Duché Guilhem

4003-571

Personal project report

Further with the raytracer

I) Project description :

The goal of my personal project was to extend the possibilities of the basic ray tracer we had to make during the quarter. Basically i wanted to implement some techniques we had talked about during the classes and try some personal ideas i had. I wanted to had more complexity to the scenes that our ray tracer was able to render and finally try to use the ray tracer to make 3D animations. I worked in the following fields : Lights, Mappings, complex objects, calculations speedup and animations.

II) Approach :

My ray tracer is a JAVA application which renders scenes. The scenes are hard coded in one of the JAVA files : Begin.java in the GUI package. The scenes defined in this file are the inputs of the program. When you launch the program, you can set up different options, choose the scene to render and the ray tracer will computes images according to what has been selected and programmed. All the images are stored on your hard drive.

The algorithms, main techniques and main features used in this ray tracer are :

- recursive raytracing algorithm,
- Phong illumination model,
- Phong-Blinn illumination model,
- Moller & Trumbore ray-triangle intersection algorithm,
- Bounding boxes,
- Texture mappings, alpha mappings, bump mapping,
- HeightField,
- Off Object,
- ...

III) User's guide:

User's guide

CGII ray tracer

Program description :

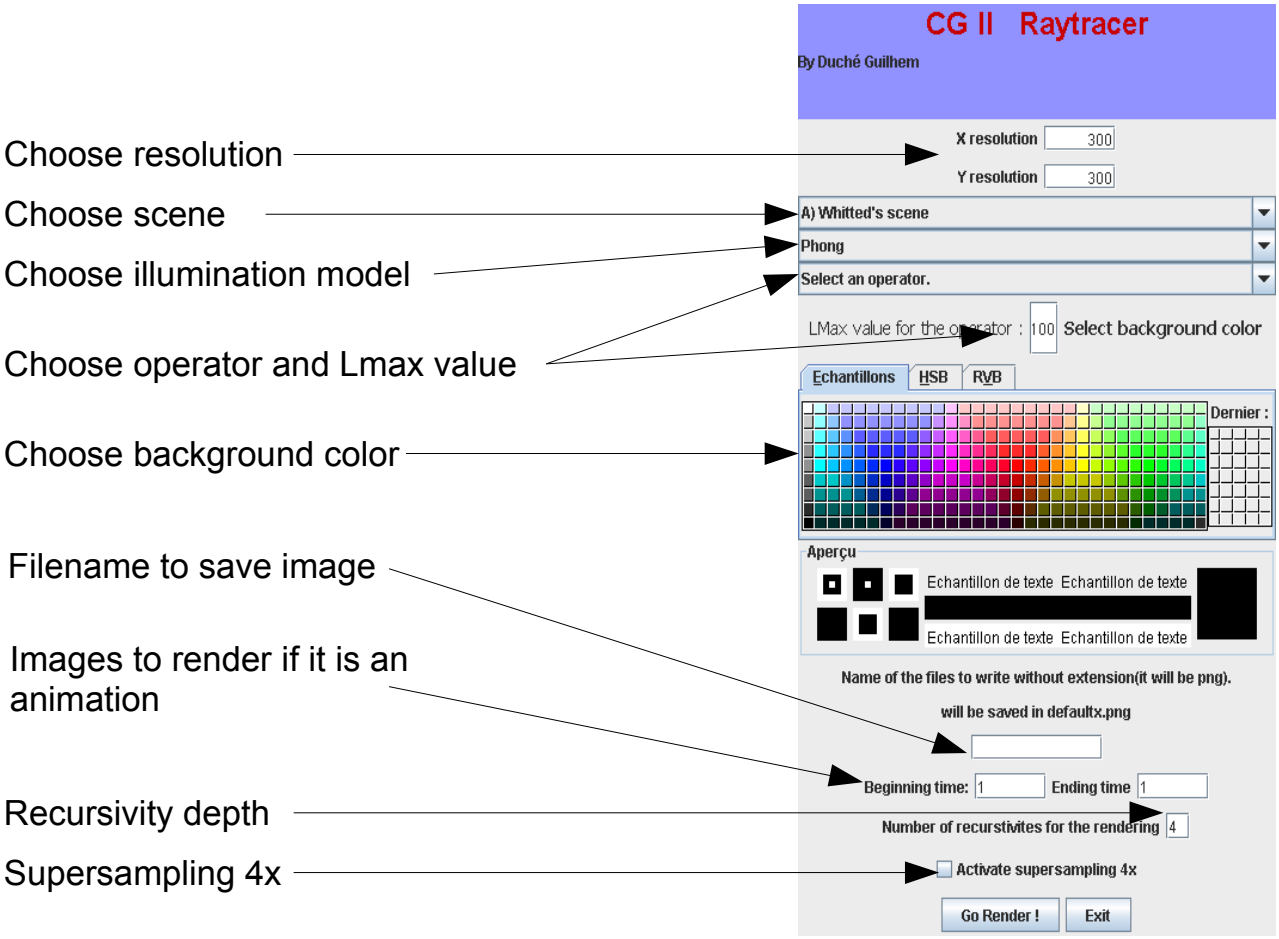
This software is a ray tracer written in JAVA. It permits the user to render 3D images and 3D animations programmed in JAVA. The kind of objects we can render are : sphere, polygons, boxes, off objects, height fields. The scene is enlightened with lights that can be : ambient light, point lights, sport lights and diffuse light. Knowledge of JAVA is required to make you own scenes.

