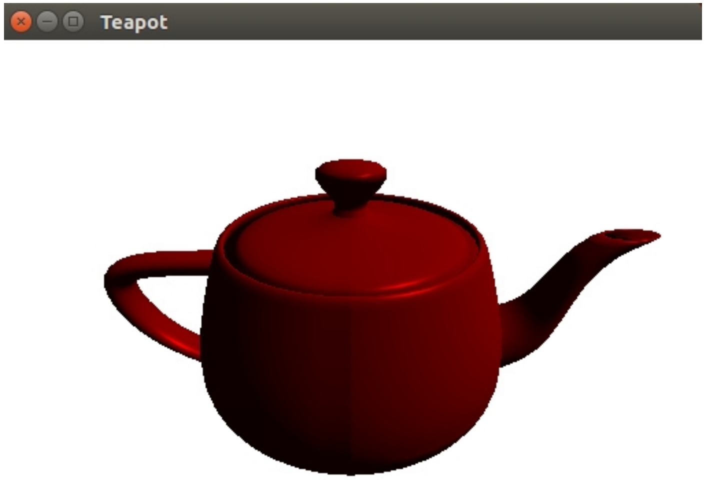
Announcement

Homework 3 has been posted.

Due Wednesday, Nov. 9

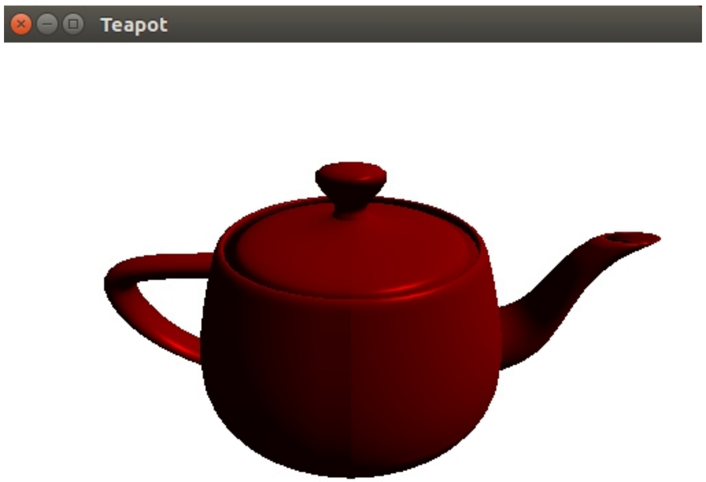
Project 2

```
vec4 light_position( 1.0, 1.0, 1.0, 0.0 );
vec4 light_ambient( 0.1, 0.1, 0.1, 1.0 );
vec4 light_diffuse( 1.0, 1.0, 1.0, 1.0 );
vec4 light_specular( 1.0, 1.0, 1.0, 1.0 );
vec4 material_ambient( 0.5, 0.0, 0.0, 1.0 );
vec4 material_diffuse( 0.5, 0.0, 0.0, 1.0 );
vec4 material_specular( 0.5, 0.0, 0.0, 1.0 );
float material_shininess = 100;
```



Project 2: Varying Light Position

vec4 light_position(1.0, 1.0, 1.0, 0.0);



vec4 light_position(-1.0, 1.0, 1.0, 0.0);

How to Choose Light Position

- **Ambient term is a constant**
- **Diffuse term** $\mathbf{I}_d = \mathbf{k}_d(\mathbf{l} \cdot \mathbf{n})\mathbf{L}_d$

Should be positive

- **Specular term** $\mathbf{I}_s = k_s L_s \max\left((\mathbf{n} \cdot \mathbf{h})^{\beta}, 0\right)$

Should be positive

Project 2: Varying Material Shininess

float material_shininess = 100;

float material_shininess = 10;

LookAt Function

```
mat4 mv = LookAt(vec4 eye, vec4 at, vec4 up);
```

Usually, “at” is the center of the object

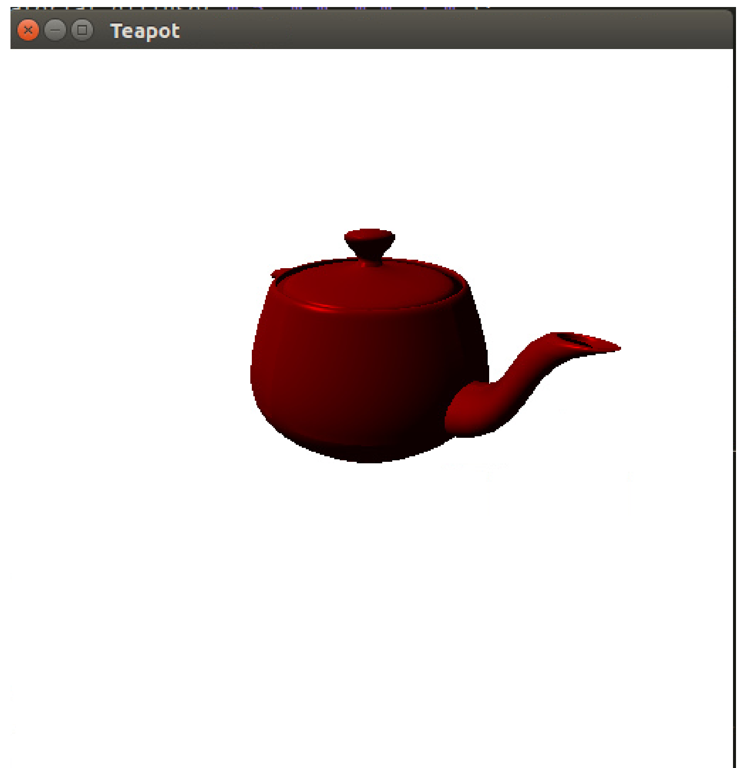
```
vec4    at( 0.0, 0.0, 0.0, 1.0 );
```

Assuming the viewer is upright

```
vec4    up( 0.0, 1.0, 0.0, 0.0 );
```

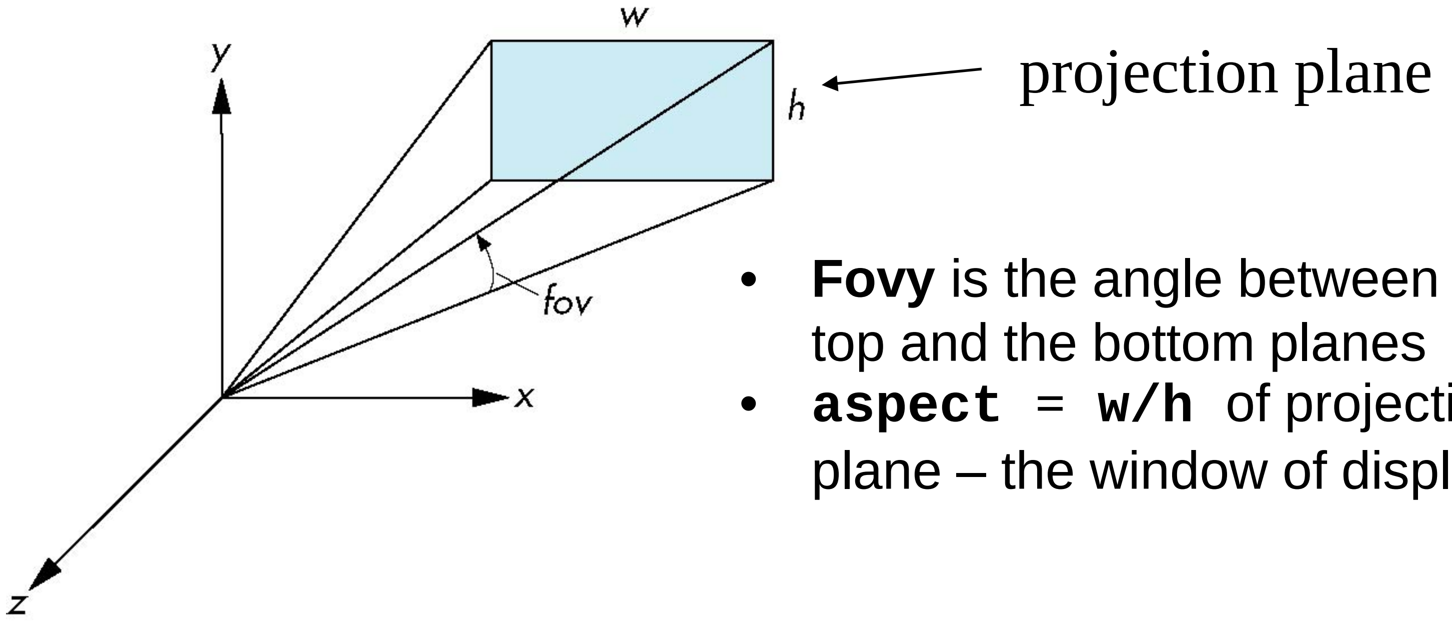
You need to choose “eye” appropriately

Project 2: Varying Eye



Perspective()

