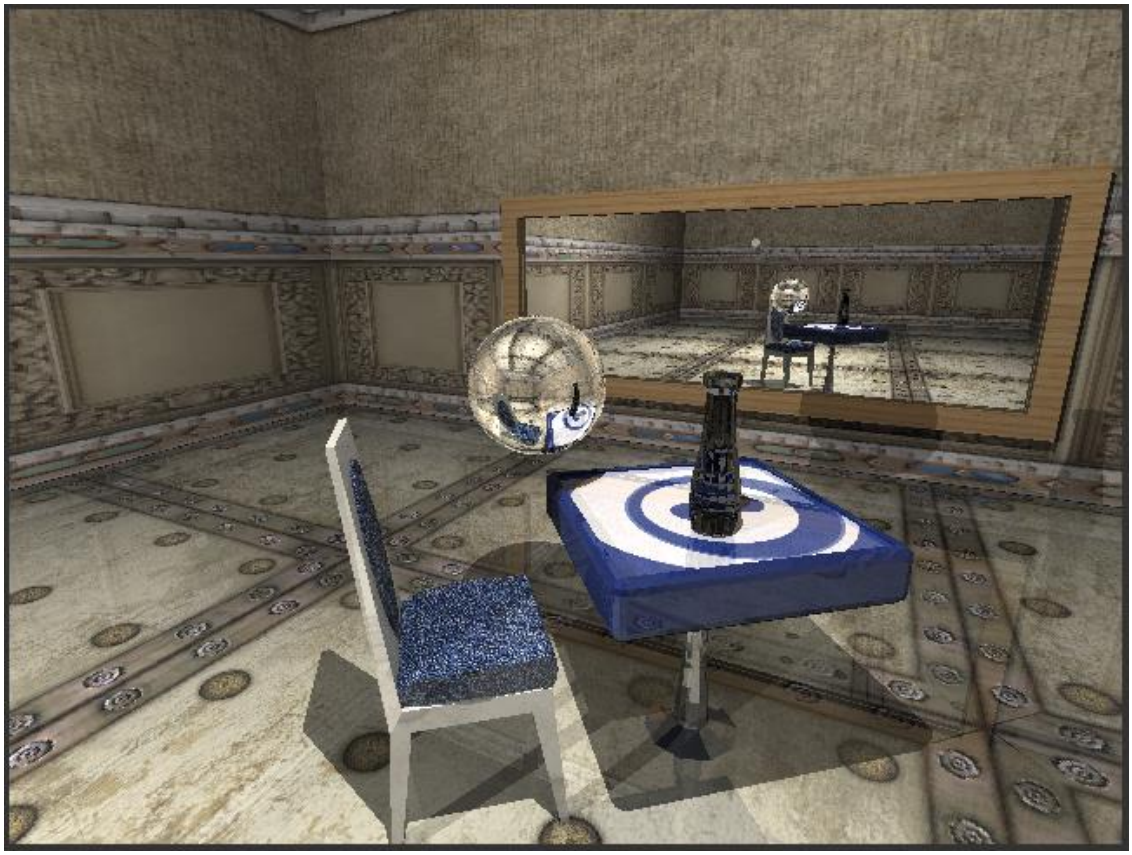
Chapter 1: INTRODUCTION

1.1 RAY TRACING CONCEPT

The Ray Tracing technique is used to imitate real life situations. It thrives to copy the real-life visualizations and act like nature. In real life, lights are transmitted everywhere and they enter the camera from every direction. Each one of those rays has a light intensity (color), and thus a picture is formed. Since it is too costly to transmit an infinite number of light rays on a PC, Ray Tracing tries as much as possible to get close to that reality.

Ray Tracing uses the same camera concept but in a reverse way. Rays are sent from the eye to the scene and then the light intensities are computed at the intersection points of the nearest objects. It is true that many rays are excluded from the real process, but this technique gives us a satisfying result.



