Algorithms and Data Structures

COMP261
3D Rendering 3

Yi Mei

yi.mei@ecs.vuw.ac.nz

Outline

- Polygon rendering
  - Linear Interpolation
  - Z-buffer to only render the closest polygons

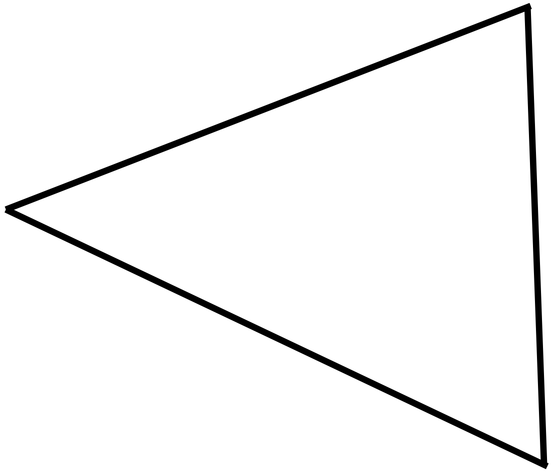
What We Have Learned

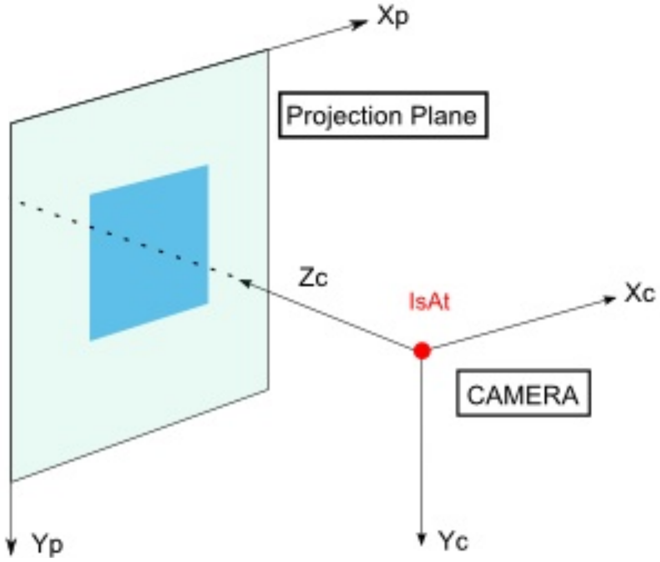
- Use triangle polygons to approximate object surface (computationally efficient when the object is translated, scaled and rotated)
  - Three vertices, each with 3D coordinates
  - Recalculate the coordinates of the vertices upon transformation
- Identify the visible/invisible part of the surface (only render the visible part)
  - Calculate *normal* (vector) of polygons by cross product
  - Order the vertices so that the the normal is pointing to the viewer
  - Setup the coordinate system so that z-axis is the viewing direction
  - Negative z-value of normal -> the polygon is visible
- Shading (color of polygons that you see)
  - Use some physical principles
  - Depends on lights and viewing direction

3D Rendering

- For each visible polygon
  - Compute its shading color
  - Render the polygon with the shading color

