Raytracer 2

February 6, 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.

This time, we learn how to add optical laws, anti-aliasing and an extended camera model into our framework. You can download the `.yaml` files, which describe added scene elements, from Nestor.

1 Optical laws

In this assignment you will implement a global lighting simulation. Using recursive ray tracing the interaction of the lights with the objects is determined. The program should be able to handle multiple colored light sources and shadows.
Tasks:

1. Extend the lighting calculation in `Scene::trace(Ray)` such that it produces shadows. First make it configurable whether shadows should be produced (e.g., Shadows: true). The general approach for producing shadows is to test whether a ray from the light source to the object intersects other objects. Only when this not the case, the light source contributes to the lighting. For the example in Figure 1.1 a large background sphere is added to the scene.

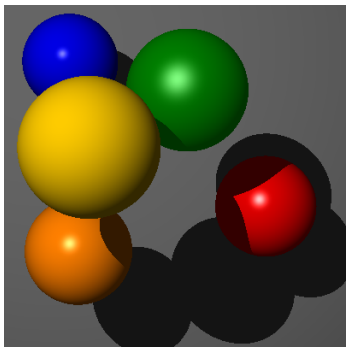

Figure 1.1: *shadow (`scene01-shadows.yaml`)*

2. Now loop over all light sources (if you didn't do that already) and use their color in the calculation. For the following result two different lights were used. The output of this scene file is in Figure 1.2.

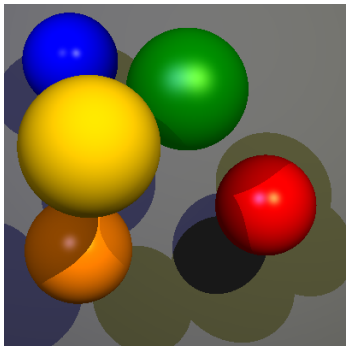

Figure 1.2: *Shadows caused by multiple lights (`scene01-lights-shadows.yaml`)*

3. Implement reflections, by recursively continuing rays in the direction of the reflection vector, treating the found values as light sources. Be sure to only compute the specular reflection for these "light sources"(ambient makes no sense at all, and diffuse reflection is better approximated by not taking it into account in this coarse approximation). An example result with a maximum of two reflections is shown in Figure 1.3.

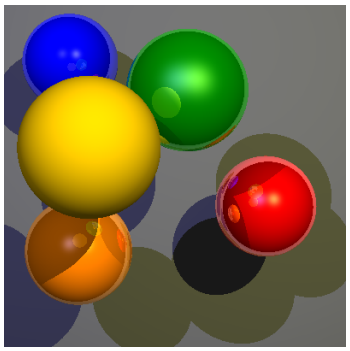

Figure 1.3: *reflection (`scene01-reflect-lights-shadows.yaml`)*

Bonus: Implement refraction, by recursively continueing rays in the direction of the transmission vector, using the parameters `refrct` and `eta`, where `eta` is the index of refraction of the material.
Bonus: You'll notice that the specular reflections from the scene are not blurred, like the light sources. One way to do something about this is to sample along multiple rays (around the reflection vector) and average the results. Note that for a correct result you should be careful about selecting your vectors and/or the way you average them. Hint: if you take a normal average you should select more rays in those areas where the specular coefficient is high.

Anti-aliasing & Extended Camera Model

Anti-aliasing will result in better looking images. In addition to this you will make it easier to move the camera position by implementing an extended camera model.

Tasks:

1. Implement super-sampling (anti-aliasing), i.e., casting multiple rays through a pixel and averaging the resulting colors. This should give your images a less jagged appearance. Note that you should position the (destinations of the) initial rays symmetrically about the center of the pixel, as in Figure 1.4 (for 1x1 and 2x2 super sampling):

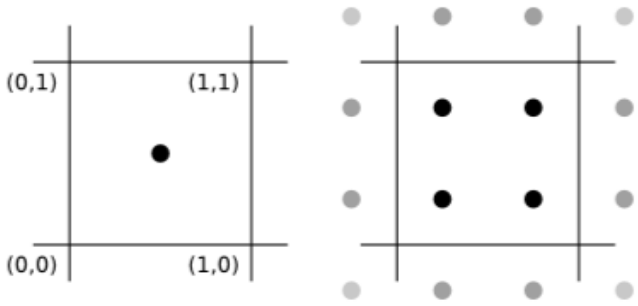

Figure 1.4: *Super sampling*

   Again, make sure this is configurable in the scene file. The default should be to have no super sampling (or, equivalently, super sampling with a factor of 1). An example of 4x4 super-sampling in Figure 1.5 (`scene01-ss.yaml`).

