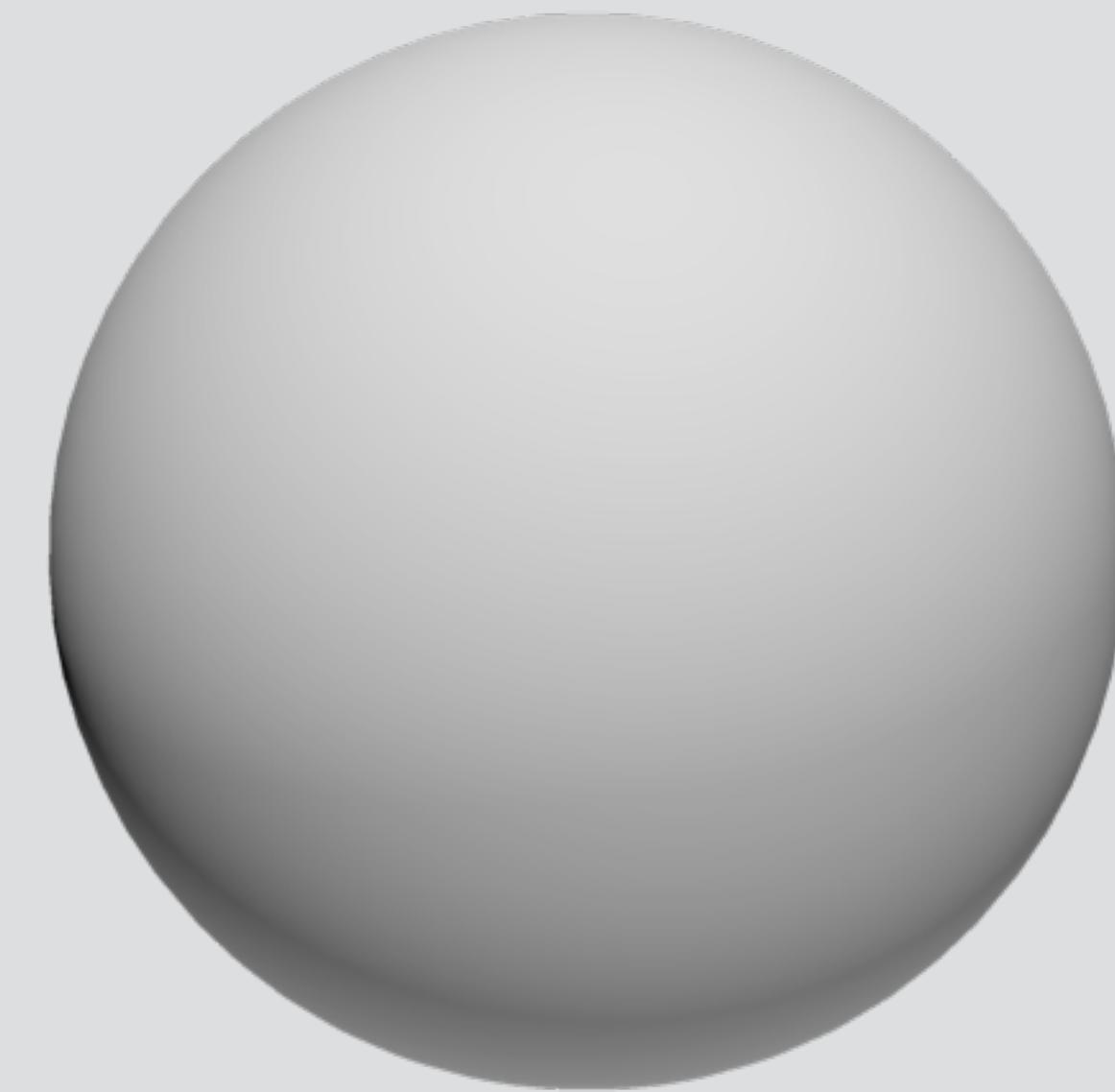
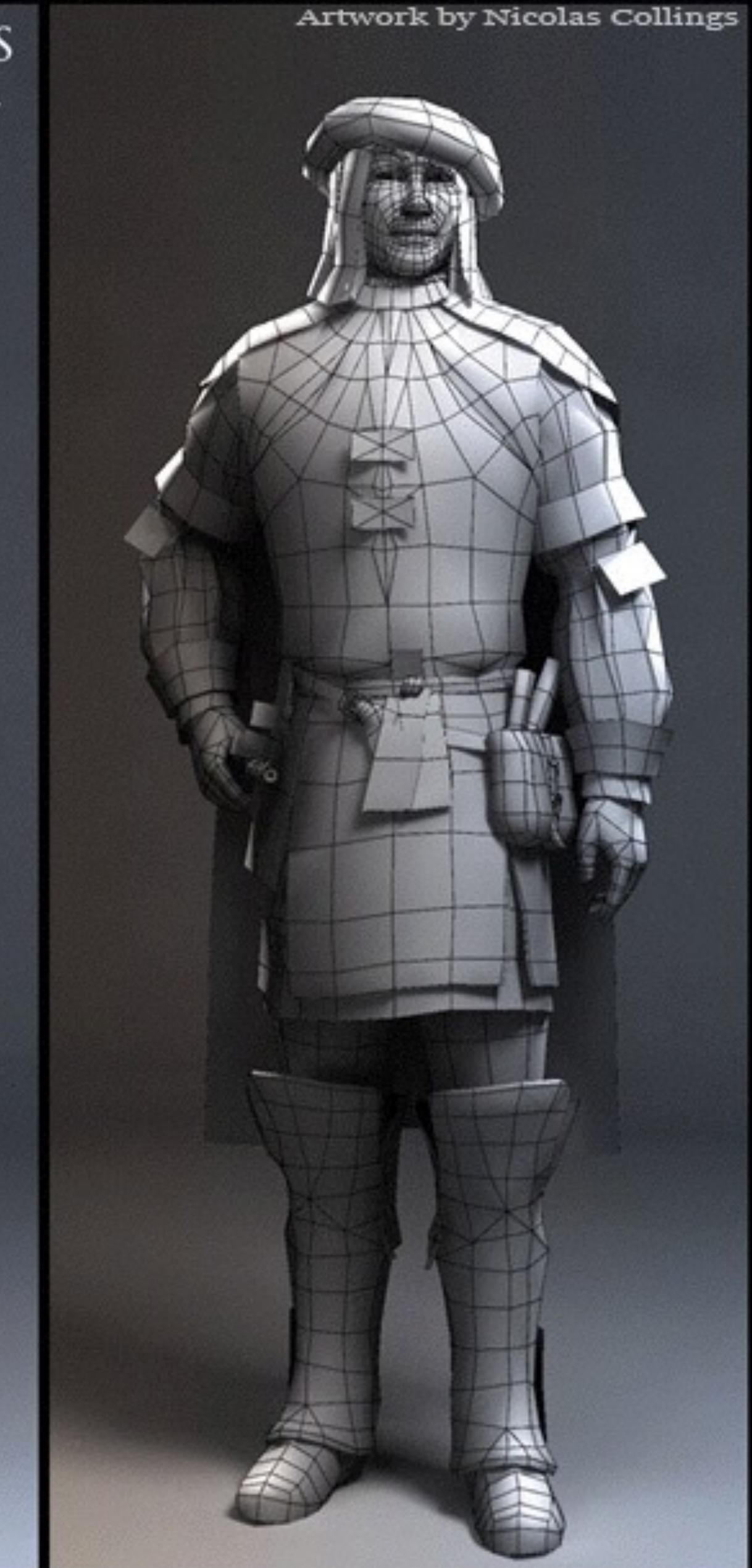
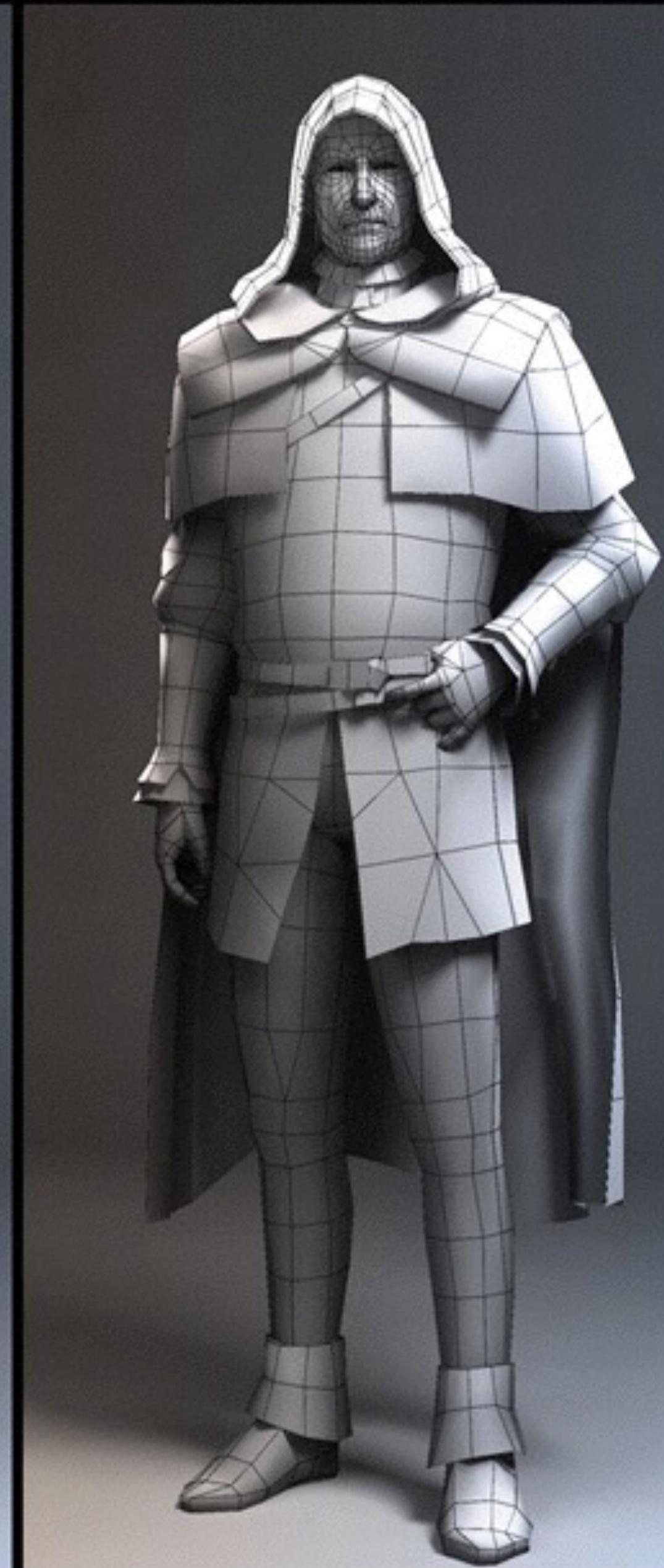


