CRT - Terms of Reference

Hamza Haiken (hamza.haiken@heig-vd.ch)
Prof. Pier Donini (pier.donini@heig-vd.ch)

B.A. Thesis, HEIG-VD 2014-2015

Contents

1 Project summary

1.1 Goal

Study, design and implement an artistic conception tool for creating realistic three-dimensional images, based on a physical simulation of light and its fundamental properties (diffusion, reflection and refraction), method known as *Ray Tracing*.

1.2 Context

Ray Tracing is a technique for computer optical calculations used for rendering an image or for optical studies. This technique involves simulating the inverse path light takes from the camera and compute the interactions of light rays with abstract primitive objects, composing a scene. This technique is used in the entertainment industry by Pixar and Dreamworks for animation, and by 3d artists for concept art.

1.3 Workflow

All work will be kept up to date on GitHub, on a dedicated repository[1]. The documentation and report will be updated on a weekly basis.

The main project will be created using Maven for easy to manage dependencies and automated building.

The documentation will be written using Markdown mixed in with some LaTeX, then converted to LaTeXusing Pandoc via templates. The documentation will include:

- This document
- The main report
- A development journal

Lastly, a small 7000 lines prototype of the core functionalities of this project was done last summer for experimenting and learning purposes. This prototype only deals with part of the rendering functionality and didn't include any of the rest (see below). Although some of the code will be refactored into the new project, most of the existing code will be totally corrected and rewritten.

[1] http://github.com/Tenchi2xh/CRT

2 Project specifications

The following subchapters will describe all planned steps and features.

2.1 Rendering

Base engine for rendering mathematical primitives: tracing rays, computing intersections, computing colors. Will be optimized for multi-threaded computations.

Extended rendering features will preferably include:

- Multiple light sources with inverse square law falloff
- Primitive objects mathematical modelisation (spheres, cuboids, cones, planes, half-planes, torus, etc.)
- Primitive objects can posess physical material properties: – Recursive partial and total reflection (Fresnel model) – Refraction (Snell-Descartes laws) – Procedural rugged surface (bump mapping, Perlin noise)
- Constructive Solid Geometry (CSG) with the primitive objects (union, difference and intersection) in order to design complex shapes
- Simulation of the physical properties of a camera: – Focal distance, field of view – Depth of field – Aperture shape for *bokeh* simulation
- Spherical and cylindrical projections of panorama pictures for scene backgrounds
- Super Sampling and partial preview of a render

2.2 Scripting language design

To control scene rendering, a user-friendly scripting language will be designed. This task includes:

- Designing user-friendly keywords and language constructs
- Designing a context-free EBNF grammar
- Generating a parser
- Writing a compiler

2.3 User interface

The user interface will aim for a professional look. Two mains views will be implemented: a code view, where the user can describe a scene by writing a script, and a live view, where the user can view a live, simple representation of the same scene, from multiple angles.

Linking the code view, live view and the saved files will be the fact that scenes will be serialized in XML.

2.3.1 Code view

The code view will include a rendering window and an editor, that should be visually and functionally close to any IDE editor, and include features such as line numbering, syntax highlighting, etc.

2.3.2 Live 3D view

The live 3D view will be written using OpenGL. The user will be able to drag & drop primitives in and out of the views.

2.3.3 Extensive properties

Both views will include the same side-panel, which will enable the user to extensively configure all aspects of the scene and final render, as well as view some relevant information:

- Camera settings
- Object properties
- Material properties
- Rendering settings
- View project informations
- View system informations and statistics

2.4 Project management and XML serialization

All views will export their scenes into XML: the format will have to be well-defined and validated with DTDs. All scene objects will implement an interface for this serialization.

Saved XML files will include the user scene, all cameras and project settings.

2.5 Basic animations

An animation mode will activate at the press of a button to display additional buttons and settings for rendering basic animations. In the code, a global variable representing time will be usable to make entity positions relative to time, and the animation mode will render the scene frame by frame, incrementing the time global.