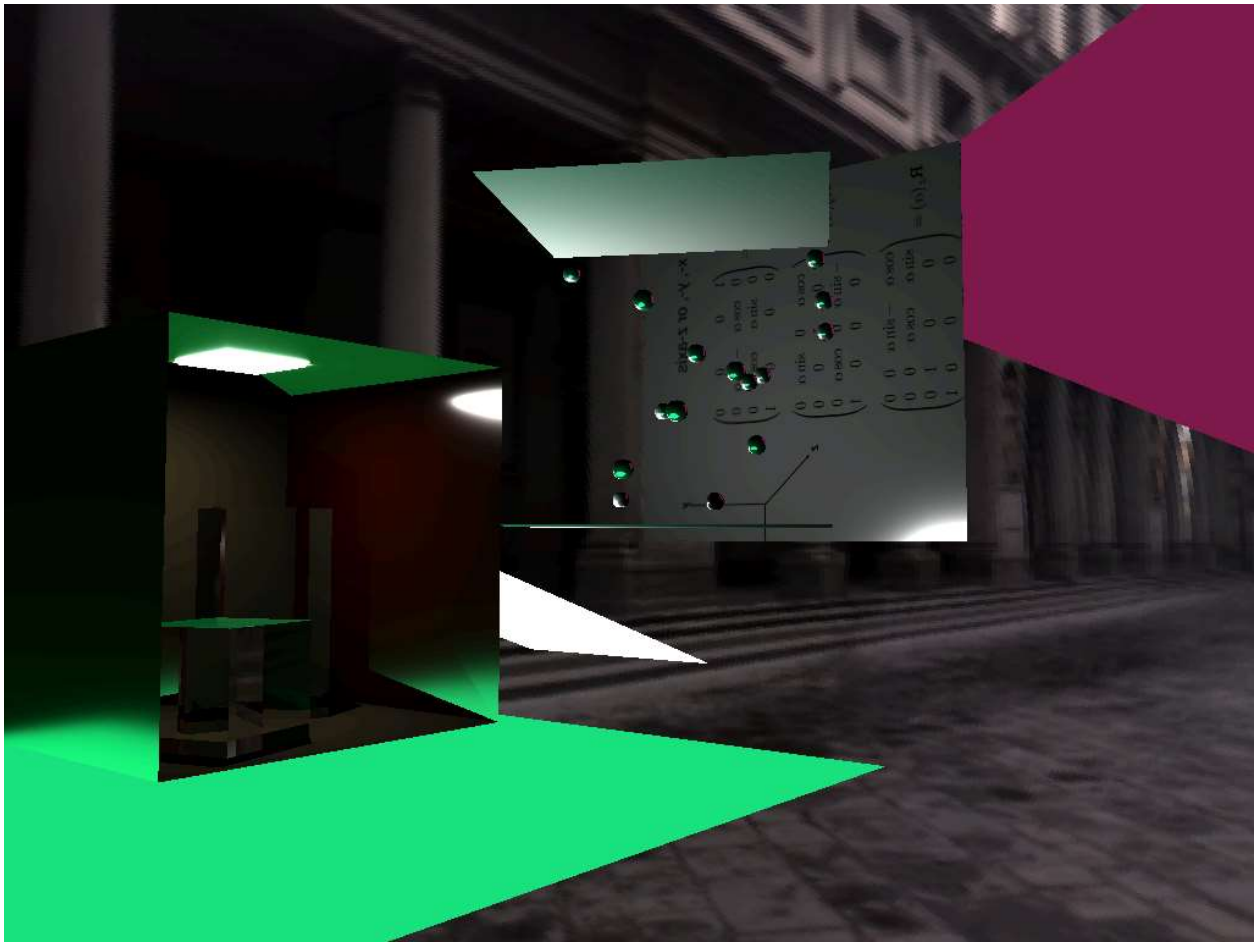
Spring 2024 CS488 Final Project



by Kevin Lee (k327lee, 20887836)

August 5th, 2024

Fresnel refraction	SAH BVH	Microfacet model
Texture Mipmaps	Area lights	Deformables (0% done)
Rigid body shape matching (80% done)	Light dispersion/spectral rendering (50% done)	Fractures (0% done)

While I was not able to get a good amount of my objectives completed, I feel that this project has still managed to achieve a level of depth that I am proud of and am happy to share with you!
For my SAH BVH, here are the performance gains in some test scenes:

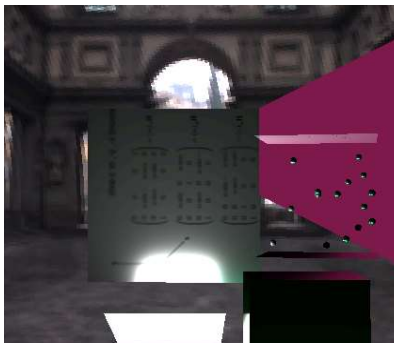
- Project scene (1668 triangles) with Fresnel glass refractions: 1303 -> 1013 ms/frame (23% boost)
- Project scene (1668 triangles) without Fresnel glass refractions: 833 -> 617 ms/frame (26% boost)
- testobj-glass.obj (10846 triangles) with Fresnel glass refractions: 2223 -> 1960 ms/frame (12% boost)
- testobj-glass.obj (10846 triangles) without Fresnel glass refractions: 585 -> 495 ms/frame (16% boost)
- sponza.obj (66454 triangles): 1848 -> 718 msec/frame (69% boost)

Clearly, the performance increase depends on not only the number of triangles, but mostly the arrangement and density of these triangles, as well as what materials and shading methods they require.
Building the BVH with Surface Area Heuristic can still be relatively fast, though it also varies by scene, at around 79 msec for testobj-glass.obj and around 470 msec for sponza.obj. Most other test scenes have negligible construction time.

For light dispersion, I think I grasped the main concept of how spectral rendering should work, but the actual implementation was far too confusing for me. I could wrap my head around a multichromatic ray, with a fixed number of sampled wavelengths and levels, splitting into monochromatic rays when hitting a dispersive material.
Each ray would be affected by the material differently based on the wavelength it held, and they would continue from there with traditional raytracing until hitting a diffuse surface or light source and adding that surface's corresponding monochrome wavelength component.
The intuitive yet intriguing theory of this concept led me to grow curious about it and pushed me to add it as an objective. However, when actually implementing the infrastructure for it, I got lost with the conversions between RGB and spectrum.
I attempted to follow the CIE color matching function implementations from PBR, as well as the method outlined in Smits' *An RGB to Spectrum Conversion for Reflectances* (2000), but I got overwhelmed very quickly while trying to fit the two methods together and had to move on. I am still intrigued by this phenomenon, but it was definitely too much effort for a result that would not be as noticeable as other objectives, leading me to drop it altogether. The remains of my attempt live on in my code at lines 434 - 637 of cs488.h.

I chose mipmapping with a raytracer as a bit of a challenge, since it is traditionally easy for rasterizers but less obvious for raytracing. I was hoping it would also help with the look of my final scene. I mainly followed Igehy's *Tracing Ray Differentials* paper for this, which provides a method involving the calculation of differentials at each ray hit with respect to an initial change in pixel x and y coordinates when shooting eye rays. Specifically, by keeping track of the derivatives of the position, direction, and normals, it is possible to estimate the change in texture coordinates with respect to the initial x, y offset. This gives an approximation for the pixel footprint of a single ray, with formulae that also enable tracing of the ray differentials through specular reflection and refraction. The texture footprint is then used to determine a mip level/level of detail, and I used trilinear resampling to smooth out the mip levels and generate an average value for the return texture (see Fig 1, 2 for results). I tried looking into bringing this

further to anisotropic filtering, but that did not happen sadly (I tried one of the methods in the paper).

a) 

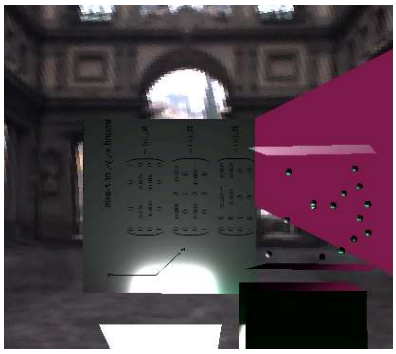
b) 

Fig 1. From afar, we can see there is evident blurring in a), and aliasing in b).

