Assignment 5: Voxel Rendering

陈九润 3180105488

1 代码

1.1 在Object3D子类构造函数中加入Boundingbox构造

以Triangle为例

```
    Triangle(Vec3f& a, Vec3f& b, Vec3f& c, Material* m) :Object3D(m), a(a),
b(b), c(c)
    {
        Vec3f::Cross3(normal, b - a, c - b);
        normal.Normalize();


        //generate bounding box
        Vec3f min;
        Vec3f::Min(min, a, b);
        Vec3f::Min(min, min, c);

        Vec3f max;
        Vec3f::Max(max, a, b);
        Vec3f::Max(max, max, c);


        boundingBox = new BoundingBox(min,max);
    }
```

1.2 Grid类

Grid类使用Object3DVector*** voxels存储nx*ny*nz个Voxel，每个Voxel为一个Object3DVector，存储指向有可能在该Voxel内Object的指针

```
    Grid(BoundingBox* bb, int nx, int ny, int nz):nx(nx),ny(ny),nz(nz)
    {
        boundingBox = bb;
        boundingBox->Get(gridMinVertex, gridMaxVertex);

        Vec3f subVertex = gridMaxVertex - gridMinVertex;
        subVertex.Divide(nx, ny, nz);
        gridStep = subVertex;

        voxelHalfDiagonalLength = gridStep.Length() / 2;
```

```
        voxels = new Object3DVector** [nx];
        for (int i = 0; i < nx; i++)
        {
            voxels[i] = new Object3DVector* [ny];
            for (int j = 0; j < ny; j++)
            {
                voxels[i][j] = new Object3DVector [nz];
            }
        }
        //other initialization code

    }
```

1.3 为Object3D子类加入insertIntoGrid函数

使用保守的估计

Triangle

```
    virtual void insertIntoGrid(Grid* grid, Matrix* m)
    {
        Vec3f minVertex;
        Vec3f maxVertex;
        boundingBox->Get(minVertex, maxVertex);
        if (m != NULL)
        {
            //get eight vertices of sub object bounding box
            Vec3f transVertices[3];
            transVertices[0] = a;
            transVertices[1] = b;
            transVertices[2] = c;

            minVertex.Set(INFINITY, INFINITY, INFINITY);
            maxVertex.Set(-INFINITY, -INFINITY, -INFINITY);
            for (int i = 0; i < 3; i++)
            {
                m->Transform(transVertices[i]);
                Vec3f::Min(minVertex, minVertex, transVertices[i]);
                Vec3f::Max(maxVertex, maxVertex, transVertices[i]);
            }
        }
        int mini, minj, mink;
        int maxi, maxj, maxk;
        assert(grid->getVoxelIndex(minVertex, mini, minj, mink));
        assert(grid->getVoxelIndex(maxVertex, maxi, maxj, maxk));
        for (int i = mini; i <= maxi; i++)
        {
            for (int j = minj; j <= maxj; j++)
            {
```

```
                for (int k = mink; k <= maxk; k++)
                {
                    grid->addObjectToVoxel(i, j, k, this);
                }
            }
        }
    }
```

Sphere

```
 //Assignment5
    virtual void insertIntoGrid(Grid* grid, Matrix* m)
    {

        //transform
        if (m != NULL)
        {
            Vec3f minVertex;
            Vec3f maxVertex;
            boundingBox->Get(minVertex, maxVertex);

            //get eight vertices of sub object bounding box
            Vec3f transVertices[8];
            transVertices[0] = minVertex;
            transVertices[1] = Vec3f(minVertex.x(), minVertex.y(),
maxVertex.z());
            transVertices[2] = Vec3f(minVertex.x(), maxVertex.y(),
minVertex.z());
            transVertices[3] = Vec3f(minVertex.x(), maxVertex.y(),
maxVertex.z());
            transVertices[4] = Vec3f(maxVertex.x(), minVertex.y(),
minVertex.z());
            transVertices[5] = Vec3f(maxVertex.x(), minVertex.y(),
maxVertex.z());
            transVertices[6] = Vec3f(maxVertex.x(), maxVertex.y(),
minVertex.z());
            transVertices[7] = Vec3f(maxVertex.x(), maxVertex.y(),
maxVertex.z());

            minVertex.Set(INFINITY, INFINITY, INFINITY);
            maxVertex.Set(-INFINITY, -INFINITY, -INFINITY);
            for (int i = 0; i < 8; i++)
            {
                m->Transform(transVertices[i]);
                Vec3f::Min(minVertex, minVertex, transVertices[i]);
                Vec3f::Max(maxVertex, maxVertex, transVertices[i]);
            }
            int mini, minj, mink;
            int maxi, maxj, maxk;
            assert(grid->getVoxelIndex(minVertex, mini, minj, mink));
            assert(grid->getVoxelIndex(maxVertex, maxi, maxj, maxk));
            for (int i = mini; i <= maxi; i++)
            {
                for (int j = minj; j <= maxj; j++)
                {
```

```
                    for (int k = mink; k <= maxk; k++)
                    {
                        grid->addObjectToVoxel(i, j, k, this);
                    }
                }
            }
            return;
        }
```

Group

Group对该函数变换矩阵参数m的处理是我遇到的一个大坑

由于使用指针传递，调用每个子object的insertIntoGrid时要使用不同的m

我一开始对所有下层函数调用使用了同一个地址m，导致m被多个object修改，出现错误，一直找不到问题

其实感觉该函数接口设计存在问题，不应该使用指针而应该使用Matrix，当没有矩阵变换时m应传入单位矩阵，会产生遇到该错误

```
    //Assignment5
    virtual void insertIntoGrid(Grid* g, Matrix* m)
    {
        for (auto it = objects.begin(); it != objects.end(); it++)
        {
            Matrix* tempM=NULL;
            if (m != NULL)
            {
                tempM = new Matrix(*m);
            }
            (*it)->insertIntoGrid(g, tempM);
            if (tempM != NULL)
            {
                delete tempM;
            }
        }
    }
```

Transform

Transform要将自己的变换矩阵乘到m上，若m为空指针，则应该new

```
    virtual void insertIntoGrid(Grid* grid, Matrix* m)
    {
        if (m == NULL)
        {
            m = new Matrix;
            m->SetToIdentity();
        }
        //cout << "My transform: " << transform << endl;
        //cout << "m: " << *m << endl;
        *m = (*m)*transform;
        //cout << "after *: " << *m << endl;
        object->insertIntoGrid(grid, m);
    }
```

1.4 修改RayTracer类

在构造函数中调用group的insertIntoGrid

```
    //Assignment4
    RayTracer(char* input_file, int width, int height, int max_bounces, float
cutoff_weight, bool shadows, bool shadeback, bool useGrid,int nx, int ny, int
nz) :
        input_file(input_file), width(width), height(height),
maxBounces(max_bounces), cutoffWeight(cutoff_weight),
shadeShadows(shadows),shadeBack(shadeback)
    {
        scene = new SceneParser(input_file);
        hits = new Hit[width * height];
        rays = new Ray[width * height];
        assert(scene != NULL);
        ambientLight = scene->getAmbientLight();

        //cout << "here2" << endl;
        //Assignment5
        if (useGrid)
        {
            //Matrix *m=new Matrix;
            //m->SetToIdentity();
            grid = new Grid(scene->getGroup()->getBoundingBox(), nx, ny, nz);
            scene->getGroup()->insertIntoGrid(grid, NULL);
        }
        else
        {
            grid = NULL;
        }
    }
```

gridShader函数用来渲染grid，调用grid->intersect函数与grid求交

```
  //Assignment5
    //based on phong shader
    void gridShader(char* outputFile)
    {
        Image outputImage(width, height);
```

```
        for (int i = 0; i < width * height; i++)
        {
            int x = i % width;
            int y = i / width;
            Hit hit;
            Ray ray = generateRayAtIndex(i);
            grid->intersect(ray, hit, scene->getCamera()->getTMin());
            if (hit.getT() == INFINITY)
            {
                outputImage.SetPixel(x, y, scene->getBackgroundColor());
                continue;
            }
            Vec3f N = hit.getNormal();
            //if shade back
            if (shadeBack && rays[i].getDirection().Dot3(N) > 0)
            {
                N.Negate();
            }
            //no shade back and ray inside object
            if (!shadeBack && rays[i].getDirection().Dot3(N) > 0)
            {
                outputImage.SetPixel(x, y, Vec3f(0, 0, 0));
                continue;
            }

            Vec3f objectColor = hit.getMaterial()->getDiffuseColor();
            Vec3f ambientColor = scene->getAmbientLight() * objectColor;
            Vec3f diffuseSpecularColor(0, 0, 0);
            for (int j = 0; j < scene->getNumLights(); j++)
            {
                Vec3f dirToLight;
                Vec3f lightColor;
                float distanceToLight;
                scene->getLight(j)->getIllumination(hit.getIntersectionPoint(),
dirToLight, lightColor, distanceToLight);
                diffuseSpecularColor += hit.getMaterial()->Shade(hit.getRay(),
hit, dirToLight, lightColor);
            }
            Vec3f pixelColor = diffuseSpecularColor + ambientColor;
            outputImage.SetPixel(x, y, pixelColor);
        }
        outputImage.SaveTGA(outputFile);
    }
```

1.5 实现Grid::paint函数

对每个存在object的voxel，绘制一个正方体，并要正确设置六个面的法向量

```
    virtual void paint(void)
    {
        glBegin(GL_QUADS);
        for (int i = 0; i < nx; i++)
        {
            for (int j = 0; j < ny; j++)
            {
```

```
                for (int k = 0; k < nz; k++)
                {
                    //if (opaque[i][j][k] == true)
                    int numObjects = voxels[i][j][k].getNumObjects();
                    if(numObjects!=0)
                    {
                        int index = numObjects-1;
                        if (index >= 16)
                        {
                            index = 15;
                        }
                        this->material = materials[index];
                        getMaterial()->glSetMaterial();
                        Vec3f offset = gridStep * Vec3f(i, j, k);
                        for (int f = 0; f < 6; f++)
                        {
                            glNormal3f(cubeNormals[f].x(), cubeNormals[f].y(),
cubeNormals[f].z());
                            for (int v = 0; v < 4; v++)
                            {
                                Vec3f vertex = cubeVertices[cubeFaces[f][v]]+
offset;
                                glVertex3f(vertex.x(), vertex.y(), vertex.z());
                            }
                        }
                    }
                }
            }
        }
        glEnd();
    }
```

1.6 实现Grid Ray Marching算法

1.6.1 Grid::intersect

该函数求光线与Grid的交

首先调用initializeRayMarch求初始交点

而后调用marchingInfo::nextCell遍历光线经过的每个Voxel，直到遇到一个不为空的Voxel，设置hit的颜色为该Voxel的颜色（与Voxel中含有几个object有关）

```
    virtual bool intersect(const Ray& r, Hit& h, float tmin)
    {
        nowMaterialIndexCell = 0;
        nowMaterialIndexFace = 0;
        MarchingInfo mi;
        initializeRayMarch(mi,r,tmin);
        if (mi.getTmin() == INFINITY)
        {
            return false;
        }
```

```
        int i,j,k;
        mi.getGridIndex(i,j,k);
        while (i>=0 && i < nx &&j>=0&& j<ny &&k>=0&& k<nz)
        {
            addHitCell(i, j, k);
            addEnteredFace(i, j, k, mi.getAxis(), mi.getSign());
            //cout << "i,j,k: " << i << " " << j << " " << k << " " << endl;
            int numObjects = voxels[i][j][k].getNumObjects();
            if (numObjects !=0)
            {
                int index = numObjects-1;
                if (index >= 16)
                {
                    index = 15;
                }
                h.set(mi.getTmin(), materials[index],mi.getNormal() , r);
                //cout << "Normal: " << mi.getNormal()<<endl;
                return true;
            }
            mi.nextCell();
            mi.getGridIndex(i,j,k);
        }

        return false;

    }
```

1.6.2 Grid::initializeRayMarch

该函数首先判断光线起始位置与grid关系，分为grid内和grid外进行处理：

若为grid内则直接得到起始Voxel的index

若为grid外则使用线和多面体求交的方法判断交点，并获得起始Vodel的index，注意处理过程中要在合适的位置引入epsilon误差

求得起始Voxel的index后要正确设置MarchingInfo类的tmin、tnext、normal等参数

```
    void initializeRayMarch(MarchingInfo& mi, const Ray& r, float tmin) const
    {
        int sign[3];
        r.getSign(sign);
        Vec3f rayInvDir = r.getInverseDirection();
        Vec3f rayOrigin = r.getOrigin();

        int startIndex[3] = { 0,0,0 };

        Vec3f startPoint = r.getOrigin() + tmin * r.getDirection();

        if (!getVoxelIndex(startPoint, startIndex))
        {
```

```
            float tNear = -INFINITY;
            float tFar = INFINITY;

            for (int i = 0; i < 3; i++)
            {
                if (rayInvDir[i] >= 0)
                {
                    tNear = fmaxf(tNear,( gridMinVertex[i] - rayOrigin[i]) *
rayInvDir[i]);
                    tFar = fminf(tFar,(gridMaxVertex[i] - rayOrigin[i]) *
rayInvDir[i]);
                }
                else
                {
                    tNear = fmaxf(tNear,( gridMaxVertex[i] - rayOrigin[i]) *
rayInvDir[i]);
                    tFar = fminf(tFar, (gridMinVertex[i] - rayOrigin[i]) *
rayInvDir[i]);
                }
            }

            //no intersection
            if (!(tNear < tFar && tNear >= tmin))
            {
                mi.setTmin(INFINITY);
                return;
            }

            //add epsilon
            startPoint = r.getOrigin() + r.getDirection() * (tNear);
            assert(getVoxelIndex(startPoint, startIndex));
            tmin = tNear;
        }

        mi.setGridIndex(startIndex);

        mi.setSign(sign);

        Vec3f dt;
        float arraydt[3];
        for (int i = 0; i < 3; i++)
        {
            arraydt[i] = gridStep[i] * fabs(rayInvDir[i]);
        }
        dt.Set(arraydt[0], arraydt[1], arraydt[2]);
        mi.setDT(dt);
        mi.setTmin(tmin);

        //get tnext
        Vec3f nextVoxelConner = getVoxelCornerBySign(startIndex[0] ,
startIndex[1] , startIndex[2],sign);
        Vec3f offset = nextVoxelConner - startPoint;
        offset.Set(fabs(offset.x()), fabs(offset.y()), fabs(offset.z()));
        Vec3f percent(offset.x() / gridStep.x(), offset.y() / gridStep.y(),
offset.z() / gridStep.z());
        Vec3f toffset = percent * dt;
        Vec3f tnext(toffset.x()+tmin,toffset.y()+tmin,toffset.z()+tmin);
```

```
        mi.setTnext(tnext);

        //set normal
        Vec3f voxelMin = getVoxelMinByIndex(startIndex[0], startIndex[1], 
startIndex[2]);
        Vec3f offset2 =  startPoint- voxelMin;
        offset2.Set(fabs(offset2.x()), fabs(offset2.y()), fabs(offset2.z()));
        Vec3f percent2(offset2.x() / gridStep.x(), offset2.y() / gridStep.y(), 
offset2.z() / gridStep.z());

        int maxI;
        float temp2 = -INFINITY;
        for (int i = 0; i < 3; i++)
        {
            if (percent2[i] > temp2)
            {
                temp2 = percent2[i];
                maxI= i;
            }
        }
        mi.setNormal(maxI);

    }
```

