Distribution Ray Tracer

Remy Oukaour, 107122849

1. Introduction

Ray tracing is a technique for generating realistic images of virtual objects by tracing the paths that light takes from various sources to the film plane of a virtual camera. Turner Whitted described how to recursively trace rays to emulate reflected and transmitted light and cast shadows, and Robert Cook extended the process by distributing multiple rays to make these phenomena "fuzzy." This paper describes an implementation of Cook's technique using C++ and OpenGL.

1. Compilation and Execution

The program is compiled and executed via the command line. To compile it, enter:

```
make trace
```

This produces the file trace.exe. To render a scene file progressively, enter:

```
trace scene.dat
```

This will begin rendering the scene in a GLUT window. Pause or resume rendering by pressing `p`, or save the current image as `out.ppm` by pressing `s`. Toggle the diagnostic information by pressing `t`. Quit the program by pressing `q`.

Since ray tracing can be a slow process, and becomes exponentially slower as the resolution, recursion depth, or distributed ray count increase, there is an option to preview a scene file with OpenGL's rasterization:

```
trace –p scene.dat
```

This is near-instantaneous, but cannot handle ray-traced phenomena like reflection, transmission, or shadows.

To render a scene file directly as a 24-bit color pixmap image, enter:

```
trace scene.dat image.ppm
```

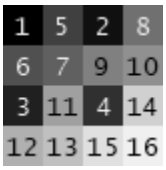
This foregoes the interactive rendering display, but is slightly faster. The PPM format was chosen for ease of output, since the buffer filled by OpenGL's `glReadPixels` function can be written directly to the image file after an appropriate header.

2. Progressive Rendering

In order to quickly judge how a scene will be rendered without waiting for the entire process to complete, images are rendered at progressively higher resolutions. The simple way to render images is scanline by scanline, top to bottom. The pixels of a 4×4 image would be drawn in this order:

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

This order is used when rendering directly to an image. When displaying its output in a window, `trace` instead renders pixels in this order:



Pixel #1 at first covers the entire image; then pixels 2, 3, and 4 each cover a quarter; and so on until all the pixels have been rendered. If a scene does not seem to look as desired, the user can stop the program and edit the scene file.

3. Scene Definitions

Scene files are written in ASCII text and can be saved with the `.dat` extension. They are intended to be easy to read, both for humans and parsers. They are parsed by `trace` as a series of tokens up to 64 characters long, separated by any number of whitespace characters. These tokens are then used to define the scene's camera, objects, and lights.

3.1. Token Formats

As tokens are read, they are compared with known key values. Most keys are to be followed by one or more parameters of a certain format. If the number or format of the parameters is not as expected, an error message prints and the program ends. The possible parameter formats are:

- integer — anything matched by C++'s `%d` string format specifier: `42`, `+1`, `-9`.
- decimal — anything matched by `%lf`: `3.0`, `+1.3e3` (which equates to 1300), `-.01`.
- string — anything matched by `%s`, i.e. any non-whitespace characters: `foo`, `c8d4`, `*x*`, `a^b`. If the parser is not expecting key parameters, tokens are parsed as strings.
- color — three consecutive decimal tokens encoding red, green, and blue color components. Each component ranges from 0 to 1, but in some cases out-of-range values may be sensible, as when the color is used as a multiplicative coefficient rather than an absolute hue.
- vector — three consecutive decimal tokens encoding *x*, *y*, and *z* components of a vector. Scenes use a right-handed coordinate system, with the *x*-axis pointing left and the *y*-axis pointing up.

3.2. Scene Keys

When a parsed token matches a known key, its parameters are parsed and used to configure the renderer or set up the scene. Most keys expect a fixed number of parameters; others delimit sections, so tokens are consumed until reaching the end delimiter and then parsed based on context. If a key is repeated, the new parameters override the ones previously specified.

- `image` $W_{\texttt{integer}}$ $H_{\texttt{integer}}$ — the width W and height H of the image in pixels. W and H must both be positive. The default is 240×240.
- `depth` $D_{\texttt{integer}}$ — the maximum number of times to reflect or transmit a ray when tracing it through the scene. The default is 0.

