Lecture 11 and 12: Ray tracing

So far we have only discussed algorithms in which are based on so-called *object-order rendering*. In this scenario the rendering can be described as follows: Each object is processed in turn and its visibility is determined using algorithms such as z-buffering. After determining its visibility, the surface is projected onto the view plane and rendered. In this scenario it is difficult to model any physical effects such as shadows, reflection and transparency which are caused by the interaction between objects. More importantly, this scenario make implicitly the assumption that the light travels from the visible objects towards the eye. This raises the following question:

Do light rays proceed from the eye to the light or from the light to the eye?

To look at this question in more detail, it is important to understand some of fundamentals about light rays:

1. Light rays travel in straight lines.
2. Light rays do not interfere with each other if they cross.
3. Light rays travel from the light source to the eye but the physics are invariant under path reversal.

An alternative to *object-order rendering* techniques are so-called *image-order rendering techniques*. One of these image-order rendering techniques is called ray tracing. Another image-order rendering technique is called volume rendering and will be discussed in more detail in lectures 15 and 16. Ray tracing allows to add considerably more realism to the rendered scene since it can be used to add shadow effects, transparency and reflections. The basic idea of ray tracing is the following: For each pixel on the screen, a ray is defined by the line joining the viewpoint and the pixel on the viewing plane (perspective projection) or by a line orthogonal to the view plane (orthographic projection). Each ray is then cast through the viewing volume and checked for any intersections with the objects inside the viewing volume. An example of a ray tracing scenario is shown in Figure 1.

For each ray we cast through the viewing volume, we need to test which surfaces of objects are intersecting the ray. If a surface is intersected, we calculate the distance from the pixel to the visible surface intersection point. The smallest calculated intersection distance identifies the visible surface for the pixel. Rays from the pixel to the nearest surface are referred to as *primary rays*. We also reflect the ray off the visible surface along the specular path (angle of reflection equals angle of incidence). If the surface is transparent, we also send a ray through the surface in the refraction direction. Reflection and refraction rays are referred to as *secondary rays*. An example of a ray tracing scenario with primary and secondary rays is shown later in Figure 10. At each intersection point we calculate the surface normal and the specular, ambient and diffuse reflection from the surface (see lecture 7).

Surface calculations for ray tracing

The key problem in ray tracing is that for each ray we must calculate all possible intersections with objects inside the viewing volume. In the following we will show how it is possible to calculate the intersection of rays with various geometric primitives. Before, we must define a geometric description of rays: In general, a ray can be described by a point $\mathbf{p}_0$ and a unit direction vector $\mathbf{d}$ as shown in Figure 2. The coordinates of any point along the ray can be calculated as follows:

$$\mathbf{p}(\mu) = \mathbf{p}_0 + \mu\mathbf{d} \tag{1}$$

Initially, we may define $\mathbf{p}_0$ as the current pixel on viewing plane $\mathbf{p}_i$. The $\mathbf{d}$ direction vector can be obtained from the position of the pixel $\mathbf{p}_i$ through which the ray passes, and the viewpoint $\mathbf{p}_v$:

$$\mathbf{d} = \frac{\mathbf{p}_i - \mathbf{p}_v}{|\mathbf{p}_i - \mathbf{p}_v|}$$

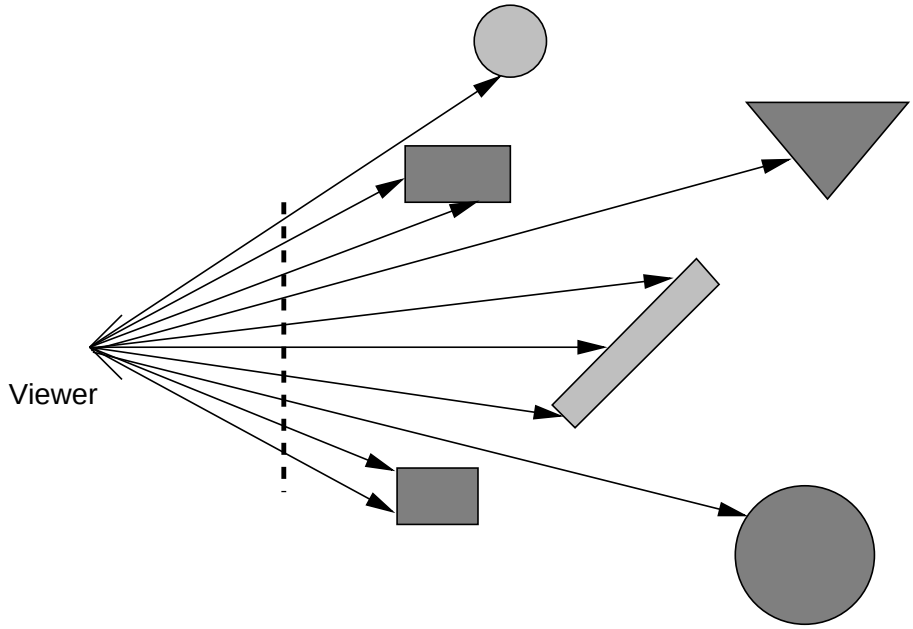

Figure 1: Ray tracing is a method which casts a separate ray from the viewpoint through each pixel on the viewing plane. For each ray the nearest intersection with an object determines the visible surfaces.

