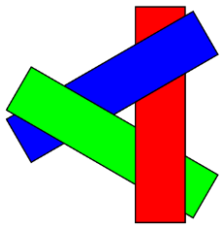
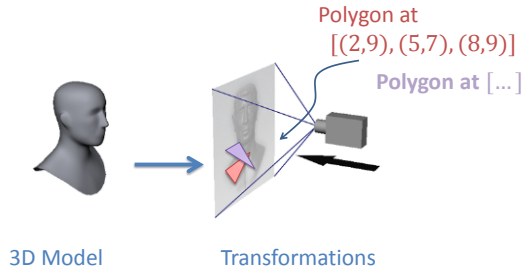
Hidden Surface Removal

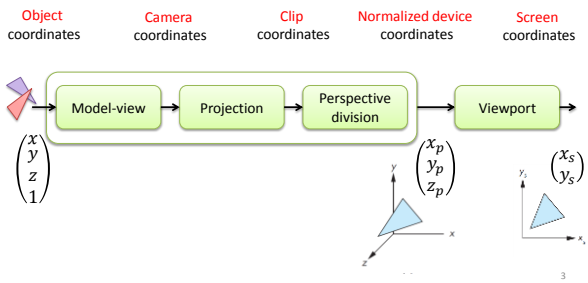


Reminder - Pipeline



2

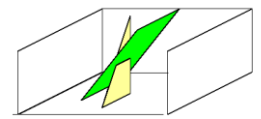
Reminder - Pipeline



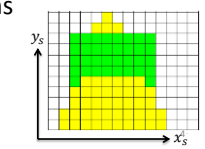
3

Hidden Surface Removal

- ***Input***: Polygons in normalized device coordinates

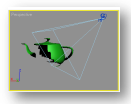


- ***Output***: 2D image of projected polygons containing only visible portions

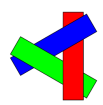


Hidden Surface Removal Algorithms

- Back face culling

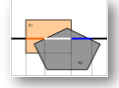


- Painter's algorithm (depth sort)



- Z-Buffer

- Scan-line Z-Buffer

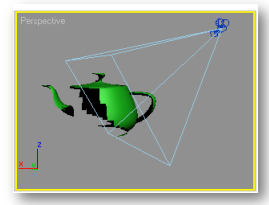


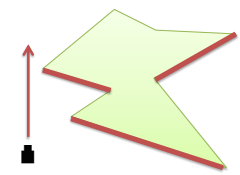
5

Back Face Culling

Object Space

In a closed polyhedron back faces are not visible



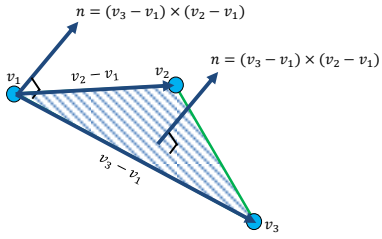


torus demo

6

JACOBS TECHNION-CORNELL INSTITUTE AT CORNELL TECH

The Normal Vector



$n(1,2,3) = n(2,3,1) = -n(2,1,3)$

7

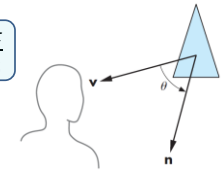
Back Face Culling
Determining back faces

- In a closed polyhedron back faces are not visible
- Assume normals point *out*

face visible iff: $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$

iff $cos\theta \geq 0$

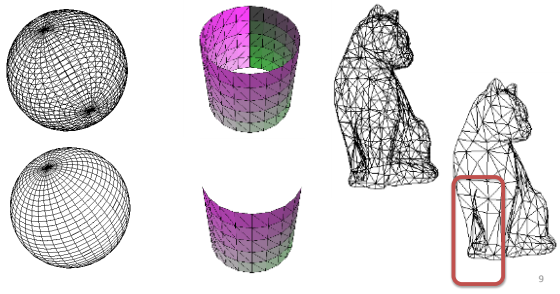
iff $v \cdot n = |v||n|cos\theta \geq 0$



8

Back Face culling
When will it work?

