2003

Ray Tracing And Global Illumination

Jason Rupard
University of North Florida

Ray Tracing And Global Illumination

Jason Rupard

Faculty Sponsor: Dr.Yap Siong Chua,
Professor of Computer Science

Abstract

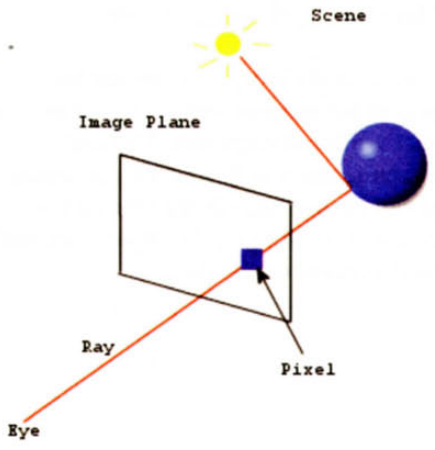
In order to represent real-world images with a computer, a program has to relate three-dimensional images on a two-dimensional monitor screen. Several ways of doing this exist with varying degrees of realism. One of the most successful methods can be grouped in a "screen-to-world method" of viewing, which is also known as "ray-tracing." This computer graphics technology simulates light rays within a 3D environment. Since light rays have predictable physical properties, the ray-tracing algorithm can attempt to calculate the exact coloring of each ray/object intersection at any given pixel. Advanced levels of ray tracing allow light rays to bounce from object to object, mimicking what they do in real life.

"Local illumination" represents the basic form of ray tracing. It only takes into account the relationship between light sources and a single object, but does not consider the effects that result from the presence of multiple objects. For instance, a light source can be intersected by another surface and therefore be obscured to any point behind that surface. Similarly, light can be contributed not by a light source, but by a reflection of light from some other object. The local illumination model does not visually show this reflection of light. Therefore, special techniques have to be used to represent these effects. In real life there are often multiple sources of light and multiple reflecting objects that interact with each other in many ways. "Global illumination," the more advanced form of ray tracing, adds to the local model by reflecting light from surrounding surfaces to the object. A global illumination model is more comprehensive, more physically correct, and it produces more realistic images.

Ray tracing is an essential subject when it comes to computer graphics. It combines issues of efficiency and realism, thus finding a favorable balance of the time and effort involved to make realistic three dimensional images. In the process of researching the many different ways of implementing a ray tracer, the study began with local illumination and graduated to global illumination, using some pre-established techniques and the development of new techniques.

Ray Tracing Basics

A basic model shown in *Figure 1* will shoot one ray per pixel. If an image is 800x600 pixels, then when the ray tracing is complete, 480,000 rays will have been shot. Each will begin at the viewer and end at its closest intersection with an object in the scene. The viewer's location is defined with the other objects of the scene in an input file. An illumination model will be applied to figure out how much light is falling on that point and what color will be produced. An illumination model is an equation used to calculate the intensity of light that we should see at a given point on the surface of an object [2].



Objects within a scene have properties describing its color, if it's reflective (mirror-like) or refractive (glass-like) and its location within the scene. Objects can be spheres, triangles (polygons), rings, cylinders, etc [5]. Any one of these shapes could be a light as well, known as area light sources when the whole surface of the object emits light. For now, we will use point light sources, light coming from a single point in the scene, for our illumination model.

Figure 1. Basic Ray Tracing Scenario

Calculating the Closest Intersection

Parametric equations for a line in a 3 dimensional space are used for calculating the closest intersection with an object from the eye (viewer)[4].

These equations are shown next to *Figure 2*. The goal is to find the smallest t value. The smallest t value will give the closest intersection to the Viewer as shown in *Figure 2*.

$$x = x_0 + t^* (x_1 - x_0)$$
$$y = y_0 + t^* (y_1 - y_0)$$
$$z = z_0 + t^* (z_1 - z_0)$$

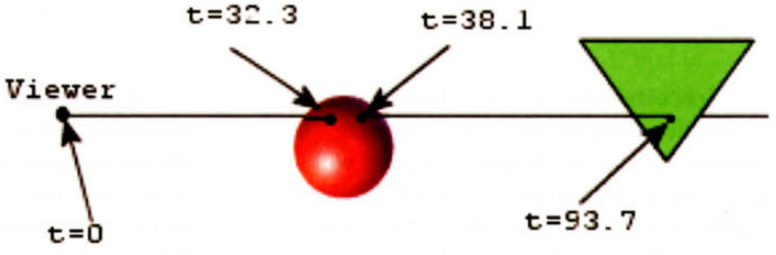


Figure 2. Intersections along a ray, where t=32.3 is the closest one.

