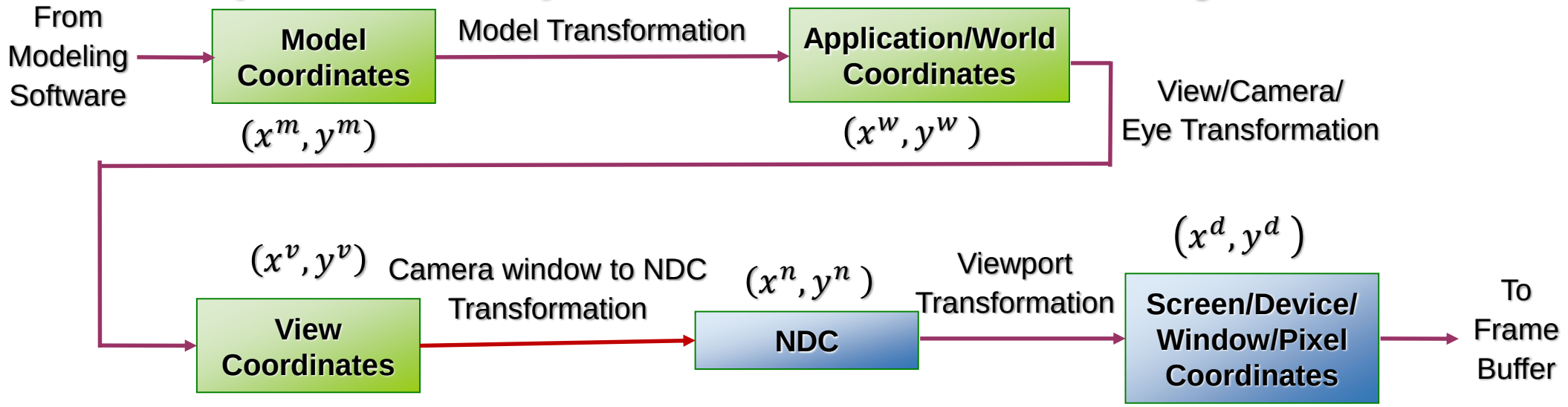
Introduction to Computer Graphics

Model and Camera Transformations

Prasanna Ghali

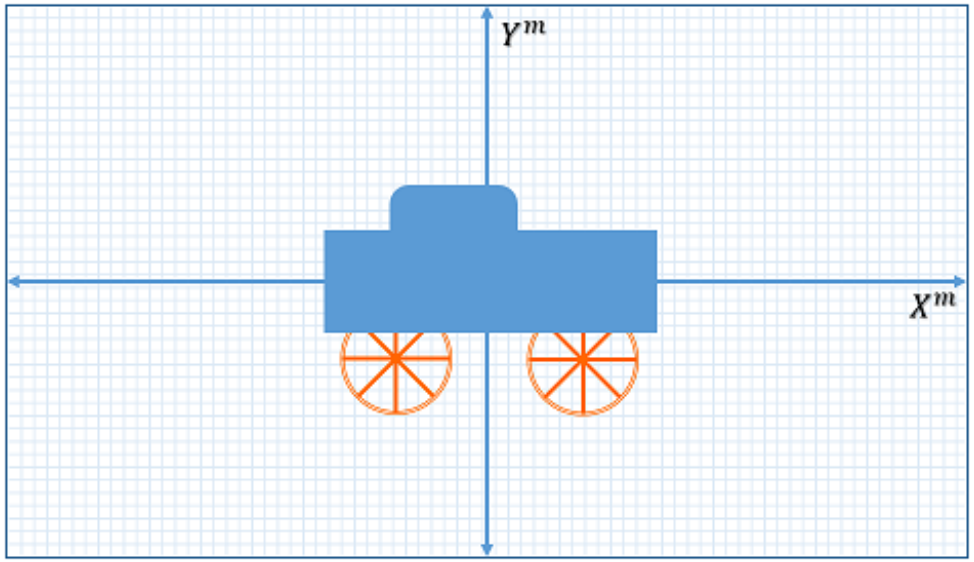
2D Graphics Pipe: Coordinate Systems



- Model coordinate system – each model has its own coordinate system
- World coordinate system – instances of models called objects are placed into common coordinate system
- Camera coordinate system – camera-centric coordinate system to generate image from point of view of camera
- NDC – standard square is axis-aligned square with sides of length 2, centered at origin

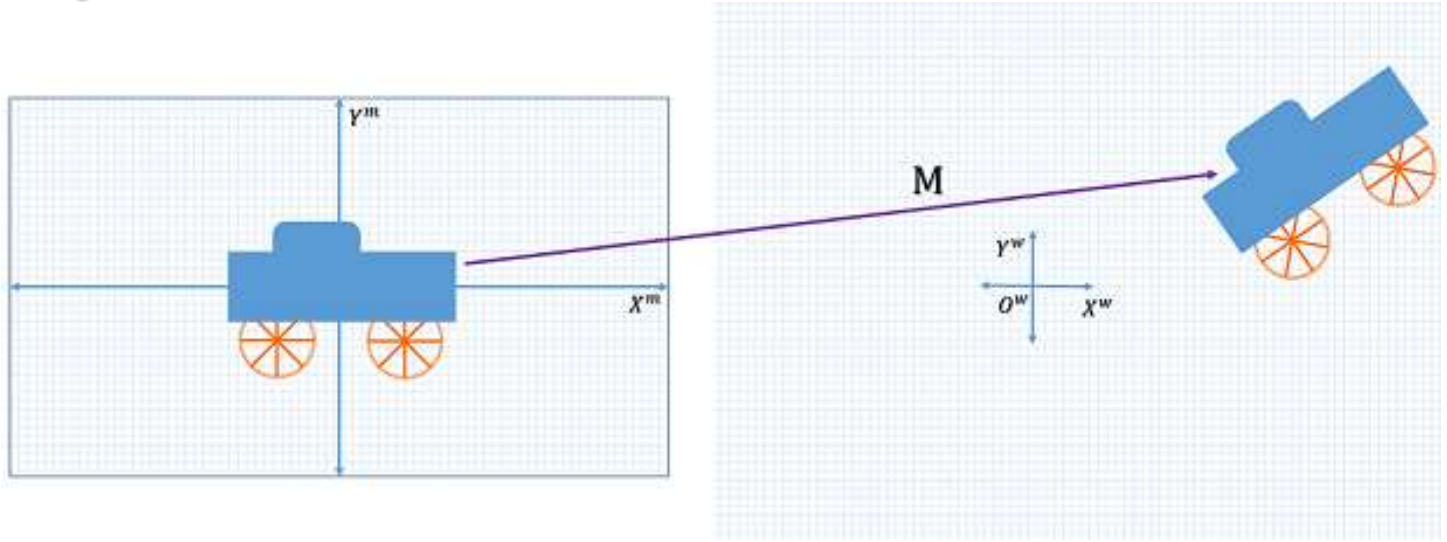
Model Transformations (1/4)

- Model described in model system by triangular mesh
  - Most common and simplest is for model system to coincide with world system



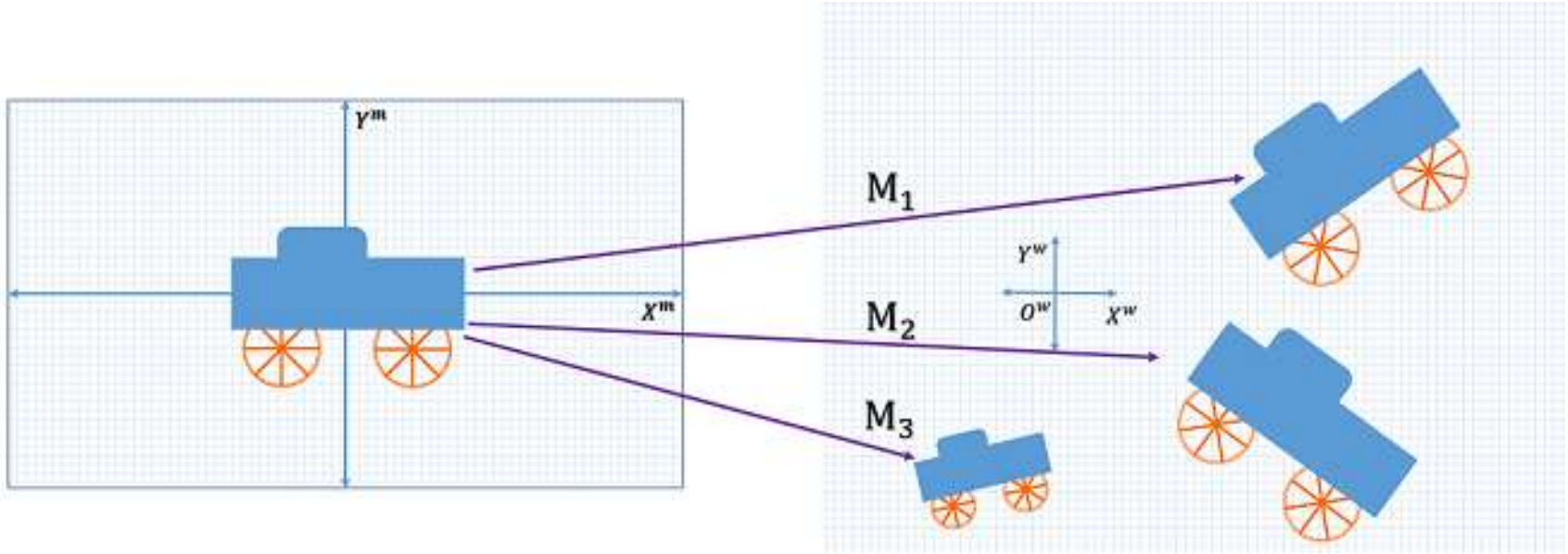
Model Transformations (2/4)