- How to render a polygon?
  - Draw 3D polygon on a 2D screen





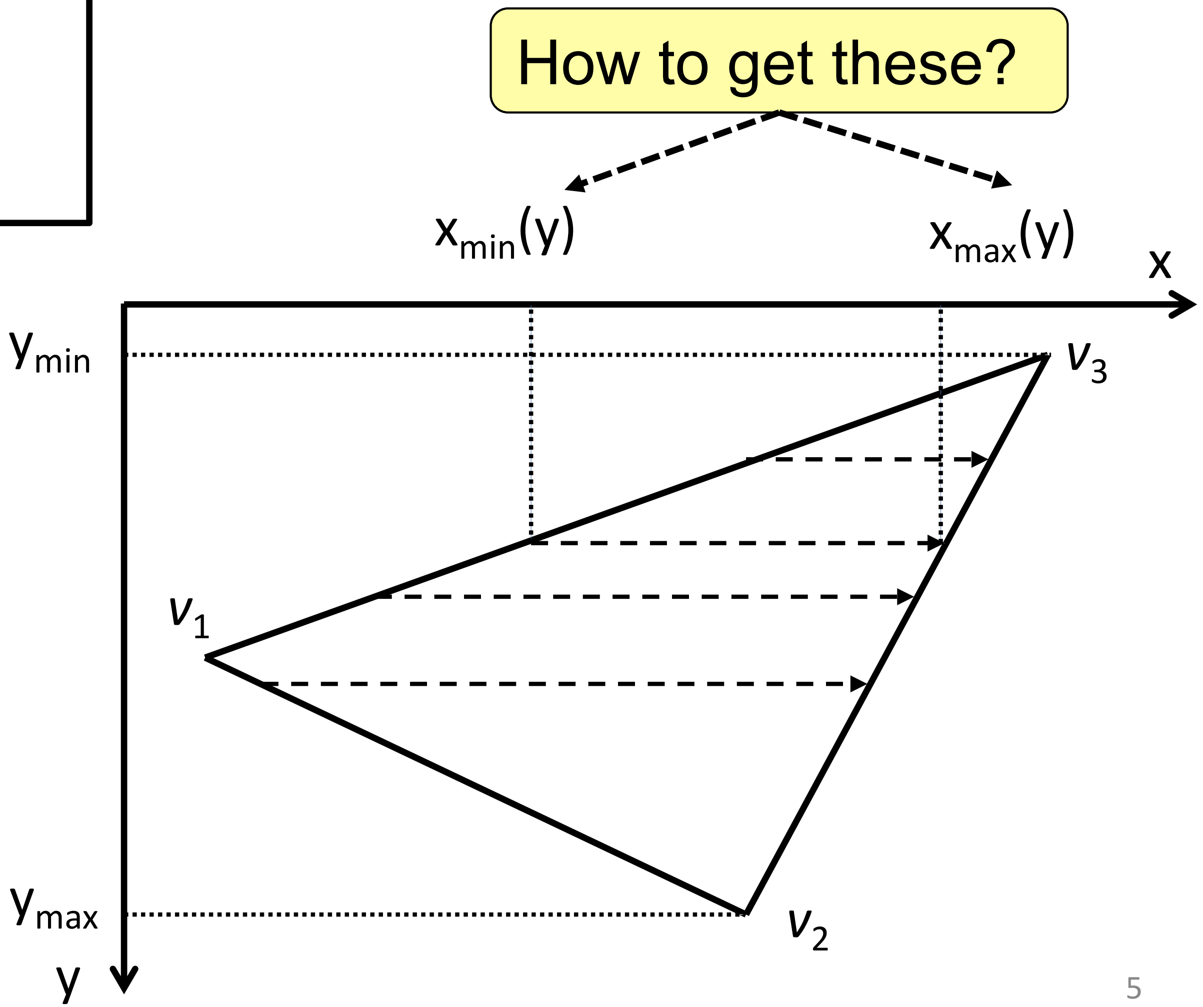
Polygon Rendering

- z-axis is the viewing direction, so the screen is x-y plane
  - Render pixels line by line

```
for ($y = y_{min}$ to $y_{max}$) {
   for ($x = x_{min}(y)$ to $x_{max}(y)$)
      pixel(x,y) = shading color;
}
```