1.6.3 MarchingInfo::nextCell

该函数选择三个方向tnext中最小的一个作为新的tmin，并更新tnext、normal

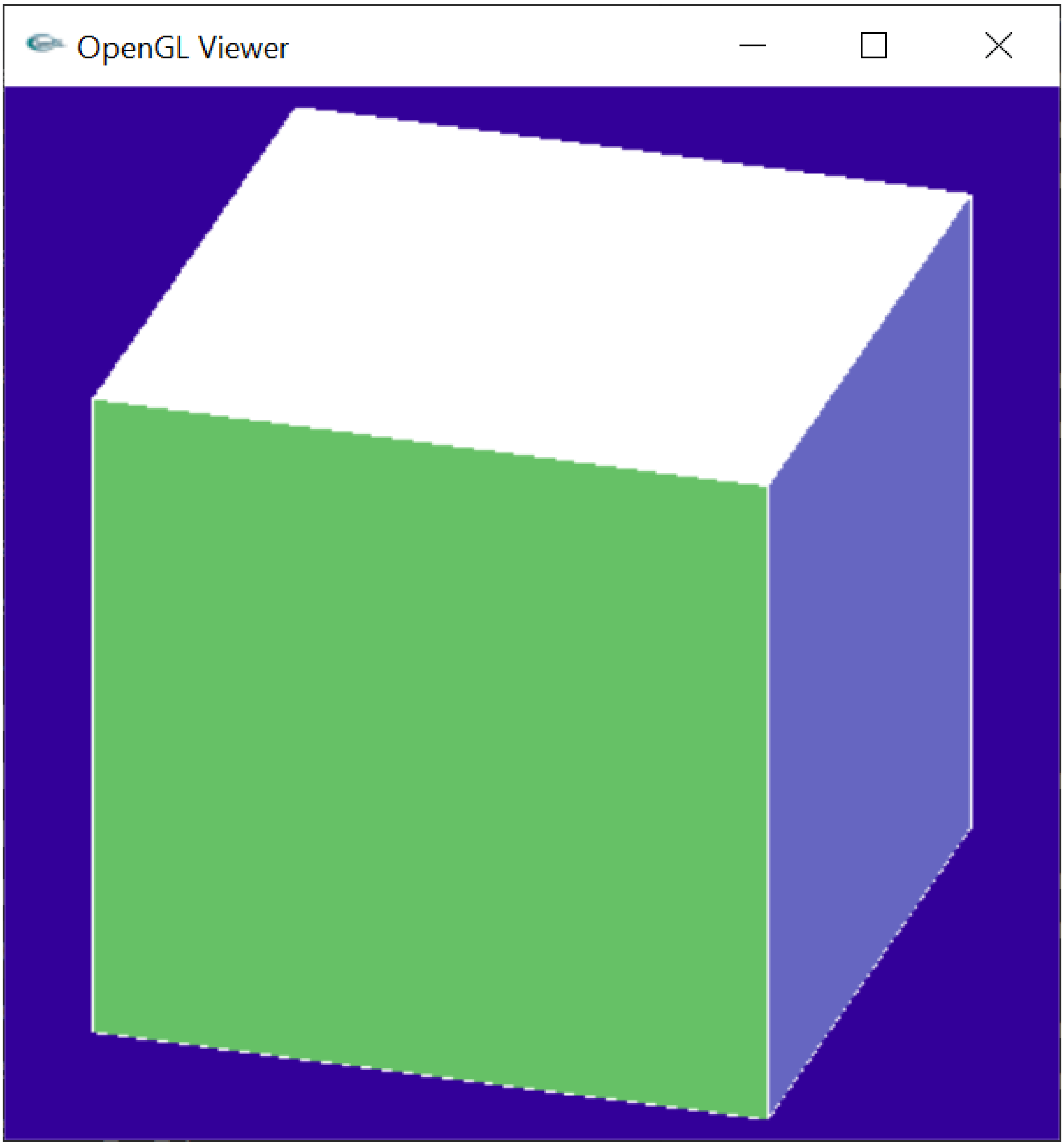
```
    //no dealing with offset
    void nextCell()
    {
        int minI;
        float temp=INFINITY;
        for (int i = 0; i < 3; i++)
        {
            if (tnext[i] < temp)
            {
                temp = tnext[i];
                minI = i;
            }
        }
        gridIndex[minI] += sign[minI];
        tmin = tnext[minI];
        tnext.Set(minI, tnext[minI] + dt[minI]);
        normal = axisNormals[minI] * sign[minI];
        axis = minI;
    }
```

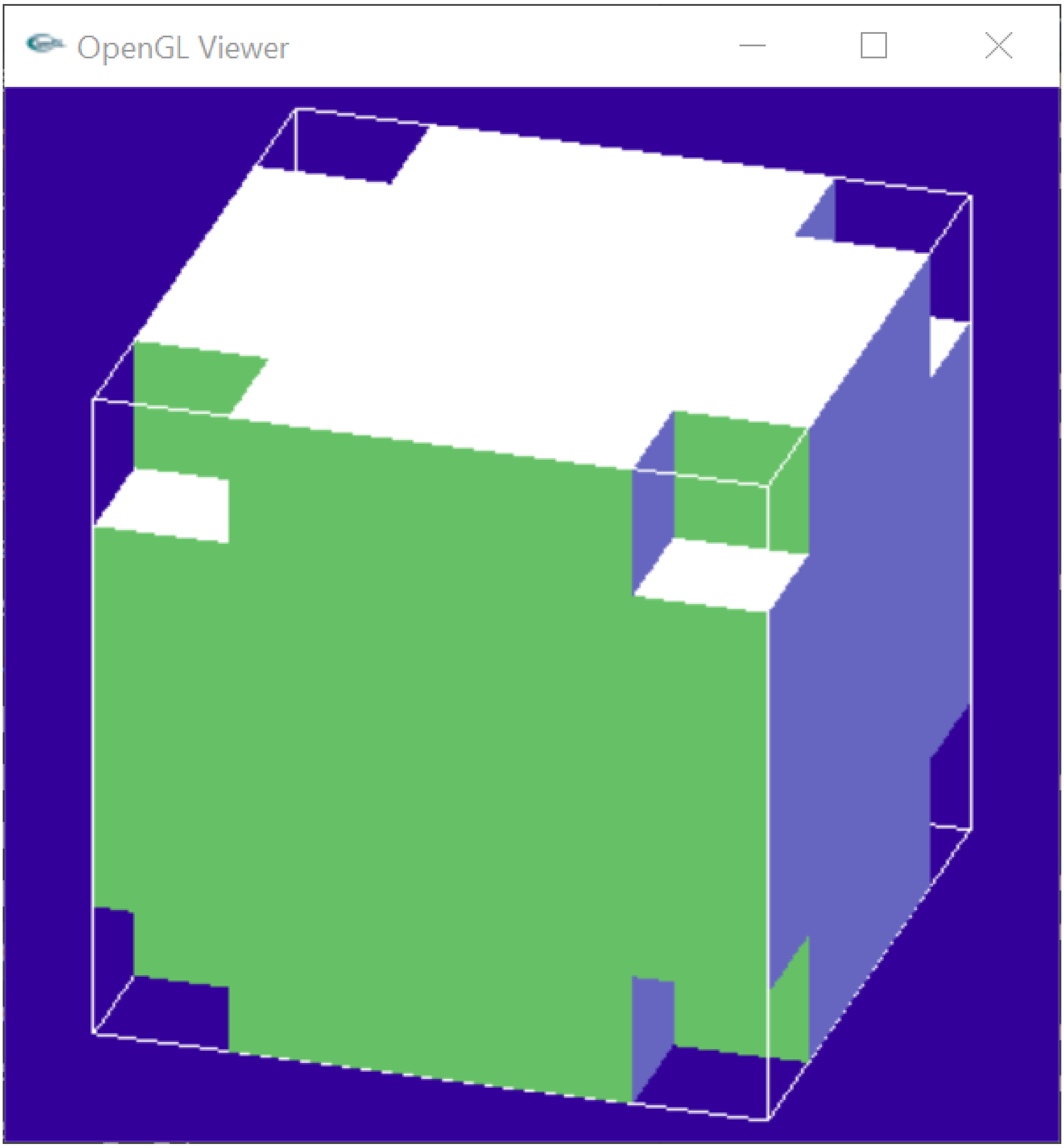
1.7 加入RayTree代码

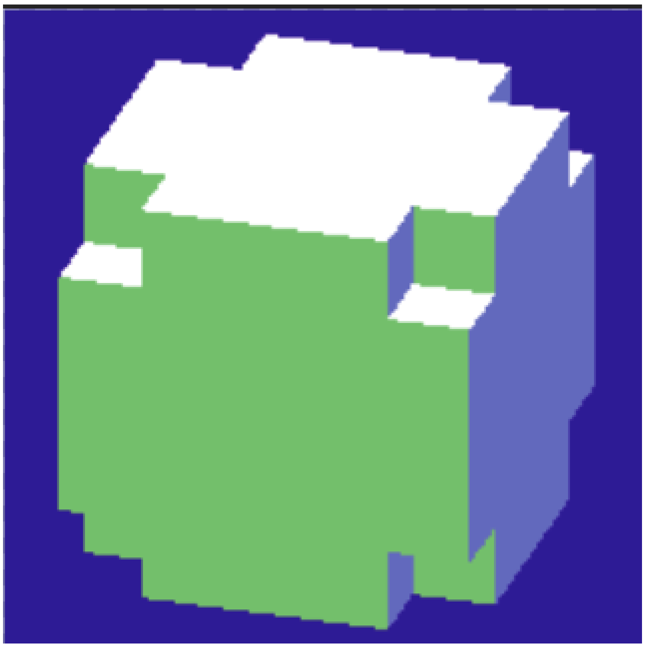
加入RayTree代码以实现Grid Ray Marching算法的可视化，便于debug

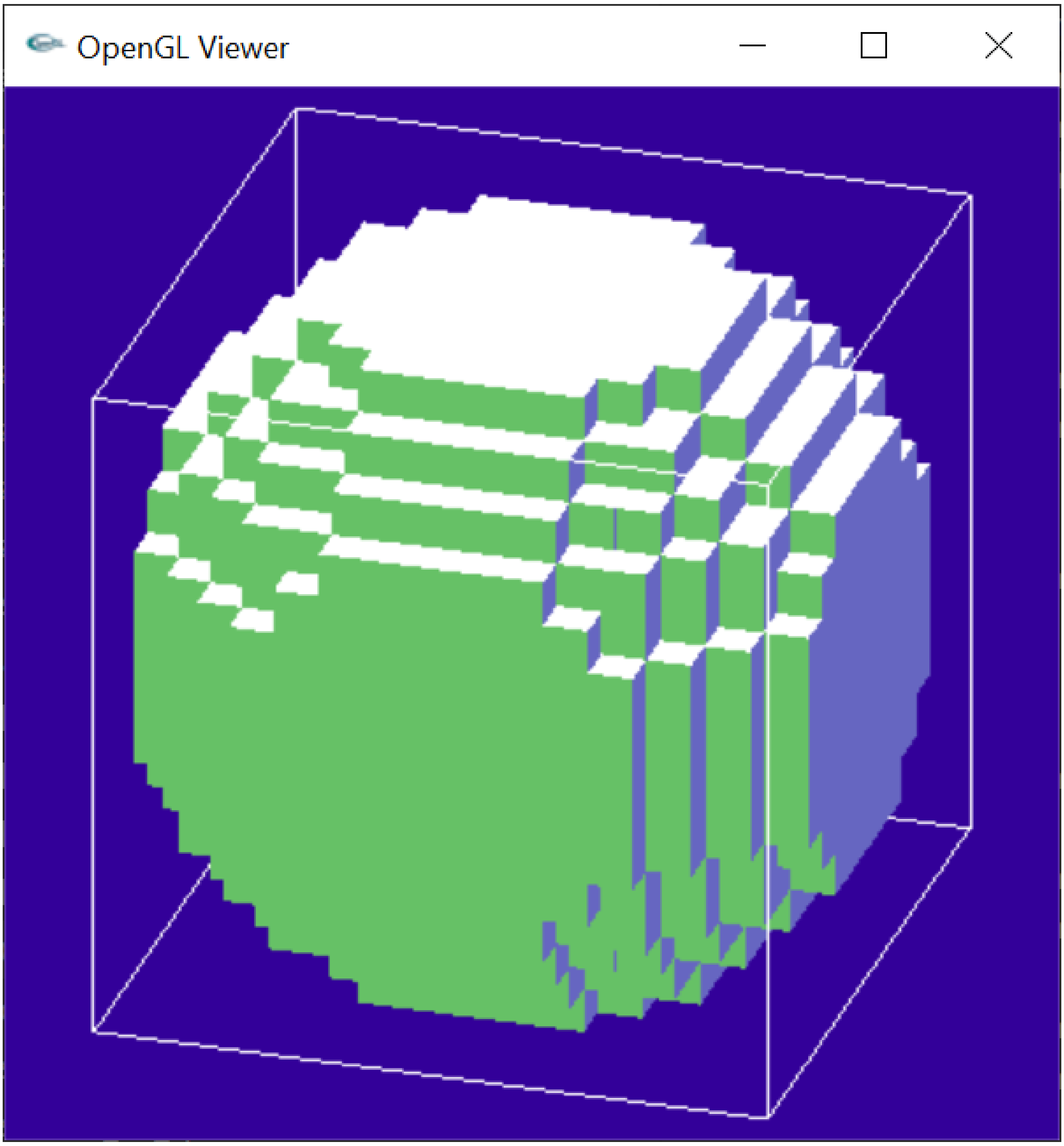
2 实验结果

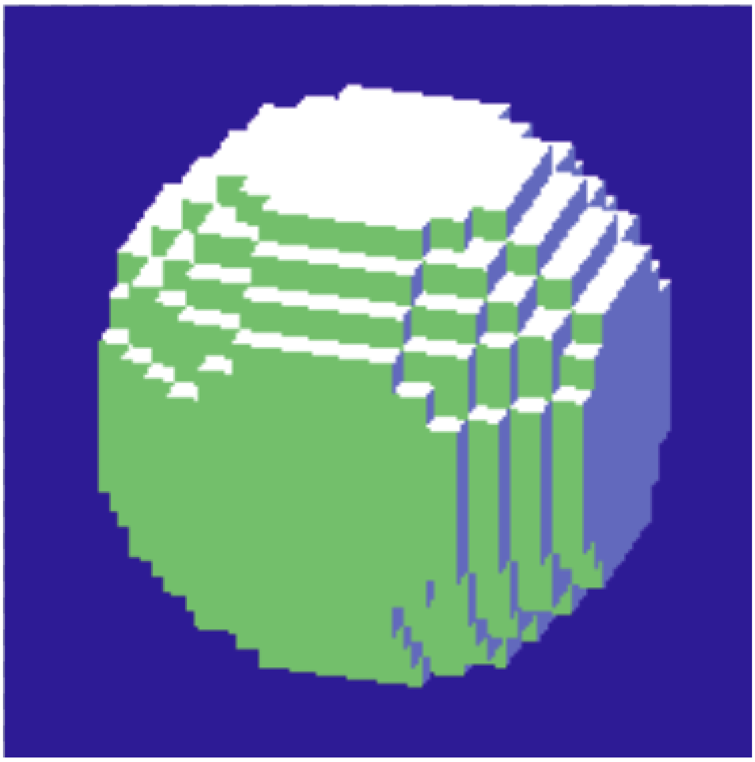
```
raytracer -input scene5_01_sphere.txt -size 200 200 -output output5_01a.tga -gui
-grid 1 1 1 -visualize_grid
raytracer -input scene5_01_sphere.txt -size 200 200 -output output5_01b.tga -gui
-grid 5 5 5 -visualize_grid
raytracer -input scene5_01_sphere.txt -size 200 200 -output output5_01c.tga -gui
-grid 15 15 15 -visualize_grid
```



