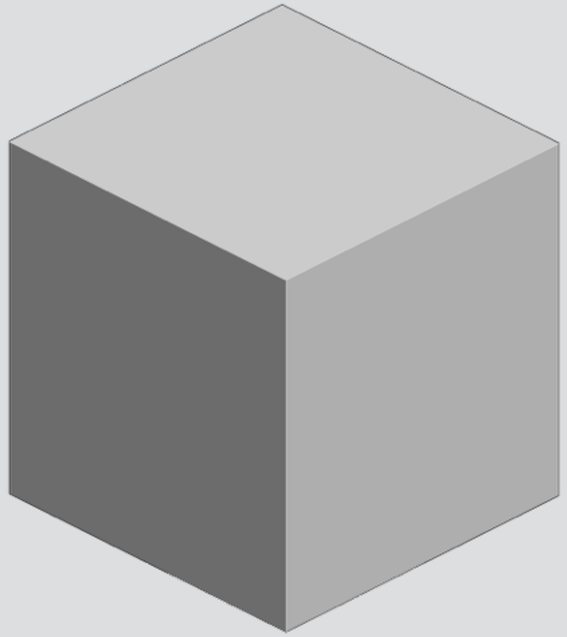
3D Graphics

Part 2




x 10
x 0
x 16

PICKED UP 4 SHOTGUN SHELLS.

14 AMMO

59% HEALTH

2 3 4
5 6 7
ARMS

0% ARMOR

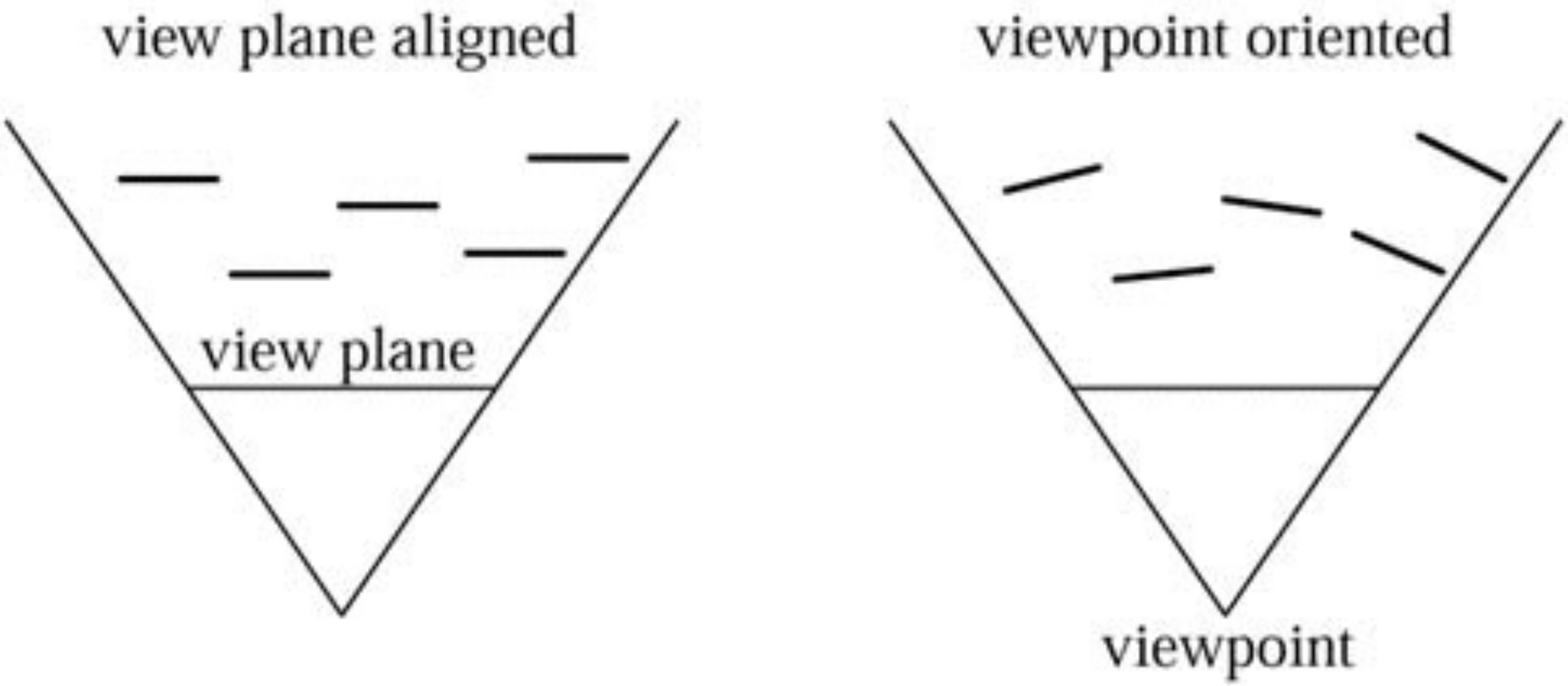
BULL	42	200
SHEL	14	50
ROKT	0	50
CELL	0	300

Billboards.

