ECSE 446/546: Realistic/Advanced Image Synthesis

Assignment 4: Path Tracing

Due: Thursday, December 12th, 2024 at 11:59pm EST on myCourses
Final weight: **30%**

Contents

1 Assignment Policies

As with previous assignments, this one builds on top your previous code bases.

1.1 Assignment Submission

Gather all the python source files within the `taichi_tracer/` folder (i.e., everything except the `scene_data_dir/` folder) and compress them into a single `.zip` file. Name your `.zip` file according to your student ID, ex:

`YourStudentID.zip`

For example, if your ID is **234567890**, your submission filename should be **`234567890.zip`**.

DO NOT ADD ANYTHING BEFORE OR AFTER THE MCGILL ID.

> Every time you submit a new file on myCourses, your previous submission will be overwritten. We will only grade the **final submitted file**, so feel free to submit often as you progress through the assignment.

In accordance with article 15 of the Charter of Students' Rights, students may submit any written or programming components in either French or English.

1.2 Late policy

All the assignments are to completed *individually*. You are expected to respect the late day policy and collaboration/plagiarism policies, discussed below.

> **Late Day Allotment and Late Policy**
>
> Every student will be allowed a total of **six (6)** late days during the entire semester, without penalty. Specifically, failure to submit a (valid) assignment on time will result in a late day (rounded **up** to the nearest day) being deducted from the student's late day allotment. Once the late day allotment is exhausted, any further late submissions will obtain a score of **0%**. Exceptional circumstances will be treated as per McGill's Policies on Student Rights and Responsibilities.

If you require an accomodation, please advise McGill Student Accessibility and Achievement (514-398-6009) as early in the semester as possible. In the event of circumstances beyond our control, the evaluation scheme as detailed on the course website and on assignment handouts may require modification.

1.3 Collaboration & Plagiarism

Plagiarism is an academic offense of misrepresenting authorship. This can result in penalties up to expulsion. It is also possible to plagiarize *your own work*, e.g., by submitting work from another course without proper attribution. **When in doubt, attribute!**

You are expected to submit *your own work*. Assignments are individual tasks. This does not need to preclude forming an environment where you can be comfortable discussing **ideas** with your classmates. **When in doubt, some good rules to follow include:**

- fully understand every step of every solution you submit,
- only submit solution code that was written (not copy/pasted/modified, not ChatGPT'ed, etc.) by you, and
- never refer to another student's code — if at all possible, we recommend that you avoid looking at another classmates code.

McGill values academic integrity and students should take the time to fully understand the meaning and consequences of cheating, plagiarism and other academic offenses (as defined in the Code of Student Conduct and Disciplinary Procedures — see these two links).

> Computational plagiarism detection tools are employed as part of the evaluation procedure in ECSE 446/546. Students may only be notified of potential infractions at the end of the semester.

Additional policies governing academic issues which affect courses can be found in the Handbook on Student Rights and Responsibilities.

2 Implicit Path Tracing

You will implement a 5-sample/path MC estimator of the (recursive) global illumination equation:

$$L(\mathbf{x}, \omega_\mathbf{i}) = L_e(\mathbf{x}, \omega_\mathbf{i}) + \int_{\mathcal{A}} L(\text{cast}(\mathbf{x}, \omega_\mathbf{i}), -\omega_\mathbf{i}) \, f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) \, \max(\mathbf{n} \cdot \omega_\mathbf{i}, 0) \, d\omega_\mathbf{i} \, ,$$

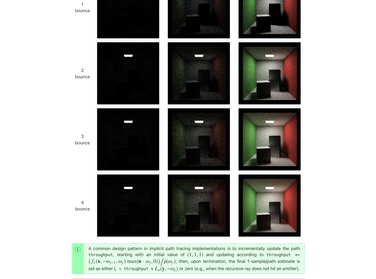
as

$$L(\mathbf{x}, \omega_\mathbf{i}) \approx L_e(\mathbf{x}, \omega_\mathbf{i}) + \frac{L(\text{cast}(\mathbf{x}, \omega_\mathbf{i}), -\omega_\mathbf{i}) \, f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) \, \max(\mathbf{n} \cdot \omega_\mathbf{i}, 0)}{p(\omega_\mathbf{i})} \, ,$$

where the $\omega_\mathbf{i}$ will be drawn using **BRDF importance sampling**, i.e. $\omega_\mathbf{i} \sim p(\omega) \propto f_r(\mathbf{x}, \omega_\mathbf{i}, \omega)$.

