
Computer Graphics

7 - Hierarchical Modeling, Mesh

Yoonsang Lee
Spring 2020

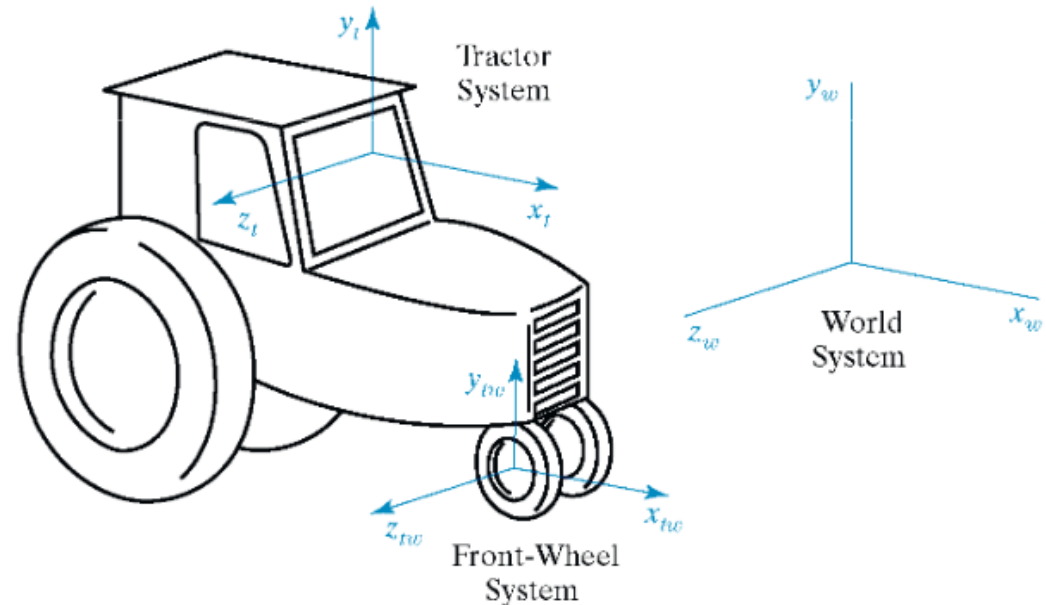
Topics Covered

- Hierarchical Modeling
 - Concept of Hierarchical Modeling
 - OpenGL Matrix Stack
- Mesh
 - Polygon mesh & triangle mesh
 - Representations for triangle meshes
 - OpenGL vertex array
 - OBJ file format

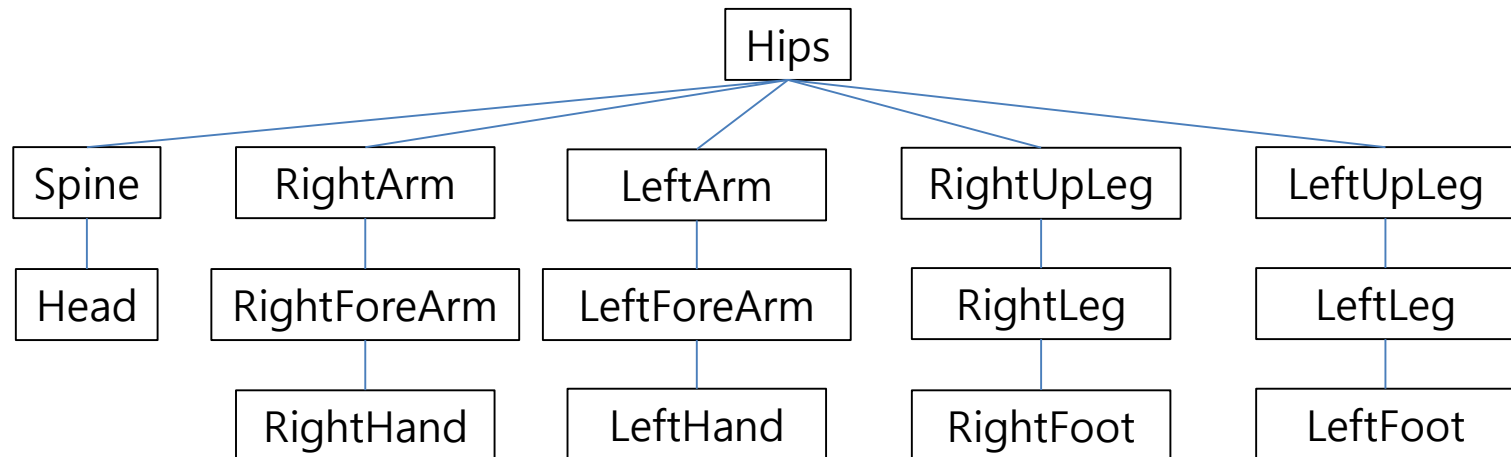
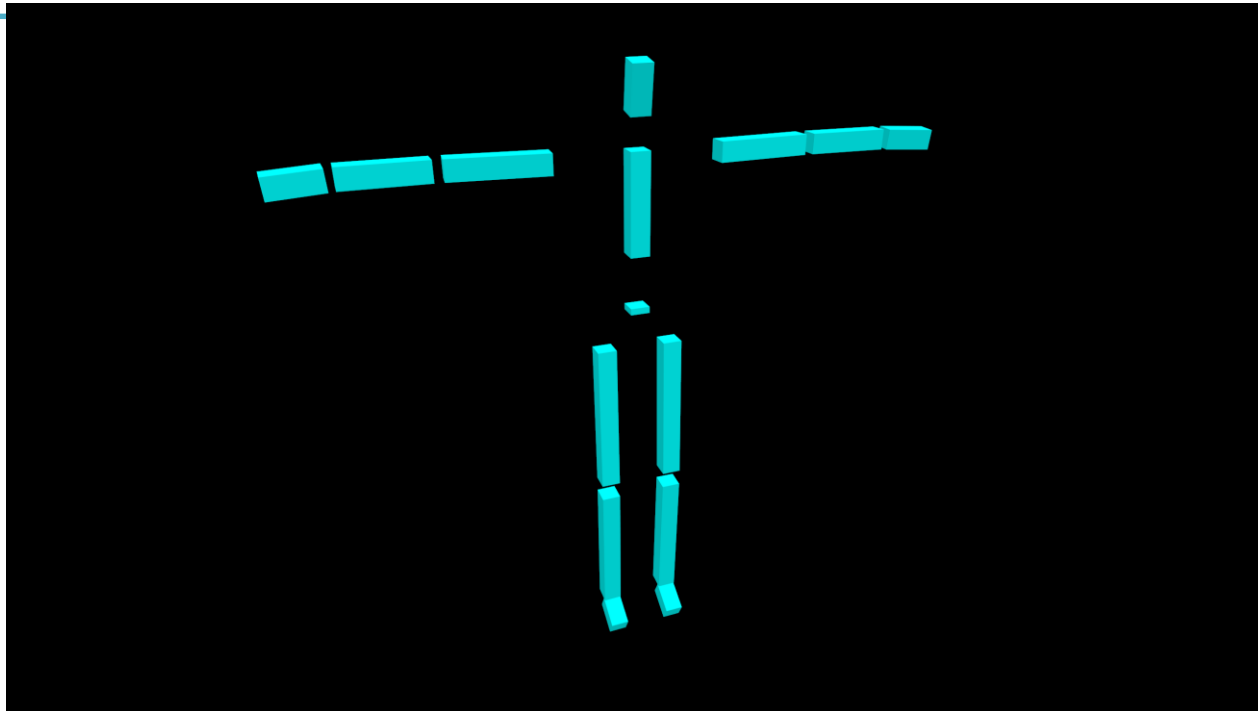
Hierarchical Modeling

