# Computer Graphics ECSE-4750 FALL 2015

CLASS 8

# Class

- Term Project
- Polygons, geometry, and should I draw them
- Textures and Coordinates
- Homework 4

# Term Project

- You will think of, implement, and document a program relating to graphics. It must demonstrate both graphics input and output; other than that it must only be legal and ethical.
- This may be done by a team, and may be combined with another course's project if you get the approval of everyone involved.
- You need to work in teams of 4-5 people
- This is on purpose, businesses need you to work well in teams.

# Polygons, geometry, and clipping

# Polygon Clipping

Not as simple as line segment clipping

- Clipping a line segment yields at most one line segment
- Clipping a polygon can yield multiple polygons

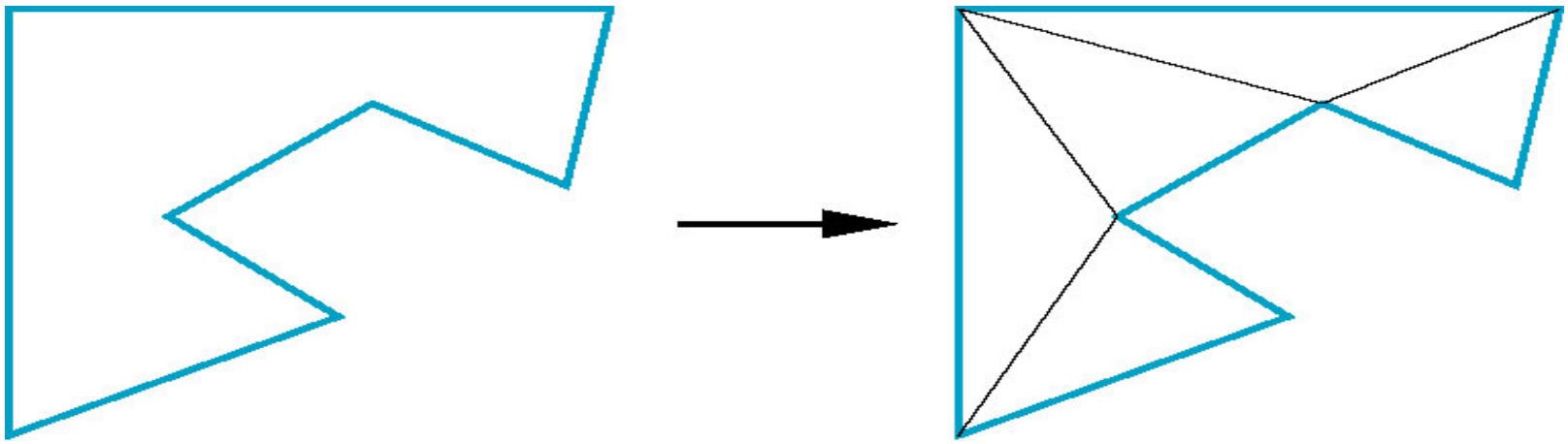
However, clipping a convex polygon can yield at most one other polygon

# Tessellation and Convexity

One strategy is to replace nonconvex (*concave*) polygons with a set of triangular polygons (a *tessellation*)