`Perspective(fovy, aspect, near, far)` often provides a better interface



- **Fovy** is the angle between the top and the bottom planes
- **`aspect = w/h`** of projection plane – the window of display

Topics

From vertices to fragments

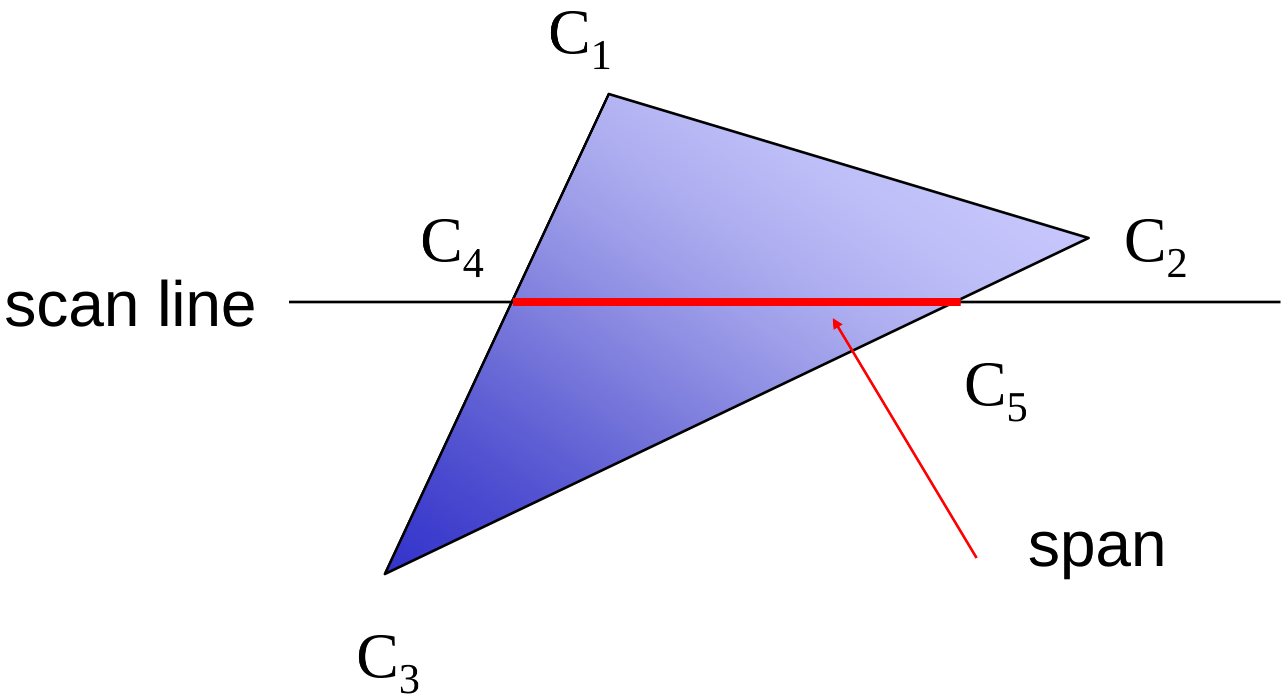
Filling in the Frame Buffer

Fill at end of pipeline: coloring a point with the inside color if it is inside the polygon

- Convex Polygons only
- Nonconvex polygons assumed to have been tessellated
- Shades (colors) have been computed for vertices (Gouraud shading)
- Scanline fill
- Flood fill

Scanline Fill: Using Interpolation

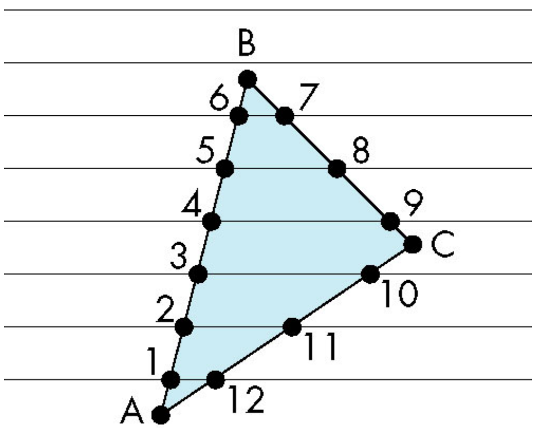
C_1 C_2 C_3 specified by `glColor` or by vertex shading
C_4 determined by interpolating between C_1 and C_2
C_5 determined by interpolating between C_2 and C_3
Interpolate points between C_4 and C_5 along span



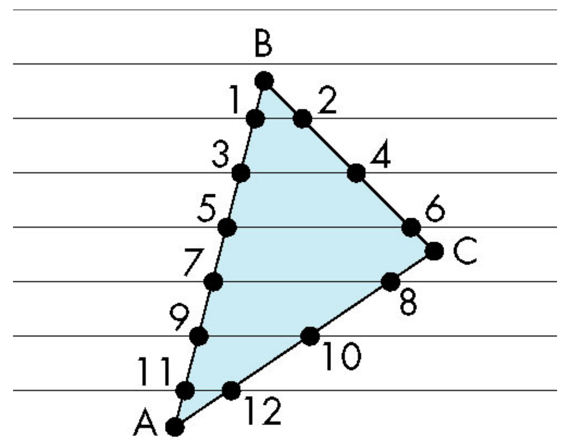
Scan Line Fill

Can also fill by maintaining a data structure of all intersections of polygons with scan lines

- Sort by scan line
- Fill each span



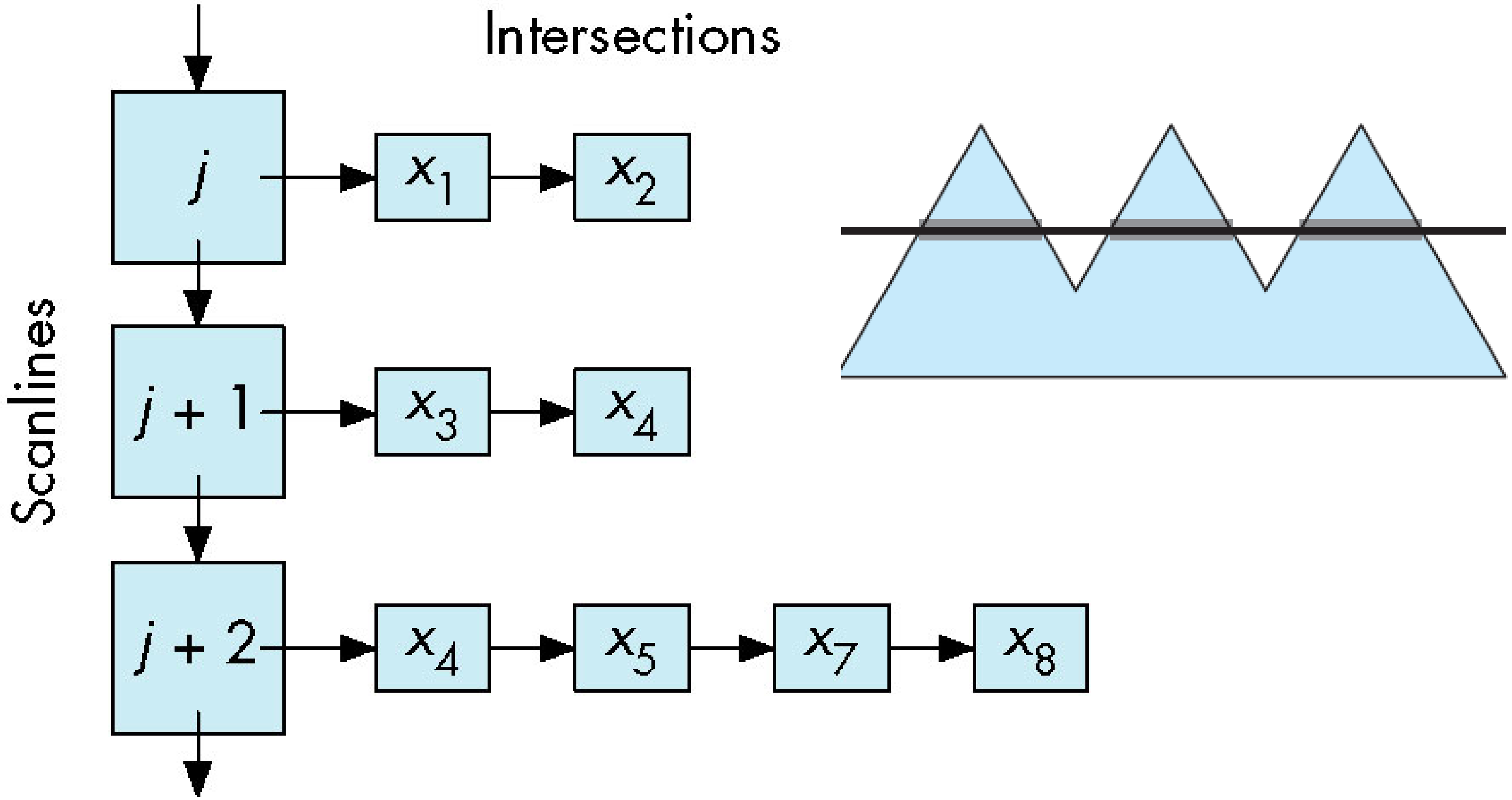
vertex order generated by vertex list



desired order

Data Structure

Insertion sort is applied on the x-coordinates for each scanline

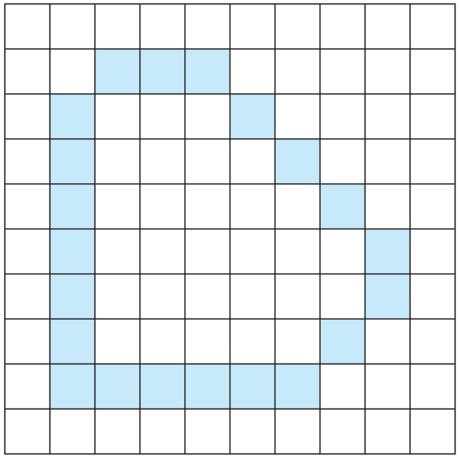


Flood Fill

Starting with an unfilled polygon, whose edges are rasterized into the buffer, fill the polygon with inside color (BLACK)

