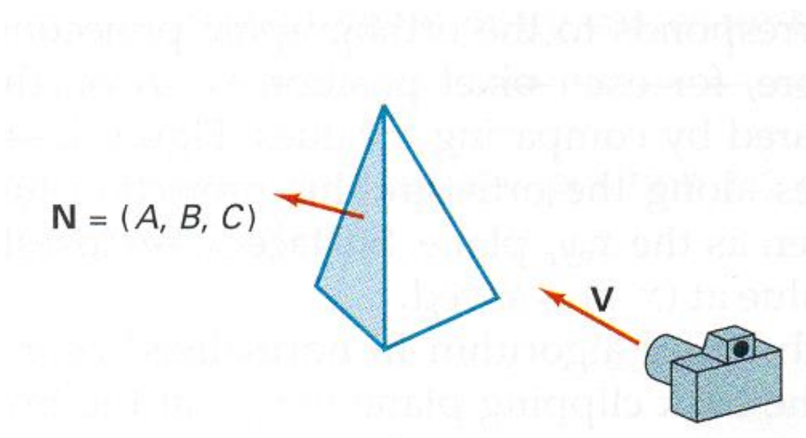
Universidade de Aveiro
Departamento de Electrónica,
Telecomunicações e Informática

Visible Surface Determination



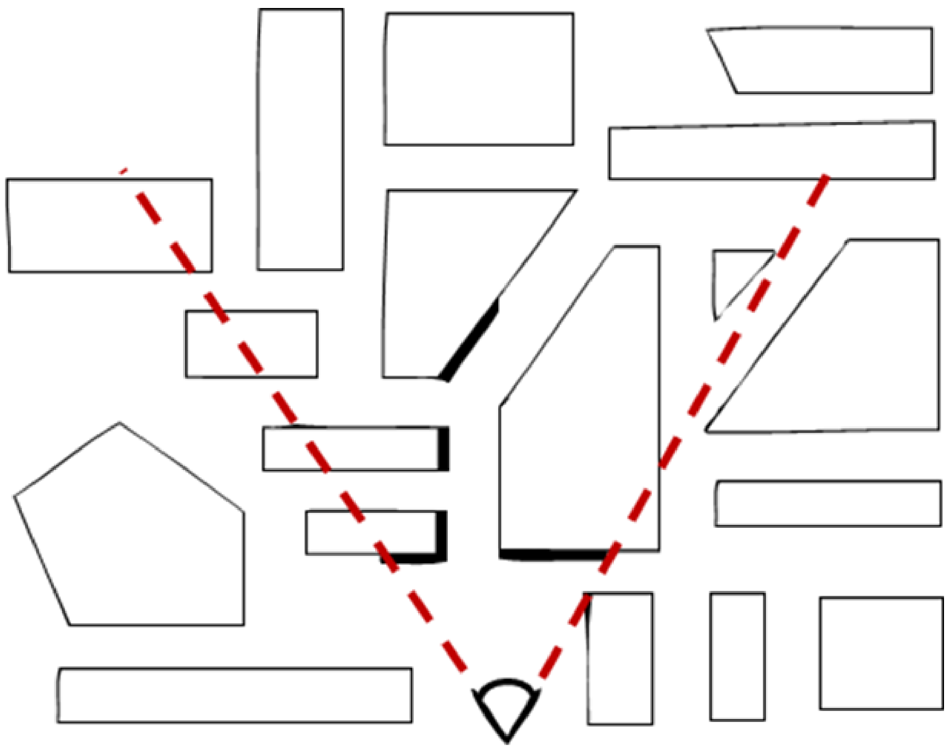
Beatriz Sousa Santos, J. Madeira

Overview

- Visible Surface Determination
- Back-Face Culling
- Z-Buffer
- Application Example
- Projections in OpenGL / WebGL

VISIBLE SURFACE DETERMINATION

Visible Surface Determination



For each object compute:

- The visible edges and surfaces

Why might objects be hidden ?

- Clipping ?
- Occlusion ?