Hierarchical Modeling

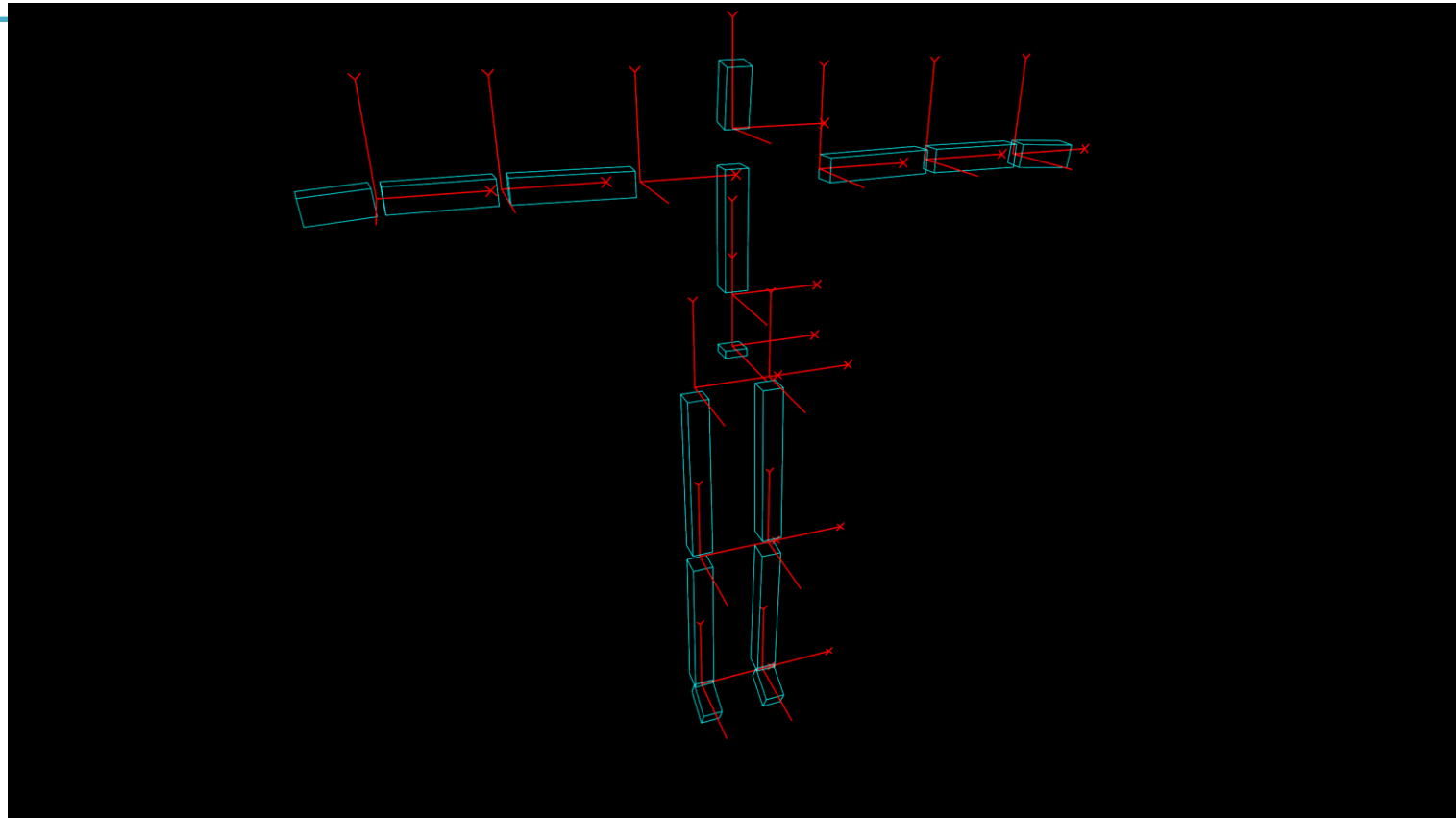
- Nesting the description of subparts (child parts) into another part (parent part) to form a tree structure
- Each part has its own reference frame (local frame).
- Each part's movement is described w.r.t. its parent's reference frame.