$y_{min} = v_3.y;$

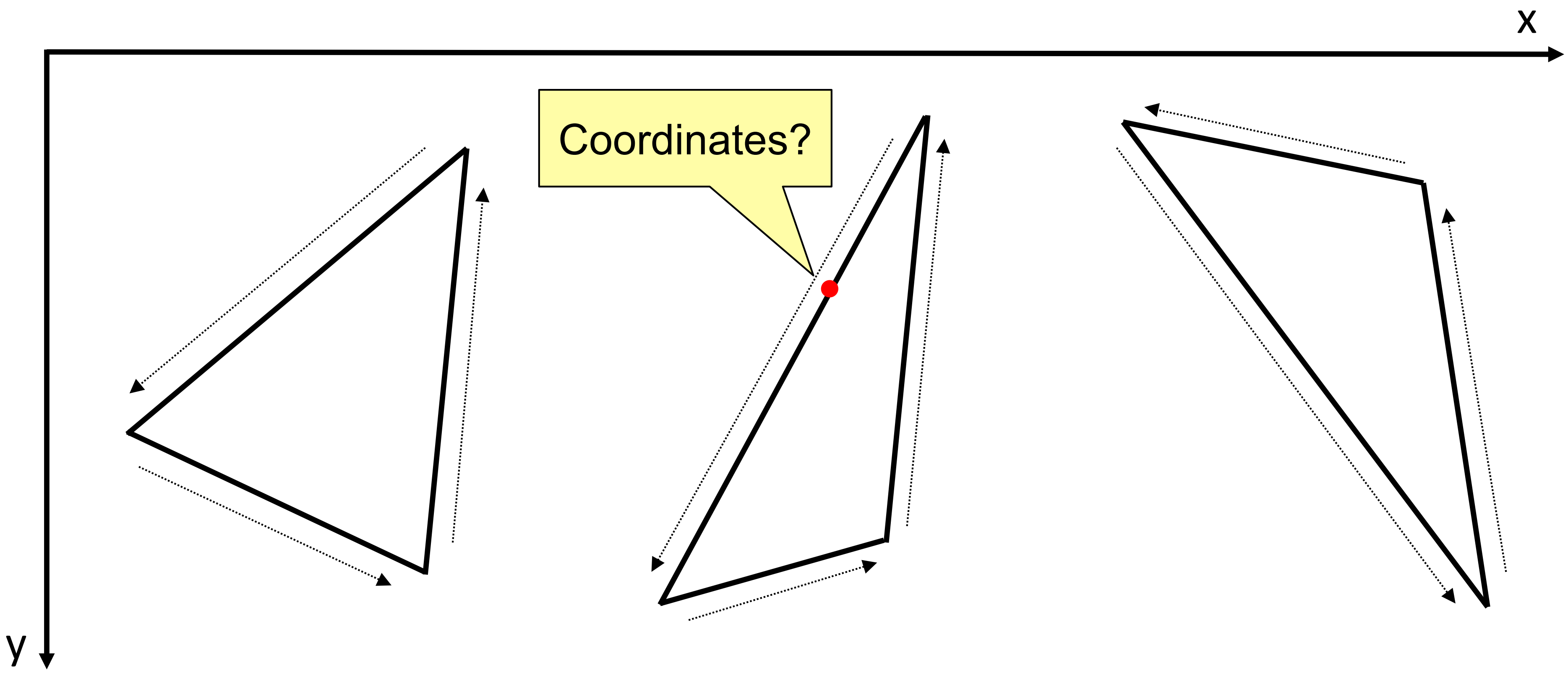
$y_{max} = v_2.y;$



How to get these?

Polygon Rendering

- For any y value, get $x_{min}(y)$ and $x_{max}(y)$
  - All the $x_{min}(y)$ and $x_{max}(y)$ are on the edges of the polygon
  - If scanning the edges anti-clockwise, then
    - When the scan is going down, then visit $x_{min}(y)$
    - When the scan is going up, then visit $x_{max}(y)$

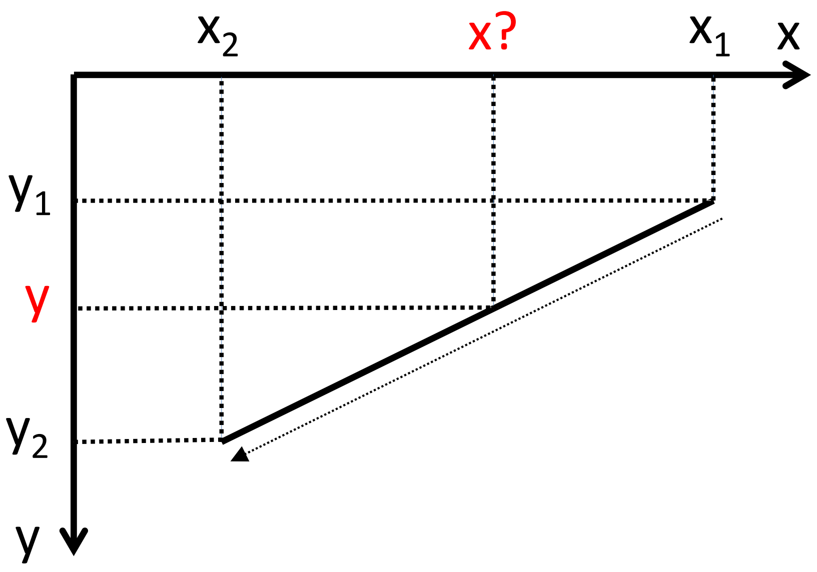


Linear Interpolation

- Given the two end-nodes of an edge (x_1,y_1) and (x_2,y_2), what is the x value of given y along the edge?