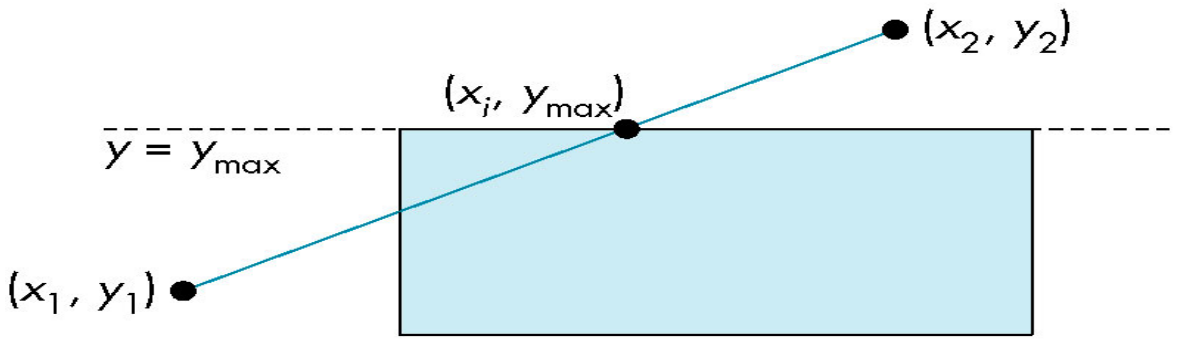
Also makes fill easier

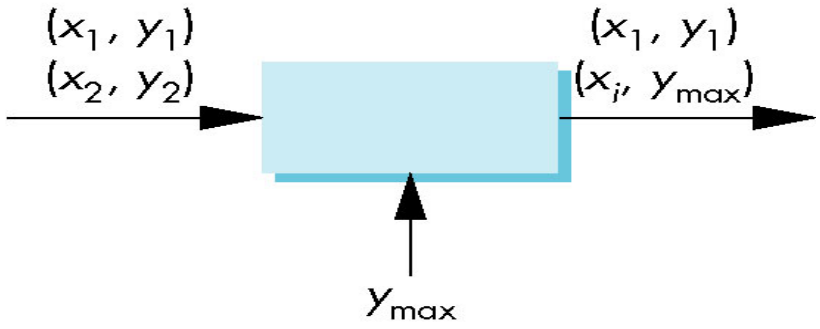
Tessellation through tesselllation shaders



# Clipping as a Black Box

Can consider line segment clipping as a process that takes in two vertices and produces either no vertices or the vertices of a clipped line segment

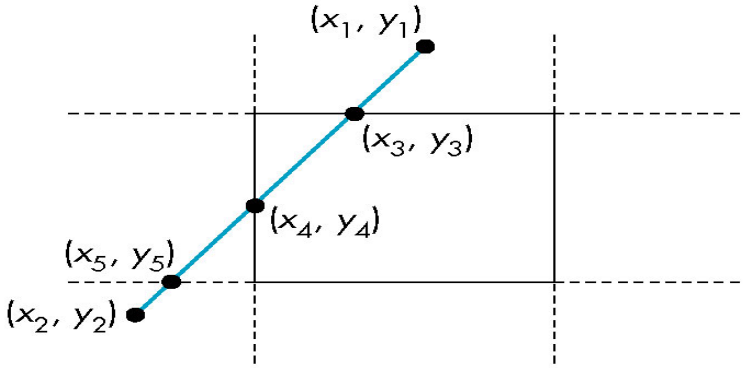


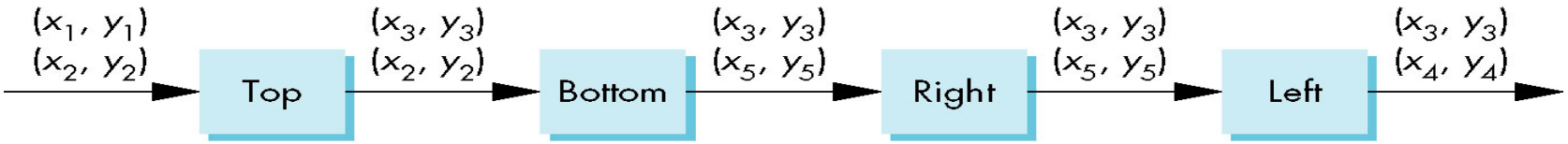



# Pipeline Clipping of Line Segments

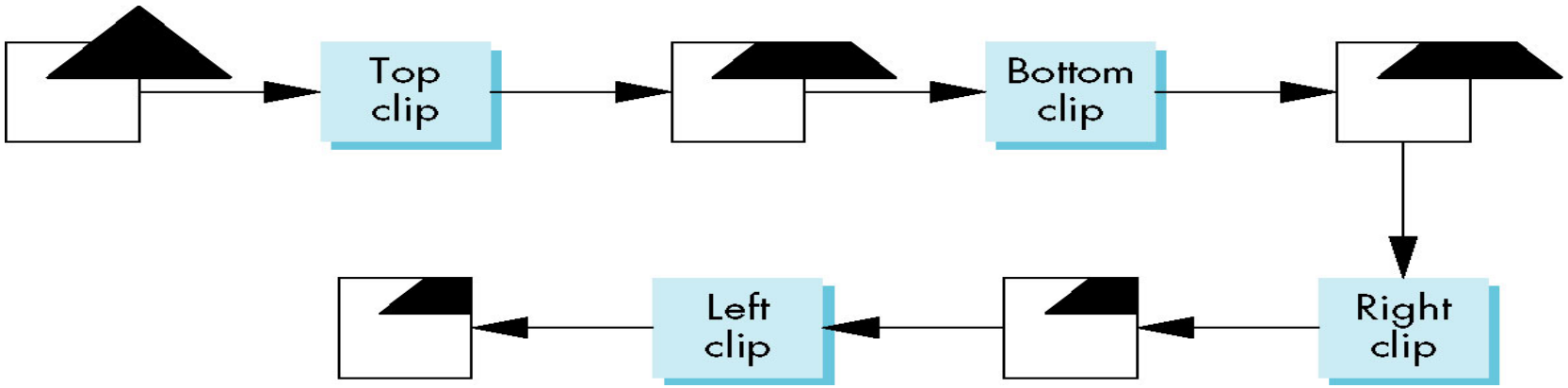
Clipping against each side of window is independent of other sides