Input :

The inputs the program uses to calculate images are scenes hard coded in a JAVA file (Begin.java).

The best way to learn how to define your own scenes is to watch the examples provided with the ray tracer and use the JAVADOC. The examples are in Begin.java and there is no need to give many details on how to use it since it's pretty obvious. Usually the JAVADOC will have the answers for your question.

The other input is the configuration of the renderer using the GUI. Here are the details of this GUI.



Normal output :

The normal output of the program is a displayed image and one or several images saved on the disk. If no name have been defined the images are saved with name default*.png otherwise it is saved with the name given (name*.png). The number of images saved is equal to Ending time – Beginning time + 1.

Exception reports :

Different kind of exceptions are possible :

- The objects created using polygons can throw the following exceptions :
  - NotEnoughtPointsInPolygonException if there isn't enough points to define a polygon.
  - PointNotInPolygonException if one of the given point is not in the plane defined by the first 3 points
- The use of camera in an animation can generate two exceptions :
  - if you try to add to an animation a camera which is in conflict with a previous camera a CameraConflictException exception is thrown.
  - if during the rendering of the animation there is no camera for a certain time a NoCameraException exception is thrown and the program will crash.
- A transformation can also throw a ObjectNotSupportedException exception if we try to apply the transformation to an object not supported by the transformations.
- Classical JAVA exceptions can also be thrown like NullPointerException exception if you define your scene badly. You will also have errors if you use some files in your scenes that are not present...

Program limitations and bugs:

The program has the limitations inherent to JAVA. Some bugs are also known :

- -If the eye point of the camera has a coordinate of type 0,X,0 an exception will be thrown because the matrix generated with the eye point can't be inversed in this case.
- The path of ray in mixed translucent objects is not accurately computed.
- A weird bug appears sometimes with the specular color of a sphere.
- Animations with smoothed objects don't work.

The user is free to extend from the given classes and add his own functionalities to the ray tracer.

Command sequence and installation :

There is no particular command to launch the program. The main method is in the Begin.java. The program is given as a Netbeans (http://www.netbeans.org/) project. The package vecmath from JAVA3D needs to be installed on the system and added to the project. The software only uses the JAVA framework and the vecmath package so it should run on every system with JAVA 1.5 installed and JAVA3D set up in the CLASSPATH variable. It should also be easy to compile and run the project outside Netbeans even if it hadn't been tried yet.

The given version also requires some images and textures that are provided in a compressed file with the ray tracer.

Author information :

Name : Duché Guilhem
Address : 2, chemin de la pouparderie 35400 Saint-Malo France
Assistance & contact : guilhem.duche@gmail.com

IV) Technical documentation :

Technical documentation

CGII ray tracer

Program description :

This software is a ray tracer written in JAVA. It permits the user to render 3D images and 3D animations programmed in JAVA. The kind of objects we can render are : sphere, polygons, boxes, off objects, height fields. The scene is enlightened with lights that can be : ambient light, point lights, sport lights and diffuse light. Knowledge of JAVA is required to make you own scenes.

Historical development of program and current status :

This program has been developed throughout the winter quarter 2005-2006. The development strictly followed the checkpoints planned by the CG2 course. In parallel to the normal development of the basic ray tracer a lot of features were added.

Description of each package and overall design structure :

An overall view of the packages and inheritances is provided with this report. Compositions are not displayed in the overall view. The JAVADOC can also be useful to understand the role of each class.

Raytracer :

Contains the kernel of the ray tracer. The class Raytracer.java is the actual algorithm of the raytracing.

The class RaytracerObject is the mother class of any objects added to a scene (camera, light, RaytracedObject).

The class Scene is the class used to store all informations about the scene. It also provides the informations needed when you need to launch a ray in the scene.

The class Camera is the class used to represent any camera. Associated to the camera we also have two exceptions that can be launched if two cameras are in conflict in an animation or if there is no camera at a given time in the animation to render it.

GUI :

Begin.java is the GUI of the ray tracer. IT also contains the definitions of the scene. To add your own scene we advise you to make your own method "*private Scene getScene*()"* and to add these lines : "*Scene s* = getScene*();Scenes.put(s*.getName(),s*);"* in the *"Begin()"* method at the beginning of the file.

RenderingWindow.java is the window where is displayed the result. It is also here that we do the loop to make all images for an animation.

Lights :

One of the most simple package which contains classes for the lights we can add in a scene.

RaytracedObjects :

This packages is the most important one. It contains all the objects we can actually display in a scene like cubes, polygons ...

We find also find the exceptions launched by the polygon if there are not enough points to define a polygon or if one of the points is not in the plane defined by the first 3 points of a polygon.

IntersectionResult is a very important class. Each time a ray is launched in the scene if it touches an object an instance of IntersectionResult is returned otherwise null is returned. IntersectionResult contains the normal at the intersection, a link to the object touched, the material of the object touched, the distance to the intersection and the point of intersection.

Materials :

Simple package which stores classes relating to the object appearance.

Transformations :

Package containing the transformations we can apply to the objects of the scene between two images of an animation.

Mapping :

Package containing the different mappings we can apply on certain objects (TextureMapping, AlphaMapping, BumpMapping, ReflectionMapping). Currently the different mappings can be applied only on rectangles, spheres and instances of GeneralCube class.

Utils :

Various utilities for the ray tracer. One class deals with images and the other permits to create bounding objects.

