Raytracer 3

March 13, 2017

Computer Graphics

General note:

In various exercises you will be asked to implement new functionality. In these cases your ray tracer should accept the (syntax of the) example scene files provided. Under no circumstances should your ray tracer be unable to read older scene files (those that do not enable the new functionality), or modify the interpretation of older scene files.

1 Texture mapping and alternative illumination models

In this assignment you will implement texture mapping and an alternative illumination model.

Tasks:

1. With textures it becomes possible to vary the lighting parameters on the surface of objects. For this a mapping from the points of the surface to texture-coordinates is needed. You might want to make a new pure virtual function in Object (and give it a non-trivial implementation in at least Sphere) for computing texture coordinates so that it only has to be done for objects which actually need it. See Link [†] for example textures to use and section 11.2 (2D texture mapping) of your book for how to compute the texture coordinates. For reading the textures you can use the following line: `Image *texture = new Image("bluegrid.png")`, and access the pixel data: `texture->colorAt(float x, float y)`, where x and y are between 0 and 1.

[†] http://planetpixelemporium.com/planets.html

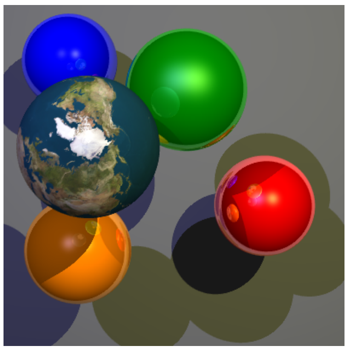

Figure 1.1: *Scene containing a texture mapped sphere*

For testing your texture mapping code you may want to use the following files:

- `bluegrid.png`
- `blue-earth.yaml`

There is extra YAML documentation on Nestor (*Lab Assignments*) for parsing textures.

2. Also implement rotation of (at least) spheres (if you haven't already). In this case you are NOT required to exactly follow the syntax given here, but you are required to implement something that gives the same degrees of freedom. Please look at the given YAML scene file to view the changes to the sphere definition. You are given an axis and an angle. The radius component of the sphere has been augmented with an axis such that the first component `radius[0]` gives the radius and the second component `radius[1]` is a vector. In the example given here the vector defined by `radius[0]*radius[1].normalized()` is mapped to $(0, 0, 1)$. This is done through the simplest possible rotation and then the whole sphere is rotated by angle degrees around the z-axis (note that this corresponds to subtracting the angle in radians from the angle gotten by the arctan). The result should look like this:

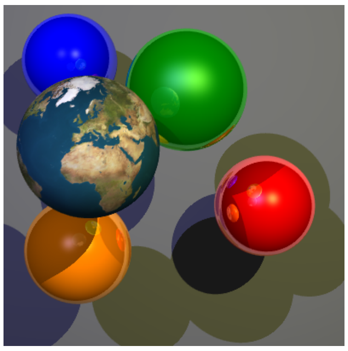

Figure 1.2: *Scene with a rotated sphere (`scene01-texture-ss-reflect-lights-shadows.yaml`)*

3. In particular for illustration purposes alternative illumination models have been developed, which are rather easy to implement in your raytracer. In this assignment you will implement one of these models. Your task is to implement the illumination model by Gooch et al [†]. Be aware of the following:

[†] `http://www.cs.northwestern.edu/~ago820/SIG98/abstract.html`

- The formula for the lighting calculation in the original paper is not correct. Use this one:

$$I = kCool * (1 - dot(\mathbf{N}, \mathbf{L}))/2 + kWarm * (1 + dot(\mathbf{N}, \mathbf{L}))/2$$

  (note that for this formula it is not necessary that $dot(\mathbf{N}, \mathbf{L}) < 0$).
- The variable `kd` in the paper can be set to: `lights[i]->color*material->color*material->kd`.
- Extend the scene description for the new parameters b, y, $alpha$ and $beta$ which are defined in the paper (reminder: your ray tracer should still accept files that do not set these parameters).
- The Gooch model should not replace the Phong model, instead which model will be used should be configurable.
- Gooch should not use ambient lighting, but it can use the same kind of highlights as in Phong shading (the specular component of Phong shading).
- When using Gooch shading you may ignore shadows and/or reflections, but you are not required to (and you might be able to get some interesting effects by not ignoring them).
- The resulting image could look like the images in Figure 1.3 (for the first image `scene01-gooch.yaml` is used):

