Visible Surface Detection

Werner Purgathofer
Visibility in the Rendering Pipeline

- scene objects in object space
- transformed vertices in clip space
- scene in normalized device coordinates
- raster image in pixel coordinates

**Vertex Stage**
- object capture/creation
- modeling
- viewing
- projection

**Viewport Transformation**
- clipping + homogenization
- viewport transformation
- rasterization
- shading

**Pixel Stage**
- pixel stage ("fragment shader")
3D Display: Wireframe Display

© ZwCAD Software Co.,Ltd
3D Display: Depth Cueing

depth cueing = intensity decreases with increasing distance
3D Display: Visibility

- visible line and surface identification

\[
\begin{array}{c}
\text{visible line and surface identification}
\end{array}
\]
3D Display: Visibility
- only visible lines
- intensity decreases with increasing distance
3D Display: Shaded Display

shading + depth cueing:

© Ansgar Philippsen
Visible-Surface Detection

- identifying visible parts of a scene (also *hidden-surface elimination*)
- type of algorithm depends on:
  - complexity of scene
  - type of objects
  - available equipment
  - static or animated displays
- *object-space* methods
  - objects compared to each other
- *image space* methods
  - point by point at each pixel location
  - often sorting and coherence used
the following algorithms are examples for different classes of methods

- back-face detection
- depth buffer method
- scan-line method
- depth-sorting method
- area-subdivision method
- octree methods
- ray-casting method
surfaces (polygons) with a surface normal pointing away from the eye cannot be visible (back faces)

⇒ eliminate them before visibility algorithm!

can be eliminated:  

![Diagram showing back-face detection](image-url)
Back-Face Detection (2)

- eliminating back faces of closed polyhedra
- view point \((x,y,z)\) “inside” a polygon surface if
  \[
  Ax + By + Cz + D < 0
  \]
- or polygon with normal \(N=(A, B, C)\) is a back face if
  \[
  V_{\text{view}} \cdot N > 0
  \]
Back-Face Detection (3)

- object description in viewing coordinates \( \Rightarrow \quad V_{\text{view}} = (0,0,V_z) \)

\[
V_{\text{view}} \cdot N = V_z C
\]

- sufficient condition: if \( C \leq 0 \) then back-face
Back-Face Detection (4)

- complete visibility test for non-overlapping convex polyhedra

- preprocessing step for other objects:
  about 50% of surfaces are eliminated
Depth-Buffer Method (1)

- z-buffer method
- image-space method
- hardware implementation
- no sorting!
Depth-Buffer Method (2)

- two buffers
  - refresh buffer (intensity information)
  - depth buffer (distance information)

size corresponds to screen resolution
(for every pixel: r, g, b, z)

draw something =
  - compare z with z in buffer
  - if z closer to viewer
  - then draw and update z in buffer
  - else nothing!
Depth-Buffer Algorithm Example

polygons with corresponding z-values

image
back-ground

depth-buffer

-1 -1 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1

-1 -1 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1

-1 -1 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1
-1 -1 -1 -1

-1 .3 .3
-1 .3 .3
-1 .3 .3
-1 .3 .3

-1 .8 .7
-1 .7 .6
-1 .7 .6
-1 .7 .6

-1 .6 .5 .4
-1 .6 .5 .4
-1 .6 .5 .4
-1 .6 .5 .4

-1 .8 .7
-1 .7 .6
-1 .7 .6
-1 .7 .6

-1 .6 .5 .4
-1 .6 .5 .4
-1 .6 .5 .4
-1 .6 .5 .4
Depth-Buffer Algorithm

for all (x,y)
  frameBuff(x,y) = backgroundColor
  depthBuff(x,y) = -1
for each polygon P
  for each position (x,y) on polygon P
    calculate depth z
    if z > depthBuff(x,y) then
      depthBuff(x,y) = z
      frameBuff(x,y) = surfColor(x,y)
Ax + By + Cz + D = 0

depth at (x,y):
\[ z = \frac{-Ax - By - D}{C} \]

depth at (x+1,y):
\[ z' = \frac{-A(x+1) - By - D}{C} = z - \frac{A}{C} \]

depth at (x,y-1):
\[ z'' = \frac{-Ax - B(y-1) - D}{C} = z + \frac{B}{C} \]
determine y-coordinate extents of polygon P
$z = \frac{-Ax - By - D}{C}$

$y' = y - 1$

$x' = x - \frac{1}{m}$

$\Rightarrow z' = \frac{-A(x-1/m) - B(y-1) - D}{C}$

$= z + \frac{A/m + B}{C}$

constant!
Scan-Line Method

- image-space method
- extension of scan-line algorithm for polygon filling
Scan-Line Method: Edge & Polygon Tables