- Can use four independent clippers in a pipeline



# Pipeline Clipping of Polygons

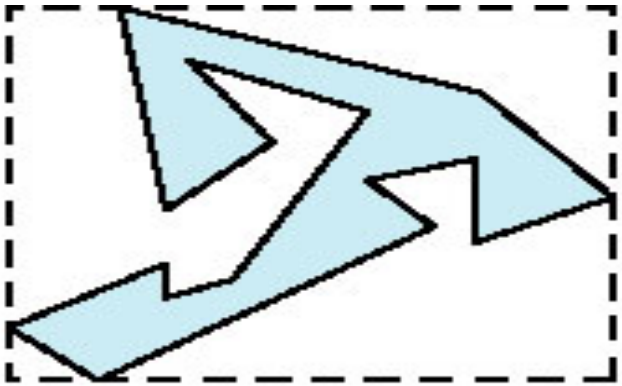



- Three dimensions: add front and back clippers
- Strategy used in SGI Geometry Engine
- Small increase in latency

# Bounding Boxes

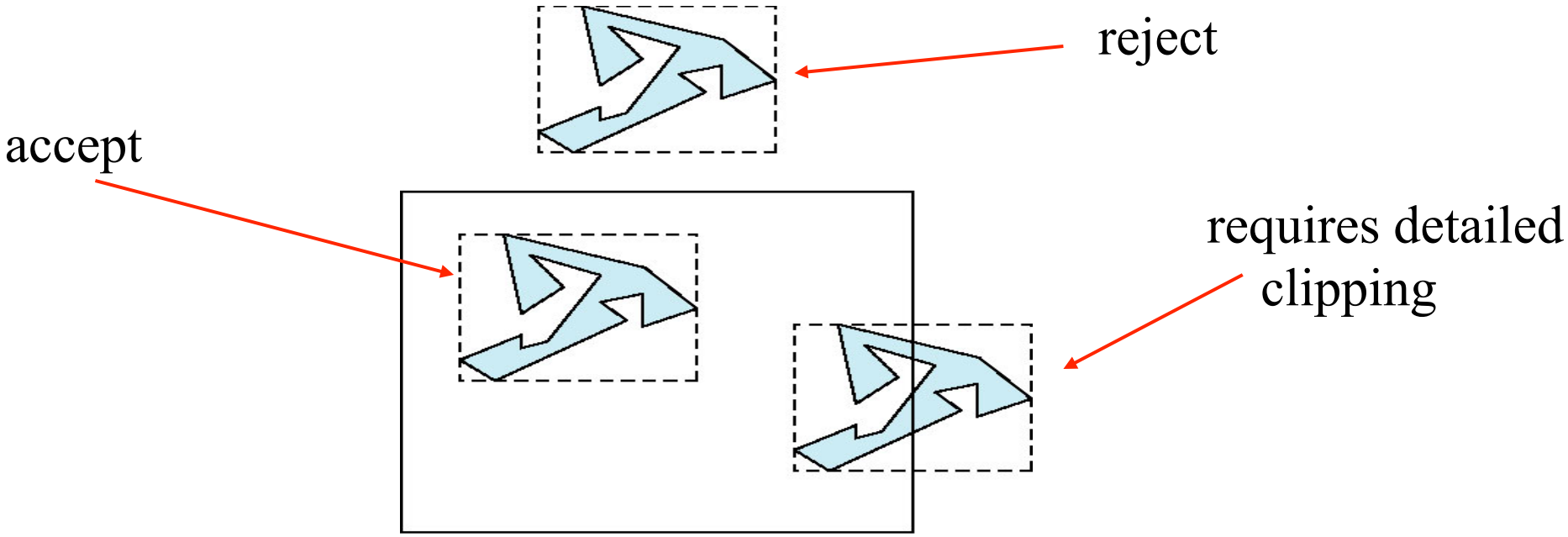
Rather than doing clipping on a complex polygon, we can use an *axis-aligned bounding box* or *extent*

