

CS 6630 Process Book

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/>

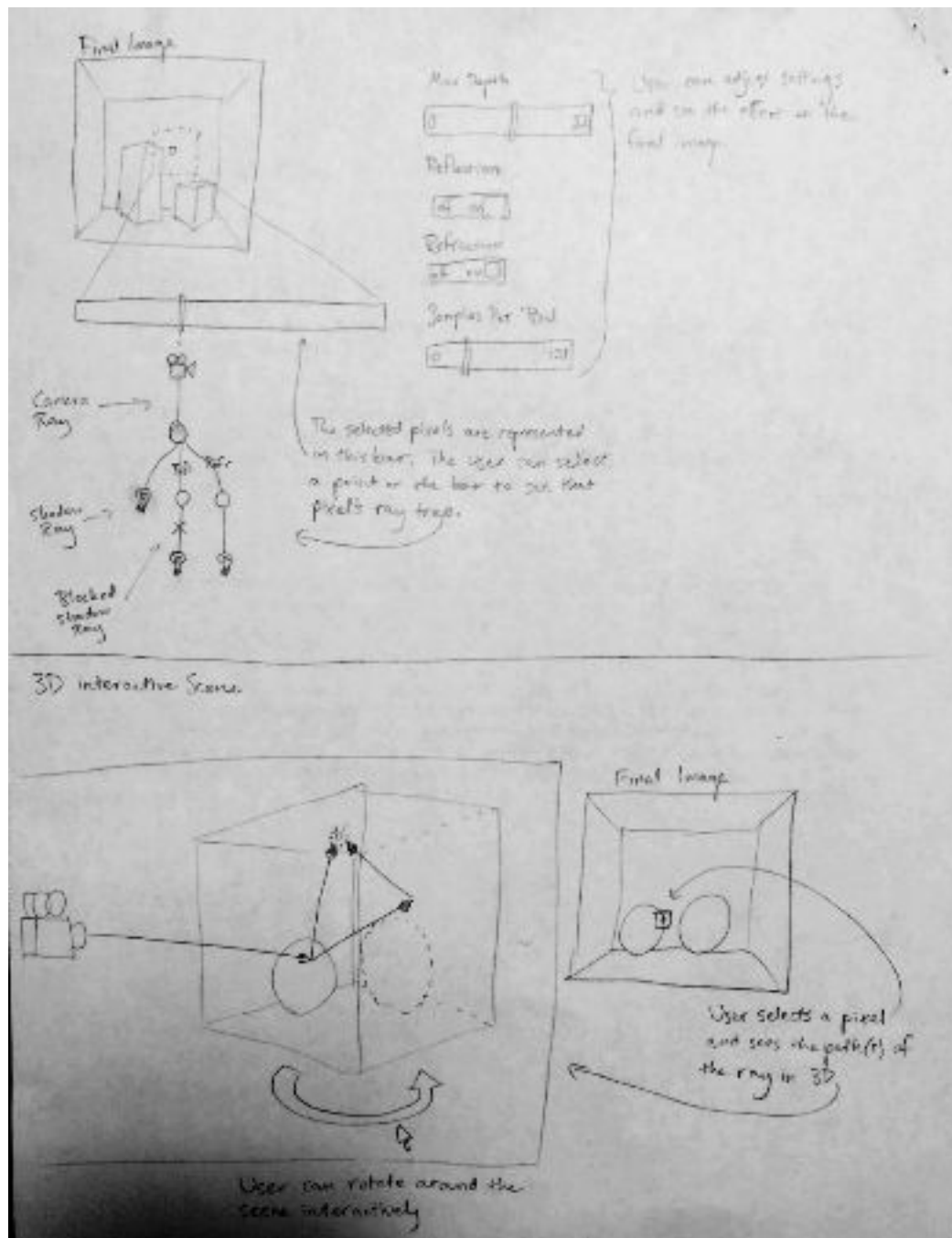
As scenes are generated, we would export intermediate results for visualization. For each pixel, we will generate a ray tree structure. Links represent rays and nodes represent intersections. Each link will store its ray direction, length, depth, type (camera, shadow, reflection, or refraction), and 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

