From 3D World to 2D Screen

Hendrik Speleers

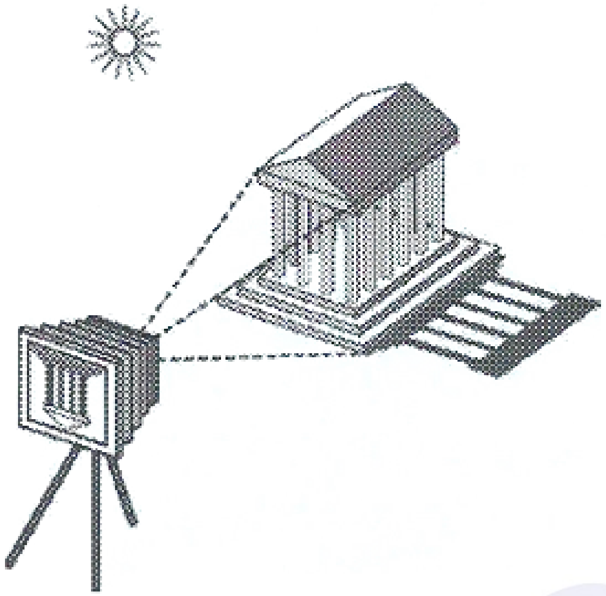
From 3D World to 2D Screen

- Overview
  - Synthetic camera
  - Rendering pipeline
  - World window versus viewport
  - Clipping
    - Cohen-Sutherland algorithm
  - Rasterizing
    - Bresenham algorithm

From 3D World to 2D Screen

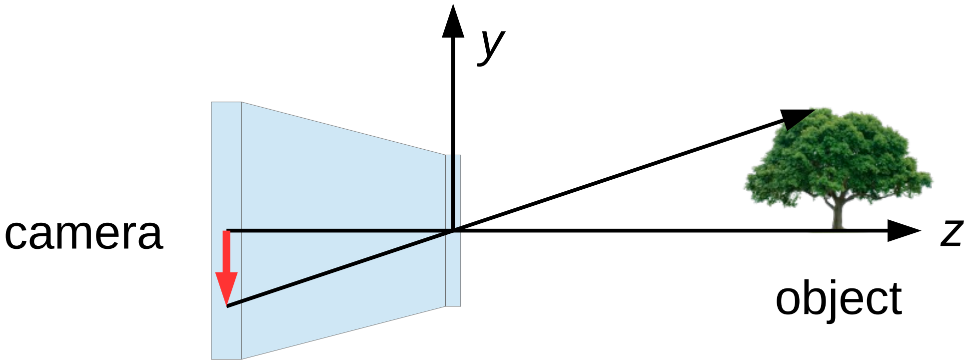
Università di Roma Tor Vergata

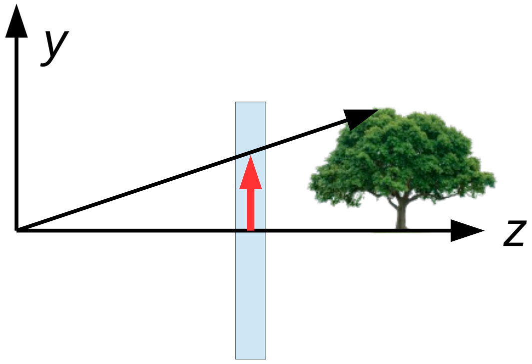
- Three different actors in a scene
  - Objects: exist in space, independent of viewer
  - Viewer: camera, human, …
  - Lights: shading, shadows, ...



From 3D World to 2D Screen

- Viewer
  - Pinhole camera (camera obscura)
    - Projection plane behind projection center: an inverted image
    - Easy mathematical description