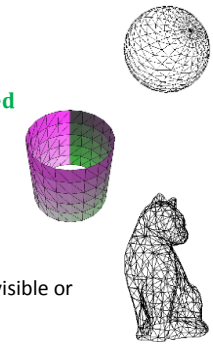
Closed, convex | Open | Closed, non convex



9

Back Face Culling

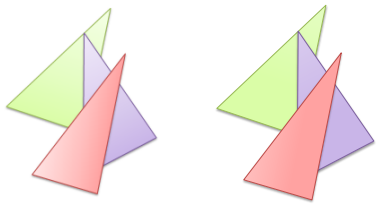
- Closed convex objects
  - All back faces invisible
  - All front faces visible
  - → **Visibility problem solved**
- Open objects
  - Back faces possibly visible
- Closed non-convex objects
  - Invisible back faces
  - Front faces can be visible, invisible or partially visible



10

Painter's Algorithm
Object space

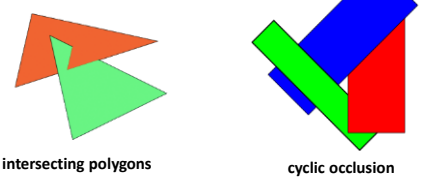
- Which **color** to draw?
- Draw everything → which **order** to draw?



11

Painter's Algorithm

- Sort polygons by depth values
- Paint back to front
- When will this fail?

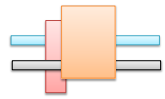


intersecting polygons | cyclic occlusion

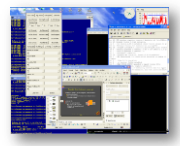
12

Painter's Algorithm

- Works in special cases
  - E.g. polygons with constant z



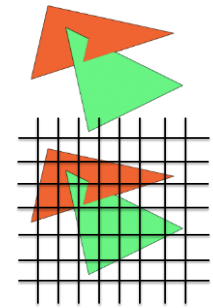
  - Where do we have polygons with constant z?



13

Painter's Algorithm
How do we fix it?

- Depth sort per **polygon** doesn't work



- Depth sort per ***pixel***
  - Image space algorithm

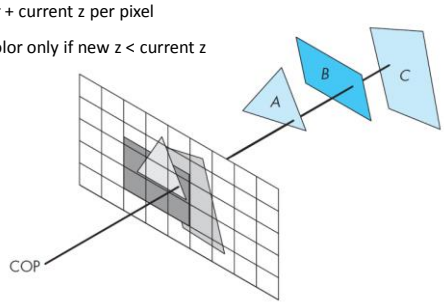
14

Z-Buffer Algorithm
Image Space

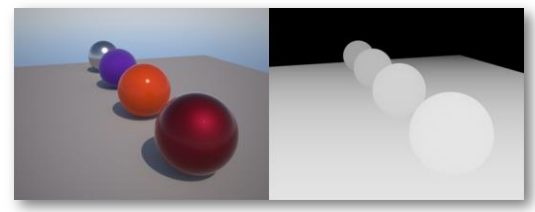
Resolve visibility at the pixel level

Store color + current z per pixel

Put new color only if new z < current z



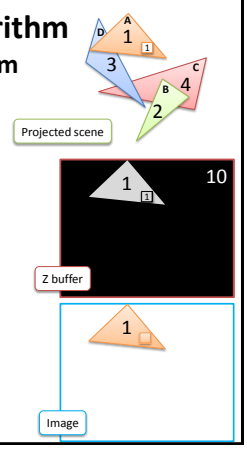
Z-Buffer Algorithm
The Z-Buffer



16

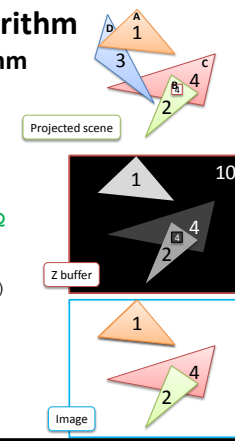
Z-Buffer Algorithm
The Algorithm

```
For every pixel (x,y)
      putZ(x,y,MaxZ)

For each polygon P
   Q = project(P)
   for each pixel (x,y) in Q
      z = depth(Q,x,y)
      if z < getZ(x,y)
         putColor(x,y,col(P))
         putZ(x,y,z)
      end
   end
end
```



