
Computer Graphics

6 - Lab - Vertex Processing 2

Yoonsang Lee
Hanyang University

Spring 2026

Outline

- Orthographic Projection
 - `glm.ortho()`
- Perspective Projection
 - `glm.frustum()`
 - `glm.perspective()`
- Viewport Transformation
 - `glViewport()`

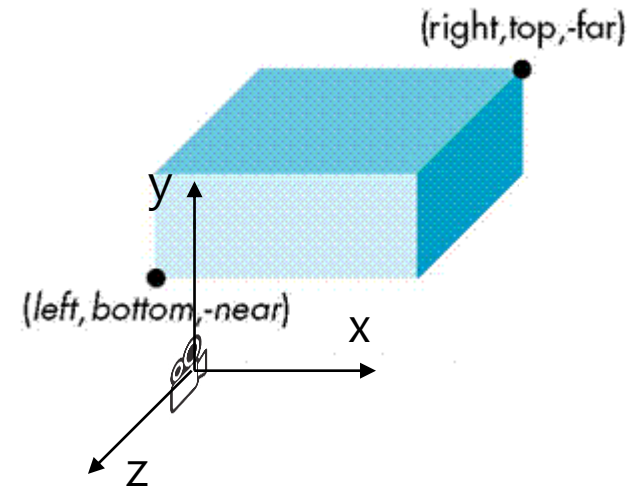
Orthographic Projection

- glm.ortho()

glm.ortho()

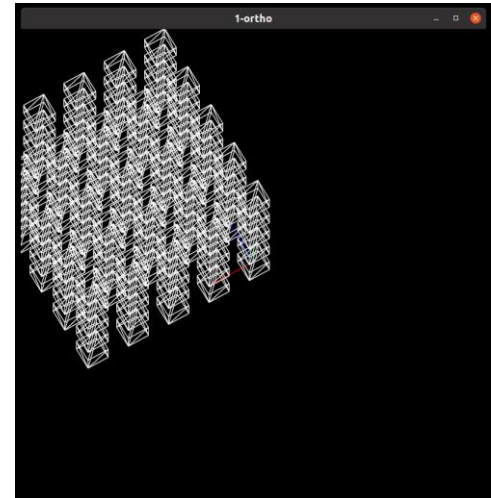
glm.ortho(left: float, right: float, bottom: float, top: float, zNear: float, zFar: float) -> fmat4x4

- : Creates an orthographic projection matrix.
- Sign of zNear, zFar:
 - positive value: the plane is in front of the camera.
 - negative value: the plane is behind the camera.



[Code] 1-ortho

- Vertex & Fragment shader: same as those in the previous example "3-lookat.py"
- Draws a cube or an array of cubes instead of a single triangle, in wireframe mode.
- Two VAOs:
 - cube VAO
 - frame VAO
- Three drawing functions:
 - draw_frame() - uses the frame VAO
 - draw_cube(), draw_cube_array() - uses the cube VAO



[Code] 1-ortho

```
def prepare_vao_cube():
    # prepare vertex data (in main memory)
    # 36 vertices for 12 triangles
    vertices = glm.array(glm.float32,
        # position                color
        -0.5 ,  0.5 ,  0.5 ,  1, 1, 1, # v0
        0.5 , -0.5 ,  0.5 ,  1, 1, 1, # v2
        0.5 ,  0.5 ,  0.5 ,  1, 1, 1, # v1

        -0.5 ,  0.5 ,  0.5 ,  1, 1, 1, # v0
        -0.5 , -0.5 ,  0.5 ,  1, 1, 1, # v3
        0.5 , -0.5 ,  0.5 ,  1, 1, 1, # v2

        ...
        -0.5 ,  0.5 ,  0.5 ,  1, 1, 1, # v0
        -0.5 ,  0.5 , -0.5 ,  1, 1, 1, # v4
        -0.5 , -0.5 , -0.5 ,  1, 1, 1, # v7
    )
    # the same vertex configuration as the previous "3-lookat.py" example
    ...

def prepare_vao_frame():
    # the same function as the previous "3-lookat.py" example
    ...
```

[Code] 1-ortho

```
def draw_frame(vao, MVP, loc_MVP):
    glBindVertexArray(vao)
    glUniformMatrix4fv(loc_MVP, 1, GL_FALSE, glm.value_ptr(MVP))
    glDrawArrays(GL_LINES, 0, 6)

def draw_cube(vao, MVP, loc_MVP):
    glBindVertexArray(vao)
    glUniformMatrix4fv(loc_MVP, 1, GL_FALSE, glm.value_ptr(MVP))
    glDrawArrays(GL_TRIANGLES, 0, 36)

def draw_cube_array(vao, MVP, loc_MVP):
    glBindVertexArray(vao)
    for i in range(5):
        for j in range(5):
            for k in range(5):
                MVP_cube = MVP *
glm.translate(glm.vec3(1*i, 1*j, 1*k)) *
glm.scale(glm.vec3(.5, .5, .5))
                glUniformMatrix4fv(loc_MVP, 1, GL_FALSE,
glm.value_ptr(MVP_cube))
                glDrawArrays(GL_TRIANGLES, 0, 36)
```

[Code] 1-ortho

```
def main():
    ...
    while not glfwWindowShouldClose(window):
        ...
        # render in "wireframe mode"
        glPolygonMode(GL_FRONT_AND_BACK, GL_LINE)
        ...
        # orthogonal projection - try changing arguments
        P = glm.ortho(-5,5, -5,5, -10,10)

        # view matrix
        V = glm.lookAt(glm.vec3(1*np.sin(g_cam_ang), g_cam_height,
1*np.cos(g_cam_ang)), glm.vec3(0,0,0), glm.vec3(0,1,0))

        # draw world frame
        draw_frame(vao_frame, P*V*glm.mat4(), MVP_loc)
        ...
        M = glm.mat4()
        # M = R

        # draw cube w.r.t. the current frame MVP
        draw_cube(vao_cube, P*V*M, MVP_loc)

        # # draw cube array w.r.t. the current frame MVP
        # draw_cube_array(vao_cube, P*V*M, MVP_loc)
        ...
```

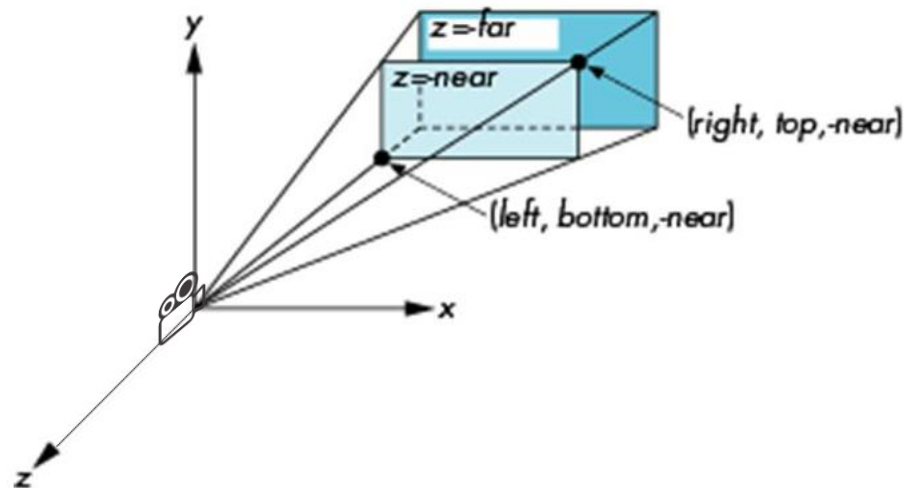
Perspective Projection

- glm.frustum()**
- glm.perspective()**

glm.frustum()

glm.frustum(**left**: float, **right**: float, **bottom**: float, **top**: float, **near**: float, **far**: float) -> **fmat4x4**

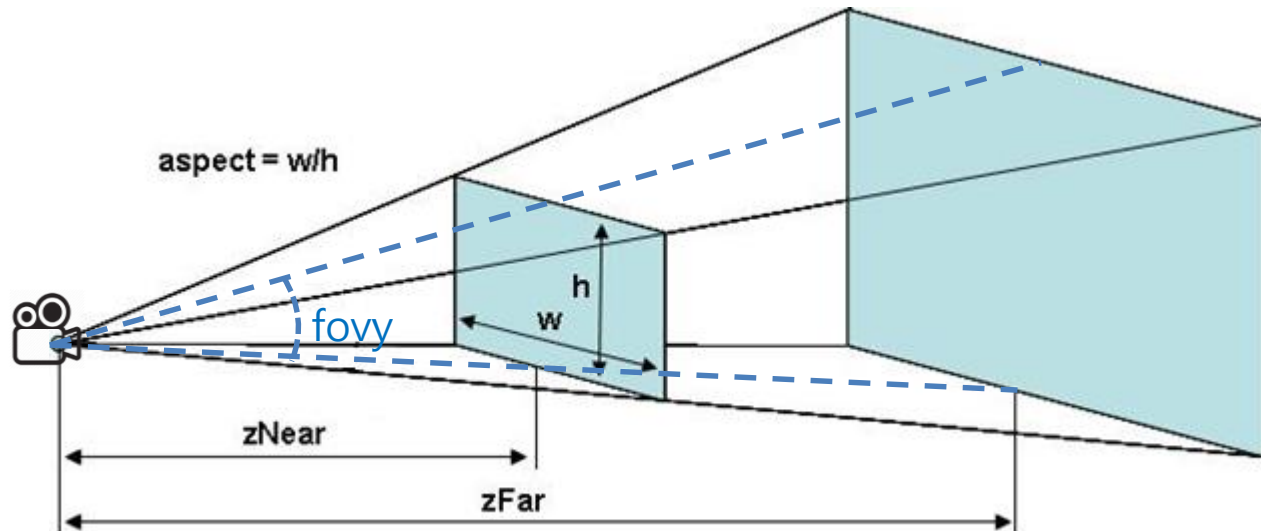
- : Creates a perspective projection matrix.
 - Actually, not easy to use to set a view frustum.
- Sign of near, far:
 - The values for both parameters must be positive.



glm.perspective()

glm.perspective(fovy: float, aspect: float, near: float, far: float) -> fmat4x4

- **fovy**: The field of view angle, in degrees, in the y-direction.
 - **aspect**: The aspect ratio that determines the field of view in the x-direction. width / height.
- : Creates a perspective projection matrix.
 - Easier to use.



[Code] 2-frustum-perspective

```
def main():
    ...
    while not glfwWindowShouldClose(window):
        ...
        # perspective projection - try changing arguments

        # perspective
        P = glm.perspective(45, 1, 1, 10)

        # # frustum
        # # P = glm.frustum(-1,1, -1,1, .1,10)
        # P = glm.frustum(-1,1, -1,1, 1,10)

        # view matrix
        V =
glm.lookAt(glm.vec3(5*np.sin(g_cam_ang), g_cam_height, 5*np.cos(g_c
am_ang)), glm.vec3(0,0,0), glm.vec3(0,1,0))

        ...
```

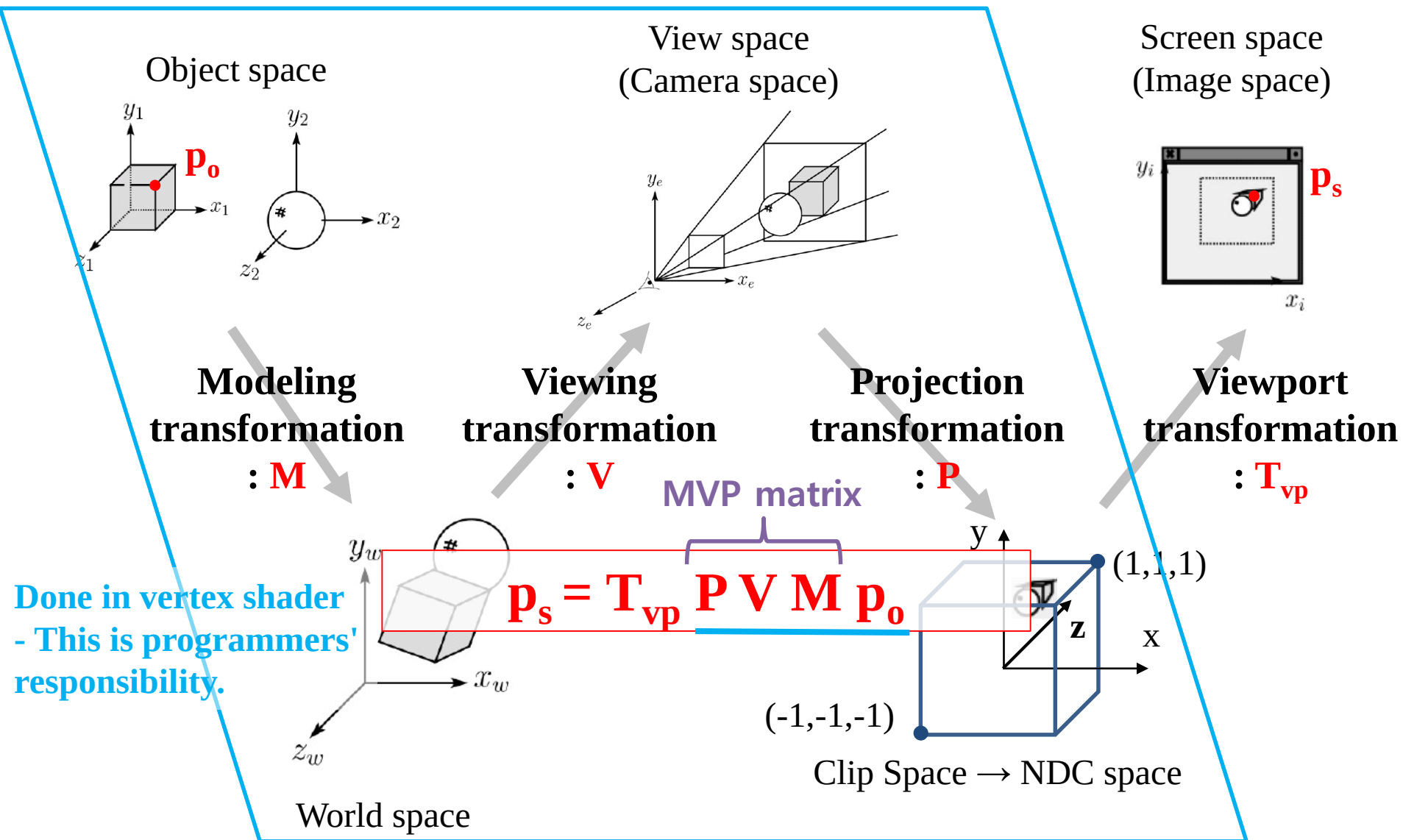
Quiz 3

- Go to <https://www.slido.com/>
- Join #cg-ys
- Click "Polls"
- Submit your answer in the following format:
 - **Student ID: Your answer**
 - e.g. **2024123456: 4.0**
- Note that your quiz answer must be submitted **in the above format** to be counted for attendance.

Viewport Transformation

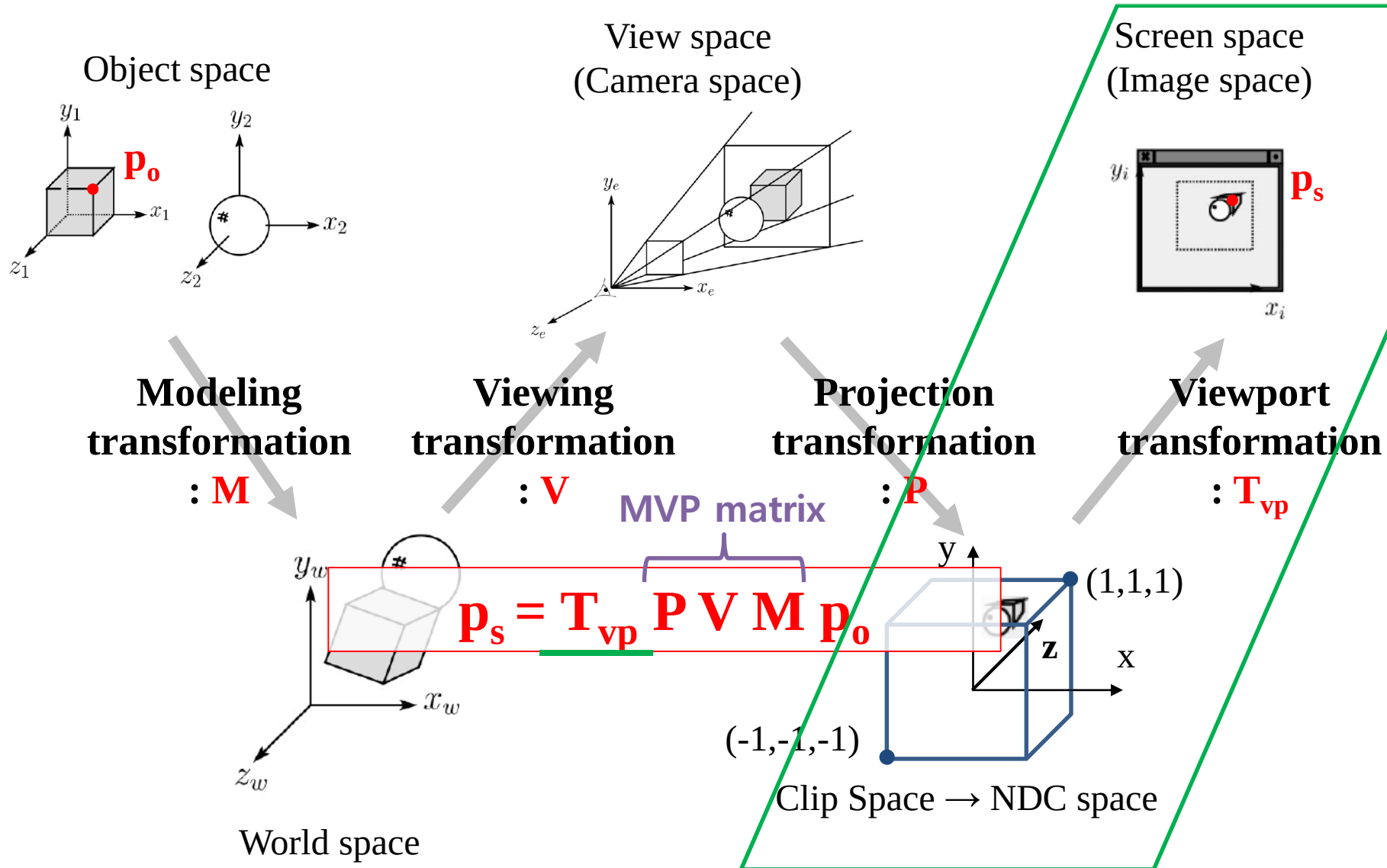
- glViewport()

Recall: Vertex Processing in OpenGL



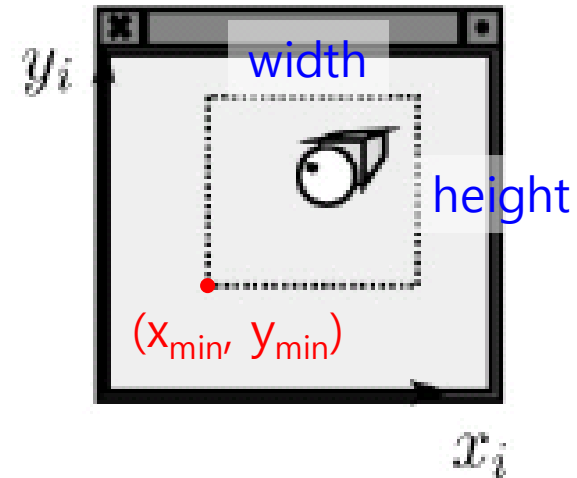
Done internally -
Programmers just
configure this with
`glViewport()`

Vertex Processing in OpenGL



glViewport()

- `glViewport(xmin, ymin, width, height)`
 - `xmin, ymin, width, height`: specified **in pixels**
- `:` Sets the viewport
- Default viewport setting for (`xmin, ymin, width, height`) is **(0, 0, window width, window height)**.
 - If you do not call `glViewport()`, OpenGL uses this default viewport setting.



[Code] 3-viewport

```
def main():  
    ...  
    glViewport(100, 100, 200, 200)  
  
    # loop until the user closes the window  
    while not glfwWindowShouldClose(window):  
        ...
```

Resizing Viewport to Fit to Window

- In the previous example, even if you resize the window, the viewport size remains the same.
- How can we make the viewport always fill the window even when the window is resized?

[Code]4-viewport-fit

```
def framebuffer_size_callback(window, width, height):  
    glViewport(0, 0, width, height)  
  
def main():  
    ...  
    # comment out  
    # glViewport(100,100, 200,200)  
    ...  
    glfwSetFramebufferSizeCallback(window,  
framebuffer_size_callback)  
    ...
```

Maintaining Aspect Ratio When Resizing Viewport

- In the previous example, resizing the viewport also changes the aspect ratio of an object.
- How to ensure that the aspect ratio of an object is always maintained?

[Code] 5-viewport-fit-preserve-objratio-ortho

```
# now projection matrix P is a global variable so that it can be accessed
from main() and framebuffer_size_callback()
g_P = glm.mat4()
...

def framebuffer_size_callback(window, width, height):
    global g_P

    glViewport(0, 0, width, height)

    ortho_height = 10.
    ortho_width = ortho_height * width/height
    g_P = glm.ortho(-ortho_width*.5,ortho_width*.5,
-ortho_height*.5,ortho_height*.5, -10,10)
```

[Code] 5-viewport-fit-preserve-objratio-ortho

```
def main():
    global g_P
    ...
    # initialize projection matrix
    ortho_height = 10.
    ortho_width = ortho_height * 800/800    # initial width/height
    g_P = glm.ortho(-ortho_width*.5, ortho_width*.5, -
ortho_height*.5, ortho_height*.5, -10, 10)

    while not glfwWindowShouldClose(window):
        ...

        # draw world frame
        draw_frame(vao_frame, g_P*V*glm.mat4(), MVP_loc)

        ...

        # # draw cube w.r.t. the current frame MVP
        # draw_cube(vao_cube, g_P*V*M, MVP_loc)

        # draw cube array w.r.t. the current frame MVP
        draw_cube_array(vao_cube, g_P*V*M, MVP_loc)
```

Time for Lab Check

- Let's start today's lab check.
- TA will guide you.