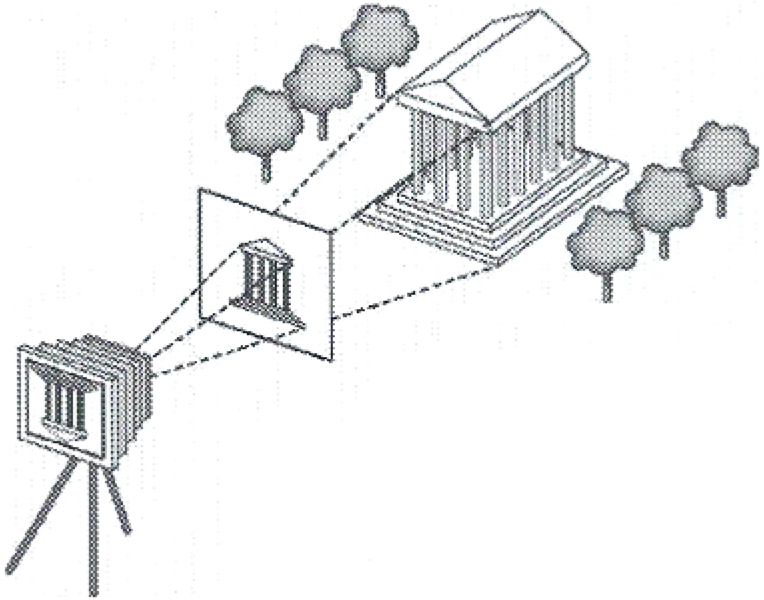




  - Synthetic camera
    - Projection plane in front of projection center: no inversion

From 3D World to 2D Screen

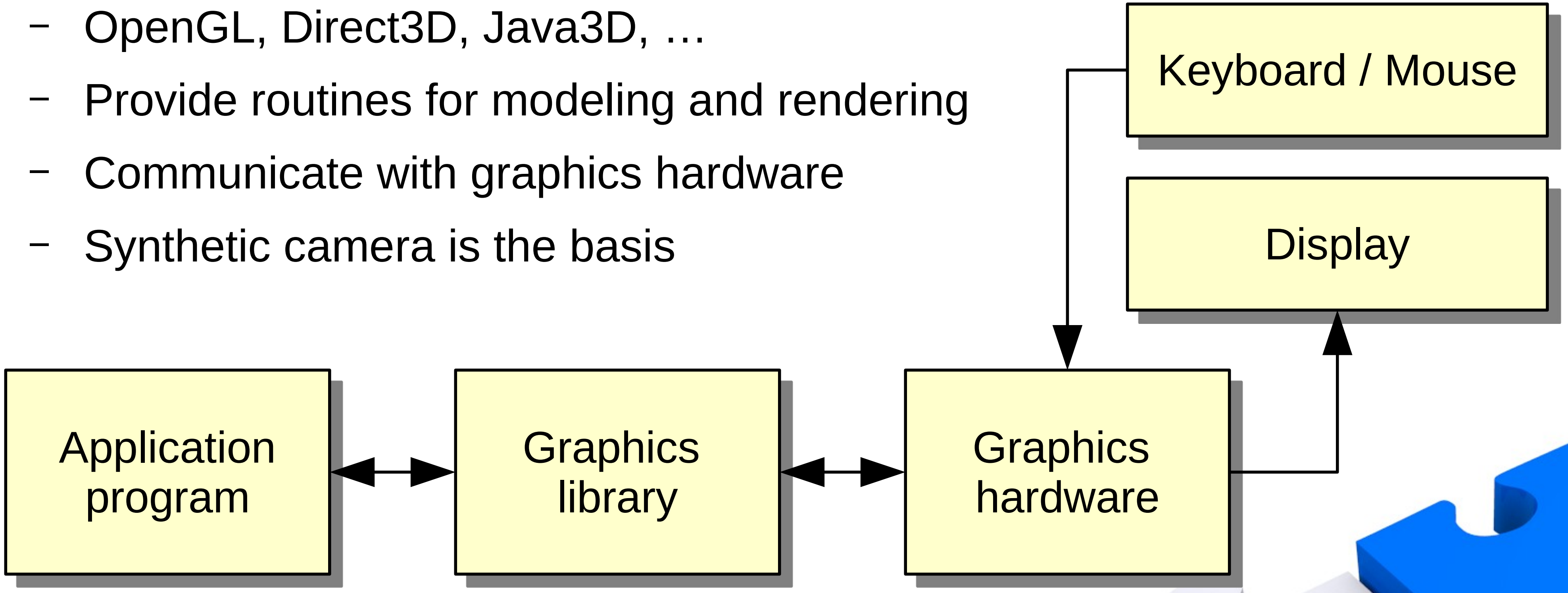
- Viewer
  - We don’t want to see everything
  - Clipping
    - Looking through a 2D window

