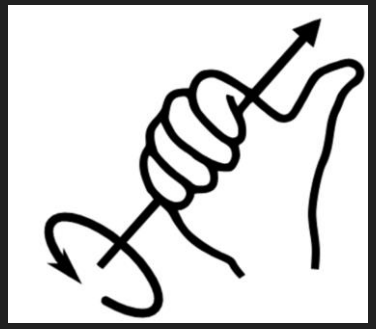
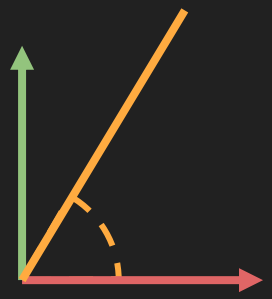
# Fundamentals of Computer Graphics and Image Processing

Computer Graphics - Exercise #02

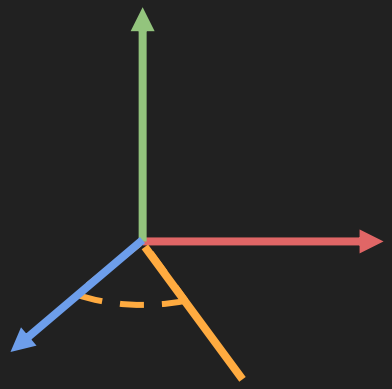
# Right-handed coordinate system



All rotational transformations are performed using right-hand rule



2D rotation
(around Z axis)



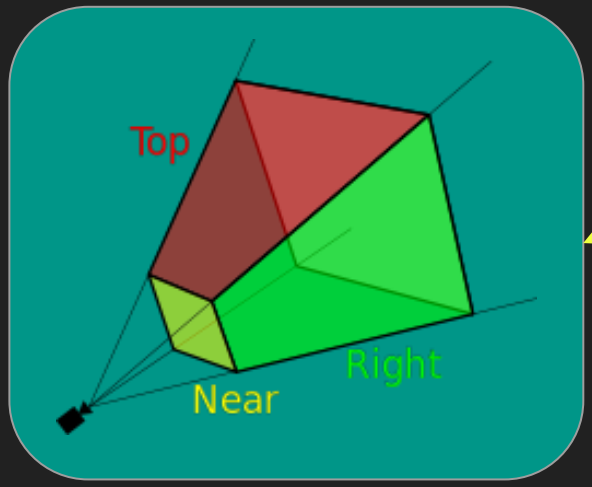
3D rotation
around Y axis

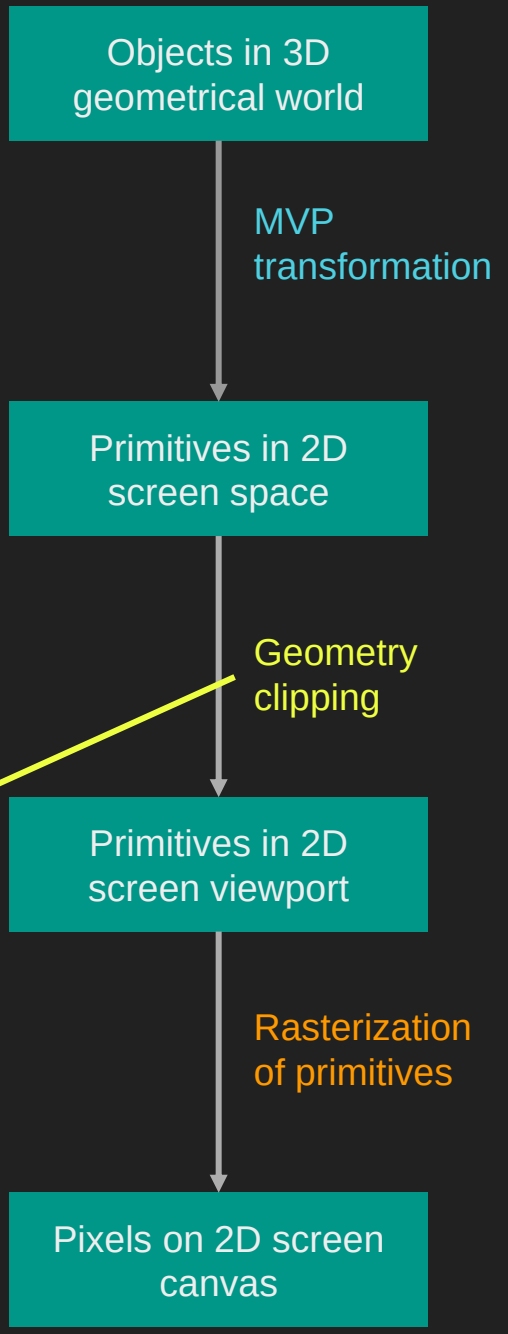
# Rendering Pipeline

Process of getting 3D models to screen pixels

Each step takes processing power (data filtering)

