CS 6630 Final Project Proposal

Nate Morrical & Justin Jensen

Basic Info

Project Title: RayTracker

- Project repository: https://github.com/n8vm/RayTracker

Team:

- Nate Morrical
    - u1137457@utah.edu, UID: u1137457
- Justin Jensen
    - u1140481@utah.edu, UID: u1140481

Background and Motivation

Both of us are in Cem Yuksel's introduction to ray tracing class, and are interested in visualizing what's going on "under the hood" of our ray tracers. In a typical ray tracer implementation, a series of rays are tracked and combined for each pixel on the screen. Multiple effects, like anti aliasing and depth of field can be created by altering how many and where these rays are generated within a pixel. As each of these ray bounce off objects, they emit a color based on a set of light and material properties. Each of these bounces can generate additional reflective and refractive secondary rays, which can in turn bounce off additional objects, collecting color as they go.

As rays travel through a scene, they must test for intersection with a variety of primitives, which can be expensive. As a result, optimizing the ray tracing process is a fairly heavily researched topic. A potential way to learn optimization insights is to study the intermediate data created during the ray traversal process. Unfortunately, the amount of data created by ray tracers is very large, and can be hard to manage and learn from.

However, If patterns can be spotted within this data, it may be possible to improve the performance of ray tracing. Rays intersecting the same geometry can be processed together for better memory coalescing. Rays that are found to have insignificant impact can be terminated early. Photon mapping has the potential to converge faster if light to camera ray paths are processed first. And scene graphs could be filtered for improved primitive intersection times.

Additionally, providing an intuitive and interactive visualization for the ray tracing process may help improve our and other students understanding of ray tracing. As raster based graphics has improved, people may be wondering what the advantages are in moving to ray tracing. Some material effects, like refraction of light, can seem counterintuitive, but could be explained with this tool. As rays bounce, subtle changes in the image could be highlighted

Project Objectives

Primary questions we would like to answer

- What rays are intersecting what geometry
- What rays are creating bottlenecks

- Which rays are most important to the final image
- Performance Cost vs Image Quality metrics
- How does texture filtering affect an image
- How does anti aliasing affect an image

We would like to accomplish

- Building a tool with high quality interaction

Benefits of this project include

- Better understanding of the ray tracing process
- Teaching the inner workings of ray traversal to students just starting to learn about computer graphics.

Data

Primary data is coming from our ray tracer implementations from Cem Yuksel's Introduction to Ray Tracing class: http://graphics.cs.utah.edu/courses/cs6620/fall2017/

