Challenge 4: Ray tracing

Jan José Hurtado Jauregui

June 27, 2016

1 Objective

Implement Ray Tracing algorithm.

2 Background

In the field of computer graphics, generating a 2D image based on a 3D scene, is an important task for a lot of applications. To address this task, there exist a lot of techniques, which depend on the problem where they will be used. Sometimes a real time solution is needed, and sometimes this is not important because you want realistic images. Ray Tracing is an algorithm proposed, initially, in the latter (realistic way). It is a simplification of the physic model of a simple camera capture.

It works as follows. Given an imaginary eye, which is the starting point, a ray should be traced through a virtual screen. This screen represents what we will see on the device's screen. So, we only need one ray for each pixel to obtain the resulting image. We have to continue tracing each ray and detect if it interesects with an object in the scene. If that happens, the pixel value will be calculated, regarding the object color and properties, and the illumination model of the scene. In Figure 1 we can see the general workflow of the algorithm, where camera represents the imaginary eye.

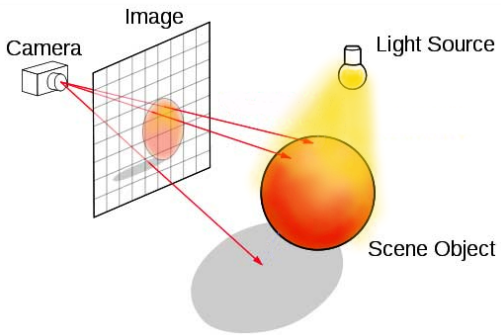


Figure 1: Ray tracing workflow

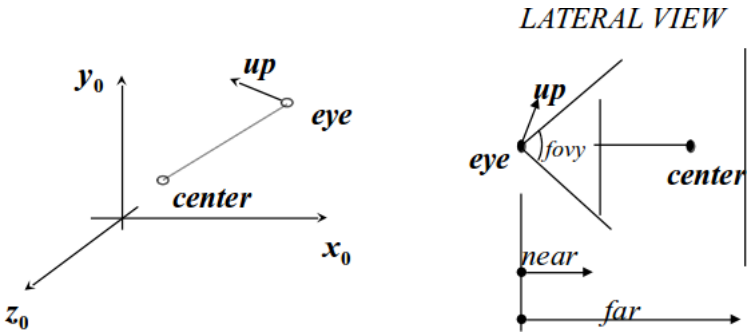


Figure 2: Intrinsic description of the camera

The algorithm can be described with pseudocode of Algorithm 1.

Algorithm 1 Ray tracing

```
for each screen pixel
  trace a ray
  for each object of the scene
    compute intersection of this object
    assign nearest intersection
  if the ray intersected an object
    compute the color
    assign this color to the pixel
```

The camera (imaginary eye) can be described using an extrinsic definition and an intrinsic definition. Remember that we have to use a simple camera model. So, we have the position of the camera (eye), the point of view of the camera (center) and a vector which determines the orientation of the camera (up). Also we need an angle of view (fovy) and the distance of the imaginary screen (near). In Figure 2 we show how it looks like. This is an intrinsic description.

Based on the height (h_p) and width (w_p) of the screen, and on the extrinsic description of the camera, we can compute some intrinsic values. The local space of the camara can be described as follows.

$$z_e = \frac{1}{||eye - center||}(eye - center)$$

$$x_e = \frac{1}{||up \times z_e||}(up \times z_e)$$

$$y_e = z_e \times x_e$$

Also we can define the focal distance (d_f), the height (h) of the screen in the global system and the width (w) of the screen in the global system.

