Computer Graphics

6 - Viewing & Projection 2, Mesh

Yoonsang Lee

Spring 2021

Midterm Exam Announcement

- 본부의 중간고사 방침: "2021학년도 1학기 중간고사는 대면시험을 원칙으로 하되 코로나19 감염병 상황과 관련해 정보 방역체계 강화 등 부득이한 경우 원격시험으로 전환될 수 있습니다"

- 이에 따라, 우리 강의에서도 대면시험으로 중간고사를 진행하기로 함.
  - 전체 인원을 대상으로 한 온라인 시험에서 환경 설정 및 시험 진행에 있어 많은 애로사항이 있던 경험을 고려

- 날짜 및 시간: **4월 19일 (월) 오전 9시30분~10시30분**
- 장소: IT.BT관 **507, 508호**에 나뉘어 사회적 거리두기를 유지하며 시험 진행 (각 실습실 별 응시 명단은 추후 공지)

- 시험 일정 즈음에서 확진판정을 받거나, 자가격리 상태이거나, 혹은 그렇지 않더라도 발열 혹은 호흡기 증상이 나타나는 경우에는 무리하게 중간고사를 보러 학교에 오지 말고 조교에게 빠르게 메일을 보내 미리 알려주면 방안을 마련해보도록 하겠습니다.

Midterm Exam Announcement

- Scope:
  - 2 - Introduction to NumPy & OpenGL
  - 3 - Transformation 1
  - 4 - Transformation 2
  - 5 - Rendering Pipeline, Viewing & Projection 1
  - 6 - Viewing & Projection 2, Mesh
  - 7 - Lighting & Shading

- **You cannot leave until 30 minutes after the start of the exam** even if you finish the exam earlier.

- That means, **you cannot enter the room after 30 minutes from the start of the exam (do not be late, never too late!).**

- **시험 시 필히 학생증을 지참하기 바랍니다.**

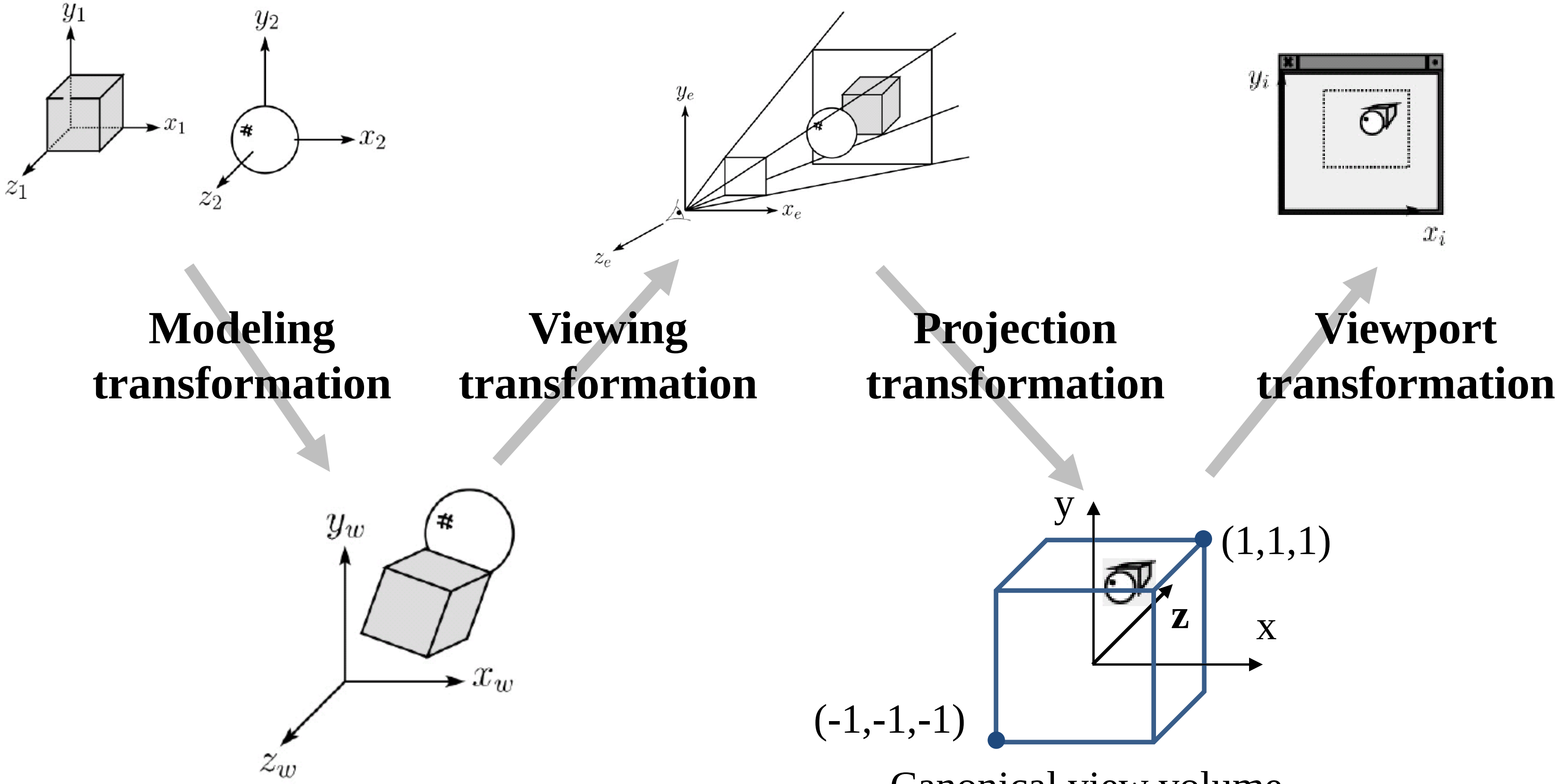
Topics Covered

- Projection Transformation
  - Perspective Projection

- Viewport Transformation

- Mesh
  - Polygon mesh & triangle mesh
  - Representations for triangle meshes
  - OpenGL vertex array
  - OBJ file format

Vertex Processing (Transformation Pipeline)



Object space

Modeling transformation

World space

Viewing transformation

View space (Camera space)

Projection transformation

Canonical view volume (Normalized device coordinates, NDC)

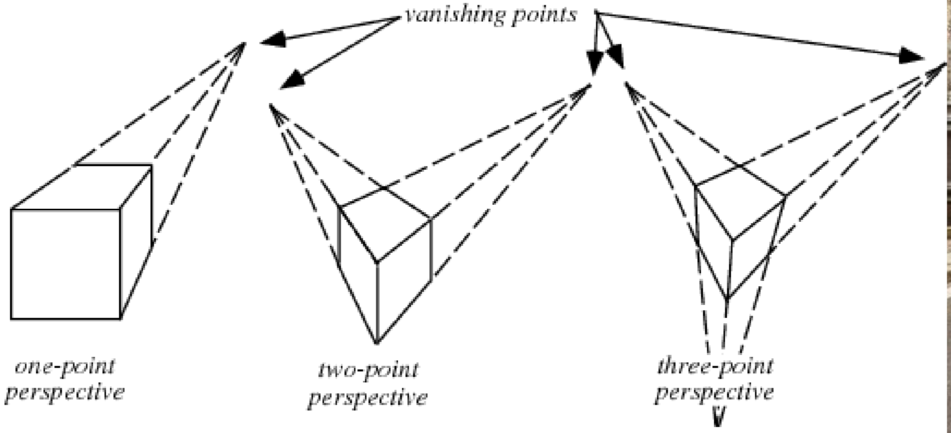
Viewport transformation

Screen space (Image space)

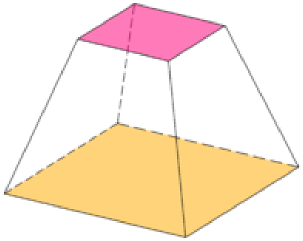
Perspective Effects

- Distant objects become small.

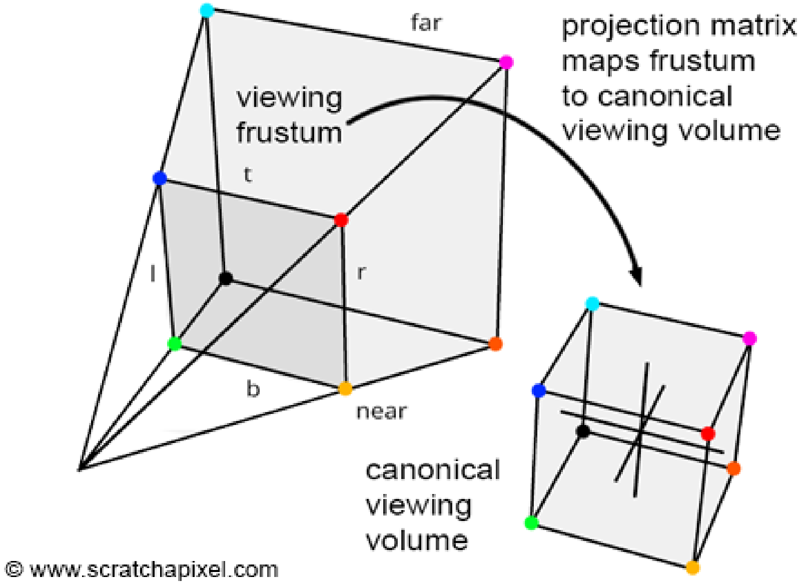
Vanishing point: The point or points to which the extensions of parallel lines appear to converge in a perspective drawing



Perspective Projection



- View volume : Frustum (절두체)
- → "Viewing frustum"
- Perspective projection : Mapping from a viewing frustum to a canonical view volume

