Table of Contents

Advanced Ray Tracing

Techniques that make images produced with ray tracing more realistic are important. For instance, the capability to form objects more complex than the generic objects we have defined so far is necessary. In addition, transmission of light is more than just a local phenomenon and hence there is a need to understand the non-local transport of light in phenomena such as translucency and reflection.

Ray Tracing Compound Objects

If we could combine in some way a number of interacting generic shapes, we could probably create arbitrarily complex shapes out of simple ones. One way of performing this is to consider the idea of boolean operations on generics. In set theory, operations such as union, intersection, and difference exist, and they can be applied to sets of generic shapes to compose more complex objects, as long as these objects share volume in the space defined by the world coordinates system. With generic shapes, boolean operations amount to:

- **Union**: Merging objects together: $A \cup B$
- **Intersection**: Object formed by the portion common to two or more objects: $A \cap B$
- **Difference**: Subtraction of one object from another: $A - B$ or, more formally $A \cap \bar{B}$.

An Example

Consider two spheres with a common volume between them. Is there a way to only ray trace their common intersection? There is, and it follows the idea of hit times per object. By keeping a hit time list for each object and labeling the intersections that way allows to define the boolean operations listed above on generic shapes. For example, consider two intersecting spheres such as these:

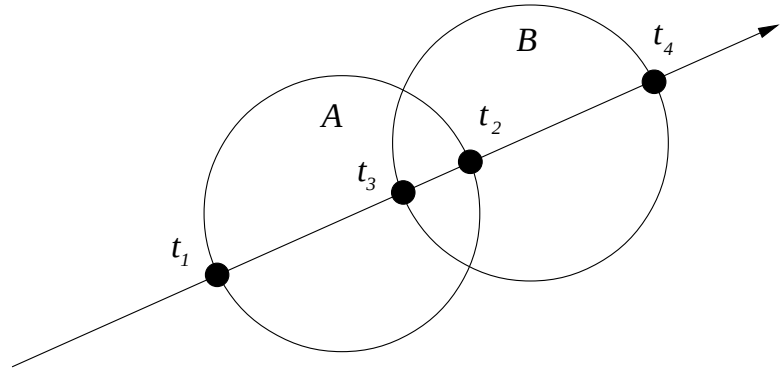


Illustration 1: Ray intersecting a compound object

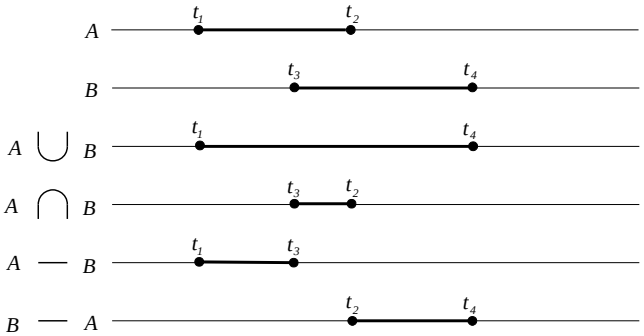
Note that the labeling of the hit times is kept per object. As we intersect sphere A we find t_1 and t_2 . The same happens with sphere B . However in this case t_3 happens to be smaller than t_2 . The ray is in the intersection of the two spheres from t_3 to t_2 , and the hit time (for local shading) is t_3 . If this object defined by the intersection of the two spheres is not opaque, then we have to consider light reflection and refraction as non-local elements of the rendering. The set of t values for which a ray is inside an object is called its inside list.

Algorithm

Consider a number of generic shapes, and a ray. The algorithm consists of the following steps:

- Construct a list of times (as in t values) at which the ray enters and exits from an object A , ordered in such a way that times are increasing
- Since A is a solid generic shape, enter and exit times alternate. The same is true for all other objects involved
- Build a list of enter/exit times for all objects
- For all objects in the compound object, form a set of t value intervals for times the ray is inside an object
- Perform the required boolean operations on these intervals

With the three boolean operators, we can build four different objects. Consider two objects that we ray-trace as a compound object, with the following enter/exit times for each of them:



Performing boolean operations on objects is equivalent to performing the same operations on lists of intervals.