# Texturing Mapping

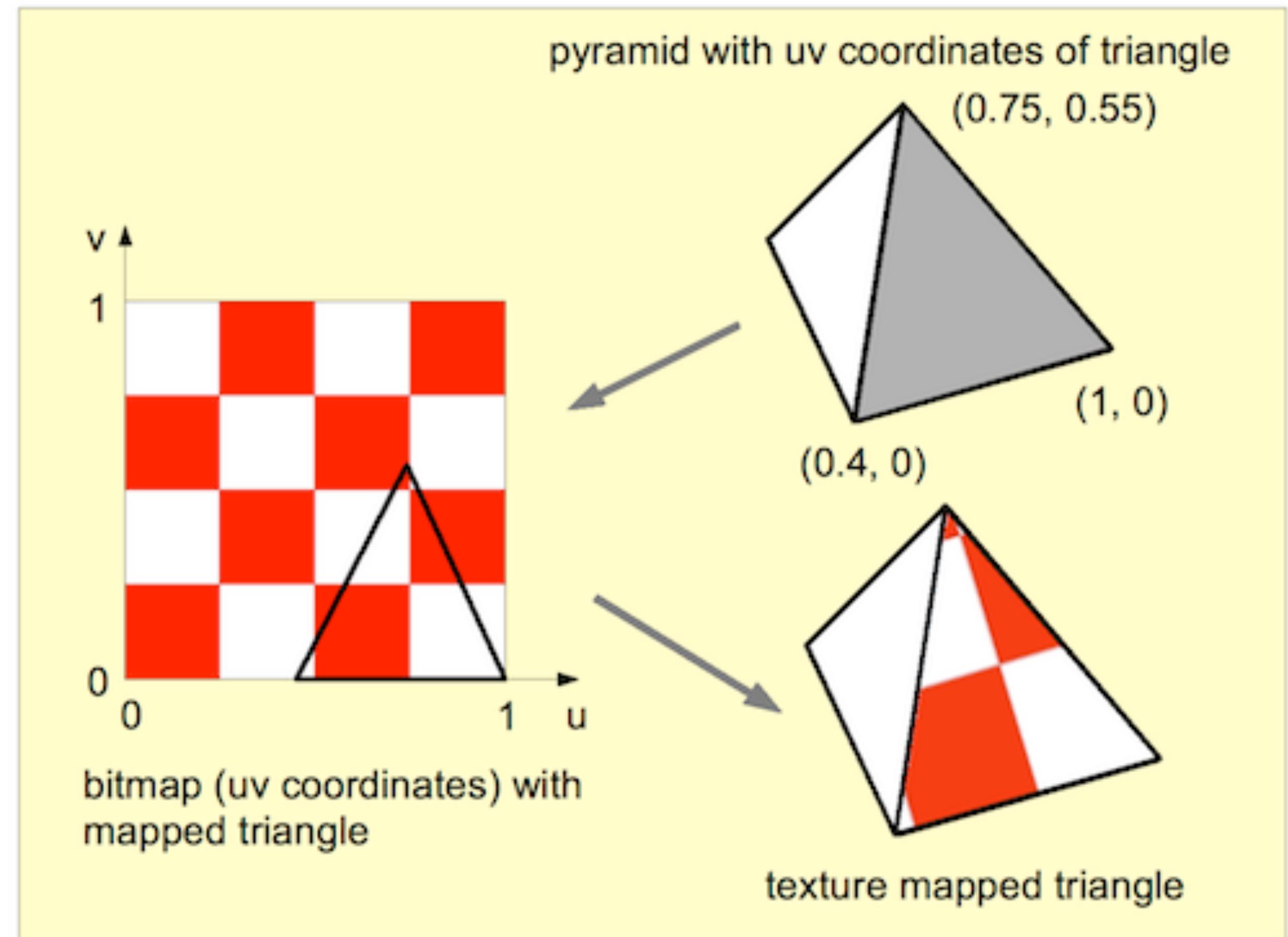


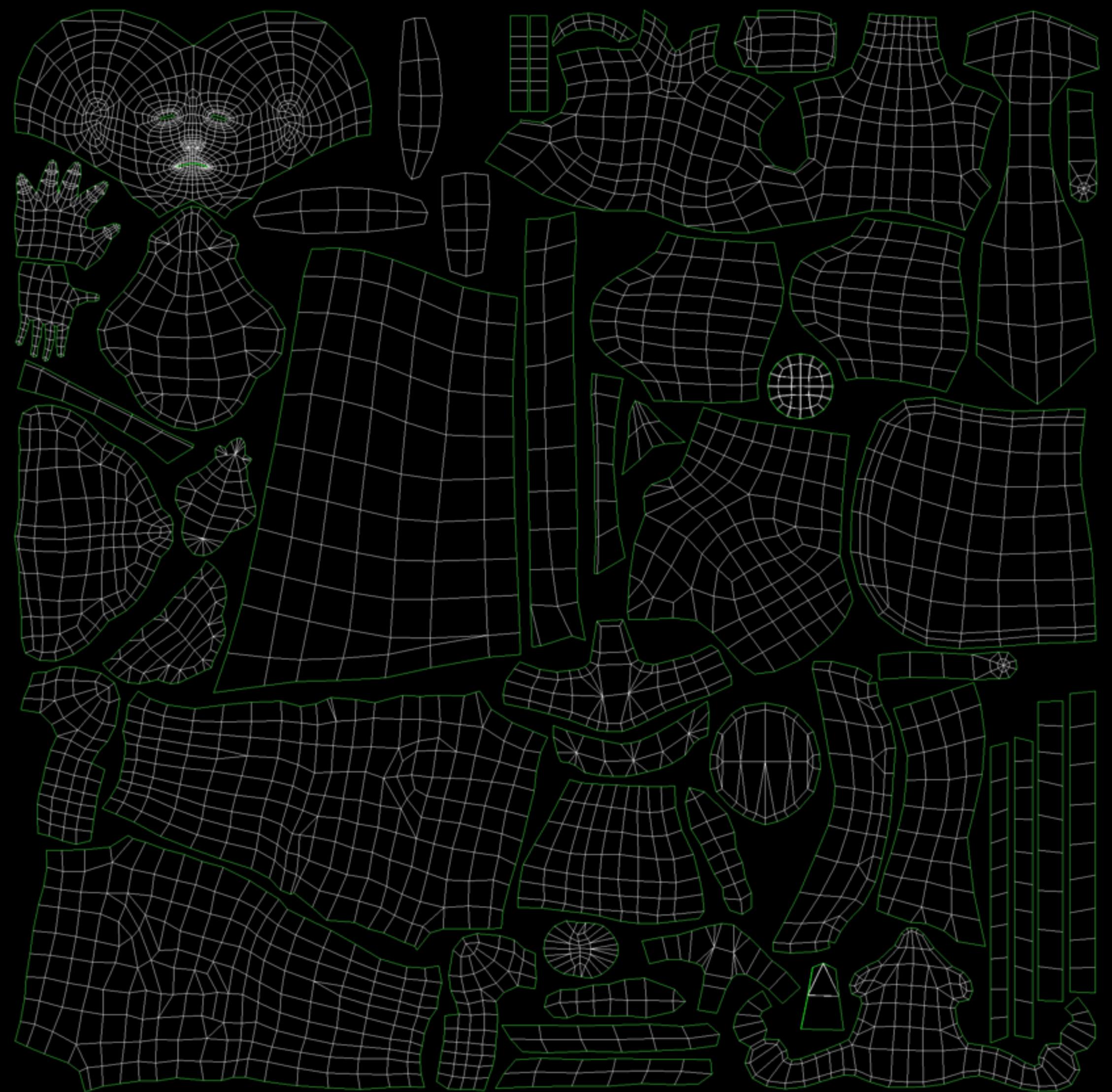
CS GY-6533 / UY-4533

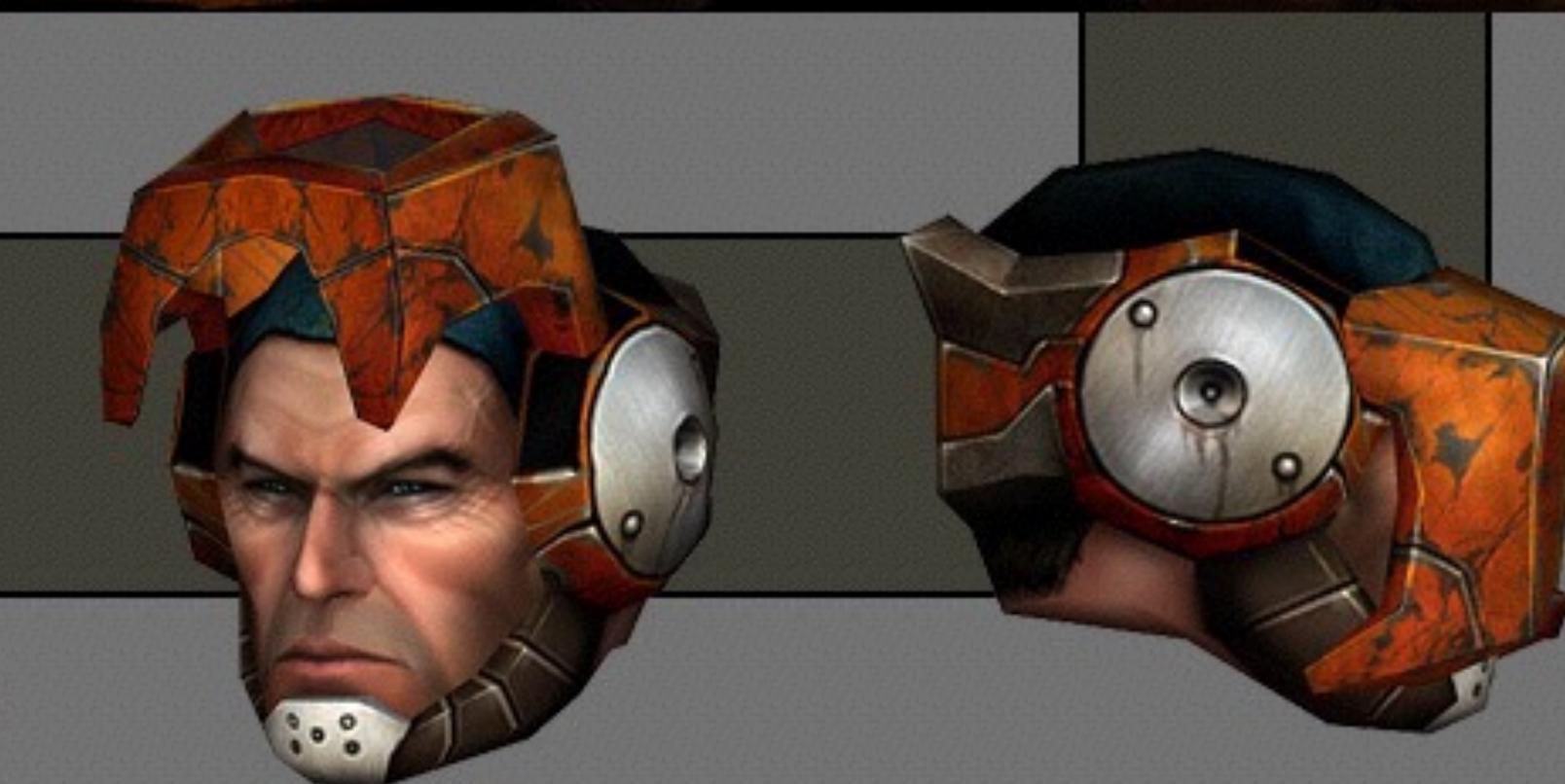
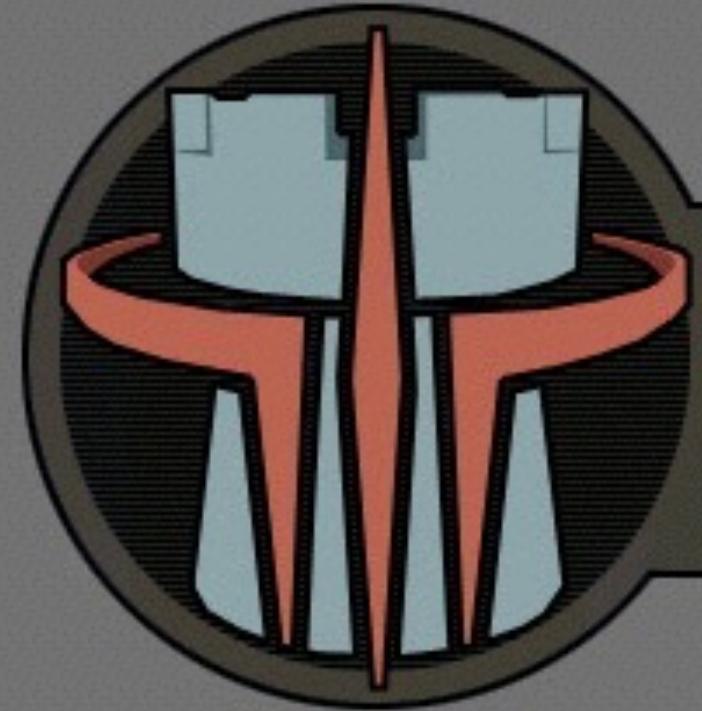




# UV coordinates







Engineer • Q3F • [www.q3f.com](http://www.q3f.com) • BoBo!

# Basic texturing

# Vertex program

```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelViewMatrix;
uniform mat4 projectionMatrix;

varying vec2 varyingTexCoord;

void main() {
    varyingTexCoord = texCoord;
    gl_Position = projectionMatrix * modelViewMatrix * position;
}
```

# Fragment program

```
varying vec2 varyingTexCoord;
uniform sampler2D diffuseTexture;

void main() {
    gl_FragColor = texture2D(diffuseTexture, varyingTexCoord);
}
```

# Need to add texture coordinate to our vertex structure.

```
struct VertexPNT {
    Cvec3f p, n;
    Cvec2f t;

    VertexPNT() {}
    VertexPNT(float x, float y, float z, float nx, float ny, float nz) : p(x,y,z), n(nx, ny, nz) {}

    VertexPNT& operator = (const GenericVertex& v) {
        p = v.pos;
        n = v.normal;
        t = v.tex;
        return *this;
    }
};

// ...

glEnableVertexAttribArray(texCoordAttribute);
glVertexAttribPointer(texCoordAttribute, 2, GL_FLOAT, GL_FALSE, sizeof(VertexPNT), (void*)offsetof(VertexPNT, t));
glEnableVertexAttribArray(texCoordAttribute);
```

Texture uniforms in GLSL are integer indexes of the GPU texture unit.  
On the C++ side, they must be bound to the appropriate texture unit  
using **glActiveTexture**.

```
GLuint diffuseTexture;
GLuint diffuseTextureUniformLocation;

// ...

diffuseTexture = loadGLTexture("someimage.png");
diffuseTextureUniformLocation = glGetUniformLocation(program, "diffuseTexture");

