TDDC32 - Analysis and Design Document 1.1

Ray Tracing Engine

Simon Vernhes <simon@vernhes.eu>

April 22, 2010

Abstract

This document give a complete analysis and design 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 a whole set of diagram of UML analysis and design, including class diagrams, class description, use case diagrams, interaction diagrams and state and activity diagrams. At the end of this document, you will find the test planning (unit tests and integration tests).

The purpose of this document is to specify, visualize, modify, construct and document the artifacts of the Ray Tracing Engine.

Contents

1 Document conventions

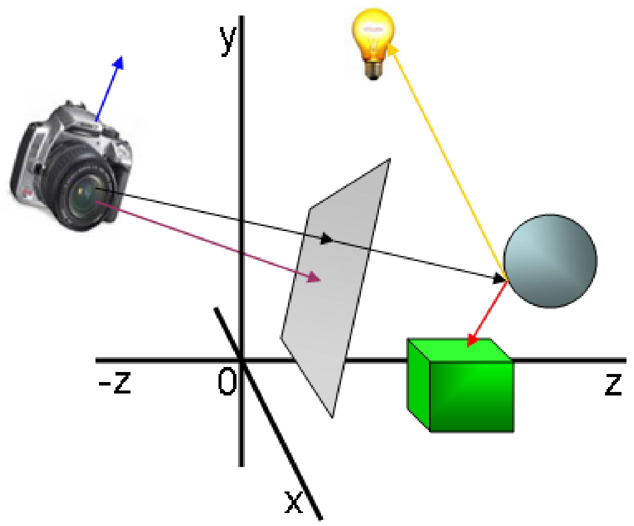

Figure 1: Ray tracing

- 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)