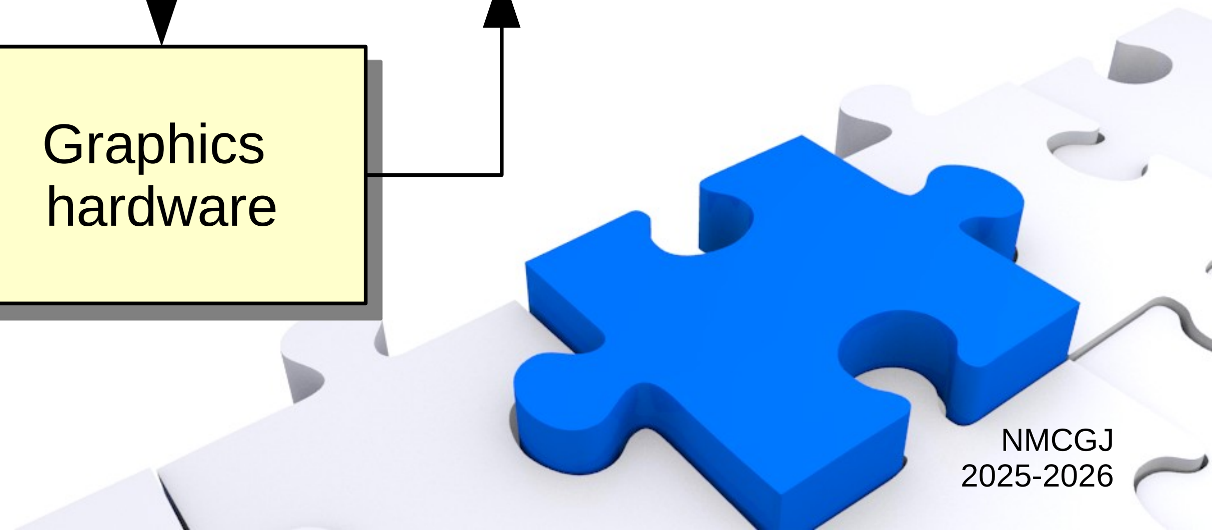


  - Two clipping models
    - 2D clipper: First project, and then cut everything outside window
    - 3D clipper: Cut everything outside view pyramid, and then project

From 3D World to 2D Screen

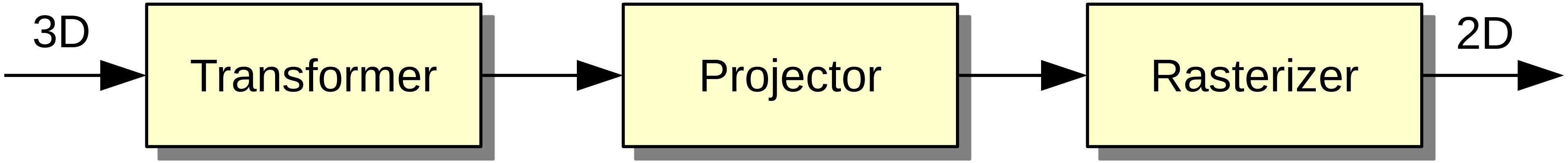
- 3D graphics libraries
  - OpenGL, Direct3D, Java3D, …
  - Provide routines for modeling and rendering
  - Communicate with graphics hardware
  - Synthetic camera is the basis





