Computer Graphics Workshop:

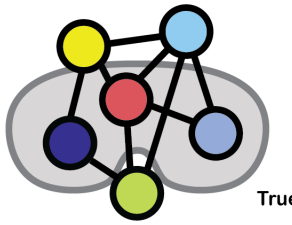
# Graphics Programming using PyOpenGL

**Trudi Qi**, Ph.D.

Assistant Professor, EE&CS

**Fowler School of Engineering**

March 13, 2023



tvil

True Virtual Intelligence Lab

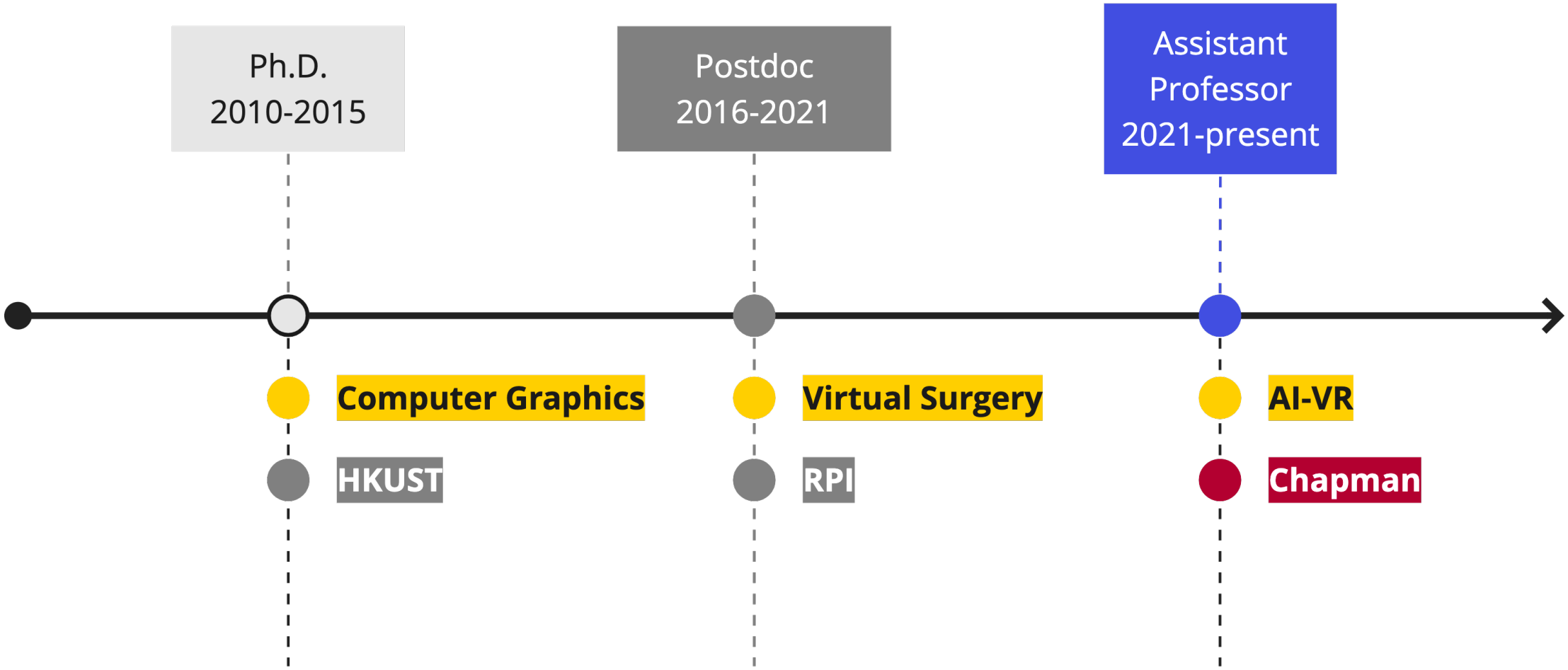
CHAPMAN UNIVERSITY

# Setup

- https://github.com/trudiQ/CG-Workshop.git
  - Installation, source code & instructional slides.

# About Dr. Trudi Qi

# Bio



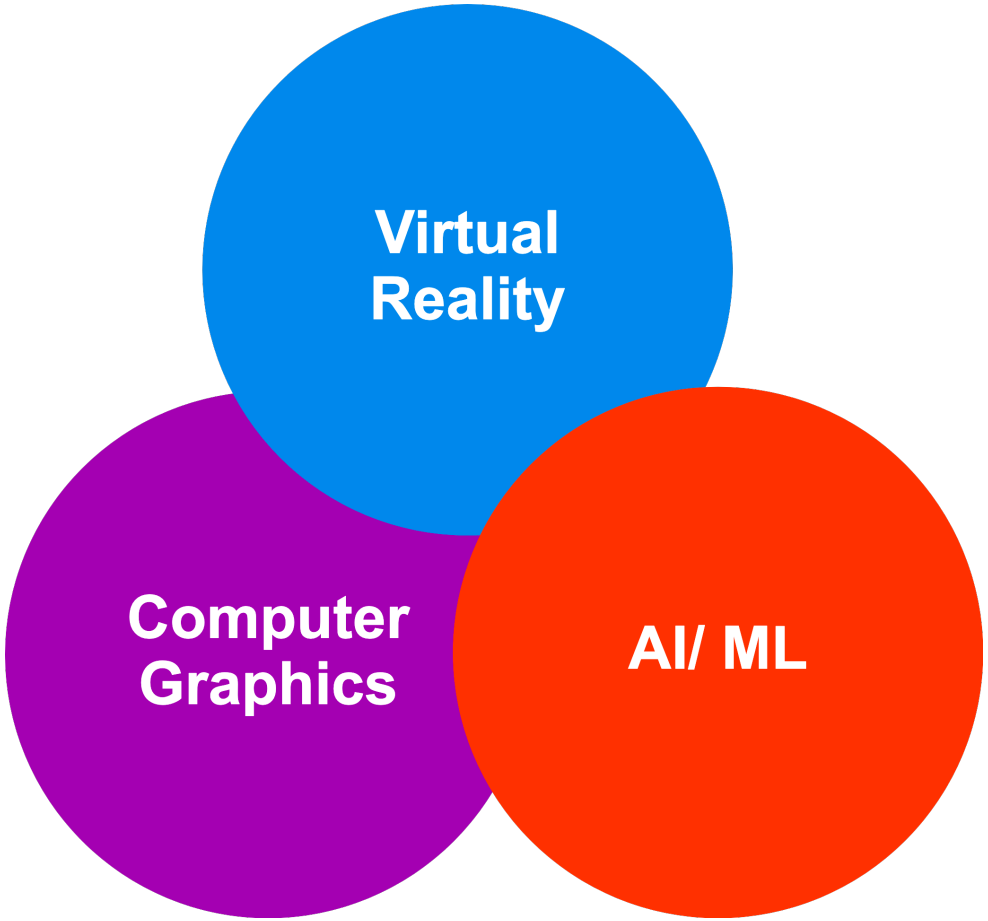

tvil
True Virtual Intelligence Lab

CHAPMAN UNIVERSITY

# Research Interest

Integrating **AI** with **visual computing** technologies to facilitate **human-centered applications,** e.g., healthcare and education, through **virtual reality**.

**AI-powered human-centered VR**




tvil
True Virtual Intelligence Lab

CHAPMAN UNIVERSITY

# What is Computer Graphics?

Computer graphics is a sub-field of computer science which studies **computational methods** and techniques for **digitally synthesizing and manipulating** visual content.

**- WIKIPEDIA**

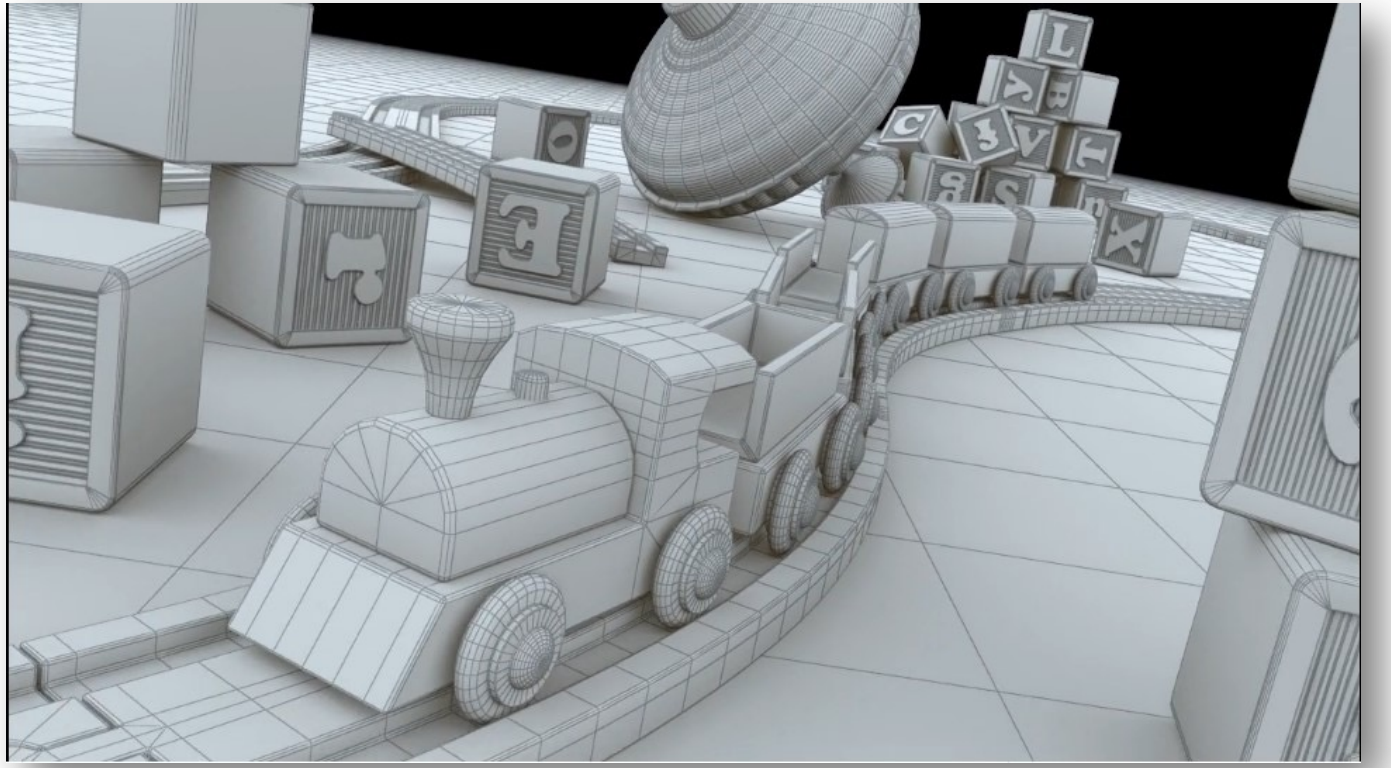
*by Daniel Guerra:*
*https://vimeo.com/106854149*