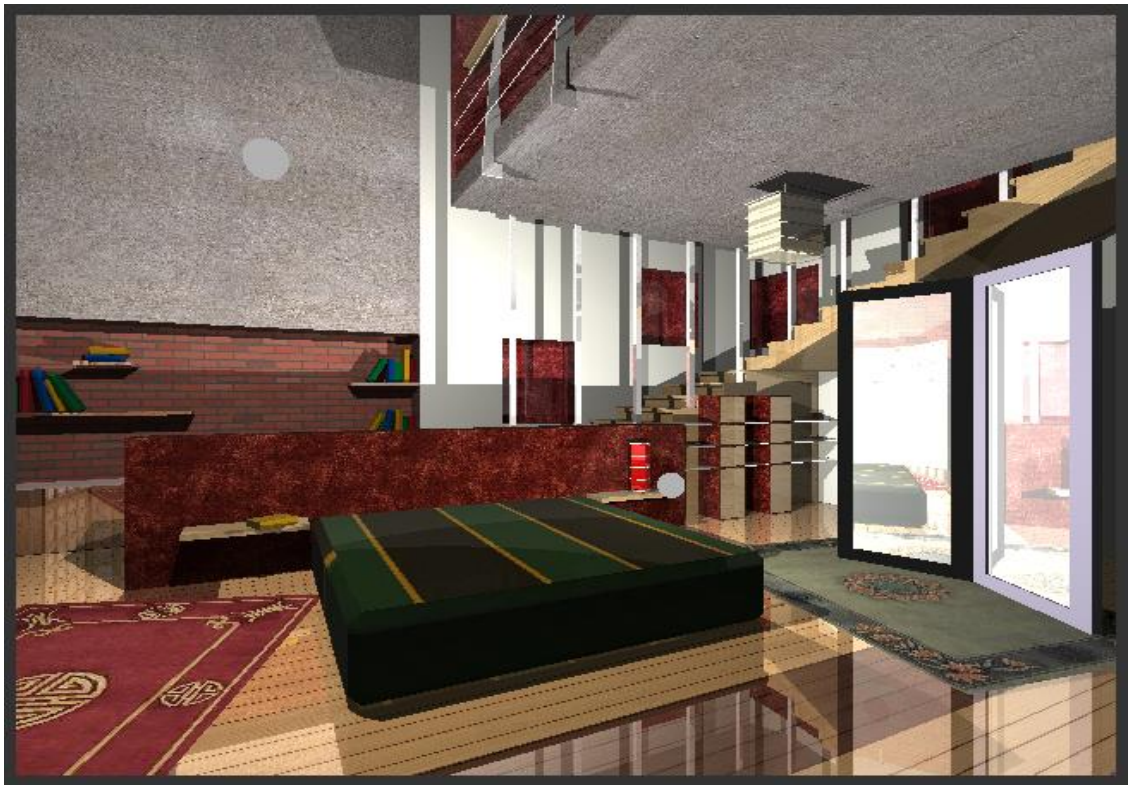
1.2 ADVANTAGES

Advantages

1. Pixel color is found in the world which means no Rasterization required. Rasterize, also

spelled rasterise, is converting a vector image (a mathematically defined image of points
and curves) to a raster image (a picture composed of discrete pixels).

2. Since the method is screen-to-world, the number of rays is finite.
3. Only the visible parts of the world are viewed thus no clipping is required.
4. The closest intersection point is taken hence we do not need to perform any Hidden Surface Removal calculations.
5. If a ray going from an intersection point P to the light intersects with an object, we can conclude that point P is a shadow point. In this case, shadowing is added.
6. Since Ray Tracing is all about tracing rays (reflected, refracted…), we do not have to worry about shiny objects, mirrors, glass objects and even water objects…