To render or not to render, that is the question…

[Andy Van Dam]

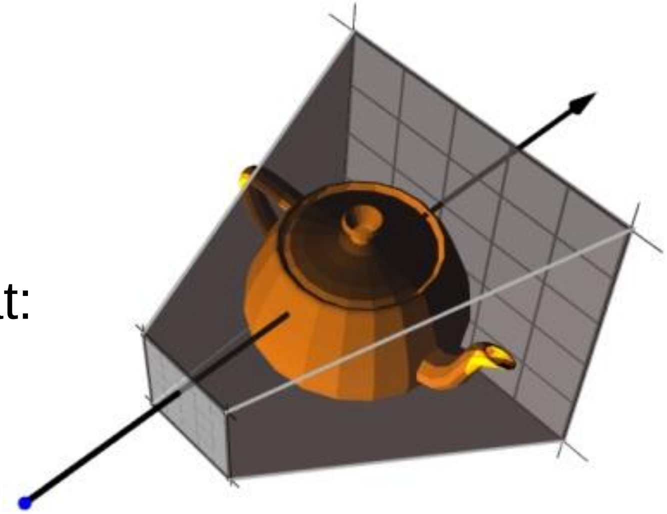
Clipping vs Occlusion

- Clipping against the view volume
  - It is done at object-level !
- Occlusion / Hidden-Surface Removal
  - It is done at scene-level !
  - Compare depth of object / edges / pixels against other objects / edges / pixels

Possible approaches

- Object-precision algorithms
  - Analyze / compare objects or parts of objects to determine which surfaces / faces / edges are fully or partially visible
  - Back-Face Culling
- Image-precision algorithms
  - Determine visibility for every pixel in the viewing plane
  - Work in 3D to get / compare depth values (i.e., z values)
  - Z-buffer, A-buffer, Ray-Casting
- Hybrid – use both approaches
- Improve performance using
  - Coherence
  - Sorting

- The visibility problem
  - Which primitives – after modeling transformations, projection and lighting calculations – contribute for each image pixel



- In general, we solve the dual problem !!

- Which are the hidden surfaces / faces that:

- are outside of the view volume ?

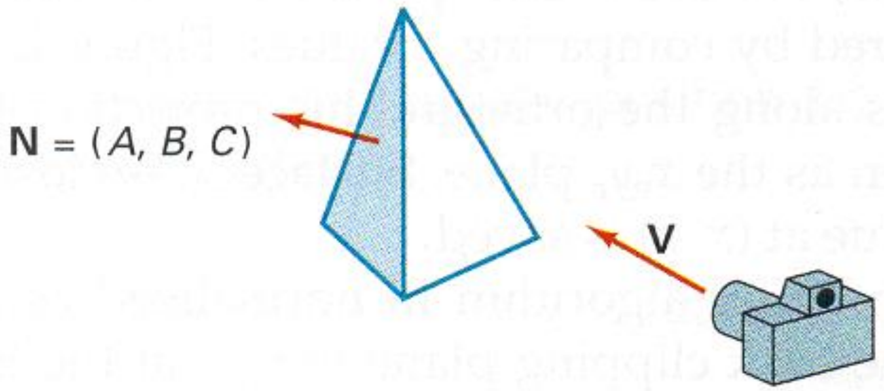
- are back-faces in a closed and convex polyhedron ?

- are hidden by other faces closer to the viewpoint / camera ?

- Determine which models or parts of models are visible from the chosen viewpoint and projection type

- There are various methods with different:

  - approaches
  - running time
  - required memory
  - ….



- More or less adequate given:

  - scene complexity
  - available hardware
  - ….

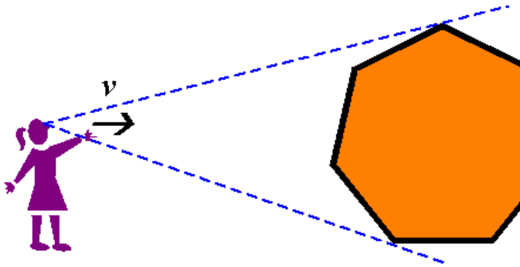
Some methods

- Back-face culling
- Depth-buffer (z-buffer)
- A-buffer
- Depth-sorting (painter's algorithm)
- Ray-casting

BACK-FACE CULLING

Back-Face Culling

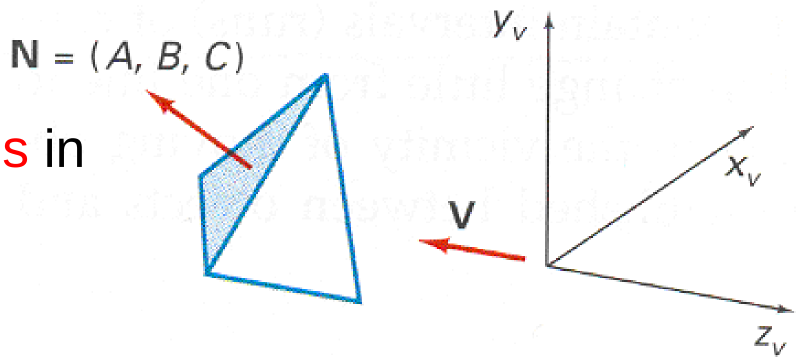
- Sufficient for a single convex polyhedron which is not sectioned by clipping