# Core Areas in Graphics

- **Modeling**
- **Rendering**
- **Imaging**
- **Animation**

# Core Areas in Graphics

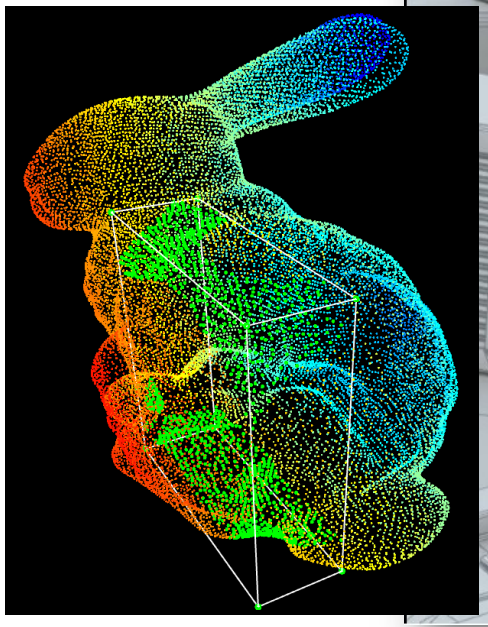
- **Modeling**



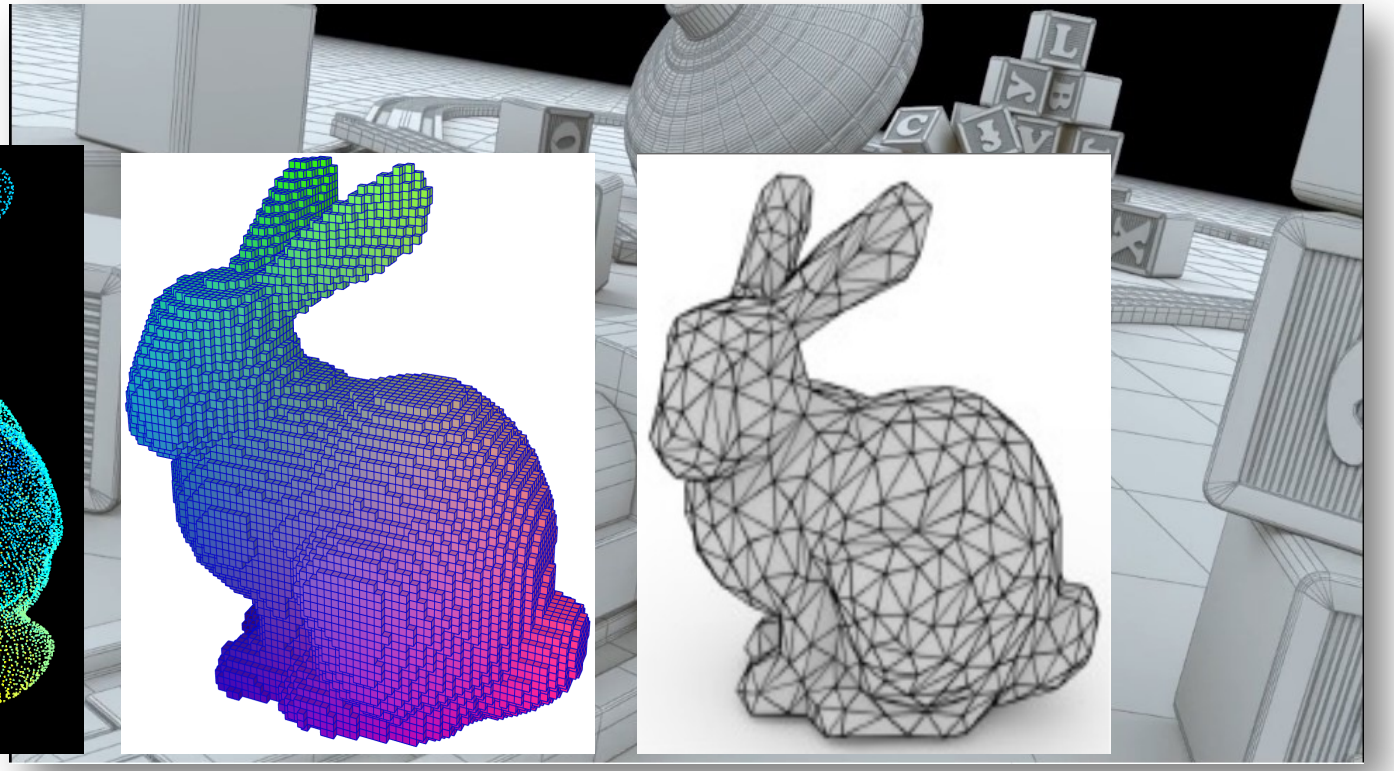
**Deals with the mathematical representation of shape and computational methods for manipulation using computers**

# Core Areas in Graphics

- **Modeling**



Point cloud



Voxels

Polygonal mesh

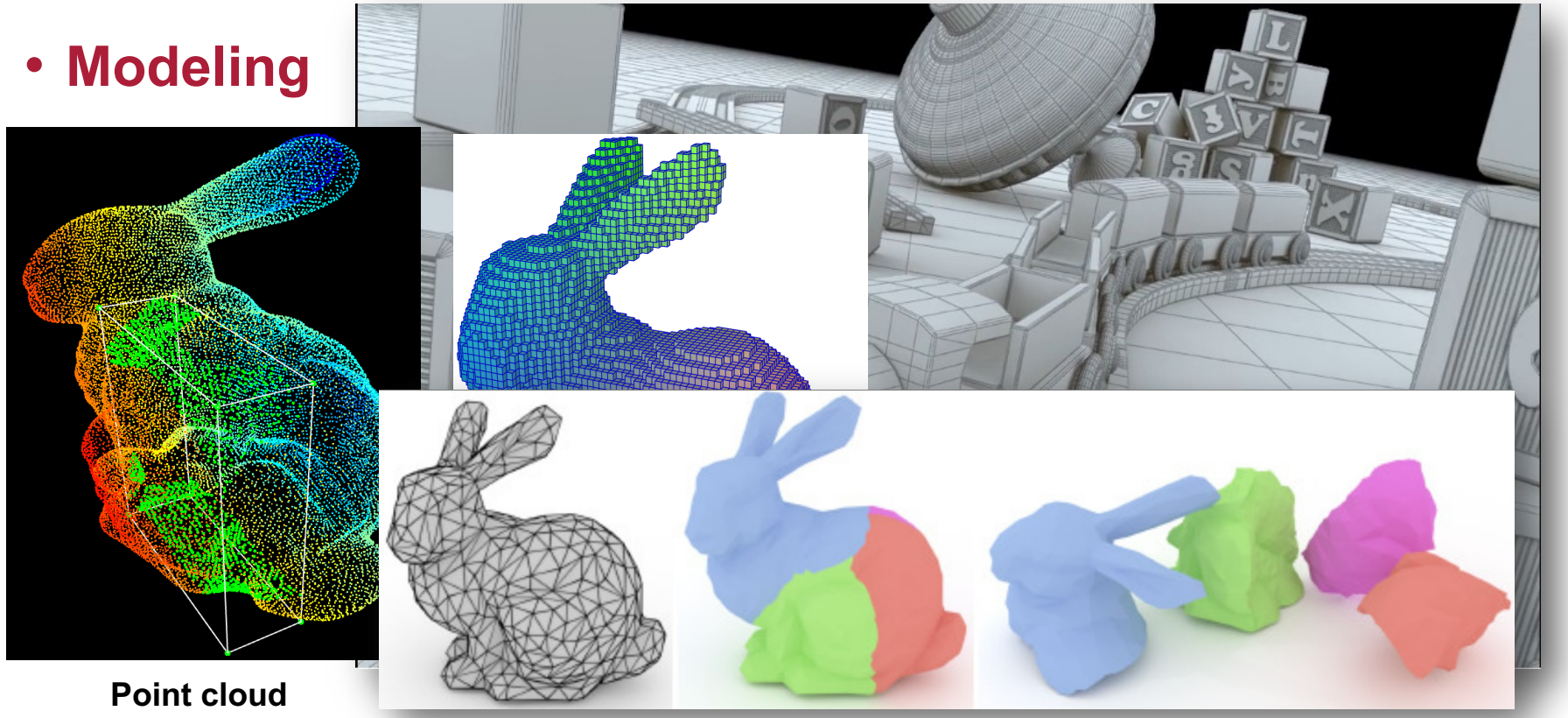
**Deals with the mathematical representation of shape and computational methods for manipulation using computers**

tvil True Virtual Intelligence Lab

CHAPMAN UNIVERSITY

# Core Areas in Graphics

- **Modeling**



Point cloud

**Deals with the mathematical representation of shape and computational methods for manipulation using computers**

tvil True Virtual Intelligence Lab

CHAPMAN UNIVERSITY

# Core Areas in Graphics

- **Rendering: texturing**

**algorithms dealing with the simulating materials and environmental effects based on the 3D computer models**

# Core Areas in Graphics

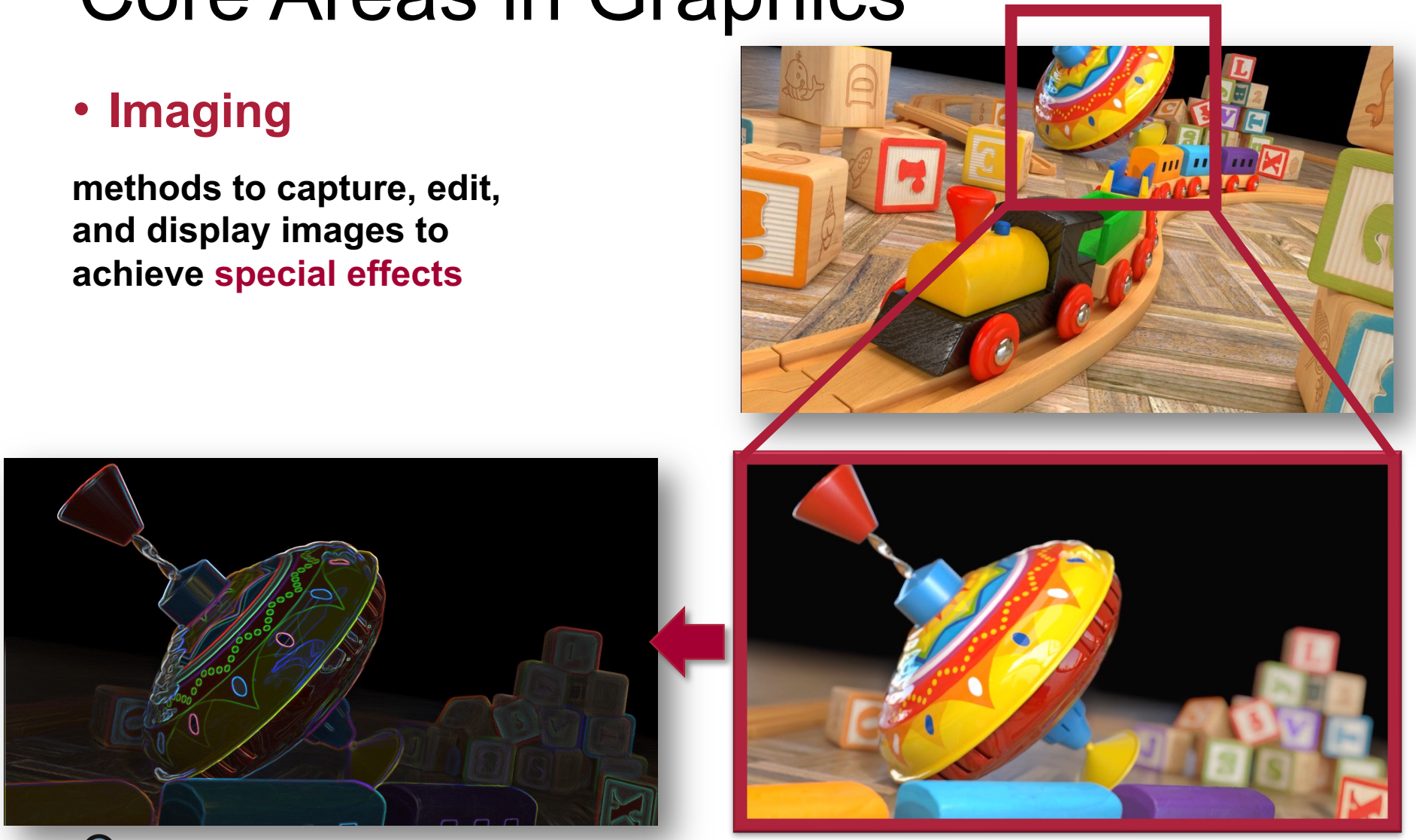
- **Rendering: Lighting**

**algorithms dealing with the simulating materials and environmental effects based on the 3D computer models**

# Core Areas in Graphics

- **Imaging**

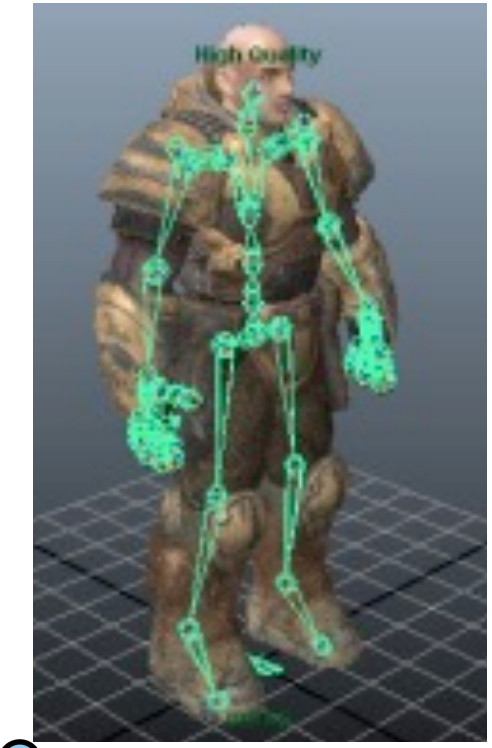
**methods to capture, edit, and display images to achieve special effects**



