Computer Graphics

8 - Lighting & Shading 2, Hierarchical Modeling

Yoonsang Lee
Spring 2022

Midterm Exam Announcement (Again)

- Date & time: **Apr 27**, 09:30 - 10:30 am
- Place: IT.BT, 508
- Scope: Lecture 2 ~ 7 **(up to last week's lecture)**

- You cannot leave the room until the end of the exam even if you finish the exam earlier.

- Please bring your student ID card to the exam.

- **If you are unable to take the offline exam** (stay abroad, corona confirmed, etc.), please contact the TA in advance.
  - Chaejun Sohn (손채준 조교), thscowns@gmail.com
  - You must inform the TA **at least two days before the exam**.

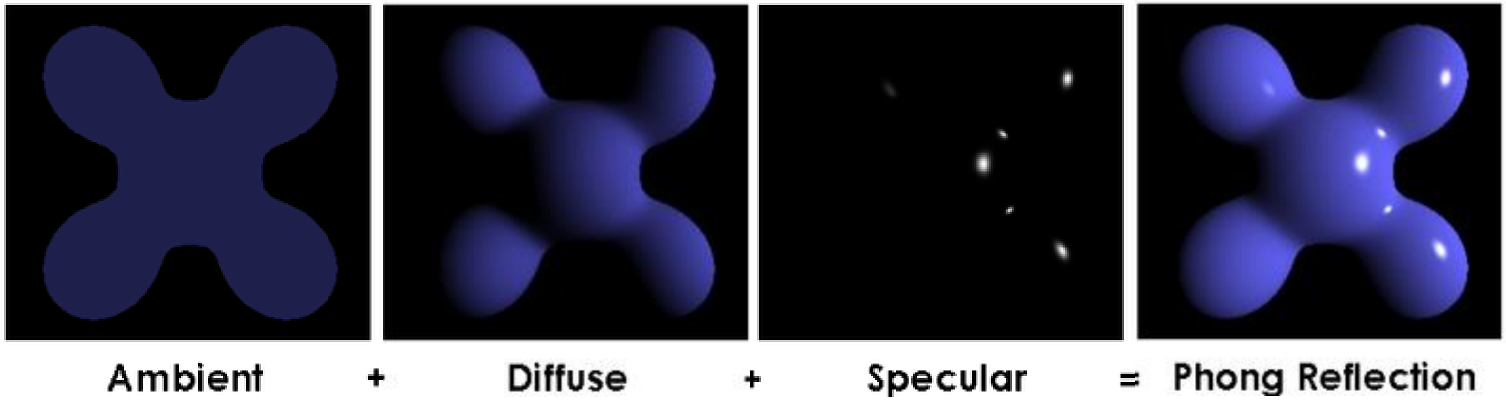
Topics Covered

- Lighting & Shading in OpenGL
- Interpretation of Composite Transformations
- Hierarchical Modeling
    - Concept of Hierarchical Modeling
    - OpenGL Matrix Stack

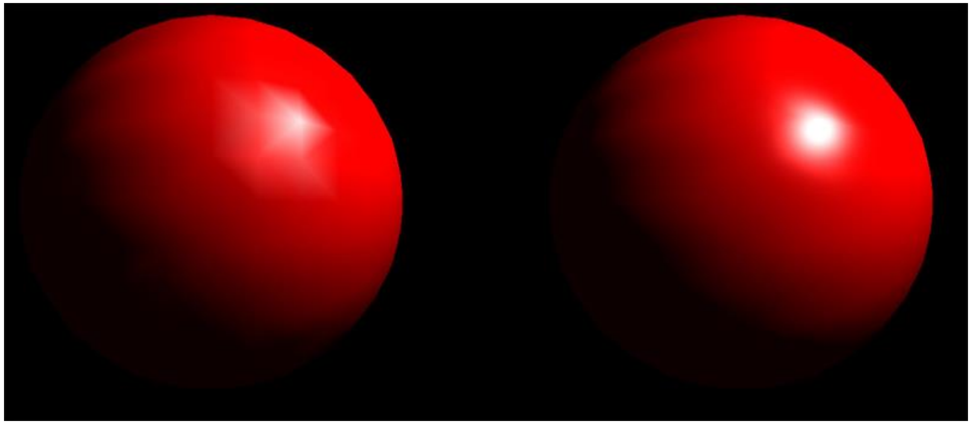
Lighting & Shading in OpenGL

Recall for Lighting & Shading

- Phong Illumination Model



- Shading



Gouraud shading Phong shading

To do Lighting & Shading in OpenGL,

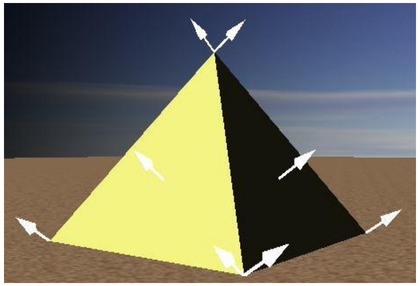
- First, you need to set vertex normal.