In this notation, $\mu > 0$ denotes the part of the ray behind the viewing plane (visible part) and $\mu < 0$ denotes the part of the ray in front of the viewing plane (invisible part). To find a visible point of intersection of a ray with an object, we must always test whether $\mu > 0$ at the intersection point.

Sphere

One way of describing objects is the use of solid models such as spheres, boxes, cylinders and other objects whose geometric shapes can be described easily. A simple object for ray tracing is a sphere. A sphere can be described by its centre $\mathbf{p}_s$ and a vector $\mathbf{q}$ originating at the centre as shown in Figure 3. The points on the surface of the sphere can be described by:

$$|\mathbf{q} - \mathbf{p}_s|^2 - r^2 = 0$$

where r is the radius of the sphere. To test whether a ray intersects a surface, we can substitute $\mathbf{q}$ using the definition of a ray in equation (1) and obtain:

$$|\mathbf{p}_0 + \mu\mathbf{d} - \mathbf{p}_s|^2 - r^2 = 0$$

Setting $\Delta\mathbf{p} = \mathbf{p}_0 - \mathbf{p}_s$ and expanding the dot product produces the following quadratic equation

$$\mu^2 + 2(\mathbf{d} \cdot \Delta\mathbf{p})\mu + |\Delta\mathbf{p}|^2 - r^2 = 0$$

which has the following solution for μ:

$$\mu = -\mathbf{d} \cdot \Delta\mathbf{p} \pm \sqrt{(\mathbf{d} \cdot \Delta\mathbf{p})^2 - |\Delta\mathbf{p}|^2 + r^2}$$

If the quadratic equation has no solution, the ray does not intersect the sphere. If the quadratic equation has two solutions and we assume that $\mu_1 < \mu_2$, the value μ_1 corresponds to the intersection point at which the ray enters the sphere while μ_2 corresponds to the point at which the ray exits the sphere.

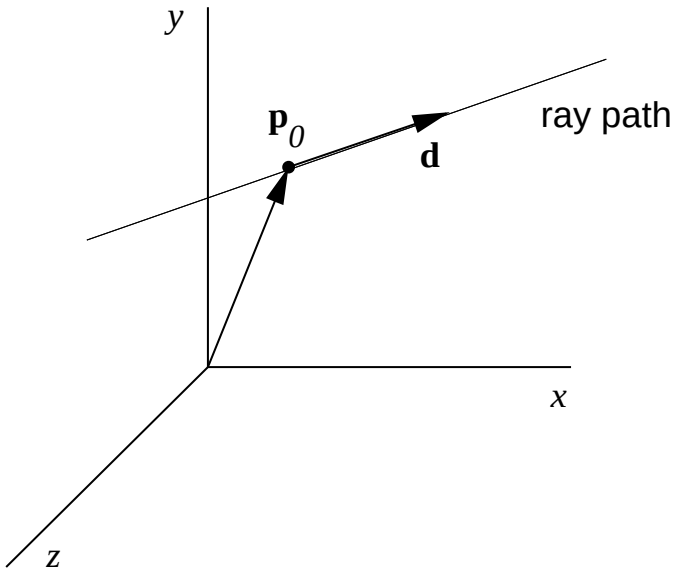

Figure 2: A ray with an initial position vector $\mathbf{p}_0$ and a direction vector $\mathbf{d}$.

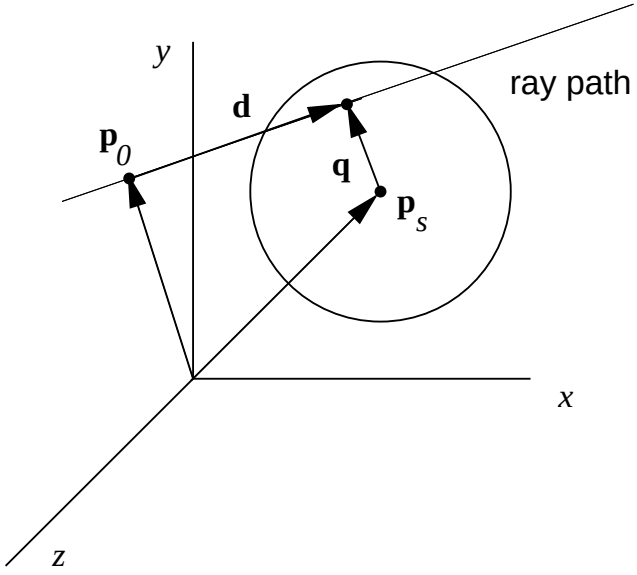

Figure 3: A ray intersecting a sphere.