Compound Object Representation

A compound object is always the combination of two (possibly compound) objects, and a binary tree provides a natural description. For example, consider the compound object:

$$((A \cap B) - C) - D$$

This compound object can now be represented as a binary tree in the following way:

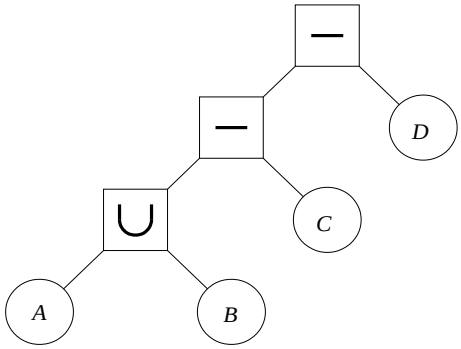


Illustration 2: Binary expression tree

in which the internal nodes of the tree are operators and leaf nodes are generic objects. Traversing this tree in-order and computing inside sets at each internal node will produce the desired final inside set. Operator precedence with boolean operators is, in decreasing precedence: $(\neg, \cap, \cup)$.

An interesting example is to create a spherical bowl as a compound object. We could use two spheres, one slightly smaller than the other, located at the same point, and a cone, that we would use to carve out the opening of the bowl:

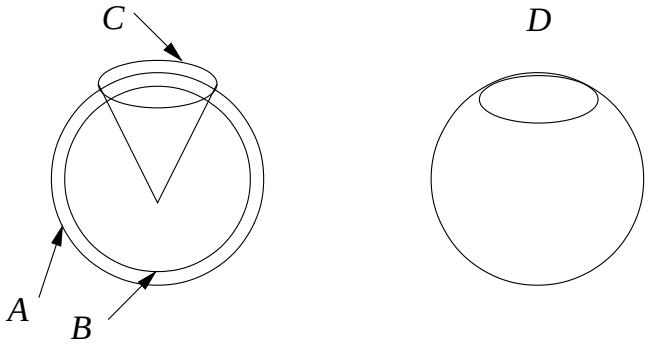


Illustration 3: Defining a compound object

The resulting compound object is defined by the following boolean operations on two spheres A and B and a cone C :

$$D=(A-B)-C$$

The solid sphere A is hollowed out by removing all the points from the inner sphere B. The top is opened by removing all the points in the cone C.

Decomposing Compound Shapes

Given a visual description of a compound object, it is possible to express this object with boolean operations on generic shapes. For instance, the following compound object:

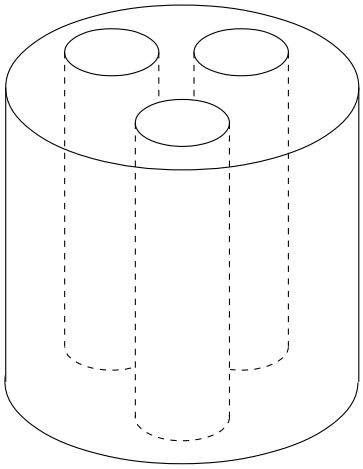


Illustration 4: Cylinder with three holes

Can be created using the following generic shapes, conveniently placed at appropriate locations:

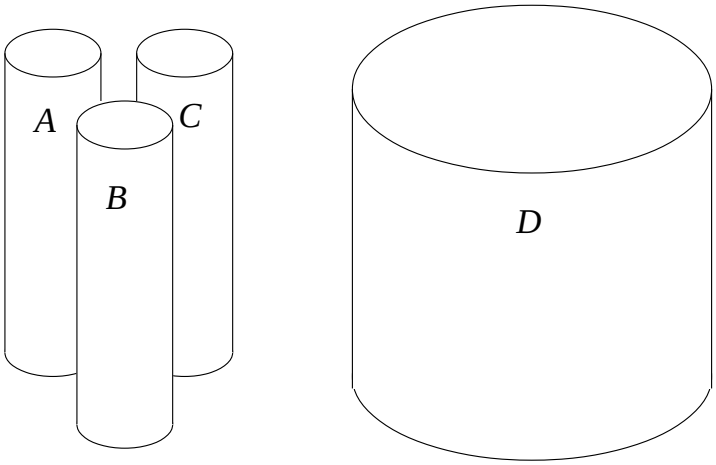


Illustration 5: Required generic shapes

The boolean equation, from which inside lists can be formed, is easily written as