2 Class diagrams

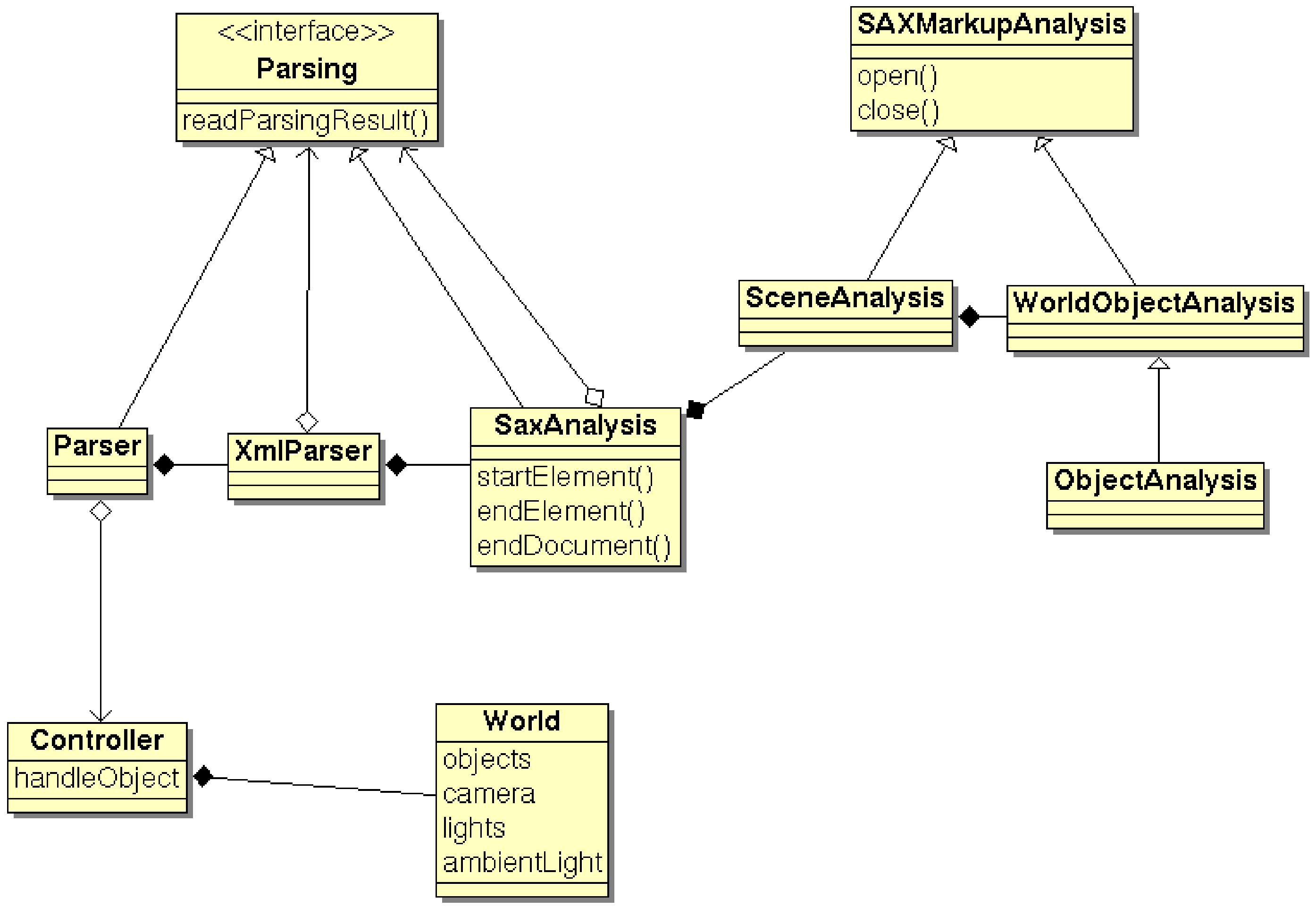

Figure 2: ray tracing parser

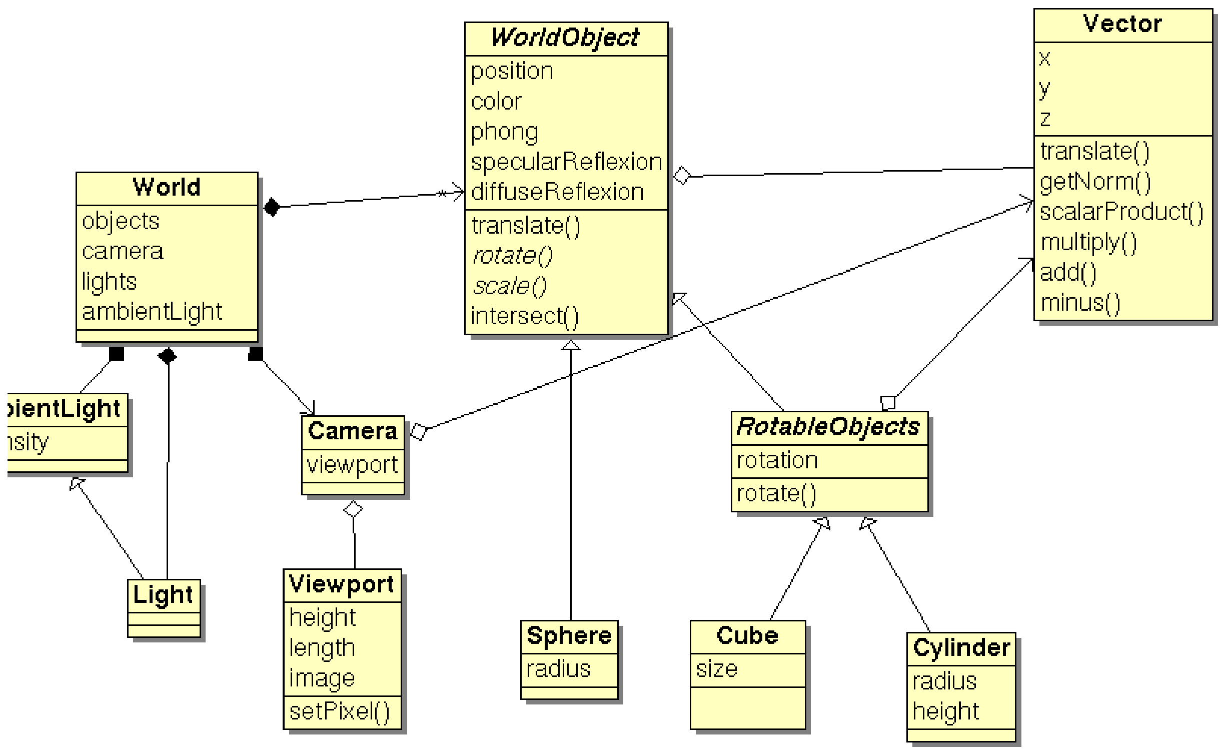

Figure 3: Ray tracing World Objects

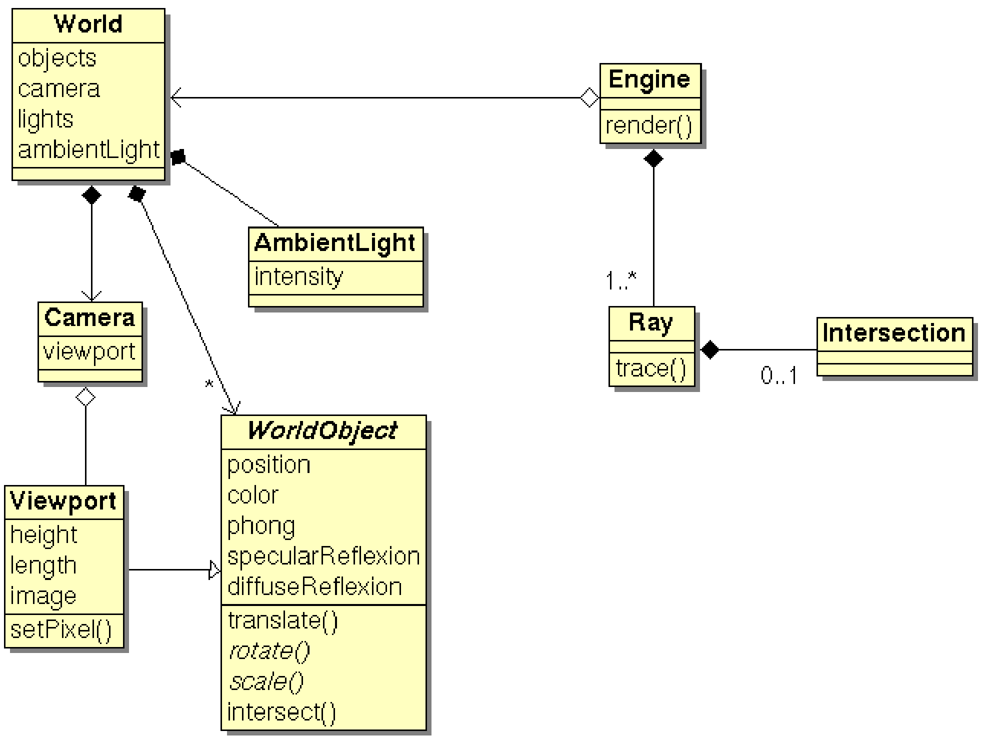

Figure 4: Ray tracing Engine

3 Class descriptions

3.1 Parser

This set of classes will be created to use efficiency the SAX parser.

– Controller : Receive all created object from parsing and add it to the World class.
– Parsing : interface, describe a unique method ”readParsingResult” to give back the newly created object from parsing to the parent class. Controller is the last called.
– Parser : prepare the xml parser.
– XMLParser : initialize SAX parser.
– SaxAnalysis : Sax Handler, analyse each markup sended by SAX and call the right SaxMarkupAnalysis class.
– SaxMarkypAnalysis : interface which describe a markup analyser
– *Analysis : one class for each markup, implementing SaxMarkupAnalysis. These classes create an object using given markup attributes.

3.2 World Objects

– Vector : 3D Vector (cartesian coordinates)
– WorldObject : abstract class use to describe a general object, this class is able to translate the center of the object. The intersect method return the point where a Ray inter
– RotableObject : abstract class, handle the possible rotation of object (derive from WorldObject).
– World : set of World Object with camera and ambient light and all other lights.
– AmbientLight : describe the ambient light, with RGB color.
– Light : derive from AmbientLight, these lights have a position.
– Sphere : describe a sphere by its center and its radius
– Cube : describe a cube by its position and its size
– Cylinder : describe a cylinder by its center, its radius and its height
– Camera : Position of the camera and it's direction. Create the Viewport for the future rendering

3.3 Engine

The class Engine will use the World to render an image.

– Intersection : describe an intersection between a Ray and a WorldObject with the distance from the origin of the ray and the position of the intersection, the normal to the surface and the WorldObject involved in this intersection.
– Ray : a ray which will be traced. Have an origin and a direction. The trace method will try to intersect with all the WorldObjects and return the closest intersection to the origin.
– Viewport : a 2D plan in front of the camera which represent the image. It contains a set of method to create the image. It can also create a dynamic visualisation of the current creation of the image.
– Renderer : The main algorithm of the Ray Tracer. It will cast a ray for each pixel of the Viewport and calculate the color of each pixel.

4 Use case diagrams

4.1 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.