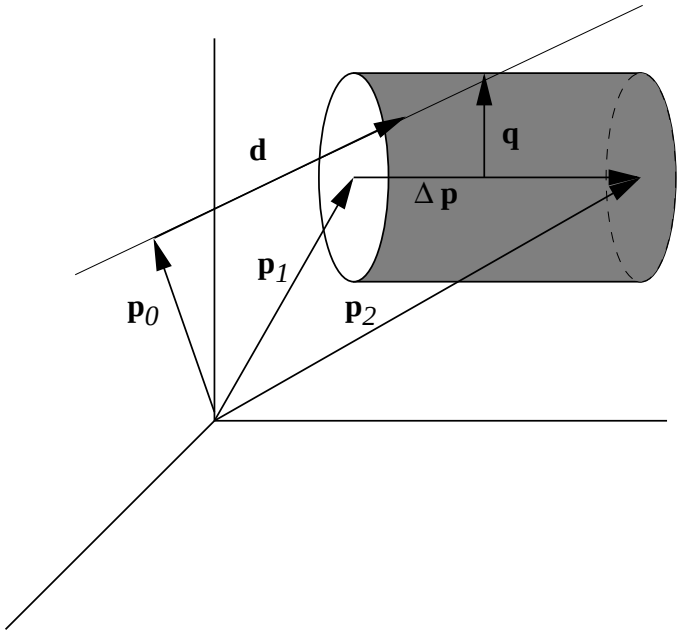

Figure 4: Calculating the intersection of a ray with a cylinder.

Cylinder

A more complex solid model for ray tracing is a cylinder. A cylinder may be defined by two position vectors $\mathbf{p}_1, \mathbf{p}_2$ which specify the two endpoints of the cylinder and its radius r (see Figure 4). The axis of the cylinder can be written as $\Delta\mathbf{p} = \mathbf{p}_1 - \mathbf{p}_2$. Assuming $\mathbf{q}$ is a radial vector, we obtain

$$\mathbf{p}_1 + \alpha\Delta\mathbf{p} + \mathbf{q} = \mathbf{p}_0 + \mu\mathbf{d} \tag{2}$$

Since $\mathbf{q} \cdot \Delta\mathbf{p} = 0$ we can write

$$\alpha(\Delta\mathbf{p} \cdot \Delta\mathbf{p}) = \mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}$$

$$\alpha = \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}$$

Substituting in equation (2) we obtain

$$\mathbf{q} = \mathbf{p}_0 + \mu\mathbf{d} - \mathbf{p}_1 - \left(\frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}\right)\Delta\mathbf{p}$$

Using the fact that $\mathbf{q} \cdot \mathbf{q} = r^2$, we can use the same approach as before to solve a quadratic equation for μ. If the equation has no solution, the ray does not intersect the cylinder. If the equation has two solutions $\mu_1 \leq \mu_2$, it is still necessary to check whether the intersection occurs between the two endpoints of the cylinder. For this we need to solve the following equation

$$\alpha_1 = \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu_1\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}$$

If the value of α_1 is between 0 and 1, it follows that there is an intersection on the outside surface of the cylinder. Otherwise it is necessary to compute α_2 using

$$\alpha_2 = \frac{\mathbf{p}_0 \cdot \Delta\mathbf{p} + \mu_2\mathbf{d} \cdot \Delta\mathbf{p} - \mathbf{p}_1 \cdot \Delta\mathbf{p}}{\Delta\mathbf{p} \cdot \Delta\mathbf{p}}$$

If the value of α_2 is between 0 and 1 it follows that there is an intersection on the inside surface of the cylinder.

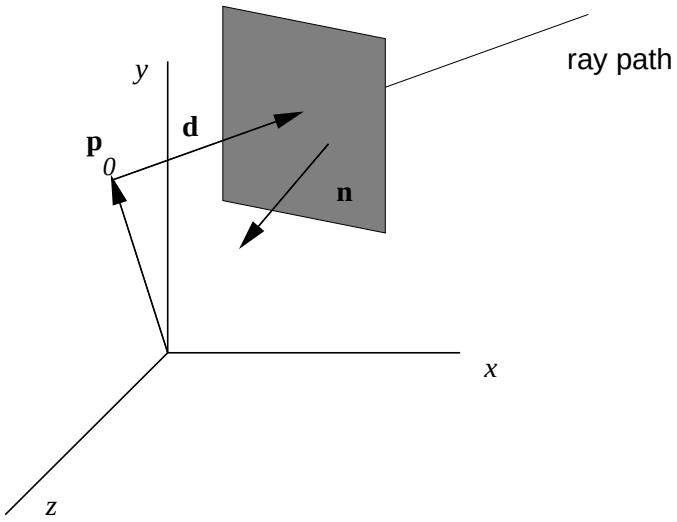

Figure 5: Calculating the intersection of a ray with a plane.

Plane

Even though the calculation of the intersection of rays and solid models is relatively simple, many objects cannot be represented easily with solid models. In practice, the surfaces of most objects will be described by a set of triangles or planar polygons. In order to test whether a ray intersects a triangle or polygon, we need to test whether the ray will intersect the plane defined by the triangle or polygon. The intersection point of a ray and a plane is given by

$$\mathbf{p}_1 + \mathbf{q} = \mathbf{p}_0 + \mu\mathbf{d}$$

where $\mathbf{p}_1$ is a point in the plane. Since the surface normal $\mathbf{n}$ is perpendicular to the plane (see Figure 5), multiplying with $\mathbf{n}$ yields:

$$\mathbf{q} \cdot \mathbf{n} = 0 = (\mathbf{p}_0 - \mathbf{p}_1) \cdot \mathbf{n} + \mu\mathbf{d} \cdot \mathbf{n}$$

Solving for μ yields the point at which the ray intersects the plane:

$$\mu = -\frac{(\mathbf{p}_0 - \mathbf{p}_1) \cdot \mathbf{n}}{\mathbf{d} \cdot \mathbf{n}}$$

Triangle

To calculate the intersection of a ray with a triangle we can use the intersection test described before to test whether the plane defined by the triangle is actually intersected by the ray. However, in order to reduce the computational complexity we can first test whether the triangle is front facing or not and only then calculate the intersection of the plane with the ray. To determine whether a triangle is front facing or not, we can use the following test:

$$\mathbf{d} \cdot \mathbf{n} < 0$$

If the triangle is front facing we proceed and calculate the intersection of the ray with the plane defined by the triangle. In the final step we must calculate whether the intersection point is inside the triangle or not.

