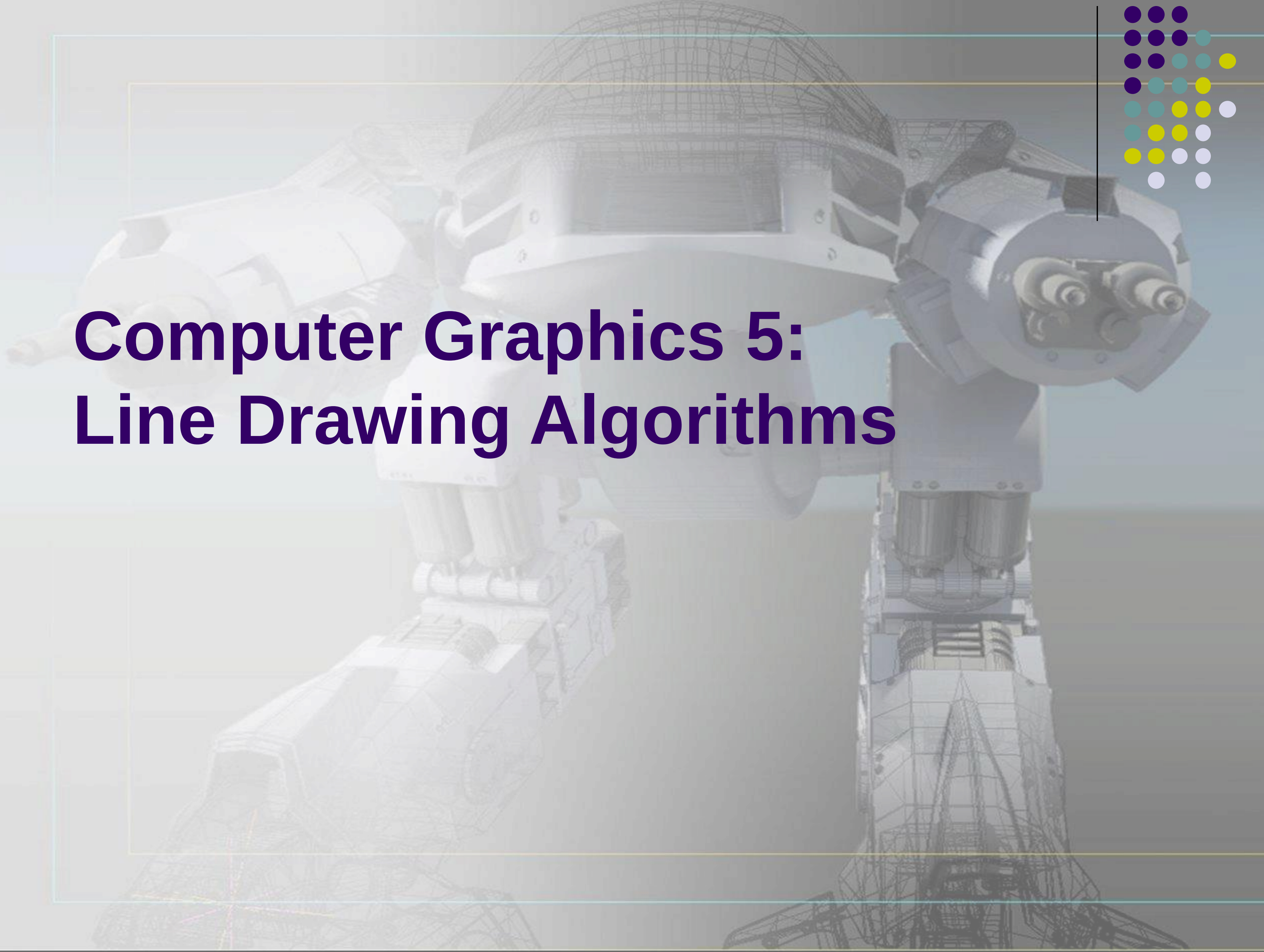
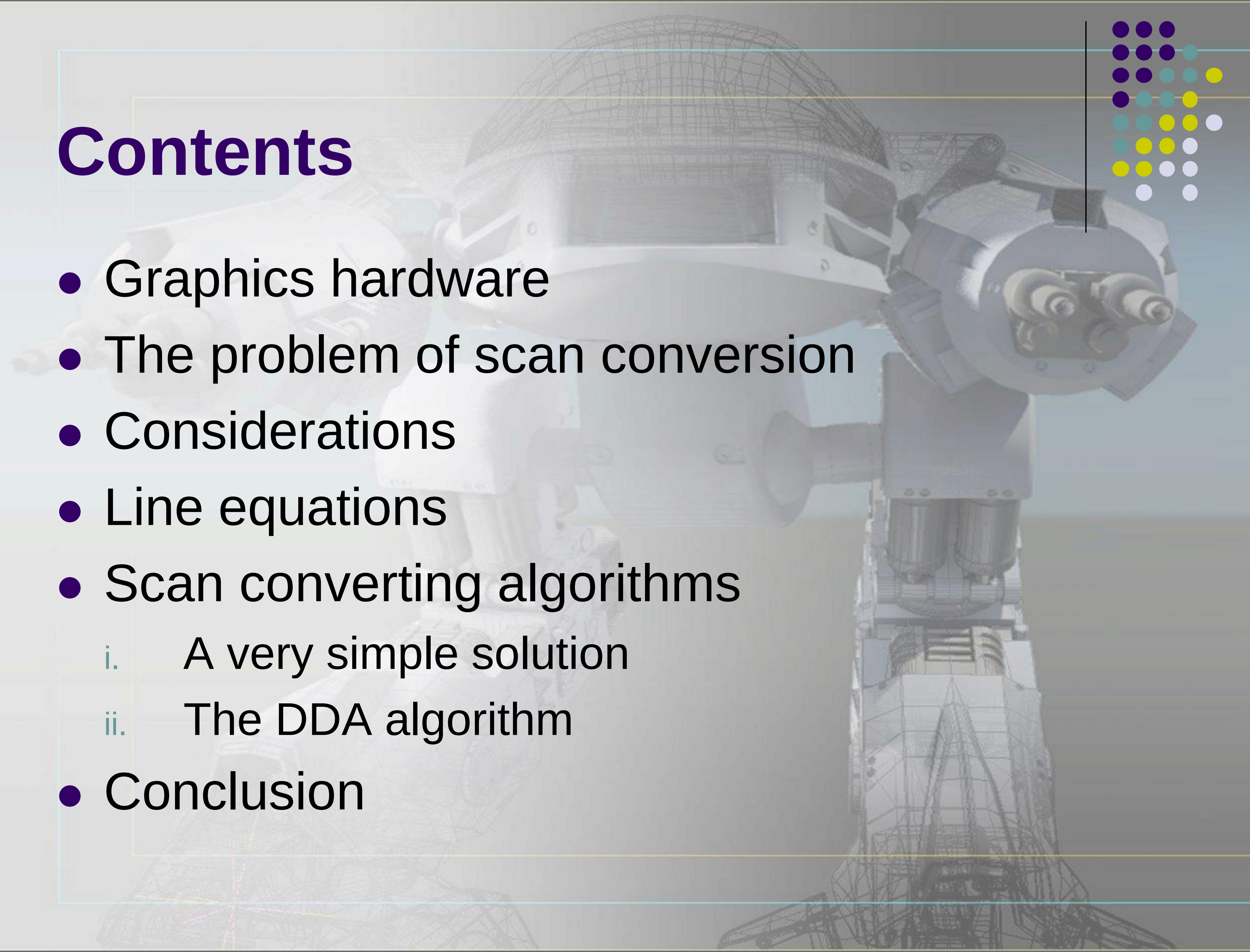
Computer Graphics 5:
Line Drawing Algorithms

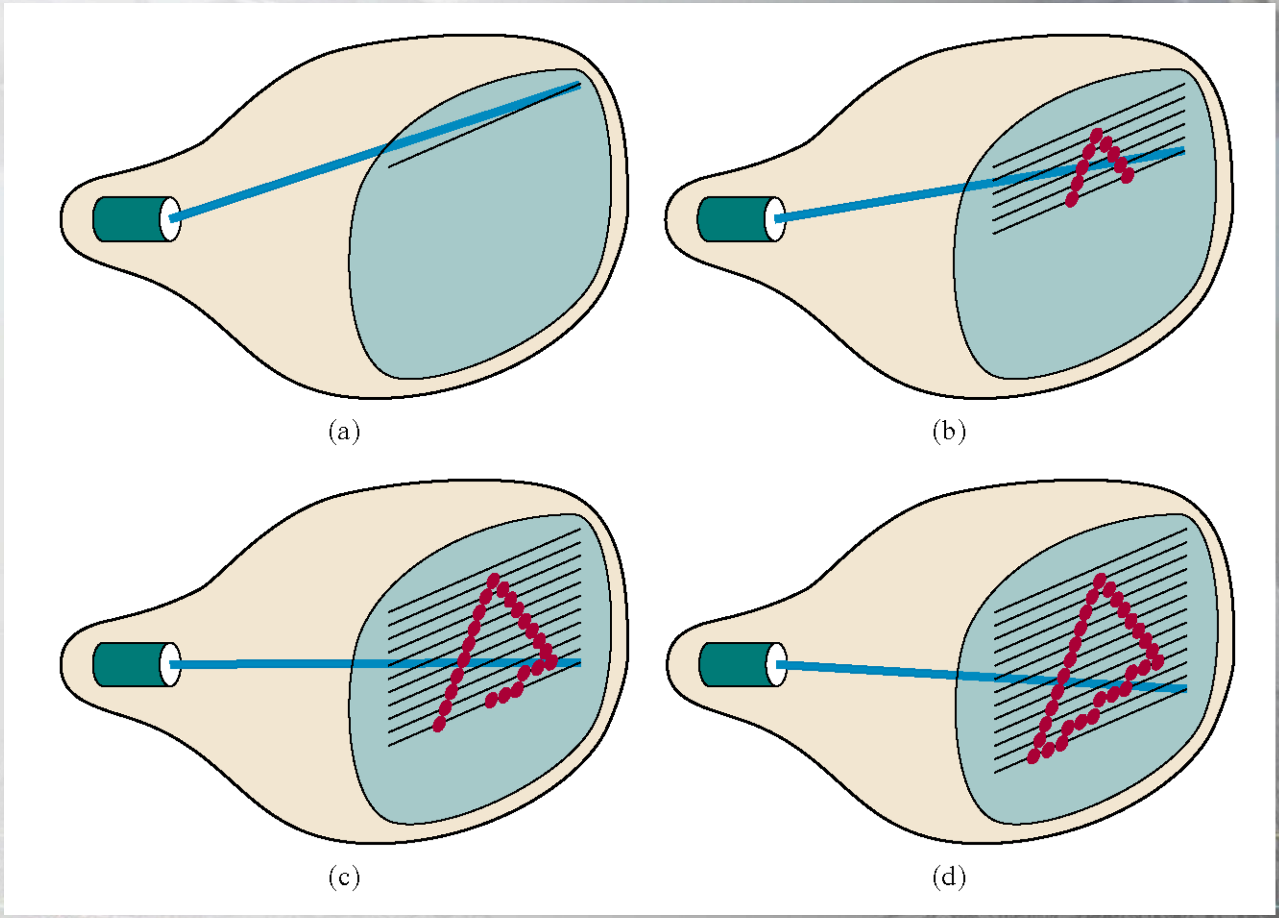




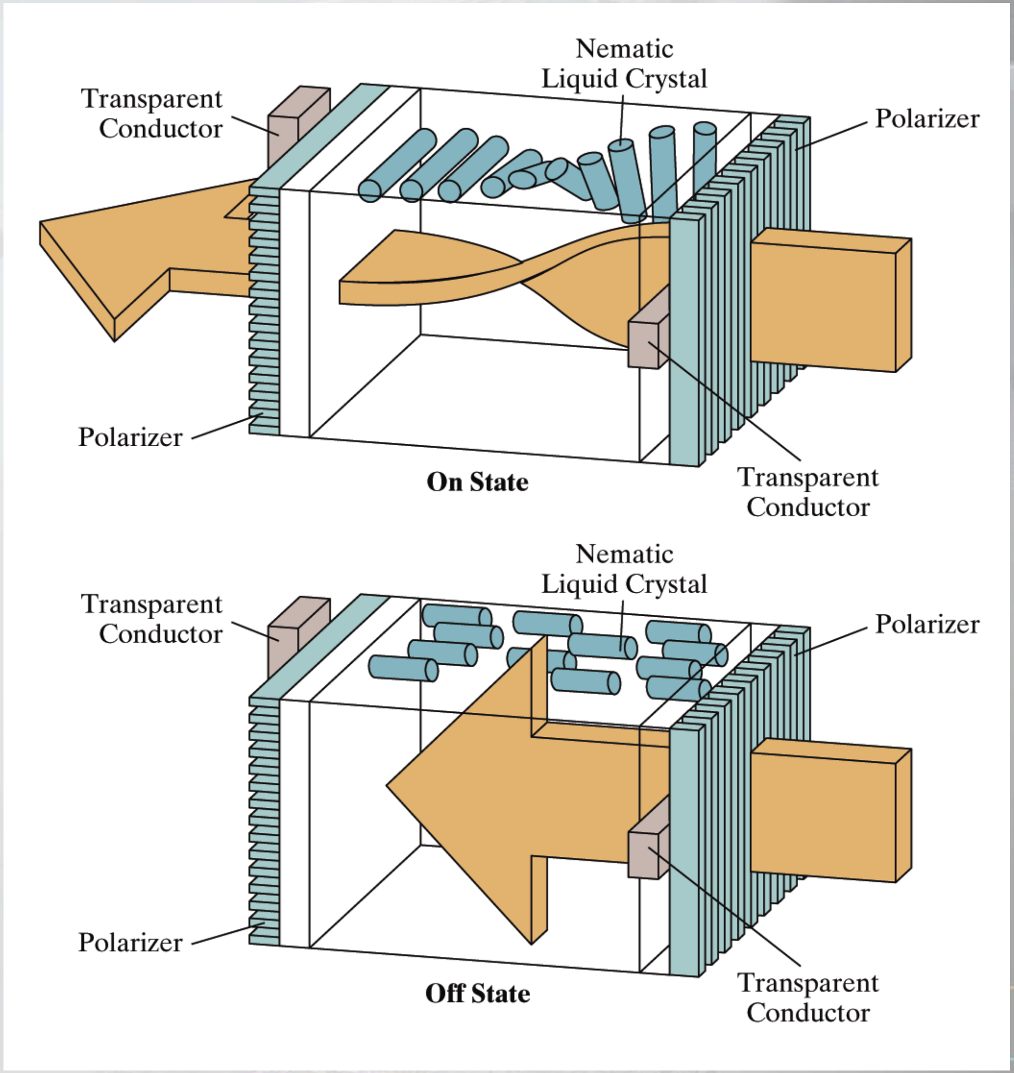
Contents

Raster Scan Systems

- Draw one line at a time



Liquid Crystal Displays

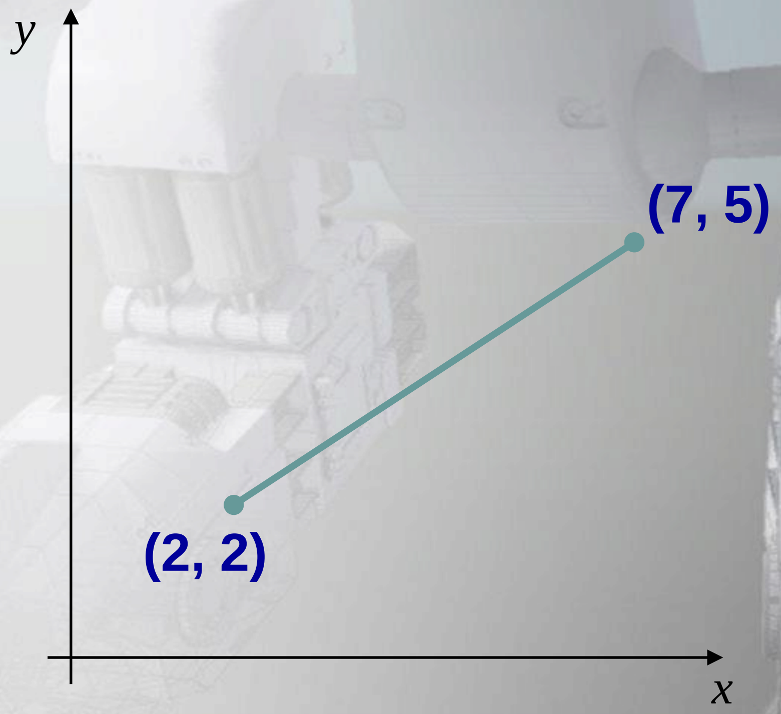


- Light passing through the liquid crystal is twisted so it gets through the polarizer
- A voltage is applied using the crisscrossing conductors to stop the twisting and turn pixels off

Images taken from Hearn & Baker, "Computer Graphics with OpenGL" (2004)

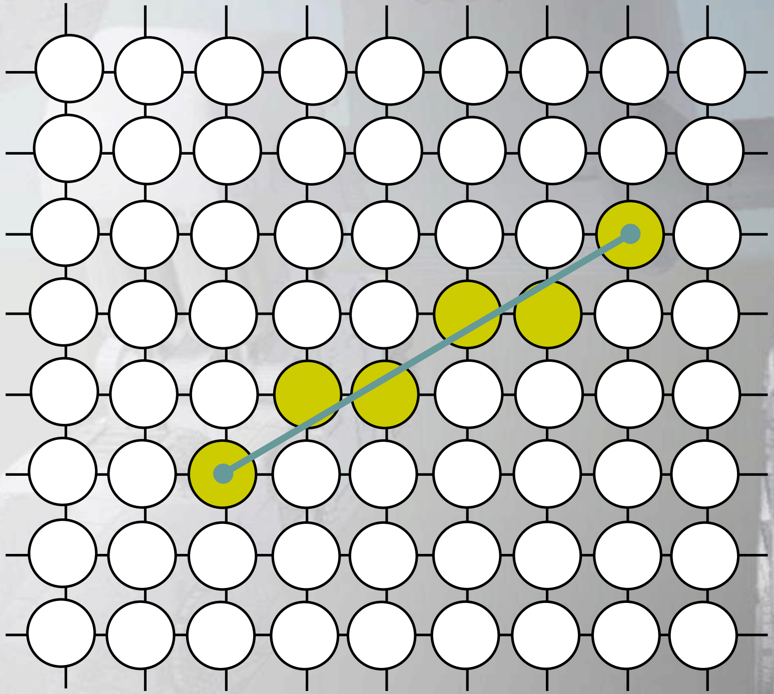
The Problem Of Scan Conversion

- A line segment in a scene is defined by the coordinate positions of the line end-points



The Problem (cont…)

- But what happens when we try to draw this on a pixel based display?



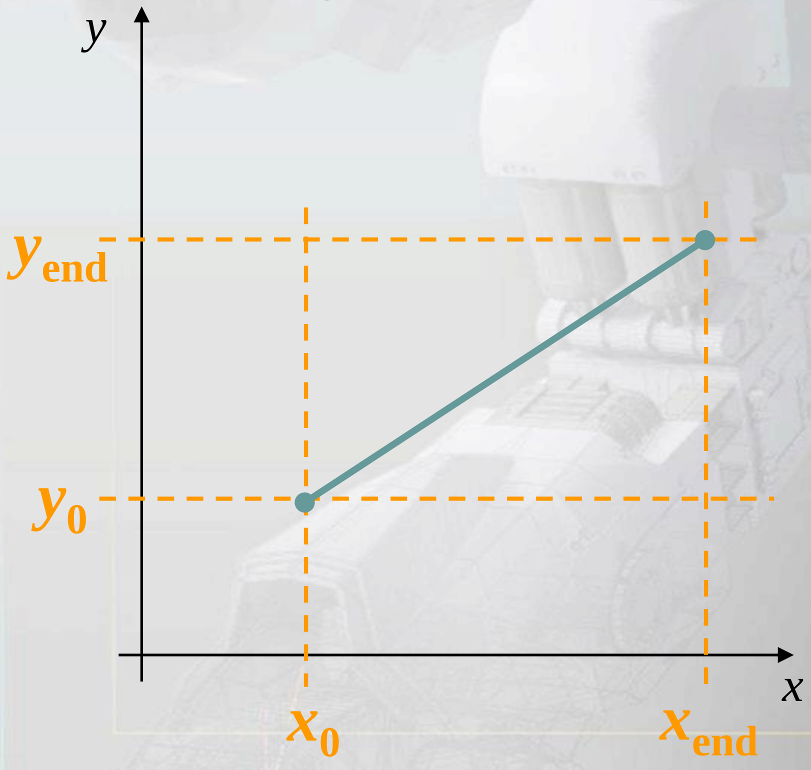
How do we choose which pixels to turn on?

Considerations

- Considerations to keep in mind:
  1. The line has to look good
     - Avoid *jaggies*
  2. It has to be lightening fast!
     - How many lines need to be drawn in a typical scene?