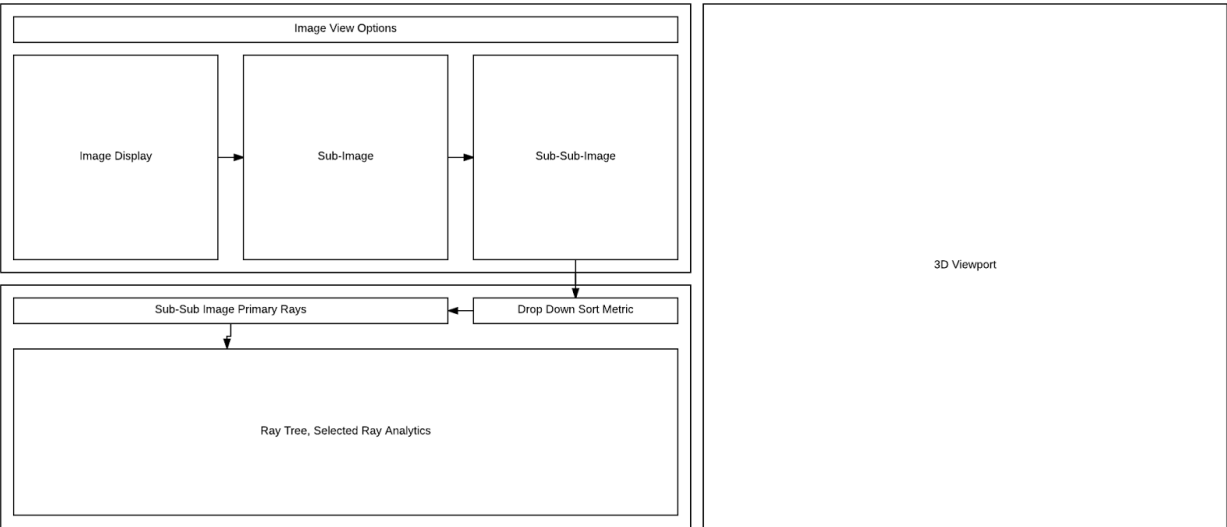
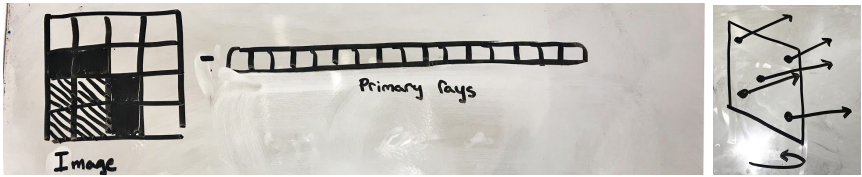
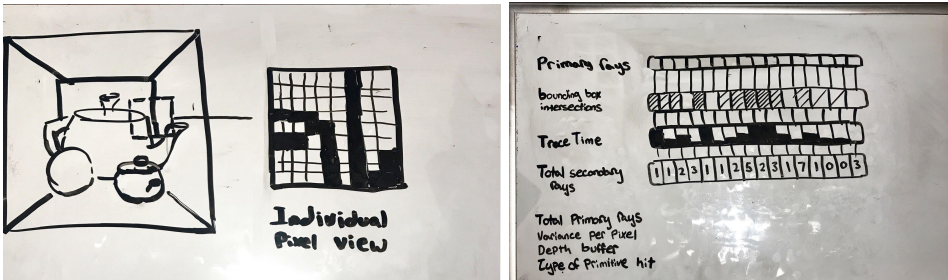
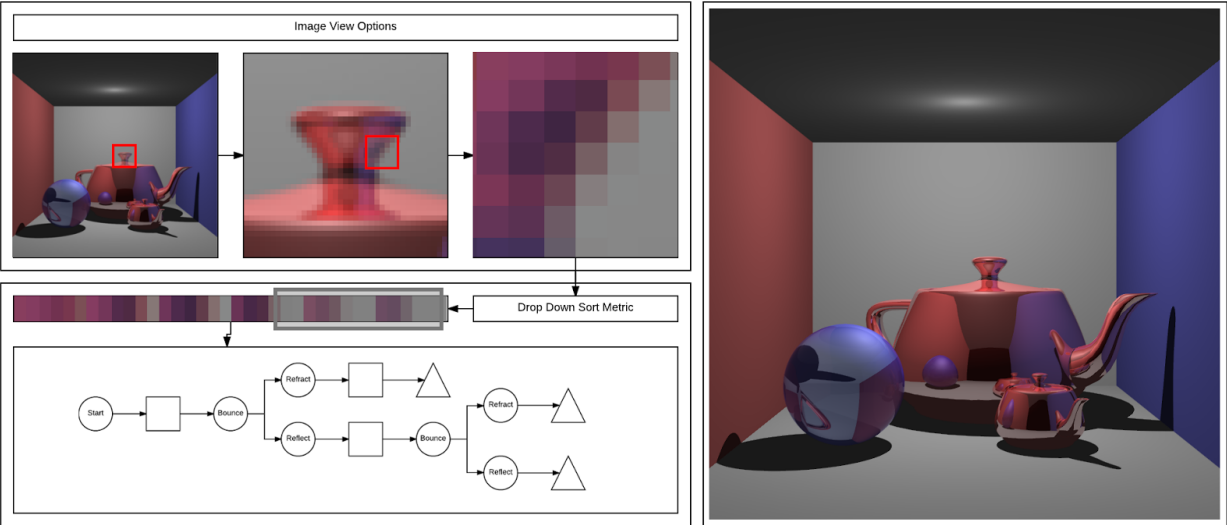
We will be generating all of the data with our ray tracer, so there should not be substantial data cleanup; we can generate exactly what data we need and organize it how we need it.