In the implicit path tracing scenario (with biased maximum path length), the incremental path construction terminates under only a few conditions:

1. ray$(\mathbf{x}, \omega_\mathbf{i})$ = ∅ lies on a light source, yielding a (non-recursive, deterministic) value for $L(\text{cast}(\mathbf{x}, \omega_\mathbf{i}), -\omega_\mathbf{i}) = L_e(\mathbf{y}, -\omega_\mathbf{i})$;
2. ray$(\mathbf{x}, \omega_\mathbf{i})$ exits the scene (i.e., does not intersect any object), or
3. you have reached the maximum num_bounces path length limit before hitting a light.



> A common design pattern in implicit path tracing implementations is to incrementally update the path throughput, starting with an initial value of $(1, 1, 1)$ and updating according to throughput *= $f_r(\mathbf{x}, -\omega_{i-1}, \omega_i) \, \max(\mathbf{n} \cdot \omega_i, 0) / p(\omega_i)$ (yes, upon termination, the final throughput estimate is set as either L = throughput * $L_e(\mathbf{y}, -\omega_i)$ or zero (e.g., when the recursive ray does not hit an emitter).

> **Deliverable 1 [10 points]**
>
> - Complete the implementation of your `AARenderer`'s `render()` function to support the implicit path tracing algorithm, as described above.

3 Explicit Path Tracing

You will next implement an explicit path tracer, separating out the direct (non-recursive) and indirect (recursive) lighting contributions as:

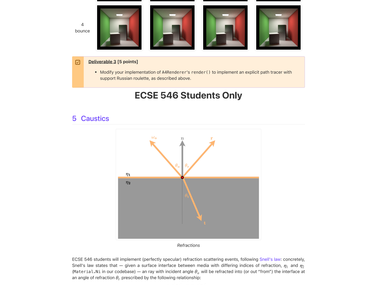
$$L(\mathbf{x}, \omega_\mathbf{i}) = L_e(\mathbf{x}, \omega_\mathbf{i}) + \underbrace{L_{\text{direct}}(\mathbf{x}, \omega_\mathbf{i})}_{\int_{\mathcal{A}} L(\text{cast}(\mathbf{x}, \omega_\mathbf{i}), -\omega_\mathbf{i}) \, f_r(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) \, \max(\mathbf{n} \cdot \omega_\mathbf{i}, 0) \, d\omega_\mathbf{i}} + \underbrace{L_{\text{indirect}}(\mathbf{x}, \omega_\mathbf{i})}_{\int_{\mathcal{A}} L_i(\mathbf{x}, \omega_\mathbf{i}, \omega_\mathbf{o}) \, d\omega_\mathbf{i}} \, ,$$

where the (non-recursive) direct illumination contribution $L_{\text{direct}}(\mathbf{x}, \omega_\mathbf{i})$ will be computed using **light importance sampling**, and the (recursive) indirect term $L_{\text{indirect}}(\mathbf{x}, \omega_\mathbf{i})$ is computed using **BRDF importance sampling**. Here, unlike the implicit case above, recursive path vertices should be rejected if they contribute (locally, i.e., at the current path vertex) to "direct illumination", adding a certain complexity to the algorithm logic. this is denoted mathematically, in part, by the exclusion of points $\mathbf{y} \notin \mathcal{A}_\mathbf{l}$ on the light from the integration domain of the recursive indirect contribution, i.e., where $L(\text{cast}(\mathbf{x}, \omega_\mathbf{i}), \omega_\mathbf{i}) = L_e(\mathbf{y}, -\omega_\mathbf{i}) \neq 0$.

Validation against the implicit path tracing results can be used to confirm that the explicit path tracer converges to the same result (albeit, usually with far fewer accumulated samples per pixel). As discussed in class, recall that — unlike the implicit case — each 1-sample estimate that your `Scene.render` explicit path tracer generates may potentially correspond to a contribution from more than one energy carrying light path.

