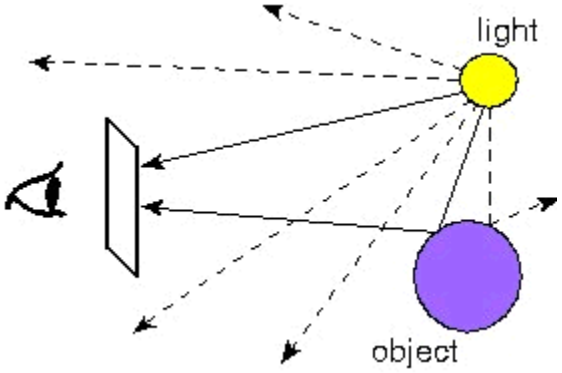
Ray Tracing

Idea



Direct approach: Trace rays from the light source to the eye. Lots of rays are wasted because they never reach the eye.

The Ray Tracing approach: Trace rays starting from the eye, through the image plane and into the scene. Primary rays are those which directly intersect an object (the part closest to the eye) from the eye point. Secondary rays are shot from this intersection point. There are three types:

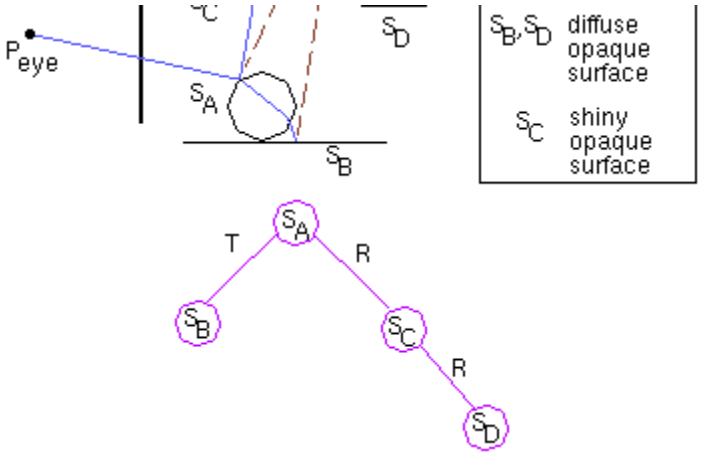
- Shadow rays
- Reflected rays
- Refracted rays.

Shadow rays are directed at the light sources and determine if the region is in shadow. Reflected and refracted rays model the mirror-like and transparency characteristics of the object. A local lighting model is applied at the intersection point (with an ambient term) and combined with the reflected ray and refracted ray contributions. Ray Tracing models specular reflection and refractive transparency well, but models global lighting contributions poorly. This is because it uses a directionless ambient light term for approximating diffuse scattering. Radiosity methods are used to overcome this deficiency.

- basic algorithm

```
raytrace(ray)
{
    find closest intersection
    cast shadow ray, calculate colour_local
    if (object is shiny) colour_reflect = raytrace(reflected_ray)
    if (object is transparent) colour_refract = raytrace(refracted_ray)
    colour = k1*colour_local + k2*colour_reflect + k3*colour_refract
    return(colour)
}
```

- points on rays $P(t) = p + t\mathbf{v}$, $t >= 0$
  - p - starting point (e.g. eye point)
  - **v** - directional unit vector
- example



- cast ray from eye point, going through image plane
  - closest intersection with S_A
  - shadow ray unobstructed, intersection point lit directly by light, so colour_local is nonzero
  - S_A is shiny, so cast reflected ray
    - closest intersection with S_C
    - shadow ray obstructed (by S_C), intersection point not lit, so colour_local is zero
    - S_C is shiny, so cast reflected ray
      - closest intersection with S_D
      - shadow ray unobstructed, intersection point lit, so colour_local is nonzero
      - S_D is diffuse, no reflected ray, so colour_reflect is zero
      - S_D is opaque, no refracted ray, so colour_refract is zero
      - return total colour
    - S_C is opaque, no refracted ray, so colour_refract is zero
    - return total colour
  - S_A is transparent, so cast refracted ray
    - closest intersection with S_B
    - shadow ray unobstructed, intersection point lit, so colour_local is nonzero
    - S_B is diffuse, no reflected ray, so colour_reflect is zero
    - S_B is opaque, no refracted ray, so colour_refract is zero
    - return total colour
  - return total colour
- need to find intersections between rays and surfaces
- so need mathematical definitions of primitives
  - e.g. sphere, cone, polygon, etc.

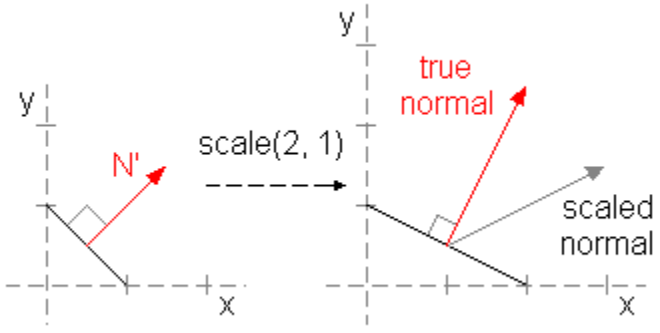
Example

- consider a simple scene with a unit sphere at origin
  - $x^2 + y^2 + z^2 = 1$
- let P = (x, y, z) be a point on sphere, then the equation reduces to
  - $P \cdot P = 1$
- solve equation to find intersection between a ray and unit sphere
  - $P \cdot P = 1$
  - $(p + t\mathbf{v}) \cdot (p + t\mathbf{v}) = 1$
  - $p^2 + 2tp\mathbf{v} + t^2\mathbf{v}^2 = 1$
  - $\mathbf{v}^2t^2 + 2p\mathbf{v}t + (p^2 - 1) = 0$