- Recall that a vertex has these attributes:
  - Vertex coordinate : specified by glVertex*()
  - Vertex color : specified by glColor*()
  - **Normal vector : specified by glNormal*()**
  - Texture coordinate : specified by glTexCoord*()

Shading in OpenGL

- The shading method is determined by the vertex normal vectors you specify.

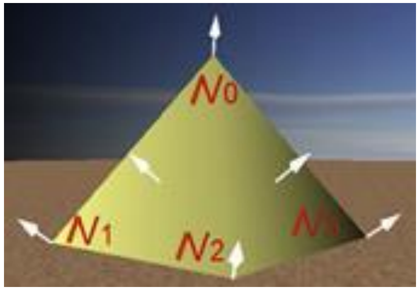
- Flat shading: Set each vertex normal to the face normal the vertex belongs to.



The normal at a vertex is the same as the face normal. Therefore, each vertex has as many normals as the number of faces it belongs.

Shading in OpenGL

- Gouraud shading: Set each vertex normal to the average of normals of all faces sharing the vertex.



Only one vertex normal per vertex; average of face normals of the faces the vertex is part of

- Phong shading is not available in legacy OpenGL.

Setting Vertex Normals in OpenGL

- You can specify normals using glNormal*() or a vertex array

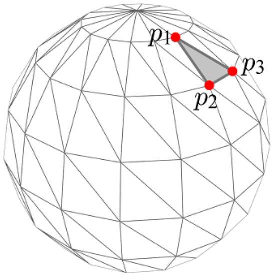
```
glBegin(GL_TRIANGLES)

glNormal3f(0,0,1) # v0,v2,v1,v0,v3,v2 normal
glVertex3f( -1 ,  1 ,  1 ) # v0 position
glVertex3f(  1 , -1 ,  1 ) # v2 position
glVertex3f(  1 ,  1 ,  1 ) # v1 position

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

glNormal3f(0,0,-1)
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
```

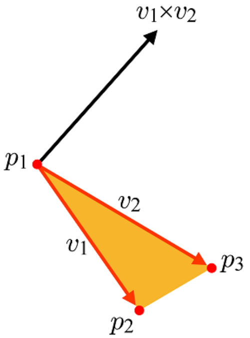
```
varr = np.array([
        (0,0,1),            # v0 normal
        ( -1 ,  1 ,  1 ), # v0 position
        (0,0,1),            # v2 normal
        (  1 , -1 ,  1 ), # v2 position
        (0,0,1),            # v1 normal
        (  1 ,  1 ,  1 ), # v1 position

        (0,0,1),            # v0 normal
        ( -1 ,  1 ,  1 ), # v0 position
        (0,0,1),            # v3 normal
        ( -1 , -1 ,  1 ), # v3 position
        (0,0,1),            # v2 normal
        (  1 , -1 ,  1 ), # v2 position

        (0,0,-1),
        ( -1 ,  1 , -1 ), # v4
        (0,0,-1),
        (  1 ,  1 , -1 ), # v5
        (0,0,-1),
        (  1 , -1 , -1 ), # v6
# ...
], 'float32')
```

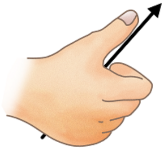
Setting Vertex Normals in OpenGL

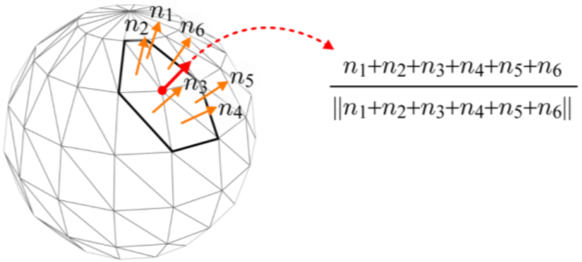
- You can hard-code normals like prev. page

- or compute normals from vertex positions









$$\frac{n_1+n_2+n_3+n_4+n_5+n_6}{\|n_1+n_2+n_3+n_4+n_5+n_6\|}$$

- or read normals from a model file such as .obj (most common case)

Lighting in OpenGL

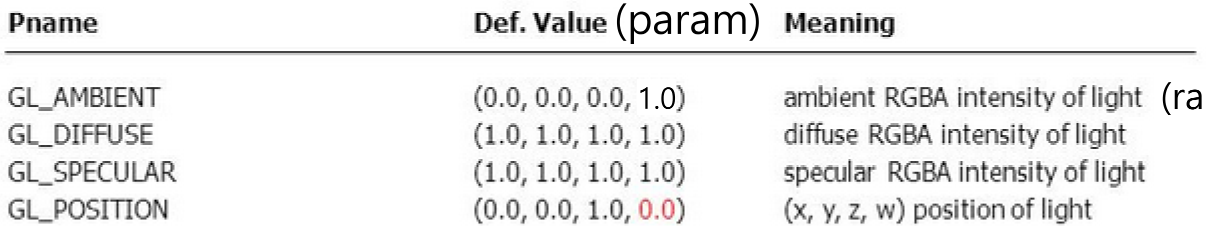
- Lighting in legacy OpenGL is too restrictive.
  - Only Blinn-Phong illumination model is available.