Billboards

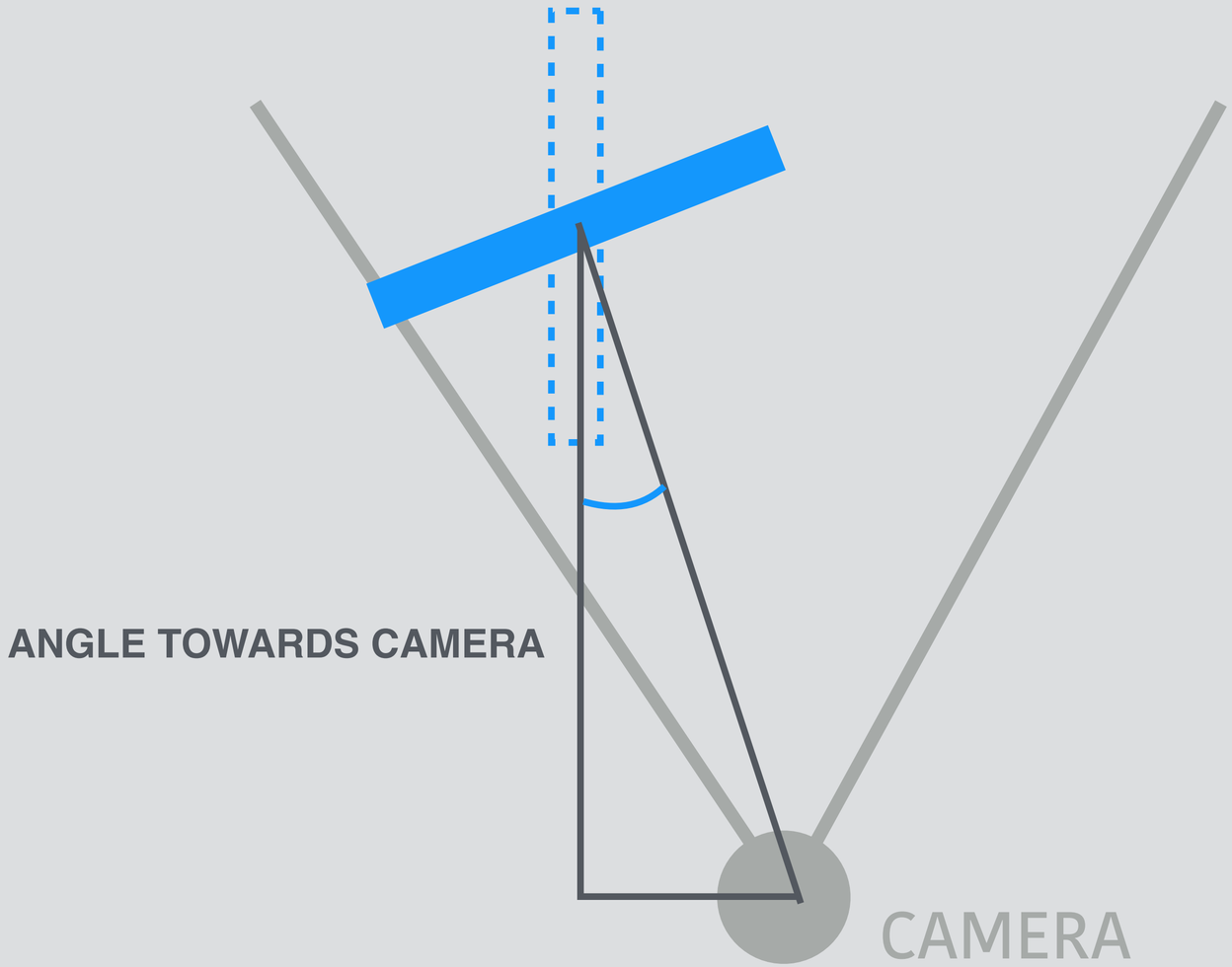
=

Sprites that **always face the camera**.


view plane aligned
view plane
viewpoint oriented
viewpoint

Method #1 - Rotate all billboards towards camera.

Viewpoint oriented



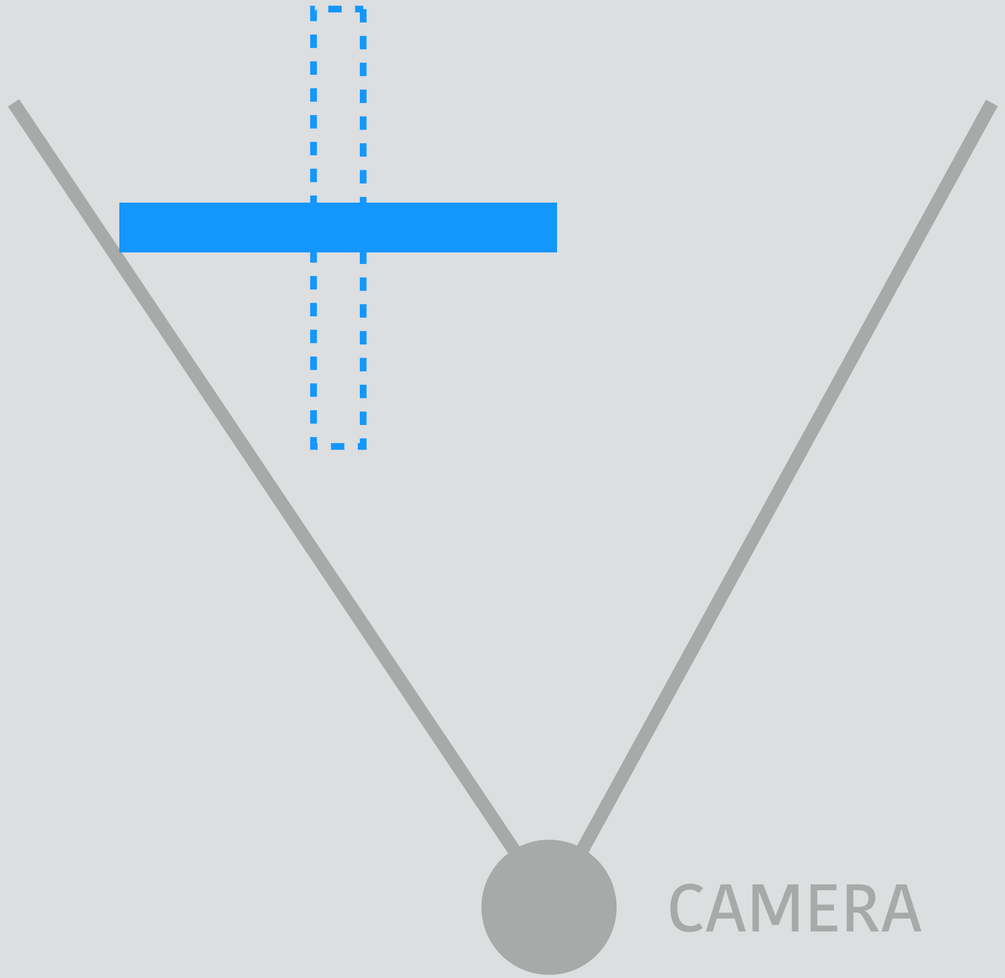
Method #1 - Rotate all billboards towards camera.

Viewpoint oriented

```
for(int i=0; i < entities.size(); i++) {

    if(entities[i]->billboard) {

        float directionX = entities[i]->position.x - camera->position.x;
        float directionZ = entities[i]->position.z - camera->position.z;

        entities[i]->rotation.y = atan2f(directionX, directionZ);
    }

    entities[i]->Render();
}
```

Method #2 - Remove all rotation from the modelview matrix.

