Computer Graphics

T5 - Mesh

Yoonsang Lee and Taesoo Kwon

Spring 2019

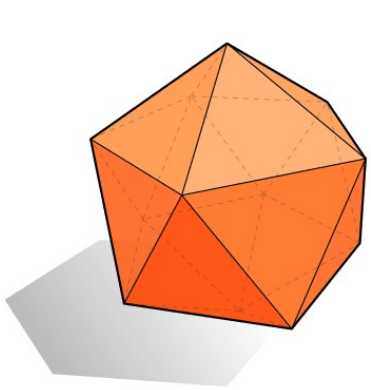
Topics Covered

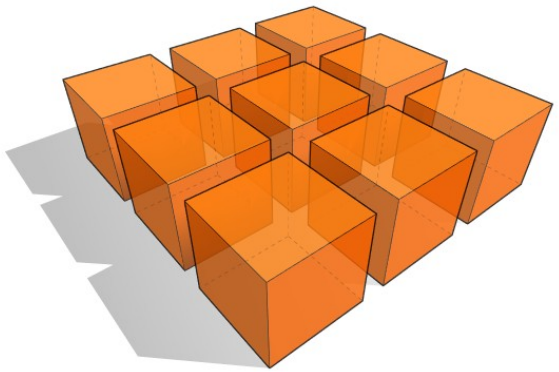
- Mesh
  - Polygon mesh & triangle mesh
  - Representations for triangle meshes
  - OpenGL vertex array
  - OBJ file

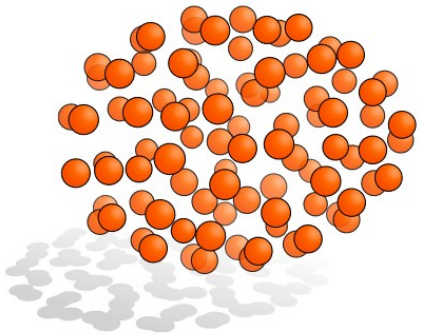
Mesh

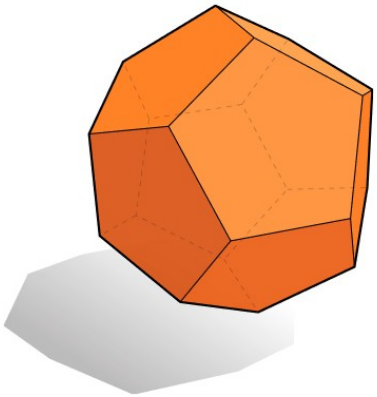
Many ways to digitally encode geometry

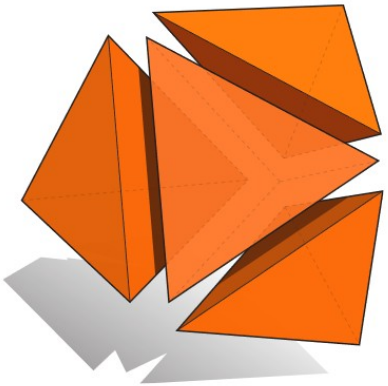
- **EXPLICIT**
  - point cloud
  - polygon mesh
  - subdivision, NURBS
  - L-systems
  - ...
- **IMPLICIT**
  - level set
  - algebraic surface
  - ...
- **Each choice best suited to a different task/type of geometry**





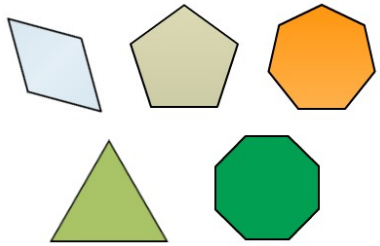






The Most Popular One : Polygon Mesh

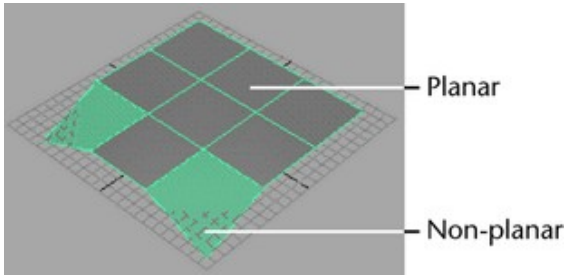
- Because this can model any arbitrary complex shapes with relatively simple representations and can be rendered fast.