- `distribute` $S_{\texttt{integer}}$ — the number of rays to distribute and average for fuzzy phenomena. S must be positive. The default is 1.
- `refresh` $R_{\texttt{integer}}$ — the number of milliseconds to wait before redrawing the OpenGL display. Drawing pixels takes time, so setting this higher can speed up rendering. R must be positive. The default is 1000.
- `background` $C_{\texttt{color}}$ — the color to use for pixels which rays do not intersect any objects. The default is black (0, 0, 0).
- `ambient` $C_{\texttt{color}}$ — the color of the ambient light. The default is black (0, 0, 0).
- `camera` … `end` — delimits keys relevant to the properties of the scene's camera.
- `material` $N_{\texttt{string}}$ … `end` — delimits keys that define a material named N.
- `light` … `end` — delimits keys that define light source.
- `sphere` … `end` — delimits keys that define a sphere.
- `plane` … `end` — delimits keys that define a plane.
- `triangle` … `end` — delimits keys that define a triangle.
- `[` … `]` — delimits a comment. Comments can be nested.

3.2.1. Camera Keys

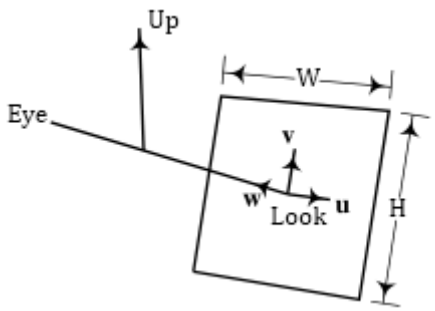
The virtual camera is conceptually a pinhole camera projecting onto a piece of pixilated film. It is specified by its location ("eye"), facing ("look-at" point), and "up" direction. A physical pinhole camera has its film behind the pinhole; this camera has its image plane in front, corresponding with the look-at point.

- `eye` $E_{\texttt{vector}}$ — the eye (pinhole) location. The default is (0, 0, 0).
- `look` $L_{\texttt{vector}}$ — the look-at location. The default is (0, 0, −1).
- `up` $U_{\texttt{vector}}$ — the upwards direction. The default is (0, 1, 0).
- `screen` $W_{\texttt{decimal}}$ $H_{\texttt{decimal}}$ — the width and height of the film plane. The default is 2×2.
- `fov` $F_{\texttt{decimal}}$ — the horizontal field of view angle in degrees. The default is whatever is implied by having the film plane contain the look-at point. Specifying a field of view modifies the look-at vector:

$$look' = eye + \frac{look - eye}{|look - eye|} \cdot \frac{width}{2 \tan fov}$$

- aperture A_{decimal} — the radius of the pinhole. A must not be negative. The default is 0.
- `range` $N_{\texttt{decimal}}$ $F_{\texttt{decimal}}$ — the distances of the near and far planes, for use with the rasterizing preview mode. N and F must be positive. The defaults are 0.1 and 10,000.

The virtual camera stores three unit vectors u, v, and w derived from the specified eye, look-at, and up vectors:



$$w = \frac{eye - look}{|eye - look|}$$
$$u = \frac{up \times w}{|up \times w|}$$
$$v = \frac{u \times w}{|u \times w|}$$

These vectors are used later when firing rays from the eye point through the image plane.

3.2.2. Material Keys

Objects in a scene have material properties that determine their shading, and since multiple objects might share the same material, a material can be defined once and reused.

- `use` $N_{\texttt{string}}$ — the properties to those of the material named N.
- `ambient` $A_{\texttt{color}}$ — the ambient coefficient. The default is (0, 0, 0).
- `diffuse` $D_{\texttt{color}}$ — the diffuse coefficient. The default is (0, 0, 0).
- `specular` $S_{\texttt{color}}$ — the specular coefficient. The default is (0, 0, 0).
- `shininess` $H_{\texttt{decimal}}$ — the shininess of the specular highlight. H must be positive. The default is 1.
- `reflect` $R_{\texttt{color}}$ — the reflection coefficient. The default is (0, 0, 0).
- `transmit` $T_{\texttt{color}}$ — the transmission coefficient. The default is (0, 0, 0).
- `ior` $I_{\texttt{decimal}}$ — the index of refraction. I must not be zero. The default is 1.
- `gloss` $G_{\texttt{decimal}}$ — the glossy reflection coefficient. G must not be negative. The default is 0.
- `translucency` $T_{\texttt{decimal}}$ — the translucent transmission coefficient. T must not be negative. The default is 0.
- `shadowless` — objects of this material will not cast shadows.

3.2.3. Light Keys

Lights in a scene exist as single points that cast sharp shadows; with a non-zero radius they are conceptually spheres that cast soft shadows, but will not appear in the rendered scene although their emitted light affects objects that do appear. To fake the appearance of a spherical light, surround it with a shadowless sphere of the same color that reflects only ambient light.