- To create scene in world system, we instantiate one or more objects from each model [that is defined in its model system]
  - Transformation matrix is necessary to appropriately size, orient, and position an instance of model in world system
  - This process is called *model*, or *model-to-object*, or *model-to-world*, *world*, or *object transformations*



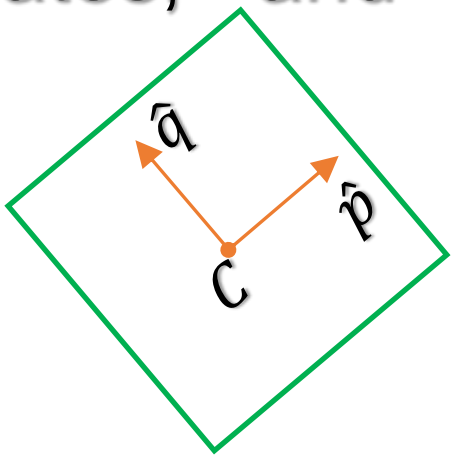
Model Transformations (3/4)

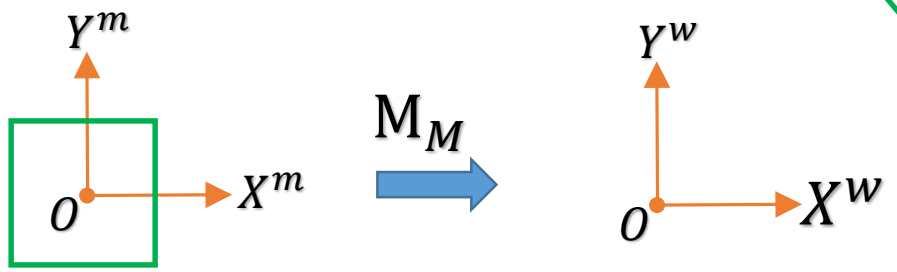
- Model specifies geometrical attributes in model system
- Object is single instantiation of model in world system
- We can have many objects instantiated from a model



Model Transformations (4/4)

- This transformation is applied only to position coordinates of mesh vertices
- For real-time lighting, normals require transformation
- Everything else including texture coordinates, and topology information are unaffected





SRT Technique

- Most common strategy for constructing model transforms is Scale-Rotate-Translate (SRT) method
- Strategy has 3 or 4 steps
  1) Translate model to origin [of model system]
     - Hopefully never necessary if model and world systems coincide
  2) Scale to target size
  3) Rotate to target orientation
  4) Translate to target location

SRT Technique: Example

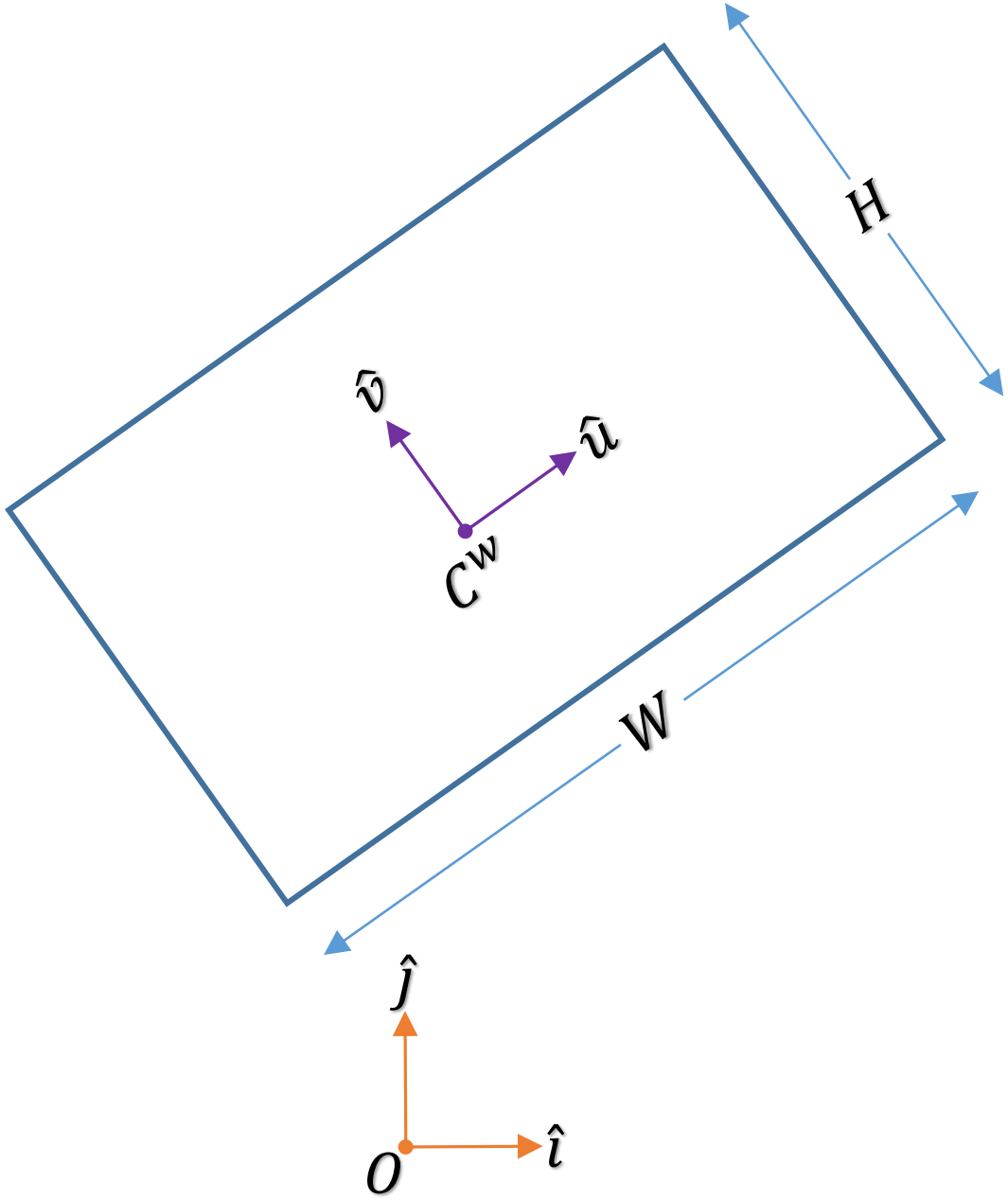
- Compute transformation matrix that maps standard square $([-1,1] \times [-1,1])$ to rectangle with width 20, height 16, rotated by $30°$, and centered at $(5, 12)$

$$T_{\langle 5,12 \rangle} \circ R_{30°} \circ H_{\langle 20/2,16/2 \rangle} = \left(T_{\langle 5,12 \rangle} \circ R_{30°}\right) \circ H_{\langle 10,8 \rangle}$$

$$= \begin{bmatrix} \sqrt{3}/2 & -1/2 & 5 \\ 1/2 & \sqrt{3}/2 & 12 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 10 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 5\sqrt{3} & -4 & 5 \\ 5 & 4\sqrt{3} & 12 \\ 0 & 0 & 1 \end{bmatrix}$$

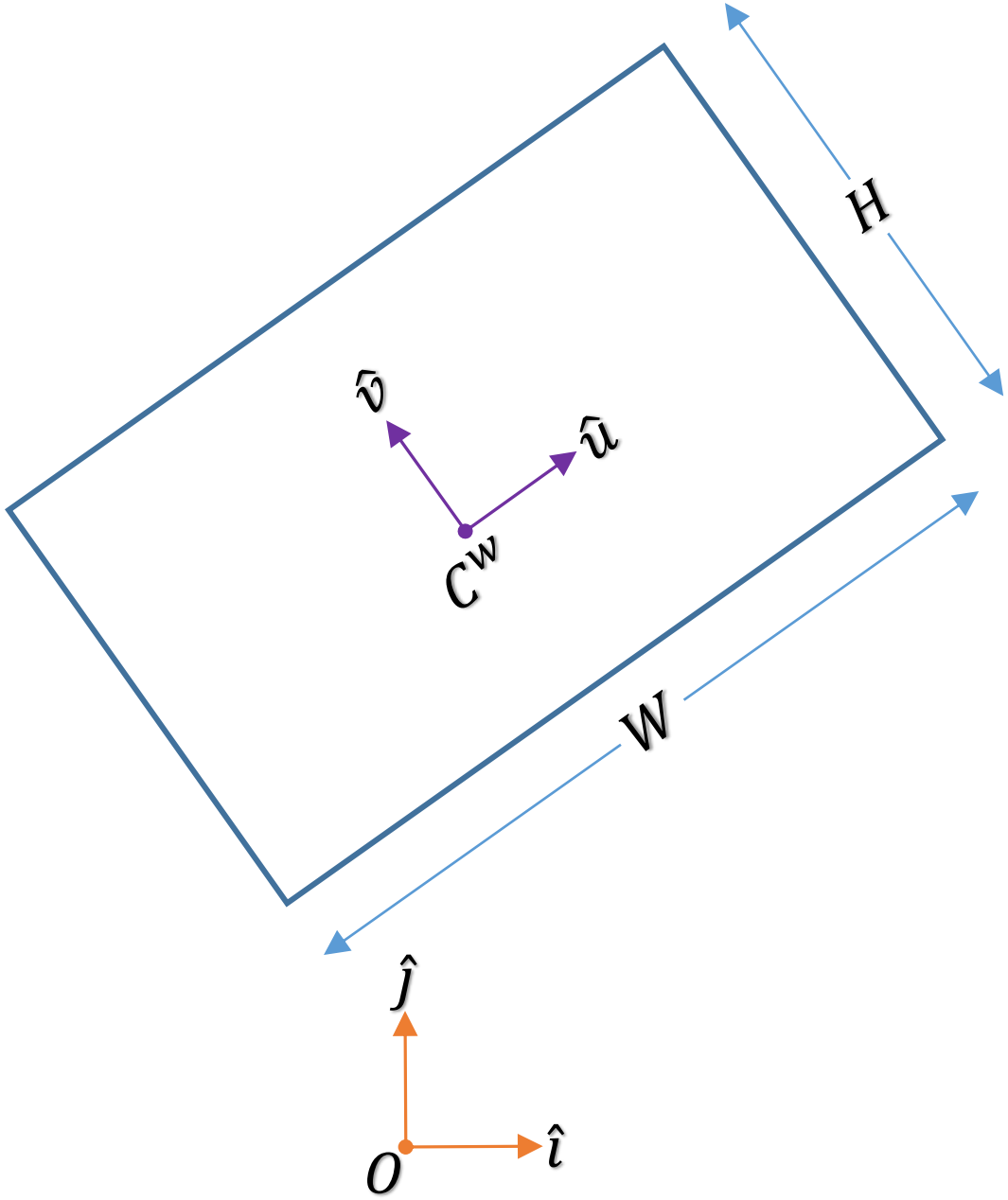
2D Camera Specs (1/2)

