Computer Graphics

2 - Introduction to OpenGL

Yoonsang Lee and Taesoo Kwon

Spring 2019

Topics Covered

- Introduction to OpenGL
  - What is OpenGL?
  - OpenGL basics
  - GLFW input handling
  - Legacy OpenGL & Modern OpenGL
  - OpenGL as a Learning Tool

Introduction to OpenGL

What is OpenGL?

- **Op**en **G**raphics **L**ibrary

- OpenGL is an **API** (Application Programming Inte rface) for graphics programming.
  - Unlike its name, OpenGL is not a library.

What is OpenGL?

- **API** is a **specification.**
  - API describes **interfaces** and **expected behavior.**

- As for OpenGL API,
  - OS vendors provide OpenGL interface (e.g. opengl32.dll on windows)
  - GPU vendors provide OpenGL implementation, the grap hic card driver (e.g. Nvidia drivers)

Characteristics of OpenGL

- Cross platform
  - You can use OpenGL on Windows, OS X, Linux, iOS, Android, ...
- Language independent
  - OpenGL has many language bindings (C, Python, Java, Javascript, ...)
  - We'll use its Python binding in this class - PyOpenGL

What can we do with OpenGL?

- **Only for drawing objects**
  - Provides a small set of low-level drawing operations
  - No functions for creating windows & OpenGL contexts, handling events (we'll discuss the "context" later)

- Additional libraries are required to use OpenGL
  - GLFW, FreeGLUT : Simple utility libraries for OpenGL
  - Fltk, wxWigets, Qt, Gtk : General purpose GUI f rameworks

Utility Libraries for Learning OpenGL

- General GUI frameworks such as Qt are powerful, but are too complex for just learning OpenGL.

- GLUT "was" the most popular library for this purp ose
  - But it's outdated and unmaintained.
  - Open-source library FreeGLUT provides a stable clone o f GLUT.

- Now, GLFW is getting more popular.
  - Provides much finer control for managing windows and

[Practice] First OpenGL Program

```python
import glfw
from OpenGL.GL import *

def render():
    pass

def main():
    # Initialize the library
    if not glfw.init():
        return
    # Create a windowed mode window and its OpenGL context
    window = glfw.create_window(640,480,"Hello World", None,None)
    if not window:
        glfw.terminate()
        return

    # Make the window's context current
    glfw.make_context_current(window)

    # Loop until the user closes the window
    while not glfw.window_should_close(window):
        # Poll events
        glfw.poll_events()

        # Render here, e.g. using pyOpenGL
        render()

        # Swap front and back buffers
        glfw.swap_buffers(window)

    glfw.terminate()

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

import X
: to access X's attributes or methods using X.attribute, X.method()

from X import *
: to access X's attributes or methods without using "X."

If the python interpreter is **running this source file as the main program**, it sets the special **__name__** variable to have a value **"__main__"**.

If this file **is being imported from another module**, **__name__** will be set to **the module's name**.

[Practice] Draw a Triangle

```
def render():
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()
    glBegin(GL_TRIANGLES)
    glVertex2f(0.0, 1.0)
    glVertex2f(-1.0,-1.0)
    glVertex2f(1.0,-1.0)
    glEnd()
```

Vertex

- In OpenGL, geometry is specified by **vertices.**
- To draw something, vertices have to be listed
- between *glBegin(primitive_type)* and *glEnd()* calls.
- *glVertex*()* specifies the coordinate values of a vert ex.