Any point $\mathbf{q}$ inside a triangle can be described as follows:

$$\mathbf{q} = \alpha\mathbf{a} + \beta\mathbf{b} \tag{3}$$

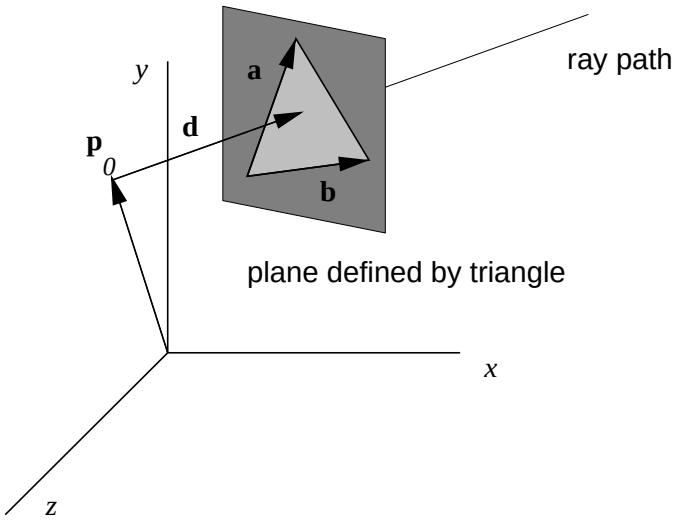

Figure 6: Calculating the intersection of a ray with a triangle.

Here $\mathbf{a}$ and $\mathbf{b}$ are two edge vectors defining the triangle (see Figure 6). The scalar multipliers α, β must satisfy the following three conditions:

$$\begin{aligned} 0 \leq \alpha \leq 1 \\ 0 \leq \beta \leq 1 \\ \alpha + \beta \leq 1 \end{aligned} \tag{4}$$

Using the point of intersection $\mathbf{q}$ and taking the dot products of both $\mathbf{a}$ and $\mathbf{b}$ with equation (3), we can derive the following two expressions for the two scalar multipliers α, β:

$$\alpha = \frac{(\mathbf{b} \cdot \mathbf{b})(\mathbf{q} \cdot \mathbf{a}) - (\mathbf{a} \cdot \mathbf{b})(\mathbf{q} \cdot \mathbf{b})}{(\mathbf{a} \cdot \mathbf{a})(\mathbf{b} \cdot \mathbf{b}) - (\mathbf{a} \cdot \mathbf{b})^2}$$

$$\beta = \frac{(\mathbf{q} \cdot \mathbf{b}) - \alpha(\mathbf{a} \cdot \mathbf{b})}{\mathbf{b} \cdot \mathbf{b}}$$

After calculation of α and β one can check whether the three conditions of equation (4) hold.

Polygon

As for triangles, we can first test whether the polygon is front facing or back facing. If the polygon is front facing we calculate the intersection of the ray with the plane defined by the polygon. Then we need to test whether the intersection point with the plane is contained by the polygon. This can be achieved by projecting the 3D polygon and the intersection point onto the $x - y$ plane and calculating whether the intersection point is inside the 2D polygon.

Space-subdivision methods

By far the largest amount time during ray tracing is spent on the calculation of intersections with objects in the scene. One way of reducing the number of intersection calculations is to enclose groups of adjacent objects into a bounding volume such as a sphere or box (see Figure 7). For example, a face represented by a large set of polygons may be enclosed in such a bounding box. During the ray tracing, each ray is

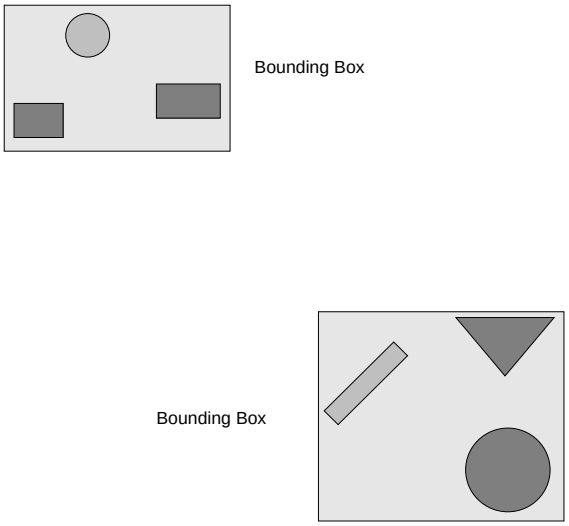

Figure 7: Grouping of objects into a bounding box for ray tracing.

first tested whether it intersects the bounding box. If the ray does intersect the bounding box, the ray will not intersect any of the polygons. If the ray intersects the bounding box, each of the objects inside the bounding box must be checked for intersections with the ray.