Description of key classes :

The most important classes are the following one :

- Raytracer.java : is the class with the algorithm of raytracing. It also contains the illumination models used and the tone reproduction operators.
- Scene.java : this is the class that defines a scene/animation. It stores all the cameras used during the animations, all the lights that light the scene and all the RaytracedObject objects in the scene.
- IntersectionResults : result of an intersection between a ray and a RaytracedObject. It contains the normal at the intersection point, the distance to the intersection point and the materials of the object touched and the object touched. If nothing is touched in the scene null is returned.
- RaytracerObject : mother class of every object stored in an instance of the Scene class. This class permits to give date of birth and death of the object. It also contains a list of transformations associated to the object and that should be applied at certain

time during the animation.

Global program behavior :

When the “Go render !“ button from the GUI is clicked an in instance of the Raytracer class with all the parameters setup in the GUI and the scene selected. Calling the draw method in the instance of Raytracer class draws the image of the scene. Calling update make the objects move once in the scene.

A scene is composed of Lights, Cameras and RaytracedObject objects. Each object stores the story of its life : when it lives, what changes are operated on itself and when to apply each change. The time unit is the number of the image rendered. When we draw the scene, the scene launches rays only on objects that are alive at the number of the image. The lights used are only the lights alive too. For each image only one camera can be used to render the image, if there is no camera an exception is thrown.

Command sequence and installation :

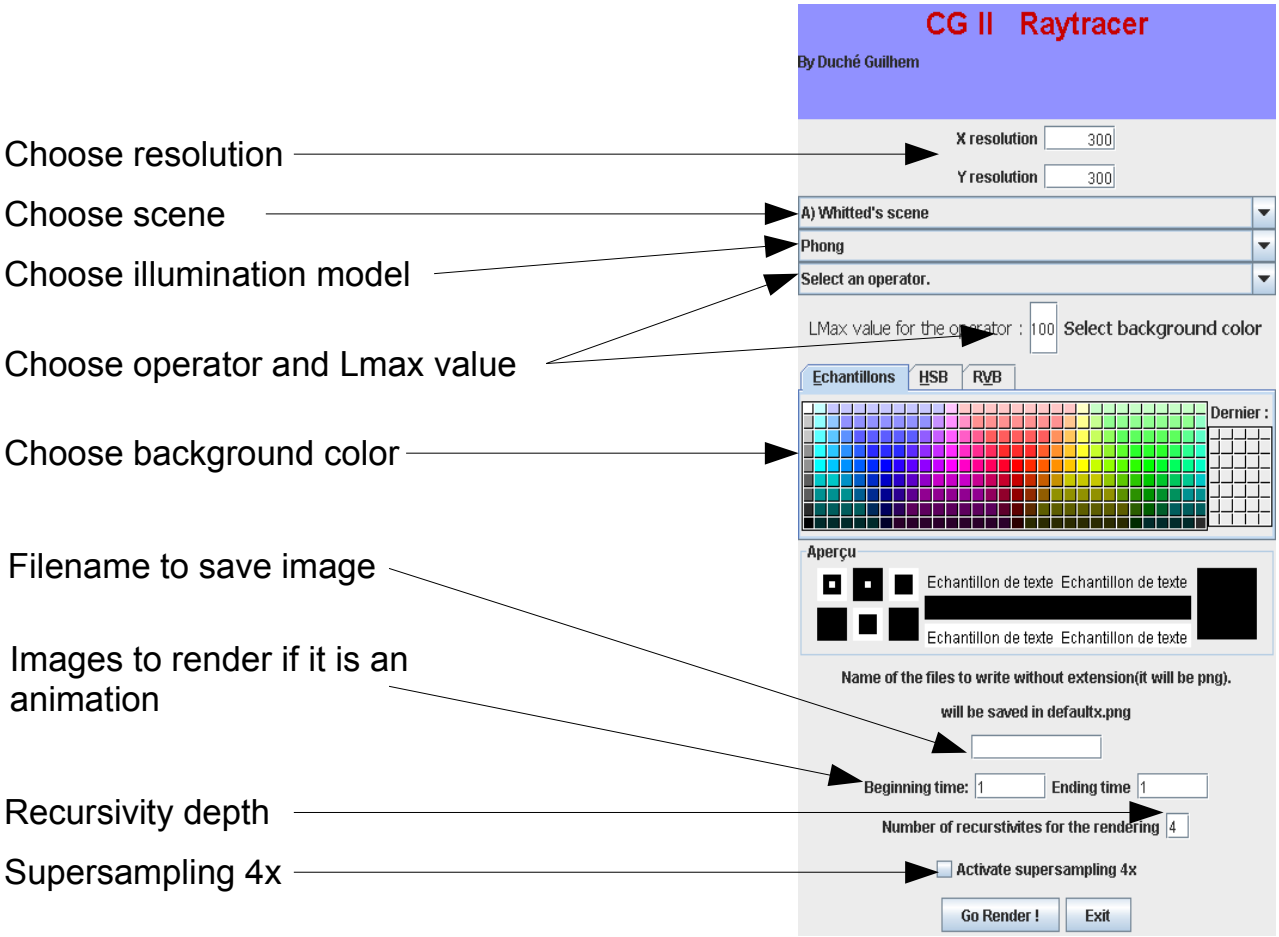
There is no particular command to launch the program. The main method is in the Begin.java. The program is given as a Netbeans (http://www.netbeans.org/) project. The package vecmath from JAVA3D needs to be installed on the system and added to the project. The software only uses the JAVA framework and the vecmath package so it should run on every system with JAVA 1.5 installed and JAVA3D set up in the CLASSPATH variable. It should also be easy to compile and run the project outside Netbeans even if it hadn't been tried yet.

