Hidden Surface Algorithms

Clipping

Image space algorithms

- Z-buffering
- Painter's algorithm

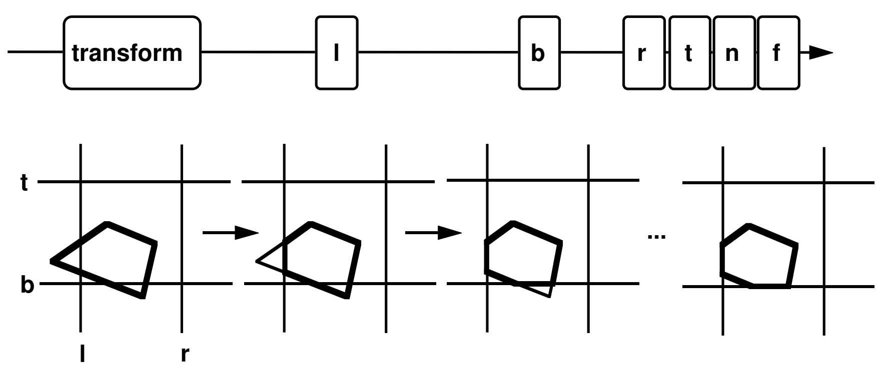
Object space algorithms

- BSP trees
- Robert's algorithm
- Portals

Quake game engine

Clipping Pipeline

Clip polygon to 6 planes of the frustum in a pipeline

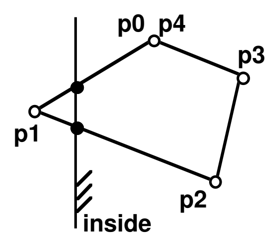


Copyright © Pat Hanrahan

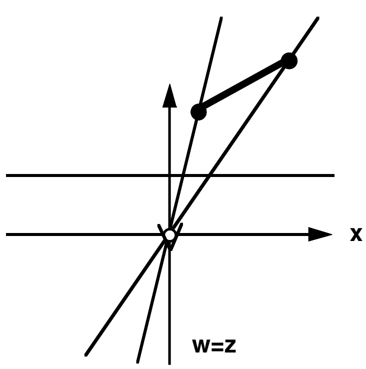
Clipping Stage

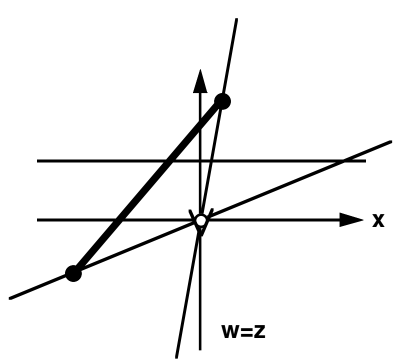
```
moveto()
  prevpt = input();
  prevside = sign(pl o pt);
  if( inside(prevside) )
    output(prevpt);
```

```
drawto()
  pt = input(); side = sign(pl o pt);
  if( side != prevside ) {
    ipt = intersect(prevpt,pt,pl);
    output(ipt);
  }
  if( inside(pt) ) output(pt)
  prevpt = pt; prevside = side;
```



Test & Clip *Before* Divide by w





Visibility Analysis

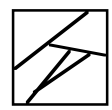
Hidden/Visible Surface/Line Elimination/Determination

Requirements

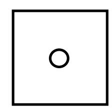
- **Handle diverse set of geometric primitives**
- **Handle large number of geometric primitives O(N)**

Classification: Sutherland, Sproull, Schumacher (1974):

- **Object Space**
  - **Geometric calculations involving polygons**
  - **Floating point precision: Exact**
  - **Often process scene in object order**



- **Image Space**
  - **Visibility at pixel samples**
  - **Integer precision**
  - **Often process scene in image order**



1. Z-buffering (Ed Catmull)

1. **Transform z at vertices**
2. **Modify scan conversion to interpolate z**
3. **Maintain depth or range image (16 or 24-bits common)**
4. **Depth comparison during drawing**

```
...
if( p.z < i[p.x][p.y].z ) {
  i[p.x][p.y].z = p.z;
  i[p.x][p.y].c = p.c;
}
p.z += p.dz;
p.c += p.dc;
...
```

Why So Popular?

