CSCI 580, PROF. SATY RAGHAVACHARY

Crepescular Rays using Ray Tracing

Team Uncanny Valley

https://sites.google.com/site/uncannyvalley580/
Sanskriti, Saurabh Mistry, Sreepada Rao Singeetham, Tushar Tiwari
December 4, 2014

1 OBJECTIVE

To achieve the above render, we make use of two well-known techniques namely Ray Tracing and Volumteric Shadowing. Ray Tracing is a technique for generating an image by tracing the path of light. Volumetric Shadowing is a technique that allows the viewer to see shadow beams called crepescular rays.

Contents

2 Introduction

The Ray Tracing algorithm very much mimics the procedure of observing the color of an object in nature. A light source emits a ray of light which travels, eventually, to a surface that interrupts its progress. One can think of this "ray" as a stream of photons traveling along the same path. There are two basic ideas for ray tracing.

Forward ray tracing and backward ray-tracing. Forward ray-tracing is how the color of an object is observed by the eye in real life. That is, a ray of light is reflected of an object and travels into the eye producing an image. However, this algorithm is impractical because there will be many rays that will miss the camera and will not be useful to us. Thus, there will be a huge computational overhead.

For CG purposes, we make use of the backward ray-tracing algorithm. The basic idea is shoot a ray from the camera through each pixel of the image plane. If this ray intersects with an object, we send out another ray in the direction of our light source to determine if the object is under a shadow of another object. We also send out a reflected ray to obtain the reflection of another object if the object is capable of reflections.

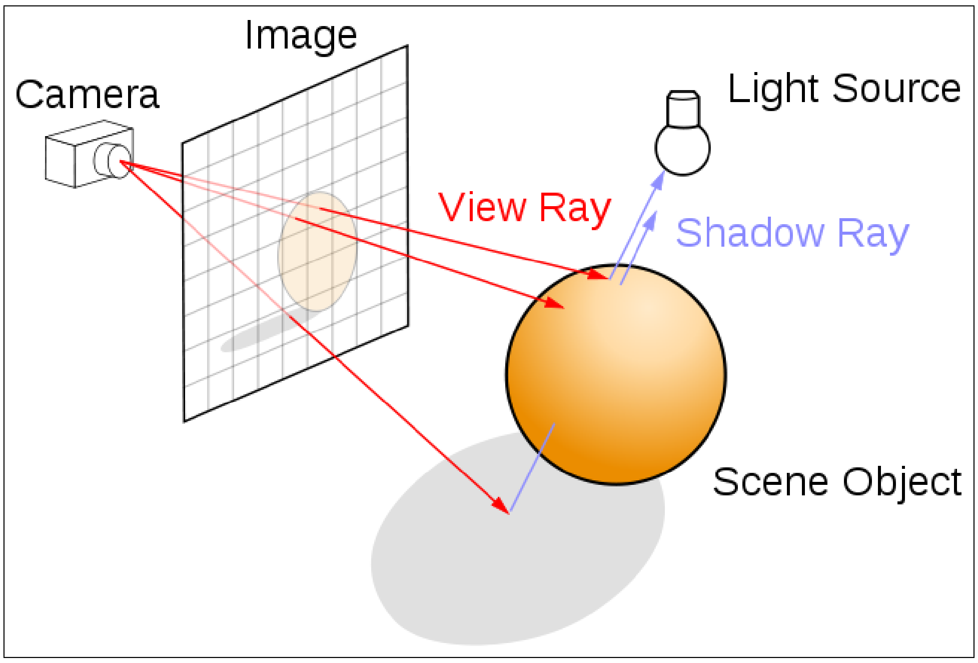


Figure 2.1: Basics of Ray Tracing

The algorithm we have used to compute the volumetric rays is quite straightforward. We use the same concept used to compute shadows on the image plane. We start from the original image plane, compute shadows for every pixel on that image plane. Then we move the image plane towards the object and obtain a newer, smaller image plane. Compute shadows for this new image plane. This process is repeated until we reach the object. The below image illustrates this process.

