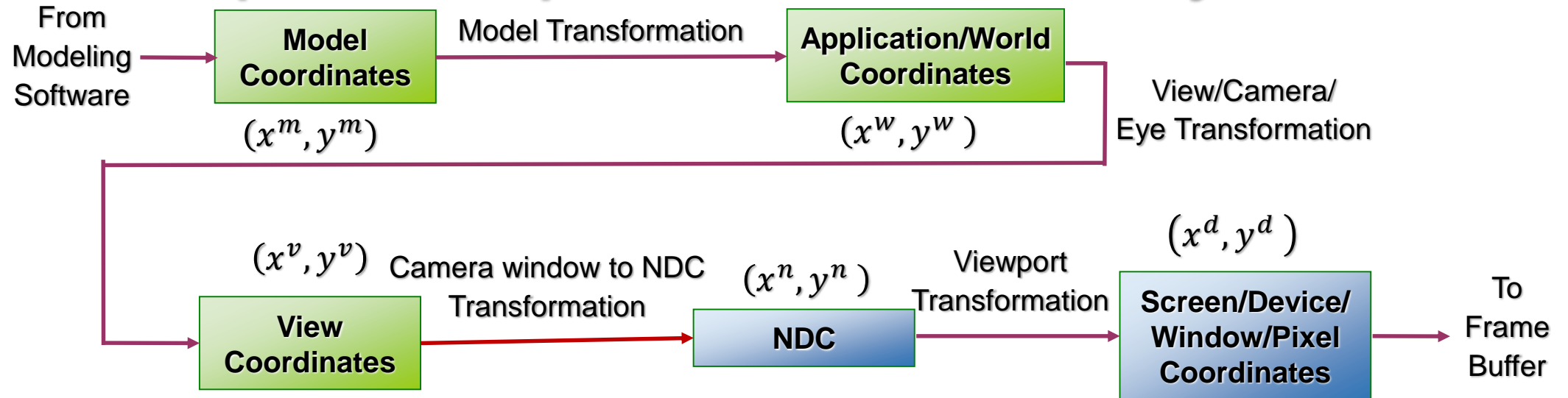


Introduction to Computer Graphics

Model and Camera Transformations

Prasanna Ghali

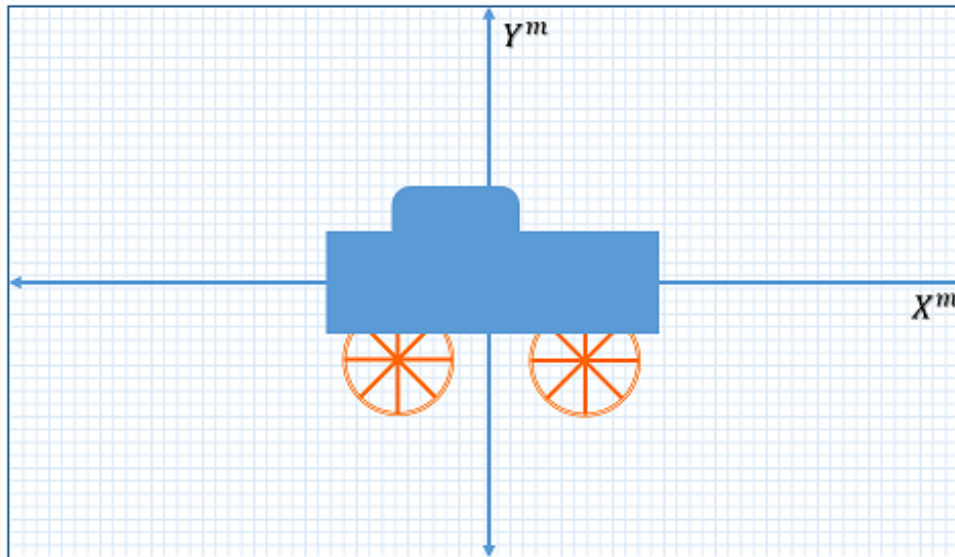
2D Graphics Pipe: Coordinate Systems



- 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

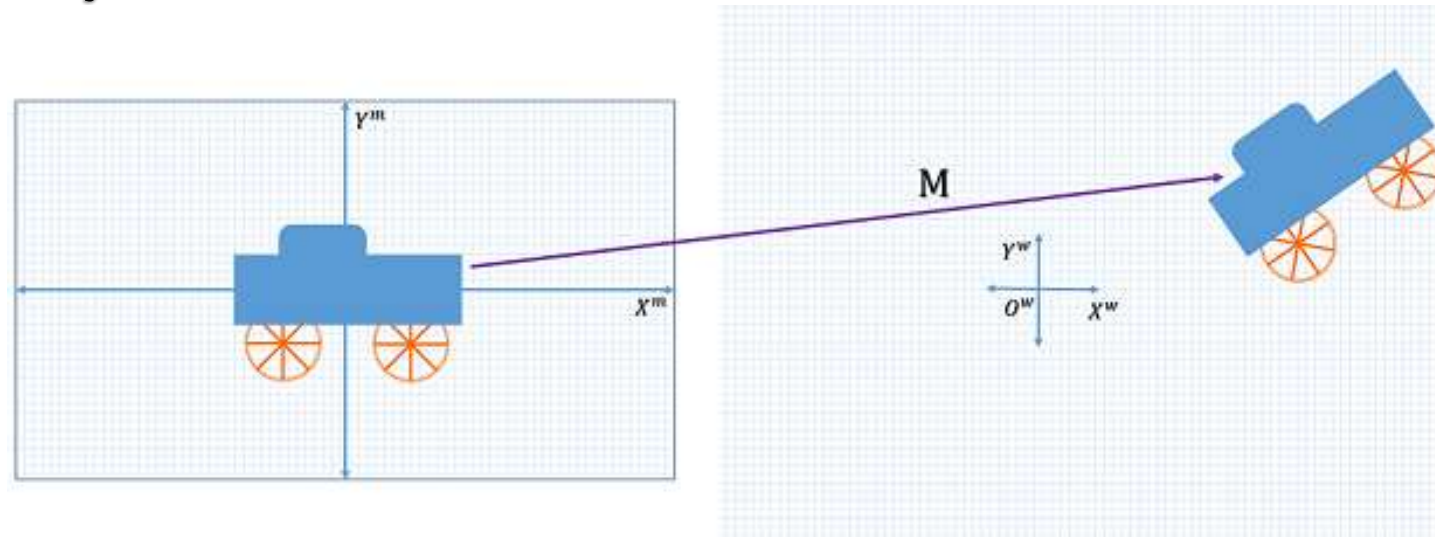
Model Transformations (1/4)

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



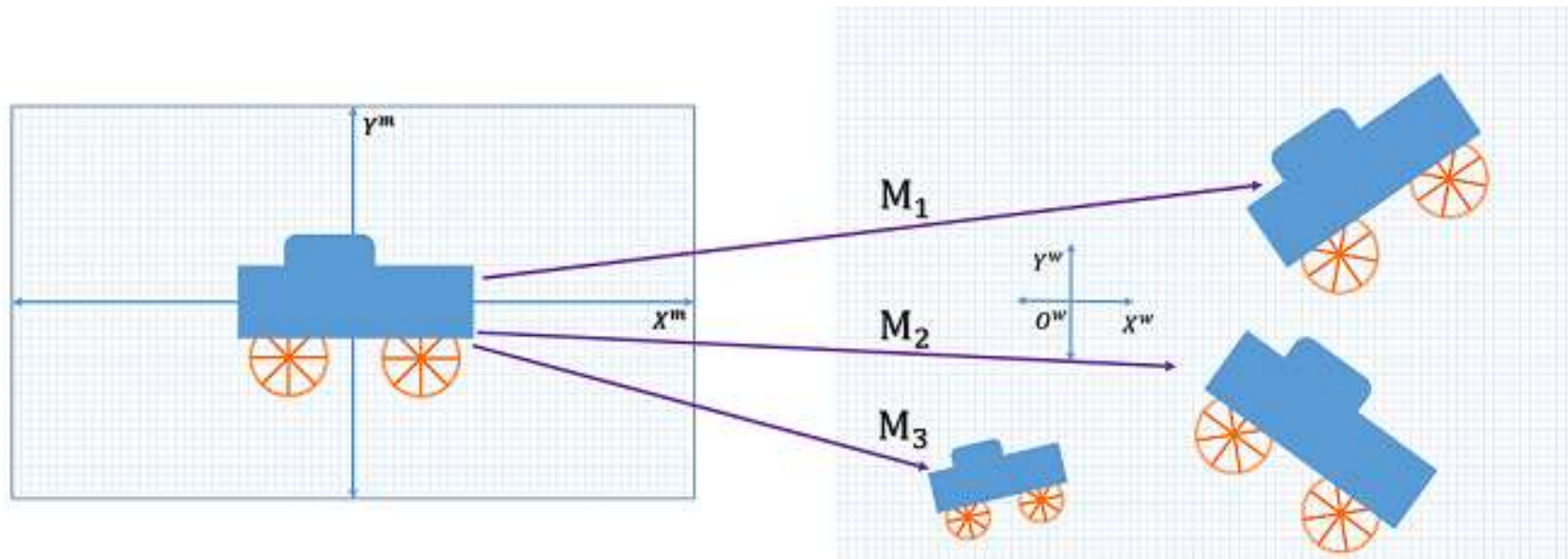
Model Transformations (2/4)

