UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4370 Fall 2023

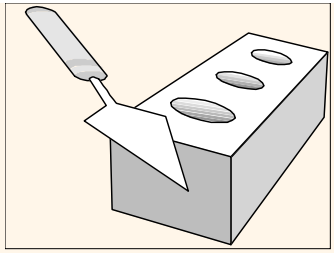
Interactive Computer Graphics

M & W 5:30 to 7:00 PM

Prof. Victoria Hilford

PLEASE TURN your webcam ON

NO CHATTING during LECTURE



COSC 4370

5:30 to 7

PLEASE

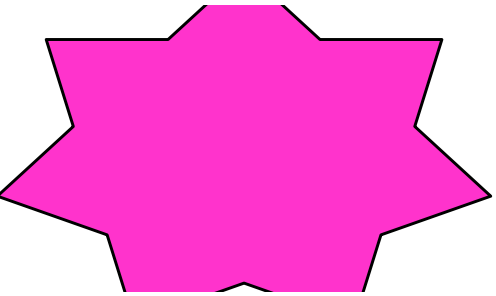
LOG IN

CANVAS

Please close all other windows.

NEXT.

10.04.2023 (W 5:30 to 7) (13)		Lecture 7 (From Vertices to Fragments)
10.09.2023 (M 5:30 to 7) (14)	Homework 6	PROJECT 2
10.11.2023 (W 5:30 to 7) (15)		EXAM 2 REVIEW
10.16.2023 (M 5:30 to 7) (16)		EXAM 2



CLASS PARTICIPATION - 15%

15% of Total

Class PARTICIPATION on Lecture 7

Not available until Oct 4 at 5:30pm | Due Oct 4 at 7pm | 100 pts

VH, publish

LECTURES

LECTURE 7 CLASS PARTICIPATION

Class Participation on Lecture 7.docx

VH, publish

Class PARTICIPATION on Lecture 7

Publish | Edit | ⋮

Download and complete this word document.

Class Participation on Lecture 7.doc

You will be prompted when to Upload completed document to CANVAS as **score.doc** (example 100.doc).

Warning:

TA, at random, will inspect the Uploaded document.

If you score is not honestly entered, you will get a zero.

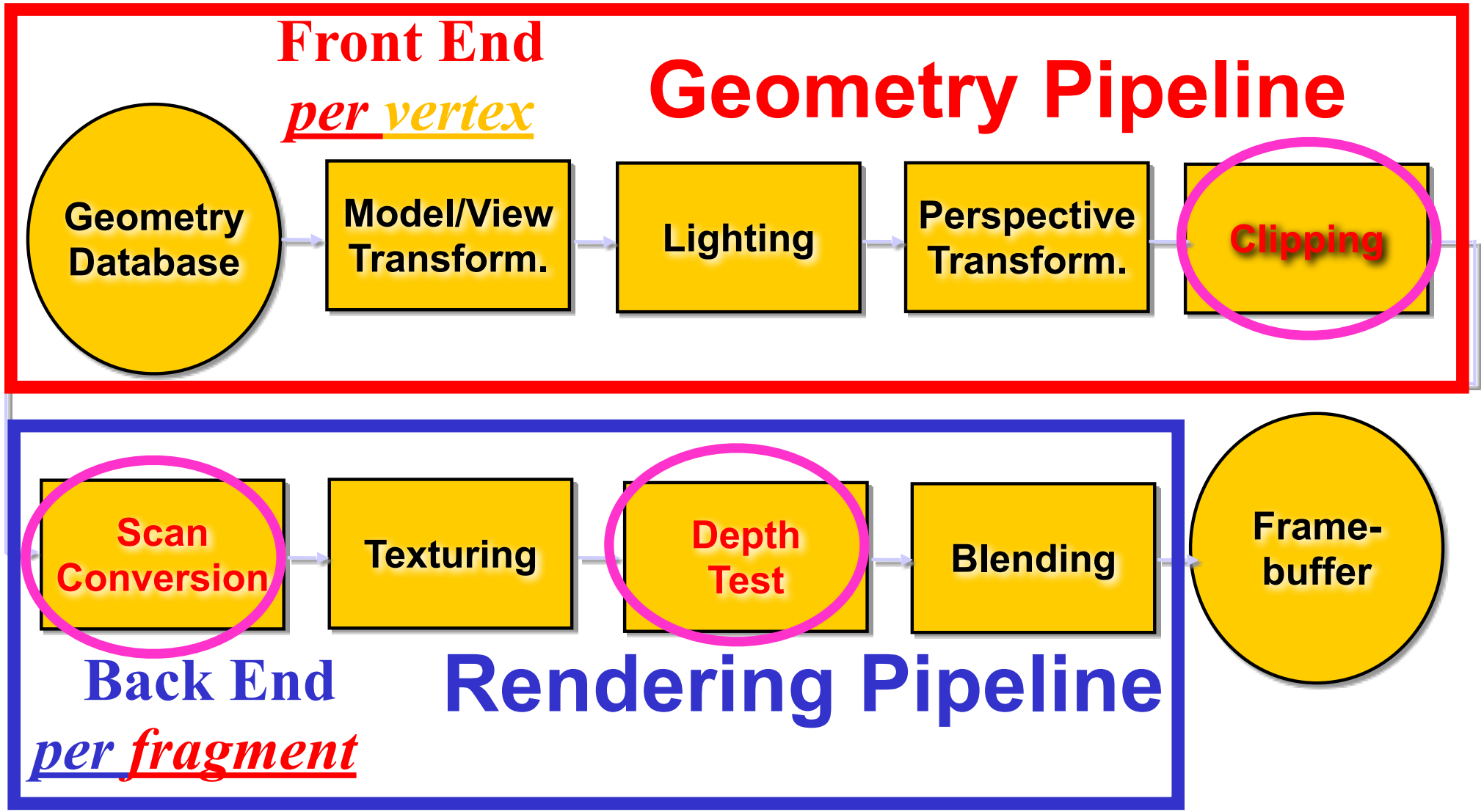
For the Grader: if the student did not submit, please skip and do not assign a ZERO (MISSING).

COSC 4370 – Computer Graphics

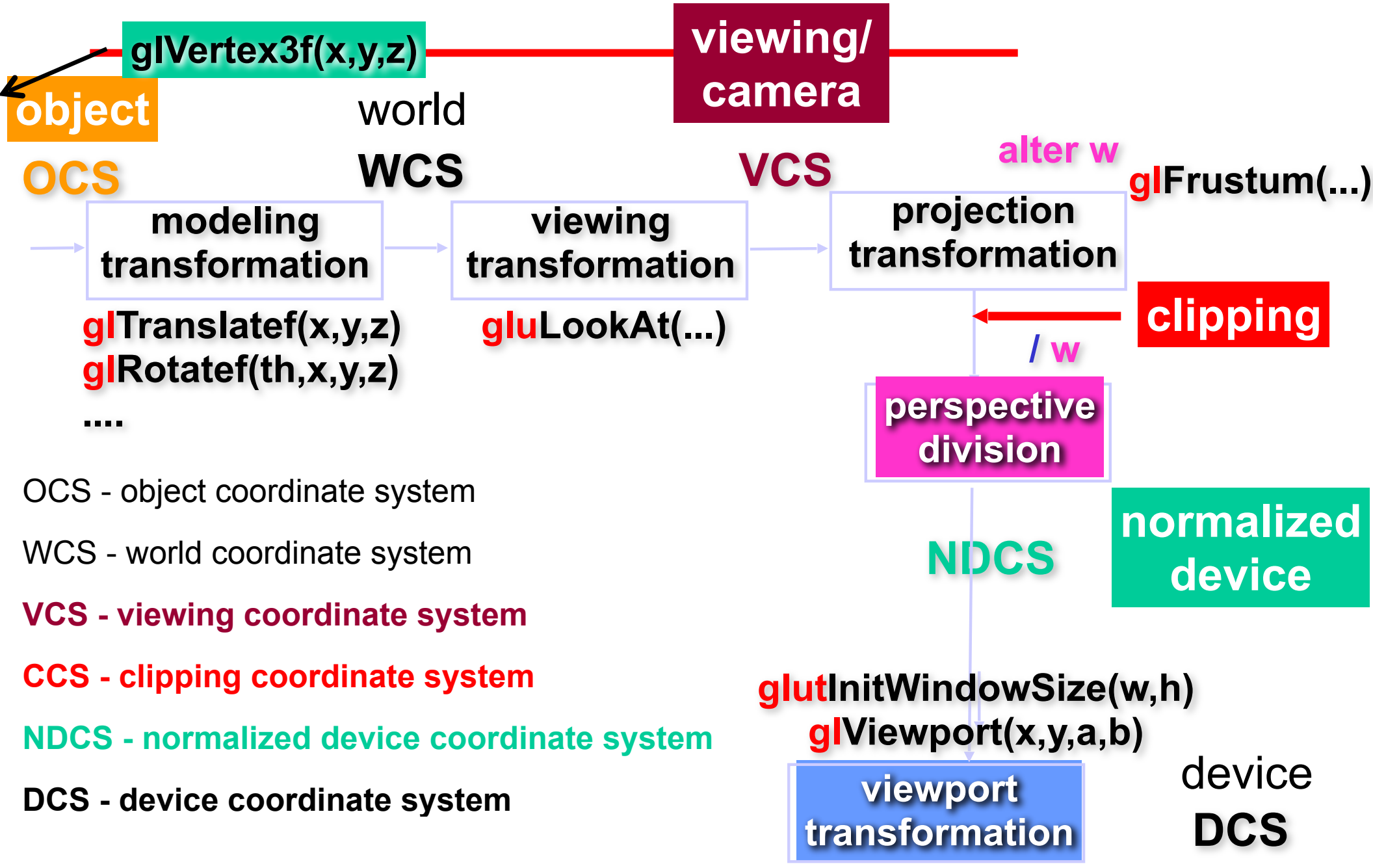
Lecture 7

From Vertices to Fragments
Chapter 7

The Graphics Pipeline



Projective Rendering Pipeline



OCS - object coordinate system

WCS - world coordinate system

VCS - viewing coordinate system

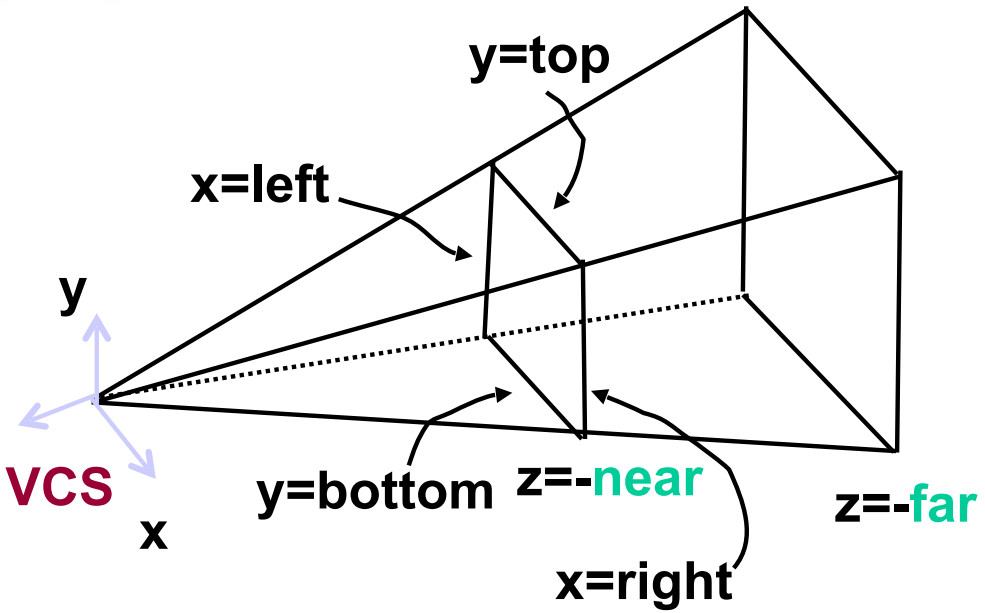
CCS - clipping coordinate system

NDCS - normalized device coordinate system

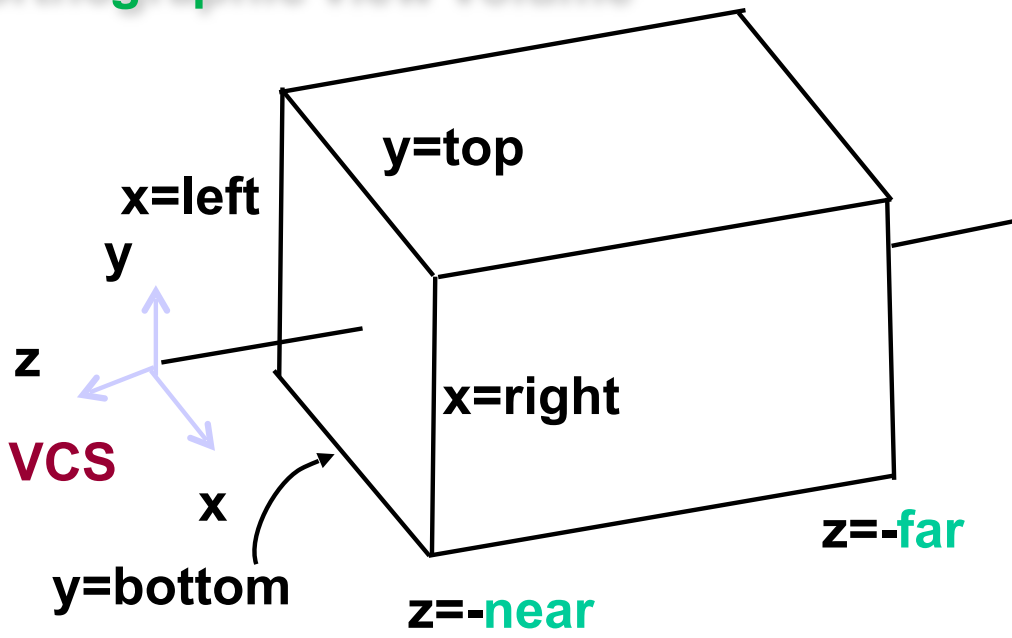
DCS - device coordinate system

Transforming View Volumes

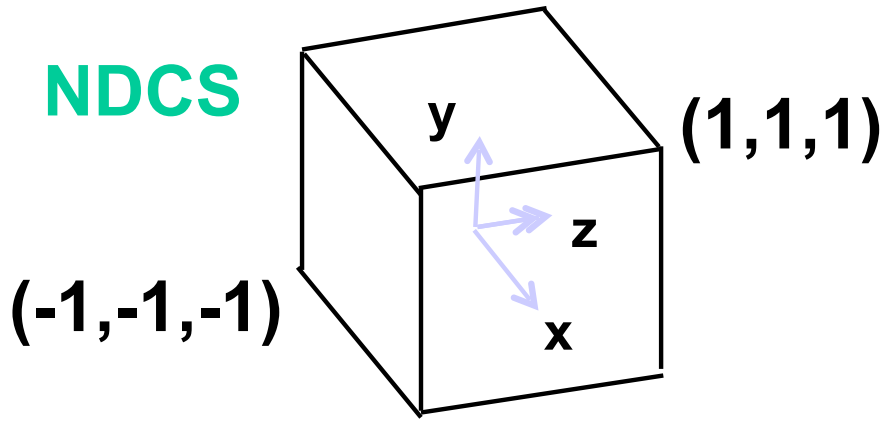
perspective view volume



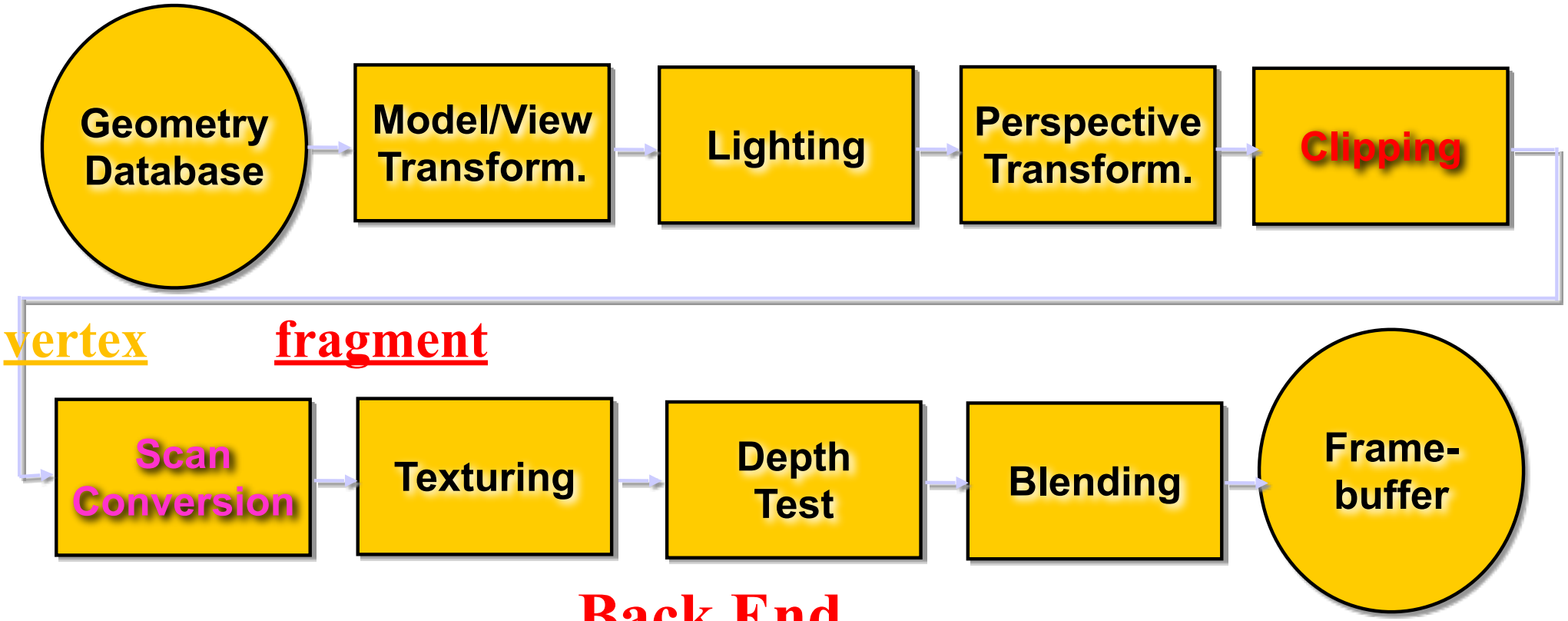
orthographic view volume



NDCS



The Rendering Pipeline



Back End

per pixel (fragment)

Objectives

- Introduce basic implementation strategies
- Clipping
- Scan conversion

Clipping

glFrustum(...)

alter w

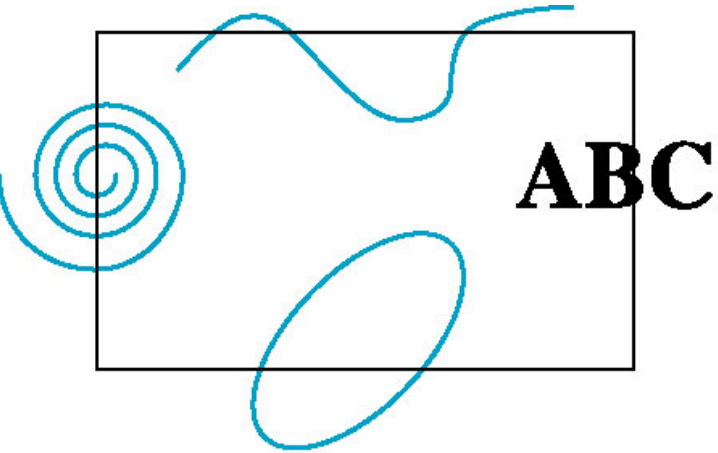
projection transformation

clipping

/ w

perspective division

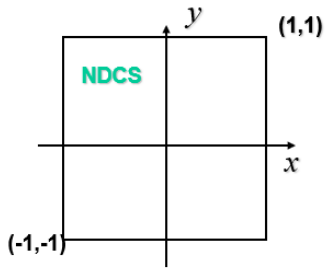
Clipping done **BEFORE** perspective division to **ELIMINATE** **UNNECESSARY** /w



NDCS

normalized device

Clipping window: Part of the scene we want to draw on the screen.



