Al-Mustaqbal University / College of Engineering	
Prosthetics & Orthotics Eng. Department	
Third Class	
Subject (Computer Application)	
Code (UOMU٠١٠٣٠٥٢)	
Asst. Lec. Ghadeer Haider	
١st term – Lecture ٥	

Computer Application

Email (ghadeer.haider @uomus.edu.iq)

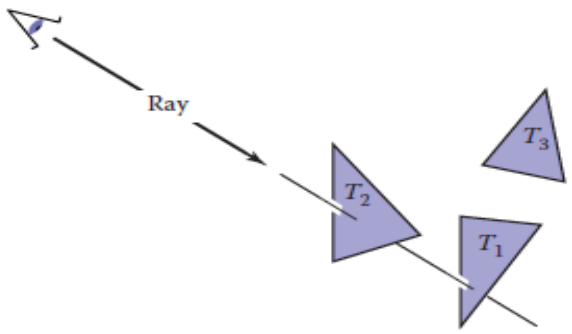
Fundamentally, rendering is a process that takes as its input a set of objects and produces as its output an array of pixels. One way or another, rendering involves considering how each object contributes to each pixel; it can be organized in two general ways. In *object-order rendering*, each object is considered in turn, and for each object all the pixels that it influences are found and updated. In *image-order rendering*, each pixel is considered in turn, and for each pixel all the objects that influence it are found and the pixel value is computed. You can think of the difference in terms of the nesting of loops: in image-order rendering the "for each pixel" loop is on the outside, whereas in object-order rendering the "for each object" loop is on the outside.

4.1 The Basic Ray-Tracing Algorithm

A ray tracer works by computing one pixel at a time, and for each pixel the basic task is to find the object that is seen at that pixel's position in the image. Each pixel "looks" in a different direction, and any object that is seen by a pixel must intersect the *viewing ray*, a line that emanates from the viewpoint in the direction that pixel is looking. The particular object we want is the one that intersects the viewing ray nearest the camera, since it blocks the view of any other objects behind it. Once that object is found, a *shading* computation uses the intersection point, surface normal, and other information (depending on the desired type of rendering) to determine the color of the pixel. This is shown in Figure 4.1, where the ray intersects two triangles, but only the first triangle hit, T_2, is shaded.

A basic ray tracer therefore has three parts:

1. *ray generation*, which computes the origin and direction of each pixel's viewing ray based on the camera geometry;
2. *ray intersection*, which finds the closest object intersecting the viewing ray;
3. *shading*, which computes the pixel color based on the results of ray intersection.



4.2 Perspective

The problem of representing a 3D object or scene with a 2D drawing or painting was studied by artists hundreds of years before computers. Photographs also represent 3D scenes with 2D images. While there are many unconventional ways to make images, from cubist painting to fisheye lenses (Figure 4.2) to peripheral cameras, the standard approach for both art and photography, as well as computer graphics, is *linear perspective*, in which 3D objects are projected onto an *image plane* in such a way that straight lines in the scene become straight lines in the image.

The simplest type of projection is *parallel projection*, in which 3D points are mapped to 2D by moving them along a *projection direction* until they hit the image plane (Figures 4.3–4.4). The view that is produced is determined by the choice of projection direction and image plane. If the image plane is perpendicular

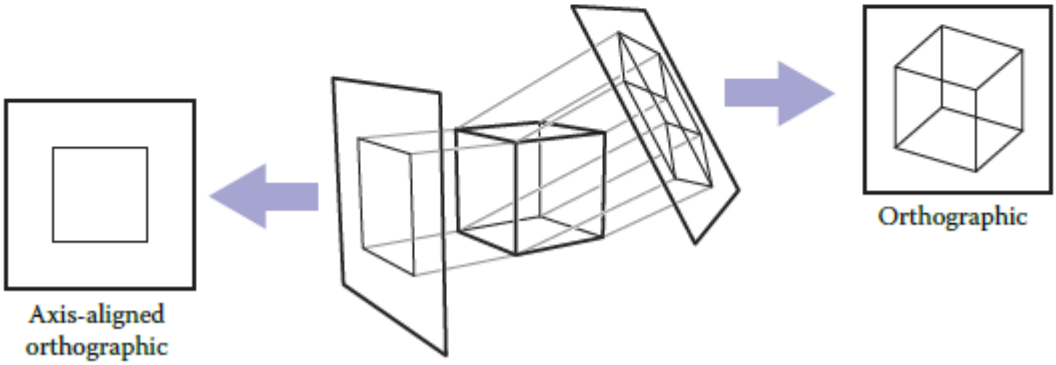


Figure 4.3. When projection lines are parallel and perpendicular to the image plane, the resulting views are called orthographic.

٤,٣ Computing Viewing Rays

From the previous section, the basic tools of ray generation are the viewpoint (or view direction, for parallel views) and the image plane. There are many ways to work out the details of camera geometry; in this section we explain one based on orthonormal bases that supports normal and oblique parallel and orthographic views.

٤,٣,١ Orthographic Views

For an orthographic view, all the rays will have the direction −w. Even though a parallel view doesn't have a viewpoint per se, we can still use the origin of the camera frame to define the plane where the rays start, so that it's possible for objects to be behind the camera.

٤,٣,٢ Perspective Views

For a perspective view, all the rays have the same origin, at the viewpoint; it is the directions that are different for each pixel. The image plane is no longer positioned at e, but rather some distance d in front of e; this distance is the *image plane distance*, often loosely called the *focal length*, because choosing d plays the same role as choosing focal length in a real camera. The direction of each ray is defined by the viewpoint and the position of the pixel on the image plane.

Email (ghadeer.haider @uomus.edu.iq)

٤,٤ Ray-Object Intersection

Once we've generated a ray e+td, we next need to find the first intersection with any object where t > ٠. In practice, it turns out to be useful to solve a slightly more general problem: find the first intersection between the ray and a surface that occurs at a t in the interval [t٠, t١]. The basic ray intersection is the case where t٠ = ٠ and t١ = +∞. We solve this problemfor both spheres and triangles. In the next section, multiple objects are discussed.

٤,٤,١ Ray-Sphere Intersection

٤,٤,٢ Ray-Triangle Intersection

Email (ghadeer.haider @uomus.edu.iq)