Another Example - Human Figure

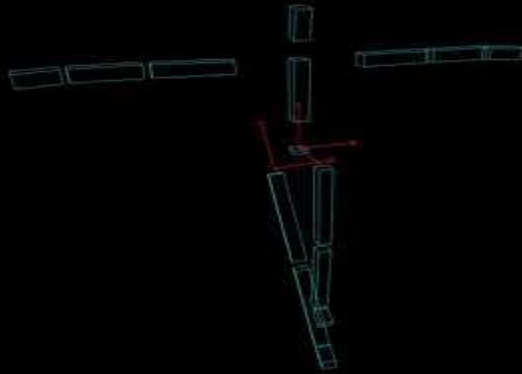


Human Figure - Frames

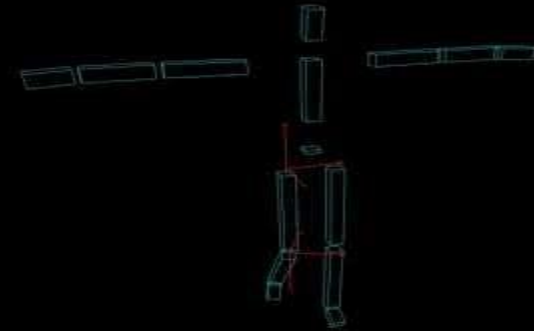


- Each part has its own reference frame (local frame).

Human Figure - Movement of rhip & rknee



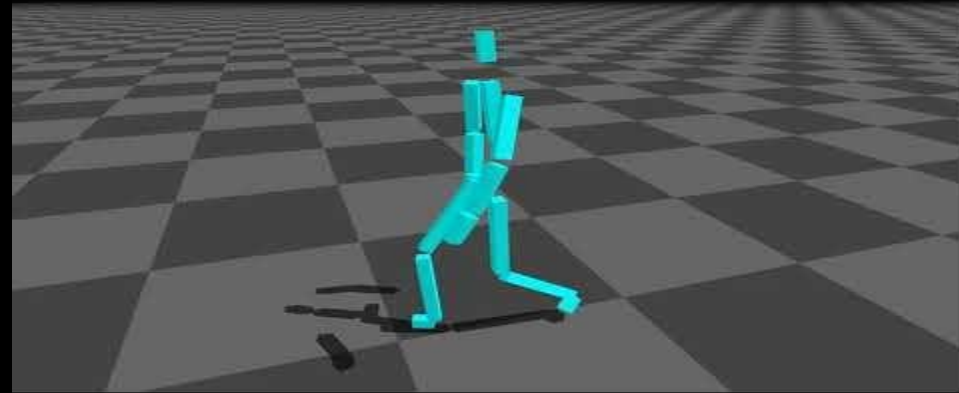
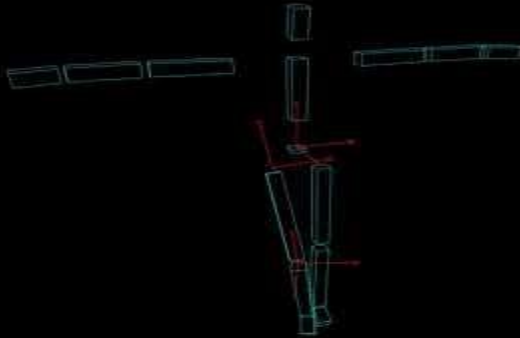
<https://youtu.be/Q7lhvMkCSCg>



<https://youtu.be/Q5R8WGUwpFU>

- Each part's movement is described w.r.t. its parent's reference frame.
 - Each part has its own transformation w.r.t. parent part's frame
 - "Grouping"

Human Figure - Movement of more joints



<https://youtu.be/9dz8bvVK9zc>

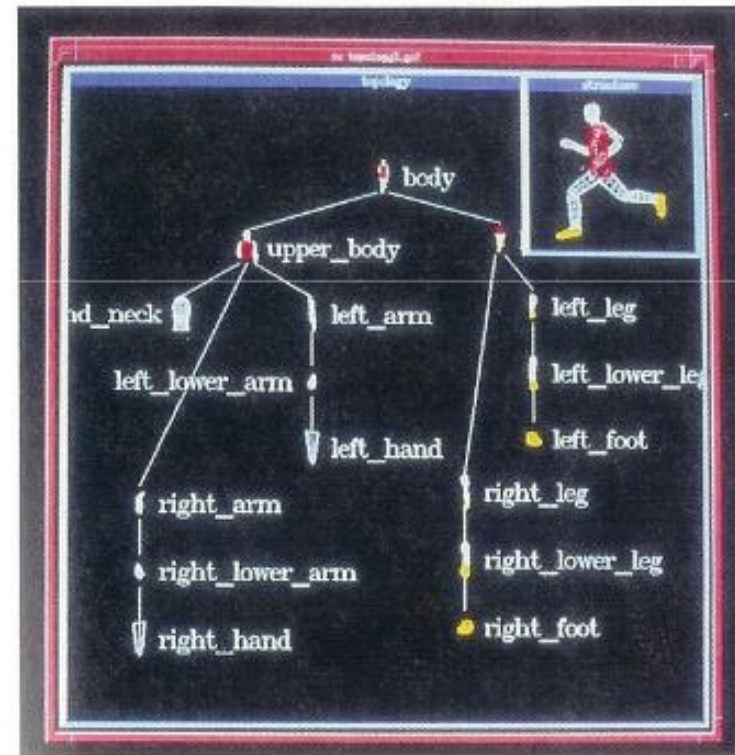
<https://youtu.be/PEhyWI8LGBY>

- Each part's movement is described w.r.t. its parent's reference frame.
 - Each part has its own transformation w.r.t. parent part's frame
 - "Grouping"

The Scene Graph (tree)

- A name given to various kinds of graph structures (nodes connected together) used to represent scenes
- Simplest form: tree
 - just saw this
 - every node has one parent

- Each node has its own transformation w.r.t. parent node's frame

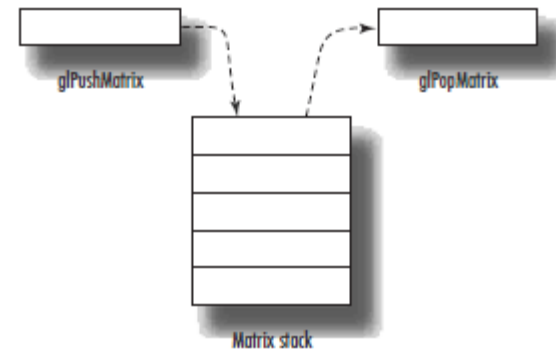


Hierarchical Modeling in OpenGL

- OpenGL provides a useful way of drawing objects in the hierarchical structure (scene graph)
- → **Matrix stack**