Line Equations

- Let's quickly review the equations involved in drawing lines



Slope-intercept line equation:

$$y = m \cdot x + b$$

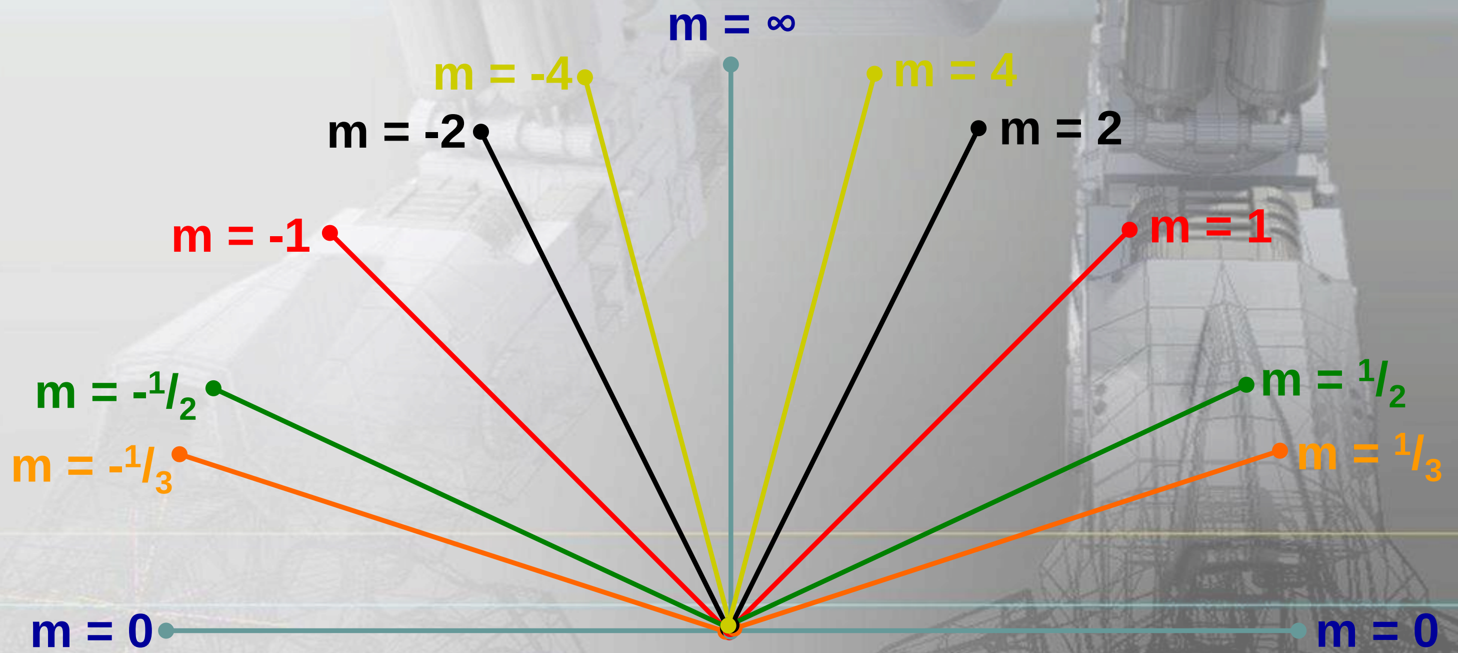
where:

$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$

$$b = y_0 - m \cdot x_0$$

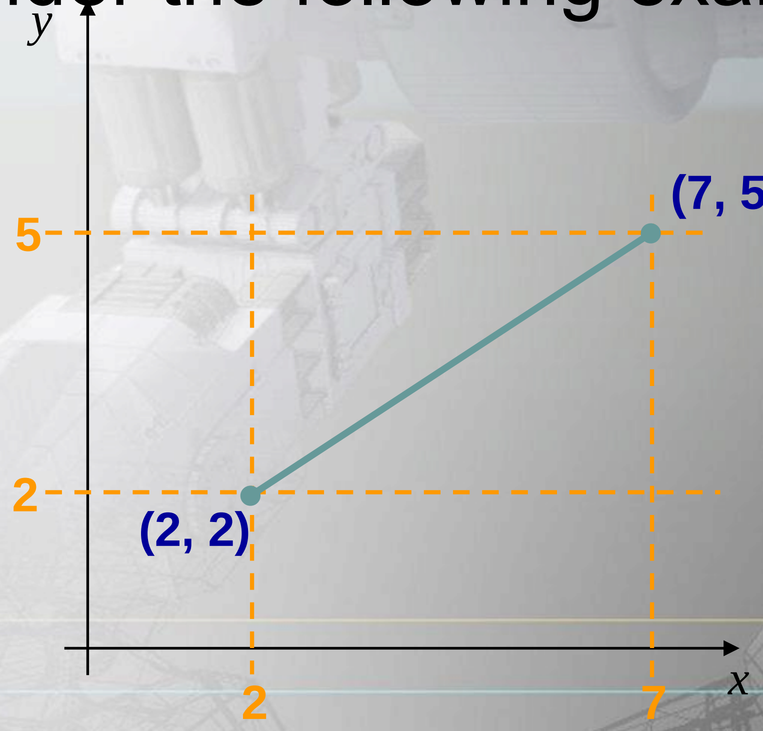
Lines & Slopes

- The slope of a line (m) is defined by its start and end coordinates
- The diagram below shows some examples of lines and their slopes

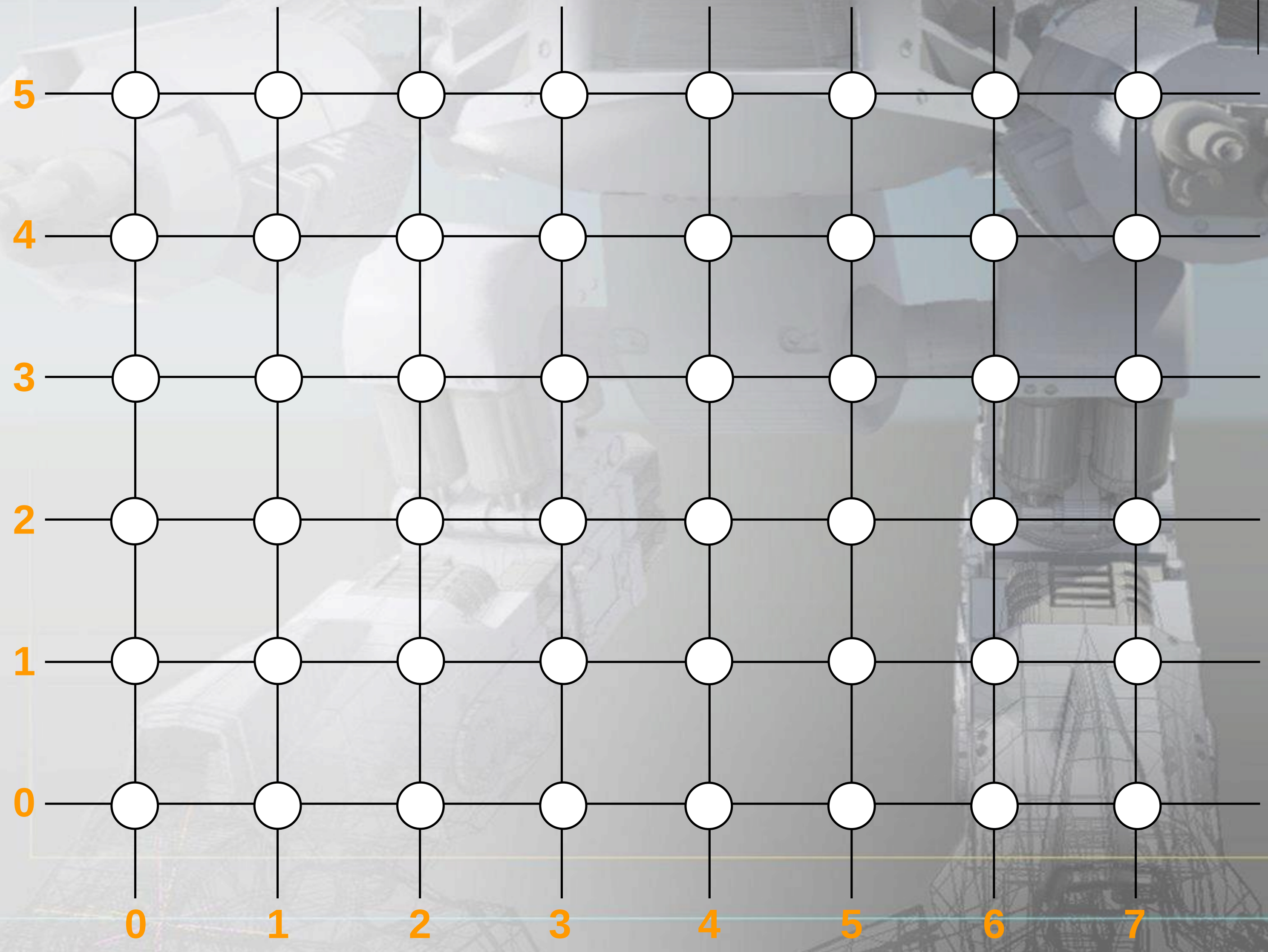


A Very Simple Solution

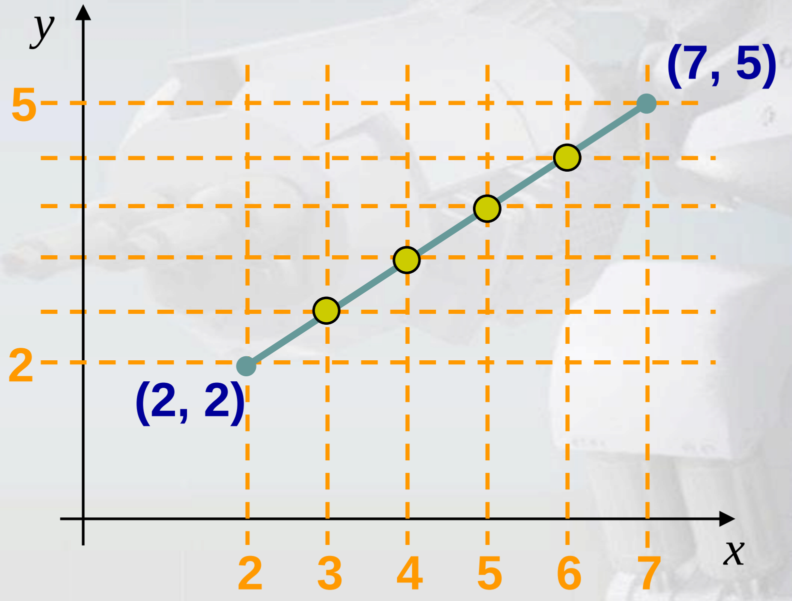
- We could simply work out the corresponding y coordinate for each unit x coordinate
- Let's consider the following example:



A Very Simple Solution (cont…)



A Very Simple Solution (cont…)



- First work out m and b:

$$m = \frac{5-2}{7-2} = \frac{3}{5}$$

$$b = 2 - \frac{3}{5} * 2 = \frac{4}{5}$$

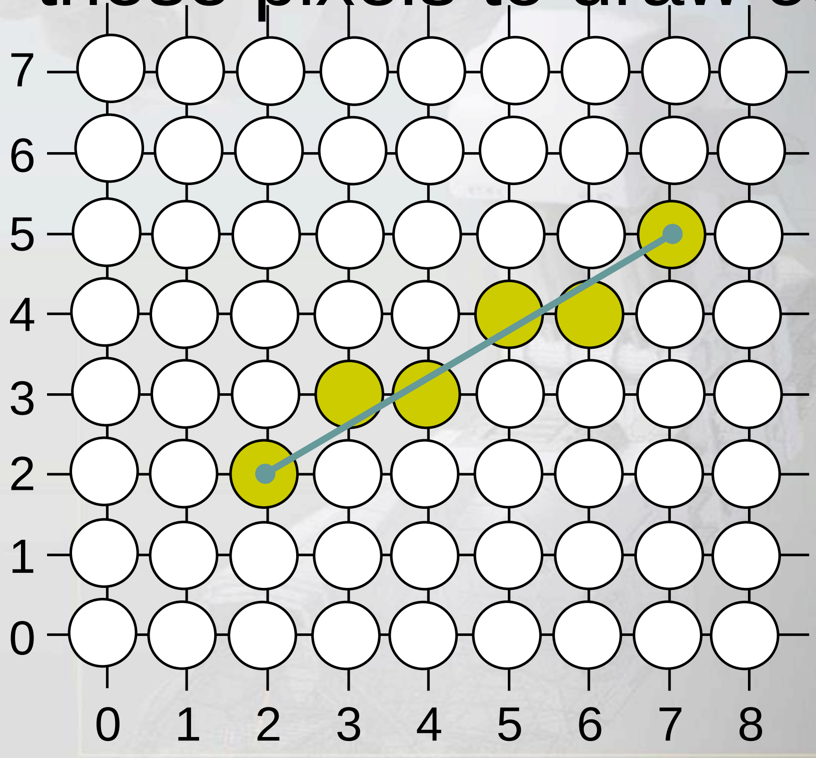
Now for each x value work out the y value:

$$y(3) = \frac{3}{5} \cdot 3 + \frac{4}{5} = 2\frac{3}{5} \qquad y(4) = \frac{3}{5} \cdot 4 + \frac{4}{5} = 3\frac{1}{5}$$

$$y(5) = \frac{3}{5} \cdot 5 + \frac{4}{5} = 3\frac{4}{5} \qquad y(6) = \frac{3}{5} \cdot 6 + \frac{4}{5} = 4\frac{2}{5}$$

A Very Simple Solution (cont…)

- Now just round off the results and turn on these pixels to draw our line



$$y(3) = 2\frac{3}{5} \approx 3$$

$$y(4) = 3\frac{1}{5} \approx 3$$

$$y(5) = 3\frac{4}{5} \approx 4$$

$$y(6) = 4\frac{2}{5} \approx 4$$

A Very Simple Solution (cont…)

- However, this approach is just way too slow
- In particular look out for:
  1. The equation $y = mx + b$ requires the multiplication of m by x
  2. Rounding off the resulting y coordinates
- We need a faster solution

A Quick Note About Slopes

- In the previous example we chose to solve the parametric line equation to give us the y coordinate for each unit x coordinate
- What if we had done it the other way around?
- So this gives us:

$$x = \frac{y - b}{m}$$

- where:

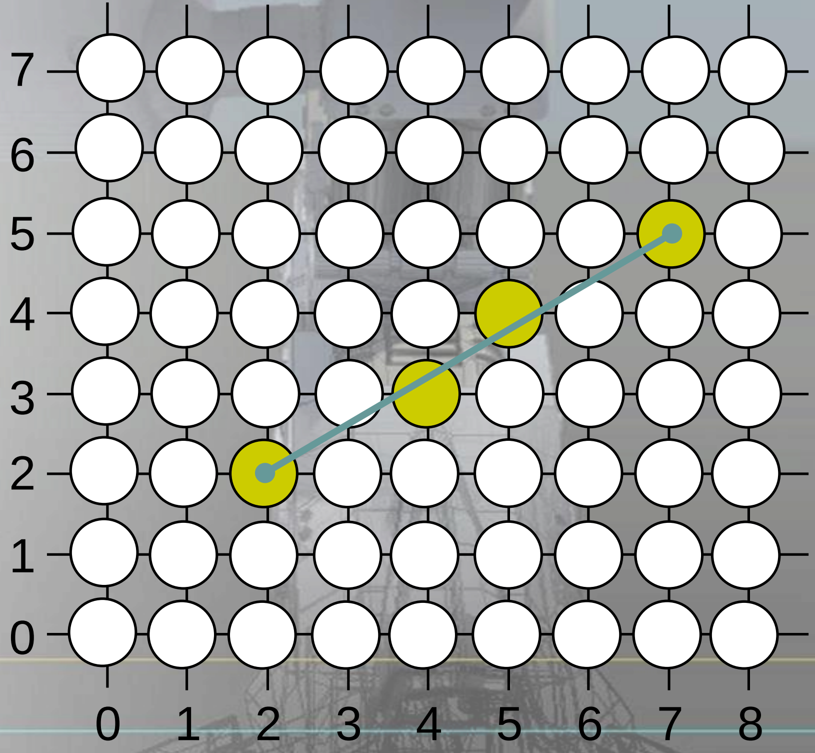
$$m = \frac{y_{end} - y_0}{x_{end} - x_0}$$ and $$b = y_0 - m \cdot x_0$$

A Quick Note About Slopes (cont…)

- Leaving out the details this gives us:

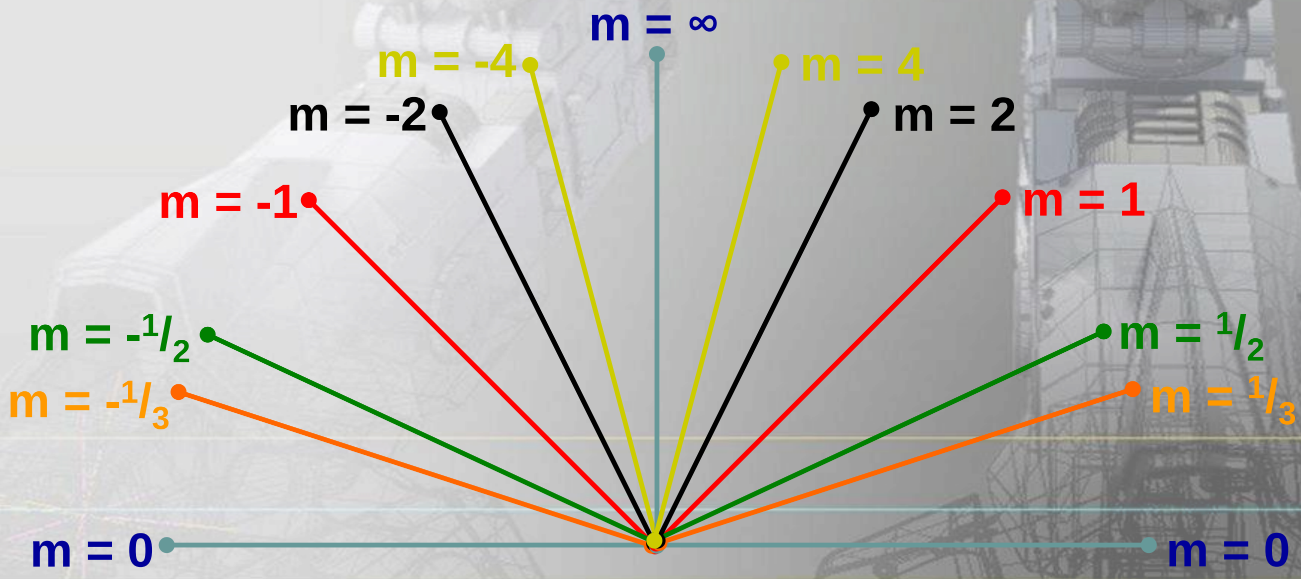
$$x(3) = 3\frac{2}{3} \approx 4 \qquad x(4) = 5\frac{1}{3} \approx 5$$

- We can see easily that this line doesn’t look very good!
- We choose which way to work out the line pixels based on the slope of the line

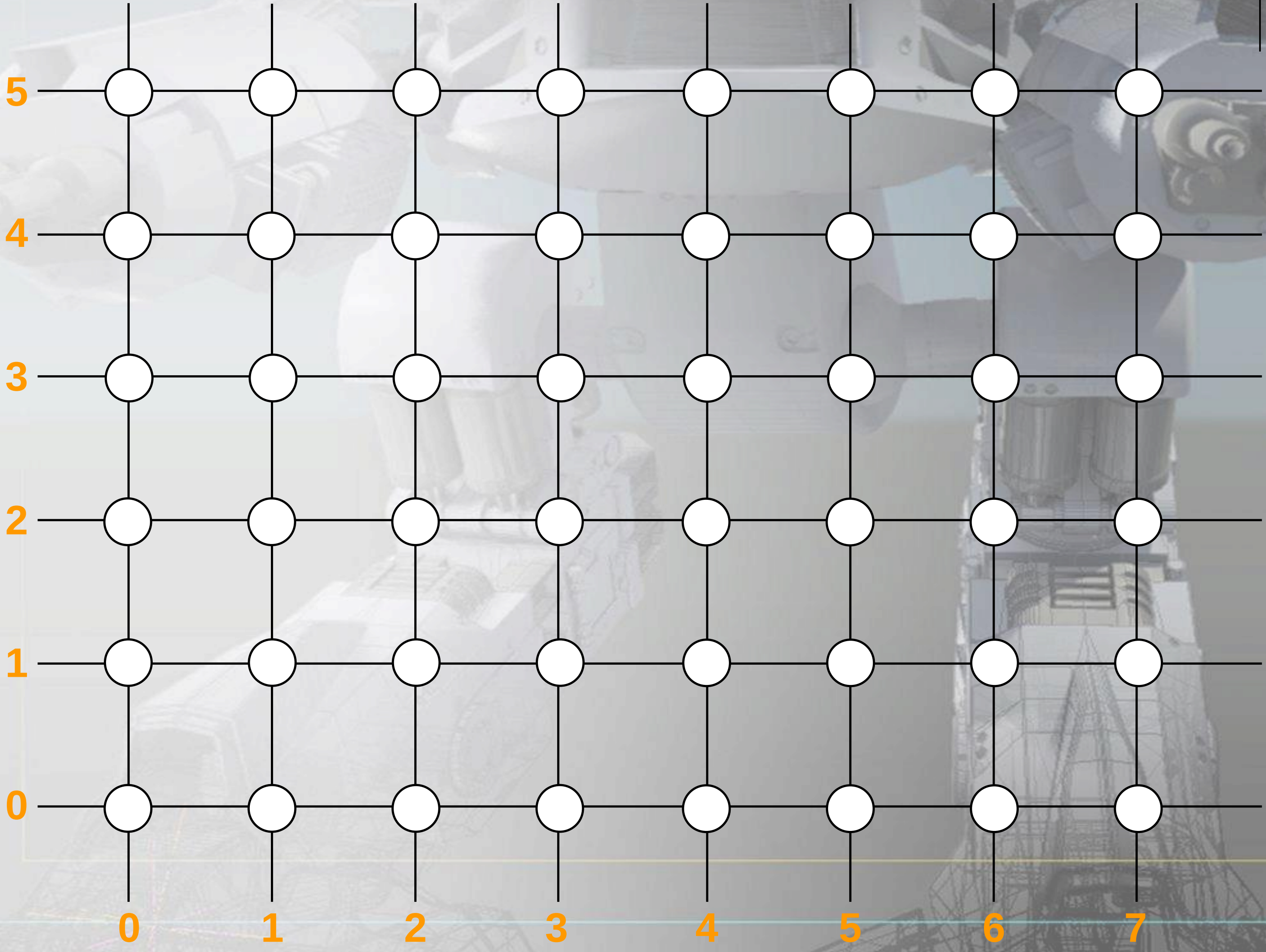


A Quick Note About Slopes (cont…)

- If the slope of a line is between -1 and 1 then we work out the *y* coordinates for a line based on it's unit *x* coordinates
- Otherwise we do the opposite – *x* coordinates are computed based on unit *y* coordinates

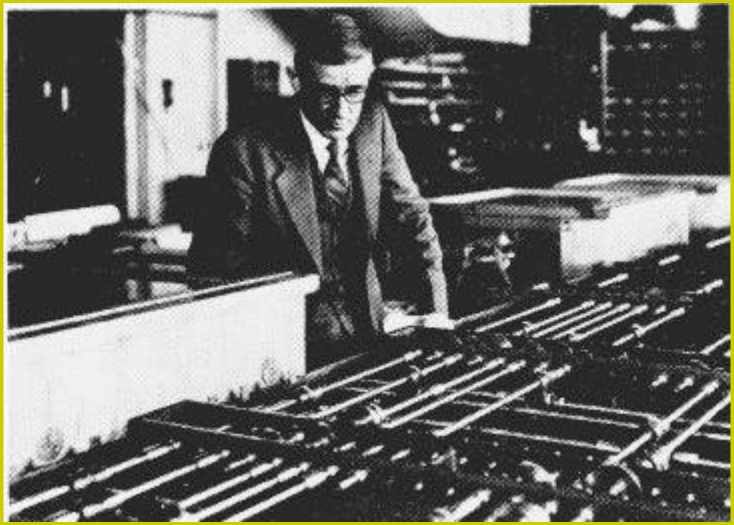


A Quick Note About Slopes (cont…)



The DDA Algorithm

- The *digital differential analyzer* (DDA) algorithm takes an incremental approach in order to speed up scan conversion
- Simply calculate y_{k+1} based on y_k



The original differential analyzer was a physical machine developed by Vannevar Bush at MIT in the 1930’s in order to solve ordinary differential equations.

More information here.