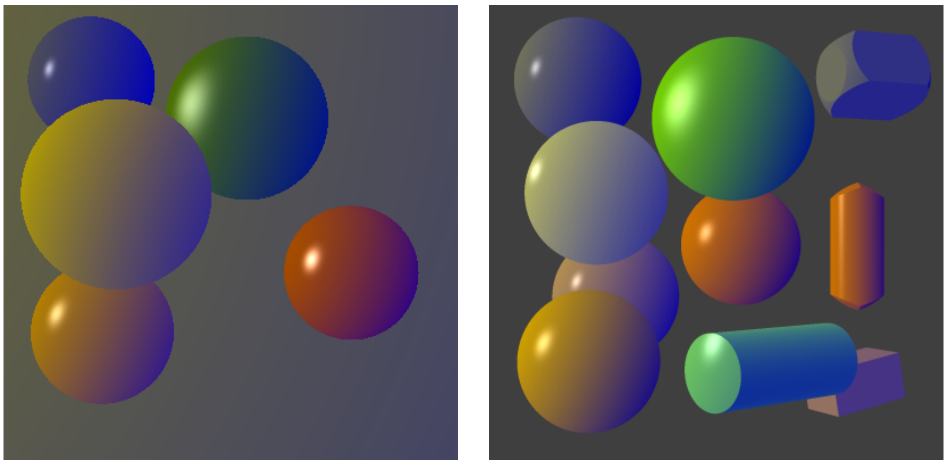

Figure 1.3: *Gooch shaded scenes*

4. **(Bonus)** Implement normal/bump-mapping. The results could be something similar to the following:

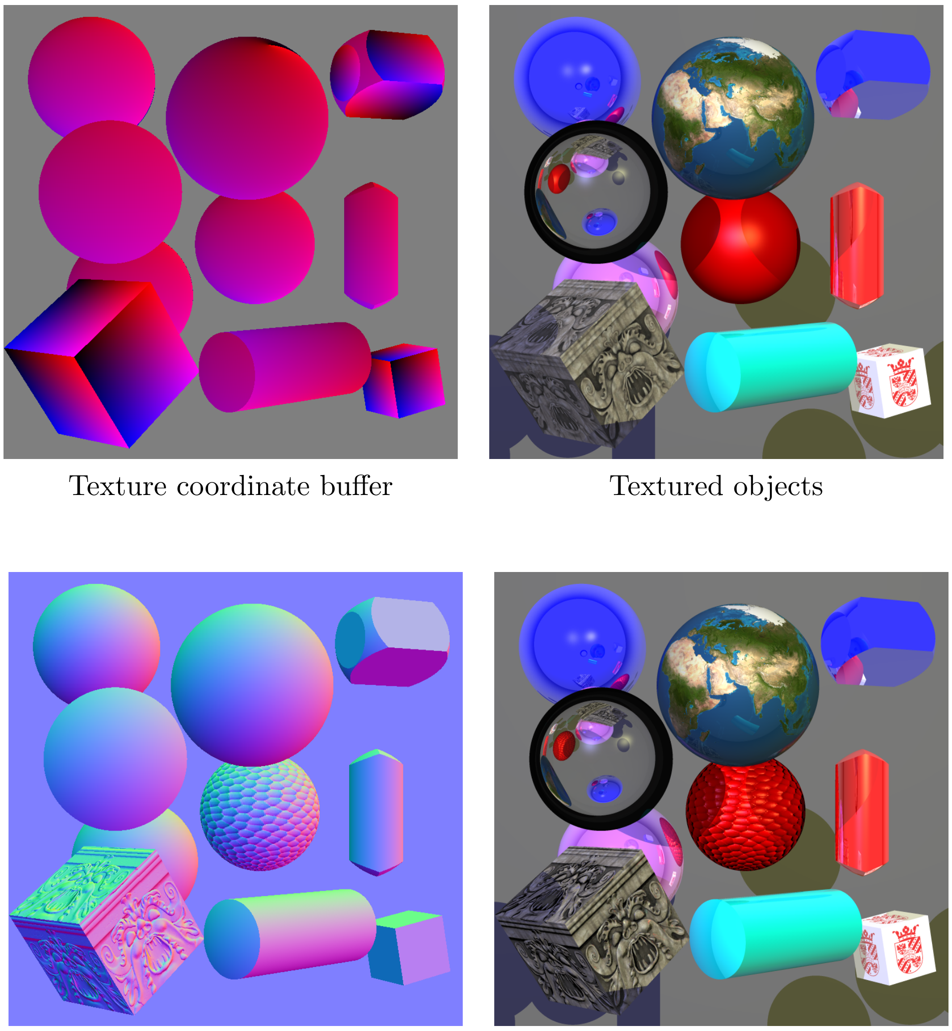

Figure 1.4: *Bump maps*

5. **(Small bonus)** For the Gooch shading: add a black line to the silhouettes of your objects. This should create a really nice effect (as was seen in the tutorial presentation).