$$d_f = n$$

$$h = 2d_f \tan(\frac{fovy}{2})$$

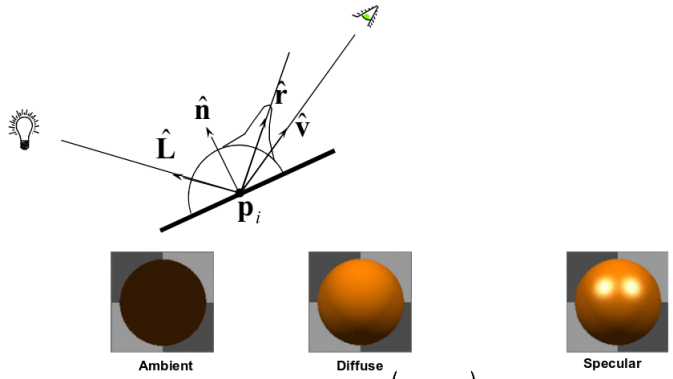


Figure 3: Illumination

$$w = \frac{w_p}{h_p} h$$

A ray can be defined as $o + td$, where o is the initial point and d is the direction vector of the ray. In the our case we use $o = eye$ and d depends on which pixel we trace. So, it can be defined as follows.

$$d = -d_f z_e + h(\frac{y}{h_p} - \frac{1}{2}) y_e + w(\frac{x}{w_p} - \frac{1}{2}) x_e$$

Usually, 3D objects are represented using polygon meshes. Triangular meshes are a good choice due its properties. To detect if the ray collides with an object, we have to detect if the ray collides with any of its triangles. Also we have to consider only the nearest collision. To address this problem we have to compute the normals of all triangles, then, if the ray collides with the plane described by the normal, it is a candidate. To deduce if the collision point is inside of the triangle we can parametrize it in barycentric coordinates of the respective triangle.

To calculate the pixel value (color), we have to consider the illumination model. In Figure 3 we show the general scheme and formula. As a first term we have the ambient illumination C_{amb_λ}, which simulates the light reflection in all directions. The second term $C_{amb_\lambda} k_{dif_\lambda} (\hat{n} \cdot \hat{L})$ represents the diffused illumination, where C_{amb_λ} is the color of the light, k_{dif_λ} is the material of the object, $\hat{n}$ the normal of the point and $\hat{L}$ a vector in light source direction. The third term $C_{amb_\lambda} k_{s_\lambda} (\hat{r} \cdot \hat{v})^n$ represents the specular reflection, where C_{amb_λ} is the color of the light, k_{s_λ} is a reduction factor, and n determines the brightness.

An object occluding another regarding the light source, produces a shadow. So, if a point is occluded, we are going to consider only the term of the ambient light. In Figure 4 we can see this case. We can trace a shadow ray in light source direction, such that, if it intersects an object, the corresponding point must be a shadow.

Antoher feature we can consider, is the reflection. Given an intersected point, if the surface has a reflection factor, the color of the corresponding pixel should reflect the object in the corresponding direction (see Figure 5). When