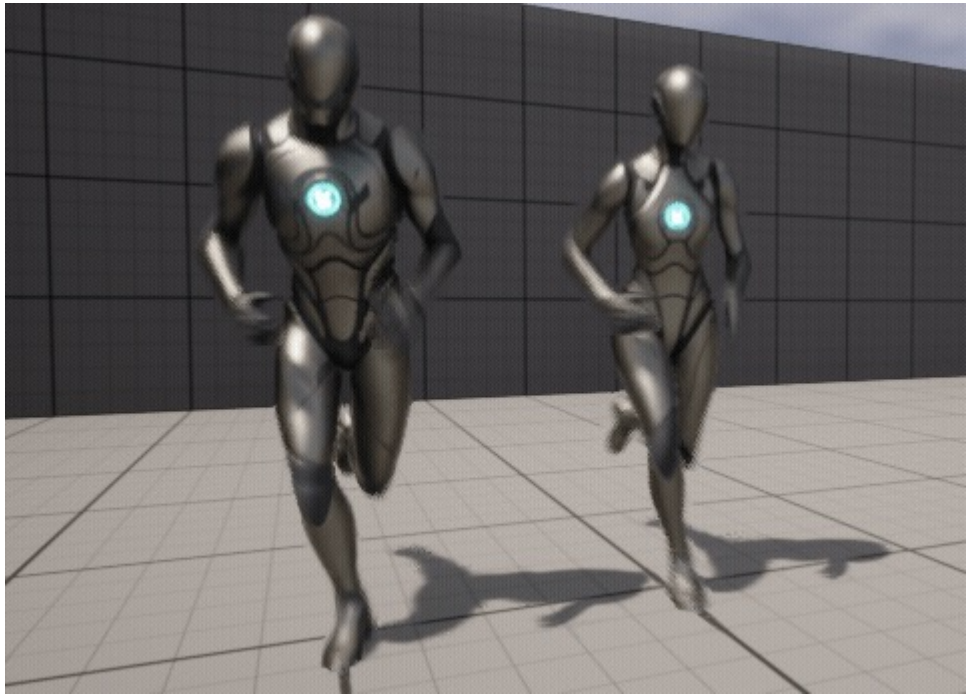
# Core Areas in Graphics

- **Animation**

**techniques to create and manipulate motion and simulate deformation behavior**



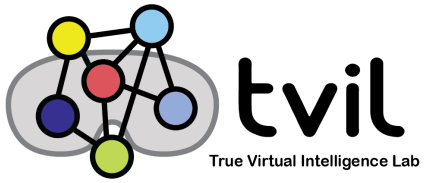

*Skeletal animation*



*Unreal Engine 5.0 documentation*

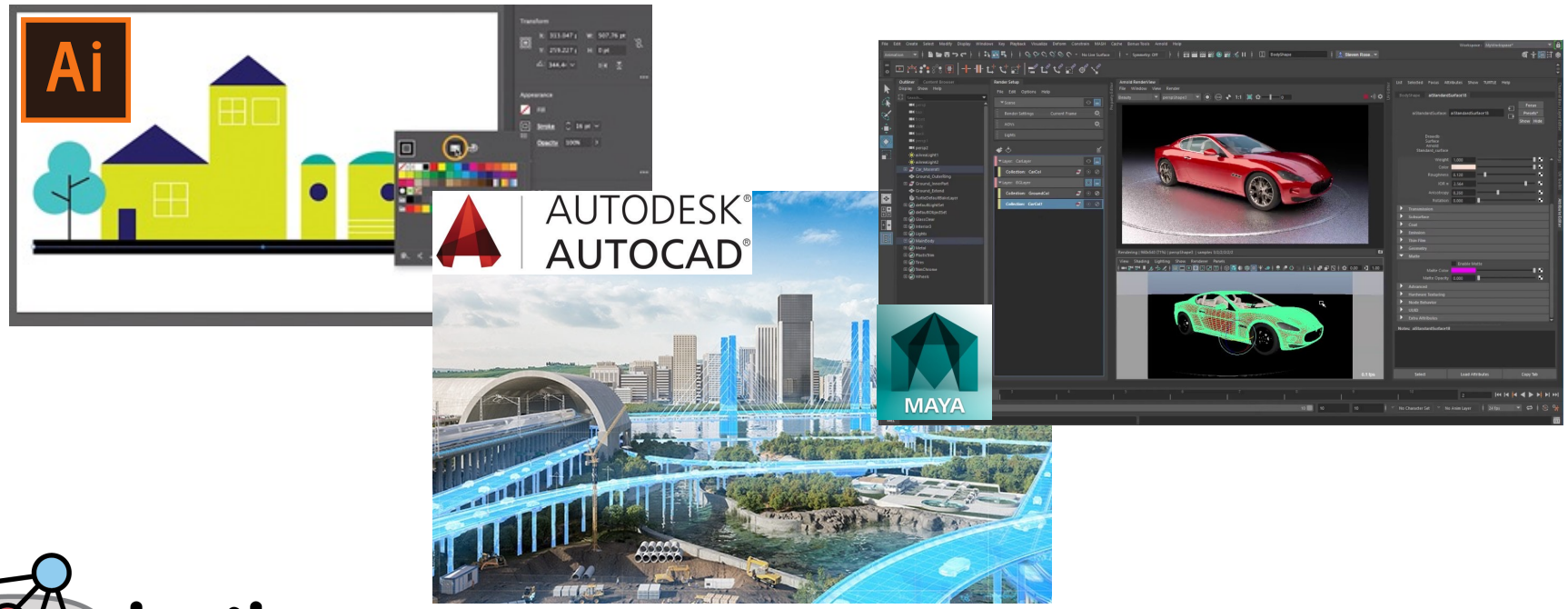
# OpenGL

**Computer graphics programming APIs, OpenGL pipeline and basics**




# Computer Graphics Software

- Two classifications
  - **Special-purpose packages**
    - Designed for non-programmers who want to generate pictures, graphics etc.
    - Examples: Adobe Illustrator, AutoCAD, Maya, etc.



# Computer Graphics Software

- Two classifications
  - **Special-purpose packages**
  - **General graphics APIs (**Application Programming Interface**)**
    - Provide **a set of graphics functions to draw and render general 2D and 3D graphics**, designed to be implemented mostly or entirely in hardware (e.g., GPU)
    - Major APIs: OpenGL, Direct3D, Vulkan etc.

# Graphics APIs

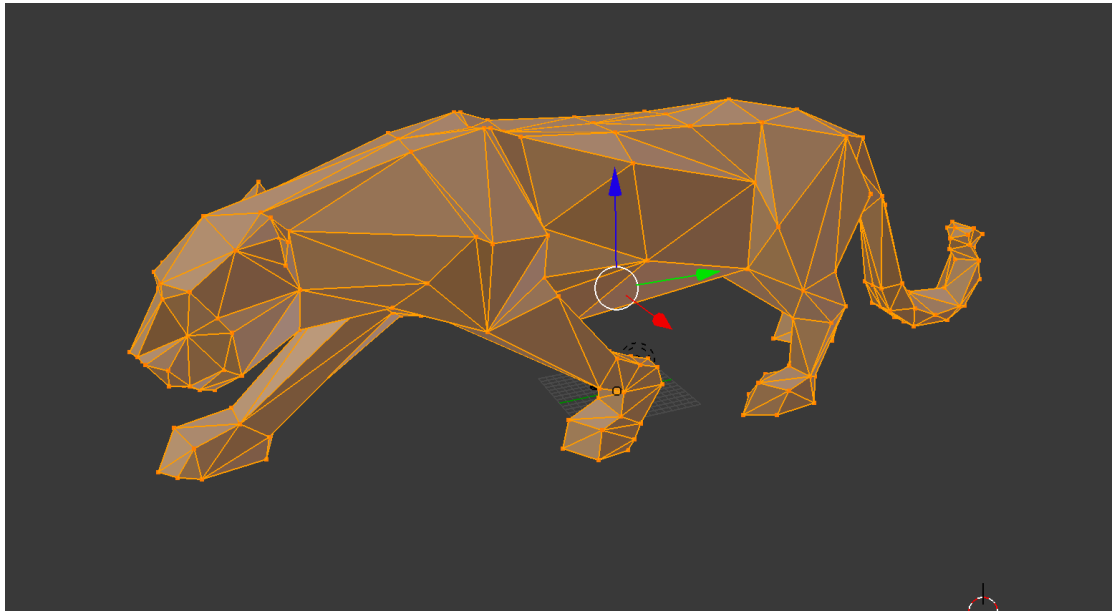
- A general-purpose graphics APIs provide users with **a variety of functions** for creating and manipulating 2D/3D shapes, including
  - **Modeling** - creating 2D/3D shapes
  - **Geometric transformations** – translating/rotating/scaling 2D/3D shapes
  - **Viewing transformations** – specifying the view projected onto the screen
  - **Rendering/Imaging** – coloring, texturing
  - **Input functions** – allowing mouse/keyboard interactions

# Graphics APIs

- A general-purpose graphics APIs provide users with **a variety of functions** for creating and manipulating 2D/3D shapes, including
  - **Modeling** - creating 2D/3D shapes

Input vertices

Primitives (triangles)