Another way of speeding up the intersection calculation is to use space subdivision methods. The simplest space subdivision method is the so-called uniform space subdivision which divides the viewing volume into uniform cubes with a pre-defined size (see Figure 8(a)). Each cube has a list of objects which are inside the cube. During the ray tracing, we calculate which cubes will intersect the ray. Only for those cubes which have objects inside we need to calculate intersection points with objects. A more sophisticated space subdivision method is the so-called adaptive space subdivision. Initially, the viewing volume forms a single cube. The cube is then recursively subdivided into eight smaller cubes. The subdivision only stops if a cube contains no other objects or if the cube size reaches a pre-defined limit (see Figure 8(b)).

Recursive ray tracing

So far, we have only discussed how to determine the intersection of rays with objects inside the viewing volume. Once the nearest intersection has been found, we can calculate the ambient, diffuse and specular reflections at the intersection point. Also, the modelling of multiple light sources can be achieved easily, since we can add separate diffuse and specular reflections for each light source. However, the ray tracing would still look like any standard object-order rendering technique. To exploit the advantages of ray tracing, we can start to incorporate new effects such as shadows to make the rendering more realistic.

The modelling of shadows with ray tracing is simple. In practice, shadows can be created as follows: For each ray, we determine the nearest intersection point. For each light source, we then determine whether a new ray starting from the light source to the nearest intersection point is intersected by any other object in the viewing volume. If the ray intersects the any other object, any contribution from this light source is ignored. If it does not intersect any other object, the contribution from this light source is added to the total intensity. Figure 9 shows an example of how shadows may be created by occlusion.

Let us assume that $\mathbf{n}$ is surface normal and $\mathbf{v}$ is the direction vector of the incoming ray. To calculate the

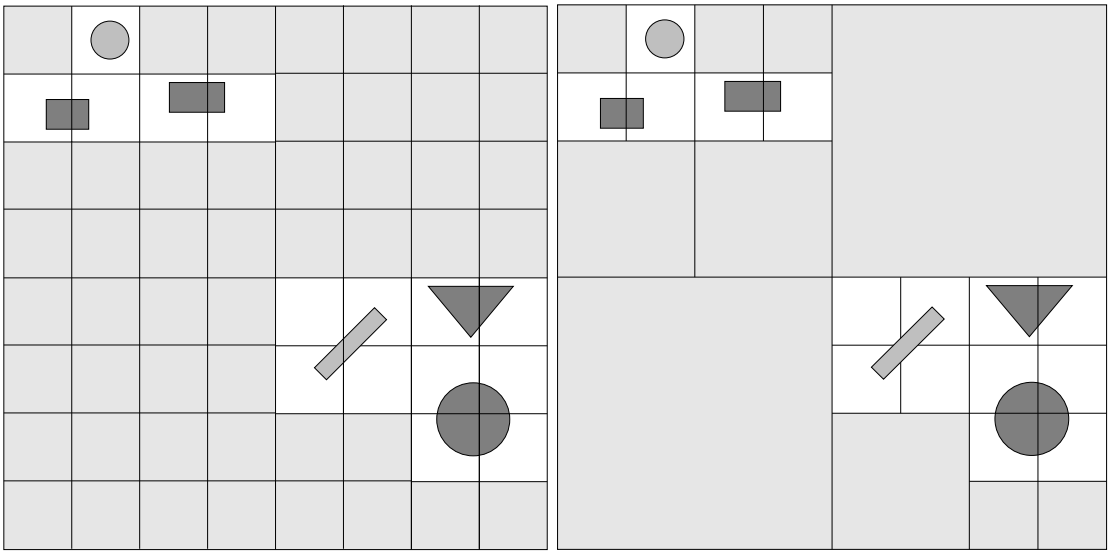

Figure 8: (a) Uniform space subdivision for ray tracing and (b) adaptive space subdivision for ray tracing.

illumination at an intersection point, we can use the following equation:

$$I = k_a I_a + \sum_i s_i I_i \left[k_d(\mathbf{n} \cdot \mathbf{l}_i) + k_s(\mathbf{v} \cdot \mathbf{l}'_i)^t \right] \tag{5}$$

In this illumination model, L_i denotes the colour of light source i, $\mathbf{l}_i$ is the direction vector of the light source and $\mathbf{l}'_i$ is the reflected direction vector from the light source. Here k_a denotes ambient material colour, I_a denotes the ambient colour of the light, k_d denotes the diffuse material colour, k_s denotes specular material colour, and t denotes the specular reflection coefficient.

To incorporate shadows, s_i represents a delta function which indicates whether a light source i is obscured or not:

$$s_i = \begin{cases} 0 & \text{if light source } i \text{ is obscured} \\ 1 & \text{if light source } i \text{ is not obscured} \end{cases}$$

In addition to shadows, ray tracing also allows the generation of other realistic images showing the reflection and transmission properties of objects. This can be achieved as follows: Once the ray tracing procedure has determined the intersection for the primary ray, the procedure can be repeated recursively for each secondary ray which may originate at the intersection. These may be either reflection or refraction rays, or both. Using this scenario we can build a binary ray tracing tree. Figure 11 shows such a binary ray tracing tree for the scene shown in Figure 10. In this binary tree each node corresponds to the intersection between a ray and a surface. Each node may have exactly one branch for reflection rays and one branch for refraction rays. This recursive ray tracing may continue infinitely and may be limited by a maximum recursion depth which determines the speed of processing and the amount of memory needed for storage.