In the explicit path tracing scenario (with biased maximum path length), the incremental path construction terminates under only a few conditions:

1. ray$(\mathbf{x}, \omega_\mathbf{i})$ = ∅ lies on a light source, terminating the path to avoid double counting direct illumination,
2. ray$(\mathbf{x}, \omega_\mathbf{i})$ exits the scene (i.e., does not intersect any object), or
3. you have reached the maximum num_bounces path length limit before hitting a light.



> **Deliverable 2 [10 points]**
>
> - Complete the implementation of your `AARenderer`'s `render()` function to support the explicit path tracing algorithm, as described above.

4 Russian Roulette

Terminating a path based on a fixed number of bounces — as we have prescribed thus far — introduces bias. **Russian roulette** is an unbiased, probabilistic termination method that relies on a user-defined termination probability q.

At every bounce in your explicit path tracer, you will evaluate the direct illumination contribution as usual, however — before continuing on with the recursive indirect contribution, you will using Russian roulette to decide whether to continue the path: first, draw a uniform variable $\xi \sim U[0, 1]$ and then, if $\xi < q$, you terminate the recursion — returning a 0 value for the (remaining recursive) indirect contribution at the current path vertex, otherwise, you continue the recursive process and adjust the remaining recursive indirect illumination estimate according to Russian roulette termination probability q, as:

$$L(\mathbf{x}', \omega_\mathbf{i}) = L_e(\mathbf{x}', \omega_\mathbf{i}) + L_{\text{direct}}(\mathbf{x}', \omega_\mathbf{i}) + \frac{L_{\text{indirect}}(\mathbf{x}', \omega_\mathbf{i})}{1 - q}$$

In **explicit path tracing with Russian roulette** (still retaining a biased maximum path length, strictly for comparison purposes with earlier implementations), the incremental path construction terminates under only a few conditions:

1. the recursive ray ray$(\mathbf{x}', \omega_\mathbf{i})$ = ∅ lies on a light source, terminating the path to avoid double counting direct illumination,
2. ray$(\mathbf{x}, \omega_\mathbf{i})$ exits the scene (i.e., does not intersect any object),
3. you have reached the maximum num_bounces path length limit before hitting a light, or
4. your path is terminated by the Russian roulette process, i.e., when $\xi < q$.

> **Deliverable 3 [5 points]**
>
> - Modify your implementation of AARenderer's `render()` to implement an explicit path tracer with support Russian roulette, as described above.

ECSE 546 Students Only

5 Caustics

Refractions

ECSE 546 students will implement (perfectly specular) refraction scattering events, following Snell's law: concretely, Snell's law states that — given a surface interface between media with differing indices of refraction, η_i and η_t (factor (a).A), in our coordinates — an ray with incident angle θ_i will be refracted into (or out "from") the interface at an angle of refraction θ_t prescribed by the following relationship:

$$\eta_i \sin\theta_i = \eta_t \sin\theta_t \, ,$$

and so $\sin\theta_t = \frac{\eta_i}{\eta_t} \sin\theta_i$. During refraction, we need to ensure that $0 \leq \sin\theta_t \leq 1$, otherwise Total Internal Reflection (TIR) occurs; in these configurations, we will treat the (recursive) TIR reflection event, as described below.

First however, assuming we are not in a TIR setting, the refracted ray direction $\mathbf{t}$ is defined as:

$$\mathbf{t} = -\frac{\eta_i}{\eta_t} \mathbf{i} + \left(\frac{\eta_i}{\eta_t} \cos\theta_i - \sqrt{1 - \sin^2\theta_t} \right) \mathbf{n} \, .$$

If there is TIR however, the ray cannot refract across the interface and, instead, has to reflect about the normal at an angle $\theta_r = \theta_i$ (i.e., perfect specular reflection).

> You can assume that the default medium is air with $\eta = 1.0$. Since all of the medium information is stored on the surface materials, one way to determine if you are exiting or entering an object (i.e., a different medium) is by testing the `hit_data.is_backfacing` flag.

> **Deliverable 4 [10 points]**
>
> - Modify your AARenderer's `render()` implicit path tracer implementation to support caustics, by implementing refraction and TIR.

6 You're Done!

Congratulations, you've completed the 4th and final assignment. Review the submission procedures and guidelines at the start of this handout before submitting your assignment solution.