The DDA Algorithm (cont…)

- Consider the list of points that we determined for the line in our previous example:
- $(2, 2), (3, 2^{3}/_{5}), (4, 3^{1}/_{5}), (5, 3^{4}/_{5}), (6, 4^{2}/_{5}), (7, 5)$
- Notice that as the x coordinates go up by one, the y coordinates simply go up by the slope of the line
- This is the key insight in the DDA algorithm

The DDA Algorithm (cont…)

- When the slope of the line is between -1 and 1 begin at the first point in the line and, by incrementing the x coordinate by 1, calculate the corresponding y coordinates as follows:

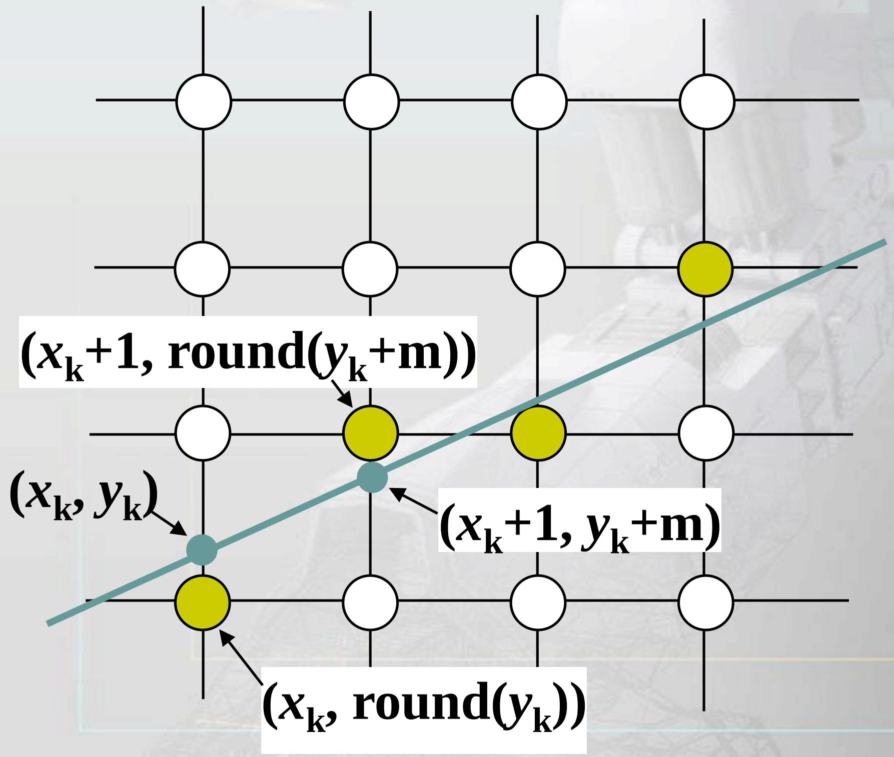
$$y_{k+1} = y_k + m$$

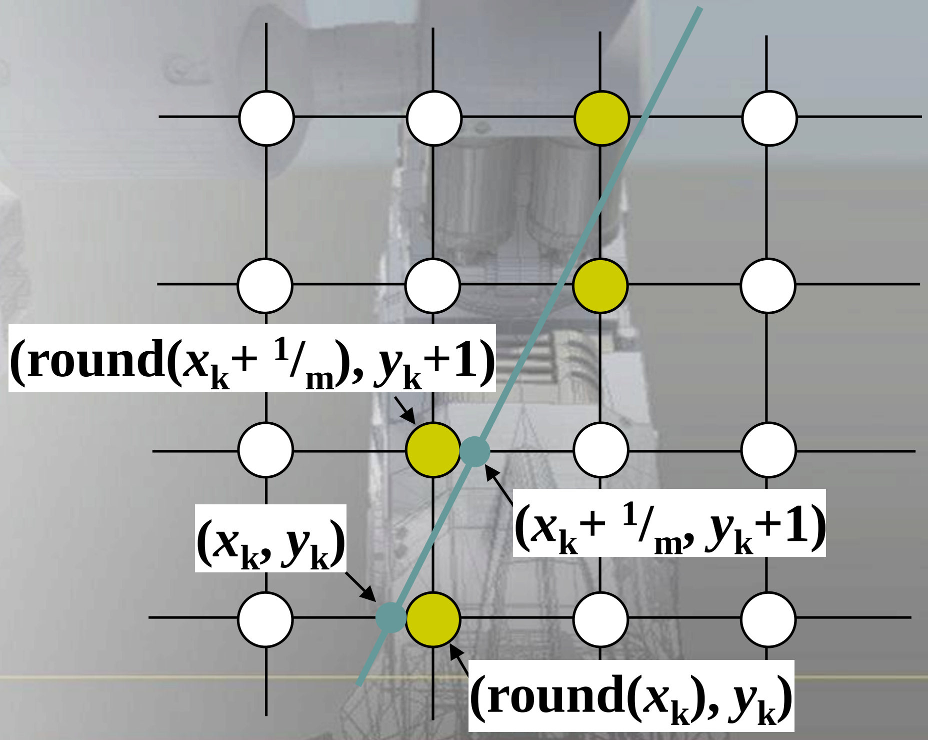
- When the slope is outside these limits, increment the y coordinate by 1 and calculate the corresponding x coordinates as follows:

$$x_{k+1} = x_k + \frac{1}{m}$$

The DDA Algorithm (cont…)

- Again the values calculated by the equations used by the DDA algorithm must be rounded to match pixel values

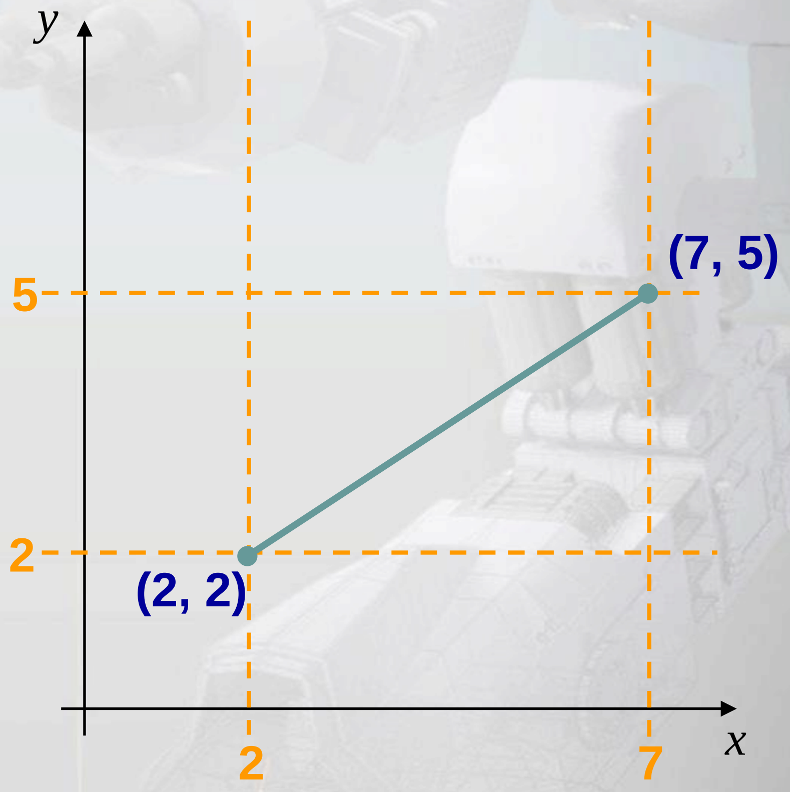


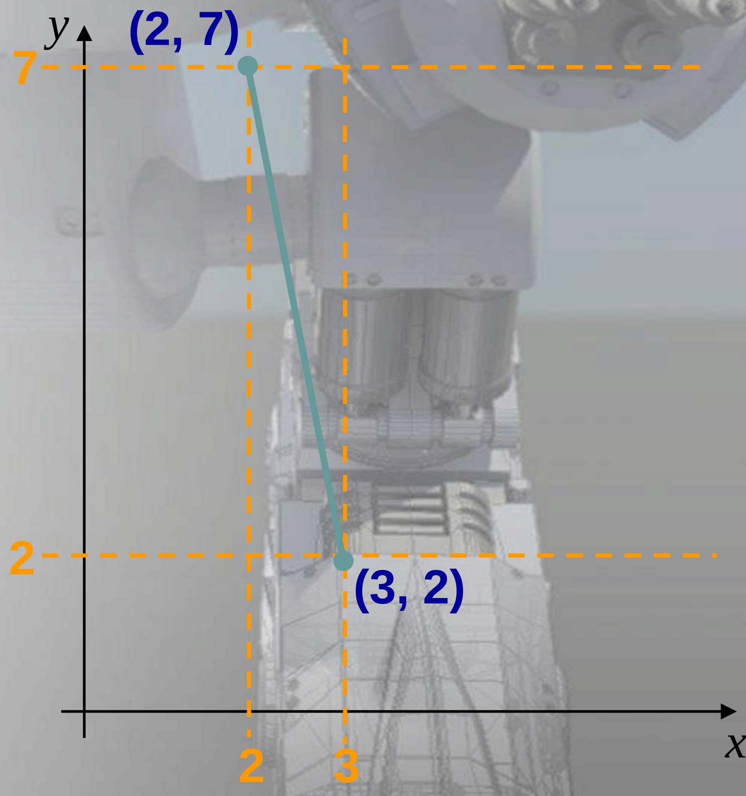


$(\text{round}(x_k+ {}^1/_m), y_k+1)$
(x_k, y_k)
$(x_k+ {}^1/_m, y_k+1)$
$(\text{round}(x_k), y_k)$

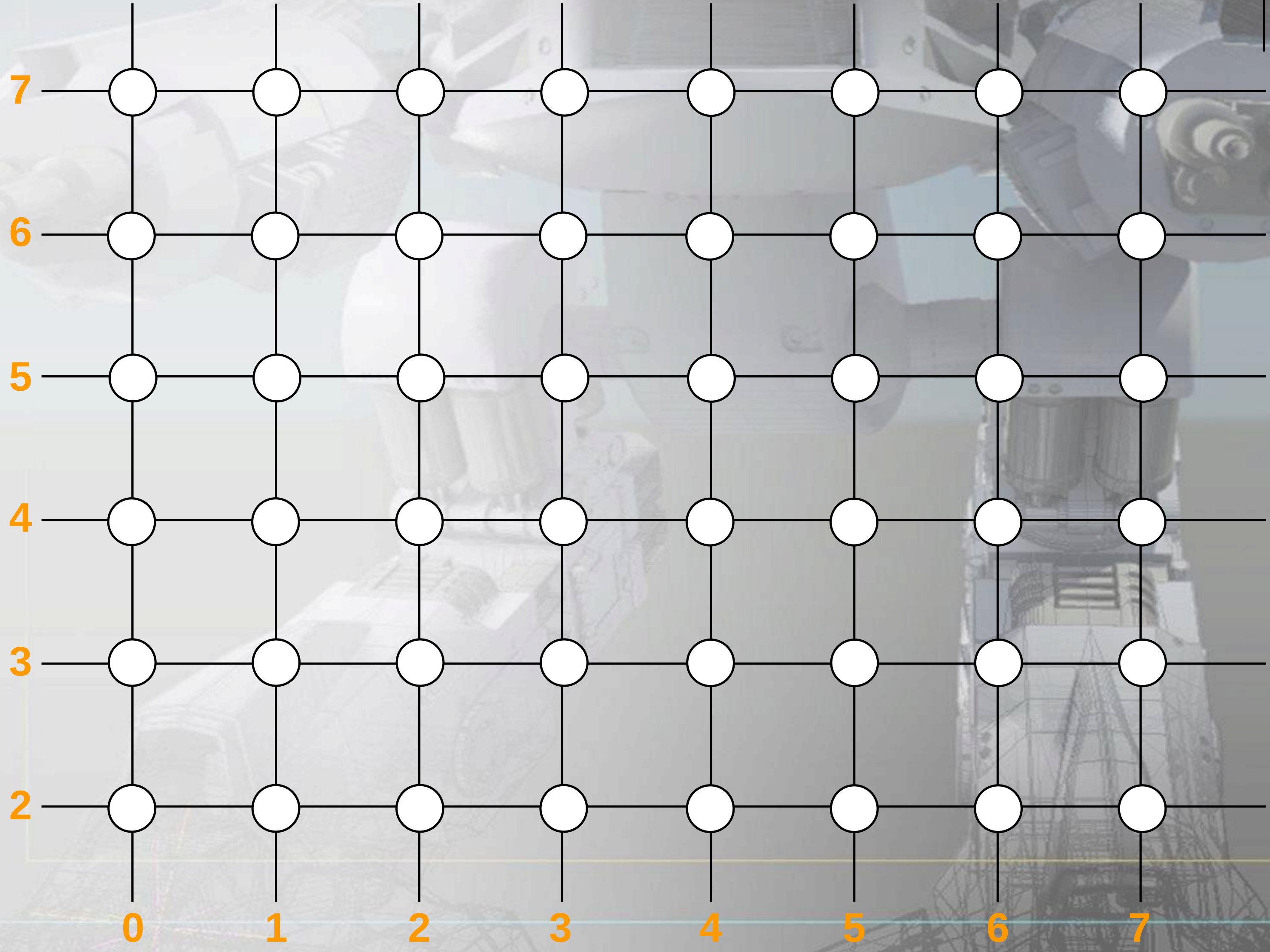
DDA Algorithm Example

- Let’s try out the following examples:





DDA Algorithm Example (cont…)



The DDA Algorithm Summary

- The DDA algorithm is much faster than our previous attempt
    1. In particular, there are no longer any multiplications involved
- However, there are still two big issues:
    1. Accumulation of round-off errors can make the pixelated line drift away from what was intended
    2. The rounding operations and floating point arithmetic involved are time consuming

BRESENHAM LINE DRAWING ALGORITHM, CIRCLE DRAWING & POLYGON FILLING

Contents

i. Bresenham’s line drawing algorithm

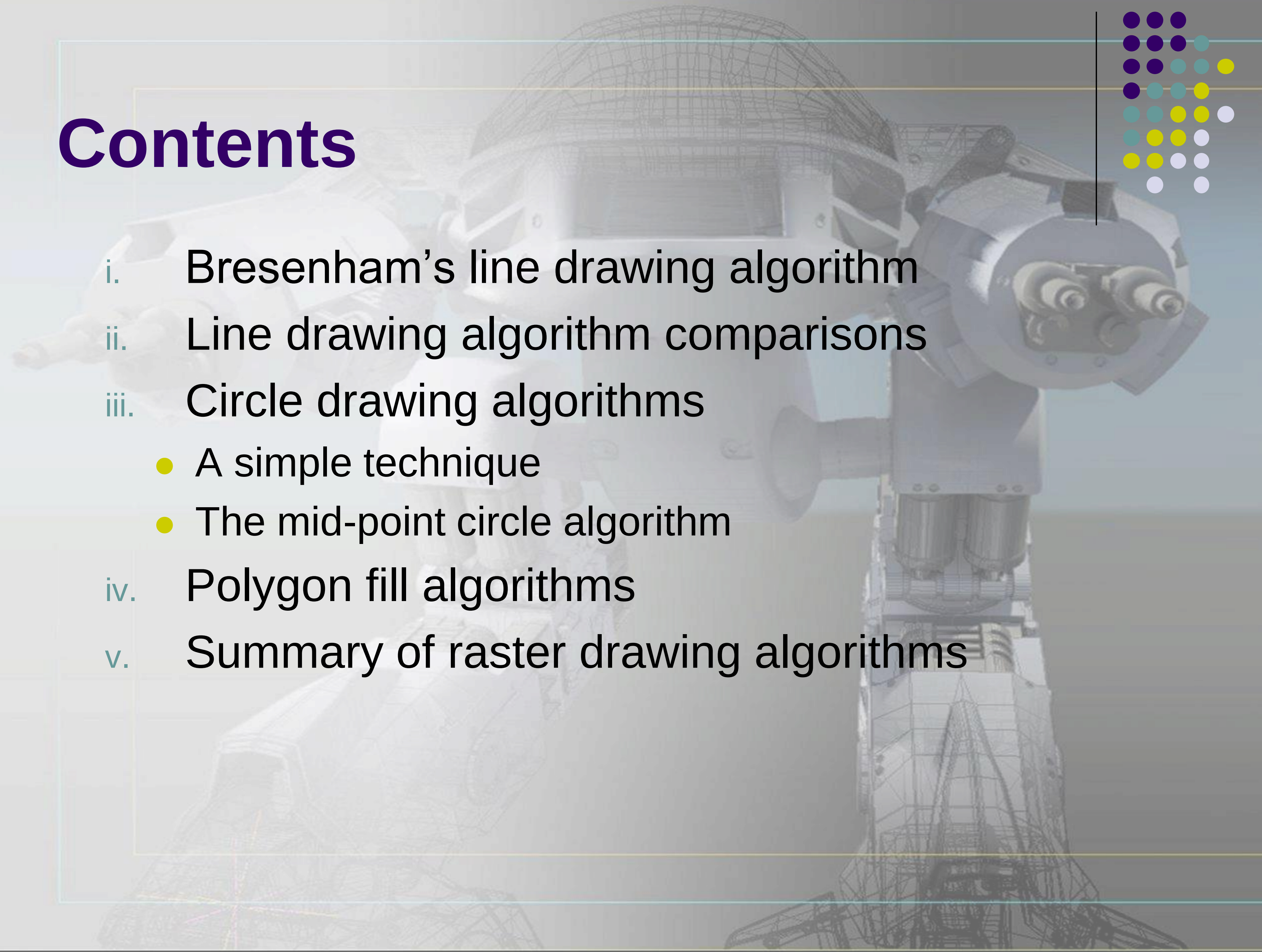
ii. Line drawing algorithm comparisons

iii. Circle drawing algorithms

- A simple technique
- The mid-point circle algorithm

iv. Polygon fill algorithms

v. Summary of raster drawing algorithms



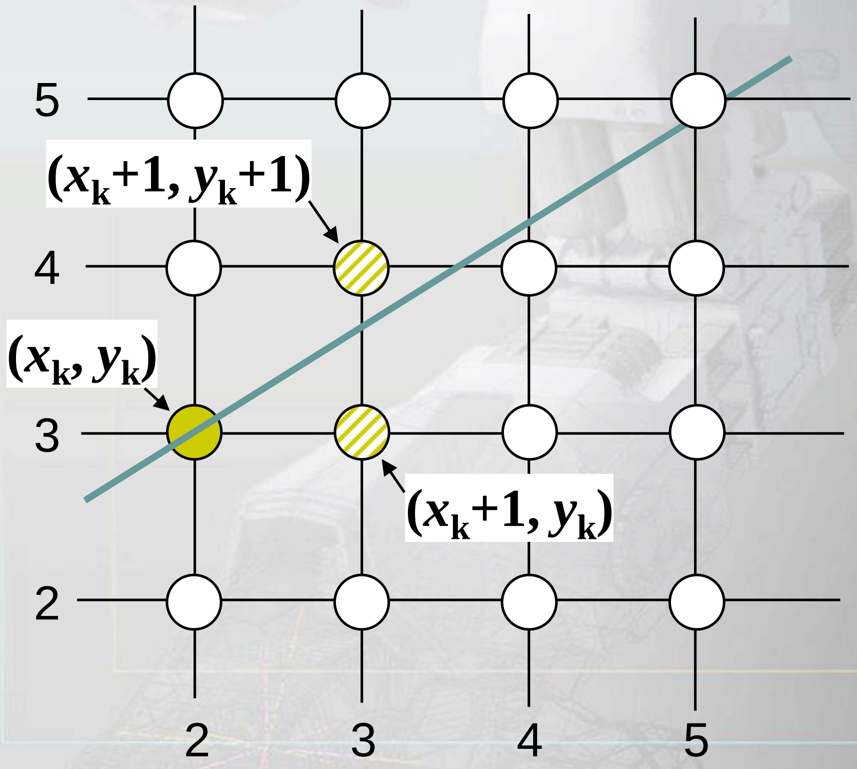
The Bresenham Line Algorithm

- The Bresenham algorithm is another incremental scan conversion algorithm
- The big advantage of this algorithm is that it uses only integer calculations

Jack Bresenham worked for 27 years at IBM before entering academia. Bresenham developed his famous algorithms at IBM in the early 1960s

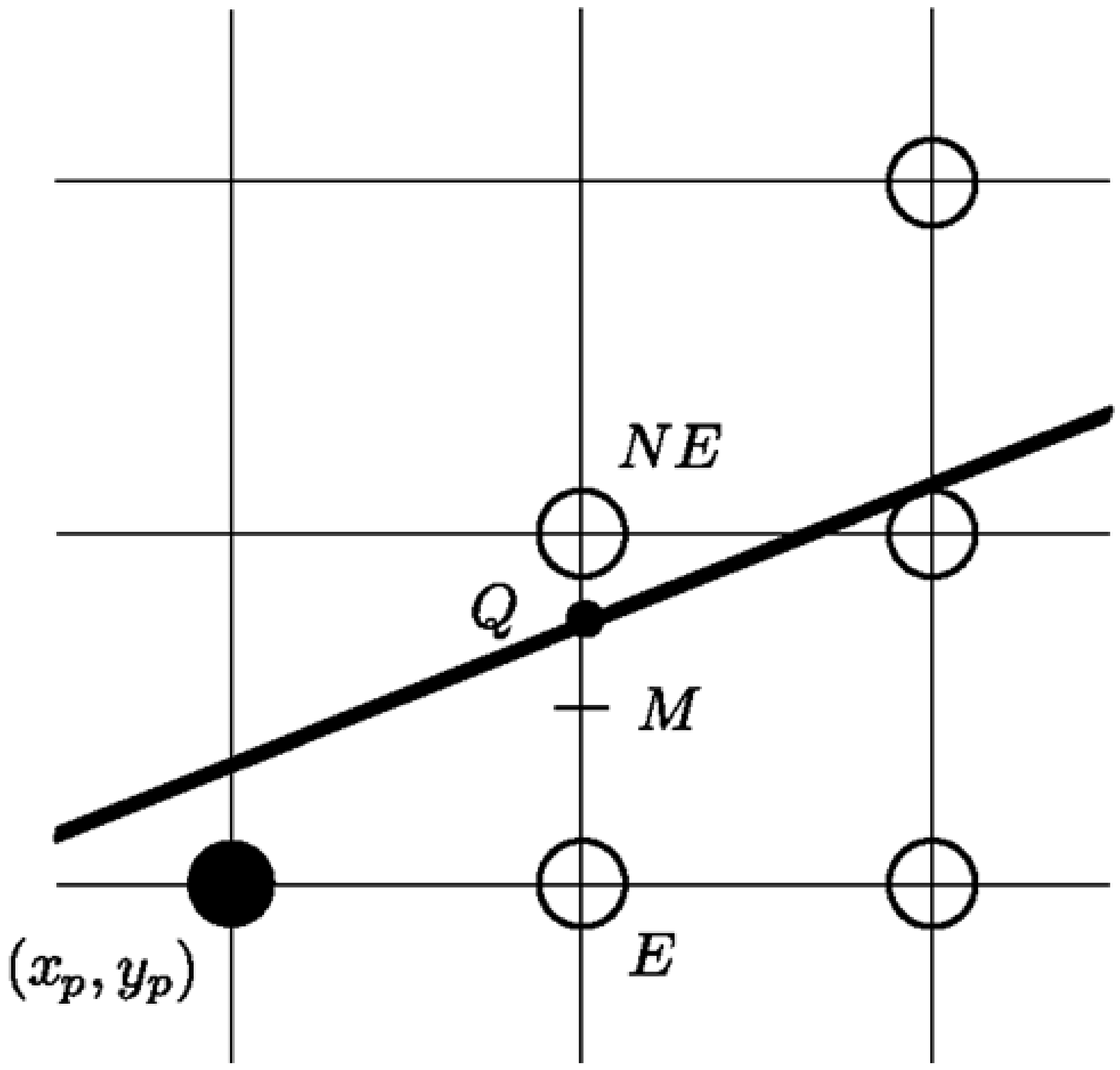
The Big Idea

- Move across the x axis in unit intervals and at each step choose between two different y coordinates



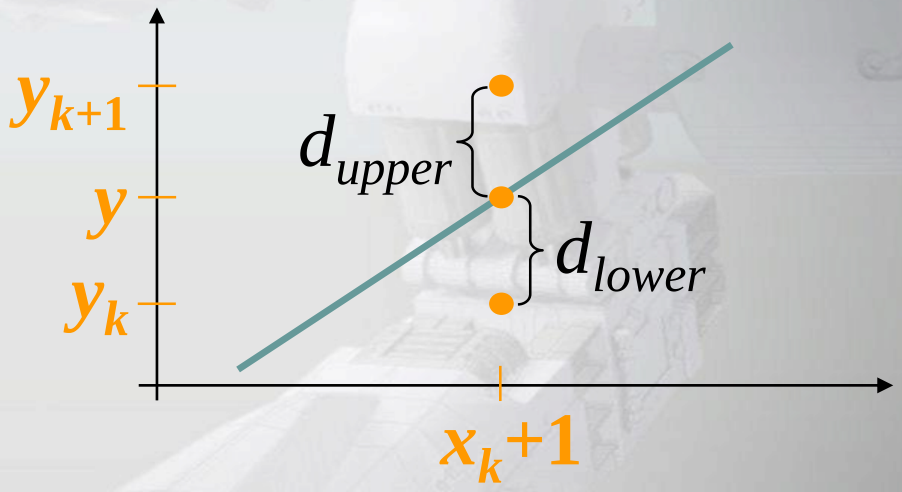
For example, from position (2, 3) we have to choose between (3, 3) and (3, 4)