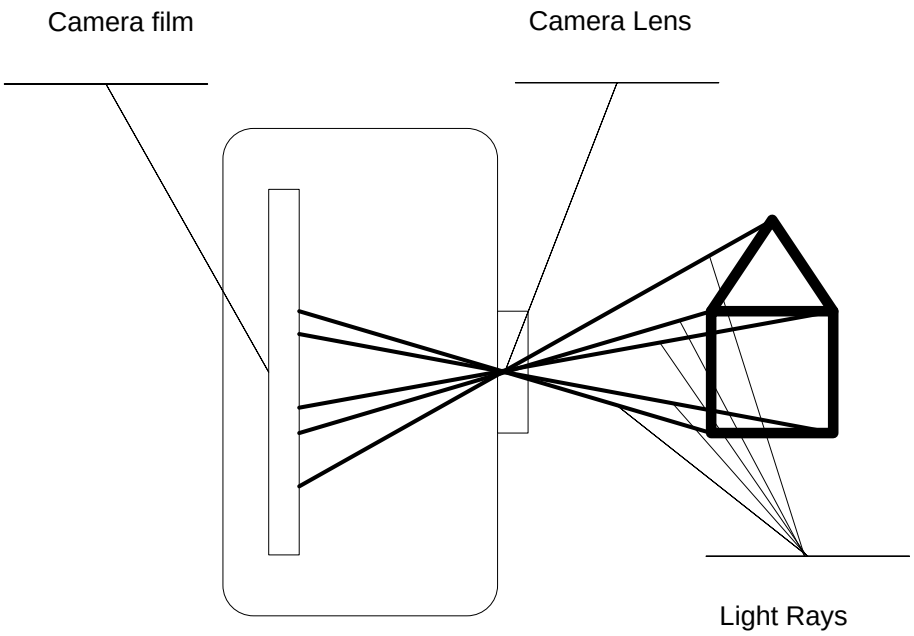
1.3 DISADVANTAGES

All the realistic and beautiful things discussed earlier that we can perform using ray tracing have one major problem. This problem is the costly computations that we have to perform to achieve such result.

In other words, for every pixel, we have to send at least one ray to the world to compute its intensity value (color). Plus, at the point of intersection of this ray with the world's objects, more rays have to be sent in case of reflection, refraction, shadowing… and those more rays continue recursively…

It is always possible to optimize the Ray Tracing algorithm in a way to have a real-time simulation in 3D. But, the more we optimize, our original algorithm becomes less simple and more complex.
If in the near future, every pixel or group of pixels is equipped with a processor, then the Ray Tracing technique will prevail over the ordinary projection based (world-to-screen) method.

1.4 CAMERA CONCEPT

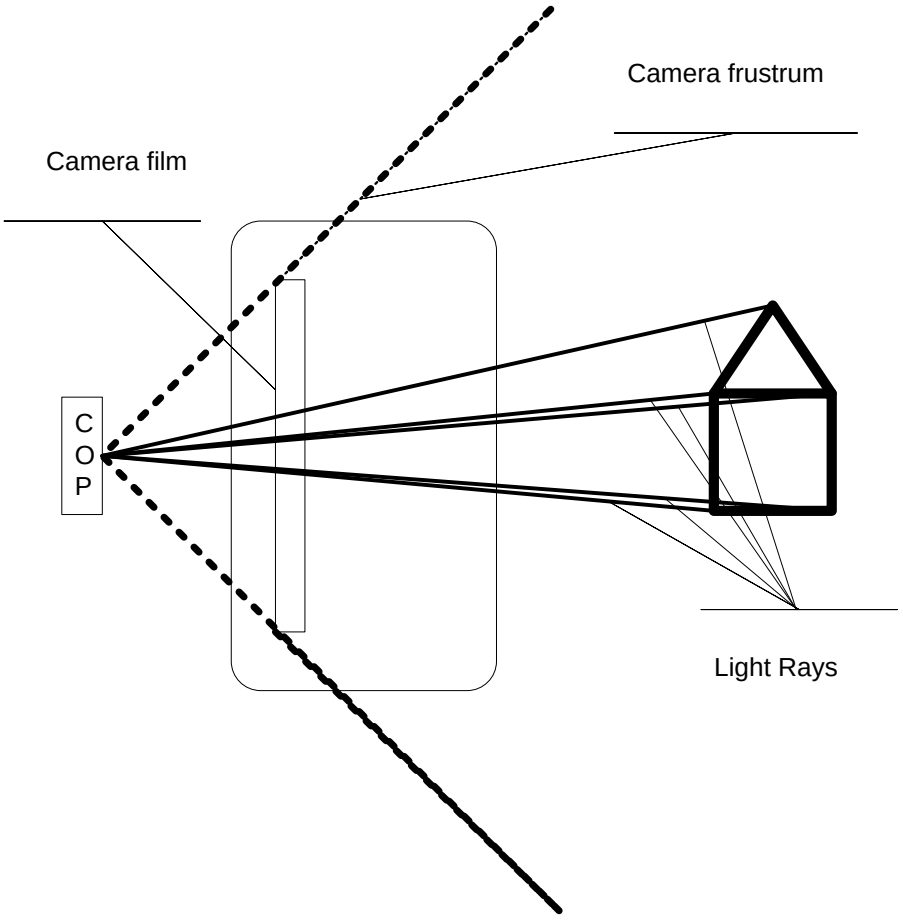


The camera concept is based on allowing rays of light to hit the negative (Camera Film) through the camera lens.