- 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*



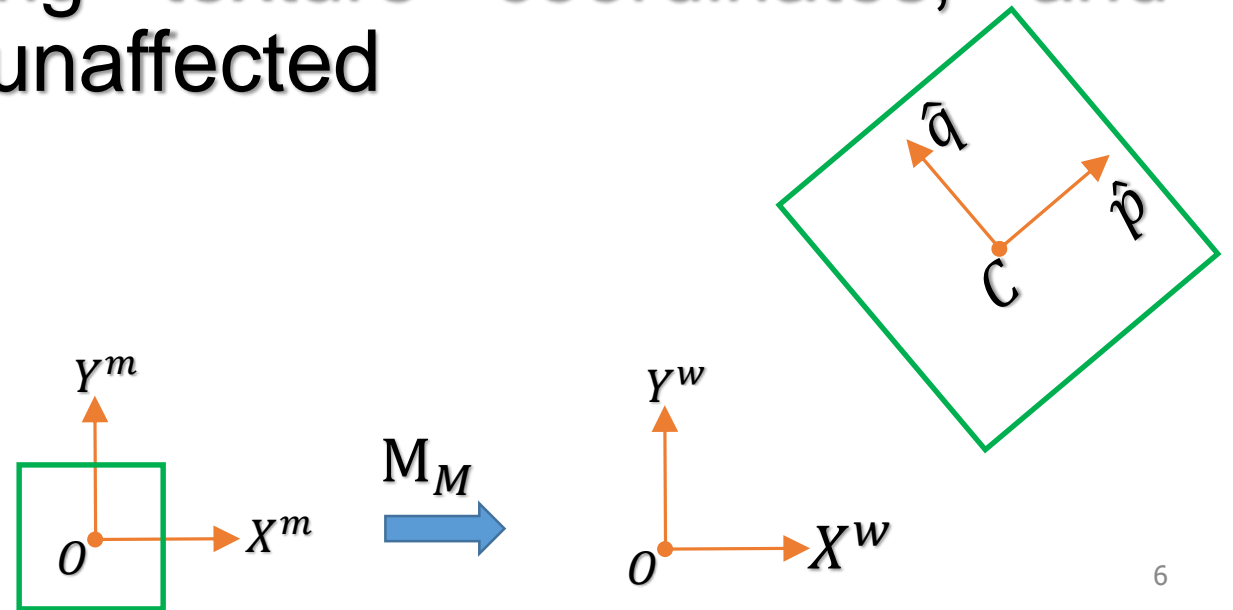
Model Transformations (3/4)

- 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

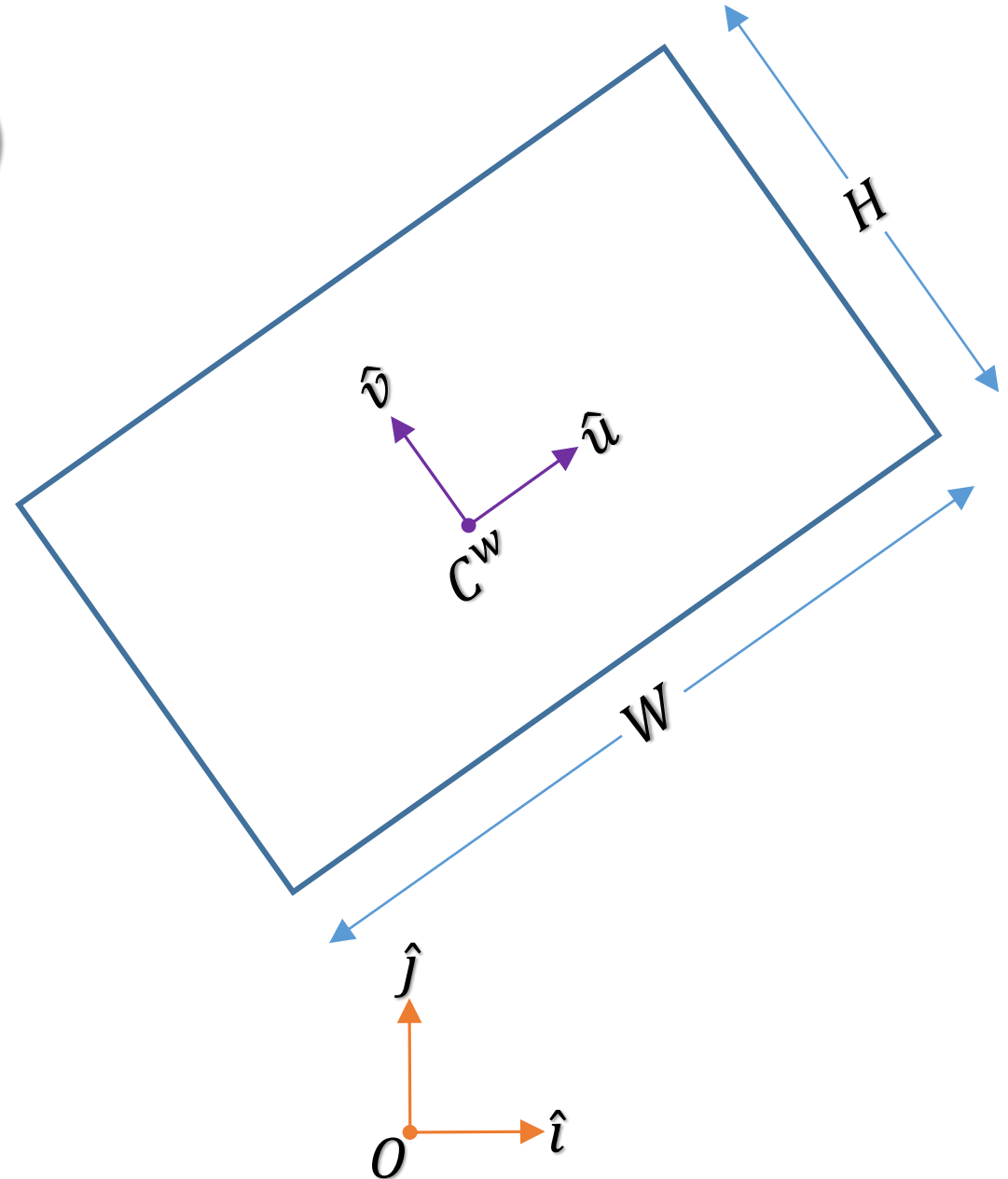
- Compute transformation matrix that maps standard square $[-1,1] \times [-1,1]$ to rectangle with width 20, height 16, rotated by 30° , and centered at $(5, 12)$

