A5 Project Proposal

Title: Ray Traced Tetris

Name: Spencer Van Leeuwen
Student ID: 20412199
User ID: srvanlee

Final Project:

Purpose :

The purpose of my project is to build a real-time Whitted raytracer. As objects in the scene move, the raytracer will be able to re-render the scene in real-time.

Statement :

As hardware gets better, more and more ray tracing researchers have become interested in the idea of trying to build real-time raytracers, since there are things that can be done much better using ray tracing than rasterization (the most simple examples are reflection and refraction).

[WMG+09] is an article overviewing different methods of ray tracing animated scenes, and is where I found most of the other papers that I am using to build my project. The foundation of this project will be quickly partitioning the objects in the scene and tracing multiple rays. This should reduce the rendering time enough to be able to create an animated image.

For my final scene, I hoping to make a raytraced tetris game (if I can't get enough FPS, I might come up with another idea for the revised proposal). The pieces will have different materials, instead of just different colours. I also might experiment with putting other primitives in the scene, like a mirror behind the game. It would also be interesting to make all of the pieces translucent and having an object behind the game to demonstrate refraction. The border of the game and the surface that the game sits on (probably just a plane), will have an interesting texture of some sort.

This project is very interesting because it is based on research that requires very innovative solutions to work. Real-time raytracers don't have access to special purpose hardware, unlike rasterization which has highly parallelized GPUs. Plus, finding ways to make something run faster is just inherently interesting to any computer scientist.

I will learn how to add basic improvements to a raytracer, like reflection, refraction, antialiasing, and texturing. I will also learn about fast building and traversal of spacial partitioning data structures.

Technical Outline :

Bounding Interval Hierarchy

This data structure was recommended in [WMG+09] for being fast and relatively easy to implement. The BIH is introduced in [WK06], and was significantly faster than any previous data structure.

When constructing a BIH, it is like a hybrid of a kd-tree and a Bounding Volume Hierarchy. First, the global bounding box is divided into a uniform grid. At each step, we choose the longest axis with respect to the uniform bounding box and get the median along that axis. Then we partition the objects in the bounding box to below and above the median, by checking which side contains more of the object's bounding box. Then for the objects below the median, we take the plane that bisects the median and increase its location on the chosen axis such that all of the objects' bounding boxes are completely below the plane. We can induce a new bounding box using this plane. We can do a similar thing for objects above the median. Again, the paper [WK06] explains this much better.

To traverse the BIH, we simply treat it as a BVH, where the bounding volumes are induced by the planes we found during the construction.

To demonstrate that this works, I will time the rendering of a few different scenes with and without the data structure.

Reflection Rays

I will just use a simple reflection model, where $\theta_{in} = \theta_{out}$, with attenuation per reflection. I might reference [PH10] to figure out a nice way to do it.

Refraction Rays

I am just going to use a basic refraction model. I will use [PH10] to figure this out.

Antialiasing

I am going to use Stratified sampling, as described in [PH10]. Basically, I am going to split each pixel into a grid, then take a random point in each grid box and sample there.

Ray Packets

For ray-packet tracing, the basic idea is to group a bunch of similar rays together at test if they intersect a primitive all at once. [BEL$^+$07] shows that it is possible to use ray packets for secondary rays, even though they don't share an origin. It suggests using the BVH traversal method for ray packets described in [WBS07]. When testing against a particular primitive, [BEL$^+$07] suggests using interval arithmetic, as described in [BWS06].

The traversal as described in [WBS07] has a few key components. First, it tests if the first ray in the packet hits the bounding box. If it does, we recurse the children. Otherwise, we perform an "all miss" test, which conservatively tests if all of the rays in the packet miss the bounding box. The paper actually shows that in the majority of cases, one of the tests succeed and saves a ton of computation.

Again, I plan on using a table with times to demonstrate that this works.

Add Tetris Primitive

I am going to add the tetris game from A1 as a lua primitive. This will allow me to place the tetris game where I want in the scene. At first, this will just place the border in the scene, but as the game progresses, blocks will be added as children. To make sure this works, I will have to update the game a few times before drawing, just so some blocks are visible.

Animate Tetris using Qt

I am going to set up Qt so that after rendering an image, it displays on the canvas. This is tied in with the previous objective, since I don't actually have to support the blocks moving until this point.

Manage user input using Qt

I will have a similar user input to A1, only without the moving camera. When the user does something, the raytracer should redraw the scene with the block moved in the right way.

Add Texturing

I will probably just use 2D texture maps, since most of the surfaces in my final scene will be flat.

Final Scene

(Note: same text as in Statement) For my final scene, I hoping to make a raytraced tetris game (if I can't get enough FPS, I might come up with another idea for the revised proposal). The pieces will have different materials, instead of just different colours. I also might experiment with putting other primitives in the scene, like a mirror behind the game. It would also be interesting to make all of the pieces translucent and having an object behind the game to demonstrate refraction. The border of the game and the surface that the game sits on (probably just a plane), will have an interesting texture of some sort.

Bibliography :

References

[BEL⁺07] Solomon Boulos, Dave Edwards, J. Dylan Lacewell, Joe Kniss, Jan Kautz, Peter Shirley, and Ingo Wald. Packet-based whitted and distribution ray tracing. In *Proceedings of Graphics Interface 2007*, GI '07, pages 177–184, New York, NY, USA, 2007. ACM.

[BWS06] Solomon Boulos, Ingo Wald, and Peter Shirley. Geometric and arithmetic culling methods for entire ray packets. *Tech. Rep. UUCS-06-010, SCI Institute, University of Utah*, 2006.

[PH10] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation (Second Edition)*, pages 346–358,428–446. Morgan Kaufmann, 2010.

[WBS07] Ingo Wald, Solomon Boulos, and Peter Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 26(1), January 2007.

[WK06] Carsten Wchter and Alexander Keller. Instant ray tracing: The bounding interval hierarchy. In *IN RENDERING TECHNIQUES 2006 PROCEEDINGS OF THE 17TH EUROGRAPHICS SYMPOSIUM ON RENDERING*, pages 139–149, 2006.

[WMG⁺09] Ingo Wald, William R. Mark, Johannes Gnther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G. Parker, and Peter Shirley. State of the art in ray tracing animated scenes. *Computer Graphics Forum*, 28(6):1691–1722, 2009.

Objectives:

Full UserID:____________________________ **Student ID:**____________________________

___ 1: Bounding Interval Hierarchy

___ 2: Reflection Rays

___ 3: Refraction Rays

___ 4: Antialiasing

___ 5: Ray Packets

___ 6: Add Tetris Primitive

___ 7: Animate Tetris using Qt

___ 8: Manage user input using Qt

___ 9: Add Texturing

___ 10: Final Scene

A4 extra objective: None