Computer Graphics

P. Healy

CS1-08
Computer Science Bldg.
tel: 202727
patrick.healy@ul.ie

Spring 2021–2022

Outline

Outline

1. Two-Dimensional Viewing
   - Two-Dimensional Viewing Pipeline

Coordinate Systems

Several coordinate systems used in graphics systems.

- **World** coordinates are the to be modelled; we select a “window” of this world for viewing
- **Viewing** coordinates are how we wish to present the view on the ; we can achieve by mapping different-sized clipping windows (via `glOrtho2D()`)to a fixed-size **viewport**; can have multiple viewports on screen simultaneously

Coordinate Systems

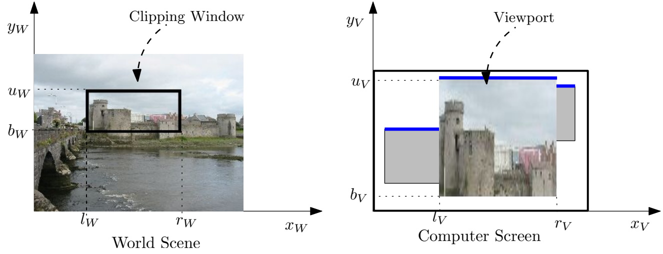
Several coordinate systems used in graphics systems.

- **World** coordinates are the [ ] to be modelled; we select a "window" of this world for viewing
- **Viewing** coordinates are how we wish to present the view on the [ ]; we can achieve [ ] by mapping different-sized clipping windows (via `glOrtho2D()`)to a fixed-size **viewport**; can have multiple viewports on screen simultaneously *viewport*

Coordinate Systems

Several coordinate systems used in graphics systems.

- **World** coordinates are the to be modelled; we select a “window” of this world for viewing
- **Viewing** coordinates are how we wish to present the view on the ; we can achieve by mapping different-sized clipping windows (via `glOrtho2D()`)to a fixed-size **viewport**; can have multiple viewports on screen simultaneously viewport



Coordinate Systems

Several coordinate systems used in graphics systems.

- **World** coordinates are the to be modelled; we select a “window” of this world for viewing
- **Viewing** coordinates are how we wish to present the view on the ; we can achieve by mapping different-sized clipping windows (via `glOrtho2D()`)to a fixed-size **viewport**; can have multiple viewports on screen simultaneously viewport

Must ensure that we can handle camera rotations, too.

Coordinate Systems (contd.)

- The selects **what**; the indicates **where** on output device and what size
- **Normalized** coordinates are introduced to (for example) make be in range of 0-1; clipping is usually and **more efficiently** done in these coordinates
- **Device** coordinates are specific to output device: printer page, screen display, etc. but **Normalized Device** co-ordinates are independent

The Viewing Pipeline.

Coordinate Systems (contd.)

- The selects **what**; the indicates **where** on output device and what size
- **Normalized** coordinates are introduced to (for example) make be in range of 0-1; clipping is usually and **more efficiently** done in these coordinates
- **Device** coordinates are specific to output device: printer page, screen display, etc. but **Normalized Device** co-ordinates are independent

The Viewing Pipeline.

Coordinate Systems (contd.)

- The $\boxed{}$ selects **what**; the $\boxed{}$ indicates **where** on output device and what size
- **Normalized** coordinates are introduced to (for example) make $\boxed{}$ be in range of 0-1; clipping is usually and **more efficiently** done in these coordinates
- **Device** coordinates are specific to output device: printer page, screen display, etc. but **Normalized Device** co-ordinates are independent

The Viewing Pipeline.

Coordinate Systems (contd.)

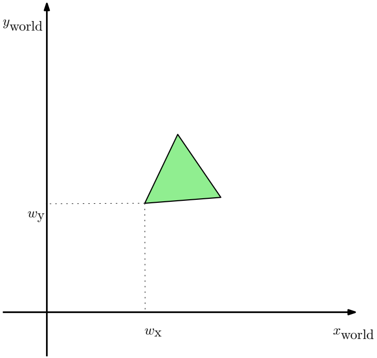
- The selects **what**; the indicates **where** on output device and what size
- **Normalized** coordinates are introduced to (for example) make be in range of 0-1; clipping is usually and **more efficiently** done in these coordinates
- **Device** coordinates are specific to output device: printer page, screen display, etc. but **Normalized Device** co-ordinates are independent

Construct World–Coordinate Scene → Convert World Coords. to Viewing Coords. → Viewing Coords. are Normalized → Map Normalized Coords. to Device Coords. →

The Viewing Pipeline.

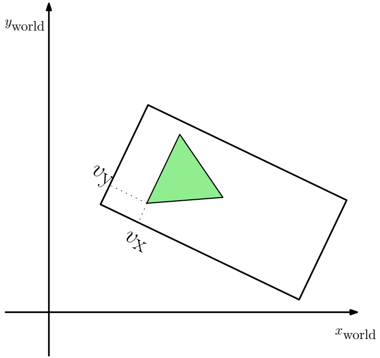
Viewing-Coordinate Clipping Window

- Given a scene in coordinates, how do we transform a point $w = (w_x, w_y)$ to coordinates $v = (v_x, v_y)$?



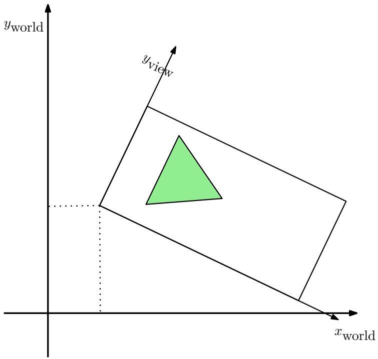
Viewing-Coordinate Clipping Window

- Given a scene in $\boxed{}$ coordinates, how do we transform a point $w = (w_x, w_y)$ to $\boxed{}$ coordinates $v = (v_x, v_y)$?



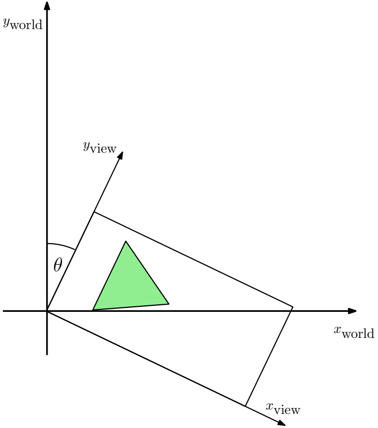
Viewing-Coordinate Clipping Window

- Given a scene in $\boxed{}$ coordinates, how do we transform a point $w = (w_x, w_y)$ to $\boxed{}$ coordinates $v = (v_x, v_y)$?



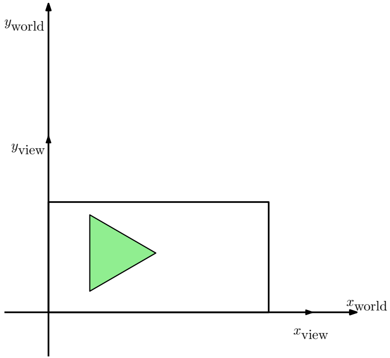
Viewing-Coordinate Clipping Window

- Given a scene in $\boxed{}$ coordinates, how do we transform a point $w = (w_x, w_y)$ to $\boxed{}$ coordinates $v = (v_x, v_y)$?



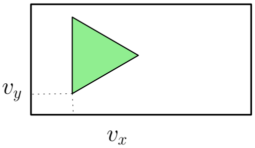
Viewing-Coordinate Clipping Window

- Given a scene in [ ] coordinates, how do we transform a point $w = (w_x, w_y)$ to [ ] coordinates $v = (v_x, v_y)$?



Viewing-Coordinate Clipping Window

- Given a scene in coordinates, how do we transform a point $w = (w_x, w_y)$ to coordinates $v = (v_x, v_y)$?



