Advanced Computer Graphics
Exercise 4 - Path Tracing

Handout date: Monday, 12.10.2015
Submission deadline: Monday, 26.10.2015, 11:00pm

Note

Undeclared copying of code or images (either from other students or from external sources) is strictly prohibited! Any violation of this rule will lead to expulsion from the class. No late submission allowed.

What to hand in

Solve the exercise in groups of 3. Hand in a `.zip` compressed file renamed to `Exercise4_#id.zip` where `#id` is your group number. It should contain the following files:

- `path.cpp`: a path tracer,
- An XML scene file plus required mesh files.
- A `readme.pdf` containing a description of your solution approach, your rendered images, how much time you needed and encountered problems (use the same numbers and titles), and providing answers to the questions.

4.1 Before Starting

Open the file `src/ex4/path.cpp` and put your group number in `#define GROUP_NUMBER`.

4.2 Path Tracing (60 points)

In this assignment, you will implement an integrator to perform path tracing. As discussed in the lecture, path tracing is an algorithm to find a numerical solution to the rendering equation. Instead of integrating over a hemisphere (Exercise 2) or over all light surfaces (Exercise 3), we can solve the rendering equation by integrating over all possible light paths of length from 1 to infinity. The light transport equation is then expressed as:

$$L(\mathbf{x}_0,\mathbf{x}_1) = L_e(\mathbf{x}_0,\mathbf{x}_1) + \sum_{i=1}^{\infty} \int_A \dots \int_A T(\mathbf{x}_0,\mathbf{x}_1,\dots,\mathbf{x}_{i+1}) L_e(\mathbf{x}_i,\mathbf{x}_{i+1}) dA(\mathbf{x}_2)\dots dA(\mathbf{x}_{i+1})$$

Later bounces usually contribute less to the final output, path tracing avoids exponential growth of rays by using *Russian Roulette*.
To evaluate the above sum, for each eye ray, we incrementally create a random path through the scene. At each intersection $\mathbf{x}_i$, we add the direct illumination (multiplied with the current path throughput) to the total radiance. After that, we sample one direction ω_i according to the current BRDF and trace it to get the next intersection $\mathbf{x}_{i+1}$.

4.2.1 Path Integrals (30 points)

Complete the method:

```
Color3f Li(const Scene *scene, Sampler *sampler, const Ray3f &ray_) const;
```

to sum up the radiance for all paths with depth ≤ 3 (which correspond to zero bounce (emitted light), one bounce (direct lighting) and two bounces) with the following steps:

1. Find the intersection of the current ray with the scene. (The initial ray is the eye ray.) If this ray does not hit any objects or if it hits a light source, what should you do? Briefly justify your solution.
2. Find the contribution of the current path by light area sampling.
3. Construct the next ray by sampling the BRDF to find a new incoming direction. Remember to update the throughput of the new path segment using the BRDF at the current intersection point.

Render the scene `ajax2-path.xml` and compare your result to Figure 1.

(a) with 4 samples per pixel

(b) with 1024 samples per pixel

Figure 1: The Ajax statue once with diffuse (left) material and once with phong (right) material. Russian Roulette was not used (only paths up to length 3).

4.2.2 Russian Roulette (15 points)

To include the contributions of later bounces, apply Russian Roulette after 2 main bounces. How do you change the throughput of new path to maintain an unbiased estimator for the light transport equation?

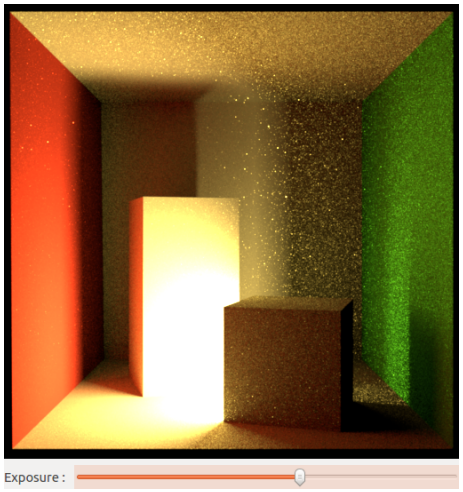
Render the scene `cbox-path.xml` with different termination probability q. Observe the difference in rendering time and rendering results (see Figure 2).

4.2.3 Creation of Scenes (15 points)

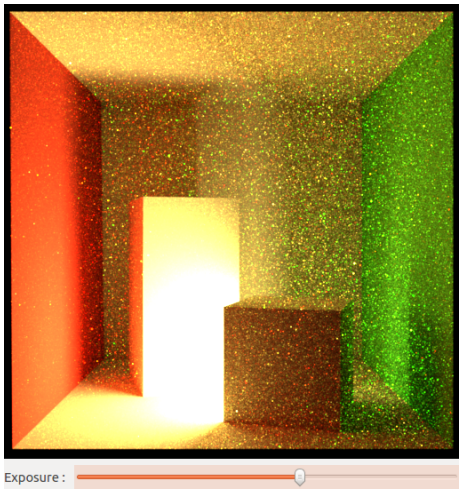
As you may have learned from the lecture, Russian Roulette allows us to estimate the infinite sum of path integrals in an unbiased way. The termination probability q determines the sampling probability of longer paths; thus affects both rendering results and rendering time. Create one new scene which **clearly** shows the difference in the rendering results when q is decreased. You

should have some areas in the image which are not visible with large q but appear and become less noisy as q gets closer to 0.0 (e.g., see how the green region on the right side of the small box emerges in Figure 2). Include in your report 4 renderings with $q = 1.0, 0.7, 0.4$ and 0.1.
Note that just taking the Cornell Box (Figure 2) and changing the colors of the objects is not sufficient. Please create your own scene from scratch.
Now reduce the number of rays per pixel by changing the value in `<integer name="sampleCount" value="256"/>` so that you can have roughly similar rendering time for the above four values of q. Comment on the new results and briefly discuss your observation. Also include the new renderings in your report.