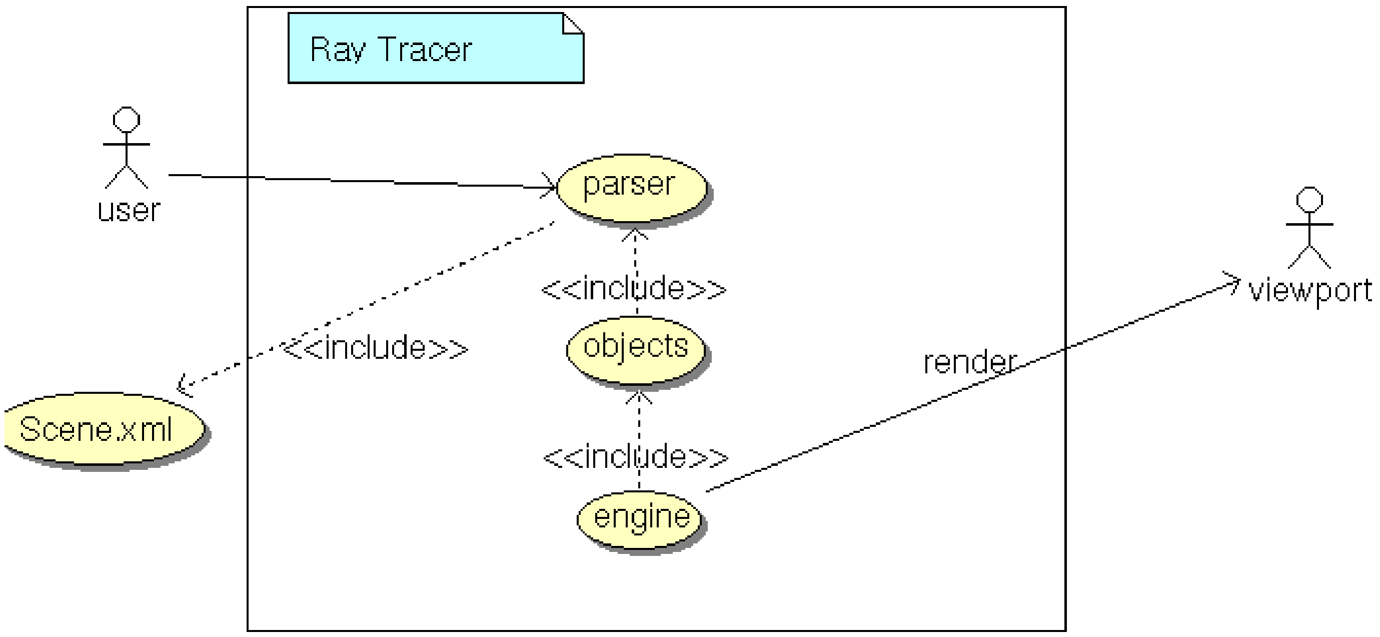

Figure 5: Use case general view

4.2 Objects

The parser create all objects of the scene.

The creation of the camera also create the viewport.