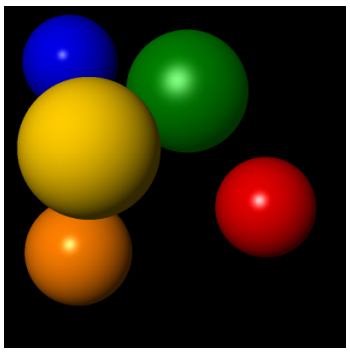

Figure 1.5: *Super sampling example*

2. Implement an extended camera model such that other image resolutions are possible and producing images becomes more flexible. You should keep support for the Eye parameter for backwards compatibility, but allow the specification of a Camera object (instead of the Eye parameter). The camera object should support:

   - an eye position
   - a reference point (center)
   - an up-vector
   - a viewSize (width and height)
   - and a pixelsize (determined by the length of the up-vector).

   Using these definitions you should be able to derive the right-vector from the viewing direction and the adjusted up-vector (such that it is orthogonal to both the

right-vector and the viewing direction). These vectors aid in defining a rectangular grid of points around the center of viewSize * pixelsize. For an example of what this should look like consult the scene files which also contain the parameters to be parsed (`scene01-camera-ss-reflect-lights-shadows.yaml` and `scene01-zoom-ss-reflect-lights-shadows.yaml`):

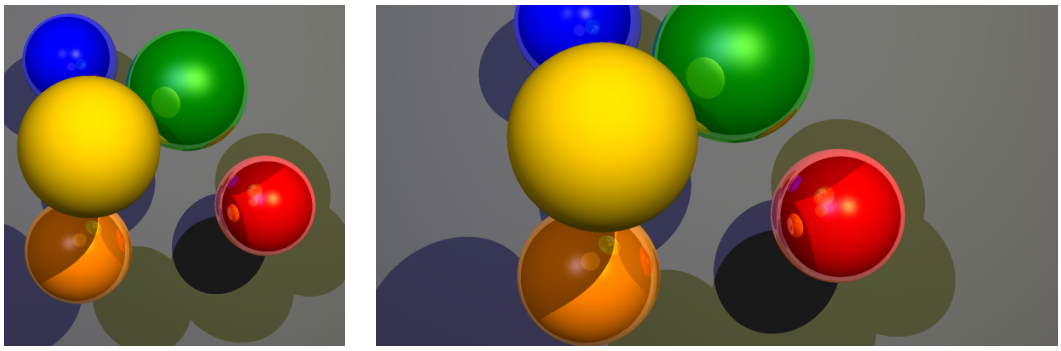

Figure 1.6: *Extended camera model*

For more information on constructing a view, see the lecture slides and labslides or this https://graphics.stanford.edu/courses/cs348b-98/gg/viewgeom.html. And keep in mind that the length of the up vector determines both the "vertical" and the "horizontal" dimensions of a pixel (you can implement additional functionality to allow for stretched views if you want).

Bonus: Implement *apertureRadius* and *apertureSamples* parameters for your camera object and use them to simulate depth of field by taking *apertureSamples* positions (uniformly) within *apertureRadius* of the eye (note that this disc should be formed using the up and right vectors). This can look like the figure below, using Vogel's model[†] with $n \in [0, apertureSamples)$,

$$c = \frac{\text{apertureRadius}}{||up|| * \sqrt{\text{apertureSamples}}} \tag{1.1}$$

$$r = c * \sqrt{n} \tag{1.2}$$

$$\theta = n * \text{goldenAngle} \tag{1.3}$$

and (Eqn. 1.2), Eqn. 1.3, Eqn. 1.1 sample the aperture Figure 1.7.

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_2` , with last names in alphabetical order, containing the following:

[†] https://en.wikipedia.org/wiki/Fermat's_spiral

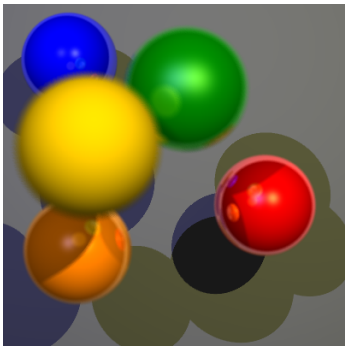

Figure 1.7: *Aperture (`scene01-dof-ss-reflect-lights-shadows.yaml`)*

- Sub-directory named Code, containing the modified C++ framework (please do **not** include executables)
- Sub-directory named Screenshots wherein 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*). An example of a file to be submitted associated with the first raytracer assignment would be: `Catmull_Clark_Raytracer_2.tar.gz`.

Assessment

See Nestor (*Assessment & Rules*).