Clipping Points

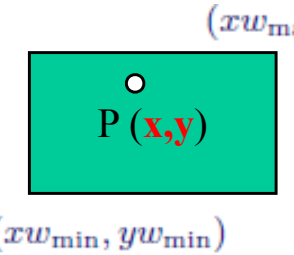
Clipping a point **(x, y)** is easy assuming that the **clipping window** is an axis aligned rectangle defined by $(xw_{\min}, yw_{\min})$ and $(xw_{\max}, yw_{\max})$

Keep point **(x, y)** if

$$xw_{\min} \le x \le xw_{\max}$$

$$yw_{\min} \le y \le yw_{\max}$$

otherwise **clip** the point.



Using **normalized coordinates**, we have

$(xw_{\min} = -1, yw_{\min} = -1)$ and

$(xw_{\max} = 1, yw_{\max} = 1)$

CLASS PARTICIPATION 1!
(Next Slide)

Name: _____

Total score: ☐

Class PARTICIPATION on Lecture 7.doc **ANSWER SHEET**

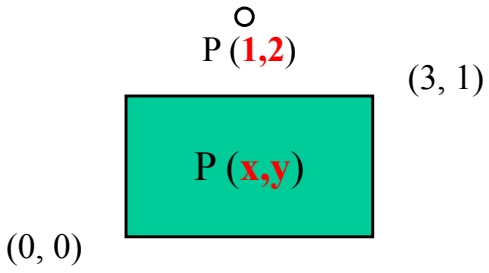
(Out of 100 points. Please record your own total score!)

(Attach as score.doc!)

1. (25 points) Is point (1,2) inside the clipping window with x coordinates min and max (0, 3) and y coordinates min and max (0,1)?

NO

Self Graded - correctly



Clipping 2D Line Segments

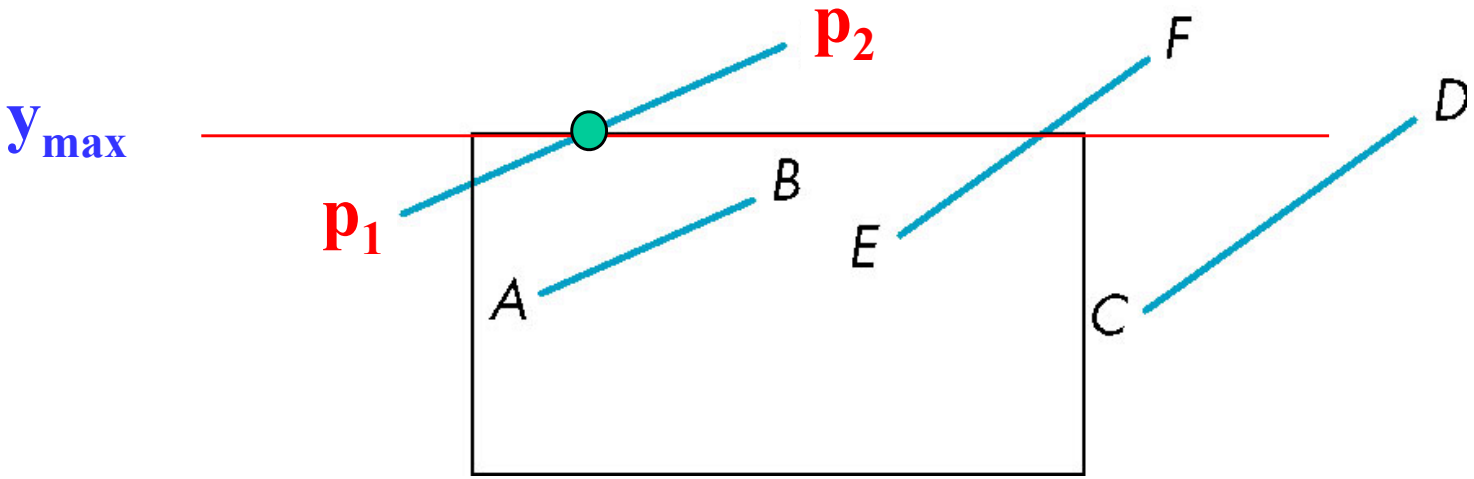
FUN !!!

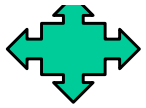
- Brute force approach: **compute intersections** with all sides of **clipping window**
  - Inefficient: one division **per intersection**

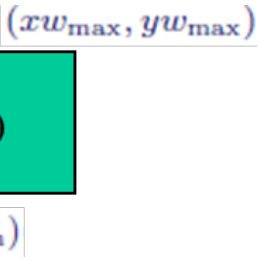
$$\alpha = (y2 - y1) \;/\; (y_{max} - y1)$$

$p_1 = (x_1, y_1)$

$p_2 = (x_2, y_2)$

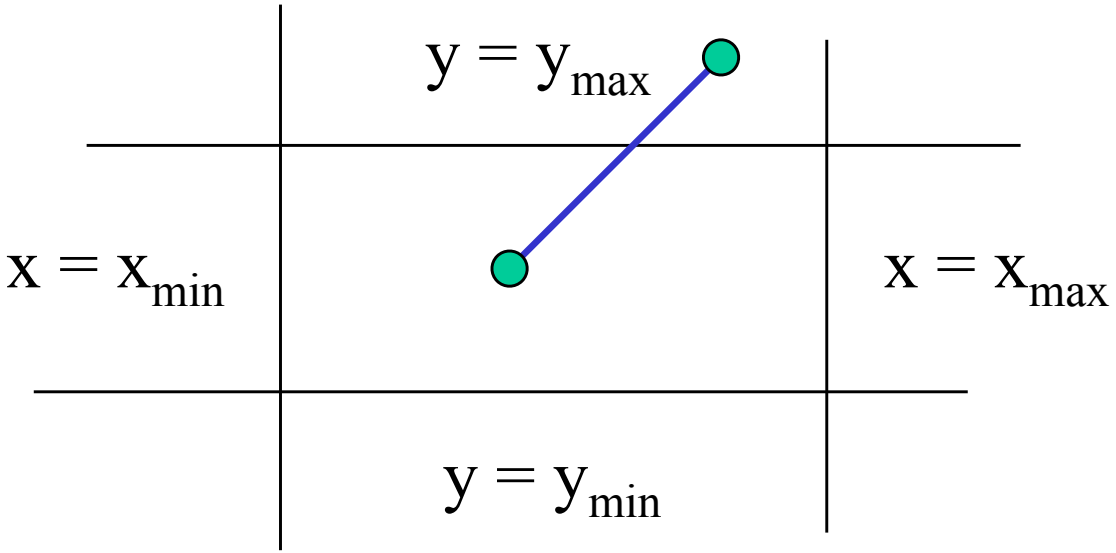


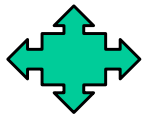




Cohen-Sutherland Algorithm

- Idea: eliminate as many cases as possible **without computing intersections**
- Start with four lines that determine the sides of the **clipping window**

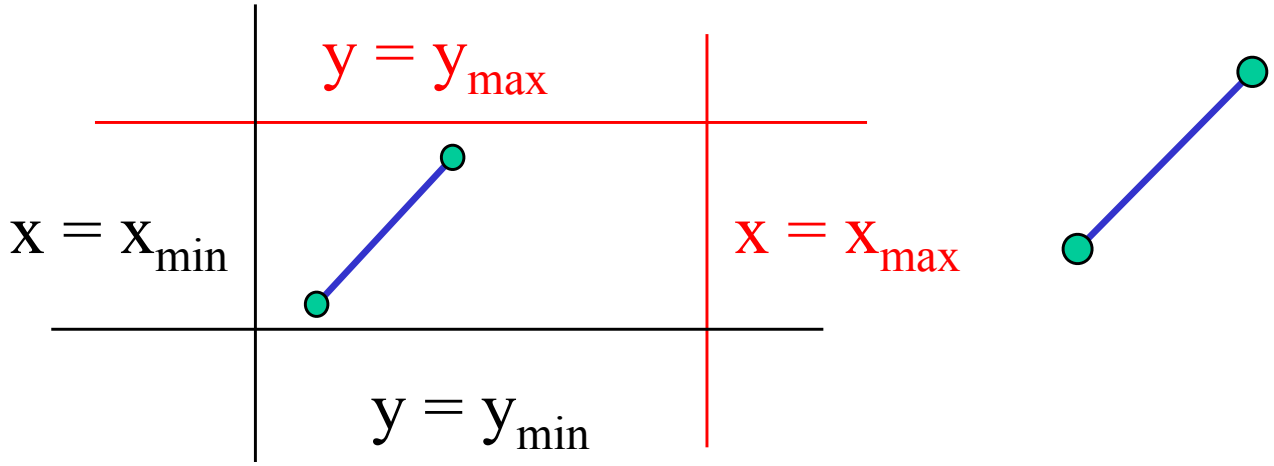




The Cases

Case 1: both endpoints of line segment inside all four lines

Draw (accept) line segment as is



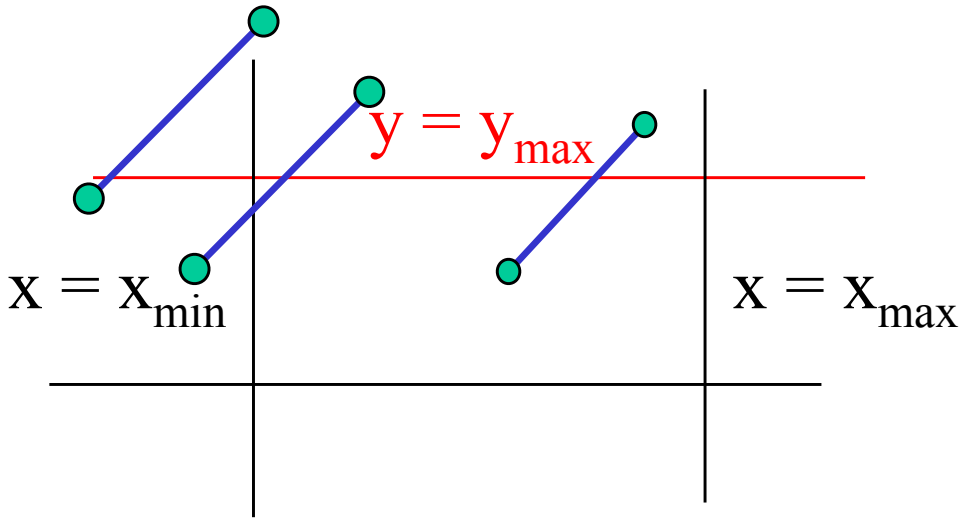
Case 2: both endpoints of line segment outside all lines an
on same side of a line

Discard (reject) the line segment

The Cases

- **Case 3**: One endpoint inside, one outside
  - **Must do at least one intersection**

- **Case 4**: Both outside
  - **May have part inside**
  - **Must do at least one intersection**

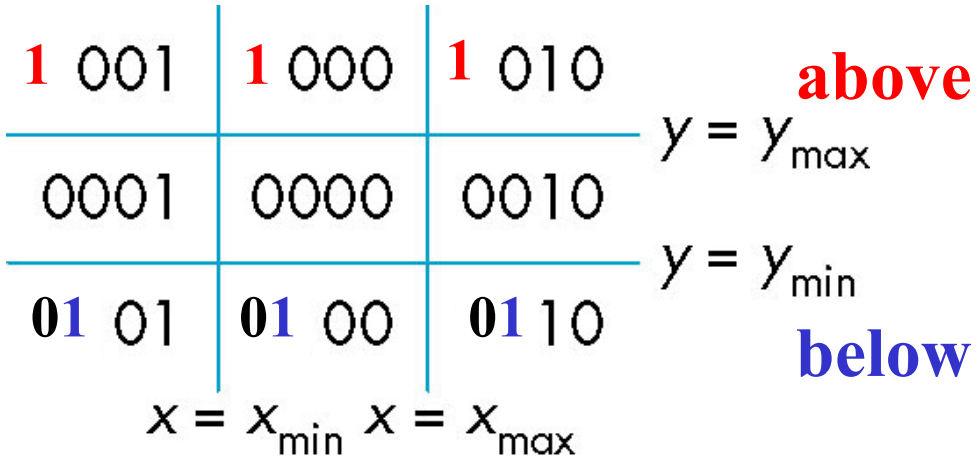


Defining Outcodes

- For each endpoint, define an outcode

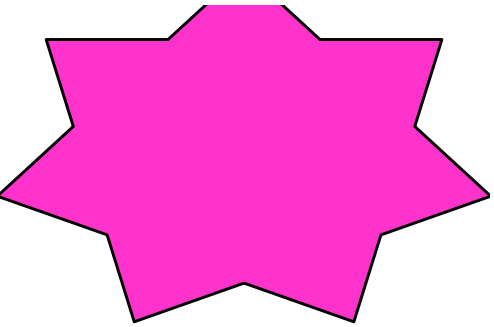
$$b_0 b_1 b_2 b_3$$

$b_0 = 1$ if $y > y_{max}$, 0 otherwise
$b_1 = 1$ if $y < y_{min}$, 0 otherwise
$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise



- Outcodes divide space into 9 regions
- Computation of outcode requires at **most 4 subtractions**

CLASS PARTICIPATION 2!
(Next Slide)

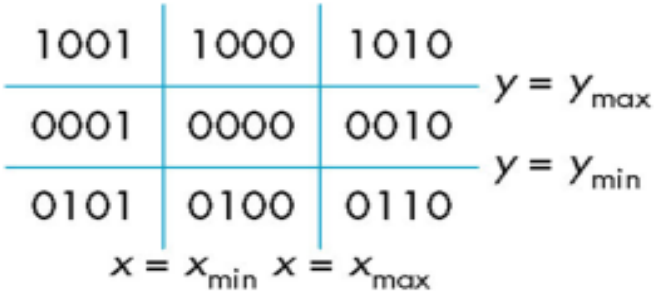


Defining Outcodes

- For each endpoint, define an outcode

$$b_0 b_1 b_2 b_3$$

$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise



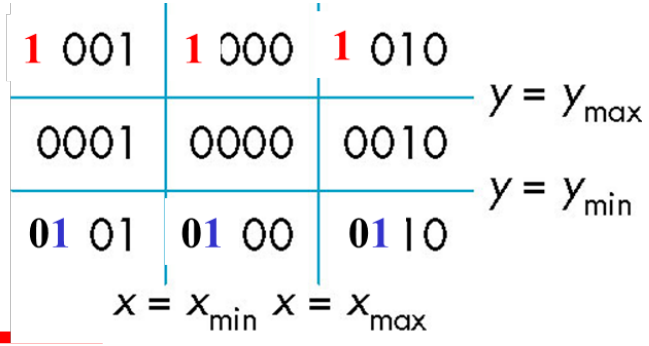
- Outcodes divide space into 9 regions
- Computation of outcode requires at **most 4 subtractions**

Self Graded - correctly

2. (25 points) Which **bit** is used for the **top part** of the clipping window?

b_0

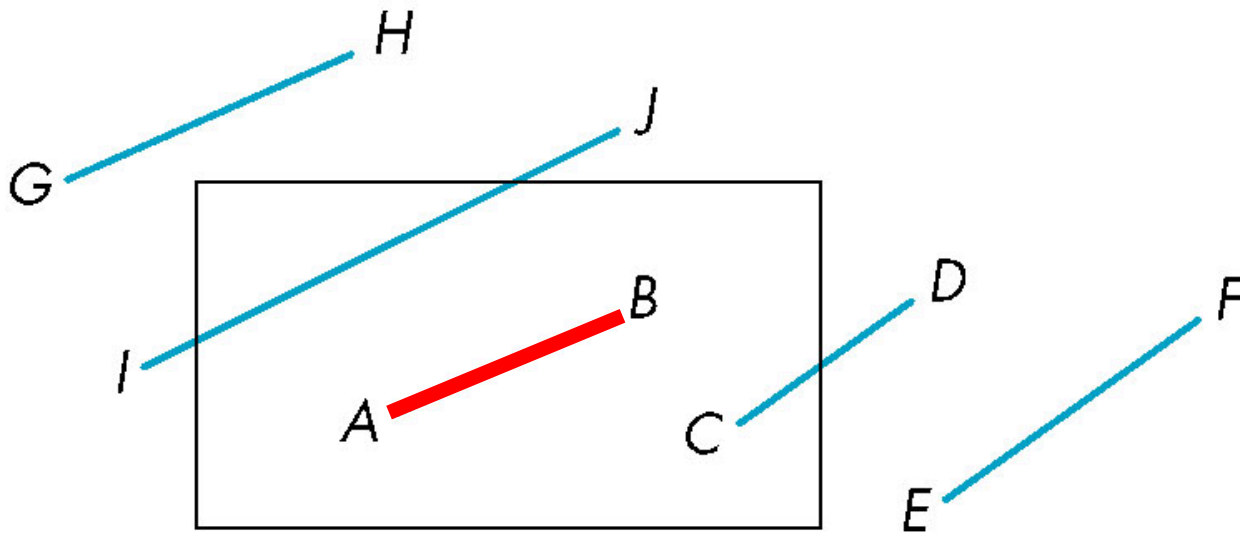
Using Outcodes



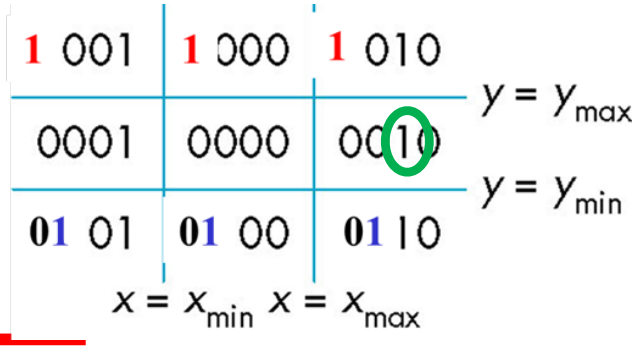
Consider the 5 cases below

- AB: outcode(A) = outcode(B) = 0

  Accept line segment



Using Outcodes

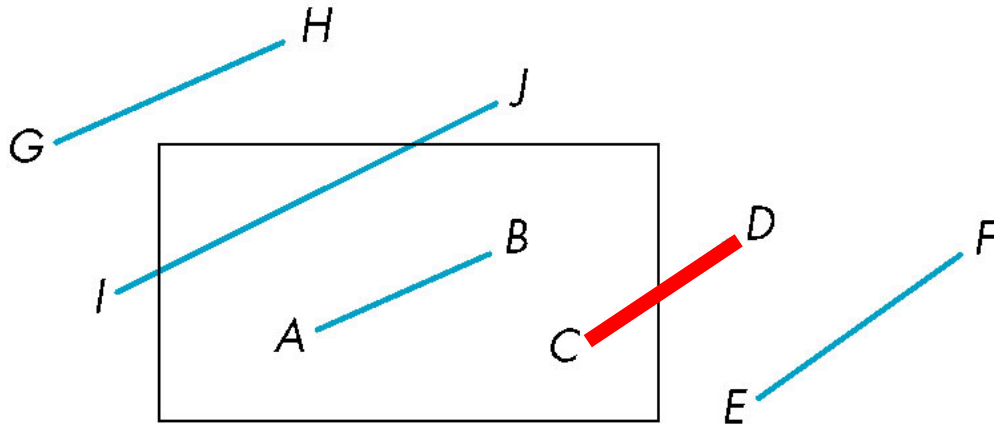


CD: outcode (C) = 0, outcode(D) ≠ 0

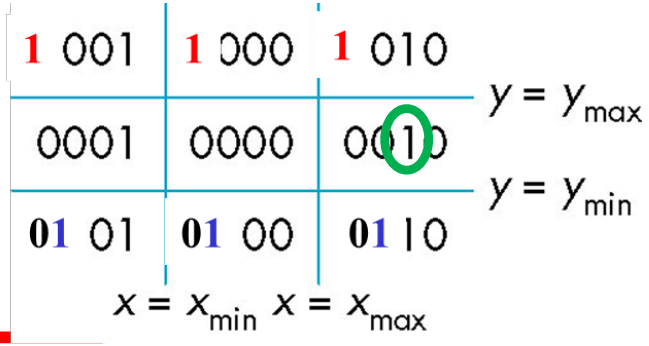
Compute intersection

Location of 1 in outcode(D) determines which edge to intersect with

Note if there were a segment from A to a point in a region with 2 ones in outcode, we might have to do **2 intersections**



