Title: Real-time raytracer written functionally in Scala

student: Jerzy Redlarski
e-mail: 5xinef@gmail.com
skype: tennousei-jin_xinef
mobile phone: +48 600902567
timezone: UTC+2

mentor: Lalit Pant pant.lalit@gmail.com

Abstract:

The aim of the project is to create a graphics engine with a raytracing renderer. The project would be written mostly in scala and adhering to the functional programming paradigms. One of the goals is to make the rendering in real-time, so that it can be used in games, 3D visualization tools and CAD applications.

Functional programming is perfect for this kind of problems, since rendering as a whole is a functional problem (take a model and generate an image). Raytracing is also an embarrassingly parallel problem (scaling near linearly with the number of processors), which means that functional programming allows for easy multithreading. The mathematical nature of 3D computer graphics makes functional programming a natural choice for expressing many of the algorithms and data structures.

The project will be written with the Kojo Learning Environment in mind (http://www.kogics.net/kojo), serving as a 3D renderer for this project founded by Lalit Pant.

Detailed idea:

1. Innovation:
   While raytracing has been used for decades, it was mainly for off-line rendering, meaning that a single picture would take more than a few dozen milliseconds to render. What is more, so far few purely functional implementations of raytracing exist, and those that do are proofs of concept or otherwise simple examples, not fully developed graphic engines that would be necessary for useful applications.

   Nowadays, real-time raytracing is slowly becoming a possibility thanks to the increasing number of cores in CPUs, as well as graphics hardware that allows for programming floating point calculations outside of the rasterization pipeline. This means that finally it is possible to use GPUs to achieve hardware acceleration of raytracing. It is thus possible to implement most of a raytracer in a high level language like Scala and only implement the bottlenecks using CUDA and/or OpenCL, and still achieve great speedups. The great support for concurrency in Scala is also important when implementing parallel calculations to make full use of multiple cores in a real-time application.

   It is also worth noting that scenes that need rendering are becoming more and more complex every year. Raytracing scales better with the increasing complexity of a scene than rasterization does. This means that while for a simple scene rasterization might be faster, for a sufficiently complex scene raytracing will always be faster.

2. Contribution to the Scala community:
   While it is possible to use Java/C# graphic libraries in Scala to render 3D scenes, those libraries are written using imperative programming and techniques that are best avoided in Scala, such as using nulls instead of Options, using exceptions for problems that could be solved otherwise etc. Implementing a graphic library in Scala would solve those inconveniences by allowing Scala users to code in an idiomatic way.

   It would also encourage more people to write graphic intensive applications, such as games and CAD software, in Scala, as well as making it easier to define geometric shapes the way we think of them, since users will not be limited to points, vertices and triangles. Creating a graphic engine in Scala would also create a codebase that other graphic software written in Scala could reuse.

   Adding a 3D renderer to the Kojo Environment will also benefit the community, as the Kojo project aims to encourage people (especially kids) to learn programming and math through interactivity, creativity, art, and games, and switching from two dimensions to three adds both many new possibilities and improves the appeal of the end results to the users.

3. Why is it needed?
   Raytracing is slowly becoming a viable alternative to rasterization, while offering many benefits in terms of quality. It better reflects real life phenomena such as reflections, refractions and shadows. It can easily render perfect spheres, fractals, as well as other complex shapes, while rasterization can only draw triangles. It offers great scalability. It's only a matter of time until raytracing (and other methods like photon mapping) outclass rasterization in real-time applications.

   The fact that major companies such as Intel (http://www.qwrt.de/), IBM (http://www-03.ibm.com/press/us/en/pressrelease/22258.wss) and Nvidia (http://www.geeks3d.com/20090814/cuda-real-time-ray-tracing-tutorial/) already made successful experimental implementations proves, that sufficiently efficient hardware is already present and with time will reach home users. It would be beneficial if a real-time raytracer written in Scala was already present at that time, rather than being implemented post factum, when people looking for real-time raytracing solutions already chose other technologies that were implemented earlier.

   From a Kojo point of view, a 3D rendering engine is needed to enrich the learning experience. Choosing a raytracer over rasterization allows for easier visualization of shapes described by mathematical formulas.

4. Plan:

What is already done:

– I have implemented simple raytracers based on tutorials such as:
  http://www.codermind.com/articles/Raytracer-in-C++-Part-I-First-rays.html
  http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm

– I have set up a github repository at https://github.com/Xinef/Xapphire
– an example „Hello raytraced world!" Scala script is in the repository. I can achieve around 8 FPS on a 3 GHz processor.
– I gathered a number of examples and links to resources that could prove useful as a reference, such as an Open Source raytracer written in Java http://sunflow.sourceforge.net/

April 23 – May 21

- Setting up the working environment
- Establishing contact with the mentor
- Doing further reasearch on raytracing and on the Kojo Environment
- Finding people who can offer essential help

May 21 – June 4

- Creating the basic skeleton of the raytracer
- Creating a data model of 3D objects
- Adding support for textures and different types of light sources

June 4 – June 18

- Implementing a choice of different shading algorithms
- Integrating importers from popular 3D model formats such as .md3 and wavefront .obj
- Creating an API that allows the creation of a 3D model from the Kojo Environment

June 18 – July 2

- Creating the acceleration structures (octrees, k-d trees)
- Testing the effectiveness of the acceleration structures and finding when to use which structure for best effect

July 2 – July 16

- Milestone – a working raytracer that can display a complex scene, doesn't have to be real-time yet.
- Time for bugfixes, finishing unfinished tasks
- Mid-term evaluation

July 16 – July 30

- Optimizations:
    - multithreading
    - hardware acceleration with CUDA/OpenCL through Java Native Interface
    - improving the algorithms
    - using stack allocation where appropriate with JStackAlloc

July 30 – August 13

- Further optimizations
- Improvements of the API to make usage of the engine in the Kojo Environment easy and intuitive (possibly developing a DSL for that purpose)
- Writing the user manual

August 13 – August 20

- Finishing the documentation
- Testing performance
- Bugfixing

August 20 ->

- Further development after GSoC ends

5. Why should you choose me for the task?

I am a student of Gdansk University of Technology, first year of master degree studies in Computer Science. I've chosen the Department of Intelligent Interactive Systems as my primary specialisation, since I am interested in advanced computer graphics programming and also the development of new graphical technologies. I'm also a member of two scientific groups: Vertex – dedicated to computer graphics and game development, and JUGGUT – the Java User Group of Gdansk University of Technology.

I've been experimenting with raytracing and OpenGL for approximately three years, and I've been learning Scala and functional programming for about a year. My bachelor project was a 3D rigid body physics engine written in Java, where I used immutable data structures to gain some of the benefits of functional programming, while the rest of the engine was written in an imperative manner. I was also responsible for writing an example 3D application in Java3D that showcased the functionality of the physics engine.

I am willing to support the Scala Community because in my opinion Scala solves many of the problems that make programming graphics engines in Java tiresome, while benefiting from the advantages of JVM. I find both functional programming and the way Scala supports easy parallelization of operations to be very helpful when working with graphics. You can be sure that I will be dedicated and will work with passion, since this subject is both something I love doing, and it is something I intend to further develop and use as a basis of an open source game engine written in Scala.

My CV is available at http://www.mediafire.com/?p6a2qg7igstdy1f