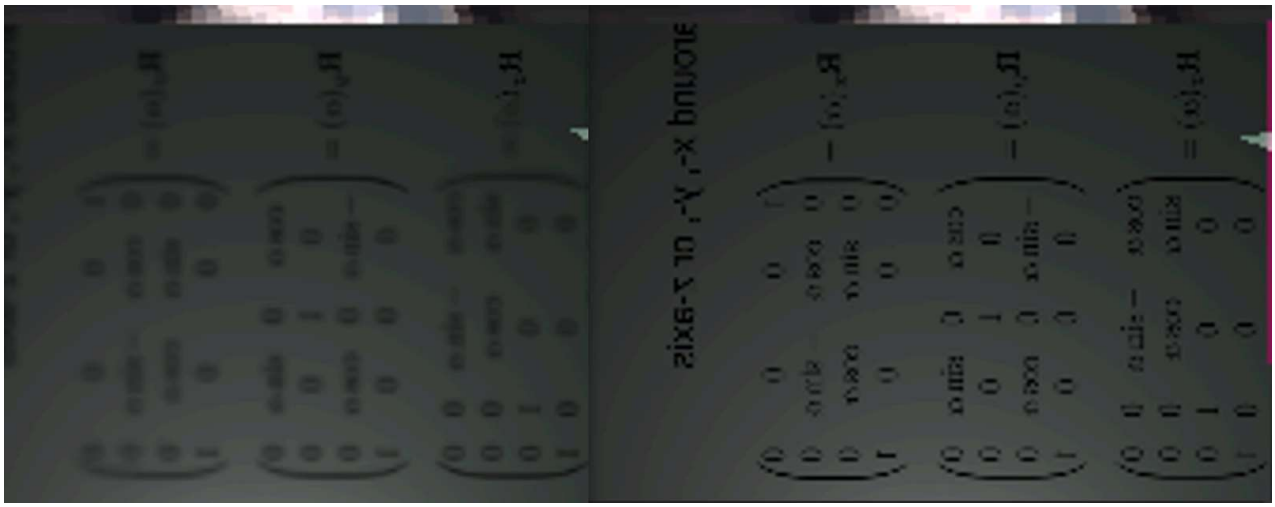


Fig 2. Here are a) (left) and b) (right) zoomed in to the text. While there is room to improve on the mipmapping, we can clearly see more smooth outlines of the characters instead of blocky.

I also attempted rigid body shape matching. The idea was to use the particle system to define groups of particles which would act as sample points to base the shape matching off of, then encase these particles with a triangle mesh whose vertices would be transformed by the rigid transformation found after updating the sample particles. I implemented the collision with triangle meshes by using the position delta vector and intersecting it with the scene BVH, then handling collisions as if the mesh were static. I split the shape matching portion into two parts: the first being computing the next positions of the sample particles and correcting them based on relative position constraints to other particles, then the second being the computation of the optimal rotation matrix. However, once I began implementing the shape matching portion according to Müller et al.'s 2020 paper on Rigid Body Deformations, I found that the triangle mesh collision no longer worked. When it came to computing the rotation matrix, I was following the method in Müller et al.'s 2005 paper *Meshless Deformations Based on Shape Matching*, which we also briefly covered in class, but yet again it did not work as intended when it came to applying the matrix to transform points. This was quite the disappointment, as it essentially closed off my ability to complete the other two body dynamic objectives, but I was close.

Finally, I implemented both area lights and a microfacet model. At first, I was unsure whether to do area lights using Monte Carlo Integration or trying to implement the analytic approximation with Linearly Transformed Cosines, a real-time solution created by Heitz et al. in 2016 with Unity. At first I did not understand the LTC method, and so I went with MC, and achieved some okay results. This was based on the lecture slides, with random sampling from the area light's mesh (Fig 3). However, it came to me like a dream, and I finally understood the relation between BRDFs, area lights, spherical distributions, and spherical polygons. The solution is not the most clean, as it requires an entire table full of a precomputed BRDF approximation for a Shlick-GGX-based Cook-Torrance microfacet model. This table acted as a texture using the angle between the view direction and halfway vector (between viewDir and normal), and the microfacet roughness parameter as texture coordinates. Given that these would be rounded (after some scaling), I also opted to linearly interpolate between adjacent values in the table, which helped with anti-aliasing (Fig 4). While I also implemented the Cook-Torrance model on its own to be used without the LTC area lights, the result from using LTCs is simply gorgeous so I chose to make that the default in my project (See Fig 5). I tried working some shadows in (since this LTC solution does not account for them) by checking visibility from each hit to the triangles of the area light meshes, and skip if any were blocked, which also acted as a small optimization.

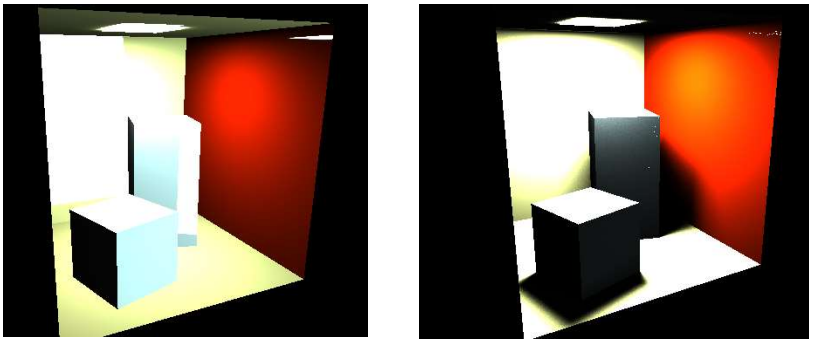


Fig. 3 left LTC, right MC. Notice the white spots in the MC example in the places where the specular reflection of the light should have been. Note that there is another upright rectangular light behind the camera in these figures.