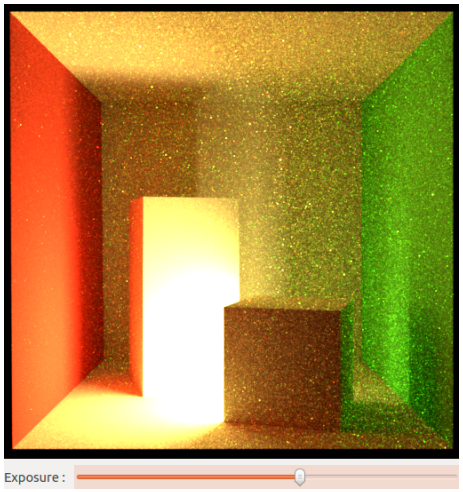
Hint: Scenes with many specular surfaces or scenes with no direct path from light sources to the eye might be interesting.



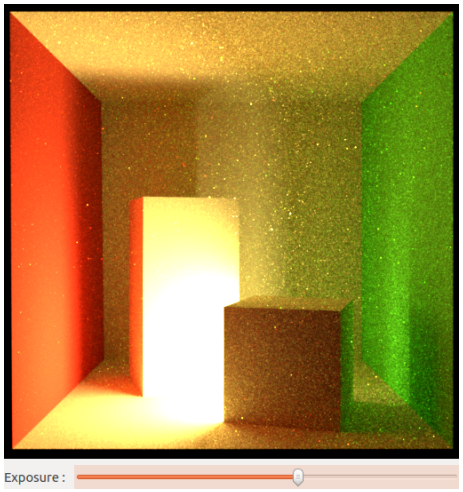
(a) $q = 1.0$ (stop after 2 bounces)



(b) $q = 0.7$



(c) $q = 0.4$

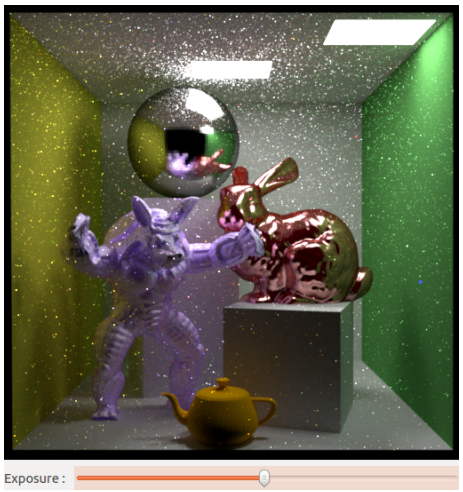


(d) $q = 0.1$

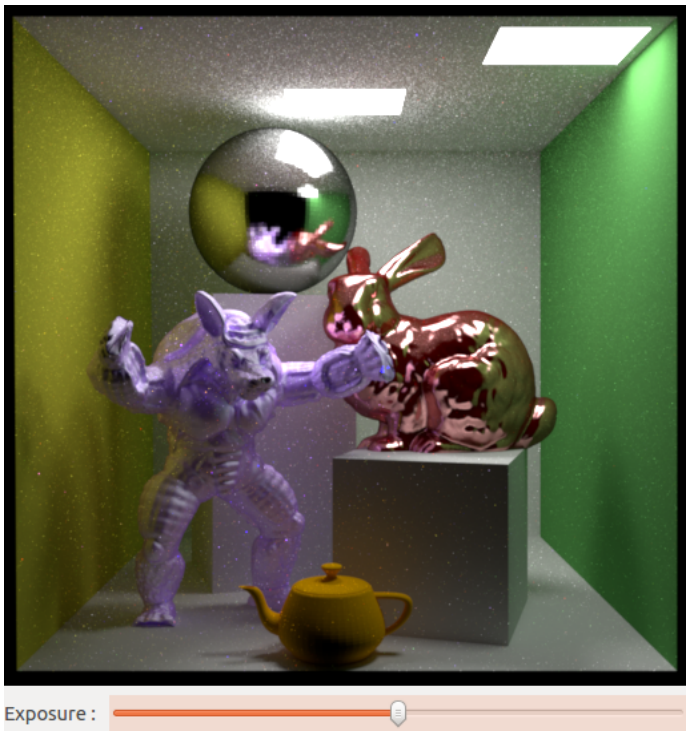
Figure 2: Path tracing of the Cornell Box with different values of the termination probability q. We used 256 samples and slightly increased the exposure in the viewer.

For your own scene you will have to create a new xml file. Have a look at the other provided scenes to understand how to compose your own setting. Your encouraged to search for or create new 3D meshes. Be creative!

(a) 4 samples per pixel



(b) 2048 samples per pixel



(c) 65536 samples per pixel

Figure 3: An example composition of several meshes in the Cornell Box. Note that you can create mirror like objects by using the phong material with a high specular exponent and by setting the diffuse color values to 0. This scene can be found in `cbox-meshes-path.xml`. All 3 images were rendered with $q = 0.1$. What technique could be applied to make the image converge faster (you don't have to implement that)?