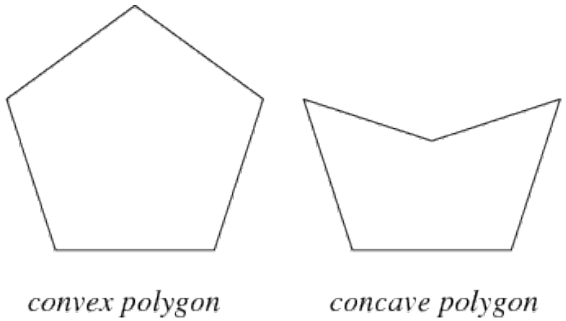


- **Polygon**: a “closed” shape with straight sides
- **Polygon mesh**: a bunch of polygons in 3D space that are connected together to form a surface
  - Usually use *triangles* or *quads* (4-sided polygon)

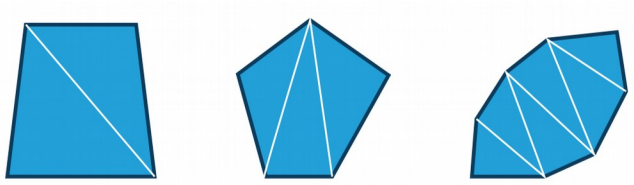
Triangle Mesh

- 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 decomposed into multiple triangles.



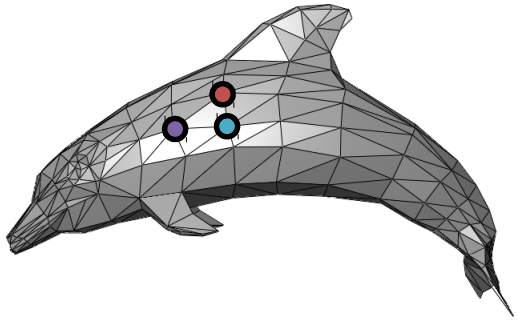
Triangle Mesh

- That’s why modern GPUs ONLY draw “triangles”
  - At the lowest level, GPUs draws everything as a set of triangles.
  - Modern OpenGL API does not support GL_QUAD or GL_POLYGON.
- On the other hand, in 3D modeling softwares such as Maya and Blender, quads are often preferred.
  - Better smoothing operation (*subdivision*) results
  - Better deformation when animating a *skinned model*
  - Easier to understand topology of a model (from rows and columns of quads)
- But those quad models are still rendered using triangles

Representation for Triangle Mesh

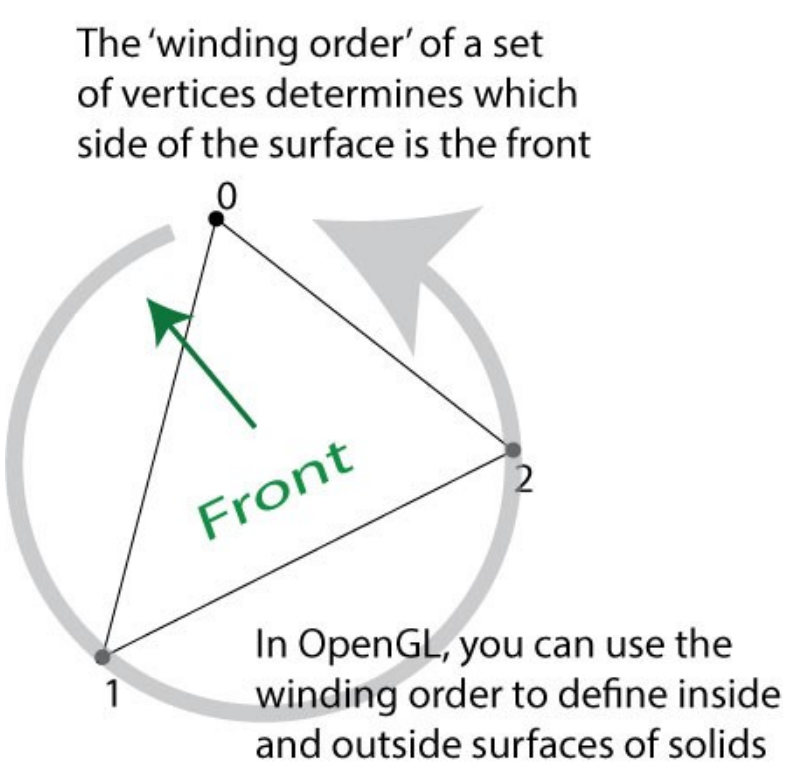
- How can we store
  - vertex positions
  - connectivity (to make triangles)