$$D-A-B-C$$

We then can use the hit list per object, sort them out on axes, and perform the boolean operations specified by the user.

Light Reflection

When the surface of an object is mirror-like or transparent, or both, the light that reaches the eye may have up to five components:

$$I = I_a + I_d + I_s + I_l + I_r$$

where the first three local components are the usual ambient, diffuse, and specular contributions. The fourth term, I_r, is from reflected light that is incident at $\vec{r}(t_0)$, along direction $-\vec{l}$ where

$$\vec{l} = \vec{r}(t_0) - 2(\vec{r}(t_0) \cdot \vec{n})\vec{n}$$

To compute this light contribution, we need to cast a ray from $\vec{r}(t_0)$ in direction $\vec{l}$ and find an intersection with a scene object, if any. If an intersection is found, then the local ambient, diffuse, and specular contributions are computed at that intersection, and another ray is cast in the same way, to also find out the contribution of reflected light at this point. The process of chaining multiple ray castings in this way is usually coded in a recursive manner.

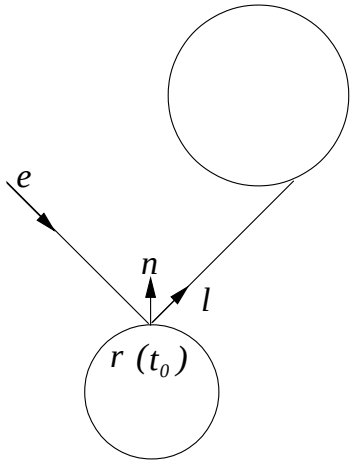


Illustration 6: Light reflection at hit point

If an object is able to reflect incident light coming from other surfaces, then we should be able to render it realistically. The following is a recursive algorithm that does this (assuming a single light source):

```
color_t shade(matrix_t e, matrix_t d, k) {

if (intersection r(t0) = e + dt0 with an object) {
  color = local shading components ;
  if (recursion level k > 0) {
    if (object has non-zero reflection coefficient) {
      compute direction of reflection l ;
```

```
        color_add(color,reflection coefficient*shade(r(t0),l,k-1)) ;
      }
    }
  }
  else {
    color = background color ;
  }
  return(color) ;
}
```

The recursion level in this algorithm is controlled by the depth of the recursion calls. However, the recursion could also be controlled by other means, such as when the contribution of the reflected light becomes smaller than a certain user-specified threshold. Graphically, the level of recursion can be displayed as in the following figure:

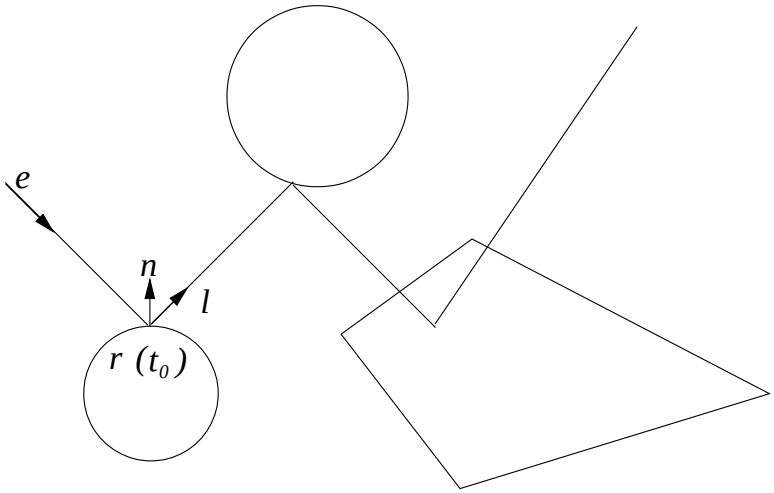


Illustration 7: Light reflection with recursion level set to 3

Light Refraction

The component I_r is the transmitted light component that is transmitted through transparent material and hits the intersection point of the ray and an object. The light transmitted through transparent material bends as it goes through the material.

