TDDC32 - Software Requirements specification 0.1

Ray Tracing Engine

Simon Vernhes <simon@vernhes.eu>

25 février 2010

Abstract

This document give a complete description of the behavior of the Ray Tracing Engine. This project is the main part of the TDDC32 - Design and implementation of a software module in Java course.

This document include the system descriptions, the user interface. It also include all the functional and non functional requirements for this project. At the end of this document, you can see see the use cases descriptions.

The purpose of this document is not to discuss about implementation issues but focus on how the Ray Tracing Engine will be.

Table des matières

1 Introduction

Early, computer graphic got the ambition the produce photo realistic images.

Ray tracing is a way to produce visual images. The main advantage of this technique among others is the photorealism of produced image.

To produce an image, a ray tracing engine trace the reverse path of the light, from the virtual eye (camera) through a virtual screen (the image) and calculate the color of visible objects. You can see an example on the figure 1.

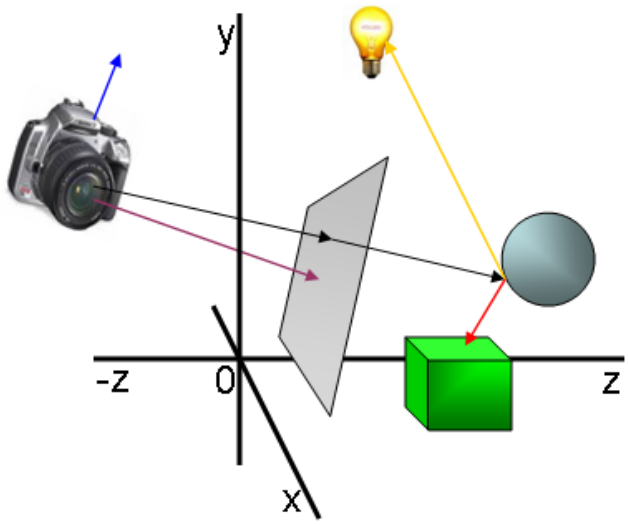

FIGURE 1 – Ray tracing

The eye (camera) look at a direction (purple arrow). A ray is traced through each pixel of the image (black arrow). The engine will try to find out where the ray intersect with an object (the sphere). The engine will find the color for this intersection by looking for the light and eventually reflexion of other objects (the cube).

2 Document conventions

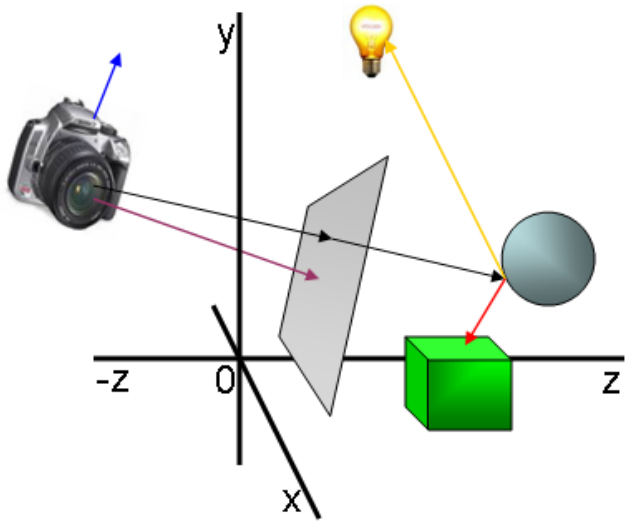

FIGURE 2 – Ray tracing

- eye : camera
- viewport : plan where the eye is looking at
- shading effect : way to render/color a pixel (ray intersection with an object)
- scene : the scene the ray tracer have to render (described in the xml scene file)

3 System description

The ray tracing engine will be divided into 3 parts :
- XML parser
- World objects
- Engine

The xml parser have an xml file as input. This xml must describe the scene which will be rendered. The syntax for this xml will be describe in the analysis phase and will be handed with a XML Scheme descriptor. This parser will produce the World object.

The world objects are all the object which can be used to describe a scene. The exhaustive list of this object are :
- Sphere
- Cube
- Cylinder

We can also do transformations on this objects : rotation, translation, scaling.

Finally, the engine will render an image using the world objects.