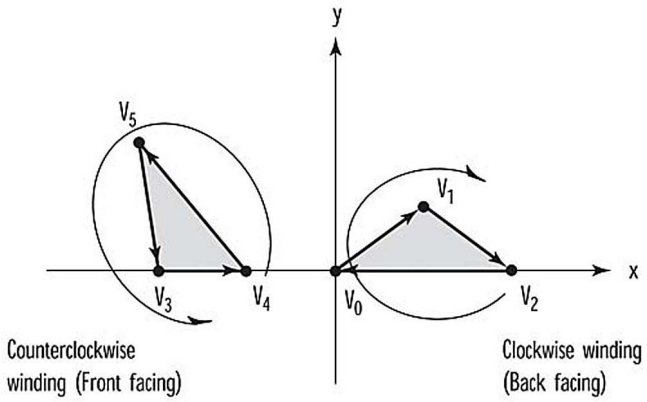
  on memory

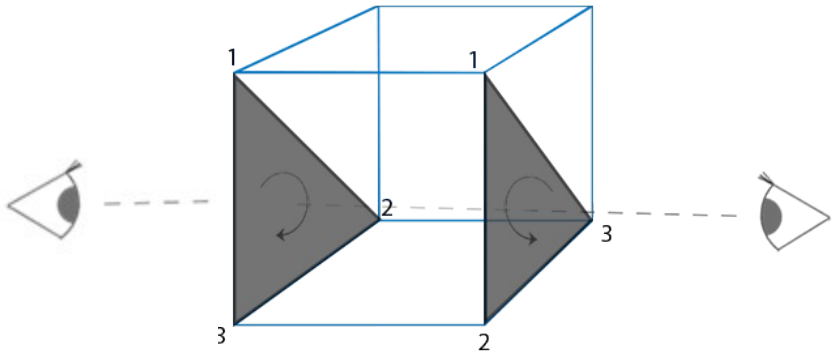


Vertex Winding Order

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







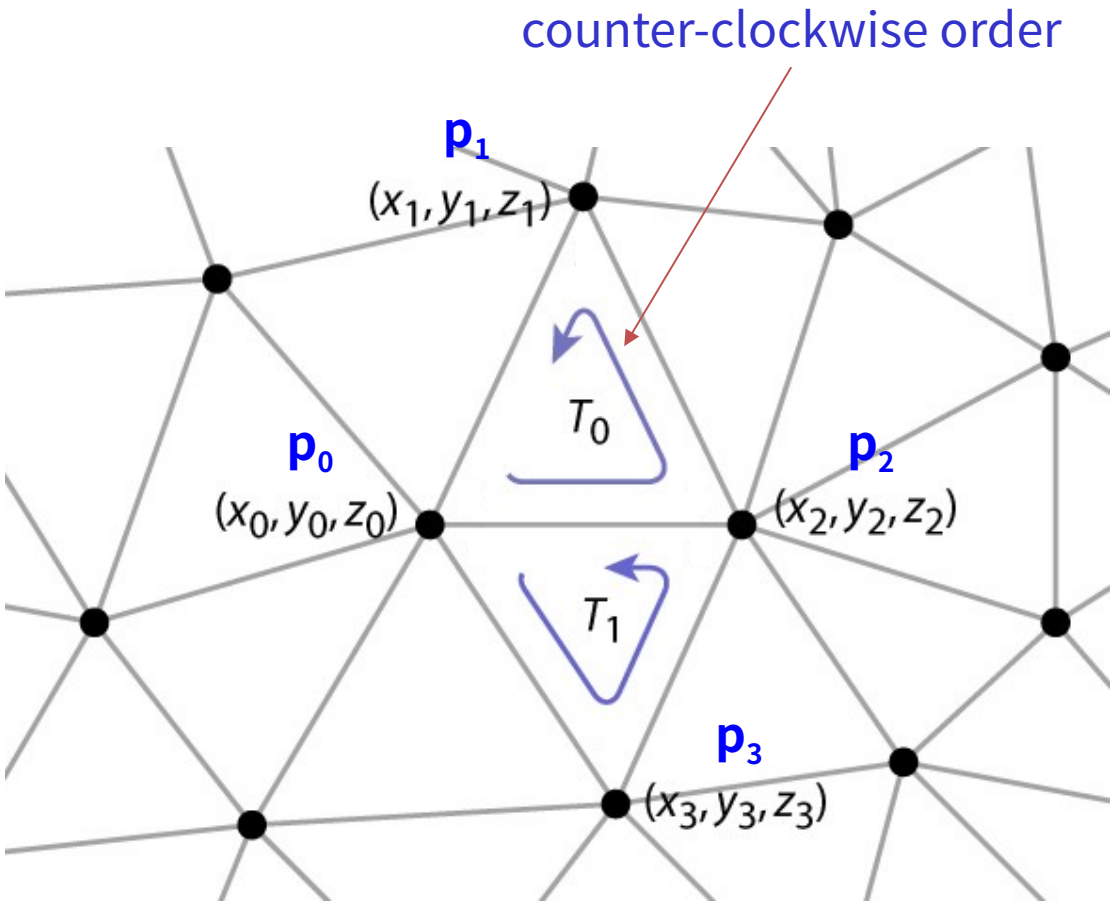
Representations for triangle meshes

- Separate triangles
- Indexed triangle set
  - shared vertices
- Triangle strips and triangle fans
  - compact representations for
  - efficiency