Visualization Design

Sketch 1:



Sketch 2



Must-Have Features

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

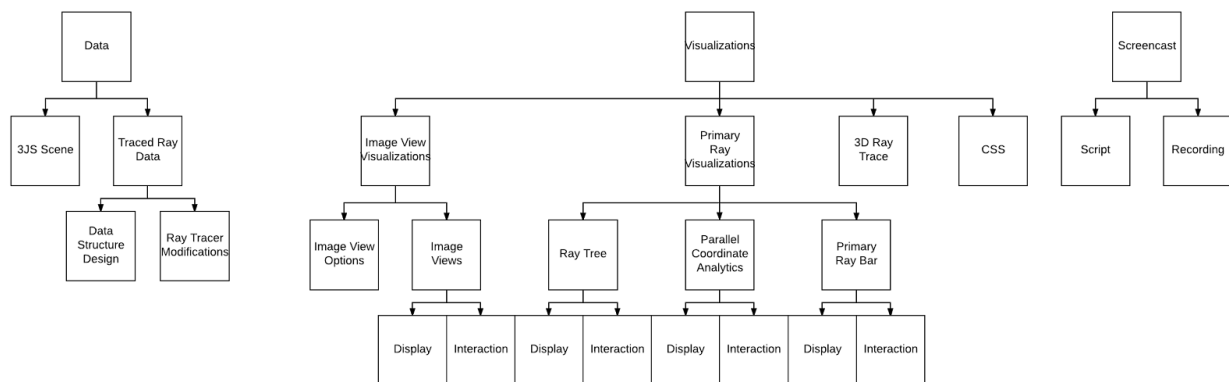
- Selecting groups of pixels in the rendered image to visualize the types of rays that passed through those pixels.
- Visualizing the scene traversal of individual rays as a tree.
- Visualizing which rays hit which objects, eg through parallel coordinates
- Visualizing distance traveled per ray
- Changing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