```
glBegin(GL_TRIANGLES)
glVertex2f(0.0, 1.0)
glVertex2f(-1.0,-1.0)
glVertex2f(1.0,-1.0)
glEnd()
```

[Practice] Draw a Triangle

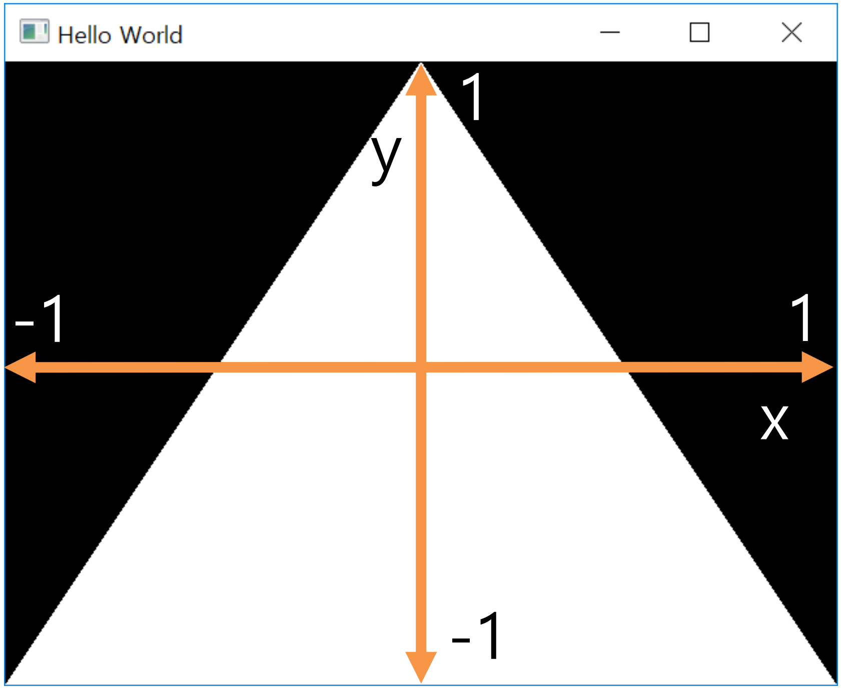
```
def render():
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()
    glBegin(GL_TRIANGLES)
    glVertex2f(0.0, 1.0)
    glVertex2f(-1.0,-1.0)
    glVertex2f(1.0,-1.0)
    glEnd()
```

Coordinate System

- You can draw the triangle anywhere in a 2D square ranging from (-1, -1) to (1, 1).

- Called “Normalized Device Coordinate” (NDC).

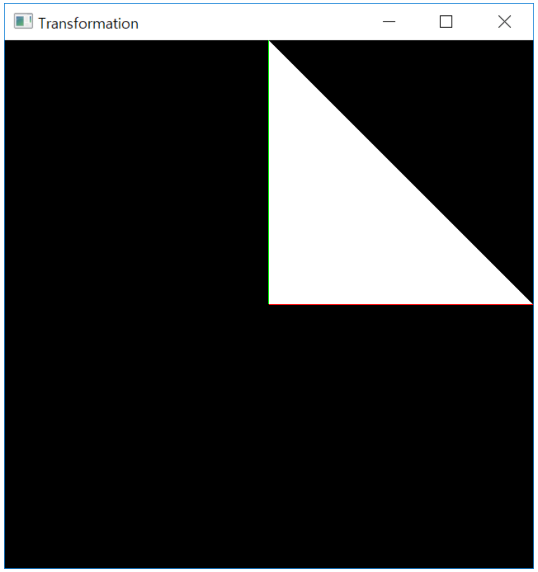
- We’ll see how objects are tr ansformed to NDC in later c lasses.



2D Transformation

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

def render(T):
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()

    # draw cooridnate
    glBegin(GL_LINES)
    glColor3ub(255, 0, 0)
    glVertex2fv(np.array([0.,0.]))
    glVertex2fv(np.array([1.,0.]))
    glColor3ub(0, 255, 0)
    glVertex2fv(np.array([0.,0.]))
    glVertex2fv(np.array([0.,1.]))
    glEnd()

    # draw triangle
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex2fv(T @ np.array([0.0,0.5]))
    glVertex2fv(T @ np.array([0.0,0.0]))
    glVertex2fv(T @ np.array([0.5,0.0]))
    glEnd()
```

Uniform Scale




```
def main():
    if not glfw.init():
        return
    window = glfw.create_window(640,640, "2D Trans", None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)

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

        T = np.array([[2.,0.],
                      [0.,2.]])
        render(T)

        glfw.swap_buffers(window)

    glfw.terminate()

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

[Practice] Animate it!