We would like the point that is closer to the original line


NE
Q
M
E
(x_p, y_p)

Deriving The Bresenham Line Algorithm

- At sample position x_k+1 the vertical separations from the mathematical line are labelled d_{upper} and d_{lower}



The y coordinate on the mathematical line at x_k+1 is:

$$y = m(x_k + 1) + b$$

Deriving The Bresenham Line Algorithm (cont…)

- So, d_{upper} and d_{lower} are given as follows:

$$\begin{aligned} d_{lower} &= y - y_k \\ &= m(x_k + 1) + b - y_k \end{aligned}$$

- and:

$$\begin{aligned} d_{upper} &= (y_k + 1) - y \\ &= y_k + 1 - m(x_k + 1) - b \end{aligned}$$

- We can use these to make a simple decision about which pixel is closer to the mathematical line

Deriving The Bresenham Line Algorithm (cont…)

- This simple decision is based on the difference between the two pixel positions:

$$d_{lower} - d_{upper} = 2m(x_k + 1) - 2y_k + 2b - 1$$

- Let's substitute *m* with $\Delta y / \Delta x$ where Δx and Δy are the differences between the end-points:

$$\Delta x(d_{lower} - d_{upper}) = \Delta x(2\frac{\Delta y}{\Delta x}(x_k + 1) - 2y_k + 2b - 1)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + 2\Delta y + \Delta x(2b - 1)$$

$$= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c$$

Deriving The Bresenham Line Algorithm (cont…)

- So, a decision parameter p_k for the kth step along a line is given by:

$$\begin{aligned} p_k &= \Delta x(d_{lower} - d_{upper}) \\ &= 2\Delta y \cdot x_k - 2\Delta x \cdot y_k + c \end{aligned}$$

- The sign of the decision parameter p_k is the same as that of $d_{lower} - d_{upper}$
- If p_k is negative, then we choose the lower pixel, otherwise we choose the upper pixel

Deriving The Bresenham Line Algorithm (cont…)

- Remember coordinate changes occur along the x axis in unit steps so we can do everything with integer calculations
- At step k+1 the decision parameter is given as:

$$p_{k+1} = 2\Delta y \cdot x_{k+1} - 2\Delta x \cdot y_{k+1} + c$$

- Subtracting p_k from this we get:

$$p_{k+1} - p_k = 2\Delta y(x_{k+1} - x_k) - 2\Delta x(y_{k+1} - y_k)$$

Deriving The Bresenham Line Algorithm (cont…)

- But, x_{k+1} is the same as x_k+1 so:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x(y_{k+1} - y_k)$$

- where y_{k+1} - y_k is either 0 or 1 depending on the sign of p_k
- The first decision parameter p0 is evaluated at (x0, y0) is given as:

$$p_0 = 2\Delta y - \Delta x$$

The Bresenham Line Algorithm

BRESENHAM'S LINE DRAWING ALGORITHM
(for $|m| < 1.0$)

1. Input the two line end-points, storing the left end-point in (x_0, y_0)
2. Plot the point (x_0, y_0)
3. Calculate the constants Δx, Δy, $2\Delta y$, and $(2\Delta y - 2\Delta x)$ and get the first value for the decision parameter as:

$$p_0 = 2\Delta y - \Delta x$$

4. At each x_k along the line, starting at $k = 0$, perform the following test. If $p_k < 0$, the next point to plot is (x_k+1, y_k) and:

$$p_{k+1} = p_k + 2\Delta y$$

The Bresenham Line Algorithm (cont…)

Otherwise, the next point to plot is (x_k+1, y_k+1) and:

$$p_{k+1} = p_k + 2\Delta y - 2\Delta x$$

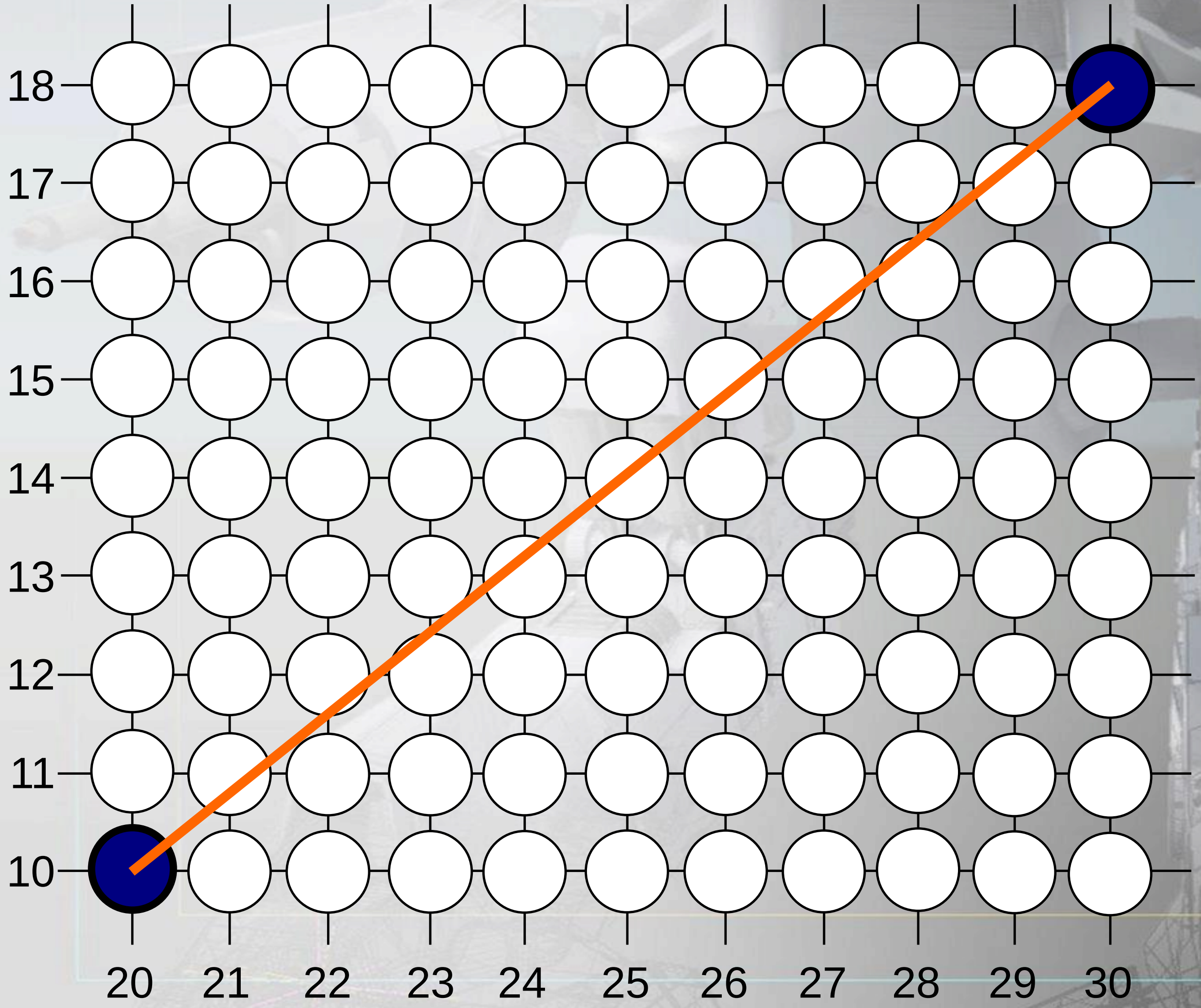
5. Repeat step 4 (Δx – 1) times

- **Be careful!** The algorithm and derivation above assumes slopes are less than 1.
- for other slopes we need to adjust the algorithm slightly

Bresenham Example

- Let’s have a go at this
- Let’s plot the line from (20, 10) to (30, 18)
- First off calculate all of the constants:
   i. Δx: 10
   ii. Δy: 8
   iii. $2\Delta y$: 16
   iv. $2\Delta y - 2\Delta x$: -4
- Calculate the initial decision parameter p_0:
   i. $p0 = 2\Delta y - \Delta x = 6$

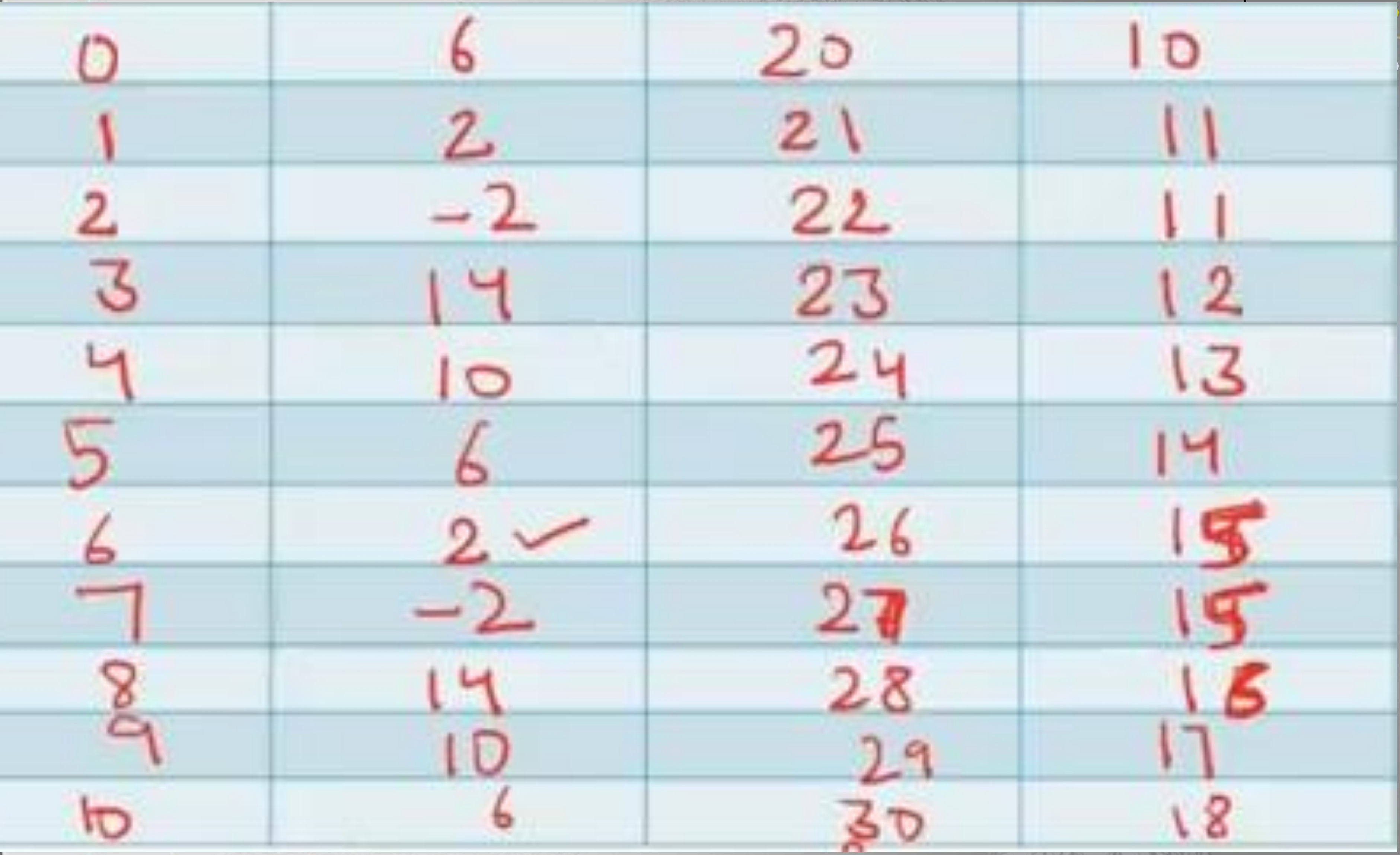
Bresenham Example (cont…)



k	p_k	(x_{k+1},y_{k+1})
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		

$$\begin{cases} \Delta y = 18 - 10 = 8 \\ \Delta x = 30 - 20 = 10 \\ 2\Delta y = 16 \\ 2\Delta x = 20 \end{cases}$$

$$\begin{cases} 2\Delta y - 2\Delta x = -4 \\ 2\Delta y - \Delta x = P_0 = \underline{\underline{6}} \end{cases}$$



0	6	20	10
1	2	21	11
2	-2	22	11
3	14	23	12
4	10	24	13
5	6	25	14
6	2 ✓	26	15
7	-2	27	15
8	14	28	16
9	10	29	17
10	6	30	18

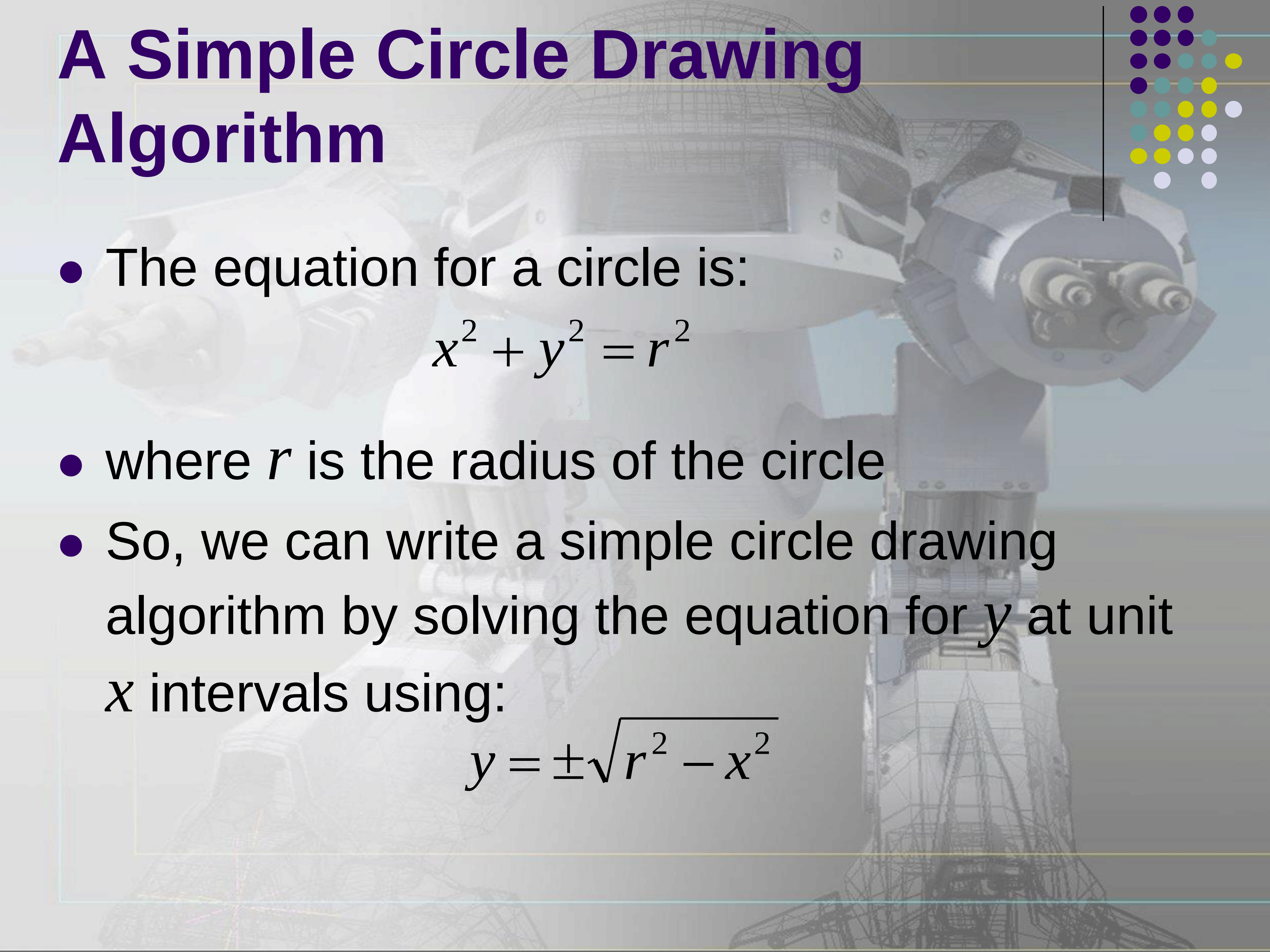
Bresenham Exercise

- Go through the steps of the Bresenham line drawing algorithm for a line going from (21,12) to (29,16)

Bresenham Line Algorithm Summary

- The Bresenham line algorithm has the following advantages:
  i. An fast incremental algorithm
  ii. Uses only integer calculations
- Comparing this to the DDA algorithm, DDA has the following problems:
  i. Accumulation of round-off errors can make the pixelated line drift away from what was intended
  ii. The rounding operations and floating point arithmetic involved are time consuming

A Simple Circle Drawing Algorithm



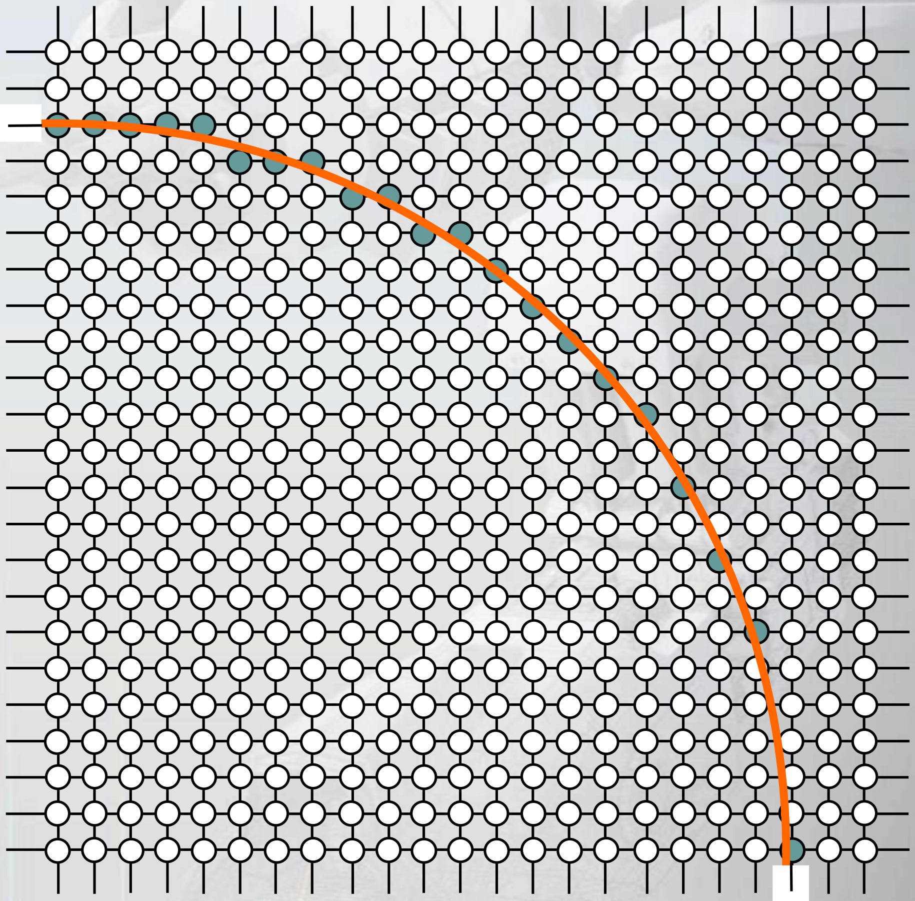
- The equation for a circle is:

$$x^2 + y^2 = r^2$$

- where *r* is the radius of the circle
- So, we can write a simple circle drawing algorithm by solving the equation for *y* at unit *x* intervals using:

$$y = \pm\sqrt{r^2 - x^2}$$

A Simple Circle Drawing Algorithm (cont…)



$$y_0 = \sqrt{20^2 - 0^2} \approx 20$$

$$y_1 = \sqrt{20^2 - 1^2} \approx 20$$

$$y_2 = \sqrt{20^2 - 2^2} \approx 20$$

⋮

$$y_{19} = \sqrt{20^2 - 19^2} \approx 6$$

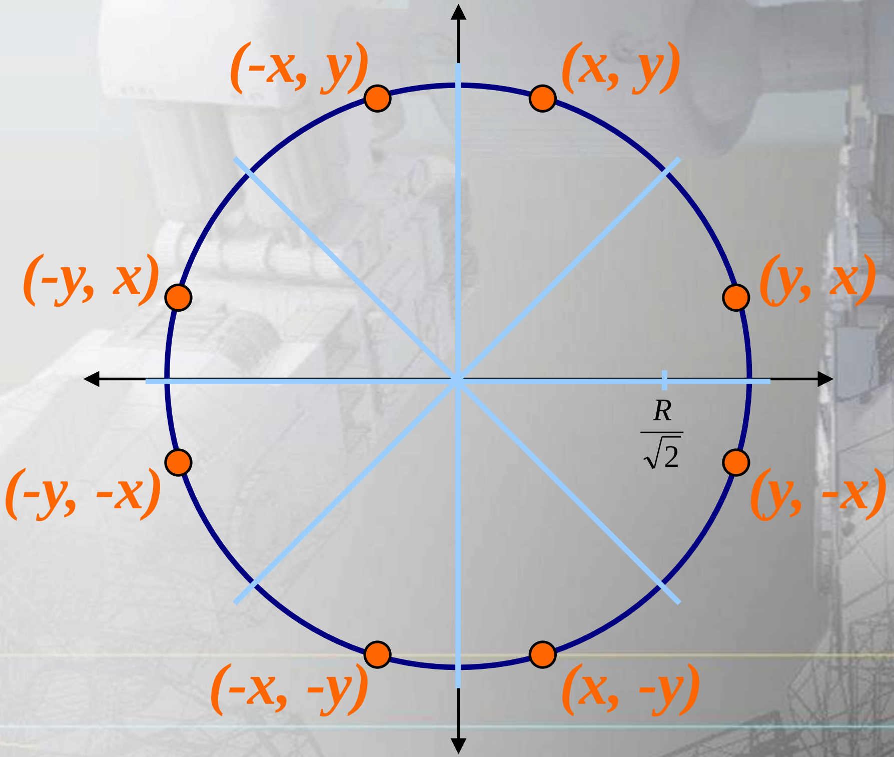
$$y_{20} = \sqrt{20^2 - 20^2} \approx 0$$

A Simple Circle Drawing Algorithm (cont…)

- However, unsurprisingly this is not a brilliant solution!
- Firstly, the resulting circle has large gaps where the slope approaches the vertical
- Secondly, the calculations are not very efficient
  - i. The square (multiply) operations
  - ii. The square root operation – try really hard to avoid these!
- We need a more efficient, more accurate solution

Eight-Way Symmetry

- The first thing we can notice to make our circle drawing algorithm more efficient is that circles centred at $(0, 0)$ have *eight-way symmetry*