The given version also requires some images and textures that are provided in a compressed file with the ray tracer.

V) Results obtained from the raytracer:

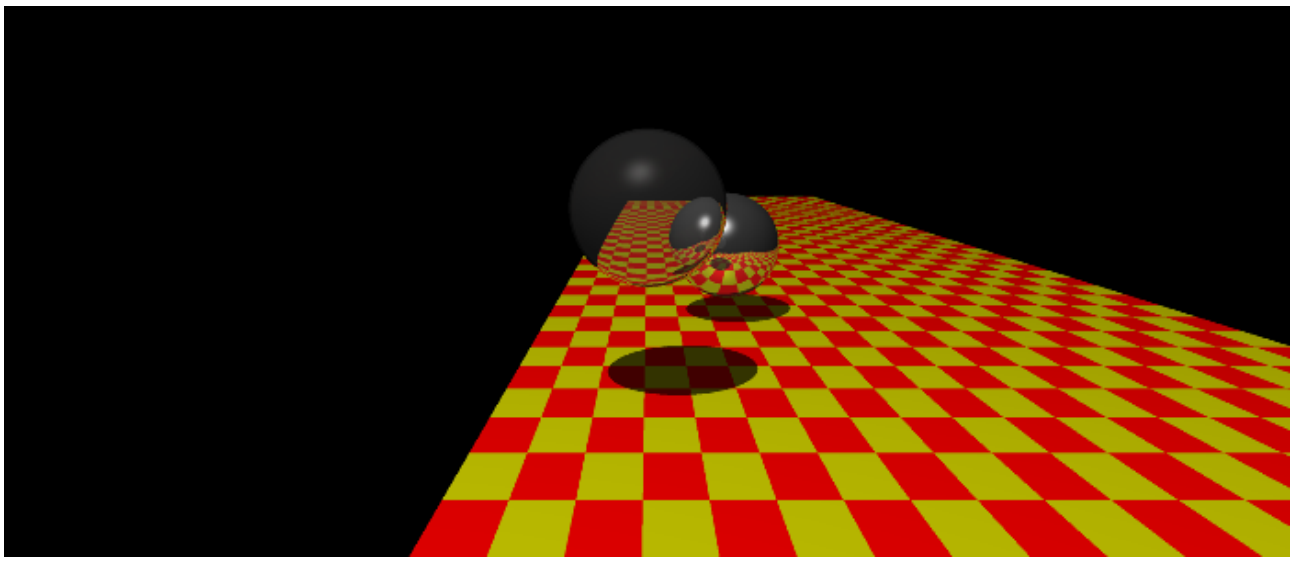
GUI :

The GUI permits to set up the rendering and launch it.



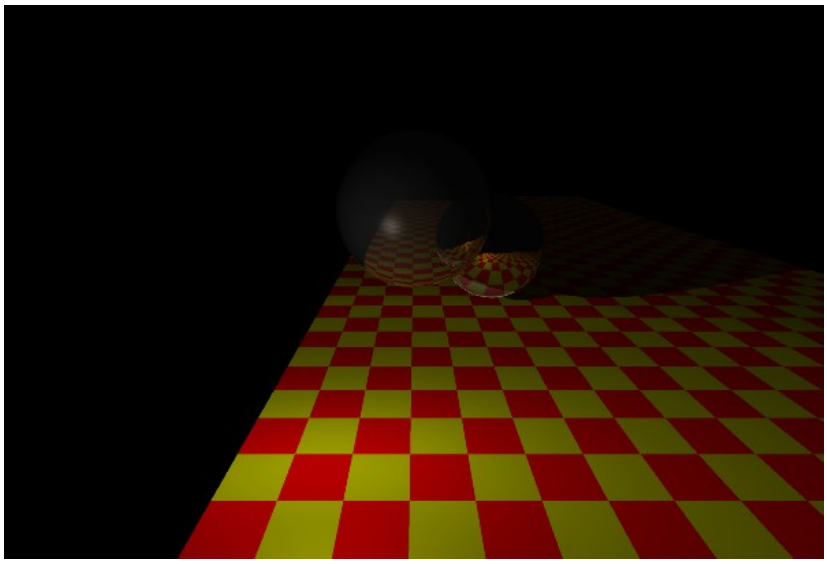
Basic Results :

Here is the result with all the features we had to implement for the basic ray tracer made for the CG2 course.