- Smallest rectangle aligned with axes that encloses the polygon
- Simple to compute: max and min of x and y



# Bounding boxes

Can usually determine accept/reject based only on bounding box

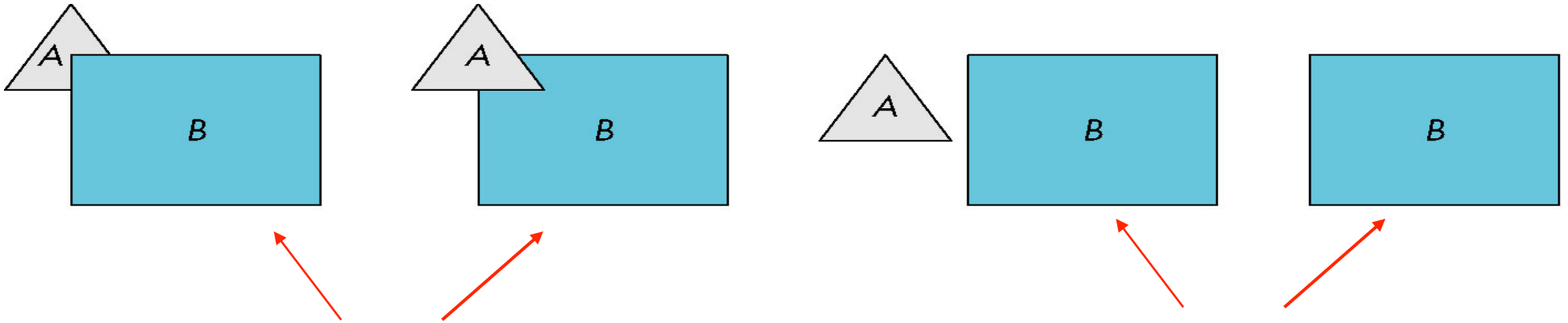


# Clipping and Visibility

- Clipping has much in common with hidden-surface removal
- In both cases, we are trying to remove objects that are not visible to the camera
- Often we can use visibility or occlusion testing early in the process to eliminate as many polygons as possible before going through the entire pipeline