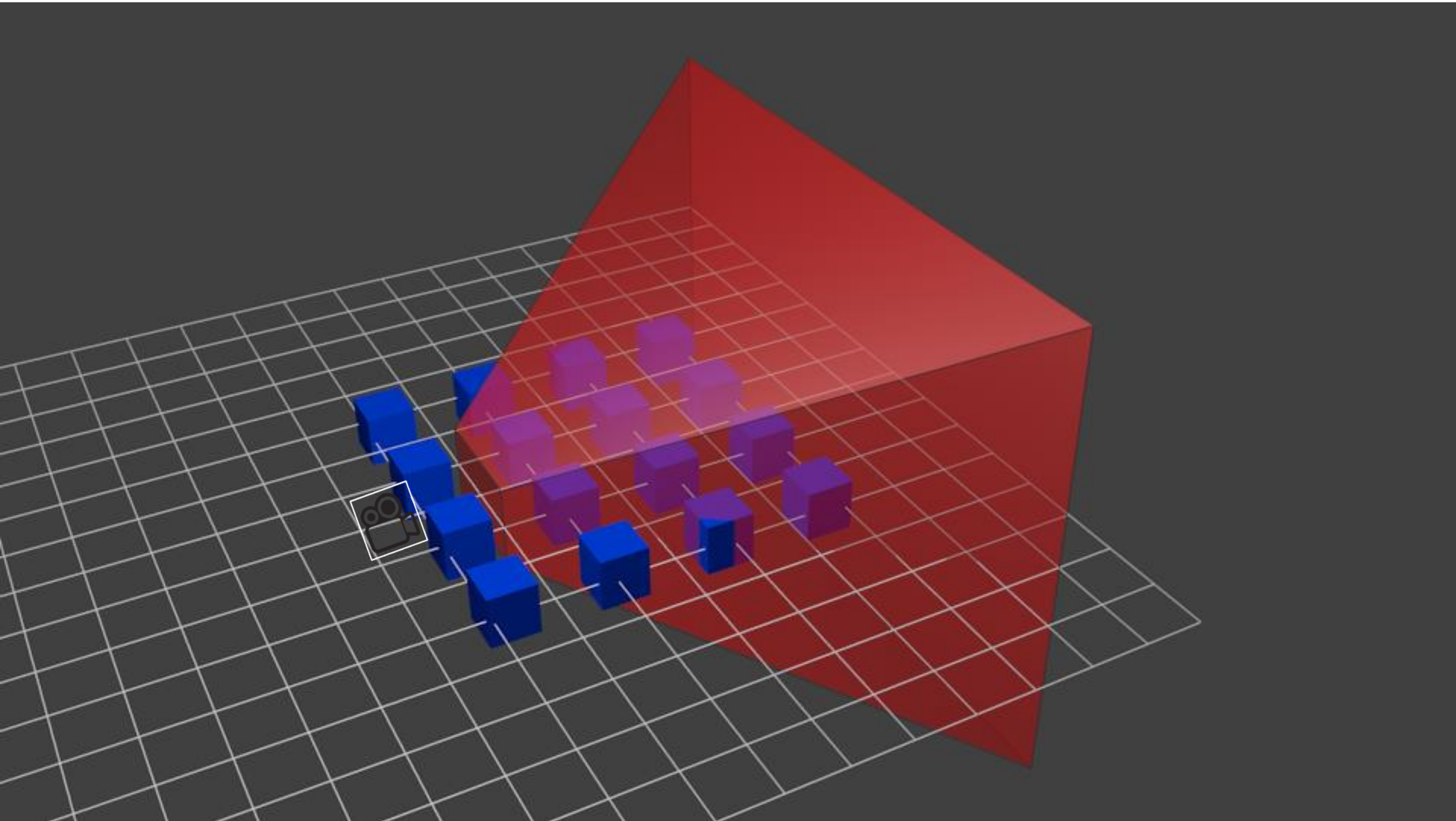


© www.scratchapixel.com

Why does this mapping generate a perspective effect?