- **glEnable(GL_LIGHTING)**
  - Enable lighting

- **glEnable(GL_LIGHT0)**
  - Enable 0th light. You can use eight lights in legacy OpenGL (GL_LIGHT0 ~ GL_LIGHT7)

glLightfv()

- **glLightfv(light, pname, param)**
  - **light**: the light to assign
    - GL_LIGHT0 ~ GL_LIGHT7
  - **pname**, **param**: light properties such as light intensity and position



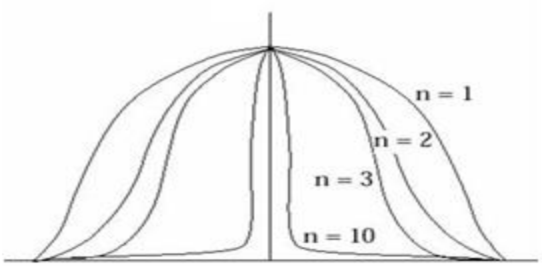
Pname	**Def. Value** (param)	**Meaning**
GL_AMBIENT	(0.0, 0.0, 0.0, 1.0)	ambient RGBA intensity of light (ranging from 0.0 to 1.0)
GL_DIFFUSE	(1.0, 1.0, 1.0, 1.0)	diffuse RGBA intensity of light
GL_SPECULAR	(1.0, 1.0, 1.0, 1.0)	specular RGBA intensity of light
GL_POSITION	(0.0, 0.0, 1.0, 0.0)	(x, y, z, w) position of light

w=0: directional light
w=1: point light
(homogeneous coordinates)

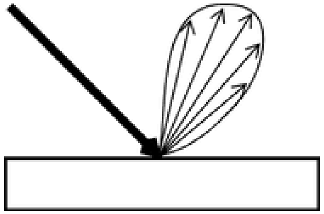
glMaterialfv()

- **glMaterialfv(face, pname, param)**
  - **face**: the face type to assign
    - GL_FRONT, GL_BACK, or GL_FRONT_AND_BACK
  - **pname**, **param**: material reflectance for each color channel
    - GL_AMBIENT, GL_DIFFUSE, GL_SPECULAR
    - GL_AMBIENT_AND_DIFFUSE
    - GL_SHININESS: Specular exponent (shininess coefficient) (0 ~ 128)

$$I = C_s k_s \cos^{\boxed{n}}(\alpha)$$



Specular falloff of (cos δ) ^{n}



Good Settings for glLightfv() & glMaterialfv()

- glLightfv()
  - GL_DIFFUSE & GL_SPECULAR: Color of the light source
  - GL_AMBIENT: The same color, but at much reduced intensity (about 10%)

- glMaterialfv()
  - GL_DIFFUSE & GL_AMBIENT: Color of the object
  - GL_SPECULAR: White (1,1,1,1)

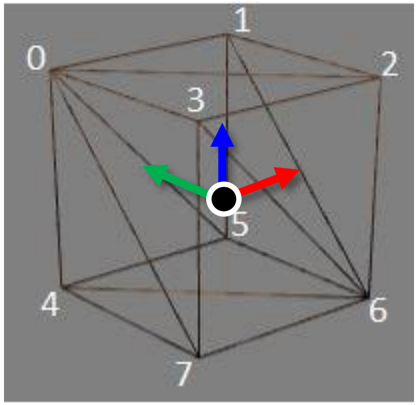
- Final color is the sum of ambient, diffuse, specular components, and
- each component is formed by **multiplying the glMaterial color by the glLight color for each color channel**.

Reference: https://www.khronos.org/opengl/wiki/How_lighting_works#glMaterial_and_glLight

Normals with Lighting

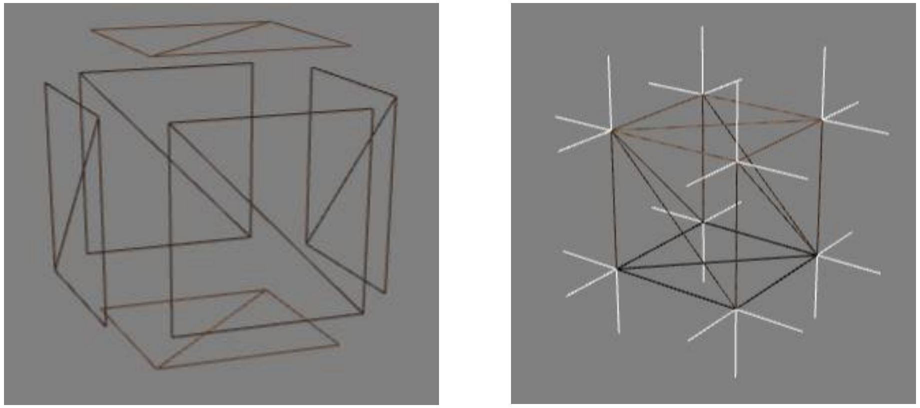
- In OpenGL, normal vectors should have *unit length*.
- Normal vectors are transformed by GL_MODELVIEW matrix, so they may not have unit length, especially if scaling are included.
- You need to use one of these:
  - **glEnable(GL_NORMALIZE)**
    - Automatically normalize normal vectors after model-view transformation
  - **glEnable(GL_RESCALE_NORMAL)**
    - More efficient, but normal vectors must be initially supplied as unit vectors and only works for uniform scaling