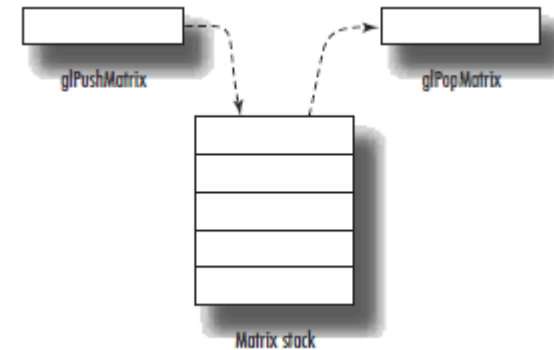
OpenGL Matrix Stack

- A *stack* for transformation matrices
 - Last In First Outs
- You can **save the current transformation matrix** and then **restore** it after some objects have been drawn
- Useful for traversing hierarchical data structures (i.e. scene graph or tree)

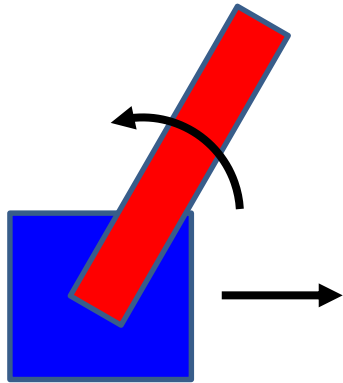


OpenGL Matrix Stack

- **glPushMatrix()**
 - Pushes **the current matrix** onto the stack.
- **glPopMatrix()**
 - Pops the matrix off the stack.
- The **current matrix** is the matrix **on the top of the stack!**
- Keep in mind that the **numbers of glPushMatrix() calls and glPopMatrix() calls must be the same.**



A simple example



Bold text is the **current transformation matrix** (the one at the top of the matrix stack)

- Start with identity matrix

I

- `glPushMatrix()`

I
I
- `glTranslate(T)` # to translate base

T
I
- `glPushMatrix()`

T
T
I
- `glScale(S)` # to draw base

TS
T
I
- Draw a box
- `glPopMatrix()`

T
I
- `glPushMatrix()`

T
T
I
- `glRotate(R)` # to rotate arm

TR
T
I
- `glPushMatrix()`

TR
TR
T
I
- `glScale(U)` # to draw arm

TRU
TR
T
I
- Draw a box
- `glPopMatrix()`

TR
T
I
- `glPopMatrix()`

T
I
- `glPopMatrix()`

I

[Practice] Matrix Stack

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

gCamAng = 0

def render(camAng):
    # enable depth test (we'll see
    details later)
    glClear(GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    glLoadIdentity()

    # projection transformation
    glOrtho(-1,1, -1,1, -1,1)

    # viewing transformation
    gluLookAt(.1*np.sin(camAng), .1,
    .1*np.cos(camAng), 0,0,0, 0,1,0)

    drawFrame()

    t = glfw.get_time()
```

```
# modeling transformation

# blue base transformation
glPushMatrix()
glTranslatef(np.sin(t), 0, 0)

# blue base drawing
glPushMatrix()
glScalef(.2, .2, .2)
glColor3ub(0, 0, 255)
drawBox()
glPopMatrix()

# red arm transformation
glPushMatrix()
glRotatef(t*(180/np.pi), 0, 0, 1)
glTranslatef(.5, 0, .01)

# red arm drawing
glPushMatrix()
glScalef(.5, .1, .1)
glColor3ub(255, 0, 0)
drawBox()
glPopMatrix()

glPopMatrix()
glPopMatrix()
```

```

def drawBox():
    glBegin(GL_QUADS)
    glVertex3fv(np.array([1,1,0.]))
    glVertex3fv(np.array([-1,1,0.]))
    glVertex3fv(np.array([-1,-1,0.]))
    glVertex3fv(np.array([1,-1,0.]))
    glEnd()

def drawFrame():
    # draw coordinate: x in red, y in
green, z in blue
    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, gComposedM
    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)

def main():
    if not glfw.init():
        return
    window =
glfw.create_window(640,640,"Hierarchy",
None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)
    glfw.swap_interval(1)

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

    glfw.terminate()

if __name__ == "__main__":
    main()

```

Quiz #1

- Go to <https://www.slido.com/>
- Join #cg-hyu
- 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”.

OpenGL Matrix Stack Types

- Actually, OpenGL maintains four different types of matrix stacks:
- **Modelview matrix stack (GL_MODELVIEW)**
 - Stores model view matrices.
 - This is the default type (what we've just used)
- **Projection matrix stack (GL_PROJECTION)**
 - Stores projection matrices
- **Texture matrix stack (GL_TEXTURE)**
 - Stores transformation matrices to adjust texture coordinates. Mostly used to implement texture projection (like an image projected by a beam projector)
- **Color matrix stack (GL_COLOR)**
 - Rarely used. Just ignore it.
- You can switch the current matrix stack type using `glMatrixMode()`
 - e.g. `glMatrixMode(GL_PROJECTION)` to select the projection matrix stack

OpenGL Matrix Stack Types

- A common guide is something like:

```
/* Projection Transformation */
glMatrixMode(GL_PROJECTION); /* specify the projection matrix */
glLoadIdentity();           /* initialize current value to identity */
gluPerspective(...);        /* or gluOrtho(...) for orthographic */
                             /* or glFrustum(...), also for perspective */

/* Viewing And Modelling Transformation */
glMatrixMode(GL_MODELVIEW); /* specify the modelview matrix */
glLoadIdentity();          /* initialize current value to identity */
gluLookAt(...);            /* specify the viewing transformation */

glTranslate(...);          /* various modelling transformations */
glScale(...);
glRotate(...);
...
```

- **Projection transformation** functions (`gluPerspective()`, `gluOrtho()`, ...) should be called with **`glMatrixMode(GL_PROJECTION)`**.
- **Modeling & viewing transformation** functions (`gluLookAt()`, `glTranslate()`, ...) should be called with **`glMatrixMode(GL_MODELVIEW)`**.
- Otherwise, you'll get wrong lighting results.

[Practice] With Correct Matrix Stack Types

```
def render(camAng):
    # enable depth test (we'll see
    details later)
    glClear(GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()