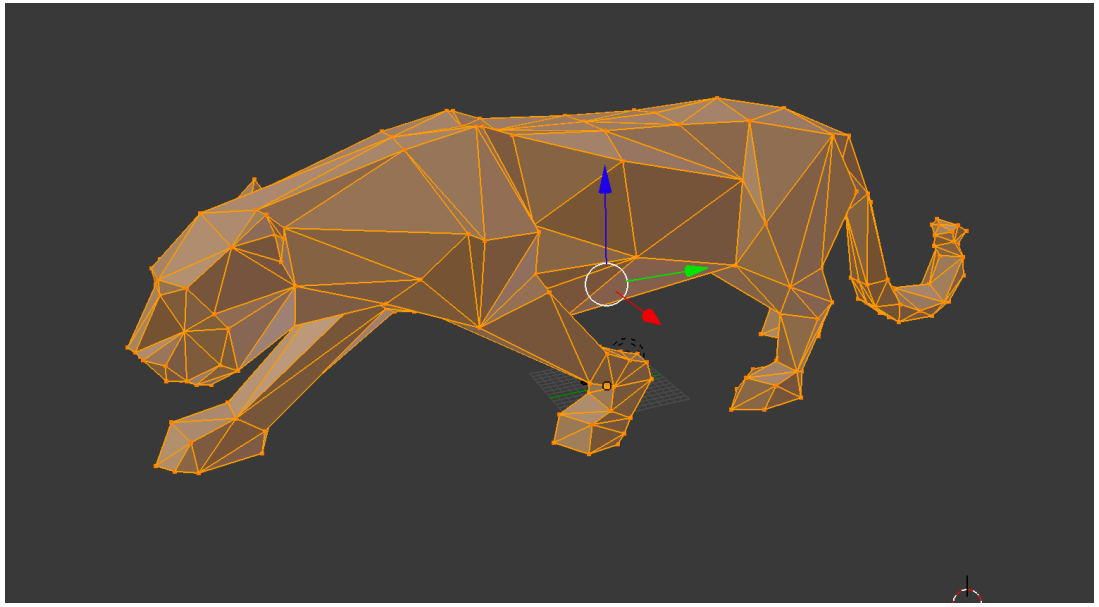
3D mesh

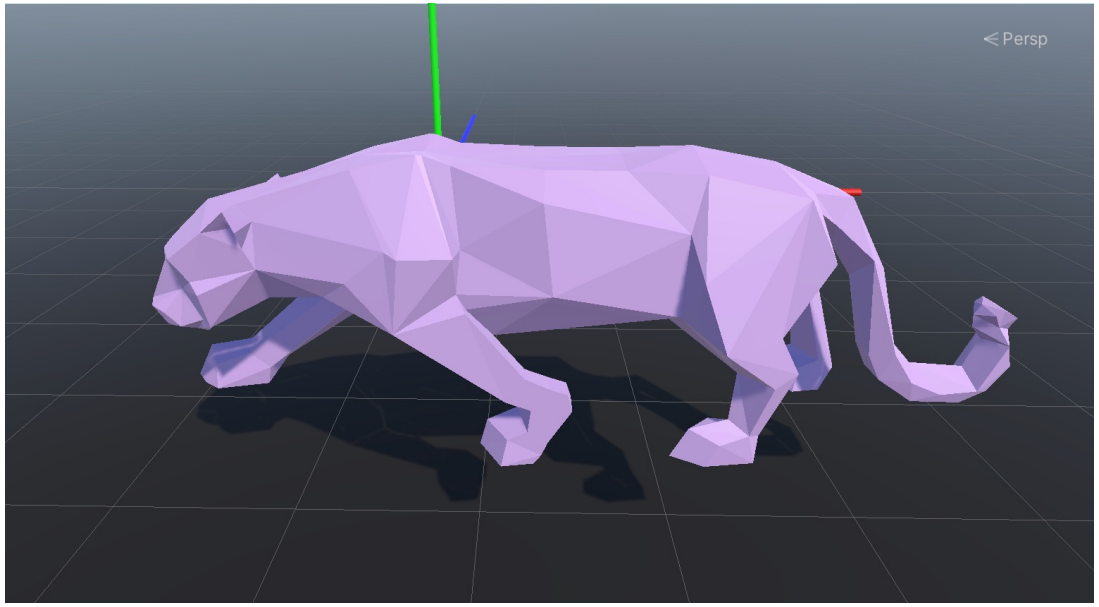


# Graphics APIs

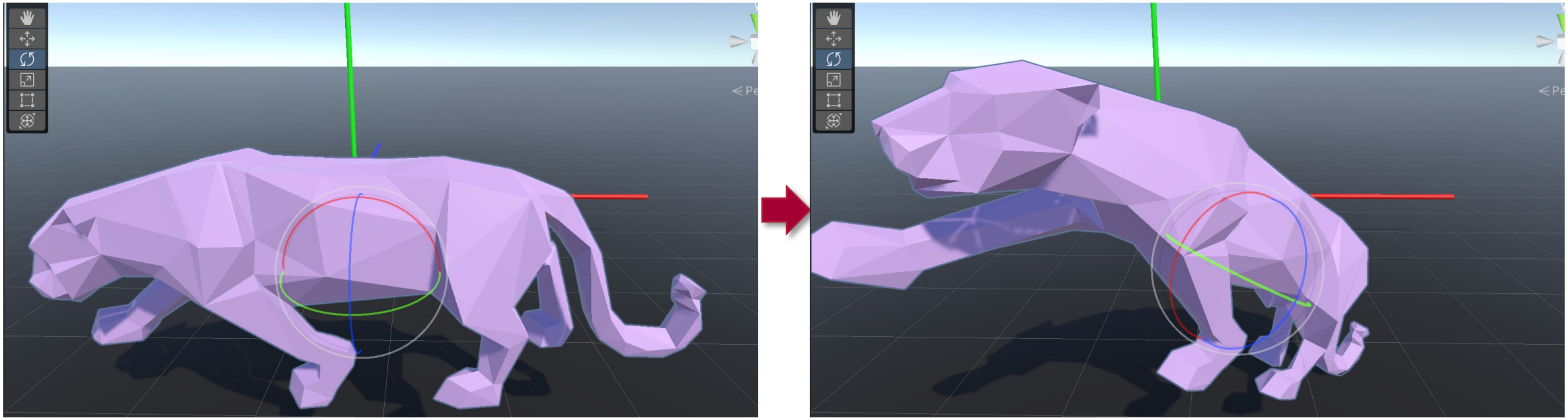
- A general-purpose graphics APIs provide users with **a variety of functions** for creating and manipulating 2D/3D shapes, including
  - **Rendering/Imaging** – coloring, texturing





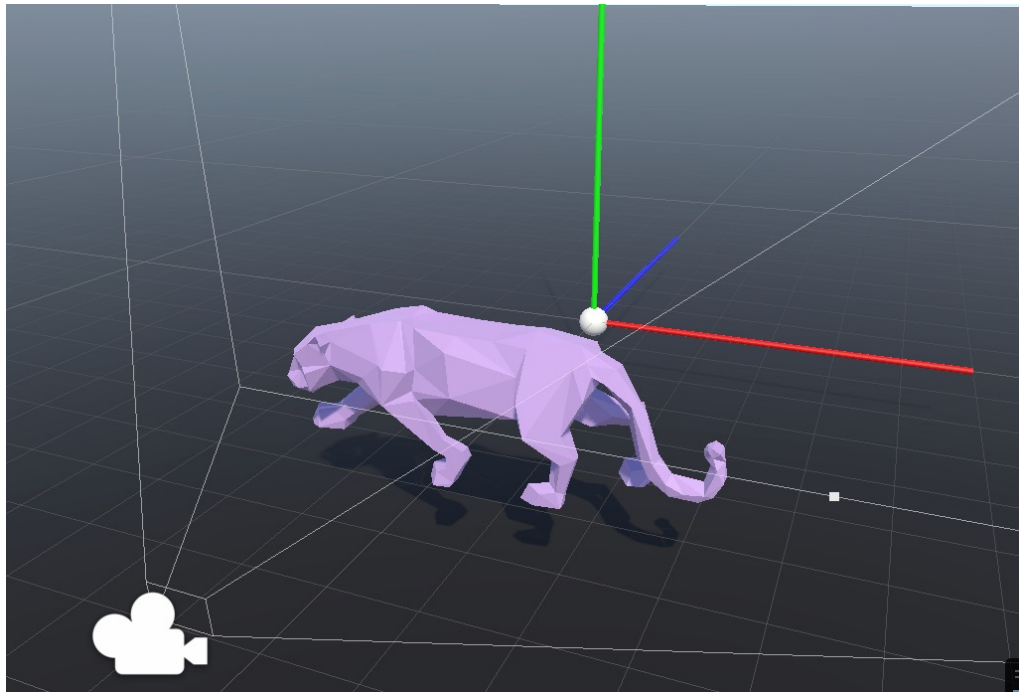
# Graphics APIs

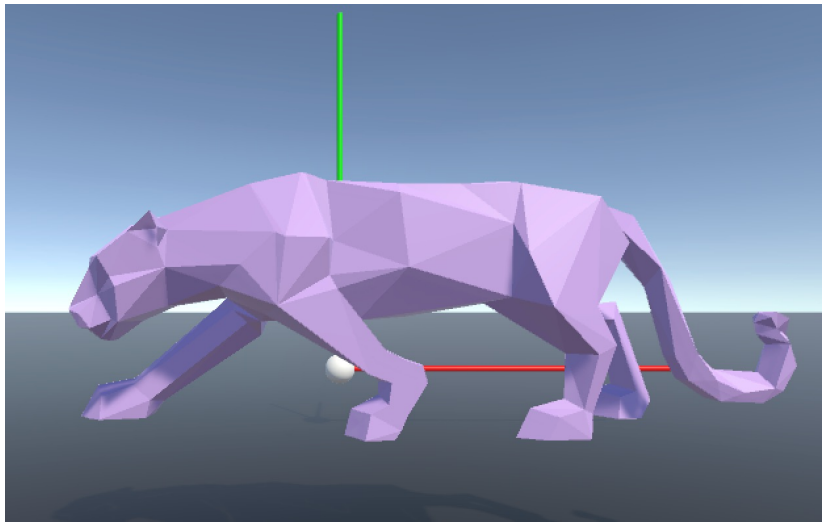
- A general-purpose graphics APIs provide users with **a variety of functions** for creating and manipulating 2D/3D shapes, including
  - **Geometric transformations** – translating/**rotating**/scaling 2D/3D shapes



# Graphics APIs

- A general-purpose graphics APIs provide users with **a variety of functions** for creating and manipulating 2D/3D shapes, including
  - **Viewing transformations** – specifying the view projected onto the screen

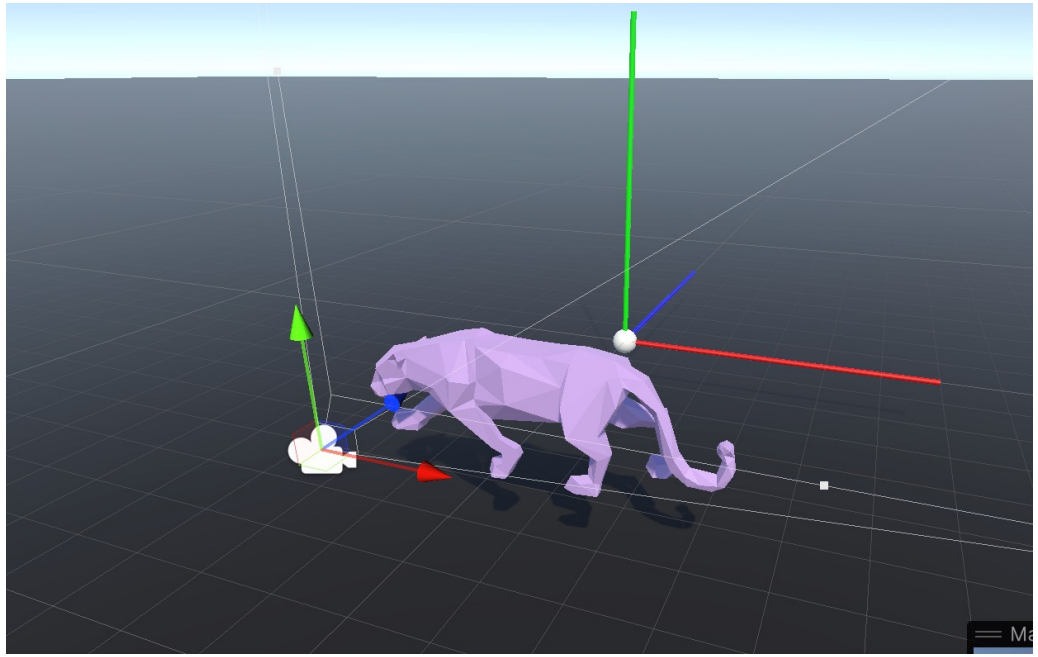


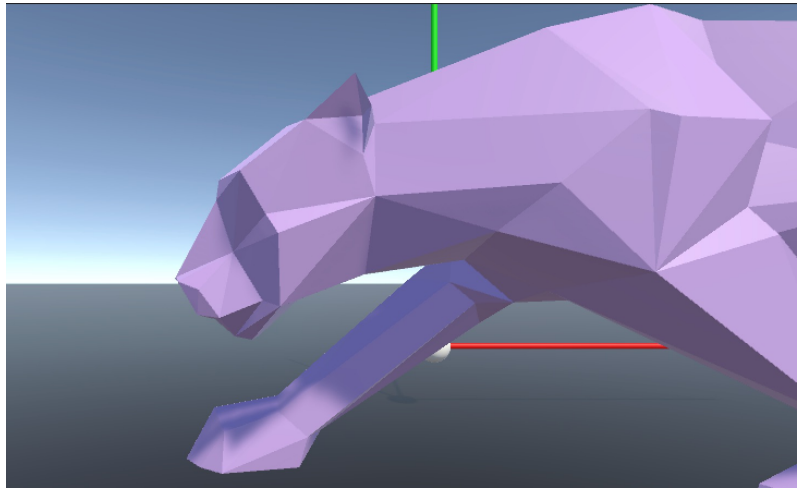


Camera transforms the 3D world to a 2D view

# Graphics APIs

- A general-purpose graphics APIs provide users with **a variety of functions** for creating and manipulating 2D/3D shapes, including
  - **Viewing transformations** – specifying the view projected onto the screen





Transform the camera changes the view

# PyOpenGL & PyGame

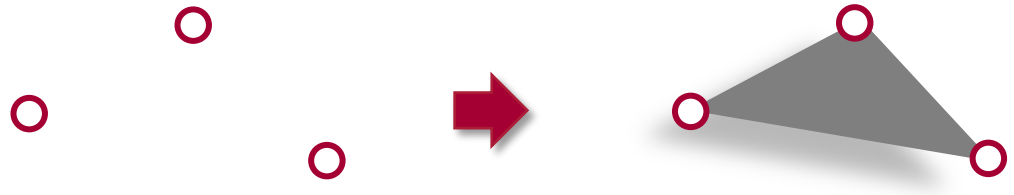
Have you been able to set things up?