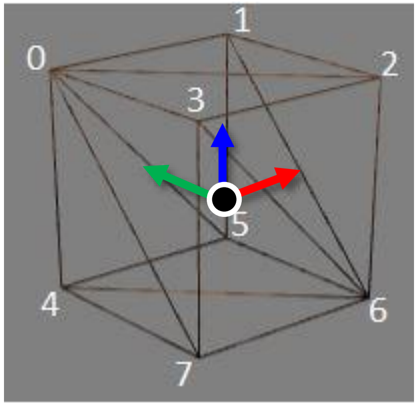
Example: a cube of length 2 again



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)

Normals of the Cube for Flat Shading





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

[Practice] OpenGL Lighting

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

gCamAng = 0.
gCamHeight = 1.

def drawCube_glVertex():
    glBegin(GL_TRIANGLES)

    glNormal3f(0,0,1) # v0, v2, v1, v0, v3, v2 
normal
    glVertex3f( -1 ,  1 ,  1 ) # v0 position
    glVertex3f(  1 , -1 ,  1 ) # v2 position
    glVertex3f(  1 ,  1 ,  1 ) # v1 position

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

    glNormal3f(0,0,-1)
    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
```

```
    glNormal3f(0,1,0)
    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

    glNormal3f(0,-1,0)
    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

    glNormal3f(1,0,0)
    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

    glNormal3f(-1,0,0)
    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()
```

```
def createVertexArraySeparate():
    varr = np.array([
            (0,0,1),         # v0 normal
            ( -1 ,  1 ,  1 ), # v0 position
            (0,0,1),         # v2 normal
            (  1 , -1 ,  1 ), # v2 position
            (0,0,1),         # v1 normal
            (  1 ,  1 ,  1 ), # v1 position

            (0,0,1),         # v0 normal
            ( -1 ,  1 ,  1 ), # v0 position
            (0,0,1),         # v3 normal
            ( -1 , -1 ,  1 ), # v3 position
            (0,0,1),         # v2 normal
            (  1 , -1 ,  1 ), # v2 position

            (0,0,-1),
            ( -1 ,  1 , -1 ), # v4
            (0,0,-1),
            (  1 ,  1 , -1 ), # v5
            (0,0,-1),
            (  1 , -1 , -1 ), # v6

            (0,0,-1),
            ( -1 ,  1 , -1 ), # v4
            (0,0,-1),
            (  1 , -1 , -1 ), # v6
            (0,0,-1),
            ( -1 , -1 , -1 ), # v7

            (0,1,0),
            ( -1 ,  1 ,  1 ), # v0
            (0,1,0),
            (  1 ,  1 ,  1 ), # v1
            (0,1,0),
            (  1 ,  1 , -1 ), # v5

            (0,1,0),
            ( -1 ,  1 ,  1 ), # v0
            (0,1,0),
            (  1 ,  1 , -1 ), # v5
            (0,1,0),
            ( -1 ,  1 , -1 ), # v4

            (0,-1,0),
            ( -1 , -1 ,  1 ), # v3
            (0,-1,0),
            (  1 , -1 , -1 ), # v6
            (0,-1,0),
            (  1 , -1 ,  1 ), # v2

            (0,-1,0),
            ( -1 , -1 ,  1 ), # v3
            (0,-1,0),
            ( -1 , -1 , -1 ), # v7
            (0,-1,0),
            (  1 , -1 , -1 ), # v6

            (1,0,0),
            (  1 ,  1 ,  1 ), # v1
            (1,0,0),
            (  1 , -1 ,  1 ), # v2
            (1,0,0),
            (  1 , -1 , -1 ), # v6

            (1,0,0),
            (  1 ,  1 ,  1 ), # v1
            (1,0,0),
            (  1 , -1 , -1 ), # v6
            (1,0,0),
            (  1 ,  1 , -1 ), # v5

            (-1,0,0),
            ( -1 ,  1 ,  1 ), # v0
            (-1,0,0),
            ( -1 , -1 , -1 ), # v7
            (-1,0,0),
            ( -1 , -1 ,  1 ), # v3

            (-1,0,0),
            ( -1 ,  1 ,  1 ), # v0
            (-1,0,0),
            ( -1 ,  1 , -1 ), # v4
            (-1,0,0),
            ( -1 , -1 , -1 ), # v7
            ], 'float32')
    return varr
```