$Point_0$ (x_0, y_0, z_0) is the location of the Viewer at the origin for the ray. $Point_1$ (x_1, y_1, z_1) is a point on the image plane. Point (x, y, z) is any point on the line defined by P_0 and P_1. Notice that $(x_1 - x_0)$ is the x component of a vector, same for y and z. So a ray can be represented by vector: $(x_1 - x_0)$, $(y_1 - y_0)$, $(z_1 - z_0)$.

Applying Illumination

Now that the calculation of what the viewer can see at a particular pixel is found, the illumination model is applied. The local illumination model is used figure out what color the pixel will be.

$$Pixel_{color} = Diffuse + Specular$$

A diffused material is a dull material, like chalk. At the point of intersection, a vector is made from the intersection to a light. This forms the light vector L. N is the normal of the surface at that intersection. L and N are shown in *Figure 3.* The normal is perpendicular to the surface. The formula to calculate the diffused component of the local illumination model is as follows [2]:

$$Diffuse = K_{diffuse} * Color_{diffuse} * cos\theta$$

$K_{diffuse}$ and $Color_{diffuse}$ are pre-defined inputs of the program describing a particular object's diffused properties. The angle between L and N is θ, which is calculated and will change according to the light's location. This will give the object a shaded look dependent on the light.

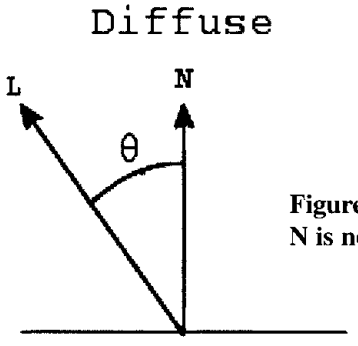


Figure 3. Diffused component, L points at a light source and N is normal to the surface.

$$Specular = K_{specular} * Color_{specular} * cos_{shiny}\ \phi$$

Specular color is viewer dependent. The closer the reflection vector R is pointing towards the eye, the brighter the pixel will get. Simply put, specular color will brighten a point more if the light reflects back into the eye. The formula to calculate the specular component of the local illumination model is as follows [2]:

$K_{specular}$, $Color_{specular}$, and shiny are input describing the object. The *shiny* exponent affects the specular spot on the object, shown in *Figure 5.* The higher *shiny* is, the more concentrated the spot becomes. ϕ is the angle between the Normal vector, N, and the Eye vector, shown if *Figure 4.*

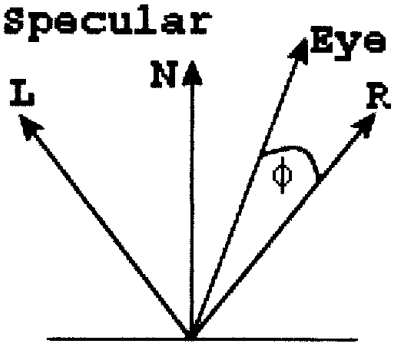


Figure 4. Specular component, R is the reflection of L

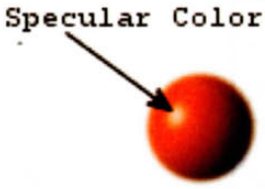


Figure 5. Specular color on a sphere, the shiny exponent will effect the area of the Specular "spot".

Pixels will be in a "Red-Green-Blue" color space known as (R, G, B) values. Each RGB component will have range [0.0 – 1.0]. White would be (1,1,1) and Black (0,0,0). The illumination model formula becomes [2]:

$$Pixel_{color\_R} = Diffused_R + Specular_R$$

$$Pixel_{color\_G} = Diffused_G + Specular_G$$

$$Pixel_{color\_B} = Diffused_B + Specular_B$$

Smoothing the Image

Antialiasing is a technique used for the smoothing of an image. It takes sharp, jagged edges of an image and blends it with colors around the edge making it smooth [1]. For instance, a black surface intersecting a white surface, at those points of intersection the colors will blend and make a grayish color.