```
def main():
    if not glfw.init():
        return
    window = glfw.create_window(640,640,"2D Trans", None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)

    # set the number of screen refresh to wait before calling glfw.swap_buffer().
    # if your monitor refresh rate is 60Hz, the while loop is repeated every 1/60 sec
    glfw.swap_interval(1)

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

        # get the current time, in seconds
        t = glfw.get_time()

        s = np.sin(t)
        T = np.array([[s,0.],
                      [0.,s]])
        render(T)

        glfw.swap_buffers(window)
    glfw.terminate()
```

[Practice] Nonuniform Scale, Rotation, Reflection, Shear

```
while not glfw.window_should_close(window):
    glfw.poll_events()
    t = glfw.get_time()

    # nonuniform scale
    s = np.sin(t)
    T = np.array([[s,0.],
                  [0.,s*.5]])
    # rotation
    th = t
    T = np.array([[np.cos(th), -np.sin(th)],
                  [np.sin(th), np.cos(th)]])
    # reflection
    T = np.array([[-1.,0.],
                  [0.,1.]])
    # shear
    a = np.sin(t)
    T = np.array([[1.,a],
                  [0.,1.]])
    # identity matrix
    T = np.identity(2)

    render(T)
    glfw.swap_buffers(window)
```

[Practice] Composition

```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        S = np.array([[1.,0.],
                      [0.,2.]])
        th = np.radians(60)
        R = np.array([[np.cos(th), -np.sin(th)],
                      [np.sin(th), np.cos(th)]])

        # compare results of these two lines
        render(R @ S)
        # render(S @ R)

        # ...
```

[Practice] Homogeneous Coordinates

```
def render(T):
    # ...
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex2fv( (T @ np.array([.0,.5,1.]))[:-1] )
    glVertex2fv( (T @ np.array([.0,.0,1.]))[:-1] )
    glVertex2fv( (T @ np.array([.5,.0,1.]))[:-1] )
    glEnd()
```

[Practice] Homogeneous Coordinates

```
def main():
    # ...
    while not glfw.window_should_close(window):
        glfw.poll_events()

        # rotate 60 deg about z axis
        th = np.radians(60)
        R = np.array([[np.cos(th), -np.sin(th),0.],
                      [np.sin(th), np.cos(th),0.],
                      [0.,         0.,         1.]])

        # translate by (.4, .1)
        T = np.array([[1.,0.,.4],
                      [0.,1.,.1],
                      [0.,0.,1.]])