Using Outcodes

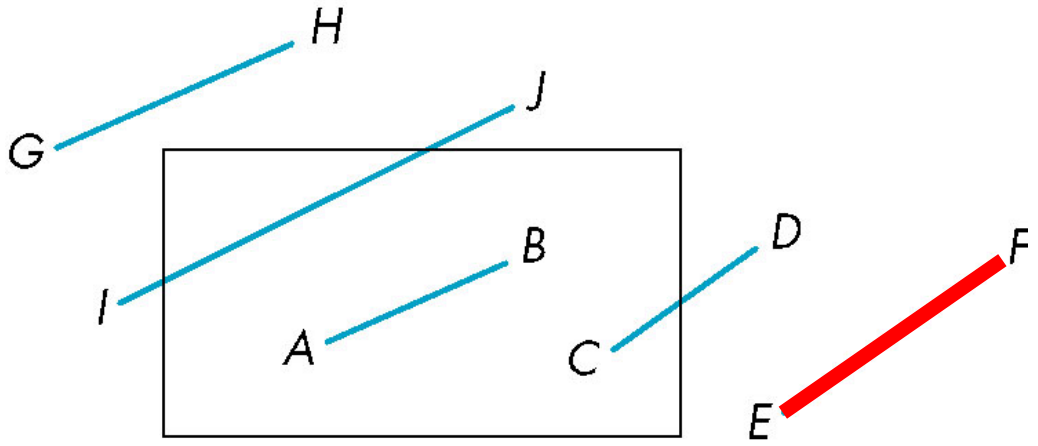


EF: outcode(E) logically **AND**ed with outcode(F) (bitwise) ≠ 0

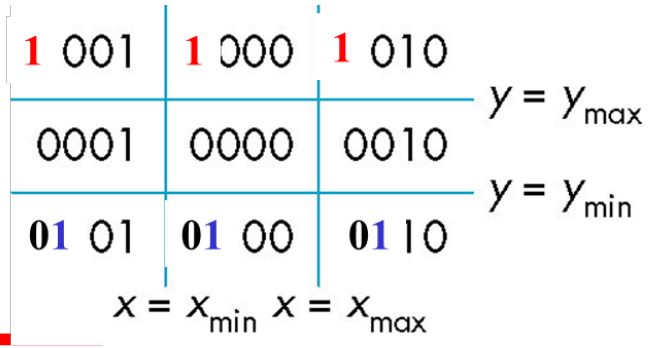
Both **outcodes** have a 1 bit in the same place

Line segment is outside of corresponding side of clipping window

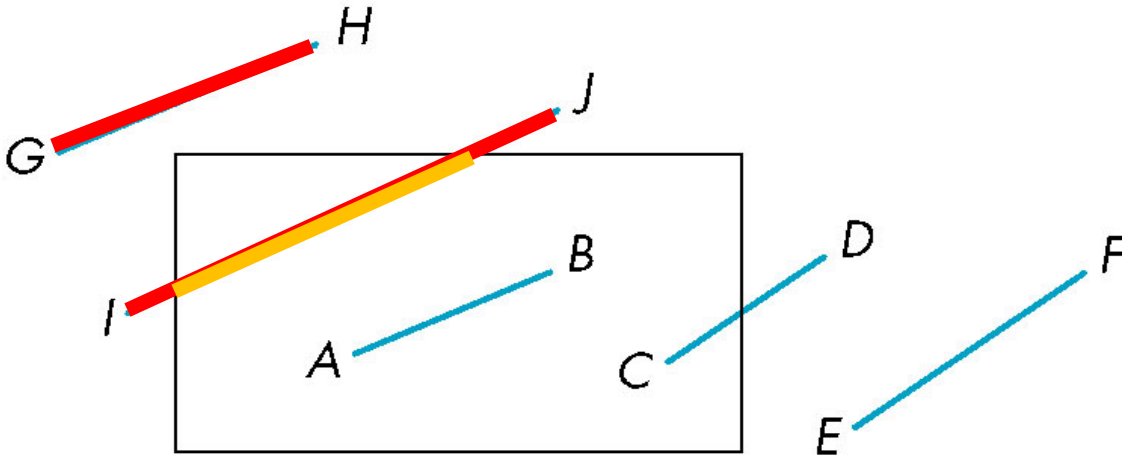
reject



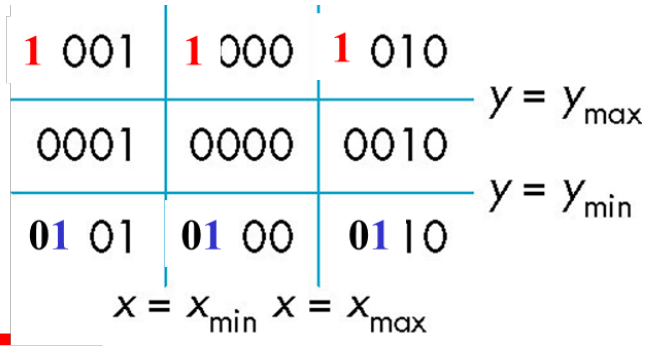
Using Outcodes



- **GH** and **IJ**: same outcodes, neither zero but logical **AND** yields zero
- **Shorten line segment by intersecting with one of sides of window**
- Compute **outcode** of **intersection (new endpoint of shortened line segment)**
- Re-execute algorithm



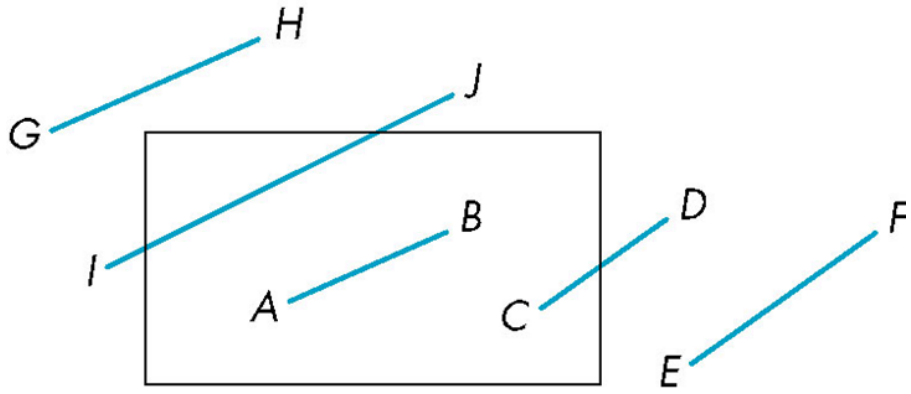
Efficiency



In many applications, the clipping window **is small** relative to the size of the entire database

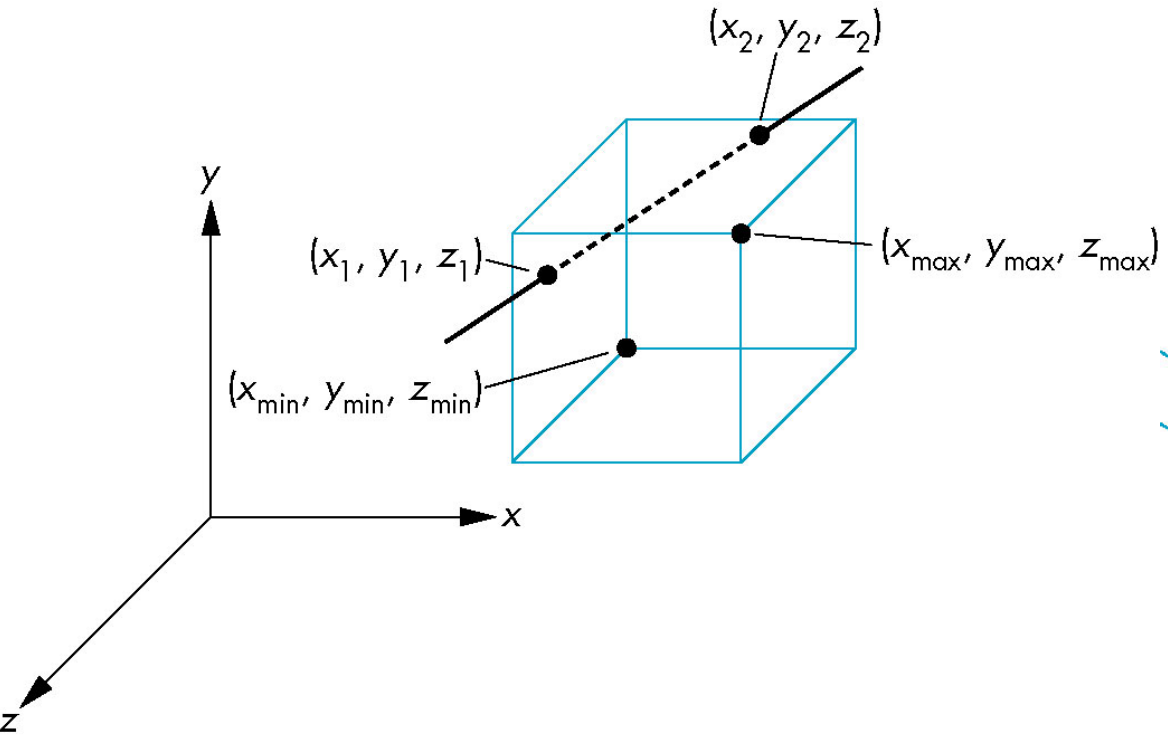
- **Most line segments are outside one or more side of the window** and can be eliminated based on their **outcodes**

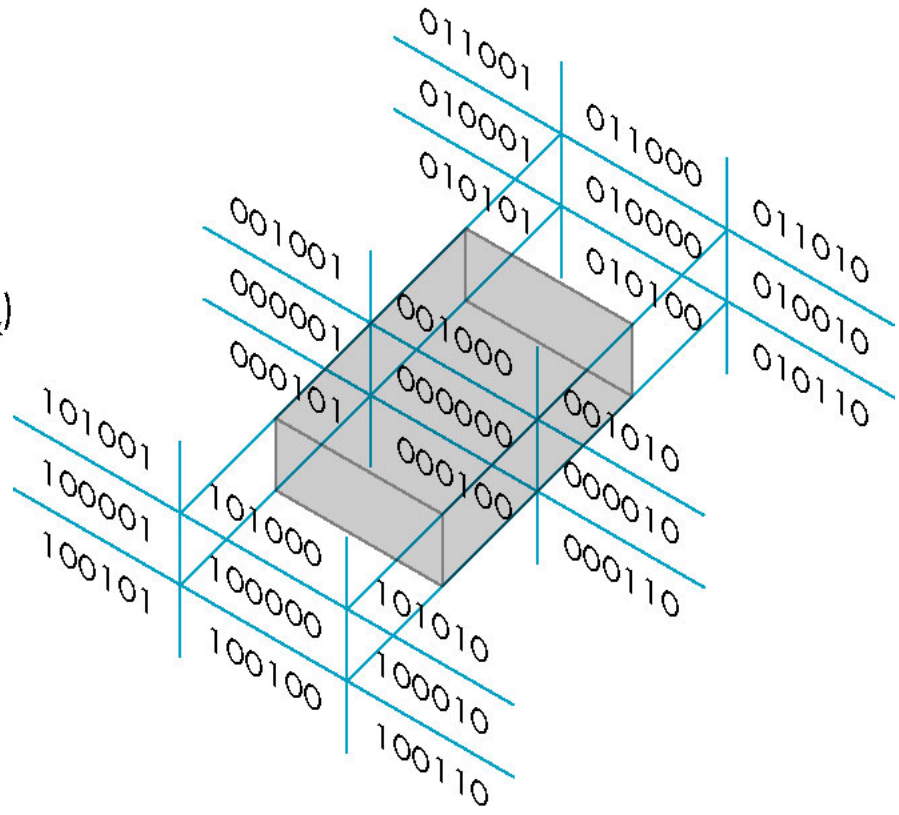
Inefficiency when code has to be **re-executed for line segments that must be shortened in more than one step**



Cohen Sutherland in 3D

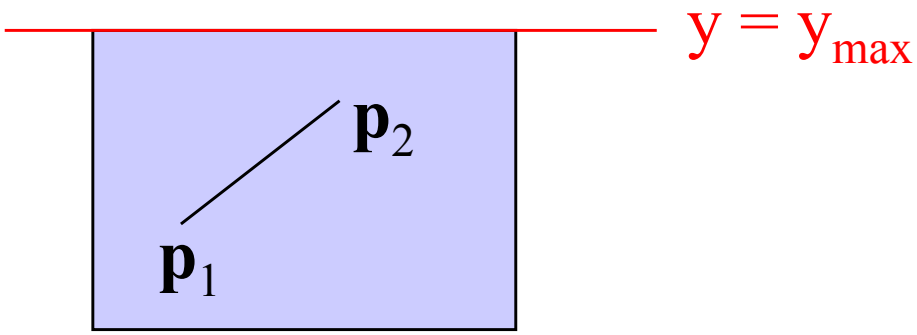
- Use 6-bit **outcodes** (6 faces!)
- When needed, **clip** **line segment** **against planes**





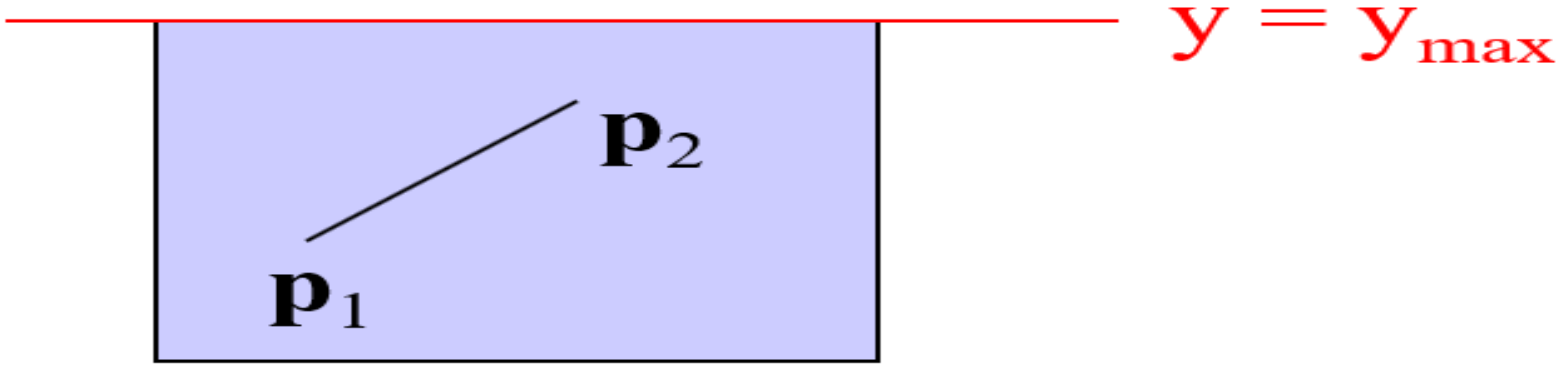
Liang-Barsky Clipping

- Consider the parametric form of a line segment



- We can distinguish between the cases by looking at the **ordering of the values** of α where the **line** determined by the **line segment crosses** the **lines that determine the window** (e.g., **$y = y_{max}$** for the top window edge)

CLASS PARTICIPATION 3!
(Next Slide)



3. (25 points) What is the **parametric** form of this **line segment**?