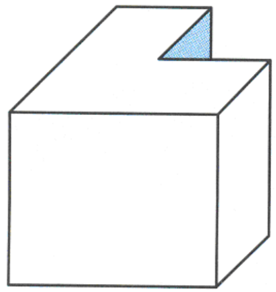
- Not sufficient for

  - concave polyhedra

  - when there are two or more models in front of each other



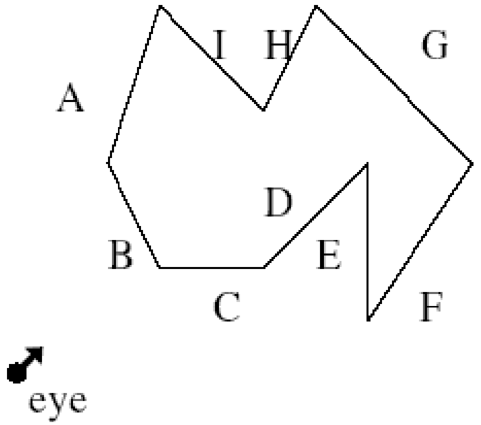
When looking at the negative ZZ' semi-axis, a polygon is a back-face if C ≤ 0

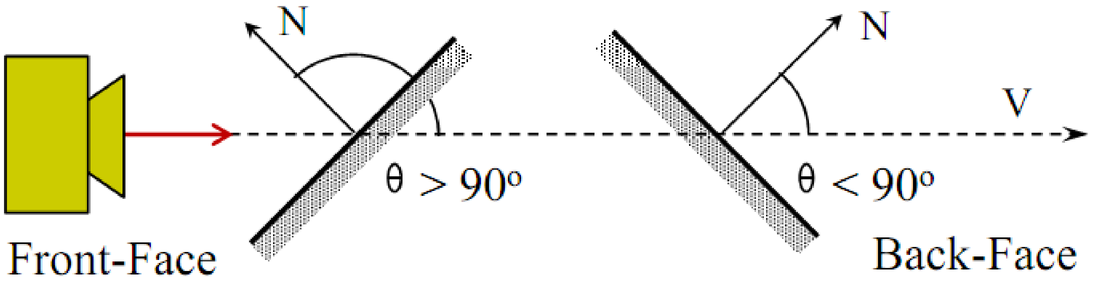


Concave model with a partially visible face

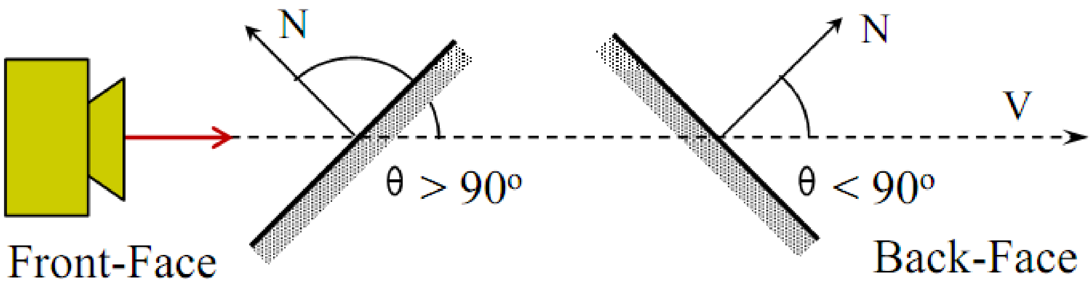
Back-Face Culling

- What to do in a general case ?
- For each face, compute the angle between
  - The normal vector to the face
  - The viewing direction, defined by the viewpoint





Back-Face Culling



- Efficiency: just compute the scalar product !!
  - Reject a face if $N \cdot V > 0$
  - What happens if $N \cdot V = 0$?
- Simplification when $V = (0, 0, -1)$!!
- On average, approx. half of the faces are removed!

OpenGL – Culling

- Back-Face Culling

```
glEnable( GL_CULL_FACE );

glCullFace( GL_BACK );
```

- Front-Face Culling

```
glEnable( GL_CULL_FACE );

glCullFace( GL_FRONT );
```

WebGL – Culling

- Back-Face Culling

```
gl.enable( gl.CULL_FACE ); // Default: disabled!

gl.cullFace( gl.BACK );    // Default
```

- Front-Face Culling

```
gl.enable( gl.CULL_FACE );

gl.cullFace( gl.FRONT );
```