- `pos` $P_{\texttt{vector}}$ — the center position. The default is (0, 0, 0).
- `color` $C_{\texttt{color}}$ — the emitted color. The default is white (1, 1, 1).

- `diffuse` $D_{\texttt{decimal}}$ — the diffuse intensity. The default is 1.
- `specular` $S_{\texttt{decimal}}$ — the specular intensity. The default is 1.
- `attenuation` $A_{\texttt{decimal}}$ $B_{\texttt{decimal}}$ $C_{\texttt{decimal}}$ — the attenuation coefficients. The defaults are 1, 0, and 0.
- `radius` $R_{\texttt{decimal}}$ — the radius. R must not be negative. The default is 0.

3.2.4. Sphere Keys

Spheres are surfaces defined as the set of points a certain distance away from a center point.

- Any material key.
- `pos` $P_{\texttt{vector}}$ — the center position. The default is (0, 0, 0).
- `radius` $R_{\texttt{decimal}}$ — the radius. R must be positive. The default is 1.

3.2.5. Plane Keys

Planes are infinite surfaces defined by a normal vector and a distance away from the origin.

- Any material key.
- `normal` $N_{\texttt{vector}}$ — the normal direction. The default is (0, 1, 0).
- `distance` $D_{\texttt{decimal}}$ — the distance from the origin. The default is 0.
- `scale` $S_{\texttt{decimal}}$ — the width, for use with the rasterizing preview mode (since ray-traced planes are infinite, but OpenGL cannot rasterize infinite planes). S must be positive. The default is 10,000.

3.2.6. Triangle Keys

Triangles are surfaces defined as being contained within three vertices. The vertices should wind counter-clockwise.

- Any material key.
- `v1` $V_{\texttt{vector}}$ — the first vertex. The default is (1, 0, 0).
- `v2` $V_{\texttt{vector}}$ — the second vertex. The default is (0, 0, 1).
- `v3` $V_{\texttt{vector}}$ — the third vertex. The default is (0, 1, 0).

4. Rendering and Depth of Field

To render the image pixel at (x, y), `trace` converts it to a point on the image plane:

$$d_u = \frac{2x+1}{2w} - \frac{1}{2}$$
$$d_v = \frac{2y+1}{2h} - \frac{1}{2}$$

Then the camera fires a ray from its eye point to the image plane point:

$$pt = look + u \cdot d_u w + v \cdot d_v h$$
$$dir = \frac{pt - eye}{|pt - eye|}$$

If the camera's aperture is non-zero, the eye point is randomly perturbed within a circle of the aperture's size before being used to calculate the ray direction. This is done for a number of rays, which are each traced through the scene and then averaged to get the final color for the pixel.

A ray is traced by testing its intersection with every object to find the closest one. If it does not intersect any objects, the pixel is colored with the scene's background color. Otherwise, the shading at the point of intersection is calculated, including contributions from any recursively traced reflection, transmission, or shadow rays.

4.1. Ray-Object Intersection

The equation for a ray with origin O and direction V is:

$$R(t) = O + Vt$$

By including this equation in the system of equations defining a surface and solving for its real solutions, one can find the point at which a ray intersects the surface.

4.1.1. Ray-Sphere Intersection

The equation for all points P on a sphere with center C and radius r is:

$$(P - C)^2 - r^2 = 0$$

The system of equations can be solved as follows:

$$\begin{aligned} Q &= O - P \\ b &= -Q \cdot V \\ d &= b^2 - Q \cdot Q + r^2 \\ t &= b \pm \sqrt{d} \end{aligned}$$

If $\sqrt{d}$ is imaginary or $b + \sqrt{d}$ is not positive, then there is no intersection. Otherwise, the ray intersects the sphere at a distance of t for whichever value of t is positive. If both values are negative, then the ray originated inside the sphere.

4.1.2. Ray-Plane Intersection

The equation for a plane with normal (a, b, c) and distance from the origin d is:

$$ax + by + cz + d = 0$$

The system of equations can be solved as follows:

$$s = \frac{-(N \cdot O + d)}{N \cdot V}$$

If s is zero, negative, or undefined due to a zero denominator, then there is no intersection. Otherwise, the ray intersects the plane at a distance of s.

4.1.3. Ray-Triangle Intersection

The equation for a triangle with vertices V_0, V_1, V_2 is:

$$T(u, v) = (1 - u - v)V_0 + uV_1 + vV_2$$

where u and v are barycentric coordinates. Equating $R(t)$ and $T(u, v)$ yields a linear system of equations which can be solved by translating the triangle to the origin and aligning the ray with the x-axis before finding their intersection. A more detailed algorithm is given in Möller and Trumbore's 1997 paper "Fast, Minimum Storage Ray/Triangle Intersection."