The intensity assigned to a pixel is determined by accumulating the intensity contributions starting a the bottom (leaf nodes in the tree) of the ray tracing tree. Surface intensity from each node in the tree is attenuated by the distance from the parent surface (next higher nodes in the tree) and added to the intensity of the parent surface. The pixel intensity is then the sum of the attenuated intensities of the root node or the ray tracing tree. If no surface intersects the ray, the ray tracing tree will be empty, and the pixel will be assigned the value of the background. If a ray intersects a non-reflecting light source, the pixel can be assigned the intensity of the light source. Figure 12 shows an example of how light may propagate from the leaf nodes to the root node.

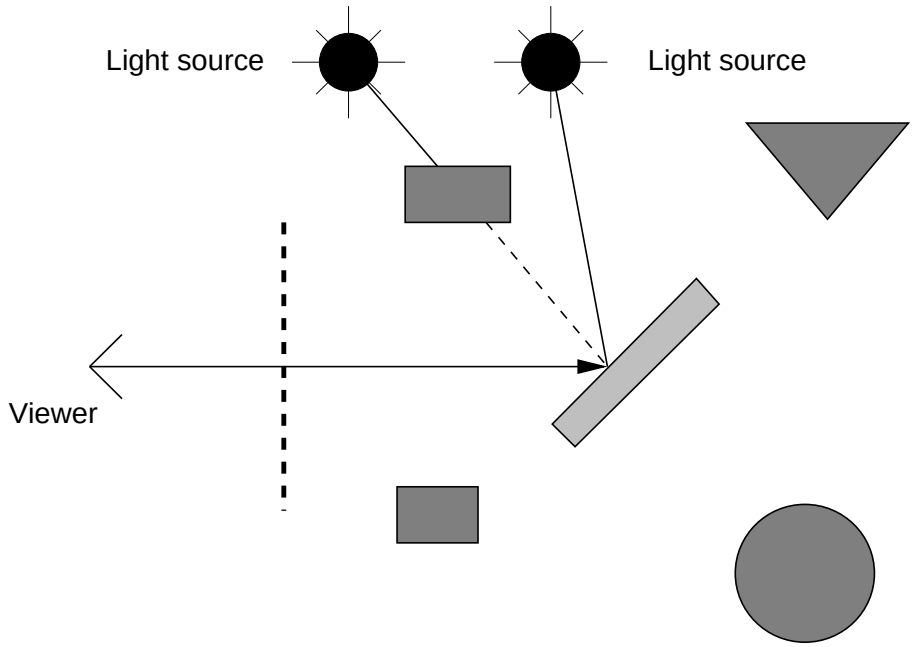

Figure 9: Ray tracing is a method which casts a ray from the viewpoint through a pixel and can model reflections and transmissions.

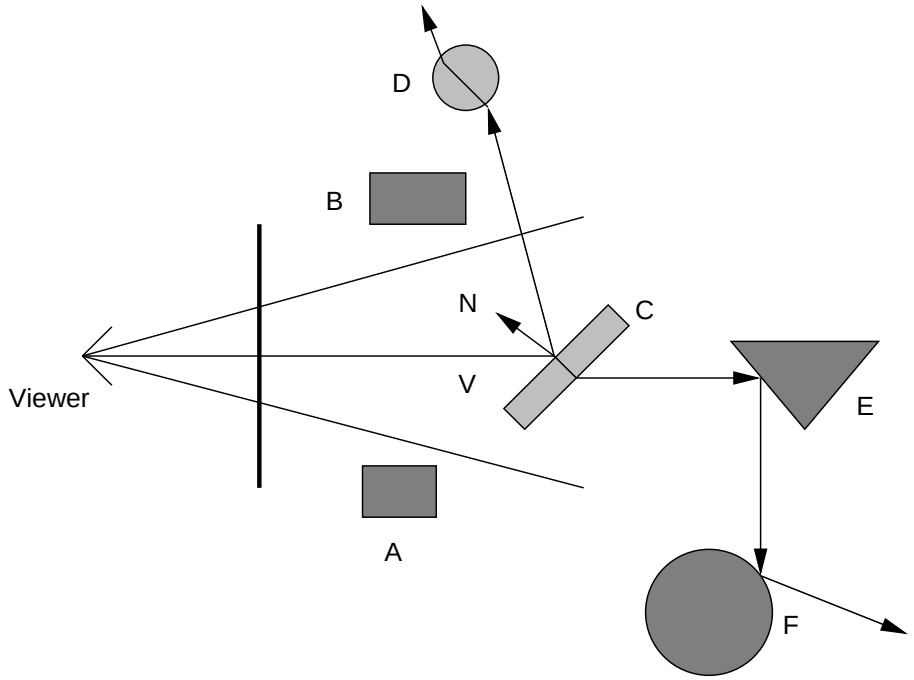

Figure 10: Ray tracing is a method which casts a ray from the viewpoint through a pixel and can model reflections and transmissions.