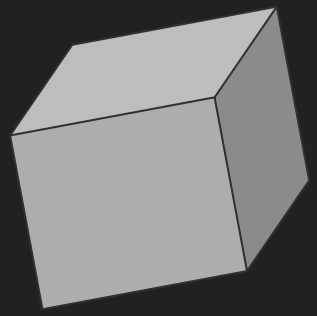
At each step, there can be additional actions performed

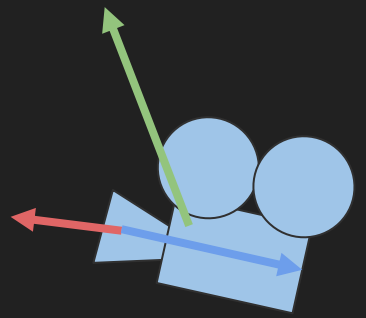


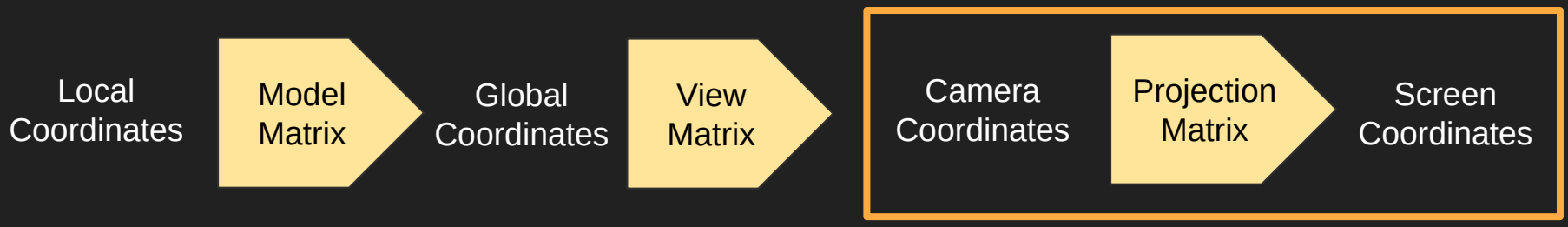



# Projection

How are primitives from camera coordinate system transformed to screen?



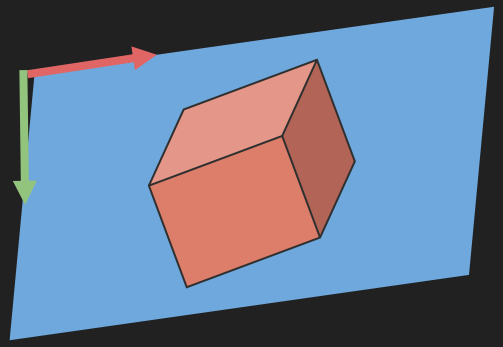


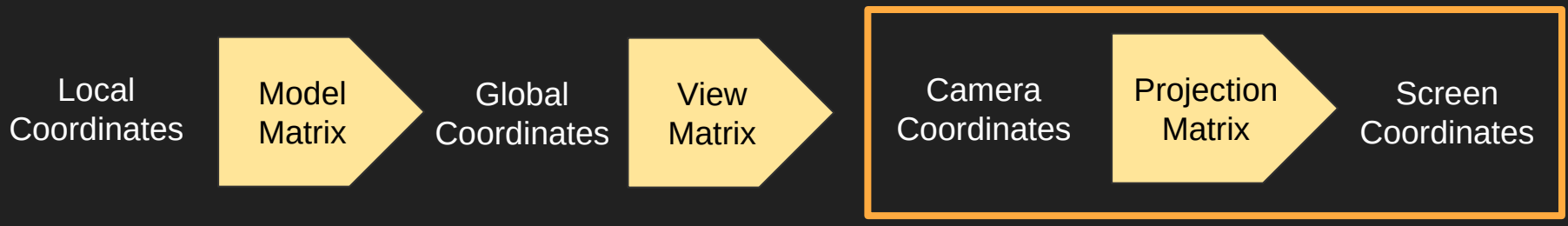


# Projection

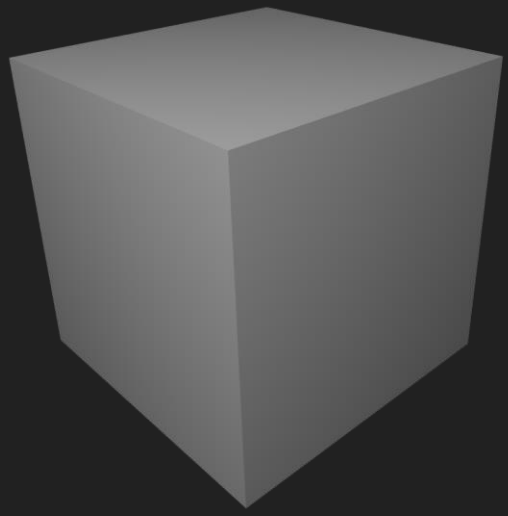
How are primitives from camera coordinate system transformed to screen?