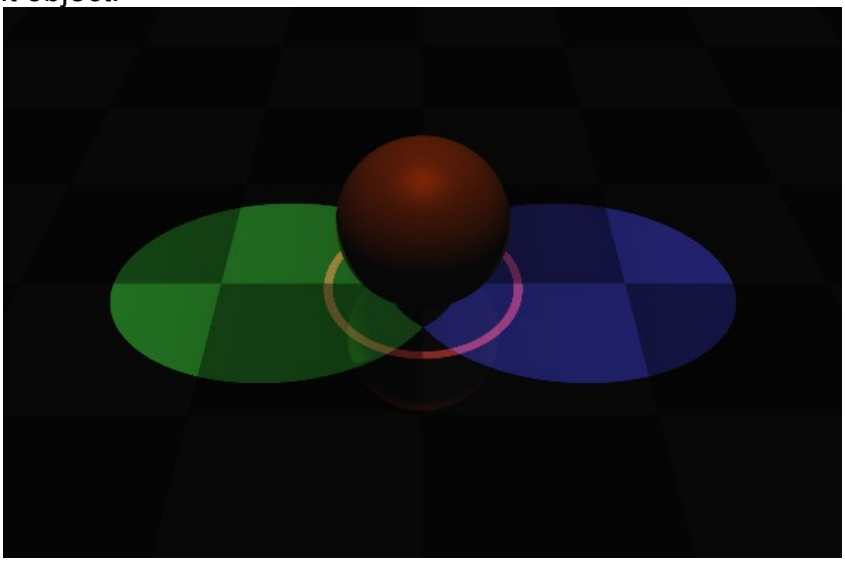


Results with new lights :

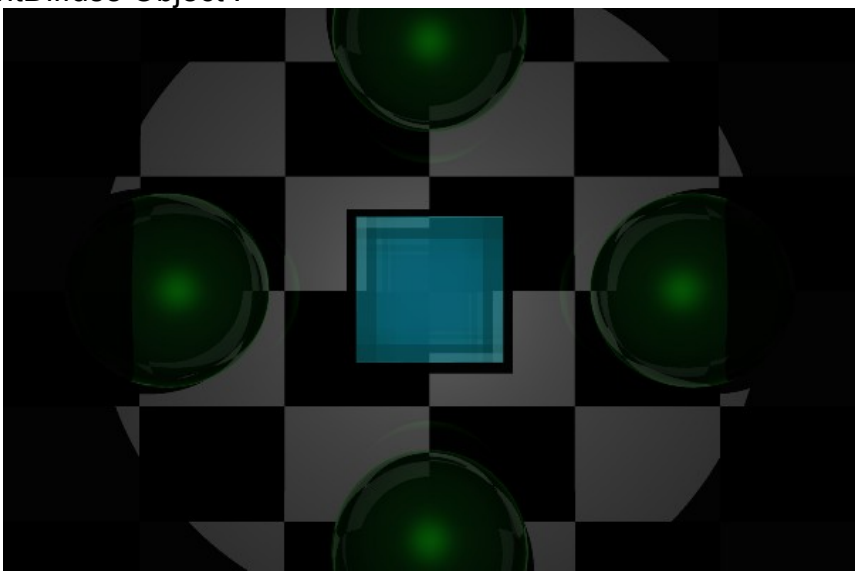
Use of DiffuseLight object:



Use of SpotLight object:



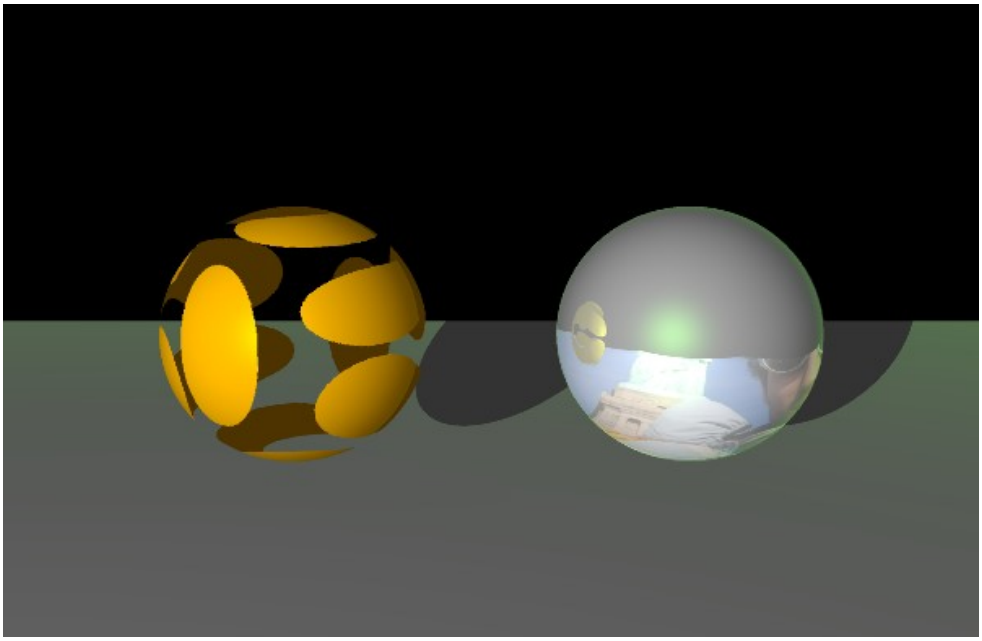
Use of SpotLightDiffuse Object :