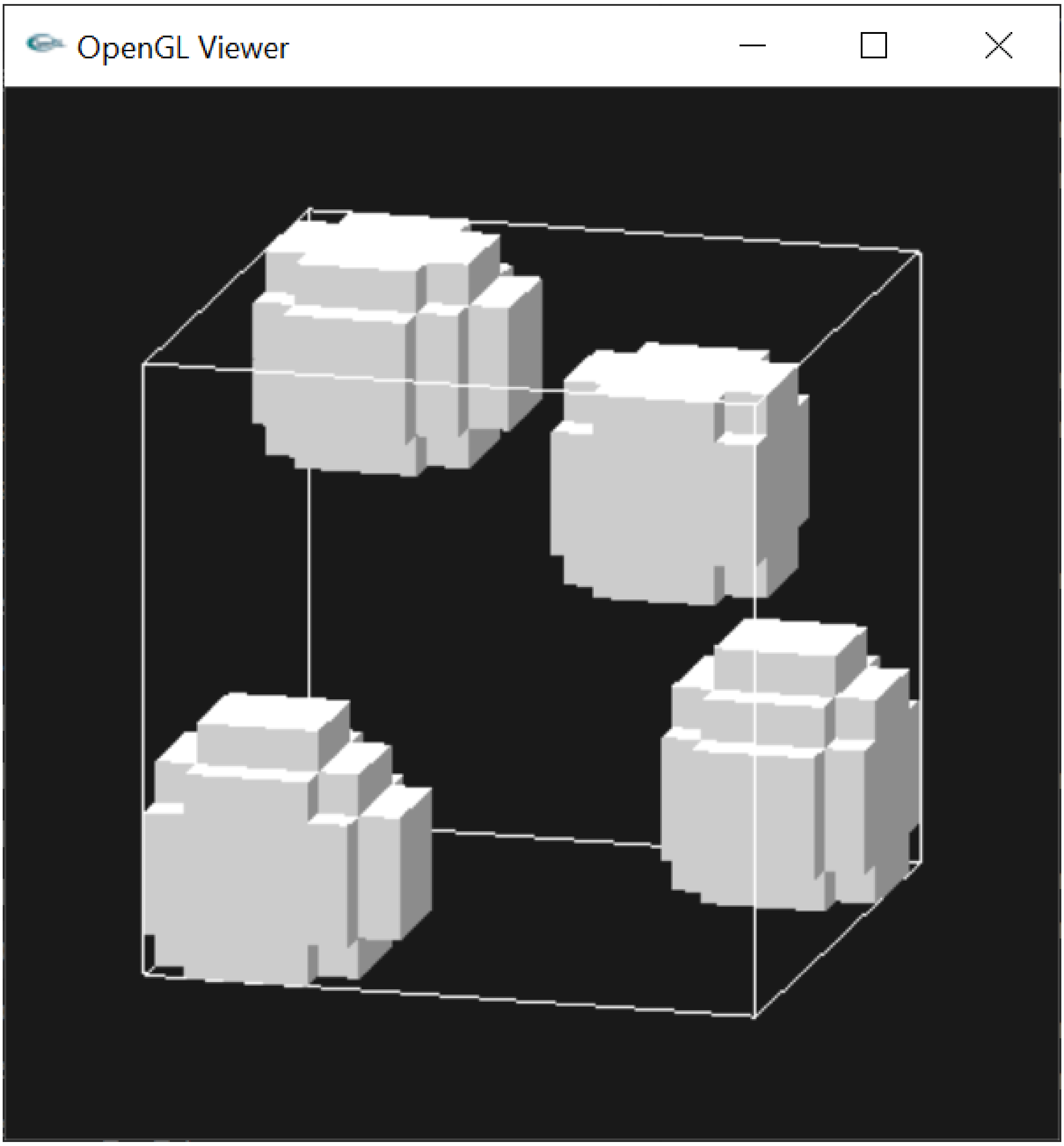

OpenGL Viewer

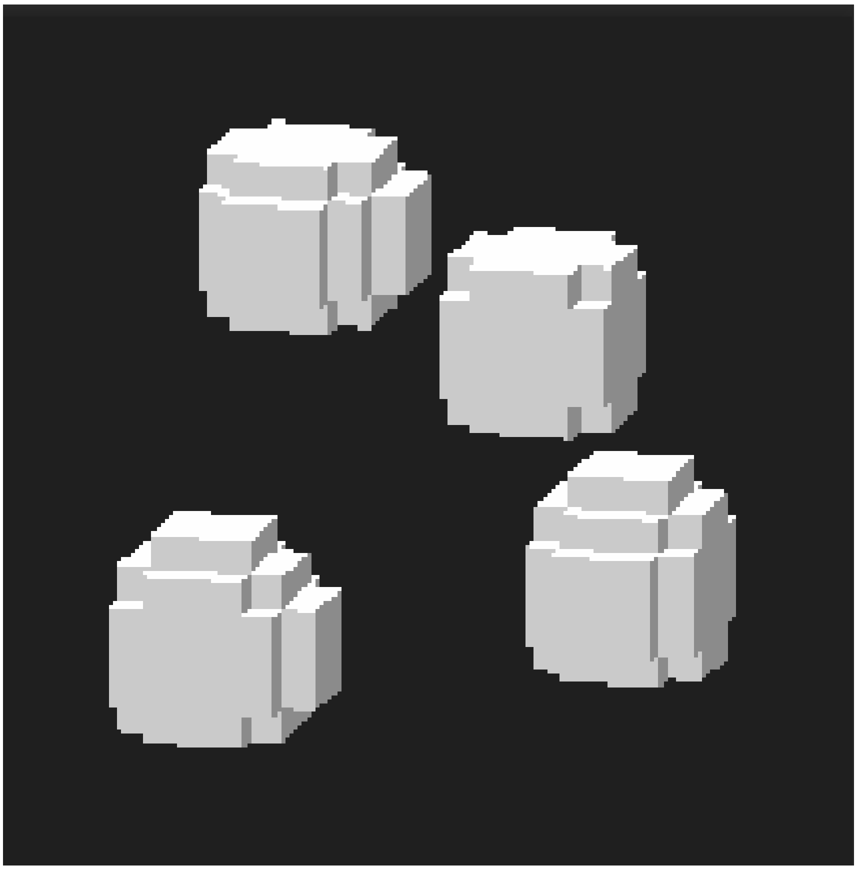



OpenGL Viewer

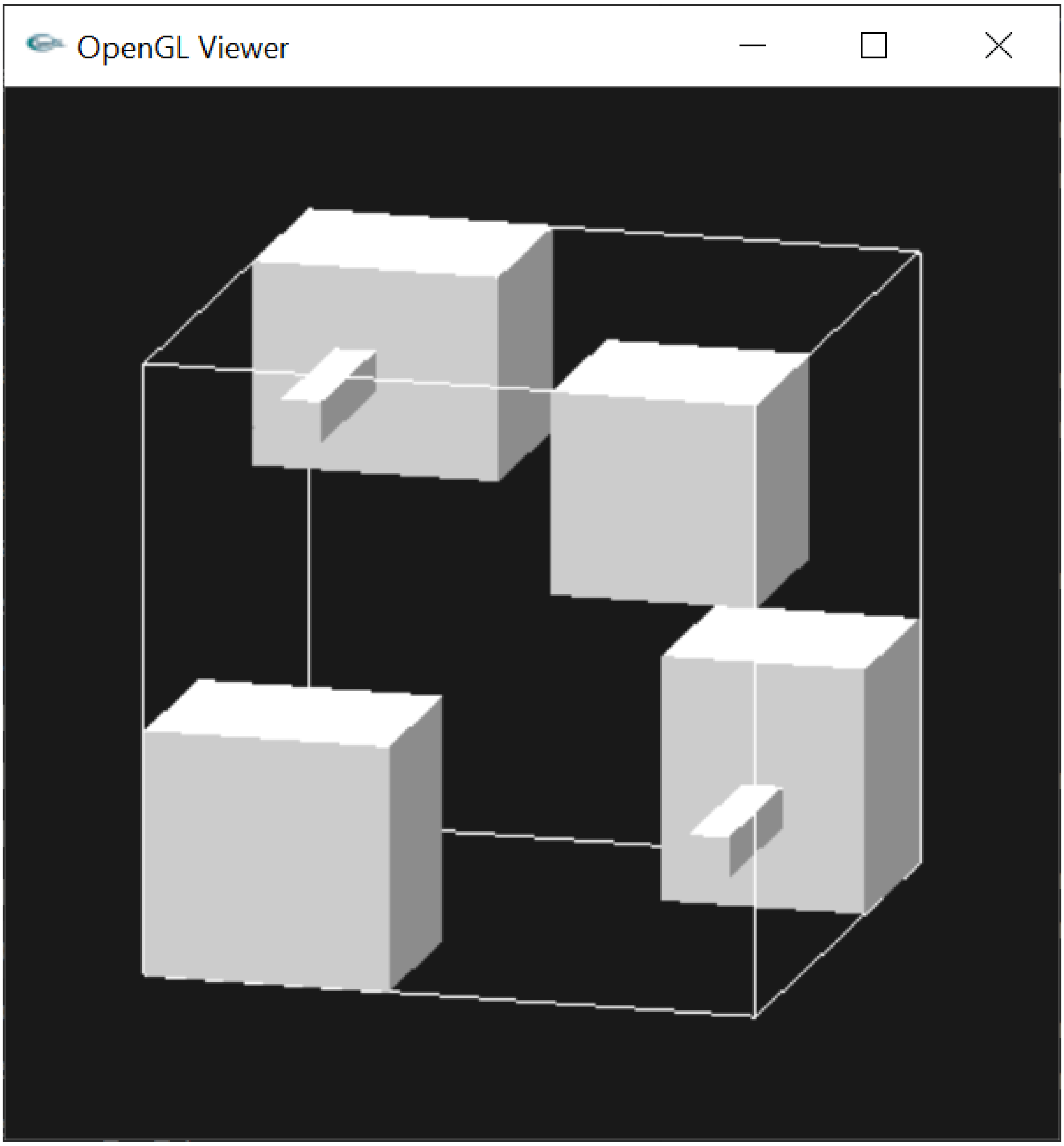


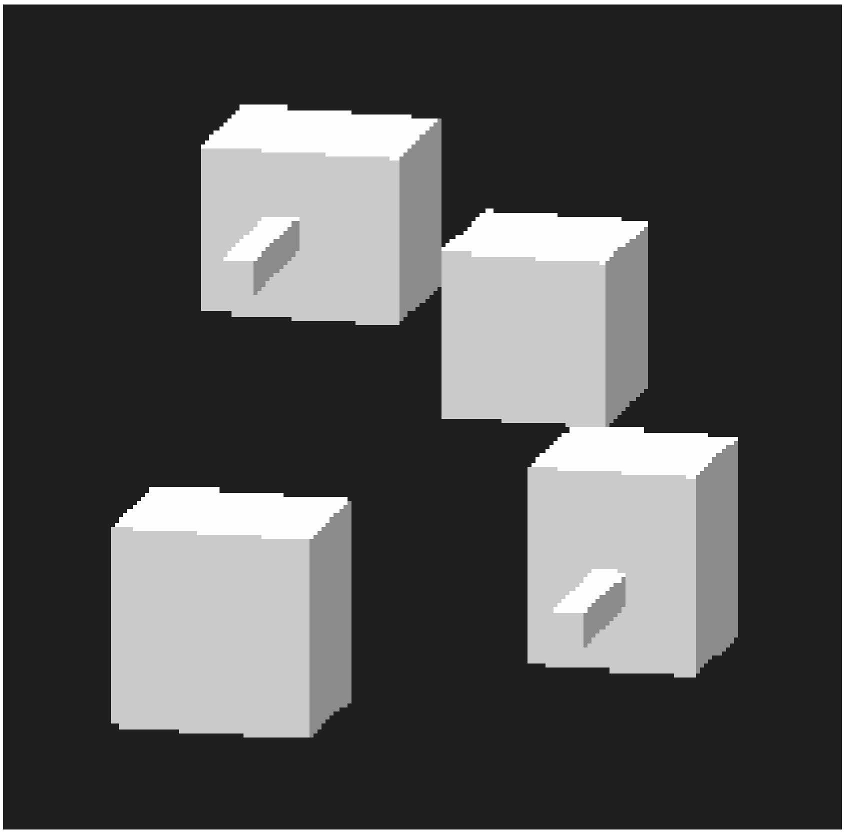
```
raytracer -input scene5_02_spheres.txt -size 200 200 -output output5_02a.tga -gui -grid 15 15 15 -visualize_grid
raytracer -input scene5_02_spheres.txt -size 200 200 -output output5_02b.tga -gui -grid 15 15 3 -visualize_grid
```


OpenGL Viewer

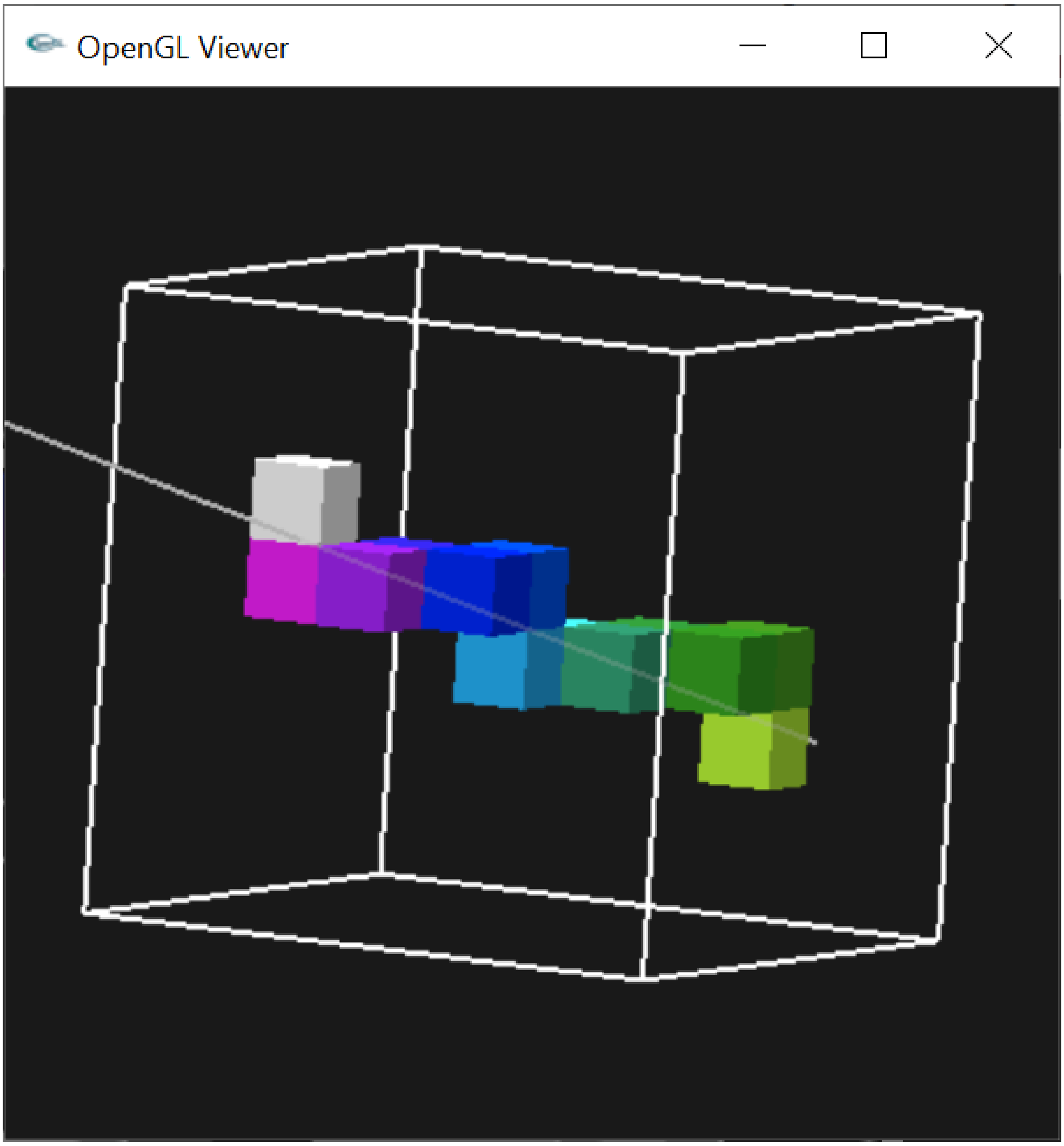


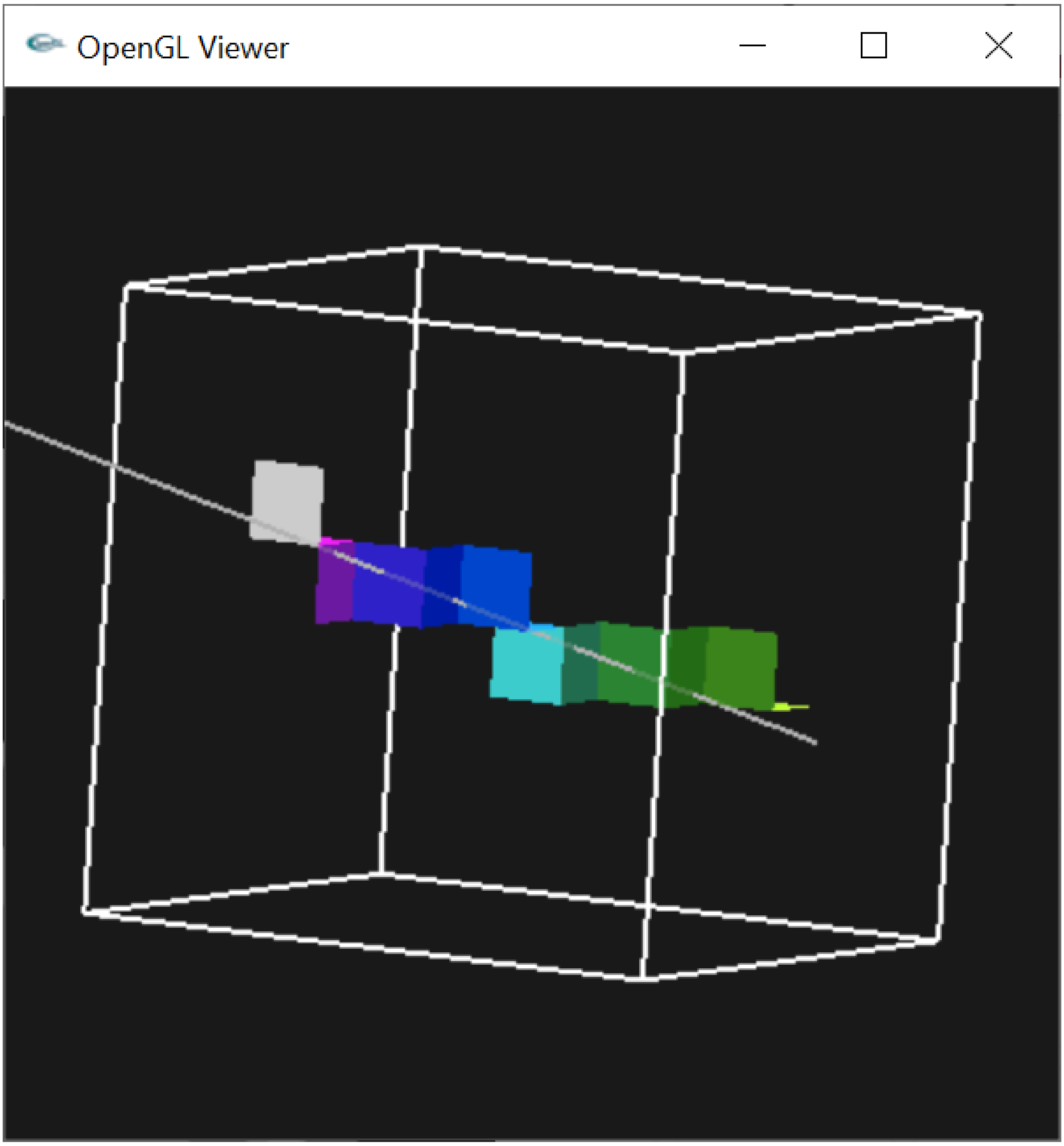
因Sphere::insertIntoGrid保守算法与原作者不同，结果有所差别