From 3D World to 2D Screen

- Rendering pipeline

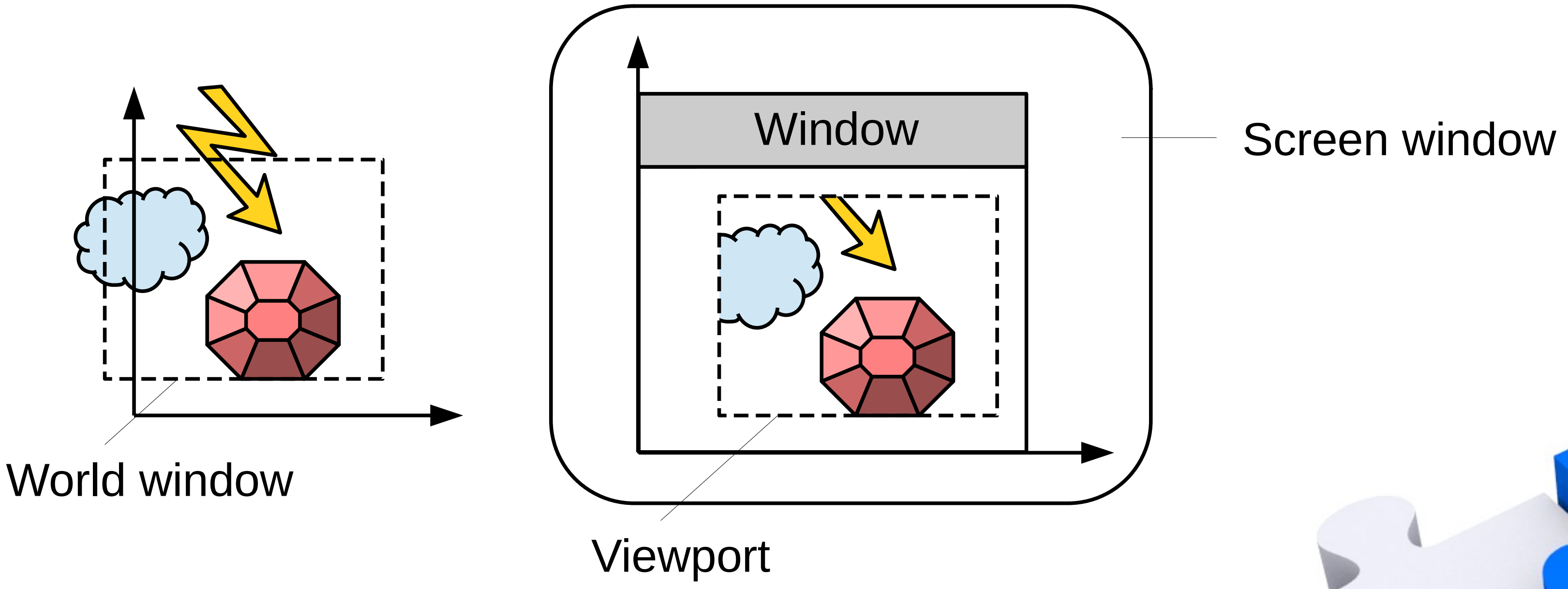


  - Conversion from 3D world vertices to 2D screen pixels
    - Transform to camera coordinate system (camera in origin)
    - Project 3D coordinates to 2D coordinates
      + Clip away everything we don’t see in window
    - Transform to pixels in the frame buffer

From 3D World to 2D Screen

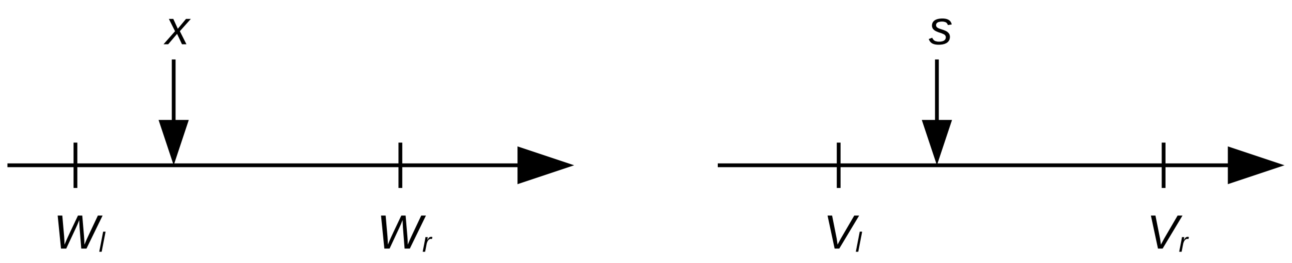
Università di Roma Tor Vergata

- World window versus viewport
  - World window: specifies which part of the world should be drawn
  - Viewport: rectangle in screen window in which we want to draw



From 3D World to 2D Screen

- World window versus viewport
  - Mapping $(x,y) \rightarrow (s,t)$ is linear



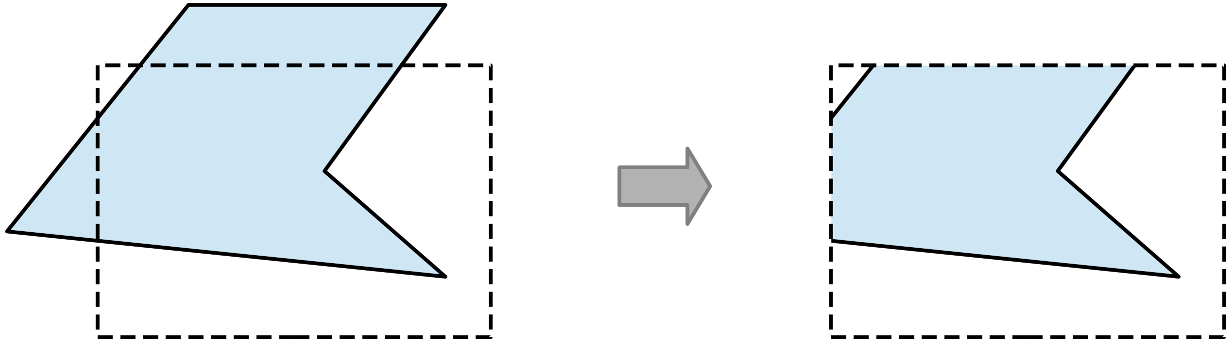
$$s = Ax + B, \qquad A = \frac{V_r - V_l}{W_r - W_l}, \qquad B = V_l - A W_l$$