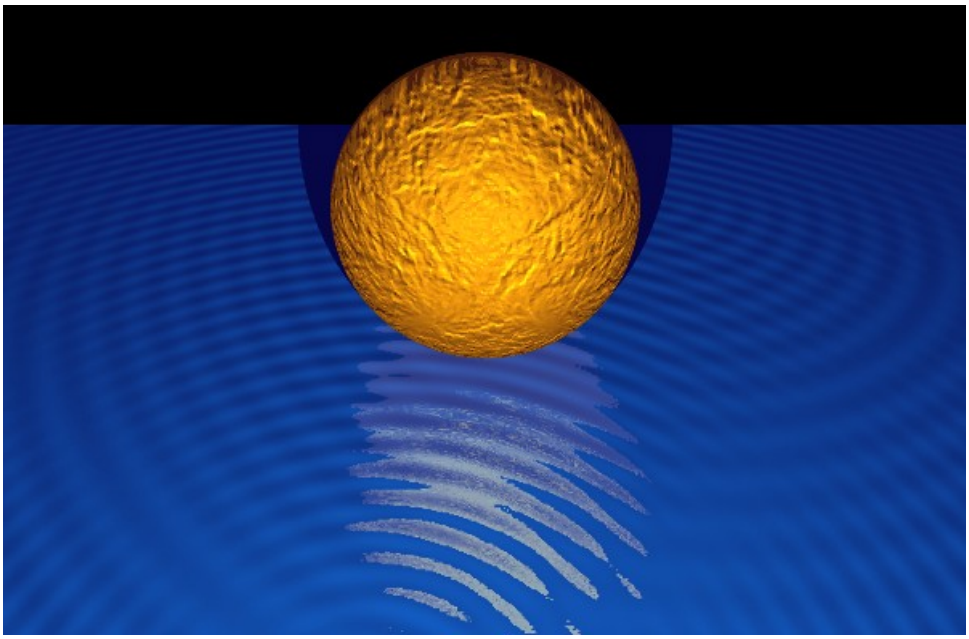
Results with mappings :

TextureMapping object :

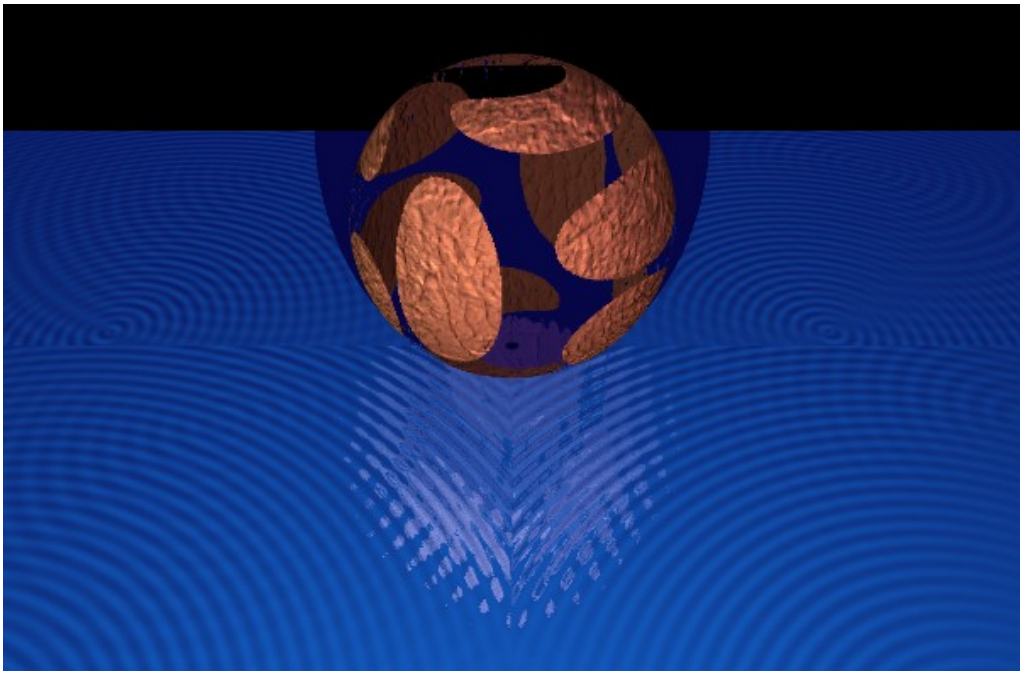
AlphaMapping object on the left and ReflectionMapping object on the right :



BumpMapping object :

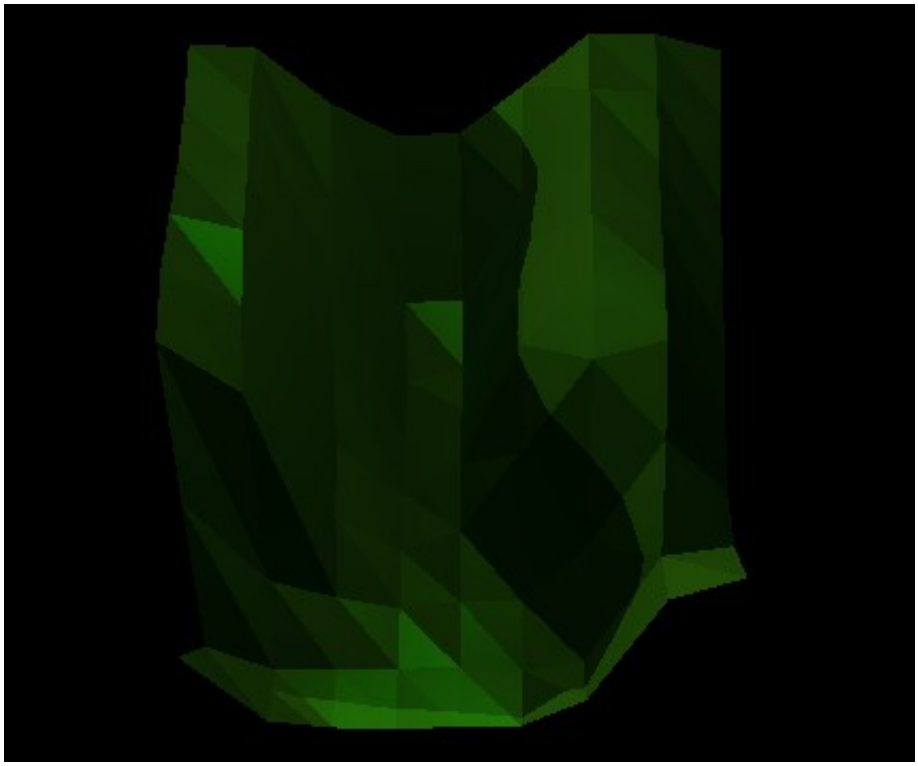


Mix several types of mappings and repeat a texture :