    # projection transformation
    glOrtho(-1,1, -1,1, -1,1)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

    # viewing transformation
    gluLookAt(.1*np.sin(camAng), .1,
    .1*np.cos(camAng), 0,0,0, 0,1,0)

    drawFrame()
    t = glfw.get_time()
```

```
# modeling transformation

# blue base transformation
glPushMatrix()
glTranslatef(np.sin(t), 0, 0)

# blue base drawing
glPushMatrix()
glScalef(.2, .2, .2)
glColor3ub(0, 0, 255)
drawBox()
glPopMatrix()

# red arm transformation
glPushMatrix()
glRotatef(t*(180/np.pi), 0, 0, 1)
glTranslatef(.5, 0, .01)

# red arm drawing
glPushMatrix()
glScalef(.5, .1, .1)
glColor3ub(255, 0, 0)
drawBox()
glPopMatrix()

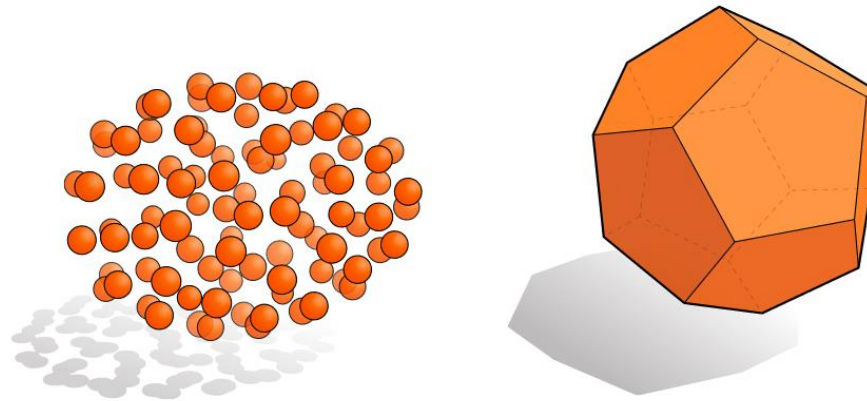
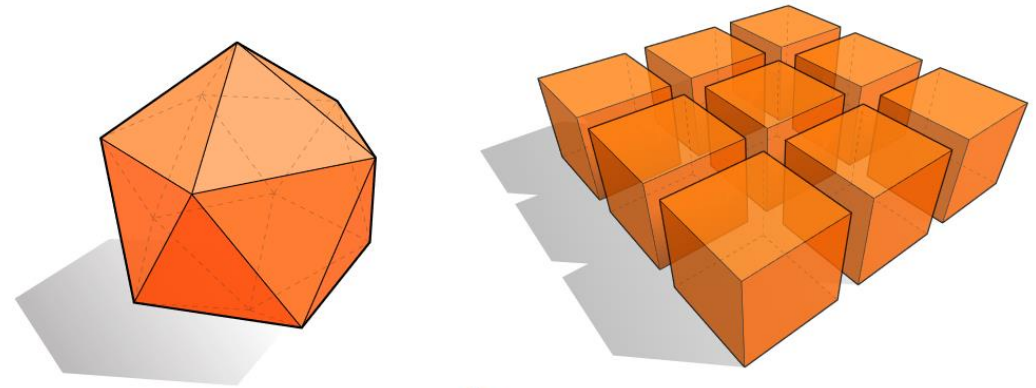
glPopMatrix()
glPopMatrix()
```

Mesh

Many ways to digitally encode geometry

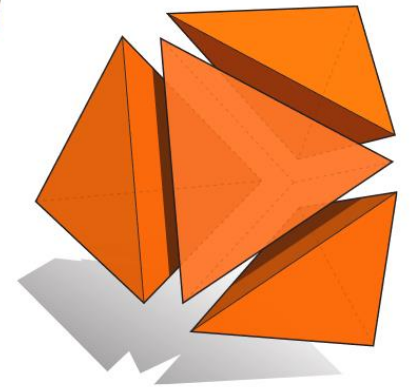
■ EXPLICIT

- point cloud
- polygon mesh
- subdivision, NURBS
- L-systems
- ...



■ IMPLICIT

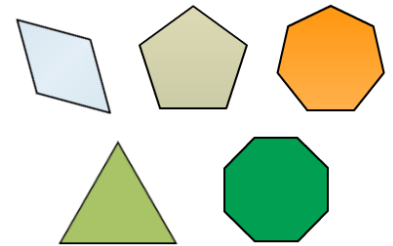
- level set
- algebraic surface
- ...



■ Each choice best suited to a different task/type of geometry

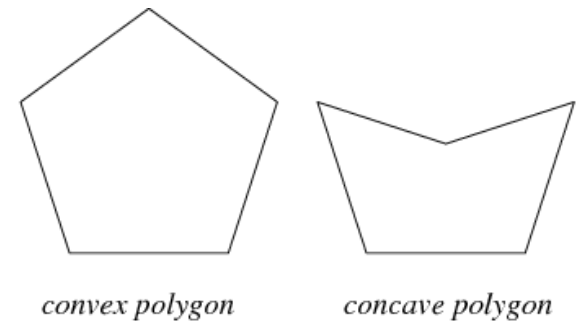
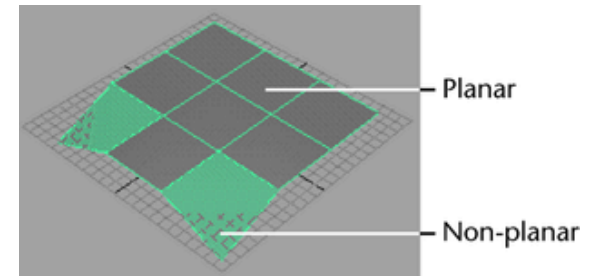
The Most Popular One : Polygon Mesh

- 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)



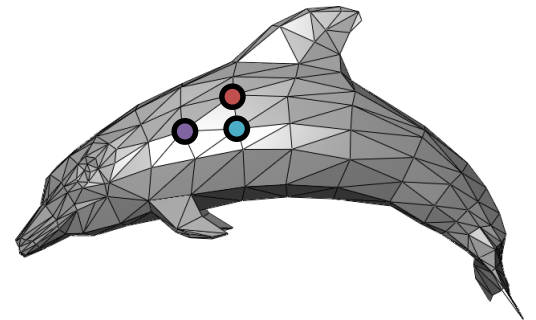
Triangle Mesh

- 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