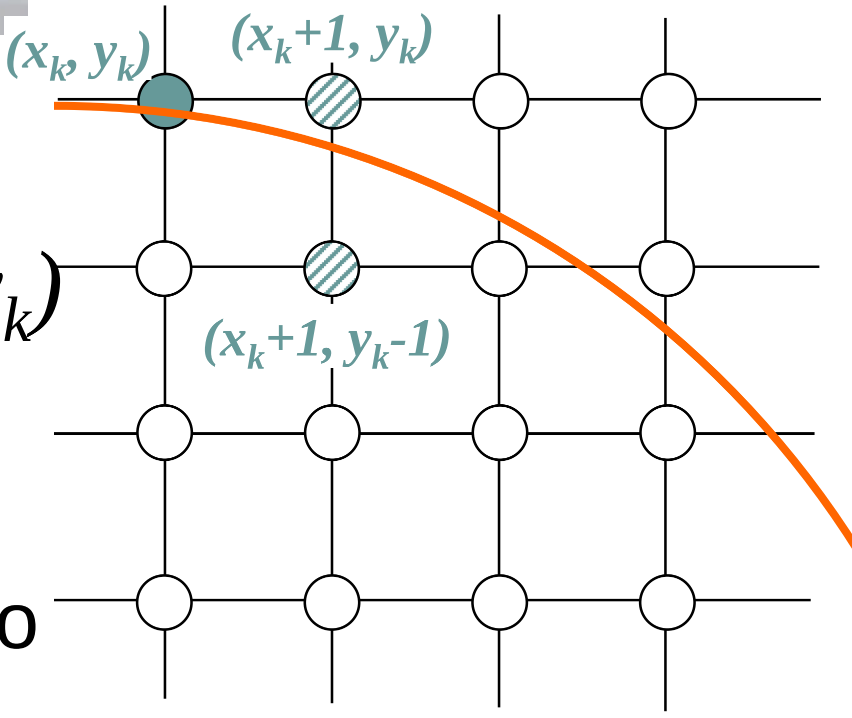
Mid-Point Circle Algorithm

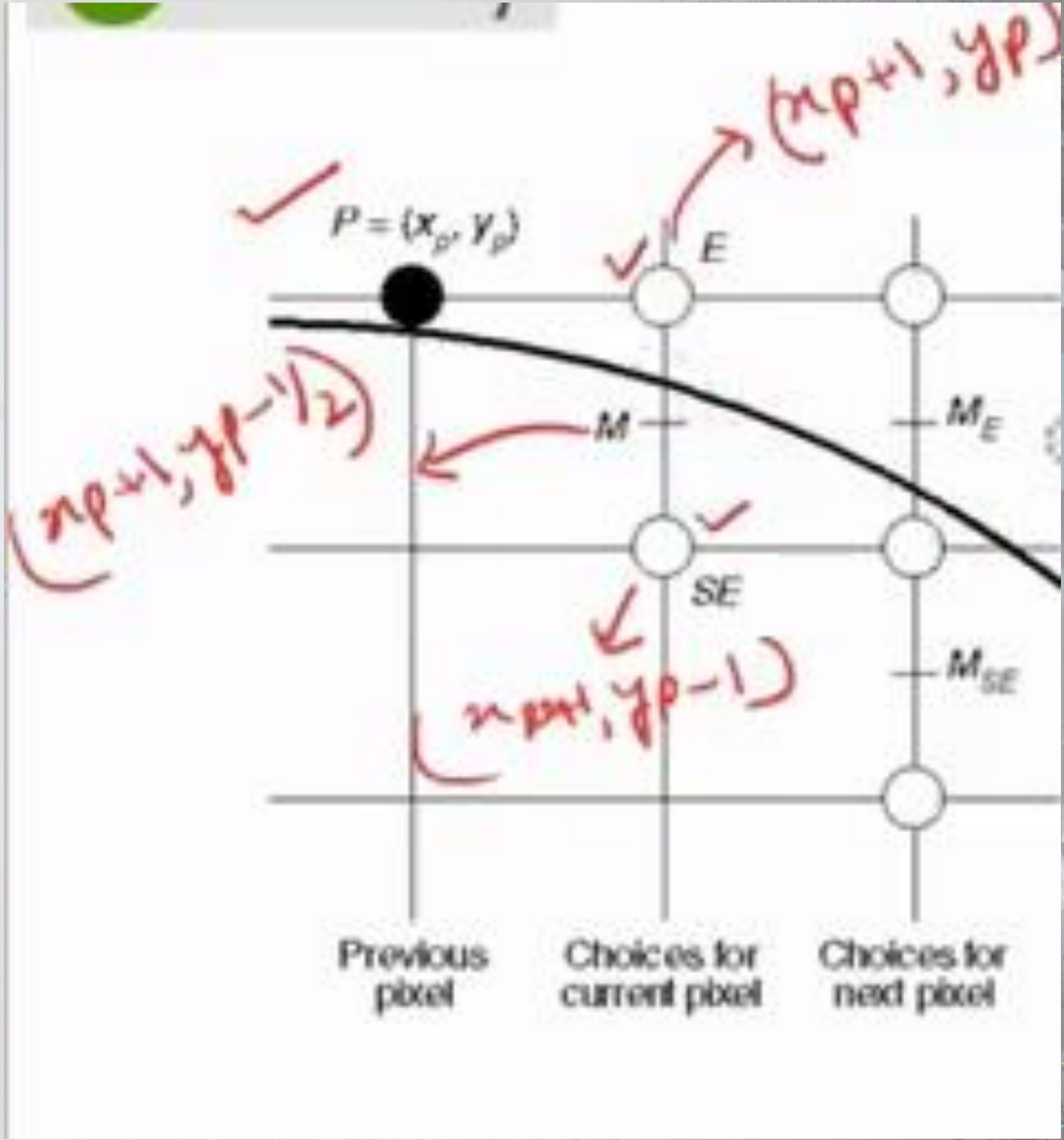
- Similarly to the case with lines, there is an incremental algorithm for drawing circles – the *mid-point circle algorithm*
- In the mid-point circle algorithm we use eight-way symmetry so only ever calculate the points for the top right eighth of a circle, and then use symmetry to get the rest of the points

The mid-point circle algorithm was developed by Jack Bresenham

Mid-Point Circle Algorithm (cont…)

- Assume that we have just plotted point (x_k, y_k)
- The next point is a choice between (x_k+1, y_k) and (x_k+1, y_k-1)
- We would like to choose the point that is nearest to the actual circle
- So how do we make this choice?





$P = (x_p, y_p)$
(x_p+1, y_p)
E
M
M_E
$(x_p+1, y_p-1/2)$
SE
M_{SE}
(x_p+1, y_p-1)
Previous pixel
Choices for current pixel
Choices for next pixel

Mid-Point Circle Algorithm (cont…)

- Let’s re-jig the equation of the circle slightly to give us:

$$f_{circ}(x, y) = x^2 + y^2 - r^2$$

- The equation evaluates as follows:

$$f_{circ}(x, y) \begin{cases} < 0, \text{ if } (x, y) \text{ is inside the circle boundary} \\ = 0, \text{ if } (x, y) \text{ is on the circle boundary} \\ > 0, \text{ if } (x, y) \text{ is outside the circle boundary} \end{cases}$$

- By evaluating this function at the midpoint between the candidate pixels we can make our decision

Mid-Point Circle Algorithm (cont…)

- Assuming we have just plotted the pixel at (x_k,y_k) so we need to choose between (x_k+1,y_k) and (x_k+1,y_k-1)
- Our decision variable can be defined as:

$$p_k = f_{circ}(x_k + 1, y_k - \tfrac{1}{2})$$

$$= (x_k + 1)^2 + (y_k - \tfrac{1}{2})^2 - r^2$$

- If $p_k < 0$ the midpoint is inside the circle and and the pixel at y_k is closer to the circle
- Otherwise the midpoint is outside and y_k-1 is closer

Mid-Point Circle Algorithm (cont…)

- To ensure things are as efficient as possible we can do all of our calculations incrementally
- First consider:

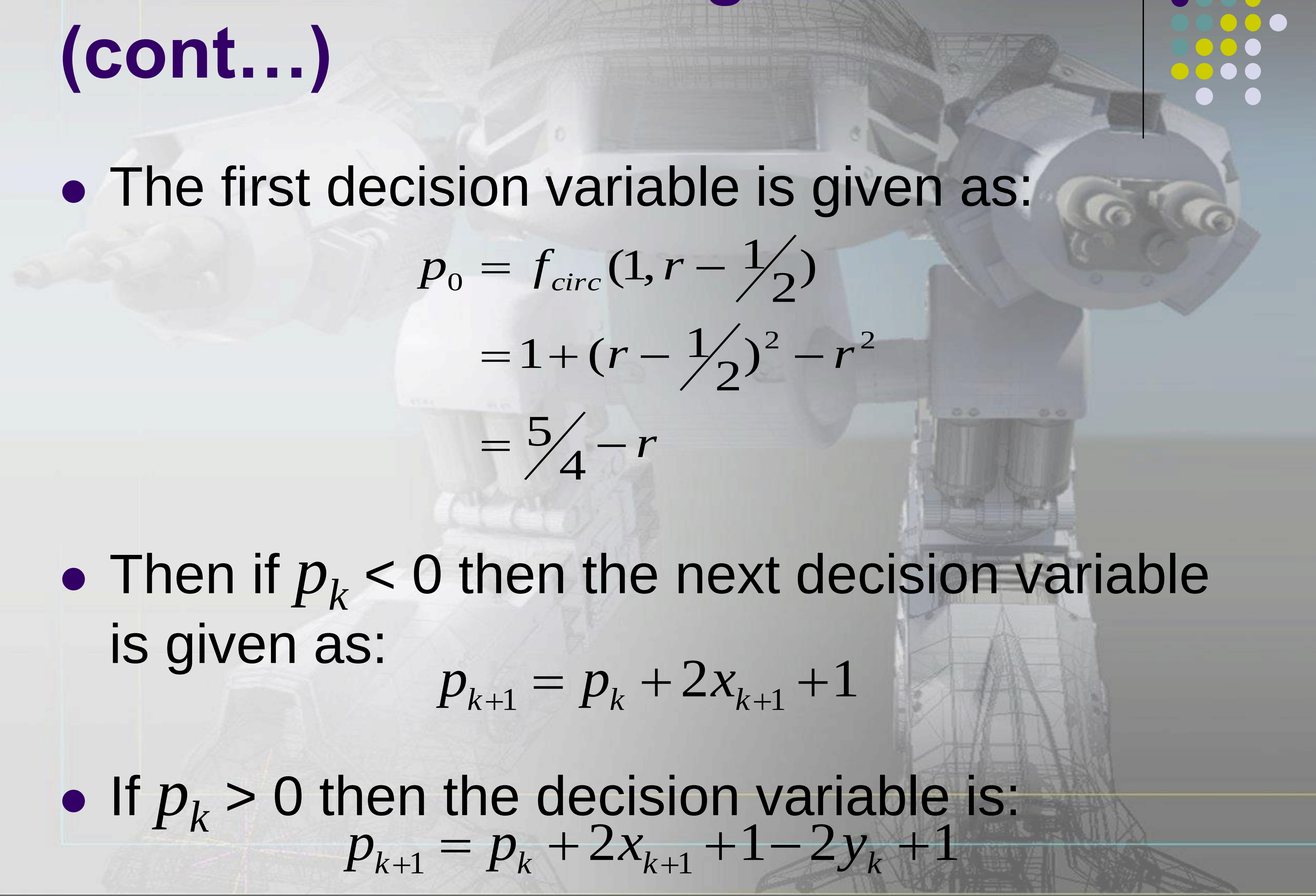
$$p_{k+1} = f_{circ}\left(x_{k+1}+1, y_{k+1}-\tfrac{1}{2}\right)$$

$$= [(x_k+1)+1]^2 + \left(y_{k+1}-\tfrac{1}{2}\right)^2 - r^2$$

- or:

$$p_{k+1} = p_k + 2(x_k+1) + (y_{k+1}^2 - y_k^2) - (y_{k+1} - y_k) + 1$$

- where y_{k+1} is either y_k or y_k-1 depending on the sign of p_k



Mid-Point Circle Algorithm (cont…)

- The first decision variable is given as:

$$\begin{aligned} p_0 &= f_{circ}(1, r - \tfrac{1}{2}) \\ &= 1 + (r - \tfrac{1}{2})^2 - r^2 \\ &= \tfrac{5}{4} - r \end{aligned}$$

- Then if $p_k < 0$ then the next decision variable is given as:

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

- If $p_k > 0$ then the decision variable is:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_k + 1$$

The Mid-Point Circle Algorithm

- MID-POINT CIRCLE ALGORITHM

- Input radius r and circle centre (x_c, y_c), then set the coordinates for the first point on the circumference of a circle centred on the origin as:

$$(x_0, y_0) = (0, r)$$

- Calculate the initial value of the decision parameter as:

$$p_0 = \tfrac{5}{4} - r$$

- Starting with $k = 0$ at each position x_k, perform the following test. If $p_k < 0$, the next point along the circle centred on $(0, 0)$ is (x_k+1, y_k) and:

$$p_{k+1} = p_k + 2x_{k+1} + 1$$

The Mid-Point Circle Algorithm (cont…)

- Otherwise the next point along the circle is (x_k+1, y_k-1) and:

$$p_{k+1} = p_k + 2x_{k+1} + 1 - 2y_{k+1}$$

4. Determine symmetry points in the other seven octants
5. Move each calculated pixel position (x, y) onto the circular path centred at (x_c, y_c) to plot the coordinate values:

$$x = x + x_c \qquad y = y + y_c$$

6. Repeat steps 3 to 5 until $x >= y$

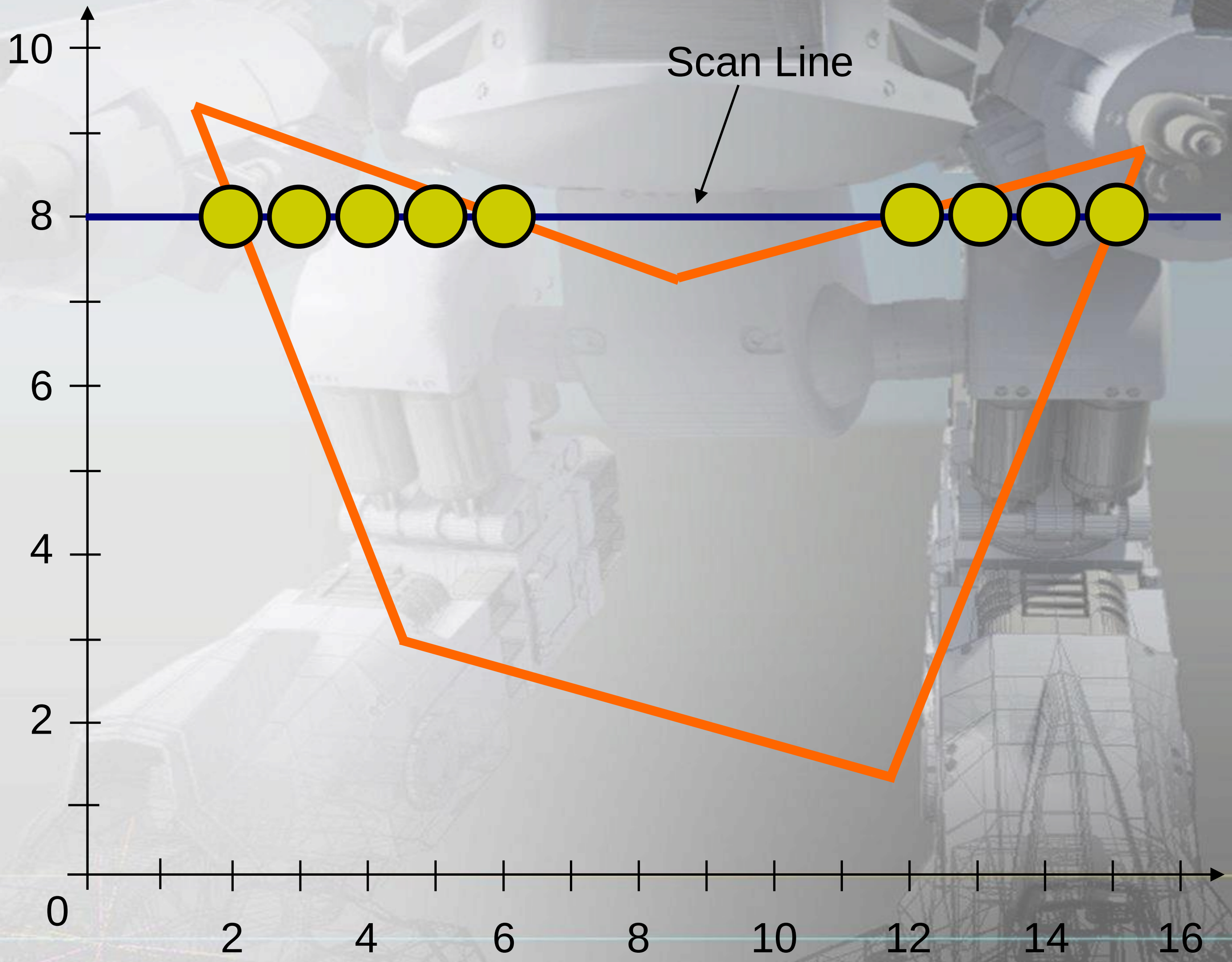
Mid-Point Circle Algorithm Summary

- The key insights in the mid-point circle algorithm are:
  1. Eight-way symmetry can hugely reduce the work in drawing a circle
  2. Moving in unit steps along the x axis at each point along the circle's edge we need to choose between two possible y coordinates

Filling Polygons

- So we can figure out how to draw lines and circles
- How do we go about drawing polygons?
- We use an incremental algorithm known as the scan-line algorithm

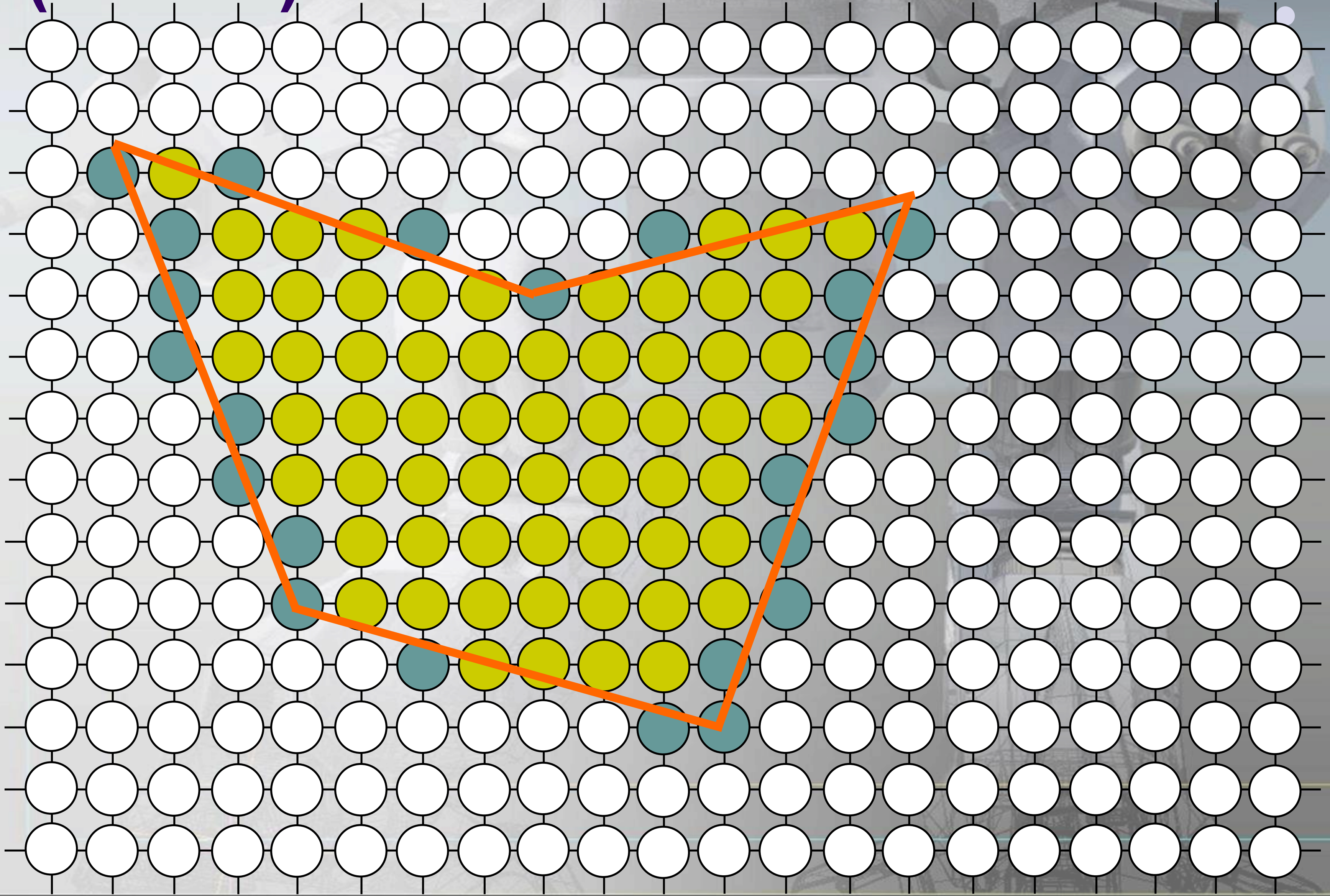
Scan-Line Polygon Fill Algorithm



Scan-Line Polygon Fill Algorithm

- The basic scan-line algorithm is as follows:
  1. Find the intersections of the scan line with all edges of the polygon
  2. Sort the intersections by increasing x coordinate
  3. Fill in all pixels between pairs of intersections that lie interior to the polygon

Scan-Line Polygon Fill Algorithm (cont…)



Line Drawing Summary

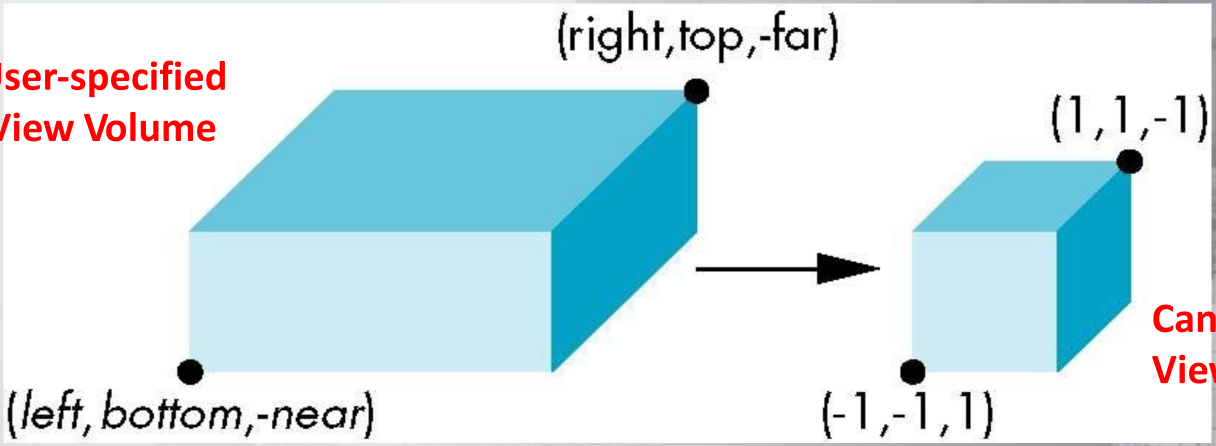
- Over the last couple of lectures we have looked at the idea of scan converting lines
- The key thing to remember is this has to be **FAST**
- For lines we have either DDA or Bresenham
- For circles the mid-point algorithm

perspective Projection

Parallel Projection

- **normalization** ⇓ find 4x4 matrix to transform **user-specified view volume** to **canonical view volume (cube)**