Viewplane aligned

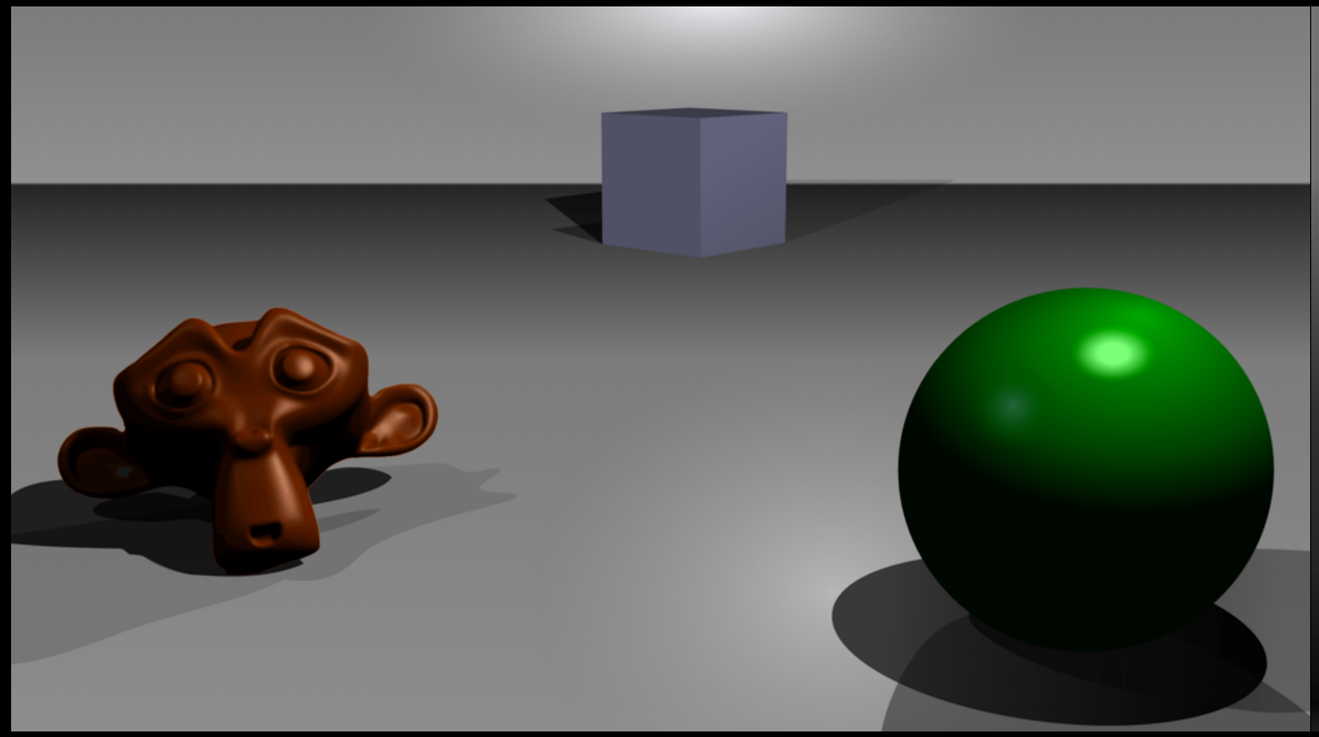


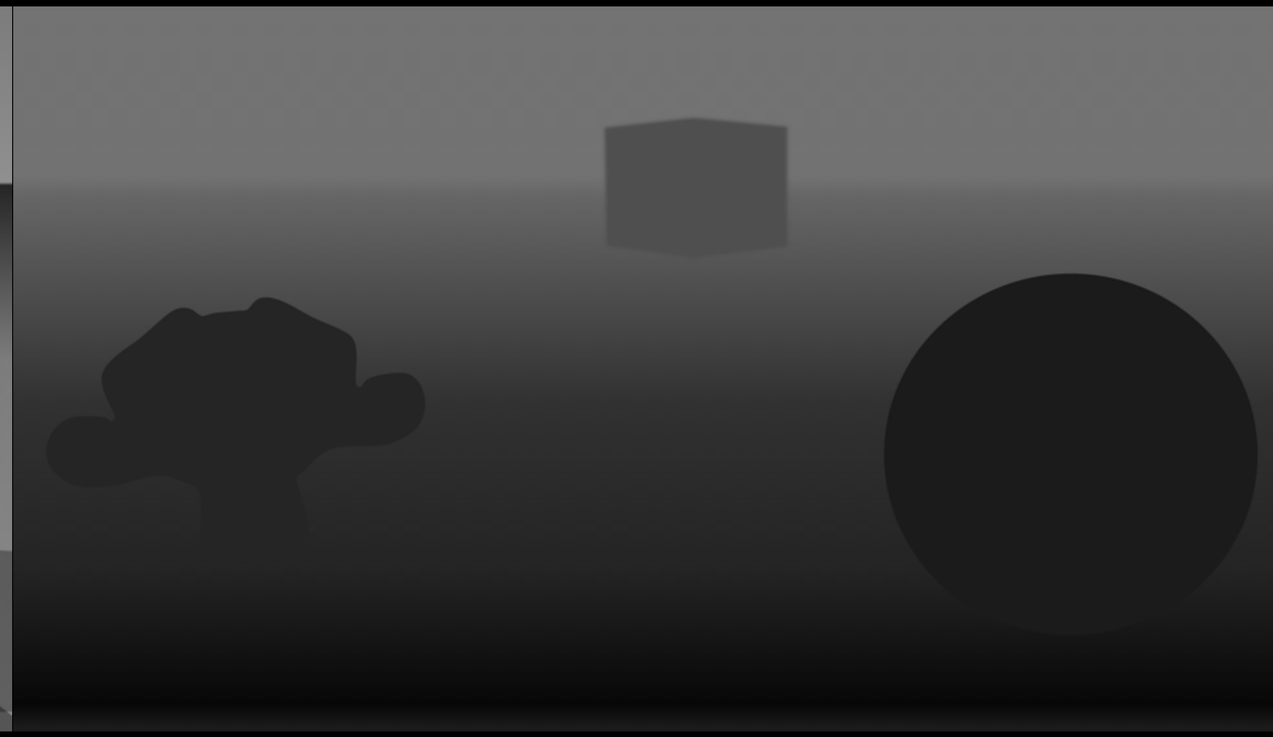
Method #2 - Remove all rotation from the modelview matrix.