To apply this technique to ray tracing is straightforward. Take a pixel and divide it into sub-pixels, shown in *Figure 6*, and shoot the sub-pixels with rays. Add all the sub-colors up and divide that by the number of sub-pixels. This gives you an average color for that whole pixel. This works because the all the sub-rays shot will all not hit the same place, some will hit the black surface and some will hit the white surface. Then averaging the colors will give you a gray.

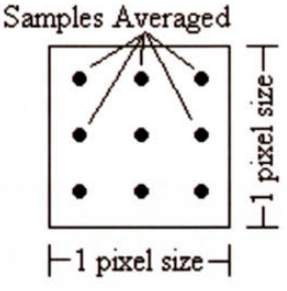


Figure 6. The subdivision of a pixel to make sub-pixels. Calculate the color for each sub-pixel and then averaging them to get a color for the whole pixel.

Accelerated Ray Tracing

Ray tracing is very time consuming algorithm. The majority of the time goes to finding the intersection of a ray [1]. To find this intersection you have to test a ray with every object in the scene, and then chose the closest intersection. Therefore, if there are 86K objects in a scene and the image is 800x600 (480K rays), 41.28 billion intersection calculations are made.

A way of speeding up this process is to use a 3-D grid to encompass all the objects in a scene. Now, instead of testing all 86k objects per ray, only test objects that are in the sub-boxes for which the ray passes through, as shown in *Figure 7*. Only objects in boxes 8,9,10,11,12 need to be test for intersection.

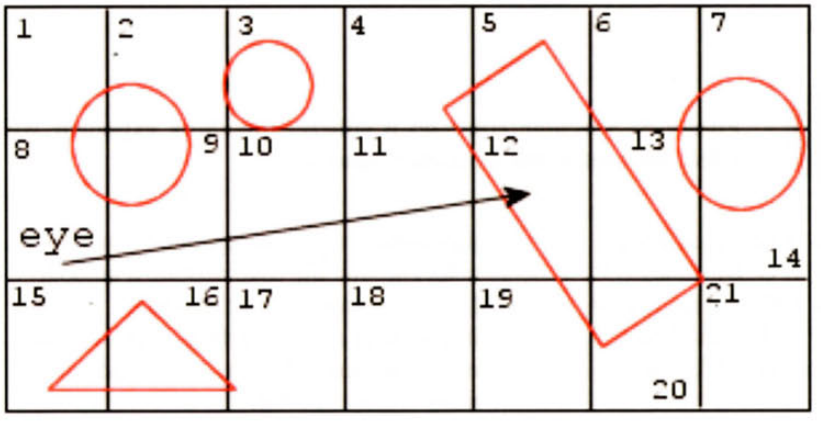


Figure7. 2-D representation of 3-D grid structure.

Another way of accelerating the rendering process is to take advantage of a computer that has multiple processors. With this capability, an image could be split into N sub-image, where N is the number of processors. Each processor having one sub-image to work on, making the run-time N times faster.

Interpolation of Normals

A normal is perpendicular to the surface at a particular point on that surface [4]. If a triangle has just one normal for all the points on the triangle then that triangle will be perfectly flat. With one normal per triangle an object made up of triangles will become patched, as seen with the teapot on the left in *Figure 8.* With interpolation, the goal is to have a slightly different normal for every point on the triangle making the object curved, as seen with the teapot on the right in *Figure 8* [2].

