Ray Tracing on Rectangular Surfaces

December 12, 2020

Team - Green Screen

Adwait Thattey (S20170010004)

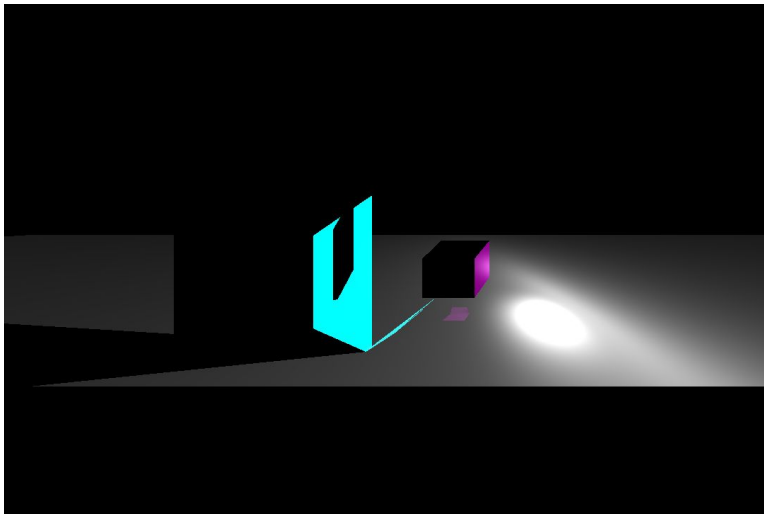
Brijesh Bumrela (S20170010031)

Siddhant Jain (S20170010151)

Dhruv Nair (S20170010101)

Venkata Sri Mukhesh Inturi (S20170010056)

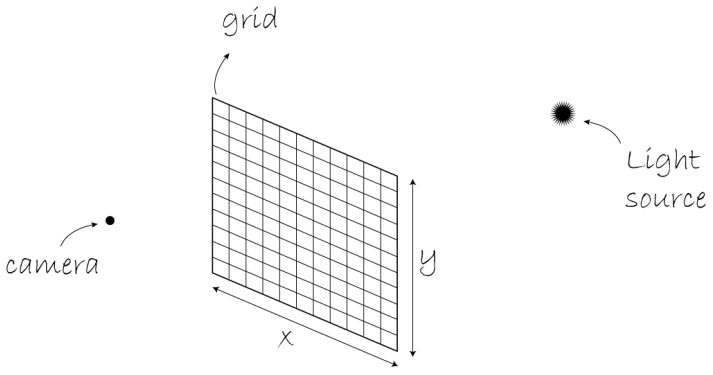
Tanay Rathore (S20170010163)



Contents:

1. Define Camera, Screen, Light Object

- Camera, Screen, Light and Objects are the main components of the 3D rendering
- *Screen* :-
    i. Screen is the 2d grid which is used as the view plane of the space
    ii. This space can be divided into a variable number of rows and columns. Each grid cell is a pixel.
- *Camera* :-
    i. Camera is present at the center of the screen 1 unit behind the screen. (This is configurable in code)
    ii. All the rays present in the animation start from camera's coordinates
- Light Source :-
    i. It is a point source which produces light in all directions.

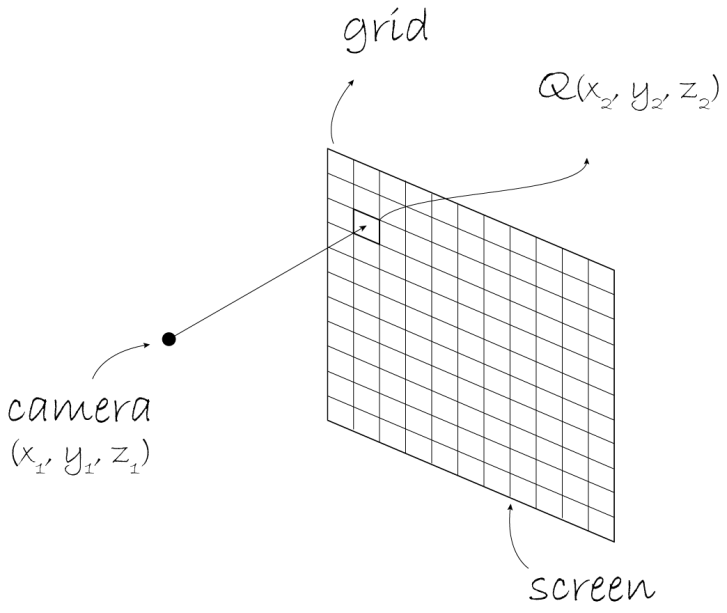


2. Show camera image & equation of ray:

- Let us consider coordinates of camera are C (x_1, y_1, z_1) and coordinates of a pixel is Q (x_2, y_2, z_2), then the equation of the ray emitting from it is,

  P(*t*) = C + (Q-C)t

  where t is the distance of point P from Camera C.