Viewing-Coordinate Clipping Window (contd.)

- If the view coordinate system is located at (a, b) we firstly align it with the world origin by **translating** by (a, b)
- The y_{view} axis is commonly known as **view-up vector**, V
- The angle, θ, that V makes with the y-world axis is what w must be rotated by to transform it in to viewing coordinates
- Then the **composite** transformation $M_{W,V}$ to make this transformation from w (world) to v (viewport) is the product (multiplication)

$$v = M_{W,V} w$$

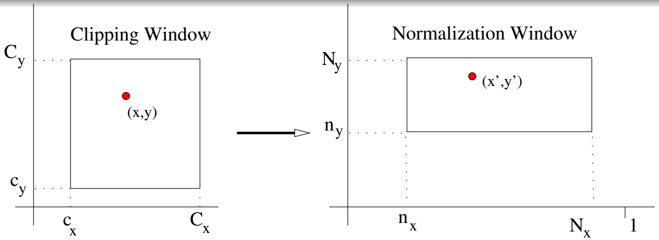
where

$$\begin{aligned} M_{W,V} &= R \times T \quad \text{Rotation } \textit{after} \text{ Translation} \\ &= \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -a \\ 0 & 1 & -b \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Normalization and Viewport Transformations

- In some graphics systems normalization and window-to-viewport transformations are combined in to one operation
- In this case axes of viewport coord. system are often given in range of $[0, 1]$
- After clipping, the square is mapped to the output display
- As an alternative, in other systems normalization and clipping are applied before viewport transformations
- In these systems, the viewport boundaries are specified in screen coords relative to the display-window position
- What both systems have in common is a need to map a point (x, y) in one rectangle to (x', y') in another

Normalization and Viewport Transformations (contd.)



- What is the transformation that maps (x, y) within the clipping window to (x', y') in another window?
- In order to maintain the same **relative** position look at ratios:

$$\frac{x' - n_x}{N_x - n_x} = \frac{x - c_x}{C_x - c_x}$$

$$\frac{y' - n_y}{N_y - n_y} = \frac{y - c_y}{C_y - c_y}$$

Normalization and Viewport Transformations (contd.)

Then

$$x' = \frac{N_x - n_x}{C_x - c_x}x + \frac{C_x n_x - c_x N_x}{C_x - c_x}$$

And after solving for y' analogously, we can write (x', y') as

$$x' = s_x x + t_x$$
$$y' = s_y y + t_y$$

- The scalings, s_x and s_y, are

$$s_x = \frac{N_x - n_x}{C_x - c_x}, \qquad s_y = \frac{N_y - n_y}{C_y - c_y}$$

- The translation factors are

$$t_x = \frac{C_x n_x - c_x N_x}{C_x - c_x} \qquad t_y = \frac{C_y n_y - c_y N_y}{C_y - c_y}$$

Normalization and Viewport Transformations (contd.)

- So we get a composite transformation, $M_{C,N}$ that combines a translation and a scaling

$$\begin{aligned} M_{C,N} &= T \cdot S \\ &= \begin{pmatrix} s_x & 0 & t_x \\ 0 & s_y & t_y \\ 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

Normalization and Viewport Transformations (contd.)

- This expression holds for **any** mapping from one rectangular coordinate system to another
- Whether the clipping window to viewport transformation is done in one step or in two, the same procedure for determining the transformation matrix can be used
- For example, if we map from the clipping window to a normalized square $[-1, 1] \times [-1, 1]$, (then clip in this square and then convert to screen coords) the **first** transformation matrix of this pipeline is

$$M_{C,NS} = \begin{pmatrix} \frac{2}{C_x - c_x} & 0 & -\frac{C_x + c_x}{C_x - c_x} \\ 0 & \frac{2}{C_y - c_y} & -\frac{C_y + c_y}{C_y - c_y} \\ 0 & 0 & 1 \end{pmatrix}$$