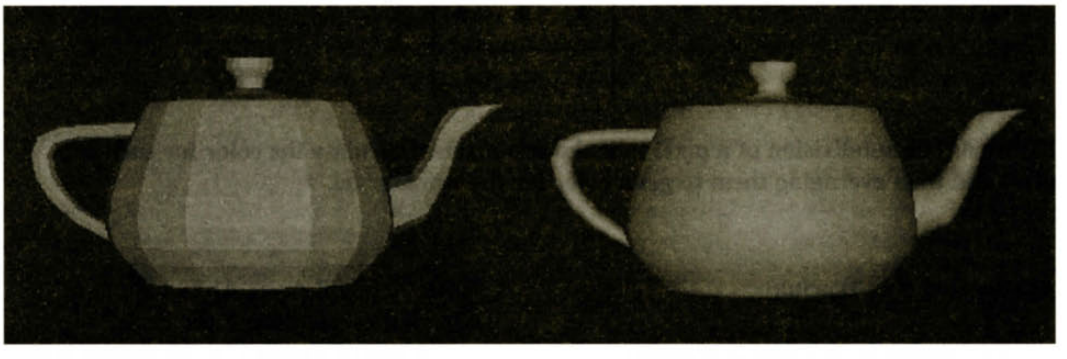


Figure 8. Left teapot is without interpolation; the right teapot was rendered with interpolation.

This new normal, N, will be calculated from three other normals, Na, Nb, and Nc, representing the normals of the three points of a triangle, A, B, and C respectively, shown in *Figure 9*. The three normals of the triangle are pre-defined inputs to the ray tracer. These normals will have been calculated from a different program. They are based upon the averaging of surrounding triangles and their normals. N is a linear combination of the vectors, Na, Nb, and Nc.

$$N = \alpha * N_a + \beta * N_b + \sigma * N_c$$

The location of point P and its distances from each normal determines which one; Na, Nb, or Nc is weighted more than the rest. In *Figure 9,* P is closer to B, therefore β will have a greater value than α and σ.

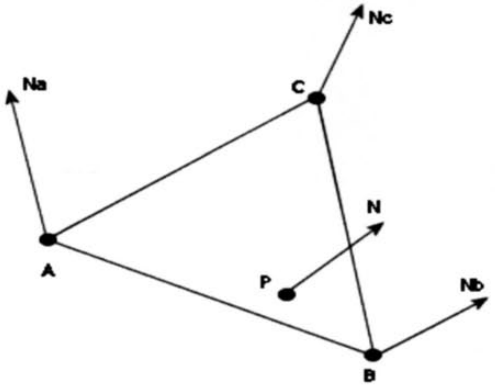


Figure 9. A triangle with three normals used to calculate N, the normal for point P.

Global Illumination

Global Illumination will give a more realistic image. It will take into account all light, direct and indirect, to form a better lighting model on a surface. With local illumination, one ray is shot to every light to calculate how much light will be falling on that surface. With global illumination, once the intersection is calculated many rays are shot out in different directions to produce the light falling on that point. To produce the most realistic image possible, all directions of light would have to be tested. This is impossible because there are infinite directions of light falling on any particular point. Instead, sample rays are shot to produce a lighting model, shown in *Figure 11*. *Figure 10* shows that the more samples that have been taken, the better the image will come out [4].

Figure 10. Left image: 100 sample rays per intersection. Middle image: 1000 sample rays per intersection. Right image: 3500 sample rays per intersection

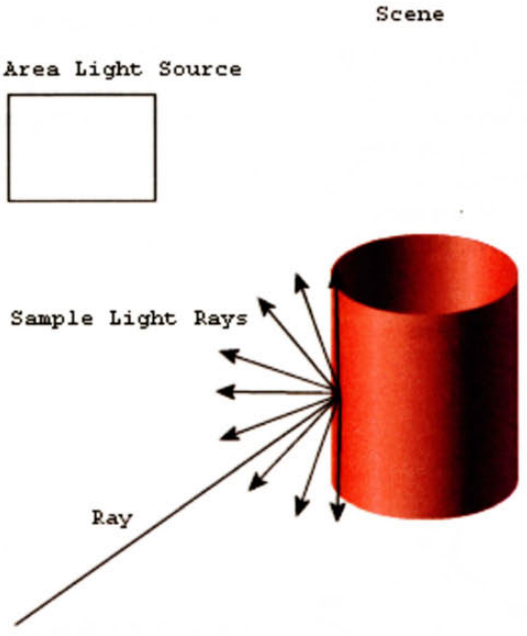


Figure 11. Basic Global Illumination Scenario

A sample ray can bounce randomly off objects until it reaches a light. If it never reaches a light in the maximum allowed bounces, then it is thrown out of the final calculation of pixel color. If a sample hits a light directly, the full intensity of light goes into the calculation. If it doesn't hit the light directly, after every bounce the light intensity is decreased by a factor of the diffused component of the object it is bouncing from. In global illumination the sample ray takes the place of the Light vector, L, in the pixel color calculations, seen in *Figures 3* and *4*. The pixel color formula is now changed to the following.