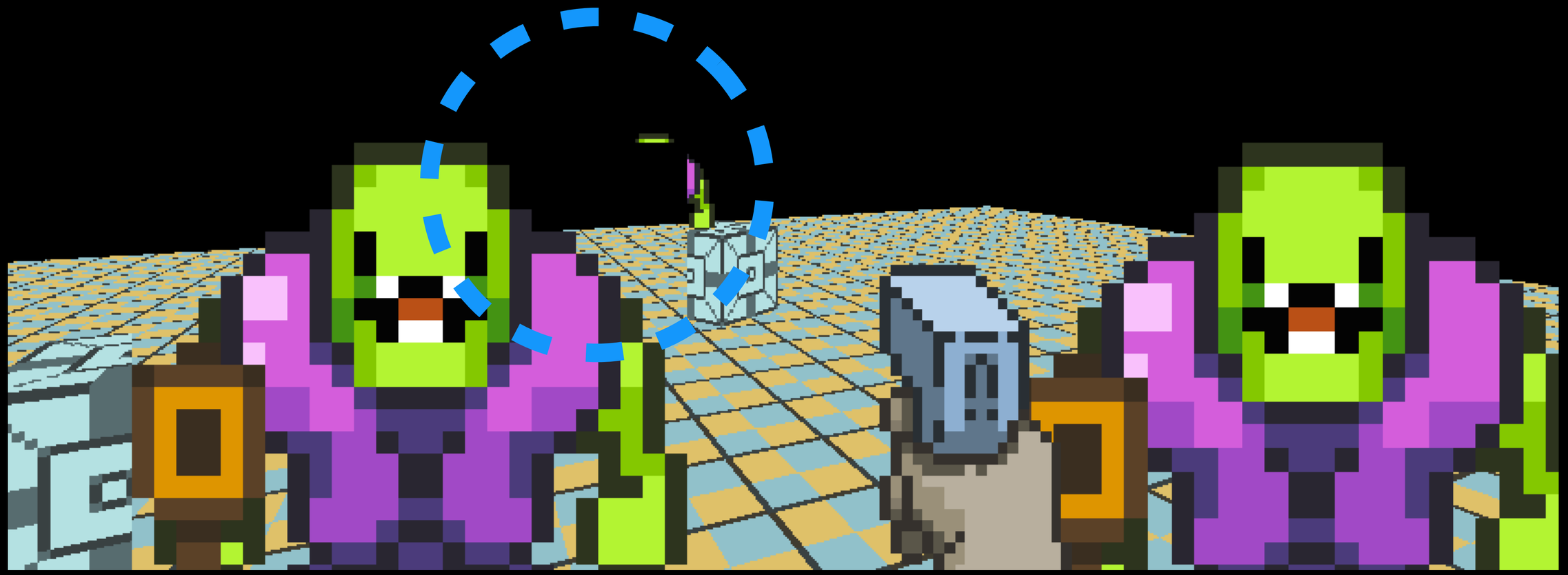
Viewplane aligned

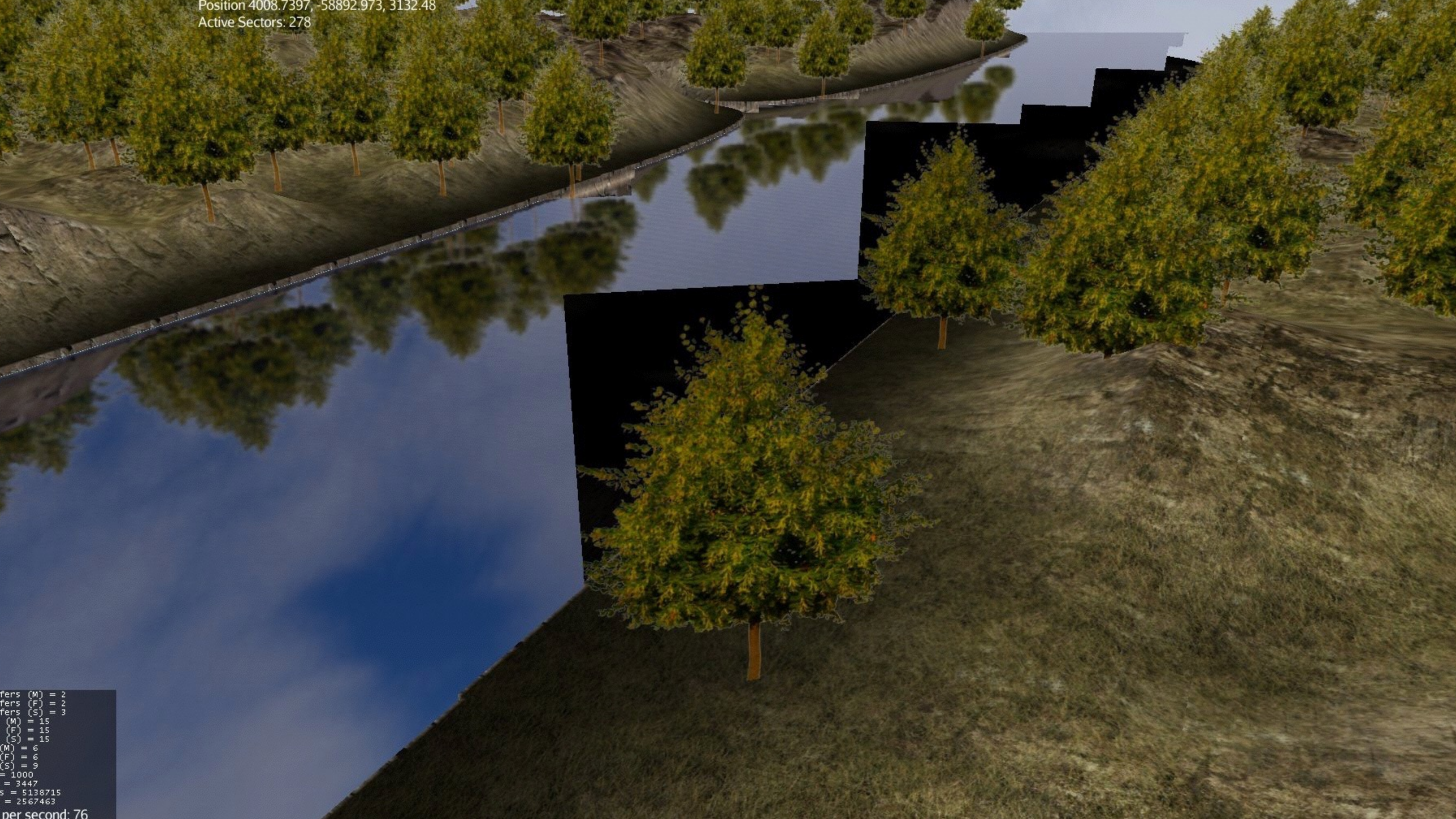
```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

varying vec2 texCoordVar;
void main()
{
    mat4 modelViewMatrix = viewMatrix * modelMatrix;

    modelViewMatrix[0][0] = 1.0;
    modelViewMatrix[0][1] = 0.0;
    modelViewMatrix[0][2] = 0.0;

    modelViewMatrix[1][0] = 0.0;
    modelViewMatrix[1][1] = 1.0;
    modelViewMatrix[1][2] = 0.0;

    modelViewMatrix[2][0] = 0.0;
    modelViewMatrix[2][1] = 0.0;
    modelViewMatrix[2][2] = 1.0;

    vec4 p =  modelViewMatrix * position;
    texCoordVar = texCoord;
    gl_Position = projectionMatrix * p;
}
```

Blending and the **Z-Buffer**








Position 4008.7397, -58892.973, 3132.48
Active Sectors: 278
fers (M) = 2
fers (F) = 2
fers (S) = 3
(M) = 15
(F) = 15
(S) = 15
(M) = 6
(F) = 6
(S) = 9
= 1000
= 3447
s = 5138715
= 2567463
ner second: 76

Discarding fragments.

