Tutorial 8
Introduction to Assignment 4

Zhanrui Zhang

Agenda

- Quick review on the Path Tracing algorithm.
- Introduction to assignment 4.

The Path Tracing algorithm

Rendering Equation

$$L_o(x, \omega_o) = \int_{\Omega^+} L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \cos\theta \, \mathrm{d}\omega_i$$

Sampling hemisphere

$$= \int_A L_i(x, \omega_i) f_r(x, \omega_i, \omega_o) \frac{\cos\theta \cos\theta'}{\|x' - x\|^2} \, \mathrm{d}A$$

Sampling light

$$L_o(p, \omega_o) \approx \frac{1}{N} \sum_{i=1}^{N} \frac{L_i(p, \omega_i) f_r(p, \omega_i, \omega_o)(n \cdot \omega_i)}{p(\omega_i)}$$

By Monte-Carlo integration

The Path Tracing algorithm (in recursive form)

```
shade(p, wo)
    # Contribution from the light source.
    L_dir = 0.0
    Uniformly sample the light at x' (pdf_light = 1 / A)
    Shoot a ray from p to x'
    If the ray is not blocked in the middle
        L_dir = L_i * f_r * cos θ * cos θ' / |x' - p|^2 / pdf_light

    # Contribution from other reflectors.
    L_indir = 0.0
    Test Russian Roulette with probability P_RR
    sample the hemisphere toward wi
    Trace a ray r(p, wi)
    If ray r hit a non-emitting object at q
        L_indir = shade(q, -wi) * f_r * cos θ / pdf_hemi / P_RR

    Return L_dir + L_indir
```

The Path Tracing algorithm (loop form)

Beta = 1, L = 0
Generate a ray from the camera to a pixel on the image plane
For i = 1 to max depth
(0) Suppose the marching ray hits at P(i)
(1) L += Beta * Direct lighting at P(i)
(2) Sample ONE next ray according to P(i)'s BRDF and find the corresponding Pdf
(3) Beta *= BRDF * cosΘ / Pdf
(4) Spawn and march the new ray

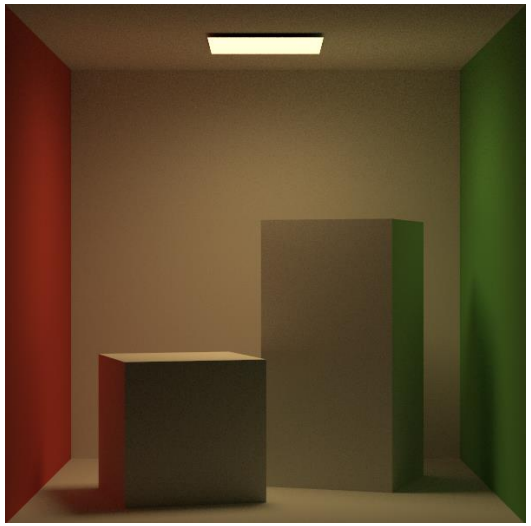
where L is the output radiance.

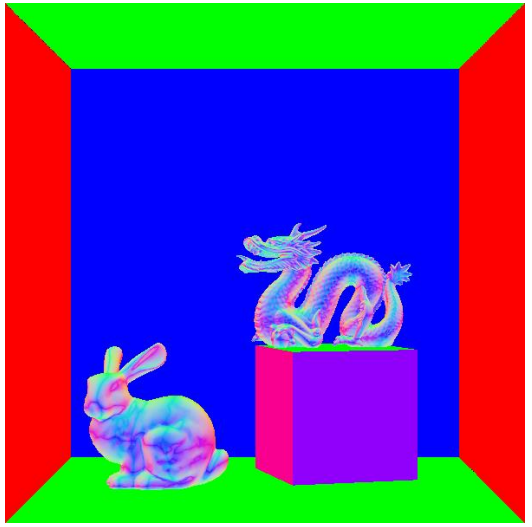
BRDF

- It is a “distribution function”
- Sample according to the distribution/evaluate pdf for given samples
- Evaluate(): compute the brdf value for given wi, wo
- Pdf(): compute pdf value from given wi, wo
- Sample(): sample the direction of next ray for given wo.