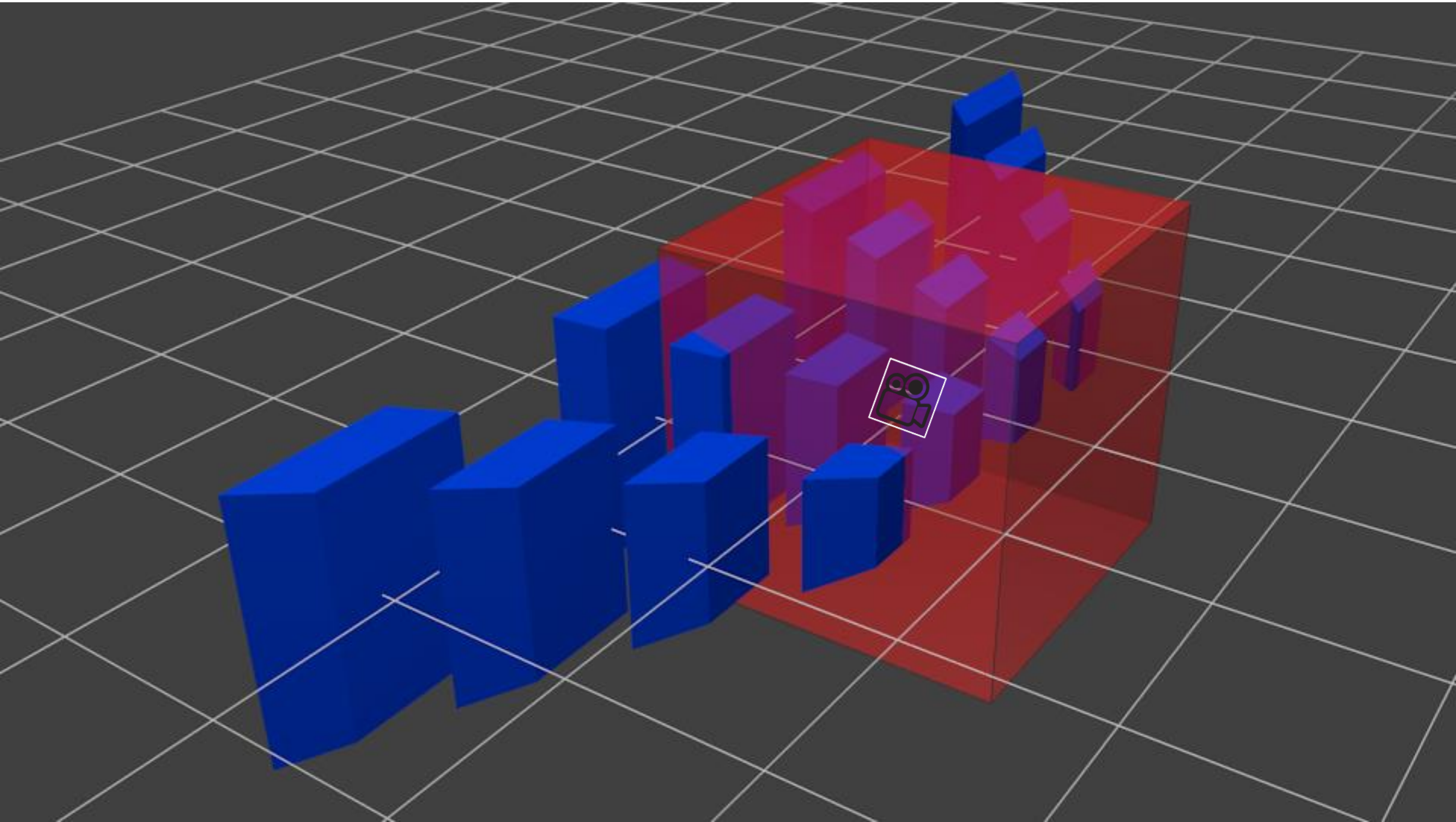
Original 3D scene

Red: viewing frustum, Blue: objects



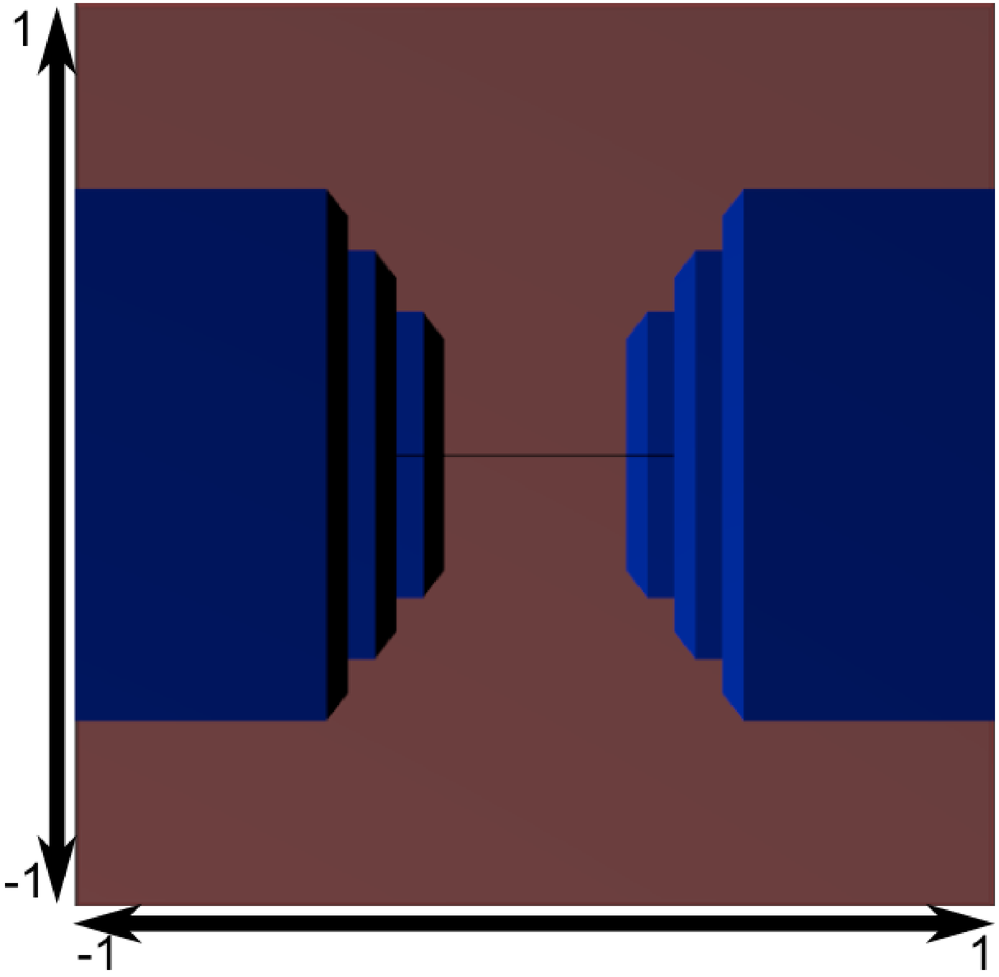
An Example of Perspective Projection

After perspective projection



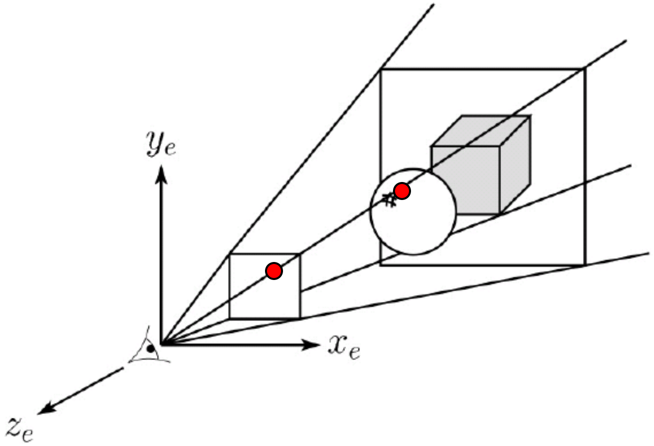
An Example of Perspective Projection

The camera view



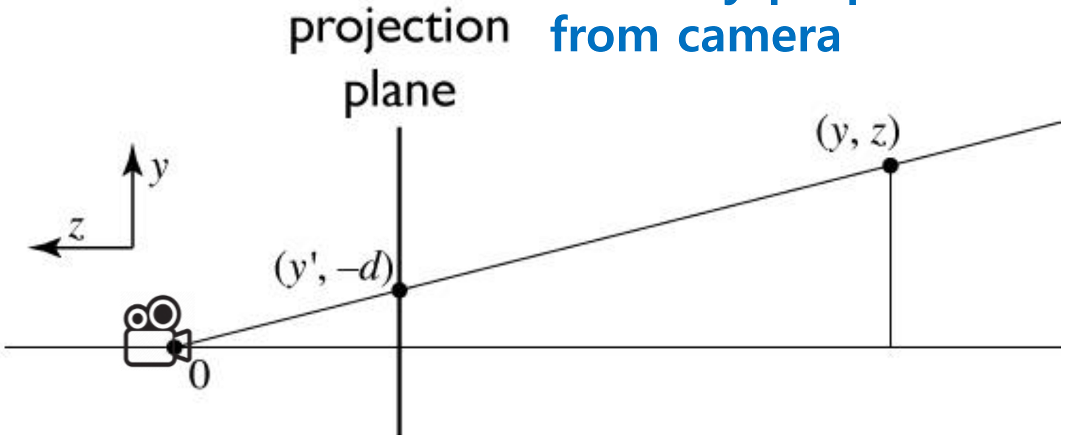
Let’s first consider
3D View Frustum→2D Projection Plane

- Consider the projection of a 3D point on the camera plane



Perspective projection

The size of an object on the screen is inversely proportional to its distance from camera



similar triangles:

$$\frac{y'}{d} = \frac{y}{-z}$$

$$y' = -dy/z$$

Homogeneous coordinates revisited

- Perspective requires division
  - that is **not** part of affine transformations
  - in affine, parallel lines stay parallel
    - therefore not vanishing point
    - therefore no rays converging on viewpoint
- “True” purpose of homogeneous coords: projection

Homogeneous coordinates revisited

- Introduced *w* = 1 coordinate as a placeholder

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

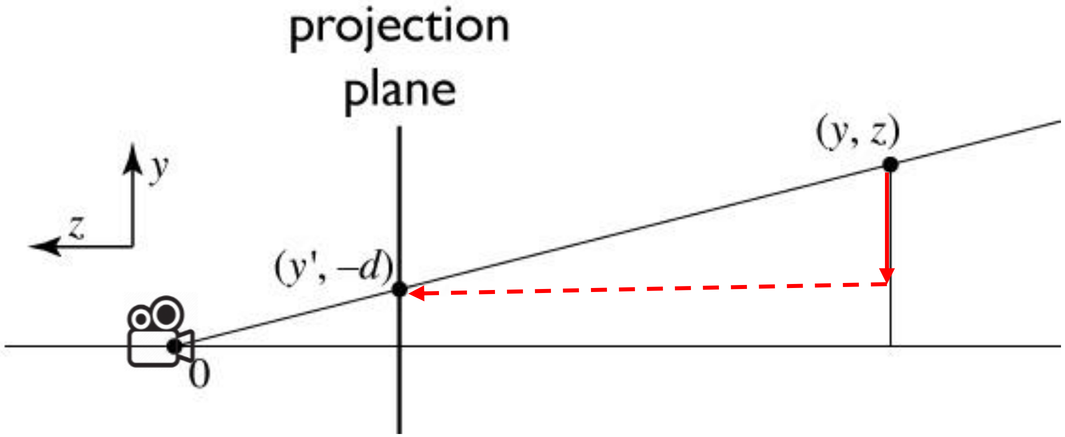
  - used as a convenience for unifying translation with linear transformation

- Can also allow arbitrary *w*

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \sim \begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix}$$

All scalar multiples of a 4-vector are equivalent

Perspective projection



to implement perspective, just move z to w:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -dx/z \\ -dy/z \\ 1 \end{bmatrix} \sim \begin{bmatrix} dx \\ dy \\ -z \end{bmatrix} = \begin{bmatrix} d & 0 & 0 & 0 \\ 0 & d & 0 & 0 \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

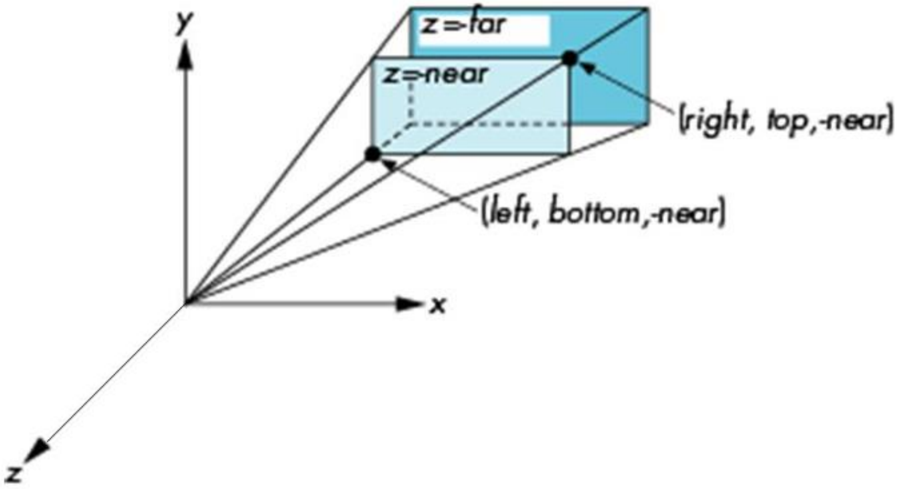
Perspective Projection Matrix

- This 3D → 2D projection example gives the basic idea of perspective projection.

- What we really have to do is 3D → 3D, View Frustum → Canonical View Volume.

- For details for this process, see *6 - reference-projection.pdf*

- M_{pers}= $$\begin{pmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

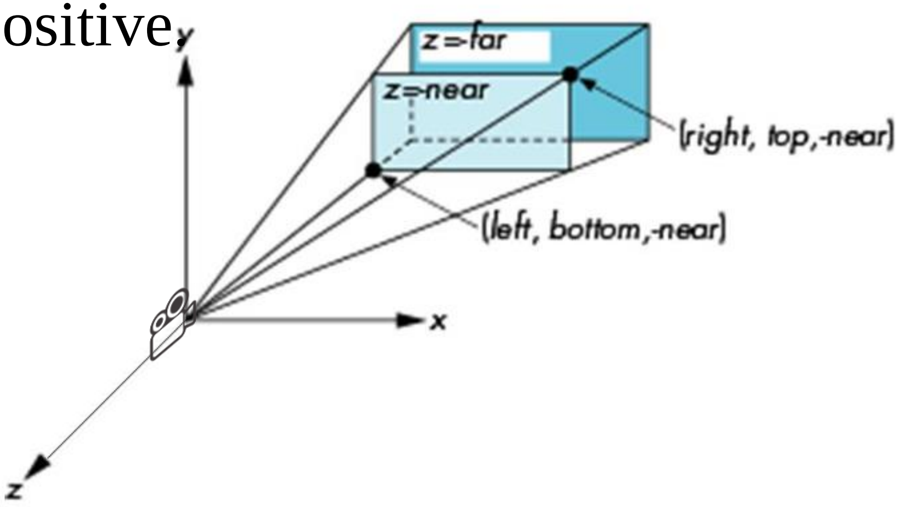


glFrustum()

- glFrustum(left, right, bottom, top, near, far)
- : Creates a perspective projection matrix and right-multiplies the current transformation matrix by it

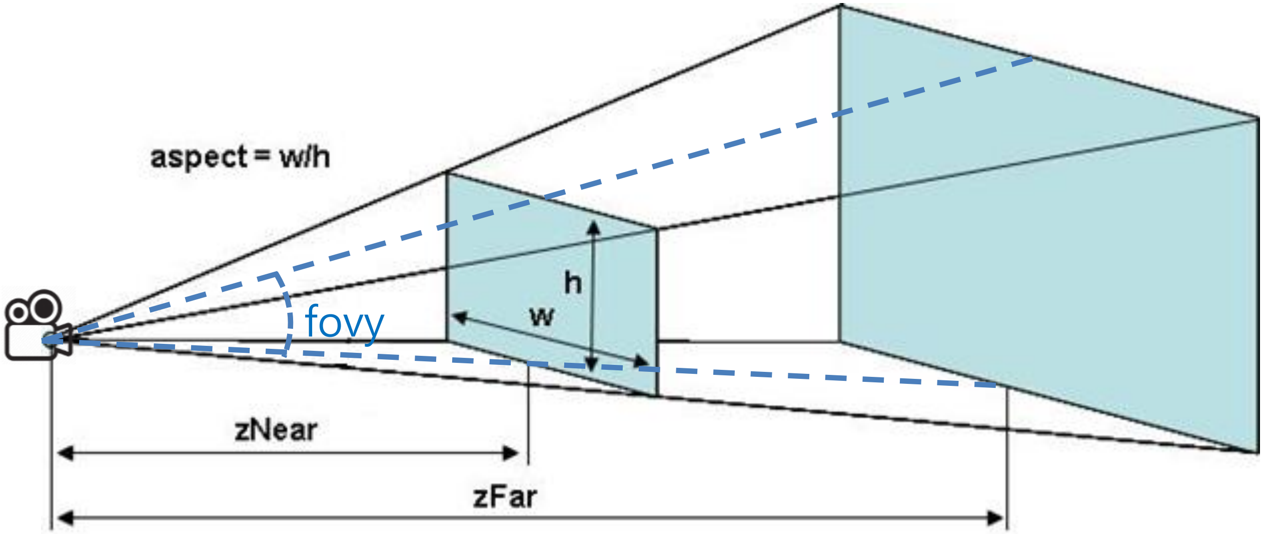
- Sign of near, far:
  - Both distances must be positive

- $C \leftarrow CM_{pers}$



gluPerspective()

- gluPerspective(fovy, aspect, zNear, zFar)
  - 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. The aspect ratio is the ratio of x (width) to y (height).
- : Creates a perspective projection matrix and right-multiplies the current transformation matrix by it
- $C \leftarrow CM_{pers}$