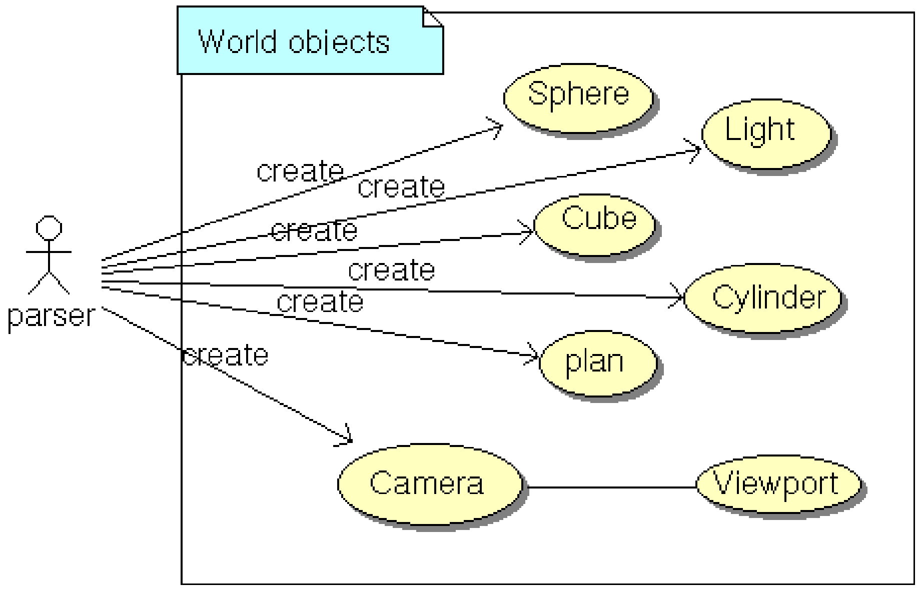

Figure 6: Use case objects creation

4.3 Engine

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

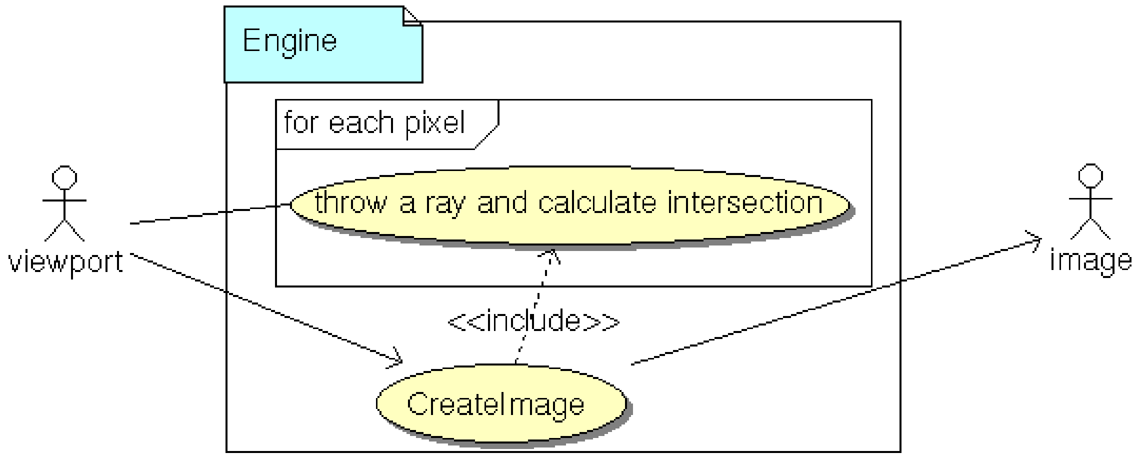

Figure 7: Use case engine

5 Interaction diagrams

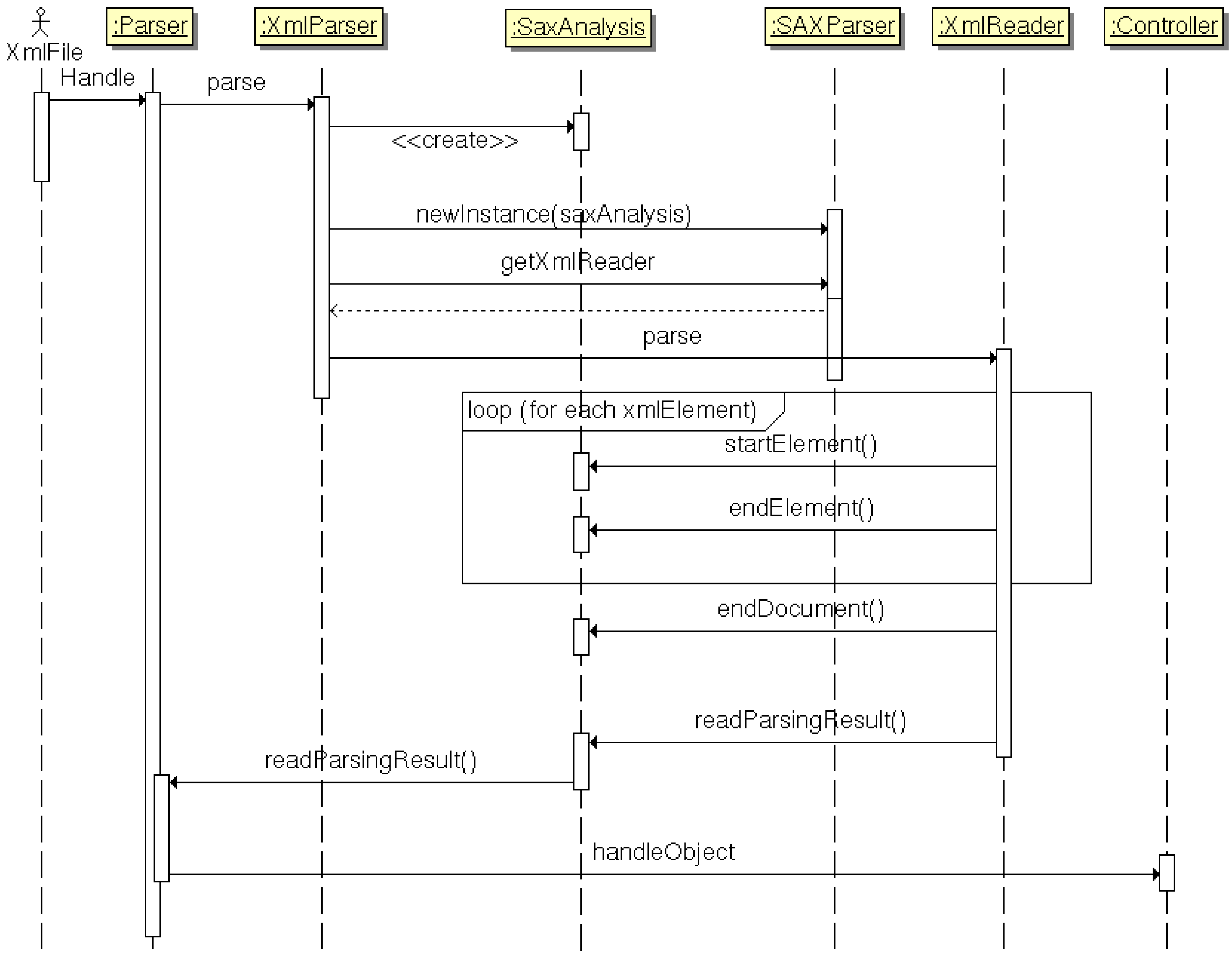

Figure 8: ray tracing parser

6 State and Activity diagrams

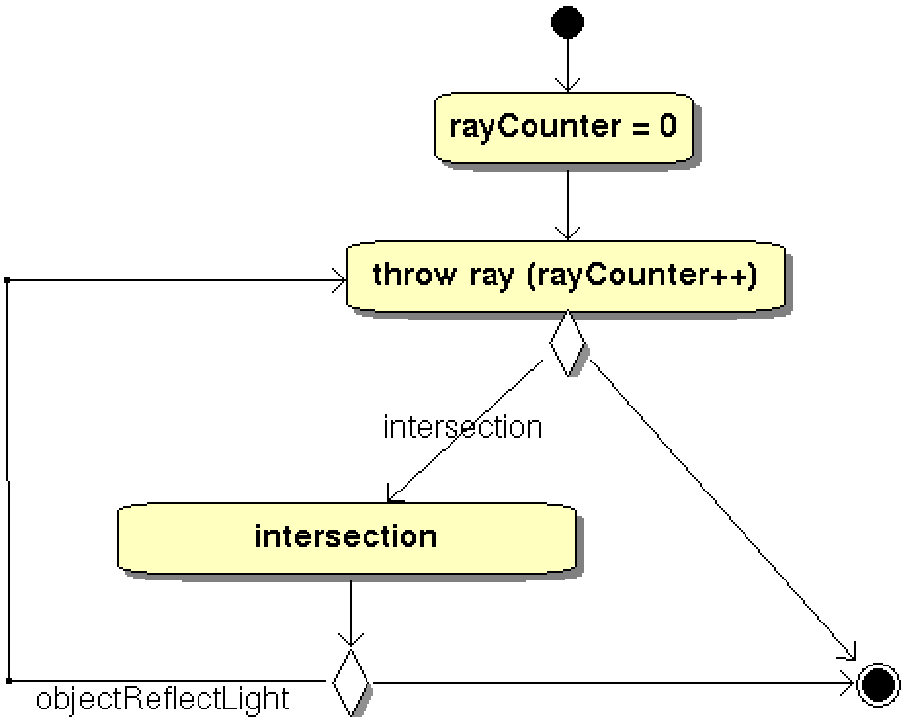

Figure 9: ray tracing parser

7 Test planning

7.1 Unit Test

Each WorldObject will be unit tested, mainly the intersect method (intersection of the object with a ray)

The ray tracing algorithm must be well tested before be really used. It cannot be really unit tested, mainly due to the fact that the result is an image, so it will be tested manually.

7.2 Integration test

The parser must be test independently and be tested mostly manually.

Finally we can test some render with the complete program.