Birthing a Ray Tracer

Mark Tigges mtigges@cpsc.ucalgary.ca

November 24, 1997

1 Introduction

Ray tracing as an image synthesis technique has enjoyed immense success. Learning about the algorithms is much easier through implementation for any technique, in the case of ray tracing one aspect learned is that it is simple to implement. Moreover, implementing a ray tracer goes a long way as a learning aid for much of computer graphics. I will discuss the very basics of implementing a ray tracer, my own personal biases will be expressed but no unaccepted methods will be taught. The one message I wish to convey is to keep it simple.

All figures show the $x = 0$ plane, scalars are lower case italic (ex. x), vectors are bold (ex. $\mathbf{p}$) and normalized vectors have a hat (ex. $\hat{\mathbf{d}}$).

2 Intersections

I will discuss a general technique for ray object intersection with application to spheres. A sphere is far and away the most common object to intersect. A plane is slightly easier to intersect but far less interesting. Consider the implicit form of a sphere at the origin with radius 1 (what I term a canonical sphere):

$$f(\mathbf{P}) = \mathbf{P}_x^2 + \mathbf{P}_y^2 + \mathbf{P}_z^2 - 1 = 0$$

Given a ray $R : \mathbf{o} + t \cdot \mathbf{d}$ we want to know for value t, $R(t) = f(\mathbf{P})$. This is easy (always remember, ray tracing is simple), let $P = \mathbf{o} + t \cdot \hat{\mathbf{d}}$.

$$\begin{aligned} f(R(t)) \quad = \quad & (\mathbf{o}_x + t \cdot \hat{\mathbf{d}}_x)^2 + \\ & (\mathbf{o}_y + t \cdot \hat{\mathbf{d}}_y)^2 + \\ & (\mathbf{o}_z + t \cdot \hat{\mathbf{d}}_z)^2 - 1 \end{aligned}$$

This is a quadratic equation in t whose coefficients are found after a little bit of algebra:

$$A = \mathbf{o}_x^2 + \mathbf{o}_y^2 + \mathbf{o}_z^2 - 1$$

$$B = 2 \cdot (\hat{\mathbf{d}}_x \cdot \mathbf{o}_x + \hat{\mathbf{d}}_y \cdot \mathbf{o}_y + \hat{\mathbf{d}}_z \cdot \mathbf{o}_z)$$

$$C = 1$$

The solution of the quadratic equation $At^2+Bt+C = 0$ will yield either two solutions or no solutions. (Note that a single solution is really a double root, hence two solutions.) You are interested in the smaller root, it indicates the first intersection of the ray and the sphere. In the event of a double root, the ray (to within the precision of the machine) is tangential to the sphere. Figure 1 illustrates the meaning of the solution of the quadratic equation. Substituting the value for t into the ray equation $\mathbf{o} + t \cdot \hat{\mathbf{d}}$ yields the point in space for the intersection (the question is which space?).

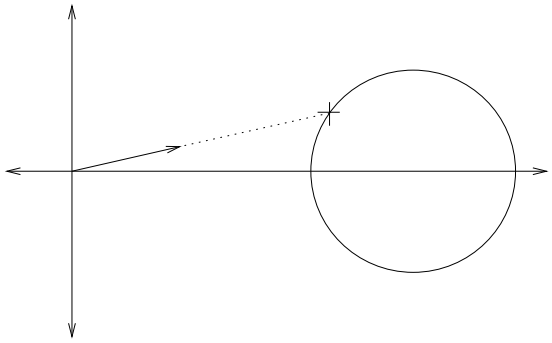

Figure 1: A ray leaving the origin intersecting a sphere, the distance from the ray's origin to the intersection point is the solution of $At^2 + Bt + C = 0$ for the surface.

After finding an intersection, it is necessary to calculate the normal for the intersected surface. The normal in the case of an algebraic surface is the partial derivatives w.r.t. x, y and z for each of the respective components of the normal. In the case of other types of surfaces the calculation of the normal should be inherent in the definition of the surface.