```
def drawCube_glDrawArray():
    global gVertexArraySeparate
    varr = gVertexArraySeparate
    glEnableClientState(GL_VERTEX_ARRAY)
    glEnableClientState(GL_NORMAL_ARRAY)
    glNormalPointer(GL_FLOAT, 6*varr.itemsize, varr)
    glVertexPointer(3, GL_FLOAT, 6*varr.itemsize,
ctypes.c_void_p(varr.ctypes.data + 3*varr.itemsize))
    glDrawArrays(GL_TRIANGLES, 0, int(varr.size/6))
```

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

glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)

    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
    gluPerspective(45, 1, 1,10)

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()

gluLookAt(5*np.sin(gCamAng),gCamHeight,5*np.cos(gCamAng), 0,0,0, 0,1,0)

    drawFrame()

    glEnable(GL_LIGHTING)   # try to comment out: no lighting
    glEnable(GL_LIGHT0)

    glEnable(GL_NORMALIZE)  # try to comment out: lighting will be incorrect if you scale the object

    # light position
    glPushMatrix()

    # glRotatef(t*(180/np.pi),0,1,0)  # try to uncomment: rotate light
    lightPos = (3.,4.,5.,1.)    # try to change 4th element to 0. or 1.
```

```
    glLightfv(GL_LIGHT0, GL_POSITION, lightPos)
    glPopMatrix()

 # light intensity for each color channel
    lightColor = (1.,1.,1.,1.)
    ambientLightColor = (.1,.1,.1,1.)
    glLightfv(GL_LIGHT0, GL_DIFFUSE, lightColor)
    glLightfv(GL_LIGHT0, GL_SPECULAR, lightColor)
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambientLightColor)

    # material reflectance for each color channel
    objectColor = (1.,0.,0.,1.)
    specularObjectColor = (1.,1.,1.,1.)
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, objectColor)
    glMaterialfv(GL_FRONT, GL_SHININESS, 10)
    glMaterialfv(GL_FRONT, GL_SPECULAR, specularObjectColor)

    glPushMatrix()
    # glRotatef(t*(180/np.pi),0,1,0)    # try to uncomment: rotate object
    # glScalef(1.,.2,1.)    # try to uncomment: scale object

    glColor3ub(0, 0, 255) # glColor*() is ignored if lighting is enabled

    # drawCube_glVertex()
    drawCube_glDrawArray()
    glPopMatrix()

    glDisable(GL_LIGHTING)
```

```
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
```

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

    if not glfw.init():
        return
    window =
glfw.create_window(640,640,'Lecture13',
None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window,
key_callback)
    glfw.swap_interval(1)

    gVertexArraySeparate =
createVertexArraySeparate()

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

    glfw.terminate()

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

glNormalPointer()

- **glNormalPointer(type, stride, pointer)**
- : specifies the location and data format of an array of normals
  - **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 normal
  - **pointer**: The pointer to the first coordinate of the first normal in the array

- **c.f.) 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

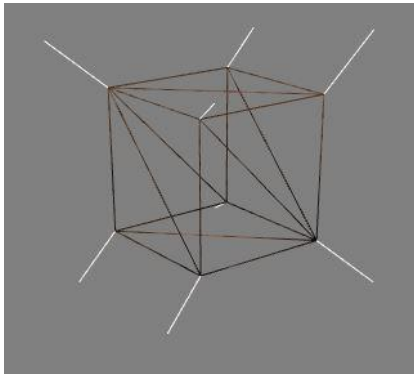
Quiz #1

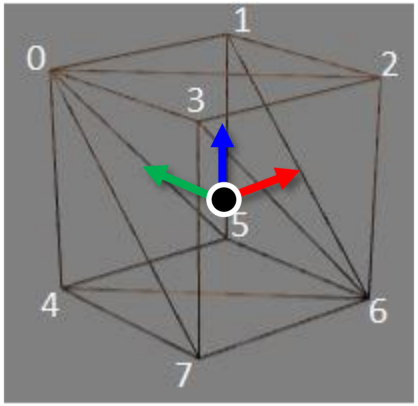
- Go to https://www.slido.com/
- Join #**cg-ys**
- Click “Polls”

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

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

Normals of the Cube for Smooth Shading





vertex index	position	normal
0	(-1 , 1 , 1)	(-0.5773502691896258 , 0.5773502691896258 , 0.5773502691896258)
1	(1 , 1 , 1)	(0.8164965809277261 , 0.4082482904638631 , 0.4082482904638631)
2	(1 , -1 , 1)	(0.4082482904638631 , -0.4082482904638631 , 0.8164965809277261)
3	(-1 , -1 , 1)	(-0.4082482904638631 , -0.8164965809277261 , 0.4082482904638631)
4	(-1 , 1 , -1)	(-0.4082482904638631 , 0.4082482904638631 , -0.8164965809277261)
5	(1 , 1 , -1)	(0.4082482904638631 , 0.8164965809277261 , -0.4082482904638631)
6	(1 , -1 , -1)	(0.5773502691896258 , -0.5773502691896258 , -0.5773502691896258)
7	(-1 , -1 , -1)	(-0.8164965809277261 , -0.4082482904638631 , -0.4082482904638631)

Lighting in Modern OpenGL

- Legacy OpenGL
  - Only allows Gouraud shading & Blinn-Phong illumination model.
  - Rendering quality is not good.