Fill can be done recursively if we know a seed point located inside. Color the neighbors to (BLACK) if they are not edges.

```
flood_fill(int x, int y) {
    if(read_pixel(x,y)= = WHITE) {
        write_pixel(x,y,BLACK);
        flood_fill(x-1, y);
        flood_fill(x+1, y);
        flood_fill(x, y+1);
        flood_fill(x, y-1);
}   }
```



Back-Face Removal (Culling)

Only render front-facing polygons

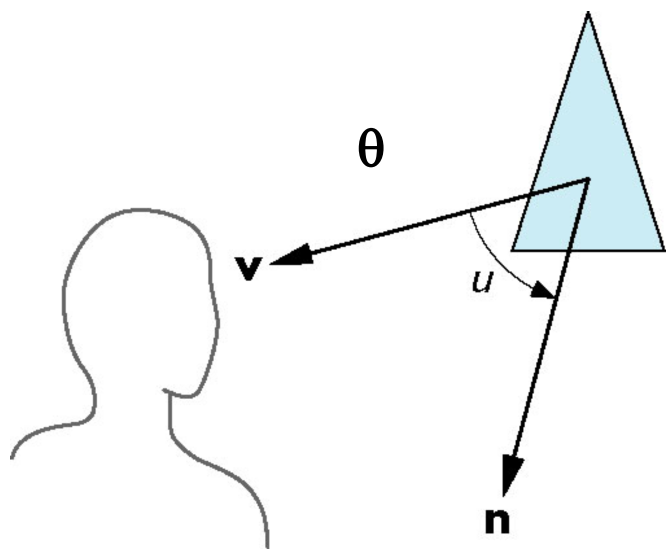
Reduce the work by hidden surface removal

Face is visible iff $-\frac{\pi}{2} \leq \theta \leq \frac{\pi}{2}$

equivalently

$\cos\theta \geq 0$ **or** $\mathbf{v} \bullet \mathbf{n} \geq 0$

Easy to compute



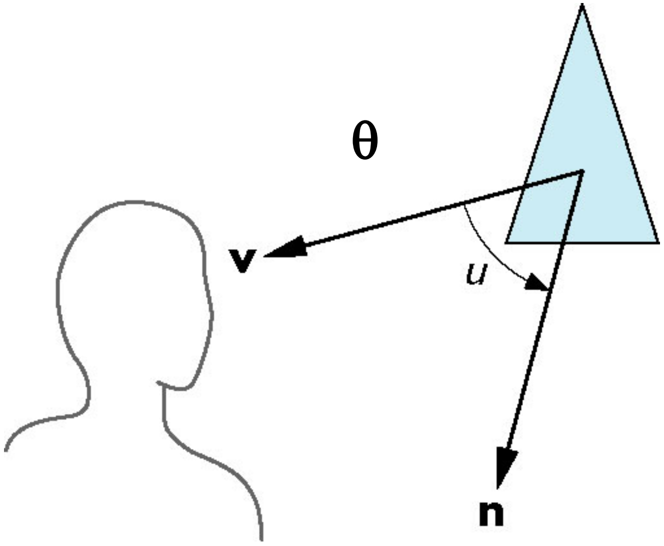
Back-Face Removal (Culling)

- After transformation (projection normalization), the view is orthographic

$$\mathbf{v} = (\ 0\ 0\ 1\ 0)^T$$

- The coordinates are normalized device coordinates
- If the plane of face has form

$$ax + by + cz + d = 0$$



E. Angel and D. Shreiner: Interactive Computer Graphics 6E © Addison-Wesley 2012

Need only test the sign of c

Why?

$$\mathbf{n} = \begin{bmatrix} a \\ b \\ c \\ 0 \end{bmatrix}, d = P_0 \cdot \mathbf{n}$$

In OpenGL we can simply enable culling but may not work correctly if we have nonconvex objects

Hidden Surface Removal

Object-space algorithms:

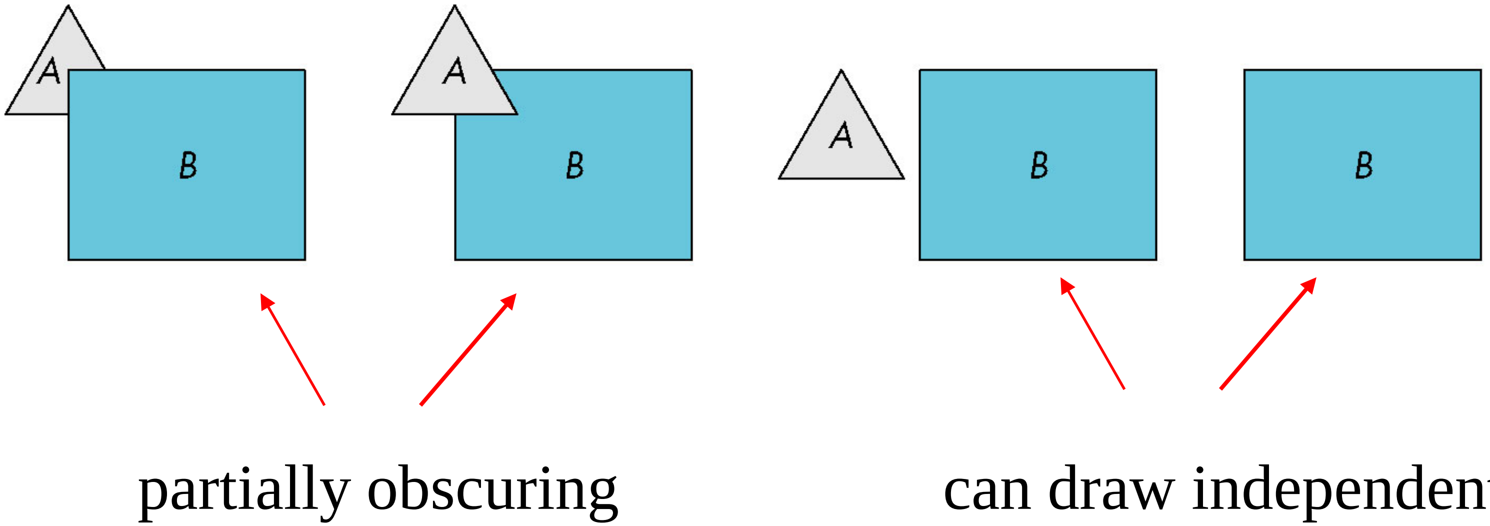
- Consider the relationships between objects
- Reduce number of polygons
- Works better for a smaller number of objects

Image-space algorithms:

- Ray casting
- Works at fragment/pixel level
- Most popular

Hidden Surface Removal

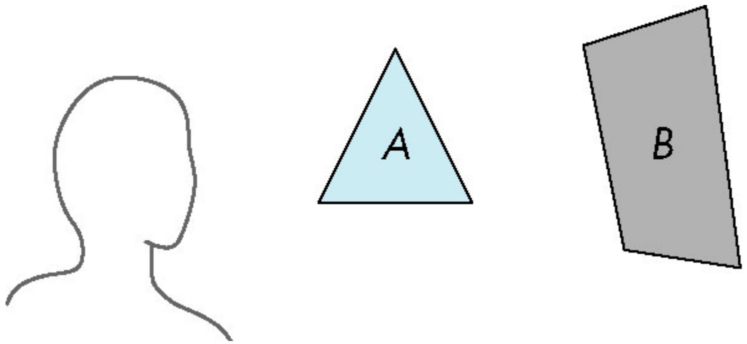
Object-space approach: use pairwise testing between polygons (objects)



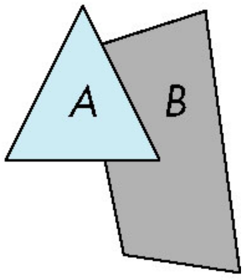
Worst case complexity $O(n^2)$ for *n* polygons

Painter's Algorithm

Render polygons a back to front order so that polygons behind others are simply painted over



B behind A as seen by viewer



Fill B then A

Back-to-front rendering

A depth sorting is needed!

