# PG612 - Mandatory Assignment 3: Ray-Tracing

In this assignment, you will learn a way of creating a 2D image from a 3D scene, ray-tracing, and you will also learn how some OpenGL-techniques work «behind-the-scenes».

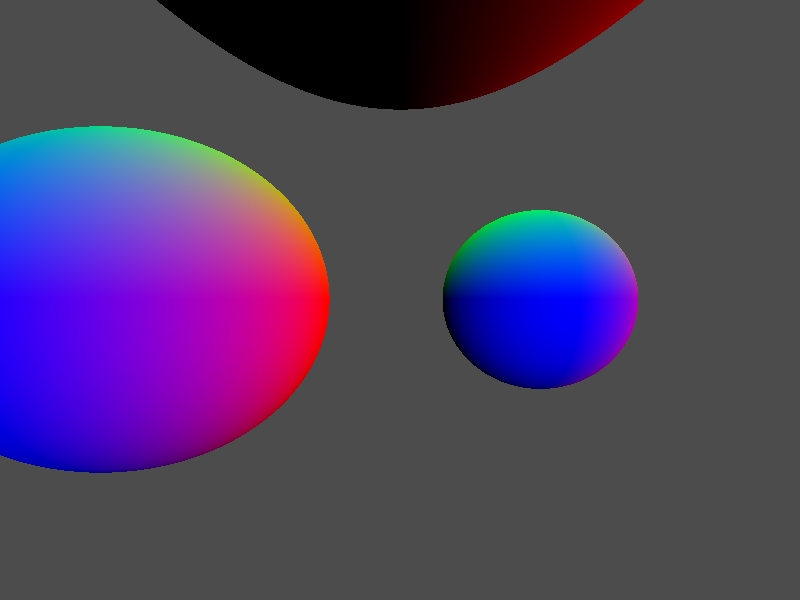
## Important information:

* **Read all parts of this text, and do as described.**
* **All subtasks are possible to implement independently. Do not get stuck on a single subtask. It is better to get 90% right of all than 100% right on one.**
* **You have to write all of the code yourself. Base your solution on the lecture notes and books in the course.**

Subtask 1: Implement ray traced reflective and Fresnel spheres

Implement a ray-tracer that renders a scene consisting of (a) Fresnel and (b) Reflective spheres based on the skeleton code.

* See lecture 9 for details on how to implement  
  Hint 1: Doing lab 9 will be a step-by-step guide to help you get vectors and tracing right  
  Hint 2: Debug your normals to check if they are correct (should be similar to image below)!

Subtask 2: Implement a ray traced cube map

Implement a cube map as a scene object based on the CubeMap class.

* See lecture 9 for details on how to implement the CubeMap::rayTrace(...) function.
* Find the four pixels nearest the texture coordinate, and perform bilinear interpolation between them in the function CubeMap::readTexture(...).

Subtask 3: Maximum ray traversal depth  
**This subtask requires that subtask 1 is implemented properly.**  
Rewrite so that rays do not have a maximum traversal depth of 8, but will terminate when their contribution to the screen pixel color is «small» (e.g., when they only contribute with 2% of the output color). Render a scene with at least two reflective spheres: what is the difference to the maximum depth approach?

* Remember that this also requires that you cannot have 100% reflective objects in your scene: each object must absorb a minimum amount of light when hit.

### Subtask 4: Multi-sampling

Implement 2x2 multi-sampling in your ray-tracer by shooting four rays for each pixel location and averaging the results

* See lecture 9 for details on how to implement

Subtask 5: Ray trace a triangle  
Implement a new scene object, Triangle, that performs ray-triangle intersection test.

* See lecture 10 for details on how to implement, and shade the triangle object using the reflection effect from subtask 1.

## Requirements

* Code must extend the skeleton from itsl
* Code must compile and run out-of-the-box using
  + Visual Studio 2010, SDL, GLM, Assimp, OpenGL 3.3+
* Short (5-25 lines, 80 columns) text (.txt) README-file
* Doxygen compliant source code comments (javadoc)
* No temporary files (Visual Studio, svn, etc.).

## Grading

The following criteria are used for grading this assignment:

* 40% Doing the assignment. For example:
  + Have you completed all subtasks?
  + Have you written every line of code yourself?
* 30% Code quality. For example:
  + Is your solution correct, simple, and elegant?
  + OpenGL efficiency,
  + Use of deprecated OpenGL functionality,
  + Useful comments,
  + Compilation errors, warnings, etc.
* 20% Visual quality and natural feel. For example:
  + Does your solution “feel” natural, does it look “right”, etc.
* 10% Overall rating. For example:
  + Anything I feel is not covered by the above points which deserves extra credit