N will begin equaling the number of samples, but every time gl( ) returns 0 it will be subtracted by 1. The gl( ) function will return 0 if the sample ray never hits a light. This throws out all noncontributing sample rays. Also, gl( ) is a recursive function. It will recurse till the maximum number of bounces has been reached, or it has hit a light. Once gl( ) has hit a light, it returns the light's intensity. That light intensity decreases with every bounce it took to get to the light. So, the more bounces the ray takes to get to the light, the less the light's intensity will factor into the final calculation of the pixel color. This Pseudocode below show this being done.

$$Pixel_{color} = {}^{1}/_{N} * \sum_{i=1}^{\#samples} (Diffused_i + Specular_i) * gl\ (samples)$$

Pseudocode: Global Illumination

```
.color_pixel(Vector ray_from_eye)
          intersection = find_intersection(ray_from_eye)
          N=#samples
          for(i=0;i<#samples; ++i)
                sample_ray = generate_random_ray(intersection)
 current_color = diffuse(intersection)+
     specular(intersection)
                factor = gl(sample_ray)

                if(factor == 0)
                      N - 1
                      goto next sample
                end_if
                sum_of_colors += current_color * factor
          end_for
          pixel_color = sum_of_colors / N
          return pixel_color
    end_color_pixel

    gl(Vector sample_ray)  //recursive function
          if(Max Bounces Reached)
                return 0
          end_if
          intersection = find_intersection(sample_ray)
          if(intersection is a light)
                return light's intensity
          end_if
 sample_ray = generate_random_ray(intersection)
          factor = gl(sample_ray)
          return diffuse(intersection) * factor
 end_gl
```

Generating Random Sample Rays

Generating sample rays is an important part of the global illumination algorithm. Sample rays will produce the light; so bad sample rays will render bad lighting. If a sample ray never hits a light it is thrown out of the calculation. We want all rays to have a "chance" of hitting the light. Sample rays need to point away from the surface they are coming from. A sample ray needs to have an angle with the normal between 0 and 90 degrees, and should be able to reach 360 degrees around the normal. This depicts a hemisphere, with the surface's normal going straight through the top "North Pole" of it, as in *Figure 12*[1].

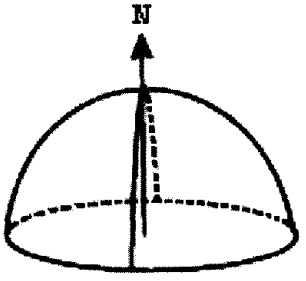


Figure 12. Hemisphere for sample rays

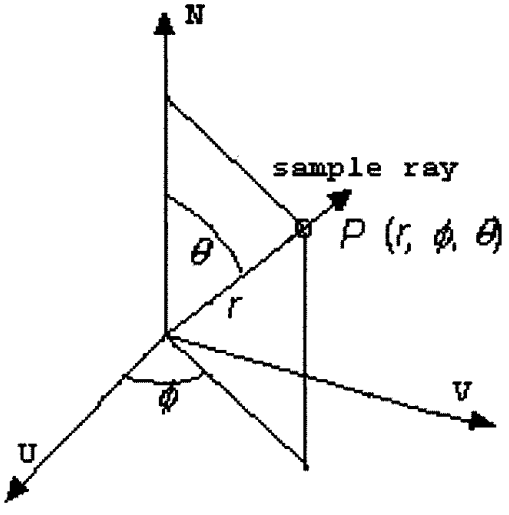


Figure 13. P is a point on the hemisphere.
r is the radius f the hemisphere.
θ has a range of [0, 90] degrees.
ϕ has a range of [0, 360] degrees.

The spherical coordinates system can be use to make random sample rays [1], shown in *Figure 13.* Point P on the hemisphere needs to be randomized. This can be accomplished by randomizing the angles ϕ and θ. Radius r can be kept constant at 1. P needs to converted to it's x, y, and z components with the following formulas.