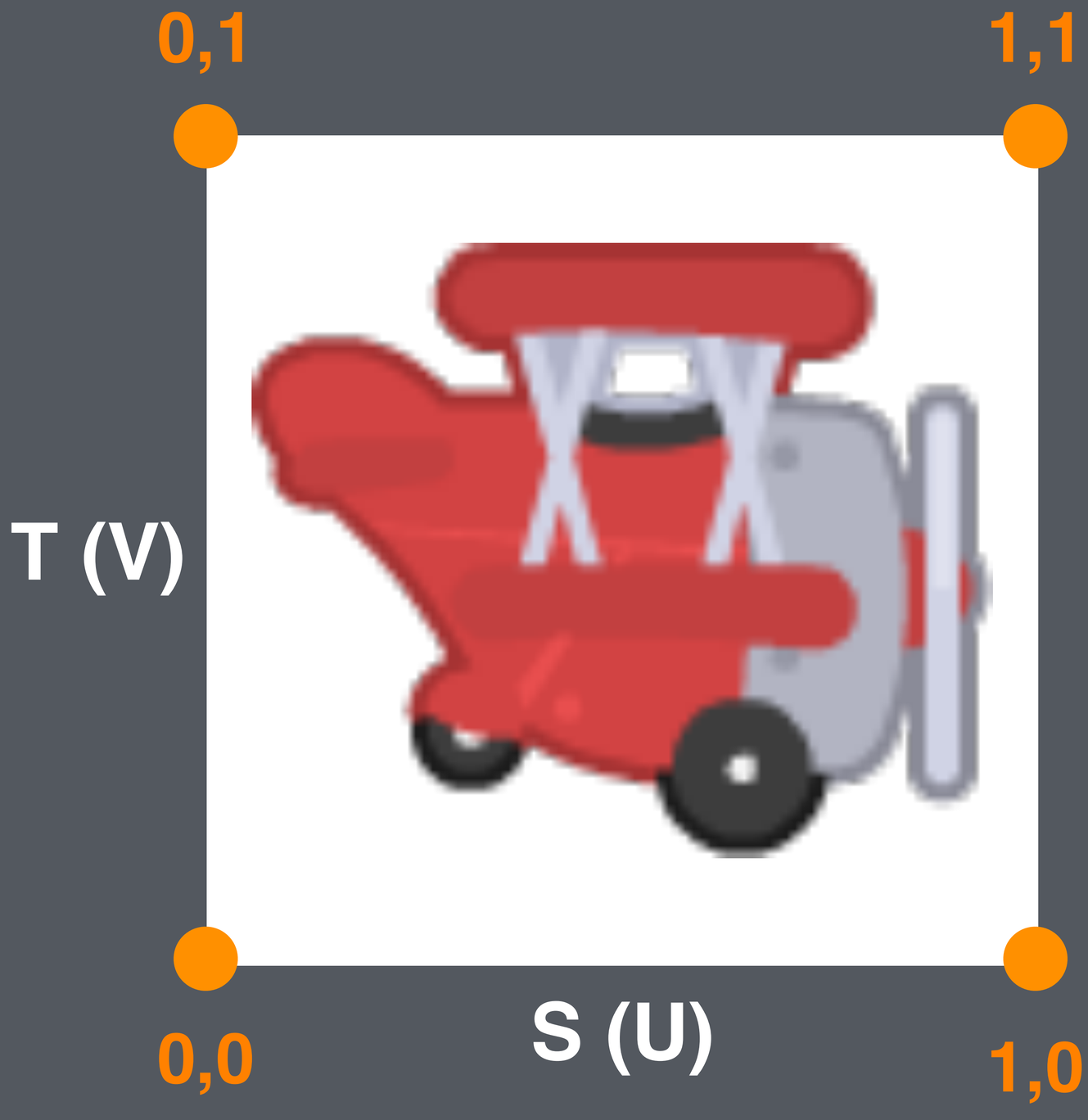
Discard fragment if alpha value is 0.

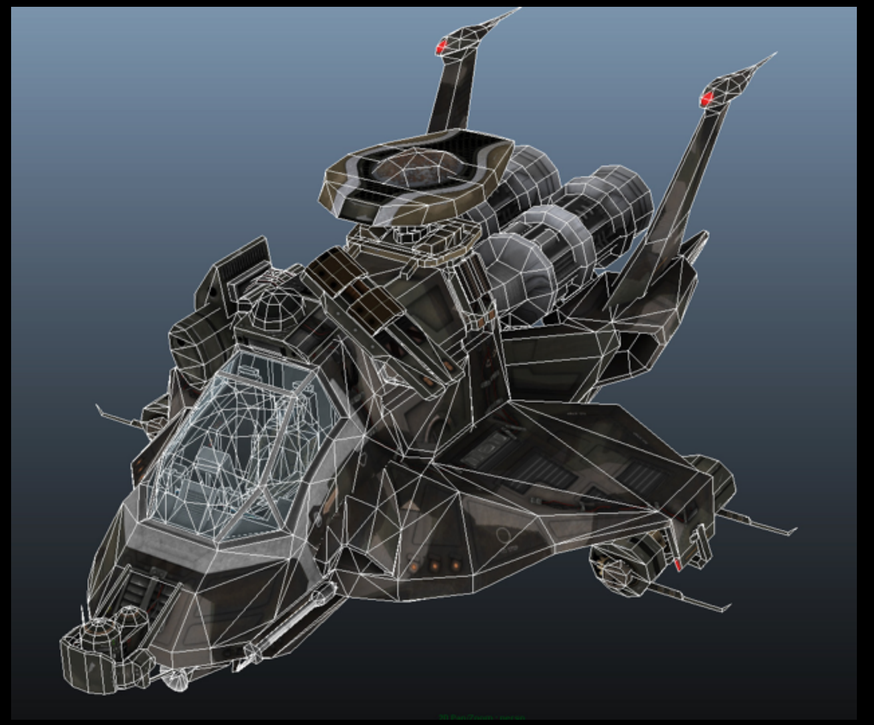
```
uniform sampler2D diffuse;
varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D(diffuse, texCoordVar);
    if(gl_FragColor.a == 0.0) {
        discard;
    }
}
```