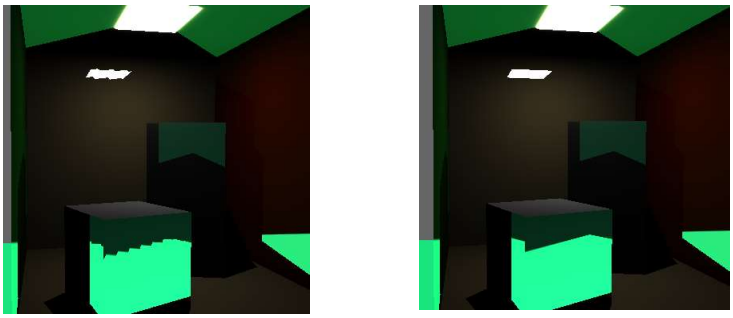


Fig 4 left squiggly (bad), right smooth (good)

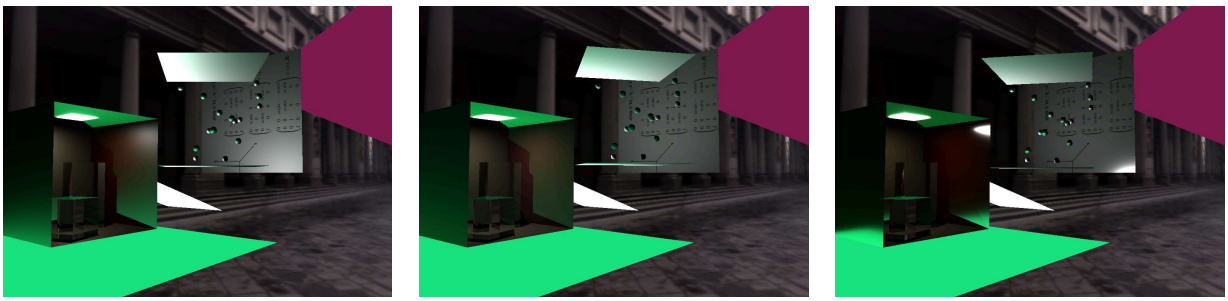


Fig. 5: Varying levels of roughness affecting the specular and diffuse components of surfaces.

Addendum: Running the Program

Unzip the contents into the cs488 repo root directory. You should have a “cornellbox-custom.obj” file in the media folder now. If not, extract it separately from the zip. There are also additional files for the LTC area lights, “ltc.h” and “ltc.inc”. Make sure these are in /src after unzipping.
Finally, you may enable environment mapping with ‘-e’ as a command line option.

Addendum: Controls for project executable

- M: toggle mipmap
- N: toggle glass Fresnel
- T: toggle MC/LTC area lights (would not recommend)
- 5/6: Decrease/Increase global roughness
- K: toggle LTC texture antialiasing

References

- Sun, Y., Fracchia, F.D., Drew, M.S. (2000). *RENDERING LIGHT DISPERSION WITH A COMPOSITE SPECTRAL MODEL*. International Conference on Color in Graphics and Image Processing - CGIP'2000
- Radziszewski, M. et al. (2009) *An Improved Technique for Full Spectral Rendering*. Journal of WSCG, Vol. 17, no. 1-3, p. 9-16.
- Akenine-M¨oller, T., Nilsson, J., Andersson, M., Barr´e-Brisebois, C., Toth, R, Karras, T. (2019). *Texture Level of Detail Strategies for Real-Time Ray Tracing*. Chapter 20 in Ray Tracing Gems.
- Ihegy, H. (1999). *Tracing Ray Differentials*. SIGGRAPH '99, pp.179-186. doi 10.1145/311535.311555.
- Smits, B. (1999). *An RGB-to-Spectrum Conversion for Reflectances*. Journal of Graphics Tools, Vol. 4, Issue 4. doi 10.1080/10867651.1999.10487511.
- Müller, M. et al. (2020). *Detailed Rigid Body Simulation with Extended Position Based Dynamics*. Computer Graphics Forum, Vol. 39, Issue 8 pp.101-112. doi 10.1111/cgf.14105.
- Ng, R. (2005). *Microfacet Models*. Stanford University. https://graphics.stanford.edu/courses/cs348b-05/lectures/lecture11/reflection_ii.pdf
- Müller, M. et al. (2005). *Meshless Deformations Based on Shape Matching*. ACM Transactions on Graphics (TOG), Vol. 24, Issue 3 pp.471-478. doi 10.1145/1073204.1073216.
- Heitz et al. (2016). *Real-time polygonal-light shading with linearly transformed cosines*. ACM Transactions on Graphics, Vol. 35, Issue 4, Article 41, pp.1-8. doi 10.1145/2897824.2925895.