$x = r \sin \phi \cos \theta$
$y = r \sin \phi \sin \theta$
$z = r \cos \phi$

P is oriented about the *Origin.* So, P needs to be transformed so it is oriented with the surface, and its normal. To do this, an orthonormal basis is made with the surface's normal. An orthonormal basis (ONB) is a set of vectors that are mutually perpendicular, and are unit length [1]. The most famous ONB is the xyz coordinate system, the natural basis. ONBs make converting to and from different coordinate systems uncomplicated. Once an orthonormal basis is made with the surface's normal, P can be transformed into P'. P' is now oriented with the new basis. To finish this process off, a vector is made from the intersection point on the surface to P' and the new sample ray is formed. The new sample ray will be used in place of the light ray in the illumination formula.

Conclusion

The study of ray tracing can lead to interesting things in the field of computer graphics. Ray tracing is a viable technique of producing two-dimensional images of a three dimensional world. It can be a tool that becomes more and more valued as our culture heads deeper into computer generated worlds via games, movies, training simulators, or even architectural modeling. Ray tracing can produce images

with varying degrees of realism. With its strong mathematical and physical foundations, ray tracing is and will remain a major concept of computer graphics.

Gallery

To see these images in their full size please visit:
http://www.unf.edu/~rupj0001/ray/

A.) One of the first images produce. The light is coming from the top right, and shadows were turned off. Image took 13min. to render and its size was 1024x768.

B.) All most same scene as image A, but with a new cylinder and two light sources with shadows turned on. Also this image has Antialiasing 10rays/pixel. Image took 10hours to render and its size was 1024x768.

C.) Two light sources, one inside the cylinder, and one coming from the top left. Shadows turned on. Image took 4min. to render and its size was 1024x768.

D.) Two light sources coming from the bottom left and the top right. 4 reflective sphere all reflecting each other's colors. Image took 1min. to render and its size was 1024x768 with 5rays/pixel.

E.) Same 4 sphere as image D, but now incase in a reflective box. One gold-colored light source in the top right front of the box. Image took 1min. to render and its size was 400x300.

F.) The Parthenon is inside of a blue reflective box. Three light sources, on in the back left bottom corner, one in the front left bottom corner and one inside the Parthenon. Image took 3hours to render and its size was 800x600 with 10rays/pixel.

G.) A glass sphere with a blue diffused sphere behind it. Image took 3mins to render and its size was 1024x768 with 10rays/pixel.

H.) 100,000 random spheres to test the 3-D grid acceleration technique. The image about 18mins at 512x512. Without a 3-D grid it would still be rendering today!

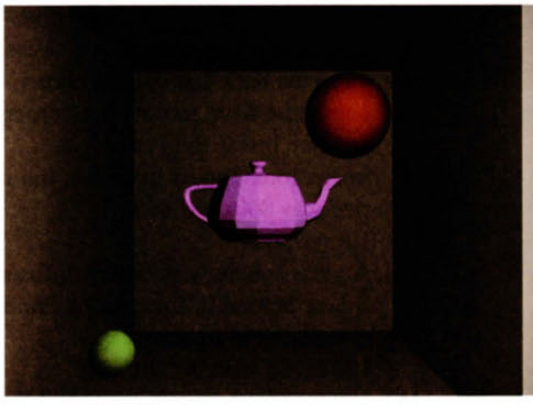
I.) A Sphereflake, this image took about 5min. to render at 1024x768.

J.) The Rhinoceros Logo, 86000 triangles. Image took 30mins on 7 processors at 1024x768.

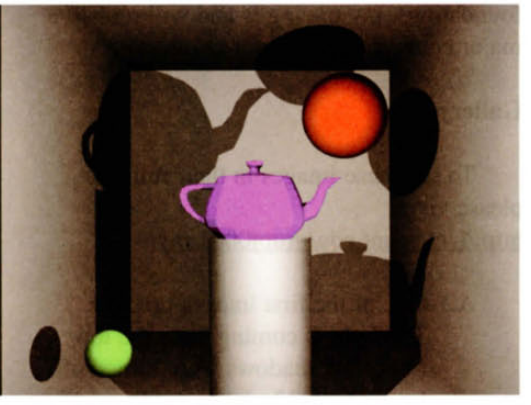
K.) This image was the goal of the research. A global illuminated scene at 800x600 with 3000samples/intersection. Two area light sources at the ceiling of the room. A reflective sphere floats at the left, with two soft shadows under it. The soft shadows are one of the products of the global illumination technique. Image took 7hours on 8 processors.