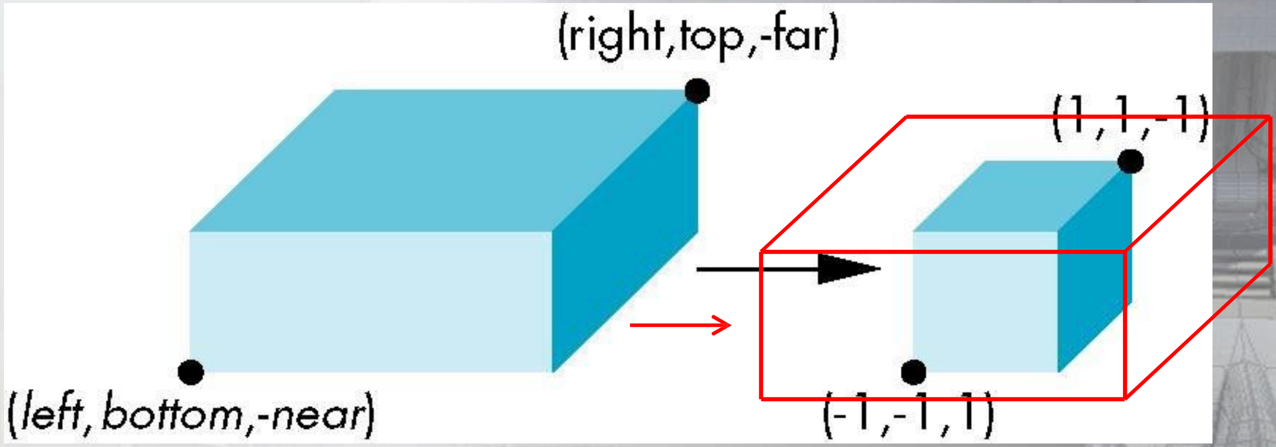
User-specified View Volume



Canonical View Volume

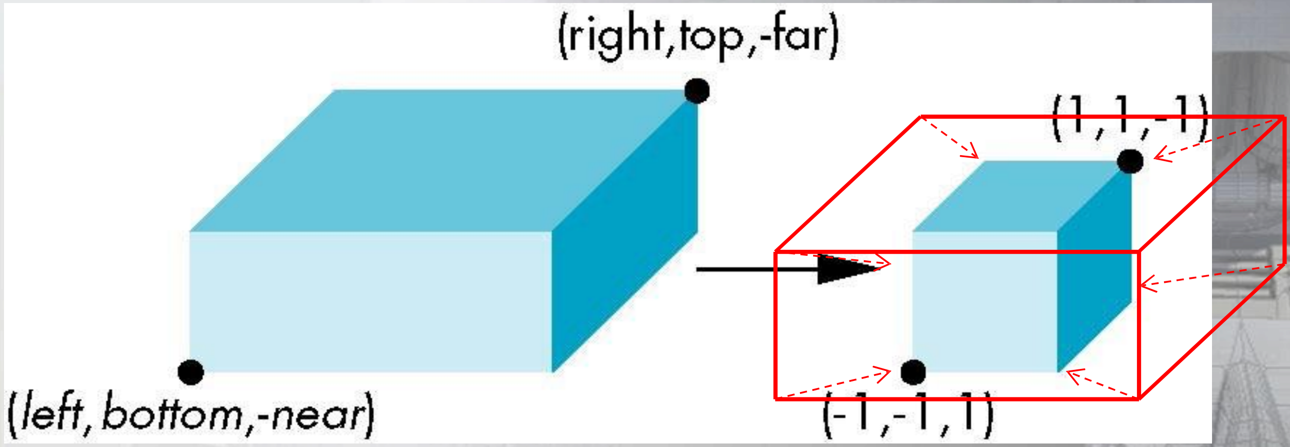
Parallel Projection: Ortho

- Parallel projection: 2 parts
  1. **Translation:** centers view volume at origin



Parallel Projection: Ortho

2. **Scaling:** reduces user-selected cuboid to canonical cube (dimension 2, centered at origin)



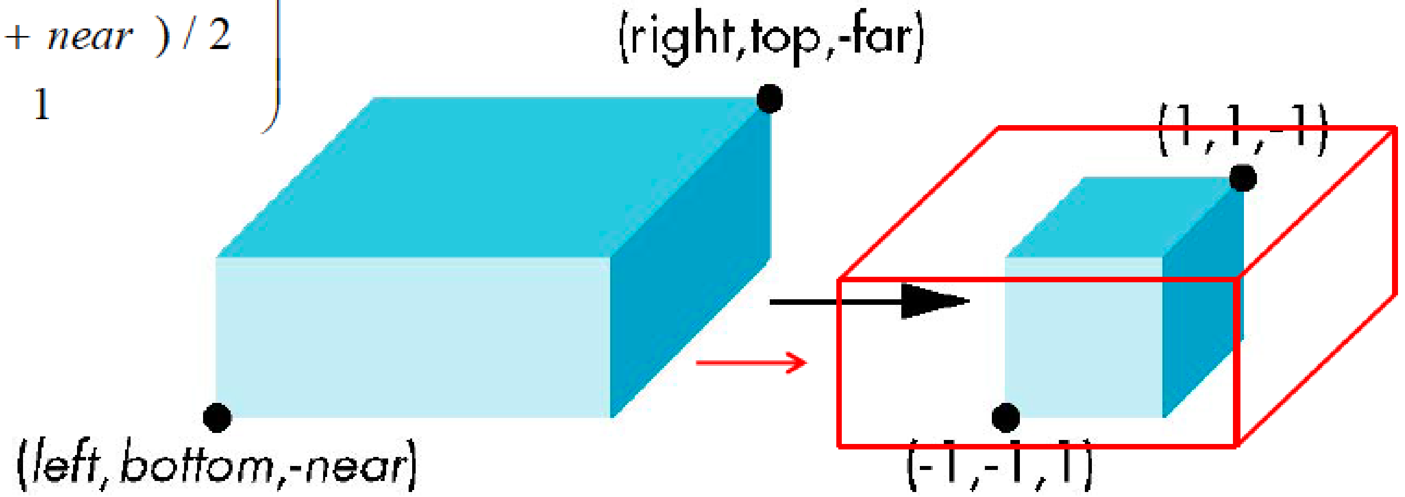
Parallel Projection: Ortho

- Translation lines up midpoints: E.g. midpoint of x = (right + left)/2
- Thus translation factors:

  -(right + left)/2, -(top + bottom)/2, -(far+near)/2

- Translation matrix:

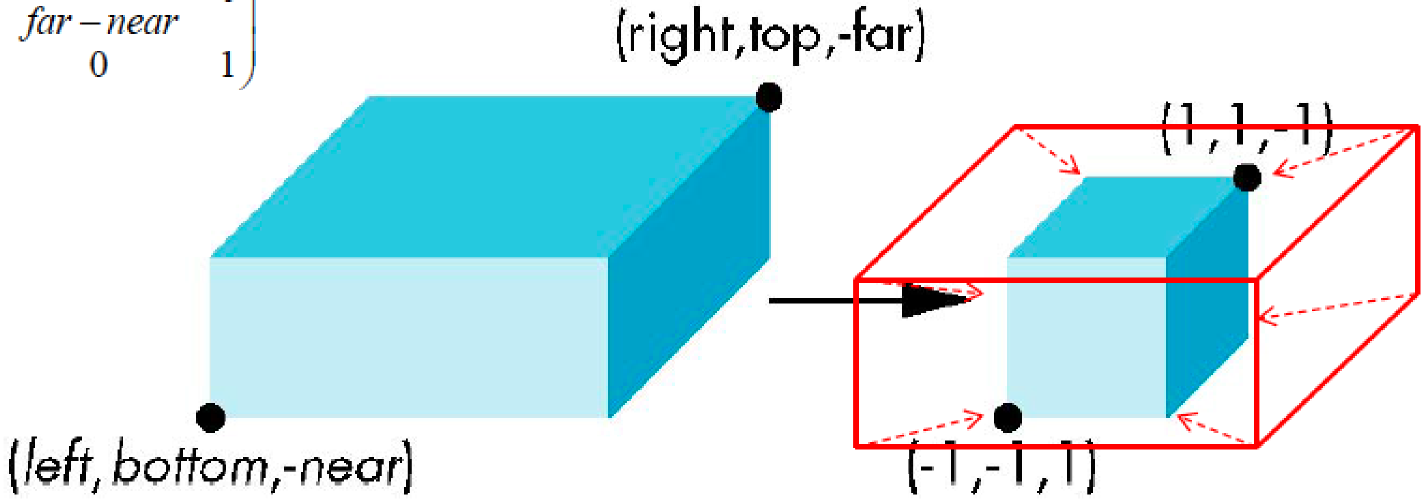
$$\begin{pmatrix} 1 & 0 & 0 & -(right + left)/2 \\ 0 & 1 & 0 & -(top + bottom)/2 \\ 0 & 0 & 1 & -(far + near)/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Parallel Projection: Ortho

- Scaling factor: ratio of ortho view volume to cube dimensions
- Scaling factors: 2/(right - left), 2/(top - bottom), 2/(far - near)
- Scaling Matrix M2:

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & \frac{2}{far-near} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Parallel Projection: Ortho

Concatenating **Translation** x **Scaling**, we get Ortho Projection matrix

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & \frac{2}{far-near} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & -(right+left)/2 \\ 0 & 1 & 0 & -(top+bottom)/2 \\ 0 & 0 & 1 & -(far+near)/2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$\mathbf{P} = \mathbf{ST} = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & -\frac{right-left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & -\frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

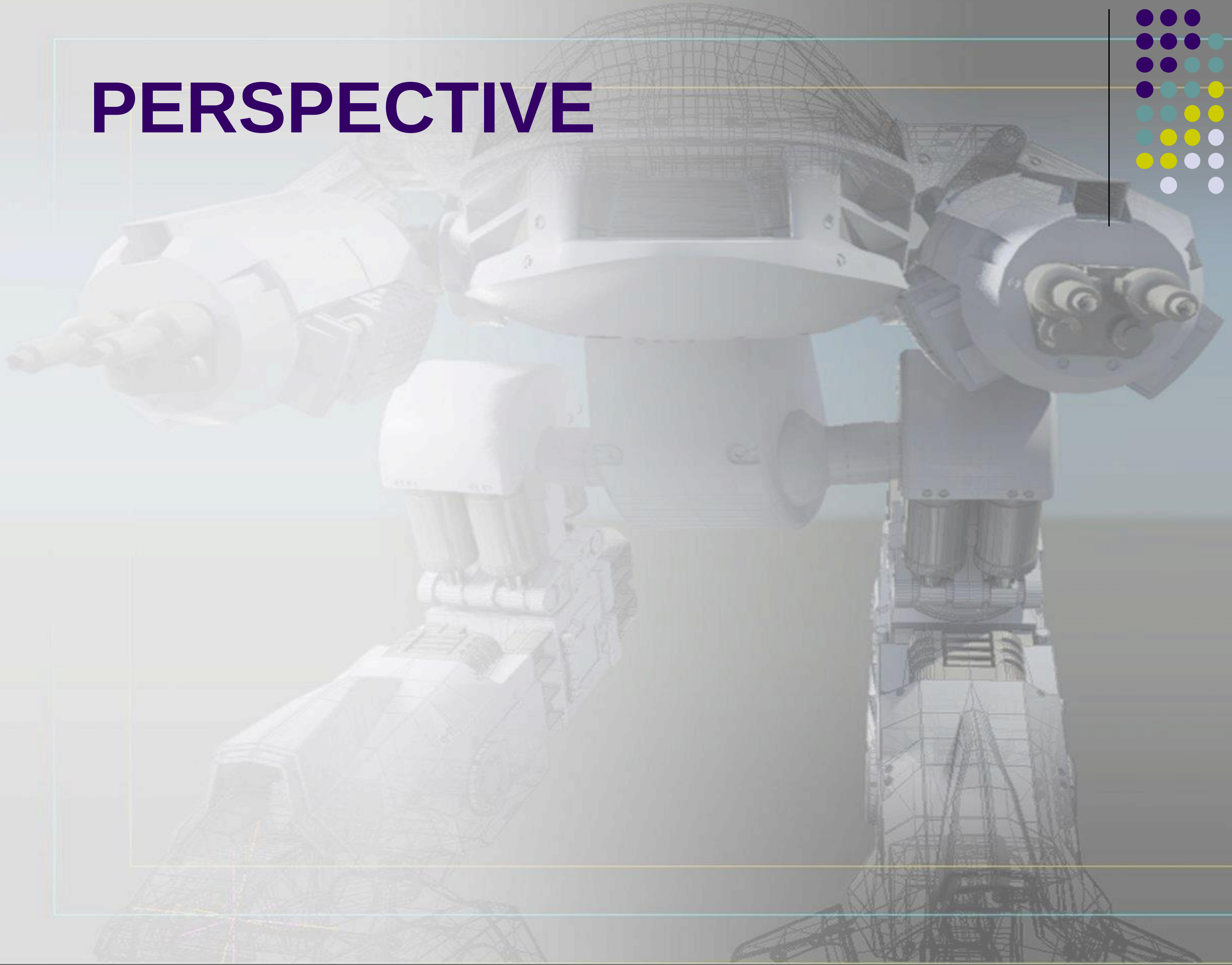
Final Ortho Projection

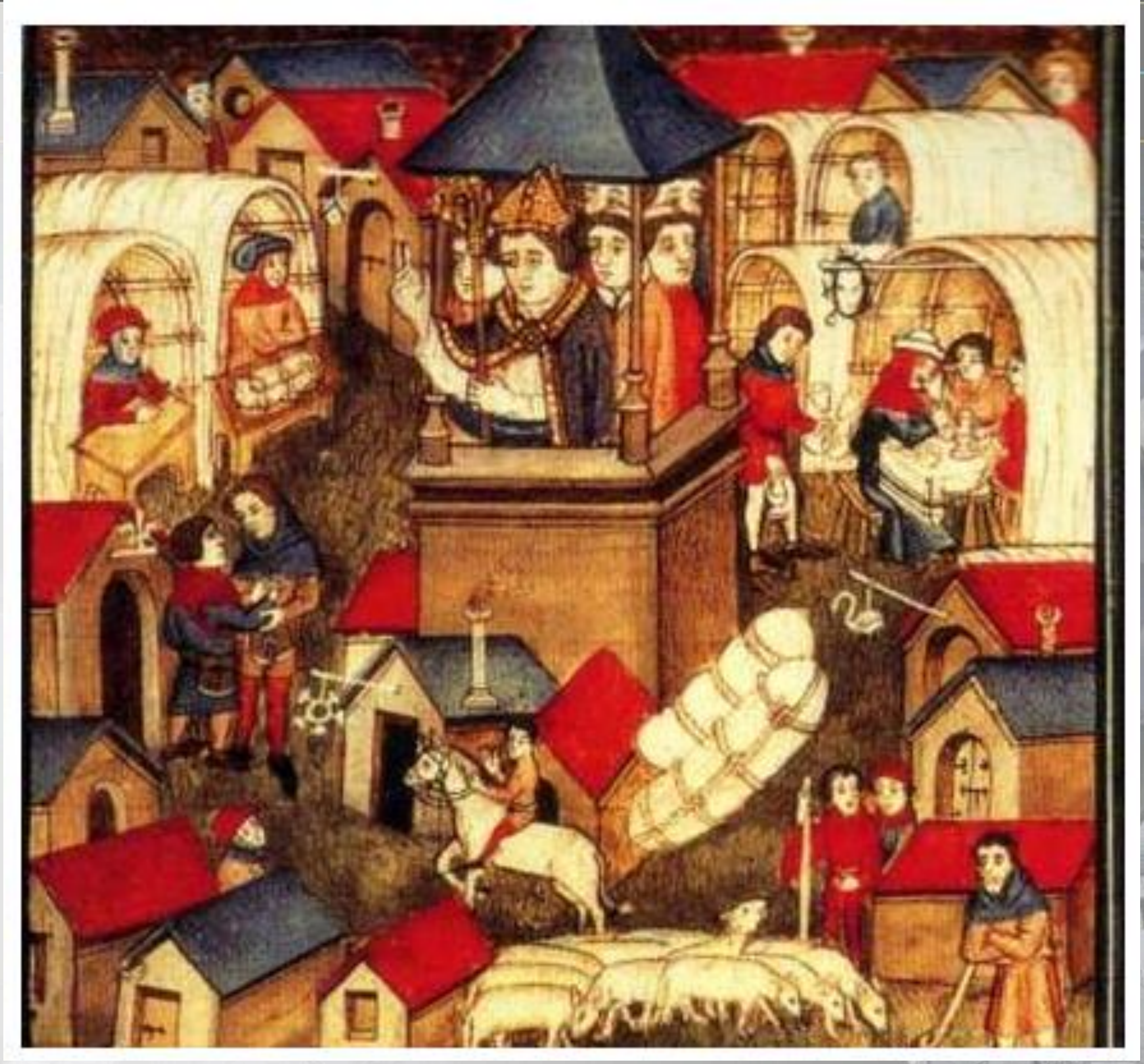
- Set $z=0$
- Equivalent to the homogeneous coordinate transformation

$$\mathbf{M}_{\text{orth}} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

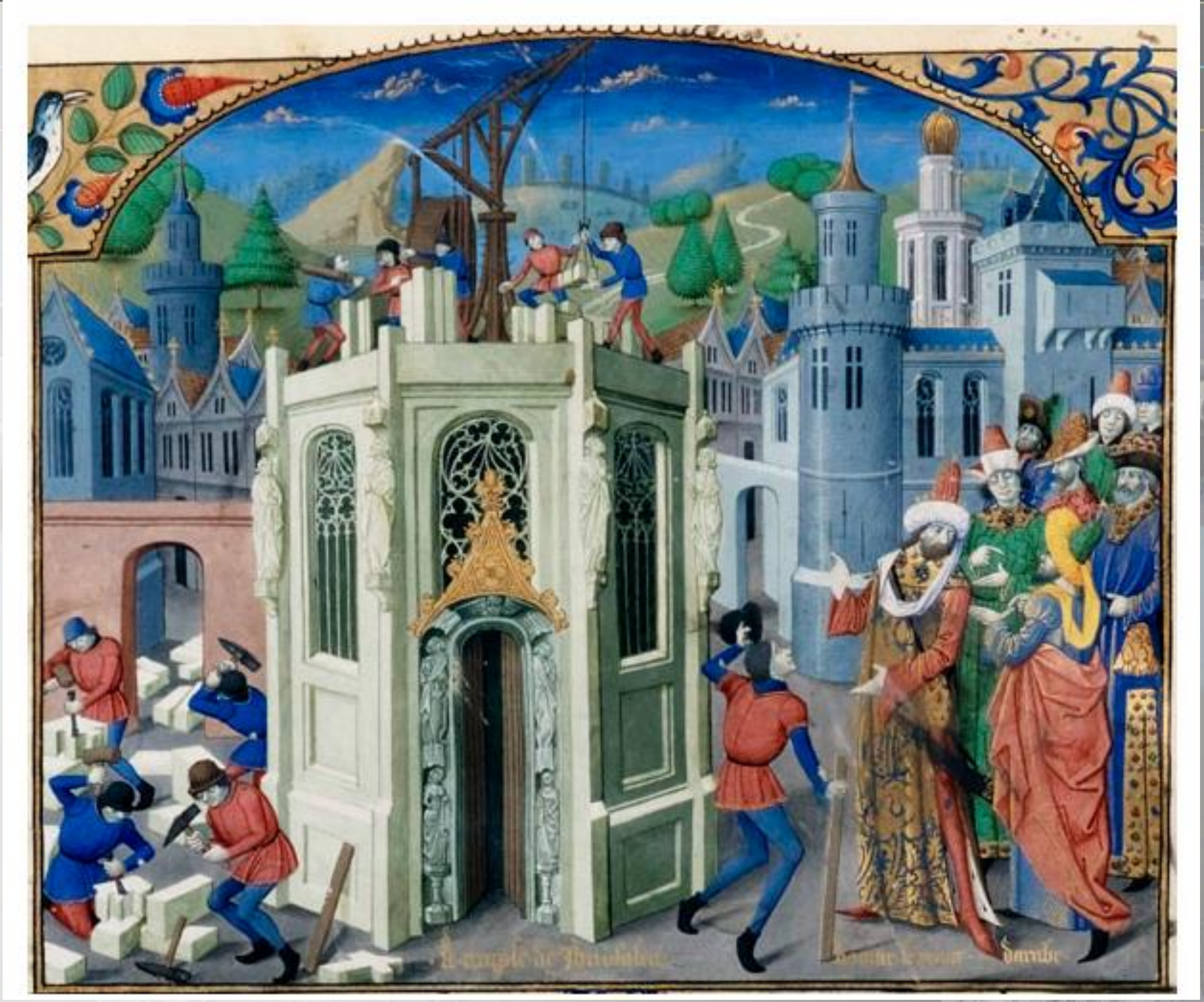
- Hence, general orthogonal projection in 4D is $\mathbf{P} = \mathbf{M}_{\text{orth}}\mathbf{ST}$