3D geometry space is transformed to 2D screen space.





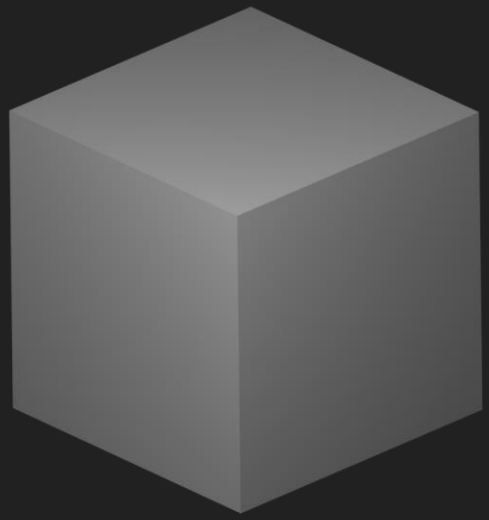
# Projection



**Perspective**

Distant objects appear smaller. Eye-like projection, looks natural for human observers.

**Orthographic**

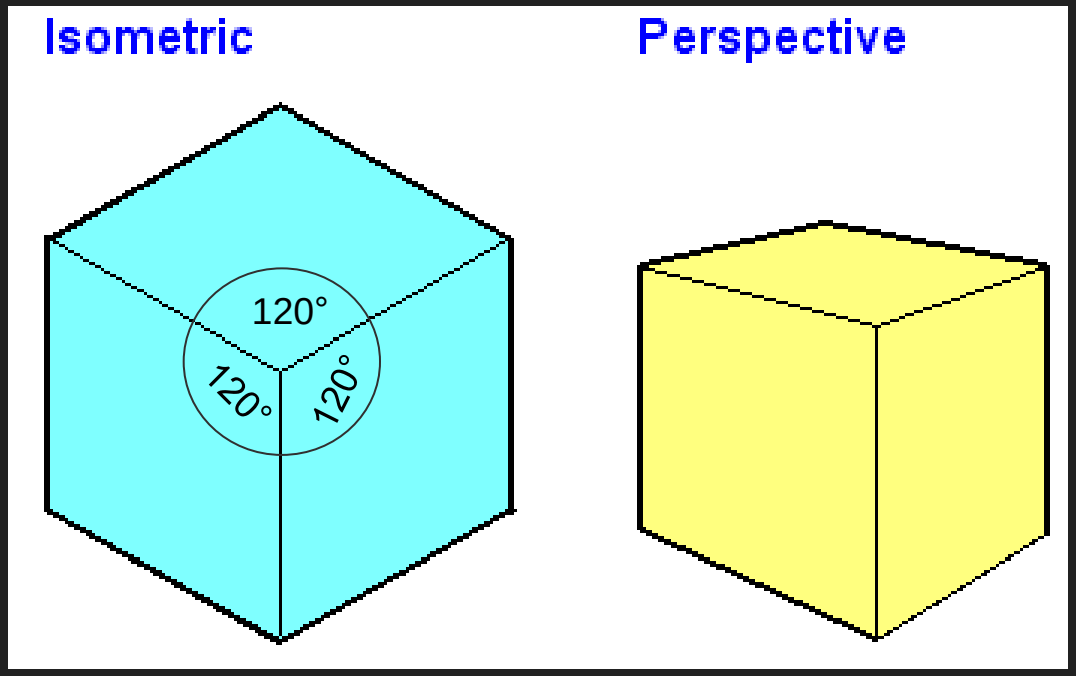


Parallel lines stay parallel, distant objects keep their size. Good for examining geometry.

Isometric

Special case of orthographic projection, where world axes X,Y,Z hold 120° in screen space.