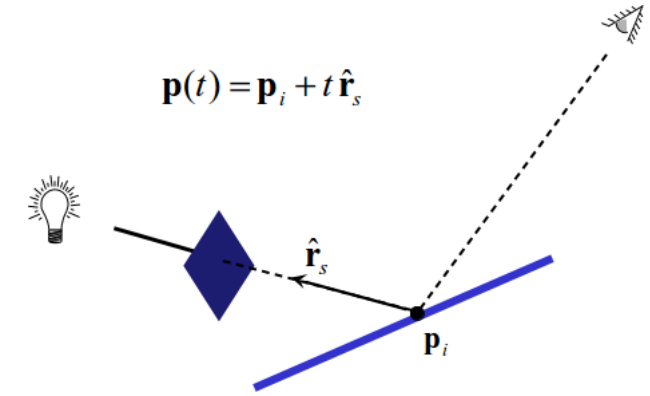


Figure 4: Shadow

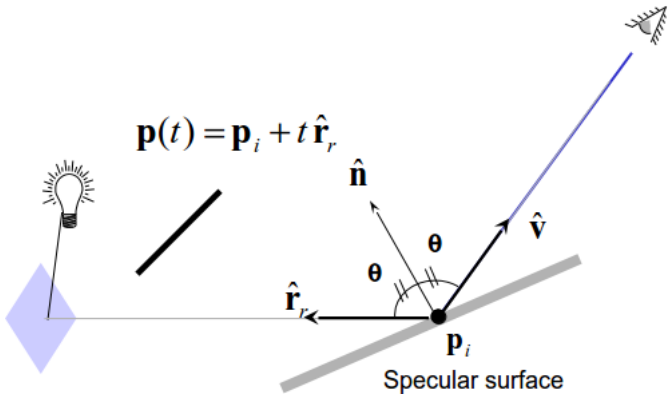


Figure 5: Reflection

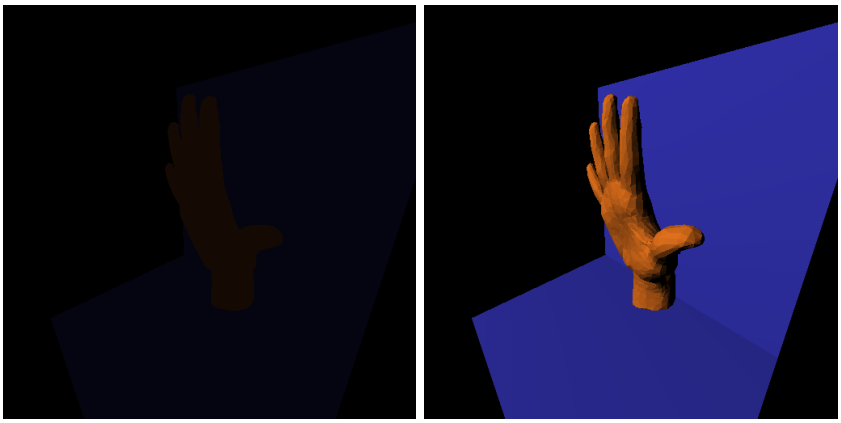
tracing the initial ray, after the collision, we can continue tracing this ray in the direction described in the Figure. So, as a resulting image we can see the reflected image in this position. We can use multiple levels of reflection.

3 Experiments

As first experiment we show the behavior of the implemented ray tracing using different specifications (see Figure 6).

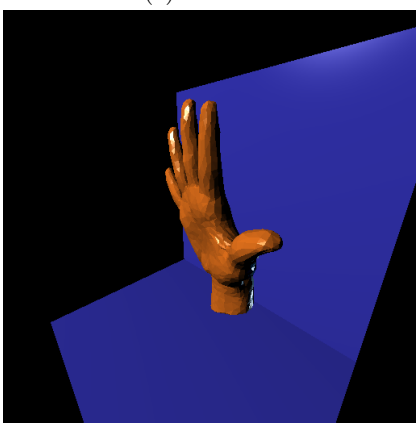
As second experiment, we compared the implemented algorithm with the Z-Buffer algorithm. We used different views (see Figure 7). Each row is a view, the first column represents Z-Buffer, the second column represents Ray tracing without shadow and reflection, and the third column represents the full version of Ray tracing using 2 ray levels (reflection level 1).

Also, we show some examples in Figure 8.