- Triangle-neighbor data structure
  - supports adjacency queries
- Winged-edge data structure
  - supports general polygon meshes

not covered in this practice class

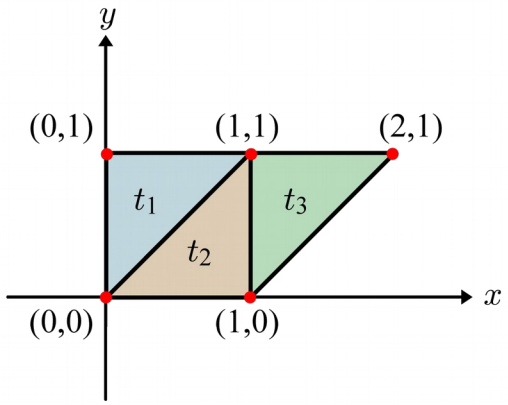
Separate triangles

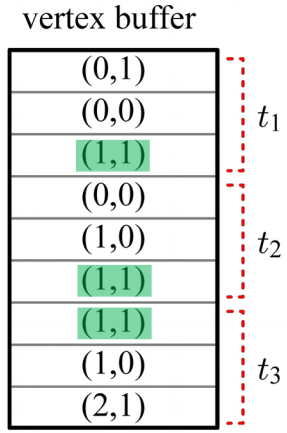
	[0]	[1]	[2]
tris[0]	x_0, y_0, z_0	x_2, y_2, z_2	x_1, y_1, z_1
tris[1]	x_0, y_0, z_0	x_3, y_3, z_3	x_2, y_2, z_2
	⋮	⋮	⋮