2 More geometries and 3D mesh files

1. Implement an additional geometry type from the above or following list. Make sure however, you have implemented at least a triangle.

   - (bi-linear) Quad
   - Planes
   - Polygon
   - Cylinder, Cone, parabolic surfaces
   - Torus
   - Blobs

- Fractals
- Free-form surfaces

So, after finishing this task your raytracer should support four different geometries: a sphere, a triangle, and two others.

2. Implement 3D mesh objects (read from a file). You can take the code from your OpenGL project and port it, or alternatively use `glm`[†]. If you choose to use `glm` you have to retrieve the triangles from the `GLMmodel` (You can do this with: `GLMmodel *model = glmReadOBJ(filename)`). You can use the same models, but be aware that producing a raytraced image of a model with many triangles can take a long time. For example, the image in Figure 2.1 of an evil golden rubber duck (with 3712 triangles) took quite a long time to generate on a reasonably fast machine (with 4×4 super-sampling, relatively un-optimised code). If you want to speed up the tracing of large models, consider employing axis-aligned bounding boxes.

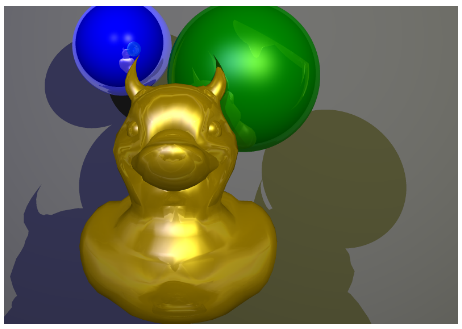

Figure 2.1: *Evil golden rubber duck*

3. **(Bonus)** Implement Phong shading for triangles so as to achieve smooth shading for your objects. In addition to vertex positions you also have to parse vertex normals from the object files.
4. Include your coolest result(s) in the archive that you hand in. You can use images (renders and screenshots). Please don't hide these files too deep in your archive, so that we can easily spot them.
5. **(Bonus)** Implement constructive solid geometry (CSG). An example of a rendering with cylinders and CSG objects or a more complex CSG shape, like the nut, can be found in Figure 2.2.

[†]`http://glm.g-truc.net/0.9.7/index.html`

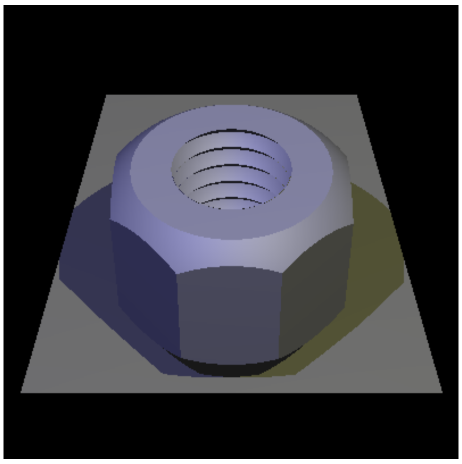

Figure 2.2: *CSG object*

Deadline

See Nestor (*Time Schedule*). Details on how to submit your work can also be found on Nestor (*Lab Assignments*).

Assignment submission

Please use the following format:

- Main directory named `Lastname1_Lastname2_Raytracer_3`, with last names in alphabetical order, containing the following:
- Sub-directory named **Code**, containing the modified C++ framework (please do not include executables or other files you would typically put in a .gitignore)
- Sub-directory named **Screenshots** where you provide the relevant screenshots/rendered images for this assignment
- ReadMe (plain text, short description of the modifications/additions to the framework along with user instructions)

The main directory and its contents should be compressed (resulting in a zip or tar.gz archive) which is the file that should be submitted (using the *Assignment Dropbox*). Example: the name of the file to be submitted associated with the third Raytracer assignment would, in our case, be `Hettinga_Talle_Raytracer_3.tar.gz`.

Assessment

See Nestor (*Assessment & Rules*).