CS7.302: Computer Graphics

Assignment 3 [100 points]

Deadline: 11:59 PM, 19th Feb 2024

Welcome to your third assignment. In this assignment, you will implement direct lighting in presence of area lights using Monte-Carlo integration. Like the previous assignment, you will build on top of an updated version of the base code. Please pull the latest version of the code from the GitHub Repository. The scenes required for this assignment have also been uploaded to scenes repository, so make sure to pull the latest version for that, too. Finally, the Blender Addon for exporting the scene has been updated to support exporting area lights[1] from Blender. Download and install the latest addon version (v0.3) if you want to test your renderer on any scenes you create. For all of the questions in the assignment, use a diffuse BRDF, i.e $f(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) = \frac{c(\mathbf{x})}{\pi}$.

1 Background

The light transport equation which describes the shading for a pixel is given as:

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = \int_C \int_\Omega L_i(\mathbf{x}, \omega_\mathbf{i}) f(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) V(\mathbf{x}, \omega_\mathbf{i}) (\omega_\mathbf{i} \cdot \mathbf{n}) d\omega_\mathbf{i} dc. \tag{1}$$

Refer to slides of lecture 8 & lecture 9 for the exact definition of each term. Note that we have two integrals in this case. This is because we will also be implementing pixel subsampling for Anti-aliasing (see section 2)

Unlike in the case of directional lights and point lights, a closed-form solution to the above equation doesn't exist in the presence of area lights. Hence, we rely on Monte Carlo integration to estimate the equation numerically. The Monte Carlo estimator for the light transport equation is given as follows:

$$L_o(\mathbf{x}, \omega_\mathbf{o}) = \frac{1}{N} \sum_{i=0}^{N} \frac{L_i(\mathbf{x}, \omega_\mathbf{i}) f(\mathbf{x}, \omega_\mathbf{o}, \omega_\mathbf{i}) V(\mathbf{x}, \omega_\mathbf{i}) (\omega_\mathbf{i} \cdot \mathbf{n})}{p(\mathbf{x}, \omega_\mathbf{i})} \tag{2}$$

Note that pixel-subsampling probability ($p(c) = 1$) has been omitted for brevity. The choice of sampling the outgoing direction has a significant impact on the convergence rate of the estimator. It is preferable to sample from a distribution that closely resembles the estimated quantity to achieve faster convergence. This is called importance sampling. Generally, it is hard to sample exactly from the integrand due to its complex nature. Hence, we settle on sampling proportional to one of the terms of the integrand. In this assignment, you will implement three importance sampling strategies (see section 4) and compare them.

[1] Only area lights of shape **Square** and **Rectangle** are supported. If the area light is of a different shape, it will be skipped.

2 Pixel Subsampling for Anti-aliasing [20 points]

In this question, you will implement pixel subsampling to Anti-alias the rendering. Doing this would remove the jagged edges due to undersampling of the underlying signal. Modify the camera ray generation to sample a ray from the entire pixel instead of only sampling the center of the pixel. You may use the `next_float()` function to generate random numbers in the range $[0, 1)$. Render the scene provided (`Assignment 3/Question 1/scene.json`) with 1 sample per pixel (spp) and 32 spp and add it to the report along with runtimes for both.

3 Area Light Support[20 points]

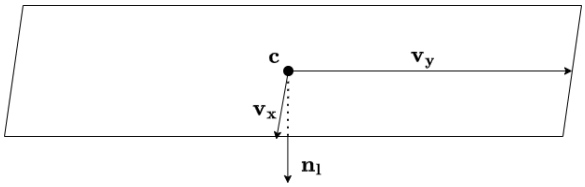

Figure 1: Area light is defined using the center $\mathbf{c}$ and two orthogonal vectors $\mathbf{v_x}, \mathbf{v_y}$ and normal $\mathbf{n_l}$

3.1 Define & load area lights[5 points]

To support area lights in our integrator, we first need to load and store them. The area lights are defined by the center $\mathbf{c}$, and two vectors $\mathbf{v_x}, \mathbf{v_y}$ which extend from the center to the edge (see fig. 1). Furthermore, the light only emits from a single face in direction of the normal $\mathbf{n_l}$. Complete the constructor for the `Light` class in `light.cpp` to load the area lights.

3.2 Intersection[15 points]

Define an intersection routine that tells if the ray intersects any of the lights in the scene. A new intersection routine, `Scene::rayEmitterIntersect` has been added, which iterates over all the lights to find the closest intersecting light, if any. Note that we could have opted to build a separate BVH on the lights or include them in the object BVH itself. However, we chose a simpler iterative method since we don't plan to have scenes with many lights.

Complete the `Light::intersectLight` method for the area light case. If the ray intersects with the light, then store the emitted radiance in `si.emissiveColor` and use that for shading the pixel in the main render loop. Render the scenes provided (`Assignment 3/Question 2`) and add them to the report. Note that the renders will be black for regions which don't intersect with area lights, since we haven't added support for rendering with area lights yet.

4 Monte-Carlo & Importance Sampling [50 points]

In this question, you will be implementing a Monte-Carlo estimator for the direct lighting equation given in eq. (1)

4.1 Uniform Hemisphere Sampling [25 points]

Implement a Monte Carlo estimator where the outgoing direction is uniformly sampled from the upper hemisphere formed at the shading point. Shoot a ray in the sampled direction to check if the closest intersection of the ray is a light source. If the intersection is a light source, then accumulate the reflected radiance according to the estimator given in eq. (2). Note that you would be required to build an Orthonormal Basis (ONB) at each shading point, which would be used to convert the vectors from shading space to world space and vice-versa. Refer to slides of lecture 8 and Möller & Hughes for more details about the function of ONB and the method to build them.

4.2 Cosine Weighted Sampling[10 points]

Instead of uniformly sampling the hemisphere, use cosine weighted sampling to find the outgoing direction ω_o.

4.3 Light Sampling [15 points]

In this sub-question, you will implement importance sampling of the lights in the scene. Sampling lights involves two steps: choosing a light to sample from and then choosing a point on the light. Use a simple uniform sampling strategy to choose the light to sample from. The sampling routine for directional and point lights has already been provided to you (`Light::sample` in `light.cpp`). Extend this sampling routine to sample the surface of the chosen area light uniformly and use it to implement the final estimator.

Render all of the provided scenes (`Assignment 3/Question 3/`) with 10, 100, and 1000 spp. Add the renders to the report and along with the render time for each.

5 Report[10 points]

Based on the results from the previous questions, answer the following in the report.

- Why can't we render point and directional lights with uniform hemisphere sampling or cosine weighted sampling?
- Why does the noise increase for the same number of samples in the case of uniform hemisphere and cosine weighted sampling as the size of the area light decreases?

6 Submission

You need to submit the modified code along with a report. Add two command line parameters allowing you to set the spp and choose the sampling method. The renderer should be invoked as follows:

```
./render <scene_path> <output_path> <spp> <sampling_method>
```

The `sampling_method` will be an integer from $0 - 2$ where each integer is defined as follows:

- 0: Uniform Hemisphere sampling
- 1: Cosine weighted sampling
- 2: Light sampling

We have provided the ground-truth images rendered with a reference renderer in the **gt** directory for each scene. You may use these to verify the correctness of your code. Since Monte-Carlo integration is stochastic, we don't expect exact matching images. However, we will check for any obvious artifacts or differences with the ground truth during the evaluation. Make a **.zip** file with the modified code, report. You may delete the **.git** and **extern** folders from the code before submitting to avoid the file-size limit issues on Moodle.

7 References

- PBRT v3, Section 13.6
- Building an Orthonormal Basis