Graphics and Imaging Lab

Assignment #4

Path Tracing

Modeling and Simulation of Appearance

Master on Robotics, Graphics and Computer Vision

Overview

In this last exercise, you will combine everything you've learned to write integrators for full light transport. After completing this assignment, you will be able to render global illumination, caustics, depth of field and more with your code.

The integrators you will implement in this assignment compute solutions to the light transport integral

$$L_o(\mathbf{x}, \omega_o) = L_e(\mathbf{x}, \omega_o) + \int_{\Omega} f_r(\mathbf{x}, \omega_o, \omega_i)\, L_i(\mathbf{x}, \omega_i)\, \cos\theta_i \, \mathrm{d}\omega_i \qquad (1)$$

where f_r is the BRDF, $\mathbf{x}$ is a surface position and L_i and L_o denote the incident and outgoing radiance functions. Unlike the previous assignment, we do not truncate the recursion - we won't just compute direct lighting, but the full solution of this equation.

Note: This assignment builds heavily on code from Assignment 3. Make sure your solution to Assignment 3 works correctly before proceeding to work on this assignment.

1 Path Tracing (90%)

1.1 Naive path tracing (30%)

In this task, you will implement a naive path tracer to compute global illumination. This integrator will be very similar to the BSDF integrator for direct illumination you implemented in the previous assignment using material-based sampling (i.e. `direct_mats.cpp`).

Create a `path.cpp` file, and create a new integrator called `PathTracing`. Use the `NORI_REGISTER_CLASS` to register the new integrator as "path" as:

```
NORI_REGISTER_CLASS(PathTracing, "path");
```

Your integrator should proceed similar to the BSDF integrator from the previous assignment: Trace a ray from the camera, sample the BSDF and trace a ray in the sampled direction. However, unlike the previous assignment, your code should repeat this process - at the hitpoint of the second ray, your code should again sample a direction and trace a new ray, and repeat until the path leaves the scene. You should use the Russian roulette technique presented in the lecture as a termination criterion to avoid infinite recursion.

In order to test this first part, you can use the scene in assignment-4/cbox/cbox_path.xml with only diffuse objects. It should converge to the reference image in scenes/assignment-4/references.

1.2 Path tracing with Next-Event Estimation (30%)

The integrator implemented in the previous subtask is a great start, but for scenes with small light sources it can perform poorly. For this reason, implement a new integrator that performs next event estimation. Create a path_nee.cpp file, and create a new integrator called PathTracingNEE. Use the NORI_REGISTER_CLASS to register the new integrator as "path_nee" as:

```
NORI_REGISTER_CLASS(PathTracingNEE, "path_nee");
```

Your integrator will now generate two rays at every bounce: a recursive ray sampled from the BSDF as before, and one ray produced using emitter sampling. That is, at each bounce your code should accumulate radiance using your emitter sampling code from the previous assignment, and then generate the next point on the path using BSDF sampling. Use your implementation of direct illumination based on light sampling in direct_ems.cpp as a guide for implementing NEE. To avoid double counting, do not accumulate radiance from light sources that were hit using BSDF sampling, except if the BSDF is perfectly smooth. This is because perfectly smooth BSDFs cannot be sampled using emitter sampling.

You can check if a BSDF is perfectly smooth by sampling a direction and checking if the measure field of the BSDFQueryRecord is EDiscrete.

Implementing a path tracer with NEE iteratively might be difficult, specially when dealing with directly visible light sources. We encourage you to implement it recursively to account for the first bounce appropriately.

For debugging, use the scene in assignment-4/cbox/cbox_nee.xml and compare it against assignment-4/cbox/cbox_path.xml. They should converge to the same result.