        render(R)
        # render(T)
        # render(T @ R)
        # render(R @ T)
        # ...
```

[Practice] 3D Transformations

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

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

    glLoadIdentity()

    # use orthogonal projection (we'll see details later)
    glOrtho(-1,1, -1,1, -1,1)

    # rotate "camera" position to see this 3D space better (we'll see details later)
    gluLookAt(.1*np.sin(camAng),.1, .1*np.cos(camAng), 0,0,0, 0,1,0)

    # 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()

    # draw triangle
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 255, 255)
    glVertex3fv((M @ np.array([.0,.5,0.,1.]))[:-1])
    glVertex3fv((M @ np.array([.0,.0,0.,1.]))[:-1])
    glVertex3fv((M @ np.array([.5,.0,0.,1.]))[:-1])
    glEnd()
```

```python
def main():
    if not glfw.init():
        return
    window = glfw.create_window(640,640, "3D Trans", None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.swap_interval(1)
    count = 0
    while not glfw.window_should_close(window):
        glfw.poll_events()
        t = glfw.get_time()

        # rotate -60 deg about x axis
        th = np.radians(-60)
        R = np.array([[1.,0.,0.,0.],
           [0., np.cos(th), -np.sin(th),0.],
           [0., np.sin(th), np.cos(th),0.],
                       [0.,0.,0.,1.]])

        # translate by (.4, 0., .2)
        T = np.array([[1.,0.,0.,.4],
                       [0.,1.,0.,0.],
                       [0.,0.,1.,.2],
                       [0.,0.,0.,1.]])

        camAng = t
        render(R, camAng)
        # render(T, camAng)
        # render(T @ R, camAng)
        # render(R @ T, camAng)

        glfw.swap_buffers(window)

    glfw.terminate()

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

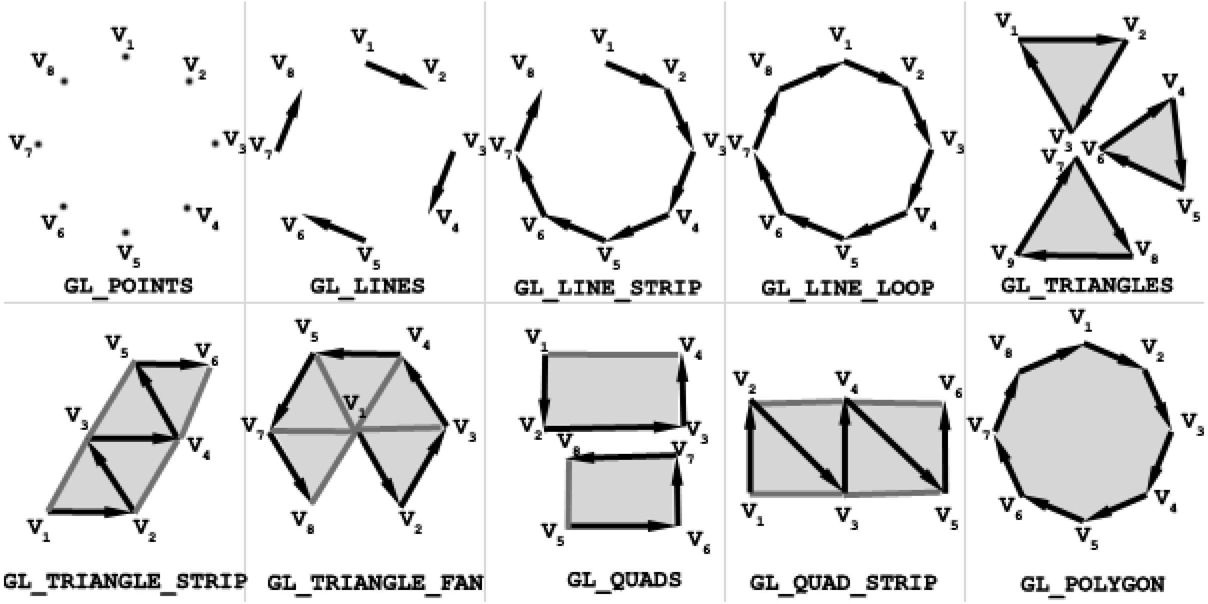
[Practice] Use Slicing

- You can use **slicing** for cleaner code (the behavior is the same as the previous page)

```
# ...

# rotate 60 deg about x axis
th = np.radians(-60)
R = np.identity(4)
R[:3,:3] = [[1.,0.,0.],
            [0., np.cos(th), -np.sin(th)],
            [0., np.sin(th), np.cos(th)]]

# translate by (.4, 0., .2)
T = np.identity(4)
T[:3,3] = [.4, 0., .2]

# ...
```

Primitive Types

- Primitive types in *glBegin(**primitive_type**)* :



- They represents how vertices are to be connected.

[Practice] Change the Primitive Type