- Modern OpenGL:
  - No specific lighting & shading model in modern OpenGL
  - Programmers have to implement Phong or other illumination model in vertex shader or fragment shader.
  - Example: the shader code in this online demo http://www.cs.toronto.edu/~jacobson/phong-demo/

Interpretation of Composite Transformations

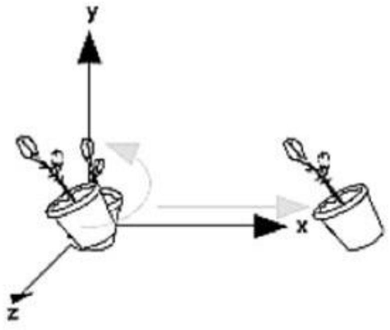
Revisit: Order Matters!

- If T and R are matrices representing affine transformations,

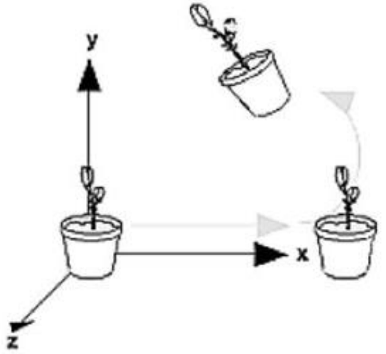
- $\mathbf{p}' = TR\mathbf{p}$
  - First apply transformation R to point **p**, then apply transformation T to transformed point R**p**

- $\mathbf{p}' = RT\mathbf{p}$
  - First apply transformation T to point **p**, then apply transformation R to transformed point T**p**

- ***w.r.t. global frame!***



Rotate then Translate



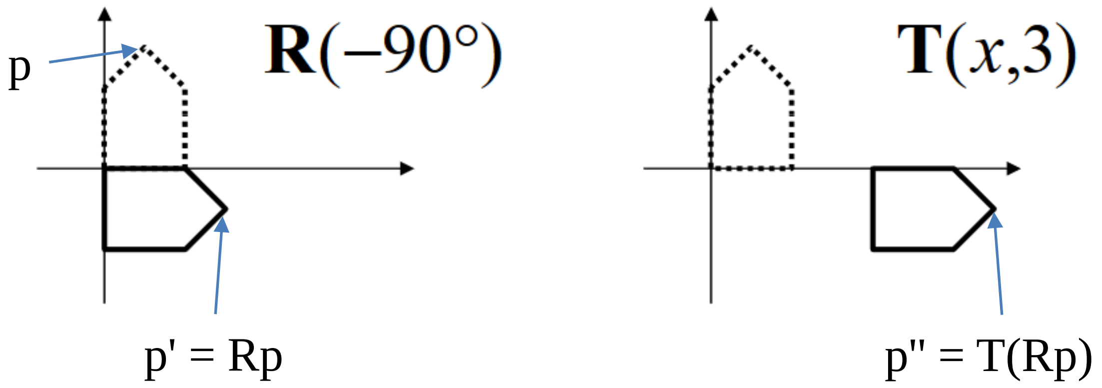
Translate then Rotate

Interpretation of Composite Transformations #1

- An example transformation:

$$M = \mathbf{T}(x,3) \cdot \mathbf{R}(-90°)$$

- This is how we've interpreted so far:
  - R-to-L: Transforms ***w.r.t. global frame***

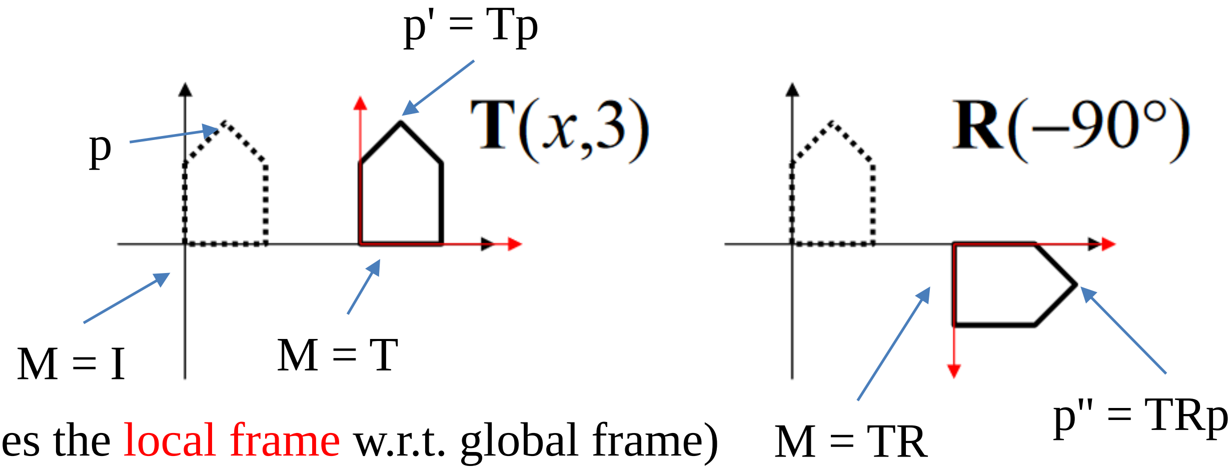


Interpretation of Composite Transformations #2

- An example transformation:

$$M = \mathbf{T}(x,3) \cdot \mathbf{R}(-90°)$$

- **Another way of interpretation:**
  - L-to-R: Transforms ***w.r.t. local frame***



$p' = Tp$

$\mathbf{T}(x,3)$

p

$\mathbf{R}(-90°)$

$M = I$

$M = T$

$M = TR$

$p'' = TRp$

(M defines the local frame w.r.t. global frame)

Left & Right Multiplication

- Thinking it deeper, we can see:

- **p' = RTp (left-multiplication by R)**
  - (R-to-L) Apply T to a point p w.r.t. global frame.
  - Then, apply R to a point Tp w.r.t. global frame.

- **p' = TRp (right-multiplication by R)**
  - (L-to-R) Apply T to a point p w.r.t. global frame.
  - Then, apply R to a point Tp w.r.t local frame.

  - ***Better view:*** Apply **R** **w.r.t the current frame T** and **set p** in the **final frame TR**.

Insert Transformation into a Series of Transformations

- $p' = M_1M_2M_3M_4M_5M_6p$

- $p' = M_1M_2M_3XM_4M_5M_6p$
  - Apply **X w.r.t the current frame $M_1M_2M_3$** and **set p** in the **final frame $M_1M_2M_3XM_4M_5M_6$**.

- $p' = RTp$
  - Apply **R w.r.t the current frame I** and **set p** in the **final frame RT**.

[Practice] Interpretation of Composite Transformations

- Just start from the Lecture 4 practice code "[Practice] OpenGL Trans. Functions".

- Differences are:

```
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()
```

[Practice] Interpretation of Composite Transformations

```python
def render(camAng):
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
    glEnable(GL_DEPTH_TEST)
    glLoadIdentity()
    glOrtho(-1,1, -1,1, -1,1)
    gluLookAt(.1*np.sin(camAng),.1,.1*np.cos(camAng), 0,0,0, 0,1,0)

    # draw global frame
    drawFrame()

    # 1) p'=TRp
    glTranslatef(.4, .0, 0)
    drawFrame()     # frame defined by T
    glRotatef(60, 0, 0, 1)
    drawFrame()     # frame defined by TR

    # # 2) p'=RTp
    # glRotatef(60, 0, 0, 1)
    # drawFrame()   # frame defined by R
    # glTranslatef(.4, .0, 0)
    # drawFrame()   # frame defined by RT

    drawTriangle()
```

Quiz #2

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

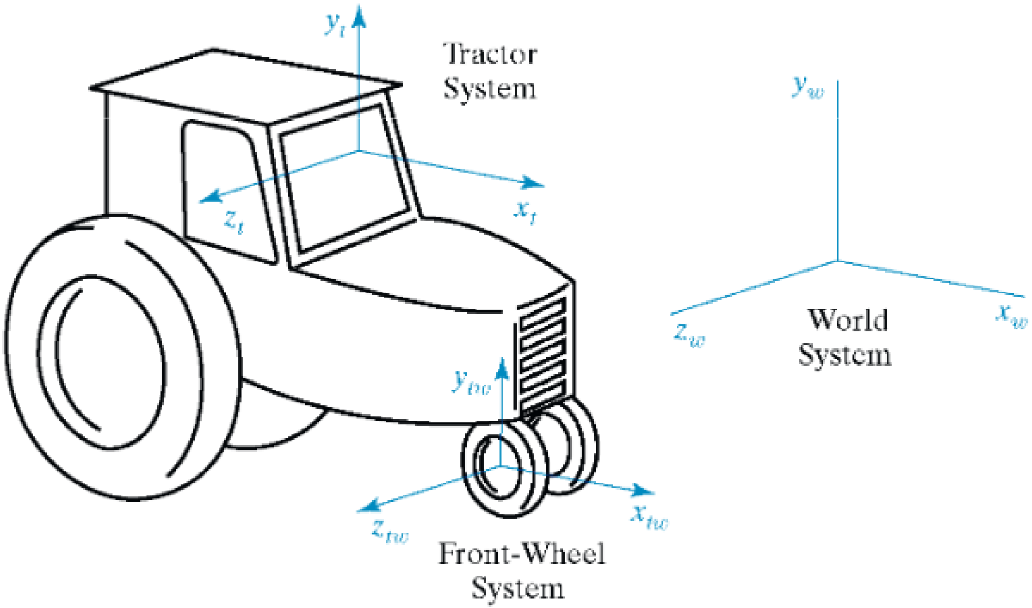
- Submit your answer in the following format:
  - **Student ID: Your answer**
  - **e.g. 2017123456: 4)**