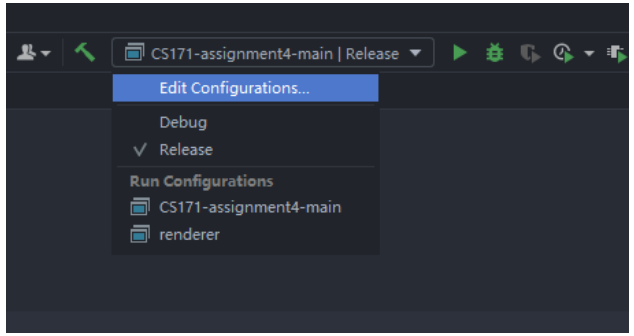
Implementation guide to assignment 4

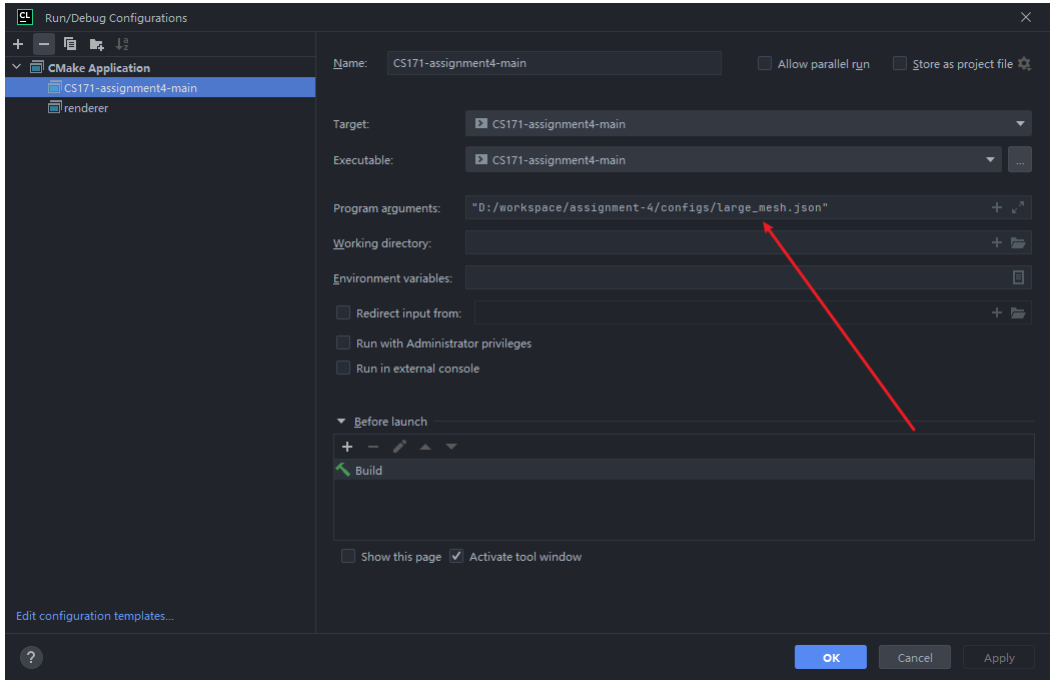
- Part. 1 BRDF & Area Light and Path Tracing algorithm.
  - Use simple.json (6*2*2 + 5*2 + 2 = 36 triangles)
  - Test intersection with every triangle (already provided in template)
- Part. 2 BVH acceleration structure.
  - Visualize with normal direction. (one ray per pixel)





About code skeleton…





Notes

- For Mac user, you may need to install OpenMP manually if you use clang as compiler.
- If you need more resources to demonstrate the effectiveness of your algorithm, you can visit Rendering Resources | Benedikt Bitterli's Portfolio (benedikt-bitterli.me).
- For scene settings
  - Origin point: center of floor
  - X-axis: left to right, pointing right
  - Y-axis: pointing up, ceiling height: 2m
  - Z-axis: positive direction: coming out from screen.

Bonus

- To support new brdf in json, you may need to modify “config.h”, “config_io.h” and “scene.cpp”.
- You need to load environment texture from file to support environment lighting.
- If you decide to implement an advanced BVH, you do not need to implement the simpler version.
- For challenging problems: be careful since there are many wrong implementations on GitHub.

Reference

- Code
  - Mitsuba & pbrt & tungsten
- Books
  - Ray tracing in one weekend (only for beginners)
  - Physically based rendering – from theory to implementations
- Online course
  - GAMES101 Lecture 16
- Do not try to copy code from GitHub, since they tend to be wrong.

START EARLY!

Good luck and have fun!