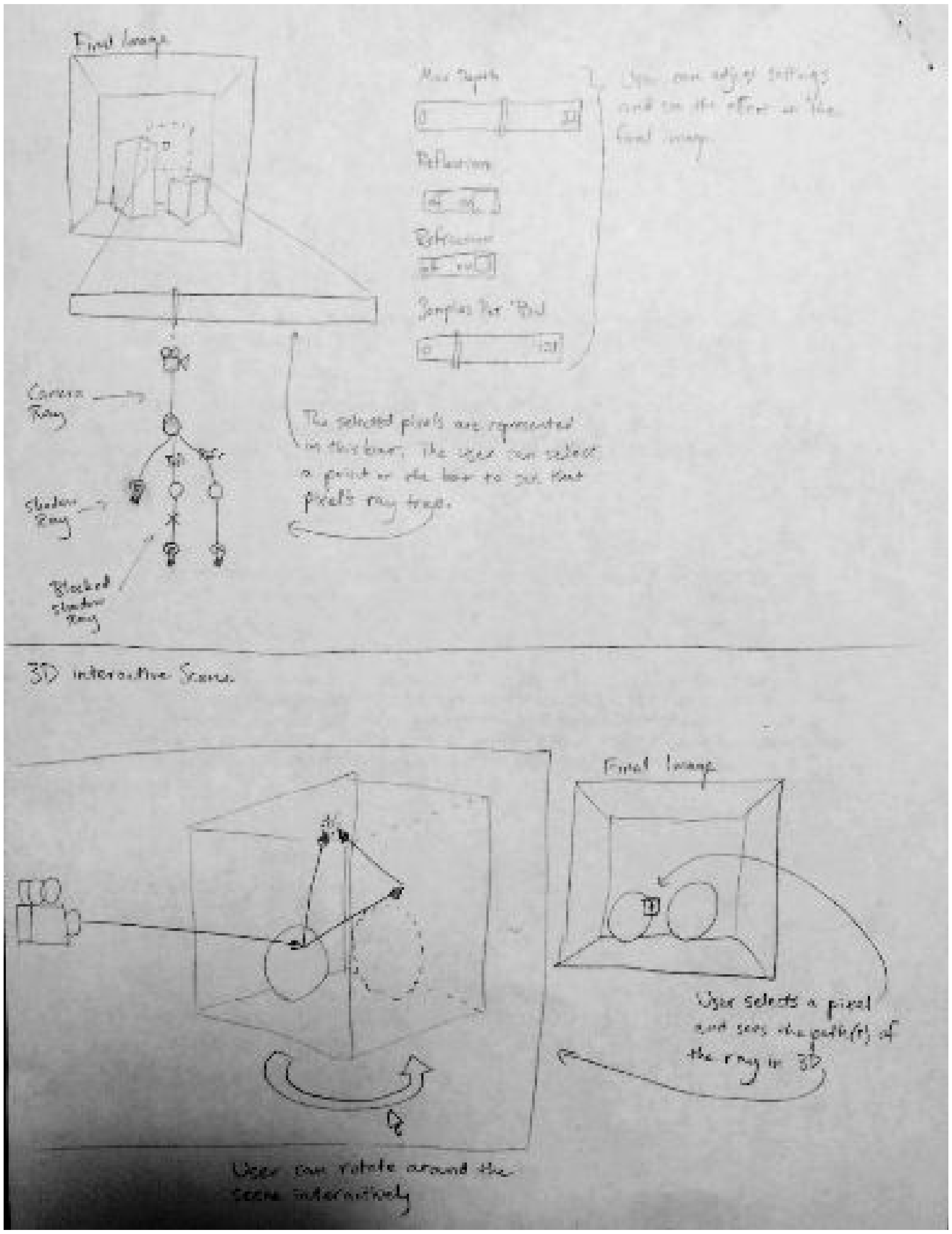
As scenes are generated, we would export intermediate results for visualization. For each pixel, we will generate a ray tree structure. Links represent rays and nodes represent intersections. Each link will store its ray direction, length, depth, type (camera, shadow, reflection, or refraction), and whether the ray was blocked if it was a shadow ray. Each node will store the object that was intersected, the position in space, and the pixel color contribution.

Data Processing

We will be generating all of the data with our ray tracer, so there should not be substantial data cleanup; we can generate exactly what data we need and organize it how we need it.

Visualization Design

Sketch 1:


Final Image
Max Depth
Reflection
Refraction
Samples Per Pixel
User can adjust settings and see the effect on the final image.
Camera Ray
Refl
Refr
Shadow Ray
Blocked shadow Ray
The selected pixels are represented in this bar. The user can select a point on the bar to see that pixel's ray tree.
3D interactive Scene
Final Image
User selects a pixel and sees the path(s) of the ray in 3D
User can rotate around the scene interactively