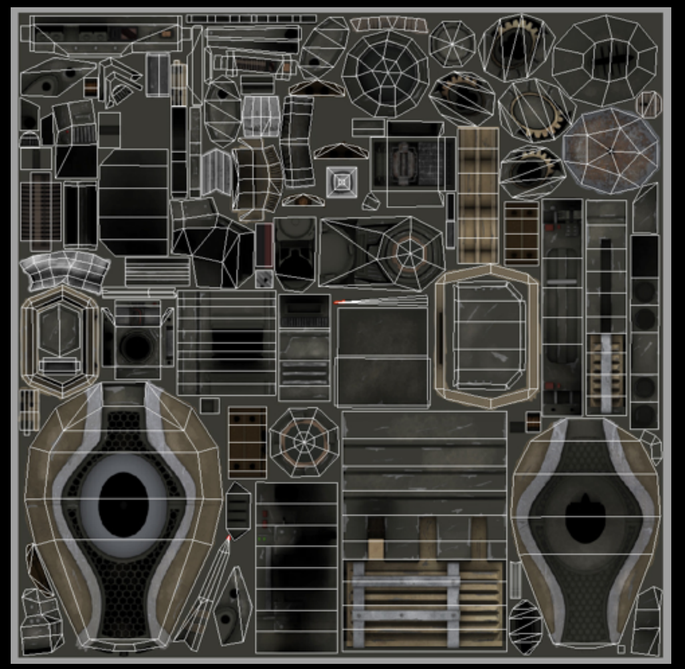
Working with **3D meshes**.

Position coordinates.

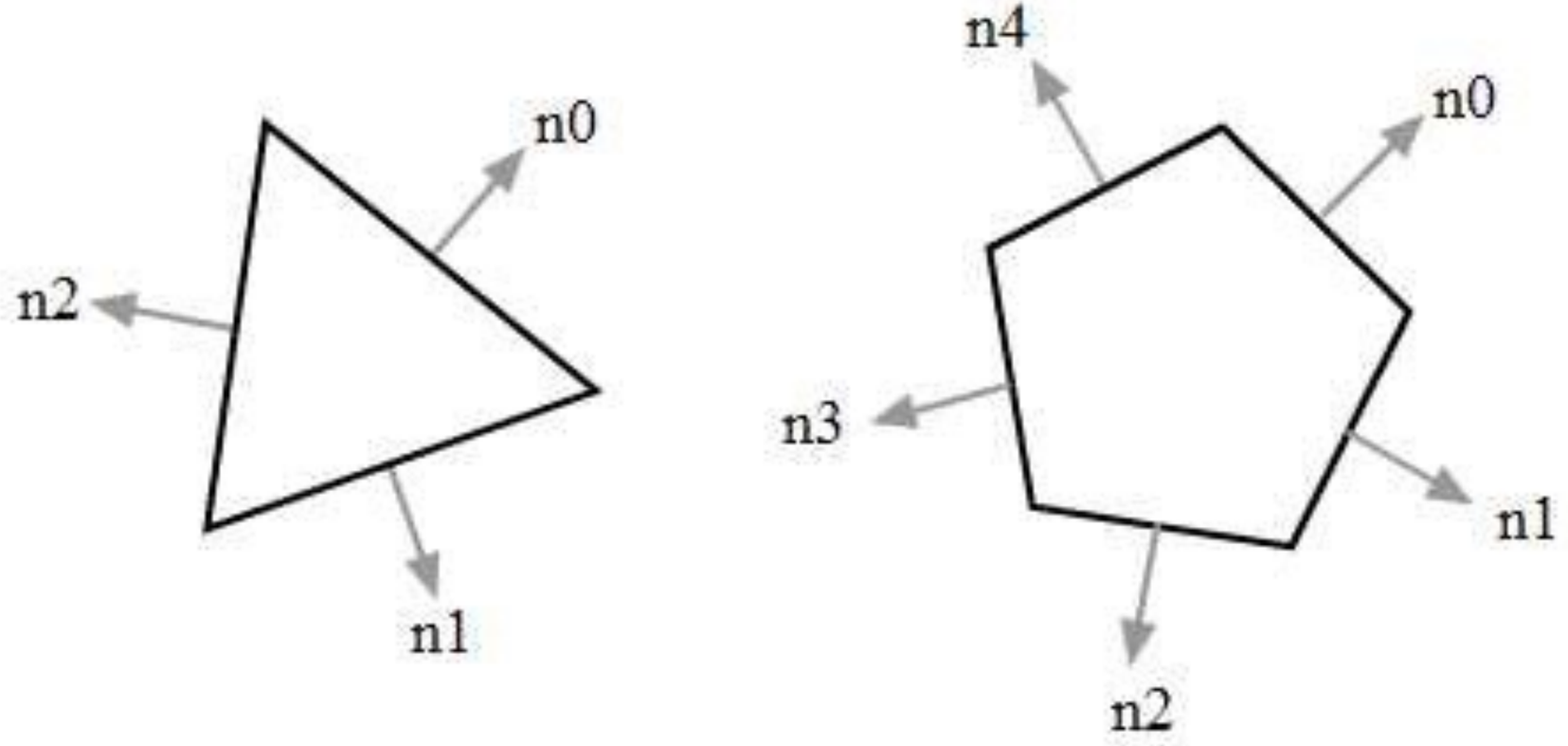
Texture coordinates.