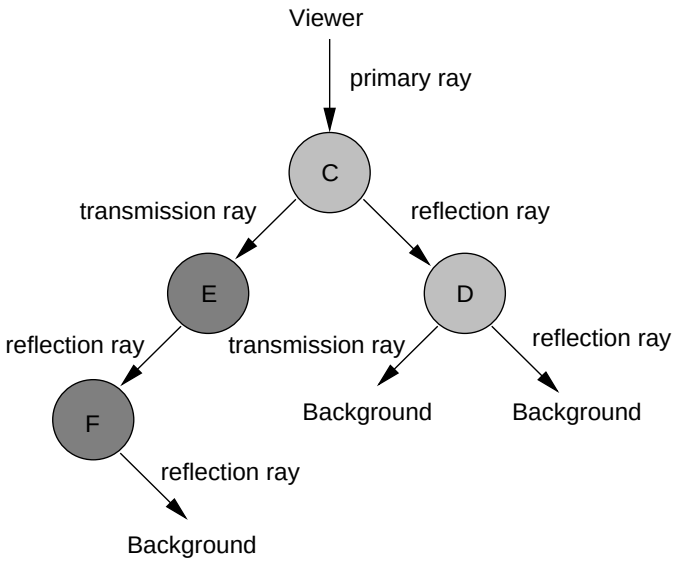

Figure 11: Propagation of rays using recursive ray tracing.

Using a recursive ray tracing algorithm as shown in Figure 14, the illumination at each pixel on the viewing plane can be calculated as follows:

$$I = k_a I_a + \sum_i s_i I_i \left[k_d (\mathbf{n} \cdot \mathbf{l}_i) + k_s (\mathbf{v} \cdot \mathbf{l}_i')^t \right] + k_r I_r + k_t I_t$$

Here, k_r is the reflection coefficient of the transmitted ray, I_r is the intensity of the reflected ray, k_t is the transmission coefficient of the transmitted ray, and I_t is the intensity of the transmitted ray.

Reflection, Refraction and Translucency

In order to calculate the illumation due to reflection, refraction and translucency we must calculate the direction of secondary rays at the intersection of a primary ray with an object. An example of a ray which is reflected from a surface is shown in Figure 13(a). The direction $\mathbf{r}$ of a reflected ray can be expressed as a function of the surface normal $\mathbf{n}$ at the intersection point and direction $\mathbf{d}$ of the incident ray:

$$\mathbf{r} = \mathbf{d} - (2\mathbf{d} \cdot \mathbf{n})\mathbf{n}$$

Figure 13(b) shows an example of a ray which changes direction as it crosses the boundary between one medium and another. To calculate the direction of the transmitted ray, we can use Snell's law which states that:

$$k_1 \sin(\phi_1) = k_2 \sin(\phi_2)$$

Here k_1 is a constant for transmission medium 1 (for example, air) and k_2 is a constant for transmission medium 2 (for example, water). ϕ_1 is the angle between the incident ray and the surface normal, and ϕ_2 is the angle between the refracted ray and the surface normal. Using $\mathbf{d}$ for the direction vector of the incident ray and $\mathbf{r}$ for the direction vector of the refracted ray, Snell's law states that:

$$k_1(\mathbf{d} \times \mathbf{n}) = k_2(\mathbf{r} \times \mathbf{n})$$

From this we can calculate the direction of the refracted ray as

$$\mathbf{r} = \frac{k_1}{k_2} \left(\left[\sqrt{\left[(\mathbf{n} \cdot \mathbf{d})^2 + \left(\frac{k_2}{k_1} \right)^2 - 1 \right]} - \mathbf{n} \cdot \mathbf{d} \right] \mathbf{n} + \mathbf{d} \right)$$

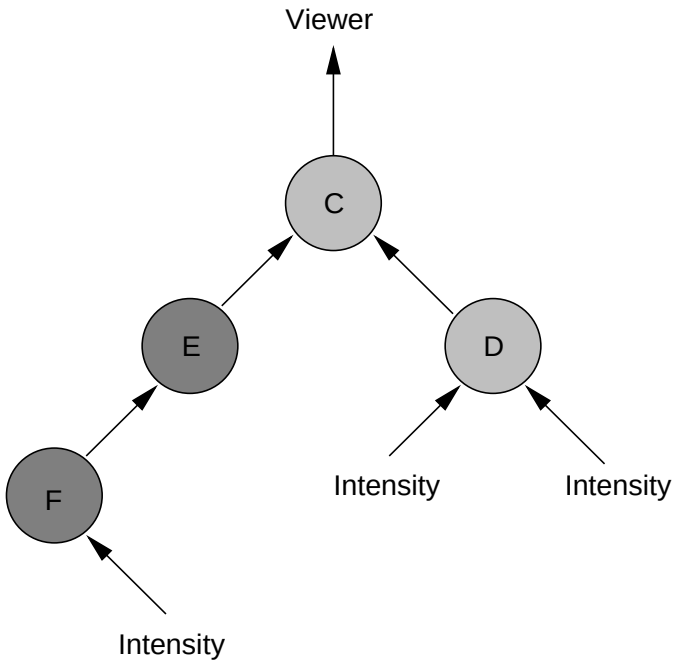

Figure 12: Propagation of light using recursive ray tracing.