- Camera is modeled as oriented rectangle in world
  - Think of rectangle as "window to world"
- Camera specs
  - Center of window $C^w = (C_{\hat{\imath}}, C_{\hat{\jmath}})$
  - Orientation defined by right vector $\hat{u}$ and up vector $\hat{v}$
  - Window's width W and height H



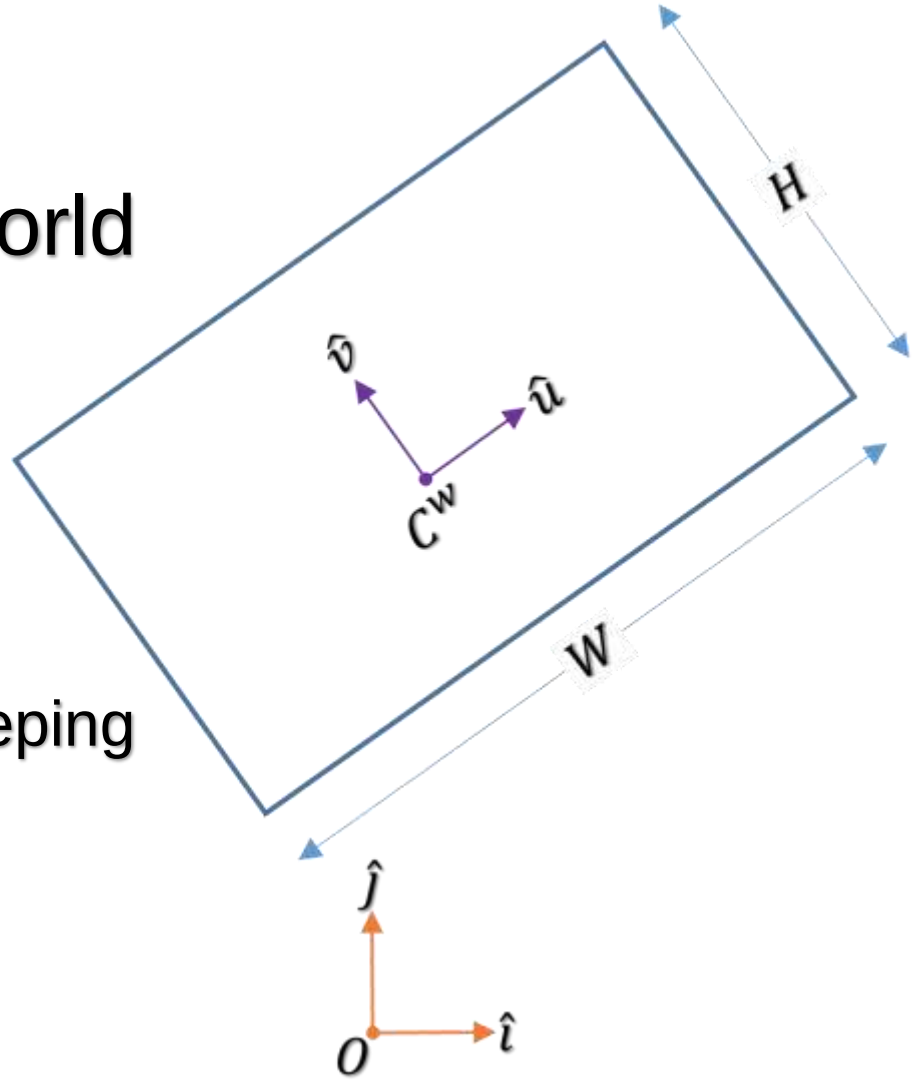
2D Camera Specs (2/2)

- Orientation vectors $\hat{u}$ and $\hat{v}$
  - Unit length: $\|\hat{u}\| = \|\hat{v}\| = 1$
  - Orthogonal: $\hat{u} \cdot \hat{v} = 0$
  - Form right-handed coordinate system: $\begin{Vmatrix} u_{\hat{\imath}} & v_{\hat{\imath}} \\ u_{\hat{\jmath}} & v_{\hat{\jmath}} \end{Vmatrix} > 0$



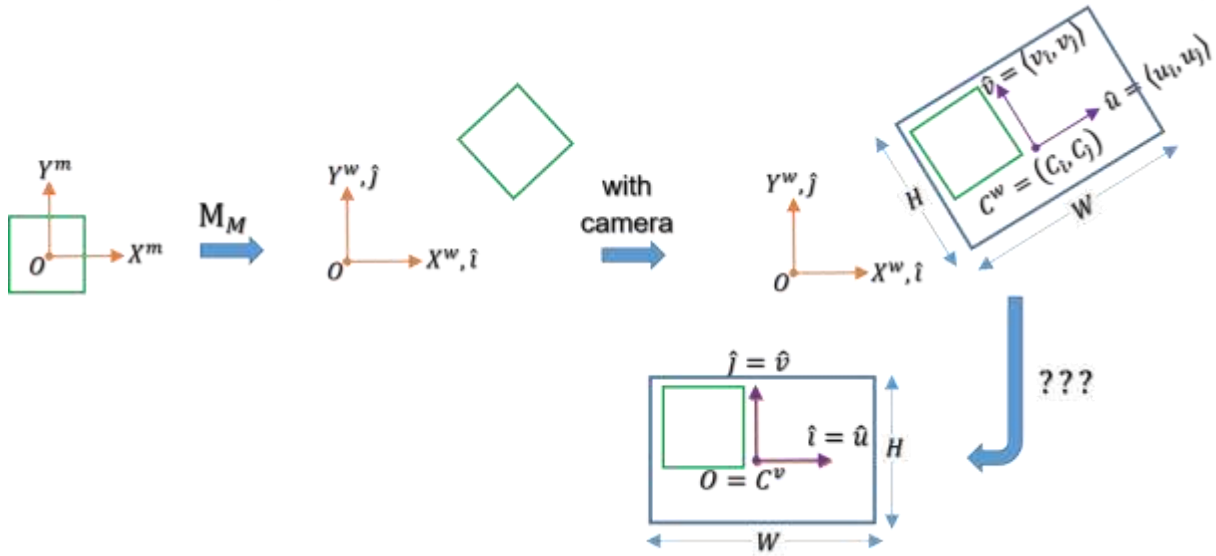
Camera Motion

- Camera and its specs “live” in world coordinate system
- Moving camera in world:
  - Translation: $C^w \leftarrow C^w + \vec{t}$
  - Rotation: $\hat{u} \leftarrow \mathrm{R}_{\theta^\circ}\hat{\imath}$, $\hat{v} \leftarrow \mathrm{R}_{\theta^\circ}\hat{\jmath}$
    - Must implement absolute rotation by keeping track of angular displacement $\theta°$
  - Zoom: $W \leftarrow \beta W$, $H \leftarrow \beta H$



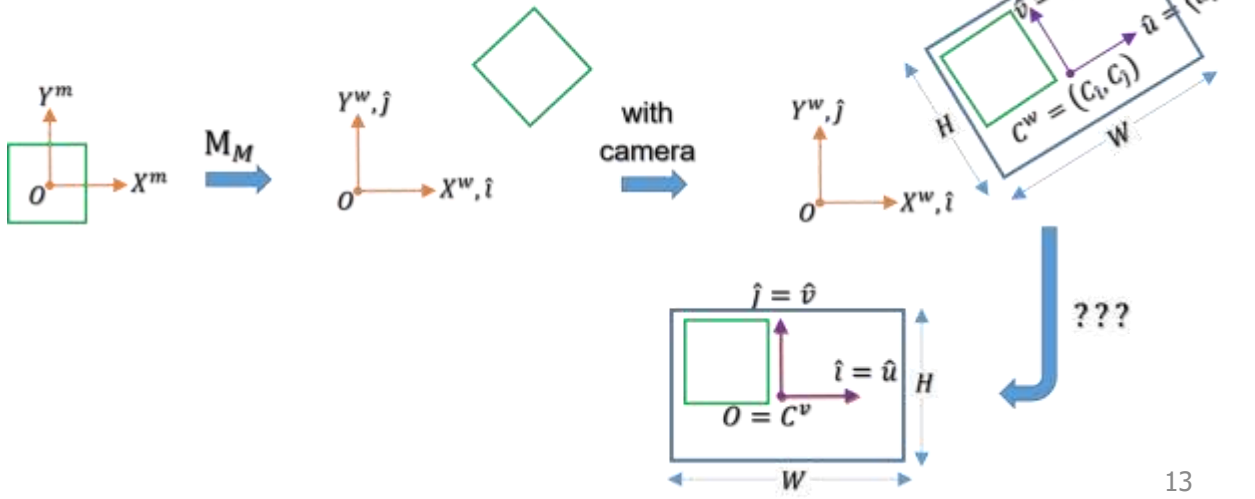
Camera (or View) Coordinate System (1/2)