PERSPECTIVE

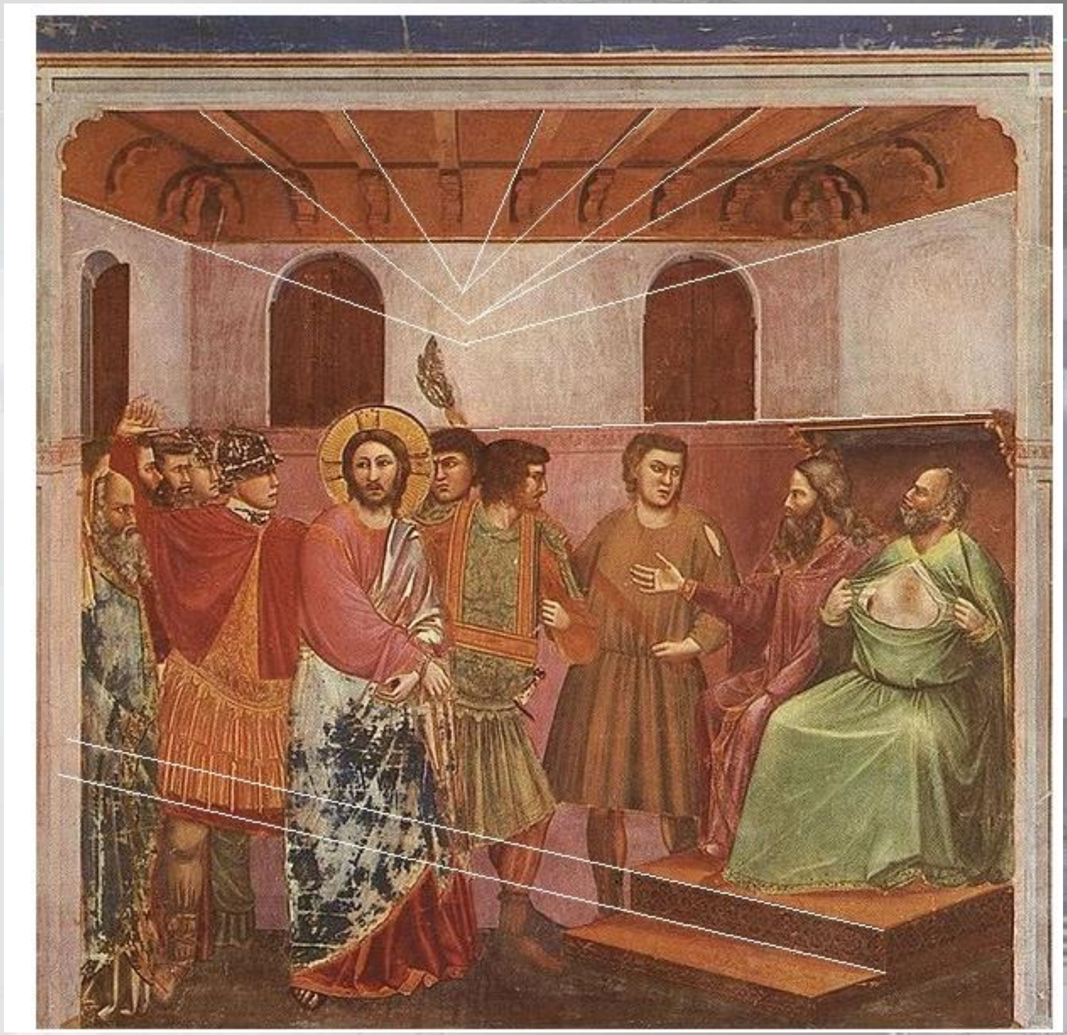




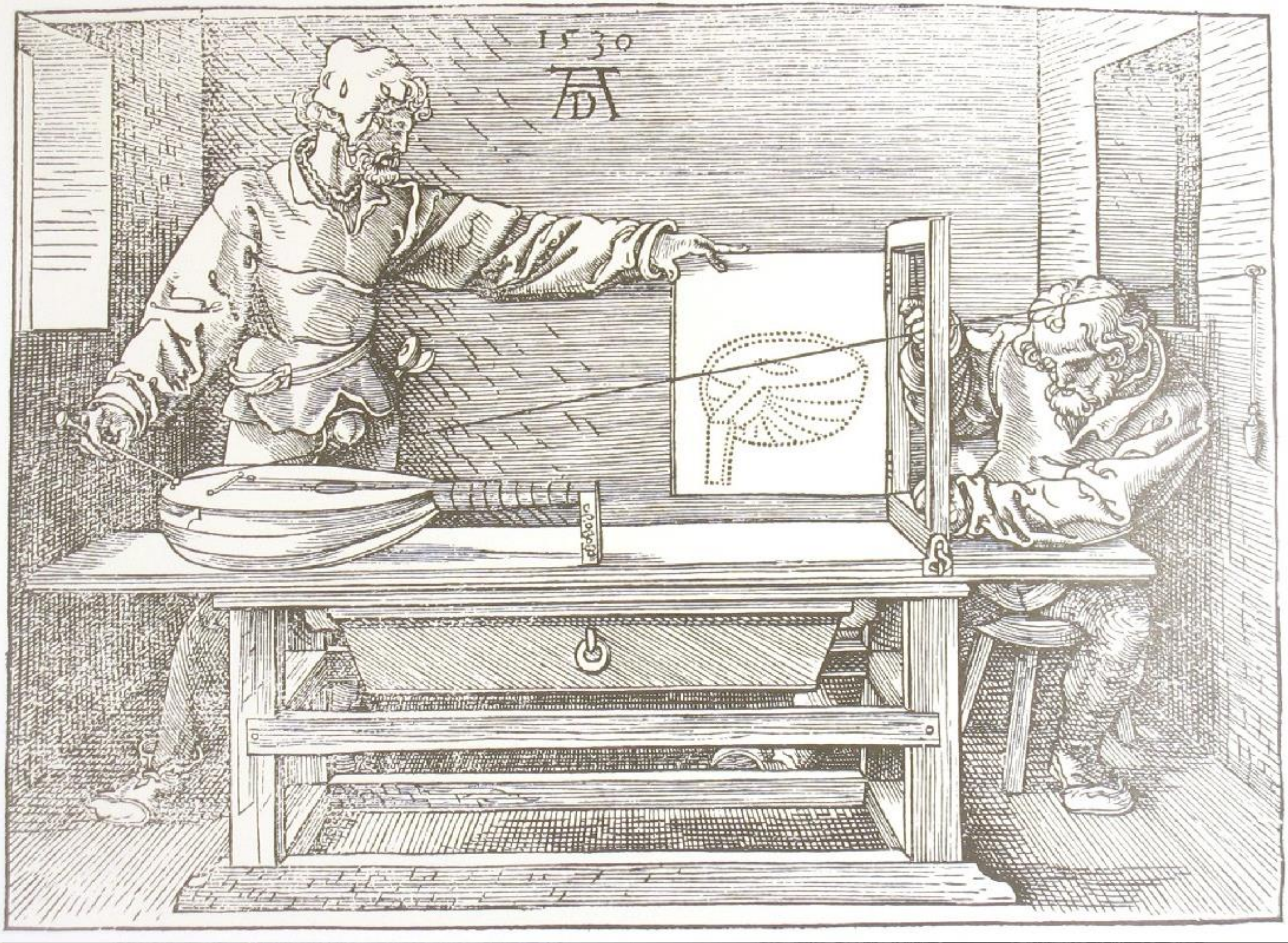
Medieval paintings do not attempt to render perspective correctly

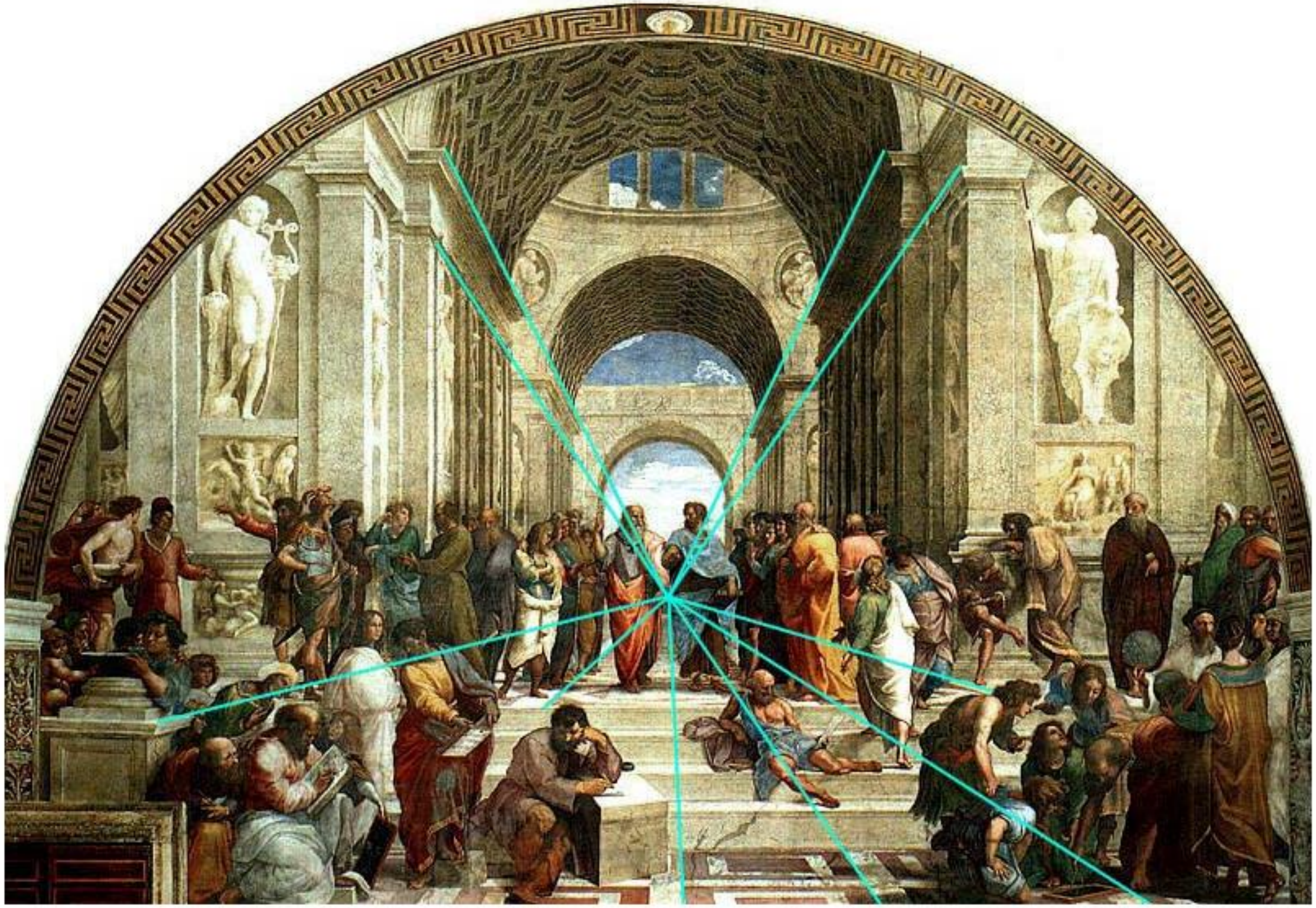


pre-renaissance often show poor understanding of perspective



Giotto was the first painter to systematically attempt to achieve perspective in his paintings.

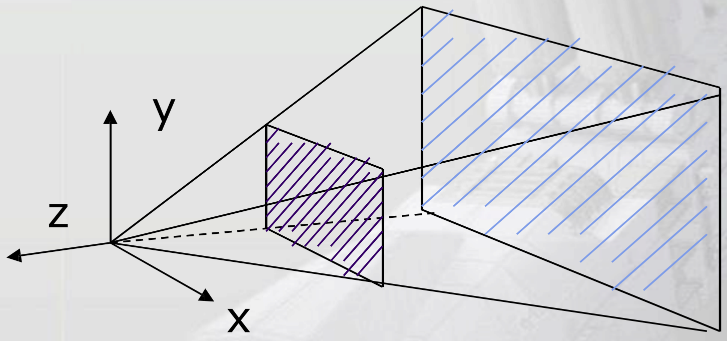

1530
AD



Raphael's "The School of Athens" shows architectural perspecive to good effect

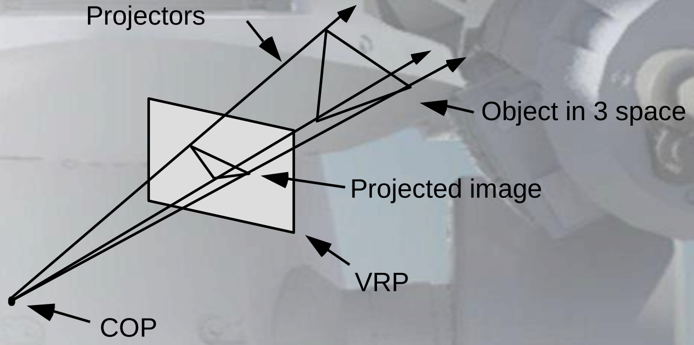
Perspective Projection

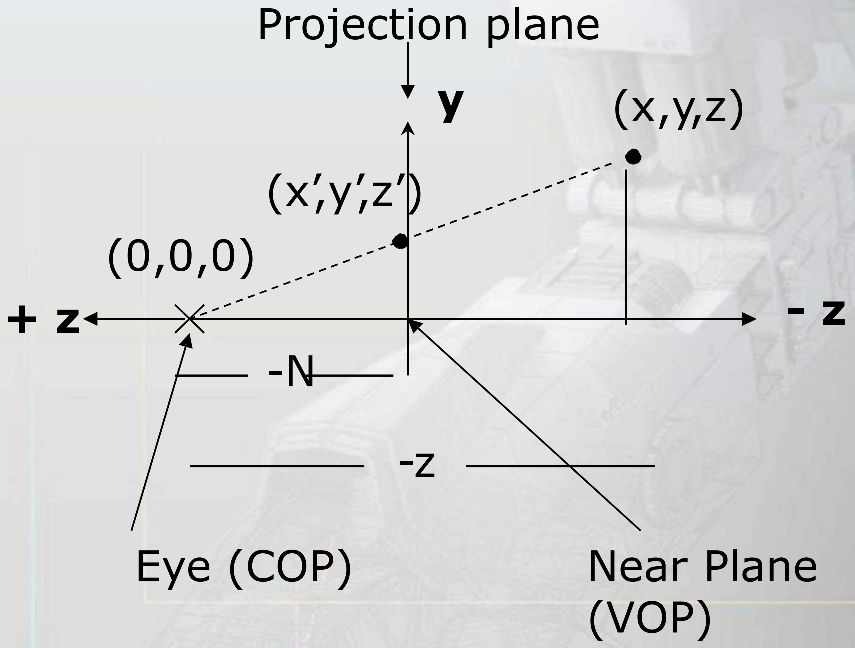
- Projection – map the object from 3D space to 2D screen



Perspective()
Frustrum( )

Perspective Projection: Classical



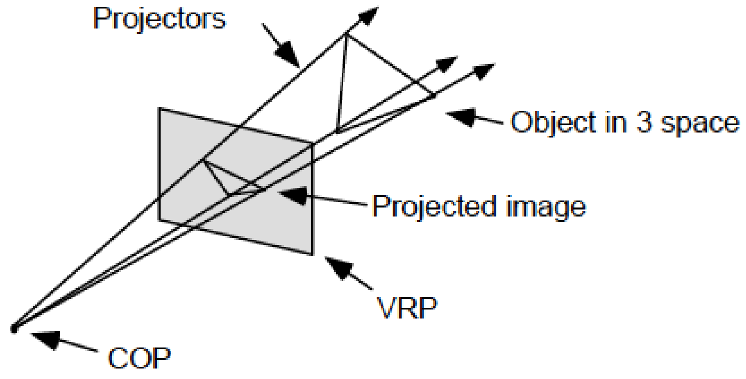


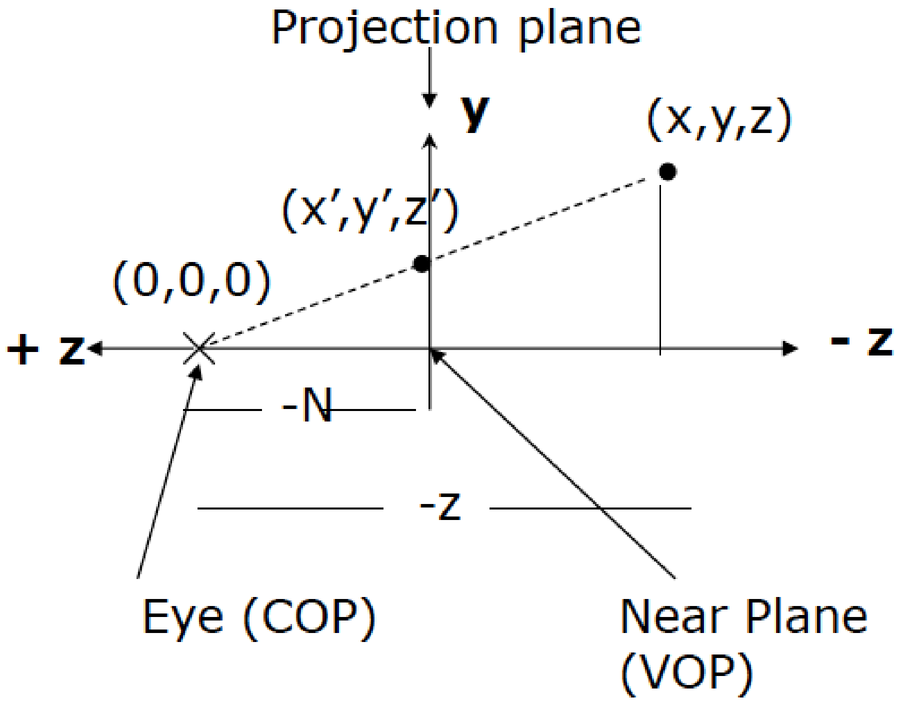
Based on similar triangles:

$$\frac{y'}{y} = \frac{N}{-z}$$

$$\Rightarrow \quad y' = y \times \frac{N}{-z}$$

Perspective Projection: Classical





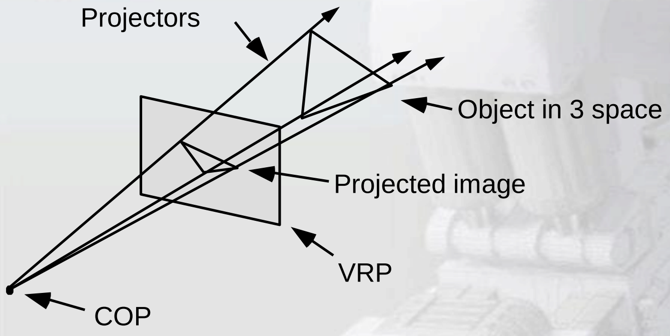
Based on similar triangles:

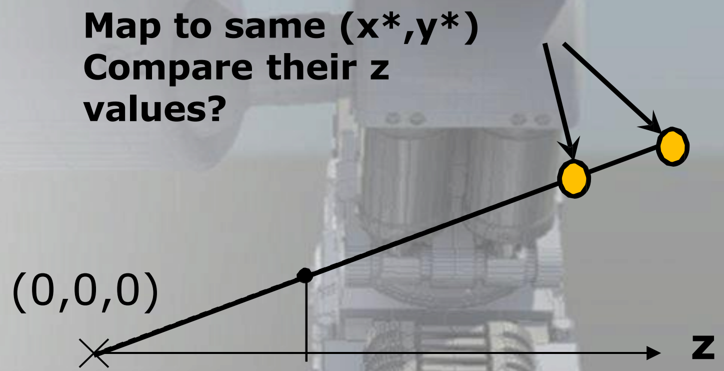
$$\frac{y'}{y} = \frac{N}{-z}$$

$$\Rightarrow \quad y' = y \times \frac{N}{-z}$$

Pseudodepth

- Classical perspective projection projects (x,y) coordinates to (x*, y*), drops z coordinates

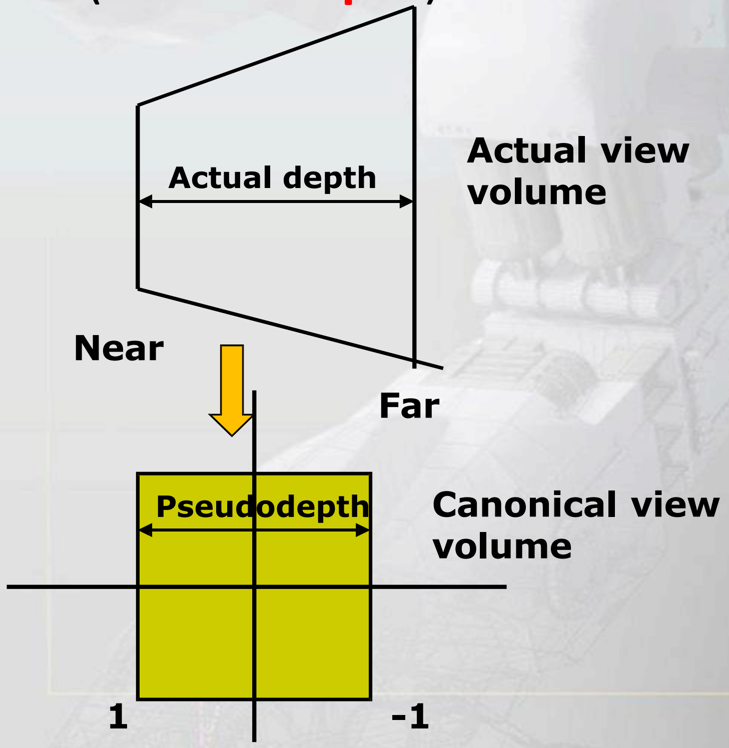




- But we need z to find closest object (depth testing)!!!

Perspective Transformation

- **Perspective transformation** maps actual z distance of perspective view volume to range [–1 to 1] (Pseudodepth) for canonical view volume

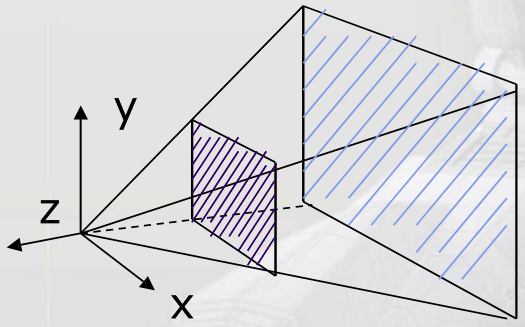


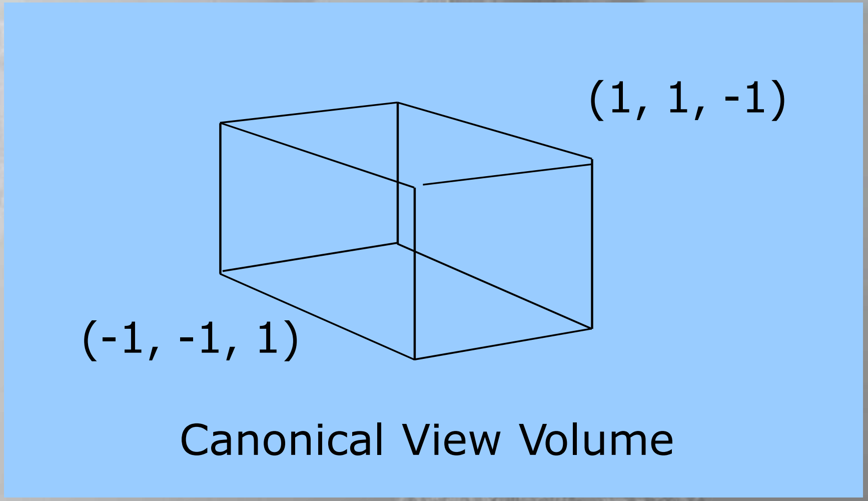
We want perspective Transformation and NOT classical projection!!

Set scaling z
Pseudodepth = az + b
Next solve for a and b

Perspective Transformation

- We want to transform viewing frustum volume into canonical view volume



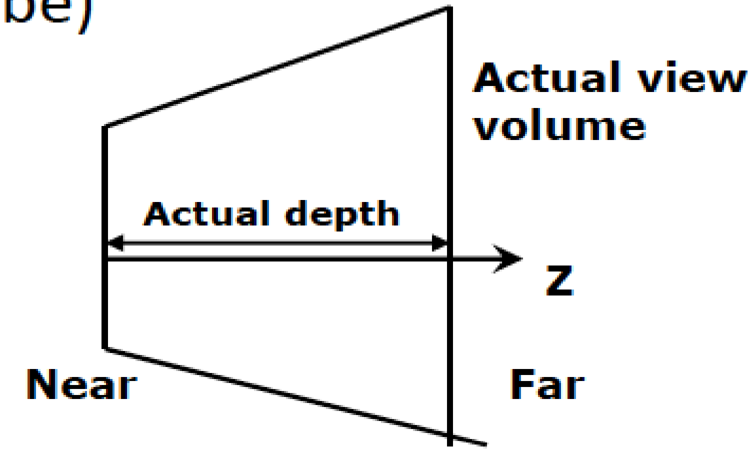


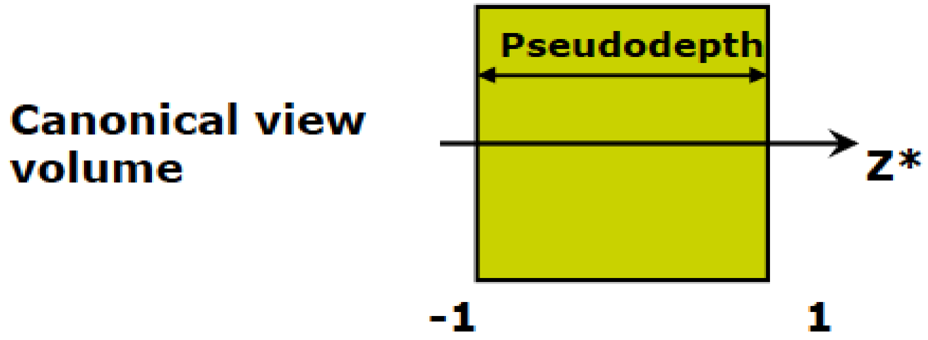
Perspective Transformation using Pseudodepth

$$(x^*, y^*, z^*) = \left(x\frac{N}{-z}, y\frac{N}{-z}, \frac{az + b}{-z} \right)$$

- Choose ***a, b*** so as z varies from **Near** to **Far**, pseudodepth varies from **–1** to **1** (canonical cube)

- Boundary conditions
  - z* = -1 when z = -N
  - z* = 1 when z = -F





Transformation of z: Solve for a and b

- Solving:

$$z^* = \frac{az + b}{-z}$$

- Use boundary conditions
  - z* = -1 when z = -N.........(1)
  - z* = 1 when z = -F..........(2)
- Set up simultaneous equations

$$-1 = \frac{-aN + b}{N} \Rightarrow -N = -aN + b........(1)$$

$$1 = \frac{-aF + b}{F} \Rightarrow F = -aF + b........(2)$$

Transformation of z: Solve for a and b

$$-N = -aN + b........(1)$$

$$F = -aF + b........(2)$$

- Multiply both sides of (1) by -1

$$N = aN - b........(3)$$

- Add eqns (2) and (3)

$$F + N = aN - aF$$

$$\Rightarrow a = \frac{F+N}{N-F} = \frac{-(F+N)}{F-N}.........(4)$$

- Now put (4) back into (3)

Transformation of z: Solve for a and b

- Put solution for *a* back into eqn (3)

$$N = aN - b........(3)$$

$$\Rightarrow N = \frac{-N(F+N)}{F-N} - b$$

$$\Rightarrow b = -N - \frac{-N(F+N)}{F-N}$$

$$\Rightarrow b = \frac{-N(F-N) - N(F+N)}{F-N} = \frac{-NF - N^2 - NF + N^2}{F-N} = \frac{-2NF}{F-N}$$

- So

$$a = \frac{-(F+N)}{F-N} \qquad b = \frac{-2FN}{F-N}$$

What does this mean?