Self Graded - correctly

$$\mathbf{p}(\alpha) = (1-\alpha)\mathbf{p}_1 + \alpha\mathbf{p}_2 \qquad 1 \geq \alpha \geq 0$$

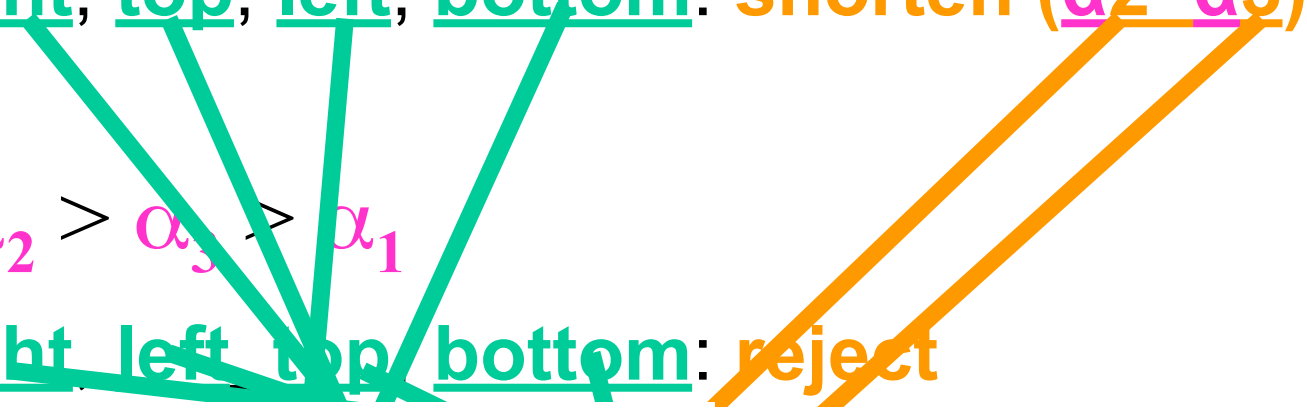
Liang-Barsky Clipping

- In (a): $\alpha_4 > \alpha_3 > \alpha_2 > \alpha_1$

  Intersect right, top, left, bottom: shorten (α_2 α_3)

- In (b): $\alpha_4 > \alpha_2 > \alpha_3 > \alpha_1$

  Intersect right, left, top, bottom: reject

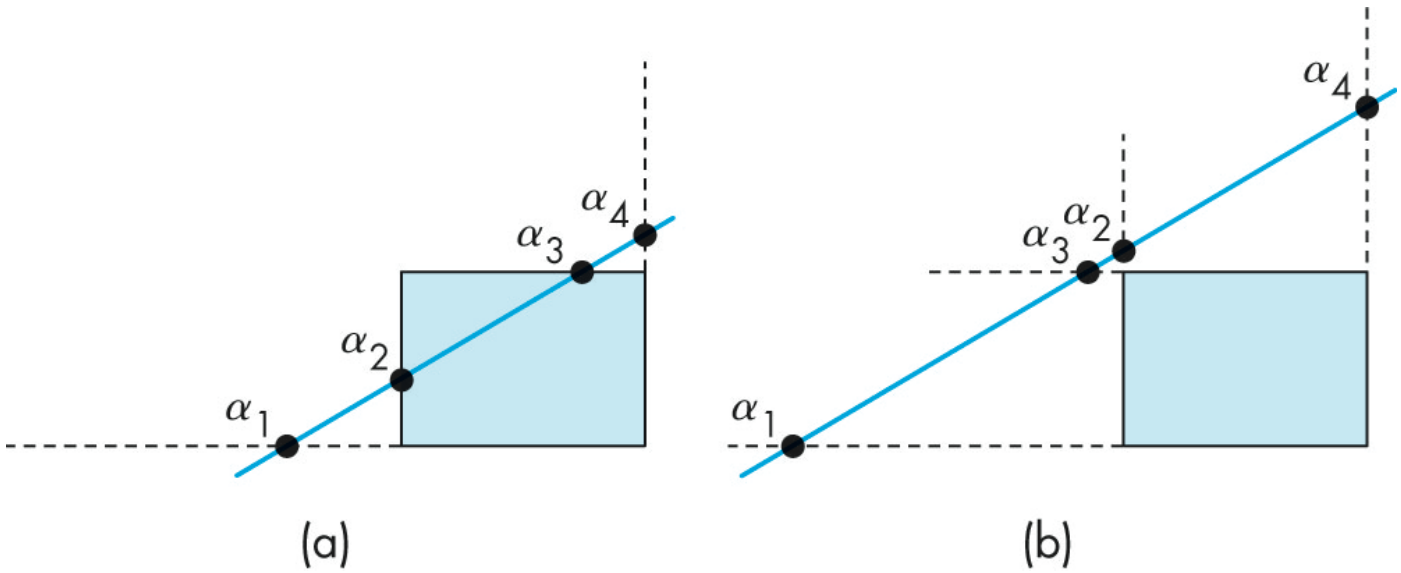


Liang-Barsky Advantages

Can accept/reject **as easily** as with Cohen-Sutherland

Using values of α, **we do not have to use** algorithm **recursively as with** Cohen-Sutherland

Extends to 3D



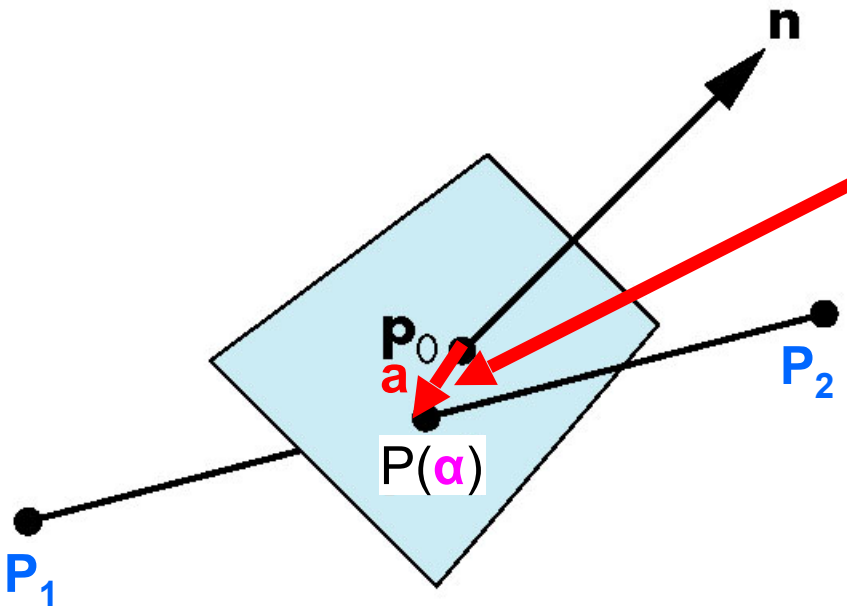
Plane-Line Intersections

Line segment between P_1 and P_2 parametric **α form**

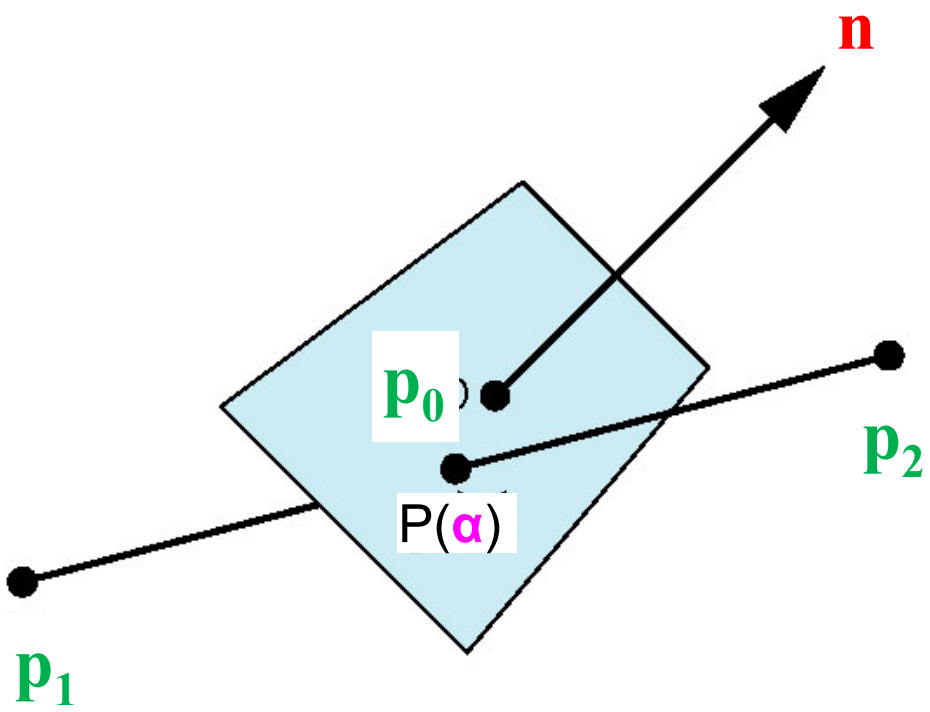
$$P(\alpha) = P_1 + \alpha\,(P_2 - P_1)$$

Plane equation **n** ● **a** = 0 **n** ● $(P(\alpha) - P_0) = 0$

What is **α** for the **point of intersection?**



Plane-Line Intersections



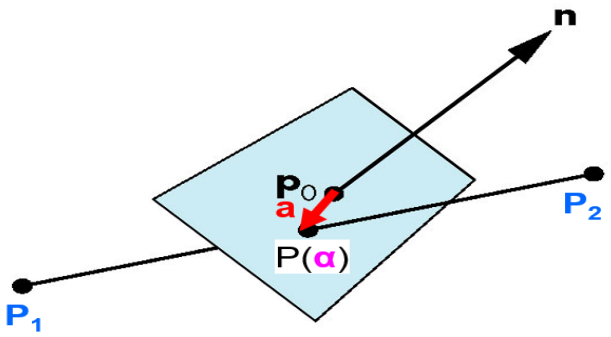
CLASS PARTICIPATION 4!
(Next Slide)

Plane-Line Intersections

Line segment between P_1 and P_2 parametric α form

$$P(\alpha) = P_1 + \alpha (P_2 - P_1)$$

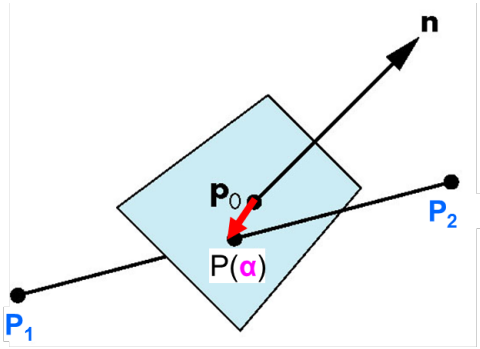
Plane equation $\mathbf{n} \bullet \mathbf{a} = 0$ $\qquad$ $\mathbf{n} \bullet (P(\alpha) - P_0) = 0$



4. (25 points) What is value of α at the Plane-Line intersection?

Self Graded - correctly

Plane-Line Intersections



$P(\alpha) = P_1 + \alpha (P_2 - P_1)$

$n \bullet (P(\alpha) - P_0) = 0$

What is **α** for the **point of intersection?**

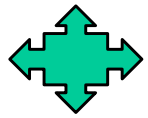
$n \bullet (P_1 + \alpha (P_2 - P_1) - P_0) = 0$

α

$n \bullet ((P_1 - P_0) + \alpha (P_2 - P_1)) = 0$

$n \bullet (P_1 - P_0) + \alpha\, n \bullet (P_2 - P_1)) = 0$

$n \bullet (P_0 - P_1) = \alpha\, n \bullet (P_2 - P_1))$



Objectives

- Introduce **clipping** algorithms for polygons

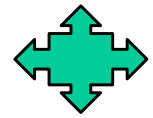
- Survey hidden-surface algorithms

Objectives

- Introduce **clipping** algorithms for polygons

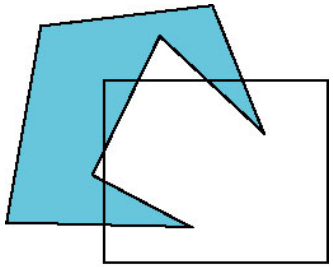
Chapter 7.5

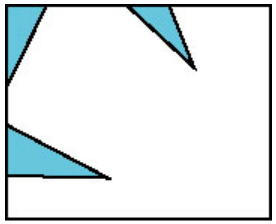
- Survey hidden-surface algorithms



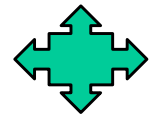
Polygon Clipping

- 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**

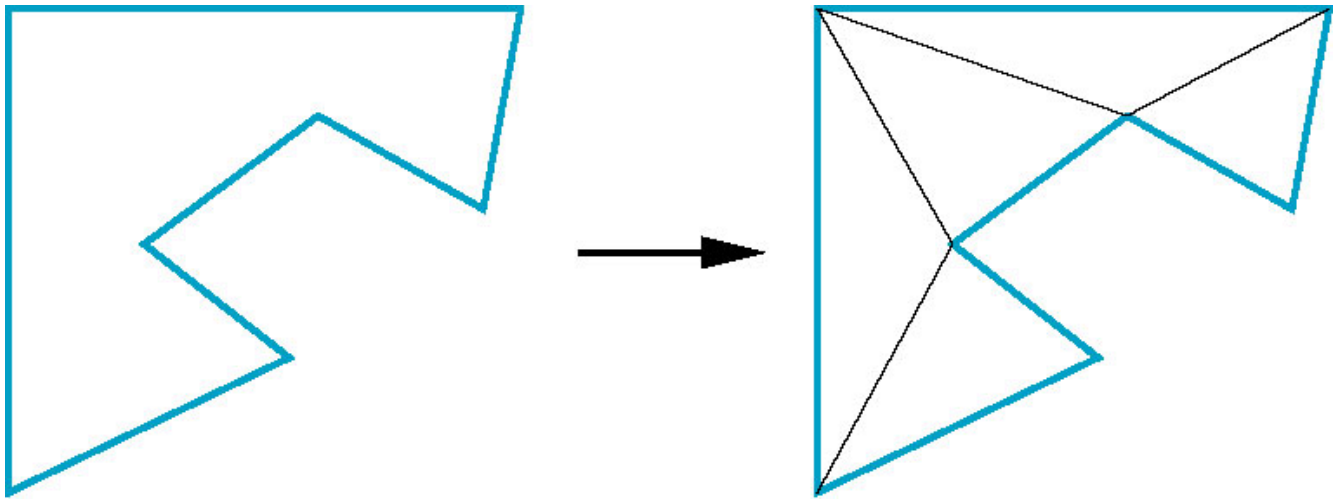


Tessellation and Convexity

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

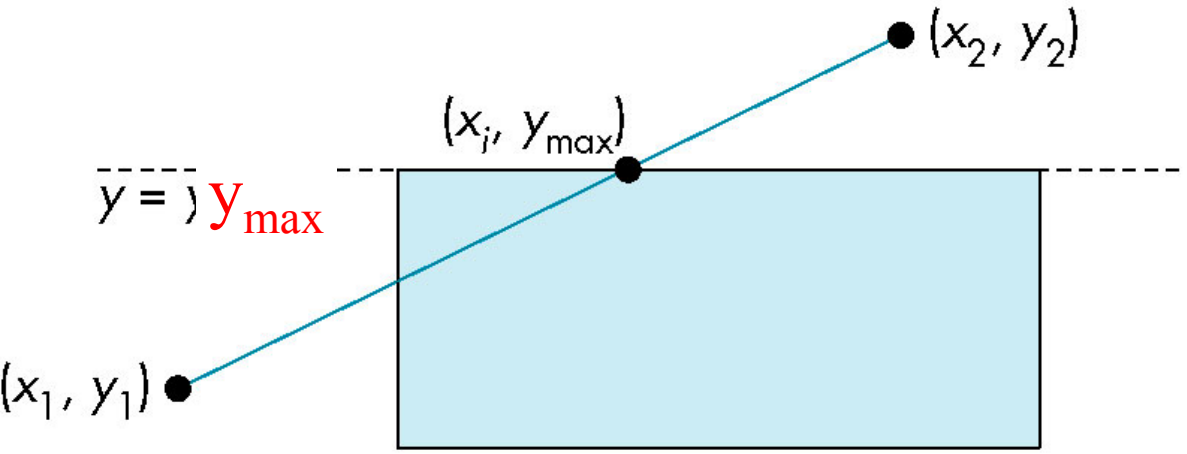
Also makes **fill** easier

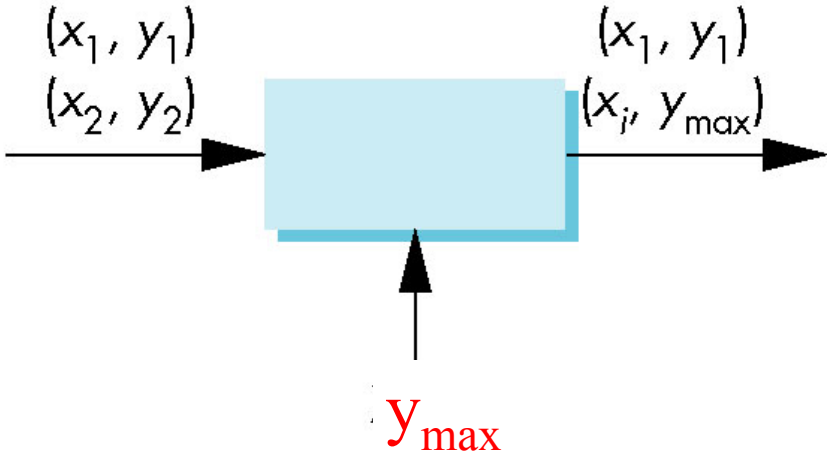
Tessellation code in GLU library



Clipping as a Black Box

Can consider **line segment clipping** as a process that takes in **two vertices** (x_1, y_1) & (x_2, y_2) and produces **either no vertices or the vertices of a clipped line segment** (x_1, y_1) & (x_i, y_{max})

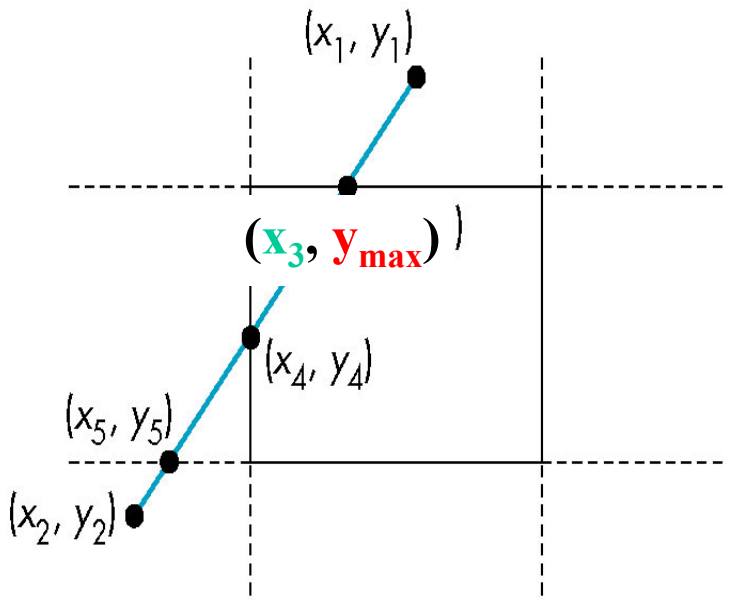




Pipeline Clipping of Line Segments

- **Clipping against each side of window is independent of other sides**

Can use **four independent clippers** in a pipeline



$x_3 = x(\alpha) = x1 + \alpha\,(x_2 - x_1)$

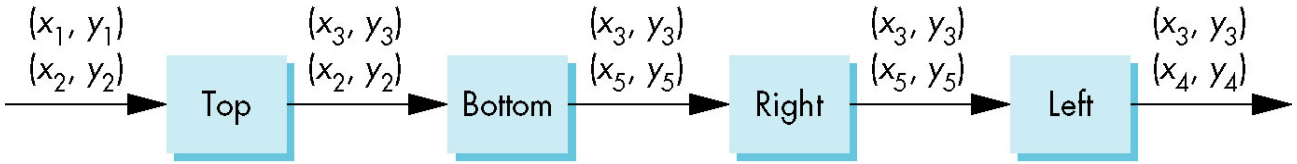
$y_3 = y_{max} = y(\alpha) = y_1 + \alpha\,(y_2 - y_1)$

$\alpha = (y_2 - y_1) / (y_{max} - y_1)$

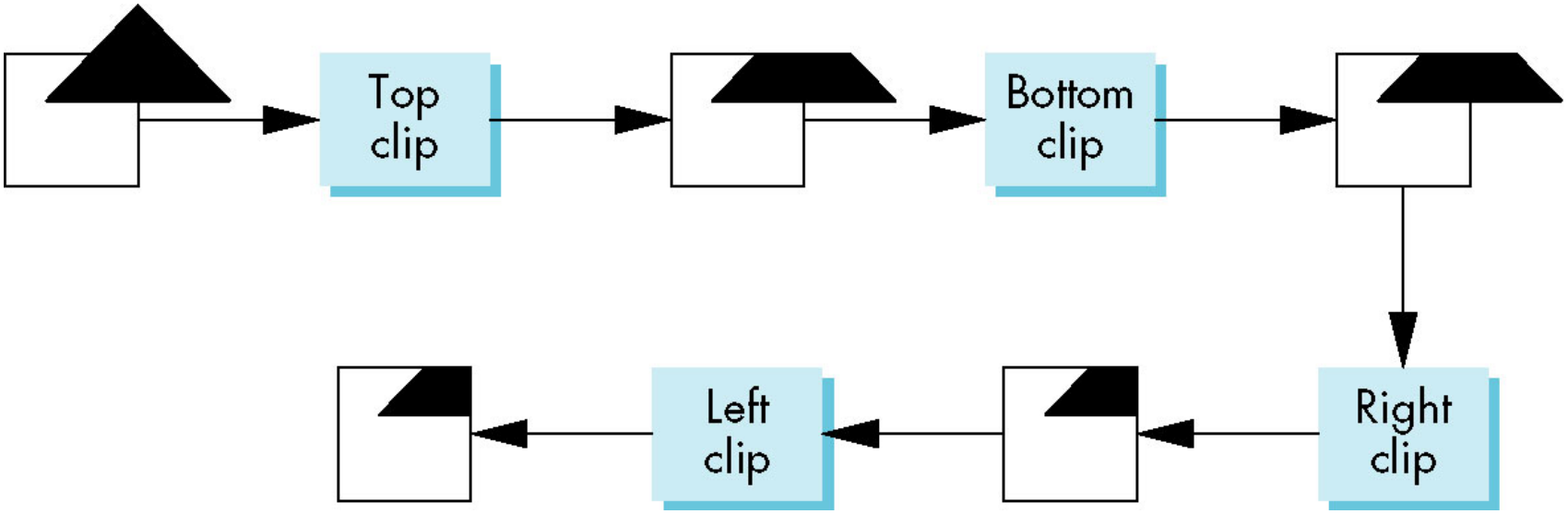
Knowing α allows us to get x coordinate of the intersection point:

$x_3 = x_1 + \alpha\,(x_2 - x_1) =$

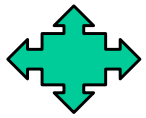
$= x_1 + (y_2 - y_1) / (y_{max} - y_1)\,(x_2 - x_1)$



Pipeline Clipping of Polygons



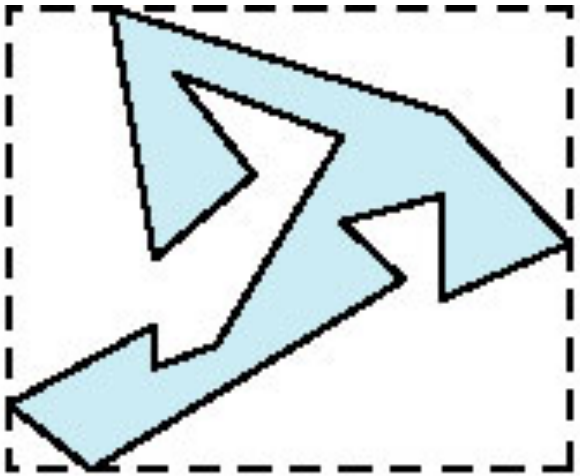
- **Three dimensions**: add front and back clippers
- Strategy used in **SGI Geometry Engine**
- Small increase in latency

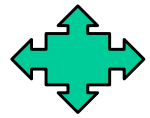


Bounding Boxes

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

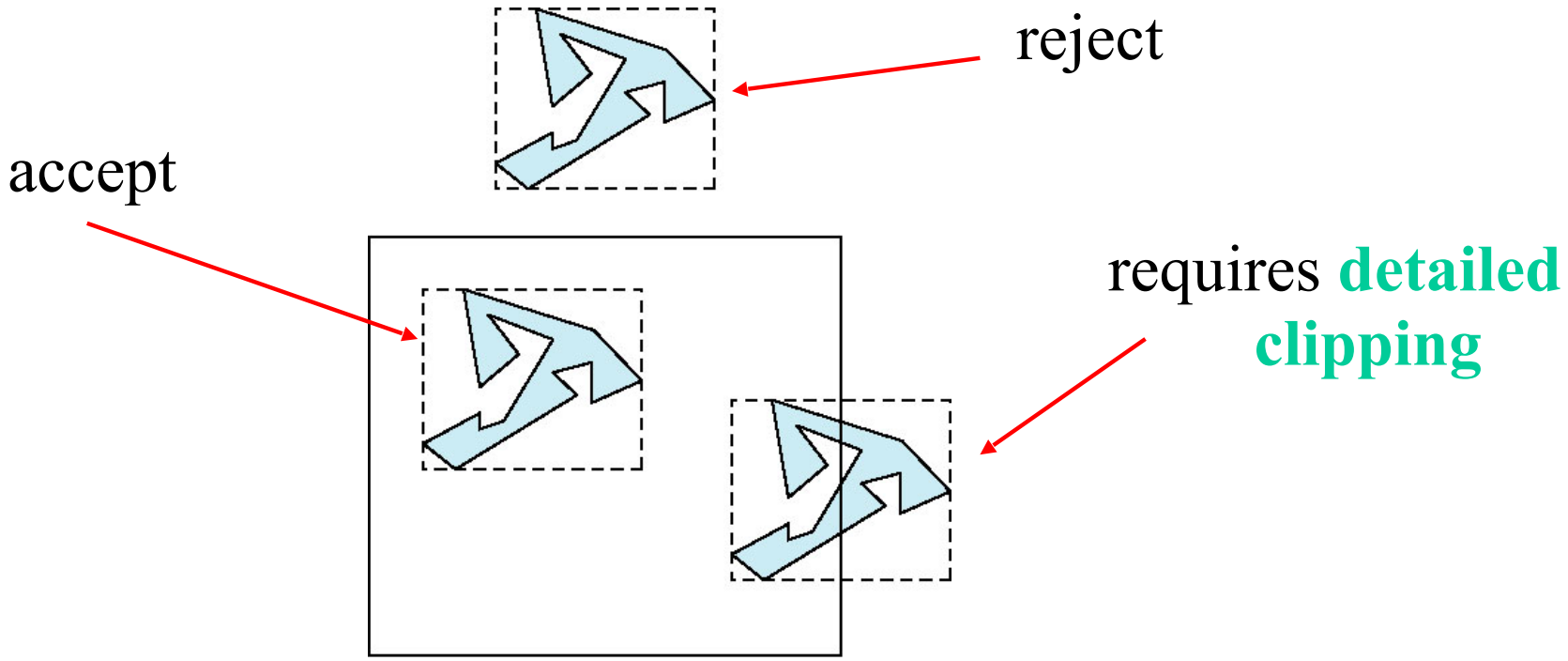
- Smallest rectangle aligned with axes **that encloses the polygon**
- Simple to compute: **max** and **min** of **x** and **y**





Bounding boxes

Can usually determine **accept**/**reject** based only on **bounding box**

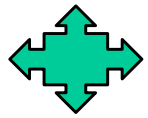


Objectives

- Introduce clipping algorithms for polygons

- Survey hidden-surface algorithms

Chapter 7.11

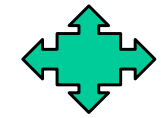


Clipping and Visibility

Clipping has much in common with hidden-surface removal

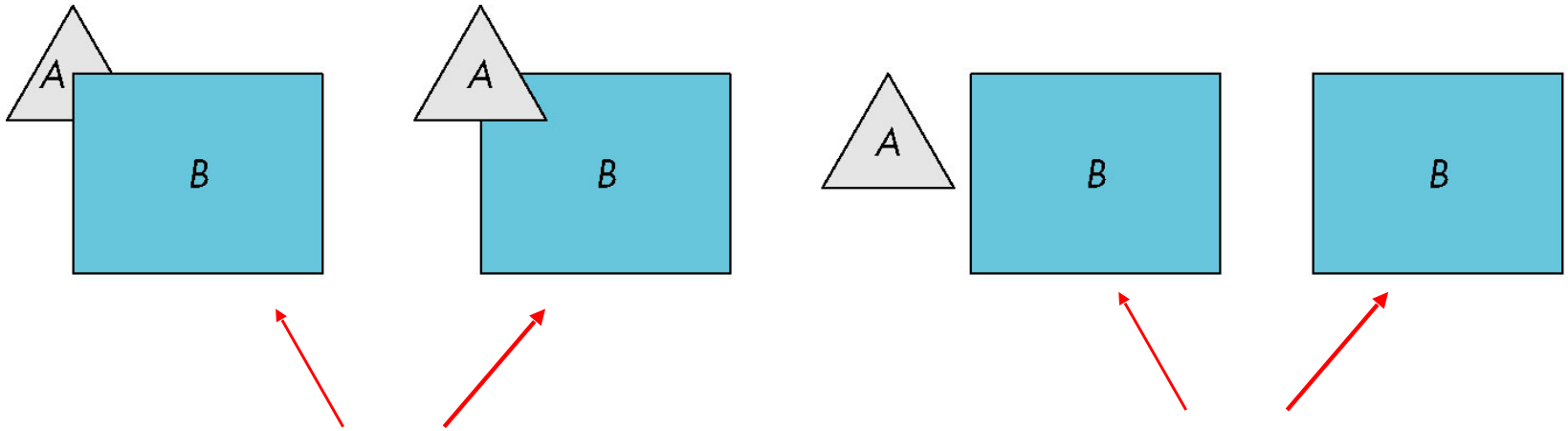
In both cases, we are trying to **remove objects** that are not visible to the **camera**

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

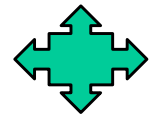
- **Object-space** approach: use pairwise testing between **polygons** (**objects**)



partially obscuring

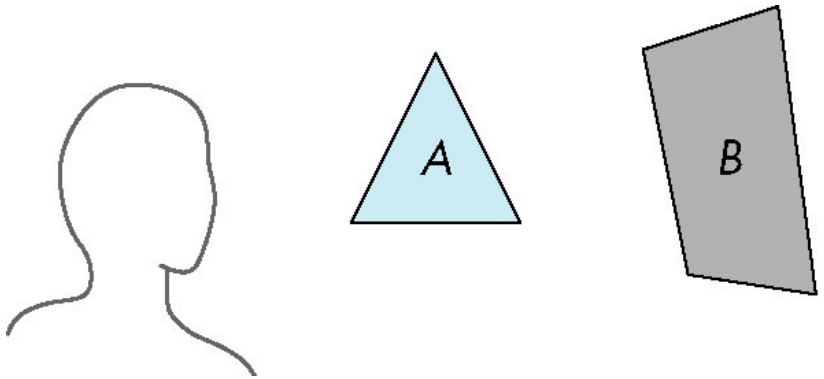
can draw independently

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

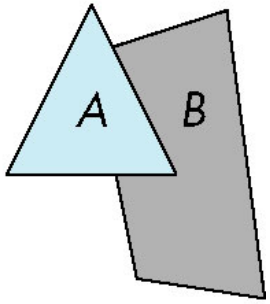


Painter's Algorithm

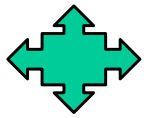
- **Render polygons in a** back to front order so that **polygons** behind others are simply **painted over**



B behind A **as seen by viewer**



Fill B then A



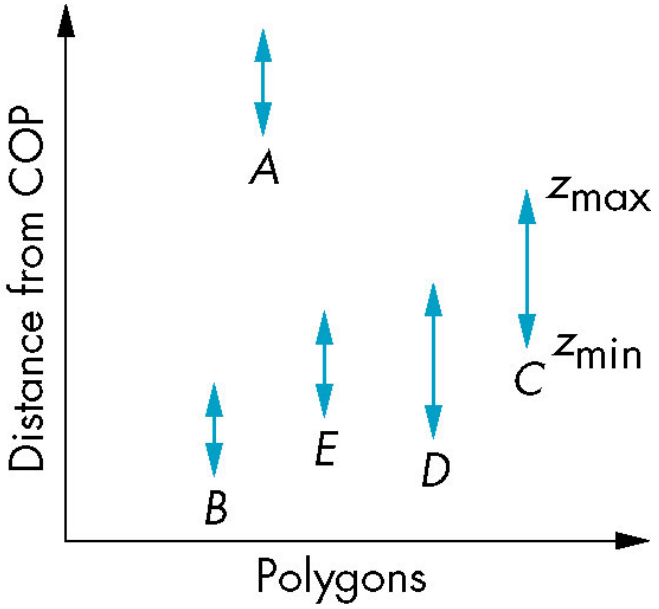
Depth Sort

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

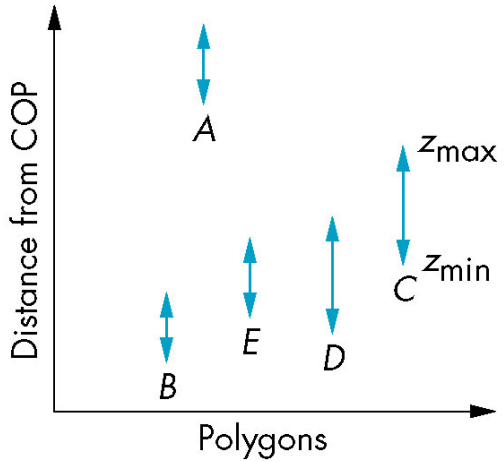
Polygons sorted by distance from COP



Easy Cases

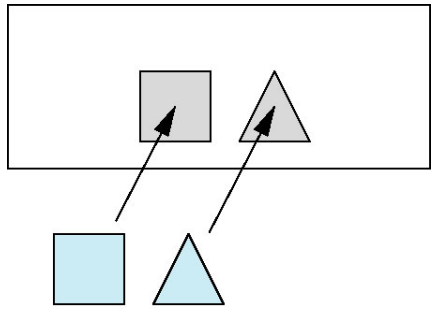
- **A** lies behind **all other polygons**

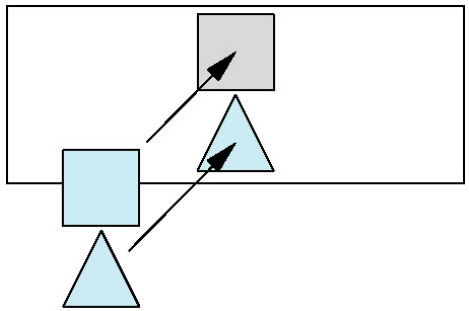
  Can **render**



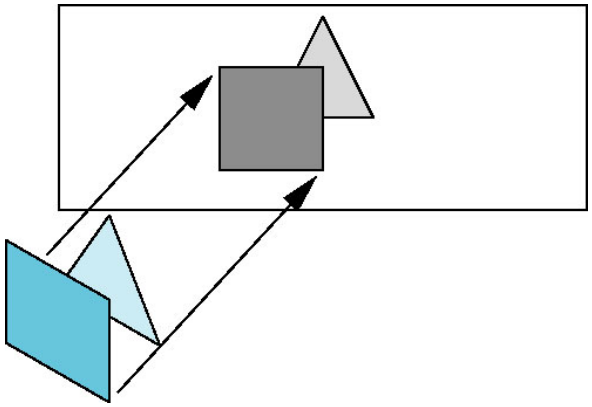
- **Polygons** overlap in **z** but not in either x or y

  Can **render independently**

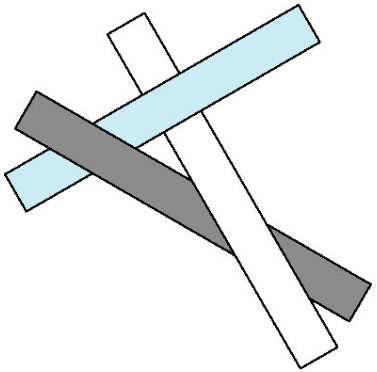




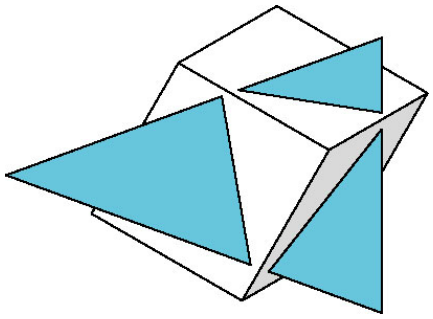
Hard Cases



Overlap in all directions
but one is fully on
one side of the **other**



cyclic overlap



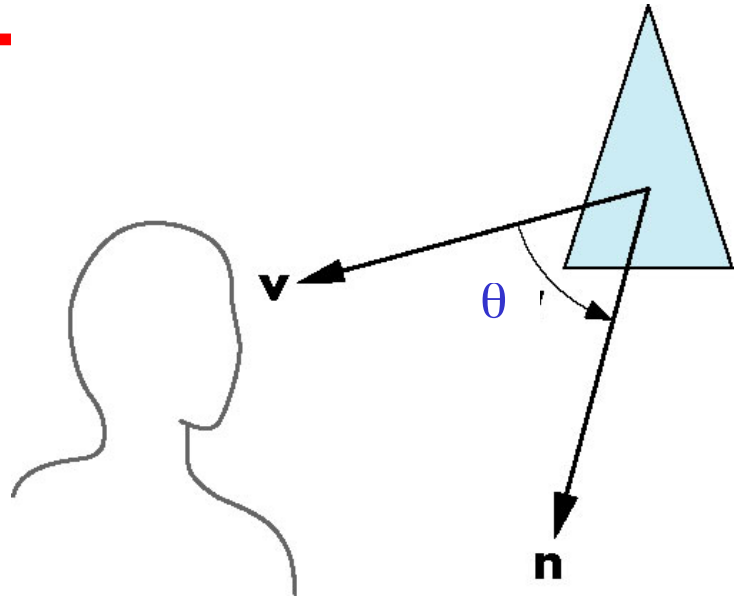
penetration

Back-Face Removal (Culling)

face is visible iff $90 \geq \theta \geq -90$

equivalently $\cos \theta \geq 0$

or $\mathbf{v} \cdot \mathbf{n} \geq 0$



plane of face has form $ax + by + cz + d = 0$

but after normalization $\mathbf{n} = (0\ 0\ 1\ 0)^T$

need only test the **sign of c**

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

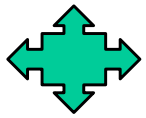
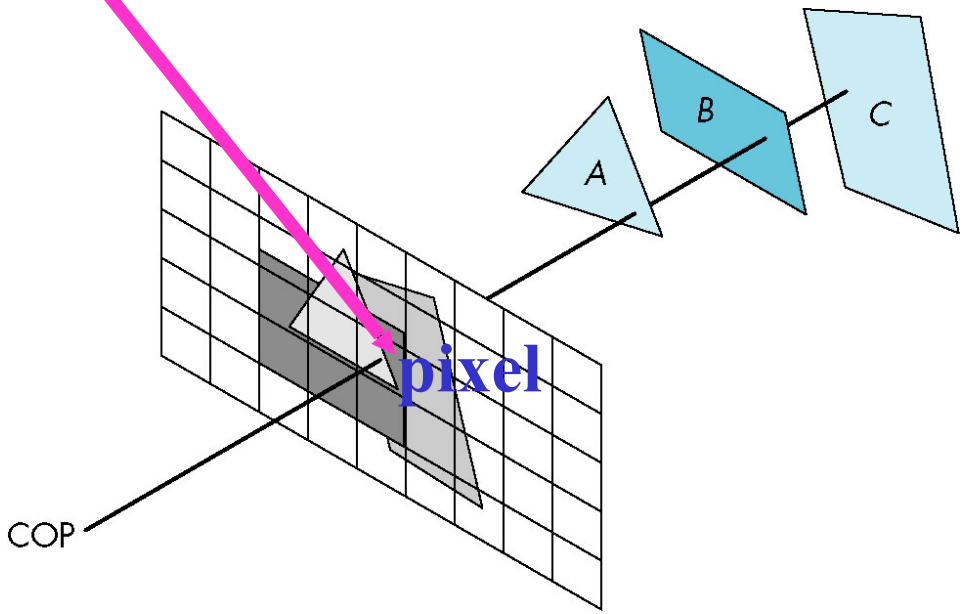


Image Space Approach

- Look at each ray projector (**nm** for an **n** x **m** frame buffer) & find closest of **k** polygons

- **Intersect ray** (**projector**) with each of the planes determine by our **k** polygons, and find the **intersection closest to the COP**.