# Hidden Surface Removal

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



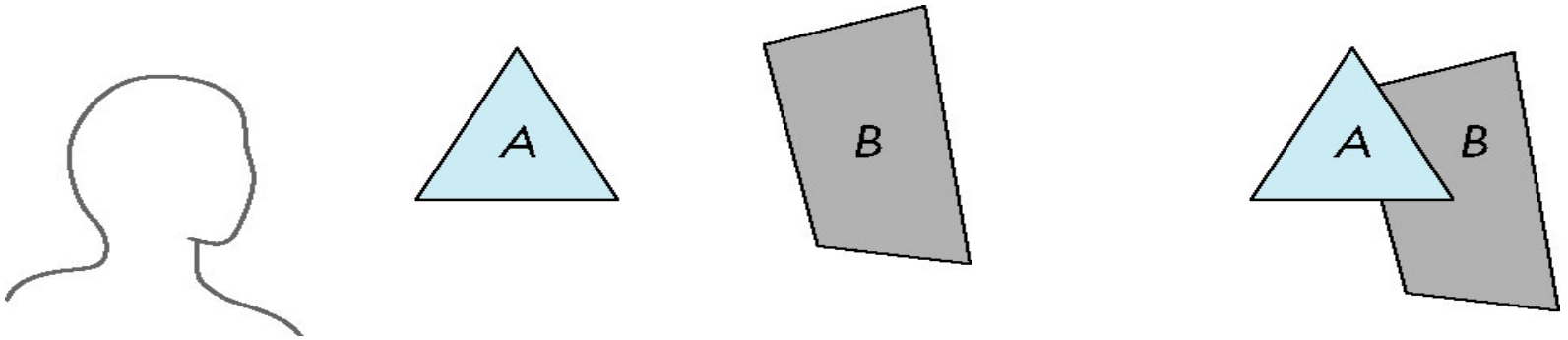

partially obscuring

can draw independently

- 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

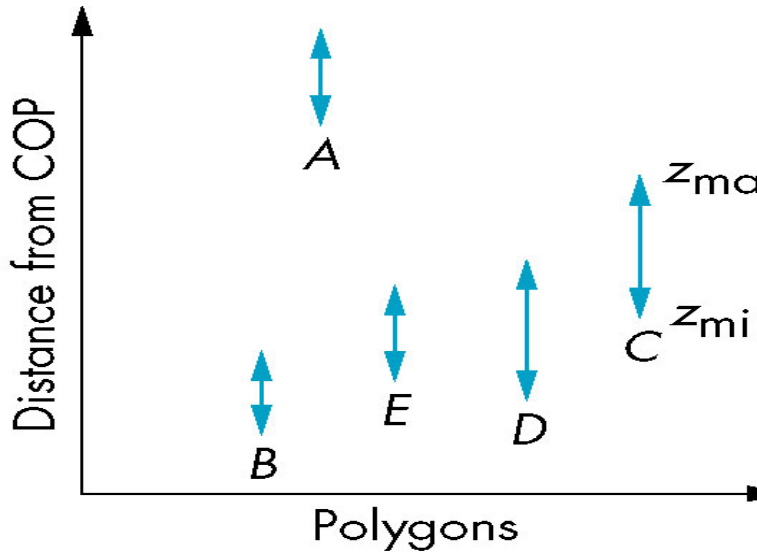
# Depth Sort

Requires ordering of polygons first

- 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

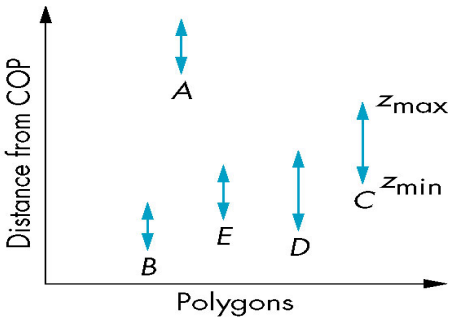
Polygons sorted by distance from COP



# Easy Cases

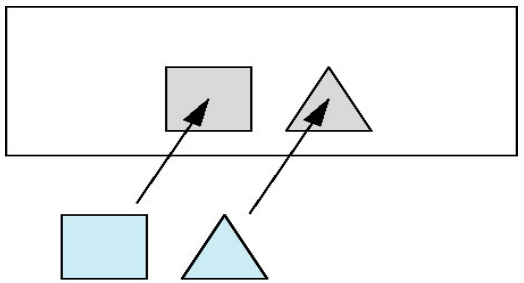
A lies behind all other polygons

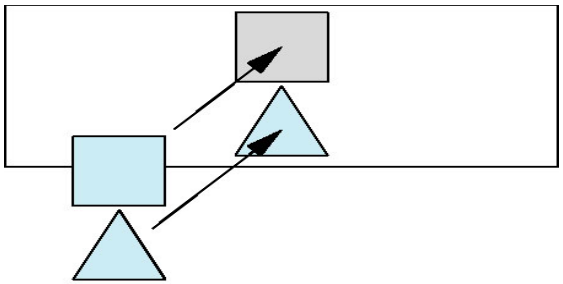
- Can render