- **edge table** (all edges, y-sorted)
  - coordinate endpoints
  - inverse slope
  - pointers into polygon table

- **polygon table** (all polygons)
  - coefficients of plane equation
  - intensity information
  - (pointers into edge table)
Scan-Line Method: Active Edge List

- active edge list (all edges crossing current scanline, x-sorted, flag)
Scan-Line Method Example

Edge Table: \(2, 3, 1, 5, 1, 1, 2, 2, 5, 4, 3, 3, 4, 4\)

Polygon Table: \(\square, \square, \square, \square\)

- active edges
  - \(3, 2\)
  - \(1, 5, 2, 1, 3, 1\)
  - \(2, 2, 5, 3, 1, 5\)
  - \(3, 3, 1, 5\)
  - \(3, 1\)

- active polygons
  - \(\square\)
  - \(\square, \square, \square\)
  - \(\square, \square, \square\)
  - \(\square, \square\)
  - \(\square\)
coherence between adjacent scan lines
- incremental calculations
- active edge lists very similar
  (easy sorting, avoid depth calculations)
Depth-Sorting Method: Overview

- surfaces sorted in order of decreasing depth (viewing in –z-direction)
  - “approximate”-sorting using smallest z-value (greatest depth)
  - fine-tuning to get correct depth order
- surfaces scan converted in order
- sorting both in image and object space
- scan conversion in image space
- also called “painter’s algorithm”
surface S with greatest depth is compared to all other surfaces $S'$

- no depth overlap $\rightarrow$ ordering correct
- depth overlap $\rightarrow$ do further tests in increasing order of complexity

2 surfaces with no depth overlap
ordering correct if
- bounding rectangles in xy-plane do not overlap
- check x-, y-direction separately

2 surfaces with depth overlap but no overlap in the x-direction
ordering correct if

- S completely behind S'
- substitute vertices of S into equation of S'

*S is completely behind ("inside") the overlapping S'*
ordering correct if
- S' completely in front of S
- substitute vertices of S' into equation of S

overlapping S' is completely in front ("outside") of S, but S is not completely behind S'
ordering correct if

- projections of $S$, $S'$ in $xy$-plane don’t overlap

surfaces with overlapping bounding rectangles
all five tests fail ⇒
- ordering probably wrong
- interchange surfaces S, S'
- repeat process for reordered surfaces

surface S has greater depth but obscures S'

sorted surface list: S, S', S" should be reordered: S', S", S
Depth-Sorting: Special Cases

- avoiding infinite loops due to cyclic overlap
  - reordered surfaces $S'$ are flagged
  - if $S'$ would have to be reordered again $\Rightarrow$ divide $S'$ into two parts

- intersecting or cyclically overlapping surfaces!
Area-Subdivision Method (1)

- image-space method
- area coherence exploited
- viewing area subdivided until visibility decision very easy
relationship polygon ⇔ rectangular view area

- outside surface
- inside surface
- overlapping surface
- surrounding surface

only these four possibilities
three easy visibility decisions

- all surfaces are outside of viewing area
- only one inside, overlapping, or surrounding surface is in the area
- one surrounding surface obscures all other surfaces within the viewing area
- a surrounding obscuring surface
- surfaces ordered according to minimum depth
- maximum depth of surrounding surface closest to view plane?
- test is conservative
Area-Subdivision Method (5)

- if all three tests fail ⇒ do *subdivision*
  - subdivide area into four equal subareas
  - outside and surrounding surfaces will remain in this status for all subareas
  - some inside and overlapping surfaces will be eliminated
- if no further subdivision possible (pixel resolution reached)
  - sort surfaces and take intensity of nearest surface
Area-Subdivision Method Example
Octree Methods

- recursive traversal of octree
  - traversal order depends on processing direction

- front-to-back:
  - pixel(x,y) written once
  - completely obscured nodes are not traversed

- back-to-front:
  - painter’s algorithm
Octree Methods

- recursive traversal of octree
  - traversal order depends on processing direction
- front-to-back:
  - pixel(x,y) written once
  - completely obscured nodes are not traversed
- back-to-front:
  - painter’s algorithm
Ray-Casting Method (1)

- line-of-sight of each pixel is intersected with all surfaces
- take closest intersected surface
Based on geometric optics, tracing paths of light rays
- Backward tracing of light rays
- Suitable for complex, curved surfaces
- Special case of ray-tracing algorithms
- Efficient ray-surface intersection techniques necessary
  - Intersection point
  - Normal vector