WebGL – Culling

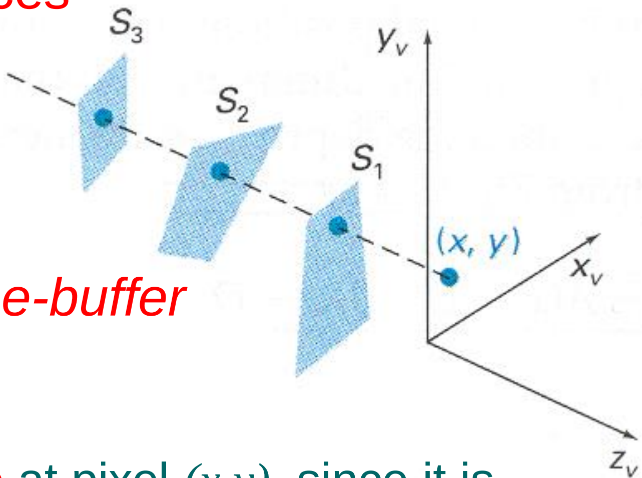
- Switching on / off

```
if( gl.isEnabled( gl.CULL_FACE ) )
{
   gl.disable( gl.CULL_FACE );
}
else
{
   gl.enable( gl.CULL_FACE );
}
```

Z-BUFFER

Depth-Buffer (z-buffer)

- Works in *image-space*
- Compares the depth of each surface relative to each pixel in the viewplane
- Fast and easy to implement for planar surfaces
- Can be adapted for curved surfaces
- Needs a *depth-buffer* in addition to the *frame-buffer*



S1 is visible at pixel (x,y), since it is closer to the viewplane

Z-Buffer Algorithm

- Draw every polygon that can’t be rejected trivially
  - I.e., it is totally outside the view volume
- If a piece (one or more pixels) of a polygon that is closer to the front is found
- Paint over whatever was behind it
- Use plane equation for polygon, z = f(x, y)
- Note: use positive z here [0, 1]

Z-Buffer Algorithm

```
void zBuffer() {
   int x, y;
   for (y = 0; y < YMAX; y++)
   for (x = 0; x < XMAX; x++) {
       WritePixel (x, y, BACKGROUND_VALUE);
       WriteZ (x, y, 1);
   }
   for each polygon {
     for each pixel in polygon's projection {
       //plane equation
       double pz = Z-value at pixel (x, y);
       if (pz <= ReadZ (x, y)) {
         // New point is closer to front of view
         WritePixel (x, y, color at pixel (x, y))
         WriteZ (x, y, pz);
       }
     }
   }
}
```

[Andy Van Dam]

Example

Initial *z-buffer* values

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Depth-values for polygon

5	5	5	5	5	5	5
5	5	5	5	5	5	
5	5	5	5	5		
5	5	5	5			
5	5	5				
5	5					
5						

Overwrite *z-buffer*

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
5	5	5	0	0	0	0	0
5	5	0	0	0	0	0	0
5	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Another polygon

3					
4	3				
5	4	3			
6	5	4	3		
7	6	5	4	3	
8	7	6	5	4	3

Final *z-buffer*

5	5	5	5	5	5	5	0
5	5	5	5	5	5	0	0
5	5	5	5	5	0	0	0
5	5	5	5	0	0	0	0
6	5	5	3	0	0	0	0
7	6	5	4	3	0	0	0
8	7	6	5	4	3	0	0
0	0	0	0	0	0	0	0

Note:
0 – background depth
Max – viewplane

Another example

255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

+

64	64	64	64	64	64	64
64	64	64	64	64	64	
64	64	64	64	64		
64	64	64	64			
64	64	64				
64	64					
64						

=

64	64	64	64	64	64	64	255
64	64	64	64	64	64	255	255
64	64	64	64	64	255	255	255
64	64	64	64	255	255	255	255
64	64	64	255	255	255	255	255
64	64	255	255	255	255	255	255
64	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

64	64	64	64	64	64	64	255
64	64	64	64	64	64	255	255
64	64	64	64	64	255	255	255
64	64	64	64	255	255	255	255
64	64	64	255	255	255	255	255
64	64	255	255	255	255	255	255
64	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255