OpenGL Viewer

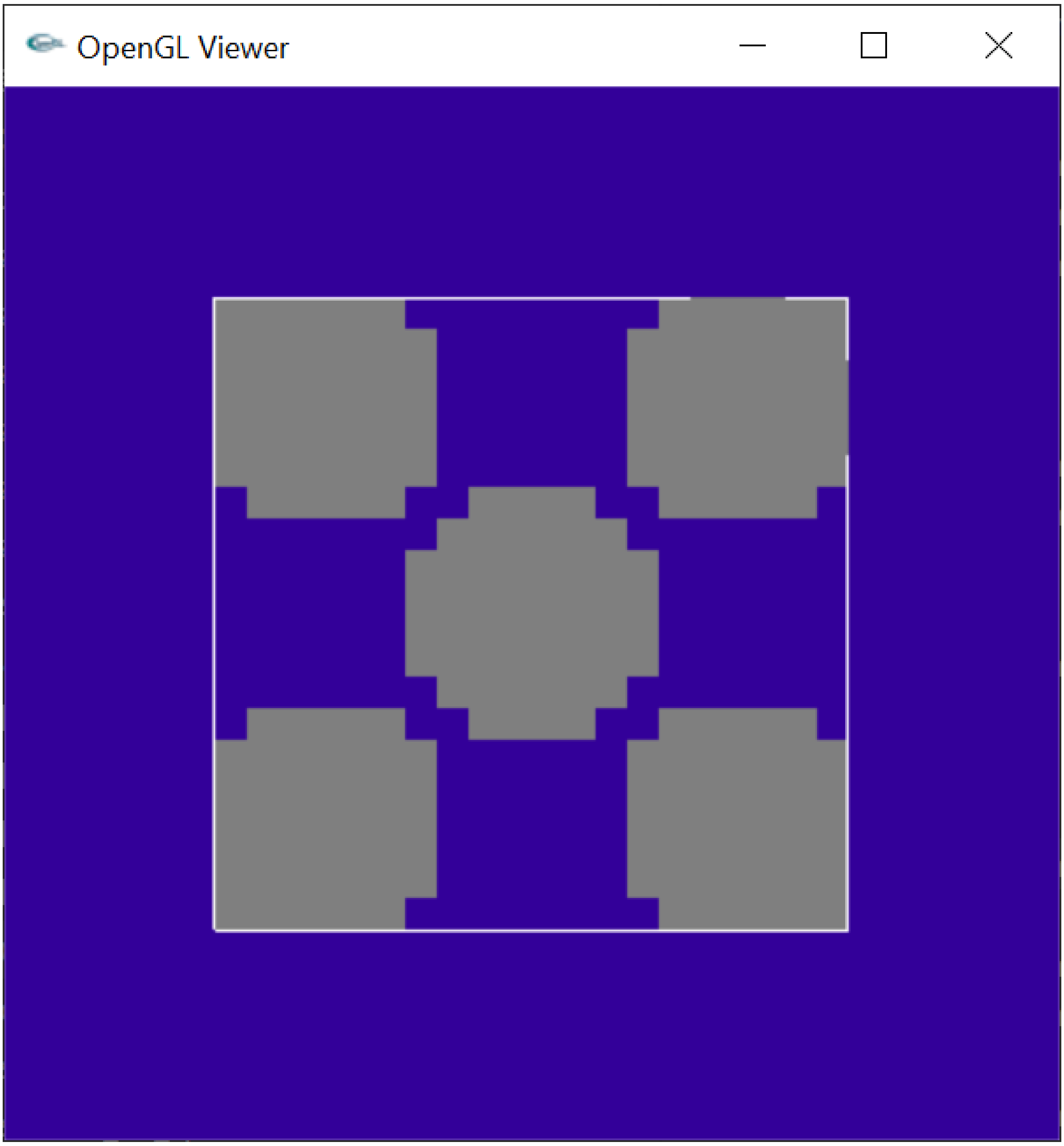


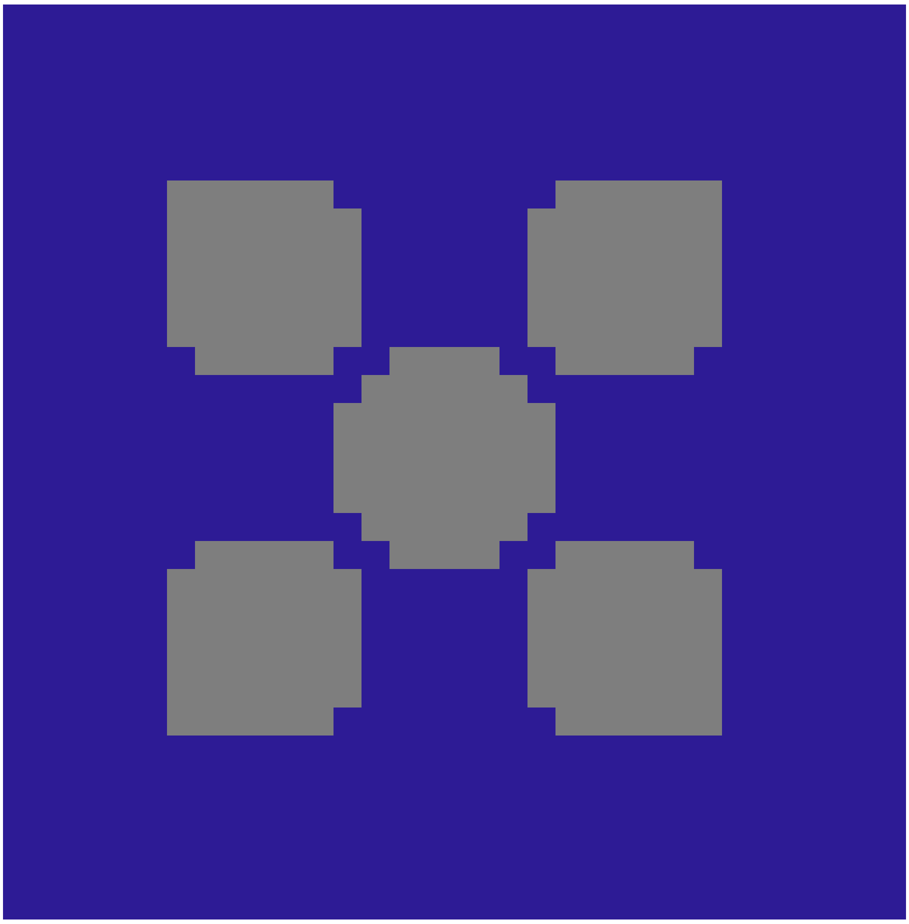
```
raytracer -input scene5_02_spheres.txt -size 200 200 -gui -grid 8 8 8 -
visualize_grid
```


OpenGL Viewer

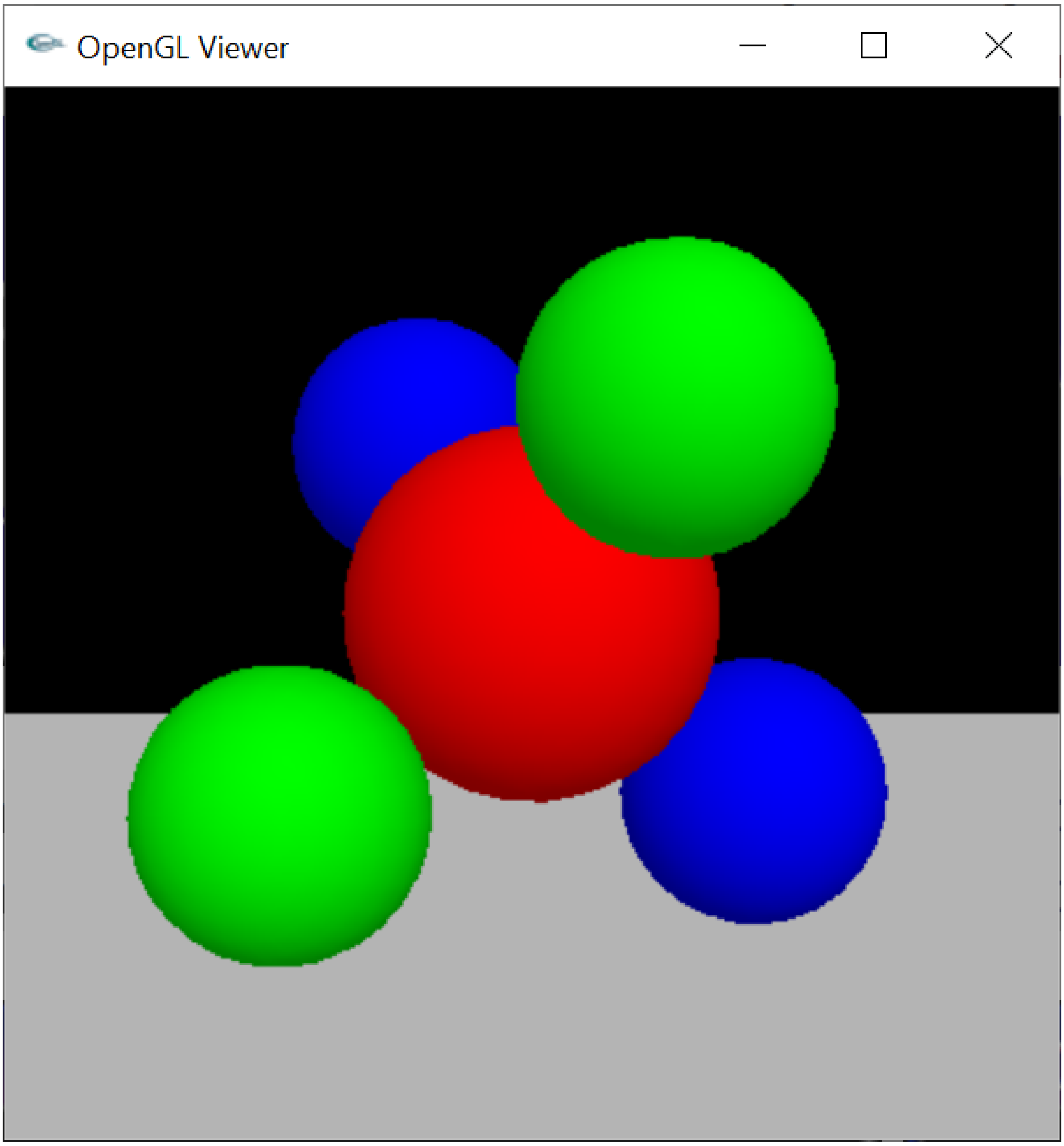


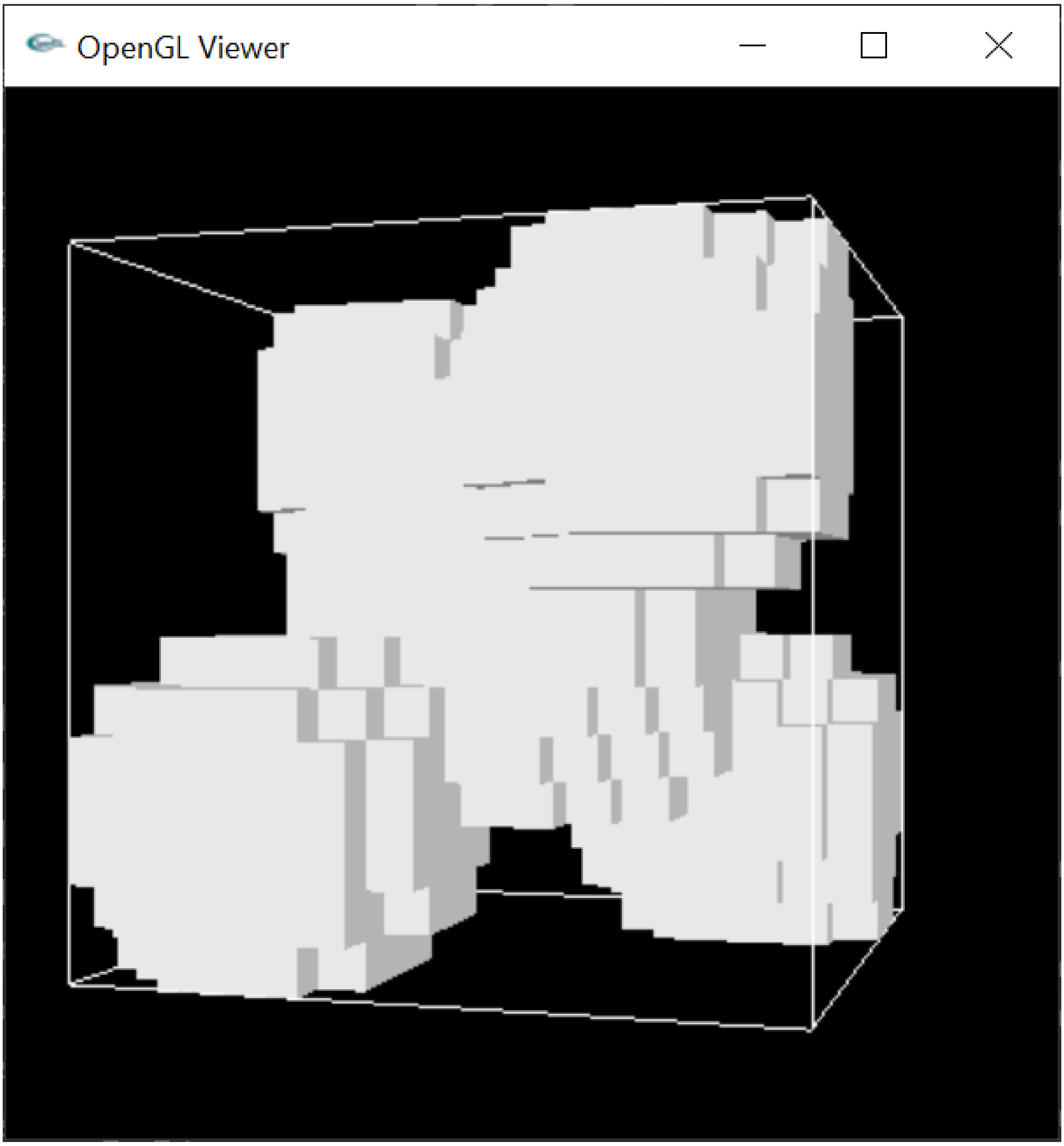

```
raytracer -input scene5_03_offcenter_spheres.txt -size 200 200 -output
output5_03.tga -gui -grid 20 20 20 -visualize_grid
```