# OpenGL Basics

- **Function names** in the OpenGL **basic library** are prefixed with gl
  - glBegin(), glEnd(). glVertex3f(), etc.
- **Arguments** of certain functions are symbolic constants, written in capital letters beginning with GL:
  - e.g., GL_TRIANGLES

**glBegin-glEnd**
building block for
each primitive

```
glBegin(GL_TRIANGLES);
    glVertex3f(x1, y1, z1); // first vertex
    glVertex3f(x2, y2, z2); // second vertex
    glVertex3f(x3, y3, z3); // third vertex
glEnd();
```



# Related Libraries

- **GLU** (OpenGL Utility)
  - GLU functions start with prefix glu
  - Included in **every** OpenGL implementation
  - Provides routines for **setting up viewing and projection matrices**
  - Describes **complex objects with line and polygon approximations**
- **GLUT** (OpenGL Utility Toolkit)
  - GLUT functions start with prefix glut
  - Provides functions for interacting with any **screen-windowing system** (OpenGL cannot create display window by itself)

# PyOpenGL

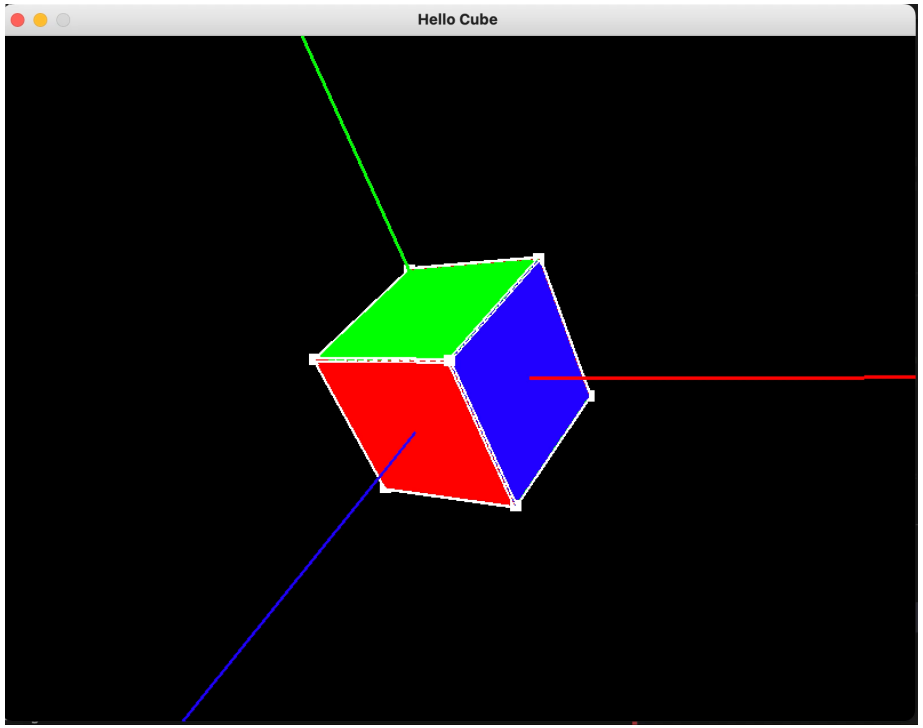
- OpenGL is a cross-platform, cross-language API.
- PyOpenGL is a **Python binding** to OpenGL and its related libraries.
- Easy to install and use
- Most of the functions in PyOpenGL are identical in calling method and functionality to that of the C specification.

# Pygame

- **A set of Python modules designed for writing video games.**
- **Provides access to video, audio, and input devices of a computer**
- **Cross-platform: Windows, MacOS, and Linux**
- **Nice features:**
  - Flexibility in controlling the main loop
  - Simple window creating and managing
  - Easy user interface interactions (keyboard, mouse)
  - Interactive GUI elements (e.g., buttons)

# Demo #1

Let’s draw a spinning cube!

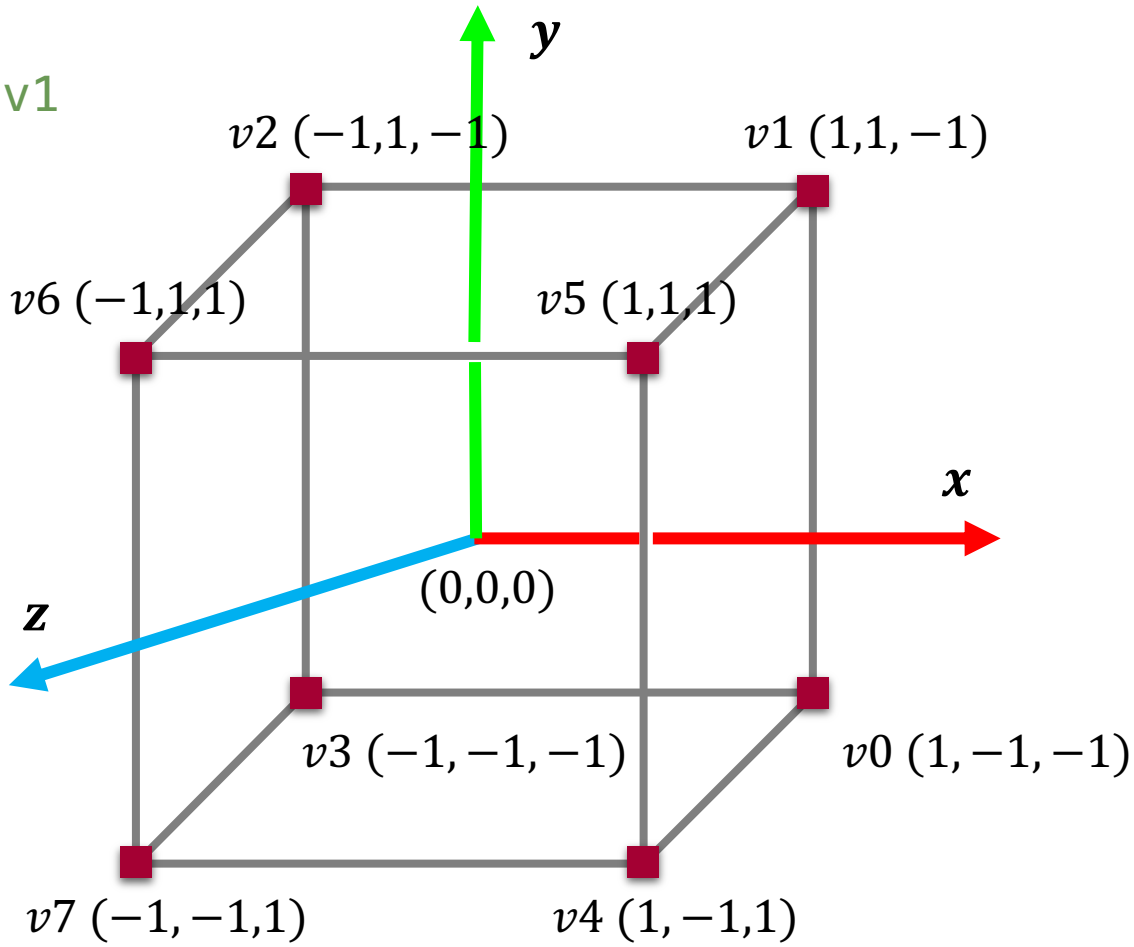


# 3D Mesh Data Structure

```
vertices = (
(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)

faces = (
(0, 1, 5, 4), #f0:v0,v1,v5,v4
(3, 2, 6, 7),
(1, 2, 6, 5),
(0, 3, 7, 4),
(0, 1, 2, 3),
(4, 5, 6, 7))
```

```
edges = (
(0, 1), #e0:v0,v1
(0, 3),
(0, 4),
(2, 1),
(2, 3),
(2, 6),
(7, 3),
(7, 4),
(7, 6),
(5, 1),
(5, 4),
(5, 6))
```



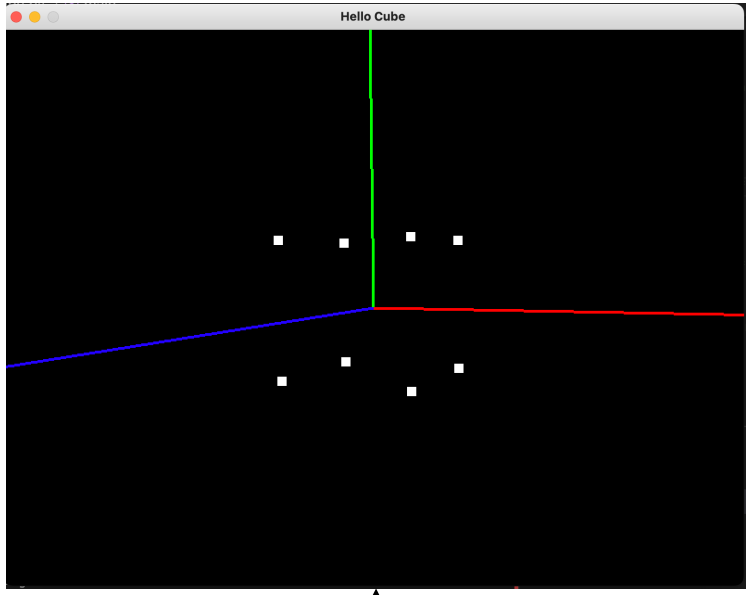
# 3D Cube

- **Vertices**
  - 3D position: vector: $(x, y, z)$
  - GL_POINTS

```
vertices = (
(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)
```

```
glColor3f(1.0, 1.0, 1.0)
glPointSize(10.0)
glBegin(GL_POINTS)
for vertex in vertices:
    glVertex3fv(vertex)
glEnd()
```

```
def Cube():
    # draw vertices
```