1.3 Path tracing with MIS (30%)

The integrator implemented in the previous subtask works great dealing with direct illumination, and will perform nicely in many scenes. However, for very specular surfaces or low-frequency illumination (e.g. large area lights) might introduce significant variance. For this reason, you would like to combine the estimate of BRDF and light sampling using Multiple Importance Sampling. Create a path_mis.cpp file, and create a new integrator called PathTracingMIS. Use the NORI_REGISTER_CLASS to register the new integrator as "path_mis" as:

```
NORI_REGISTER_CLASS(PathTracingMIS, "path_mis");
```

Similar to the previous integrator, PathTracingMIS will now generate two rays at every bounce: a recursive ray sampled from the BSDF as before, and one ray produced using emitter sampling. However, at each bounce now you should accumulate radiance using your emitter sampling code from the previous assignment, and weight it using multiple importance sampling. Then, generate the next point on the path using BSDF sampling; if the next hitpoint is on a light source, accumulate its radiance, weighted by the MIS weight for BSDF sampling.

For debugging, use the scene in assignment-4/cbox/cbox_mis.xml and compare it against assignment-4/cbox/cbox_path.xml. They should converge to the same result.

Note: Each subtask is needed to complete the following one (i.e. for a path tracer with NEE, you will need to implement a path tracer first, and for accounting for MIS in NEE, you'll need to have implemente NEE. Reuse as much code as needed, but be aware that a bug in a simpler version of the code will propagate to the rest. Additionally, note that NEE is in essence combining direct light sampling with recursive estimate of the BRDF sampling. Therefore, you can reuse code from the previous assignment.

2 Interesting Scene (10%)

Test your implementation of the three integrators in the scenes provided in the scenes patch in Moodle, including /cbox/cbox_INT.xml, /cbox/cbox_glossy_INT.xml, /cbox/cbox_caustics_INT.xml, /veach_mi/veach_INT.xml,
and /table/table_INT.xml, where INT accounts for path, nee and mis depending on the integrator used). Note that this time there is no code patched for the assignment.

In addition, make your own interesting scene and submit a rendering (or several) using any combination of the implemented techniques. Name it as interesting_X.exr and interesting_X.png, with X the image index in case you submit more than one image. Be creative, the best renderings will be shown in class. Note that all implemented features need to be shown and properly commented in at least one of the submitted scenes (see Section 4); **otherwise, they will count as not presented!**

3 References

You may find the following general references useful:

- *"Physically Based Rendering, Fourth Edition: From Theory To Implementation"* by Matt Pharr, Wenzel Jakob, and Greg Humphreys. The MIT Press, 4th edition, March 2023. http://www.pbr-book.org/
- *"Global Illumination Compendium - The Concise Guide to Global Illumination Algorithms"*, Philip Dutre, 2003. https://people.cs.kuleuven.be/philip.dutre/GI/
- *"Robust Monte Carlo Methods for Light Transport Simulation"*, PhD Thesis by Eric Veach, Stanford University, December 1997. https://graphics.stanford.edu/papers/veach_thesis/thesis-bw.pdf
- Online Nori docs: https://www.cs.cornell.edu/courses/cs6630/2012sp/nori/api/annotated.html/

Feel free to consult additional references when completing projects, but remember to cite them in your writeup. When asked to implement feature X, we request that you don't go and read the source code of the implementation of X in some other renderer, because you will likely not learn much in the process. The PBRT book is excluded from this rule. If in doubt, get in touch with the course staff.

4 What to submit?

You should submit all your rendered figures and the source code you generated in this assignment. The submission should be a a `zip` file with name `p4_NIP1_NIP2.zip`, and with the following structure:

- A file `README.txt` including the name of the authors (as sanity check) and all references consulted during the development of the assignment. Also, include a brief one-line comment about the features shown in the scene submited for Section 2.
- A folder `./figures` including all generated figures (both `exr` and `png`); the figures should be named with the name of the `xml` scene, appended with the number of samples used to generate these figures.
- A folder with all source files you modified or added in your implementation.

In case your file gets too big for the submission system, you can just include a link to a shared folder (e.g. Google Drive) with additional figures in `README.txt`.

The **deadline** for this task will be just before the first session of the final project corresponding to the lab group you are enrolled on:

- **Group 1 (Tuesdays B): December 9, 2024.**
- **Group 2 (Thursdays A): November 27, 2024.**