Shadows
One Slide Solution

- It is really very simple
- Can you see something from the eye position? Yes, then visible. No, then not visible (occluded)
- Can you see something from a light source position? Yes, then not in shadow. No, then in shadow
- If you know HLHSR, then do that from the light instead of the eye location
Multiple Slides Solution

- But there can be multiple light sources
- The light source might not be a single point or a single direction (e.g., extended sources)
- Want to determine both visibility and lighting without multiple transforms
Two-Pass Object Precision

- 1st pass: transform to light position
  - hidden surface determination (polygons which are not in shadow)
- 2nd pass: transform to original world coordinate sys
  - polygons not in shadow are merged to become surface detail polygons (which algorithm?)
- Postprocessing: transform to eye coordinate
  - visible surface determination + surface details
Two-pass Image Precision

- Z buffer from eye (e): what the viewer can see
- Z buffer from light (l): what the light source can see
- for each (xe, ye, ze)
  - transform to (xl, yl, zl)
  - is zl more distant than z(xl, yl)
    - yes, (xe, ye) is in shadow
    - no, (xe, ye) is not in shadow
Computer Graphics
Shadow Volume

- Light Source
- Shadow Volume Outline
- Model
- Lit Scene
- Shadowed Scene
Shadow Volume

- Enclosed by
  - (side) shadow polygons
  - scene polygon
  - back shadow polygon (scaled version of the original scene polygon)

- Shadow polygons are invisible and not rendered (used to determine whether an object is in shadow)

- SV polygons = scene polygon + all shadow polygons
Shadow Volume

- From the viewer
  - each front-facing (normal pointing to the viewer) SV polygon causes object to be in shadow
  - each back-facing (normal pointing away from the viewer) SV polygon causes object to be out of shadow
  - #FF intersections $\geq$ #BF intersections to be in shadow
Shadow Volume

- How do you do this?
- A modified depth-sort type algorithm
  - include SV polygons in the depth-sort list but process them front-to-back (instead of back-to-front)
  - determine whether the eye is in any SV
  - then count how many times the projection ray intersects FF and BF SV polygons
  - easier said than done
Soft Shadow

Fig. 16.48 Umbra and penumbra.
Soft Shadow

Polyhedron casting shadow

Area light

Umbra

Penumbra

Penumbra volume
Using BSP Tree

- Stationary light source
- Stationary scene
- Moving camera
- Basic BSP tree algorithm
  - Construct a tree based on scene polygons
  - Determine rendering order
- Enhancement
  - Polygons need surface details for right order and appearance
  - Order is taken care of by basic BSP
  - How about surface details?
Intuition

- Surface details (in shadow or not) are stationary regardless of camera position
  - Find once
    - if a polygon is in shadow or not, and
    - Which part is in shadow (surface detail polygons)

- Which polygon is NOT in shadow
  - The one that is closest to the light source

- The polygon 2\textsuperscript{nd} closest to the light source can only have shadow from the closest polygons
- The polygon 3\textsuperscript{rd} closest to the light source can only have shadow from the 1\textsuperscript{st} and 2\textsuperscript{nd} closest polygons, etc.
SVBSP Tree

- A binary tree
- Each node is a SV polygon (instead of a scene polygon)
- Space is divided into IN/OUT by a node (a SV polygon, normal pointing out)
- Leaf nodes are labeled IN/OUT
Computer Graphics
Computer Graphics
SVBSP Tree Construction

- Ordering is important
  - the polygon which is closest to the light source must be used first
  - the polygon which is 2nd closest to the light source then filtered down the SVBSP tree to generate surface details polygons
  - add the 2nd closest polygons to SVBSP tree
  - the polygon which is 3rd closest to the light source then filtered down the SVBSP tree to generate surface details polygons
  - add the 3rd closest polygons to SVBSP tree
  - ...

Computer Graphics
How to know which polygon is closest (2nd, 3rd closest ....) to the light source?

Use the regular BSP Tree

- traverse according to the light source position
  - first the half containing light
  - then the partition plane
  - then the half not containing light

First pass (SVBSP): surface details

Second pass (BSP): eye locations for rendering
Other Possibilities

- Ray Tracing
  - with shadow rays to the sources
- Radiosity
  - with form factor computation
- Later
Fake Shadow

- Shadow generation is not trivial
  - OpenGL does not do it

- Reason
  - Shading calculation can be based entirely on “local” information, while shadow calculation cannot (need to know the relative position of many objects)

- In reality
  - Shadow does not have to be entirely correct, it just has to be realistic
Fake Shadow (cont.)

- Usually, in an indoor environment
  - Light is on the ceiling
  - Walls and floor enclose the scene (and they are planar)
  - Cast shadows on those enclosing surfaces by projecting objects onto them
Example

- Figure out the projection transform from \((x, y, z, 1)\) to \((i, j, 1)\)
- Apply this transform to all scene polygons
- Draw projected polygons in dark (shadow) colors
Math

\[ \begin{align*}
\text{line} & \quad \begin{aligned}
x &= l_x + t(p_x - l_x) \\
y &= l_y + t(p_y - l_y) \\
z &= l_z + t(p_z - l_z)
\end{aligned} \\
\text{plane} & \quad \begin{aligned}
z &= 0 \\
\Rightarrow l_z + t(p_z - l_z) &= 0 \\
\Rightarrow t &= -\frac{l_z}{(p_z - l_z)} \\
\Rightarrow x &= \frac{l_z p_x - l_x p_z}{(p_z - l_z)}, \\
y &= \frac{l_z p_y - l_y p_z}{(p_z - l_z)}
\end{aligned}
\end{align*} \]