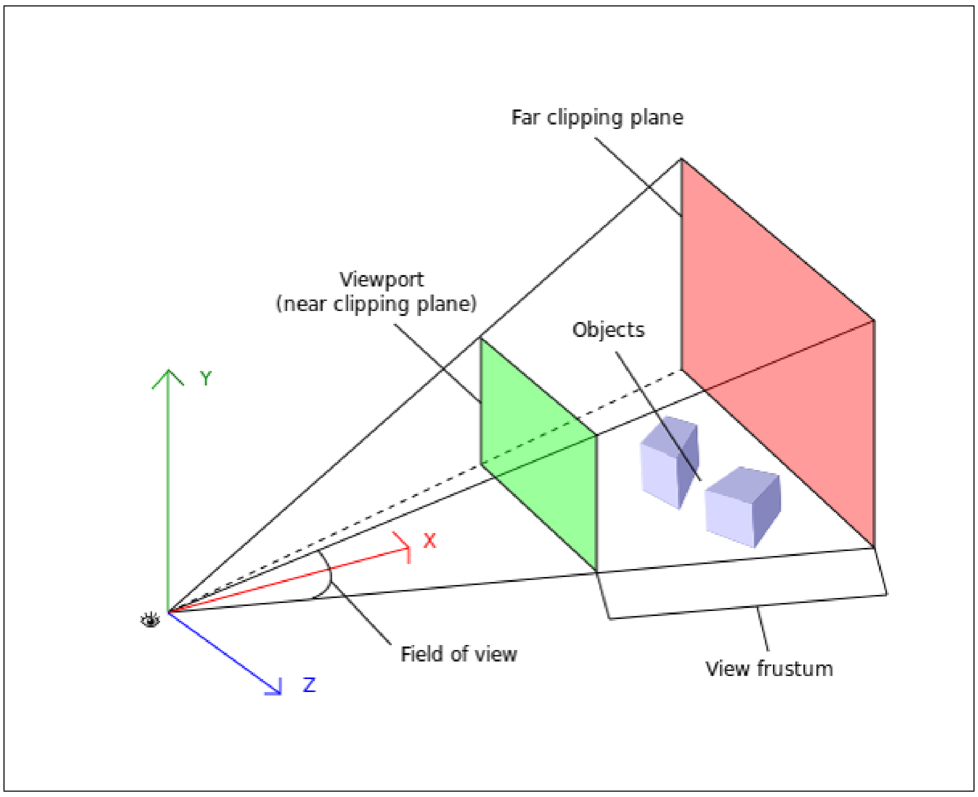


Figure 2.2: Basics of Volumetric Shadowing

3 System Schematic

The System Schematic is a flowchart of operations done in different aspects of the project. Is represents the different modules in the program and thus depicts the flow of code.