[Practice] glFrustum(), gluPerspective()

```python
import glfw
from OpenGL.GL import *
from OpenGL.GLU import *
import numpy as np

gCamAng = 0.
gCamHeight = 1.

# draw a cube of side 1, centered at the origin.
def drawUnitCube():
    glBegin(GL_QUADS)
    glVertex3f( 0.5, 0.5,-0.5)
    glVertex3f(-0.5, 0.5,-0.5)
    glVertex3f(-0.5, 0.5, 0.5)
    glVertex3f( 0.5, 0.5, 0.5)

    glVertex3f( 0.5,-0.5, 0.5)
    glVertex3f(-0.5,-0.5, 0.5)
    glVertex3f(-0.5,-0.5,-0.5)
    glVertex3f( 0.5,-0.5,-0.5)

    glVertex3f( 0.5, 0.5, 0.5)
    glVertex3f(-0.5, 0.5, 0.5)
    glVertex3f(-0.5,-0.5, 0.5)
    glVertex3f( 0.5,-0.5, 0.5)

    glVertex3f( 0.5,-0.5,-0.5)
    glVertex3f(-0.5,-0.5,-0.5)
    glVertex3f(-0.5, 0.5,-0.5)
    glVertex3f( 0.5, 0.5,-0.5)
    glVertex3f(-0.5, 0.5, 0.5)
    glVertex3f(-0.5, 0.5,-0.5)
    glVertex3f(-0.5,-0.5,-0.5)
    glVertex3f(-0.5,-0.5, 0.5)

    glVertex3f( 0.5, 0.5,-0.5)
    glVertex3f( 0.5, 0.5, 0.5)
    glVertex3f( 0.5,-0.5, 0.5)
    glVertex3f( 0.5,-0.5,-0.5)
    glEnd()

def drawCubeArray():
    for i in range(5):
        for j in range(5):
            for k in range(5):
                glPushMatrix()
                glTranslatef(i,j,-k-1)
                glScalef(.5,.5,.5)
                drawUnitCube()
                glPopMatrix()

def drawFrame():
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([1.,0.,0.]))
    glColor3ub(0, 255, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([0.,1.,0.]))
    glColor3ub(0, 0, 255)
    glVertex3fv(np.array([0.,0.,0]))
    glVertex3fv(np.array([0.,0.,1.]))
    glEnd()
```

```
def render():
    global gCamAng, gCamHeight

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)
    glPolygonMode( GL_FRONT_AND_BACK, GL_LINE )

    glLoadIdentity()

    # test other parameter values
    glFrustum(-1,1, -1,1, .1,10)
    # glFrustum(-1,1, -1,1, 1,10)

    # test other parameter values
    # gluPerspective(45, 1, 1,10)

    # test with this line
gluLookAt(5*np.sin(gCamAng),gCamHeight,5*np.cos(
gCamAng), 0,0,0, 0,1,0)

    drawFrame()
    glColor3ub(255, 255, 255)

    drawUnitCube()

    # test
    # drawCubeArray()
```

```
def key_callback(window, key, scancode, action,
mods):
    global gCamAng, gCamHeight
    if action==glfw.PRESS or
action==glfw.REPEAT:
        if key==glfw.KEY_1:
            gCamAng += np.radians(-10)
        elif key==glfw.KEY_3:
            gCamAng += np.radians(10)
        elif key==glfw.KEY_2:
            gCamHeight += .1
        elif key==glfw.KEY_W:
            gCamHeight += -.1

def main():
    if not glfw.init():
        return
    window =
glfw.create_window(640,640,‘glFrustum()',
None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)

    while not glfw.window_should_close(window):
        glfw.poll_events()
        render()
        glfw.swap_buffers(window)

    glfw.terminate()

if __name__ == "__main__":
    main()
```

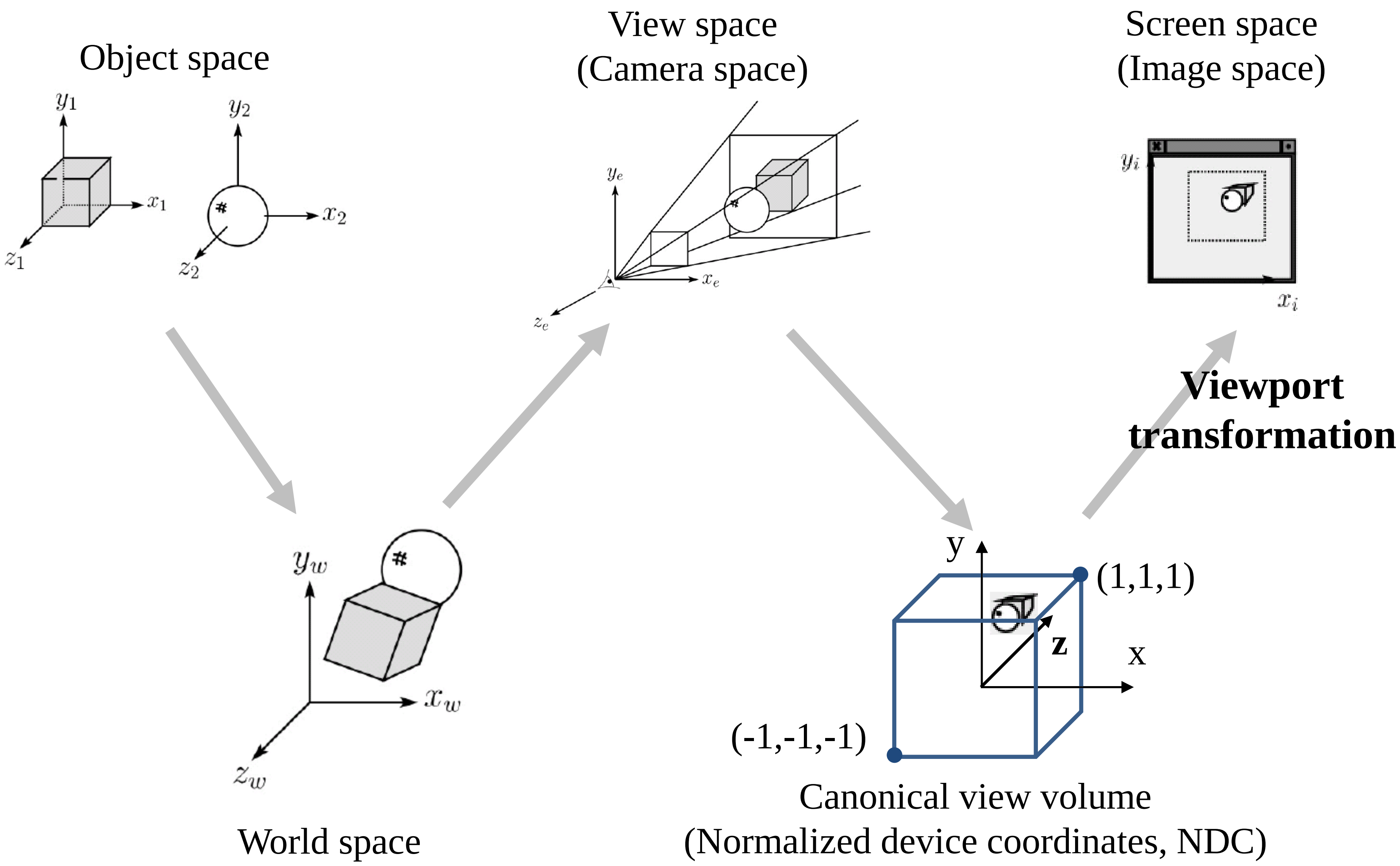
Quiz #1

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Viewport Transformation



Object space

View space (Camera space)

Screen space (Image space)

Viewport transformation

World space

Canonical view volume (Normalized device coordinates, NDC)

Recall that...

- 1. Placing objects

→ **Modeling transformation**

- 2. Placing the “camera”

→ **Viewing transformation**

- 3. Selecting a “lens”

→ **Projection transformation**

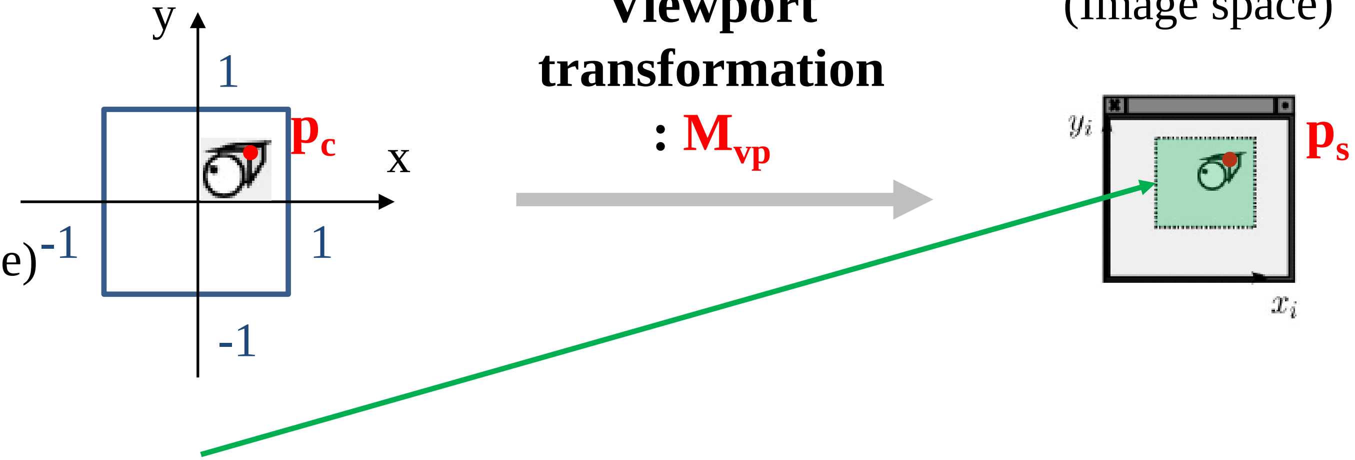
- 4. Displaying on a “cinema screen”

→ **Viewport transformation**

Viewport Transformation

Canonical view volume
(looking down +z direction)

$-1 \le z \le 1$
(z range of canonical view volume)



Viewport transformation : M_{vp}

Screen space
(Image space)

$0 \le z \le 1$
(default depth buffer range)

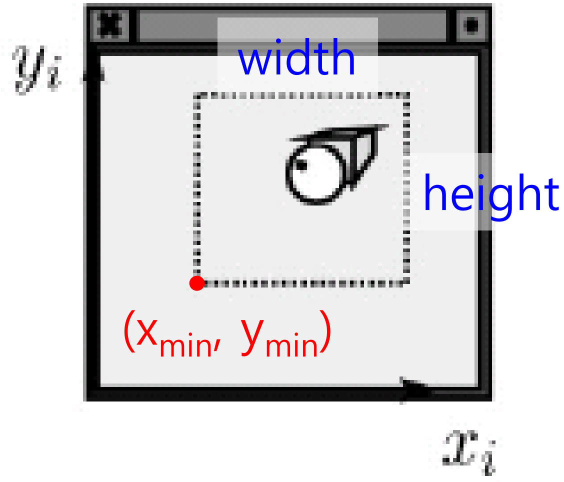
- Viewport: a rectangular viewing region of screen

- So, viewport transformation is also a kind of windowing transformation.