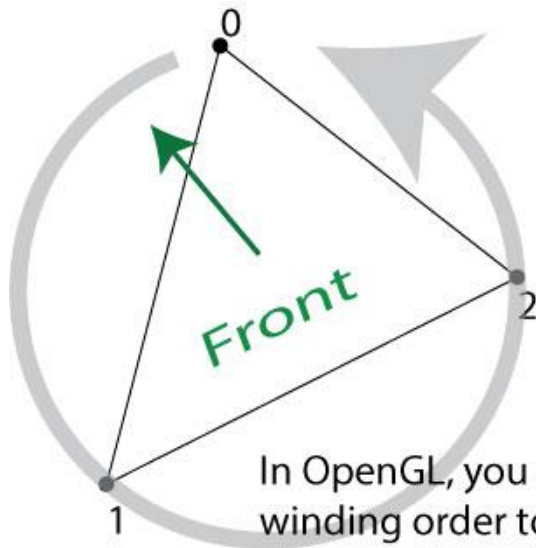
- It's about how to store
 - vertex positions
 - relationship between vertices (to make triangles)
- on memory.
- We'll see
 - Separate triangles
 - Indexed triangle set



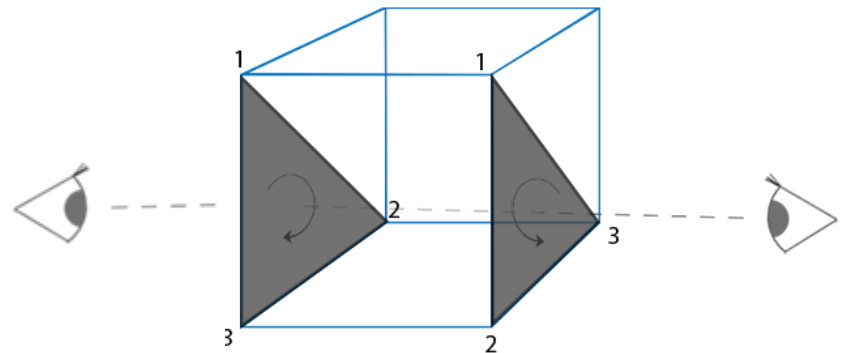
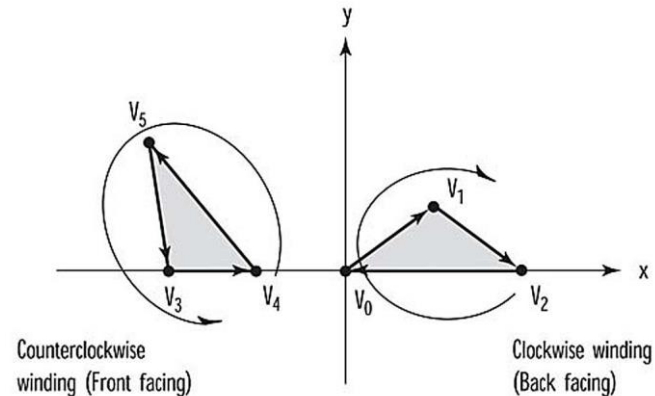
Vertex Winding Order

- 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

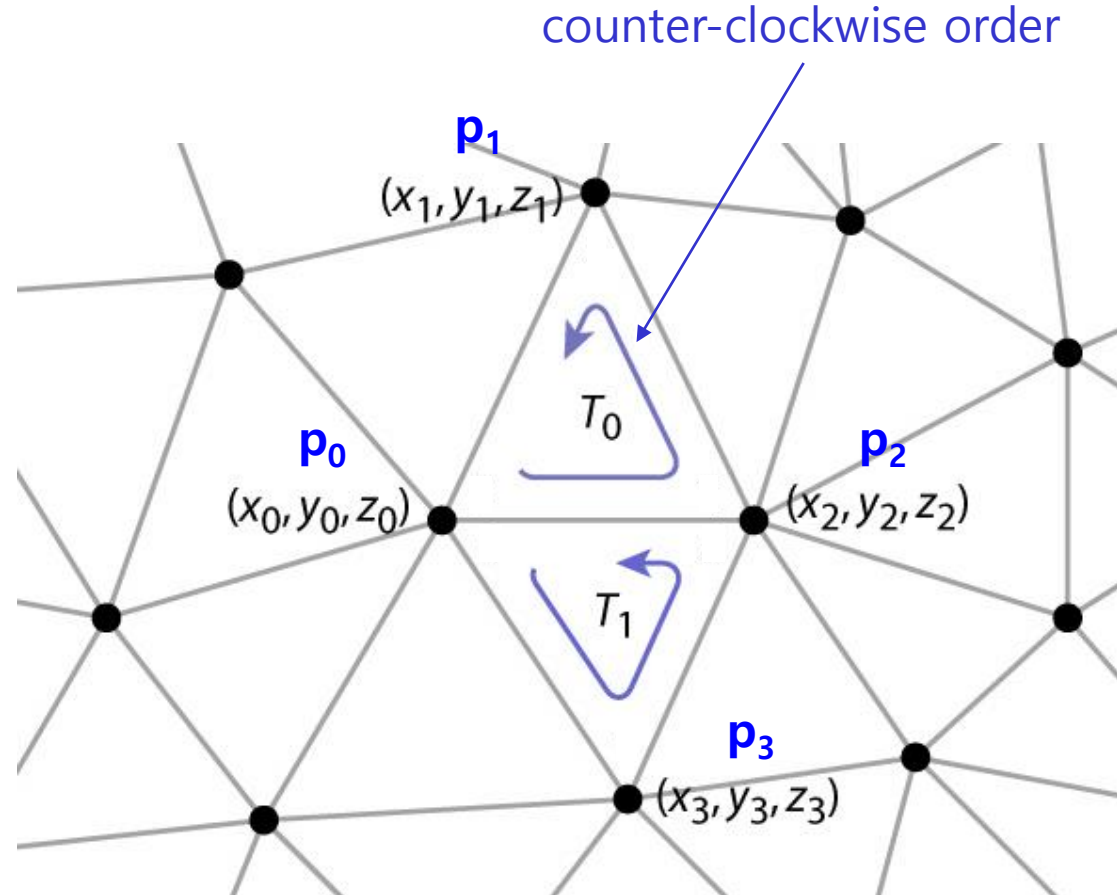


In OpenGL, you can use the winding order to define inside and outside surfaces of solids



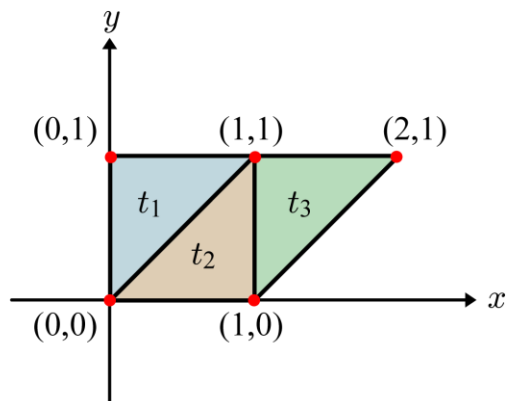
Separate triangles

	[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
	⋮	⋮	⋮



Separate Triangles

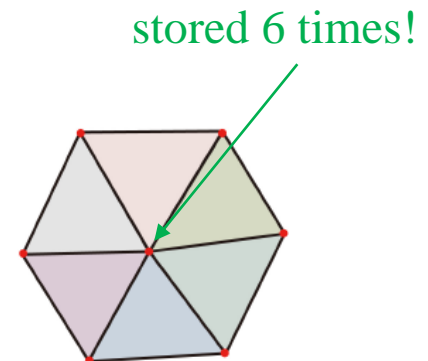
- Various problems
 - Wastes space
 - Cracks due to roundoff
 - Difficulty of finding neighbors



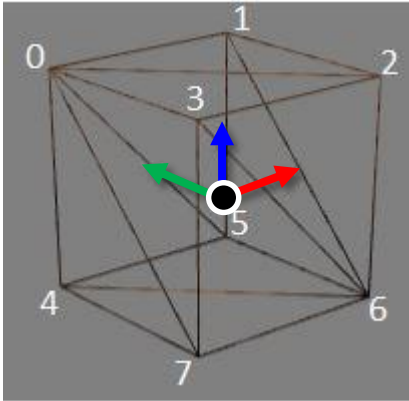
vertex buffer

(0,1)	t ₁
(0,0)	
(1,1)	t ₂
(0,0)	
(1,0)	t ₃
(1,1)	
(1,1)	
(1,0)	
(2,1)	

(1,1) is stored 3 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

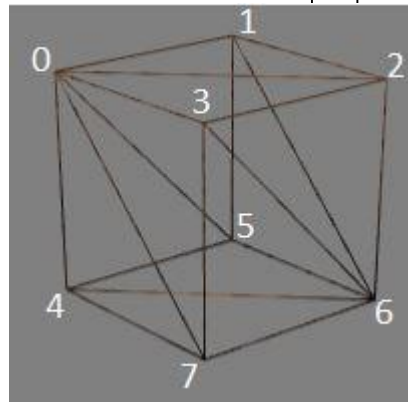
    glVertex3f( -1 , 1 , 1 ) # v0
    glVertex3f( -1 , -1 , 1 ) # v3
    glVertex3f( 1 , -1 , 1 ) # v2

    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.

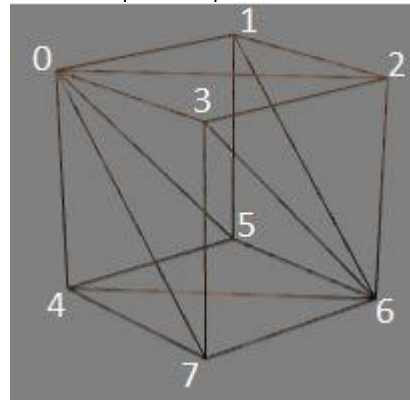
def createVertexArraySeparate():
    varr = np.array([
        (-1, 1, 1), # v0
        (1, -1, 1), # v2
        (1, 1, 1), # v1

        (-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