3.0.1 Ray Tracing

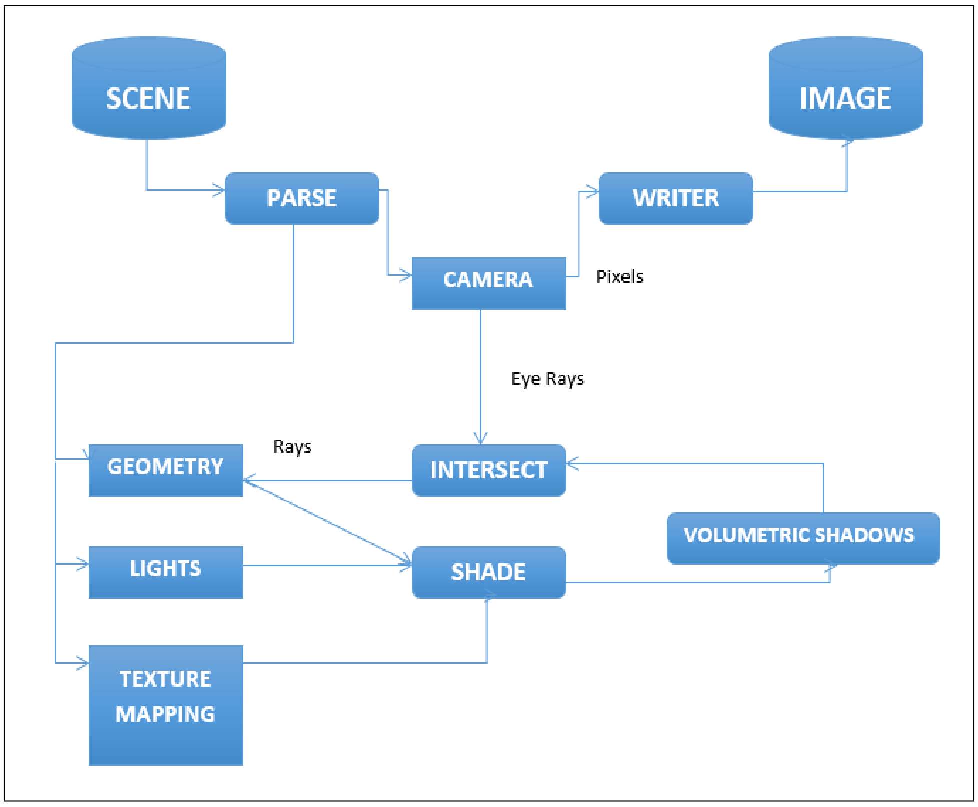


Figure 3.1: System Schematic for Ray Tracing

3.0.2 Volumetric Shadowing

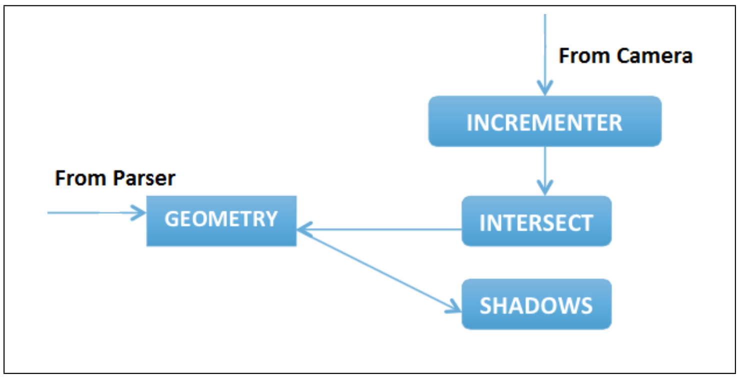


Figure 3.2: System Schematic for Volumetric Shadowing

3.0.3 Scan Line

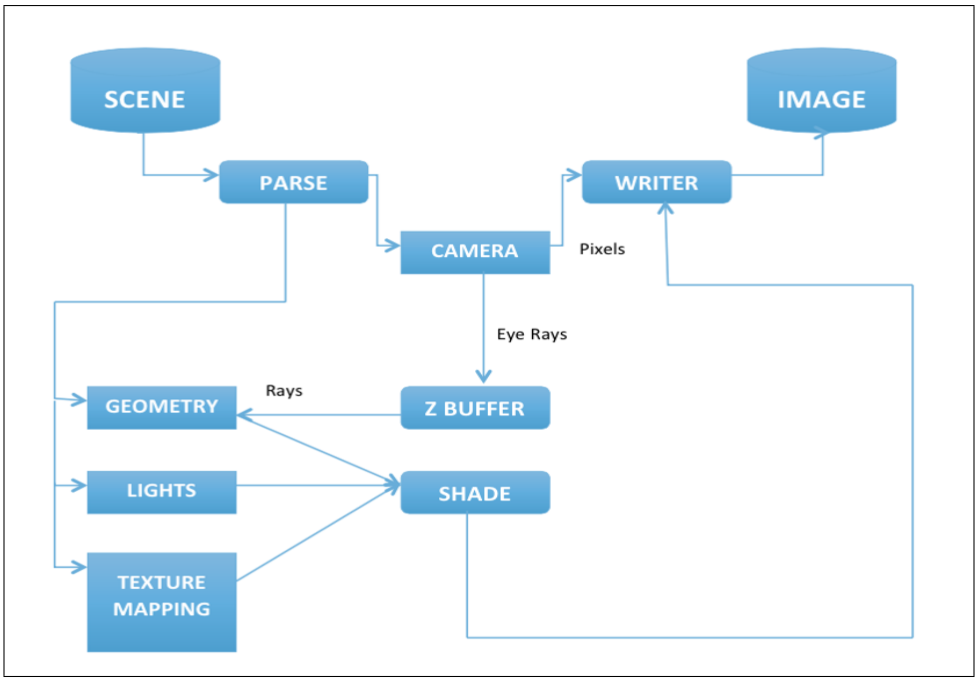


Figure 3.3: System Schematic for Scan Line Rendering

4 Implementation

4.1 Scene Creation and Obj Format

Since most objects out there use the v, vt, f and quadrilateral format, our program had to be modified to read data in that format and modify it into triangles.

Moreover, a single leaf obj wouldn't suffice for the purposes of this project. So we took the same leaf obj, translated it, rotated it and scaled it to create a scene that represented falling leaves. All these combinations of leaves in different orientations were merged to create a single render. The scenes rendered can be dynamic or static.

4.2 Backward Ray Tracing

4.2.1 Compute the Camera Rays

We represent a ray by obtaining the ray's origin and its direction. The first ray we look at is the viewing or camera ray. This ray is computed by making use of the camera position and the current pixel which has to be coloured. We first transform the pixel coordinates to $x, y \in [-1, 1]$.

$$i = \frac{x * 2}{imagePlaneWidth} - 1$$

$$j = \frac{y * 2}{imagePlaneHeight} - 1$$

The ray to the object can be represented by using the origin of the ray and the direction of the ray computed by using the object.

$$c_n = camera_{lookat}(c_l) + camera_{position}(c_p)$$

$$\overrightarrow{c_v} = camera_{lookat}(c_l) \times camera_{worldup}(c_u)$$

$$\hat{c_v} = \frac{\overrightarrow{c_v}}{|\overrightarrow{c_v}|}$$

$$ray\_camera\_to\_object_{position}(r_p) = c_n - c_v \cdot i - c_u \cdot j$$

$$ray\_camera\_to\_object_{direction}(\overrightarrow{r_c}) = r_p - c_p$$

4.2.2 Compute the intersection of the camera ray with the plane of the triangle