Whenever the camera lens shutter is open, light rays will hit the film negative causing a chemical change that will result in a certain color. If the shutter is open for a longer period, then the image get brighter since more light is coming in. If the camera lens pinhole is larger, then each bit of the negative film receives more light rays and consequently the image becomes blurry.

The computer graphics synthetic camera moves the camera film or the projection plane in front of the lens or the center of projection. Of course the synthetic camera design does not work in real life, but it has the same component as the real camera where each ray of light passes through the eye or center of projection. Since the projection plane or the camera film is located in front of the eye, the light ray needs to pass through the projection plane.

Camera Frustum
It is the portion of a cone or pyramid which remains after its upper part has been cut off by a plane parallel to its base, or which is intercepted between two such planes



In the synthetic camera, all objects visible to the camera are located within the viewing frustum.

1.5 FORWARD RAY TRACING

Naturally, light rays travel from light sources. Light sources are considered as point lights, directional lights or spot lights.

Directional Light

When a light source is far away the light rays coming from the light source are close to parallel to each other. It looks like all the light rays are coming from the same direction, regardless of where the object and/or the viewer is. When a light source is modeled to be infinitely far away it is called a directional light since all its light rays have the same direction; it is independent of the location of the light source.

A fine example of a directional light source is the sun as we know it. The sun is not infinitely far away from us, but it is so far away that we can perceive it as being infinitely far away in the lighting calculations.

Because all the light rays are parallel it does not matter how each object relates to the light source's position since the light direction remains the same for each object in the scene. Because the light's direction vector stays the same, the lighting calculations will be similar for each object in the scene.

We can model such a directional light by defining a light direction vector instead of a position vector.

```
struct Light {
    // vec3 position; // no longer necessary when using directional lights.
    vec3 direction;

    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
};
[...]
void main()
{
  vec3 lightDir = normalize(-light.direction);
  [...]
}
```

Point lights

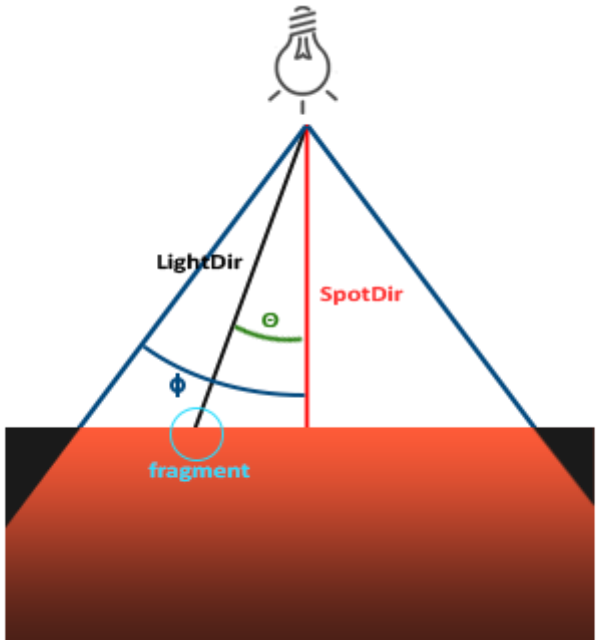
Directional lights are great for global lights that illuminate the entire scene, but we usually also want several point lights scattered throughout the scene. A point light is a light source with a given position somewhere in a world that illuminates in all directions, where the light rays fade out over distance. Think of light bulbs and torches as light casters that act as a point light.

Spotlight

A spotlight is a light source that is located somewhere in the environment that, instead of shooting light rays in all directions, only shoots them in a specific direction. The result is that only the objects within a certain radius of the spotlight's

direction are lit and everything else stays dark. A good example of a spotlight would be a street lamp or a flashlight.