```

```

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

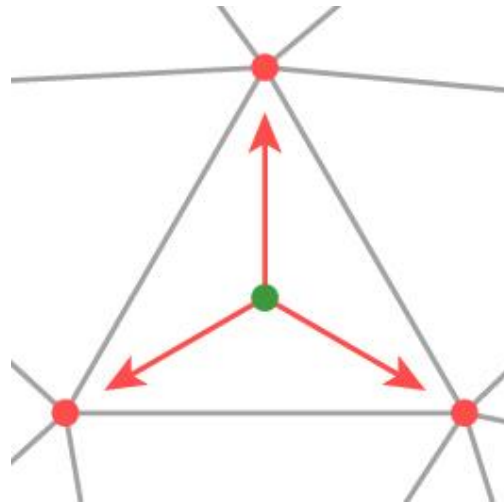
- Go to <https://www.slido.com/>
- Join #cg-hyu
- Click “Polls”

- 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



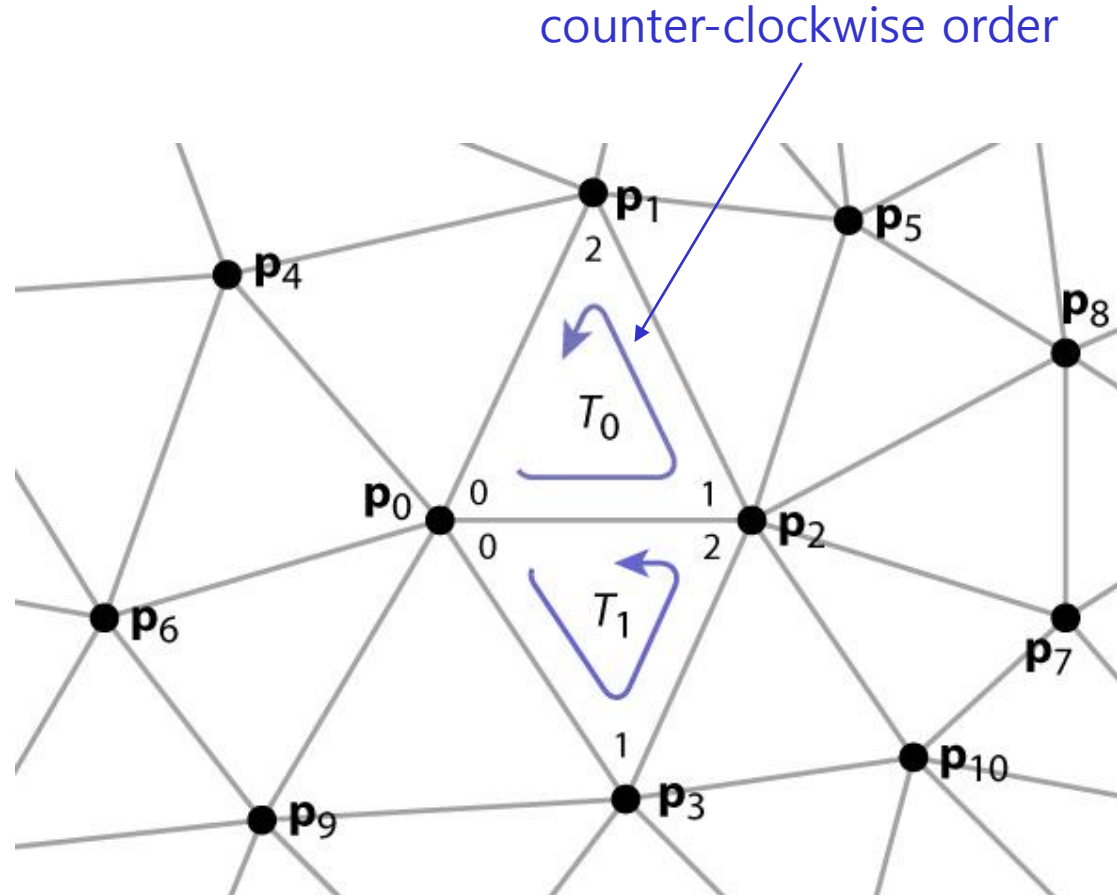
Indexed triangle set

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
	\vdots

index array

tInd[0]	0, 2, 1
tInd[1]	0, 3, 2
	\vdots



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

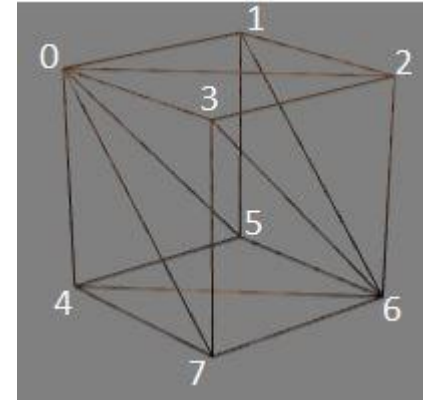
- 1. Create a vertex array & **index array** for your mesh
 - The vertex array **should not have duplicate vertex data**
- 2. Specify “pointer” to this vertex array
 - Same with the separate triangles case
- 3. Render the mesh using the specified “pointer” & **indices of vertices to render**
 - Using `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 glVertexArrayIndexed, glIndexArray
    varr = glVertexArrayIndexed
    iarr = glIndexArray
    glEnableClientState(GL_VERTEX_ARRAY)
    glVertexPointer(3, GL_FLOAT, 3*varr.itemsize, varr)
    glDrawElements(GL_TRIANGLES, iarr.size, GL_UNSIGNED_INT, iarr)

# ...
glVertexArrayIndexed = None
glIndexArray = None

def main():
    # ...
    global glVertexArrayIndexed, glIndexArray

    # ...
    glVertexArrayIndexed, glIndexArray = createVertexAndIndexArrayIndexed()

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