Since there is no way knowing which triangle the ray will intersect with, we have to try out all triangles. To check if the ray intersects with the plane of the triangle is easy. By finding out the normal to the triangle and taking the dot product of the ray and this normal we can verify if the triangle is parallel to the ray or starts behind the triangle or neither of those.

To begin with we need to make sure that we get the normal outwards rather than inwards. This will ensure that we get the correct normals. We first arrange all vertices in a counter-clockwise manner. If $v_0, v_1 and v_2$ are the vertices in counter-clockwise order, then the normal is obtained by,

$$n = \overrightarrow{(v_1 - v_0)} \times \overrightarrow{(v_2 - v_0)}$$

$$\triangle_{normal} = \hat{n} = \frac{n}{|n|}$$

Then the dot product of the $\triangle_{normal}$ and the ray can provide us with information whether the $\triangle$ is parallel to the ray or not.

$$\triangle_{normal} \cdot \overrightarrow{r_c} = 0 \implies \triangle \parallel \overrightarrow{r_c}$$

We are interested in the last condition because this signifies that the camera ray intersects the plane of the $\triangle$.

4.2.3 Check if intersection point is inside the Triangle

To obtain the point of intersection we must first compute the triangle distance:

$$d(\triangle) = \triangle_{normal} \cdot v_0$$

Then, the following equation gives us the distance to the plane:

$$d(plane) = -1 \times \frac{\triangle_{normal} \cdot (\overrightarrow{r_c} + (\triangle_{normal} \times -d(\triangle)))}{\triangle_{normal} \cdot \overrightarrow{r_c}}$$

Next we need to find the point that intersects with the plane. This point, say q, is found by:

$$q = d(plane) \cdot \overrightarrow{r_c} + r_p$$

Now, that we have the point, we need to verify if this point lies inside the $\triangle$.

$$\frac{Area(q, v_1, v_2) + Area(q, v_0, v_2) + Area(q, v_0, v_1)}{Area(v_0, v_1, v_2)} \approx 1$$

If the above condition holds then our point is inside the $\triangle$.