+

127				
127	127			
127	127	127		
127	127	127	127	
127	127	127	127	127

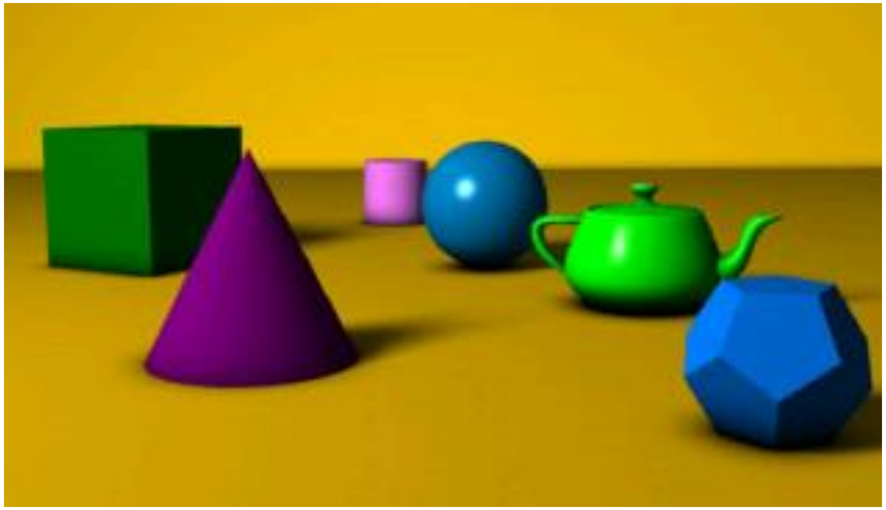
=

64	64	64	64	64	64	64	255
64	64	64	64	64	64	255	255
64	64	64	64	64	255	255	255
64	64	64	64	255	255	255	255
64	64	64	255	255	255	255	255
64	64	127	255	255	255	255	255
64	127	127	127	255	255	255	255
127	127	127	127	127	255	255	255

integer Z-buffer with
near = 0, far = 255

[Andy Van Dam]

3D scene



Z-buffer

1. Initialize the depth buffer and refresh buffer so that for all buffer positions (x, y),

$$\text{depth}(x, y) = 0, \qquad \text{refresh}(x, y) = I_{\text{backgnd}}$$

2. For each position on each polygon surface, compare depth values to previously stored values in the depth buffer to determine visibility.

   - Calculate the depth z for each (x, y) position on the polygon.
   - If $z > \text{depth}(x, y)$, then set

$$\text{depth}(x, y) = z, \qquad \text{refresh}(x, y) = I_{\text{surf}}(x,y)$$

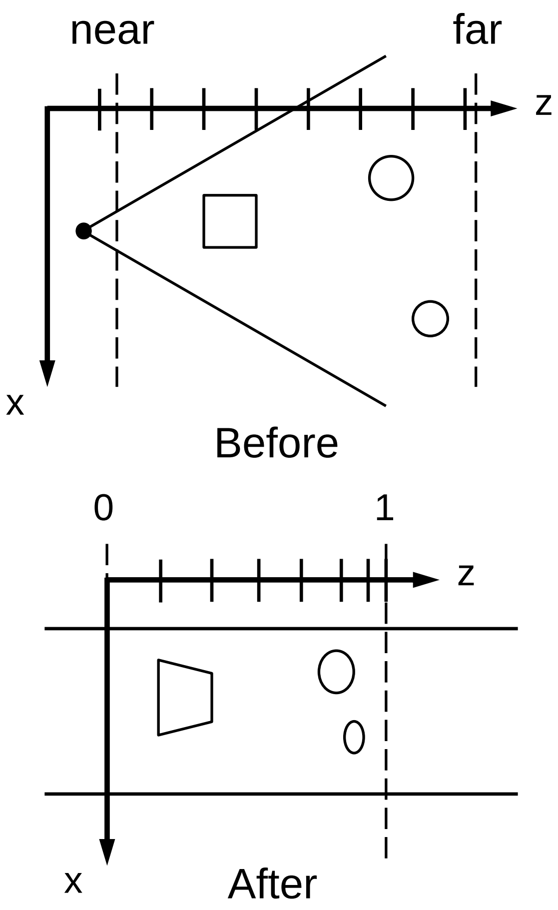
where I_{backgnd} is the value for the background intensity, and $I_{\text{surf}}(x,y)$ is the projected intensity value for the surface at pixel position (x,y). After all surfaces have been processed, the depth buffer contains depth values for the visible surfaces and the refresh buffer contains the corresponding intensity values for those surfaces.

