Assignment 8: Advanced Rendering

CS148 Fall 2015-2016

Introduction

In this assignment, you will be exploring more advanced topics in rendering. There are three options: depth of field, motion blur, and photon map rendering. You only have to implement one and show it running on a scene of your choice. There will be no starter code provided aside from the Assignment8.h/cpp files so be sure to start early and ask questions if you have any. By default, the code will load up the Cornell Box with a checkerboard floor and ceiling. It will be useful to review the lecture slides before starting this assignment. Do not forget that there is an alternative Assignment 8 this quarter: in class Thursday there will be a quiz consisting of multiple-choice and true-false questions that will probably take around 30 minutes. If you do not want to work on the homework throughout Thanksgiving break and/or none of these effects are of particular interest to you, feel free to take the quiz instead! The quiz will cover the lectures from 11/10, 11/12, and 11/17.

Downloading the Code

As always, run:

```bash
git pull origin release
```

to download the new code! See this Piazza thread if git complains about merges. You may have to re-run CMake to get the project to compile properly.

Assignment (Choose ONE)

Note that each possible assignment has some difficulty attached to it. We will take the difficulty of the task you choose into consideration come grading time (this does not apply to the quiz, the difficulty there is just for your reference).

In-Class Quiz (Difficulty: Easy-Medium)

Just another reminder that there will be an in-class quiz on Thursday covering the lectures from: 11/10, 11/12, and 11/17. We will get your quiz grade back ASAP (Thursday night) so you have time to decide whether or not you want to attempt the programming assignment instead.
Depth of Field (Difficulty: Easy)

In a pinhole camera, every object is in focus. However, in reality, cameras have apertures that control which and how many light rays hit the image plane. If you look at the slides from November 17th (Tuesday), you will see that you will need to add support for the “focal plane” as well as a circular “lens.” Here are some pointers for how to get started (note that there are many ways of implementing this and this advice is not complete guide to what you need to do):

- See common/Scene/Camera/Perspective/PerspectiveCamera.h/cpp and look for the “GenerateRayForNormalizedCoordinates” function. This takes in a normalized random sample (values from the range 0 to 1 for both the x and y direction) and constructs a ray that passes through that point on the image plane.
- This function gets called from 'Run' in common/RayTracer.cpp. Increasing the number of samples per pixel will cause ‘GenerateRayForNormalizedCoordinates’ to be called for the same pixel multiple times. However, each sample may receive a different normalized coordinate.
- You should find the ‘GetUpDirection’ and ‘GetRightDirection’ functions helpful for sampling in a circular disk around the camera position.

Motion Blur (Difficulty: Easy)

Cameras typically have shutters that are open for a certain amount of time. During that time, objects may move causing rays from a single point on the object to fall on different points on the image plane. This results in what we call “motion blur.” See the lecture slides from November 17th (Tuesday) for more details. Here are some pointers for how to get started (note that there are many ways of implementing this and this advice is not complete guide to what you need to do):

- You will need to introduce the concept of time to the ray tracer in common/RayTracer.cpp (namely the ‘Run’ function). At each time point that you sample, you will need to update the transformation of your objects and reupdate their acceleration structures and bounding boxes (via the “Finalize” function).
- You should also introduce the concept of ‘temporal samples’ into the code. Modify the loop in the ‘Run’ function accordingly. Currently, the image writer assumes that the call to ‘ComputeSamplesAndColor’ function returns the final color of the pixel. You will have to change it to average together the temporal samples.
- The scene as well as the individual scene objects need to have some concept of time added to them.

Photon Map: Gathering (Difficulty: Medium)

Last week you implemented the photon generation part of photon mapping. This week you have the option of extending it such that you can use your photon mapper, in conjunction with the existing backward renderer, to achieve global illumination. To use the photon map for rendering, you just have to query for the photons near the intersection point and treat each photon you find as a mini-light. Using the photon map this way can lead to “splotchy” results, which usually leads to implementing a “final gather” step (optional, not required for this assignment). Here are some pointers for how to get started (note that there are many ways of implementing this and this advice is not complete guide to what you need to do):

- It is critical that your photons are generated properly. If your assignment 7 has wrong photons/incorrect rays/bugs in general, make sure you fix them first.
- The bulk of the work will be done in common/Rendering/Renderer/Photon/PhotonMappingRenderer.cpp in the “ComputeSampleColor” function. The ‘find_within_range’ function finds the photons
within some distance $r$ (0.003 in the current code to visualize photons – you will have to play around with this value to see what works best for your scene).