Depth Sort

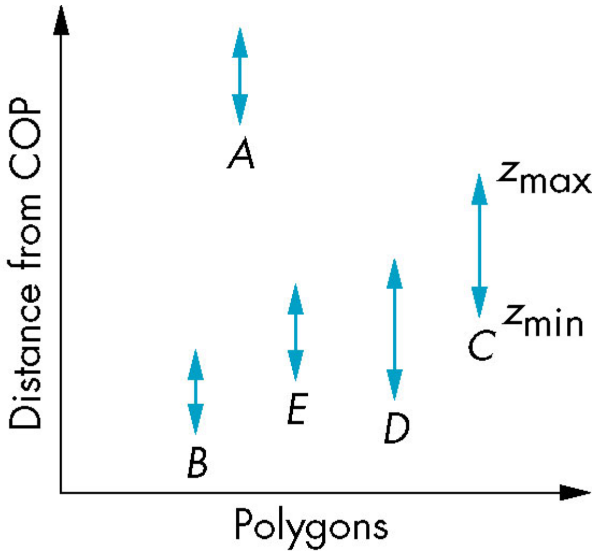
Requires ordering of polygons first

- Object-oriented hidden-surface removal
- O(n log n) calculation for ordering
- Not every polygon is either in front or behind all other polygons

Order polygons and deal with

easy cases first, harder later

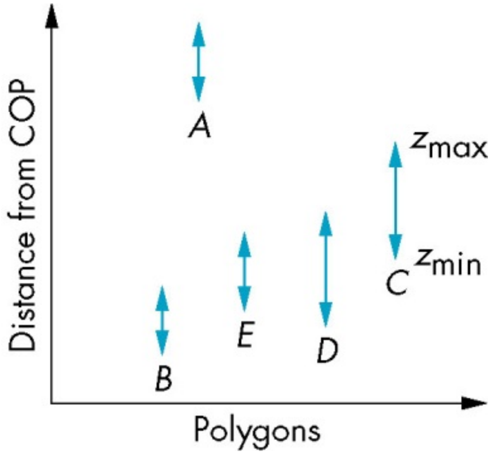
Polygons sorted by distance from COP



Easy Cases

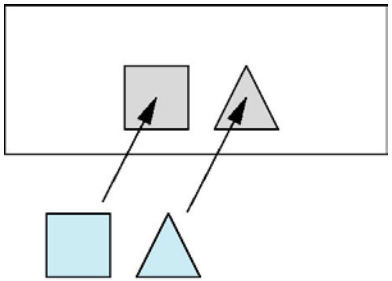
Case 1: A lies behind all other polygons

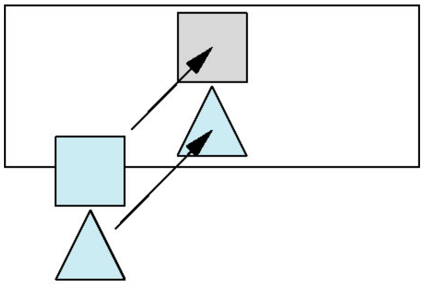
- Minimum depth of A is larger than maximum depth of the others
- Render A first



Case 2: Polygons overlap in z but not in either x or y

- Can render independently

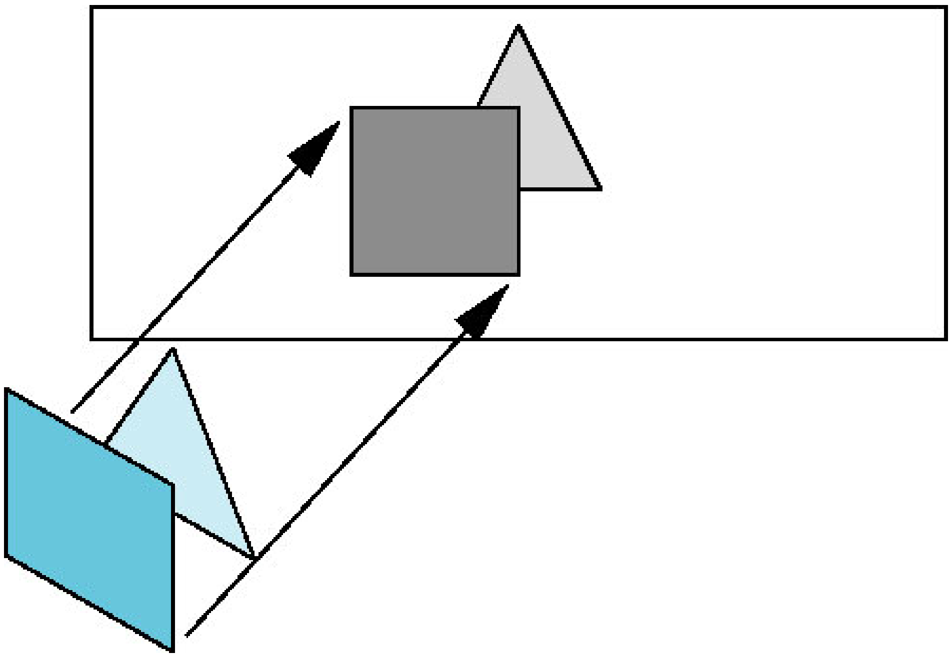




Hard Cases: Overlap in All Directions

Case 3: Two polygons overlap

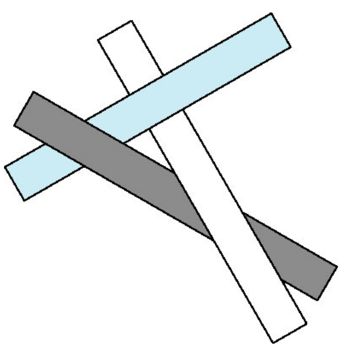
All vertices of one polygon are on one side of the other



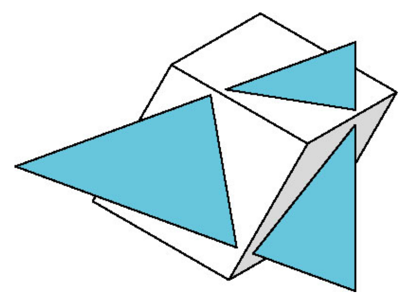
Hard Cases: Overlap in All Directions

Three or more polygons overlap

Need to divide at least one of the polygons to several parts and find the depth order of the new polygons



cyclic overlap



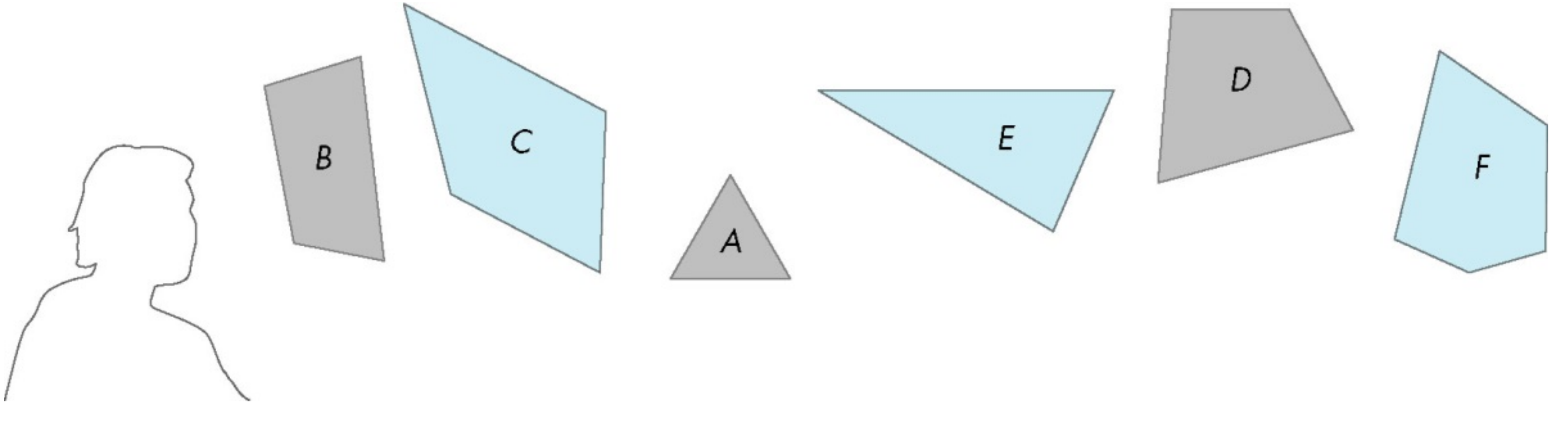
penetration

Visibility Testing

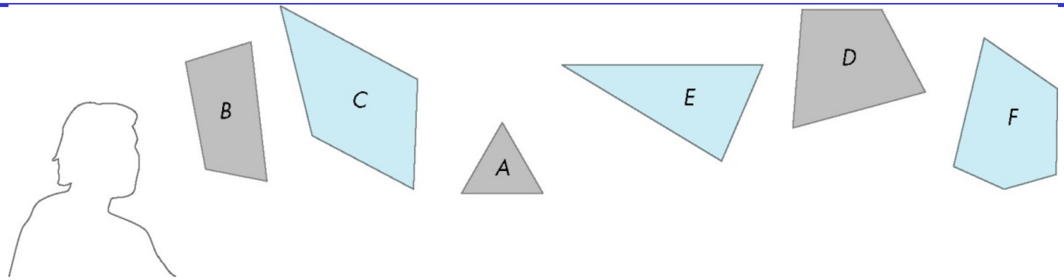
In many realtime applications, such as games, we want to eliminate as many objects as possible within the application