Viewport Transformation Matrix

- In the windowing transformation matrix,
- By substituting x_h, x_l, x_h’, … with corresponding variables in viewport transformation,

$$M_{vp} = \begin{bmatrix} \frac{width}{2} & 0 & 0 & \frac{width}{2} + x_{min} \\ 0 & \frac{height}{2} & 0 & \frac{height}{2} + y_{min} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

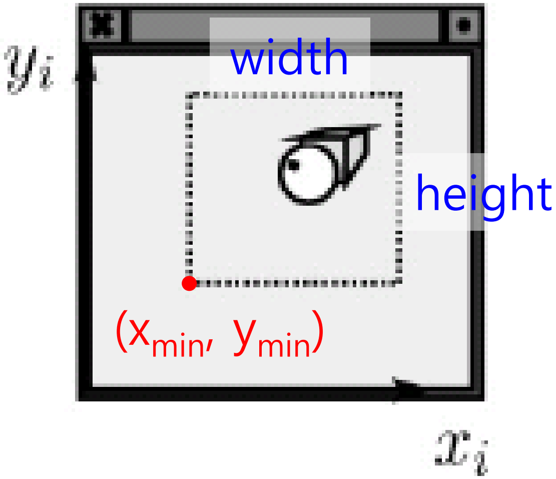


glViewport()

- glViewport(xmin, ymin, width, height)
  - xmin, ymin, width, height: specified **in pixels**

- : Sets the viewport
  - This function does NOT explicitly multiply a viewport matrix with the current matrix.
  - Viewport transformation is internally done in OpenGL, so you can apply transformation matrices **starting from a canonical view volume**, not a screen space.

- 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.



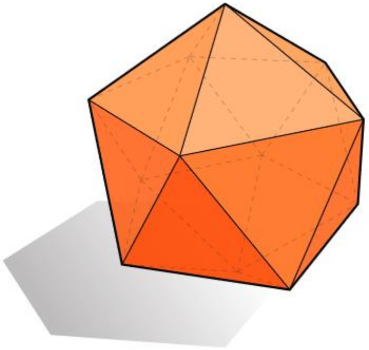
[Practice] glViewport()

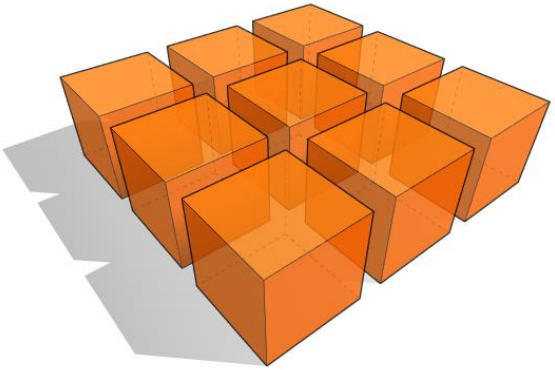
```
def main():
    # ...
    glfw.make_context_current(window)
    glViewport(100,100,200,200)
    # ...
```

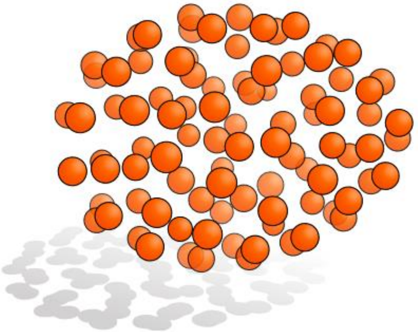
Mesh

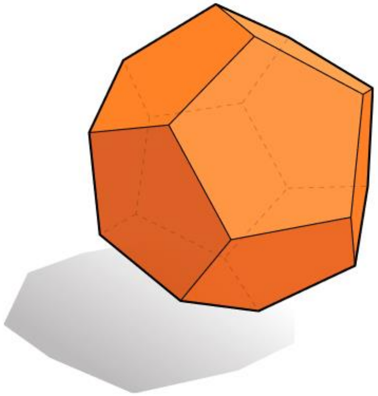
Many ways to digitally encode geometry

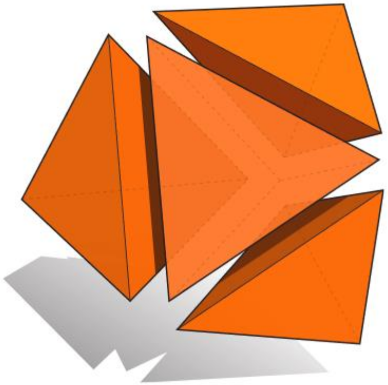
- EXPLICIT
  - point cloud
  - polygon mesh
  - subdivision, NURBS
  - L-systems
  - ...
- IMPLICIT
  - level set
  - algebraic surface
  - ...
- Each choice best suited to a different task/type of geometry





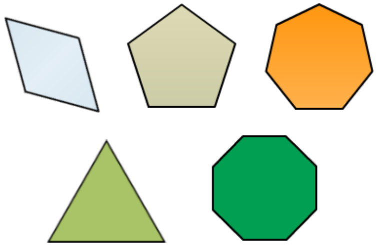




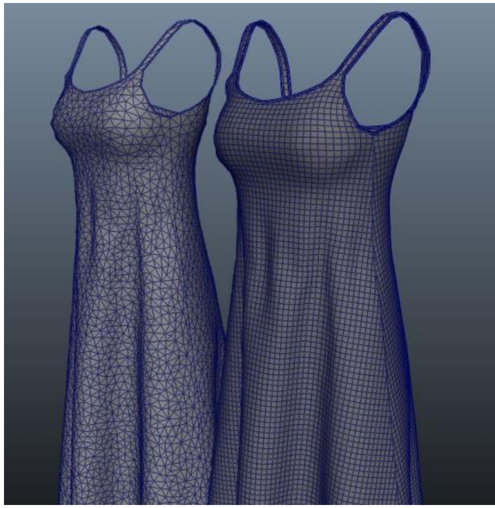


The Most Popular One : Polygon Mesh

- Because this can model any arbitrary complex shapes with relatively simple representations and can be rendered fast.



- **Polygon**: a “closed” shape with straight sides



- **Polygon mesh**: a bunch of polygons in 3D space that are connected together to form a surface
  - Usually use *triangles* or *quads* (4 side polygon)

Triangle Mesh

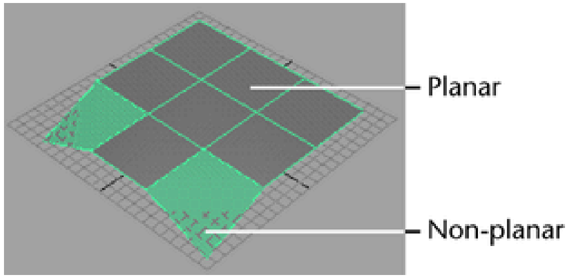
- A general N-polygon can be
  - Non-planar
  - Non-convex
- , which are not desirable for fast rendering.

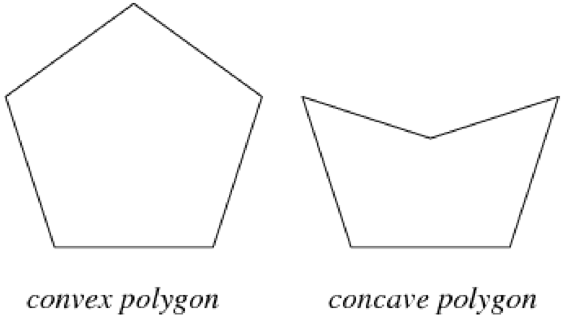
- A triangle does not have such problems. It’s always planar & convex.

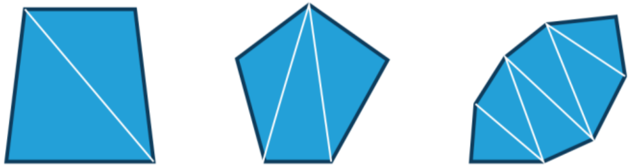
- and N-polygons can be composed of multiple triangles.

- That's why modern GPUs draw everything as a set of triangles.

- So, we'll focus on triangle meshes.

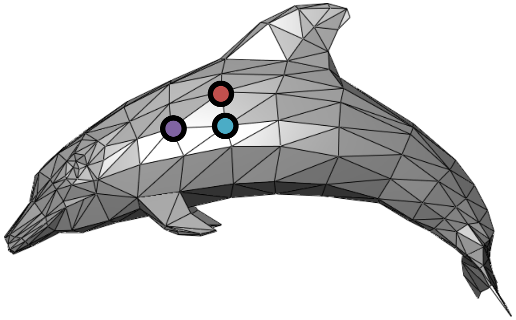






Representation for Triangle Mesh

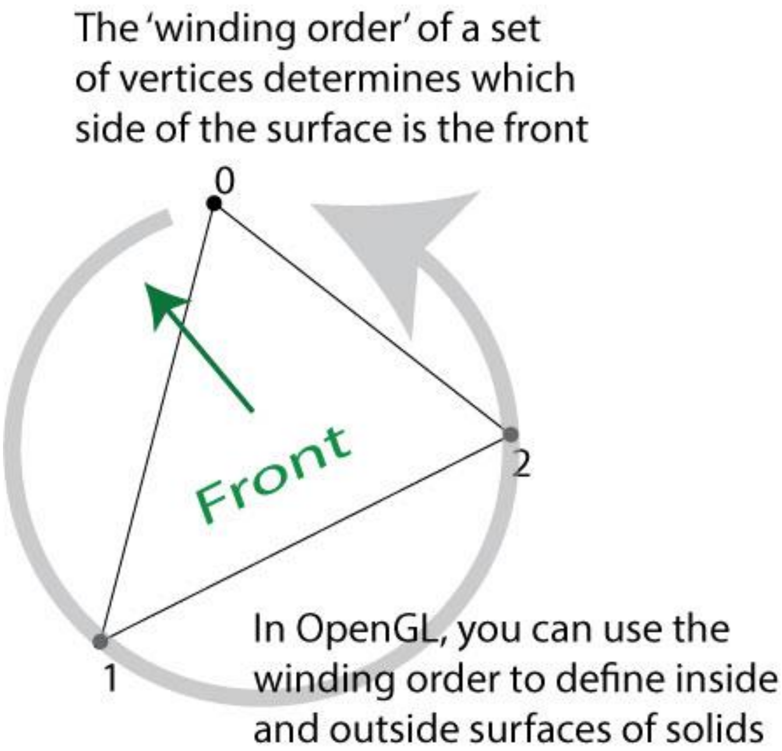
- It’s about how to store
  - vertex positions
  - relationship between vertices (to make triangles)
- on memory.