```

Quiz #3

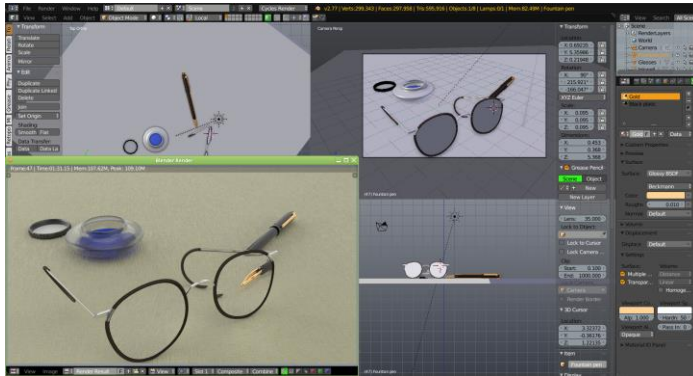
- Go to <https://www.slido.com/>
- Join #cg-hyu
- Click “Polls”

- 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

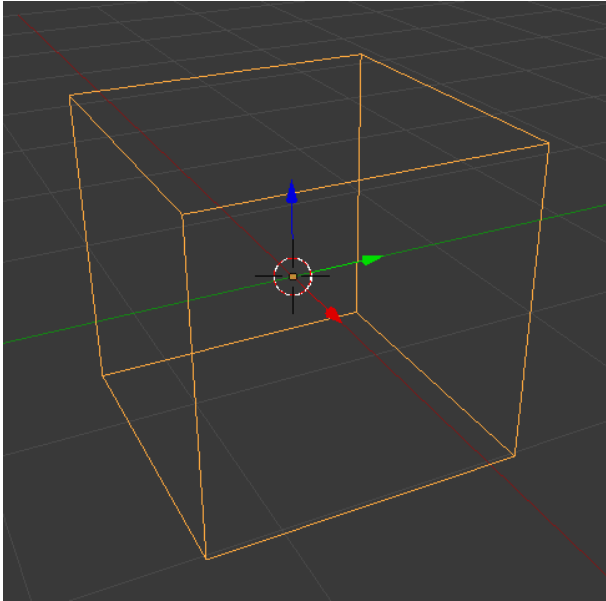
f int/int int/int int/int ... (vertex & texture)

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)
- Right 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

- Right click to select a vertex and move it 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 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 7

- Next lecture:
 - 8 - Lighting & Shading

- Acknowledgement: Some materials come from the lecture slides of
 - Prof. Jehee Lee, SNU, http://mrl.snu.ac.kr/courses/CourseGraphics/index_2017spring.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>
 - Prof. Kayvon Fatahalian and Keenan Crane, CMU, <http://15462.courses.cs.cmu.edu/fall2015/>