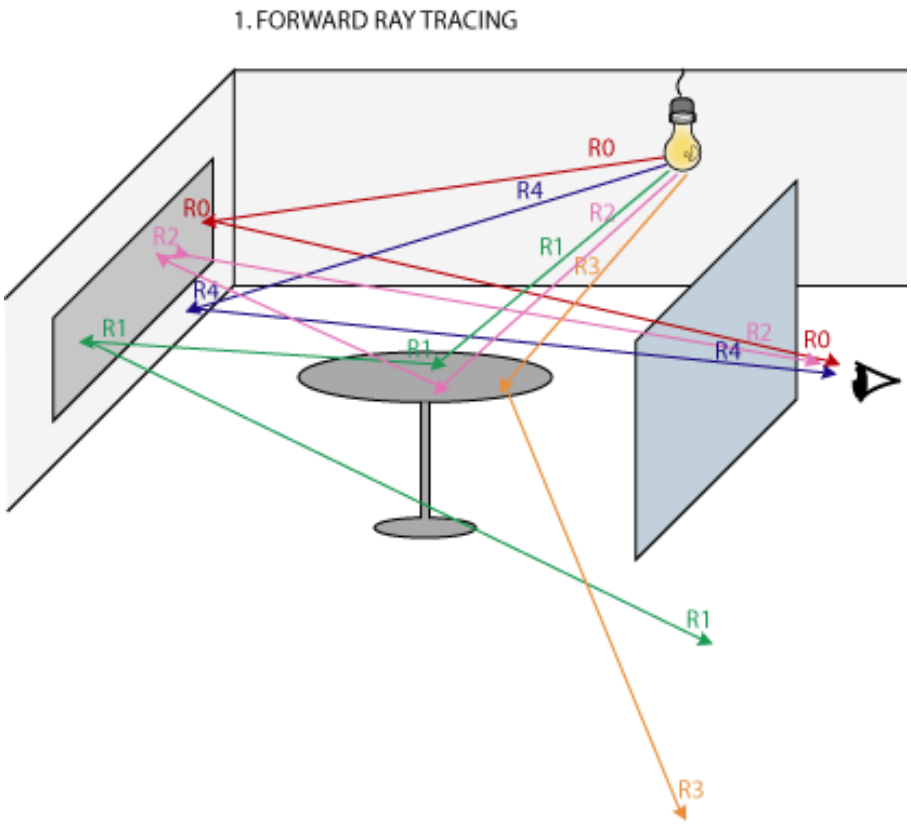
A spotlight is represented by a world-space position, a direction and a cutoff angle that specifies the radius of the spotlight. For each fragment we calculate if the fragment is between the spotlight's cutoff directions (thus in its cone) and if so, we lit the fragment accordingly. The following image gives you an idea of how a spotlight works:



Flashlight

A flashlight is a spotlight located at the viewer's position and usually aimed straight ahead from the player's perspective. A flashlight is basically a normal spotlight, but with its position and direction continually updated based on the player's position and orientation.

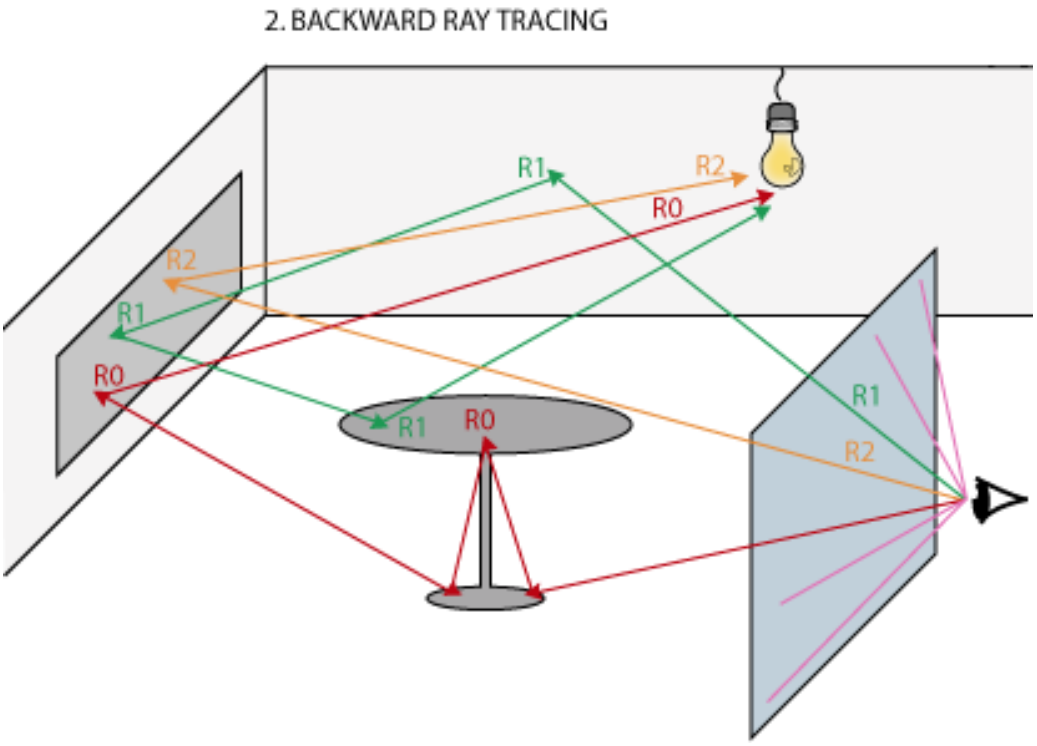
The process is to consider each light ray starting from each light source and trace its ray. Then upon intersection with an object, the object color is reflected according to the incident light ray and the light ray is reflected or refracted into a new direction based on the object material. Eventually, some light rays will end up hitting the camera film and many others will miss it. Consequently, a lot of ray intersection and light calculations would be wasted on rays that will never hit the camera film.