L.) A comparison of local illumination vs. global illumination.

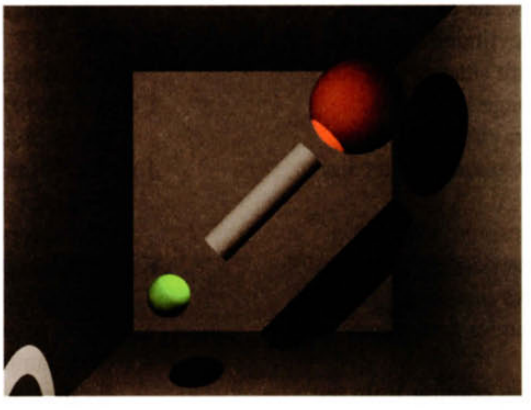
A.



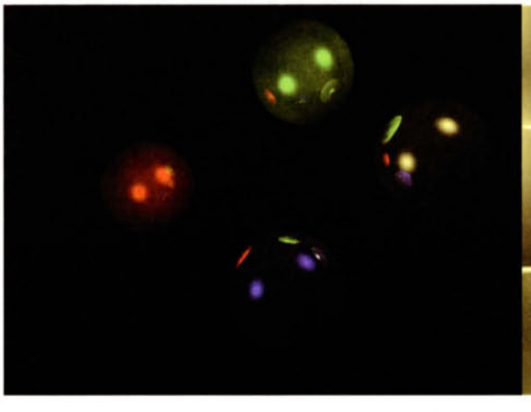
B.



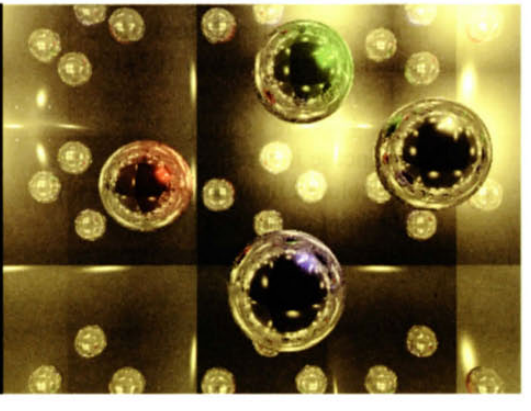
C.



D.

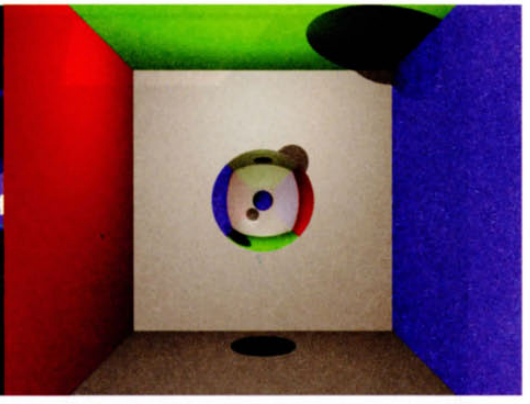


E.

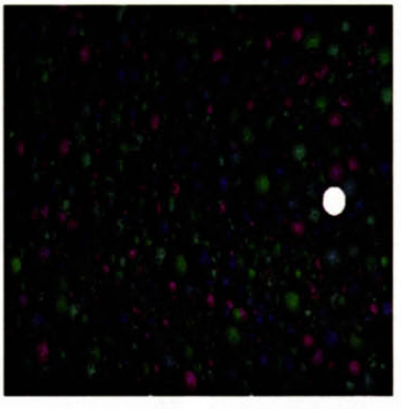


F.

G.