For completeness I should give the partial derivatives for the sphere. However in this discussion I have left something out. I prefer all intersections to be calculated in object space, if this is the case, for a sphere the normal is simply the intersection point (scaled to unit length, if it isn't already). Transformation between object and world space is discussed in section 3.

The method as presented for sphere intersection is completely general. However for surfaces of greater than fourth degree you must use a root solver to locate intersection points.

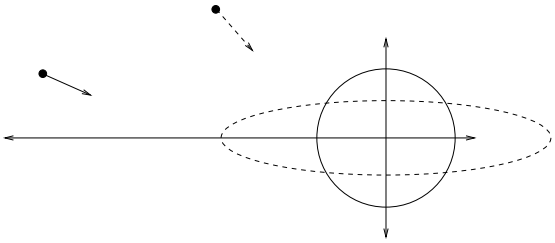

Figure 3: Transformation of ray into object space, the original ray and sphere are in solid, the dashed ray is the transformation into object space to effect the dashed sphere.

3 Rays & Object Space

Since I advocate simplicity in a ray tracer I argue for a system which does not know how to intersect a general object. That is to say I would prefer being able to only intersect a canonical sphere instead of arbitrarily positioned spheres or ellipses in arbitrary orientation. Consider a sphere of radius R translated to position $\mathbf{P}$. Instead of accounting for generality

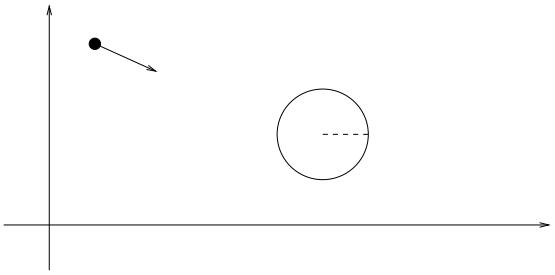

Figure 2: A sphere of radius R and position P and a ray for which to calculate intersection.

in ray object intersection calculation, transform the ray by the inverse of the affine transformation for the object. That is to say, *transform the ray into object space*. So for example, if your ray tracer can only intersect canonical spheres then to render a sphere positioned at $< 0, 0, 3 >$, translate the ray origin by $< 0, 0, -3 >$. If an ellipse is desired then scale both the ray origin *and* the ray direction by the inverse of the scale factor. In figure 3 the ray has been transformed by the inverse scale of $< x, 0.5, 2 >$. That is the ray has been scaled by $< 1/x, 2, 0.5 >$. An intersection calculation from a transformed ray yields t_{os} (object space ray time), we need t_{ws} (world space ray time). We transform the object space intersection to world space through application of the objects transformation matrix, and calculate the distance between it and the world space ray origin.

In section 2 I discussed the need to compute the normal after locating the intersection. The whole story is that not only do we need to compute it, we have to transform it into world space as well. This transformation is achieved by applying the inverse of the object transformation matrix. However, a caveat, since the normal represents a direction and not a location (and hence this caveat applies to transforming ray directions as well) we *do not* want to translate it. This means only multiply the direction vectors by the upper 3x3 portion of your transformation matrices.

This method of transformation of the ray into object space is completely general. It does not depend in any way on the object. The steps are as follows, transform the ray into object space through application of the inverse of the transformation matrix for the object. Perform the intersection calculation with the transformed ray. If there was an intersection, scale the intersection back into world space and compute the time to the world space ray. Whenever we talk about an inverse it is necessary to wonder about division by zero - or in our case a singular matrix. What does it mean for our transformations if we have a singular matrix? . . . It means we tried to scale by zero, this is something you probably don't want to do anyway.

Why would we want to use this technique? The general implicit form for the equation of a plane is $f(x, y, z) = ax^2 + by^2 + cz^2 + 2dxy + 2eyz + 2fxz + 2gx + 2hy + 2jz + k = 0$, which is the general form of a quadric. For a canonical plane ($y = 0$, and normal $< 0, 1, 0 >$) the ray intersection calculation amounts to $t_{os} = -\mathbf{o_y}/\hat{\mathbf{d}}_\mathbf{y}$. This argument is an ex-

Object	Normal
Plane	$< 0, 1, 0 >$
Cylinder	$< x, 0, z >$
Sphere	$< x, y, z >$
Cone	$< x, -y, z >$

Table 1: Normals for quadric surface object space intersections at $< x, y, z >$.

ageration for effect, but it illustrates the point. Moreover, the normal is usually much easier to compute in object space. As an example, table 1 shows the *non-computations* for the normal when given object space intersections for four common quadric surfaces.

4 Looking At It

Concering setting up a camera view, it may be our first idea to use an arbitrary position and orientation. If so go back and read the introduction, keep it simple. Leave the camera at the origin and have it point down an axis. In order to view objects at a desired distance and orientation transform them there.

Consider the definition of a simple camera view. We need nothing more (in the case of ray tracing) than specification of the field of view angle (θ) and the resolution (dx,dy) of the image to generate. To set up a view volume for screen space with square pixels we employ a tiny bit of trigonometry. The field

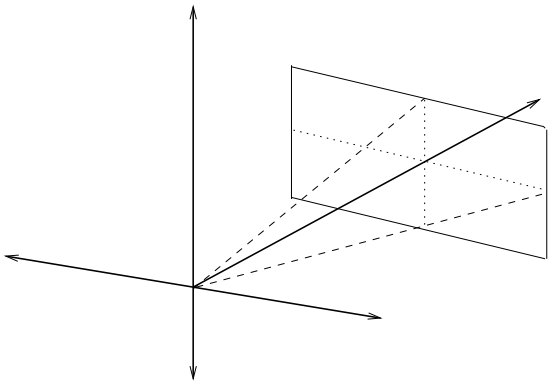

Figure 4: A view volume looking down the z-axis, θ_x and θ_y are the angles between the z-axis and the dashed lines.

of view specified by the user is θ_x, the angular extent of horizontal view, the vertical view angle is calculated $\theta_y = \theta_x \cdot dy/dx$. Any world space point on the view plane correspondent to screen space coordinate (s_x,s_y) through which rays should be traced is easily computed.

$$\mathbf{d} = \left\langle \frac{2s_x - dx}{dx} \cdot \tan(\theta_x), \frac{2s_y - dy}{dy} \cdot \tan(\theta_y), z \right\rangle \quad (1)$$

This assumes that the viewing direction is along the z axis, z is either 1 or -1, depending on the handedness of world space. Of course, the ray direction $\hat{\mathbf{d}}$ must be computed through normalizing the above vector.

5 Making it Look Good

So far we have discussed intersection calculation, simplification thereof and generating initial rays, now we discuss computing a colour for a pixel through which a ray has been traced. The colour to compute will be dependent on the surface properties of the first object that the ray struck, the distribution of luminaires in the scene and relations between objects. The simplest model to employ for illumination is due to Whitted (it is assumed that you are familiar with the basic Phong illumination model):

$$I_\lambda = Amb_\lambda + \sum_{i=1}^{N_l} l_i\lambda \left[Diff_\lambda + Spec_\lambda\right] + k_s I_{r\lambda} + k_t I_{t\lambda} \quad (2)$$