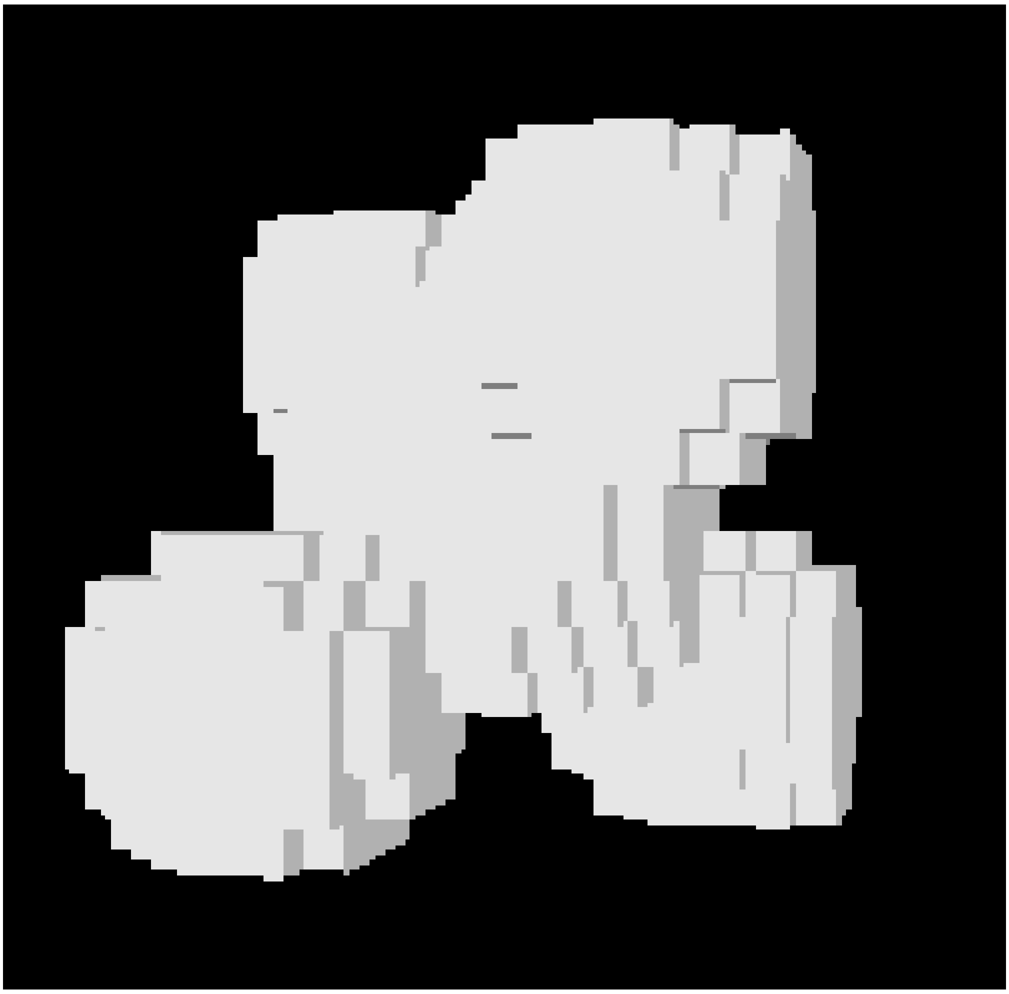

OpenGL Viewer



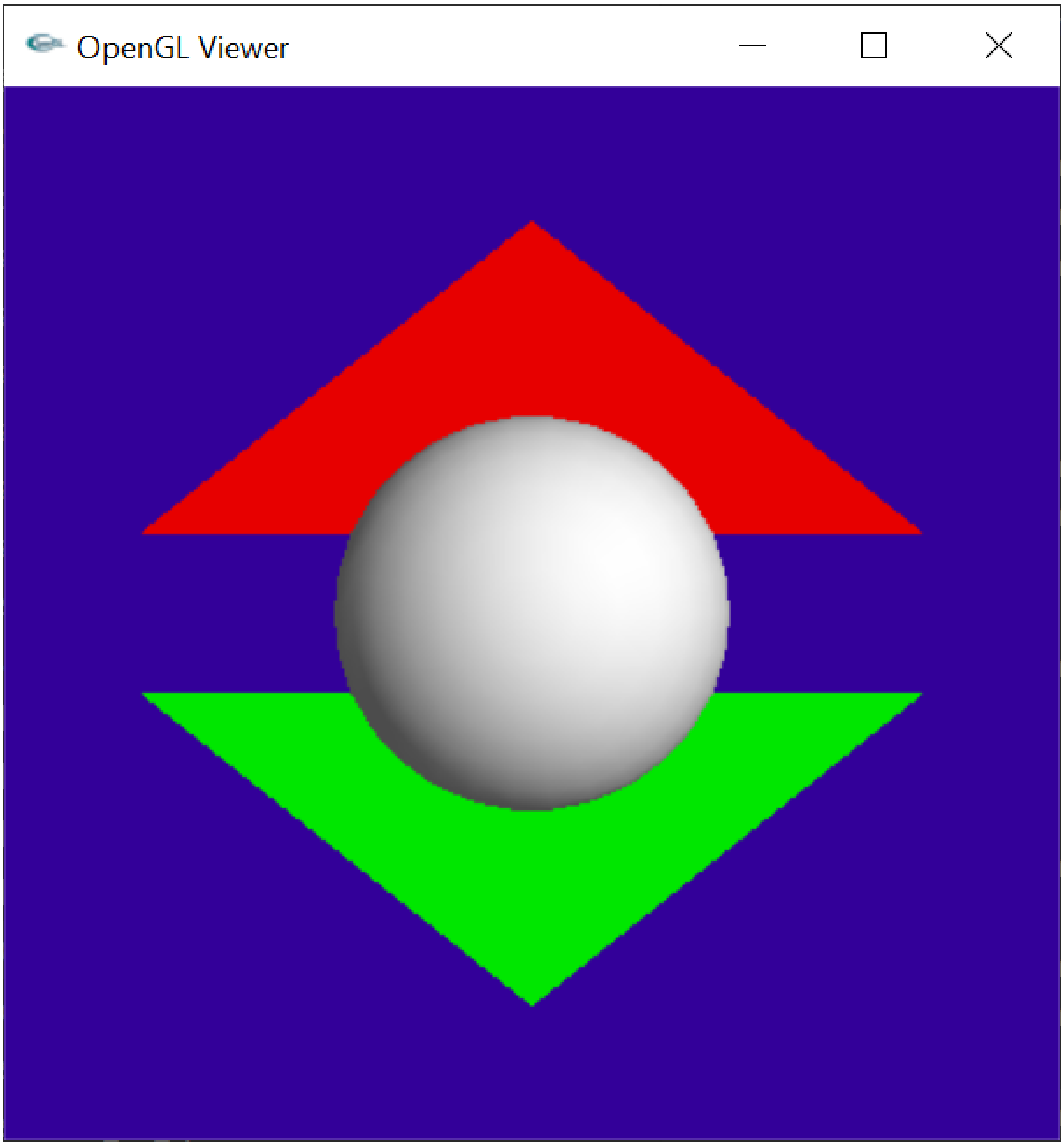
```
raytracer -input scene5_04_plane_test.txt -size 200 200 -gui -tessellation 30 15
-gouraud
raytracer -input scene5_04_plane_test.txt -size 200 200 -output output5_04.tga -
gui -grid 15 15 15 -visualize_grid
```

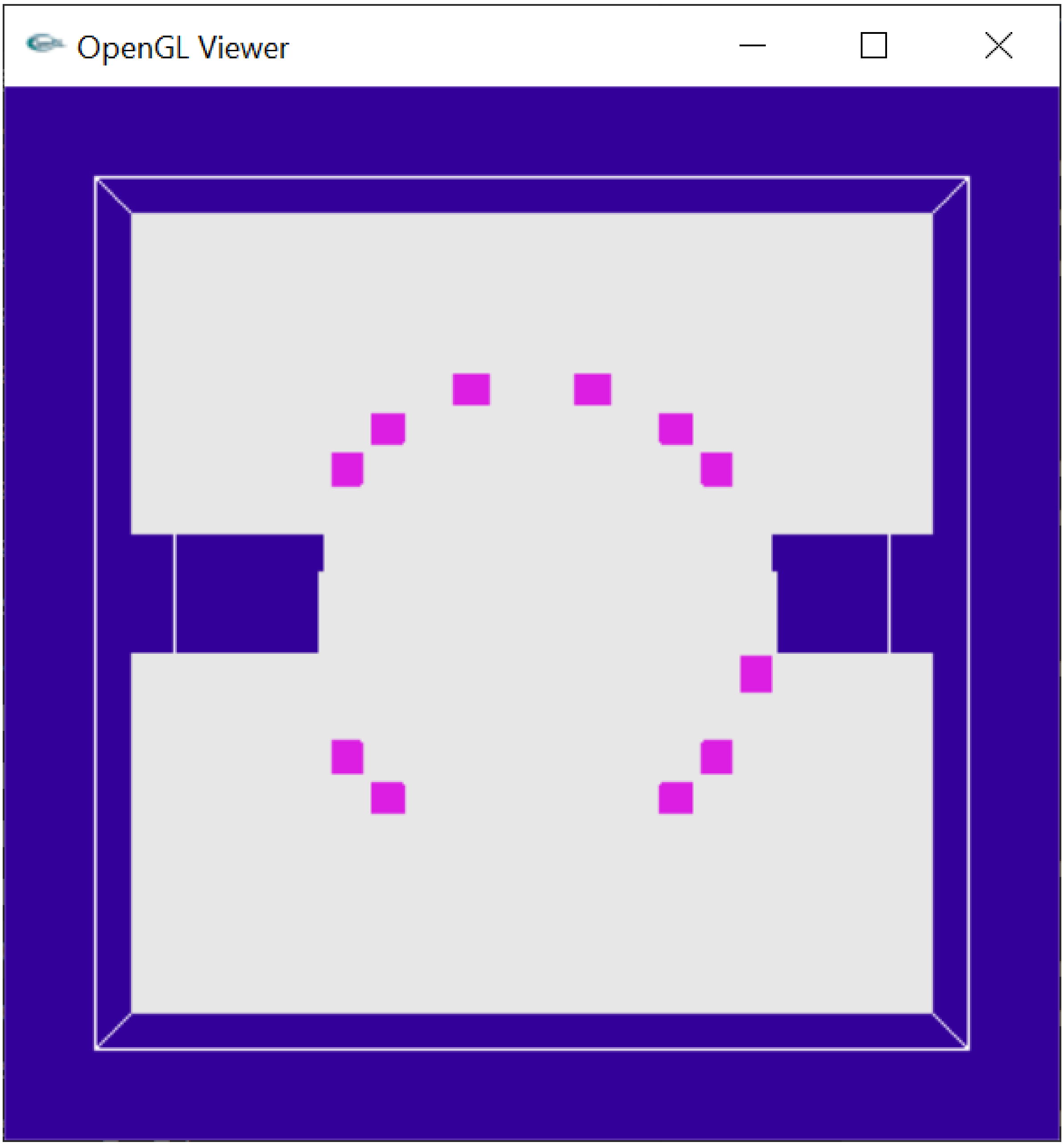

OpenGL Viewer

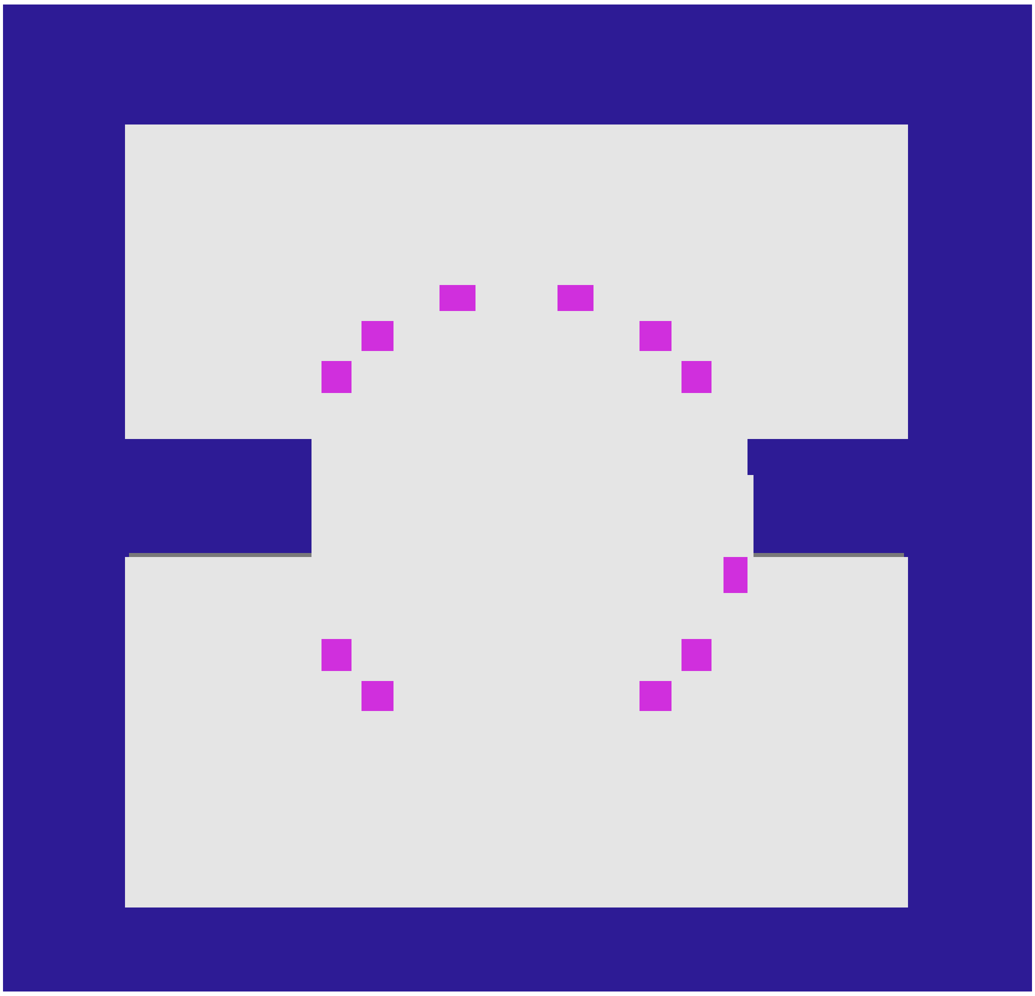

OpenGL Viewer



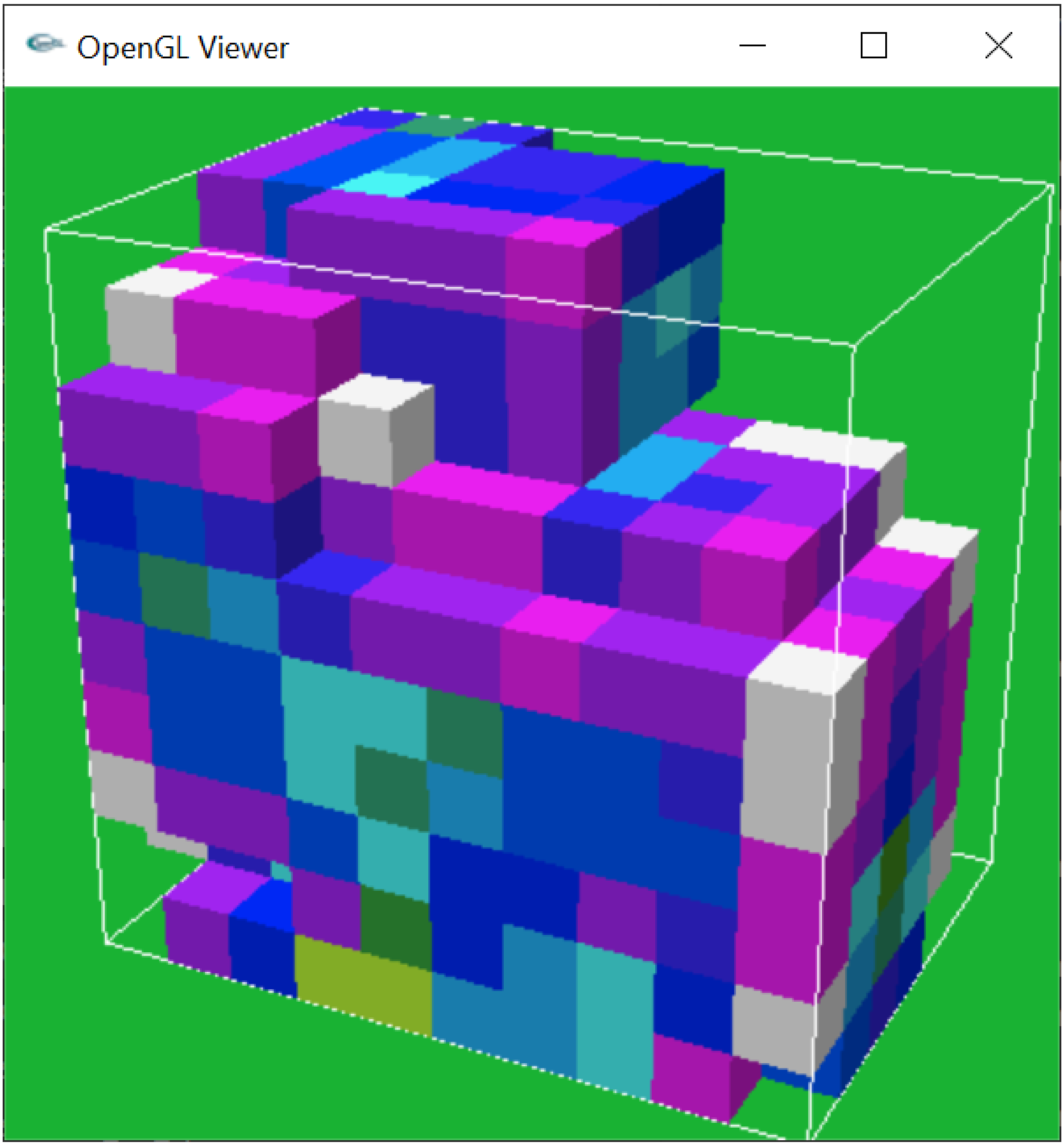
```
raytracer -input scene5_05_sphere_triangles.txt -size 200 200 -gui -tessellation
30 15 -gouraud
raytracer -input scene5_05_sphere_triangles.txt -size 200 200 -output
output5_05.tga -gui -grid 20 20 10 -visualize_grid
```