Some authors present different versions of the algorithm, using other depth values for the viewplane and the background and/or normalized coordinates

- Advantages:
  - Easy, no need to previously sort the various surfaces
  - Fast
- Disadvantages:
  - Need for additional memory
  - Depth precision problems / limitations
  - It finds one visible surface for each pixel
  - I.e. it can only handle opaque surfaces

Depth precision loss

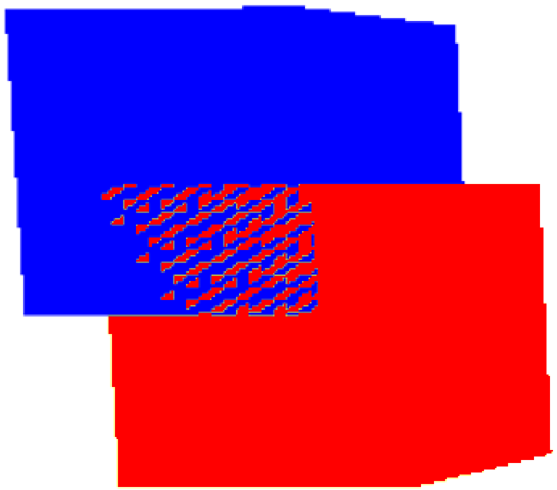
- Perspective foreshortening
  - Compression in z-axis in post-perspective space
  - Objects far away from camera have z-values very close to each other
- Depth information loses precision rapidly
  - Leads to z-ordering bugs called z-fighting



[Andy Van Dam]

Z-Fighting

- Z-fighting occurs when two primitives have similar values in the z-buffer
  - Coplanar polygons (two polygons that occupy the same space)
  - One is arbitrarily chosen over the other, but z varies across the polygons and binning will cause artifacts
  - Behavior is deterministic: the same camera position gives the same z-fighting pattern



Two intersecting cubes

[Andy Van Dam]

Edwin Catmull, the z-buffer inventor,

former president of Walt Disney and

Pixar Animation Studios, received an

Oscar in 2001

Pixar's RenderMan

http://renderman.pixar.com/

OpenGL – Depth-Test

- Enabling

```
glEnable( GL_DEPTH_TEST );
```

- Buffers

```
glutInitDisplayMode( GLUT_RGB |
                     GLUT_DOUBLE | GLUT_DEPTH );
```

- Clearing the buffers before rendering

```
glClear( GL_COLOR_BUFFER_BIT |
         GL_DEPTH_BUFFER_BIT );
```

WebGL – Depth-Test

- Enabling

```
gl.enable( gl.DEPTH_TEST ); // Default: disabled!
```

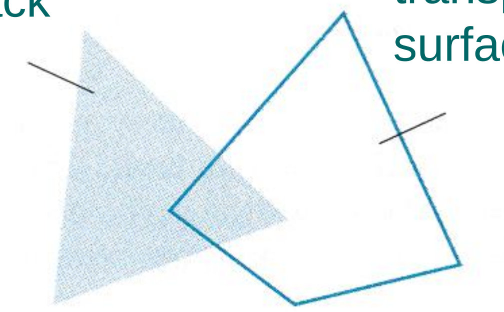
- Clearing the buffers before each rendering

```
gl.clear( gl.COLOR_BUFFER_BIT |
          gl.DEPTH_BUFFER_BIT );
```

A-BUFFER

A-Buffer

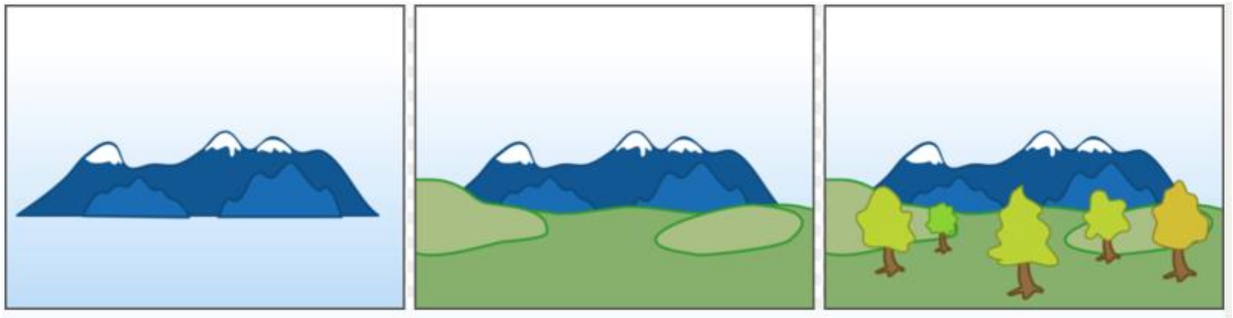
- Anti-aliased area-average, accumulation buffer
- An extension of the *depth-buffer*, allows to take into account more than one surface, i.e., handling transparent surfaces
- It also allows for *lesser aliasing* of model edges



To see an opaque surface through a transparent surface requires the accumulation of the contributions from both surfaces

THE PAINTER’S ALGORITHM

Depth-Sorting
(Painter's Algorithm)