- Complexity O(**nmk**)
- Ray tracing (casting)
- z-buffer

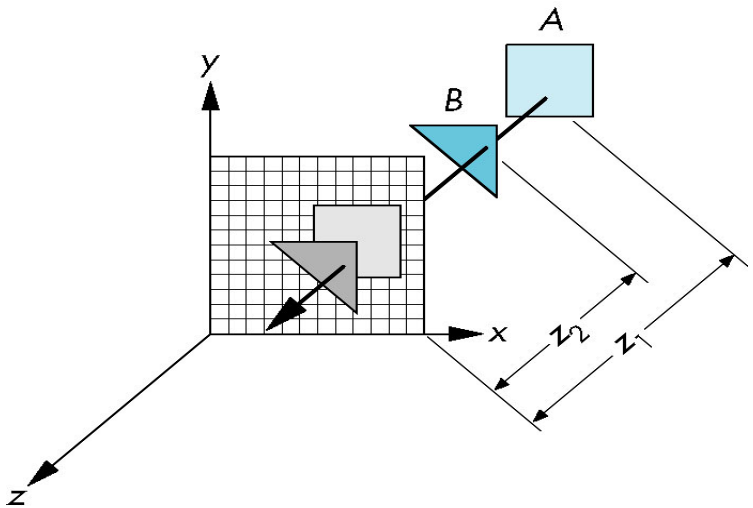


z-Buffer Algorithm

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

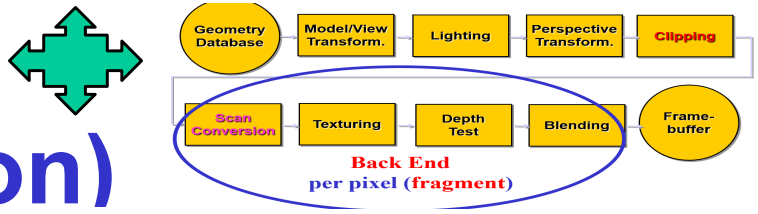
If less, place **shade of pixel in color buffer and update z buffer**



Objectives

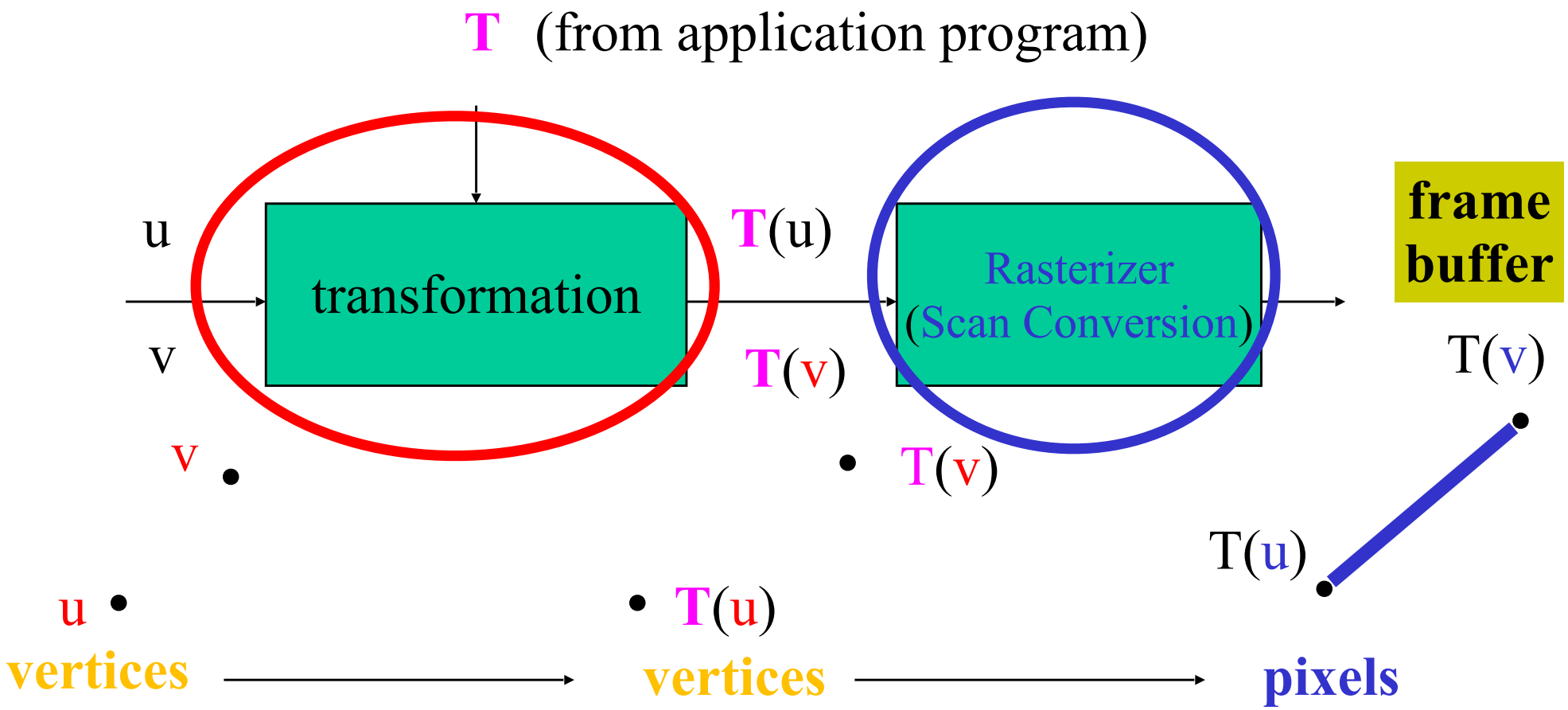
- Survey Line Drawing Algorithms
  - DDA
  - Bresenham

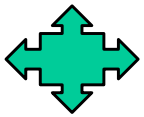
Chapter 7.8



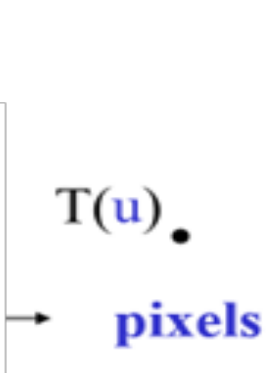
Rasterizer (Scan Conversion)

We need only to transform the homogeneous-coordinate representation of the endpoints of a **line segment** to determine completely a **transformed line**. Thus, we can implement our graphics systems as a **pipeline that passes endpoints through affine transformation** units, and **generates the interior points at the rasterization stage**.





Rasterization



Rasterization (scan conversion)

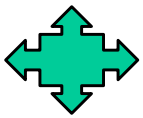
Determine **which pixels that are inside primitive** specified **a set of vertices**

Produces a **set of fragments**

frame buffer

Fragments have a location (pixel location) and other attributes such color and texture coordinates that are determined by interpolating values at vertices

Pixel colors determined **later using color, texture, and other vertex properties**



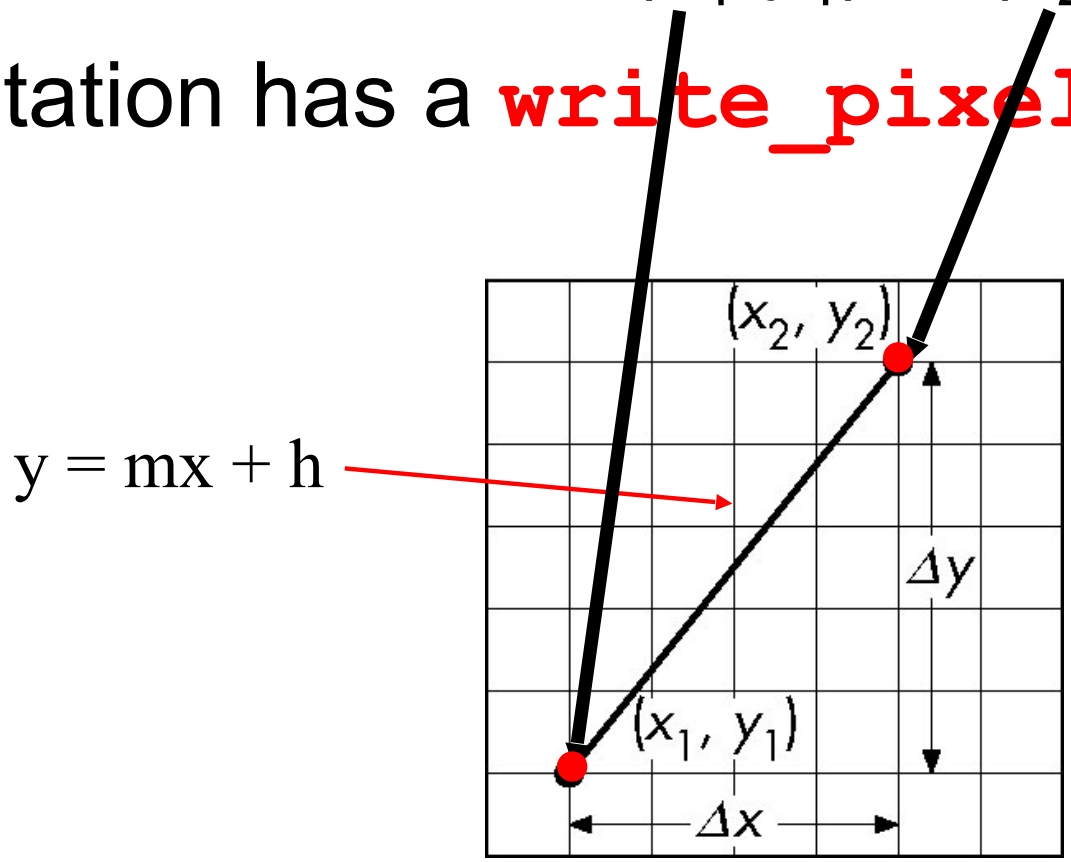
T(v)

T(u)

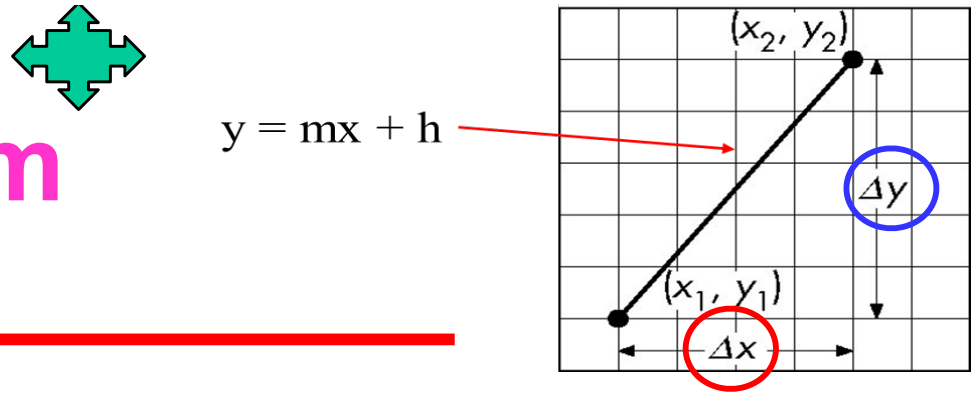
→ pixels

Scan Conversion of Line Segments

- Start with **line segment** **in window coordinates** with integer values for **endpoints** (x_1,y_1) & (x_2,y_2)
- Assume implementation has a **`write_pixel`** function



DDA Algorithm



Digital **D**ifferential **A**nalyzer

DDA was a mechanical device for numerical solution of differential equations

Line $y = m x + h$ satisfies differential equation

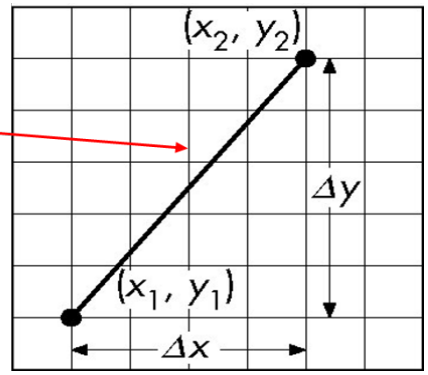
$$dy/dx = m = \Delta y/\Delta x = (y_2-y_1)/(x_2-x_1)$$

Along **scan line** $\Delta x = 1$

```
for(x= x1; x<=x2,ix++)
                {
               y+=m;
write_pixel(x, round(y), line_color)
                }
```

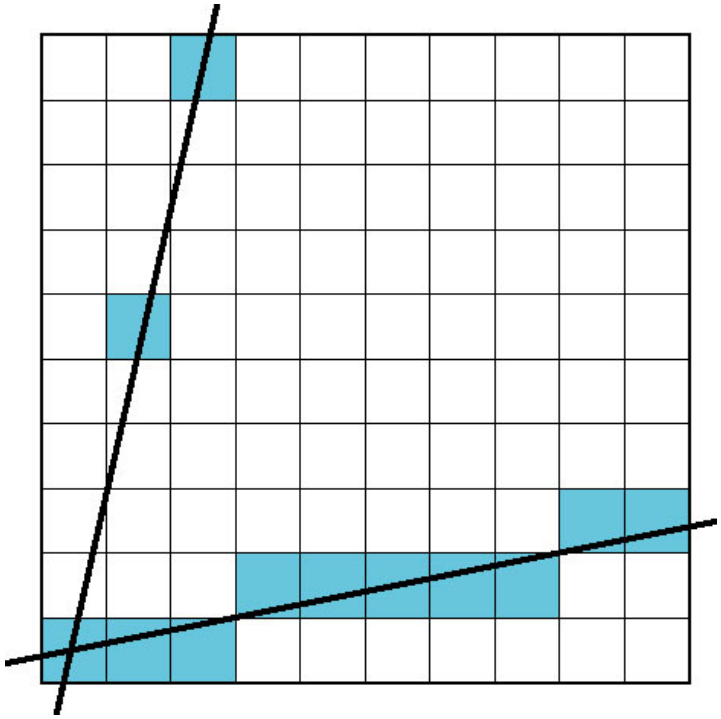
Problem

$y = mx + h$



- **DDA** = for each **x** plot pixel **at closest y**

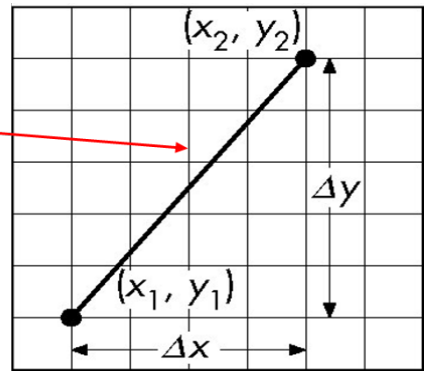
  Problems for **steep lines**



No problems for horizontal, vertical, or 45º lines

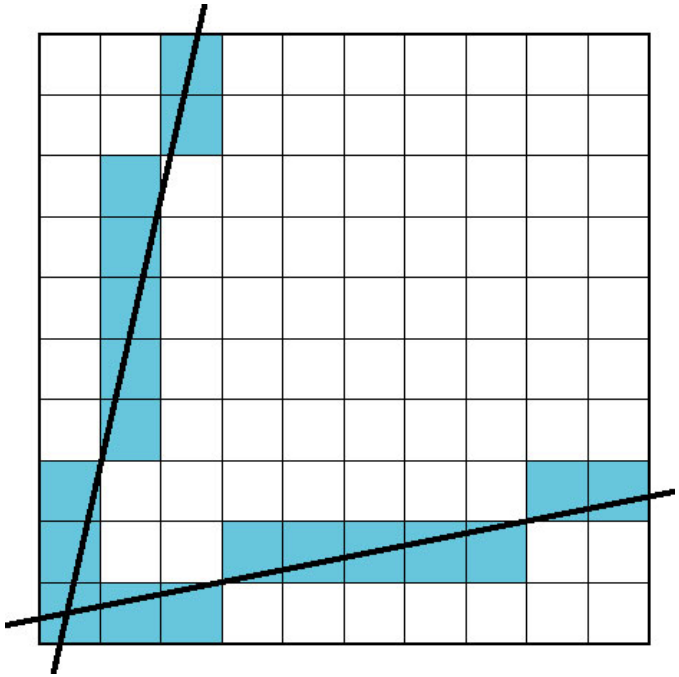
Solution 1: Using Symmetry

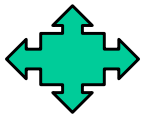
$y = mx + h$



- Use for $1 \geq \mathbf{m} \geq 0$
- For **m** > 1, swap role of x and y

  For each **y**, plot closest **x**





Bresenham's Algorithm

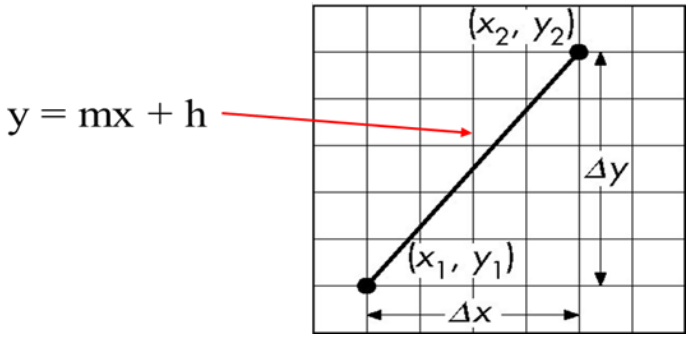
```
y+=m;
```

- DDA requires one floating point addition per step

We can eliminate all floating point through Bresenham's algorithm

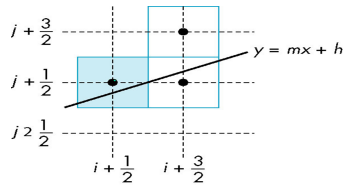
Consider only 1 ≥ m ≥ 0

Other cases by symmetry



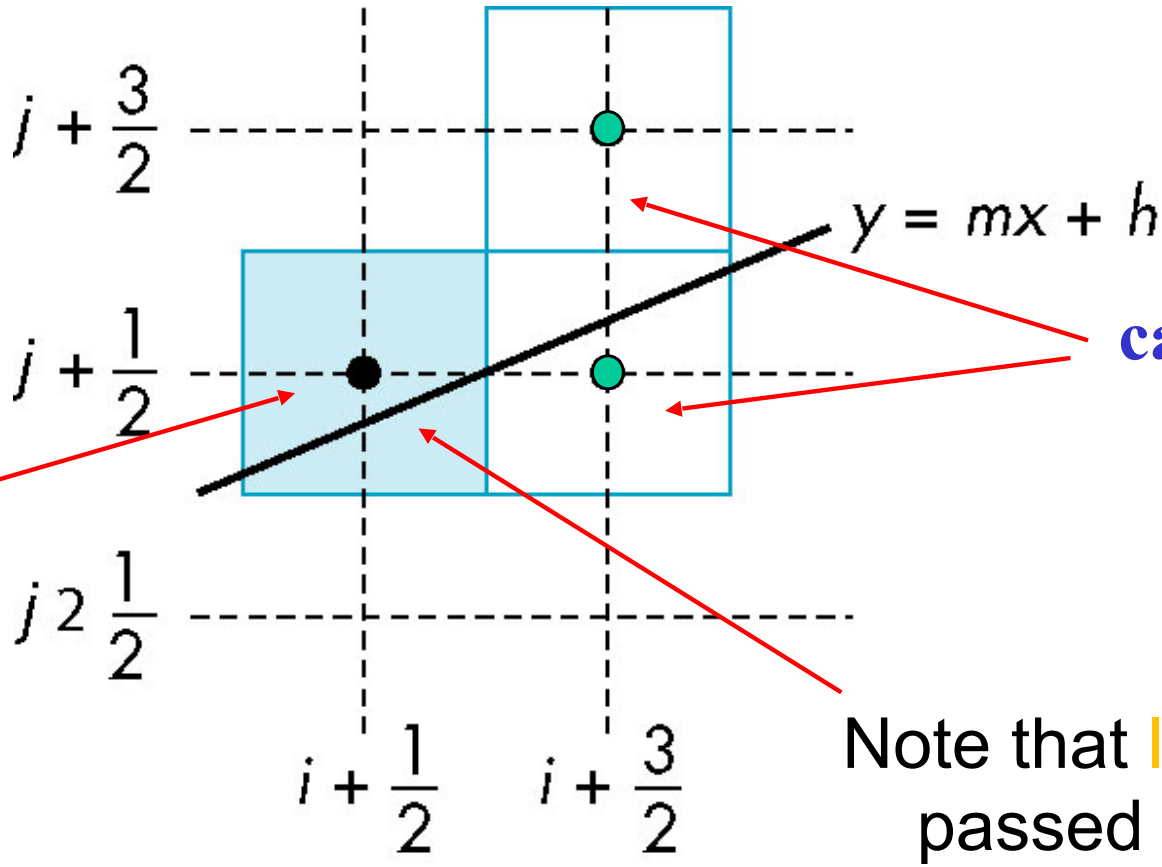
Assume pixel centers are at half integers