- Reduce burden on pipeline
- Reduce traffic on bus

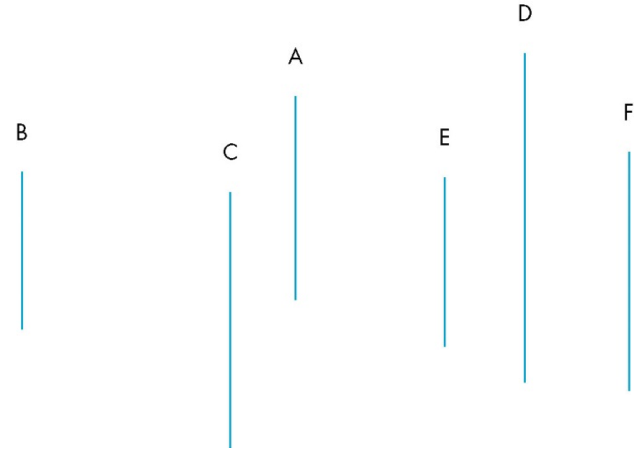
Partition space with Binary Spatial Partition (BSP) Tree



Simple Example



consider 6 parallel polygons



top view

The plane of A separates B and C from D, E and F

BSP Tree

Can continue recursively

- Plane of C separates B from A
- Plane of D separates E and F

Can put this information in a BSP tree

- Use for visibility and occlusion testing

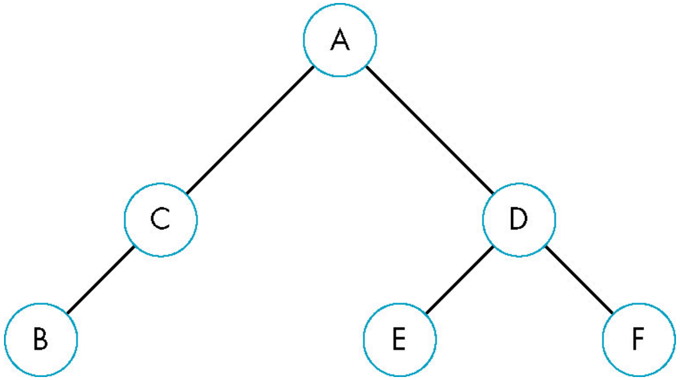
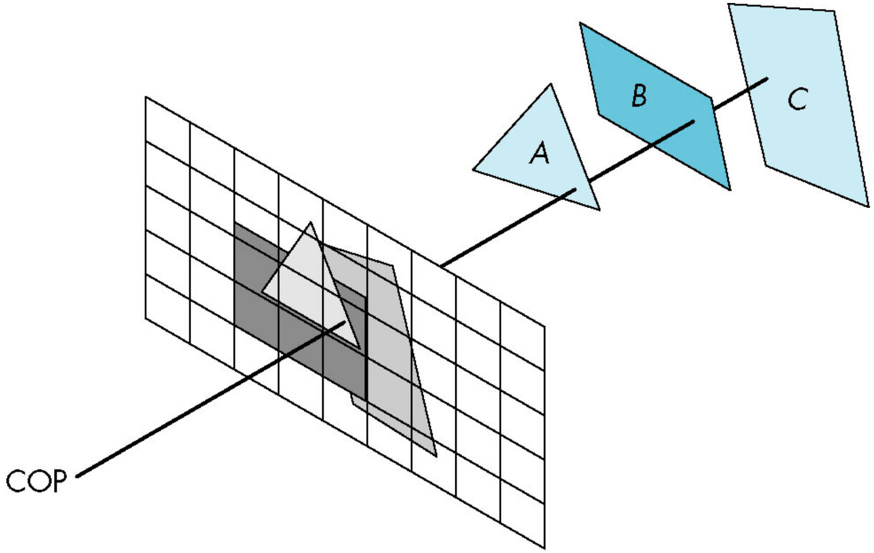


Image Space Approach

Look at each projector (nm for an $n \times m$ frame buffer) and find the closest among k polygons to COP

- Complexity $O(nmk)$
- Ray tracing
- z-buffer

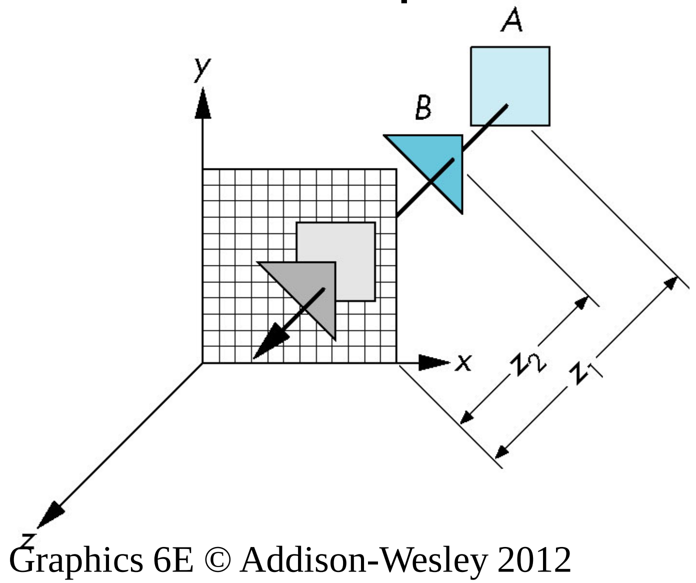


z-Buffer Algorithm

Use a buffer called the z or depth buffer to store the depth of the closest object at each pixel found so far

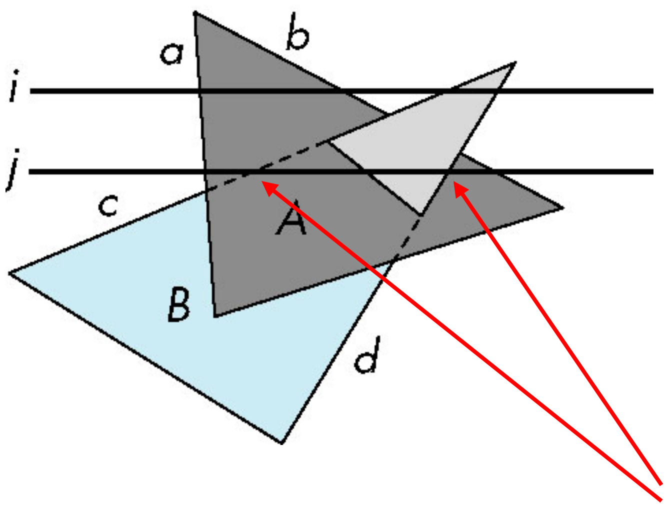
As we render each polygon, compare the depth of each pixel to depth in z buffer