4.2. Phong Illumination and Shadows

Once a point on the surface of an object has been intersected by a ray, it is shaded according to the Phong illumination model:

$$I = K_a L_a + \sum_{l \in L} (K_d I_d C_l (N \cdot L_l) + K_s I_s C_l ((L_l - 2(N \cdot L_l) \cdot N) \cdot V)^n) a_l S_l$$

Here K_a, K_d, and K_s are the object's material's ambient, diffuse, and specular coefficients; L_a is the scene's ambient light color; C_l is the color of light l and I_d and I_s are the light's diffuse and specular intensities; N is the normal vector at the shaded point; L_l is the unit vector in the direction of light l; V is the intersecting ray's direction; n is the object's material's shininess coefficient; a_l is the attenuation coefficient of light l, calculated according to the light's distance d from the shaded point:

$$a_l = \frac{1}{a + bd + cd^2}$$

S_l is the shadow coefficient of light l. For point sources of light it is calculated as the product of the transmission coefficients of all objects intersected by a shadow ray cast from the shaded point to the light's center. If any of the coefficients are zero, the point is in complete shadow and is only illuminated by ambient light; non-zero coefficients from transmissive objects result in colored shadows.

For light sources with a radius, a number of shadow rays are distributed to random points in the light's spherical volume, and the shadow coefficients for each of them are averaged to give the final shadow coefficient.

4.3. Reflection and Gloss

Objects which material has a non-zero reflection coefficient spawn reflection rays, which are traced through the scene the same way as the initial camera rays and added to the object's color at the point of reflection. Their direction is calculated as:

$$R = V + 2(N \cdot L) \cdot N$$

Here V is the intersecting ray's direction; N is the normal vector at the shaded point; and L is the unit vector in the direction of the light. This results in a reflected ray that makes the same angle with the normal as the incident ray, but on the opposite side.

For objects with glossy materials, multiple rays are reflected, each one being randomly distributed in a solid angle about the vector R. The colors yielded by tracing the reflected rays are averaged to give the final glossy reflected color. Glossy reflections are only performed for the initial camera rays, not for subsequent reflected or transmitted rays, since the extra computation time is not worth the slight visual improvement.

4.4. Transmission and Translucency

Objects which material has a non-zero transmission coefficient spawn transmission rays, which are traced through the scene the same way as the initial camera rays and added to the object's color at the point of transmission. Their direction is calculated as:

$$T = nV + (n(N \cdot V) - \sqrt{1 - n^2(1 - (N \cdot V)^2)})N$$

Here V is the intersecting ray's direction; N is the normal vector at the shaded point; and n is the ratio of the present index of refraction to that of the transmitting object's material. (The initial index of refraction for camera rays is 1.) This results in a transmitted ray that bends about the normal according to the change in the index of refraction. If the square root in the equation for T is imaginary, no ray is transmitted.

For objects with translucent materials, multiple rays are transmitted, each one being randomly distributed in a solid angle about the vector T. Like the glossy reflection rays, their colors are averaged to give a single value, and translucency effects only apply to the initial camera rays for the sake of speed.

5. Conclusion

Robert Cook's 1984 paper describes how rays can be distributed in space or time to render fuzzy effects; he lists soft shadows, glossy reflections, translucent transmissions, depth of field effects, and motion blur. The `trace` program described here implements all of those except for motion blur, although the lack of spatial subdivision or low-level optimizations means that distributed rays take a long time to render.

References

1. Cook, Robert L., Thomas Porter, and Loren Carpenter, "Distributed Ray Tracing," *Computer Graphics*, vol. 18, no. 3, pp. 137–145, July 1984.
2. Whitted, Turner, "An Improved Illumination Model for Shaded Display," *Communications of the ACM*, vol. 23, no. 6, pp. 343–349, June 1980.
3. Sederberg, Thomas W., "Vector Algebra for Ray Tracing," February 2005.
4. B. T. Phong, "Illumination for Computer Generated Pictures," *Communications of the ACM*, vol. 18, no. 6, pp. 311–317, June 1975.
5. Möller, Tomas and Ben Trumbore, "Fast, Minimum Storage Ray/Triangle Intersection," *Journal of Graphics, GPU, and Game Tools*, vol. 2, no. 1, pp. 21–28, 1997.
6. Bikker, Jacco, "Raytracing Topics & Techniques," *flipcode*, September 2004.
7. Webster, Aaron, "CG Lighting 101," *cgCoach*, January 2011.