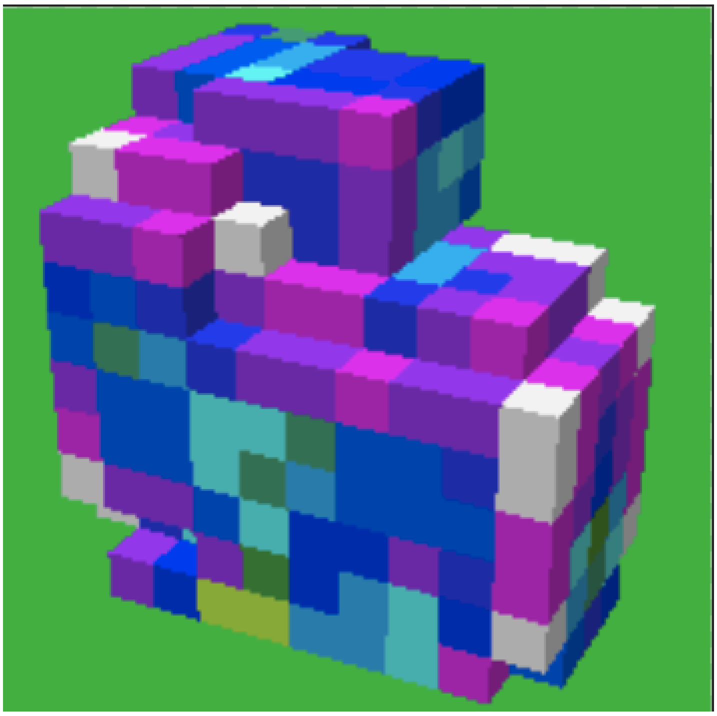

OpenGL Viewer

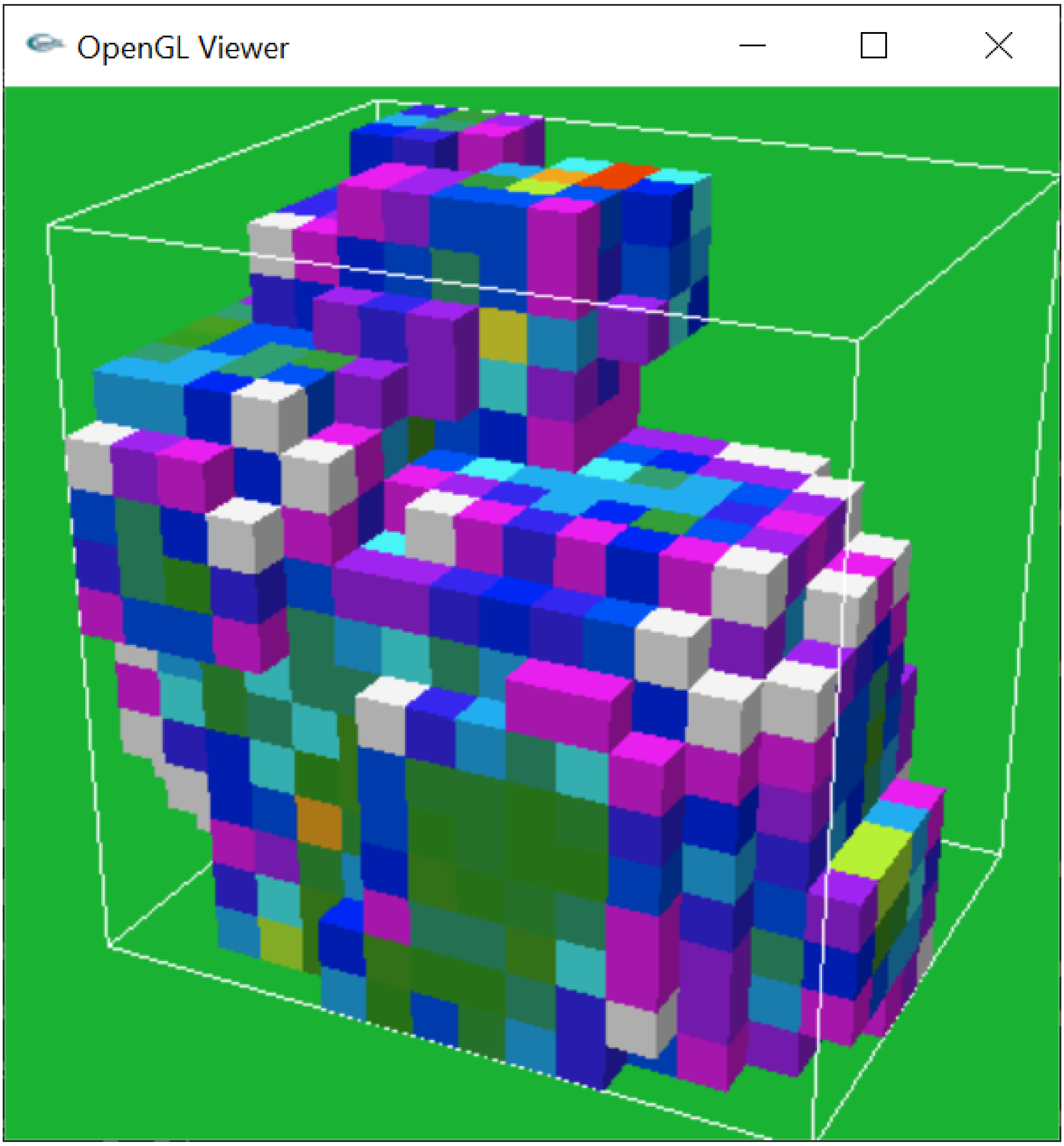

OpenGL Viewer

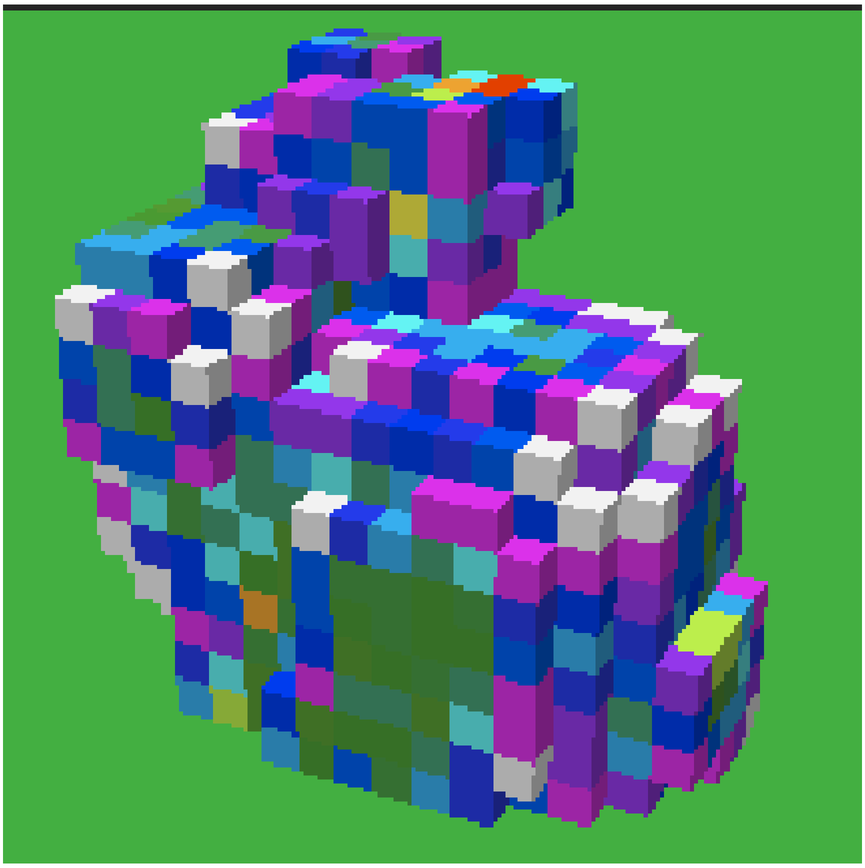


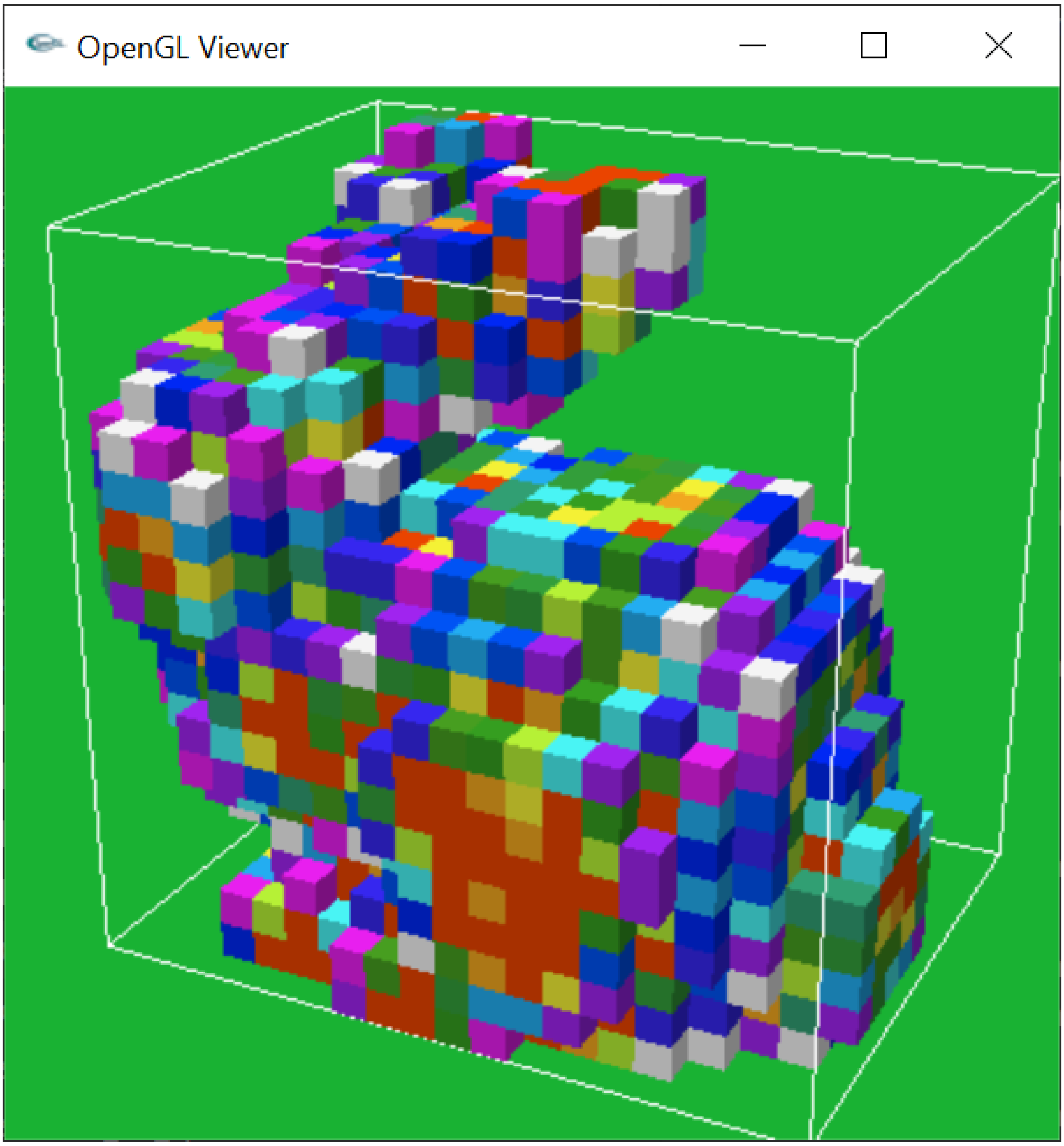
```
raytracer -input scene5_06_bunny_mesh_200.txt -size 200 200 -output
output5_06.tga -gui -grid 10 10 7 -visualize_grid
raytracer -input scene5_07_bunny_mesh_1k.txt -size 200 200 -output
output5_07.tga -gui -grid 15 15 12 -visualize_grid
raytracer -input scene5_08_bunny_mesh_5k.txt -size 200 200 -output
output5_08.tga -gui -grid 20 20 15 -visualize_grid
raytracer -input scene5_09_bunny_mesh_40k.txt -size 200 200 -output
output5_09.tga -gui -grid 40 40 33 -visualize_grid
```