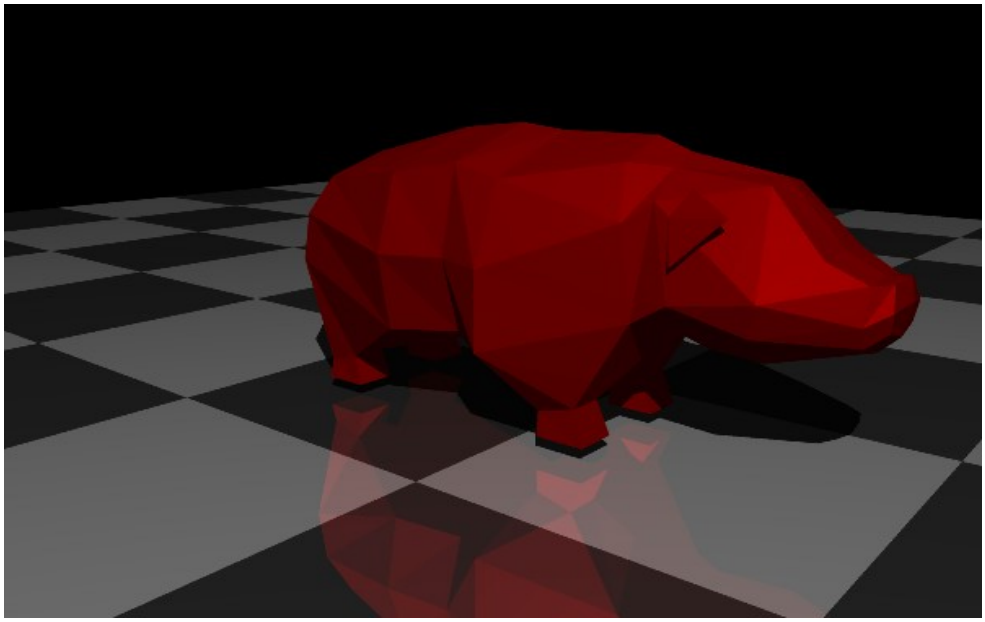


Results with complex objects :

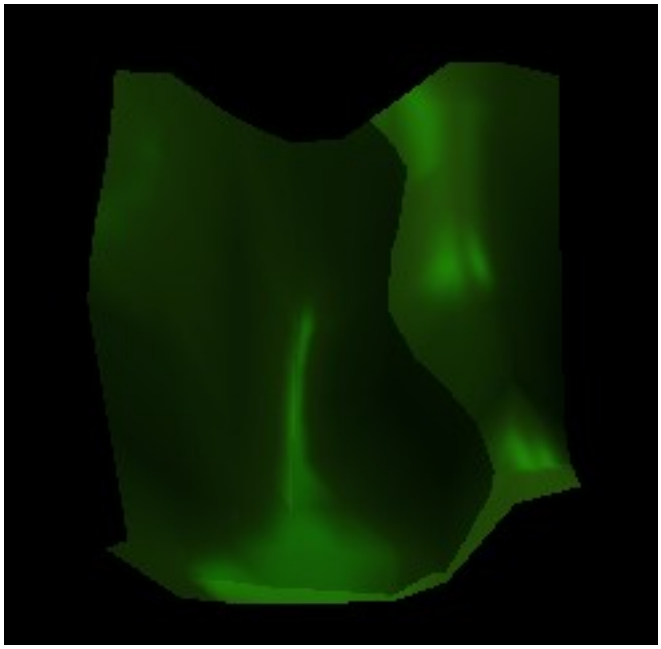
HeightField object :