  - Preserving aspect ratio (width/height) of world window
  - Maximizing and centering in viewport

From 3D World to 2D Screen

Università di Roma Tor Vergata

- 2D Clipping
  - Lines outside world window are not to be drawn



  - Algorithm *clipSegment*(...)
    - If line is within window then return *true* (accept)
    - If line is outside window then return *false* (reject)
    - Otherwise clip and return *true*

From 3D World to 2D Screen

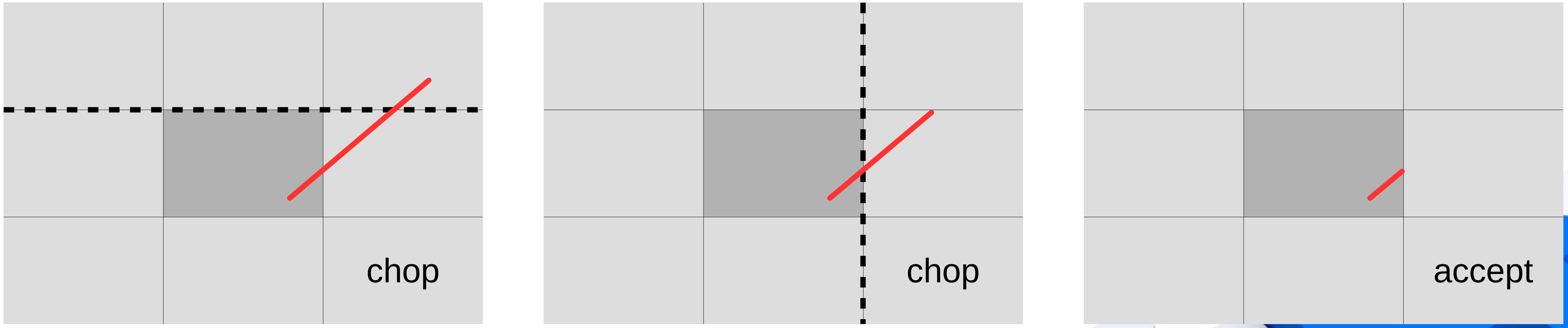
Università di Roma Tor Vergata

- 2D Clipping
  - Cohen-Sutherland region outcodes
    - Divide space into 9 regions
    - 4 bits per region
      - Left? Above? Right? Below?
    - Trivial accept:
      - Both endpoints are FFFF
    - Trivial reject:
      - Both endpoints have T in the same position

TTFF	FTFF	FTTF
TFFF	FFFF	FFTF
TFFT	FFFT	FFTT

From 3D World to 2D Screen

- 2D Clipping
  - Cohen-Sutherland chopping
    - If line is neither trivial accept nor reject
    - Then clip against edges of window repeatedly



From 3D World to 2D Screen

- 2D Clipping
  - Cohen-Sutherland line clipper

```
boolean clipSegment(Point p1, Point p2) {
    do {
        if (trivial accept) return true;
        if (trivial reject) return false;
        if (p1 is outside) {
            if (p1 is left) chop left;
            if (p1 is above) chop above;
            ...
        }
        if (p2 is outside) { ... }
    } while (true)
}
```