- Image displayed to viewport is portion of scene viewed from perspective of camera that is in the interior of “window” of camera
- This portion of scene is easier to define if oriented camera “window” in world system is transformed into axis-aligned box in *camera* [also known as *view*] system
  - Things inside “window” are rendered; those outside “window” will be culled, clipped, and scissored out



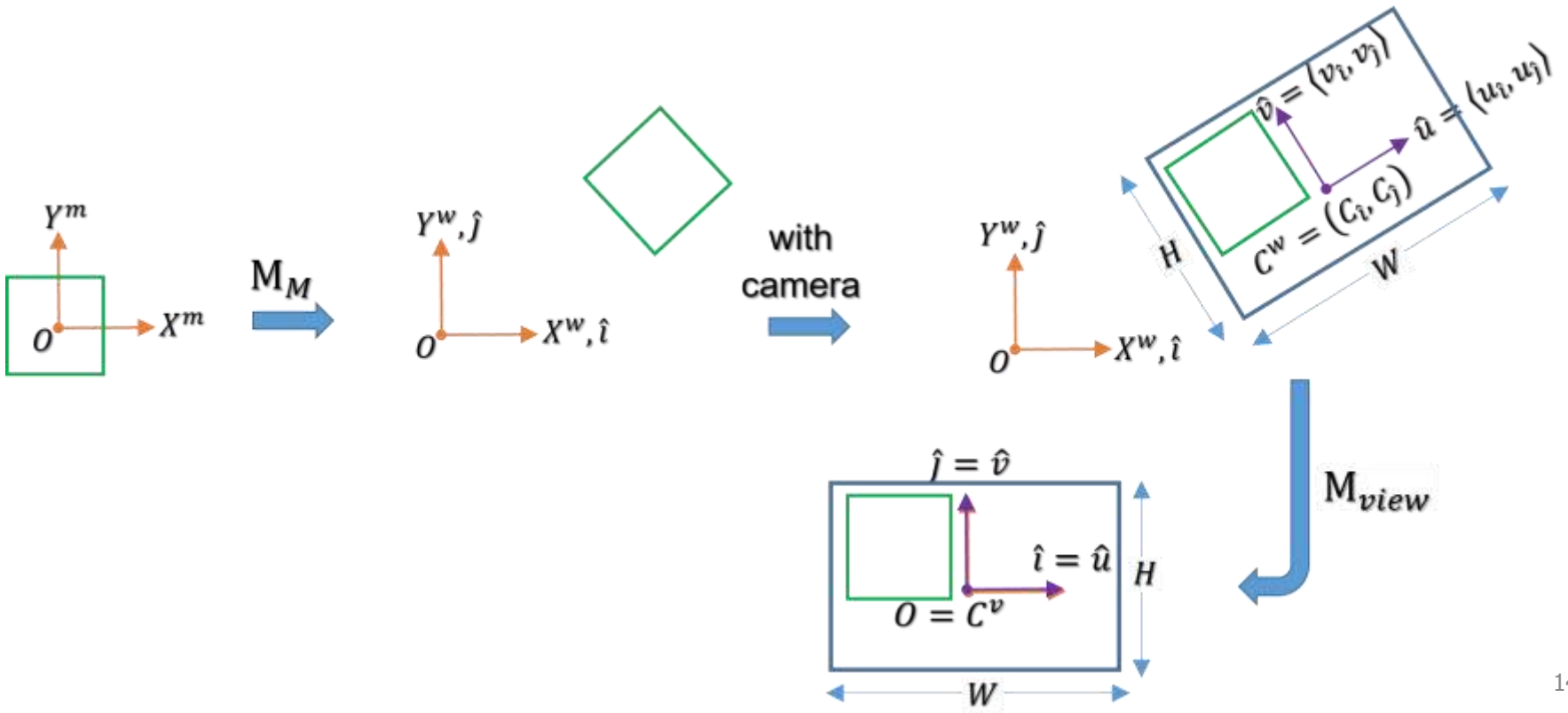
Camera (or View) Coordinate System (2/2)

- *Canonical* camera location and orientation [in camera system]:
  - Center of "window" is origin: $C^v = (0,0) \leftarrow C^w = \left(C_{\hat{\imath}}, C_{\hat{\jmath}}\right)$
  - Orientation vectors $\hat{u}$ and $\hat{v}$ transformed to view coordinate axes: $\hat{u} = \langle 1,0 \rangle \leftarrow \hat{u} = \left\langle u_{\hat{\imath}}, u_{\hat{\jmath}} \right\rangle$, $\hat{v} = \langle 0,1 \rangle \leftarrow \hat{v} = \left\langle v_{\hat{\imath}}, v_{\hat{\jmath}} \right\rangle$



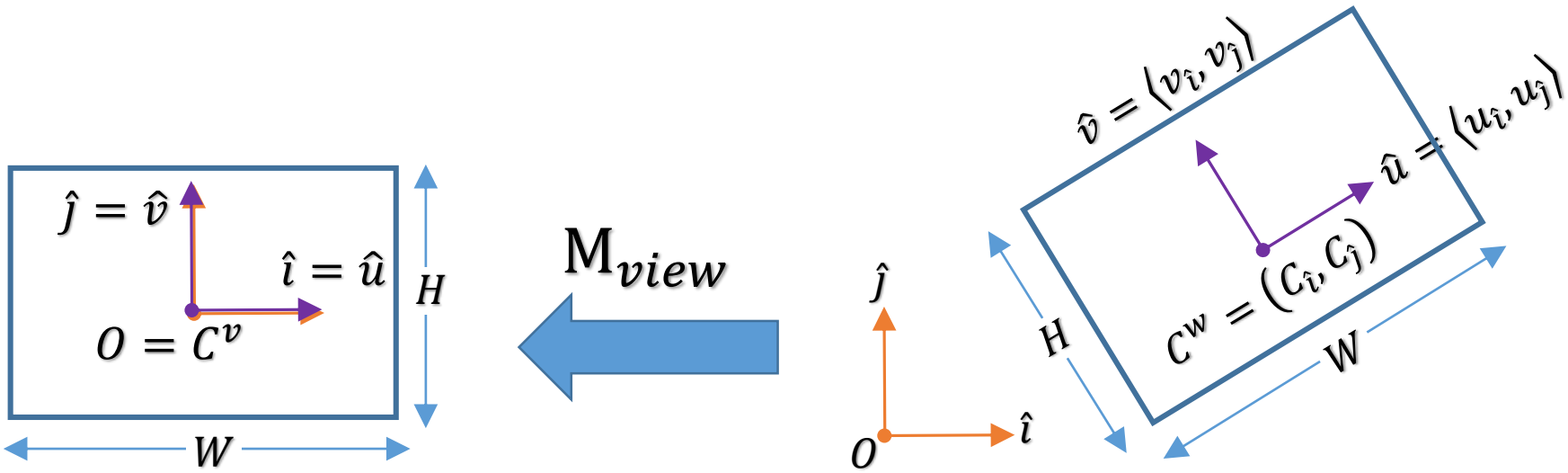
World-to-Camera (or View) Transformation (1/4)

- Transforms objects in world system to camera system so that view of scene is defined from camera's point of view