4.3 Shadows

From the point of intersection we need to shoot another ray in the direction of the light and find if it intersects with any of the objects. If it does, then that point is in shadow. But for the purposes of this project where the shadow falls on the image plane, we shoot out a ray from each pixel on the image plane in the direction of the light source (multiple shadows can be cast by shooting rays in the directions of the multiple light sources). This is done by performing the following calculations.

$$s = \overrightarrow{l} + \overrightarrow{c_l}$$

$$v = \overrightarrow{c_u} \times -\overrightarrow{c_l}$$

Transform the coordinates such that $x, y \in [-1, 1]$

$$r_l = s - ((v \cdot -i) - (c_l \cdot -j)$$

$$\overrightarrow{r_l} = \overrightarrow{l} - r_l$$

$$\hat{r}_l = \frac{\overrightarrow{r_l}}{|r_l|}$$

Now that the light ray is calculated, we can go ahead and calculate the shadows by performing similar calculations done with the camera ray above, i.e. calculating the intersections and checking if it is contained in the triangle. If it is then the point is in shadow.

The shadow rendered for a small leaf object.

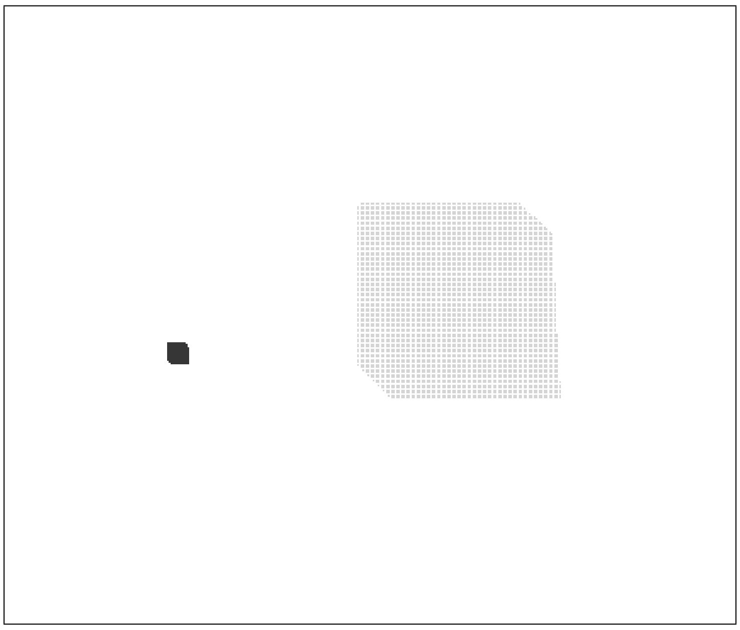


Figure 4.1: Shadow for a single leaf using ray tracing

4.4 Volumetric Shadows

The computation for Volumetric Shadows is very similar to computing shadows in ray tracing. The only difference here is that we compute the shadows for every step in the image plane, indicating that the image step is moving towards the object. The image step indicates how fine the shadows will be. A lower shadow step will make the shadows more granular. The ray from each pixel of the image plane to the light source will have a different equation with each render step.

$$\overrightarrow{r_l} = \overrightarrow{r_l} \cdot shadow\_step$$

The volumetric shadow rendered for a small leaf object.

