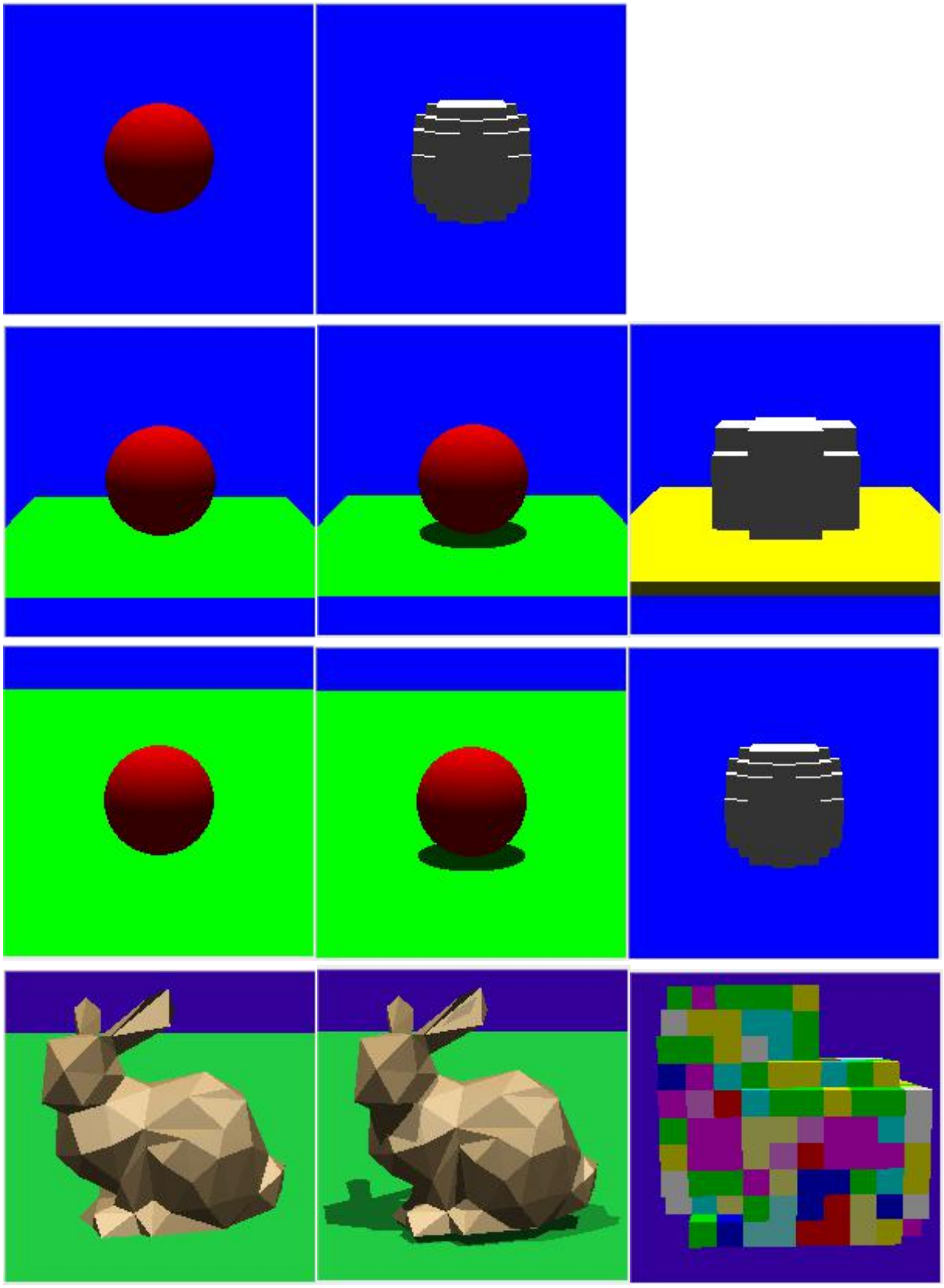
注意 N 函数的边界条件。

4、构造 Material 的子类 Marble 类，需要用到 noise 类里的函数：

```
class Marble :public Material {
public:
    Marble(){}
    Marble(Matrix *_m, Material *_mat1, Material *_mat2, int _octaves, float _frequency,
float _amplitude) {
        m = _m;
        mat1 = _mat1;
        mat2 = _mat2;
        octaves = _octaves;
        frequency = _frequency;
        amplitude = _amplitude;
    }
    ~Marble() {}
    double M(double x, double y, double z) const {
        Noise noi(m, mat1, mat2, octaves);
        return sin(frequency * x + amplitude * noi.N(x, y, z));
    }
    Vec3f Shade(const Ray &ray, const Hit &hit, const Vec3f &dirToLight, const Vec3f
&lightColor) const {
        Vec3f p = hit.getIntersectionPoint();
        m->Transform(p);
        double noi = M(p.x(), p.y(), p.z());
        noi = (noi + 1) / 2;
        Vec3f col1 = mat1->Shade(ray, hit, dirToLight, lightColor);
        Vec3f col2 = mat2->Shade(ray, hit, dirToLight, lightColor);