// ...

glUniform1i(diffuseTextureUniformLocation, 0);
glActiveTexture(GL_TEXTURE0);
 glBindTexture(GL_TEXTURE_2D, diffuseTexture);
```

```
#define TINYOBJLOADER_IMPLEMENTATION
#include "tiny_obj_loader.h"

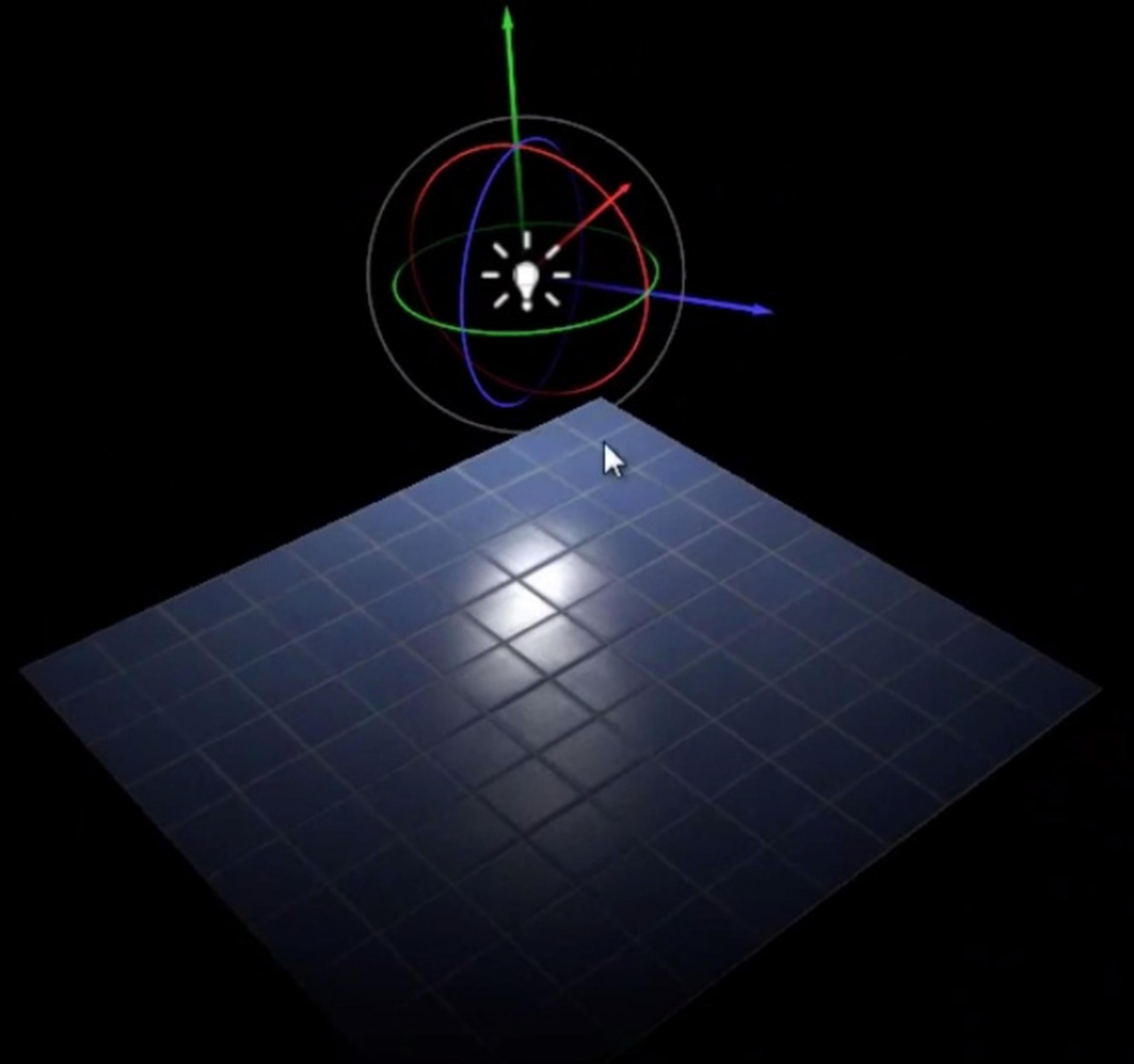
////

void loadObjFile(const std::string &fileName, std::vector<VertexPNT> &outVertices, std::vector<unsigned short> &outIndices) {
    tinyobj::attrib_t attrib;
    std::vector<tinyobj::shape_t> shapes;
    std::vector<tinyobj::material_t> materials;
    std::string err;
    bool ret = tinyobj::LoadObj(&attrib, &shapes, &materials, &err, fileName.c_str(), NULL, true);
    if(ret) {
        for(int i=0; i < shapes.size(); i++) {
            for(int j=0; j < shapes[i].mesh.indices.size(); j++) {
                unsigned int vertexOffset = shapes[i].mesh.indices[j].vertex_index * 3;
                unsigned int normalOffset = shapes[i].mesh.indices[j].normal_index * 3;
                unsigned int texOffset = shapes[i].mesh.indices[j].texcoord_index * 2;
                VertexPNT v;
                v.p[0] = attrib.vertices[vertexOffset];
                v.p[1] = attrib.vertices[vertexOffset+1];
                v.p[2] = attrib.vertices[vertexOffset+2];
                v.n[0] = attrib.normals[normalOffset];
                v.n[1] = attrib.normals[normalOffset+1];
                v.n[2] = attrib.normals[normalOffset+2];
                v.t[0] = attrib.texcoords[texOffset];
                v.t[1] = 1.0-attrib.texcoords[texOffset+1];
                outVertices.push_back(v);
                outIndices.push_back(outVertices.size()-1);
            }
        }
    } else {
        std::cout << err << std::endl;
        assert(false);
    }
}
```