- We'll see
  - Separate triangles
  - Indexed triangle set

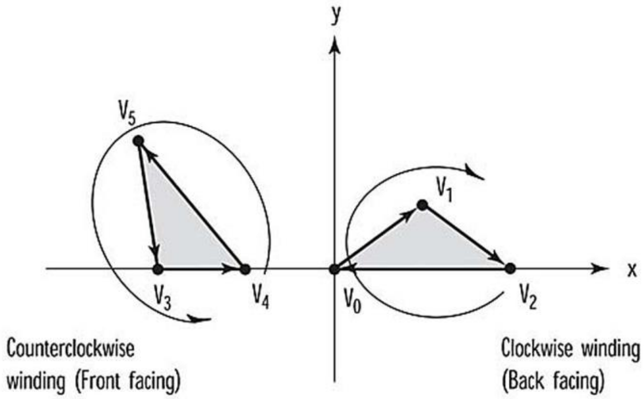
Vertex Winding Order

- In OpenGL, by default, polygons whose vertices appear in **counterclockwise** order on the screen is front-facing

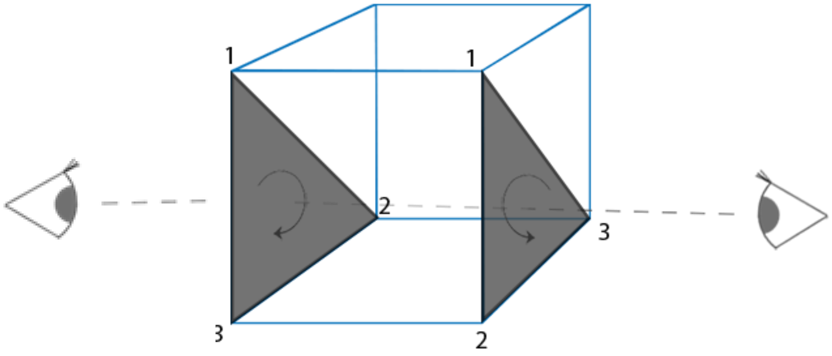


The 'winding order' of a set of vertices determines which side of the surface is the front

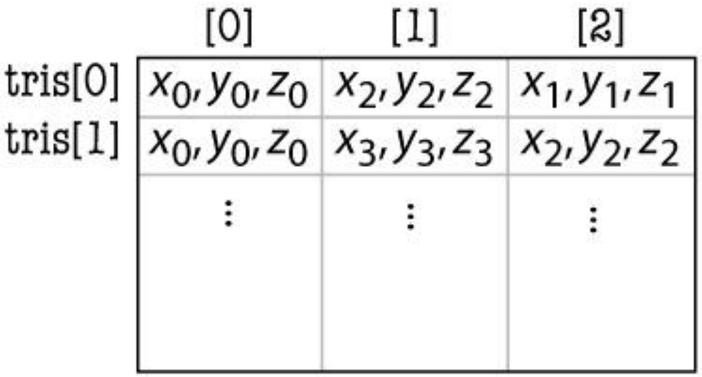
In OpenGL, you can use the winding order to define inside and outside surfaces of solids



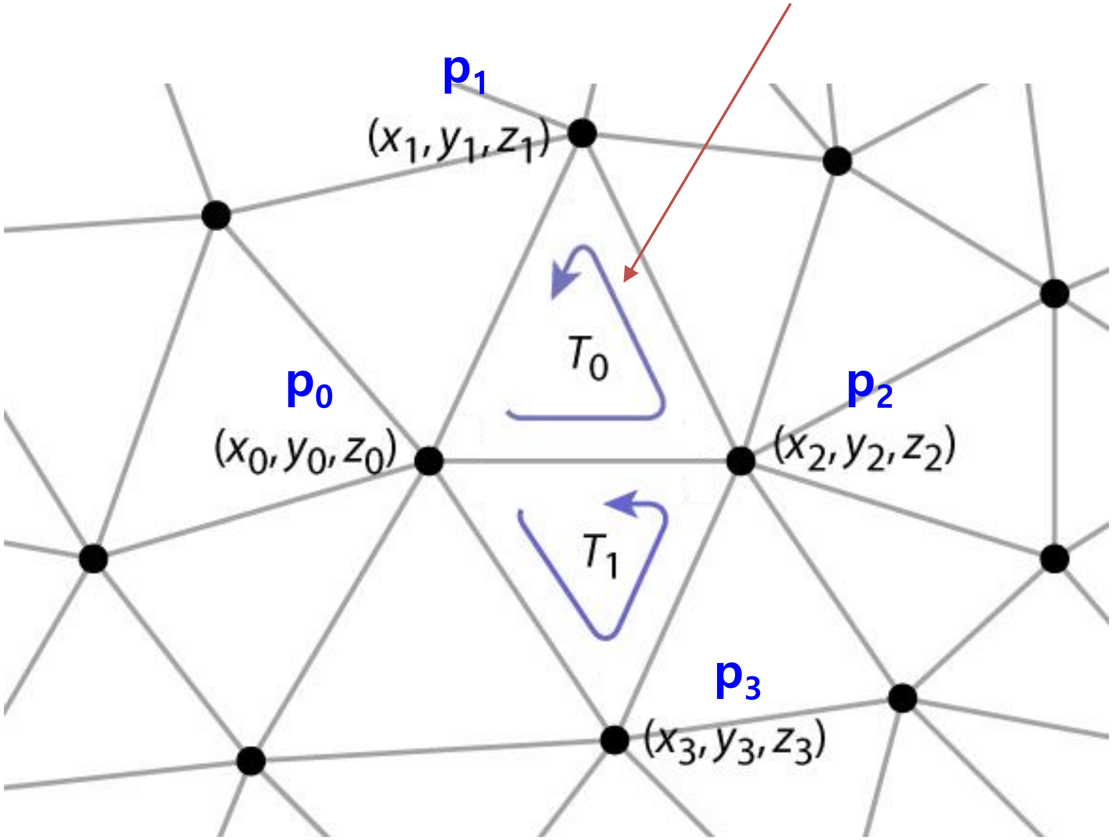
y
x
V_5
V_3
V_4
V_0
V_1
V_2
Counterclockwise winding (Front facing)
Clockwise winding (Back facing)



Separate triangles



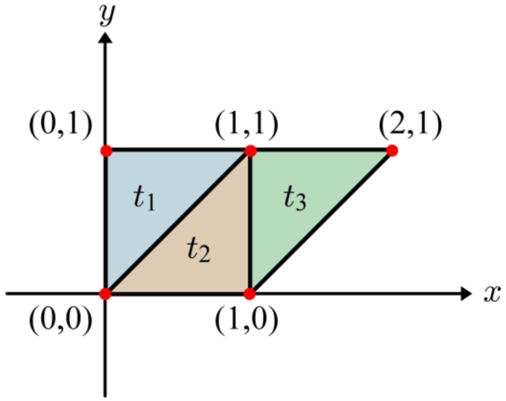
	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮

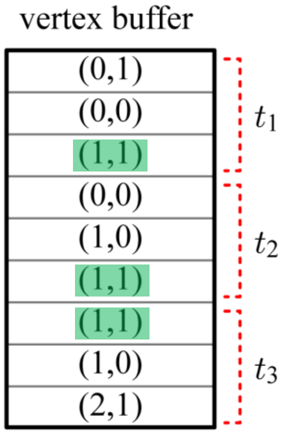


counter-clockwise order

Separate Triangles

- Various problems
  - Wastes space
  - Cracks due to roundoff
  - Difficulty of finding neighbors

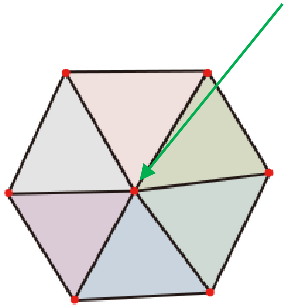




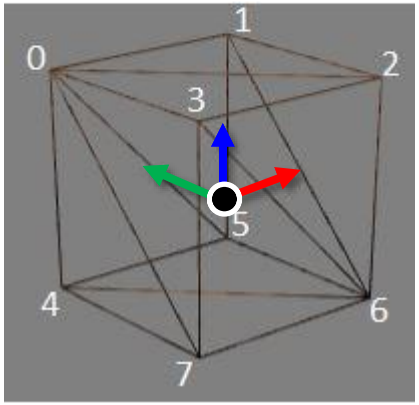
vertex buffer	
(0,1)	t_1
(0,0)	
(1,1)	
(0,0)	t_2
(1,0)	
(1,1)	
(1,1)	t_3
(1,0)	
(2,1)	

(1,1) is stored 3 times!

stored 6 times!



Example: a cube of length 2



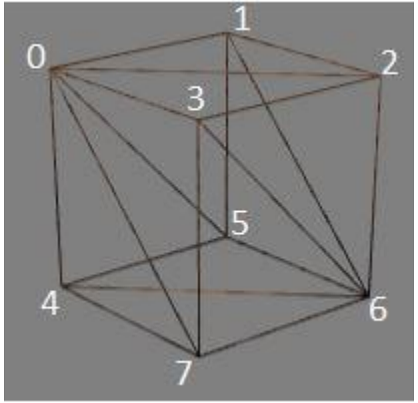
vertex index	position
0	(-1 , 1 , 1)
1	(1 , 1 , 1)
2	(1 , -1 , 1)
3	(-1 , -1 , 1)
4	(-1 , 1 , -1)
5	(1 , 1 , -1)
6	(1 , -1 , -1)
7	(-1 , -1 , -1)

Drawing Separate Triangles using glVertex*()