        return col1 * noi + col2 * (1 - noi);
    }
    void glSetMaterial(void) const {
        mat1->glSetMaterial();
        return;
    }
private:
    Matrix *m;
    Material *mat1;
    Material *mat2;
    int octaves;
    float frequency;
    float amplitude;
};
```

M 函数的返回值范围是-1 到 1，插值的时候需要注意。

二、实验结果



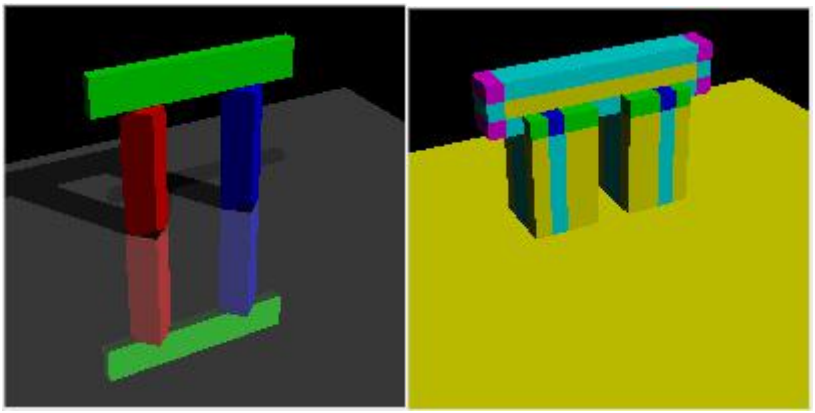
```
raytracer -input scene6_04_bunny_mesh_200.txt -output output6_04a.tga -size 200 200 -stats

**********************************************
RAY TRACING STATISTICS
  total time                445976:11:42
  num pixels                0 (0x0)
  scene bounds              NULL
  num grid cells            NULL
  num non-shadow rays       40000
  num shadow rays           66040
  total intersections       21314040
  total cells traversed     0
  rays per second           0.0
  rays per pixel            inf
  intersections per ray     201.0
  cells traversed per ray   0.0
**********************************************
success!
```