Sketch 2

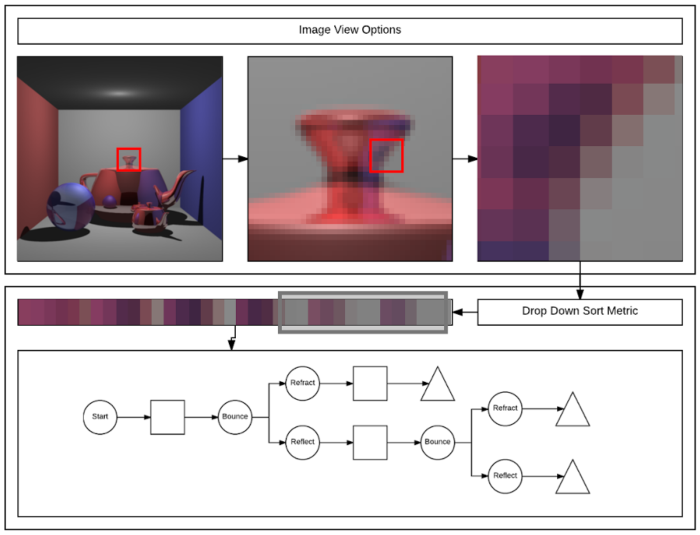
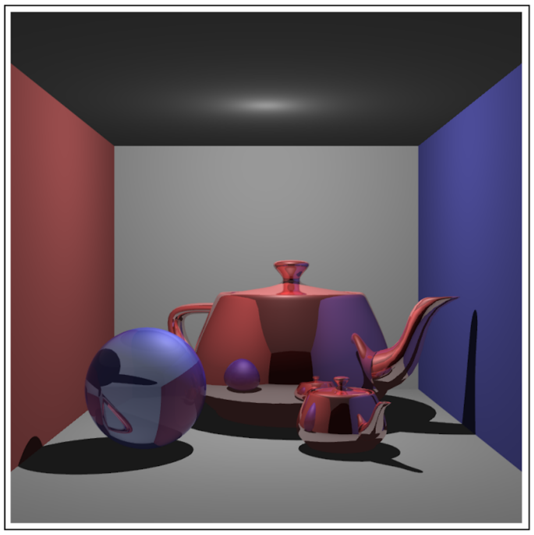
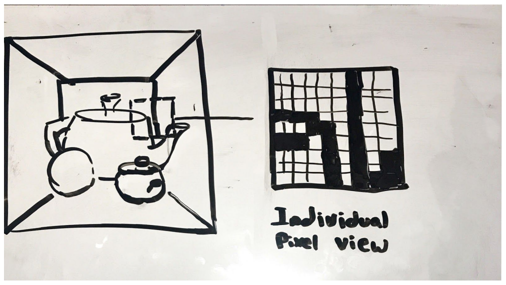
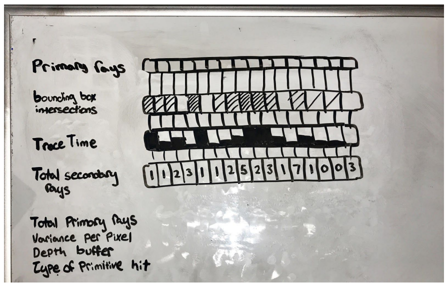


Image View Options
Drop Down Sort Metric
Start
Bounce
Refract
Reflect
Bounce
Refract
Reflect




Individual
Pixel view


Primary rays
bounding box intersections
Trace Time
Total secondary rays
1 1 2 3 1 1 2 5 2 3 1 7 1 0 0 3
Total Primary Rays
Variance per Pixel
Depth buffer
Type of Primitive hit

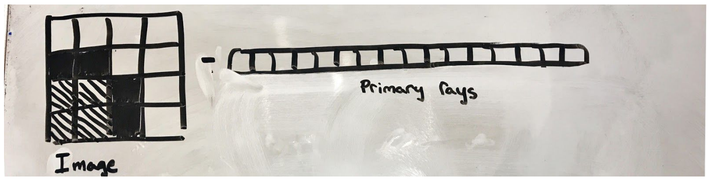


Image
Primary rays

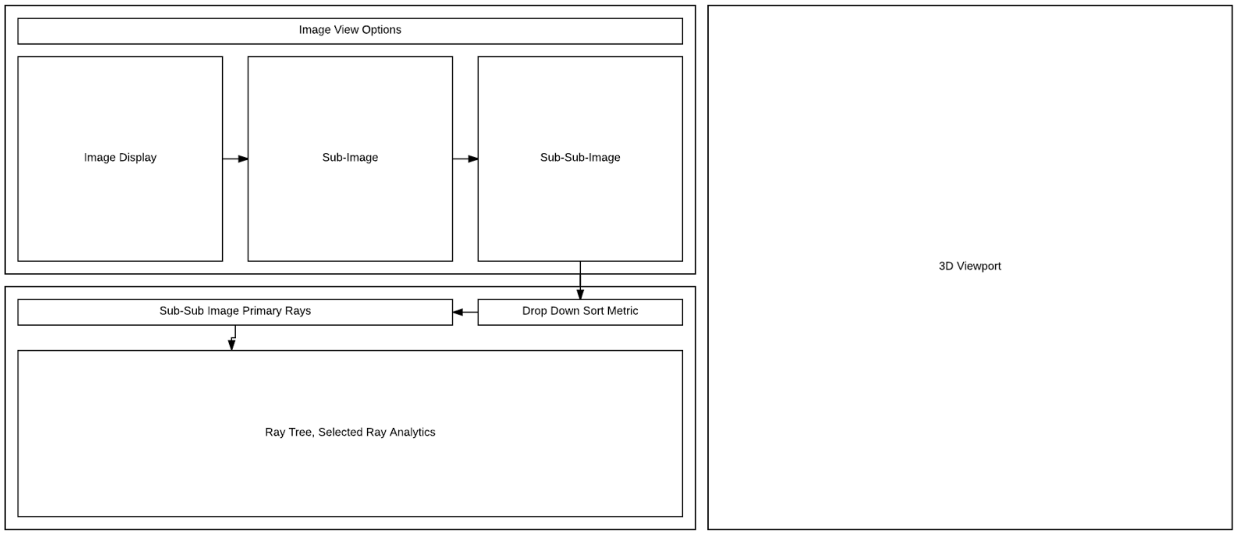


Image View Options
Image Display
Sub-Image
Sub-Sub-Image
Sub-Sub Image Primary Rays
Drop Down Sort Metric
Ray Tree, Selected Ray Analytics
3D Viewport

Must-Have Features

List the features without which you would consider your project to be a failure.