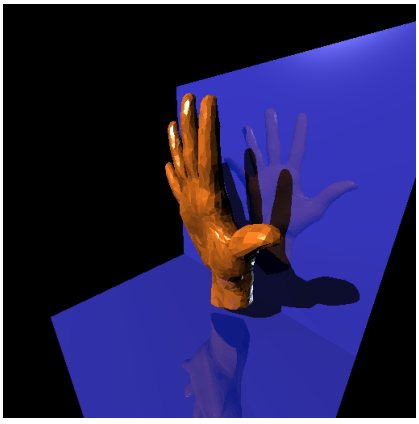
(a) Ambient

(b) Ambient + Diffuse

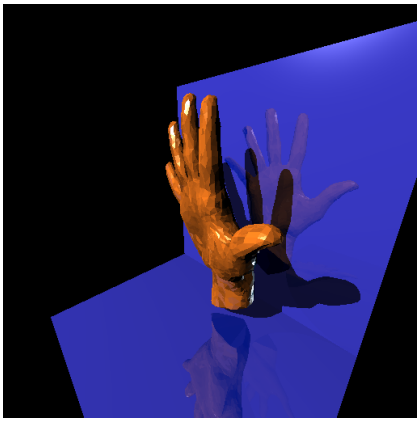


(c) Ambient + Diffuse + Specular

(d) Shadow

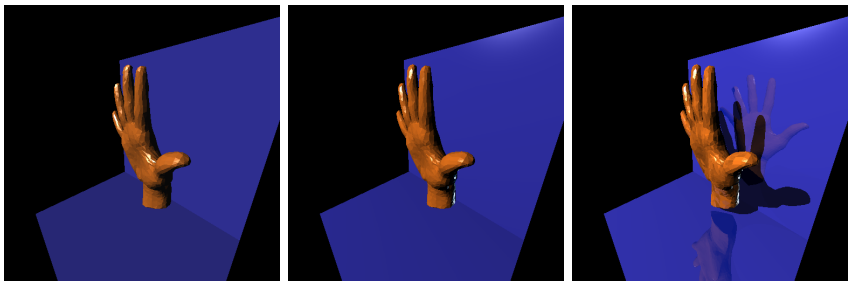


(e) Reflection level 1

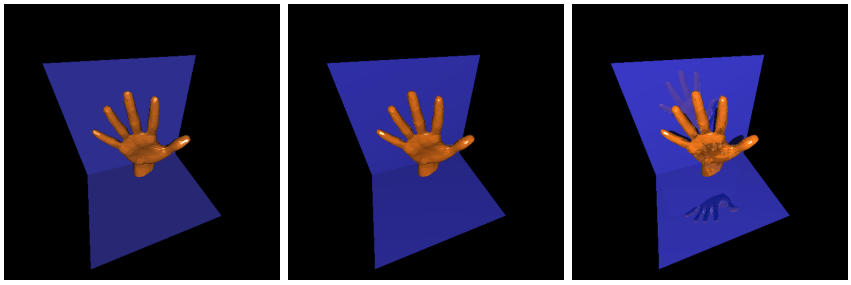


(f) Reflection level 2

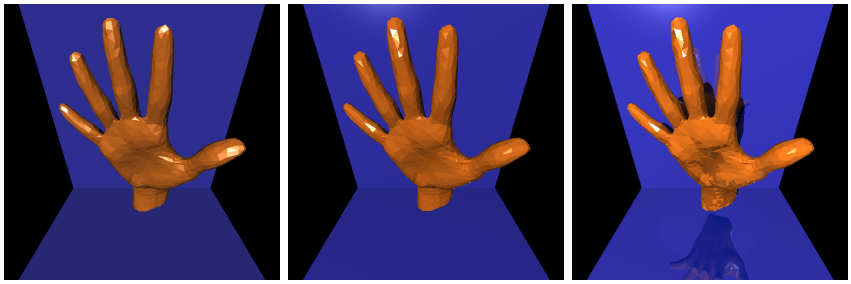
Figure 6: Ray tracing behavior



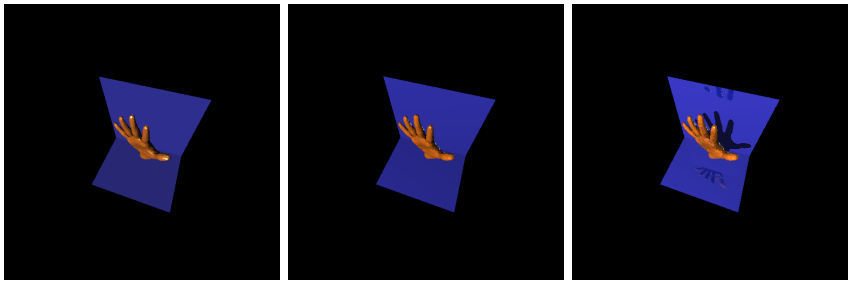
(a) Z-Buffer (b) Ray tracing (c) Ray tracing full



(d) Z-Buffer (e) Ray tracing (f) Ray tracing full



(g) Z-Buffer (h) Ray tracing (i) Ray tracing full



(j) Z-Buffer (k) Ray tracing (l) Ray tracing full

Figure 7: Comparisson of different views between Z-Buffer algorithm and Ray Tracing.