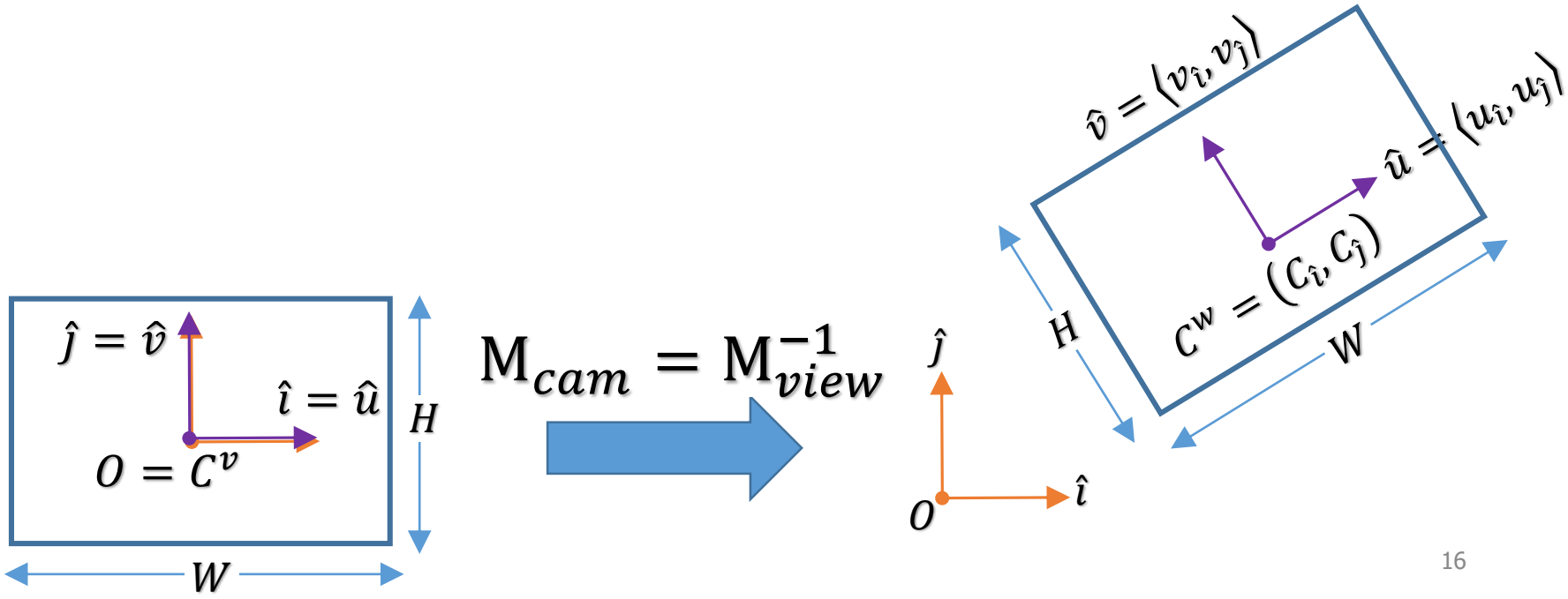
World-to-Camera (or View) Transformation (2/4)

- Transforms everything in world system to coordinate system called camera (or view) system such that
  - Camera's location $C^w = (C_{\hat{\imath}}, C_{\hat{\jmath}})$ in world system is now origin $C^v = (0, 0)$
  - Camera's orientation vectors $\hat{u} = \langle u_{\hat{\imath}}, u_{\hat{\jmath}} \rangle$ and $\hat{v} = \langle v_{\hat{\imath}}, v_{\hat{\jmath}} \rangle$ are transformed into $\hat{u} = \langle 1, 0 \rangle$ and $\hat{v} = \langle 0, 1 \rangle$, respectively



World-to-Camera (or View) Transformation (3/4)

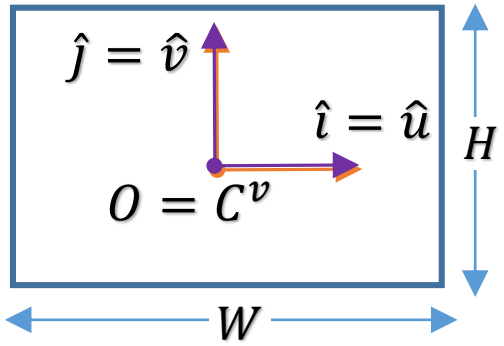
- Simpler to deduce matrix to transform axis-aligned camera window from camera system to oriented camera window in world system



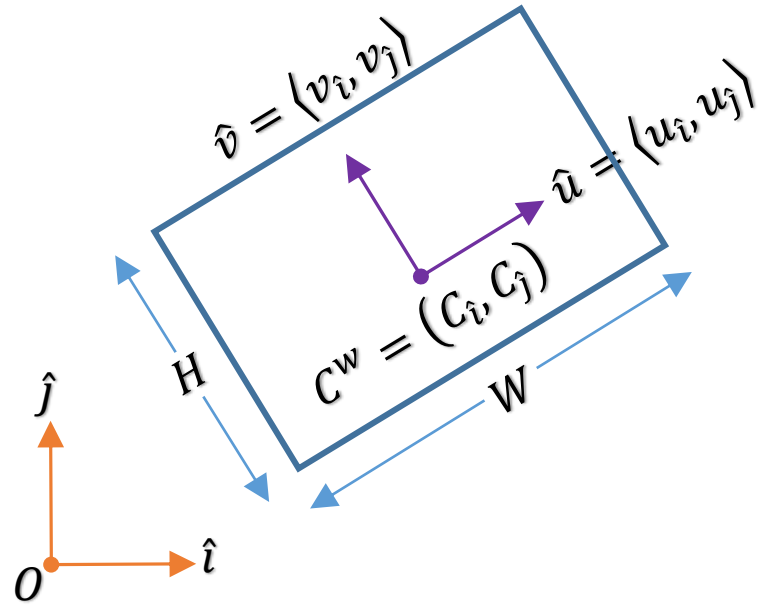
Camera-to-World Transformation

- Camera-to-world transformation transforms axis-aligned camera window from camera system to oriented camera window in world system

$$\mathrm{M}_{cam} = \mathrm{M}_{view}^{-1} = \begin{bmatrix} u_{\hat{\imath}} & v_{\hat{\imath}} & C_{\hat{\imath}} \\ u_{\hat{\jmath}} & v_{\hat{\jmath}} & C_{\hat{\jmath}} \\ 0 & 0 & 1 \end{bmatrix}$$



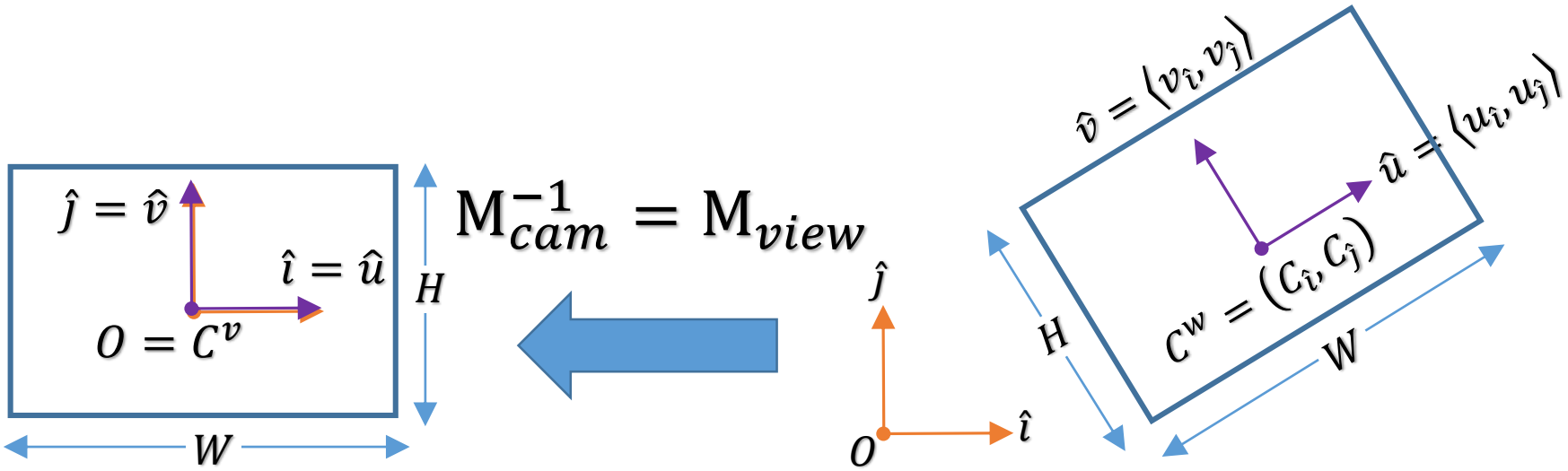
$\mathrm{M}_{cam} = \mathrm{M}_{view}^{-1}$



World-to-Camera (or View) Transformation (4/4)

$$\mathrm{M}_{cam} = \mathrm{M}_{view}^{-1} = \begin{bmatrix} u_{\hat{\imath}} & v_{\hat{\imath}} & C_{\hat{\imath}} \\ u_{\hat{\jmath}} & v_{\hat{\jmath}} & C_{\hat{\jmath}} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\mathrm{M}_{view} = \mathrm{M}_{cam}^{-1} = \begin{bmatrix} u_{\hat{\imath}} & v_{\hat{\imath}} & C_{\hat{\imath}} \\ u_{\hat{\jmath}} & v_{\hat{\jmath}} & C_{\hat{\jmath}} \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} u_{\hat{\imath}} & u_{\hat{\jmath}} & -\hat{u} \cdot C \\ v_{\hat{\imath}} & v_{\hat{\jmath}} & -\hat{v} \cdot C \\ 0 & 0 & 1 \end{bmatrix}$$



Inverse of Affine Transformation (1/2)

- Recall formula for inverse of 2×2 non-singular matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}, ad - bc \neq 0$$

- Inverse of a composition is $(\mathrm{A} \circ \mathrm{B})^{-1} = \mathrm{B}^{-1} \circ \mathrm{A}^{-1}$

Inverse of Affine Transformation (2/2)

- Inverse of 3×3 affine transformation matrix $\mathrm{A} = \mathrm{T}_{\vec{v}} \circ \mathrm{L}$ is:

$$\mathrm{A}^{-1} = (\mathrm{T}_{\vec{v}} \circ \mathrm{L})^{-1}$$

$$\Longrightarrow \mathrm{A}^{-1} = \mathrm{L}^{-1} \circ \mathrm{T}_{\vec{v}}^{-1}$$

$$\Longrightarrow \mathrm{A}^{-1} = \mathrm{L}^{-1} \circ \mathrm{T}_{-\vec{v}}$$