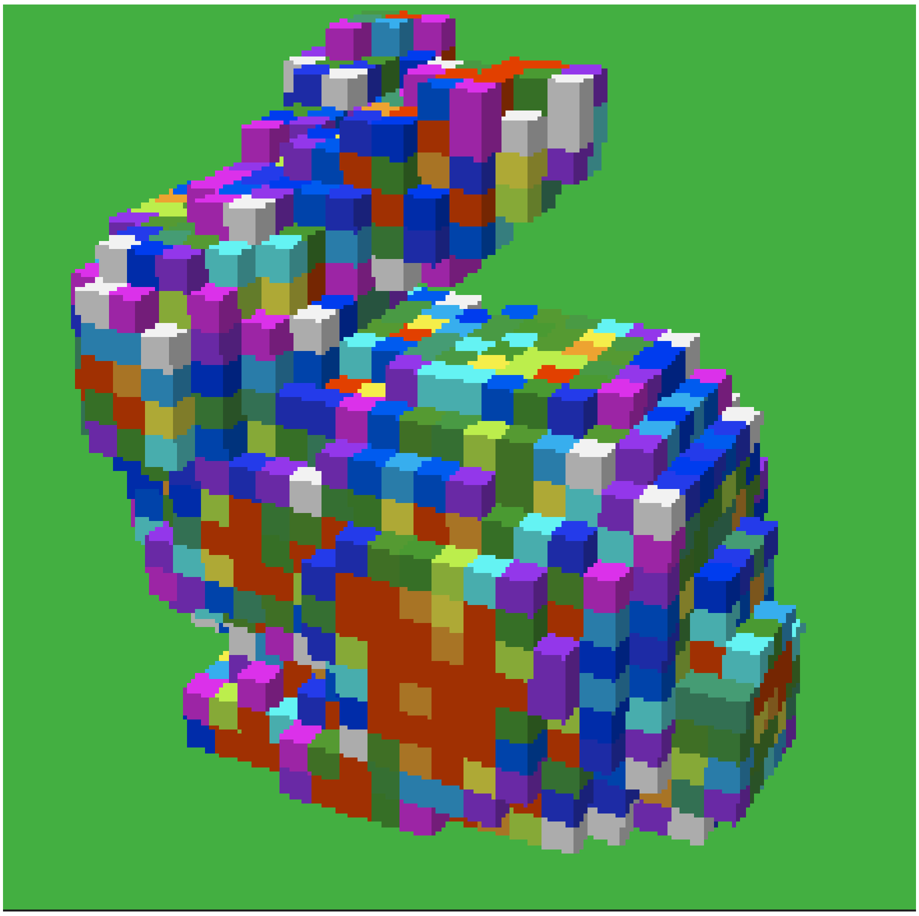

OpenGL Viewer

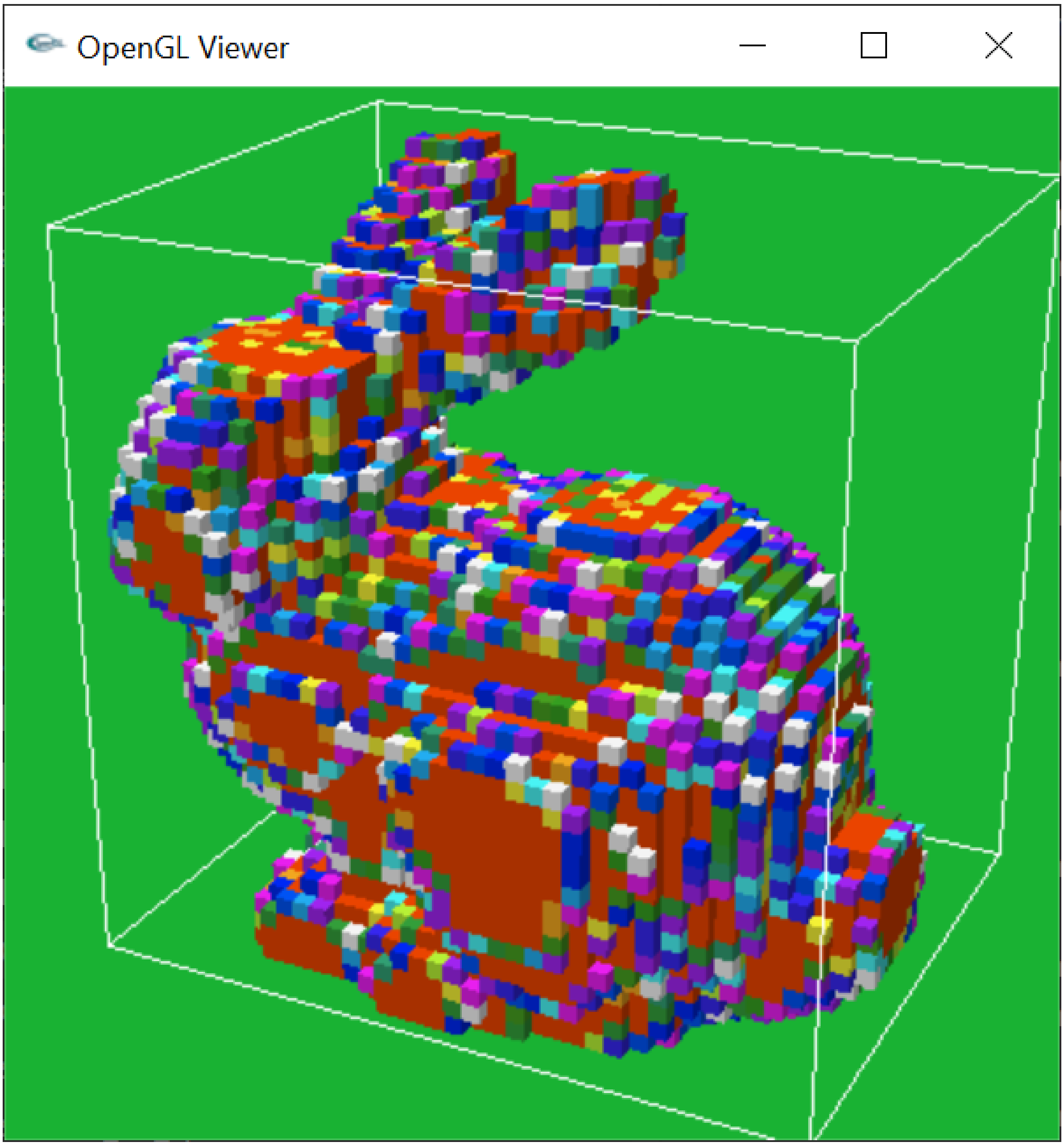


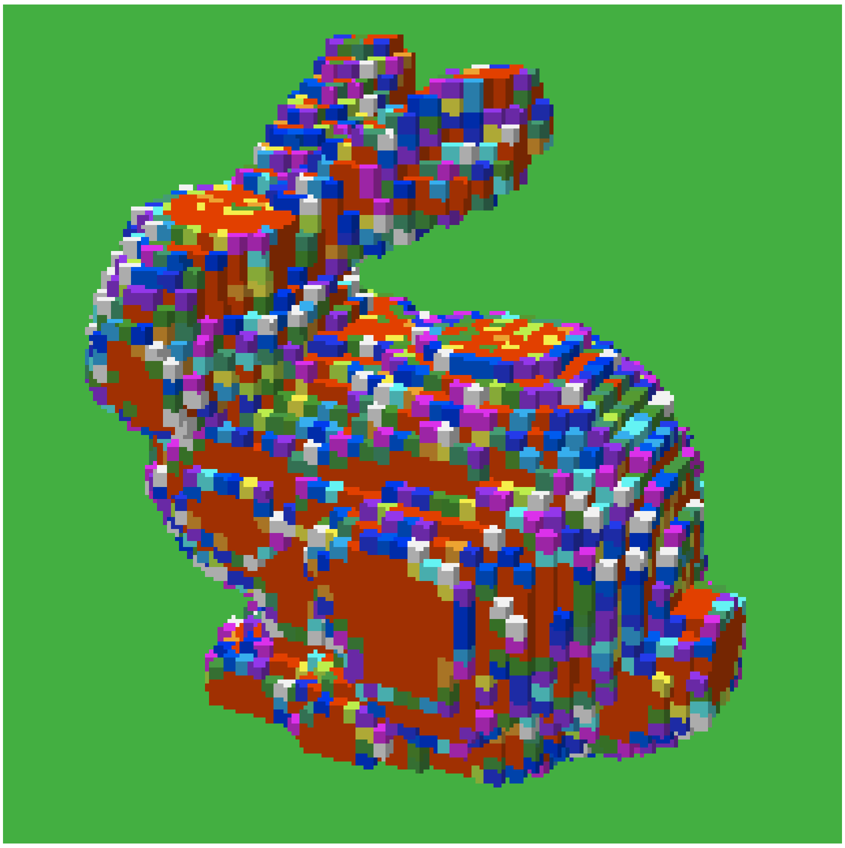

OpenGL Viewer



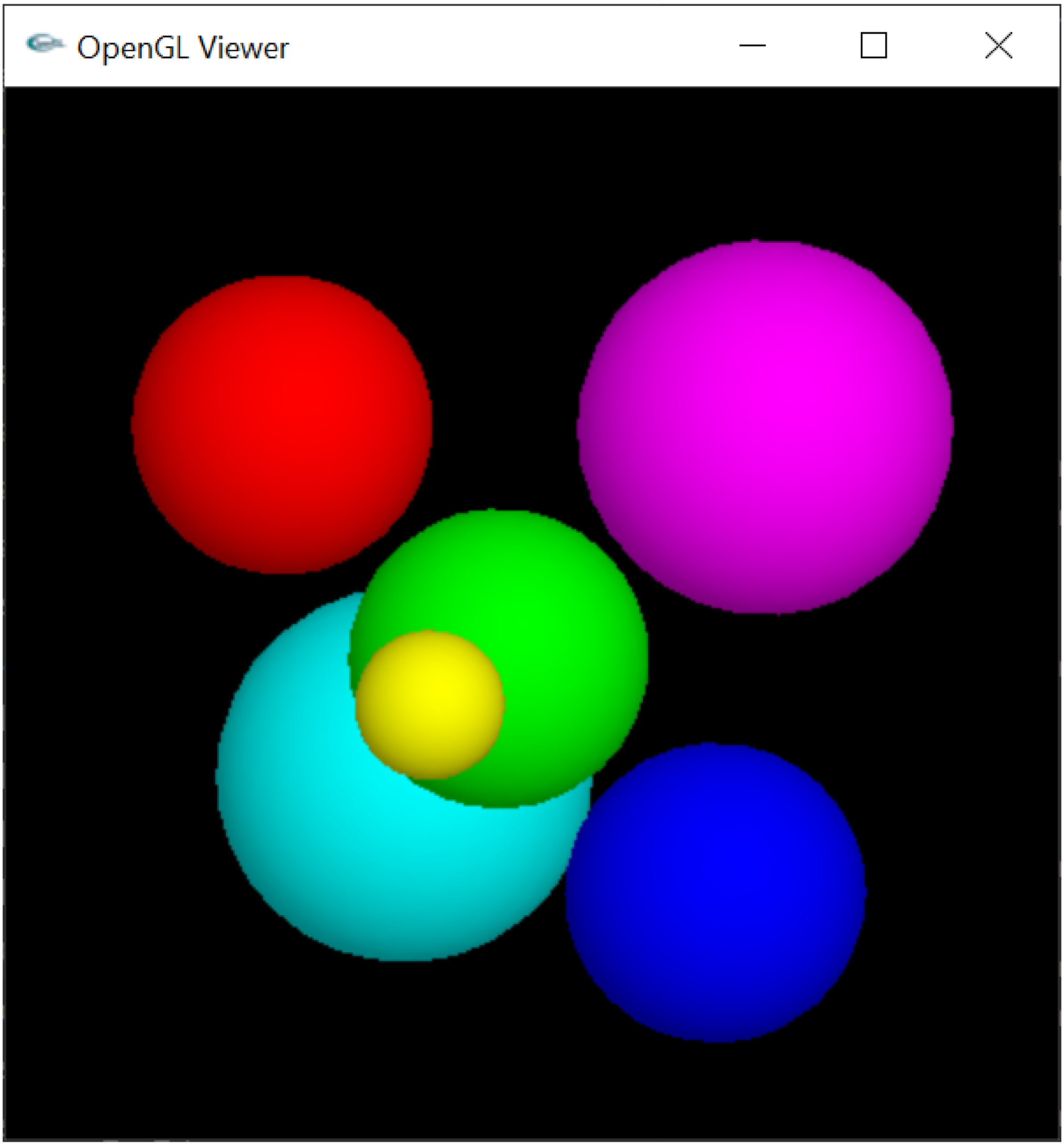

OpenGL Viewer

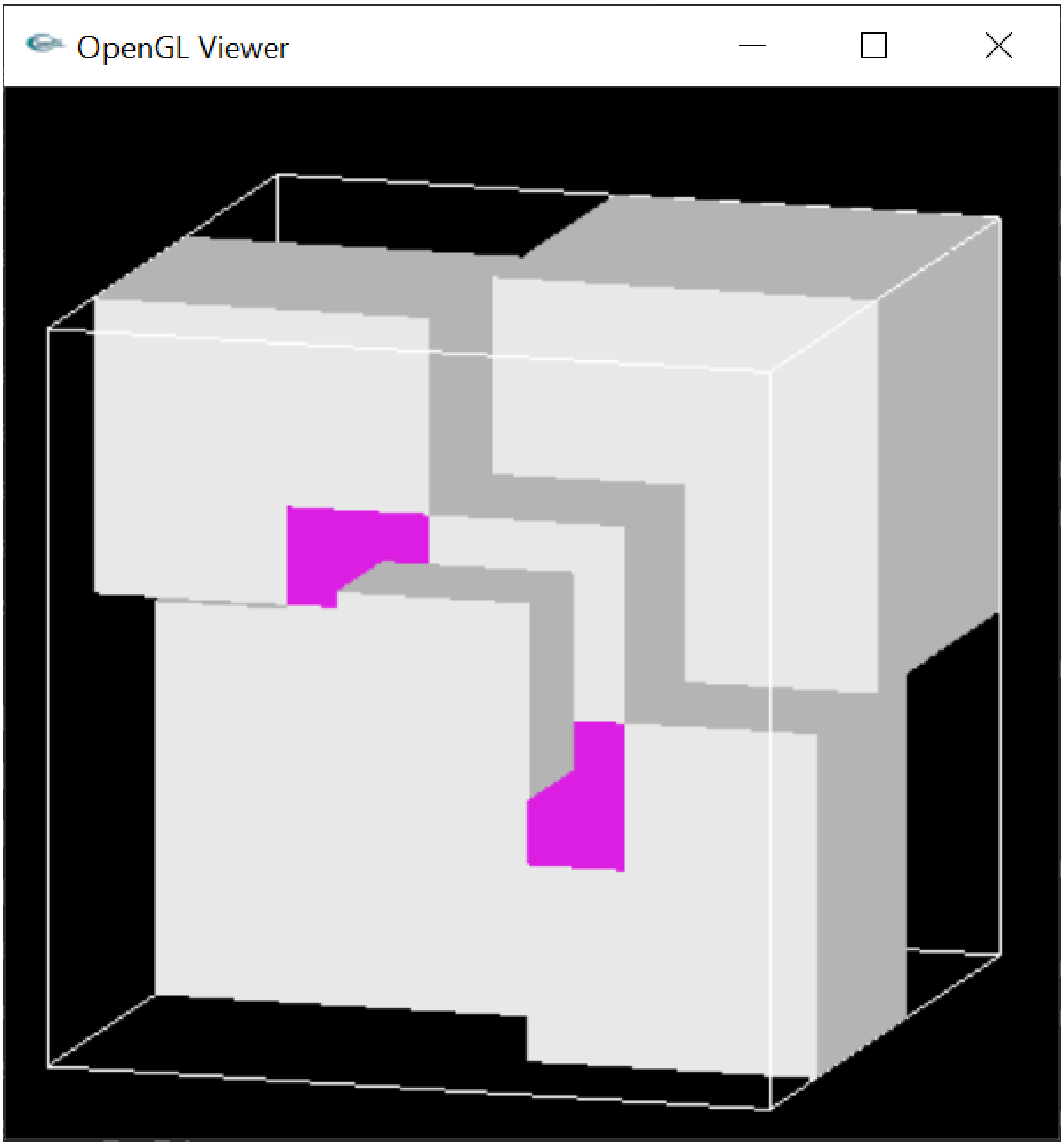


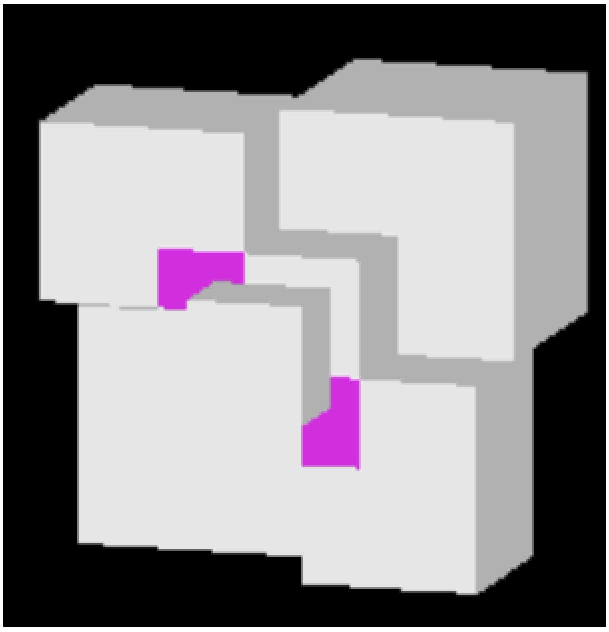

OpenGL Viewer



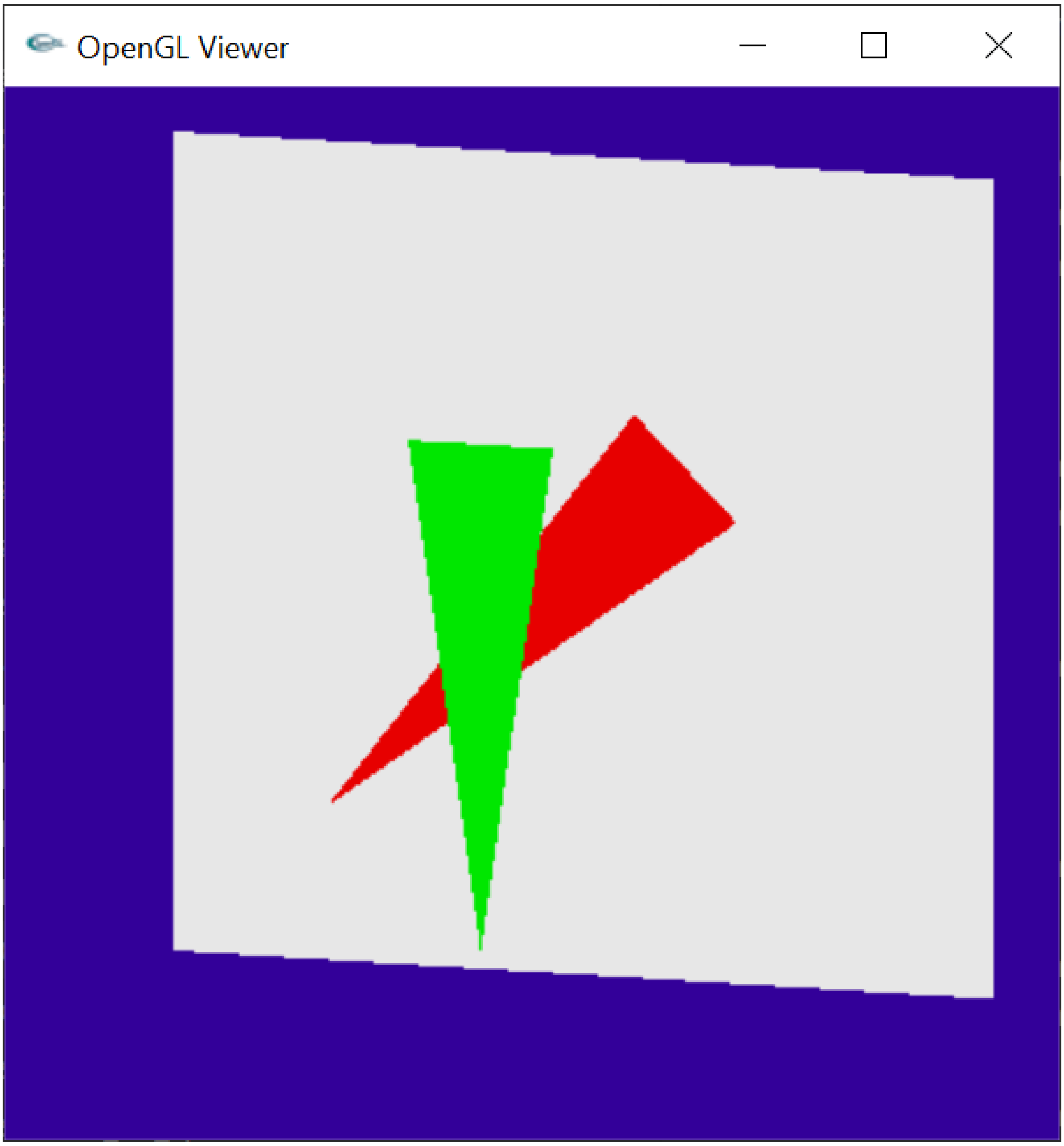
```
raytracer -input scene5_10_scale_translate.txt -size 200 200 -gui -tessellation
30 15 -gouraud
raytracer -input scene5_10_scale_translate.txt -size 200 200 -output
output5_10.tga -gui -grid 15 15 15 -visualize_grid
```

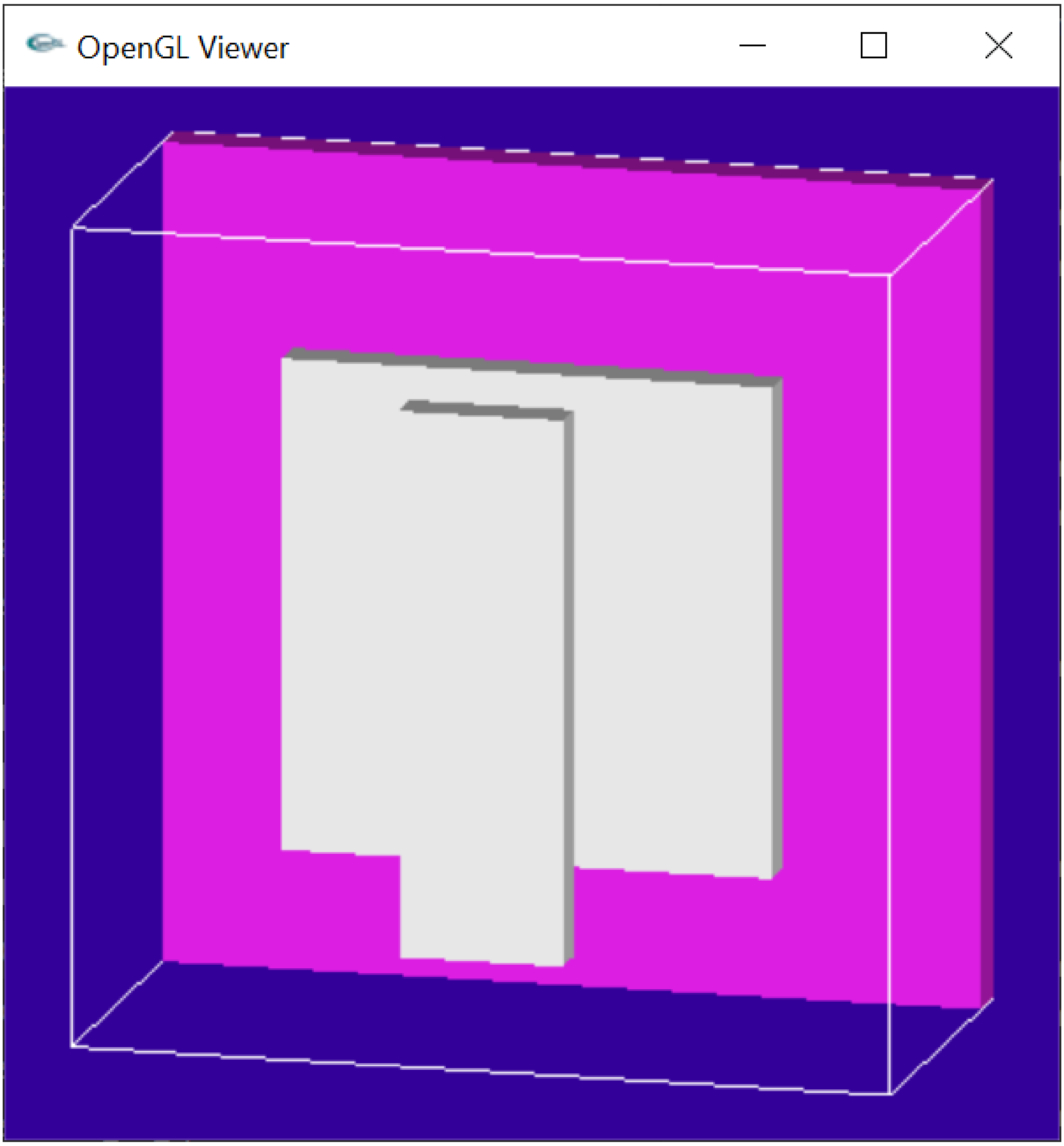