- y changed from y_1 to $y_{2,}$ x changed from x_1 to x_2
- For each unit change of y, x changed $\frac{x_2-x_1}{y_2-y_1}$ (*slope*)

- $x(y) = x_1 + slope\times(y - y_1)$

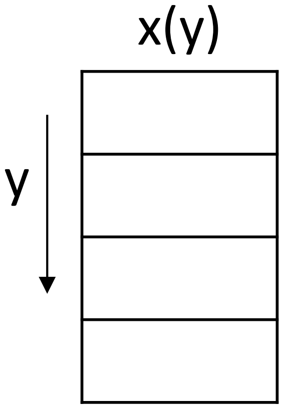


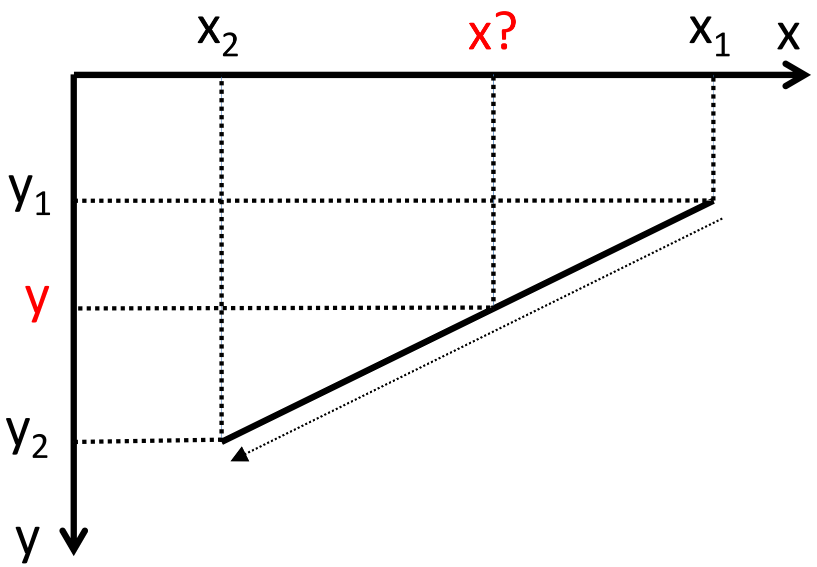
Linear Interpolation

- Given the two end-nodes of an edge (x_1,y_1) and (x_2,y_2), what is the x value of given y along the edge?

- Get the x value of **ALL** y's along the edge (store a list)

```
slope = $(x_2 – x_1) / (y_2 – y_1)$;
x = $x_1$;
for (y = $y_1$ to $y_2$) {
    x(y) = x;
    x = x + slope;
}
```





Scan **three edges** to get **two lists**

- **$x_{min}(y)$** list
- **$x_{max}(y)$** list

Edge List

- Scan each of the three edges, and update 2-column **EdgeList**
  - If scanning **up**, then update **$x_{max}(y)$** column
  - If scanning **down**, then update **$x_{min}(y)$** column
  - Use liner interpolation for each edge