Optional Features

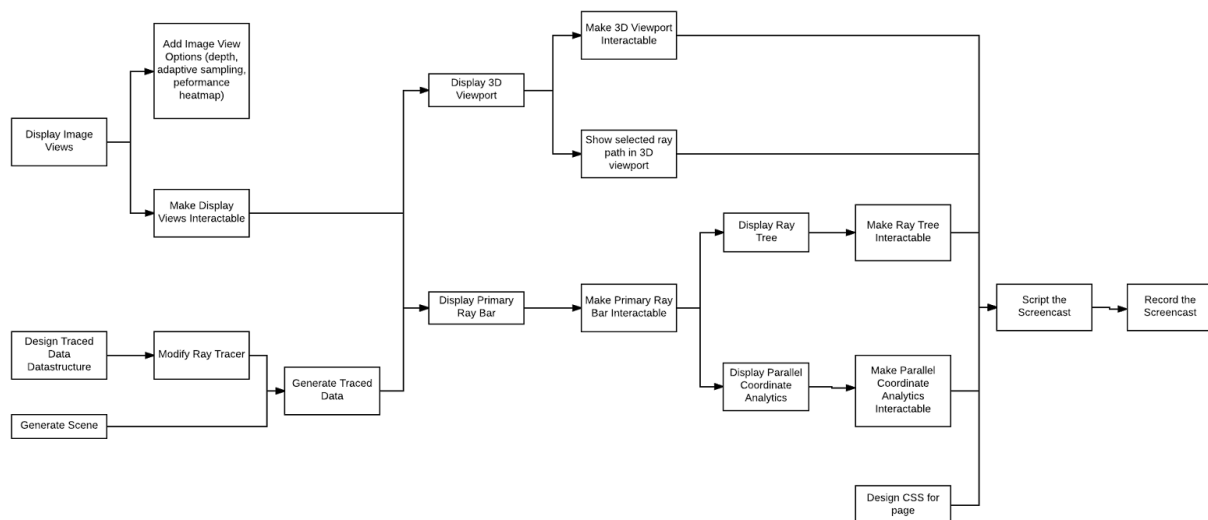
List the features which you consider to be nice to have, but not critical.