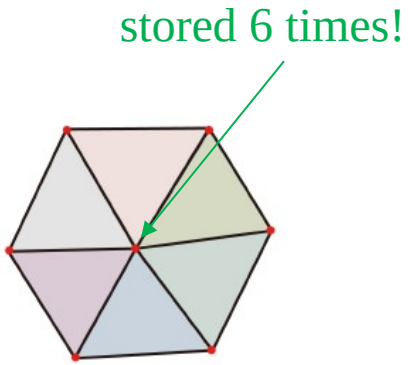
Separate Triangles

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

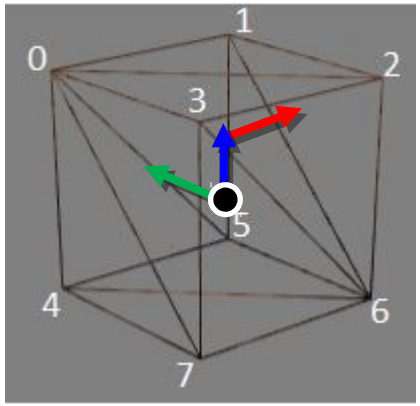




(1,1) is stored 3 times!



Example: a cube of length 2

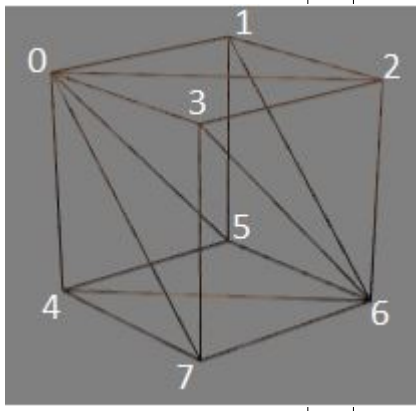


vertex index	position
0	(-1 , 1 , 1)
1	(1 , 1 , 1)
2	(1 , -1 , 1)
3	(-1 , -1 , 1)
4	(-1 , 1 , -1)
5	(1 , 1 , -1)
6	(1 , -1 , -1)
7	(-1 , -1 , -1)

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 efficient method to draw polygons: *Vertex array*

Vertex array: an array containing vertex data such as vertex positions, normals, texture coordinates and colors

By using a vertex array, you can draw a mesh by calling a single OpenGL function (instead of a number of glVertex*() calls!)

→ Faster than glVertex* function calls, but ...

(VBOs are even faster.)

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

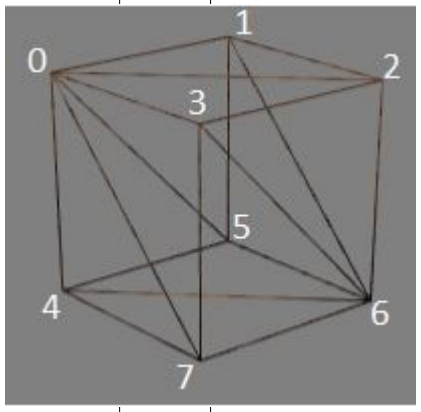
glVertexPointer() & glDrawArrays()

- **glVertexPointer(size, type, stride, pointer)**
- : specifies the location and data format of an array of vertex coordinates
  - **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 number of bytes to 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 vertex array data
  - **mode**: The primitive type to render. GL_POINTS, GL_TRIANGLES, ...
  - **first**: The starting index in the array “enabled” by glVertexPointer()
  - **count**: The number of vertices to be

[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
```

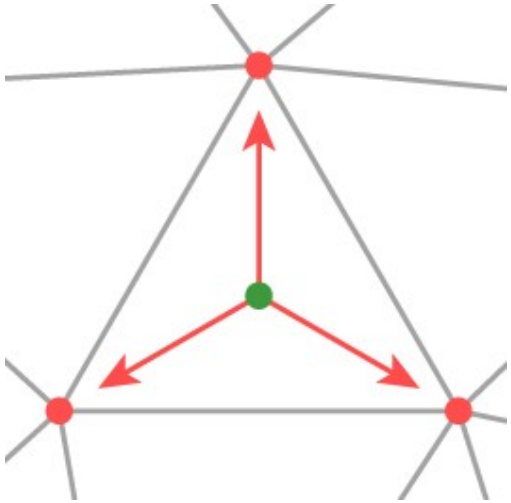
```
def render():
    global gCamAng, gCamHeight
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)
    glPolygonMode( GL_FRONT_AND_BACK, GL_LINE )