# Combining with lighting

```
vec3 intensity = (texture2D(diffuseTexture, varyingTexCoord).xyz * diffuseColor) + specularColor;  
gl_FragColor = vec4(intensity.xyz, 1.0);
```

# Specular maps





## Specular Map



```
vec3 intensity = (texture2D(diffuseTexture, varyingTexCoord).xyz * diffuseColor)
+ (specularColor * texture2D(specularTexture, varyingTexCoord).x);
```

# Multiple textures

# GLSL

```
uniform sampler2D diffuseTexture;  
uniform sampler2D specularTexture;
```

```
// . . .
```

# C++

```
GLuint diffuseUniformLocation;  
GLuint specularUniformLocation;
```

```
// . . .
```

```
diffuseUniformLocation = glGetUniformLocation(program, "diffuseTexture");  
specularUniformLocation = glGetUniformLocation(program, "specularTexture");
```

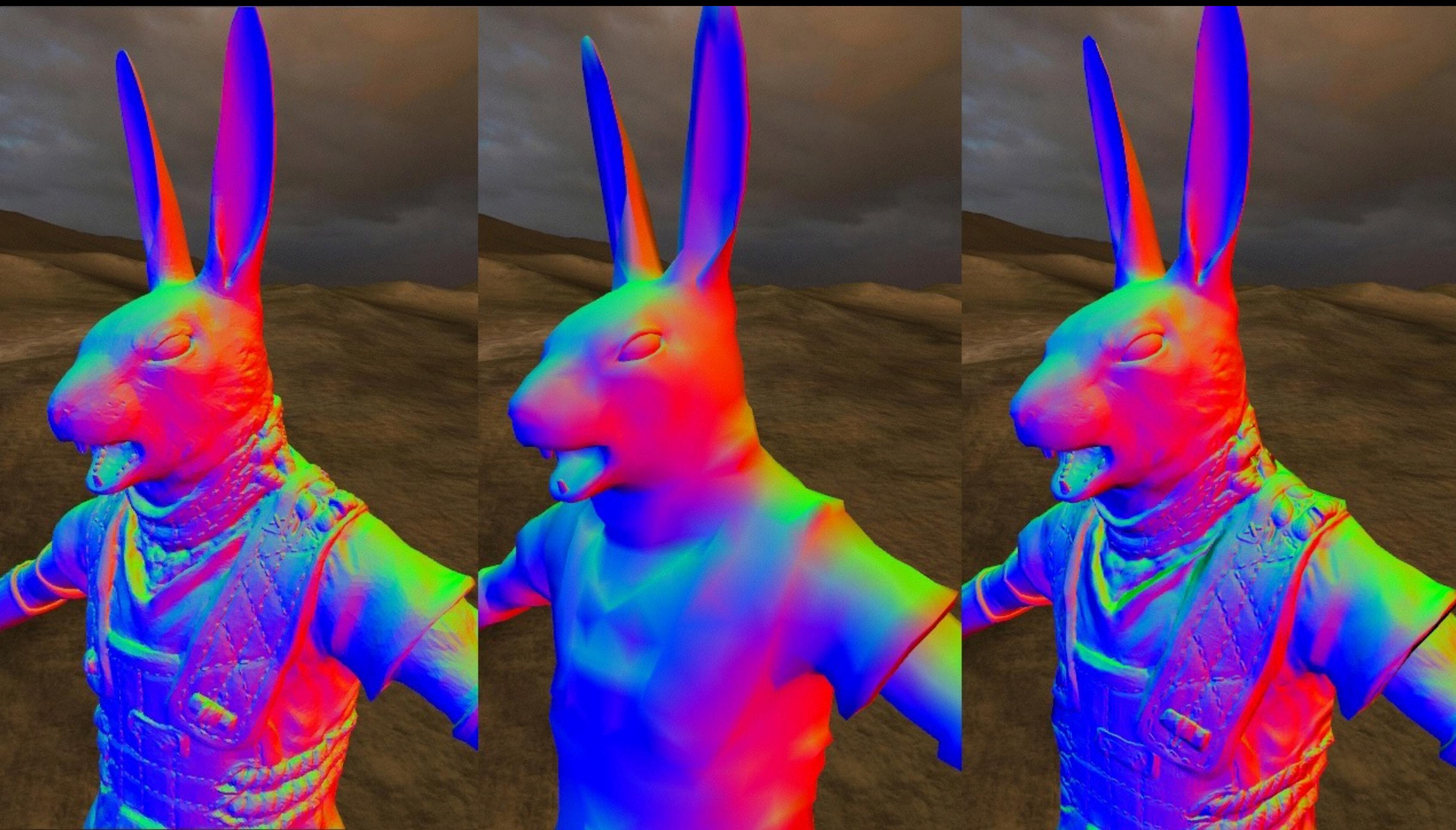
```
// . . .
```

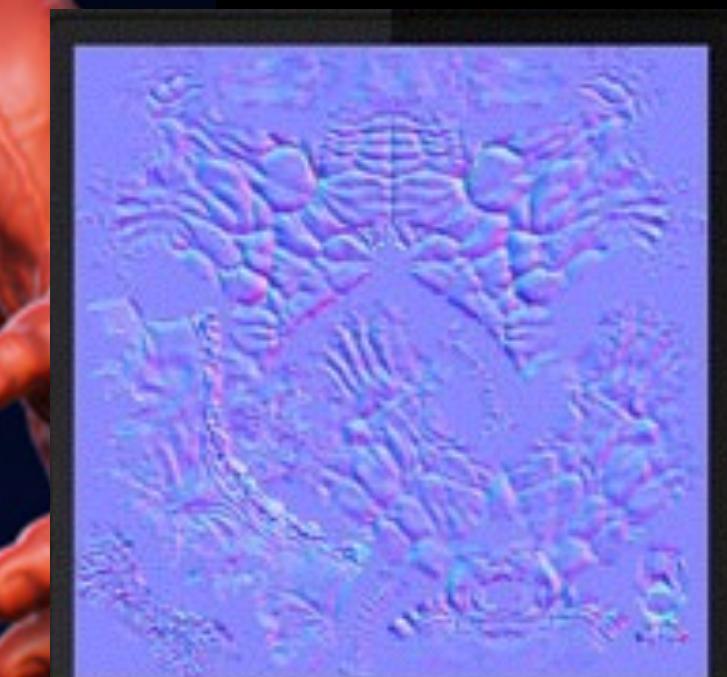
```
glUniform1i(diffuseUniformLocation, 0);  
glActiveTexture(GL_TEXTURE0);  
glBindTexture(GL_TEXTURE_2D, diffuseTexture);
```

```
glUniform1i(specularUniformLocation, 1);  
glActiveTexture(GL_TEXTURE1);  
glBindTexture(GL_TEXTURE_2D, specularTexture);
```