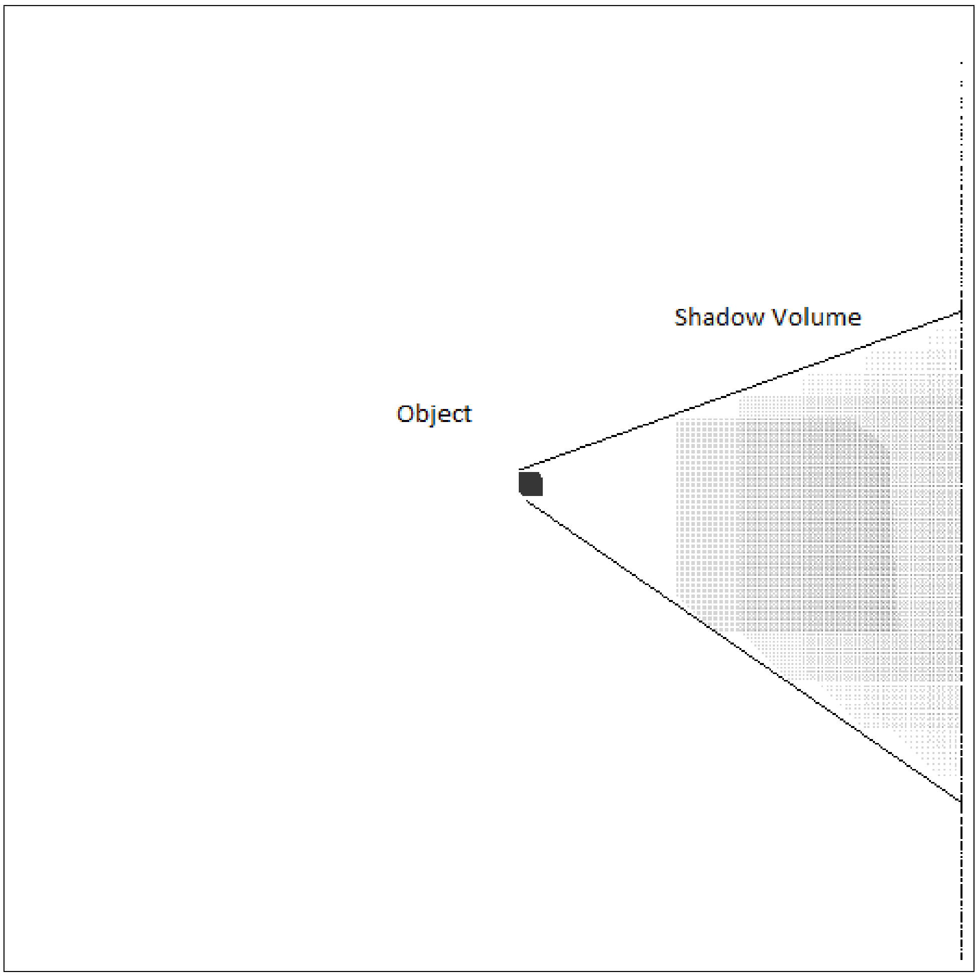
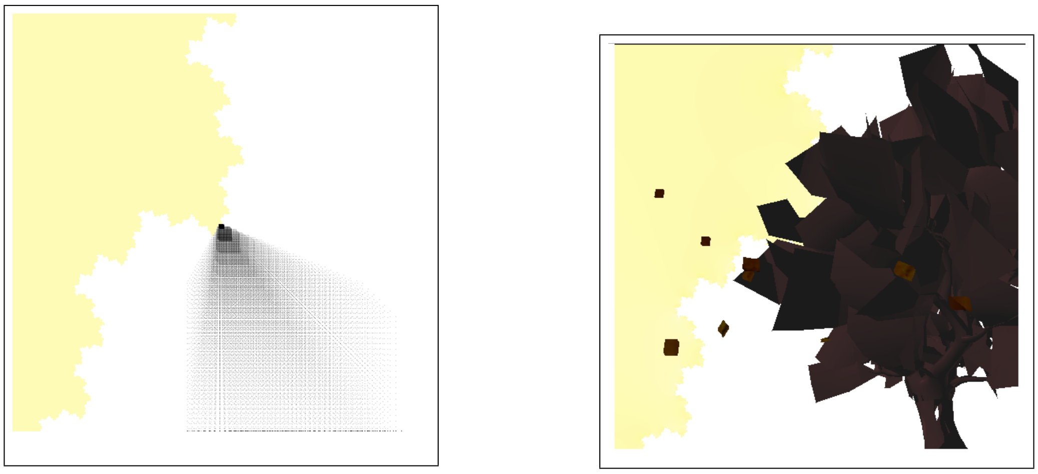