    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
```

Indexed triangle set

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



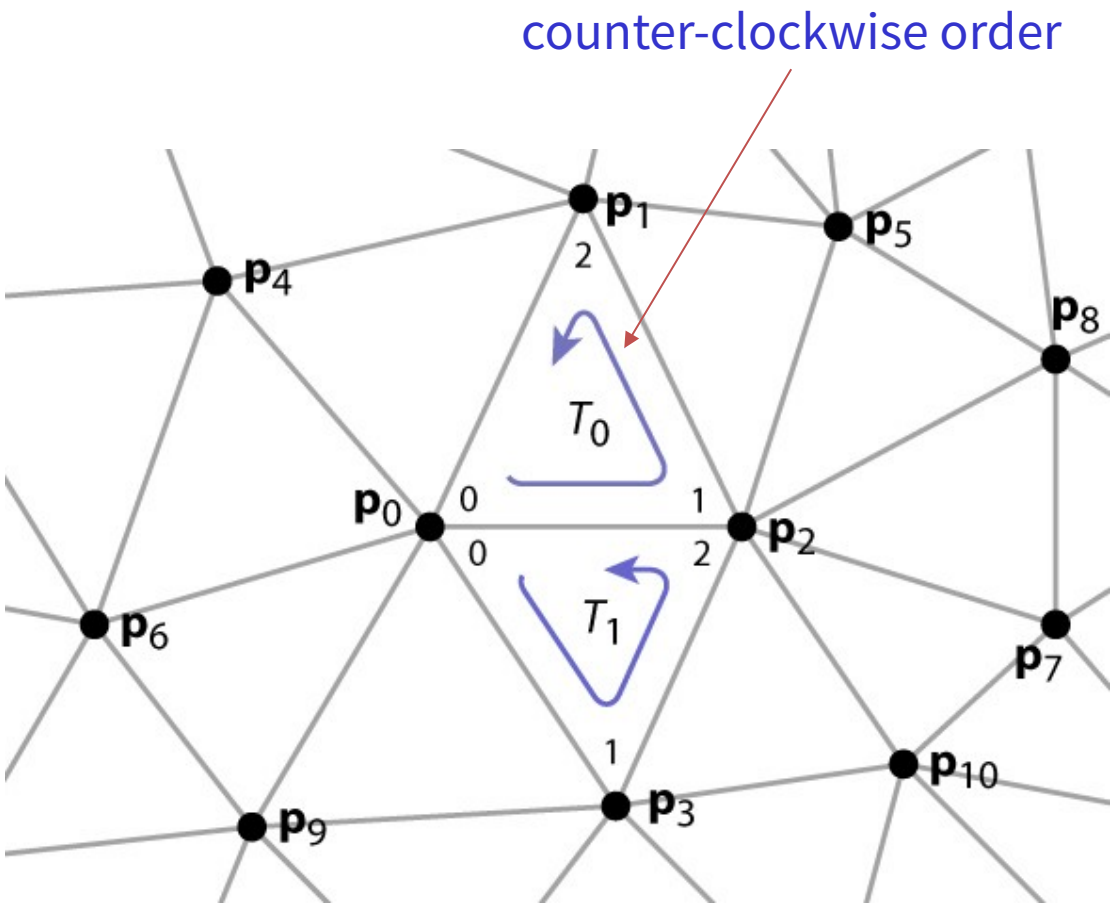
Indexed triangle set

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
	⋮



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

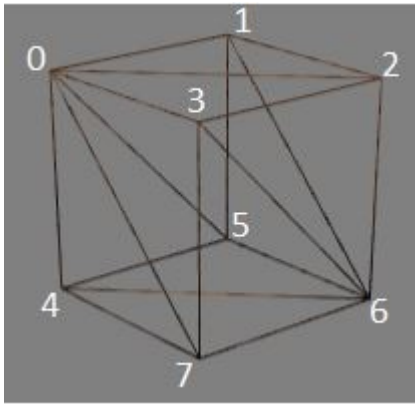
- 1. Create a vertex array & **index array** for your mesh
  - The vertex array **should not have duplicate vertex data**
- 2. Specify “pointer” to this vertex array
  - Same with the separate triangles case
- 3. Render the mesh using the specified “pointer” & **indices of vertices to render**
  - Using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

[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):
    # ...
```

More about Vertex Array...