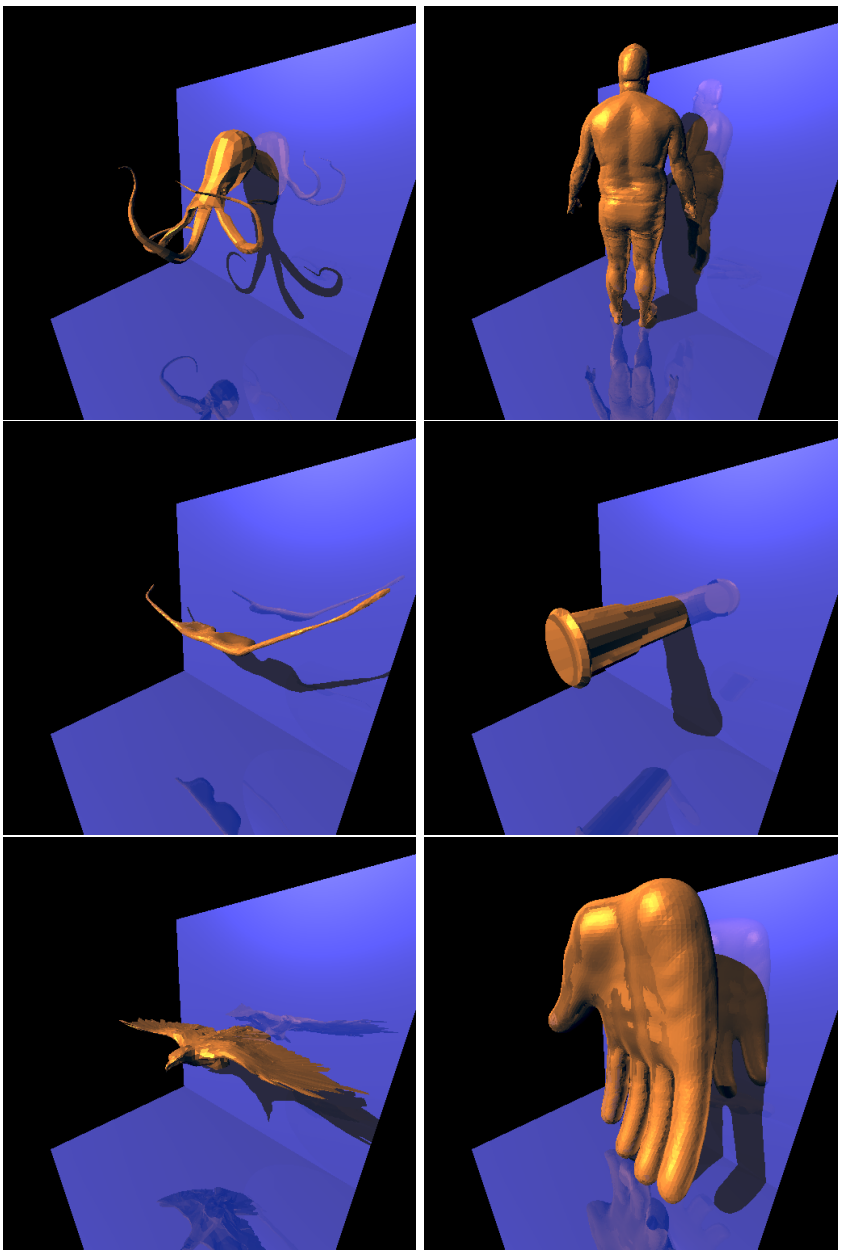


Figure 8: Examples.