- You can use glVertex*() like this:

```
def drawCube_glVertex():
    glBegin(GL_TRIANGLES)
    glVertex3f( -1 ,  1 ,  1 ) # v0
    glVertex3f(  1 , -1 ,  1 ) # v2
    glVertex3f(  1 ,  1 ,  1 ) # v1

    glVertex3f( -1 ,  1 ,  1 ) # v0
    glVertex3f( -1 , -1 ,  1 ) # v3
    glVertex3f(  1 , -1 ,  1 ) # v2

    glVertex3f( -1 ,  1 , -1 ) # v4
    glVertex3f(  1 ,  1 , -1 ) # v5
    glVertex3f(  1 , -1 , -1 ) # v6

    glVertex3f( -1 ,  1 , -1 ) # v4
    glVertex3f(  1 , -1 , -1 ) # v6
    glVertex3f( -1 , -1 , -1 ) # v7

    glVertex3f( -1 ,  1 ,  1 ) # v0
    glVertex3f(  1 ,  1 ,  1 ) # v1
    glVertex3f(  1 ,  1 , -1 ) # v5

    glVertex3f( -1 ,  1 ,  1 ) # v0
    glVertex3f(  1 ,  1 , -1 ) # v5
    glVertex3f( -1 ,  1 , -1 ) # v4
```

```
    glVertex3f( -1 , -1 ,  1 ) # v3
    glVertex3f(  1 , -1 , -1 ) # v6
    glVertex3f(  1 , -1 ,  1 ) # v2

    glVertex3f( -1 , -1 ,  1 ) # v3
    glVertex3f( -1 , -1 , -1 ) # v7
    glVertex3f(  1 , -1 , -1 ) # v6

    glVertex3f(  1 ,  1 ,  1 ) # v1
    glVertex3f(  1 , -1 ,  1 ) # v2
    glVertex3f(  1 , -1 , -1 ) # v6

    glVertex3f(  1 ,  1 ,  1 ) # v1
    glVertex3f(  1 , -1 , -1 ) # v6
    glVertex3f(  1 ,  1 , -1 ) # v5

    glVertex3f( -1 ,  1 ,  1 ) # v0
    glVertex3f( -1 , -1 , -1 ) # v7
    glVertex3f( -1 , -1 ,  1 ) # v3

    glVertex3f( -1 ,  1 ,  1 ) # v0
    glVertex3f( -1 ,  1 , -1 ) # v4
    glVertex3f( -1 , -1 , -1 ) # v7
    glEnd()
```



Vertex Array

- But from now on, let’s use a more advanced method to draw polygons: *Vertex array*

- **Vertex array**: an array of vertex data including vertex positions, normals, texture coordinates and color information
  - For now, consider vertex positions only

- By using a vertex array, you can draw a whole mesh just by calling a OpenGL function **once**! (instead of a huge number of glVertex*() calls!)
- → Tremendous increase in rendering performance!

Drawing Separate Triangles using Vertex Array

- 1. Create a vertex array for your mesh
  - Using numpy.ndarray or python list

- 2. Specify “pointer” to this vertex array
  - Using glVertexPointer()

- 3. Render the mesh using the specified “pointer”
  - Using glDrawArrays()

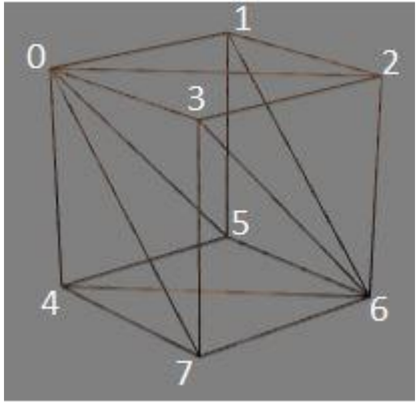
glVertexPointer() & glDrawArrays()

- **glVertexPointer(size, type, stride, pointer)**
- : specifies the location and data format of a vertex array
  - **size**: The number of vertex coordinates, 2 for 2D points, 3 for 3D points
  - **type**: The data type of each coordinate value in the array. GL_FLOAT, GL_SHORT, GL_INT or GL_DOUBLE.
  - **stride**: The byte offset to the next vertex
  - **pointer**: The pointer to the first coordinate of the first vertex in the array

- **glDrawArrays(mode , first , count)**
- : render primitives from the vertex array specified by glVertexPointer()
  - **mode**: The primitive type to render. GL_POINTS, GL_TRIANGLES, ...
  - **first**: The starting index in the array specified by glVertexPointer()
  - **count**: The number of vertices to be rendered (duplicate vertices also should be counted separately)

[Practice] Drawing Separate Triangles using Vertex Array

```
import glfw
from OpenGL.GL import *
import numpy as np
from OpenGL.GLU import *

gCamAng = 0
gCamHeight = 1.

def createVertexArraySeparate():
    varr = np.array([
            ( -1 ,  1 ,  1 ), # v0
            (  1 , -1 ,  1 ), # v2
            (  1 ,  1 ,  1 ), # v1

            ( -1 ,  1 ,  1 ), # v0
            ( -1 , -1 ,  1 ), # v3
            (  1 , -1 ,  1 ), # v2

            ( -1 ,  1 , -1 ), # v4
            (  1 ,  1 , -1 ), # v5
            (  1 , -1 , -1 ), # v6

            ( -1 ,  1 , -1 ), # v4
            (  1 , -1 , -1 ), # v6
            ( -1 , -1 , -1 ), # v7

            ( -1 ,  1 ,  1 ), # v0
            (  1 ,  1 ,  1 ), # v1
            (  1 ,  1 , -1 ), # v5
```



```
            ( -1 ,  1 ,  1 ), # v0
            (  1 ,  1 , -1 ), # v5
            ( -1 ,  1 , -1 ), # v4

            ( -1 , -1 ,  1 ), # v3
            (  1 , -1 , -1 ), # v6
            (  1 , -1 ,  1 ), # v2

            ( -1 , -1 ,  1 ), # v3
            ( -1 , -1 , -1 ), # v7
            (  1 , -1 , -1 ), # v6

            (  1 ,  1 ,  1 ), # v1
            (  1 , -1 ,  1 ), # v2
            (  1 , -1 , -1 ), # v6

            (  1 ,  1 ,  1 ), # v1
            (  1 , -1 , -1 ), # v6
            (  1 ,  1 , -1 ), # v5

            ( -1 ,  1 ,  1 ), # v0
            ( -1 , -1 , -1 ), # v7
            ( -1 , -1 ,  1 ), # v3

            ( -1 ,  1 ,  1 ), # v0
            ( -1 ,  1 , -1 ), # v4
            ( -1 , -1 , -1 ), # v7
            ], 'float32')
    return varr
```

```python
def render():
    global gCamAng, gCamHeight
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)
    glPolygonMode( GL_FRONT_AND_BACK, GL_LINE )

    glLoadIdentity()
    gluPerspective(45, 1, 1,10)
    gluLookAt(5*np.sin(gCamAng),gCamHeight,5*np.cos(gCamAng), 0,0,0, 0,1,0)

    drawFrame()
    glColor3ub(255, 255, 255)

    # drawCube_glVertex()
    drawCube_glDrawArrays()


def drawCube_glDrawArrays():
    global gVertexArraySeparate
    varr = gVertexArraySeparate
    glEnableClientState(GL_VERTEX_ARRAY)  # Enable it to use vertex array
    glVertexPointer(3, GL_FLOAT, 3*varr.itemsize, varr)
    glDrawArrays(GL_TRIANGLES, 0, int(varr.size/3))
```

```
gVertexArraySeparate = None
def main():
    global gVertexArraySeparate

    if not glfw.init():
        return
    window = glfw.create_window(640,640,'Lecture10', None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)

    gVertexArraySeparate = createVertexArraySeparate()

    while not glfw.window_should_close(window):
        glfw.poll_events()
        render()
        glfw.swap_buffers(window)

    glfw.terminate()

if __name__ == "__main__":
    main()
```

```
def drawFrame():
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([1.,0.,0.]))
    glColor3ub(0, 255, 0)
    glVertex3fv(np.array([0.,0.,0.]))
    glVertex3fv(np.array([0.,1.,0.]))
    glColor3ub(0, 0, 255)
    glVertex3fv(np.array([0.,0.,0]))
    glVertex3fv(np.array([0.,0.,1.]))
    glEnd()


def key_callback(window, key, scancode, action,
mods):
    global gCamAng, gCamHeight
    if action==glfw.PRESS or action==glfw.REPEAT:
        if key==glfw.KEY_1:
            gCamAng += np.radians(-10)
        elif key==glfw.KEY_3:
            gCamAng += np.radians(10)
        elif key==glfw.KEY_2:
            gCamHeight += .1
        elif key==glfw.KEY_W:
            gCamHeight += -.1
```

Quiz #2

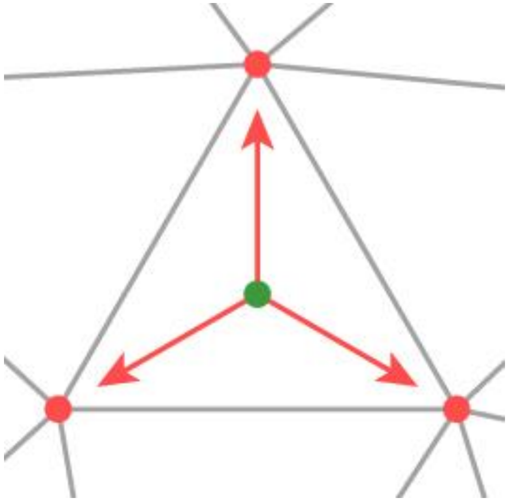
- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

Indexed triangle set

- Store each vertex once
- Each triangle points to its three vertices



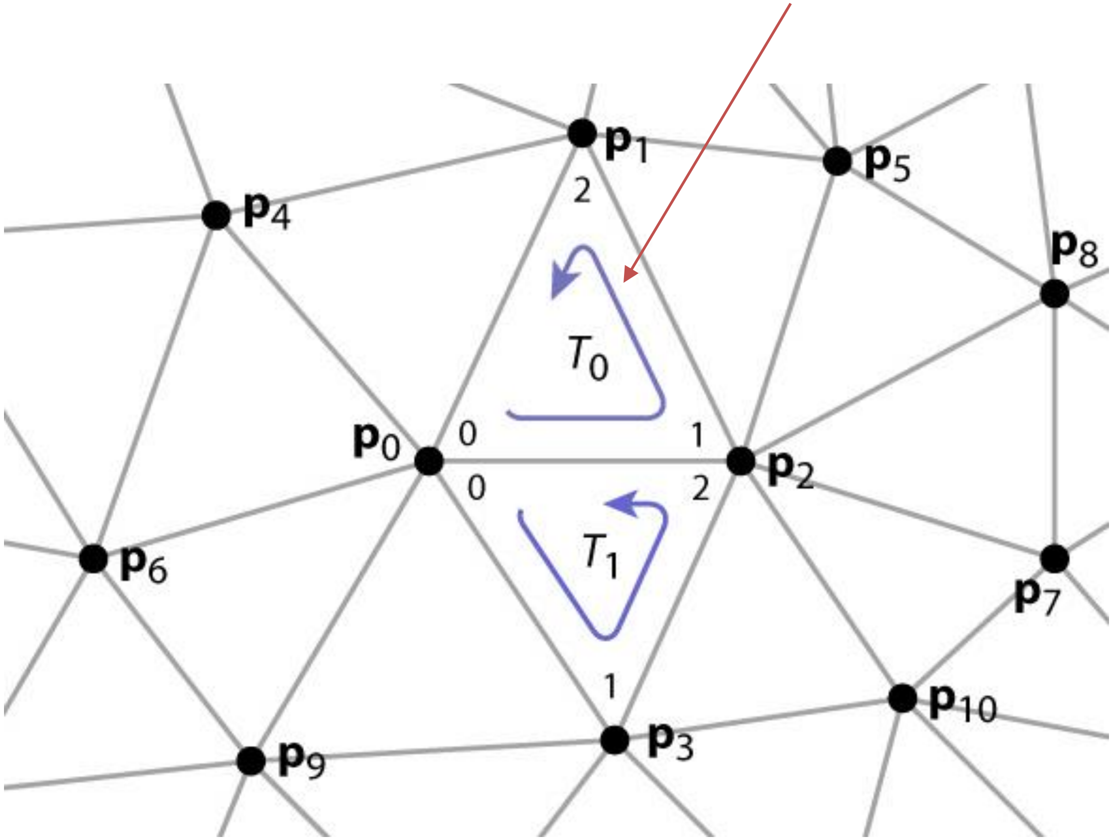
Indexed triangle set

vertex array

verts[0]	x_0, y_0, z_0
verts[1]	x_1, y_1, z_1
	x_2, y_2, z_2
	x_3, y_3, z_3
	⋮

index array

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	⋮


counter-clockwise order

Indexed Triangle Set

- Memory efficient: each vertex position is stored only once.