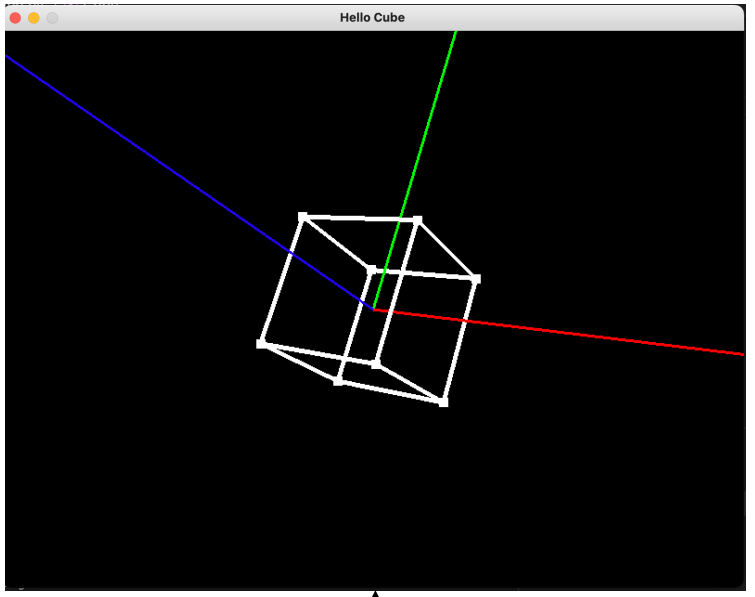
# 3D Cube

- **Edges**
  - Two vertices per edge:
    - $e.g., v0, v1$
  - GL_LINES

```
edges = (
(0, 1), #e0: v0,v1
(0, 3),
(0, 4),
(2, 1),
(2, 3),
(2, 6),
(7, 3),
(7, 4),
(7, 6),
(5, 1),
(5, 4),
(5, 6))
```

```
glColor3f(1.0, 1.0, 1.0)
glLineWidth(5.0)
glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
glEnd()
```

```
def Cube():
    # draw edges
```

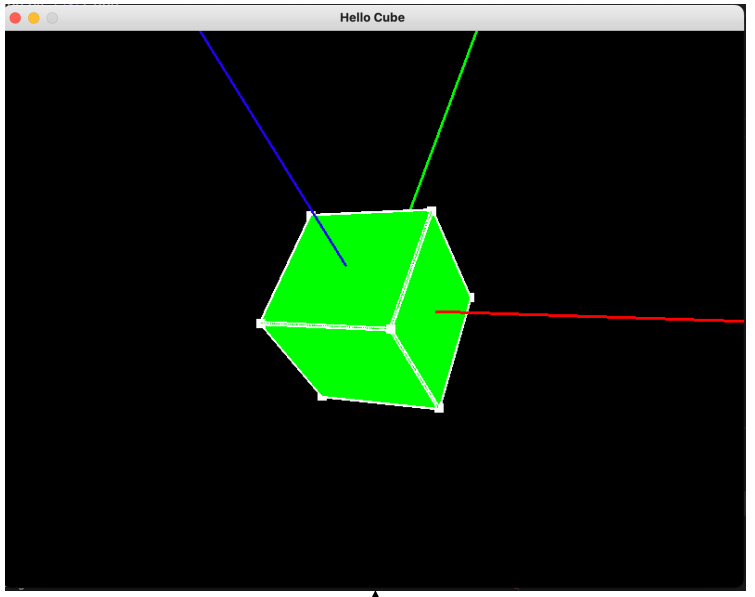


# 3D Cube

- **Faces**
  - Four vertices per face (quad):
    - $e.g., v0, v1, v2, v3$
  - GL_QUADS

```
faces = (
(0, 1, 5, 4), #f0: v0,v1,v5,v4
(3, 2, 6, 7),
(1, 2, 6, 5),
(0, 3, 7, 4),
(0, 1, 2, 3),
(4, 5, 6, 7))
```

```
glColor3f(0.0, 1.0, 0.0)
glBegin(GL_QUADS)
for face in faces:
    for vertex in face:
        glVertex3fv(vertices[vertex])
glEnd()
```

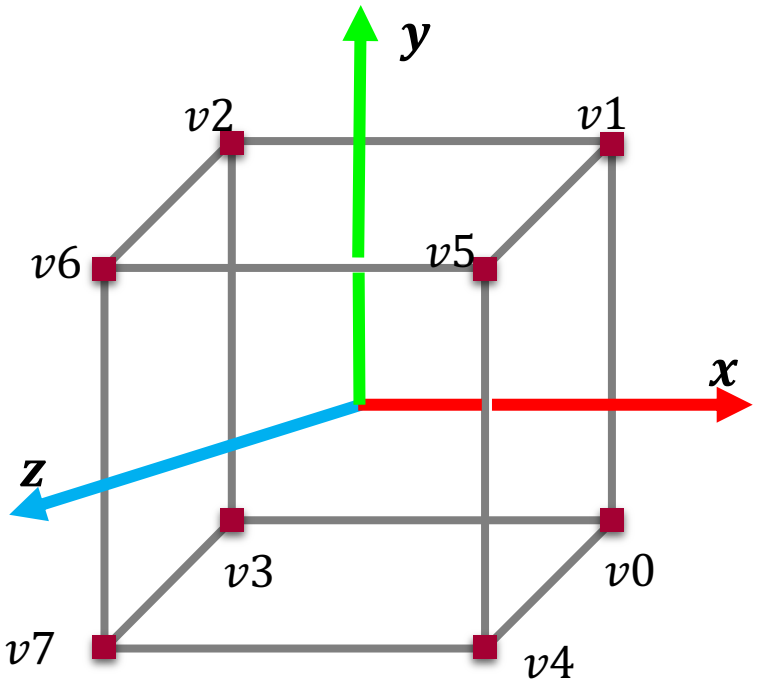



```
def Cube():
    # draw faces
```

# 3D Cube

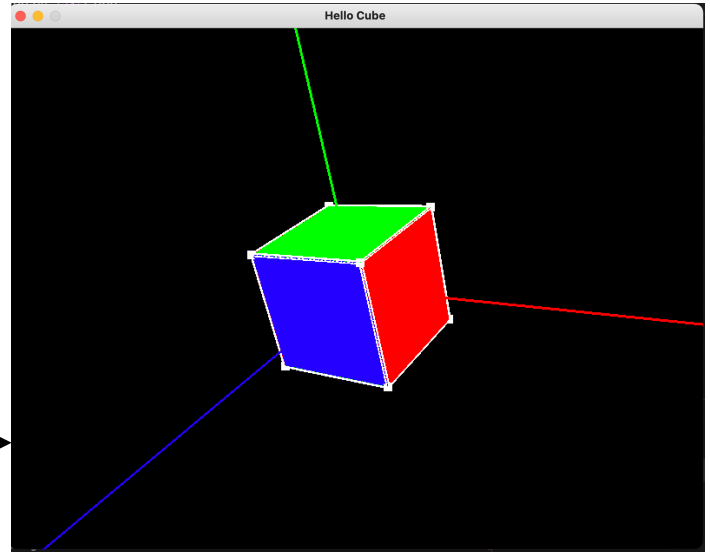
- **Let's make it colorful!**

```
faces = (
(0, 1, 5, 4), #f0 -> red
(3, 2, 6, 7), #f1 -> red
(1, 2, 6, 5), #f2 -> green
(0, 3, 7, 4), #f3 -> green
(0, 1, 2, 3), #f4 -> blue
(4, 5, 6, 7)) #f5 -> blue
```




```
colors = (
(1.0, 0.0, 0.0), #red
(0.0, 1.0, 0.0), #green
(0.0, 0.0, 1.0)  #blue
)
colorCount = 0
glColor3f(0.0, 1.0, 0.0)
glBegin(GL_QUADS)
for face in faces:
    glColor3fv(colors[int(colorCount/2)])
    for vertex in face:
        glVertex3fv(vertices[vertex])
    colorCount += 1
glEnd()
```

```
def Cube():
    # draw faces
```

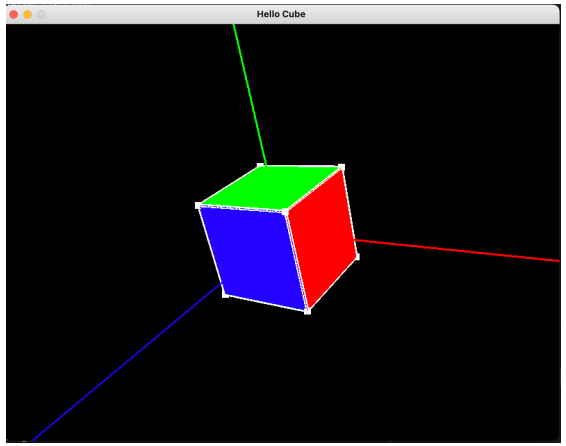




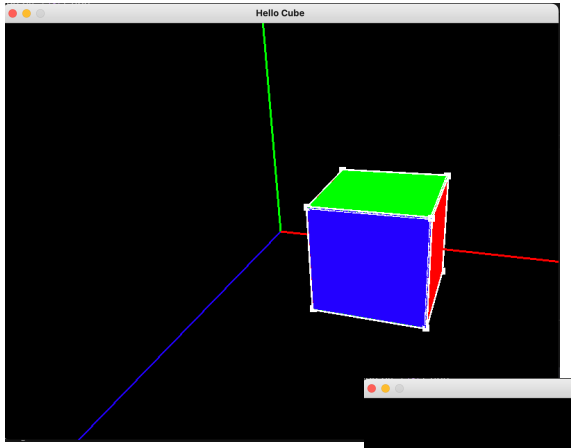

 

# Geometric Transformation

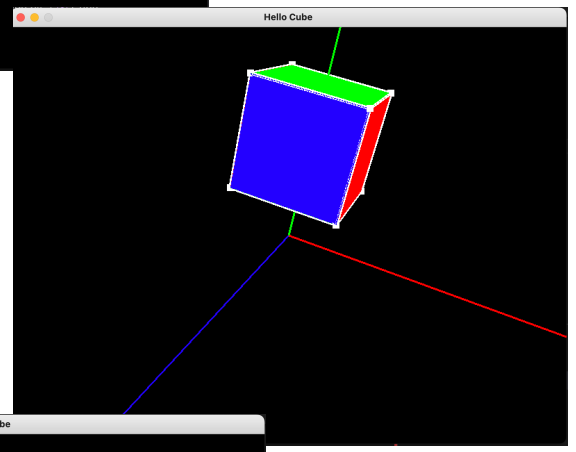
- **Translation: glTranslatef($t_x, t_y, t_z$)**



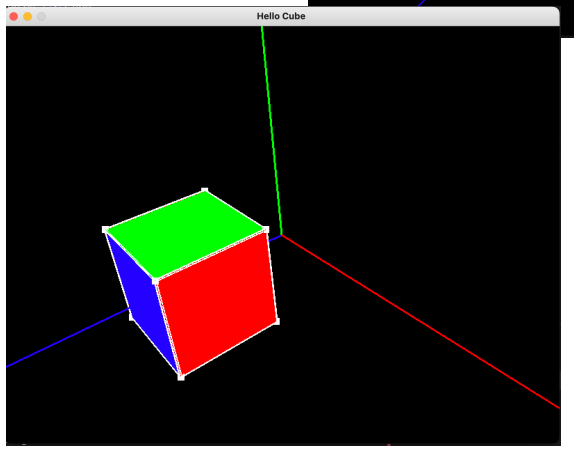

glTranslatef(2, 0, 0)




glTranslatef(0, 2, 0)



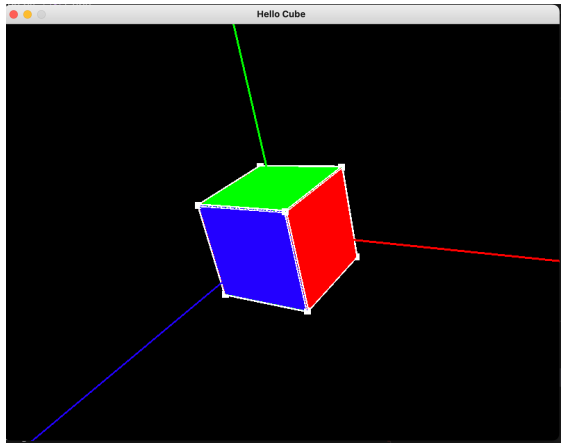

glTranslatef(0, 0, 2)