```python
def render():
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()
    glBegin(GL_POINTS)
    # glBegin(GL_LINES)
    # glBegin(GL_LINE_STRIP)
    # glBegin(GL_LINE_LOOP)
    # ...
    glVertex2f(0.0, 0.5)
    glVertex2f(-0.5,-0.5)
    glVertex2f(0.5,-0.5)
    glEnd()
```

Vertex Attributes

- In OpenGL, a vertex can have these attributes:
  - **Vertex coordinate** : specified by glVertex*()
  - **Vertex color** : specified by glColor*()
  - **Normal vector** : specified by glNormal*()
  - **Texture coordinate** : specified by glTexCoord*()

- We’ll see normal vector & texture coord. in later
- classes.

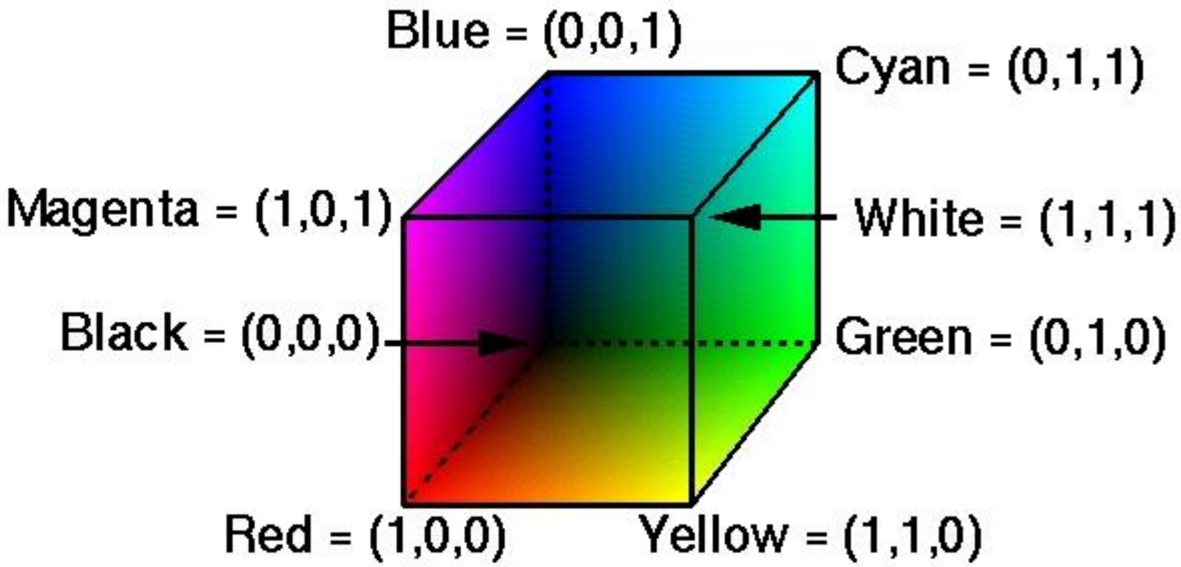
- Now, let’s have a look at the **vertex color**.

[Practice] Colored Triangle

```
def render():
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()
    glBegin(GL_TRIANGLES)
    glColor3f(1.0, 0.0, 0.0)
    glVertex2f(0.0, 1.0)
    glColor3f(0.0, 1.0, 0.0)
    glVertex2f(-1.0,-1.0)
    glColor3f(0.0, 0.0, 1.0)
    glVertex2f(1.0,-1.0)
    glEnd()
```

Color

- OpenGL uses the RGB color model.



- Vertex colors are interpolated for e

How to draw a "red" triangle?

- Set red color for each vertex.

- Or you can specify the

```
def render():
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()
    glBegin(GL_TRIANGLES)
    glColor3f(1.0, 0.0, 0.0)
    glVertex2f(0.0, 1.0)
    glVertex2f(-1.0,-1.0)
    glVertex2f(1.0,-1.0)
    glEnd()
```

OpenGL is a State Machine

- If you set a value for a state (or mode), **it remains i n effect until you change it.**
  - For example, “current” color
  - Others states:
    - “current” viewing and projection transformations
    - “current” polygon drawing modes
    - “current” positions and characteristics of lights
    - “current” material properties of the objects
    - ...
- **OpenGL context** stores all the states associated wit h

OpenGL Functions

`glVertex3fv( v )`

Number of components

2	(x,y)
3	(x,y,z)
4	(x,y,z,w)

Data Type

b	byte
ub	unsigned byte
s	short
us	unsigned short
i	int
ui	unsigned int
f	float
d	double

Vector

omit "v" for scalar form

`glVertex2f( x, y )`

[Practice] Using other forms of OpenGL Functions