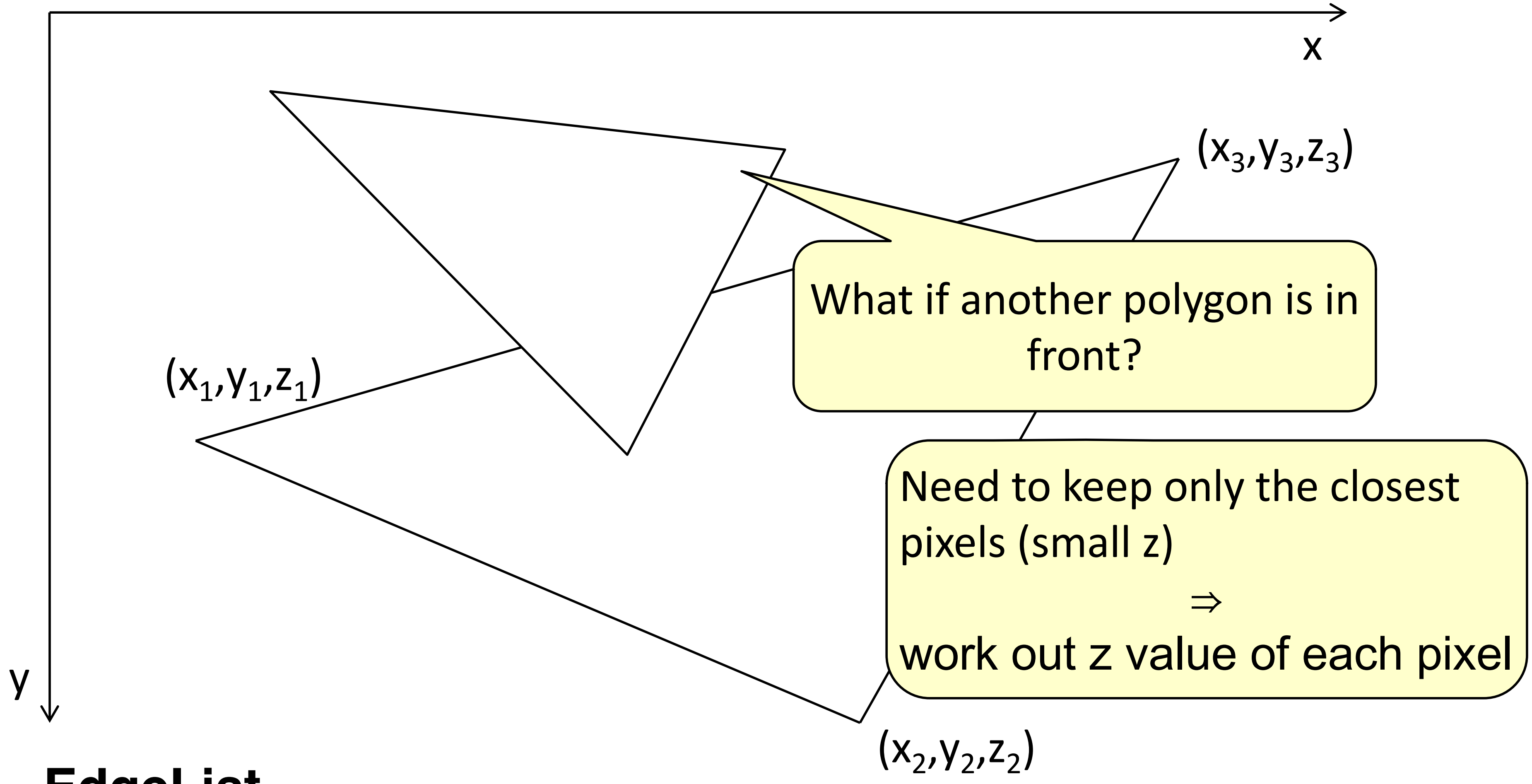
```
for (edge (a, b) in {($v_1$,$v_2$), ($v_2$, $v_3$), ($v_3$, $v_1$)}) {
    slope = (b.x – a.x) / (b.y – a.y);
    x = a.x, y = round(a.y);
    if (a.y < b.y) {// going down, update $x_{min}(y)$
        while (y <= round(b.y))
            $x_{min}(y)$ = x, x = x + slope, y++;
    else // going up, update $x_{max}(y)$
        while (y >= round(b.y))
            $x_{max}(y)$ ← x, x ← x – slope, y--
```

Anti-clockwise ordered

EdgeList

	$x_{min}(y)$	$x_{max}(y)$
y_{min}		
$y_{min}+1$		
⋮		
y_{max}		

Multiple Polygons



What if another polygon is in front?