When a ray of light strikes a transparent object, a portion of the ray penetrates the object. This light ray will change direction if the speed of light changes from the current medium to the transparent medium it penetrates. Suppose light from medium 1 hits the surface of a translucent medium 2. The vector of the light ray and the normal vector to medium 2 form a plane in which the direction of refracted light will also lie. If the angle of incidence is θ_1 (angle between the ray of light and the translucent surface normal), then Snell's law defines the angle of refraction θ_2 as:

$$\sin(\theta_2)=\frac{c_2}{c_1}\sin(\theta_1)$$

where c_1 is the speed of light in medium 1 and c_2 that of medium 2. The ratio

$$\frac{c_2}{c_1}$$

is called the index of refraction of medium 2 with respect to medium 1. Note that if the angle of incidence is zero, so is the angle of refraction. Light striking a medium interface at a right angle does not bend.

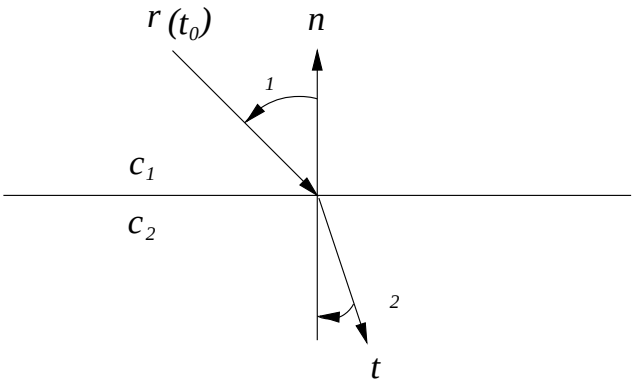


Illustration 8: A ray of light bent by entry into a different medium

Rays of light are bent more to the direction of the surface normal when they enter a medium with a lower speed of light ($c_2/c_1<1$). The reverse is true when light goes into a medium with faster speed of light. Hence, when the light goes from a slower medium into a faster one, it is physically impossible for θ_2 to exceed 90 degrees. Hence, when θ_1 passes the point where it pushes θ_2 to 90, a total internal reflection occurs and no light is refracted. Since

$$\sin(\theta_1)=\frac{c_1}{c_2}\sin(\theta_2)$$

angle θ_1 takes its largest possible value when $\theta_2=90$ degrees, or $\theta_1=\sin^{-1}(c_1/c_2)$. For example, when light moves from water to air we have $c_1/c_2=0.7519$, so the critical angle for water is 48.75 degrees. This explains a lot of things you may have witnessed at the lake.

The direction vector $\vec{t}$ dictating the path of refracted light is obtained with the translucent surface normal, the ray direction $\vec{d}\,t$ from the ray $\vec{r}(t)=\vec{e}+\vec{d}\,t$, and c_1 and c_2 :

$$\vec{t} \;=\; \frac{c_2}{c_1}\vec{d}+\left[\frac{c_2}{c_1}\vec{n}\cdot\vec{d}-\sqrt{1-\left(\frac{c_2}{c_1}\right)^2(1-(\vec{n}\cdot\vec{d})^2)}\right]\vec{n}$$

The phenomenon of total internal reflection manifests itself within this equation when the quantity in the square root is negative.

There are crucial aspects that need to be considered when implementing light refraction in a ray tracing application. We examine them in the following section.

Considerations on Light Refraction

Suppose the ray hits an object that is translucent. The algorithm then computes the direction $\vec{t}$ of refracted light using $c_1 = 1$ for air, and obtaining c_2 from the properties of the object. The new ray is built, the shading function is then called recursively and ultimately returns a color which is scaled by the transparency coefficient of the object, and added to the colors accumulated so far.