3. Defining objects

- Our model creates 3D rectangular shaped objects
- Creation of the objects involves
    i. Defining the vertices
    ii. Defining the faces of objects
        1. For each face, following operations are performed
            - checking if the 4 vertices are coplanar
            - calculating and storing the normal of the face
            - solve for the **D** part of the plane equation
        2. Defining the face includes solving for the equation of the plane with the given data like the plane normal and one of the point present on the plane
            - using the plane equation represented as

$$Ax + By + Cz + D = 0$$

            - where (A, B, C) are the directional components of the normal and (x, y, z) is a point on the plane, we can calculate ***D***

Ray Tracing

The scene has multiple objects. The objects have multiple faces.

Finally we want an image of objects projected on the screen. For every pixel we have to calculate RGB values.

Number of Rays starting from camera = Number of pixels on the screen

For a ray **P(*t*) = C + (Q-C)t**, hitting a plane **Ax + By + Cz + D = 0,** algorithm is as follows -

1. **Find Intersection point**

   C and Q are given. We have to find 't' for the given plane.

$$
\begin{aligned}
&P = O + tR \\
&Ax + By + Cz + D = 0 \\
&A * P_x + B * P_y + C * P_z + D = 0 \\
&A * (O_x + tR_x) + B * (O_y + tR_y) + C * (O_z + tR_z) + D = 0 \\
&A * O_x + B * O_y + C * O_z + A * tR_x + B * tR_y + C * tR_z + D = 0 \\
&t * (A * R_x + B * R_y + C * R_z) + A * O_x + B * O_y + C * O_z + D = 0 \\
&t = -\frac{A * O_x + B * O_y + C * O_z + D}{A * R_x + B * R_y + C * R_z} \\
&t = -\frac{N(A,B,C) \cdot O + D}{N(A,B,C) \cdot R}
\end{aligned}
$$

   Where,

   T = Distance of point P from the Camera`
   N = Plane's Normal
   D = Distance of plane from Origin (Camera)
   O = Origin (Camera)

R = direction vector of ray (pixel - camera vector)

Intersection point P(t) = C + Q(t)

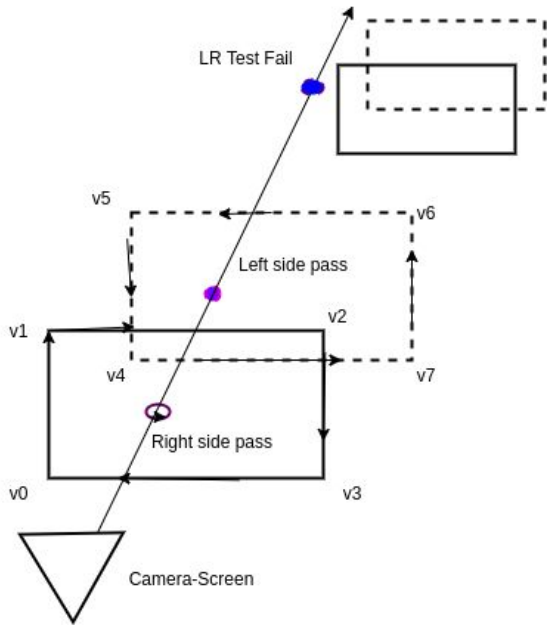
2. **Checks**
   a. **Check if the ray is not parallel to the given plane**

      dot(Plane's Normal, direction of ray) $\neq$ 0

   b. **Check if the intersection point is not behind the camera -** if it is, then this intersection point is not visible in the scene and thus not considered.
      If t > 0; intersection point is visible to the camera
      If t < 0; intersection point is behind the camera and hence not visible

3. **Is Intersection Point inside or outside the face.**
   Until now, we saw how to find the intersection point with a given plane. Remember we got this plane because it has at least one face of one of the objects.

   Now simply if the intersection point is on the face, then only we consider it.



For a point to be on the face, it should be either on the left or on the right of **all** the **edges**.

edge0 = v1 - v0 C0 = P - v0

edge1 = v2 - v1 C1 = P - v1

edge2 = v0 - v2 C2 = P - v2

N = Normal to the face

IF (dot(edge0, cross(C0, N)) > 0 && dot(edge0, cross(C0, N)) > 0 && dot(edge0, cross(C0, N)) > 0)

$\Rightarrow$ *towards left of all edges* $\Rightarrow$ *Intersection Point is on the face*

For our case i.e. rectangular faces, we only compute intersection points for faces where only the right hand rule satisfies.