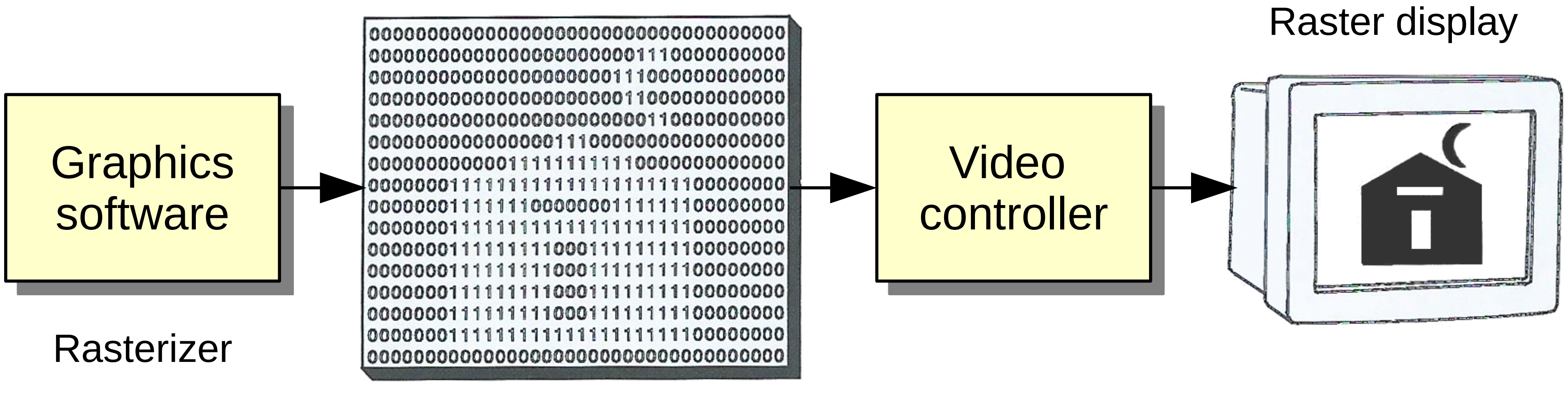
From 3D World to 2D Screen

- Rasterizing
  - Viewport on raster display
    - Cathode ray tube (CRT) monitor
    - Liquid crystal display (LCD) monitor
    - Image is discrete
  - Framebuffer
    - Raster image is stored in memory as a matrix of pixels (= picture elements)
    - The color of each pixel specifies the beam intensity
    - Video hardware scans framebuffer at 60Hz
      - Changes in framebuffer visible on screen => double buffering
      - Switch buffers when one buffer is finished

From 3D World to 2D Screen

Università di Roma Tor Vergata

- Rasterizing

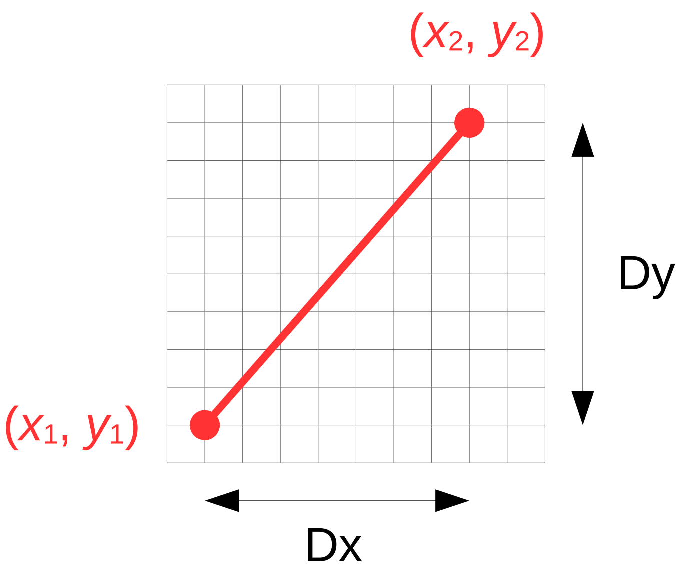


  - How to convert lines/polygons to pixels?
    - Continuous to discrete: scan conversion

From 3D World to 2D Screen

- Rasterizing
  - Scan converting lines
    - Find the pixels closest to the ideal line

$$(y - y_1) = m(x - x_1), \qquad m = \frac{Dy}{Dx} = \frac{y_2 - y_1}{x_2 - x_1}$$