The figure shows clearly that ray1 and ray3 miss the viewing plane. In reality, the problem is much more severe. In order to hit all the viewing plane pixels starting from light sources, a lot of rays must be traced.

1.6 BACKWARD RAY TRACING

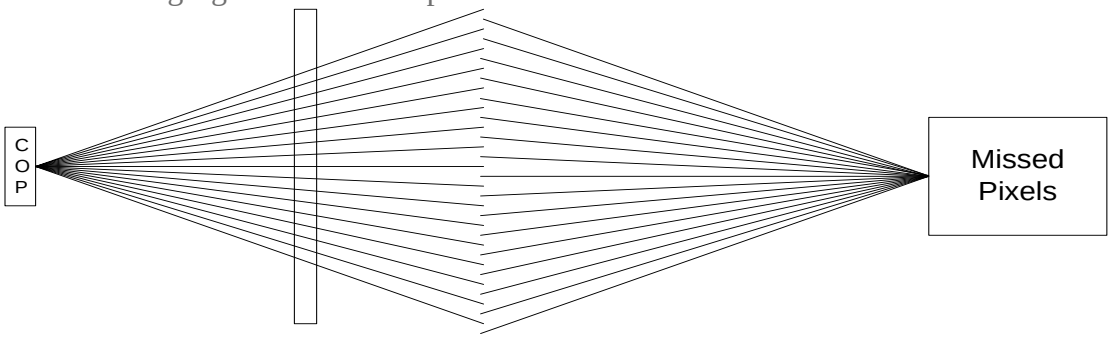
Instead of applying the ray tracing algorithm the natural way, the Ray Tracing algorithm will adopt the backward ray tracing instead of the forward ray tracing. In this way, we will consider all the rays starting from the eye through all the camera film or projection plane and trace the ray recursively until it hits a light source. In this way, we limit our study to the rays that are of interest to us.



1.7 PIXELS AND RAYS

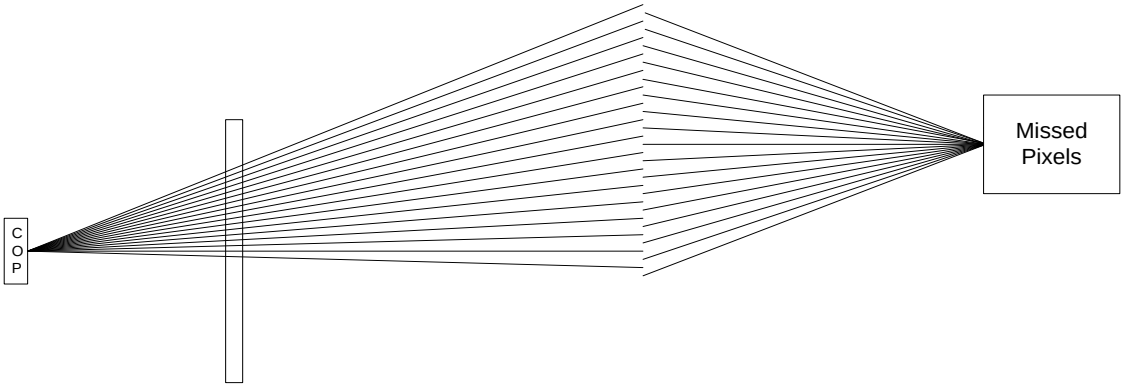
The Rasterization process converts a geometrical primitive from an infinite representation of points to a finite representation of pixels. The same concept is found in the Ray Tracing or in the screen-to-world method. But still, one question is asked: How many rays do we need to send from the eye for each pixel? In the best case, sometimes one ray is enough and sometimes we need an array of light rays for each pixel in order to avoid wrong color calculation.