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

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

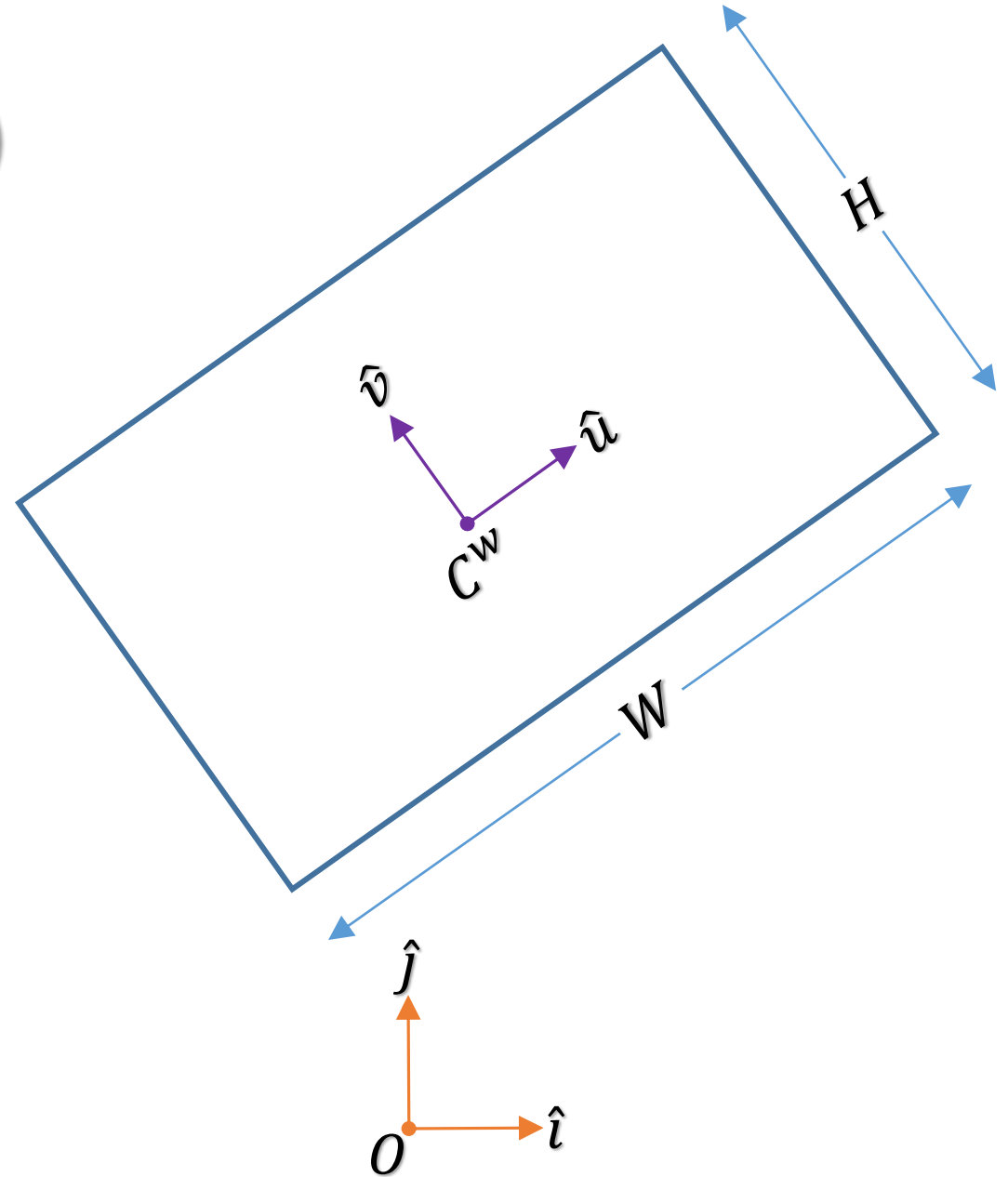
2D Camera Specs (1/2)

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



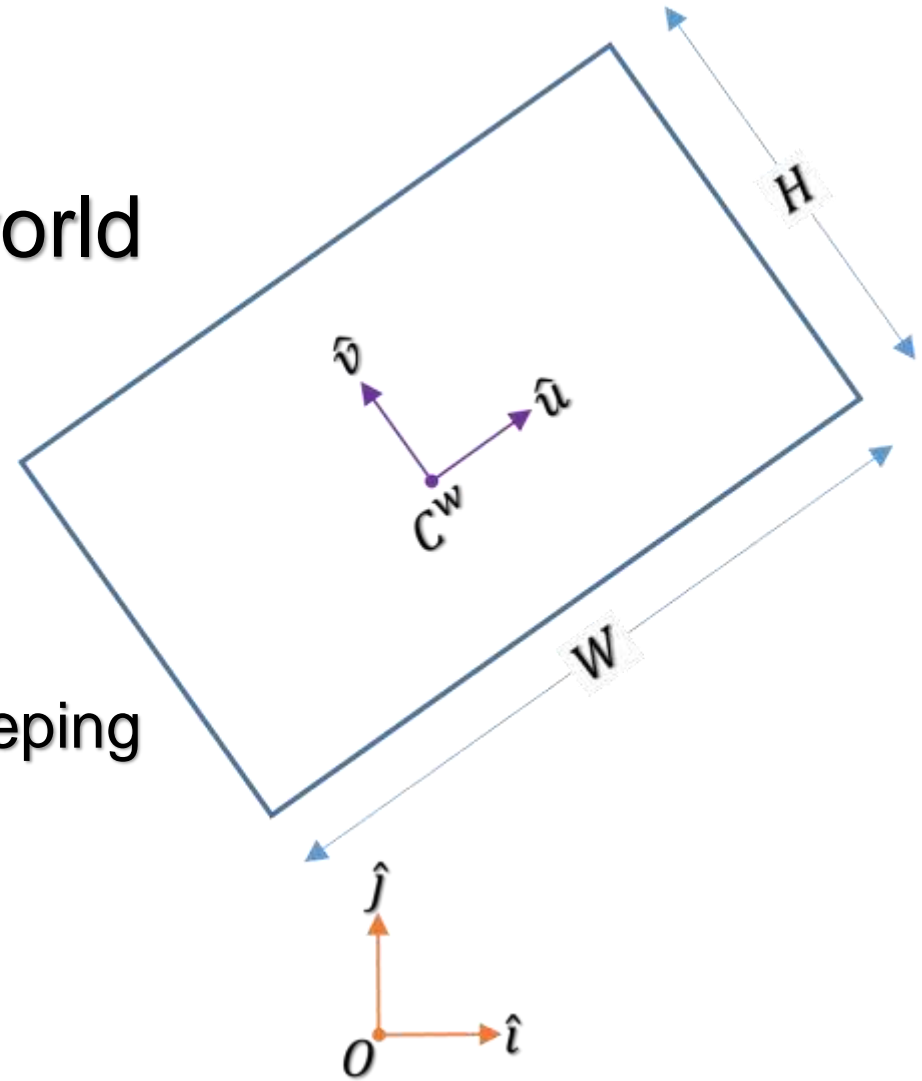
2D Camera Specs (2/2)

- 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: $\begin{vmatrix} u_{\hat{i}} & v_{\hat{i}} \\ u_{\hat{j}} & v_{\hat{j}} \end{vmatrix} > 0$