If less, place shade of pixel in color buffer and update z buffer



Scan-Line Algorithm

Can combine shading and hidden surface removal through scan line algorithm



scan line i: no need for depth information, can only be in no or one polygon

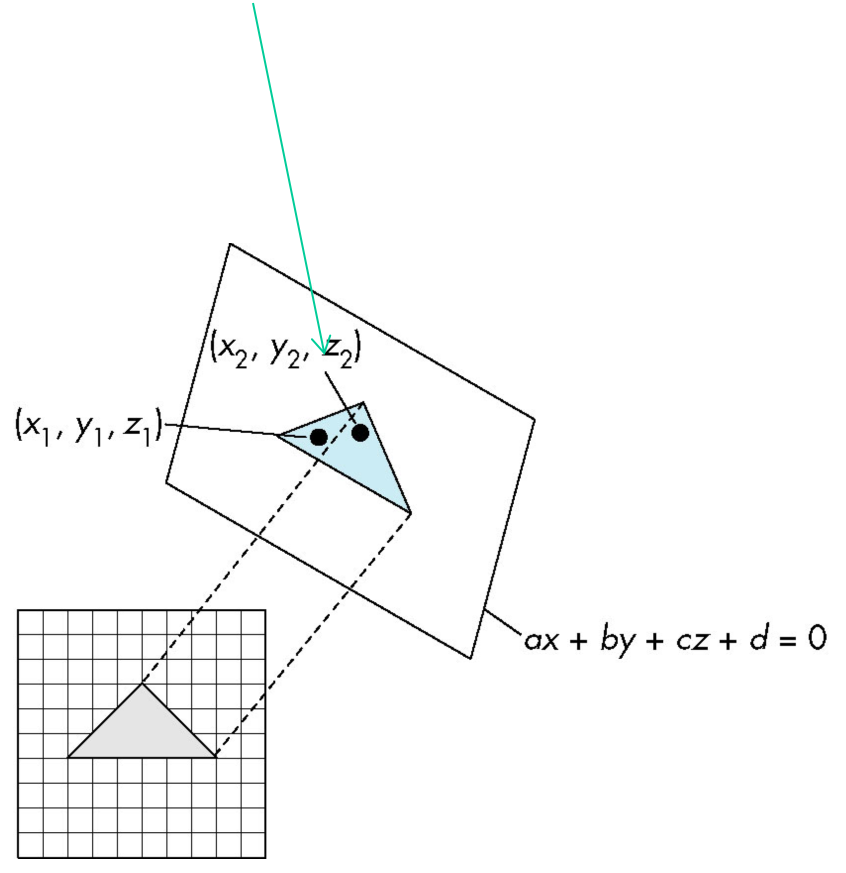
scan line j: need depth information only when in more than one polygon

Scan-Line Algorithm

A polygon is on a plane $ax + by + cz + d = 0$.

Two points on the plane with
$\Delta x = x_2 - x_1$
$\Delta y = y_2 - y_1$
$\Delta z = z_2 - z_1$

Then the plane equation becomes
$a\Delta x + b\Delta y + c\Delta z = 0$



Scan-Line Algorithm

As we move across a scan line, the depth changes satisfy

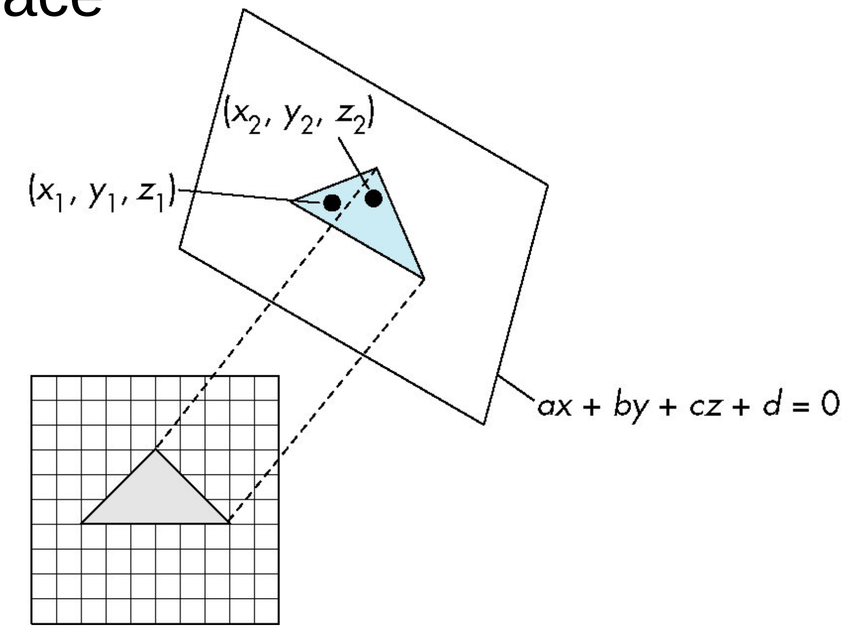
$$a\Delta x + b\Delta y + c\Delta z = 0$$

Along scan line, in screen space

$\Delta x = 1$

$\Delta y = 0$

$\Rightarrow \quad \Delta z = -\frac{a}{c}\Delta x$



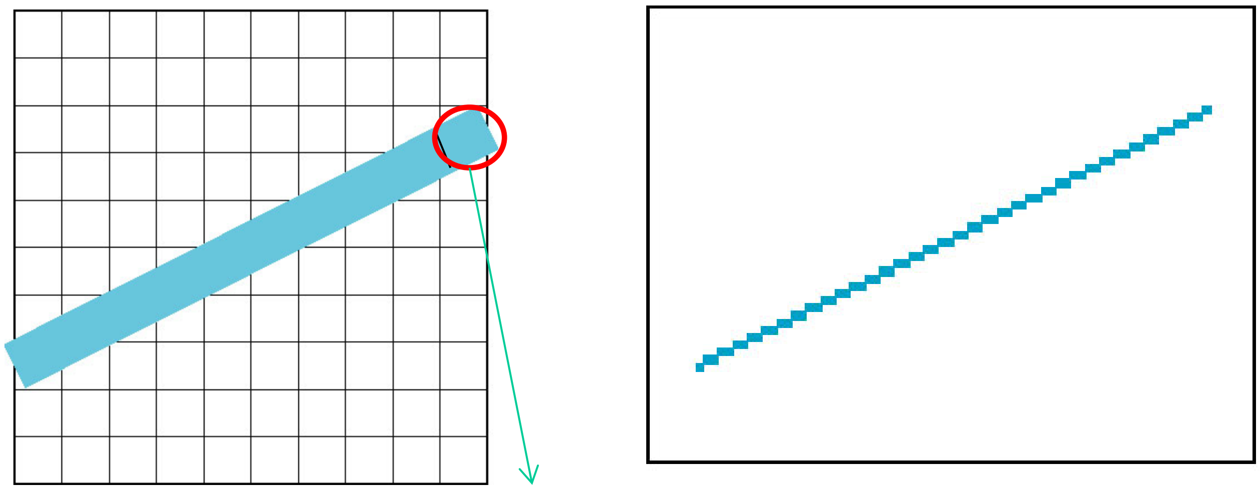
Implementation

Need a data structure to store

- Flag for each polygon (inside/outside)
- Incremental structure for scan lines that stores which edges are encountered
- Parameters for planes

Aliasing

Ideal rasterized line should be 1 pixel wide



ideal

An ideal point covers multiple pixels

actual

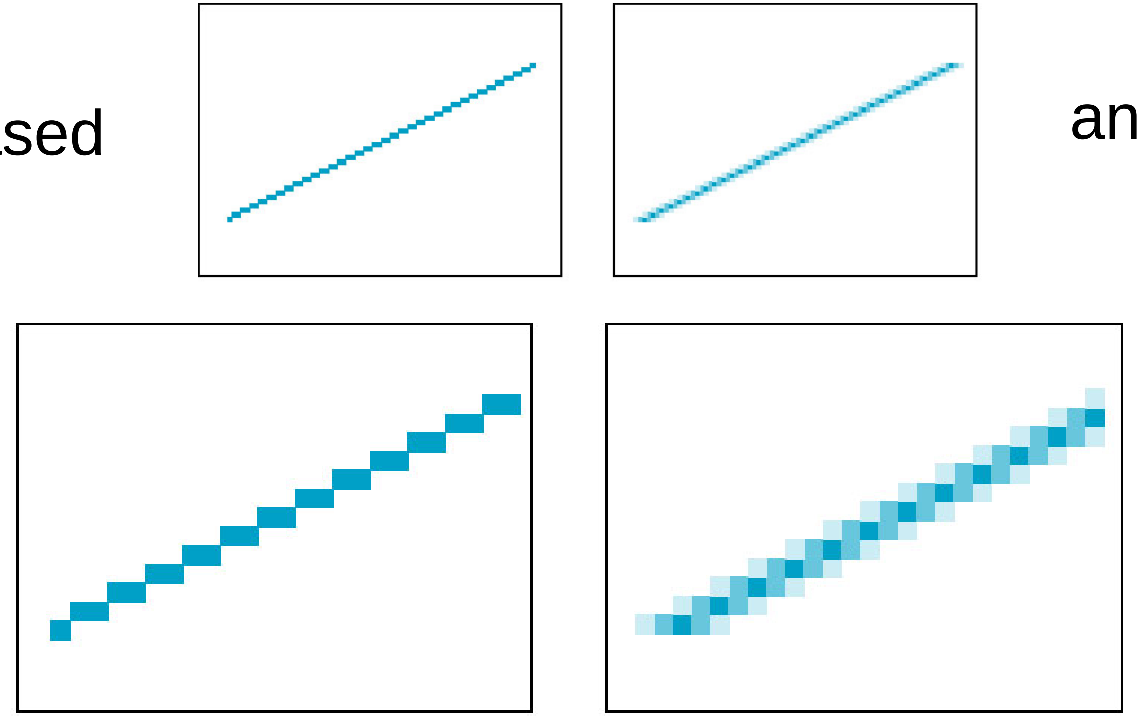
Choosing best y for each x (or visa versa) produces aliased raster lines