- 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

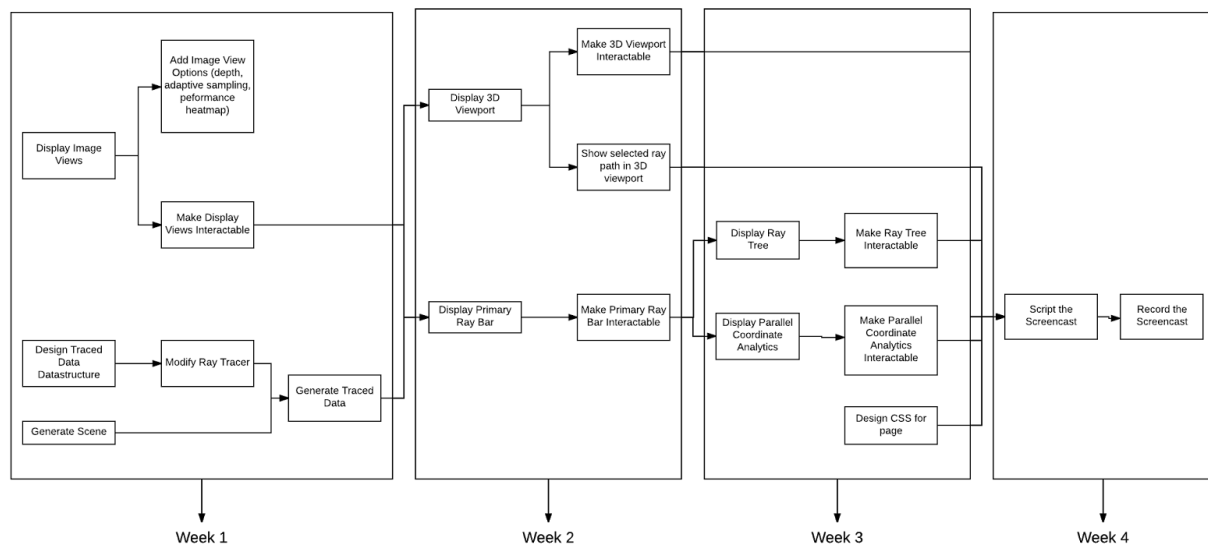
Work Breakdown Structure



Project Network Diagram



Project Schedule



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

Data Design

Ray data to collect:

For the scene:

- Camera info:
 - camera position
 - camera target
 - up vector
 - field of view
 - Image dimensions
- Scene info:
 - For each object
 - transformation
 - type (sphere, plane, mesh)
 - if mesh, filename
 - name
 - Scene coordinate system
 - What is up?
 - What is right?
 - What is forward?

For each pixel:

- Final pixel color
- Time taken to render
- (derived) Total number of secondary rays
- How many samples were taken
- Depth buffer
- Variance of the samples

For each sample:

- Time taken to render
- (derived) Total number of secondary rays

For each ray (which includes the traversal through the scene):

- Origin point
- Direction
- Which object we hit
- Number of ray-box intersection tests
- Number of ray-primitive intersection tests
- Type (camera, reflection, refraction, shadow)
- Bounce color contribution
- A list of secondary rays created by this ray

Storytelling Study

Reader-driven

- no prescribed ordering of images
- high degree of interactivity
- Dashboards, tool that is used over and over by professionals

← not so much a new, potentially better

Story driven

- constrained interaction
- various checkpoints
- users don't feel off-curve

Project is a bit of both.

Add "narrative"! Add "checkpoints" more guided exploration

Stories need beginning, middle, & end

Exposition → Climax → Pulling action → Resolution

- ~~end~~ could be through "pop out"
- start Popout w. less degree
- climax Popout w. med degree Popout etc

Conclusion may be read first, that's okay

- in their heads, user will 'order' story linearly

Add "Anxiety" through Popout too

Know what you really want to say?

- KWRWTS

Know what your data is saying

- KWRWTS

- trying to back up your opinion w. data instead of the other way around can be dishonest.
- avoid biases from filtering what your data is saying.

Know what your Audience needs to hear

- not communicating in vacuum

What Do I want to say?

What is data saying?
lots of metrics is primary
time spent on Refs, Refs, high/low time

What does audience need to hear?