- $t = [-2p\mathbf{v} \pm \text{sqrt}((2p\mathbf{v})^2 - 4\mathbf{v}^2(p^2 - 1))] / 2\mathbf{v}^2$
- numerical example
  - eye point = p = (0, sqrt(2)/2, 3)
  - direction = v = (0, 0, -1)
  - find closest intersection point
    - $p\mathbf{v}$ = (0, sqrt(2)/2, 3) · (0, 0, -1) = -3
    - p^2 = (0, sqrt(2)/2, 3) · (0, sqrt(2)/2, 3) = 9.5
    - $\mathbf{v}^2$ = (0, 0, -1) · (0, 0, -1) = 1
    - so $t = [-2(-3) \pm \text{sqrt}((2(-3))^2 - 4(1)((9.5) - 1))] / 2(1)^2 = [6 \pm \text{sqrt}(2)] / 2 = 3 \pm \text{sqrt}(2)/2$
    - smallest positive t value is our intersection point
    - so t = 3 - sqrt(2)/2
    - so p = (0, sqrt(2)/2, 3) + (3 - sqrt(2)/2)(0, 0, -1) = (0, sqrt(2)/2, sqrt(2)/2)
  - for colour calculation, we also need the surface normal at the intersection point
    - since this is a unit sphere, **N** = (0, sqrt(2)/2, sqrt(2)/2)
- we need a method for arbitrary spheres (i.e. not unit and not at origin)

In general

- we want to find the intersection between an arbitrary primitive and a ray
- there is a series of transformations M (the combined transformation matrix) that transform a unit primitive (of the correct type) into our arbitrary primitive
  - i.e. M transforms the unit primitive frame into the arbitrary primitive frame
- so M^{-1} transforms our ray into the unit primitive frame, where we can apply the unit primitive equations to solve for an intersection
- idea
  - given ray = p + t**v**
  - calculate ray' = M^{-1} · ray = M^{-1} · p + t M^{-1} · **v** = p' + t**v'**
  - calculate intersection using ray' with unit primitive, solve for t
  - substitute t into original ray = p + t**v** to get intersection point
- for surface normal
  - at first glance it seems we can apply the transformation M to the surface normal of the unit primitive to get the normal for our arbitrary primitive
  - this will work for translations, rotations and uniforming scaling
  - but consider non-uniform scaling (we'll work with unnormalized surface normals to simplify the math)



  - an edge from (0, 1) to (1, 0) has a surface normal (1, 1)
  - if we apply scale(2, 1) (a stretch in the x direction by a factor of 2) to the normal, we get (2, 1)

- but the true surface normal is (1, 2)

- to transform a surface normal **N'** from a unit primitive into a surface normal **N** for our arbitrary primitive
  - $\mathbf{N} = M^{-1T} \cdot \mathbf{N'}$ (may need to re-normalize)
- for the example above

$$\mathbf{N'} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, M = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, M^{-1T} = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\text{so } \mathbf{N} = \begin{bmatrix} 1/2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1/2 \\ 1 \\ 0 \end{bmatrix} = 1/2 \times \text{true surface normal}$$

  - i.e. **N** and the true surface normal have the same direction, are the same normalized vector
- why the transpose of the inverse of M works
  - clearer explanation (thanks Andriy!)
    - for a point P on the the unit primitive, the normal **N'** is defined to be the vector perpendicular to all tangent vectors at P
    - let **T'** be a tangent vector at P (on the unit primitive)
    - then $\mathbf{N'} \cdot \mathbf{T'} = 0$
    - so $\mathbf{N'}^T\mathbf{T'} = 0$ (using matrix multiplication notation)
    - $\mathbf{N'}^T(M^{-1}M)\mathbf{T'} = 0$
    - $(\mathbf{N'}^TM^{-1})(M\mathbf{T'}) = 0$
    - $(M^{-1T}\mathbf{N'})^T(M\mathbf{T'}) = 0$, since $(AB)^T = B^TA^T$
    - $(M^{-1T}\mathbf{N'}) \cdot (M\mathbf{T'}) = 0$
    - $\mathbf{T} = M\mathbf{T'}$ is the transformed tangent vector (on our arbitrary primitive)
    - since $(M^{-1T}\mathbf{N'}) \cdot \mathbf{T} = 0$, $M^{-1T}\mathbf{N'}$ is perpendicular to **T**
    - thus $\mathbf{N} = M^{-1T}\mathbf{N'}$ is the surface normal for our arbitrary primitive, and M^{-1T} is the correct transformation
  - old explanation
    - since we're applying M^{-1T} to a vector, translations within M have no effect
      - vectors represent directions, not points in space
    - M^{-1T} preserves any rotations in M
      - let M_1 = rotate(z, a), M_2 = rotate(z, -a)
      - then $M_1 = M_2^{-1}$, or $M_2 = M_1^{-1}$ (M_2 reverses the effect of M_1, and vice versa)
      - also

$$M_1 = \begin{bmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$M_2 = \begin{bmatrix} \cos(-a) & -\sin(-a) & 0 & 0 \\ \sin(-a) & \cos(-a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos(a) & \sin(a) & 0 & 0 \\ -\sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = M_1^T$$

        - so $M_2 = M_1^T$, or $M_1 = M_2^T$
        - thus $M_1 = M_2^T = (M_1^{-1})^T = M_1^{-1T}$
        - notice in the example above
            - the orientation of the scaled surface is rotated slightly counterclockwise, and so the orientation of the true normal is rotated slightly counterclockwise
            - whereas the orientation of the scaled normal is rotated slightly clockwise
    - the transpose of the inverse alters any scaling in M so that the resultant vector is rotated in the correct direction