Antialiasing by Area Averaging

Shade each pixel by the percentage of the area covered by the ideal line

aliased

antialiased



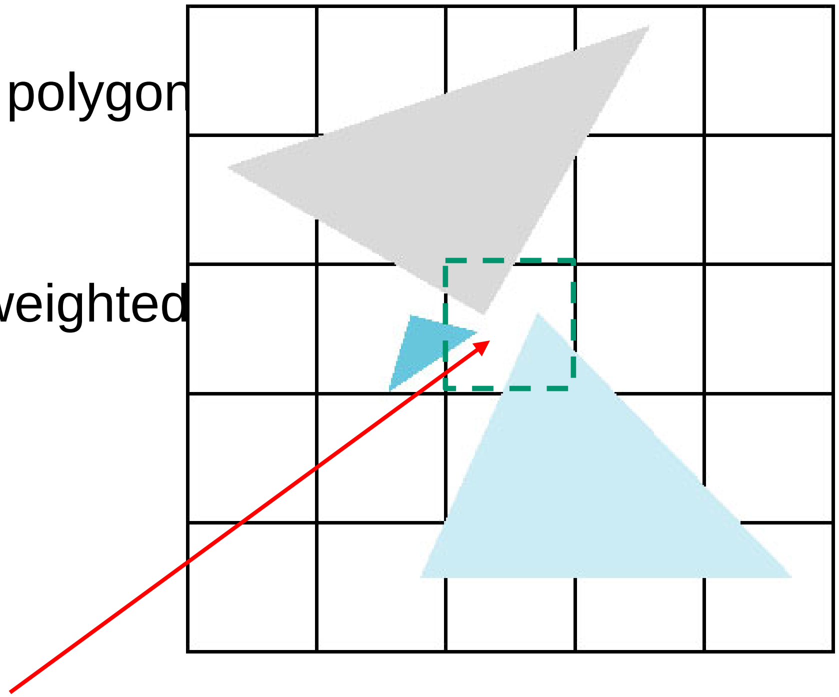
magnified

Polygon Aliasing

Aliasing problems can be serious for polygons

- Jaggedness of edges
- Small polygons neglected
- Color of pixel is determined by the polygon closest to the COP

Composing the color based on the weighted average color of all the polygons



All three polygons should contribute to color

Reading Assignment

Chapter 6.13 of Angel & Shreiner

Chapter 7 of Shreiner et al

Buffers

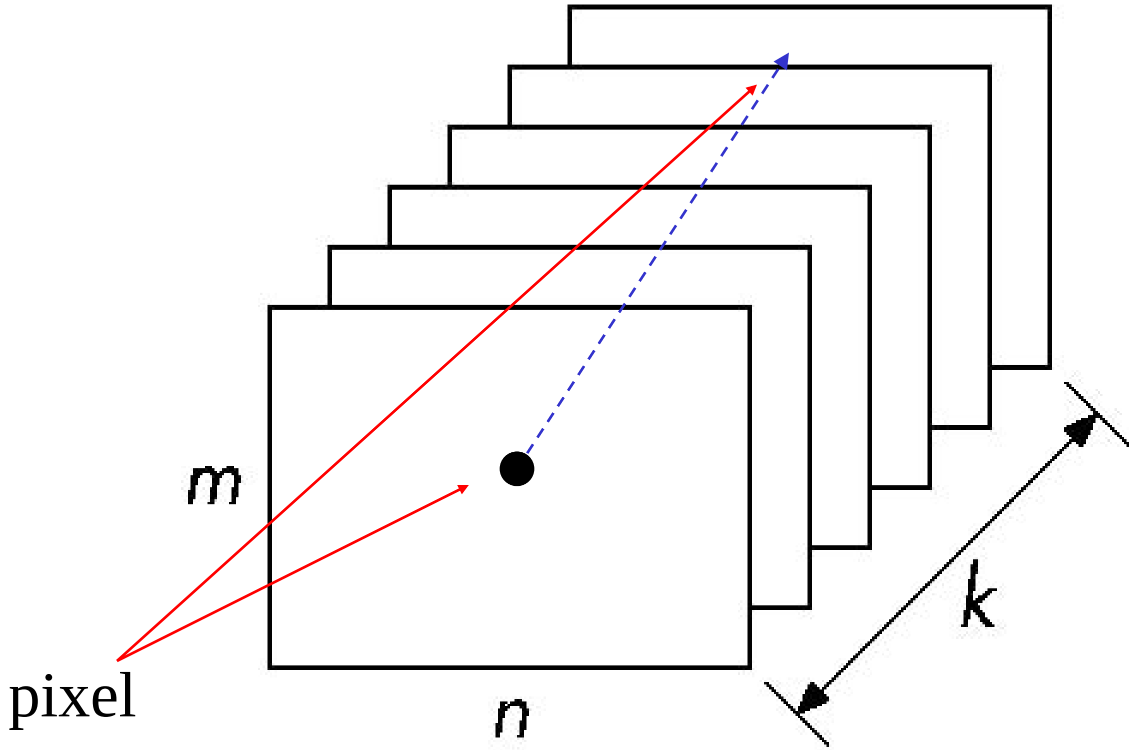
Introduce additional OpenGL buffers

Learn to read from buffers

Learn to use blending

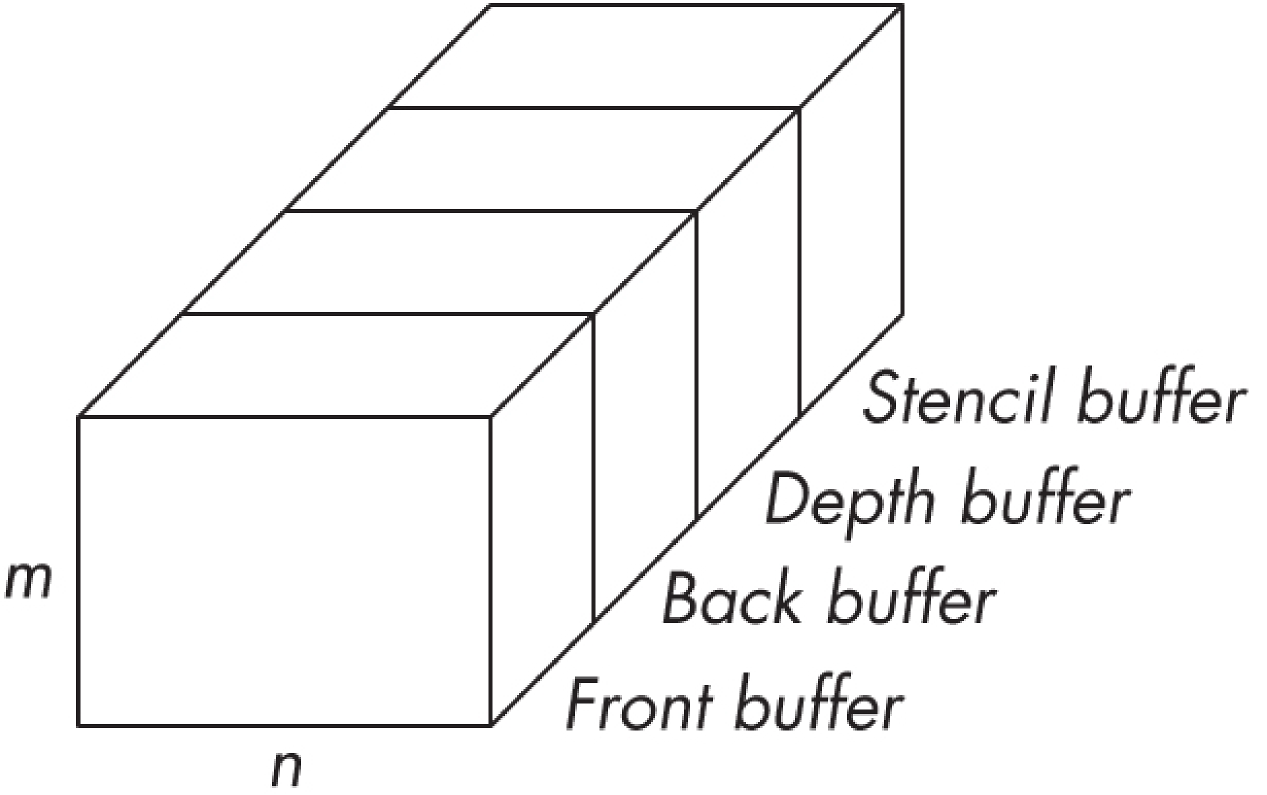
Buffer

Define a buffer by its spatial resolution (n x m) and its depth (or precision) k, the number of bits/pixel