The following figure shows how pixels can be missed:



A typical solution to the problem would be to send more rays for each pixel on the projection plane, but still some pixels would be missed.

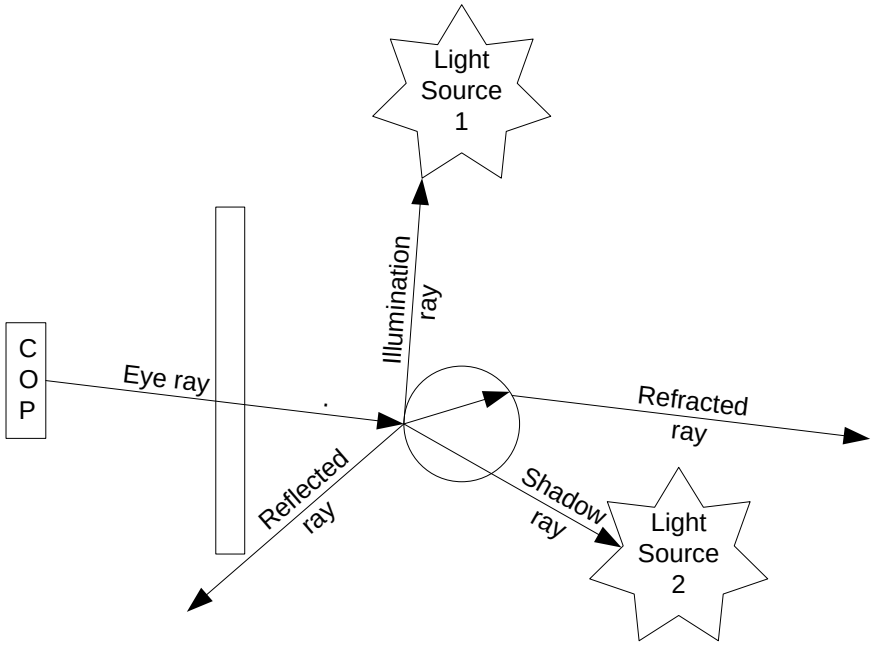
The following figure shows how pixels can be missed at a higher resolution:



1.8 RAY TYPES

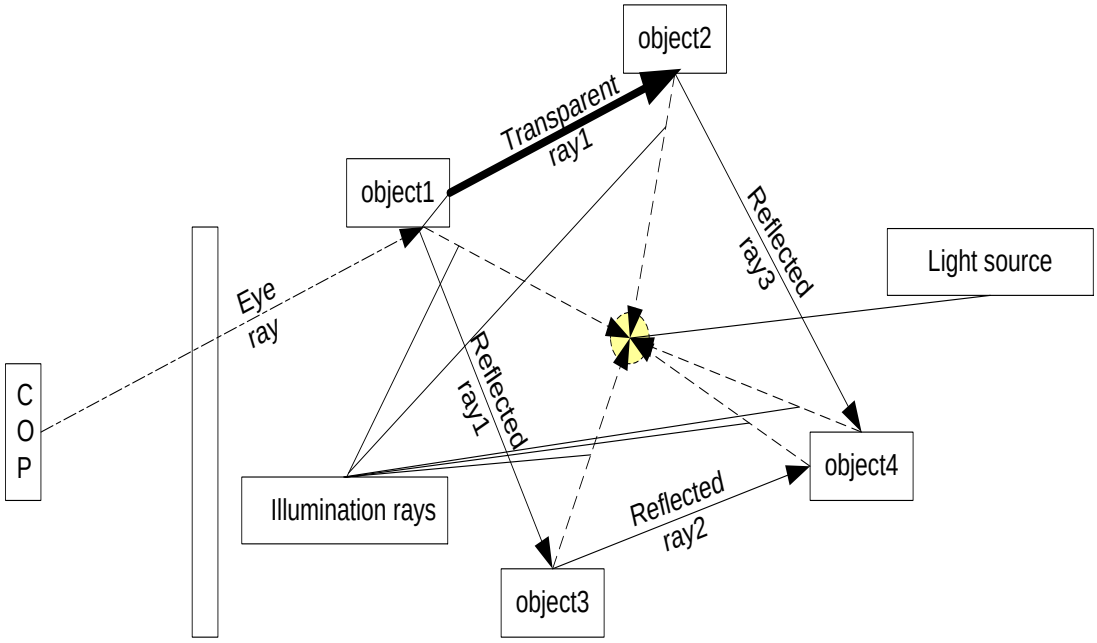
There are four different types of rays:

1. Eye Rays: These are the original rays that are sent from the eye to the projection plane (original rays).
2. Light Rays (illumination rays): These are the direct lights coming from the light source directly to the object surface. If the rays intersect with an object before arriving to the light, these rays will be called Shadow Rays.
3. Reflected Rays: These are the rays that are reflected off the object's surface.
4. Refracted Rays: These are the rays that are refracted from the object's surface.



1.9 RECURSIVE RAYS

When a ray is followed and an object intersection point P is found, we need to find the illumination at point P. In other words, since we are following the ray backwards, our goal is to find the illumination of the light leaving the object which is the opposite direction of the incident ray. The illumination at point P is a combination of different kinds of light arriving at point P. Now what is the color of the different lights arriving at point P from source S? Again, it is a combination of different kinds of lights arriving at point S. Again, we may ask what is the color of the different lights arriving at point S from source S' and so on.
According to the previous paragraph, the natural solution of a recursive problem is to adopt a recursive algorithm where the final illumination is based on the recursive combination of the reflected and refracted colors at each intersection point starting from the light source.



In this example, we assume that each pixel corresponds to one ray.

Object1 reflects rays at 30% and refracts rays at 60%
Object2, object3, object4 just reflect rays at 30%
Let p1 be the intersection point of the eye ray with object1
Let p2 be the intersection point of the transparent ray1 with object2
Let p3 be the intersection point of the reflected ray1 with object3
Let p4 be the intersection point of the reflected ray2 with object4
Let p5 be the intersection point of the reflected ray3 with object4

When the eye ray intersects with object1 at p1, the transparent ray1, reflected ray1 and an illumination light are generated. Consequently, 100% of p1's color is added to 60% of p2's color and 30% of p3's colour.
Colorp1 = (100% (object1 & light Source at p1))+(30% colorp3)+(60% colorp2)

Reflected ray1 intersects with object3 at p3, reflected ray2 and an illumination light are generated. Consequently, 100% of p3's color is added to 30% of p4's color.
Colorp3 = (100% (object3 & light Source at p3))+(30% colorp4)

Reflected ray2 intersects with object4 at p4, the recursion ends here and an illumination light is generated. Consequently, 100% of p4's color is calculated.
Colorp4 = (100% (object4 & light Source at p4))

In this way, colorp4 contributed by 9% to the final pixel color at p1 and colorp3 contributed by 30% to the final pixel color at p1. Please note that colorp1 is attenuated before it is blitted.

Transparent ray1 intersects with object2 at p2, reflected ray3 and an illumination light are generated. Consequently, 100% of p2's color is added to 30% of p5's color.
Colorp2 = (100% (object2 & light Source at p2))+(30% colorp5)

Reflected ray3 intersects with object4 at p5, the recursion ends here and an illumination light is generated. Consequently, 100% of p5's color is calculated.
Colorp5 = (100% (object4 & light Source at p5))

In a more complex example, we can have more than one light source. In this case we have an additional illumination light from each point to the new light. Also, we could have the reflected light hitting directly the light source. In this case, the transparent light is considered as an illumination light but only 60% of the light is taken into consideration.
The example shows clearly the recursion, where the contribution of individual ray down the tree contributes less and less to the final pixel. The example could be translated into the following tree: