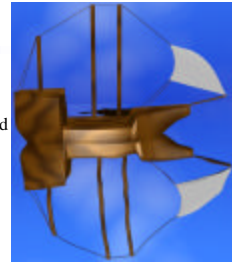


Ray Tracing

John C. Hart
 CS 319
 Advanced Computer Graphics

Why Write a Ray Tracer?

- More elegant than polygon scan conversion
- Testbed for numerous techniques and effects
 - modeling
 - rendering
 - texturing
- Easiest photorealistic renderer to implement

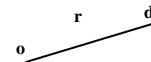


Overview

- Vector arithmetic
- Lookat viewing coordinates
- Ray casting
- Shadows
- Reflection
- Refraction

Homogeneous Coordinates

- Point $\mathbf{o} = (x_o, y_o, z_o, 1)$
- Direction $\mathbf{d} = (x_d, y_d, z_d, 0)$
- Points translate, directions do not
- Homogeneous coordinate always zero or one (no perspective distortion)
- Ray $\mathbf{r} = (\mathbf{o}, \mathbf{d}) = ((x_o, y_o, z_o, 1), (x_d, y_d, z_d, 0))$
- Ray direction \mathbf{d} always unit length
 - Not absolutely necessary
 - Makes life easier



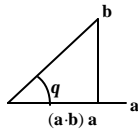
Dot Product

$$\mathbf{a} \cdot \mathbf{b} = \text{dot}(\mathbf{a}, \mathbf{b}) = \langle \mathbf{a}, \mathbf{b} \rangle$$

$$= x_a x_b + y_a y_b + z_a z_b$$

$$= |\mathbf{a}| |\mathbf{b}| \cos \theta$$

- Cosine of angle between \mathbf{a} and \mathbf{b} if both unit length
- Used to "cast a shadow" of one unit vector onto another



Cross Product

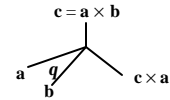
$$\mathbf{a} \times \mathbf{b} = \text{cross}(\mathbf{a}, \mathbf{b})$$

$$= (y_a z_b - z_a y_b, z_a x_b - x_a z_b, x_a y_b - y_a x_b, 0)$$

$$= \begin{vmatrix} \mathbf{x} & \mathbf{y} & \mathbf{z} \\ x_a & y_a & z_a \\ x_b & y_b & z_b \end{vmatrix}$$

$$\mathbf{a} \times \mathbf{b} = |\mathbf{a}| |\mathbf{b}| \sin \theta$$

- Returns direction perpendicular to the plane spanned by \mathbf{a} , \mathbf{b}
- Used to set up coordinate systems

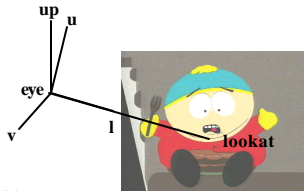


Eye LUV Coordinates

$$\mathbf{l} = (\text{lookat} - \text{eye}) / \|\text{lookat} - \text{eye}\|$$

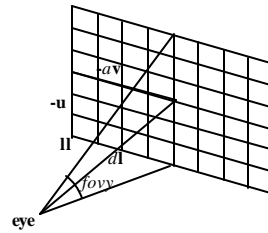
$$\mathbf{v} = (\mathbf{l} \times \text{up}) / \|\mathbf{l} \times \text{up}\|$$

$$\mathbf{u} = \mathbf{v} \times \mathbf{l}$$



- Really called Gram-Schmidt Orthonormalization

Pixels in World Coordinates



$$\mathbf{p} = \text{eye} + d\mathbf{l} - a\mathbf{v} - \mathbf{u}$$

aspect ratio $a = w/h$
focal length $d = 1/\tan(\text{fovy}/2)$

Casting Rays through Pixels

```

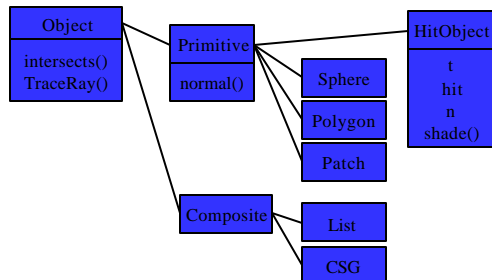
for (j = 0; j < VRES; j++) {
  for (i = 0; i < HRES; i++) {
    p = eye + 2av (double)i/HRES + 2u (double)j/VRES;
    d = (p - eye) / ||p - eye||;
    r = (eye, d);
    color = TraceRay(r);
    plot(i,j,color);
  }
}

```

TraceRay Attributes

- Invoked with ray parameter
 - Better if object database is global
 - Best if TraceRay is a member function of object database object (yikes)
- Returns a color vector (R,G,B, α)
 - α indicates transparency
 - color = α foreground + (1 - α) background

Object Hierarchy



TraceRay Definition

```

Color TraceRay(Ray r, int depth) {
  HitObject ho;
  color c = background;
  if (intersects(r,ho)) {
    c = ho.shade(lights); /* casts a shadow ray */
    c += ho.ks*TraceRay(ho.reflect(r), depth-1);
    c += ho.kr*TraceRay(ho.refract(r), depth-1);
  }
  return c;
}

```

List Definition

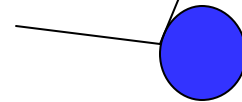
```

class List : public Composite {
    Object item[100];
    int n;
    boolean intersects(Ray r, HitObject &ho) {
        min_t = 1.e20;
        for (i = 0; i < n; i++)
            if (item[i].intersects(r,o) && o.t < min_t) {
                min_t = o.t; ho = o;
            }
        combine_shade(ho);
        return (min_t < 1.e20);
    }
}
    
```

shade Definition

```

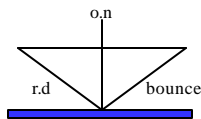
Color shade(Lights lights) {
    Color c = ka;
    for (l = 0; l < lights.n; l++) {
        ldir = normalized(lights[l].pos - hit);
        if (!intersects(ray(hit,ldir)))
            c += kd*cd*dot(n, ldir) + ...
    }
    return(c);
}
    
```



reflect Definition

```

Ray reflect(Ray r) {
    Ray bounce;
    bounce.o = hit;
    bounce.d = 2.0*dot(n,-r.d)*n + r.d;
    return(bounce);
}
    
```

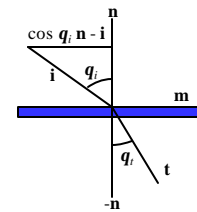


Calculating Refraction

Snell's Law: $n_i \sin q_i = n_t \sin q_t$
 Let $h = n_i / n_t = \sin q_t / \sin q_i$

$$\begin{aligned}
 t &= \sin q_t \mathbf{m} - \cos q_t \mathbf{n} \\
 &= (\sin q_t / \sin q_i) (\cos q_i \mathbf{n} - \mathbf{i}) - \cos q_t \mathbf{n} \\
 &= (h \cos q_i - \cos q_t) \mathbf{n} - h \mathbf{i}
 \end{aligned}$$

$$\mathbf{t} = (h(\mathbf{n} \cdot \mathbf{i}) - \sqrt{1 - h^2(1 - (\mathbf{n} \cdot \mathbf{i})^2)}) \mathbf{n} - h \mathbf{i}$$



$$\mathbf{m} = (\cos q_i \mathbf{n} - \mathbf{i}) / \sin q_i$$

$$\cos q_t = \sqrt{1 - \sin^2 q_i} = \sqrt{1 - h^2 \sin^2 q_i}$$

refract Definition

$$\mathbf{t} = (h(\mathbf{n} \cdot \mathbf{i}) - \sqrt{1 - h^2(1 - (\mathbf{n} \cdot \mathbf{i})^2)}) \mathbf{n} - h \mathbf{i}$$

```

Ray refract(Ray r) {
    Ray t;
    double ni = dot(n,-r.d);
    eta = current_index/new_index;
    t.o = hit;
    t.d = (eta*ni - sqrt(1.0 - eta*eta*(1-ni*ni)))*n + eta*r.d;
    return(t);
}
    
```