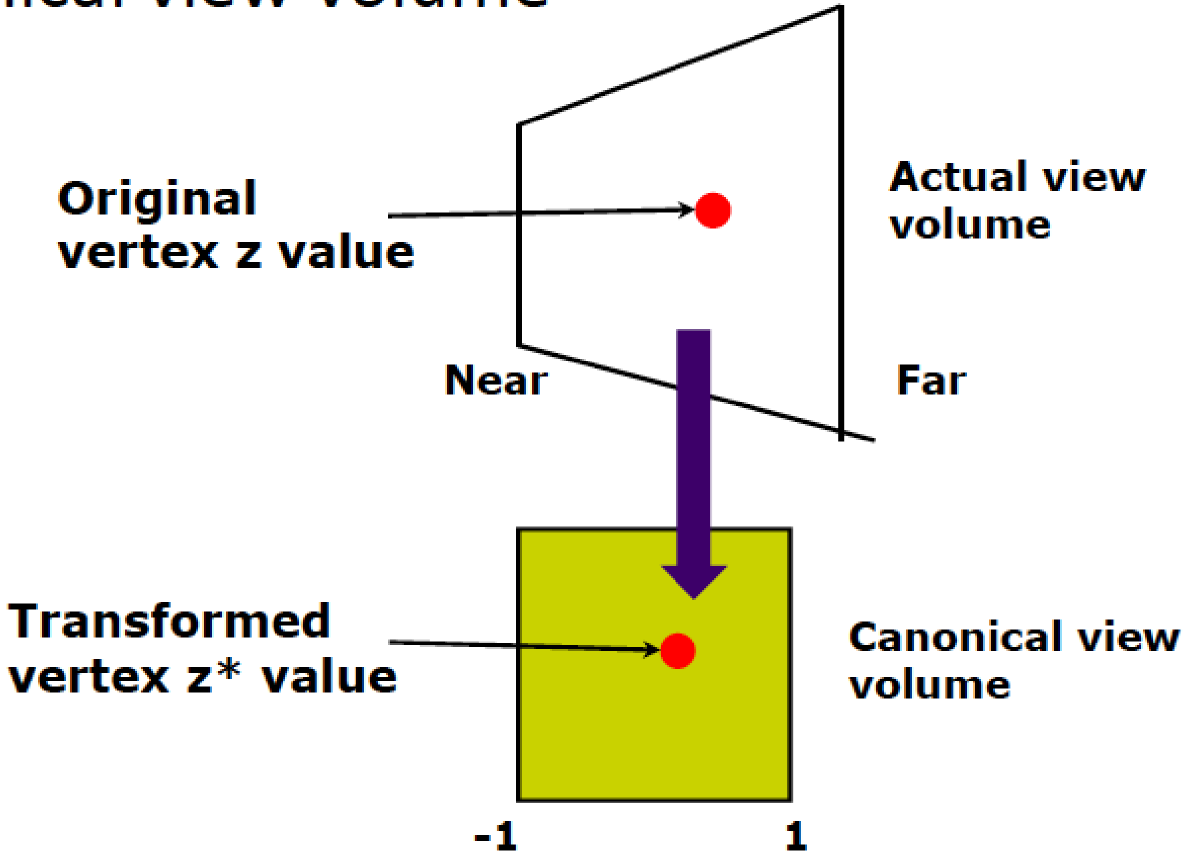
- Original point z in original view volume, transformed into z* in canonical view volume

$$z^* = \frac{az + b}{-z}$$

- where

$$a = \frac{-(F+N)}{F-N}$$

$$b = \frac{-2FN}{F-N}$$



Homogenous Coordinates

- Want to express projection transform as 4x4 matrix
- Previously, homogeneous coordinates of P = (Px,Py,Pz) => (Px,Py,Pz,1)
- Introduce arbitrary scaling factor, w, so that
  P = (wPx, wPy, wPz, w) (**Note:** w is non-zero)
- For example, the point P = (2,4,6) can be expressed as
  - (2,4,6,1)
  - or (4,8,12,2) where w=2
  - or (6,12,18,3) where w = 3, or….
- To convert from homogeneous back to ordinary coordinates, first divide all four terms by **w** and discard 4th term

Perspective Projection Matrix

- Recall Perspective Transform

$$(x^*, y^*, z^*) = \left(x\frac{N}{-z}, y\frac{N}{-z}, \frac{az+b}{-z} \right)$$

- We have: $x^* = x\frac{N}{-z}$ $y^* = y\frac{N}{-z}$ $z^* = \frac{az+b}{-z}$

- In matrix form:

$$\begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} wx \\ wy \\ wz \\ w \end{pmatrix} = \begin{pmatrix} wNx \\ wNy \\ w(az+b) \\ -wz \end{pmatrix} \Rightarrow \begin{pmatrix} x\frac{N}{-z} \\ y\frac{N}{-z} \\ \frac{az+b}{-z} \\ 1 \end{pmatrix}$$

Perspective Transform Matrix

Original vertex

Transformed Vertex

Transformed Vertex after dividing by 4th term

Perspective Projection Matrix

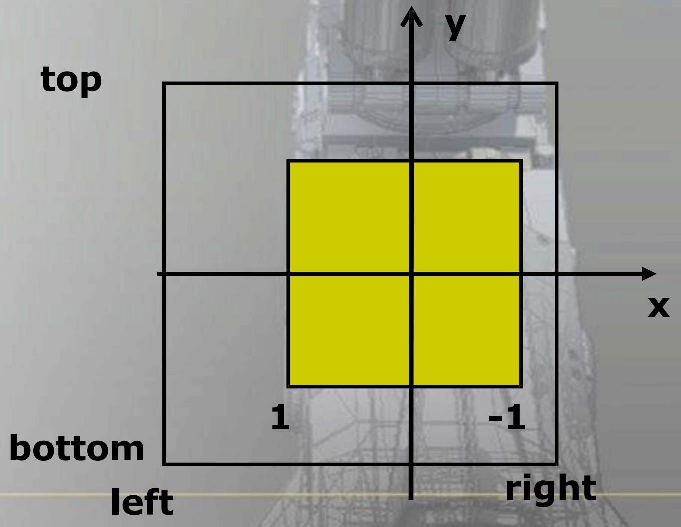
$$\begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} wP_x \\ wP_y \\ wP_z \\ w \end{pmatrix} = \begin{pmatrix} wNP_x \\ wNP_y \\ w(aP_z + b) \\ -wP_z \end{pmatrix} \Rightarrow \begin{pmatrix} x\frac{N}{-z} \\ y\frac{N}{-z} \\ \frac{az+b}{-z} \\ 1 \end{pmatrix}$$

$$a = \frac{-(F+N)}{F-N} \qquad b = \frac{-2FN}{F-N}$$

- In perspective transform matrix, already solved for ***a*** and ***b***:
- So, we have transform matrix to transform **z** values

Perspective Projection

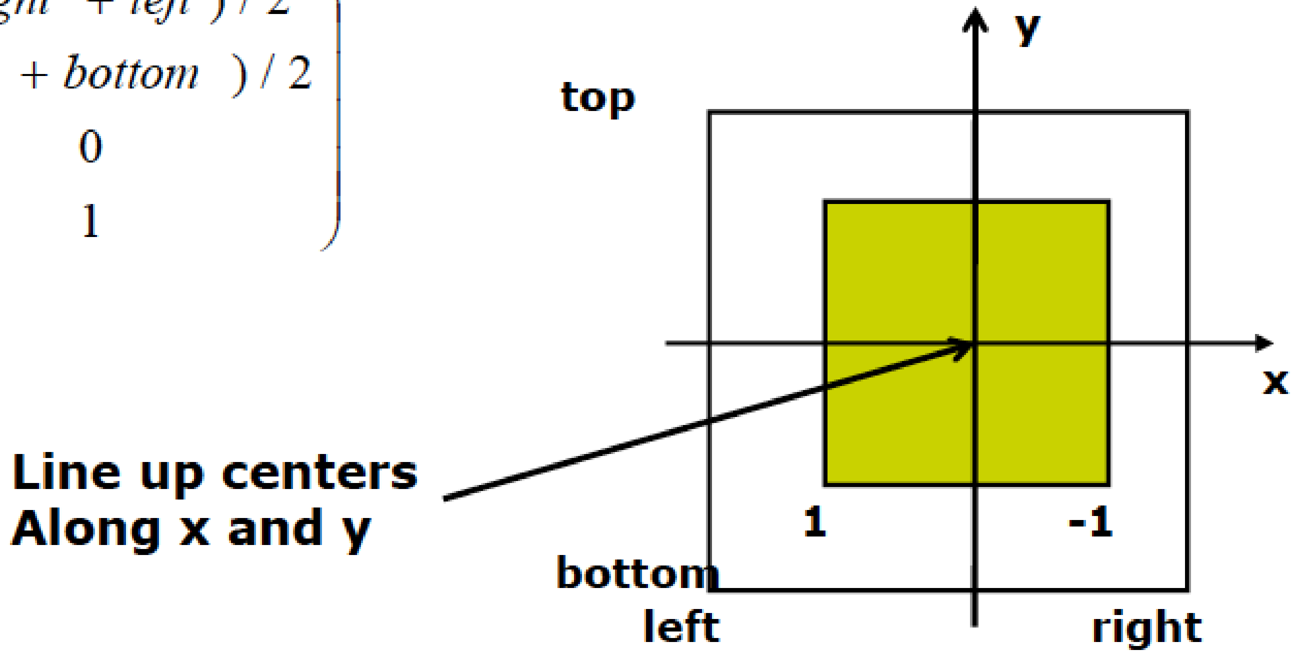
- Not done yet!! Can now transform z!
- Also need to transform the **x** = (left, right) and **y** = (bottom, top) ranges of viewing frustum to [-1, 1]
- we need to translate and scale previous matrix along x and y to get final projection transform matrix
- we translate by
  - –(right + left)/2 in x
  - -(top + bottom)/2 in y
- Scale by:
  - 2/(right – left) in x
  - 2/(top – bottom) in y



Perspective Projection

- Translate along x and y to line up center with origin of CVV
  - −(right + left)/2 in x
  - -(top + bottom)/2 in y
- Multiply by translation matrix:

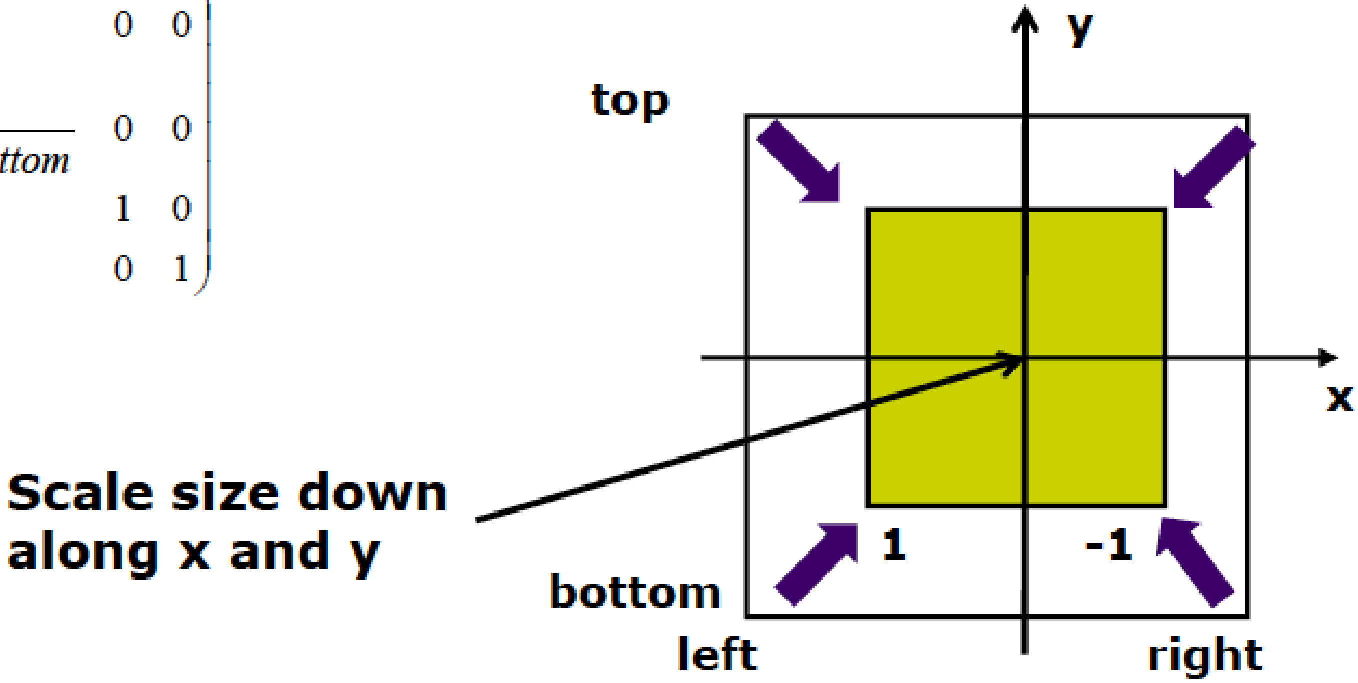
$$\begin{pmatrix} 1 & 0 & 0 & -(right + left)/2 \\ 0 & 1 & 0 & -(top + bottom)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Perspective Projection

- To bring view volume size down to size of of CVV, scale by
  - 2/(right – left) in x
  - 2/(top – bottom) in y
- Multiply by scale matrix:

$$\begin{pmatrix} \frac{2}{right-left} & 0 & 0 & 0 \\ 0 & \frac{2}{top-bottom} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Perspective Projection Matrix

Scale **Translate** **Previous Perspective Transform Matrix**

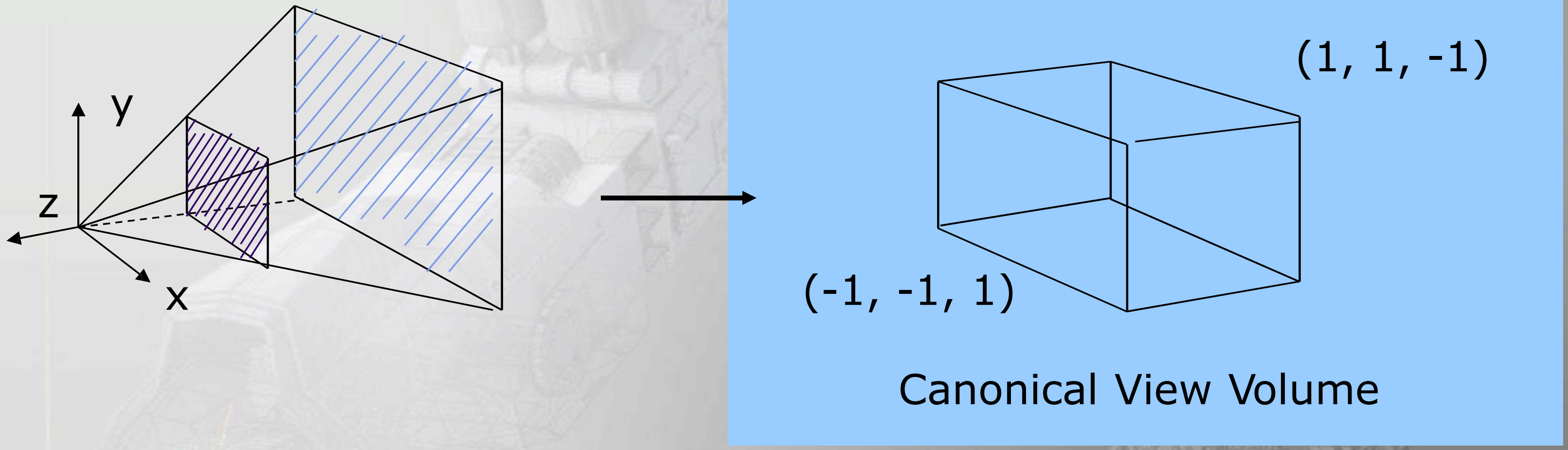
$$\begin{pmatrix} \frac{2}{right - left} & 0 & 0 & 0 \\ 0 & \frac{2}{top - bottom} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} 1 & 0 & 0 & -(right + left)/2 \\ 0 & 1 & 0 & -(top + bottom)/2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} \frac{2N}{x\max - x\min} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2N}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & \frac{-(F+N)}{F-N} & \frac{-2FN}{F-N} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Final Perspective Transform Matrix

Perspective Transformation

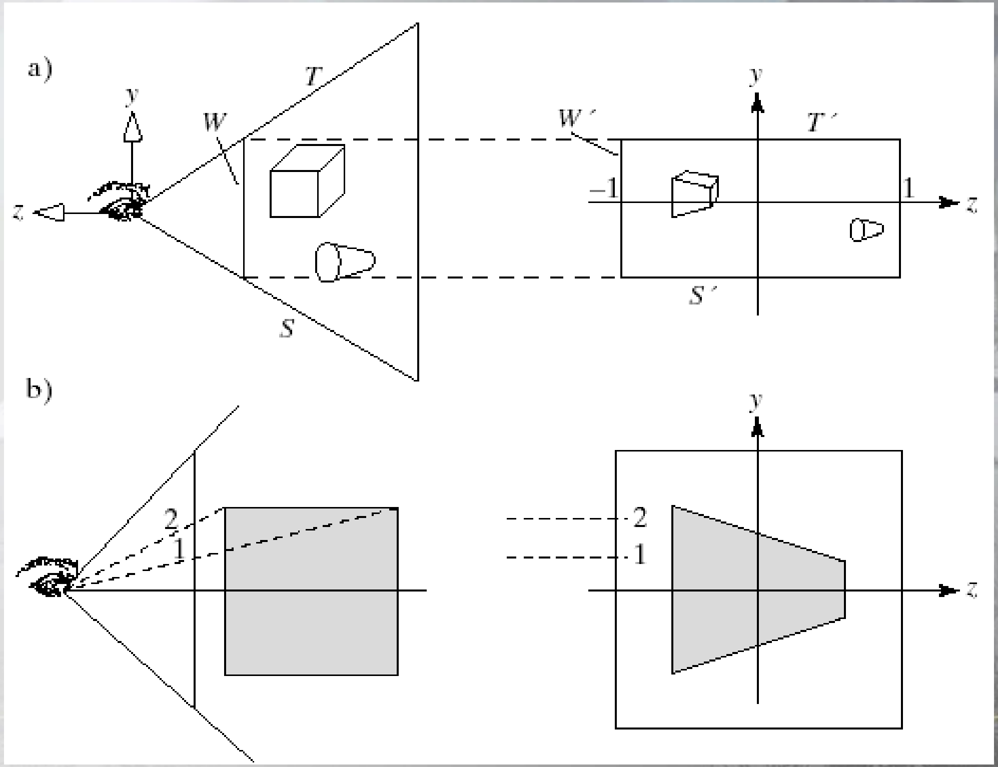
- After perspective transformation, viewing frustum volume is transformed into canonical view volume



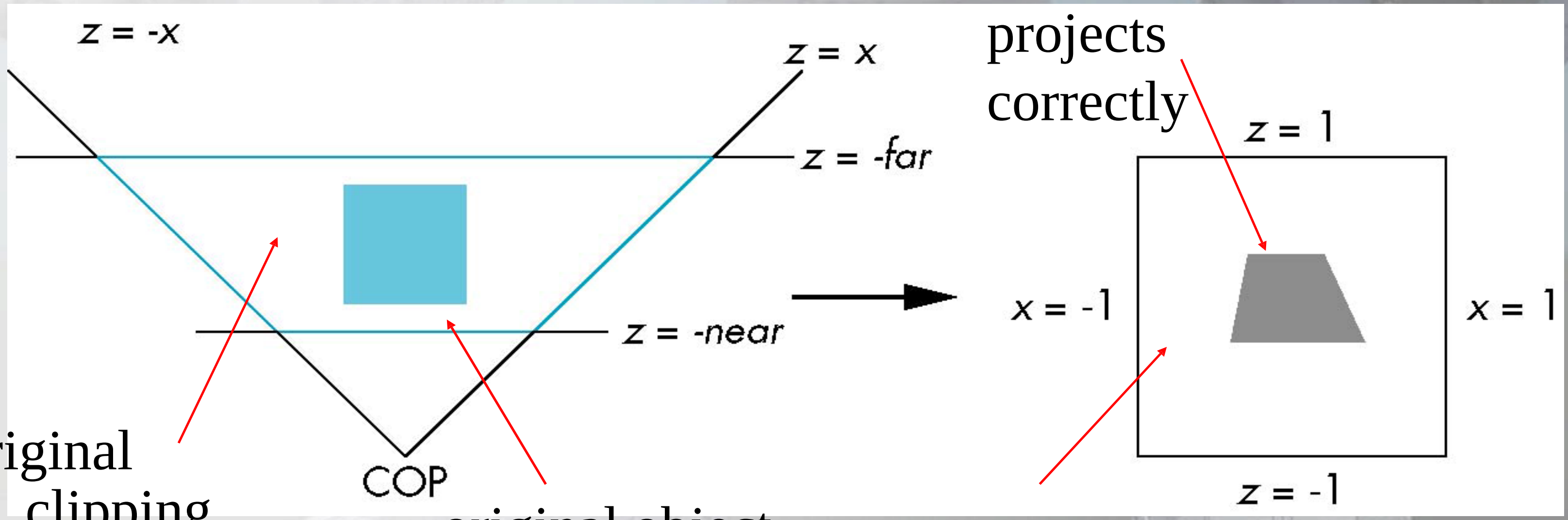
Geometric Nature of Perspective Transform

a) Lines through eye map into lines parallel to z axis after transform

b) Lines perpendicular to z axis map to lines perp to z axis after transform



Normalization Transformation



distorted object projects correctly

original clipping volume

original object

new clipping volume