Advantage

+ Simple to implement in hardware
    - Add additional z interpolator
    - Memory for z-buffer
+ Diversity of primitives
+ Unlimited complexity

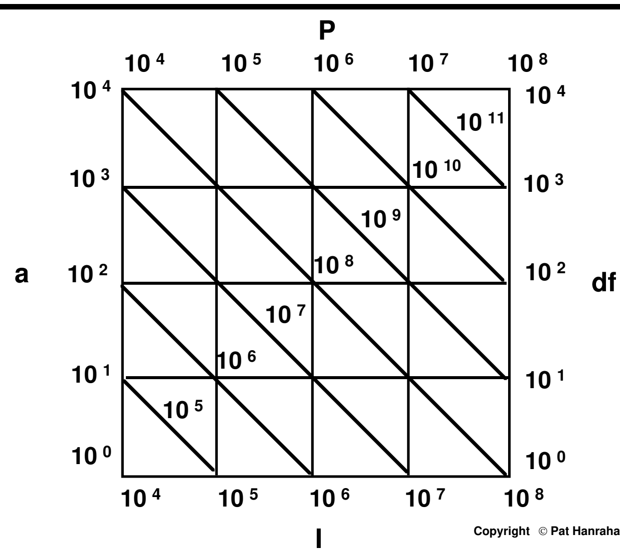
Disadvantage

- Extra memory and bandwidth
- Waste time drawing hidden objects
- Z-precision errors

? (Must use point sampling)

Depth Complexity

S = df I = a P



Rendering Tricks: Decals

Graphics packages

May draw directly in the z-buffer

May mask bits to be written to framebuffers

Decal

1. Draw base polygon using z-buffer compare,
   but, *don't update z-buffer*
2. Draw decal polygon using z-buffer compare,
   but, *don't update z-buffer*
3. Draw base polygon using z-buffer compare,
   and, *update z-buffer*, but, don't *update color buffer*

2. Painter's Algorithm

1. Draw surfaces in back-to-front order

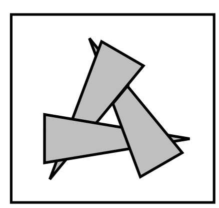
Front surfaces are drawn over back surfaces, hiding them

2. Draw surfaces in front-to-back order with under operator

Previously drawn surfaces mask drawing of those behind

Advantages:

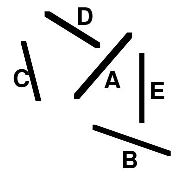
- Both supports transparency
- Front-to-back supports early termination



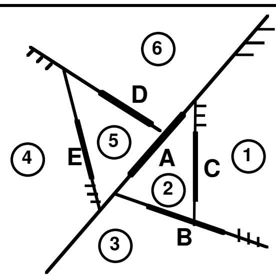
Key problem is priority determination

- Manually
- Exploit spatial properties of the model
- BSP (Binary Space Partition) Tree

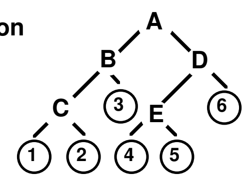
BSP Tree



Scene



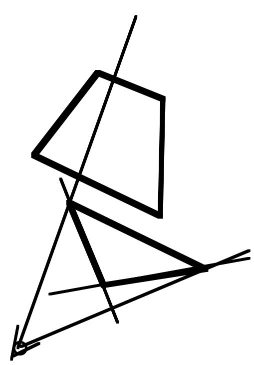
Binary
Space
Partition



3. Object-Space Algorithm

Robert's Algorithm
Input: Set of convex polyhedra
Output: Set of visible line segments

```
for each polyhedron
  form shadow volume
    discard back-facing faces
    keep front-facing faces
    create silhouette planes
for each edge
  for each shadow volume
    clip edge to shadow volume
  draw visible edge
```



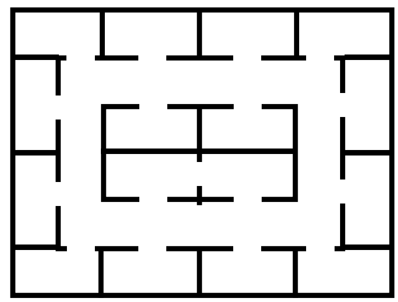
Portals

Model scene as a graph

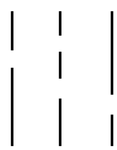
- Nodes: Rooms
- Edges: Portals

Potentially visible set

- Superset of visible polys
- Room to room visibiity



Stabbing line test



Quake Game Engine

Neat combination of all of the above!!

- Environment
  - Use portals to determine potentially visible set
  - Use bsp-tree to order polygons from front-to-back
  - Maintain front-to-back ordering of the spans on the active edge list
  - Maintain scanline visibility mask when drawing spans
  - Output only visible pixels (color and depths)
- Objects
  - Draw using the environment z-buffer