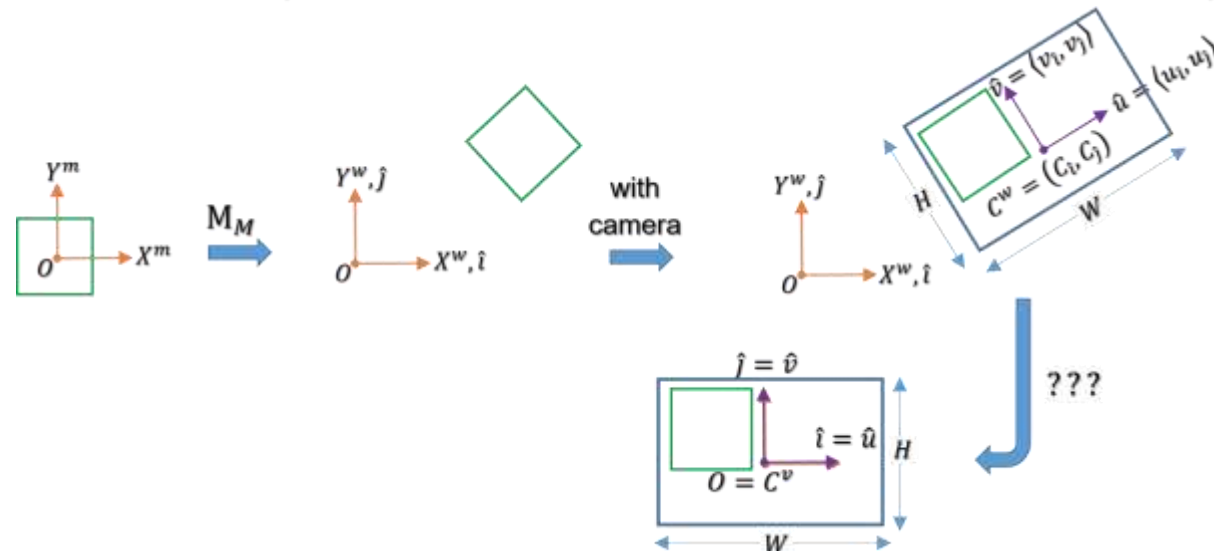
Camera Motion

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



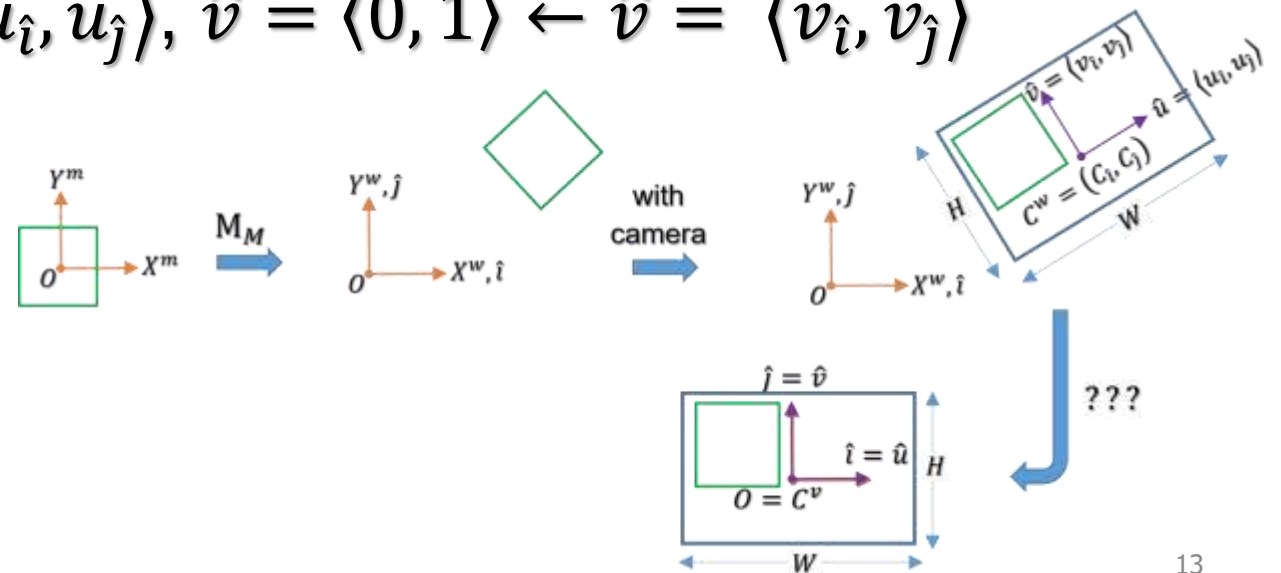
Camera (or View) Coordinate System (1/2)

- 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



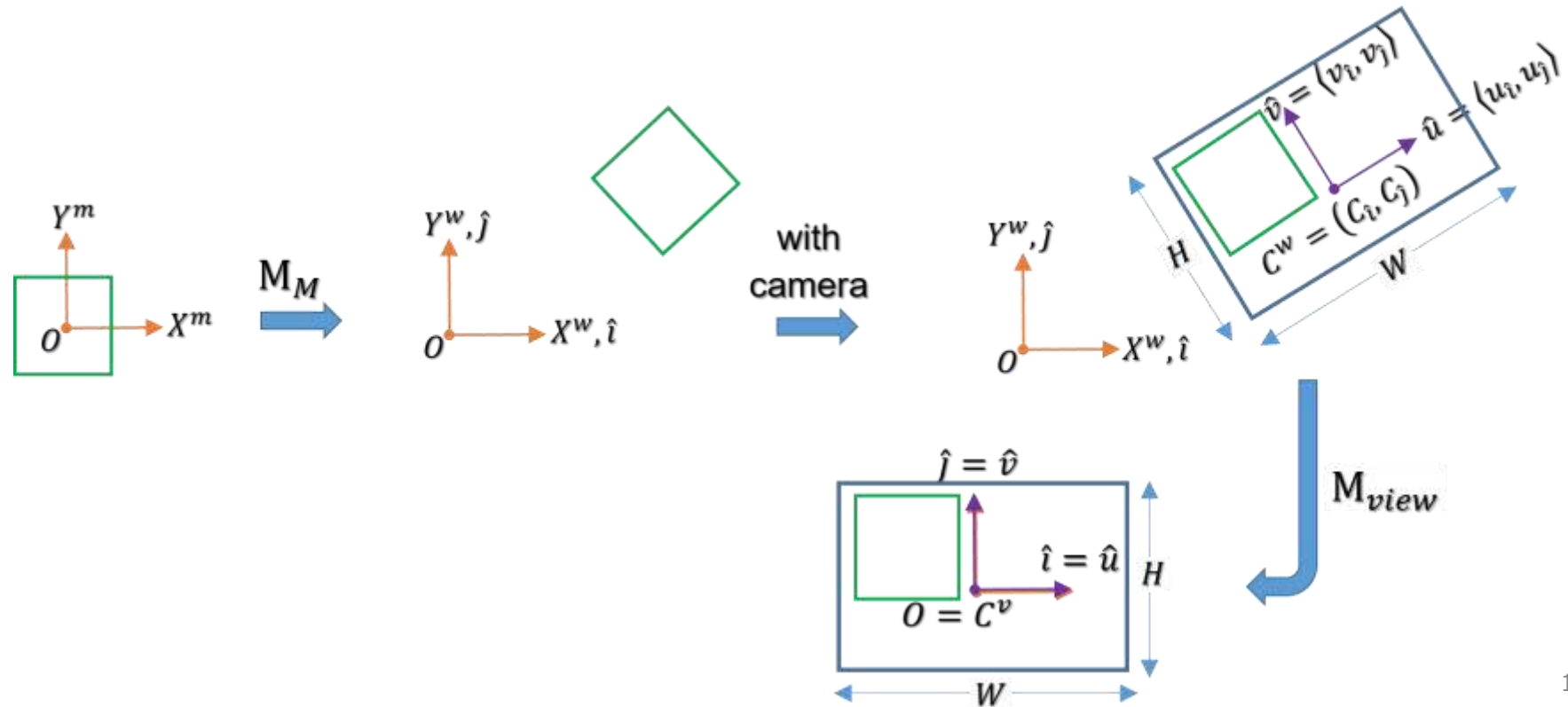
Camera (or View) Coordinate System (2/2)

- *Canonical* camera location and orientation [in camera system]:
 - Center of “window” is origin: $C^v = (0, 0) \leftarrow C^w = (C_i, C_j)$
 - Orientation vectors \hat{u} and \hat{v} transformed to view coordinate axes: $\hat{u} = \langle 1, 0 \rangle \leftarrow \hat{u} = \langle u_i, u_j \rangle$, $\hat{v} = \langle 0, 1 \rangle \leftarrow \hat{v} = \langle v_i, v_j \rangle$



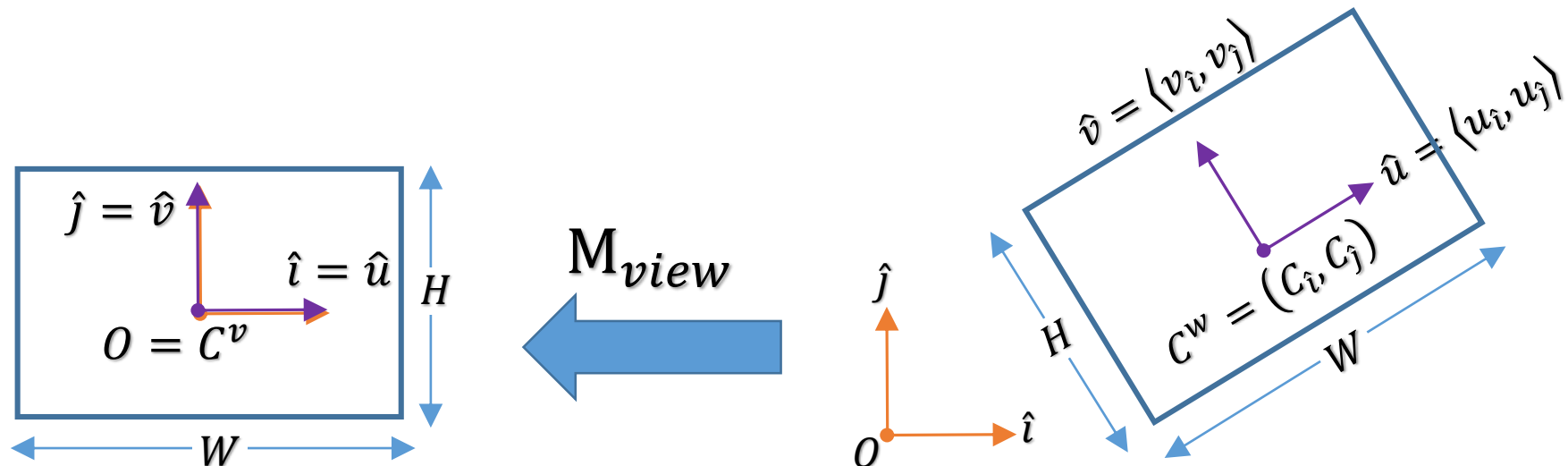
World-to-Camera (or View) Transformation (1/4)

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



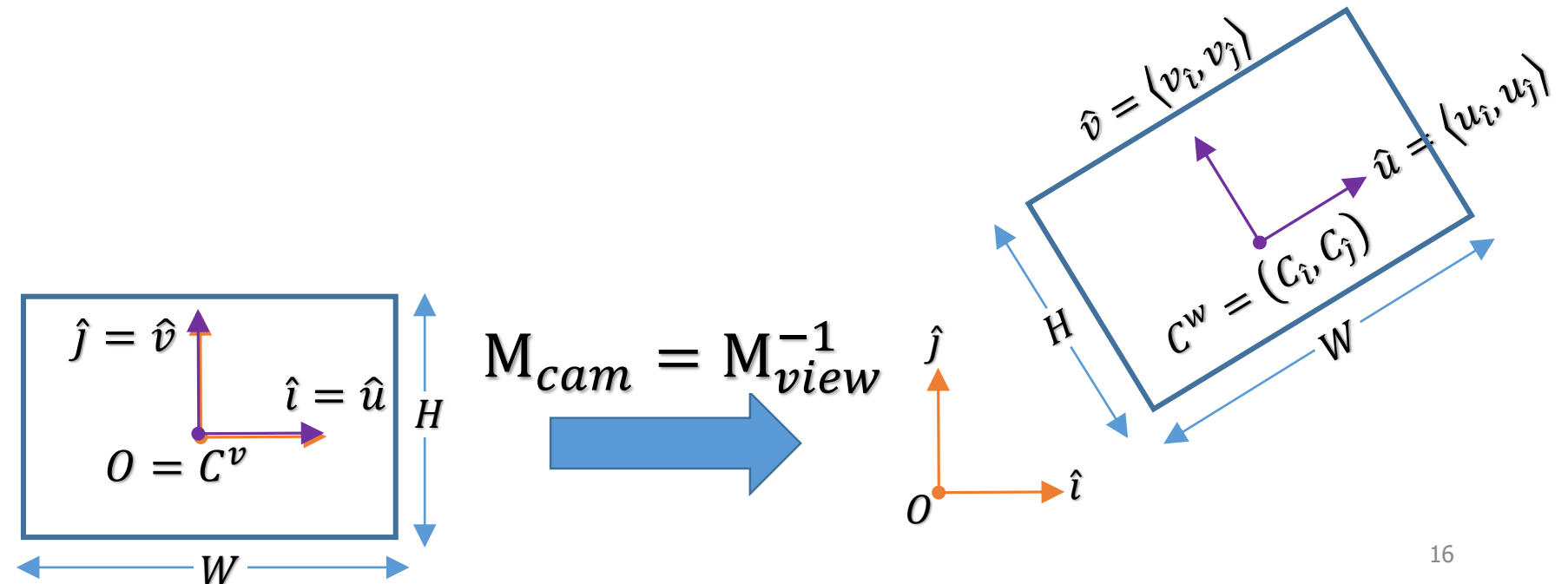
World-to-Camera (or View) Transformation (2/4)

- Transforms everything in world system to coordinate system called camera (or view) system such that
 - Camera's location $C^w = (C_i, C_j)$ in world system is now origin $C^v = (0, 0)$
 - Camera's orientation vectors $\hat{u} = \langle u_i, u_j \rangle$ and $\hat{v} = \langle v_i, v_j \rangle$ 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)

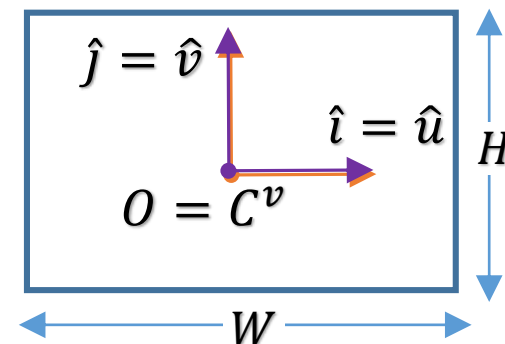
- Simpler to deduce matrix to transform axis-aligned camera window from camera system to oriented camera window in world system



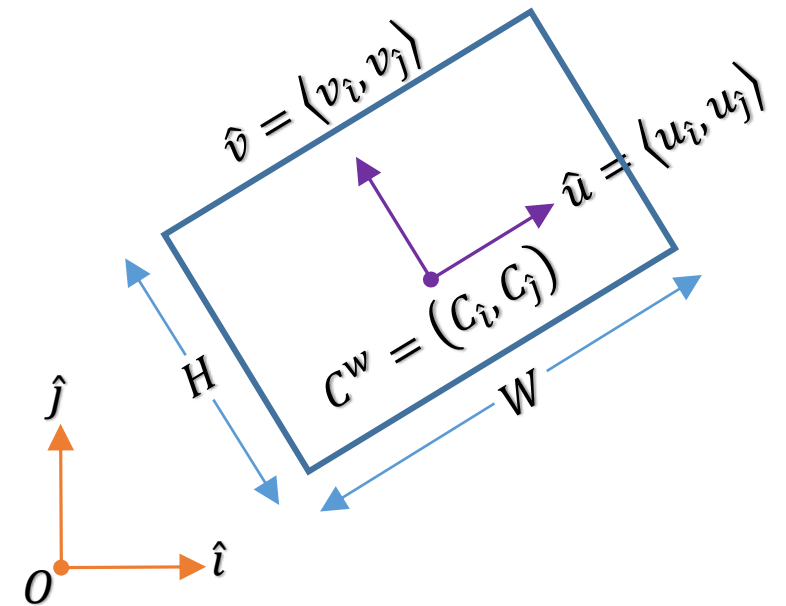
Camera-to-World Transformation

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

$$M_{cam} = M_{view}^{-1} = \begin{bmatrix} u_{\hat{i}} & v_{\hat{i}} & C_{\hat{i}} \\ u_{\hat{j}} & v_{\hat{j}} & C_{\hat{j}} \\ 0 & 0 & 1 \end{bmatrix}$$



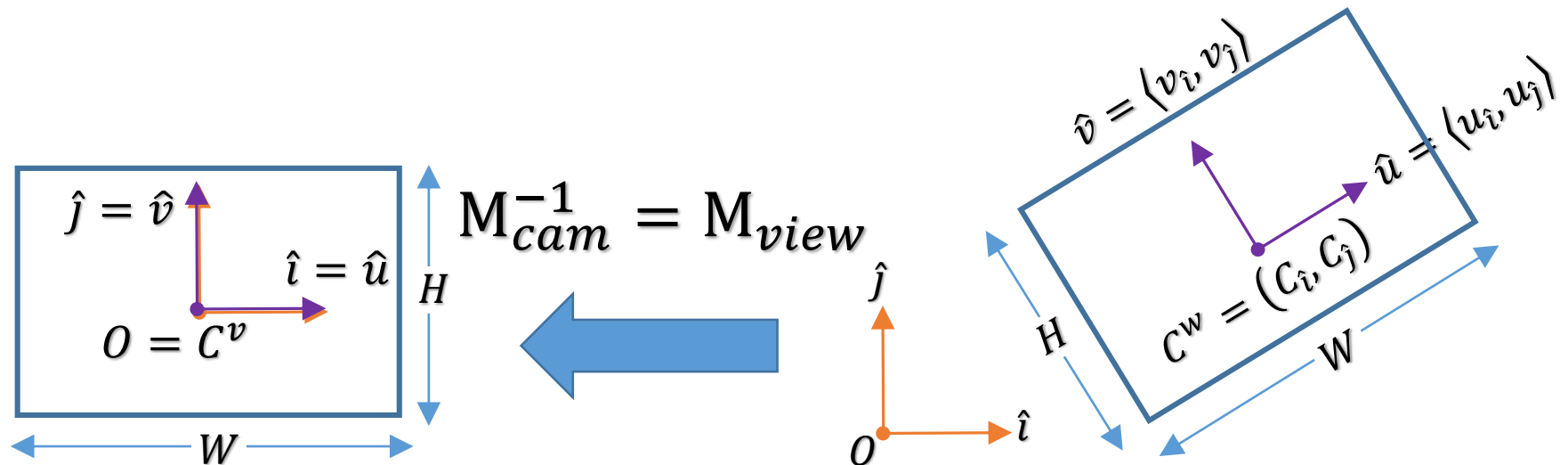
$$M_{cam} = M_{view}^{-1}$$



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

$$M_{cam} = M_{view}^{-1} = \begin{bmatrix} u_{\hat{i}} & v_{\hat{i}} & C_{\hat{i}} \\ u_{\hat{j}} & v_{\hat{j}} & C_{\hat{j}} \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_{view} = M_{cam}^{-1} = \begin{bmatrix} u_{\hat{i}} & v_{\hat{i}} & C_{\hat{i}} \\ u_{\hat{j}} & v_{\hat{j}} & C_{\hat{j}} \\ 0 & 0 & 1 \end{bmatrix}^{-1} = \begin{bmatrix} u_{\hat{i}} & u_{\hat{j}} & -\hat{u} \cdot C \\ v_{\hat{i}} & v_{\hat{j}} & -\hat{v} \cdot C \\ 0 & 0 & 1 \end{bmatrix}$$



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 $(A \circ B)^{-1} = B^{-1} \circ A^{-1}$

Inverse of Affine Transformation (2/2)

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

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

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

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

Inverse of Affine Transformation: Example

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

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

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

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

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

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)

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

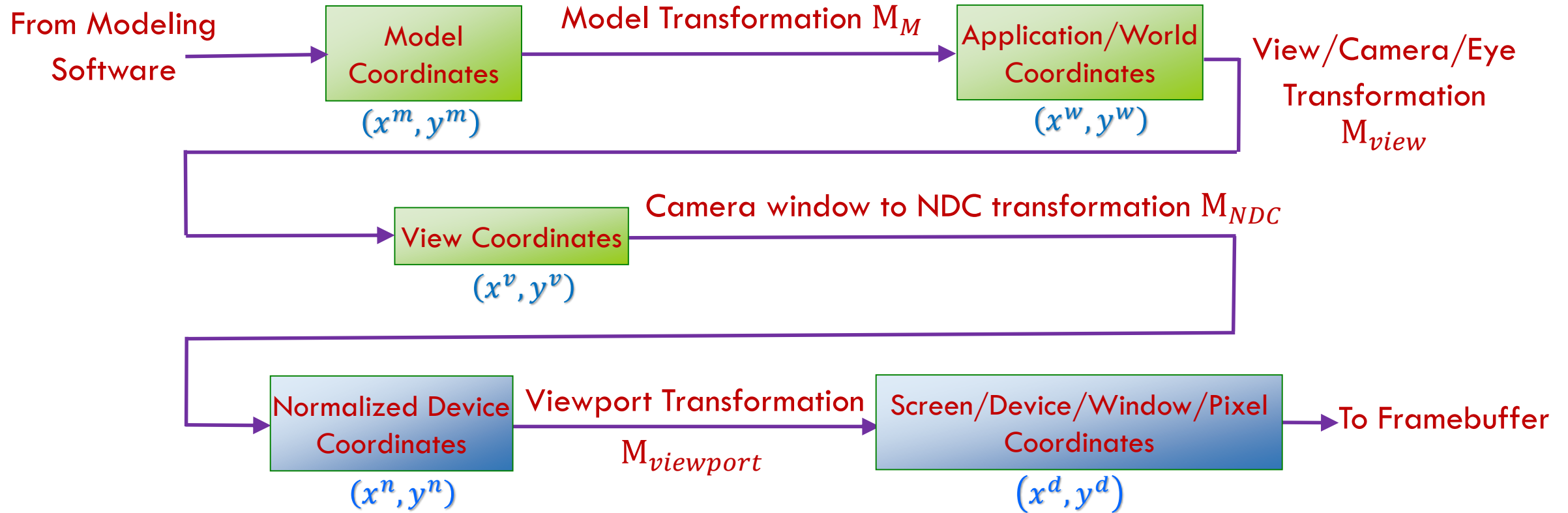
$$M_{cam} = M_{view}^{-1} = [\hat{u} \quad \hat{v} \quad C] = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & -3 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

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

$$\begin{aligned}[P_0^v \quad P_1^v \quad P_2^v] &= M_{view}[P_0^w \quad P_1^w \quad P_2^w] \\&= \begin{bmatrix} 0 & -1 & -3 \\ 1 & 0 & -2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 2 & 3 \\ -2 & -4 & -3 \\ 1 & 1 & 1 \end{bmatrix} \\&= \begin{bmatrix} -1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix}\end{aligned}$$

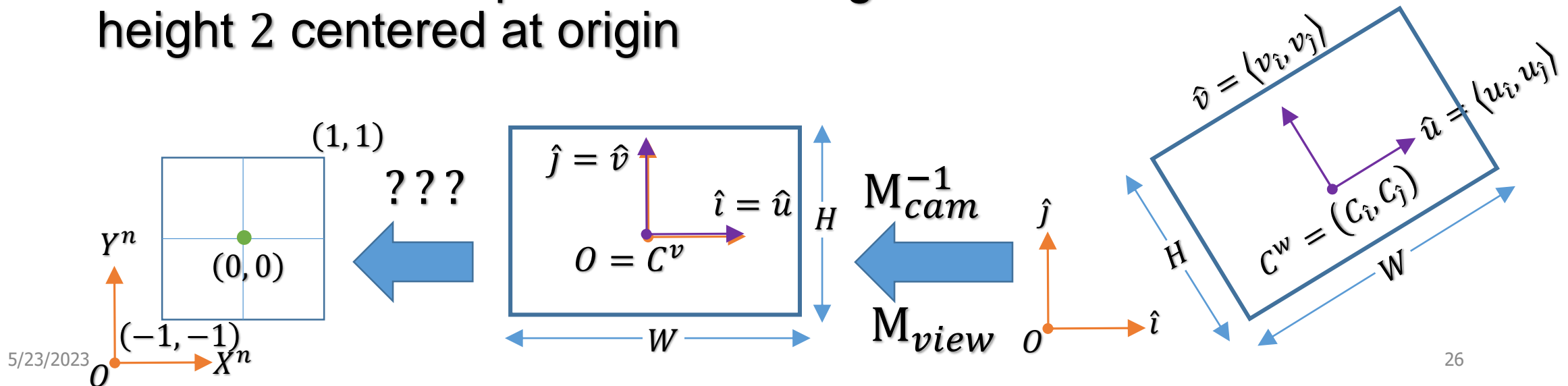
Camera-to-NDC Transform (1/3)



- 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?

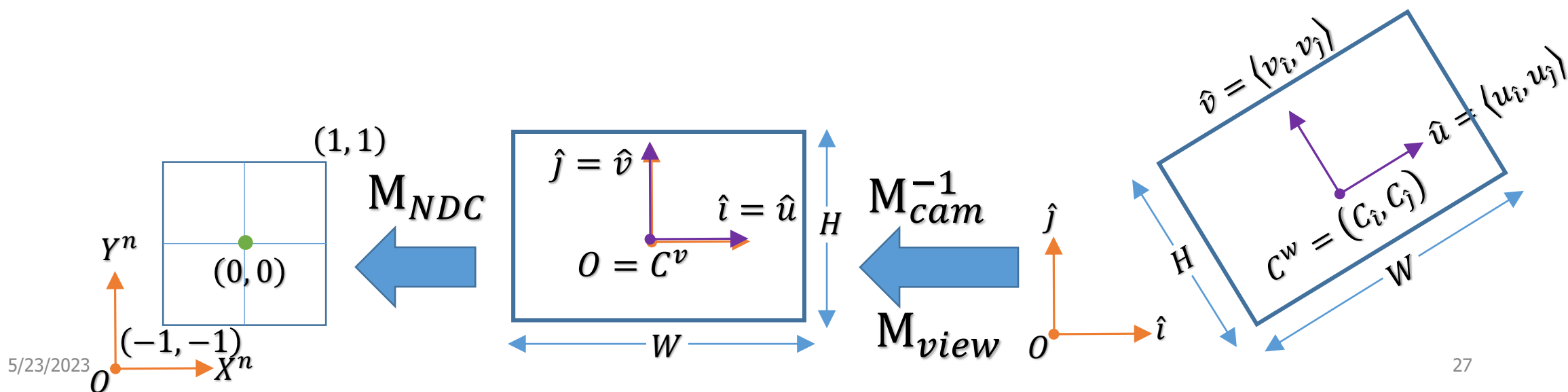
Camera-to-NDC Transform (2/3)