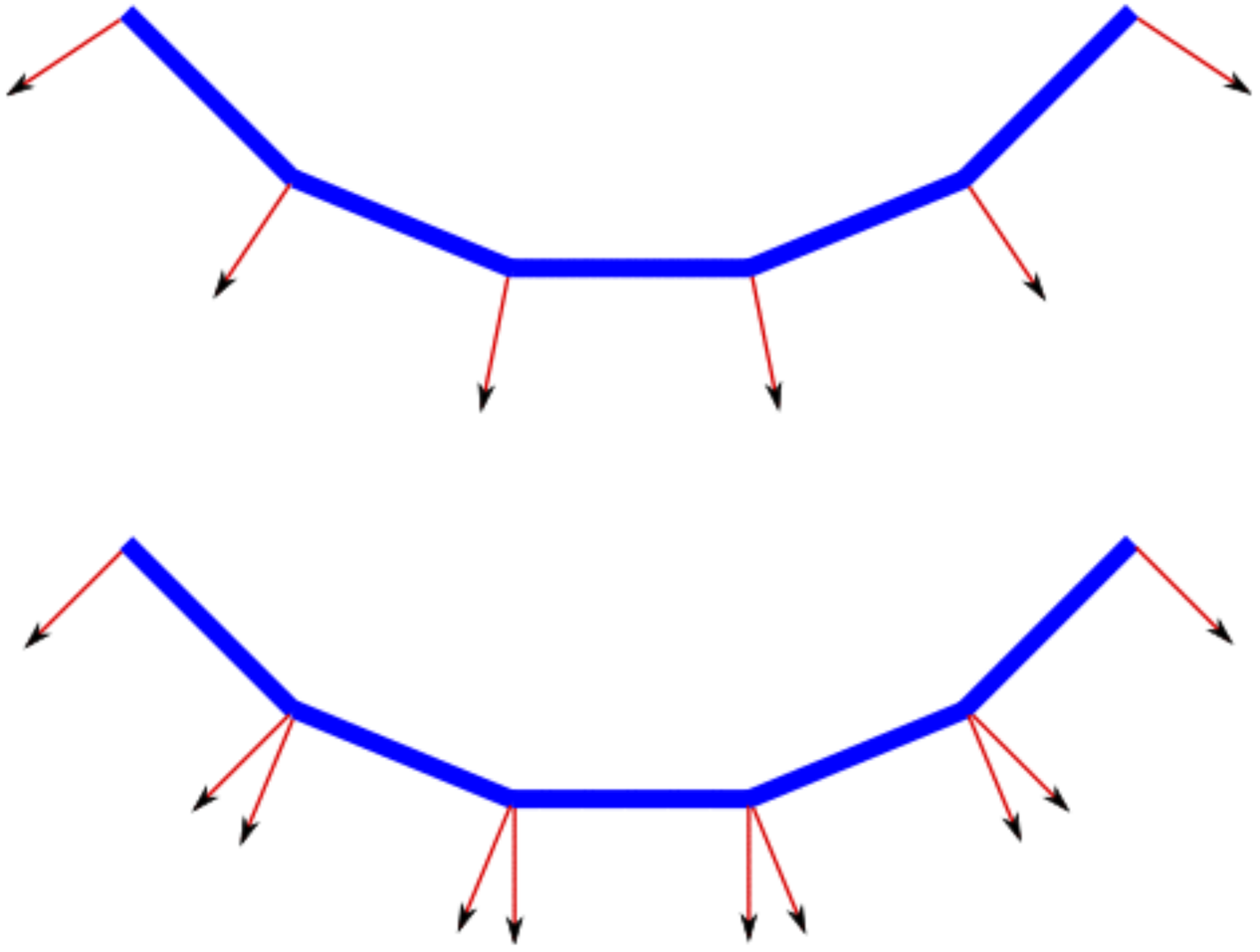

0,1
1,1
T (V)
S (U)
0,0
1,0

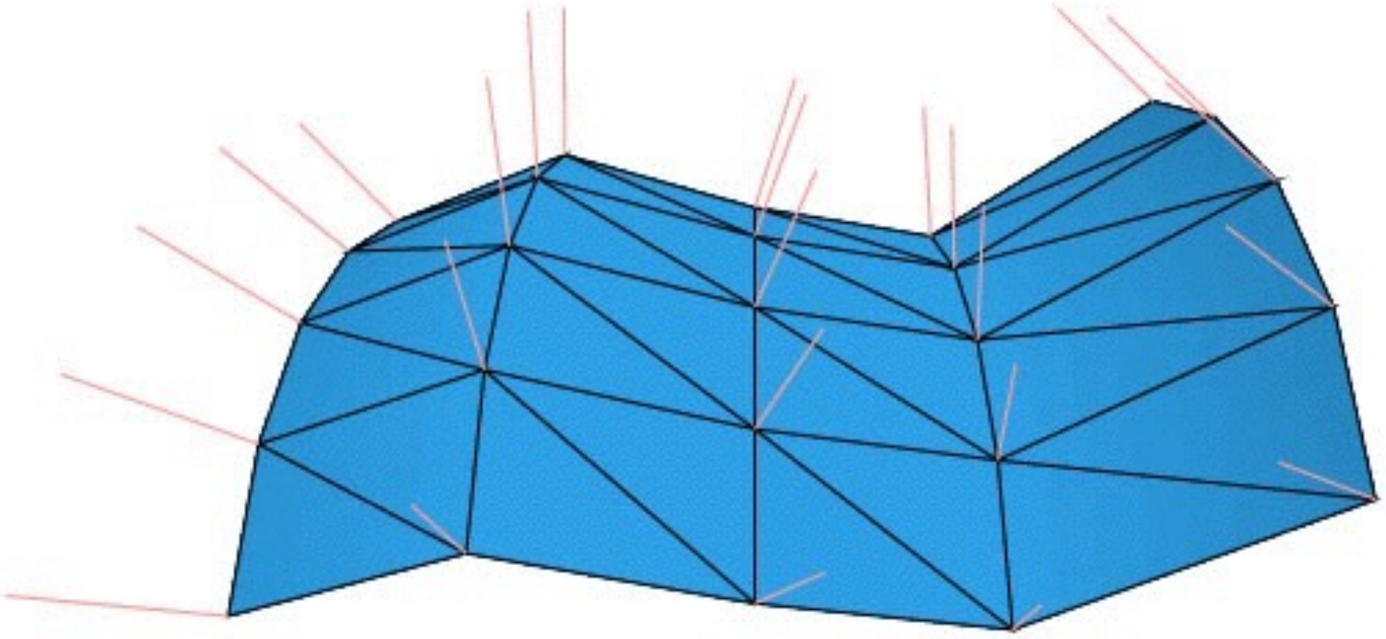


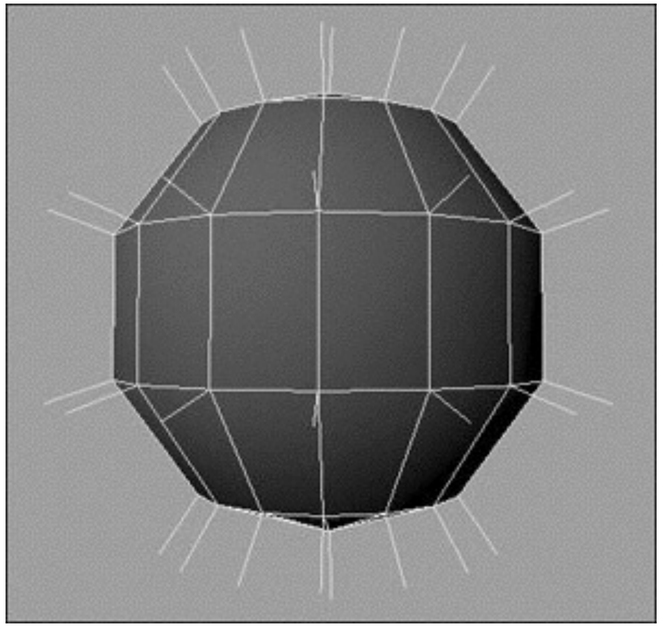


Vertex **normals**.


n0
n2
n1
n4
n0
n3
n1
n2







```
class Mesh {
    public:

        void Render(ShaderProgram *program);
        void loadOBJ(const char *fileName);

        std::vector<float> vertices;
        std::vector<float> uvs;
        std::vector<float> normals;
};

void Mesh::Render(ShaderProgram *program) {

    glVertexAttribPointer(program->positionAttribute, 3, GL_FLOAT, false, 0, vertices.data());
    glEnableVertexAttribArray(program->positionAttribute);

    glVertexAttribPointer(program->texCoordAttribute, 2, GL_FLOAT, false, 0, uvs.data());
    glEnableVertexAttribArray(program->texCoordAttribute);

    glDrawArrays(GL_TRIANGLES, 0, vertices.size()/3);

    glDisableVertexAttribArray(program->positionAttribute);
    glDisableVertexAttribArray(program->texCoordAttribute);
}
```

Loading meshes from **OBJ files**.