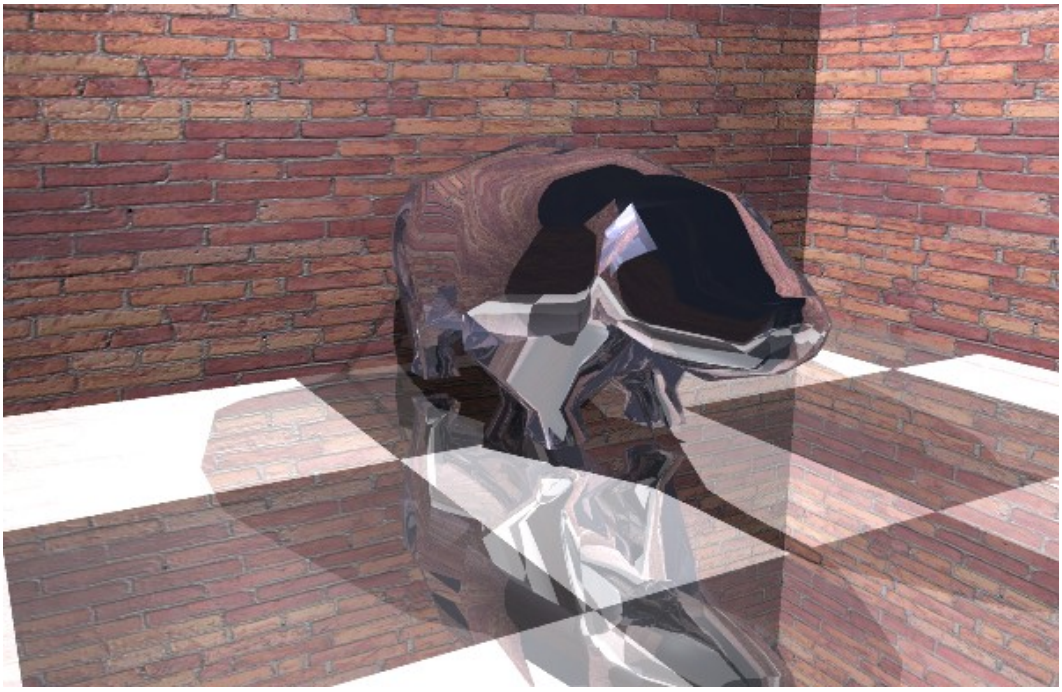
OffObject object :



SmoothHeightField object :



SmoothOffObject object :

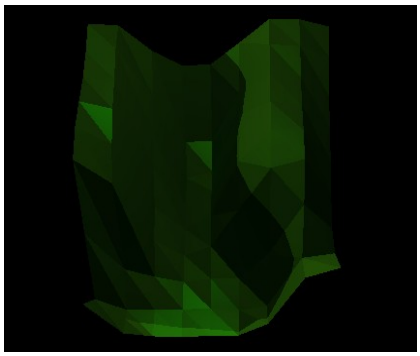


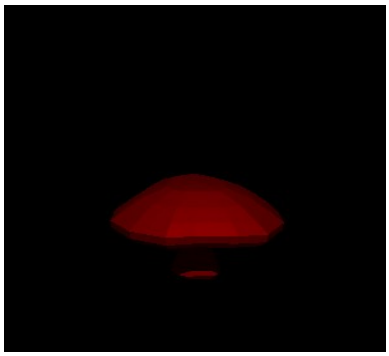
Some benchmarks :

Here are some results of the speedup improvements obtained using bounding objects and a new ray-triangle intersection algorithm.

Heightfield

128 triangles
2 lights





Mushroom.off

240 faces
448 triangles
1 light

	1st algorithm	Box	Sphere	2nd algorithm	Box	Sphere
HeightField	**55,5**	**25,6**	**30,7**	**6,8**	**3,8**	**4,3**
Mushroom	**107,2**	**32,9**	**25,0**	**15,6**	**5,2**	**4,2**

Animations :

An example of animation can be download in from this URL : http://www.easy-dj.com/cg2/grosgros.avi.

This animation can also be computed with the ray tracer. The name is “Animation final 1”.

VI) Future enhancements :

A lot of enhancements could be added. Some simple concepts could be implemented simply extending from well chosen classes. For example new transformations could be implemented extending from the class Transformation and modify any property of each object. A transformation i will certainly add soon will be a transformation to modify the color of the objects.

Some new objects could be also implemented extending from any class of the RaytracedObjects package and implementing the required functions.

Since everything is object oriented it would be interesting to develop an xml format to load and store scenes.

I'm not sure yet of the features i will add since I'm going on a coop. I will probably try to play with this ray tracer a bit more on my free time to make nice pictures or animations.

VII) Appendix :

Some references used for this project :

Off file format :

- http://shape.cs.princeton.edu/benchmark/documentation/off_format.html
- http://www.csit.fsu.edu/~burkardt/data/off/off.html

Mappings :

- http://en.wikipedia.org/wiki/Texture_mapping
- http://gamefreaks.net/defrag/textures.html
- http://www.mvps.org/directx/articles/spheremap.htm
- http://en.wikipedia.org/wiki/Normal_mapping
- http://en.wikipedia.org/wiki/Displacement_mapping
- http://web.cs.wpi.edu/~matt/courses/cs563/talks/bump/bumpmap.html
- http://www.gamedev.net/reference/articles/article1903.asp
- http://freespace.virgin.net/hugo.elias/graphics/x_polybm.htm
- http://www.debevec.org/ReflectionMapping/

Raytracing :

- http://raphaello.univ-fcomte.fr/Ig/RayTracing/LancerDeRayons.htm
- http://artis.imag.fr/~Gilles.Debunne/Enseignement/RayTracer/extensions.html
- http://tfpsly.free.fr/francais/3d/Raytracing.html
- http://www-csl.csres.utexas.edu/users/billmark/teach/cs384g-04-fall/projects/ray/handout.html
- http://www2.cs.fit.edu/~wds/classes/adv-graphics/raytrace/raytrace.html

- http://glasnost.itcarlow.ie/~powerk/Graphics/Notes/node12.html
- http://www.flipcode.com/articles/
- http://www.devmaster.net/articles/raytracing_series/part7.php

Triangles :

- http://heigeas.free.fr/laure/ray_tracing/triangle.html
- http://www.devmaster.net/wiki/Ray-triangle_intersection
- http://www.cfxweb.net/modules.php?name=News&file=article&sid=1308

HeightField:

- http://www.daylongraphics.com/products/leveller/hf.htm

Class diagram :

Check the classdiagram.png image provided with the report.