**Be aware of GPU limits for texture units (usually 16 or 32).**

# Normal maps

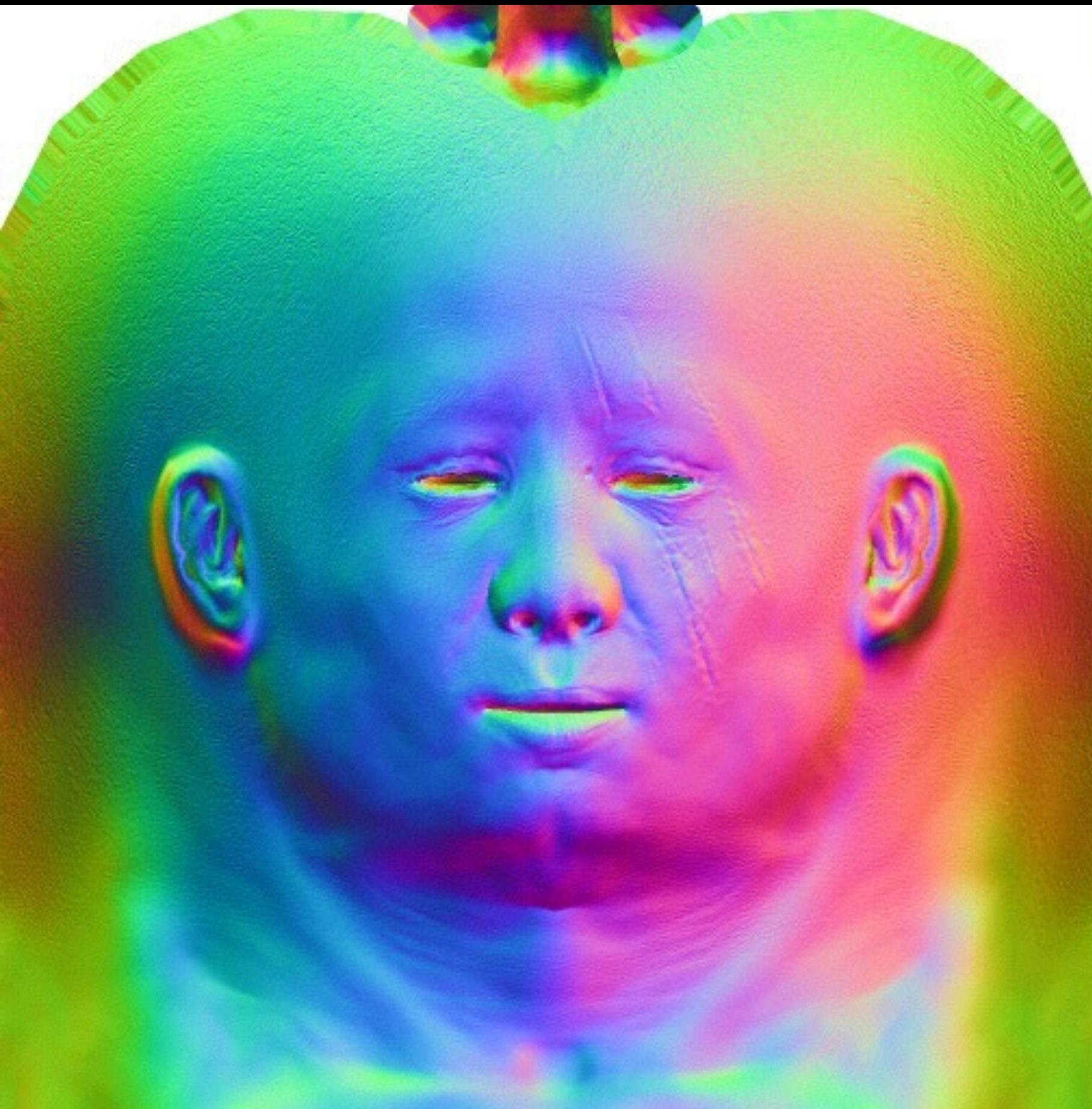




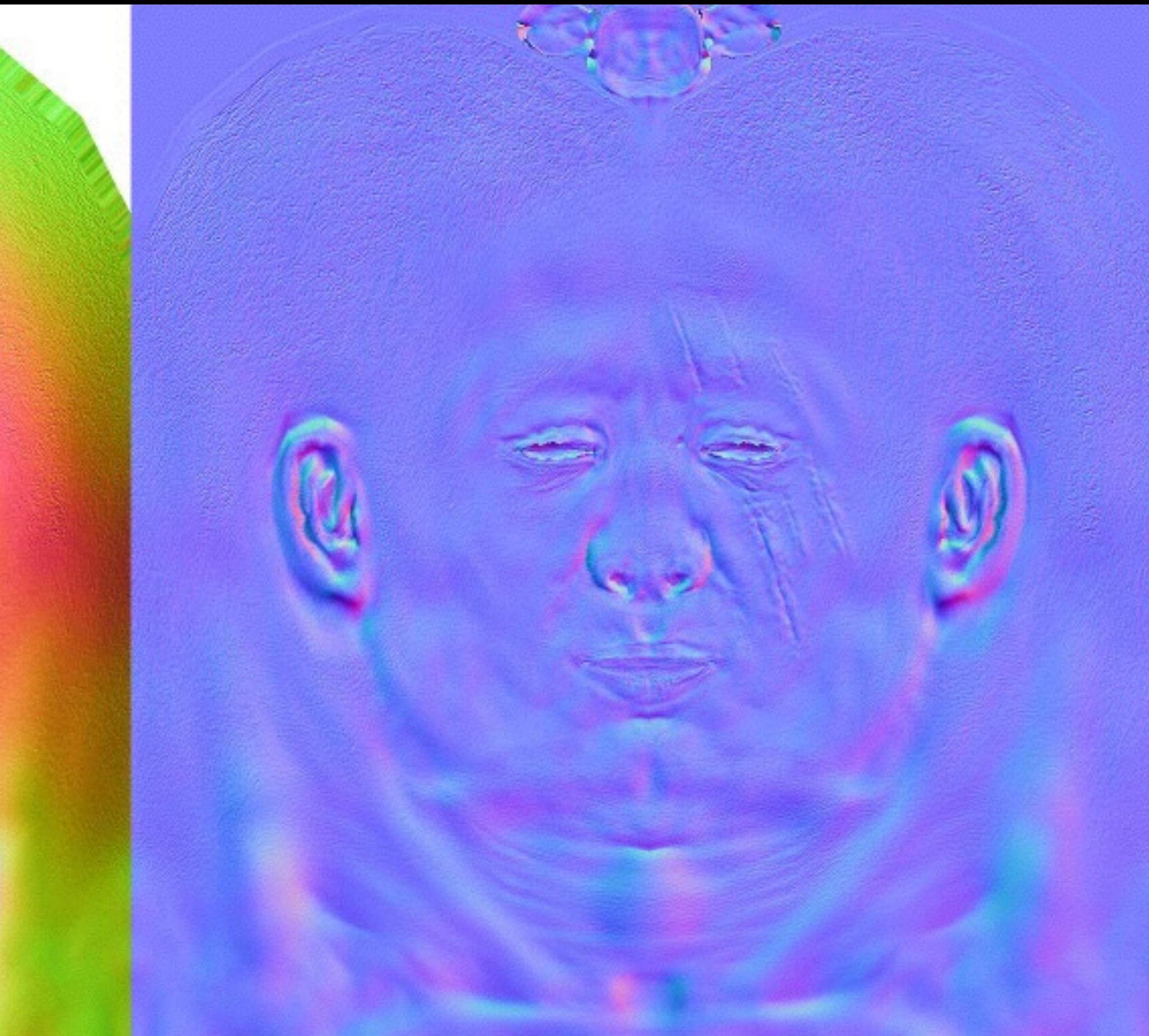
Current Texture  
ZTexture01\_NRM3\_NRM  
Width=2048  
Height=2048  
Depth=32