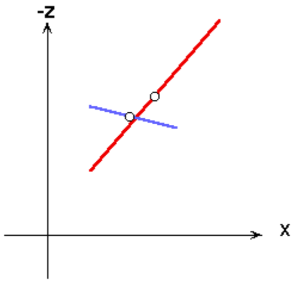


- *Image-space* and *object-space* operations

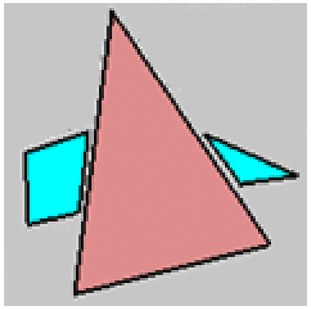
- Main steps:

  1- Sort the surfaces by their depth relative to the viewplane

  2- Back-to-front projection of the sorted surfaces



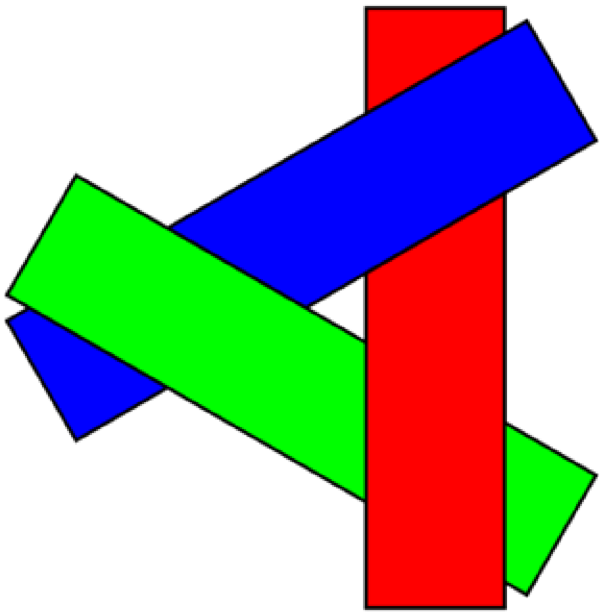
- What if superpositions occur ?
  - Subdivide the surfaces or change their order



- No need for a depth-buffer !!

- BUT, it renders distant surface parts that, in the end, will not be visible

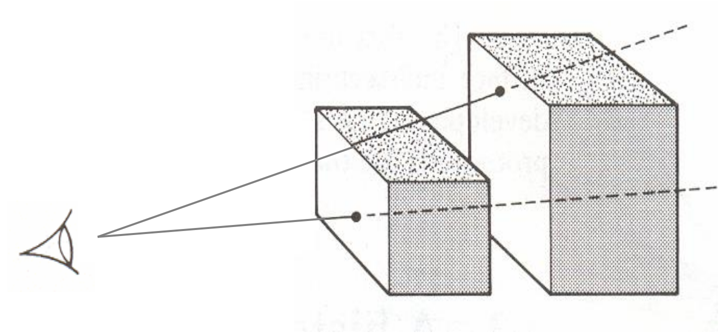
Painter's algorithm – What happens ?



[Andy Van Dam]