Basic **OBJ** file structure

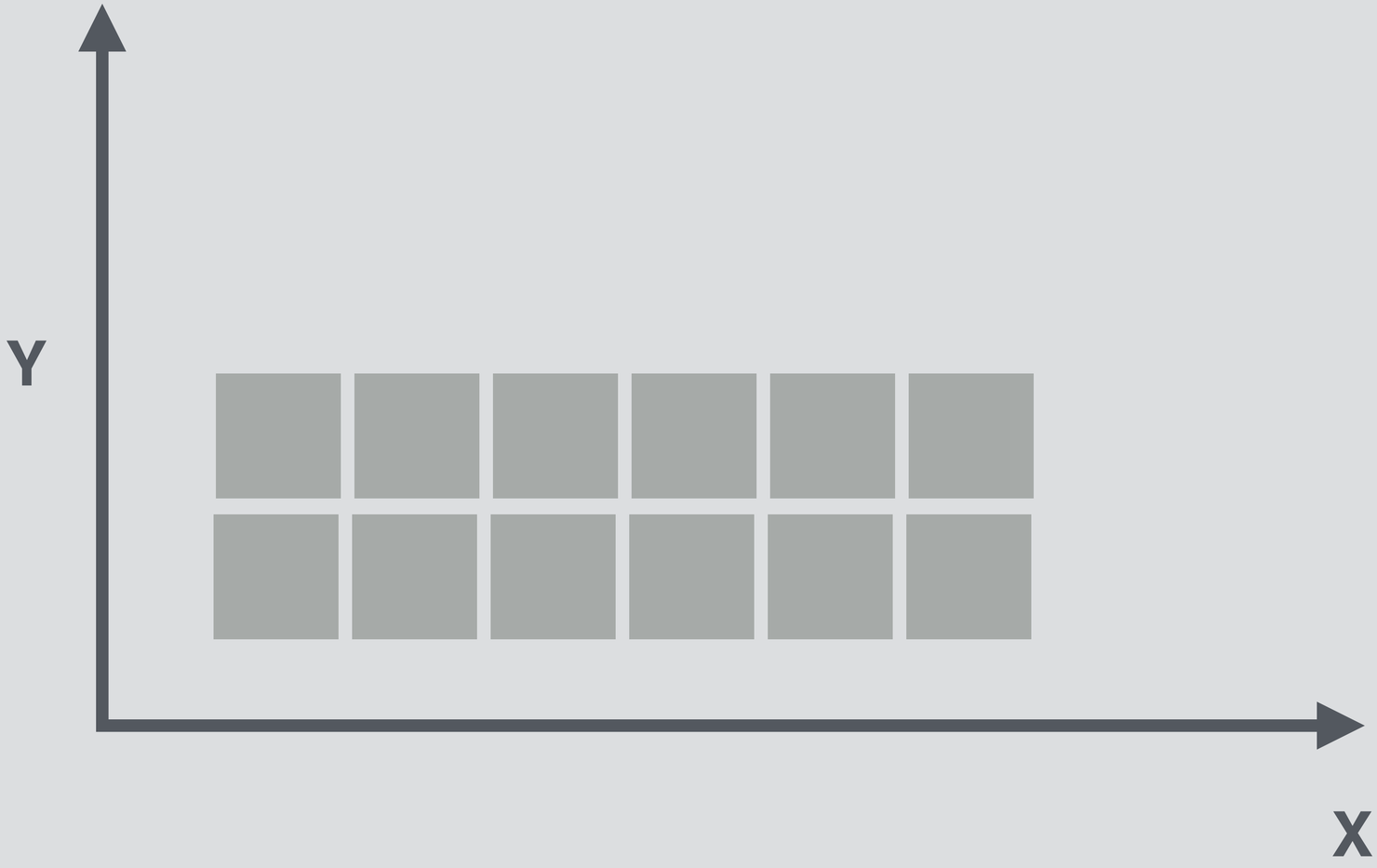
Indexes are **1-based**, V is **inverse in Texture Coordinates**.

```
# comment

# vertex
v -0.436826 -0.208037 1.060532

# texture coordinate
vt 0.859375 0.500000

# vertex normal
vn 0.707 0.000 0.707

# indices
# can reference vertex
# or vertex / texture coordinate
# or vertex / texture coordinate / normal
f 21/10/3 16/11/5 1/7/2
```

```
void Mesh::loadOBJ(const char *fileName) {
    ifstream infile(fileName);
    string line;
    std::vector<float> fileVertices;
    std::vector<float> fileUVs;
    std::vector<float> fileNormals;
    while (getline(infile, line)) {
        istringstream sStream(line);
        string token;
        getline(sStream, token, ' ');
        if(token == "v") {
            while(getline(sStream, token, ' ')) {
                fileVertices.push_back(atof(token.c_str()));
            }
        } else if(token == "vn") {
            while(getline(sStream, token, ' ')) {
                fileNormals.push_back(atof(token.c_str()));
            }
        } else if(token == "vt") {
            while(getline(sStream, token, ' ')) {
                fileUVs.push_back(atof(token.c_str()));
            }
        } else if(token == "f") {
            while(getline(sStream, token, ' ')) {
                istringstream faceStream(token);
                string faceToken;
                int type = 0;
                while(getline(faceStream, faceToken, '/')) {
                    int index = atoi(faceToken.c_str())-1;
                    switch(type) {
                        case 0:
                            vertices.push_back(fileVertices[index*3]);
                            vertices.push_back(fileVertices[(index*3)+1]);
                            vertices.push_back(fileVertices[(index*3)+2]);
                        break;
                        case 1:
                            uvs.push_back(fileUVs[(index*2)]);
                            uvs.push_back(1.0f - fileUVs[(index*2)+1]);
                        break;
                        case 2:
                            normals.push_back(fileNormals[(index*3)]);
                            normals.push_back(fileNormals[(index*3)+1]);
                            normals.push_back(fileNormals[(index*3)+2]);
                        break;
                    }
                    type++;
                }
            }
        }
    }
}
```

Loading a **2D tilemap level** in 3D

In 2D



In 3D