If we **start at a pixel that has been written**, there are **only two candidates for the next pixel to be written into the frame buffer**



Candidate Pixels

$1 \geq \mathbf{m} \geq 0$



Note that line could have passed through **any part of this pixel**

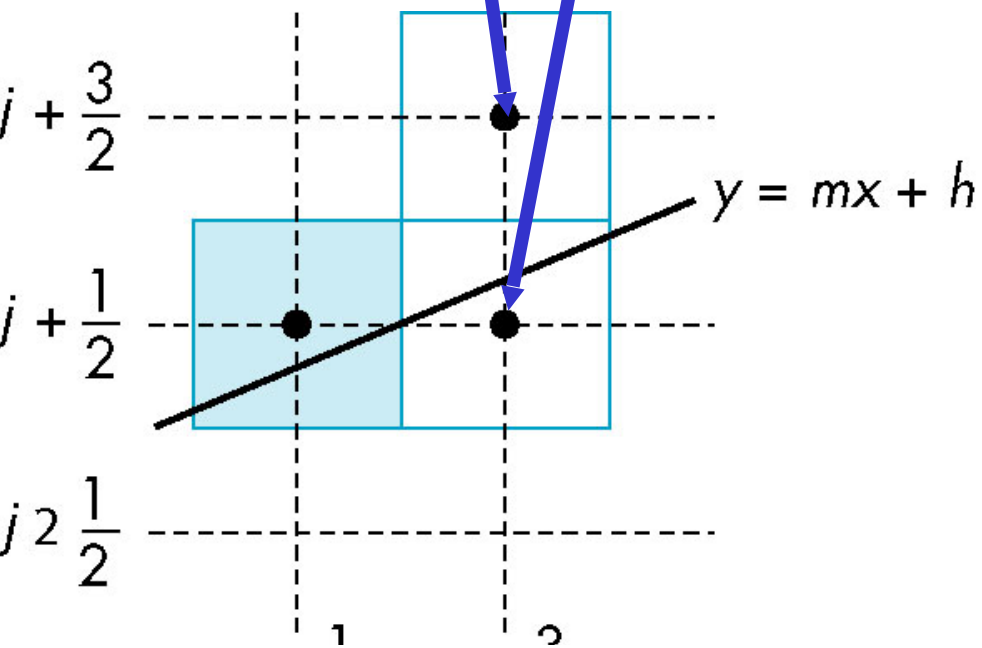
Decision Variable

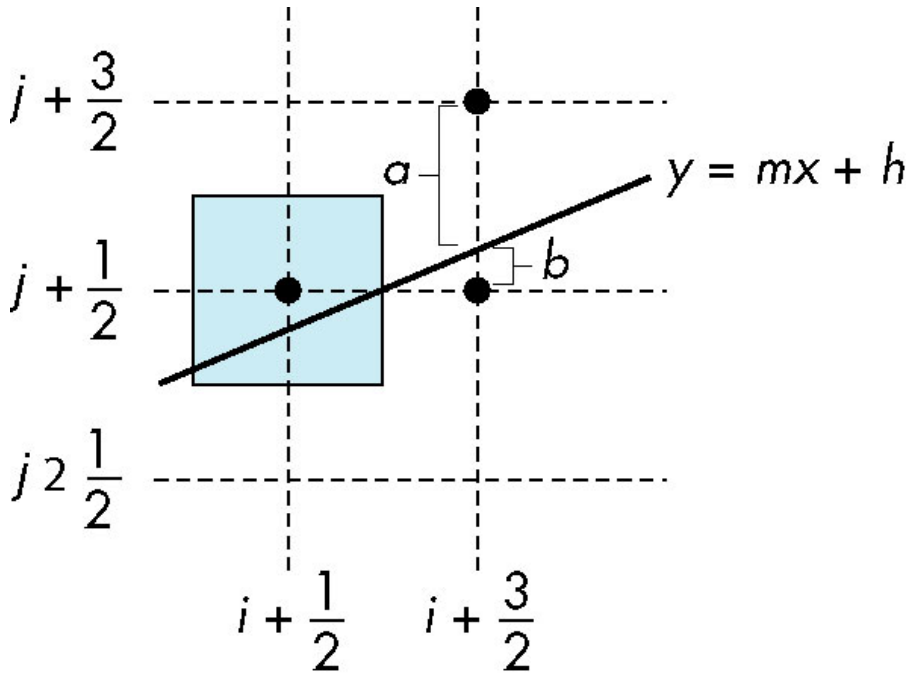
d = Δx(b-a)

d is an integer

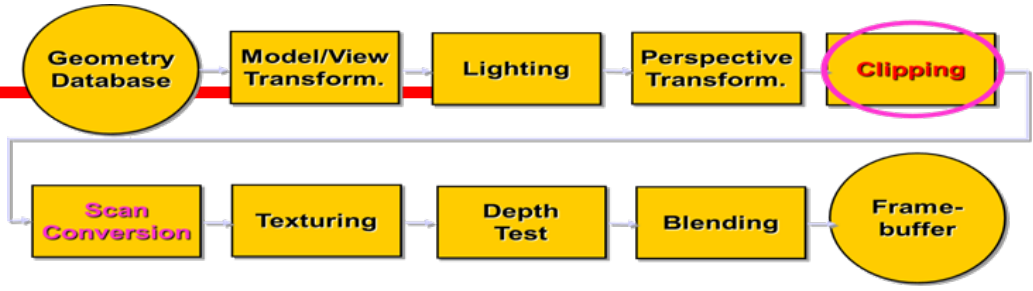
d > 0 use upper pixel

d < 0 use lower pixel





Filling in the Frame Buffer



Fill at end of pipeline

Convex Polygons only

Nonconvex polygons assumed to have been tessellated

Shades (colors) have been computed for **vertices** (**Gouraud shading**)

Combine with **z-buffer algorithm**

- **March across scan lines interpolating shades**
- **Incremental work small**

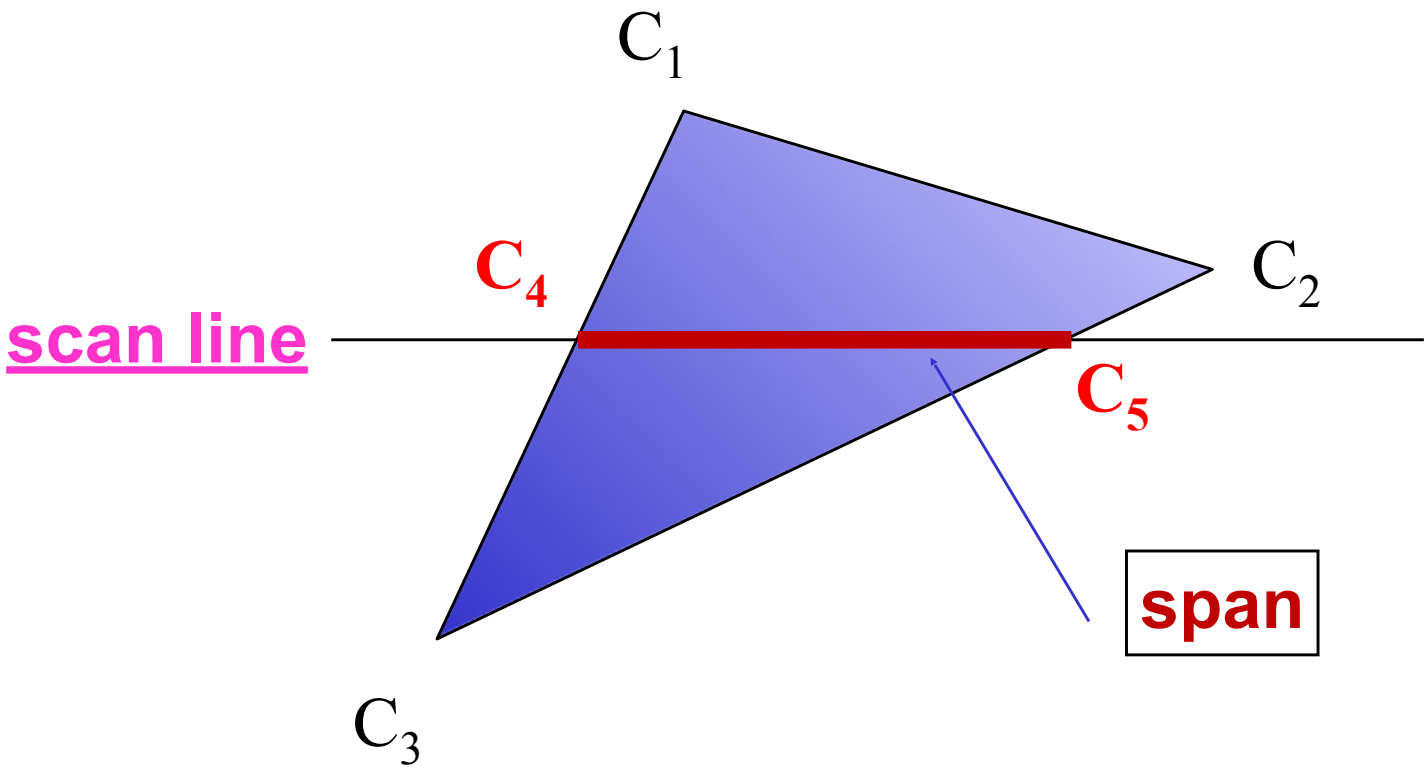
Using Interpolation

C_1 C_2 C_3 specified by `glColor` or by vertex shading

C_4 determined by interpolating between C_1 and C_3

C_5 determined by interpolating between C_2 and C_3

interpolate between C_4 and C_5 along span

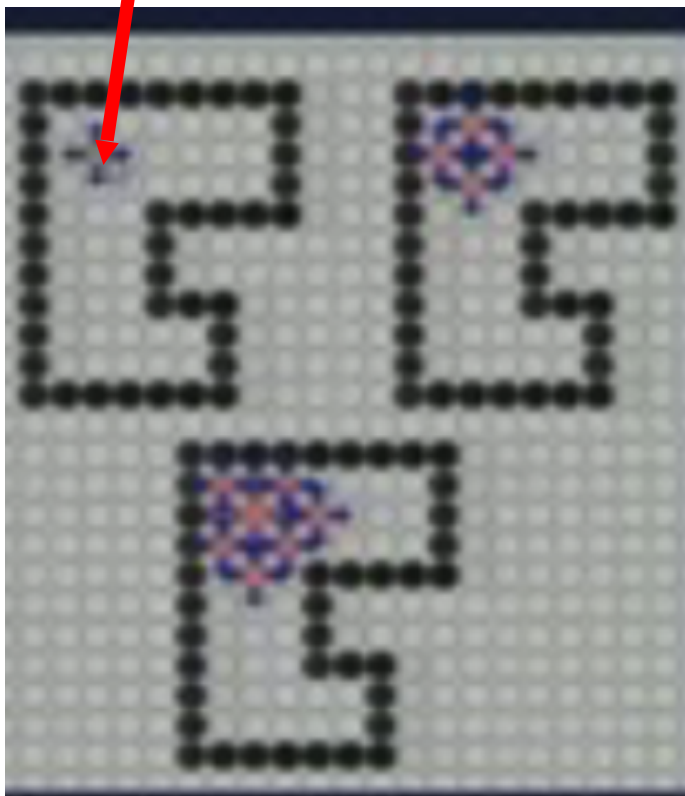


Flood Fill

Fill can be done recursively if we know a **seed point** **located inside** (WHITE)

Scan convert edges into buffer in edge/inside color (BLACK)

```
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);
          }   }
```

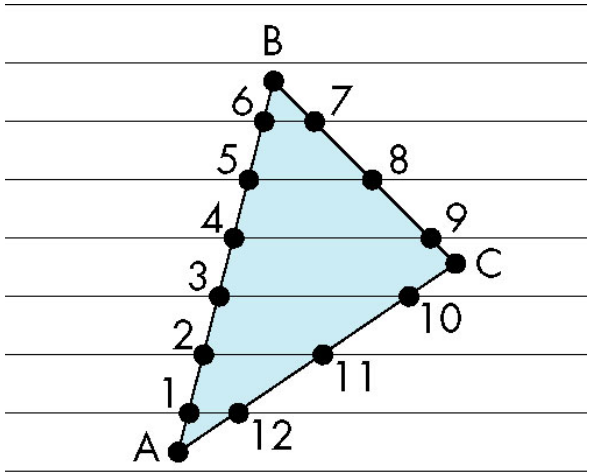


Scan Line Fill

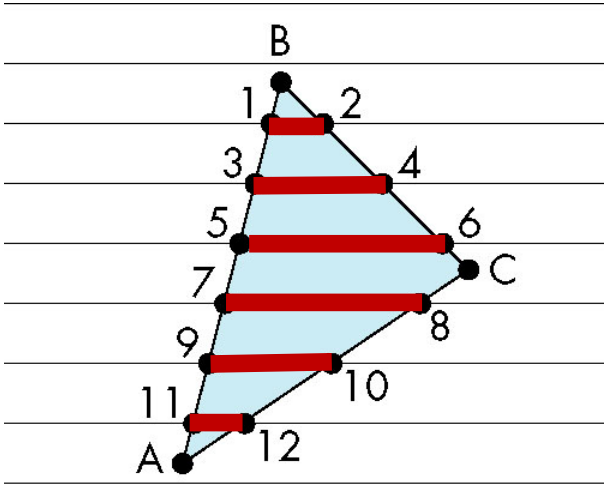
Can also fill by maintaining a data structure of **all intersections** of polygons with **scan lines**

Sort by **scan line**

Fill each **span**

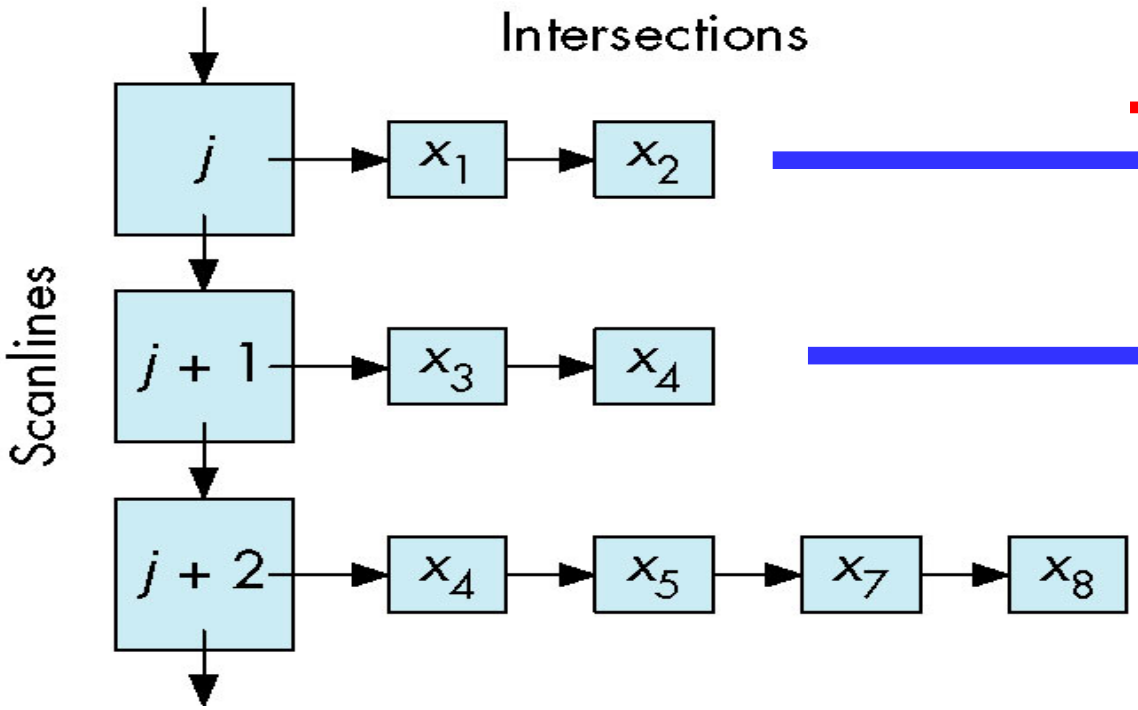


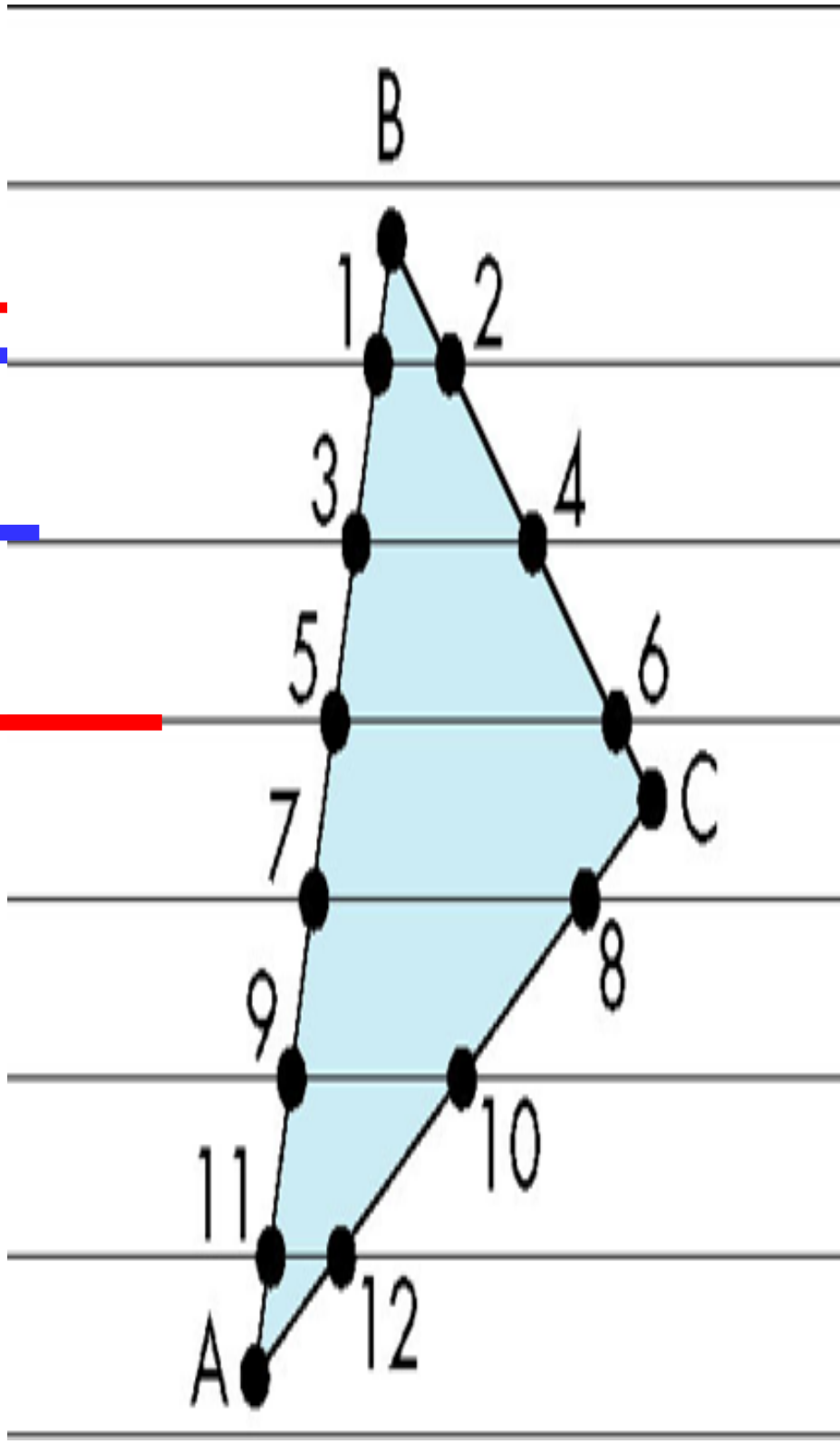
vertex order generated by **vertex list**