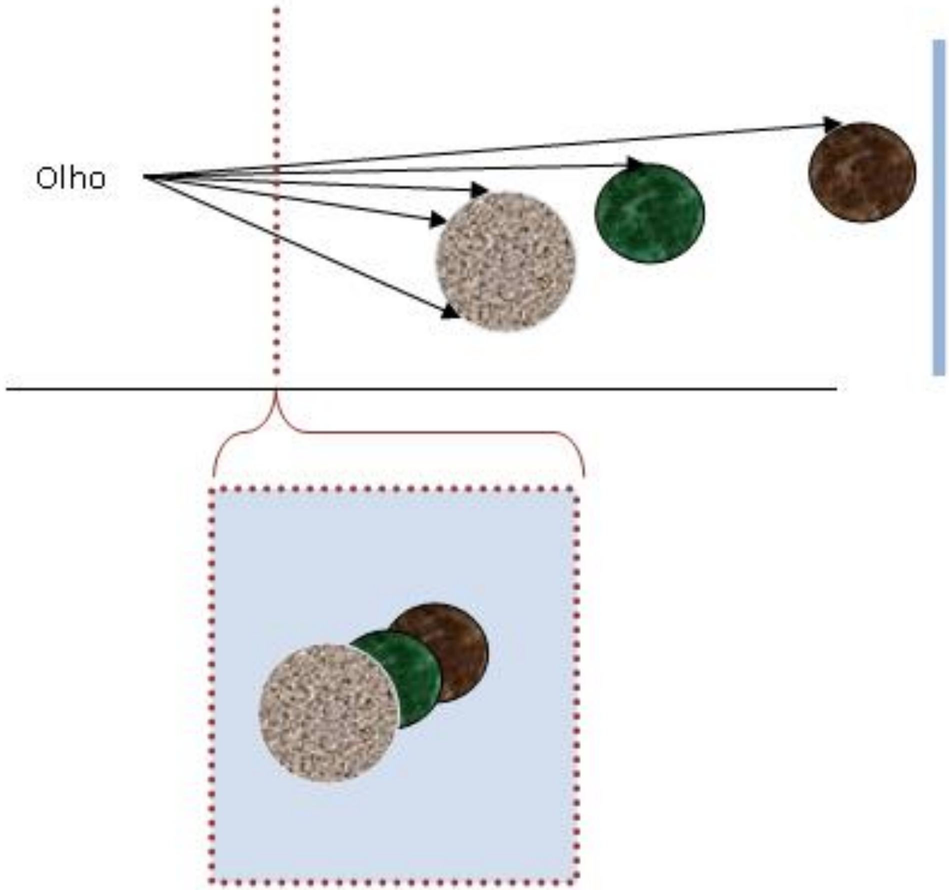
RAY-CASTING

Ray-Casting



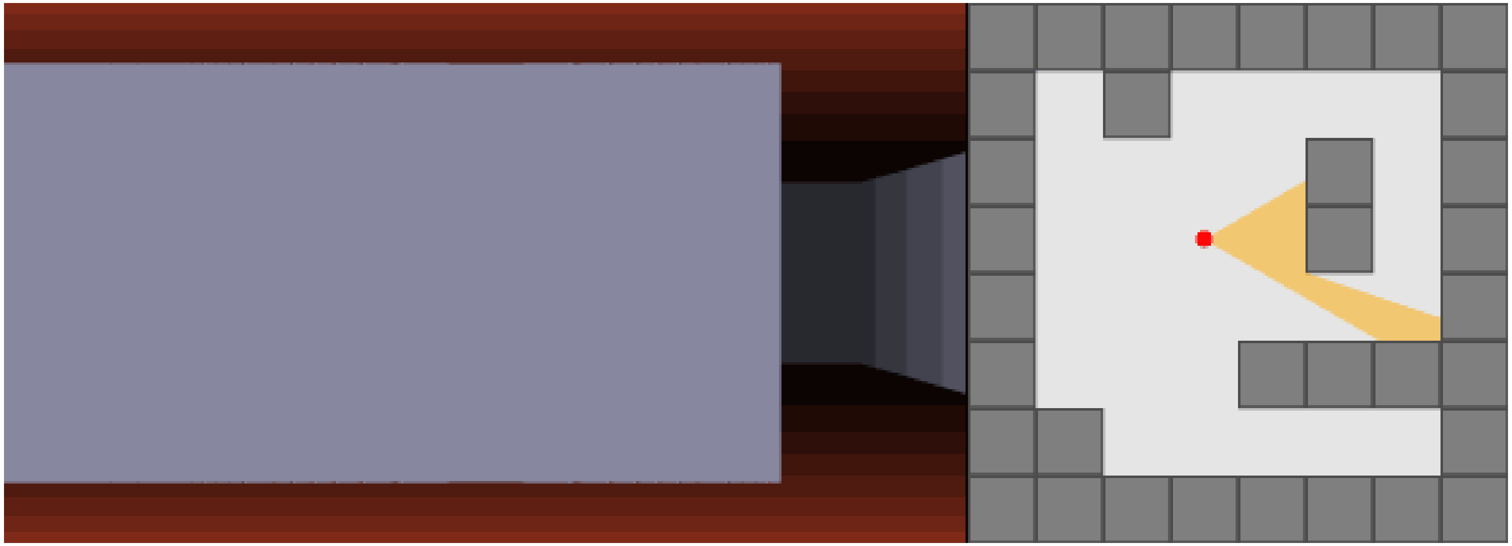
- Image-space method
- Which models are intersected by a ray cast from the viewpoint that passes through the center of a pixel ?
- If models are opaque, the intersection point closer to the viewpoint determines the color (i.e., visible surface) for the corresponding pixel

Ray-Casting



[Wikipedia]

Ray-Casting



[Wikipedia]

REFERENCES

References

- Hearn, D., P. Baker, *Computer Graphics with OpenGL*, Addison Wesley, 2004
- Hearn, D., P. Backer, *Computer Graphics*, 2nd. Ed., Prentice Hall, 1994
- Foley, J., S. Van Dam, S. Feiner, J. Hughes, *Computer Graphics, Principles and Applications*, 2nd. Ed., Addison Wesley, 1991
- Watt, A., F. Policarpo, *The Computer Image*, Addison Wesley, 1998