- 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)

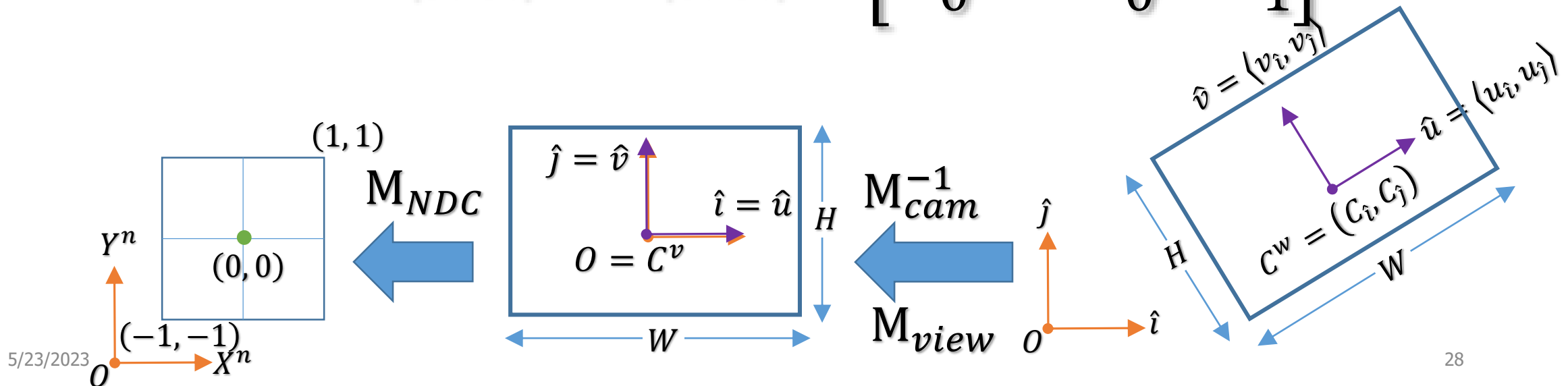
$$\mathbf{M}_{NDC} = \mathbf{H}_{\langle \frac{2}{W}, \frac{2}{H} \rangle} = \begin{bmatrix} 2/W & 0 & 0 \\ 0 & 2/H & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Camera-to-NDC Transform: Example

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

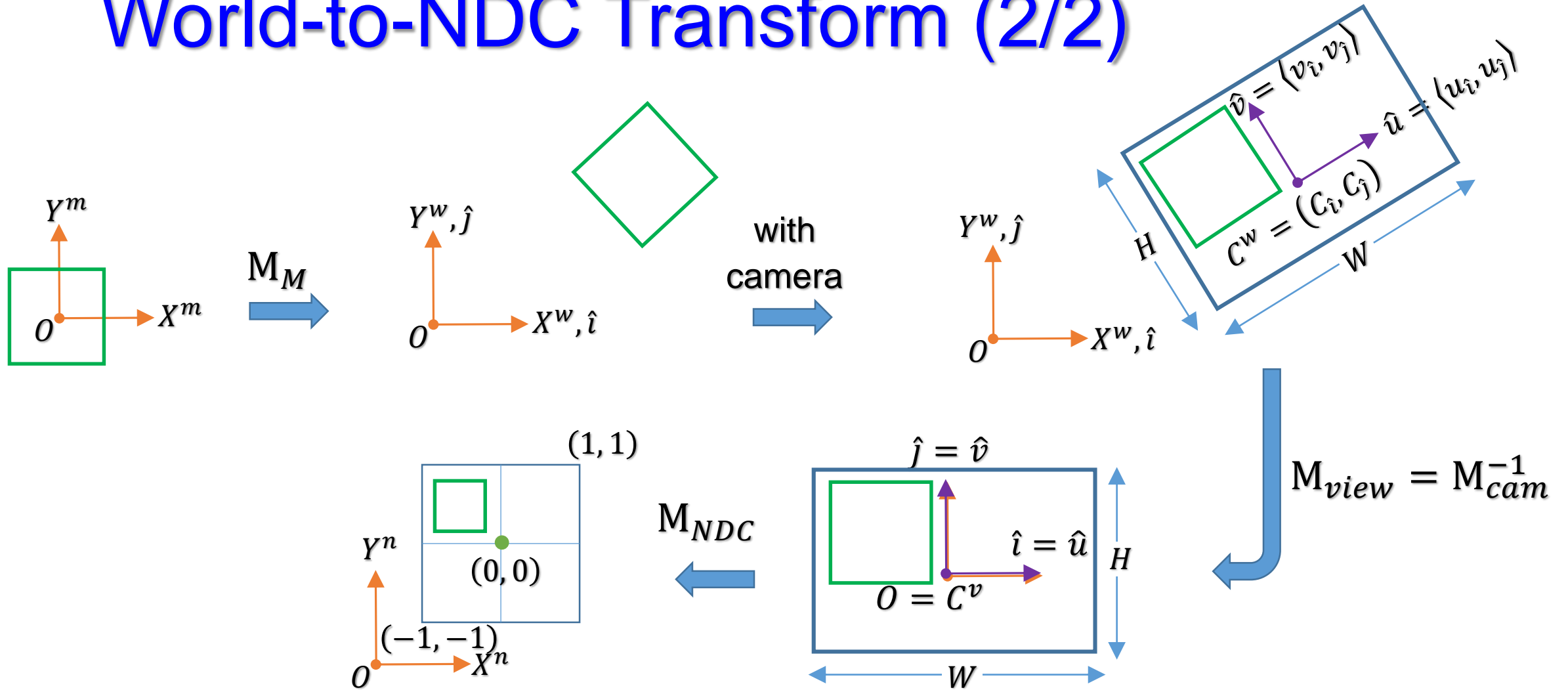
$$M_{NDC} = H \left\langle \frac{2}{80}, \frac{2}{60} \right\rangle = H \left\langle \frac{1}{40}, \frac{1}{30} \right\rangle = \begin{bmatrix} 1/40 & 0 & 0 \\ 0 & 1/30 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



World-to-NDC Transform (1/2)

- Combine world-to-camera and camera-to-NDC transforms: $M_{NDC} \circ M_{view}$
- 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 $\hat{v} = (4/5, 3/5)$
 - 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
 - $\hat{u} = R_{\theta^\circ} \hat{v} \Rightarrow \begin{bmatrix} u_{\hat{i}} \\ u_{\hat{j}} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \begin{bmatrix} v_{\hat{i}} \\ v_{\hat{j}} \end{bmatrix} \Rightarrow \begin{bmatrix} u_{\hat{i}} \\ u_{\hat{j}} \end{bmatrix} = \begin{bmatrix} v_{\hat{j}} \\ -v_{\hat{i}} \end{bmatrix}$

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

- Next, compute camera-to-world transformation

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

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

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

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

- Compute camera-to-NDC transformation

$$M_{NDC} = H_{\langle \frac{2}{10}, \frac{2}{12} \rangle} = \begin{bmatrix} 1/5 & 0 & 0 \\ 0 & 1/6 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