Inverse of Affine Transformation: Example

- Given $A = \begin{bmatrix} 5 & 1 & -3 \\ 8 & 2 & 6 \\ 0 & 0 & 1 \end{bmatrix}$, compute A^{-1}

$$A = T_{\vec{v}} \circ L = \begin{bmatrix} 5 & 1 & -3 \\ 8 & 2 & 6 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow L = \begin{bmatrix} 5 & 1 \\ 8 & 2 \end{bmatrix}, \vec{v} = \langle -3, 6 \rangle$$

$$A = T_{\vec{v}} \circ L \Longrightarrow A^{-1} = L^{-1} \circ T_{-\vec{v}}$$

$$L^{-1} = \frac{1}{(5)(2) - (8)(1)} \begin{bmatrix} 2 & -1 \\ -8 & 5 \end{bmatrix} = \begin{bmatrix} 1 & -1/2 \\ -4 & 5/2 \end{bmatrix}$$

$$A^{-1} = L^{-1} \circ T_{-\vec{v}} = \begin{bmatrix} 1 & -1/2 & 0 \\ -4 & 5/2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & -6 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -1/2 & 6 \\ -4 & 5/2 & -27 \\ 0 & 0 & 1 \end{bmatrix}$$

World-to-Camera (or View) Transformation: Example (1/3)

- Given camera's position $C^w = (2, -3)$ and orientation $\hat{u} = \langle 0, -1 \rangle, \hat{v} = \langle 1, 0 \rangle$, transform world system points $P_0^w = (3, -2), P_1^w = (2, -4)$, and $P_2^w = (3, -3)$ into points in camera (or view) system

World-to-Camera (or View) Transformation: Example (2/3)

$C^w = (2, -3), \hat{u} = \langle 0, -1 \rangle, \hat{v} = \langle 1, 0 \rangle$

$$\mathrm{M}_{cam} = \mathrm{M}_{view}^{-1} = [\hat{u} \quad \hat{v} \quad C] = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & -3 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{cam}^{-1} = \mathrm{M}_{view} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -2 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & -1 & -3 \\ 1 & 0 & -2 \\ 0 & 0 & 1 \end{bmatrix}$$

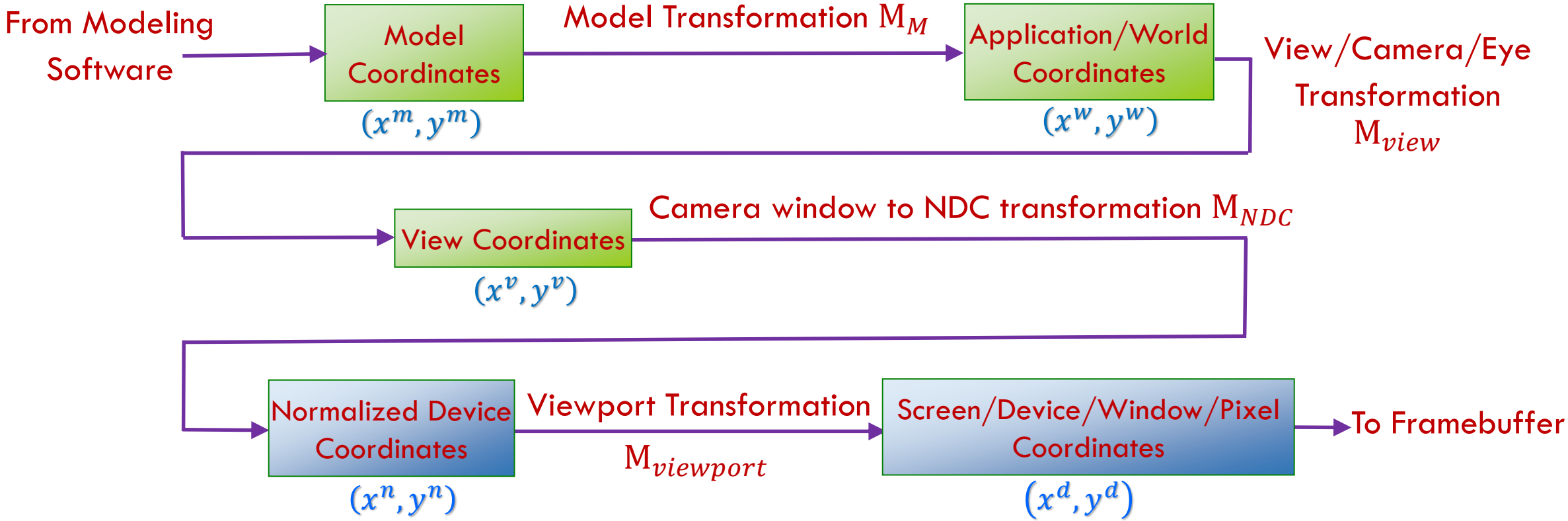
World-to-Camera (or View) Transformation: Example (3/3)

$$[P_0^v \quad P_1^v \quad P_2^v] = \mathrm{M}_{view}[P_0^w \quad P_1^w \quad P_2^w]$$

$$= \begin{bmatrix} 0 & -1 & -3 \\ 1 & 0 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 & 3 \\ -2 & -4 & -3 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

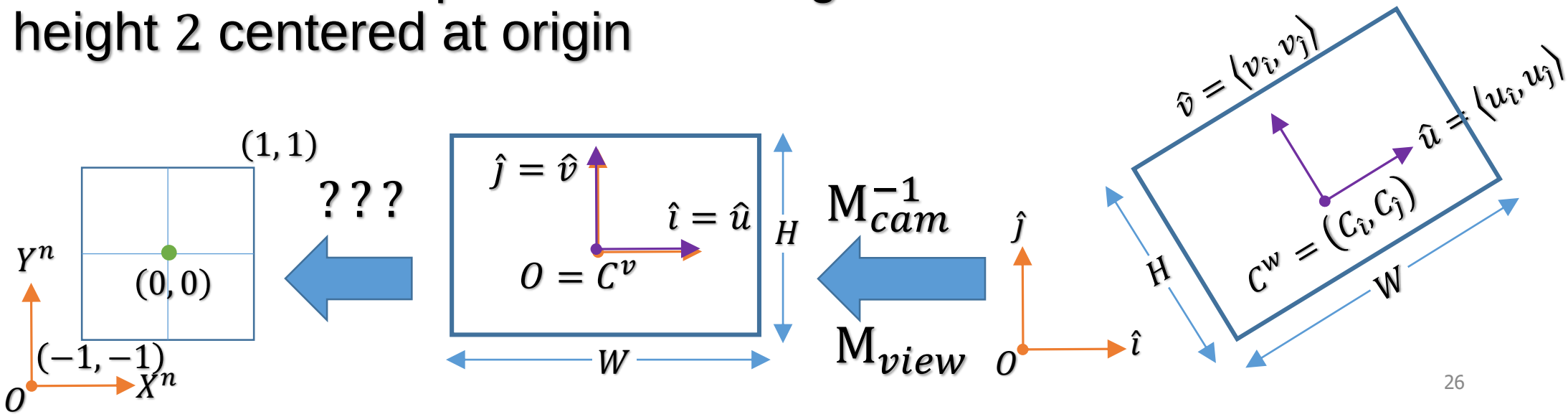
Camera-to-NDC Transform (1/3)



- In OpenGL, to display scene from camera's point of view, transformation from camera (or view) coordinates to NDC is required
- What does it look like geometrically?

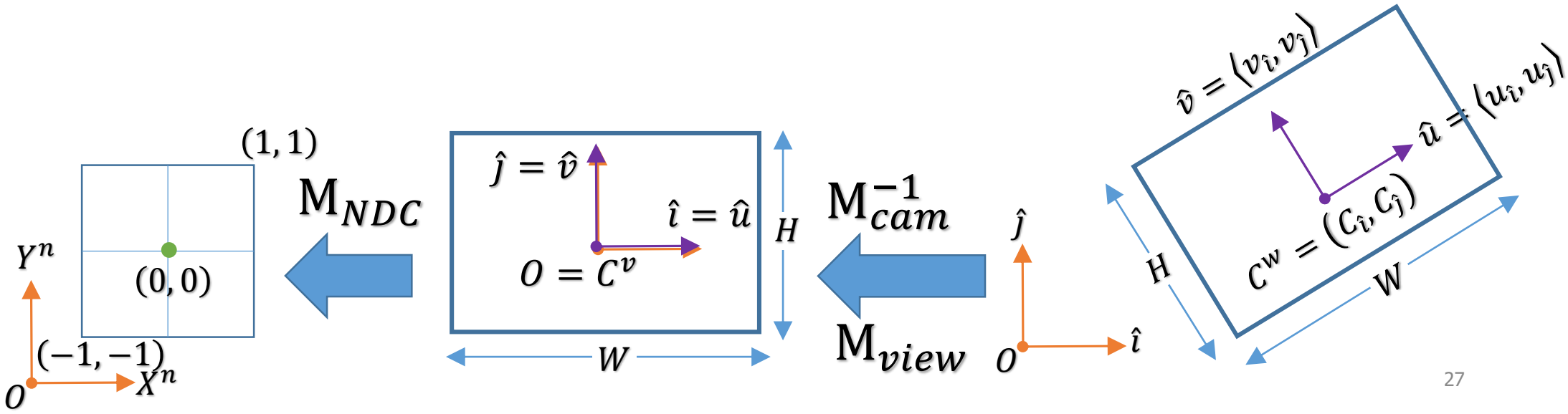
Camera-to-NDC Transform (2/3)

- Must map camera window [which is defined in camera system] to standard NDC square
  - In camera system, camera window is axis aligned rectangle with width W and height H centered at origin
  - Standard NDC square is axis aligned box with width 2 and height 2 centered at origin



