Ray Tracing

Ray Tracing/Casting: Recursive photon tracking vs. visibility probing

• Setting: eyepoint, virtual screen (an array of virtual pixels, and scene are organized in convenient coordinate frames (e.g., all in view or world frames)
• Ray: a half line determined by the eyepoint and a point associated with a chosen pixel
• Interpretations:
  – Ray is the path of photons that successfully reach the eye
    (we simulate selected photon transport through the scene)
  – Ray is a sampling probe that gathers color/visibility information
**Ray Tracing:** Recursive

- Eye-screen ray is the *primary ray*
- Backward tracking of photons that could have arrived along primary
- Intersect with objects
- Determine nearest object
- Generate secondary rays
  - to light sources
  - in reflection-associated directions
  - in refraction-associated directions
- Continue recursively for each secondary ray
- Terminate after suitably many levels
- Accumulate suitably averaged information for primary ray
- Deposit information in pixel
Ray Casting: Nonrecursive

- As above for ray tracing, but stop before generating secondary rays
- Apply illumination model at nearest object intersection with no regard to light occlusion
- Ray becomes a sampling probe that just gathers information on
  - visibility
  - color
Intersection Computations

General Issues:

- Ray: express in parametric form

\[ E + t(P - E) \]

where \( E \) is the eyepoint and \( P \) is the pixel point
- Scene Object: direct implicit form
  - express as \( f(Q) = 0 \) when \( Q \) is a surface point, where \( f \) is a given formula
  - intersection computation is an equation to solve:
    find \( t \) such that \( f(E + t(P - E)) = 0 \)
- Scene Object: procedural implicit form
  - \( f \) is not a given formula
  - \( f \) is only defined procedurally
  - \( \mathcal{A}(f(E + t(P - E)) = 0) \) yields \( t \), where \( \mathcal{A} \) is a root finding method (secant, Newton, bisection, etc.)
Quadric Surfaces:

- Surface given by
  \[ A x^2 + B x y + C x y + D y^2 + E y z + F z^2 + G x + H y + J z + K = 0 \]

- Ray given by
  \[
  \begin{align*}
  x &= x_E + t(x_P - x_E) \\
  y &= y_E + t(y_P - y_E) \\
  z &= z_E + t(z_P - z_E)
  \end{align*}
  \]

- Substitute ray \( x, y, z \) into surface formula
  - quadratic equation results for \( t \)
  - organize expression terms for numerical accuracy; i.e., to avoid
    * cancelation
    * combinations of numbers with widely different magnitudes
Polygons:

- The plane of the polygon should be known
  \[ Ax + By + Cz + D = 0 \]
  - \((A, B, C, 0)\) is the normal vector
  - pick three successive vertices
    \[ v_{i-1} = (x_{i-1}, y_{i-1}, z_{i-1}) \]
    \[ v_i = (x_i, y_i, z_i) \]
    \[ v_{i+1} = (x_{i+1}, y_{i+1}, z_{i+1}) \]
  - should subtend a “reasonable” angle
    (bounded away from 0 or 180 degrees)
  - normal vector is the cross product \(v_{i-1} \times v_{i+1} - v_i\)
  - \(D = -(Ax + By + Cz)\) for any vertex \((x, y, z, 1)\) of the polygon
- Substitute ray \(x, y, z\) into surface formula
  - linear equation results for \(t\)
Solution provides planar point \((\overline{x}, \overline{y}, \overline{z})\) – is this inside or outside the polygon?
Planar Coordinates:

- Take origin point and two independent vectors on the plane

\[ \mathcal{O} = (x_0, y_0, z_0, 1) \]
\[ \vec{b}_0 = (u_0, v_0, w_0, 0) \]
\[ \vec{b}_1 = (u_1, v_1, w_1, 0) \]

- Express any point in the plane as

\[ P - \mathcal{O} = \alpha_0 \vec{b}_0 + \alpha_1 \vec{b}_1 \]

- intersection point is \((\vec{\alpha}_0, \vec{\alpha}_1, 1)\)
- clipping algorithm against polygon edges in these \(\alpha\) coordinates

Alternatives: Must this all be done in world coordinates?