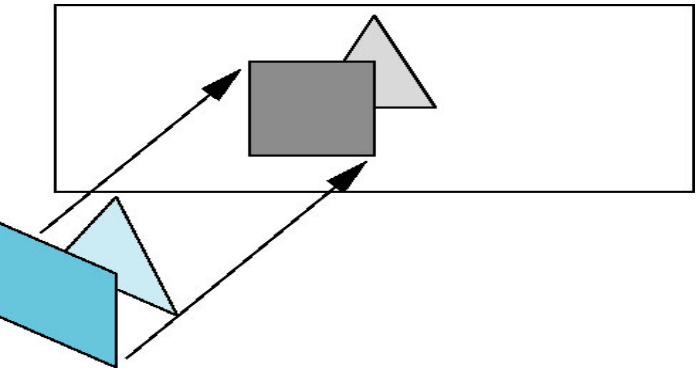

Polygons overlap in z but not in either x or y

- Can render independently

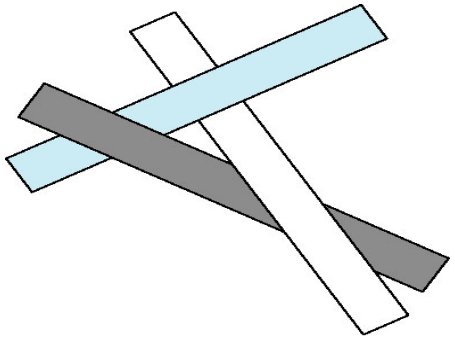




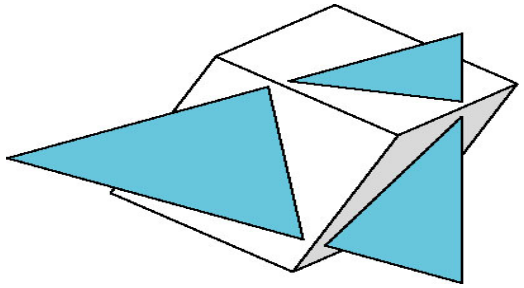
# Hard Cases



erlap in all directions
can one is fully on
side of the other

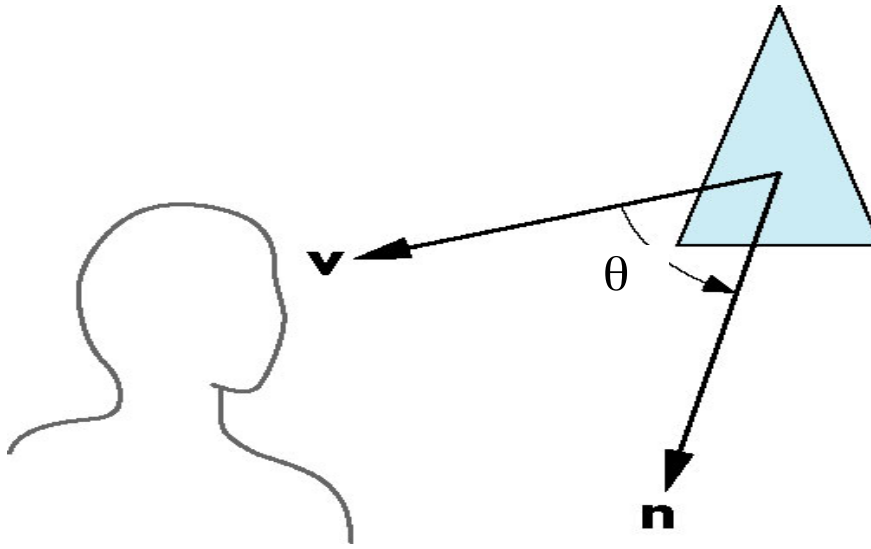


cyclic overlap



penetration

# Back-Face Removal (Culling)




ce is visible iff $90 \geq \theta \geq -90$
uivalently $\cos \theta \geq 0$
$\mathbf{v} \cdot \mathbf{n} \geq 0$

ne of face has form $ax + by + cz + d = 0$
after normalization $\mathbf{n} = (0\ 0\ 1\ 0)^T$