# Isometric Projection



# Orthographic Projection

Geometry primitive in camera coordinate system = Z-axis vector and view direction are collinear

Ignoring Z coordinate performs an orthographic projection

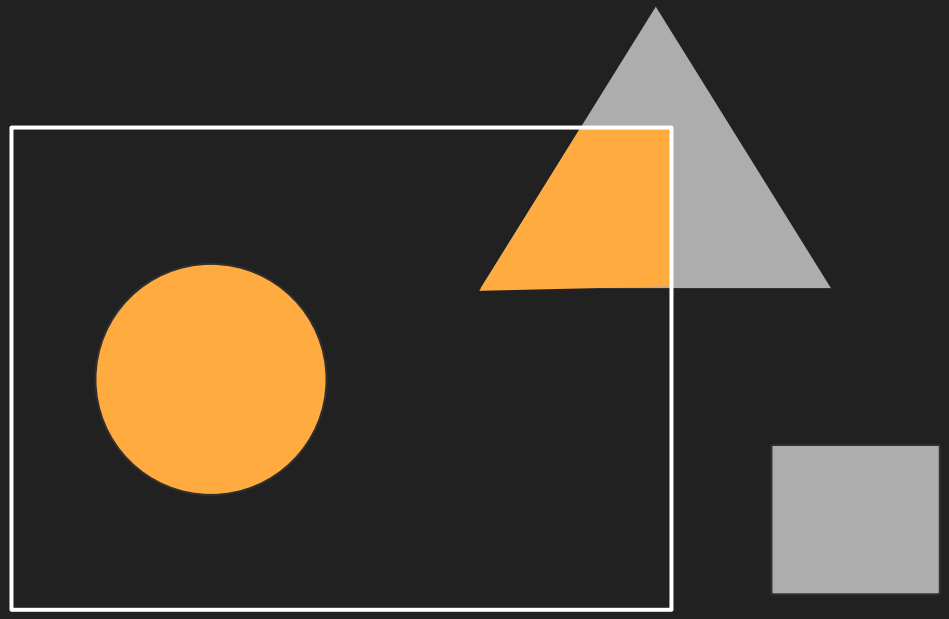
Then, we map the coordinates onto range (-1,1) => Normalized screen space

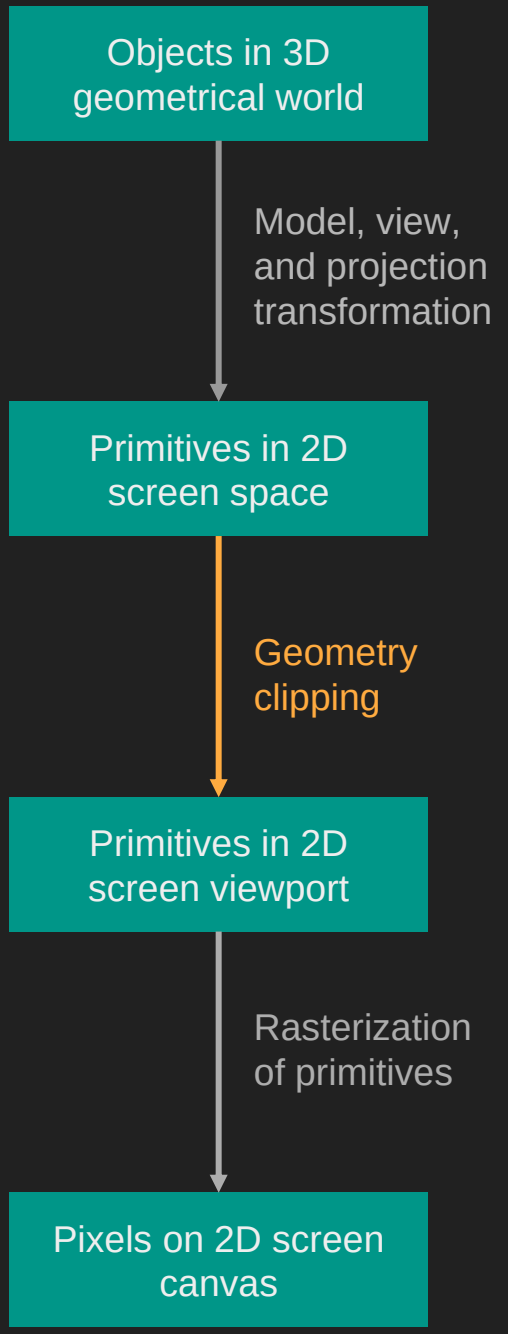
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \mathbf{0} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthographic projection matrix

# Clipping

Determines parts of geometry which are truly visible, therefore should be rendered





# Cohen-Sutherland Line Clipping Algorithm

Works only with rectangular clip regions (viewports)

1) Create binary code for both ending points of a line (b1, b2) using comparison to 4 sides of clip region ($x_{min}$ ,$y_{min}$ , $x_{max}$ , $y_{max}$)

   **$y > y_{max}$ | $y < y_{min}$ | $x > x_{max}$ | $x < x_{min}$**

2) Compare both codes using binary operations

   **b1 OR b2 = 0** ~ line is whole inside viewport (draw!)
   **b1 AND b2 != 0** ~ line is whole outside viewport (don't draw!)
   **ELSE** ~ line may be partially in viewport
   (continue to point 3)

3) Get a point which binary code is not equal to 0 and correct it by finding intersection with clip region sides.
   Take a new/corrected point and continue from point 1