- Alternative: Intersections in model space
  - form normals and planar coordinates in model space and store with model
– backtransform the ray into model space using inverse modeling transformations
– perform the intersection and illumination calculations

• Alternative: Intersections in world space
  – form normals and planar coordinates in model space and store
  – forward transform using modeling transformations
Normal Transformations: How do affine transformations affect surface normals?

- Let $P_a$ and $P_b$ be any two points on an object
  - arbitrarily close
  - $\vec{n} \cdot (P_b - P_a) = 0$
  - using transpose notation: $(\vec{n})^T(P_b - P_a) = 0$
- After an affine transformation on $P_a, P_b$:

  $$M(P_b - P_a) = MP_b - MP_a$$

we want $(N\vec{n})$ to be a normal for some transformation $N$:

  $$(N\vec{n})^T M(P_b - P_a) = 0$$
  \[\implies \vec{n}^T N^T M(P_b - P_a)\]

and this certainly holds if $N = (M^{-1})^T$

- Only the upper 3-by-3 portion of $M$ is pertinent for vectors
• Translation:

\[
(M^{-1})^T = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
-\Delta x & -\Delta x & -\Delta x & 1
\end{bmatrix} \quad \rightarrow \quad \begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x_n \\
y_n \\
z_n
\end{bmatrix} = \begin{bmatrix}
x_n \\
y_n \\
z_n
\end{bmatrix}
\]

\[\implies \text{no change to the normal}\]
Rotation (example):

\[(M^{-1})^T = \begin{bmatrix}
\cos(-\theta) & \sin(-\theta) & 0 & 0 \\
-\sin(-\theta) & \cos(-\theta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}^T
\]

\[= \begin{bmatrix}
\cos(-\theta) & \sin(-\theta) & 0 & 0 \\
-\sin(-\theta) & \cos(-\theta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

\[\rightarrow \begin{bmatrix}
\cos(-\theta) & \sin(-\theta) \\
-\sin(-\theta) & \cos(-\theta) \\
0 & 0 \\
0 & 0
\end{bmatrix}
\]

\[\Rightarrow \text{rotation applied unchanged to normal}\]
• Scale:

\[(M^{-1})^T = \begin{bmatrix}
\frac{1}{s_x} & 0 & 0 & 0 \\
0 & \frac{1}{s_y} & 0 & 0 \\
0 & 0 & \frac{1}{s_z} & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \longrightarrow \begin{bmatrix}
\frac{1}{s_x} & 0 & 0 \\
0 & \frac{1}{s_y} & 0 \\
0 & 0 & \frac{1}{s_z}
\end{bmatrix}
\]

\[
\begin{bmatrix}
\frac{1}{s_x} & 0 & 0 \\
0 & \frac{1}{s_y} & 0 \\
0 & 0 & \frac{1}{s_z}
\end{bmatrix} \begin{bmatrix}
n_x \\
n_y \\
n_z
\end{bmatrix} = \begin{bmatrix}
n_x \\
n_y \\
n_z
\end{bmatrix}
\]

\[\Rightarrow \text{reciprocal scale applied to normal}\]
Surface Information

Surface Normals:

- Illumination models require:
  - surface normal vectors at intersection points
  - ray-surface intersection computation must also yield a normal
  - light-source directions must be established at intersection
  - shadow information determined by light-ray intersections with other objects

- Normals to polygons:
  - provided by planar normal
  - provided by cross product of adjacent edges

- Normals to any implicit surface (e.g., quadrics)
  - move from \((x, y, z)\) to \((x + \Delta x, y + \Delta y, z + \Delta z)\) which is maximally far from the surface
  - direction of greatest increase to \(f(x, y, z)\)
• Taylor series:

\[ f(x + \Delta x, y + \Delta y, z + \Delta z) = f(x, y, z) + [\Delta x, \Delta y, \Delta z] \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial z} \end{bmatrix} + \cdots \]

- maximality for the gradient vector of \( f \)
- not normalized

• Normal to a quadric surface

\[
\begin{bmatrix}
2Ax + By + Cz + G \\
Bx + 2Dy + Ez + H \\
Cx + Ey + 2Fz + J \\
0
\end{bmatrix}
\]