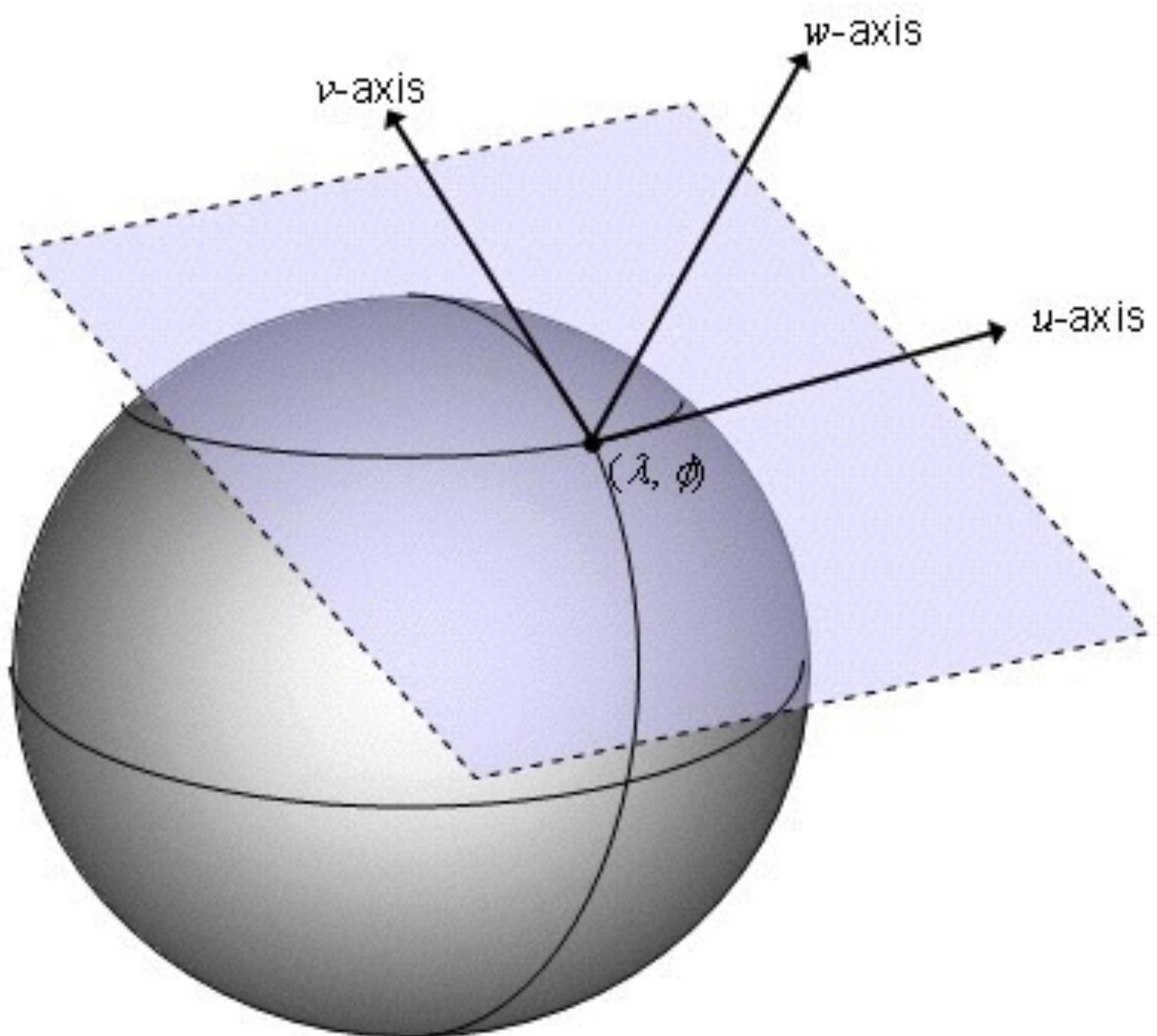
**Object space normal map**

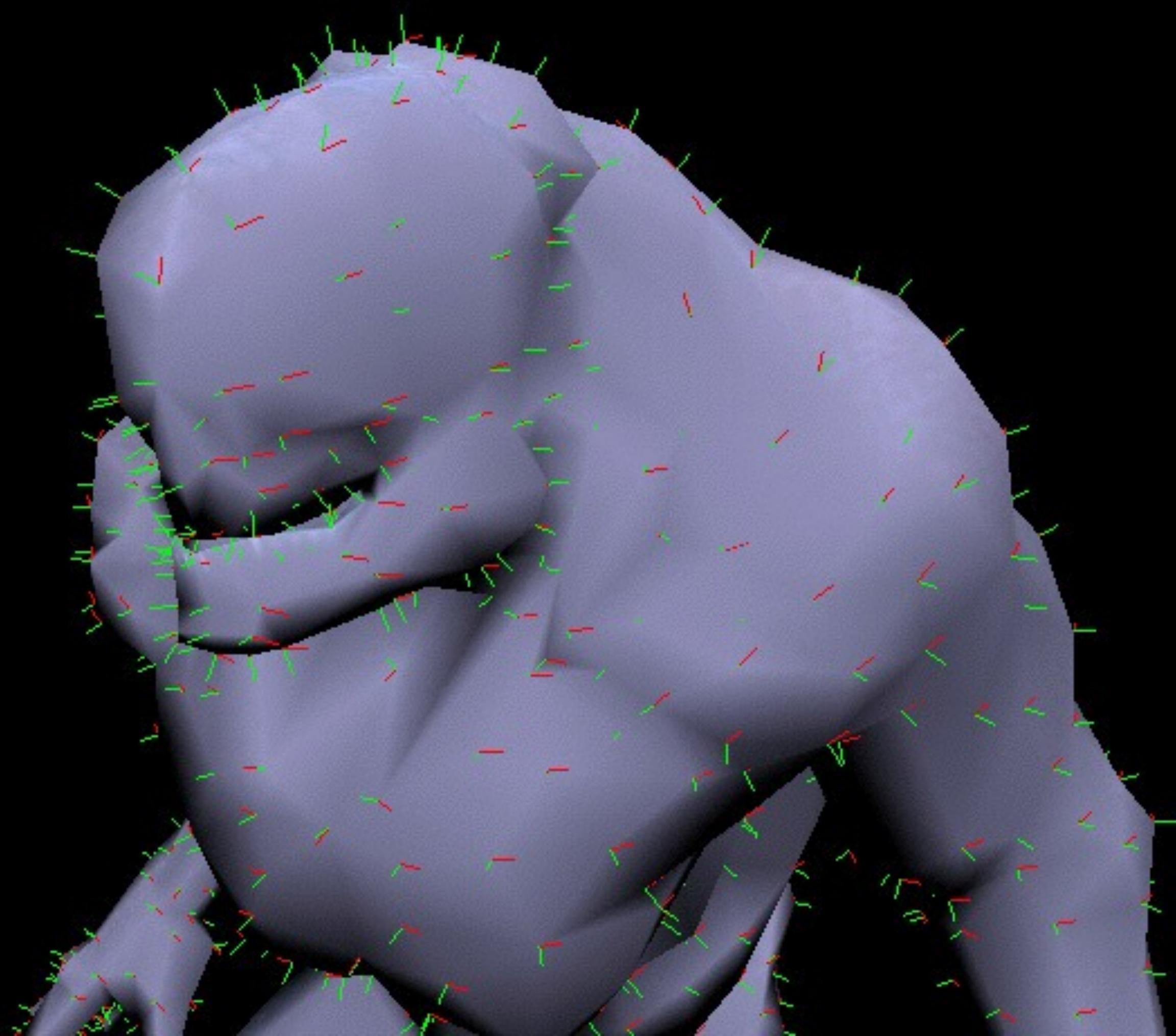
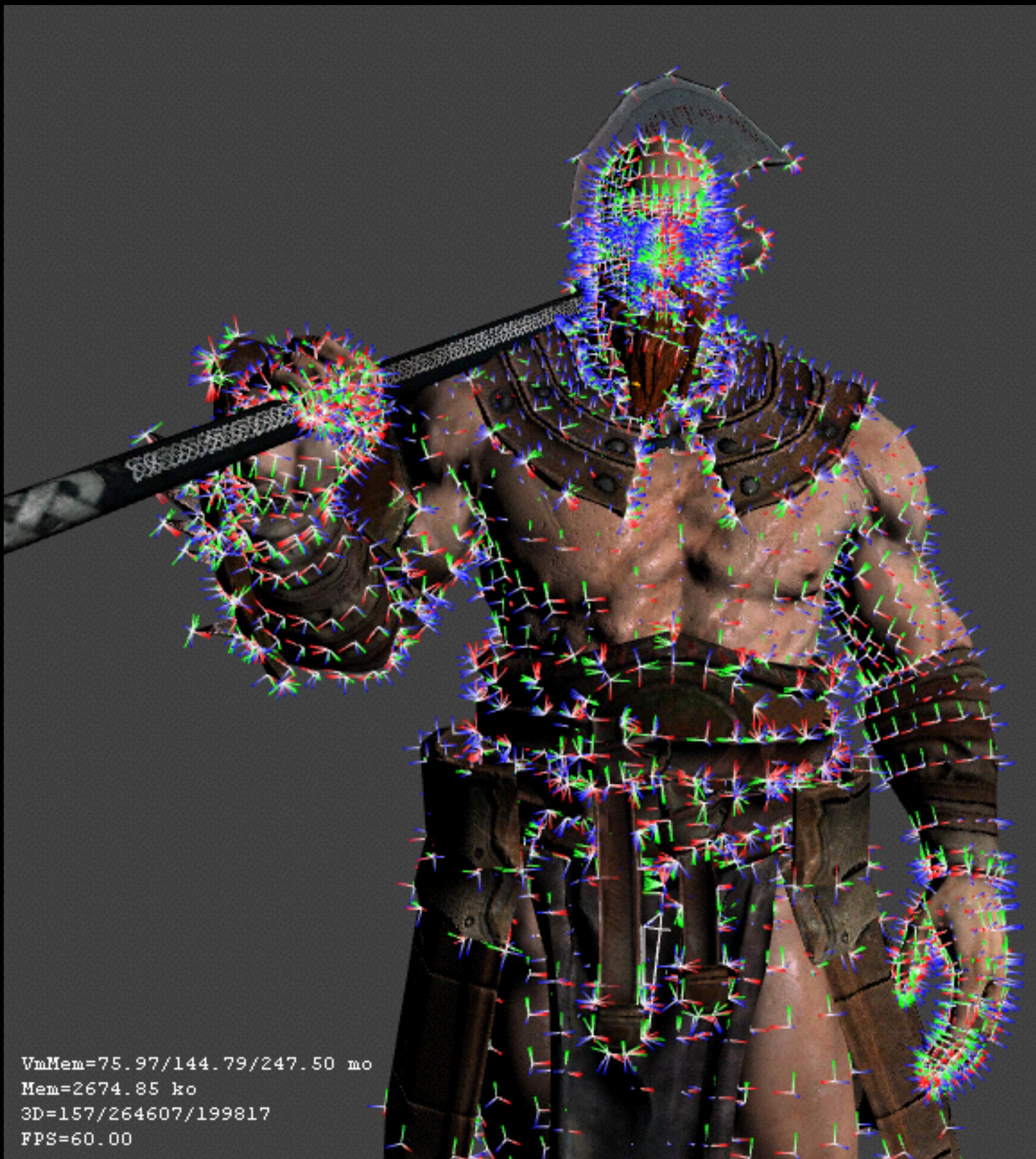


**Tangent space normal map**



# Tangent space





# Vertex program

```
attribute vec4 normal;
attribute vec4 binormal;
attribute vec4 tangent;

varying mat3 varyingTBNMatrix;

// . . .

varyingTBNMatrix = mat3(normalize((normalMatrix * tangent).xyz), normalize((normalMatrix * binormal).xyz), normalize((normalMatrix * normal).xyz));
```

# Fragment program

```
uniform sampler2D normalTexture;
varying mat3 varyingTBNMatrix;

// . . .

vec3 textureNormal = normalize((texture2D(normalTexture, varyingTexCoord).xyz * 2.0) - 1.0);
textureNormal = normalize(varyingTBNMatrix * textureNormal);
```

```
struct VertexPNTBTG {
    Cvec3f p, n, b, tg;
    Cvec2f t;

    VertexPNTBTG() {}
    VertexPNTBTG(float x, float y, float z, float nx, float ny, float nz) : p(x,y,z), n(nx, ny, nz) {}

    VertexPNTBTG& operator = (const GenericVertex& v) {
        p = v.pos;
        n = v.normal;
        t = v.tex;
        b = v.binormal;
        tg = v.tangent;
        return *this;
    }
};

// . . .

glEnableVertexAttribArray(binormalAttribute);
glVertexAttribPointer(binormalAttribute, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPNTBTG), (void*)offsetof(VertexPNTBTG, b));
glEnableVertexAttribArray(binormalAttribute);

glEnableVertexAttribArray(tangentAttribute);
glVertexAttribPointer(tangentAttribute, 3, GL_FLOAT, GL_FALSE, sizeof(VertexPNTBTG), (void*)offsetof(VertexPNTBTG, tg));
glEnableVertexAttribArray(tangentAttribute);
```

# Calculating tangents and binormals.

```
void calculateFaceTangent(const Cvec3f &v1, const Cvec3f &v2, const Cvec3f &v3, const Cvec2f &texCoord1, const Cvec2f &texCoord2,
const Cvec2f &texCoord3, Cvec3f &tangent, Cvec3f &binormal) {

    Cvec3f side0 = v1 - v2;
    Cvec3f side1 = v3 - v1;
    Cvec3f normal = cross(side1, side0);
    normalize(normal);
    float deltaV0 = texCoord1[1] - texCoord2[1];
    float deltaV1 = texCoord3[1] - texCoord1[1];
    tangent = side0 * deltaV1 - side1 * deltaV0;
    normalize(tangent);

    float deltaU0 = texCoord1[0] - texCoord2[0];
    float deltaU1 = texCoord3[0] - texCoord1[0];

    binormal = side0 * deltaU1 - side1 * deltaU0;
    normalize(binormal);
    Cvec3f tangentCross = cross(tangent, binormal);

    if (dot(tangentCross, normal) < 0.0f) {
        tangent = tangent * -1;
    }
}
```