Figure 4.2: Volumetric Shadow for a single leaf with a large step size

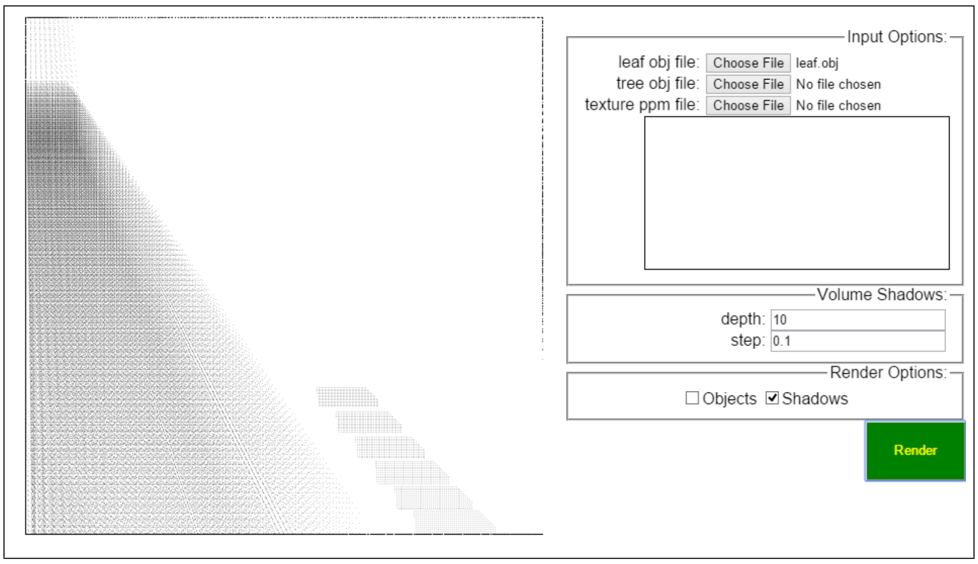
5 Results



(a) A sun fractal acting as light source with a leaf producing volumetric shadows (Took 28 min)

(b) Scene with a sun fractal, trees and leaves

Figure 5.1: Render of different scenes with a sun fractal



Input Options:

leaf obj file: Choose File leaf.obj

tree obj file: Choose File No file chosen

texture ppm file: Choose File No file chosen

Volume Shadows:

depth: 10

step: 0.1

Render Options:

☐ Objects ☑ Shadows

Figure 5.2: Render of multiple leaves with increased step size (Took 28 min)

6 Challenges

- One of the biggest challenges in the project is the slow computation time of ray tracing. Since every pixel is rendered against every triangle in the scene. The computation time is very huge. Therefore the debugging is a longer and tiring process.
- Visualizing everything in object space was a difficult process.
- Using the correct light positions to get a shadow.
- Volumteric Shadows were also very challenging. It was a fairly new algorithm.
- The volumetric shadow calculation with a new ray at every intermediate plane led to a lot of calculations.

7 Conclusion

Although we did not arrive at the exact result we were hoping to, we have acheived the purpose of this project, that is to have a better understanding of ray tracing and volumetric shadowing techniques.

While researching and implementing Ray-Tracing we were exposed to many methods that showed us how to implement photorealistic images. We were also made aware of the enormous computations required in the ray-tracing algorithm. In the process, we also mastered the ray-tracing algorithm and found out various alternatives to perform certain tasks in ray-tracing.

We also investigated into the several ways volumetric shading is done. And since, the shadows fall on the camera, we tweaked the algorithm to produce desired result.

References

[1] *A Hierarchical Volumetric Shadow Algorithm for Single Scattering, Ilya Btaran*