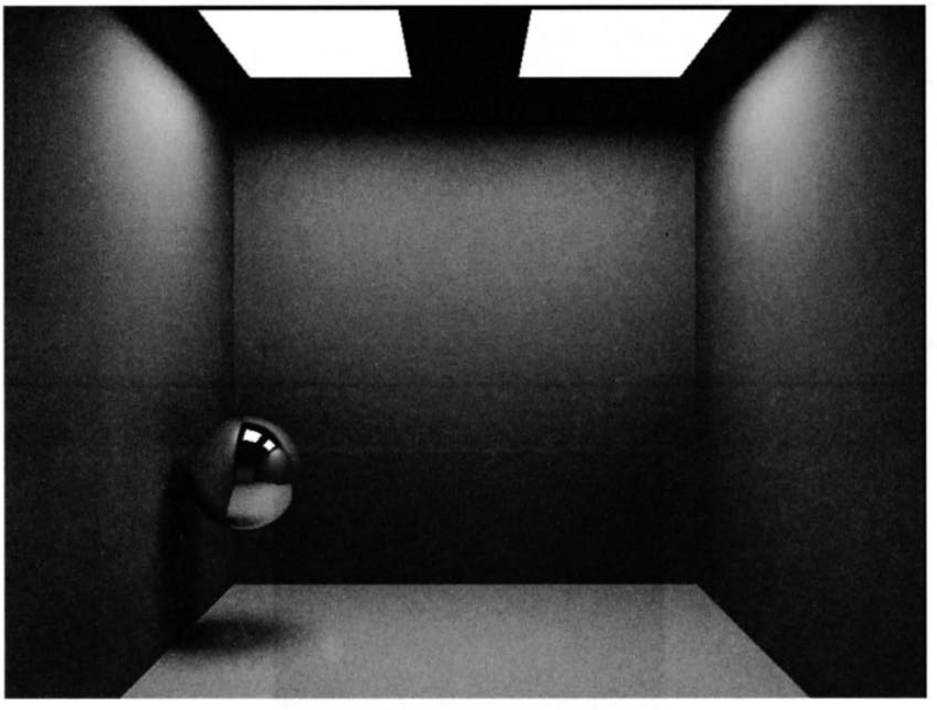
H.



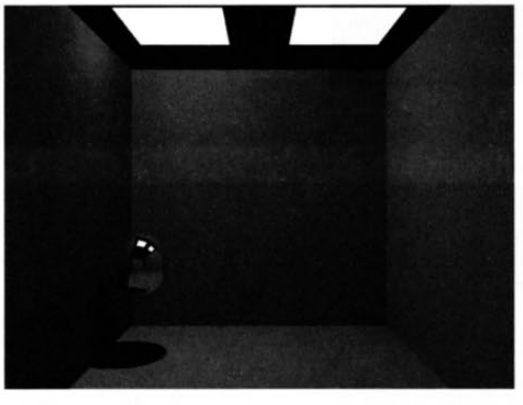
I.

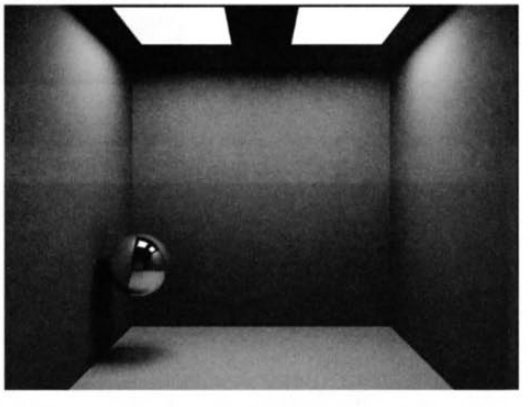
J.

K.



L. Local vs. Global





References

[1] Shirley, Peter. 2000. Realistic Ray Tracing. A K Peters, Natick, Massachusetts.

[2] Baker, M. Pauline. Hearn Donald. 1986. Computer Graphics C Version. Prentice Hall, Upper Saddle River, New Jersey.

[3] Bourke, Paul. Personal Pages. http://astronomy.swin.edu.au/~pbourke/

[4] Larson, Roland. Hostetler, Robert. Edwards, Bruce.1998. Calculus Sixth Edition. Houghton Mifflin Company, Boston, New York.

[5] Ward, Greg. Materials and Geometry Format. http://radsite.lbl.gov/mgf/.