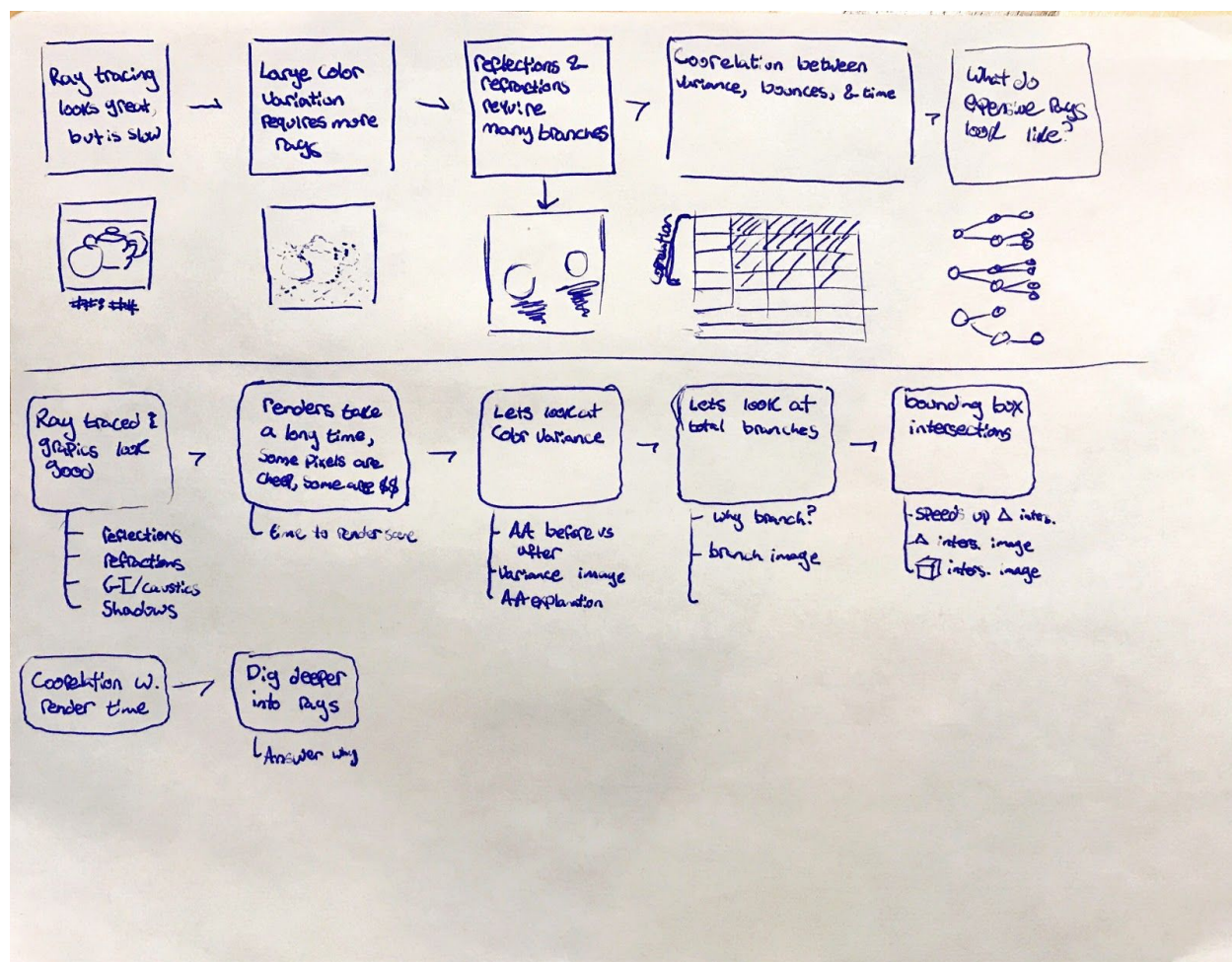
```

graph LR
    A[High Res] --> B[Pop Per Pixel/Low Res]
    B --> C[Variance Per Pixel]
    C --> D[Bounce Per Pixel]
    D --> E[Generation Evaluation]
    E --> F[Why is this low pixel]
  
```

As I designed the interface, I tried to consider how I could strike a balance between creating a reader driven visualization, and a story driven visualization. I ended up creating both a story mode and an explore mode. Story mode is more story driven and explore mode is more reader driven.