This equation has only a solution if

$$(\mathbf{n} \cdot \mathbf{d})^2 > 1 - \left(\frac{k_2}{k_1}\right)^2$$

which indicates the physical phenomenon of the limiting angle after which total reflection occurs. When light passes from one medium into another one whose index of refraction is low, the angle of the transmitted ray is greater than the angle of the incident ray. If the angle of the incident ray is large, the angle of the transmitted ray is larger than $90°$, and the ray is reflected from the interface between the media, rather than being transmitted. This is called *total reflection* and the smallest angle at which this occurs is called *critical angle*.

So far it has been assumed that refraction is perfect. This assumption provides an adequate model for clear glass or water. However, in practice the transmitted ray is distributed in a small cone around the direction computed by Snell's law. This is illustrated in Figure 13(c). This is also know as specular refraction. Clearly, it is impossible to model the ray tracing process using a large number of rays. However, realistic effects such as frosted glass can be produced using two or three rays randomly distributed within this cone.

Raytracing and Radiosity

In summary ray tracing can be used to create realistic effects such as:

- shadows
- reflections
- transparency

In raytracing each light ray is simulated one by one, tracing their path forwards or backwards. As a result of the illumination in ray tracing scenes in view dependent. In particular ray tracing can model specular reflection but not diffuse reflections. To model diffuse relections methods such as *radiosity* may be used.

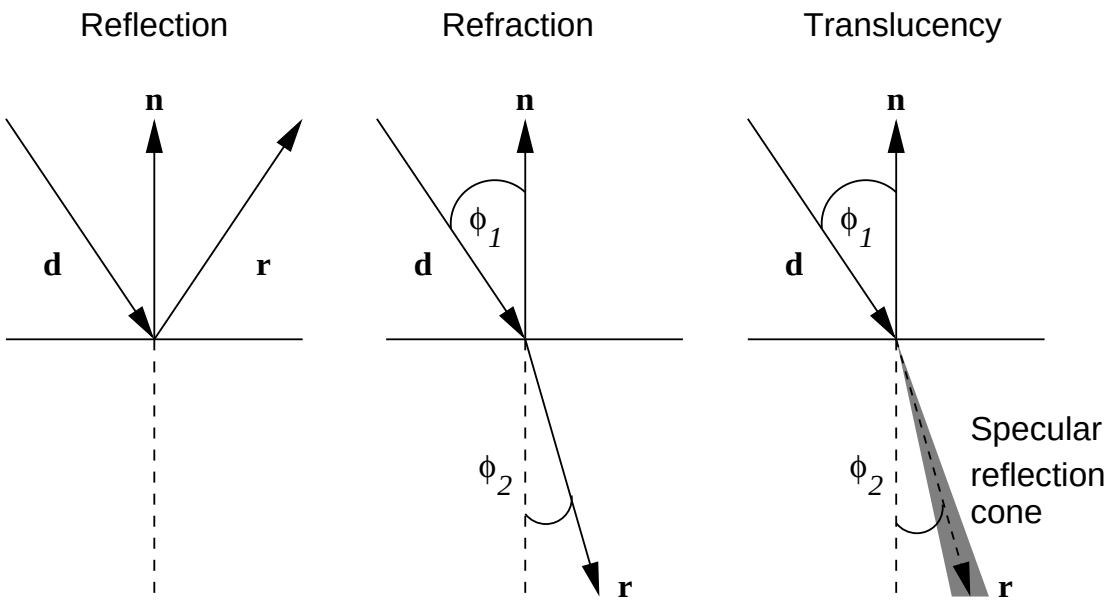

Figure 13: Rays intersecting a surface can be used to achieve effects such as (a) reflection, (b) refraction and (c) translucency.

Radiosity simulates the interreflection of light in 3D scenes and attempts to model ambient illumination more accurately. This enables rendering based on global illumination models which take into account that the light is emitted from surfaces is not only the result of relections from designated light sources. This requires setting up a system of linear equations whose solutions yields the light distribution in the rendered scene. As a result surfaces can no longer be shaded individually since they are interrelated and the solution is view independent. In contrast to ray tracing, radiosity is well suited for the rendering of diffuse scenes (such indoor scences).

```
v = calculate viewpoint
for (x = x_min; x < x_max; x++){
  for (y = y_min; y < y_max; y++){
    r = calculate ray passing from viewpoint v to pixels x, y
    c = calculate color using cast_ray(r, 1)
    draw color c and pixel x, y
  }
}

procedure cast_ray(r: ray, d: depth): color
  determine closest intersection i with closest object c
  if (intersection = true)
    compute normal n at intersection
    return color_ray(r, c, i, n, d)
  else
    return color_background
end

procedure color_ray(r: ray, c: object, i: intersection,
                    n: normal, d: depth): color
  set color to ambient color
  for each light do
    calculate ray from intersection to light
    if dot product of normal and direction to light is positive then
       compute how much light is blocked by opaque and transparent
       surfaces and use to scale diffusion and specular terms
       before adding them to color
  end

  if (d < maxDepth) then
    if (c is reflective) then
      rray = calculate ray in reflection direction
      rcol = cast_ray(rray, depth+1)
      scale rcol by specular coefficient and add to color
    end
    if (c is transparent) then
      tray = calculate ray in refraction direction
      tcol = cast_ray(tray, depth+1)
      scale tcol by transmission coefficient and add to color
    end
  end
  return color
end
```

Figure 14: Basic algorithm for recursive ray tracing.