Camera-to-NDC Transform (4/4)

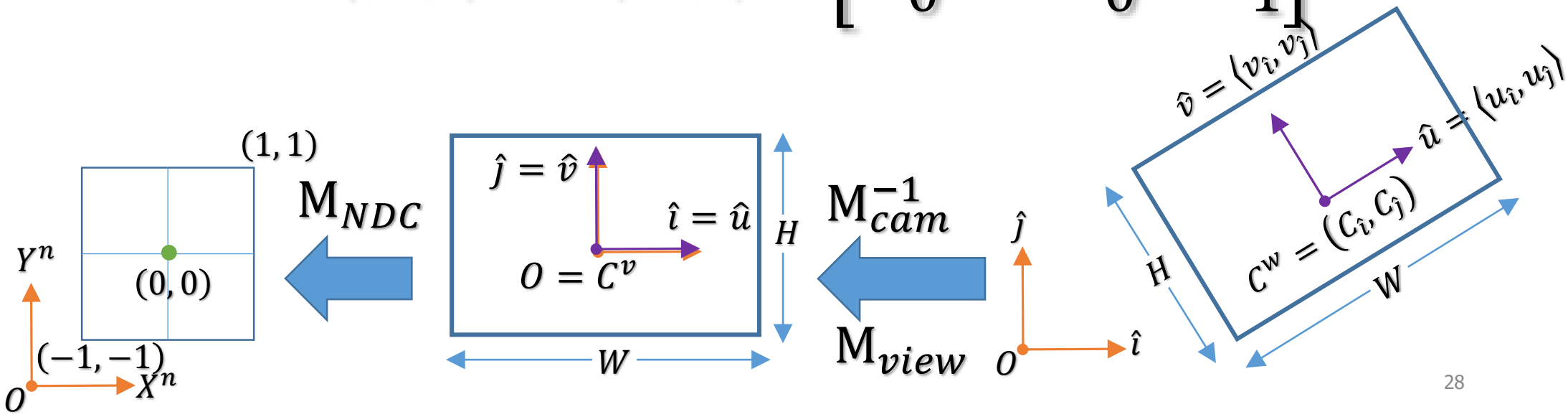
$$\mathrm{M}_{NDC} = \mathrm{H}_{\left\langle \frac{2}{W}, \frac{2}{H} \right\rangle} = \begin{bmatrix} 2/W & 0 & 0 \\ 0 & 2/H & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Camera-to-NDC Transform: Example

- If camera window has width $W = 80$ and height $H = 60$, compute M_{NDC}

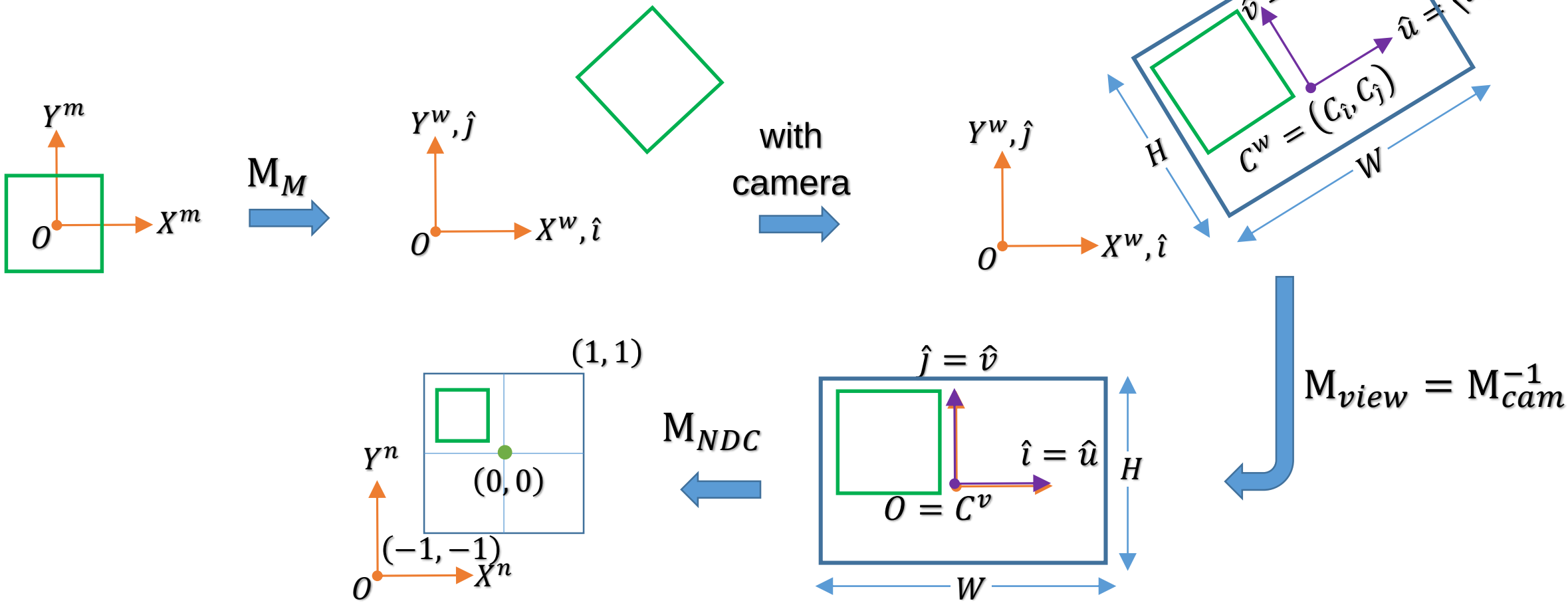
$$\mathrm{M}_{NDC} = \mathrm{H}_{\left\langle \frac{2}{80}, \frac{2}{60} \right\rangle} = \mathrm{H}_{\left\langle \frac{1}{40}, \frac{1}{30} \right\rangle} = \begin{bmatrix} 1/40 & 0 & 0 \\ 0 & 1/30 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



World-to-NDC Transform (1/2)

- Combine world-to-camera and camera-to-NDC transforms: $\mathrm{M}_{NDC} \circ \mathrm{M}_{view}$
- World-to-NDC transformation matrix maps oriented camera window in world system to standard NDC square [in NDC system]

World-to-NDC Transform (2/2)



World-to-NDC Transform: Example (1/3)

- Given following camera settings, compute world-to-NDC transformation matrix
  - Camera position in world system $C^w = (6, 7)$
  - Camera aimed so that its *up* vector is $\hat{v} = \langle 4/5\,, 3/5 \rangle$
  - Camera window has width $W = 10$ and height $H = 12$
- First step is to completely specify camera's orientation
  - Compute side vector $\hat{u}$ by rotating *up* vector $\hat{v}$ thro $-90°$ about z axis
  - $\hat{u} = \mathrm{R}_{\theta°}\hat{v} \Rightarrow \begin{bmatrix} u_{\hat{\imath}} \\ u_{\hat{\jmath}} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} v_{\hat{\imath}} \\ v_{\hat{\jmath}} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{\hat{\imath}} \\ u_{\hat{\jmath}} \end{bmatrix} = \begin{bmatrix} v_{\hat{\jmath}} \\ -v_{\hat{\imath}} \end{bmatrix}$

World-to-NDC Transform: Example (2/3)

- Next, compute camera-to-world transformation

$$\mathrm{M}_{cam} = [\hat{u} \quad \hat{v} \quad C^w] = \begin{bmatrix} 3/5 & 4/5 & 6 \\ -4/5 & 3/5 & 7 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 6 \\ 0 & 1 & 7 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3/5 & 4/5 & 0 \\ -4/5 & 3/5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Using camera-to-world transformation, compute world-to-camera transformation:

$$\mathrm{M}_{view} = \mathrm{M}_{cam}^{-1} = \begin{bmatrix} 3/5 & -4/5 & 0 \\ 4/5 & 3/5 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -6 \\ 0 & 1 & -7 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3/5 & -4/5 & 2 \\ 4/5 & 3/5 & -9 \\ 0 & 0 & 1 \end{bmatrix}$$

World-to-NDC Transform: Example (3/3)

- Compute camera-to-NDC transformation

$$\mathrm{M}_{NDC} = \mathrm{H}_{\left\langle \frac{2}{10}, \frac{2}{12} \right\rangle} = \begin{bmatrix} 1/5 & 0 & 0 \\ 0 & 1/6 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