desired order

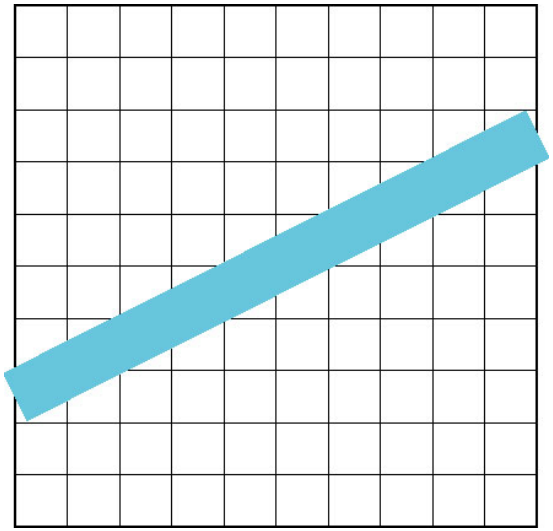
Data Structure

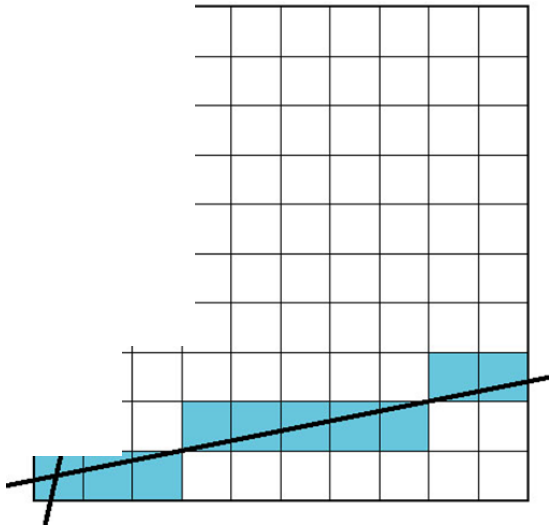




Aliasing

Ideal **rasterized** line should be 1 pixel wide



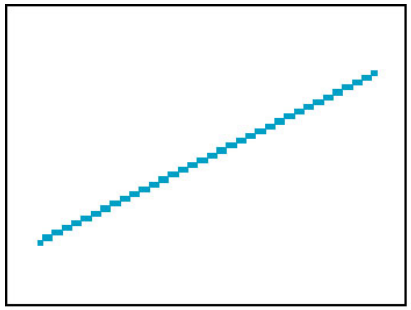


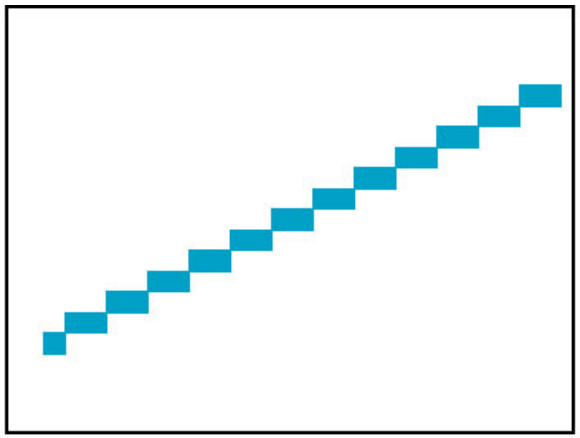
Choosing best y for each x (or vice versa) produces **aliased raster lines**

Antialiasing by Area Averaging

- **Color multiple pixels for each x** depending on coverage by **ideal line**

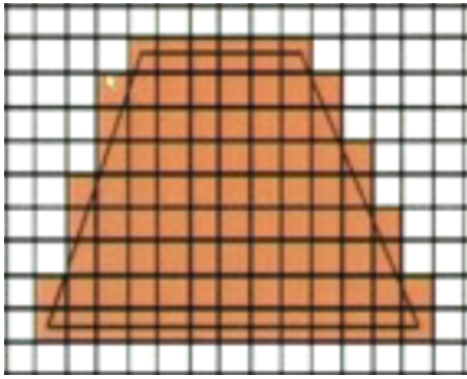
original



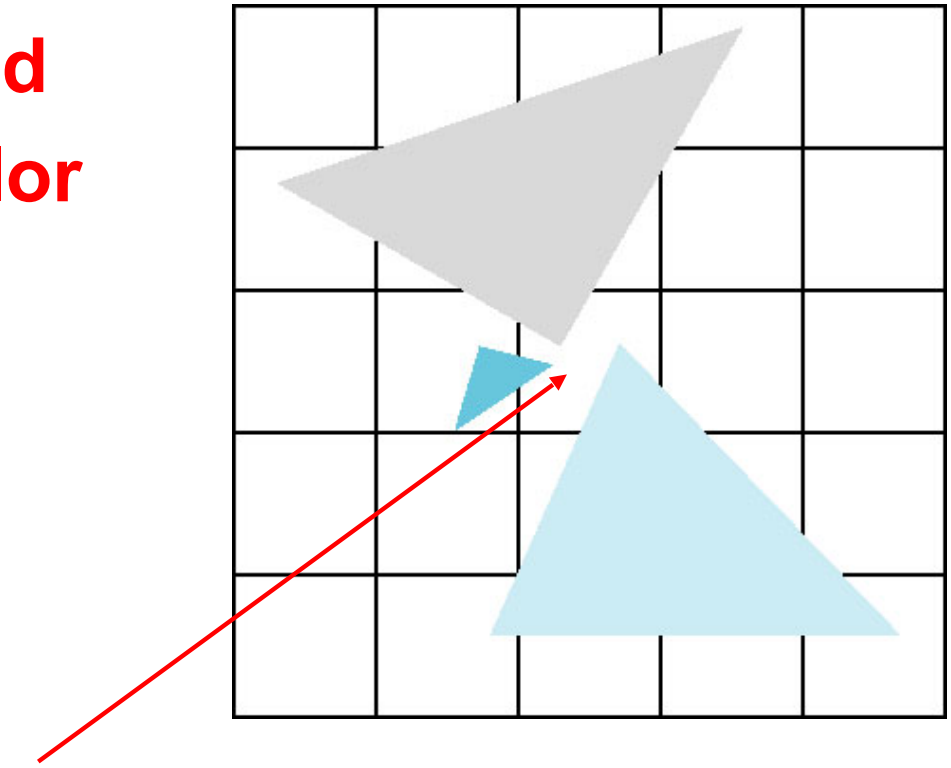


magnified

Polygon Aliasing



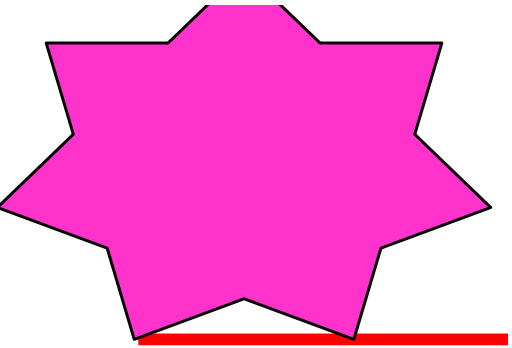
- Aliasing problems can be **serious** for **polygons**
  - **Jaggedness of edges**
  - **Small polygons neglected**
  - Need **compositing so color** of **one polygon** does not totally determine **color of pixel**



All three polygons should contribute to **color**

Color Plate 8 Rendering of a small part of the sun object with an environment map. (Courtesy of Full Dome Project, University of New Mexico.)

(a) Without antialiasing



You will be prompted when to Upload completed document to CANVAS as **score.doc** (example 100.doc).

Warning:

TA, at random, will inspect the Uploaded document.

If you score is not honestly entered you will get a zero.

Please rename document to score.doc (example 100.doc)
Warning: if you score is not honestly honest you will get a zero.

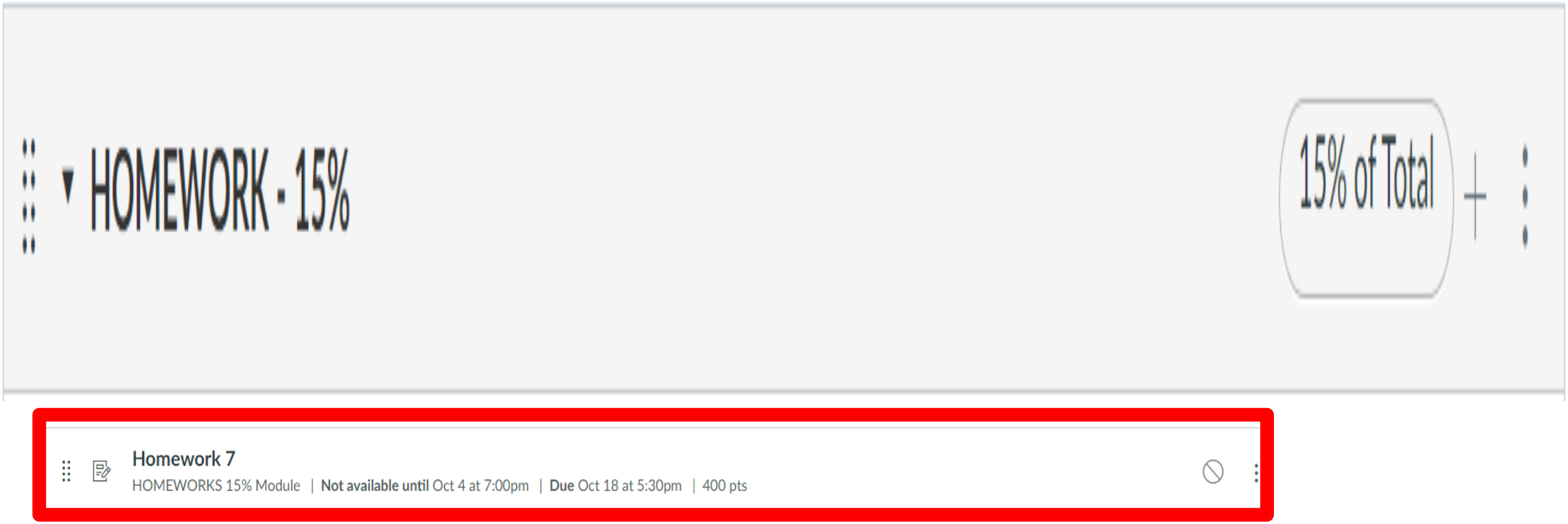
Name: _____

Total score: ☐

Class PARTICIPATION on Lecture 7.doc **ANSWER SHEET**
(Out of 100 points. Please record your own total score!)
(Attach as score.doc!)

VH, it closes at 7:00 PM

10.18.2023 (W 5:30 to 7) (17)	Homework 7	Lecture 8



VH, publish

NEXT.

10.09.2023 (M 5:30 to 7) (14)	Homework 6	PROJECT 2
10.11.2023 (W 5:30 to 7) (15)		EXAM 2 REVIEW
10.16.2023 (M 5:30 to 7) (16)		EXAM 2

NEXT.

10.09.2023 (M 5:30 to 7) (14)	Homework 6	PROJECT 2
10.11.2023 (W 5:30 to 7) (15)		EXAM 2 REVIEW
10.16.2023 (M 5:30 to 7) (16)		EXAM 2

PROJECT 2
(100 points)

1. The **glutCreateWindow** OpenGL call will take as parameter "**YourLastName** FirstName **BUS Version 2**".

2. Start with the **BUS.c** that you have created in **PROJECT 1** and call the file **MyBUS.c.** Please make sure it is based on your FINAL "**Blueprint**". Please make sure that the colors schema is observed.

3. Create a **menu** or **keyboard (f, b, r, l, t, u, i)** that allows that you to view the BUS from (**F1** is the Help Key):

- front;
- back;
- right side;
- left side;
- top;
- under;
- isometric.

Get a screenshot of the "YourLastName FirstName **BUS Version 2" window for each view.**

4. Add the **fly through navigation** to the **BUS** from (see **Homework 3**).

PROJECT 2

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 2.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 2".* (-30 points)
3. *The "PROJECT 2 Acceptance Testing".* (-20 points)
4. *The "PROJECT 2 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 2 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 2 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

Due	For	Available from	Until
Oct 9 at 5:30pm	Everyone	Sep 25 at 7pm	Oct 9 at 5:30pm

PROJECT 2

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 2.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 2".* (-30 points)
3. *The "PROJECT 2 Acceptance Testing".* (-20 points)
4. *The "PROJECT 2 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 2 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 2 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

PROJECT 2
(100 points)

Hours spent: 

The PROJECT is to be turned in before 5:30 PM of the due date class.
(Please do not submit this file)

Submit:

1. ***Your updated "Blueprint" matching the image given.***
   (-20 points)

2. ***The zipped Visual Studio2019 or WEBGL "PROJECT 2".***
   (-30 points)

3. ***The "PROJECT 2 Acceptance Testing".***
   (-20 points)

4. ***The "PROJECT 2 Acceptance Testing Check Sheet" as "score.OPENGL.doc or score.WEBGL.doc" to CANVAS.***
   (-10 points)
5. ***The ppt "PROJECT 2 PRESENTATION (no more than 20 slides)".***
   (-20 points)

(Please have your presentation and your PROJECT 2 project code opened)

Unexcused absence or not responding when your name is called to present.
(-20 points)

Please do not change the file names.

PROJECT 2

(100 points)

Hours spent:

The PROJECT is to be turned in before 5:30 PM of the due date class.
(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the image given.*
   (-20 points)

2. *The zipped Visual Studio2019 or WEBGL "PROJECT 2".*
   (-30 points)

3. *The "PROJECT 2 Acceptance Testing".*
   (-20 points)

4. *The "PROJECT 2 Acceptance Testing Check Sheet" as "score.OPENGL.doc or score.WEBGL.doc" to CANVAS.*
   (-10 points)
5. *The ppt "PROJECT 2 PRESENTATION (no more than 20 slides)".*
   (-20 points)

(Please have your presentation and your PROJECT 2 project code opened)

Unexcused absence or not responding when your name is called to present.
(-20 points)

Please do not change the file names.

PROJECT 2

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 2.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 2".* (-30 points)
3. *The "PROJECT 2 Acceptance Testing".* (-20 points)
4. *The "PROJECT 2 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 2 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 2 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

PROJECT 2 Acceptance Testing:

Insert screenshots of the output produced and insert it in this file ……..

1. Keyboard f so you can get the Front image of BUS.
SCREENSHOT Output of your program is:
……..
Copy and Paste your gluLookAt here
……..

2. Keyboard b so you can get the Back image of BUS.
SCREENSHOT Output of your program is:
……..
Copy and Paste your gluLookAt here
……..

3. Keyboard r so you can get the right Side image of BUS.
SCREENSHOT Output of your program is:
……..
Copy and Paste your gluLookAt here
……..

4. Keyboard t so you can get the Top image of BUS.
SCREENSHOT Output of your program is:
……..
Copy and Paste your gluLookAt here
……..

5. Keyboard i so you can get the Isometric image of BUS.
SCREENSHOT Output of your program is:
……..
Copy and Paste your gluLookAt here
……..

6. Fly Through the BUS through the door so you are in the middle of the BUS, looking towards the front of the BUS.
SCREENSHOT Output of your program is:
……..

7. Copy and Paste your .c (.cpp) file here:
……..

When done attach this file to "PROJECT 2" CANVAS assignment.

PROJECT 2

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 2.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 2".* (-30 points)
3. *The "PROJECT 2 Acceptance Testing".* (-20 points)
4. *The "PROJECT 2 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 2 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 2 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

PROJECT 2 Hours:

Acceptance Testing Check Sheet:

CHECKLIST (YOU MUST SCORE YOURSELF!):

1. Submitted Completed Blue Print & BUS? -20 points
2. Submitted PROJECT2.zip? -30 points
3. Submitted Acceptance Testing.docx? -20 points
4. Submitted Acceptance Testing Check Sheet as score.OPENGL.docx or score.WEBGL.docx? -10 points
5. Submitted PROJECT 2 PRESENTATION.pptx? -20 points

6. ACCEPTANCE TESTING Step 1? 15 points
7. ACCEPTANCE TESTING Step 2? 15 points
8. ACCEPTANCE TESTING Step 3? 15 points
9. ACCEPTANCE TESTING Step 4? 15 points
10. ACCEPTANCE TESTING Step 5? 15 points
11. ACCEPTANCE TESTING Step 6? 15 points
12. ACCEPTANCE TESTING Step 7? 10 points

Score

I CERTIFY THAT THIS IS MY OWN WORK and THE CHECKBOXES reflect the completion of these DELIVERABLES.

PROJECT 2

Publish | Edit | ⋮

The PROJECT is due by 5:30 PM of the due date class.

PROJECT 2.doc

(Please do not submit this file)

Submit:

1. *Your updated "Blueprint" matching the provided BUS Image* (-20 points)
2. *The zipped Visual Studio2019 or WEBGL "PROJECT 2".* (-30 points)
3. *The "PROJECT 2 Acceptance Testing".* (-20 points)
4. *The "PROJECT 2 Acceptance Testing Check Sheet".* (-10 points)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)

   Rename it to score.WEBGL.docx. (MUST BE A .DOCX DOCUMENT)
5. *The ppt "PROJECT 2 PRESENTATION (no more than 20 slides)".* (-20 points)

Please have your presentation and your PROJECT 2 project code opened.

Unexcused absence or not responding when your name is called to present. (-20 points)

Please do not change the file names.

At 6:45 PM.

End Class 13

VH, Download Attendance Report
Rename it:
10.4.2023 Attendance Report FINAL

VH, upload Lecture 7 to CANVAS.