$$Amb_\lambda = I_{a\lambda} k_a O_{d\lambda}$$

$$Diff_\lambda = k_d O_{d\lambda} (\hat{\mathbf{N}} \cdot \hat{\mathbf{L}})$$

$$Spec_\lambda = k_s (\hat{\mathbf{R}} \cdot \hat{\mathbf{L}})^n$$

$$l_i\lambda = S_i f_{att_i} I_{p\lambda_i}$$

There is nothing that much new here over the Phong illumination model. The new terms are S_i, $I_{r\lambda}$, k_t and $I_{t\lambda}$. The k_t term is the *transmission coefficient*, a factor indicating the efficiency of the surface in transmission of light (it ranges from 0..1). The interesting parts are S_i, $I_{r\lambda}$ and $I_{t\lambda}$, these terms are the recursive rays. New rays are traced from the intersection location (world space) to the light, in the reflected direction and in the transmitted direction respectively. Shadow rays are traced in the direction of the i lights, S_i is a factor indicating the success or failure of a ray reaching a light source. $I_{r\lambda}$ is the result of computing the illumination equation for the surface struck first by a reflected ray, and $I_{t\lambda}$ for a transmitted ray. Reflection rays are traced in the direction of view vector

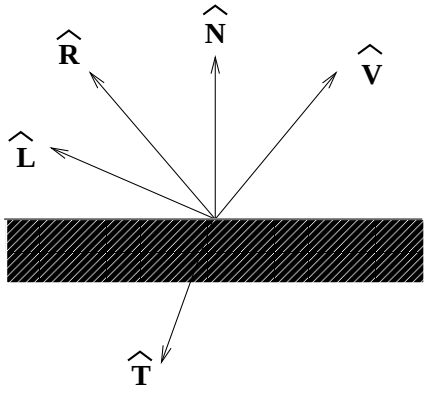

Figure 5: Interaction between an incoming ray $-V$ and a struck surface.

$\hat{V}$ reflected across the surface normal $\hat{N}$. Transmitted rays are traced in the direction indicated as $\hat{T}$ in 5. Calculations for these vectors is:

$$\hat{\mathbf{R}} = (2(\hat{\mathbf{N}} \cdot \hat{\mathbf{V}})) \cdot \hat{\mathbf{N}} - \hat{\mathbf{V}}$$

$$\hat{\mathbf{T}} = \left(\nu(\hat{\mathbf{N}} \cdot \hat{\mathbf{V}}) - \sqrt{1 - \nu^2(1 - (\hat{\mathbf{N}} \cdot \hat{\mathbf{V}})^2)}\right) - \nu\hat{\mathbf{V}}$$

where $\nu = \nu_t/\nu_i$, the transmision and incident indices of refraction. A full discussion of indices of refraction is not warranted, but needless to say ν for a vacuum is 1 everything else is larger (ν for glass is approximately 1.66).

6 Putting it all Together

To construct the beginning of a ray tracer you need to assemble these components. To compute an image, simply trace rays from $< 0, 0, 0 >$ through the locations computed in equation 1 for each screen space coordinate. Use the surface properties from the first surface struck for equation 2, the color returned from this computation is used for the color of the pixel for which the ray was traced. This color should be saved in a two dimensional array which is saved to file at the completion of the scene. The file format should not be GIF, JPG, or even BMP. Use an extremely simple format like ppm. It is not within the domain of a renderer to compress images - this is the job of an image manipulation program.

A quick note about precision. A ray tracer suffers from many aliasing problems. One of those is the inability to exactly calculate a surface point. You should always test a ray intersection calculation for being greater than some small value (I use 0.001), if an intersection is less than this value it is probably the case that you are tracing a recursive ray from a surface and you intersected exactly the same surface that the ray is attempting to leave.

7 Further Reading

There are many excellent resources for ray tracing information. Two very good texts that treat many aspects of ray tracing from theoretical and practical aspects are given here.

Watt, Mark and Alan Watt. *Advanced Animation and Rendering Techniques*, ACM Press, New York, 1992.

Glassner, Andrew ed. *An Introduction to Ray Tracing* Academic Press, 1989.

And of course the four wise men is always a good resource.

Foley, Van Damme, Feiner, Hughes. *Computer Graphics Principles and Practise*, 2nd Ed. Addison Wesley, New York, 1992.