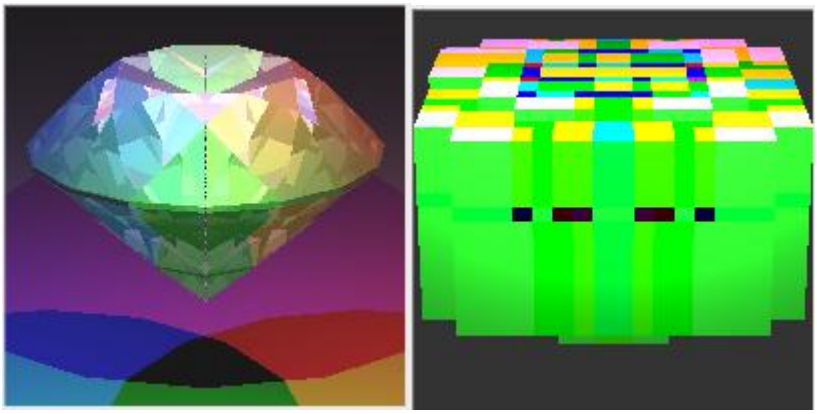
```
raytracer -input scene6_04_bunny_mesh_200.txt -output output6_04b.tga -size 200 200 -grid 10 10 7 -stats

**********************************************
RAY TRACING STATISTICS
  total time                0:00:04
  num pixels                40000 (200x200)
  scene bounds              -0.191055 0.067364 -0.114066  -> 0.121072 0.372852 0.116906
  num grid cells            700 (10x10x7)
  num non-shadow rays       40000
  num shadow rays           66040
  total intersections       859041
  total cells traversed     208993
  rays per second           26510.0
  rays per pixel            2.7
  intersections per ray     8.1
  cells traversed per ray   2.0
**********************************************
success!
```

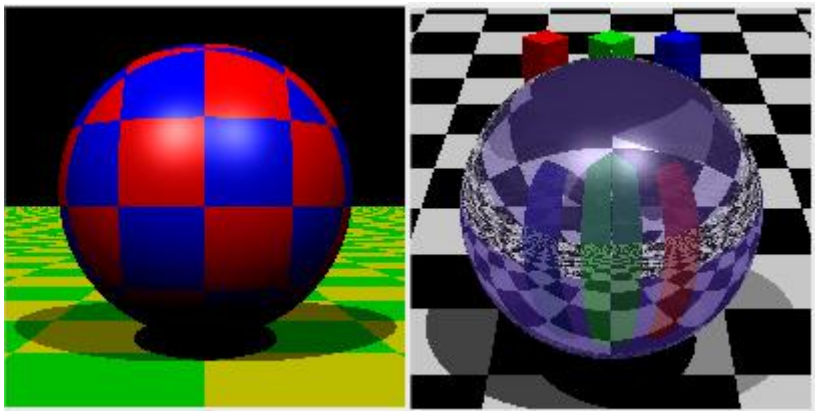
在上面的几个案例中，通过分析输出的 stats 表可知（上面是渲染没有阴影的兔子的报告）。在使用没有加速的渲染方法时，面对较少的物体，由于不需要构建边界盒并将物体插入到体素里，渲染速度更快；但是面对由较多物体组成的图像时（尤其是兔子这个例子），每条射线都要和所有物体测试是否相交，计算量急剧增加，最终的 intersect 调用比加速过的多了近 25 倍（如果物体更多，这个倍数还将扩大），更多的时间得到的图像还是一样的。

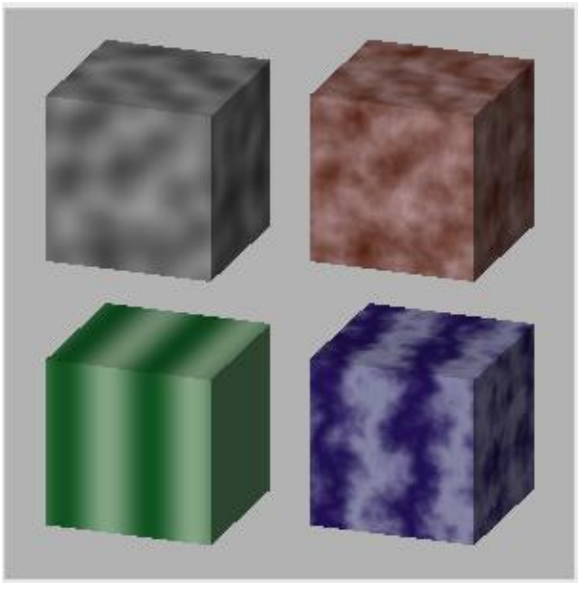
因此在渲染较少物体组成的图像时，不选择 grid（或使用-grid 1 1 1）；在渲染较多物体时（数量>=20 且不需要阴影和折射），使用 grid 选项，参数的选取应尽量使分割后的单位体素趋近于正方体，且最小的体素大小不能小于最小的边界盒的一半（仅个人测试的推测结论，可能不够准确）。