4. Finding intersection point nearest to the camera

Since there can be multiple intersection points that too on the faces, we just have to find the nearest intersection point as the corresponding face will block the light ray.

For every intersection point

Find distance of this point from Camera

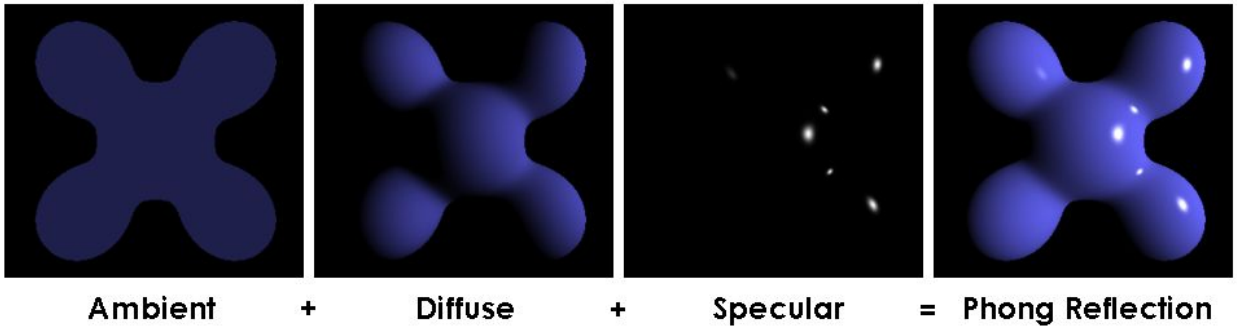
Pick the intersection with least distance.

5. **Checking if the pixel is shadowed or not:**

- The next step is to check if a particular pixel is shadowed by some object or not.
- For this, we find the ray from the intersection point to the light source and check if any object comes along the path.
- But in many cases, the object itself will be blocking the way.
- In order to solve this problem, we use the concept of **shifted point** which means that we take a point on the normal which is shifted away from the original point by a little threshold.
- The pixel is shadowed if there is any object along the ray from the shifted point to the light source. Else the pixel is not shadowed.
- **If shadowed,** we assign no color to it. (we make it black)
- **Else,** we calculate **resultant color** at that pixel by calculating
    - Illumination of that pixel
    - Color due to reflections by other objects on that pixel

6. **Calculating the Illumination:**

We calculate illumination of that pixel using **Blinn-Phong model** using the formula



$$I_p = k_a * i_a + k_d * i_d * L \cdot N + k_s * i_s * (N \cdot \frac{L+V}{\|L+V\|})^{\frac{\alpha}{4}}$$

Here,

- Ka,kd and ks are the ambient, diffuse and specular properties of the object
- Ia,id and is are the ambient, diffuse and specular properties of the light source
- L is a direction unit vector from the intersection point towards the light

- N is the unit normal vector to the surface of the object at the intersection point
- V is a direction unit vector from the intersection point towards the camera
- Alpha is the shininess of the object.

7. **Calculating Color due to Reflections caused by other objects:**
    - First we calculate the equation of reflected ray
    - Then we find the nearest object that this ray intersects
        - If there is a object
            - We calculate the color due to this reflection by the formula: reflection coefficient * illumination
            - We again repeat this process upto desired number of iterations (we have taken upto 5 iterations)
            - Each time the reflection coefficient is multiplied by the reflection coefficient of the object.
        - If there is no object
            - We stop the iteration

8. **Calculating the overall color:**

    Overall color = color due to illumination + color due to reflections by other objects

    We assign this color to the pixel. We do this process for all of the pixels and finally get the image.

Improving Performance of the above Ray Tracer

1. GPU processing
    a. We can use GPU to perform computations on the vectors. This will give a huge performance boost as GPUs are much faster for such use cases.
2. Parallel processing
    a. Multiple objects can be processed in parallel when trying to find the nearest object that the ray intersects. Also once a nearest object is found the process to check for shadow and find reflections can be done in parallel.
3. Using K-D trees to reduce the number of objects that are checked for each ray
    a. KD tree recursively partitions the space with the planes that are perpendicular to the axes of a coordinate system. There are various build and traversal algorithms that can be used.
    b. This approach gives a tremendous performance boost especially when there are a large number of objects which is the case in most scenarios.

References:

- https://medium.com/swlh/ray-tracing-from-scratch-in-python-41670e6a96f9
- https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle
- Hapala, M., & Havran, V. (2011). Review: Kd-tree Traversal Algorithms for Ray Tracing. Computer Graphics Forum, 30(1), 199–213. https://doi.org/10.1111/j.1467-8659.2010.01844.x
- Ray Tracing Acceleration by Felix Alexander (https://slideplayer.com/slide/10459895/)
- CGM Course Notes