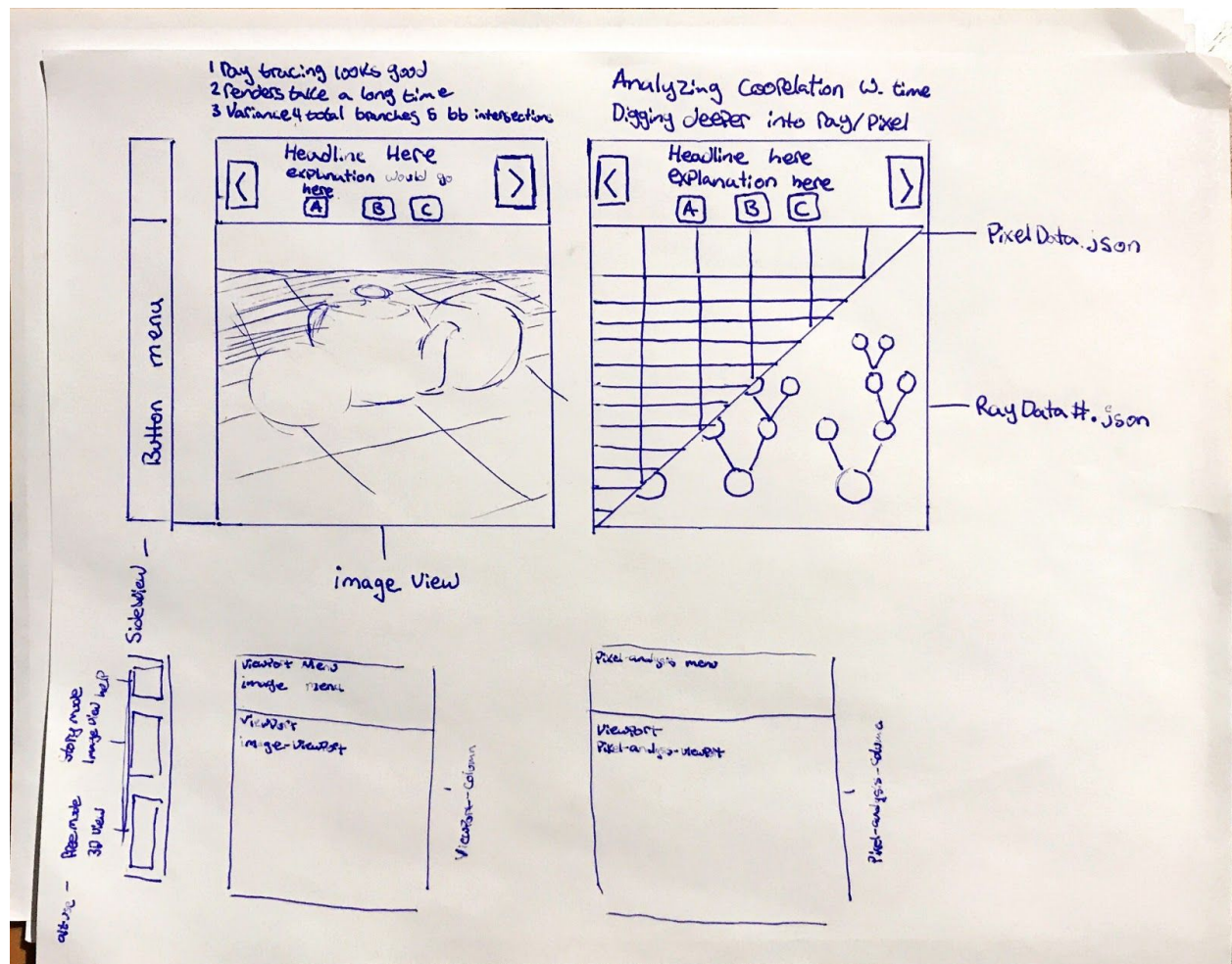
After meeting with Carolina, I decided to focus mainly on the elements of a basic ray tracer for my story. Interesting details can be teased out by sorting the parallel bars, sweeping over pixels, comparing images, and looking at the resulting ray trees.

Storyboarding Study



To incorporate more storytelling in the project, I tried storyboarding my project to clarify what I wanted to say, or what I wanted the user to find out. The top of this image is a first pass storyboard, where I was trying to determine how I'd visualize different concepts I was trying to get across. Most of these line up with the final product except for the ray tree, where I switched over to a radial tree to handle multiple trees cluttering the interface.

