Project Report - Second assignment - GAME ENGINES

Andrea Distler

05 December, 2012

Contents

1 Introduction

This is the project report for the second assignment of the Game Engines course.

A short description of the source code and the used libraries can be found in section 2. The files are available on GitHub. The libraries are also made available on GitHub for easier installation.

In section 3 the outcome is described. In section 4 used mechanics and an overview of the classes can be found.

2 The Files

All files can be found on GitHub.
`https://github.com/JungleJinn/GE---SimplePlatformerEngine`.

2.1 Used Libraries

The project was written in C++, using additional libraries:

- glm. Installed manually. Files in zip
- SDL 1.2.15. `http://www.libsdl.org/download-1.2.php` Used environment variable named SDK_SDL

3 Outcome

The assignment was to create a 3-dimensional wire-frame renderer from scratch. On the screenshot a cube rendered with the created renderer is depicted.

4 Classes and Mechanics

4.1 Renderer

It is the class managing the meshes and the right application of the camera's and the meshes' matrices. It has got a Render method which goes through all meshes and converts their vertex coordinates to drawable screen coordinates.

The conversion calculations used are like shown in listing 1.

Listing 1: The conversion from object coordinates to screen coordinates.

```
// convert vertex from object coords to clip coords
mat4 modelViewProjectionMatrix = camera.projectionMatrix *
  camera.viewMatrix * mesh->modelMatrix;

v = modelViewProjectionMatrix * v;

// normalize clip coordinates / get normalized device
    coordinates
v = v / v.w;

// clip vertices outside the viewport
if ((v.x < -1 || v.x > 1) ||
  (v.y < -1 || v.y > 1))
{
  // go to next vertex if this one is outside
  continue;
}

// map clip coordinates to screen coordinates
float width = screen->w;
float height = screen->h;
v.x = v.x * width * 0.5f + width * 0.5f;
v.y = -v.y * height * 0.5f + height * 0.5f;

vertex->screenPos = vec2(v.x, v.y);

drawPixel(vertex->screenPos, screen, 255, 0, 0);
```

4.2 Camera

It creates the projection and the view matrix. The view matrix is constructed like in listing 2. The camera has a position, a target which it looks at and an up vector.

Listing 2: The View Matrix

```
vec3 f = normalize(target - position); // lookdirection vector
vec3 s = normalize(cross(f, up)); // orthogonal to layer
    defined by lookdirection and up
vec3 u = cross(s, f);
```

```

viewMatrix = mat4(
  s.x, u.x, -f.x, 0,
  s.y, u.y, -f.y, 0,
  s.z, u.z, -f.z, 0,
  -dot(s, position), -dot(u, position), dot(f, position), 1
  );
```

In the following listing (listing 3) the perspective projection matrix calculation is shown.

Listing 3: The Perspective Projection Matrix

```
float e = 1 / tan(frustum.fov * 0.5f);
float A = e / frustum.aspect;
float B = (-frustum.far - frustum.near) / (frustum.far -
    frustum.near);
float C = (-2 * frustum.far * frustum.near)/(frustum.far -
    frustum.near);


projectionMatrix = mat4(
  e, 0, 0, 0,
  0, A, 0, 0,
  0, 0, B, C,
  0, 0, -1, 0
  );
```

4.3 Mesh

A mesh contains a vector of vertices which store their positions. Meshes are represented as line loops. The vertices are drawn in the same order in which they are stored in the vector. When the end of the vector has been reached, the last vertex is connected to the first one. Meshes have a homogeneous model matrix describing the mesh's orientation, position and scale.

Listings

List of Figures

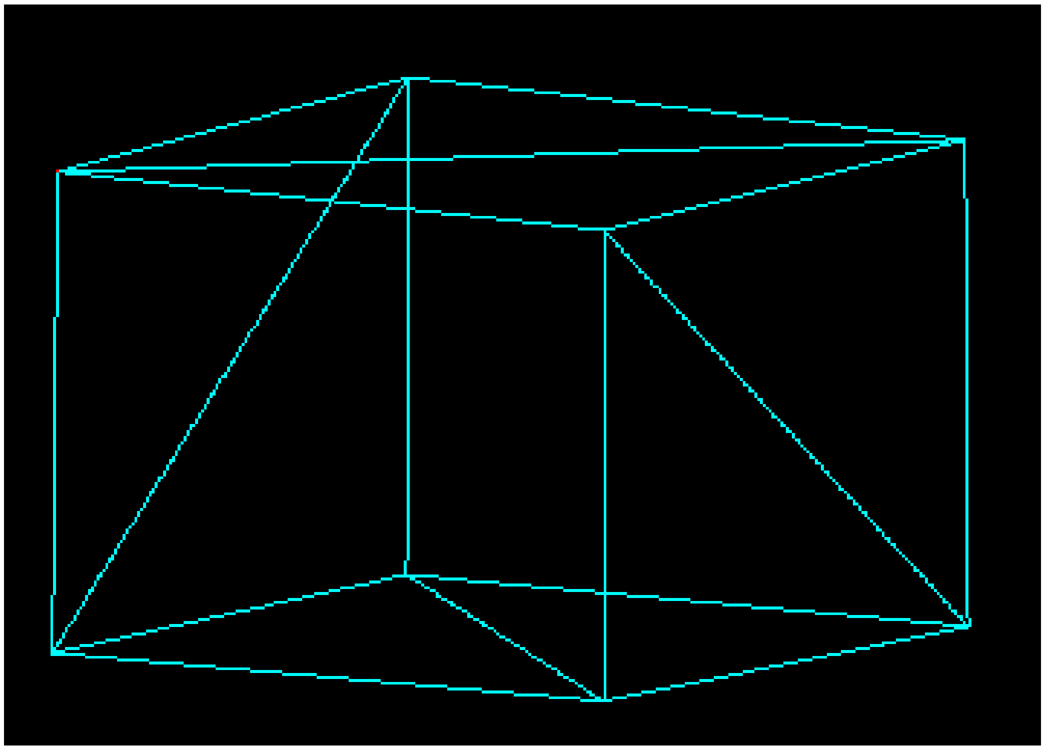

Figure 1: A screenshot of the result.