OpenGL Viewer

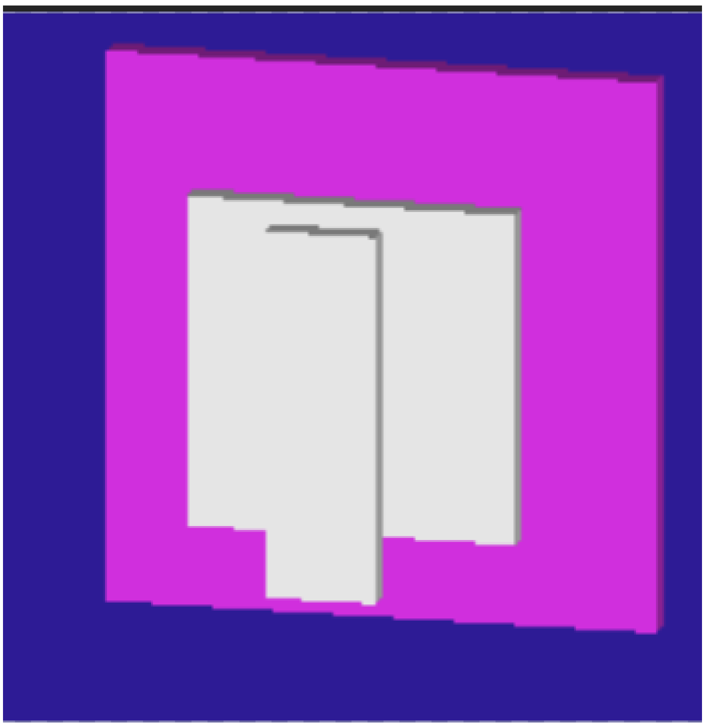

OpenGL Viewer



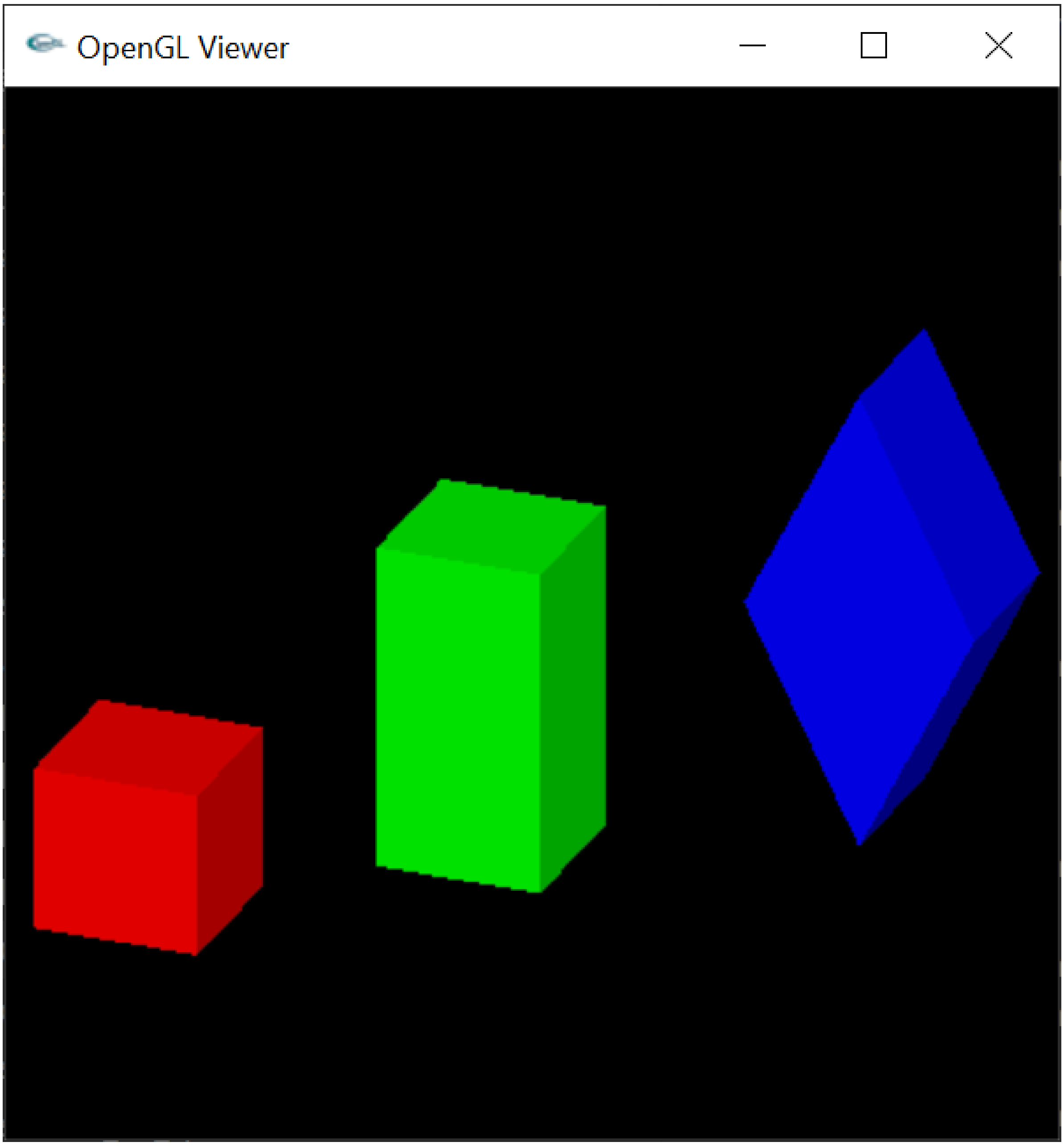
```
raytracer -input scene5_11_rotated_triangles.txt -size 200 200 -gui
raytracer -input scene5_11_rotated_triangles.txt -size 200 200 -output
output5_11.tga -gui -grid 15 15 9 -visualize_grid
```

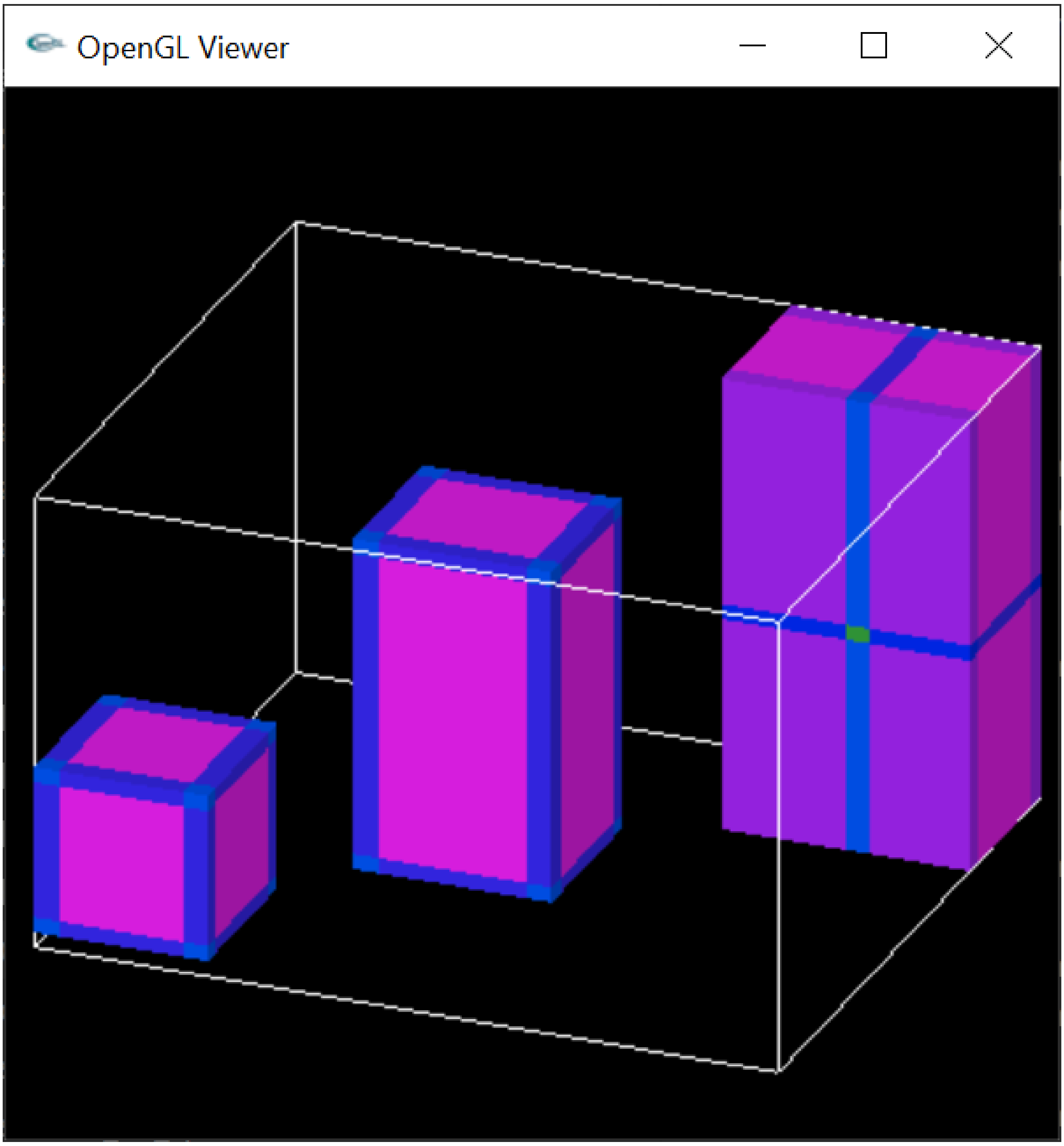



OpenGL Viewer



```
raytracer -input scene5_12_nested_transformations.txt -size 200 200 -gui
raytracer -input scene5_12_nested_transformations.txt -size 200 200 -output
output5_12.tga -gui -grid 30 30 30 -visualize_grid
```




OpenGL Viewer