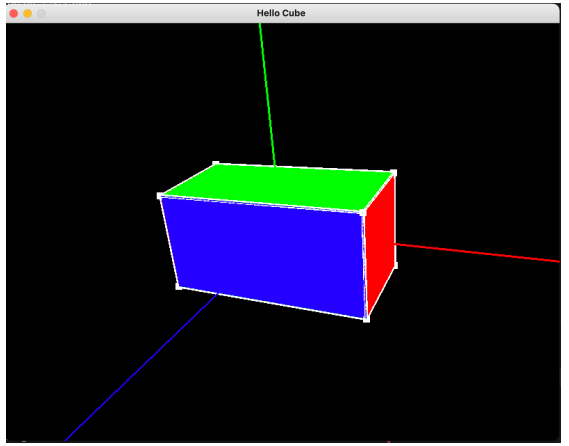

```
def Cube():
    # TODO: transform the cube
```

# Geometric Transformation

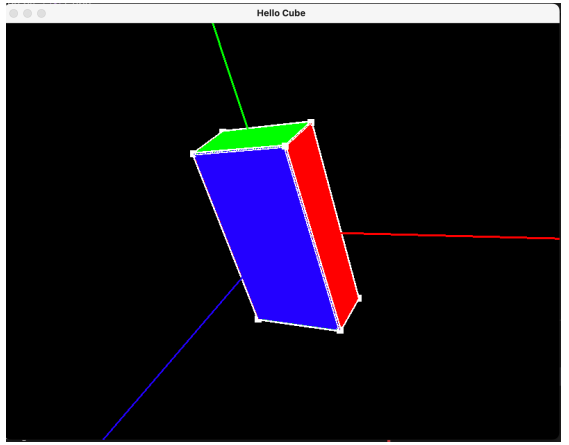
- **Scaling: glScalef($s_x, s_y, s_z$)**




glScalef(2, 1, 1)



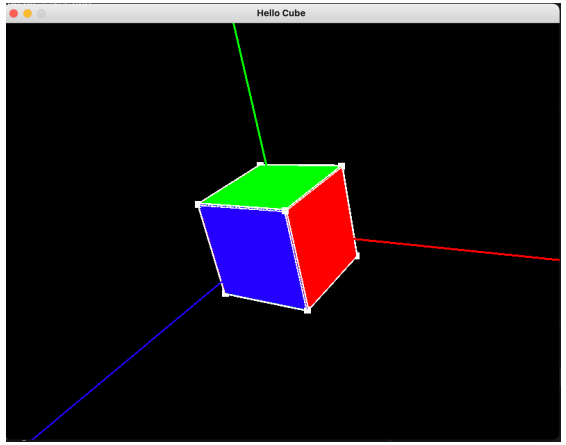

glScalef(1, 2, 0.5)



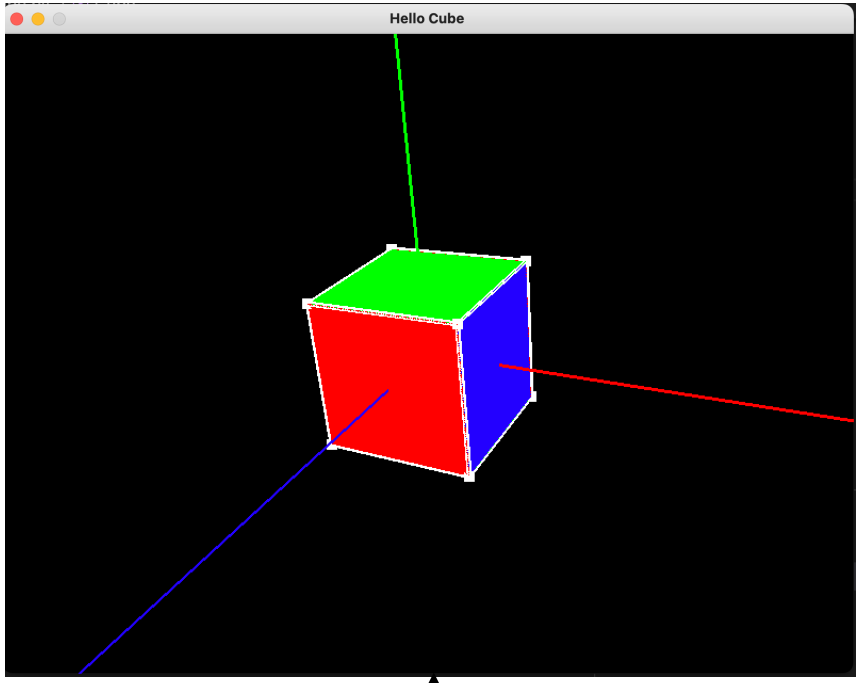

```
def Cube():
    # TODO: transform the cube
```

# Geometric Transformation

- **Rotation: glRotatef($angle, x, y, z$)**



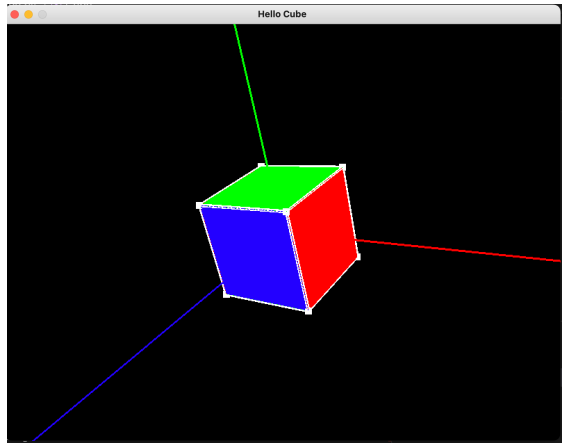

**glRotatef(90, 0, 1, 0)**

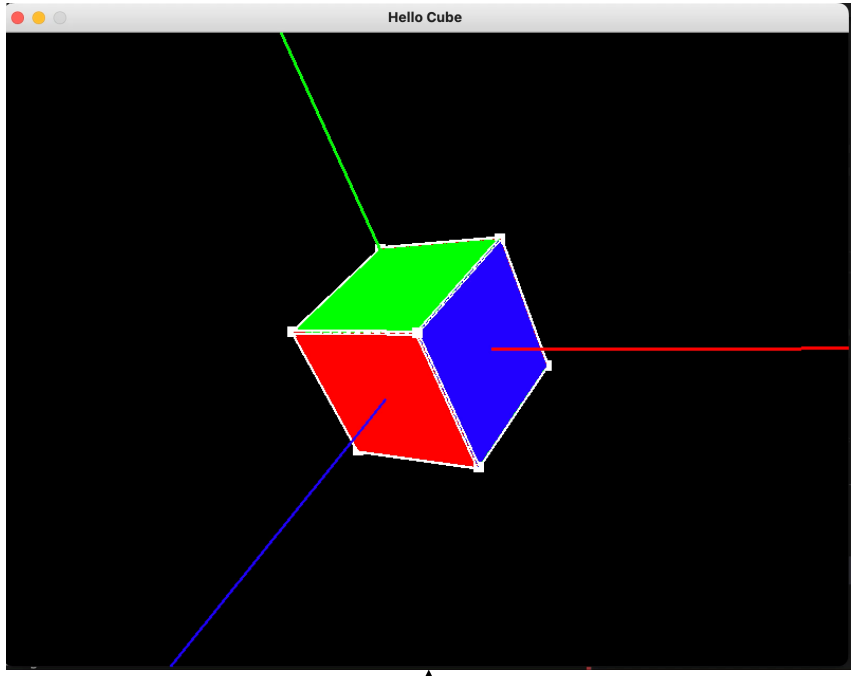



```
def Cube():
    # TODO: transform the cube
```

# Geometric Transformation

- **Rotation: glRotatef($angle, x, y, z$)**



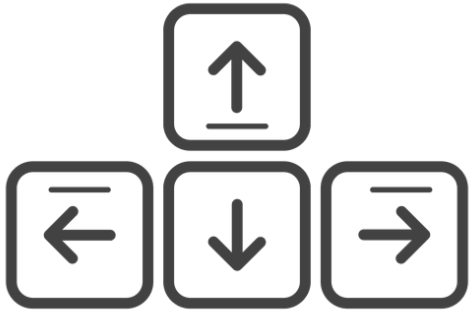



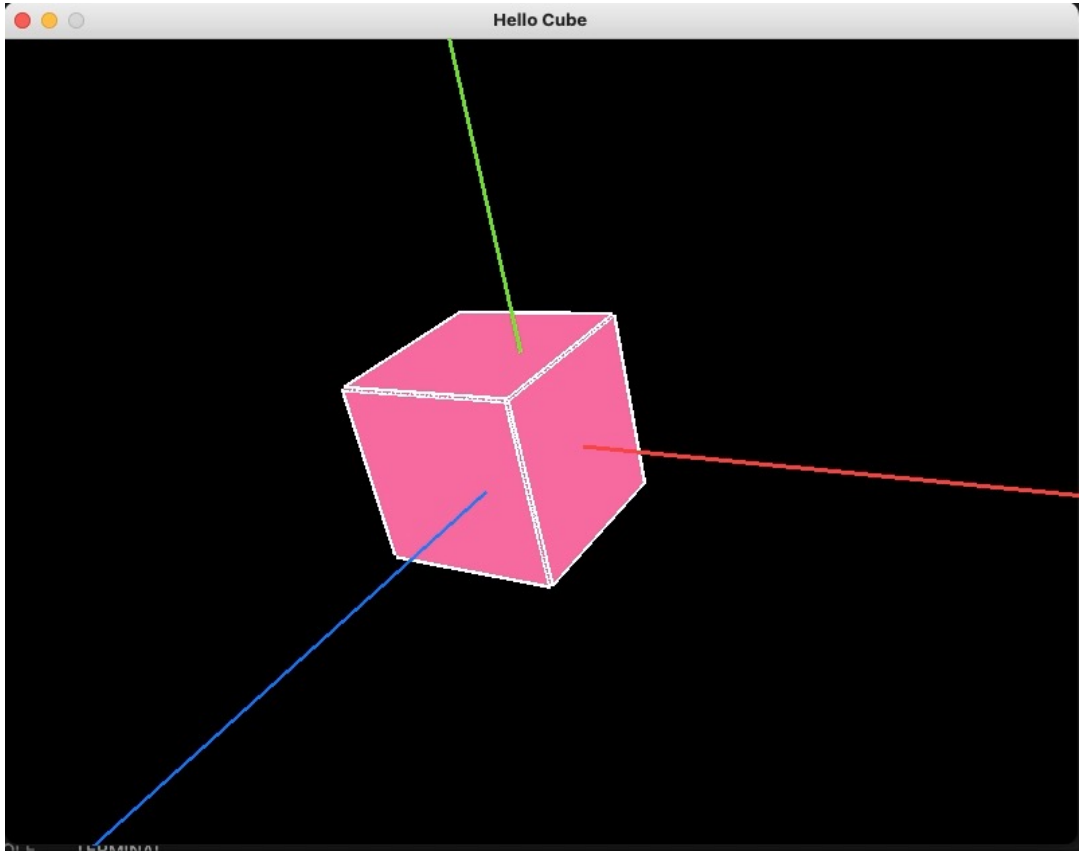

```
global rotAngle
rotAngle += 1
glRotatef(rotAngle, 0, 1, 0)
```

```
def Cube():
    # TODO: transform the cube
```