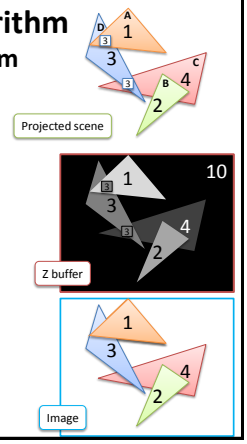
Z-Buffer Algorithm
The Algorithm

```
For every pixel (x,y)
      putZ(x,y,MaxZ)

For each polygon P
   Q = project(P)
   for each pixel (x,y) in Q
      z = depth(Q,x,y)
      if z < getZ(x,y)
         putColor(x,y,col(P))
         putZ(x,y,z)
      end
   end
end
```



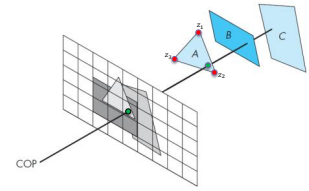
Z-Buffer Algorithm
The Algorithm

```
For every pixel (x,y)
     putZ(x,y,MaxZ)

For each polygon P
   Q = project(P)
   for each pixel (x,y) in Q
      z = depth(Q,x,y)
      if z < getZ(x,y)
         putColor(x,y,col(P))
         putZ(x,y,z)
      end
   end
end
```



Computing depth(Q,x,y)

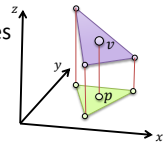
- Have z coordinate at **3 vertices**
- How do we compute z at **pre-image** of **projected point**?



20

Computing depth(Q,x,y)

- We know
  - 3D coordinates at vertices
  - 2D coordinates at projected vertices
  - 2D coordinates at $p = (x, y)$
- We need
  - 3D coordinates at v

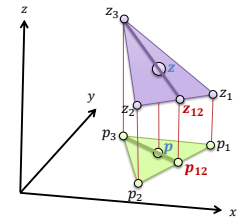


21

Computing depth(Q,x,y)

- Linear transformations preserve straight lines

1. Compute z_{12} and p_{12}
2. Express p with p_3 and p_{12}
3. Compute z in the same way from z_3 and z_{12}



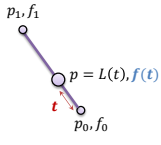
22

Linear Interpolation
On a line

- Input: 2 points, 2 values
- Output: **value** at any point p on the line $L(t)$ between them

$$L(t) = tp_1 + (1-t)p_0 \quad t \in [0,1]$$

$$L(0) = p_0$$
$$L(1) = p_1$$
$$L(0.5) = \frac{p_0 + p_1}{2}$$



$$p_1, f_1, p_0, f_0, p$$

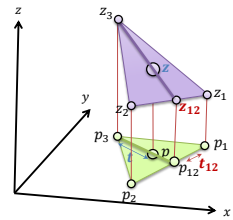
$$t = \frac{||p-p_0||}{||p_1-p_0||} = \frac{|y-y_0|}{|y_1-y_0|} = \frac{|x-x_0|}{|x_1-x_0|}$$

$$f(p) = f(t) = tf_1 + (1-t)f_0$$

23

Computing depth(Q,x,y)

- Apply linear interpolation on a line twice:

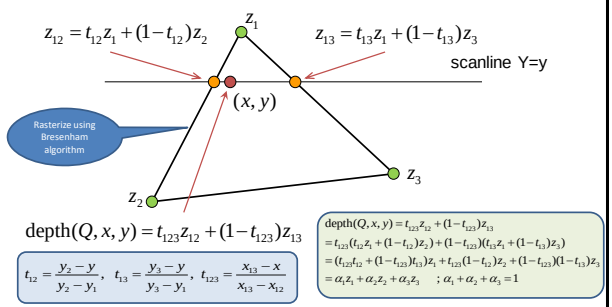


$$z_{12} = t_{12}z_2 + (1-t_{12})z_1$$
$$t_{12} = \frac{||p_{12}-p_1||}{||p_2-p_1||}$$

$$z = tz_{12} + (1-t)z_3$$
$$t = \frac{||p-p_3||}{||p_{12}-p_3||}$$

24

Z-Buffer – depth(Q,x,y)



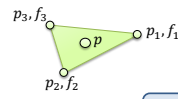
$$\text{depth}(Q,x,y) = t_{123}z_{12} + (1-t_{123})z_{13}$$

$$t_{12} = \frac{y_2 - y}{y_2 - y_1},\quad t_{13} = \frac{y_3 - y}{y_3 - y_1},\quad t_{123} = \frac{x_{13} - x}{x_{13} - x_{12}}$$

$$\begin{aligned}\text{depth}(Q,x,y) &= t_{123}z_{12} + (1-t_{123})z_{13} \\ &= t_{123}(t_{12}z_1 + (1-t_{12})z_2) + (1-t_{123})(t_{13}z_1 + (1-t_{13})z_3) \\ &= (t_{123}t_{12} + (1-t_{123})t_{13})z_1 + t_{123}(1-t_{12})z_2 + (1-t_{123})(1-t_{13})z_3 \\ &= \alpha_1 z_1 + \alpha_2 z_2 + \alpha_3 z_3 \quad ;\ \alpha_1 + \alpha_2 + \alpha_3 = 1\end{aligned}$$

Linear Interpolation
On a triangle

- Input: 3 points, 3 values
- Output: **value** at any point p in the triangle the points span



$$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$$
$$\alpha_1 + \alpha_2 + \alpha_3 = 1$$
$$\alpha_i \geq 0$$

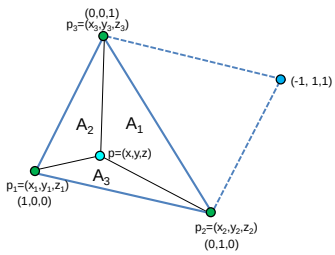
$$p_1, f_1, p_2, f_2, p_3, f_3, p$$

$$f(p) = f(\alpha_1, \alpha_2, \alpha_3) = \alpha_1 f_1 + \alpha_2 f_2 + \alpha_3 f_3$$

But α_i =?

Barycentric Coordinates

$$\alpha_i = \frac{A_i}{A_1 + A_2 + A_3}$$

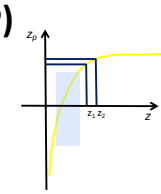


Barycentric coordinates of p = (α_1, α_2, α_3)
B.C. are **unique**.
B.C. of all interior points are ≥ 0.
Triangle centroid = (1/3,1/3,1/3).

Z-Buffer - Project(P)

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{d}{d-\alpha} & \frac{-\alpha d}{d-\alpha} \\ 0 & 0 & 1/d & 0 \end{bmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ \frac{d(z-\alpha)}{d-\alpha} \\ z/d \end{pmatrix}$$

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} = \begin{pmatrix} \frac{x}{z/d} \\ \frac{y}{z/d} \\ \frac{d^2(z-\alpha)}{z(d-\alpha)} \end{pmatrix}$$