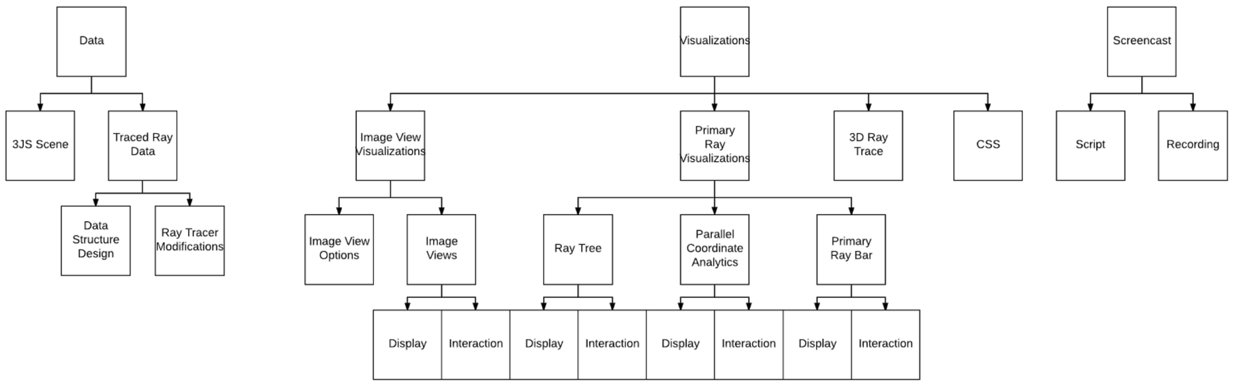
- Selecting groups of pixels in the rendered image to visualize the types of rays that passed through those pixels.
- Visualizing the scene traversal of individual rays as a tree.
- Visualizing which rays hit which objects, eg through parallel coordinates
- Visualizing distance traveled per ray
- Changing rendering parameters and visualizing the effect on the final image

Optional Features

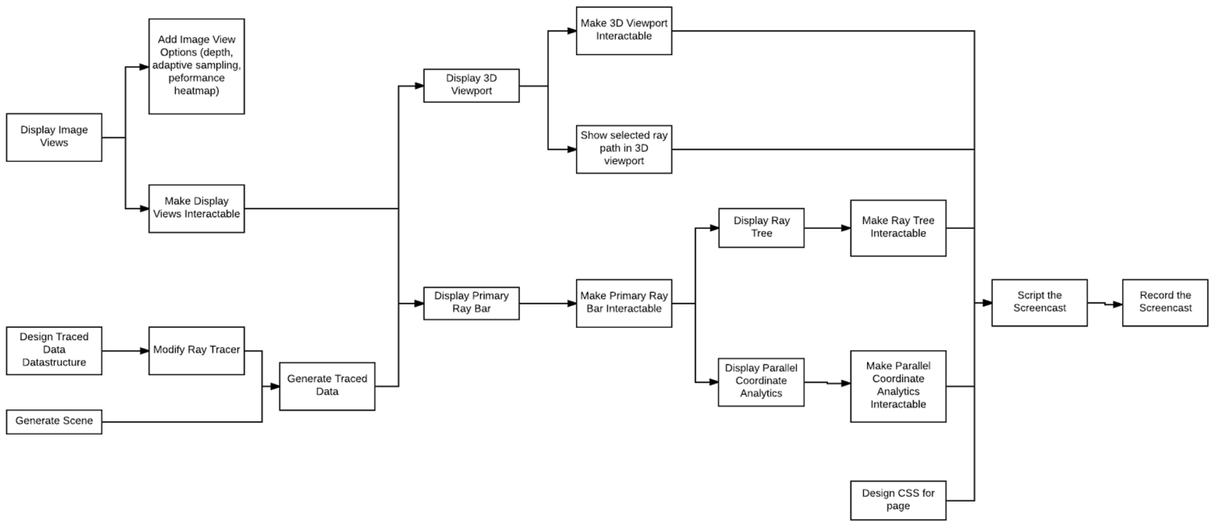
List the features which you consider to be nice to have, but not critical.

