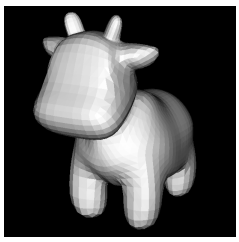


Homework 3: Pathtracer

Table of Contents

Introduction

This project was about implementing raytraced rendering of scenes. We started by getting the camera to generate random rays out into the scene through specific pixels, which we then checked to see if they intersected anything in the scene. In particular, we had to implement intersection finding for the triangle and sphere primitives. If we find an intersection, then we can add the color we think the material should be (our *estimate of the global illumination* of that point) to our average color for that pixel, otherwise we don't add anything to the color. Then, we optimized our intersection algorithm by implementing a bounding volume hierarchy (BVH), which consisted of putting our primitives in groups and then putting them inside bounding boxes; if a ray misses the bounding box, it will miss all the primitives in the scene, allowing us to cut down substantially on intersection checks.

Then, we implemented the real ray tracing part: we followed the path of rays into the scene, and updated the light they received if we either hit a light source (which just emitted its light directly into the camera), or hit an object which had light cast directly onto it. This is called direct illumination. Handling objects with light cast onto them was a little difficult, and involved two different methods: *uniform hemisphere sampling* and *importance sampling by lights*, which we discuss in more detail in Task 3. We then upgraded this ray tracing to handle multiple bounces of light in the scene (global illumination), which improved the visual quality and, you know, lit up the ceiling. Finally, we went back and optimized our pixel sampling; sometimes, we don't need to sample that many times per pixel because the result we get back is almost exactly the same every time. So, we can adapt our sampling strategy to avoid doing a lot of extra work in places where it doesn't make much of a difference.

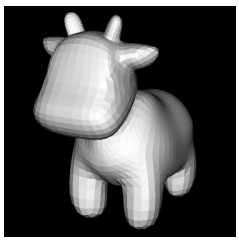
Debugging woes

We ran into a pretty large number of bugs in this project. When implementing task 3, we ran into an unexplainable bug where at most 10–20 rays would hit a light and update the corresponding pixel, even in an empty room, leaving the room basically completely dark. We tried every fix under the sun until we realized that the bug was caused by broken task 2 code. The issue was that bounding box intersections didn't work properly when the ray started *inside* the bounding box itself. This was fine when we were just rendering the scene to a file, because we always start the camera outside of any bounding boxes. However, when rays needed to suddenly bounce off of objects in the scene in task 3, everything just broke. Only when rays magically originated outside of the bounding box of the object they hit (through cursed floating point arithmetic) were they getting a chance to intersect things properly at all, which (along with the low probability of pointing in the right direction in the first place when it came to hemisphere sampling) was what caused the 10–20 ray phenomenon earlier.

There was another medium-sized bug that we didn't catch for quite some time, because I (Aditya) was not looking at the output images very carefully. After finishing task 4 (yes, it took me that long to realize), I finally realized that there were no shadows in any of the importance sampling renders. The problem was simple: the code never checked that light could actually reach the point that it intersected, so no shadow rays were ever cast and the light just teleported through objects. An easy fix, but one that took a baffling amount of time.

Also after finishing task 4, we noticed that our outputs were kind of grainy when compared to staff renders: not

overall. But this wasn't the whole story: the lighting was still slightly grainy after this too. Honestly, we're not sure we even fully fixed this, but our best guess was that this was caused by using a different (smaller) epsilon in our bounding box intersection than provided by the code, and this was leading to more rays disappearing... somehow? Changing the epsilon to `EPS_F` appeared to help, but we're not 100% sure why. After that point, we weren't sure if we were still detecting more graininess than staff, or we were just seeing things that weren't there, so we moved on.



When implementing Task 5, we were really mystified by somehow indexing fractionally into our array... until we realized we accidentally shadowed the `x` variable `;-;`. For task 2, we ran into some weird bugs based on not pruning the nodes properly, and this was because of our un-implemented `Triangle->has_intersection()` algorithm. Once we fixed this, we got the speedup we desired.

Reflection

Overall, this project was really, really fun. The feeling of getting everything to work right and being rewarded with a beautiful render was really amazing, no matter how many times it happened. More seriously, applying all of the lighting and raytracing concepts we learned in lecture was both enlightening (no pun intended) and enjoyable. I (Aditya) feel like I learned a lot of new information from this project, and it solidified my understanding in other places. When it comes to our workflow, well, we probably should have started in earnest a little sooner. But working together was pretty equitable overall; we didn't have much time to meet up, so we generally split tasks up and finished them individually, helping each other through bugs when they came up.

I (Andrew) feel like this project really helped me learn a lot about ray tracing since I've seen a lot of videos and lectures on the subject, but I haven't tried to actually implement it. The most satisfying moment for the entire project was seeing the BVH implementation actually work. When we rendered the cow in 0.14 seconds, I was really happy. I was even happier to see Max Planck get rendered in 0.13 seconds... until Max turned into swiss cheese because of incorrect pruning logic. Once we fixed that though... rendering Max in 0.17 seconds correctly was super satisfying.

2024-04-12