- More reference
  - http://www.songho.ca/opengl/gl_vertexarray.html

- Actually, what we've just used are **client-side** vertex and index arrays
  - Vertex and index arrays were stored in **main memory (RAM).**

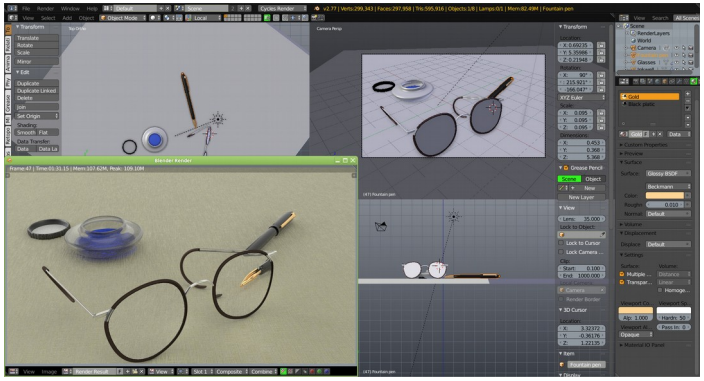
- **Vertex buffer object (VBO)**: **server-side** vertex and index arrays
  - This allows vertex array data to be stored in **high-performance graphics card memory**, so much faster than client-side arrays.

  - Even using VBO is the only way to provide vertex data in modern OpenGL.

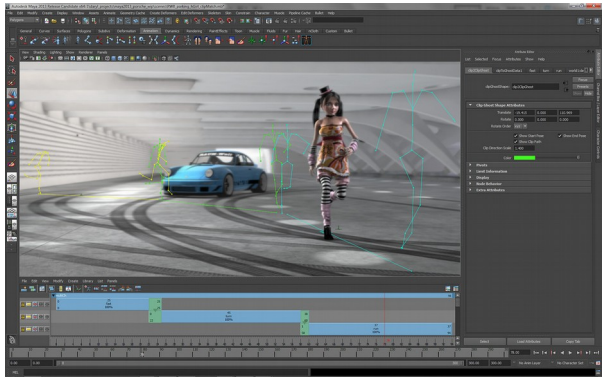
  - But VBO will not be covered in this class. For more information, see:
    - http://www.songho.ca/opengl/gl_vbo.html
    - http://www.falloutsoftware.com/tutorials/gl/gl3.htm
    -

Modeling tools

- How can we create meshes?
- An *object file* or *model file* storing polygon mesh data is usually created using 3D authoring 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
- FBX
  - Support animations and skins

KAIST

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

KAIST

OBJ Face Varieties

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

or

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

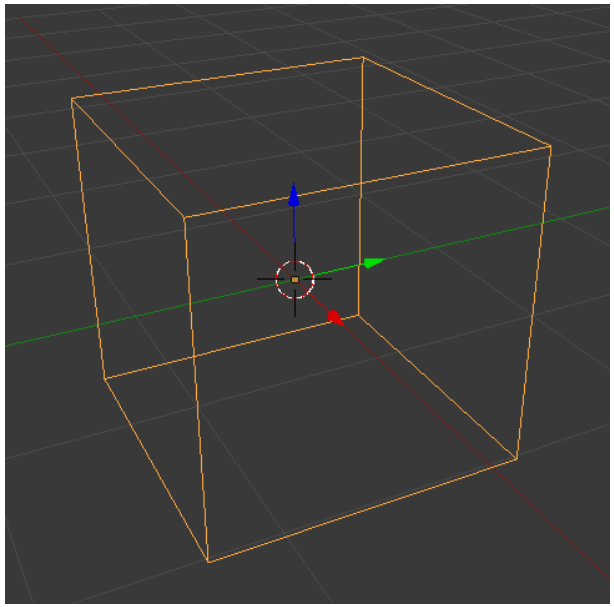
or

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

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

KAIST

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)
- Import cube.obj into Blender (File-Import)
  - Press ‘z’ to render in wireframe mode
- Edit cube.obj somehow (using a text editor)
- 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 6

- 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/