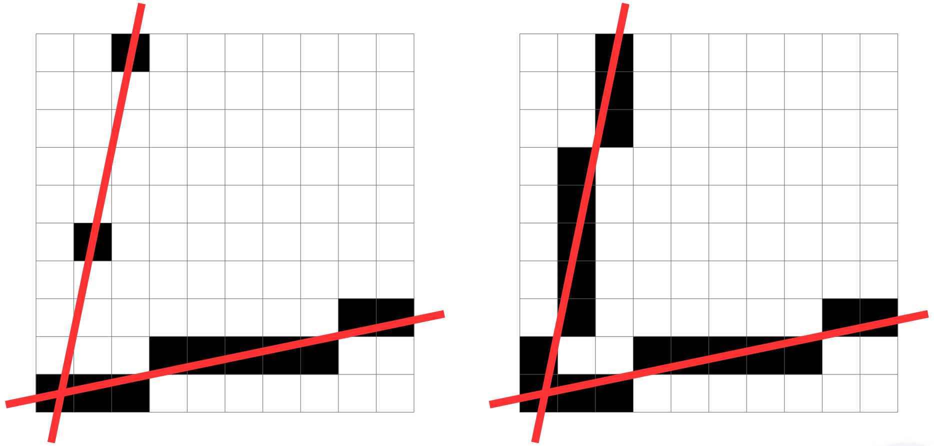


  - Naive algorithm
    - If slope $|m| \leq 1$: illuminate one pixel per column; work incrementally
    - If slope $|m| > 1$: illuminate one pixel per row; work incrementally (just $x \leftrightarrow y$)

From 3D World to 2D Screen

- Rasterizing
  - Scan converting lines: slope $|m| \leq 1$

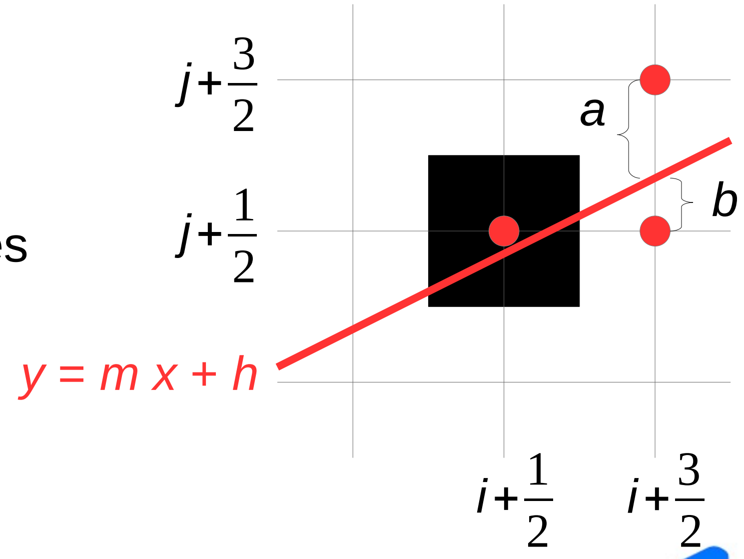
```
y = y1;
for (i = x1; i <= x2; i++) {
   plotPixel(i, Math.round(y));
   y += m;
}
```

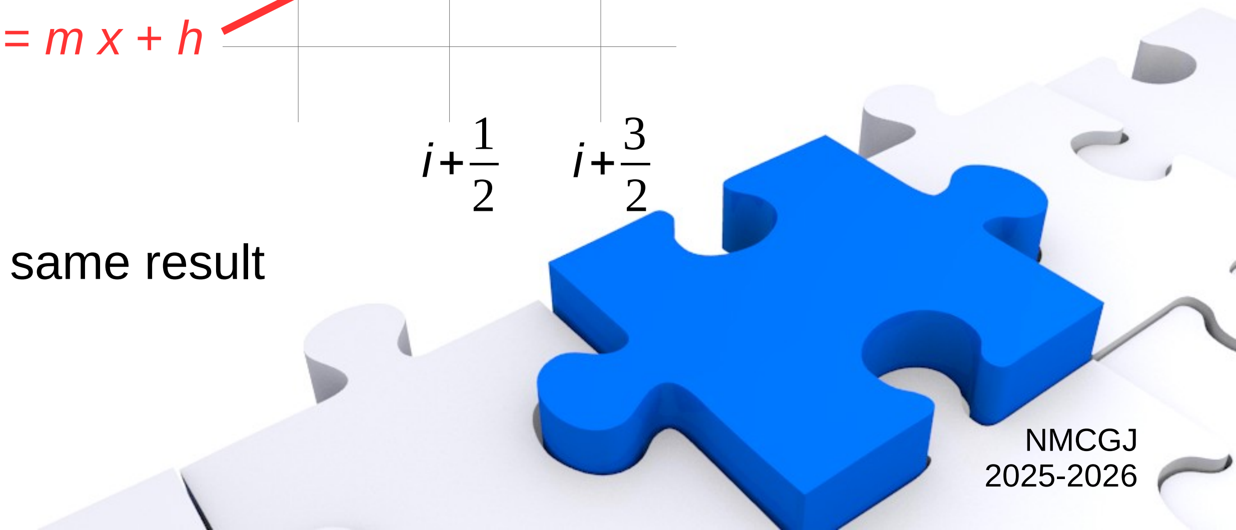


  - Inefficient:
    - Computation of round(y) for each integer x
    - And floating point addition