- Do not forget to turn off the photon map visualization by setting VISUALIZE_PHOTON_MAPPING to 0.
- Do note that PhotonMappingRenderer inherits from BackwardRenderer so you can call BackwardRenderer functions and have access to its public/protected variables.
- It may be helpful to look at the BackwardRenderer implementation in common/Rendering/Renderer/Backward/BackwardRenderer.cpp to see how to calculate an object’s BRDF given some light source.

### Photon Map: Dispersion (Difficulty: Medium)

Instead of just doing diffuse reflection in your photon map generation, you will now want to also account for refraction. In addition to refraction, you will want to handle the dispersion effect as well. Now, instead of sending out photons with the same color, you will want to send out photons that are sampled from the color spectrum (different wavelengths). Then when you do refraction, you will have to account for the change in the index of refraction due to the different wavelengths. For this assignment, you only have to show the photon map with the various different color photons (note: this is the same as assignment 7, except that your photons should be colored based on its wavelength instead of being all red). Here are some pointers for how to get started (note that there are many ways of implementing this and this advice is not complete guide to what you need to do):

- It is critical that your photons are generated properly. If your assignment 7 has wrong photons/incorrect rays/bugs in general, make sure you fix them first.
- Note that you will want to keep these refracted rays in a separate photon map than the diffuse photons you generated last week. See the ‘InitializeRenderer’ function in common/Rendering/Renderer/Photon/PhotonMappingRenderer.cpp.
- It may be useful to only shoot photons towards objects that refract light. See the ‘IsTransmissive’ function in common/Rendering/Material/Material.h.
- Note that you will have to keep track of the index of refraction of the object you are in. The skeleton for that already exists as intersection state and the ‘TracePhoton’ function both keep track of the current index of refraction (‘currentIOR’).
- See the ‘ComputeSampleColor’ in PhotonMappingRenderer.cpp for how the photons are currently displayed.
- Note that you should only care about the photons that get refracted. You do not have to perform any diffuse bounces.

For this task, you may want to use a simpler scene than the Cornell Box. You will want to create a scene with a light source, a transmissive prism, and a ground plane.

### Volumetric Rendering (Difficulty: Very Hard)

Volumetric rendering will allow you to render effects such as smoke and clouds. The first thing you want to do is to create some 3D grid in your scene and assign each ‘voxel’ some density and create some function to convert the density to $c$ (see the slides from 11/17, notably the inhomogenous Beer’s Law slides). Then when you go to trace the volumetric 3D grid, you will want to step through the grid with some finite step size to calculate the total attenuation along the ray due to the volume. You should be able to perform this attenuation for both the shadow rays and the camera rays. After you handle the attenuation, you will have to actually add in the color of the volumetric effect. To do this you will want to create a volumetric light map for each of your volumes. Then when you go to actually ray-trace the volume, as you step through the volume to compute the total attenuation, you will want to take into account the ‘in-scattering’ light which you can take from the previously computed volumetric light map and add that to color from the next sample
point in the volume multiplied by by the percentage of light that did not get absorbed. Then you will add in the color that you would get without the volume multiplied by the final attenuation value.

Once again, note that this task is very hard. Review the slides, review the actual lecture video on these slides, make extensive use of Google, and ask questions if you have any. The relevant slides for this task are slides 22 to 35.

**Grading (Programming)**

The requirements for this assignment are:

- Implement ONE of the above effects correctly.
- Display the effect CLEARLY on a scene of your choice. By default, the Cornell Box is given to you but you can modify/change the scene as necessary to display your effect. It should be painfully obvious to the CA that your effect is in the scene.
- Explain to the CA what effect(s) you implemented and how you went about doing it.

You will be graded as follows (note that the difficulty of the task will be taken into consideration):

- + – Exceed the requirements with one or more technical or artistic contributions.
- ✓ – Meet the requirements.
- – – Show up to the grading session and discuss what you tried.
- 0 – Do not show up to the grading session. :(

4