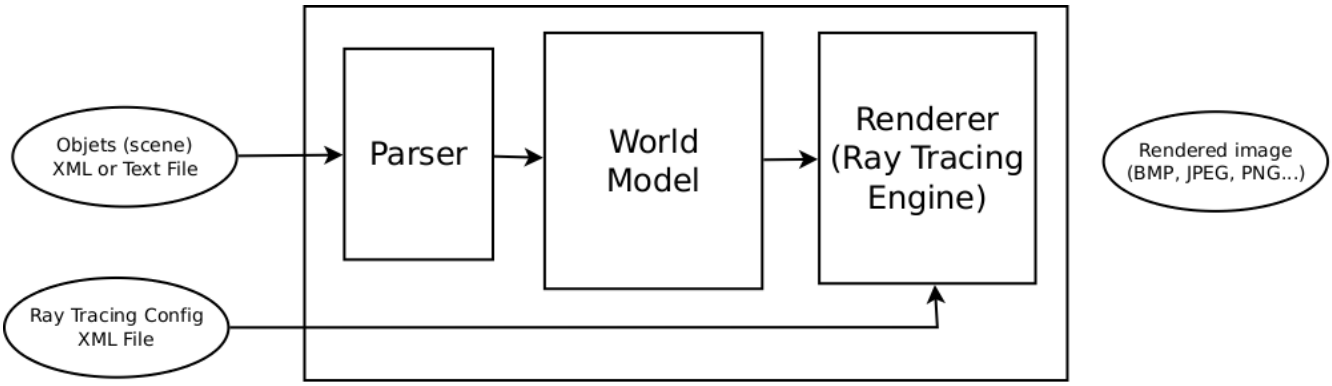

FIGURE 3 – Modules

4 User interface

The first user interface will only be command line.

The user will give a xml file as argument of the command call. Eventually a configuration for the ray tracing engine could be given. If no configuration file is given, the software will check if a file called .raytracer.xml exists in the home directory of the user (ie.*~/.raytracer.xml*). If not, the default parameters will be used.

Standard call :

```
$ raytracer scene.xml
```

Call with configuration file :

```
$ raytracer scene.xml -c config.xml
```

5 System Functions

5.1 Mandatory functionalities

We expect than the ray tracing engine should be able to render basic scene, composed by sphere, cube, cylinder, plan. The will be only light, supposed to be placed at the infinite. Each object should have a color.

So, the ray tracing engine should render a single image with on input scene description xml file.

The ray tracing engine muse let the user adjust :
- list of objects (sphere, cube, cylinder, plan)
- parameters of these objects (position, size, rotation, color)
- a phong coefficient for each object
- a specular and diffuse reflexion for each object
- give the position of a unique light
- give the position and direction of the eye

The parser should show errors in incorrect scene xml files.

5.2 Optional functionalities

As optional functional, we have a lot of options.
- Adding new objects : torus, more generic surface of revolution...
- Allowing more than one light not necessarily placed at the infinite.
- Adding textures to objects
- Implement reflexion of objects on others objects.
- multi-threaded ray tracing (and eventually distrubuted among a set of machine)
- adding a GUI
- provide live view generation of the image

6 Non functional requirements

No performance or security requirement for this project.

The xml scene description must be easy to create.

The software must be really sustainable.

7 Storage of permanent date

Eventually, a config file could be store by the user in his home directory (i.e.*~/.raytracer.xml*)

8 Limitations

This ray tracer is not a real-time raytracer so it firstly can only be use only to produce one image.

We will not take care of too long computations to render an image.

9 Use Case Description

9.1 General view

The user should call the ray tracer engine providing a scene xml file. The parser will create the world objects. And finally the engine will render the image using the world objects.