From 3D World to 2D Screen

- Rasterizing
  - Scan converting lines: Bresenham algorithm
    - Only integer arithmetic
  - What is the next pixel?
    - Assuming slope $0 \leq m \leq 1$, two possibilities
    - Decision variable: $d = a - b$
      - If ($d > 0$) … Else …
    - Alternative: $d = Dx\,(a - b)$
      - Only interested in sign, so this gives the same result
      - Incremental computation

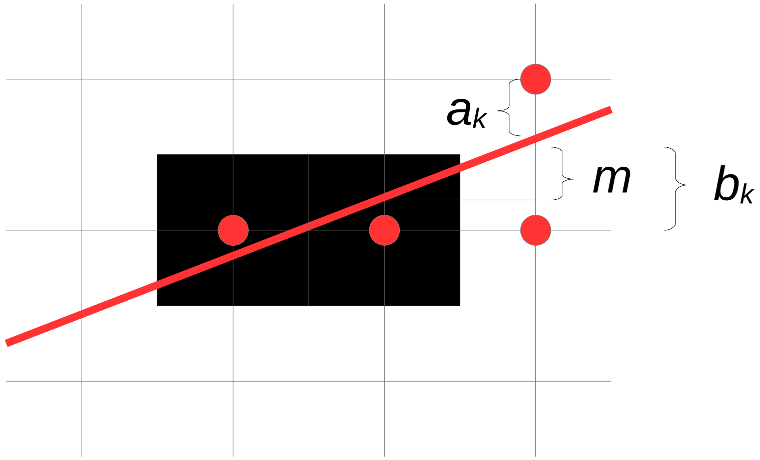


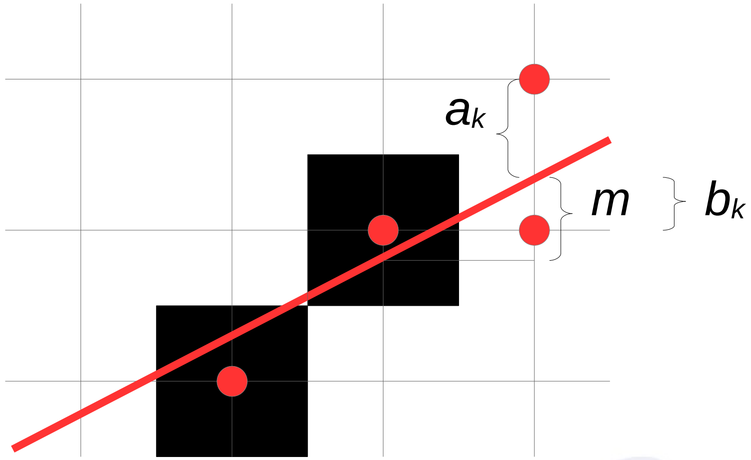


From 3D World to 2D Screen

- Rasterizing
  - Scan converting lines: Bresenham algorithm

$d_k = d_{k-1} - 2\,Dy$ or $d_k = d_{k-1} - 2\,(Dy - Dx)$





From 3D World to 2D Screen

Università di Roma Tor Vergata

- Rasterizing
  - Scan converting lines: Bresenham algorithm
    - $d_k = Dx\,(a_k - b_k)$
      $= Dx\,((a_{k-1} - m) - (b_{k-1} + m))$
      $= Dx\,(a_{k-1} - b_{k-1}) - 2\,Dx\,m$
      $= d_{k-1} - 2\,Dy$
    - $d_k = Dx\,(a_k - b_k)$
      $= Dx\,((2 - m - b_{k-1}) - (m - a_{k-1}))$
      $= Dx\,(a_{k-1} - b_{k-1}) - 2\,Dx\,(m - 1)$
      $= d_{k-1} - 2\,(Dy - Dx)$
    - Exercise: Write the entire algorithm