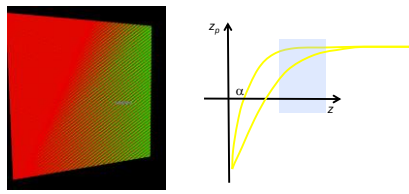


z_p monotone with respect to z – use as depth to determine order

Z-Buffer Algorithm

- Image space algorithm
- Data structure: Array of depth values
- Implemented in hardware due to simplicity
- Depth resolution of 32 bits is common
- ***Scene may be updated on the fly, adding new polygons***

Z Fighting

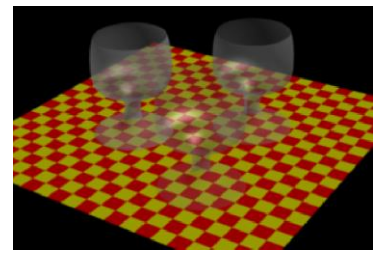


When Z-buffer has low precision and/or
α is not chosen correctly

31

Transparency Z-Buffer

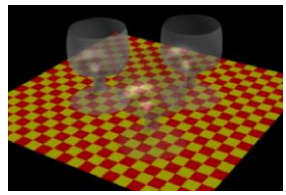
How can we emulate transparent objects?



32

Transparency Z-Buffer

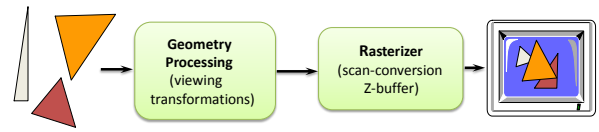
- Extension to the basic Z-buffer algorithm
- Save *all* pixel values
- At the end – have list of polygons & depths (order) for each pixel
- Simulate transparency by weighting the different list elements, in order
- Do we really need to store *all* pixel values?



33

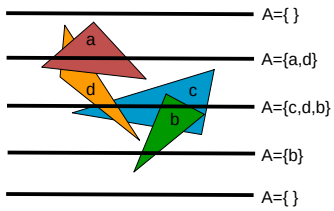
The Graphics Pipeline

- Hardware implementation of screen Z-buffer:
  - Polygons sent through pipeline one at a time
  - Display updated to reflect each new polygon



34

Scan-Line Z-Buffer Algorithm



35

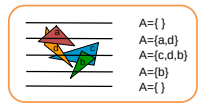
Scan-Line Z-Buffer Algorithm

- In software implementations - amount of memory required for screen Z-buffer may be prohibitive
- Scan-line Z-buffer algorithm:
  - Render the image one line at a time
  - Take into account only polygons affecting this line
- Combination of polygon scan-conversion & Z-buffer algorithms
- Only Z-buffer the size of scan-line is required.
- Entire scene must be available in advance
- Image cannot be updated incrementally

36

Scan-Line Z-Buffer Algorithm

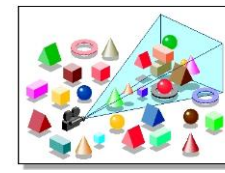
```
ScanLineZBuffer(Scene)
Scene2D := Project(Scene);
Sort Scene2D into buckets of polygons P in  increasing YMin(P) order;
A := EmptySet;
for y := YMin(Scene2D) to YMax(Scene2D) do
    for each pixel (x, y) in scanline Y=y do  PutZ(x, MaxZ);
    A := A + {P in Scene : YMin(P)<=y};
    A := A - {P in A : YMax(P)<y};
    for each polygon P in A
        for each pixel (x, y) in P's span(s) on the scanline
            z := Depth(P, x, y);
            if (z<GetZ(x)) then
                PutColor(x, y, Col(P));
                PutZ(x, z);
            end;
        end;
    end;
end;
```

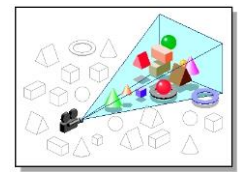


37

Line and Polygon Clipping Algorithms

- Cohen-Sutherland
- Sutherland-Hodgman
- Liang-Barksy
- Cyrus-Beck





38