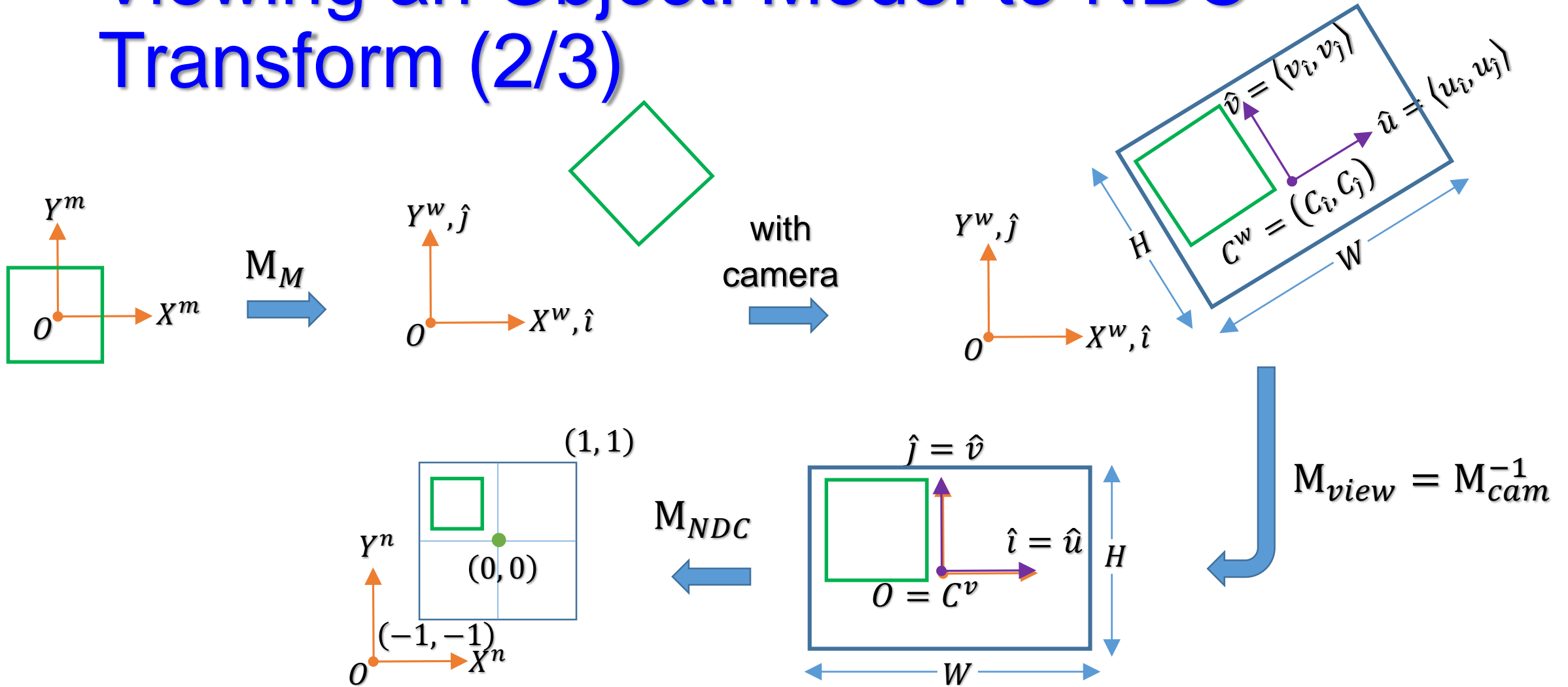
- Finally, we're ready to compute world-to-NDC transformation:

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

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

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

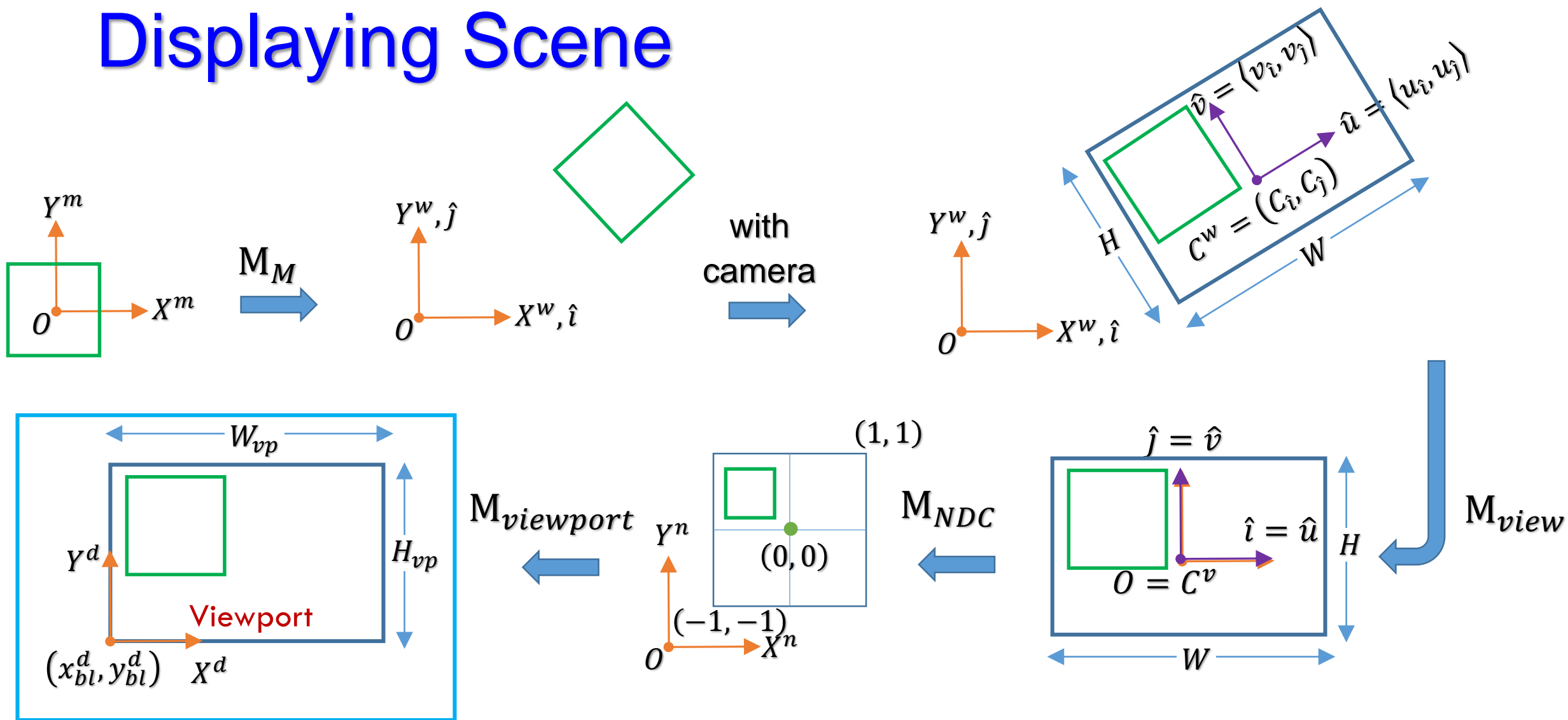
Viewing an Object: Model-to-NDC Transform (2/3)



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

- Compute model-to-NDC transform: $M_{NDC} \circ M_{view} \circ M_M$ 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

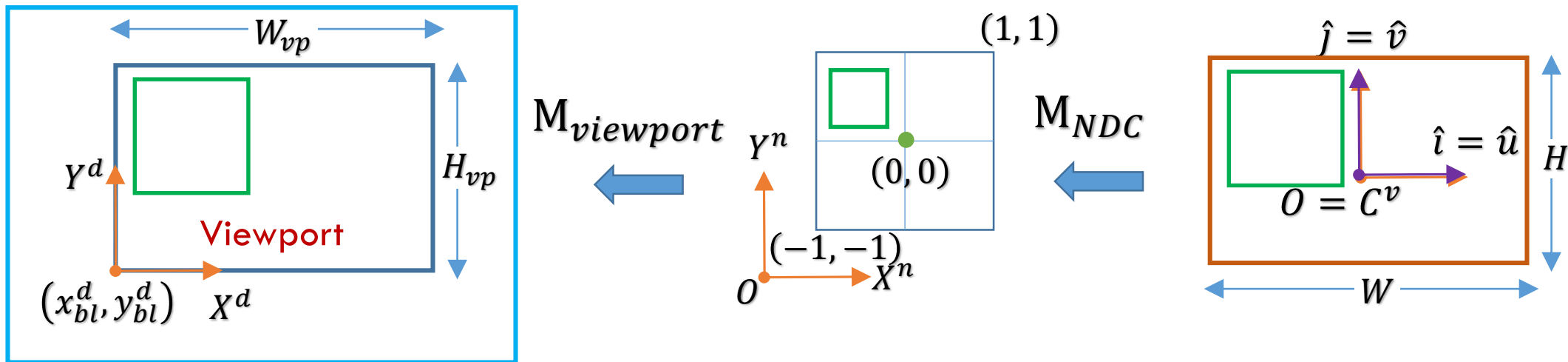
Displaying Scene



Computer Display

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 $W = H\alpha_{vp}$]