# Demo #2

Moveable cube – keyboard control





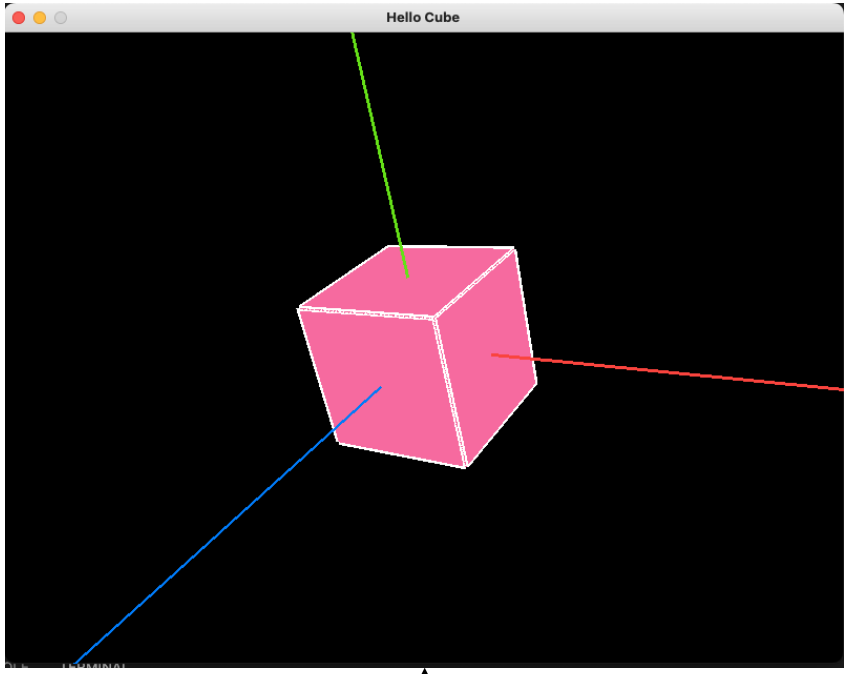
# Movable Cube

- **Now, let's create a new cube**

```
def movableCube():
    glPushMatrix()

    # draw faces (quads)
    glColor3f(1.0, 0.2, 0.6)
    glBegin(GL_QUADS)
    for face in faces:
        for vertex in face:
            glVertex3fv(vertices[vertex])
    glEnd()

    # draw edges (lines)
    glColor3f(1.0, 1.0, 1.0)
    glLineWidth(5.0)
    glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()
    glPopMatrix()
```

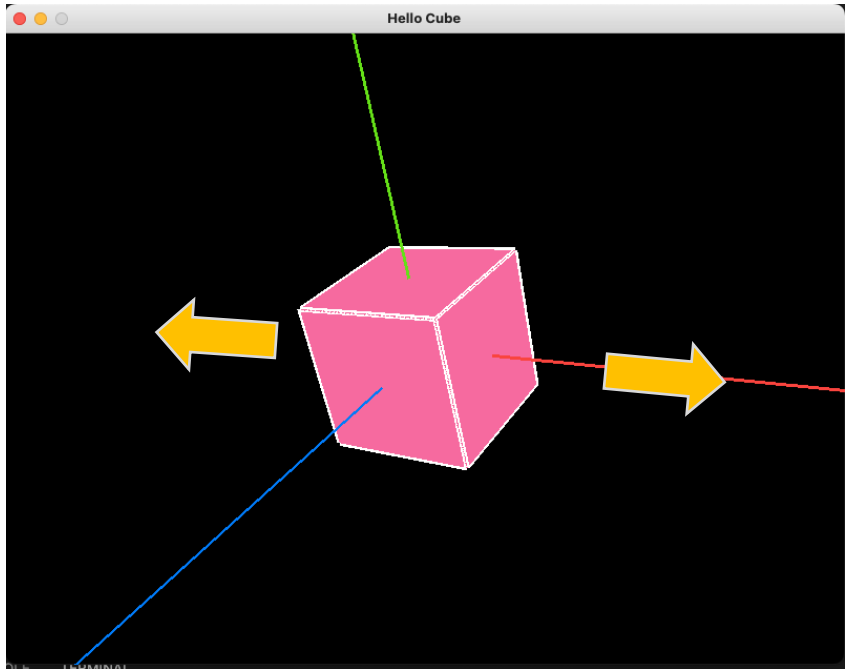



```
def draw():
    ...
    # TODO: create a movable cube
    movableCube()
```

# Movable Cube

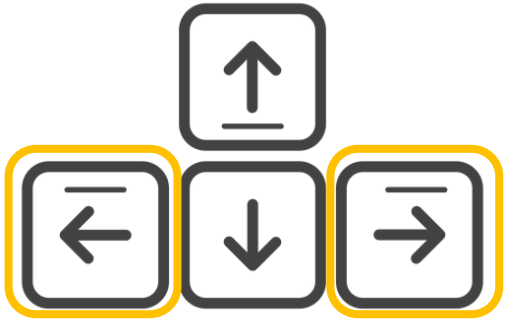
- **Keyboard input event**
  - **Left / right**

```
while True:
    ...
    # TODO: keyboard controls "movableCube"
    if event.key == pygame.K_LEFT:
        translateVec[0] += -0.5
    if event.key == pygame.K_RIGHT:
        translateVec[0] += 0.5
```




**Recall: glTranslatef($t_x, t_y, t_z$)**

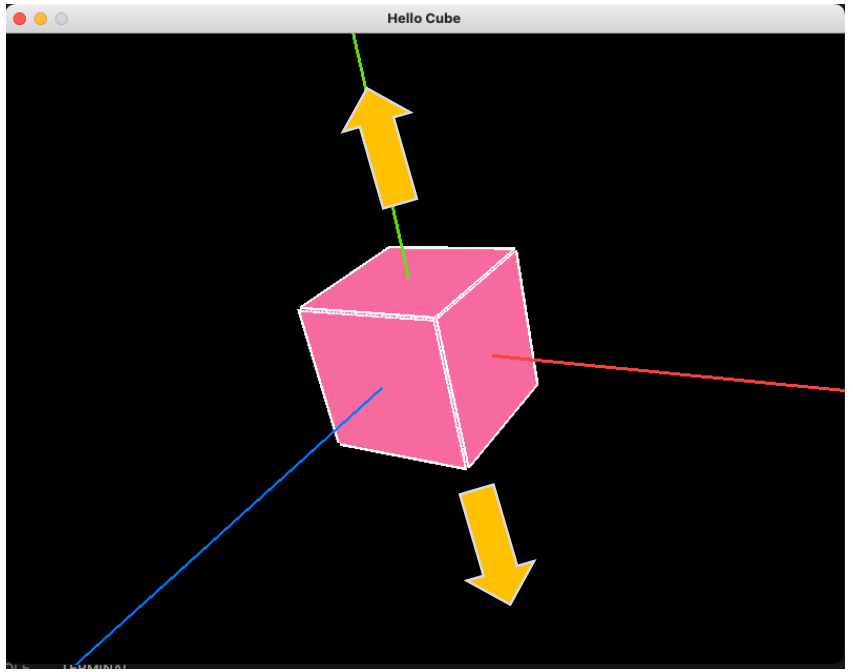
```
translateVec = [0.0, 0.0, 0.0]
```

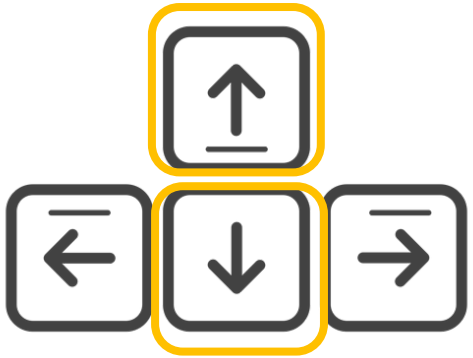


# Movable Cube

- **Keyboard input event**
  - **Down / Up**

```
while True:
    ...
    # TODO: keyboard controls "movableCube"
    if event.key == pygame.K_DOWN:
        translateVec[1] += -0.5
    if event.key == pygame.K_UP:
        translateVec[1] += 0.5
```



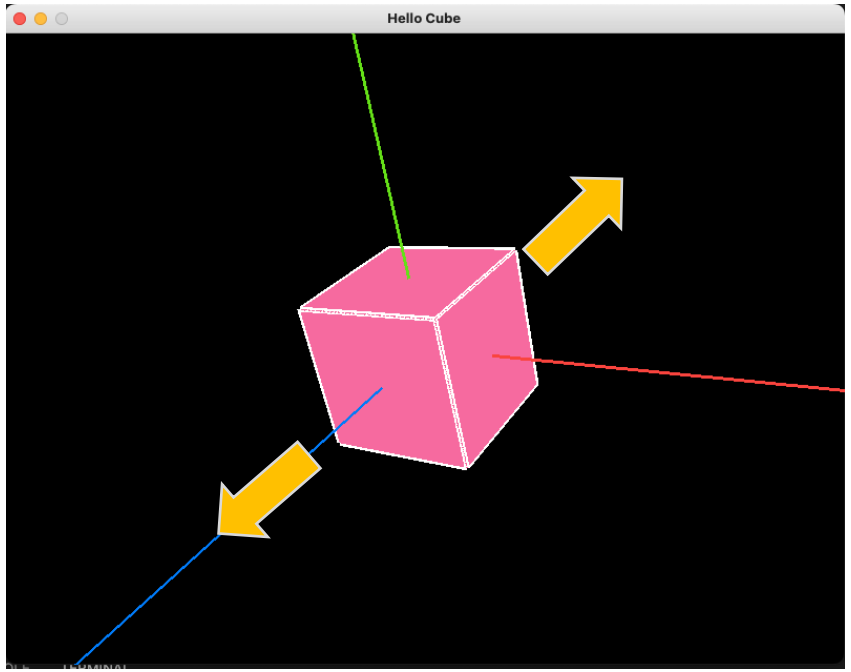
**Recall: glTranslatef($t_x, t_y, t_z$)**

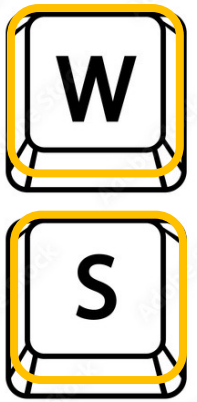
```
translateVec = [0.0, 0.0, 0.0]
```

# Movable Cube

- **Keyboard input event**
  - **Backward / Forward**

```
while True:
    ...
    # TODO: keyboard controls "movableCube"
    if event.key == pygame.K_w:
        translateVec[2] += -0.5
    if event.key == pygame.K_s:
        translateVec[2] += 0.5
```




**Recall: glTranslatef($t_x, t_y, t_z$)**

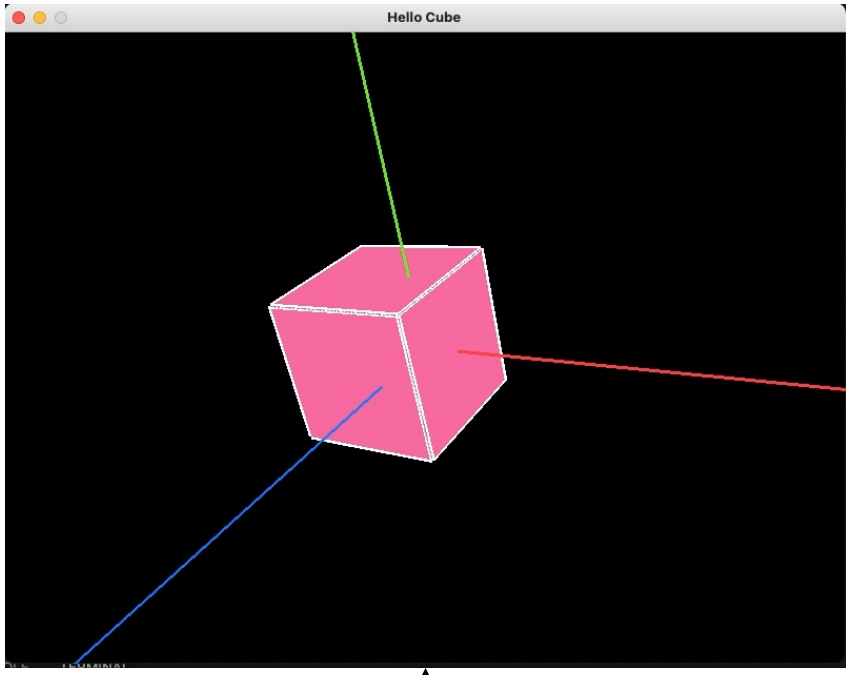
```
translateVec = [0.0, 0.0, 0.0]
```

# Movable Cube

- **Translate the cube based on keyboard input**

```
def movableCube():
    glPushMatrix()
    # TODO: rotate the cube based on the
    keyboard input
    glTranslatef(translateVec[0],
                 translateVec[1],
                 translateVec[2])

    ...
```



# Thanks for attending!

Complete code will be uploaded to https://github.com/trudiQ/CG-Workshop.git shortly.