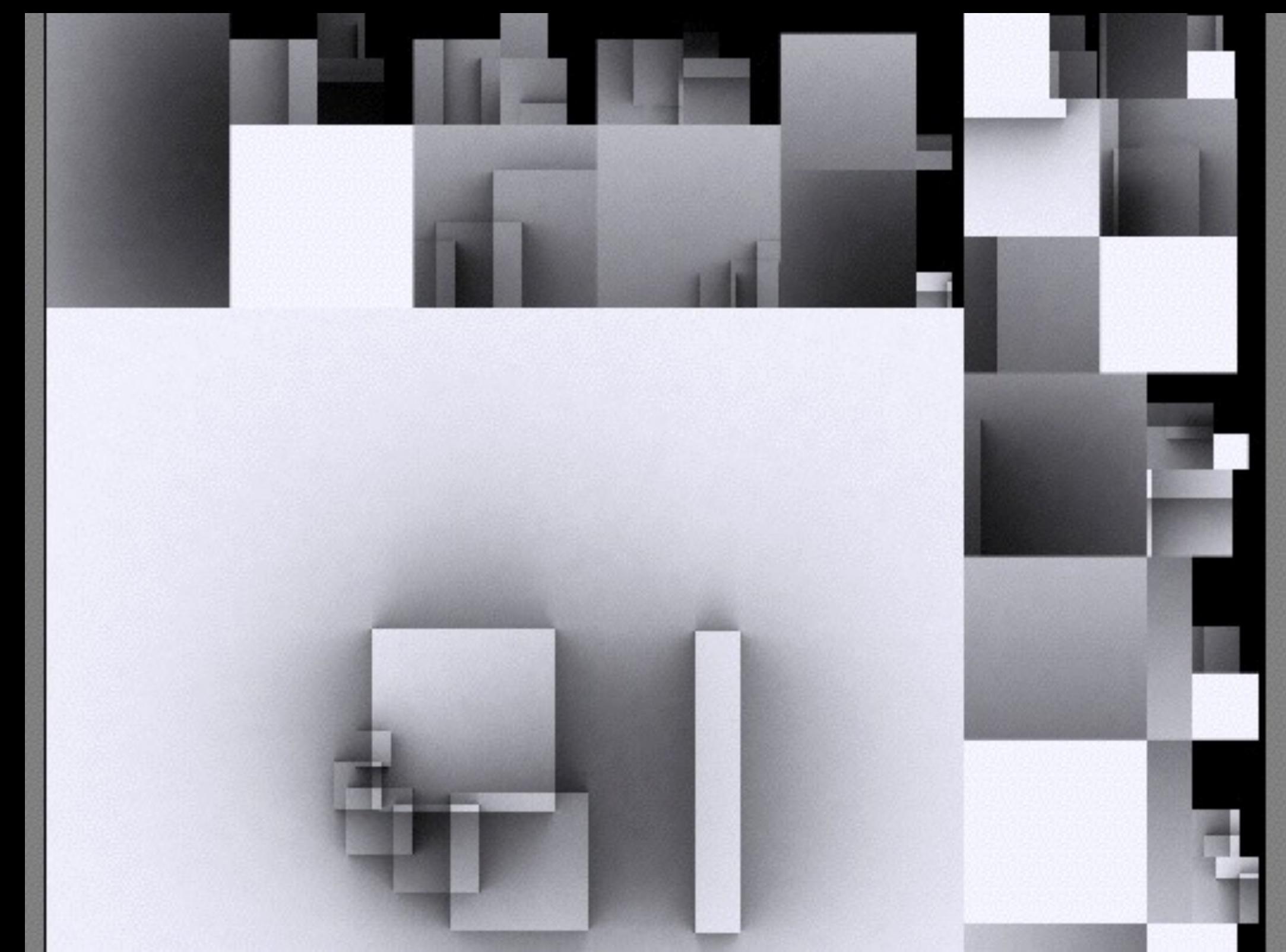
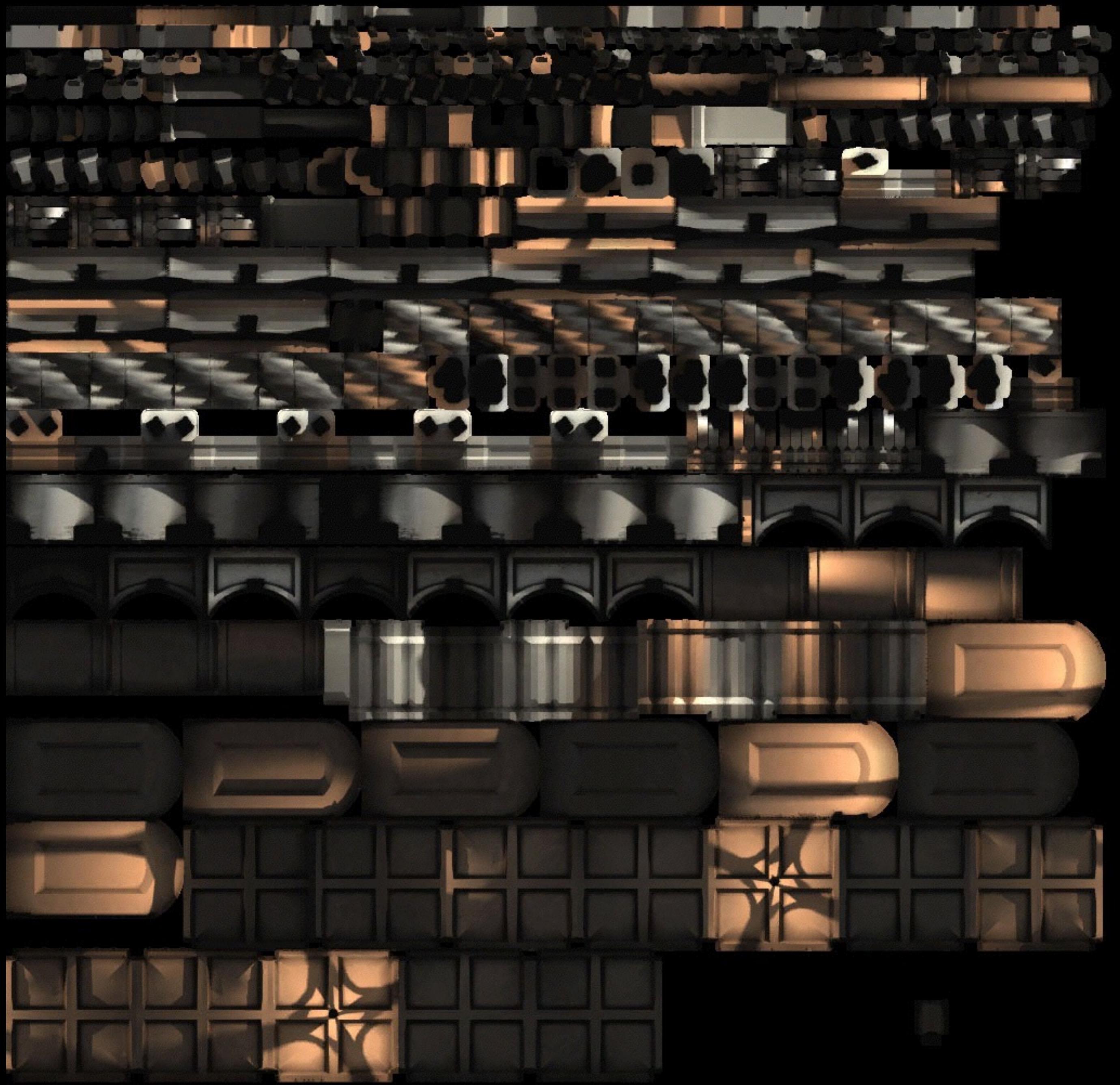
```
for(int i=0; i < outVertices.size(); i += 3) {
    Cvec3f tangent;
    Cvec3f binormal;

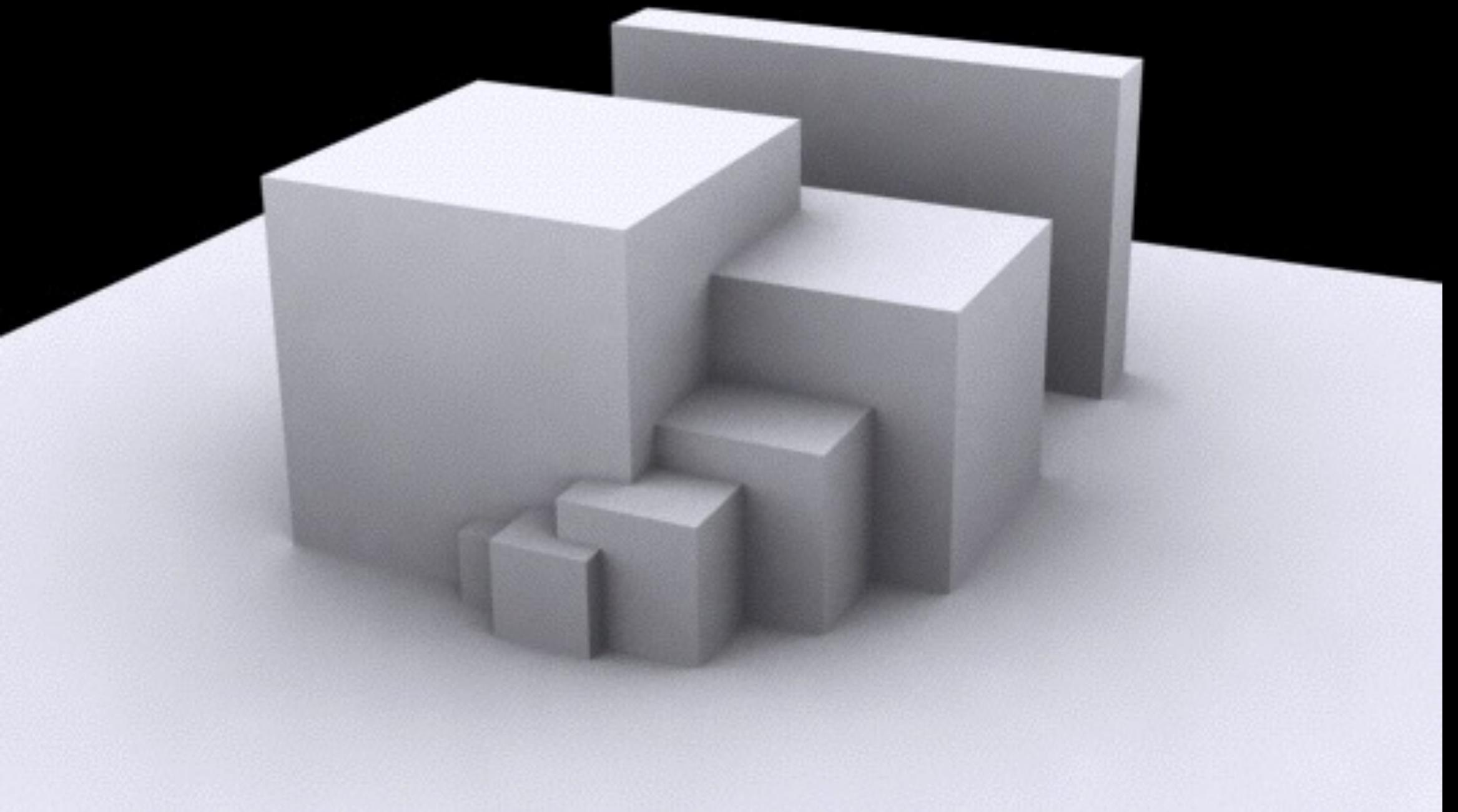
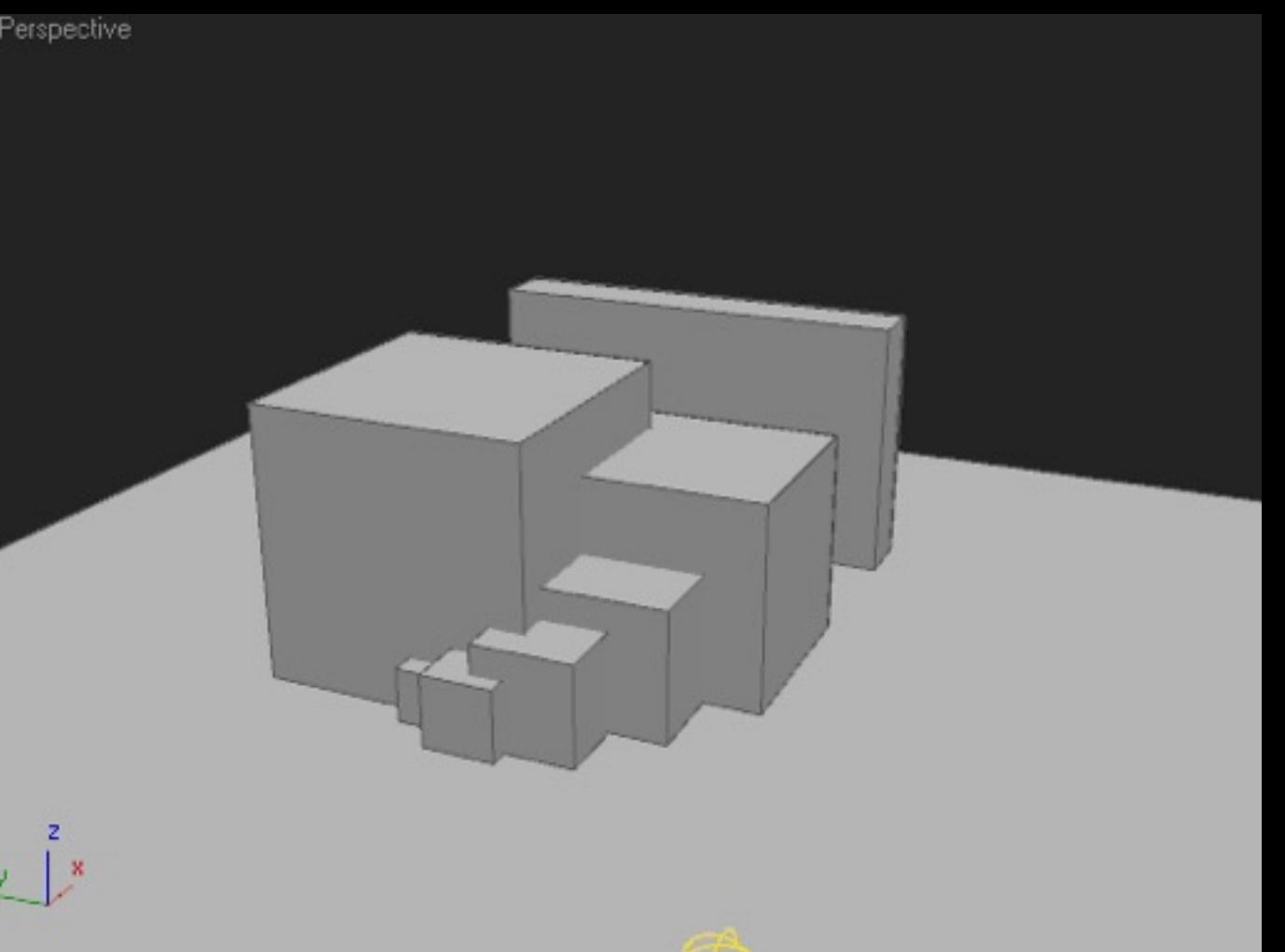
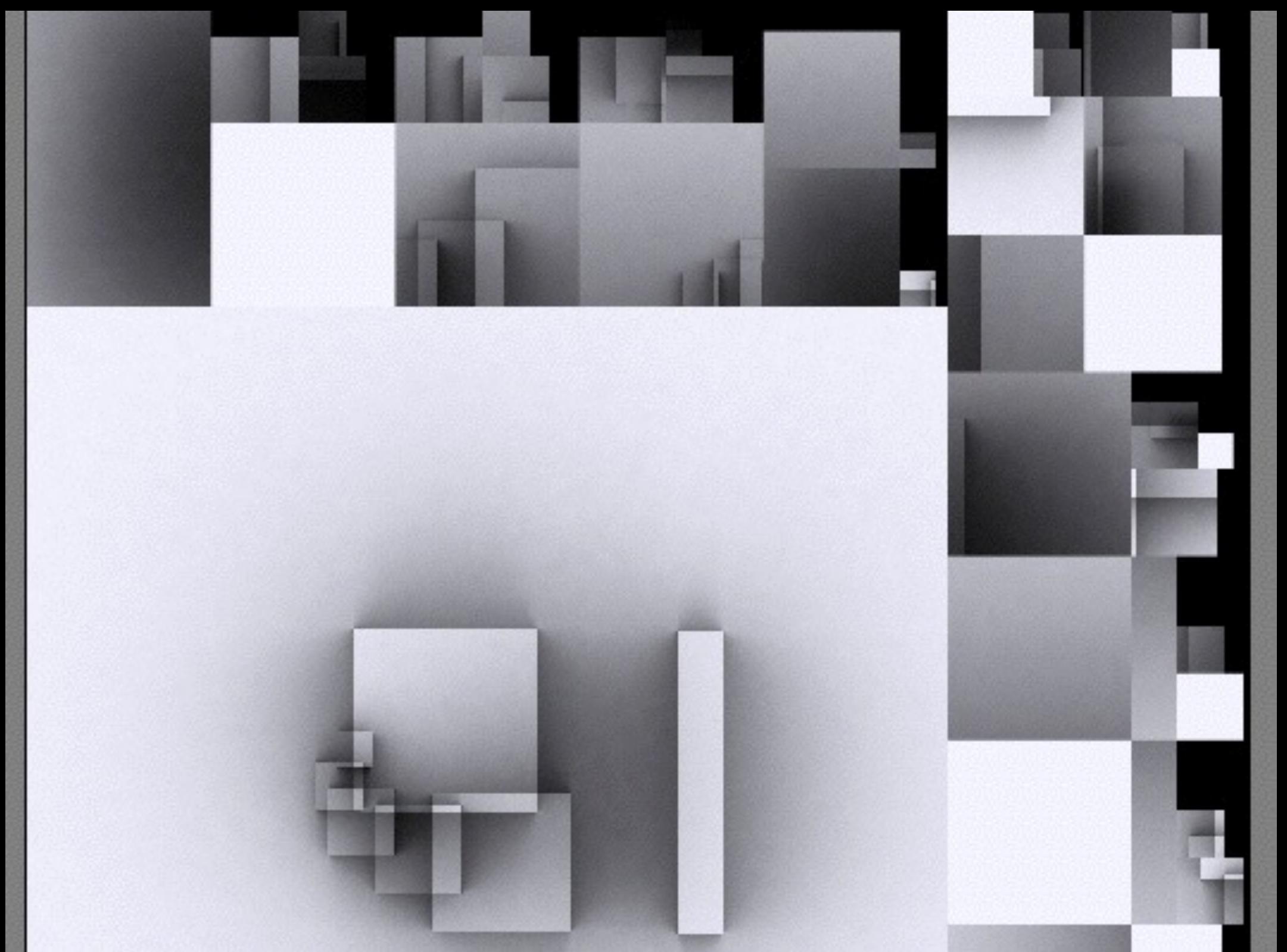
    calculateFaceTangent(outVertices[i].p, outVertices[i+1].p, outVertices[i+2].p,
outVertices[i].t, outVertices[i+1].t, outVertices[i+2].t, tangent, binormal);

    outVertices[i].tg = tangent;
    outVertices[i+1].tg = tangent;
    outVertices[i+2].tg = tangent;

    outVertices[i].b = binormal;
    outVertices[i+1].b = binormal;
    outVertices[i+2].b = binormal;
}
```

# Multiple UVs and lightmaps





# Assignment #3

- Create a simple scene with multiple positional lights.
- Use a normal map and a specular map on at least one object.
- Use either the object loader or the geometry generator to create objects in your scene.