```
import numpy as np

def render():
    glClear(GL_COLOR_BUFFER_BIT)
    glLoadIdentity()
    glBegin(GL_TRIANGLES)
    glColor3ub(255, 0, 0)
    glVertex2fv((0.0, 1.0))
    glVertex2fv([-1.0,-1.0])
    glVertex2fv(np.array([1.0,-1.0]))
    glEnd()
```

GLFW Input Handling

- *glfw.poll_events()*
  - Processes events that have already been received and then returns immediately.
  - Calls a user-registered callback function for each type of

Event type	Set a callback using…
Key input	*glfw.set_key_callback()*
Mouse cursor position	*glfw.set_cursor_pos_callback()* or just poll the position using *glfw.get_cursor_pos()*
Mouse button	*glfw.set_mouse_button_callback()*
Mouse scroll	*glfw.set_scroll_callback()*

```python
import glfw
from OpenGL.GL import *

def render():
    pass

def key_callback(window, key, scancode, action, mods):
    if key==glfw.KEY_A:
        if action==glfw.PRESS:
            print('press a')
        elif action==glfw.RELEASE:
            print('release a')
        elif action==glfw.REPEAT:
            print('repeat a')
    elif key==glfw.KEY_SPACE and action==glfw.PRESS:
        print ('press space: (%d, %d)'%glfw.get_cursor_pos(window))

def cursor_callback(window, xpos, ypos):
    print('mouse cursor moving: (%d, %d)'%(xpos, ypos))

def button_callback(window, button, action, mod):
    if button==glfw.MOUSE_BUTTON_LEFT:
        if action==glfw.PRESS:
            print('press left btn: (%d, %d)'%glfw.get_cursor_pos(window))
        elif action==glfw.RELEASE:
            print('release left btn: (%d, %d)'%glfw.get_cursor_pos(window))

def scroll_callback(window, xoffset, yoffset):
    print('mouse wheel scroll: %d, %d'%(xoffset, yoffset))
```

```python
def main():
    # Initialize the library
    if not glfw.init():
        return
    # Create a windowed mode window and its OpenGL context
    window = glfw.create_window(640, 480, "Hello World", None, None)
    if not window:
        glfw.terminate()
        return

    glfw.set_key_callback(window, key_callback)
    glfw.set_cursor_pos_callback(window, cursor_callback)
    glfw.set_mouse_button_callback(window, button_callback)
    glfw.set_scroll_callback(window, scroll_callback)

    # Make the window's context current
    glfw.make_context_current(window)

    # Loop until the user closes the window
    while not glfw.window_should_close(window):
        # Poll for and process events
        glfw.poll_events()
        # Render here, e.g. using pyOpenGL
        render()
        # Swap front and back buffers
        glfw.swap_buffers(window)

    glfw.terminate()
if __name__ == "__main__":
    main()
```

Documentation for glfw

- http://www.glfw.org/documentation.html

- Note there are changes in the python binding:
  - function names use the pythonic **words_with_underscores** notation instead of camelCase
  - **GLFW_ and glfw prefixes have been removed**, as their function is replaced by the module namespace
  - functions like glfwGetMonitors **return a list instead of a pointer and an object count**
  - see https://pypi.python.org/pypi/glfw for more information

Legacy OpenGL & Modern OpenGL

- Legacy OpenGL (OpenGL 1.x)
  - Invented when “fixed-function” hardware was standard
  - No shaders
  - Easier to use & good for rapid prototyping
  - Deprecated since OpenGL 3.0

- Modern OpenGL (OpenGL 2.x~)
  - Now programmable hardwares became standard
  - Use programmable shaders
  - More difficult to program but far more efficient & power fu

OpenGL as a Learning Tool

- Our focus in this class is on fundamental computer graphics concepts,
- not on efficient implementations.

  So we mostly used legacy OpenGL examples

  because of its simplicity.

Now,

- Lab in this week:
  - Lab assignment 2