glEngine : Technical Specifications

Gaël Jochaud du Plessix
Loick Michard

2012

Contents

Part I
Introduction

This document describes the technical specifications for the glEngine project. It comes with the Specifications document that explain the functional specifications. The technical specifications document describes the retained solutions for the glEngine project. These specifications are divided in two parts: the description of the scene management and how the 3D rendering work.

Part II
Scene management

In order to render a virtual world in 3D, the engine must manage a lot of data describing a scene. This part of the specifications describes the different elements that can be part of a scene and how they are arranged hierarchically by the engine.

1 Scene elements

A scene is composed of various kinds of elements that describe all the aspects of the virtual world to render.

1.1 Materials

Materials allow to specify how an object visually appear when rendered. It contains the following information:

- Ambient color
- Diffuse color
- Diffuse intensity
- Specular color
- Specular intensity
- Shininess
- Color map (texture)

A material also have a name, in order to identify it.
All these information are stored in a uniform buffer, in order to be efficiently used by the GPU.

1.1.1 Creation

The engine allows the creation of materials programmatically via the gle::Material class. All the properties of a material can be accessed and set via accessors.

1.1.2 Import

Materials can be imported from .mtl files through the .obj importer (Cf. section 1.2.2).

1.2 Models

Models are the virtual representation of 3D objects. They are characterized by:

- A geometry
- A position
- A rotation matrix
- A scale matrix

Models must also have a material (Cf. Section 1.1) that describes how they appear.

1.2.1 Creation

The user can create a model programmatically. Thus, glEngine provides the gle::Mesh class.
Position, rotation and scale can be accessed via simple functions.
To specify the mesh geometry, there are several accessors that store their data directly on the hardware through the OpenGL APIs:

- gle::Mesh::setVertexes
- gle::Mesh::setNormals
- gle::Mesh::setTextureCoords
- gle::Mesh::setIndexes

All these data can be provided in two ways: simple (GLfloat|GLuint)* buffers or gle::Array<(GLfloat|GLuint)> arrays.

1.2.2 Import

The engine allows the import of a mesh from a Wavefront OBJ file, thanks to the gle::ObjLoader class. Its function gle::ObjLoader::load returns a mesh imported from a .obj file. If the file includes .mtl files describing the model materials, they will also be imported (Cf. Section 1.1).

1.2.3 Update

Models can be updated at any time. To set new values on a gle::Mesh, the user can use the corresponding accessors functions.

1.3 Lights

Lights describe how a scene is illuminated. There are two types of lights: Directional an Point lights.
Point lights are characterized by a position, a diffuse color and a specular color. Directional lights have a direction vector and a diffuse color.

1.3.1 Creation

Lights can be created programmatically by the user through the classes gle::DirectionalLight and gle::PointLight, specifying all their properties manually.

1.3.2 Update

It is possible to update the properties of the lights at any time. Once the properties of a light has changed, the user must call gle::Scene::updateLights on the scene to update the lights meta-data.

1.4 Cameras

Cameras permit to specify the point of view of the observer in a scene. They have several properties common to other scene elements such as a position and an orientation (specified by a target). A camera also have specific information: a field o view, an aspect and the distances to near and far plane.

1.4.1 Creation

Cameras are created programmatically by the user via the class gle::PerspectiveCamera, specifying all properties at object construction.

1.4.2 Update

It is possible to update the properties of a camera at any time, through accessors methods provided by gle::Camera.

2 Structural information

Scene elements are structured hierarchically and have particular meta-data in order to improve the rendering process.
These data include information about whether the element is static or dynamic, its position in the scene graph and a representation of the world based on spatial partitioning.

2.1 Elements hierarchy

Each element of the scene must is represented as the node of a graph, in order to hierarchy them. Thus, each element have a parent and a list of child elements.
In addition, it is possible to name the elements and retrieve them easily within the scene graph, thanks to the functions gle::Mesh::getChildren and gle::Mesh::getChildrenByName.

2.2 Static vs Dynamic

In the scene, there are two main types of elements: static ones and dynamic ones.
As their name suggest, static elements are not subject to change with time. Thereby, the renderer can improve its performance when processing these elements.
Dynamic elements may rather be updated very often, so the engine provides ability to efficiently update the data of these elements, through the mesh accessors functions.

2.3 Spatial partitioning

In order to improve the rendering performance, the scene have a representation of its elements based on spatial partitioning. Thanks to that representation, the engine is able to efficiently treat a lot of elements.
This partitioning is made with an Octree. Each mesh has an associated bounding volume computed from its shape. Thanks to that, the partitioning tree can be made efficiently.

Part III

3D Rendering

This part of the technical specifications describes the how the different features for 3D rendering work.
It lists the basic and advanced features of the renderer and describes the techniques retained for performance improvements.

3 Basic features

This section describes the main features that are supported by the glEngine renderer.

3.1 Rasterisation

The renderer is able to rasterize triangles processed from the scene graph, using the OpenGL 4.2 API.

3.2 Textures

Triangles rasterized by the renderer can be textured. It is possible to configure the quality of the textures and how they are rendered.

4 Advanced features

Some visual effects are added to the renderer in order to improve the realism of the rendered image.
These advanced features are be:

- Diffuse shading
- Phong shading

5 Performance

In order to provide an efficient rendering process, some techniques are used by the glEngine to improve its performance capabilities.

5.1 Metadatas processing

The engine is able to process the scene meta-data in order to improve the rendering performance. Furthermore, it is able to perform a frustum culling.

5.2 OpenGL 4.2

Since this is the main purpose of this project, the glEngine takes advantage of the features of OpenGL 4.2 APIs to improve the rendering performance.
Thus, it uses Uniform Block Arrays to pack together all the mesh uniforms and reduce the number of draw calls.