When the ray exits the surface of a translucent object, it does so into the air. Because the ray is inside the object, the normal to the hit point must be reversed in sign, as the normal must be defined into the medium in which the ray is traveling. In addition, when a ray is inside an object, we assume it is not bathed in light and thus no local ambient, diffuse, or specular local contributions are computed upon the ray exiting the translucent material back into the air.

The inside wall of the translucent object may be reflective and therefore a reflection ray may be cast back into the object.

Upon exit of the refracted ray from the object, the value c_1 is obtained from the translucent object properties and c_2 is set to 1 as the ray makes it back to air. If the angle of incidence is less than the critical angle, a new ray is spawned and sent on its way recursively.

Optimizing the Ray-Tracing Process

Ray-Tracing is computationally intensive. Fortunately, there are ways to optimize the process without giving every pixel a dedicated processor. One of the simplest methods is to consider extents. The extent of an object is a shape that encloses the object. It speeds up the ray-tracing by revealing when the current ray could not possibly hit a particular object.

Since we are ray-tracing with an inversely transformed ray, the intersections are computed with untransformed objects. Hence, we could define an extent such as the generic cube, formed of six planes, that encloses all other generics (sphere, plane, cylinder, cone, etc.) and intersect the ray coming from the center of projection of the synthetic camera with the generic cube first. If we obtain an intersection, then there is a high probability that the ray also intersects with the generic object it contains. This speeds up the process if computing the ray intersection with a generic cube is faster that with the generic object it contains. Because the generic cube has a simple shape, computing a ray intersection with it is faster than with other generics, such as the sphere, for instance.

This rather crude method can be refined to gain even more efficiency. If we consider the 8 corners of the generic cube extent containing a generic object, we can transform its corners with the transformation matrix M associated with the generic. As a result, the transformed cube encloses the object, as placed in the scene (and possibly distorted) by the scene designer. Following this, we can project the 3D corner points of the cube onto the near plane, where the pixels are, and compute the 2D bounding rectangle that contains them. At this point, we know that there is a high probability of a ray intersection with the generic object If the ray is cast through a pixel located within the 2D bounding rectangle. This 2D rectangle is known as the projective extent of an object.

Repeating this process with each object, we obtain the sum of the 2D projective extents for the objects forming the scene. We then have a list of objects for each pixel to ray-trace that

we can look up to. The algorithm is as follows:

- for each generic object in the scene
  - transform its corresponding generic cube extent corners (8) by the object matrix M
  - compute the projection of these points onto the near plane, in pixel coordinates
  - compute the 2D bounding box that contains the 8 points
- Start the ray-tracing process and for each pixel
  - if the ray is within the bounding box of one or more objects, perform the intersection computation only for these objects

This algorithm can achieve significant computational savings, especially with objects such as polygonal meshes. However, it does not provide any advantage for the rays we cast for reflection and refraction, as there is no screen pixels at these intersection points to project any object extent. To solve this problem, we must resort to 3D-space partitioning techniques.

Binary Space Partitioning (BSP) trees provide an answer. We first consider the volume of the scene that we need to ray-trace. As we have seen, this volume is a parallelepiped with right angles, after the projective transformation performed by the synthetic camera. This space is then partitioned into 8, equal right-angled parallelepipeds. If no object is present in any of these sub-volumes, we do not partition it any further. On the other hand, we continue the partitioning for each sub-volume containing (entirely or partially) any object from the scene, up to a minimal preset volume size.

Once the ray-tracing space is partitioned in this way (the tree representing these subdivisions is called an octree), it becomes possible to consider a secondary ray (emanating from reflection or refraction) spawned from the surface of an object, and determine which sub-volumes it intersects with and which ones of these contain an object to ray-trace. This is an advanced technique that provides the rest of the optimizations required for ray-tracing algorithms to perform adequately on complex 3D scenes.