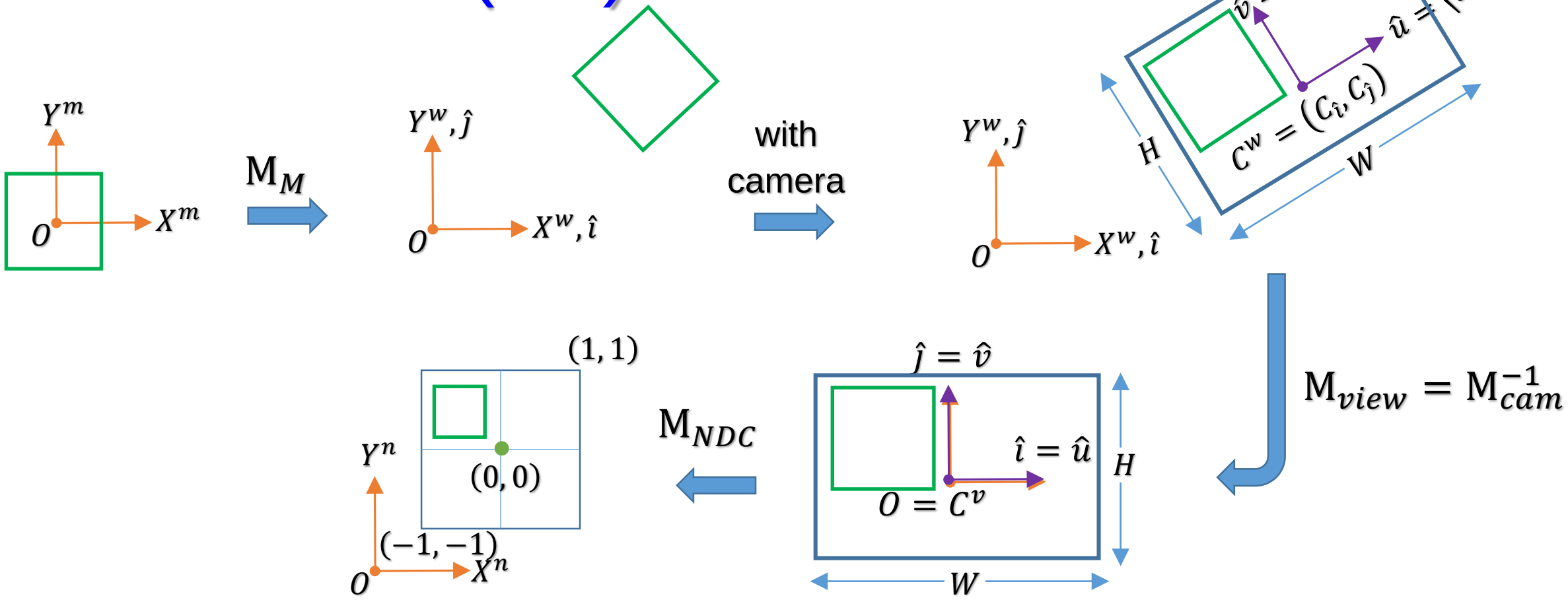
- Finally, we're ready to compute world-to-NDC transformation:

$$\mathrm{M}_{NDC} \circ M_{view} = \begin{bmatrix} 1/5 & 0 & 0 \\ 0 & 1/6 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3/5 & -4/5 & 2 \\ 4/5 & 3/5 & -9 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 3/25 & -4/25 & 2/5 \\ 2/15 & 1/10 & -3/2 \\ 0 & 0 & 1 \end{bmatrix}$$

Viewing an Object: Model-to-NDC Transform (1/3)

- Compute model-to-NDC transform: $\mathrm{M}_{NDC} \circ \mathrm{M}_{view} \circ \mathrm{M}_{M}$ where M_{M} is model-to-world (model) transformation

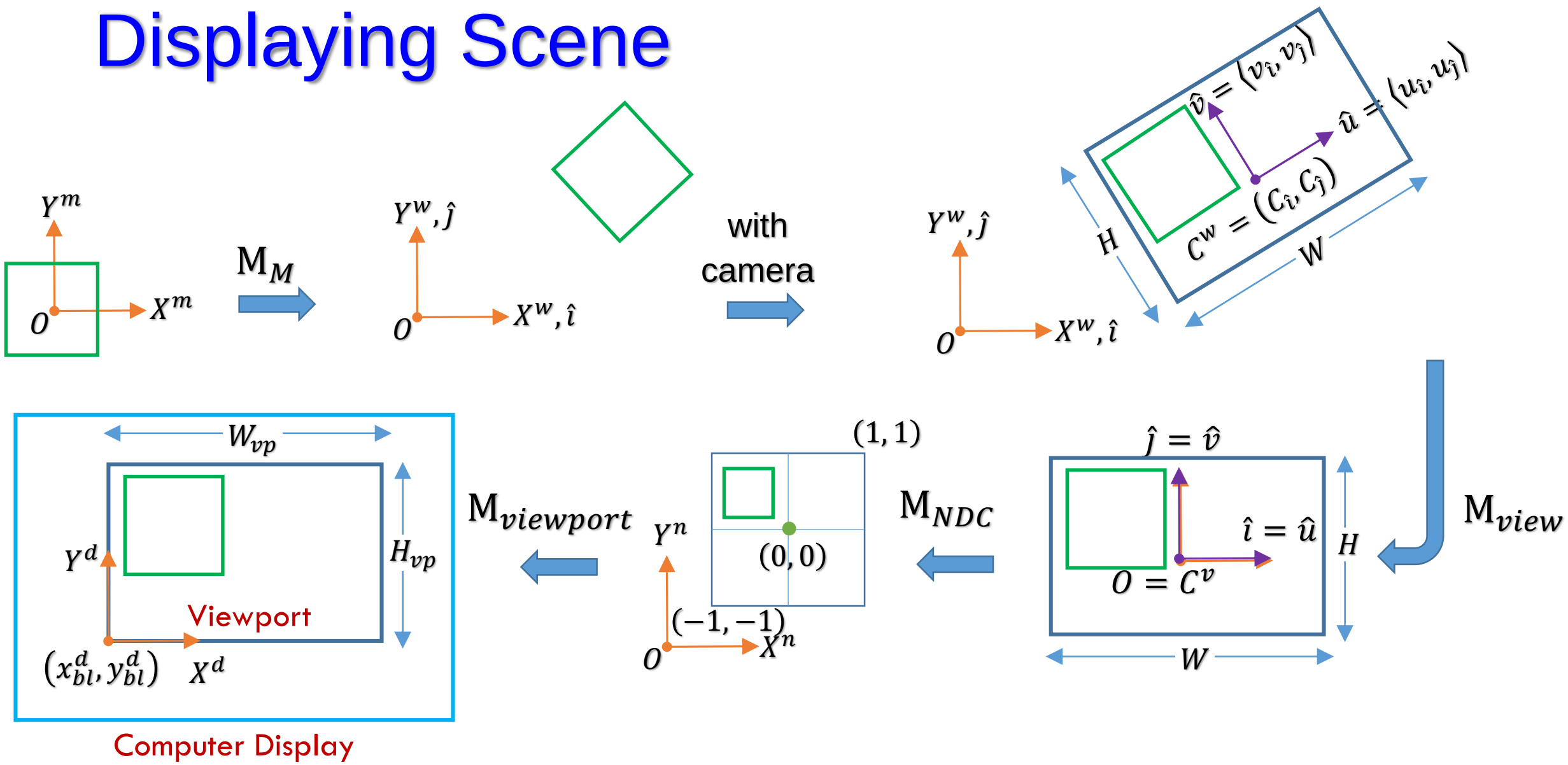
Viewing an Object: Model-to-NDC Transform (2/3)



Viewing an Object: Model-to-NDC Transform (3/3)

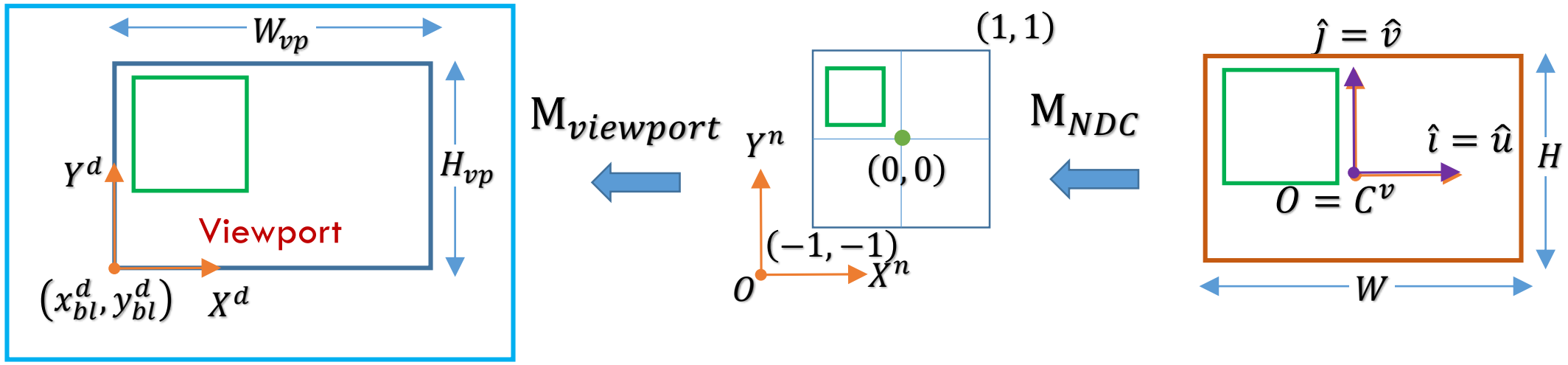
- Compute model-to-NDC transform: $\mathrm{M}_{NDC} \circ \mathrm{M}_{view} \circ \mathrm{M}_{M}$ where M_{M} is model-to-world (model) transformation
  - This matrix sent to vertex shader
  - Vertex shader will map vertices in model system directly to NDC system
- Subsequent stages will rasterizer and render the object using transformed vertices

Displaying Scene



WYSIWYG Images (1/2)

- To obtain WYSIWYG images, camera's "window to world" should've same *aspect ratio* as viewport
- If camera's window width and height are W and H, aspect ratio [of camera's window] is: $\alpha = \frac{W}{H}$



WYSIWYG Images (2/2)

- In general, we first specify viewport's dimensions from which we compute aspect ratio $\alpha_{vp} = \frac{W_{vp}}{H_{vp}}$
- We interactively control camera window dimensions by setting just its width W [or just its height H]
- Then camera window height is $H = \frac{W}{\alpha_{vp}}$ [or camera window width is $W = H\alpha_{vp}$]