- 3D interactive visualization of the scene
- In 3D, visualize the paths of one or more rays through the scene
- The ability to bookmark different rendering parameters and compare the resulting images quickly

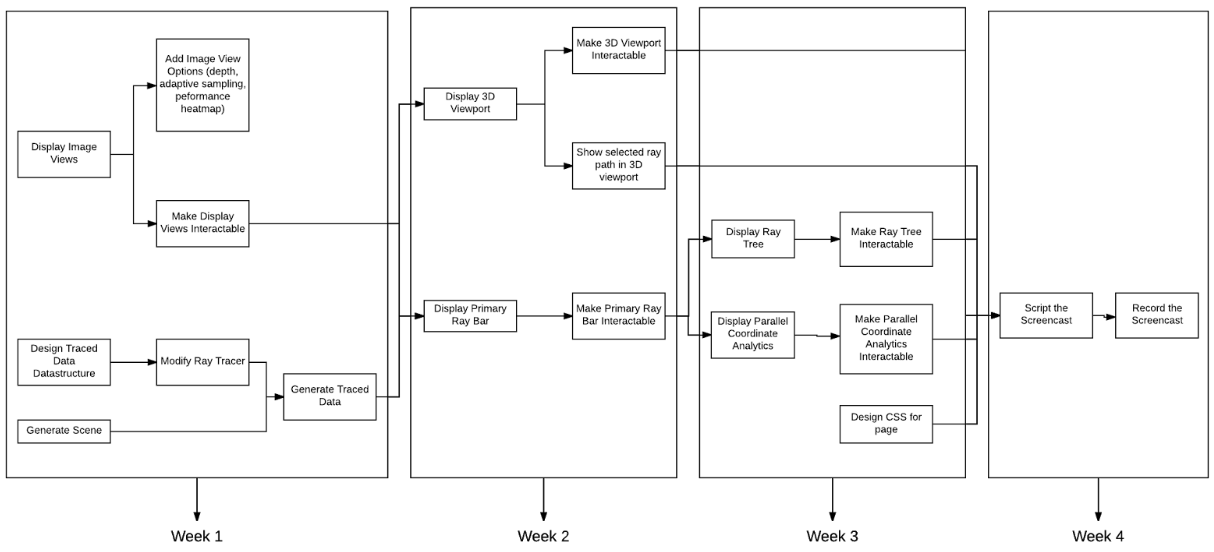
Work Breakdown Structure



Project Network Diagram


Display Image Views
Add Image View Options (depth, adaptive sampling, peformance heatmap)
Make Display Views Interactable
Design Traced Data Datastructure
Modify Ray Tracer
Generate Scene
Generate Traced Data
Display 3D Viewport
Make 3D Viewport Interactable
Show selected ray path in 3D viewport
Display Primary Ray Bar
Make Primary Ray Bar Interactable
Display Ray Tree
Make Ray Tree Interactable
Display Parallel Coordinate Analytics
Make Parallel Coordinate Analytics Interactable
Design CSS for page
Script the Screencast
Record the Screencast

Project Schedule


Display Image Views
Add Image View Options (depth, adaptive sampling, peformance heatmap)
Make Display Views Interactable
Design Traced Data Datastructure
Modify Ray Tracer
Generate Scene
Generate Traced Data
Display 3D Viewport
Make 3D Viewport Interactable
Show selected ray path in 3D viewport
Display Primary Ray Bar
Make Primary Ray Bar Interactable
Display Ray Tree
Make Ray Tree Interactable
Display Parallel Coordinate Analytics
Make Parallel Coordinate Analytics Interactable
Design CSS for page
Script the Screencast
Record the Screencast
Week 1
Week 2
Week 3
Week 4

November 3rd - Get Week 1 Deliverables Done
November 10th - Get Week 2 Deliverables Done, Stubs for other Visualizations. (Milestone)
November 17th - Get Week 3 Deliverables Done.
November 24th - Catch Up, Final Details
December 1st - Finish Screencast (Project Deadline)

Milestones

- Code for gathering data written
  - Demo scene modeled, shaded, and textured
  - Demo scene rendered and data collected
- Raw data processed and organized
- Visualizations completed
- Optional visualizations coded
- Final product completed
- Screencast script written
  - Screencast recorded
- Project website completed
- Process Book completed