Visibility and Ray Casting

OUTLINE:

Rendering=Visibility+Shading

Ray Casting

Visibility Algorithms

rendering = visibility + shading

(what's visible?) *(what color is it?)*

(old terminology: visibility = "hidden surface removal")

The Visibility Problem

PROBLEM STATEMENT:

GIVEN: a set of 3-D surfaces, a projection from 3-D to 2-D screen (typically perspective or parallel)

DETERMINE: the nearest surface encountered at any point on 2-D screen

Two approaches:

- Continuous *(object space)* methods provide visibility info. for all points of screen

  The output, a planar subdivision, can be converted into a raster image with a variety of techniques.

- Discrete *(image space)* methods provide visibility info. for only a sampling of screen points

  Output is a raster image.

Discrete *(Image Space)* Visibility

early taxonomy: [Sutherland-Sproull-Schumacker74]

Four of the simplest algorithms:

Z-buffer

```
initialize z-buffer
loop objects
    loop y
        loop x
            if z(x,y) < zbuf[x,y]
                zbuf[x,y] = z(x,y)
                write image pixel
```

Ray Casting (nonrecursive Ray Tracing)

```
loop y
    loop x
        loop objects
            find object with min z
        write pixel
```

Painter's

```
sort objects by z (back-to-front)
loop objects
    loop y
        loop x
            write pixel
```

Scanline

```
sort objects by y
loop y
    sort active objects by x
    loop x
        find active object with min z
        write pixel
```

Ray Casting

A very flexible visibility algorithm.

```
loop y
    loop x
        shoot ray from eye point through pixel (x,y) into scene
        intersect with all surfaces, find first one the ray hits
        shade that point to compute the color of pixel (x,y)
```

A ray: **p**+t**d**, where **p** is the origin of the ray and **d** is the direction of the ray.

Typically assume |**d**|=1.

p and **d** typically computed in world space.

Notation and Terminology

We represent affine and perspective transformations with homogeneous 4x4 matrices.

Transform a point from *a* space to *b* space (e.g. *a*=world, *b*=screen).

Superscript denotes space (a or b); subscript denotes coordinate (x, y, z, or w)

To transform (p_x^a, p_y^a, p_z^a), a point in *a* space, to *b* space:

$$\mathbf{p^b} = \mathbf{M^{ab}p^a} = \begin{bmatrix} p_x^b \\ p_y^b \\ p_z^b \\ p_w^b \end{bmatrix} = \begin{bmatrix} M_{00} & M_{01} & M_{02} & M_{03} \\ M_{10} & M_{11} & M_{12} & M_{13} \\ M_{20} & M_{21} & M_{22} & M_{23} \\ M_{30} & M_{31} & M_{32} & M_{33} \end{bmatrix} \begin{bmatrix} p_x^a \\ p_y^a \\ p_z^a \\ 1 \end{bmatrix}$$ homog. b space coords.

Project to w=1 plane with homogeneous division:

$$\mathbf{p^{b'}} = \begin{bmatrix} p_x^b / p_w^b \\ p_y^b / p_w^b \\ p_z^b / p_w^b \end{bmatrix}$$ projected b space coords.

Generating Rays from the Eye, part 1

PROBLEM:

Given

$\mathbf{M}_{sw}$, screen-to-world transformation matrix

(if all you have is world-to-screen matrix, then invert it to get $\mathbf{M}^{sw}$)

(x,y): screen space coordinates of pixel of interest

Generate a ray from the eye point through pixel (x,y), yielding a point **p** and a unit direction vector **d** in world space.

APPROACH: take two points on the ray in screen space, and transform them to world space.

First point: $\mathbf{p}_s$ = eye point

$= (0,0,-\infty)^T$ in projected screen space

$= (0,0,-1,0)^T$ in homogeneous screen space (since $-1/0 = -\infty$)

Second point: $\mathbf{q}_s$ = a point on ray at depth z=0

$= (x,y,0,1)^T$ in homogeneous screen space

Generating Rays from the Eye, part 2

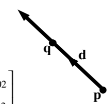
Transform point **p** to world space:

$$\mathbf{p}^w = \mathbf{M}^{sw}\mathbf{p}^s = \mathbf{M}^{sw}\begin{bmatrix}0\\0\\-1\\0\end{bmatrix} = \begin{bmatrix}-M_{02}\\-M_{12}\\-M_{22}\\-M_{32}\end{bmatrix} = \begin{matrix}\text{homog.}\\\text{world}\\\text{space}\end{matrix} \xrightarrow{\textit{project}} \mathbf{p}^{w'} = \begin{bmatrix}M_{02}/M_{32}\\M_{12}/M_{32}\\M_{22}/M_{32}\end{bmatrix} = \begin{matrix}\text{projected}\\\text{world}\\\text{space}\end{matrix}$$

note: if using transposed matrices (as in SGI's GL) then swap subscripts.

Transform point **q** to world space:

$$\mathbf{q}^w = \mathbf{M}^{sw}\mathbf{q}^s = \mathbf{M}^{sw}\begin{bmatrix}x\\y\\0\\1\end{bmatrix} = \begin{bmatrix}q_x^w\\q_y^w\\q_z^w\\q_w^w\end{bmatrix} \xrightarrow{\textit{project}} \mathbf{q}^{w'} = \begin{bmatrix}q_x^w/q_w^w\\q_y^w/q_w^w\\q_z^w/q_w^w\end{bmatrix}$$



Find direction vector $\mathbf{d}_{w'}$:

$$\mathbf{d}^{w'} = \mathbf{q}^{w'} - \mathbf{p}^{w'} = \begin{bmatrix}q_x^w/q_w^w\\q_y^w/q_w^w\\q_z^w/q_w^w\end{bmatrix} - \begin{bmatrix}M_{02}/M_{32}\\M_{12}/M_{32}\\M_{22}/M_{32}\end{bmatrix} \quad \text{or} \quad \mathbf{d}^{w'} = \begin{bmatrix}q_x^w M_{32} - q_w^w M_{02}\\q_y^w M_{32} - q_w^w M_{12}\\q_z^w M_{32} - q_w^w M_{22}\end{bmatrix}$$

then normalize $\mathbf{d}_{w'}$ to unit length. Your ray is $\mathbf{p}_{w'} + t\mathbf{d}_{w'}$ for $t \geq 0$.

Generating Rays from the Eye, part 3

Another way to generate rays in a ray tracer is to use the camera parameters (such as viewpoint, up vector, view direction, angle of view) to generate the origin and direction of the day directly in world space.

But that method requires that the modeling software pass additional information to the renderer, and restricts the set of transformations that are allowed between eye space and screen space.

Working from the matrix $\mathbf{M}_{sw}$, as in the previous slides, while perhaps trickier conceptually, is more general and leads to better modularization.