OpenGL Frame Buffer

64 bits for front and back buffers



OpenGL Buffers

Color buffers can be displayed

- Front
- Back
- Stereo

Depth

Stencil

- Holds masks (per-pixel integers) to control rendering

Most RGBA buffers 8 bits per component

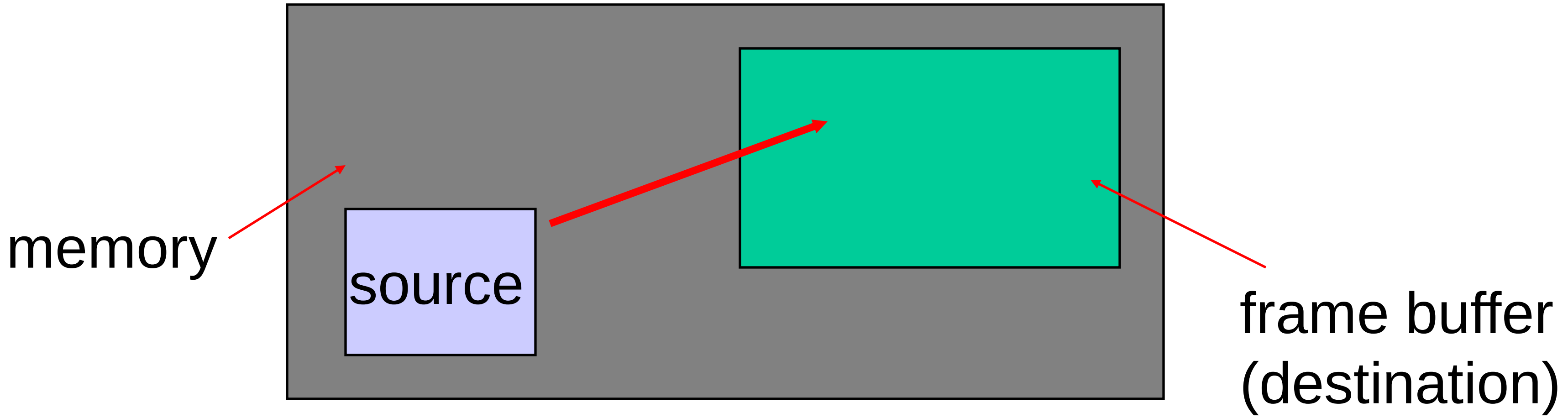
Writing in Buffers

Conceptually, we can consider all of memory as a large two-dimensional array of pixels

In practice, we read and write rectangular blocks of pixels

- *Bit block transfer* (*bitblt*) *operations*

The frame buffer is part of this memory



writing into frame buffer

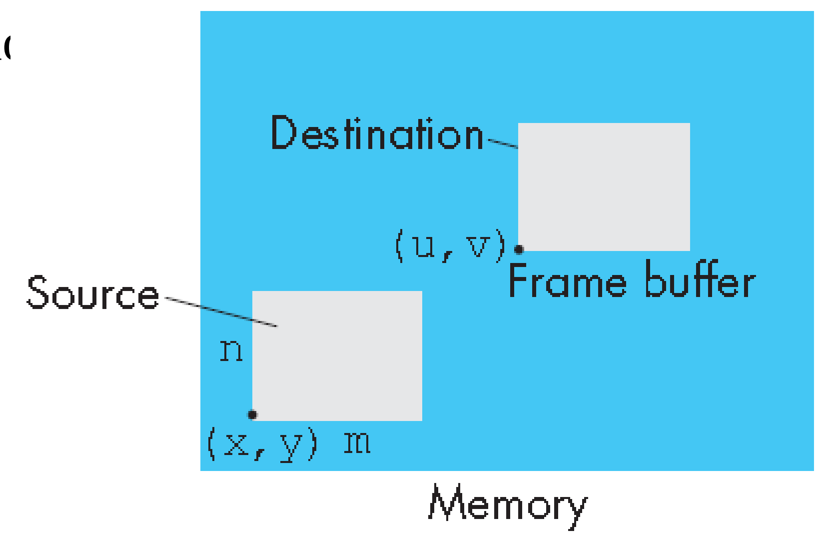
Writing in Buffers

Write an nxm source block with

Lower-left corner of destination block

```
write_block(source, n, m, x, y, destination, u, v);
```

Lower-left corner of source bl

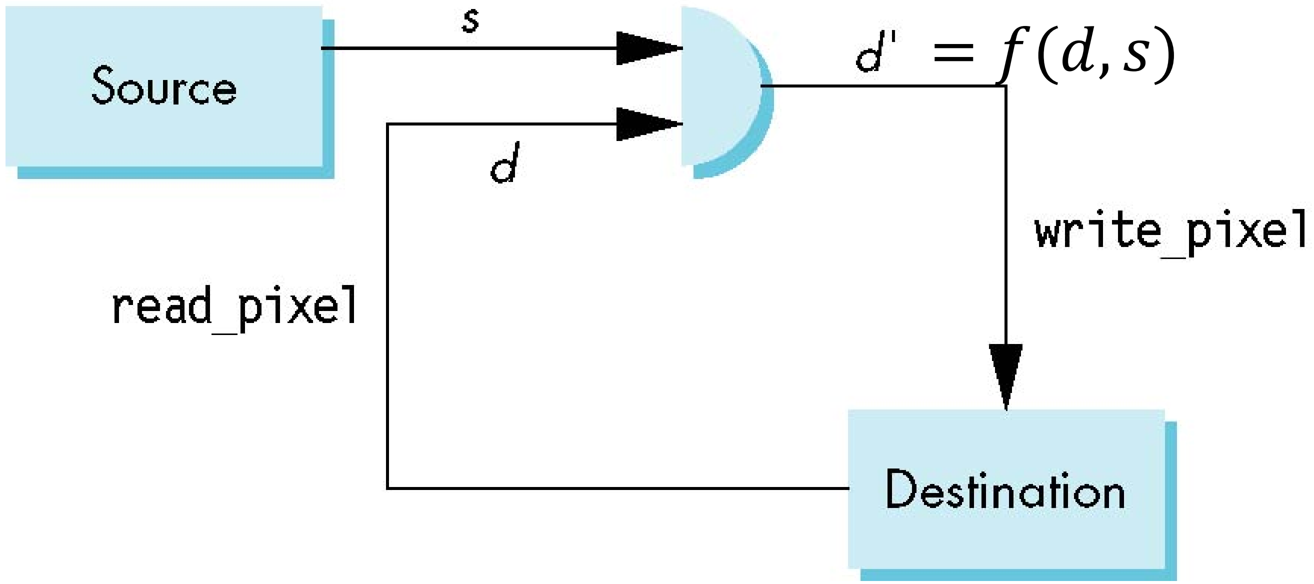


Writing Model

s: source bit

d: destination bit

Read destination pixel before writing source



Bit Writing Modes

Source and destination bits are combined bitwise

16 possible functions (one per column in table)

0 and 15: clear mode; 3 and 7: write mode

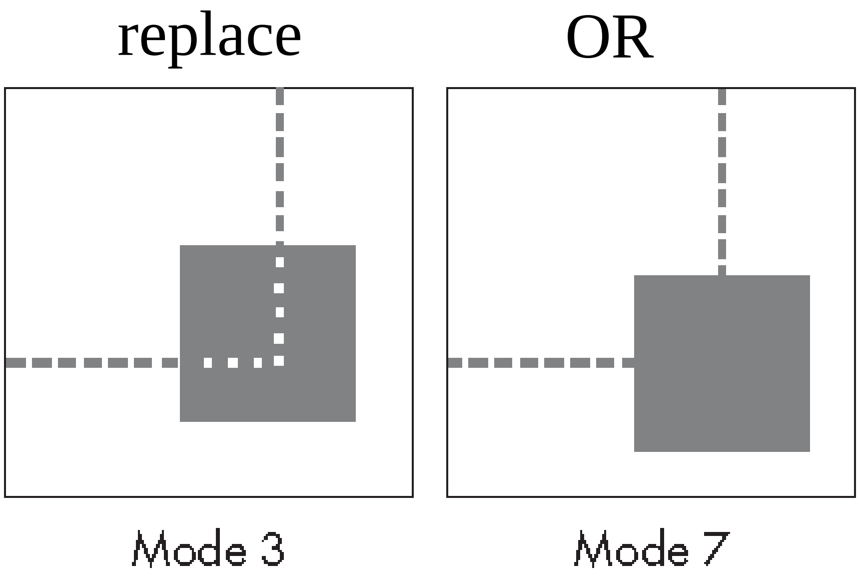
replace (3) XOR (6) OR (7)

s	*d*	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Bit Writing Modes

Background color: white

Foreground color: black



XOR (Exclusive OR) Mode

Property of XOR: return the original value if apply XOR twice

$$d = (d \oplus s) \oplus s$$

XOR is especially useful for swapping blocks of memory such as menus that are stored off screen (***backing store***)

If S represents screen and M represents a menu, the sequence

$$S \leftarrow S \oplus M$$

$$M \leftarrow S \oplus M$$

$$S \leftarrow S \oplus M$$

swaps S and M

For example, S=1010, M=1100

$S = S \oplus M = 0110$

$M = S \oplus M = 1010$

$S = S \oplus M = 1100$