Need to keep only the closest pixels (small z)

⇒

work out z value of each pixel

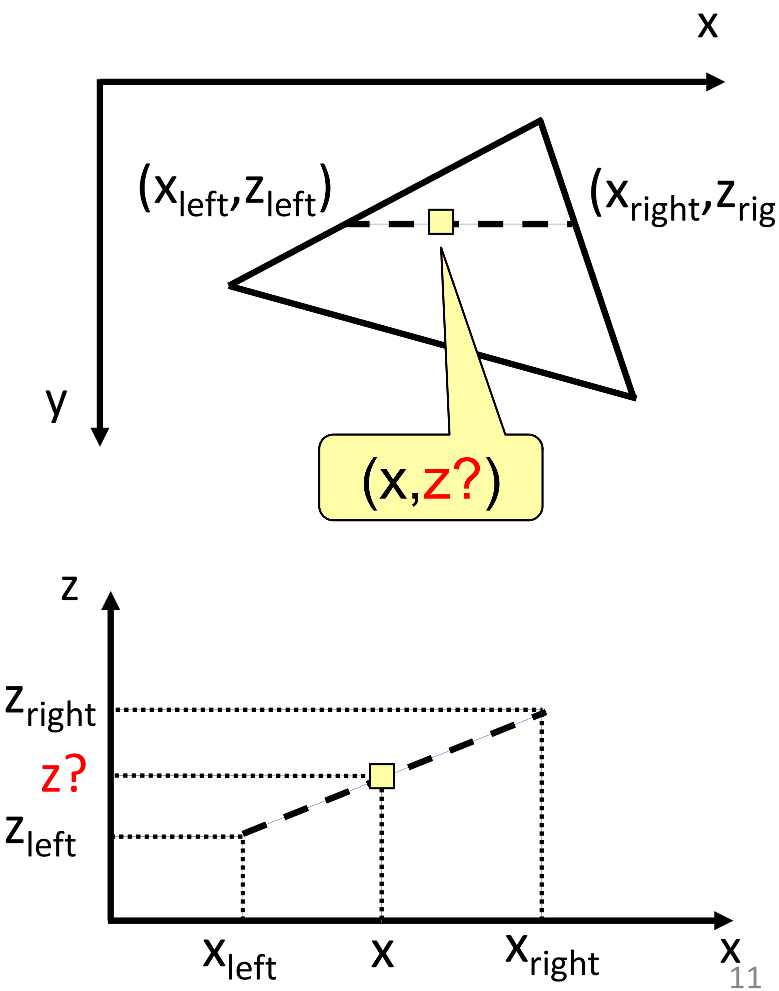
- **EdgeList**
  - **$x_{min}(y)$** and **$x_{max}(y)$** for the edges
  - **$z(x, y)$** for each pixel
    - If a pixel is on multiple polygons, render the polygon where it has the smallest z value

Render with EdgeList and z-buffer

- Compute the EdgeList for both x and z for the vertices on the edges
  - Compute the z value for each pixel inside the polygon using another linear interpolation

EdgeList

	Left		Right	
	x	z	x	z
y_{min}				
$y_{min}+1$				
⋮				
y_{max}				



Render with EdgeList and z-buffer

```
renderedImg = new Color[imageWidth][imageHeight];
zdepth = new double[imageWidth][imageHeight], initialise all entries to $\infty$;
for (each polygon) {
    calculate the x and z EdgeList (EL) of this polygon;
    for (y from $EL.y_{min}$ to $EL.y_{max}$) {
        slope = $(EL.z_{right}(y) – EL.z_{left}(y)) / (EL.x_{right}(y) – EL.x_{left}(y))$;
        x = round($EL.x_{left}(y)$), z = $EL.z_{left}(y)$ + slope * (x – $EL.x_{left}(y)$);
        while (x <= round($EL.x_{right}(y)$)) {
            if (z < zdepth(x,y)) {
                renderedImg(x,y) = shading color of this polygon, zdepth(x,y) = z;
                $z \leftarrow z + slope$, x++;
}}}}
return renderedImg;
```

Summary

- Polygon rendering
  - Render pixels line-by-line
  - Find Edge List and use linear interpolation for line boundaries
  - Multiple polygons: only render the closest parts
    - x and z EdgeList
    - z-buffer: store the current closest zdepth of each pixel