Back to the Drawing Board

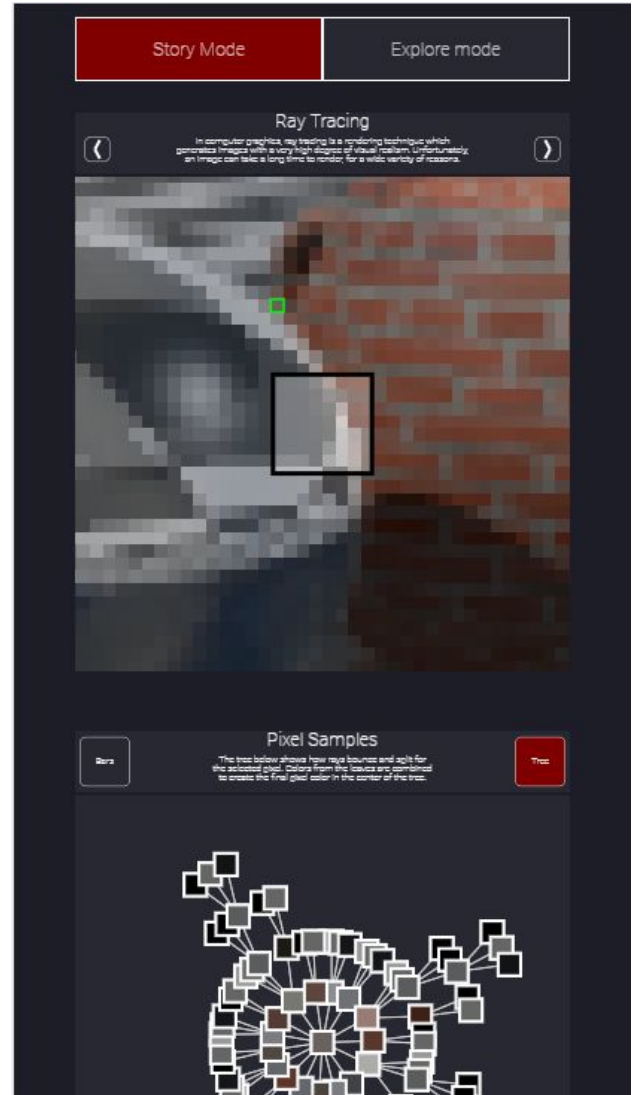


I used this quite a bit to update flexbox styling for both mobile and desktop modes, and to try incorporating more of a storytelling element to my visualization.

I was also determining which visualizations would require what data.

Final Screenshots

Portrait:



One of the goals was to make the project work on mobile. Originally, the navigable image spanned the entire width of the phone, but that made scrolling very difficult.

The visualizations all resize correctly when going from portrait to landscape and back.

Landscape


[illegible]

Story Mode

Explore mode

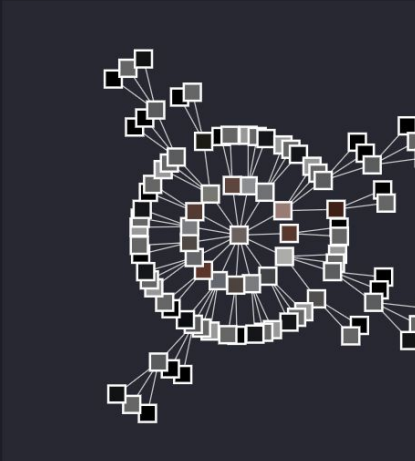
Ray Tracing

In computer graphics, ray tracing is a rendering technique which generates images with a very high degree of visual realism. Unfortunately, an image can take a long time to render, for a wide variety of reasons.



Pixel Samples

The tree below shows how rays bounce and split for the selected pixel. Colors from the leaves are combined to create the final pixel color in the center of the tree.



Conclusion

Overall I'm fairly satisfied with this project. There were some technical issues at the beginning, since my teammate had to drop the class for personal reasons, but I still was able to implement the majority of the features I wanted implemented. There are still a couple things which I'd like to improve on, like making the ray tree a bit more interactive, displaying minimum and maximum values instead of just relative comparisons, and visualizing global illumination, which complicates things quite a bit.