ed only test the sign of c

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

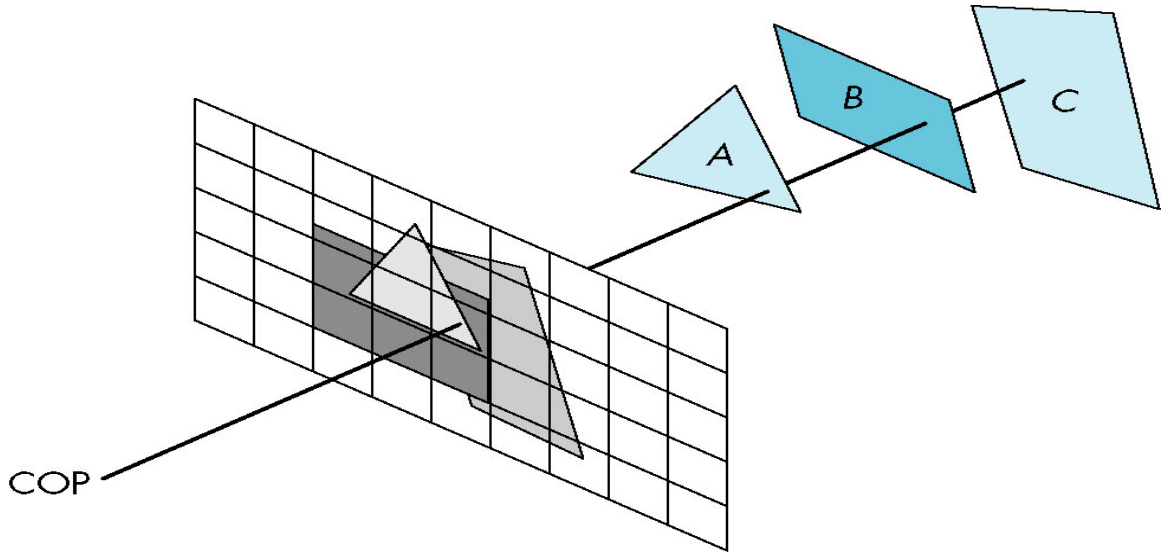
# mage Space Approach

Look at each projector ($nm$ for an $n$ x $m$ frame buffer) and find closest of $k$ polygons

Complexity O($nmk$)