- Represents topology and geometry separately.

- Finding neighbors is at least well defined.
  - Neighbor triangles share same vertex indices.

Drawing Indexed Triangles using Vertex & Index Array

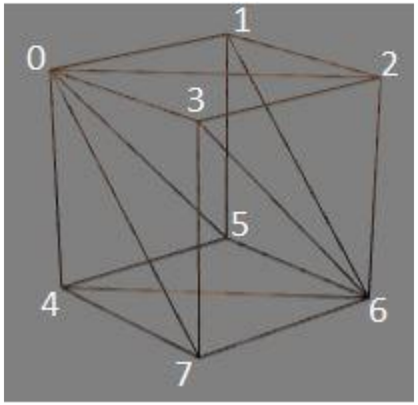
- 1. Create a vertex array & **index array** for your mesh
  - The vertex array **should not have duplicate vertex data**
- 2. Specify “pointer” to this vertex array
  - Same with the separate triangles case
- 3. Render the mesh using the specified “pointer” & **indices of vertices to render**
  - Using **glDrawElements()**

glDrawElements()

- **glDrawElements(mode , count , type , indices)**
- : render primitives from vertex & index array data
  - **mode**: The primitive type to render. GL_POINTS, GL_TRIANGLES, ...
  - **count**: The number of indices to be rendered
  - **type**: The type of the values in **indices**. GL_UNSIGNED_BYTE, GL_UNSIGNED_SHORT, or GL_UNSIGNED_INT
  - **indices**: The pointer to the index array

[Practice] Drawing Indexed Triangles using Vertex & Index Array

```
def createVertexAndIndexArrayIndexed():
    varr = np.array([
            ( -1 ,  1 ,  1 ), # v0
            (  1 ,  1 ,  1 ), # v1
            (  1 , -1 ,  1 ), # v2
            ( -1 , -1 ,  1 ), # v3
            ( -1 ,  1 , -1 ), # v4
            (  1 ,  1 , -1 ), # v5
            (  1 , -1 , -1 ), # v6
            ( -1 , -1 , -1 ), # v7
            ], 'float32')
    iarr = np.array([
            (0,2,1),
            (0,3,2),
            (4,5,6),
            (4,6,7),
            (0,1,5),
            (0,5,4),
            (3,6,2),
            (3,7,6),
            (1,2,6),
            (1,6,5),
            (0,7,3),
            (0,4,7),
            ])
    return varr, iarr
```



vertex index	position
0	(-1 , 1 , 1)
1	(1 , 1 , 1)
2	(1 , -1 , 1)
3	(-1 , -1 , 1)
4	(-1 , 1 , -1)
5	(1 , 1 , -1)
6	(1 , -1 , -1)
7	(-1 , -1 , -1)

```
def render():
    # ...
    drawFrame()
    glColor3ub(255, 255, 255)
    drawCube_glDrawElements()


def drawCube_glDrawElements():
    global gVertexArrayIndexed, gIndexArray
    varr = gVertexArrayIndexed
    iarr = gIndexArray
    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 3*varr.itemsize, varr)
    glDrawElements(GL_TRIANGLES, iarr.size, GL_UNSIGNED_INT, iarr)


# ...
gVertexArrayIndexed = None
gIndexArray = None

def main():
    # ...
    global gVertexArrayIndexed, gIndexArray

    # ...
    gVertexArrayIndexed, gIndexArray = createVertexAndIndexArrayIndexed()

    while not glfw.window_should_close(window):
    # ...
```

Quiz #3

- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

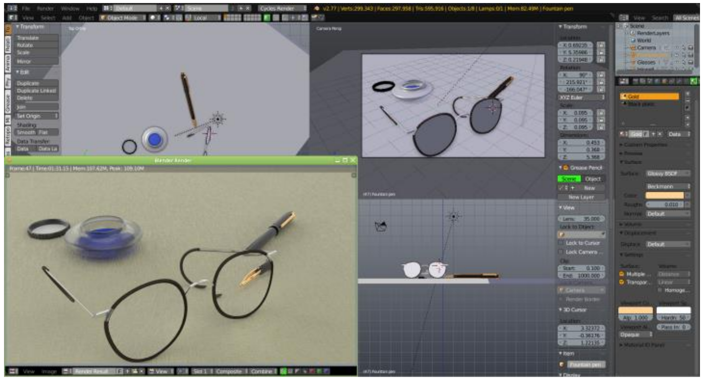
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

- Note that you must submit all quiz answers in the above format to be checked for “attendance”.

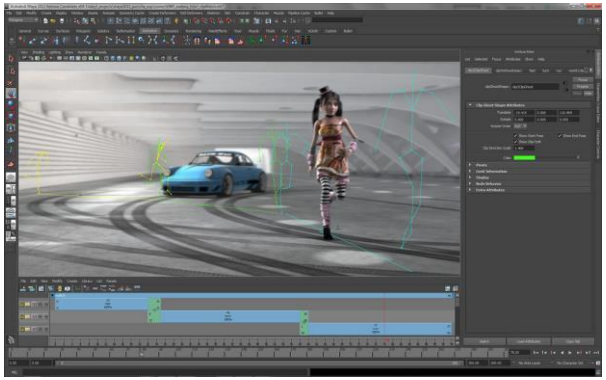
Do we need to hard-code all vertex positions and indices?

- Of course not

- An *object file* or *model file* storing polygon mesh data is usually created using 3D modeling tools.



Blender



Maya

- Applications usually load vertex and index data from an *object file* and draw the object using the loaded data.

3D File Formats

- DXF – AutoCAD
  - Supports 2-D and 3-D; binary
- 3DS – 3DS MAX
  - Flexible; binary
- VRML – Virtual reality modeling language
  - ASCII – Human readable (and writeable)
- OBJ – Wavefront OBJ format
  - ASCII
  - Extremely simple
  - Widely supported

OBJ File Tokens

- File tokens are listed below

# some text

Rest of line is a comment

v *float float float*

A single vertex's geometric position in space

vn *float float float*

A normal

vt *float float*

A texture coordinate

OBJ Face Varieties

f *int int int* ... (vertex only)

or

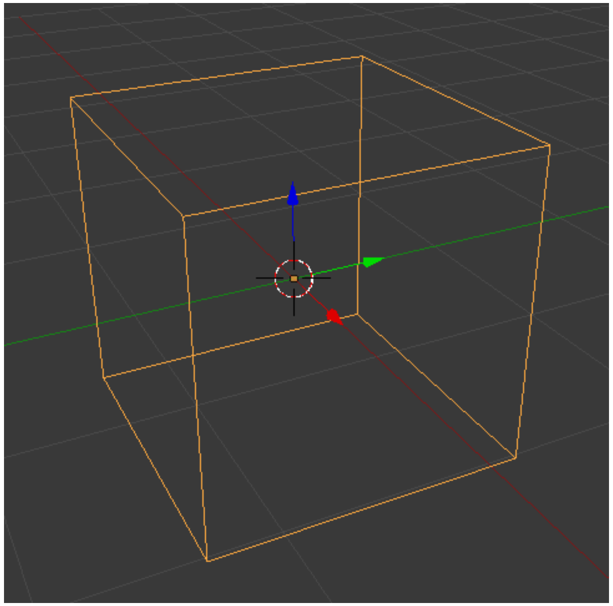
f *int/int int/int int/int* . . . (vertex & texture)

or

f *int/int/int int/int/int int/int/int* ... (vertex, texture, & normal)

- The arguments are **1-based** indices into the arrays
  - Vertex positions
  - Texture coordinates
  - Normals, respectively

An OBJ Example



```
# A simple cube
v 1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 1.000000
v -1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
```

[Practice] Manipulate an OBJ file with Blender

- Blender
  - https://www.blender.org/
  - Open source
  - Full 3D modeling/rendering/animation tool

- Install & launch Blender

- Reference for basic mouse actions in Blender
  - https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/3D_View_Windows#Changing_Your_Viewpoint,_Part_One

[Practice] Manipulate an OBJ file with Blender

- Save the obj example in the prev. page as cube.obj (using a text editor)

- Click the "start-up" cube object in the Blender and press Del key to delete it.

- Import cube.obj into Blender (File-Import)
  - Press ‘z’ to render in wireframe mode

- Edit cube.obj somehow (using a text editor)

- Delete the loaded cube and re-import cube.obj into Blender again

- Press ‘tab’ to switch to *Edit mode*

[Practice] Manipulate an OBJ file with Blender

- Click to select a vertex and click "move" icon from the left icons (or press 'G')

- Move the selected vertex by dragging red/blue/green arrows

- Export this mesh to cube.obj (File – Export)

- Open cube.obj using a text editor and check what is changed

- Reference for *Edit mode* in Blender
  - https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Mesh_Edit_Mode

- Reference for *Object mode* in Blender
  - https://en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/Object_Mode

OBJ Sources

- https://free3d.com/
- https://www.cgtrader.com/free-3d-models

- You can download any .obj model files from these sites and open them in Blender.

- OBJ file format is very popular:
  - Most modeling programs will export OBJ files
  - Most rendering packages will read in OBJ files

Next Time

- Lab in this week:
  - Lab assignment 6

- Next lecture:
  - 7 - Lighting & Shading

- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Jinxiang Chai, Texas A&M Univ., http://faculty.cs.tamu.edu/jchai/csce441_2016spring/lectures.html
  - Prof. Taesoo Kwon, Hanyang Univ., http://calab.hanyang.ac.kr/cgi-bin/cg.cgi
  - Prof. Steve Marschner, Cornell Univ., http://www.cs.cornell.edu/courses/cs4620/2014fa/index.shtml