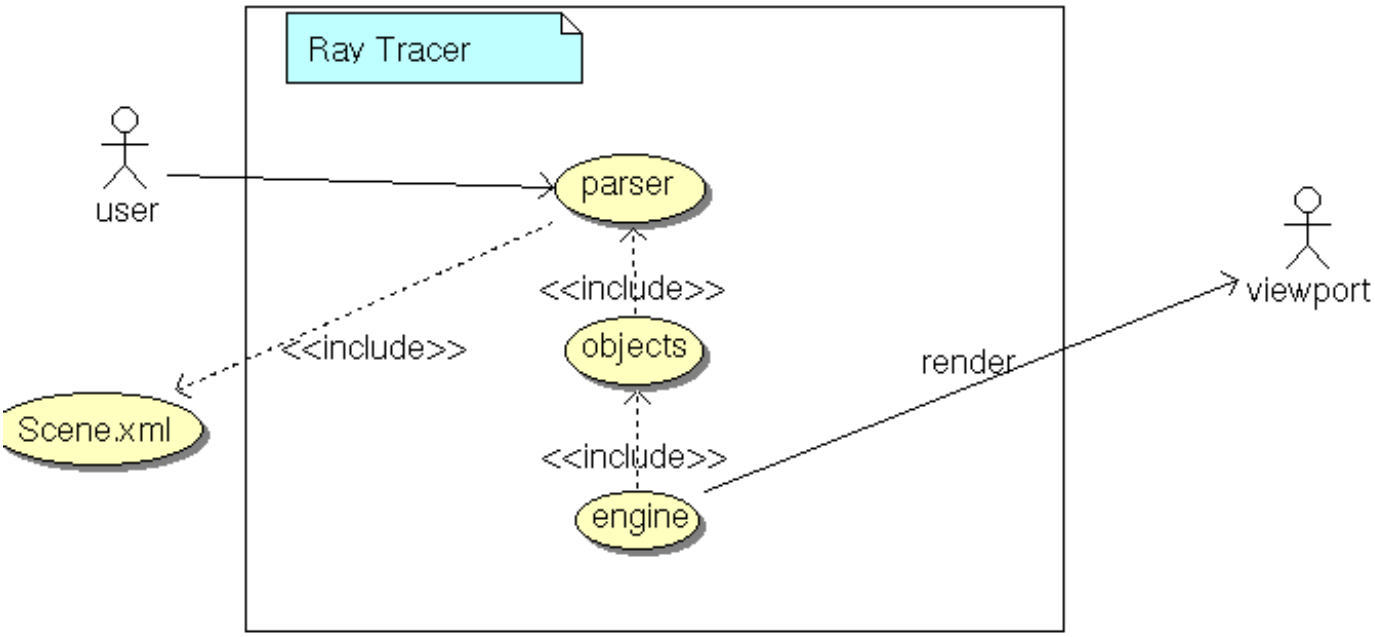

FIGURE 4 – Use case general view

9.2 Objects

The parser create all objects of the scene.
The creation of the camera also create the viewport.

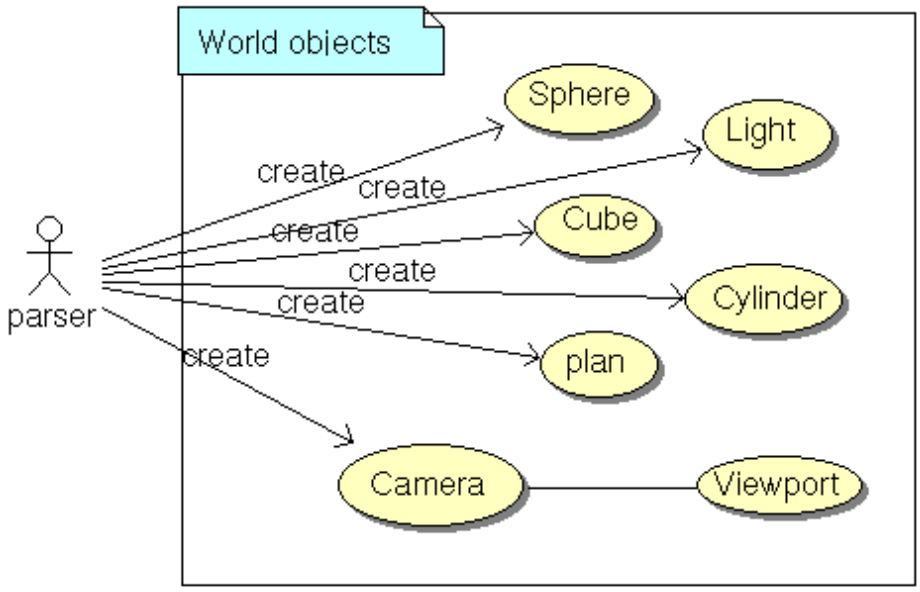

FIGURE 5 – Use case objects creation

9.3 Engine

Engine algorithm : for each pixel of the viewport, throw a ray and calculate the color where the ray intersect with an object.

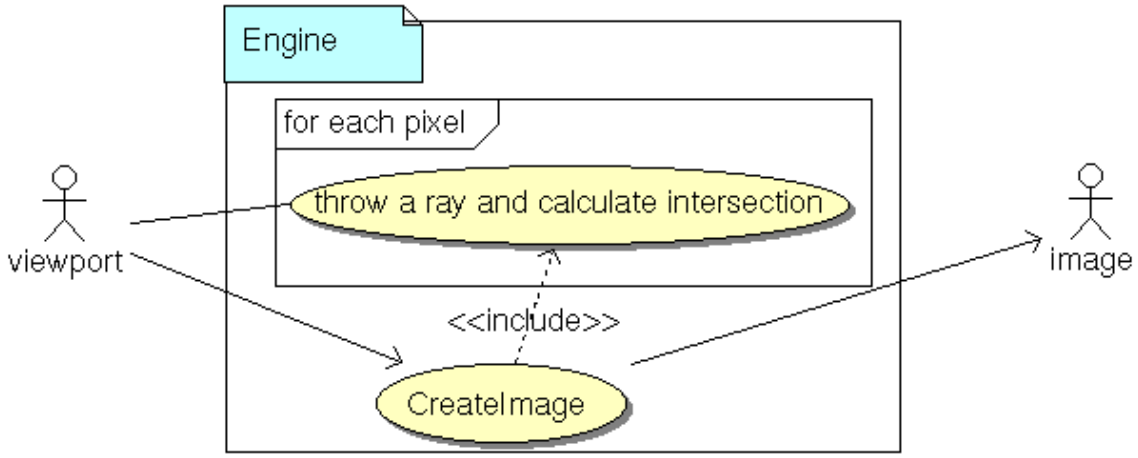

FIGURE 6 – Use case engine