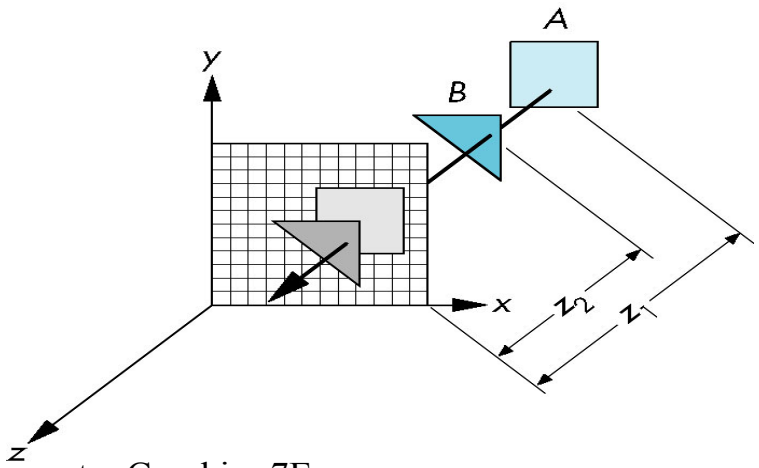
Ray tracing

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

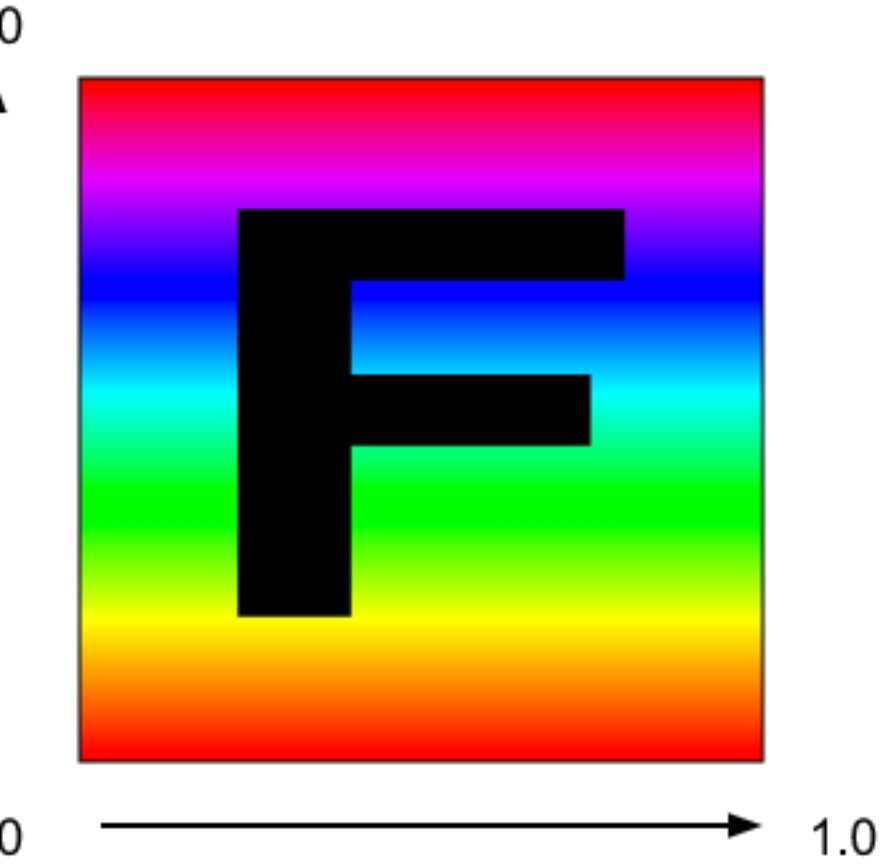


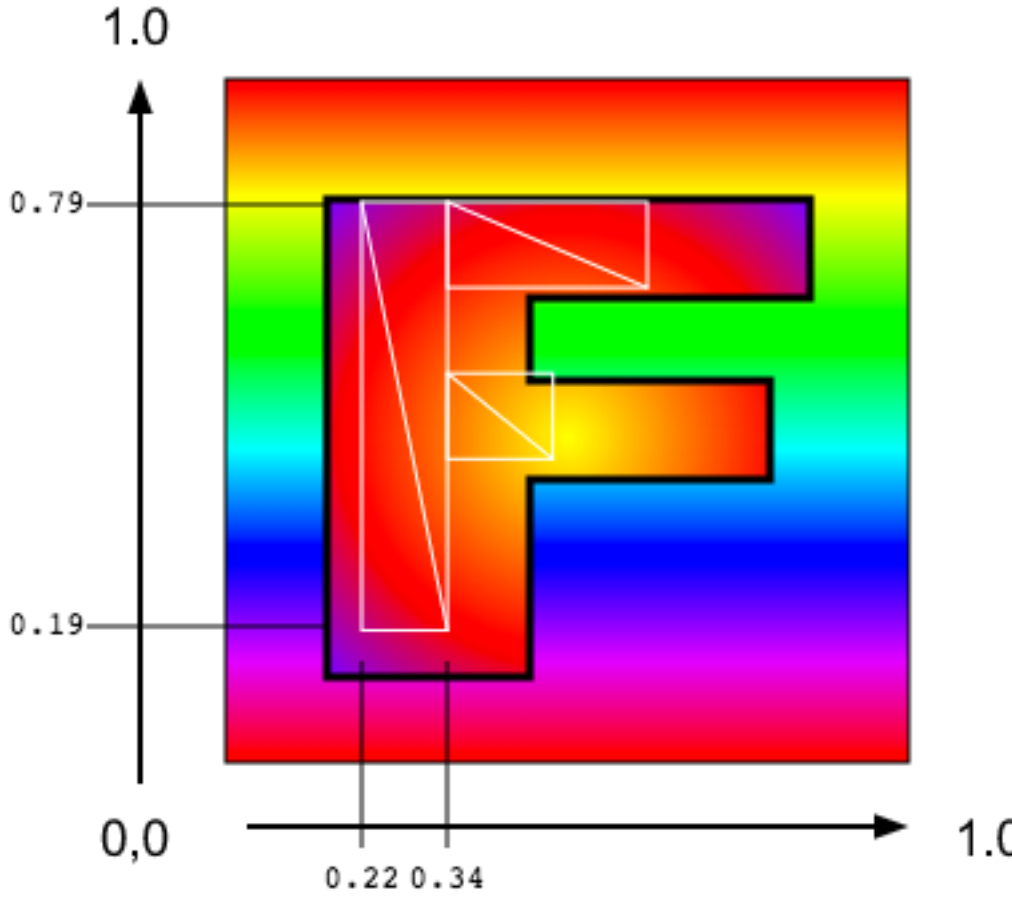
Textures

# Textures

- Texture coordinates are specified at each vertex of a given geometry
- We are drawing polygons
- These need to be tessellated
- Topology and geometry saves us
- Only need to redefine the triangle end points
- Geometry stays the same

# Textures



# Texture coordinate generation

Previous example nice for simple translation

How to map to real world


Pixel: (84, 111)
Coord: (-120, 45)

Pixel: (611, 417)
Coord: (-75, 20)

Interpolate on the geometries
world coordinates and find th
appropriate texture coordinat

# Homework 4

- Textures are very good, use your code from the in class challenge and textures to the Hw4 code.
- Add code to interpolate texture coordinates for each state
- Add one texture and reuse across all the geometries
- Each state will “cut” themselves out of the entire image.
- Try turning off some states

Hint: To open local files you can turn off Chrome file security with the command line option: `--args --disable-web-security`, just remember to close the browser when finished)

Or use Firefox ;)