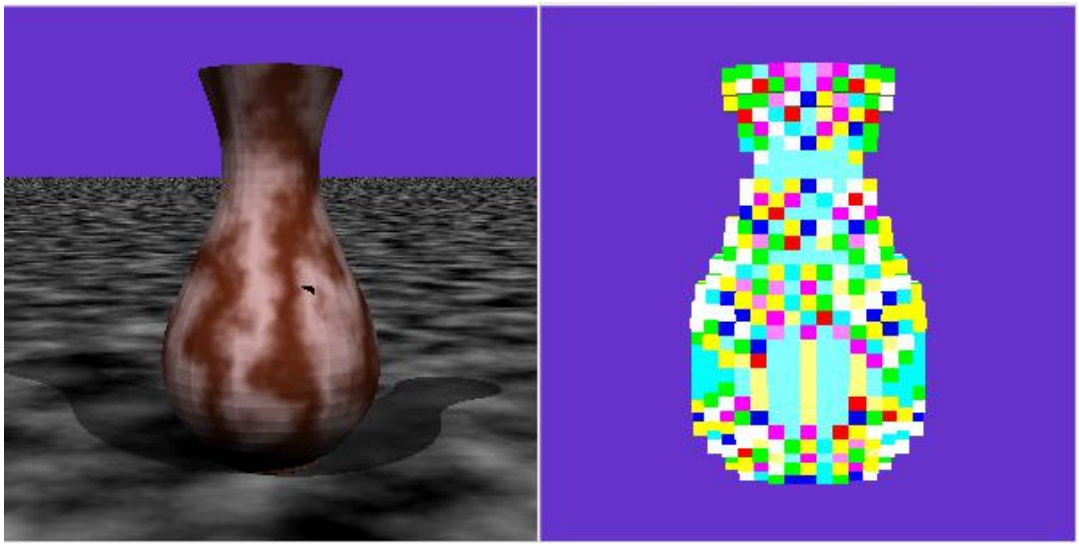
Scene 8-10 的测试在 Assignment05 里做过了，不再展示结果。

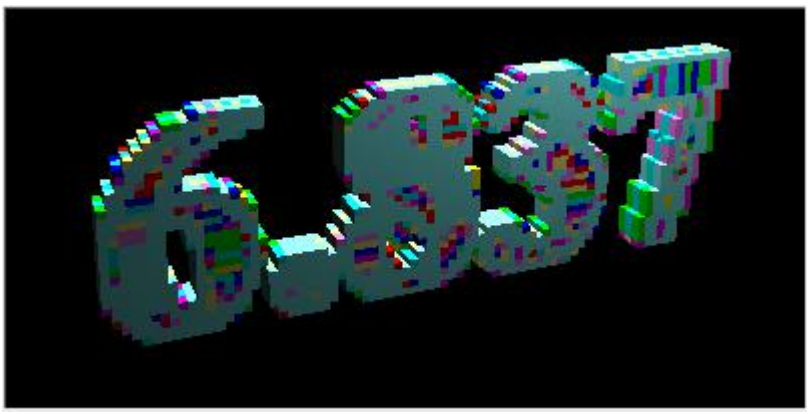












三、实验心得

在该实验中，大部分难点已经在 Assignment 5 中解决，但是在做到 Noise 的时候，关于插指的代码我一直没有做出最好的效果，感觉是 N 函数返回值的取值范围没有处理好，需要进一步的改进。另外 Wood 的代码与 Marble 代码一致，这里直接复制了一份，是为了让最后一个案例正确执行。