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

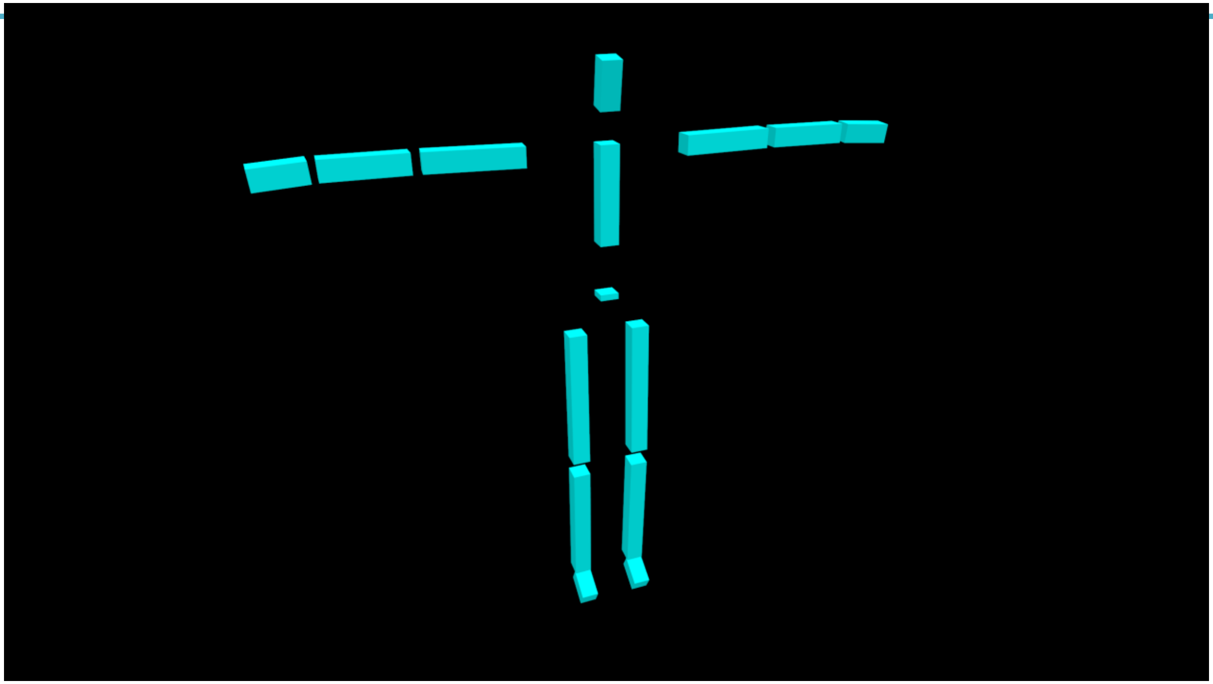
Hierarchical Modeling

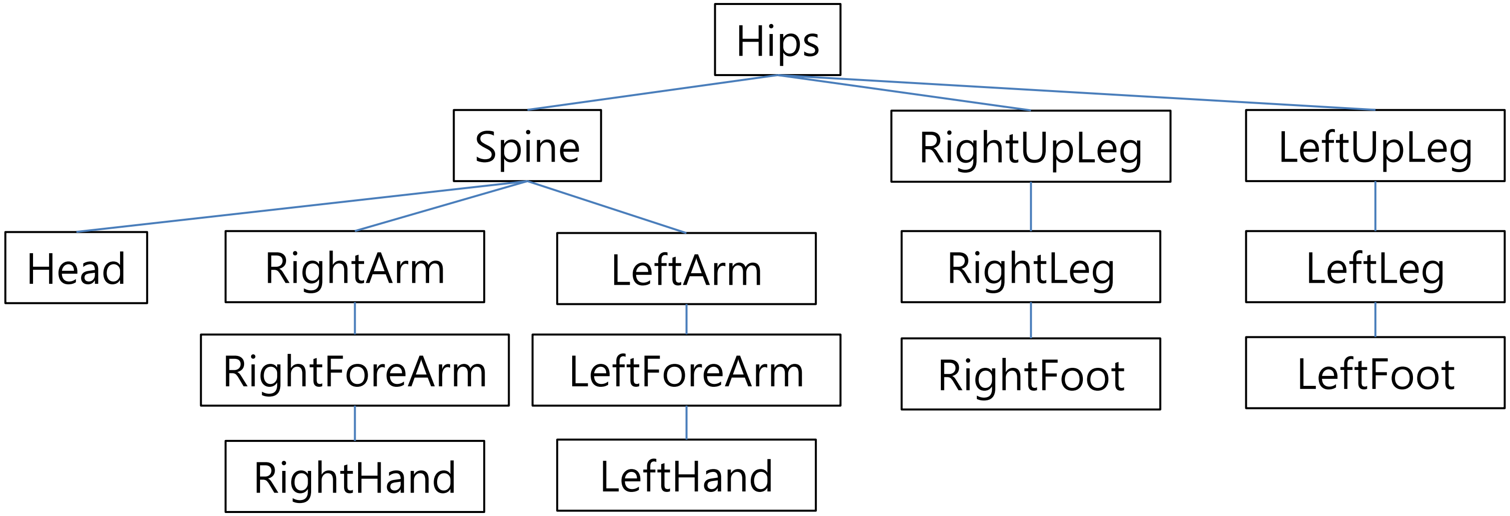
Hierarchical Modeling

- 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





Hips

Spine

RightUpLeg

LeftUpLeg

Head

RightArm

LeftArm

RightLeg

LeftLeg

RightForeArm

LeftForeArm

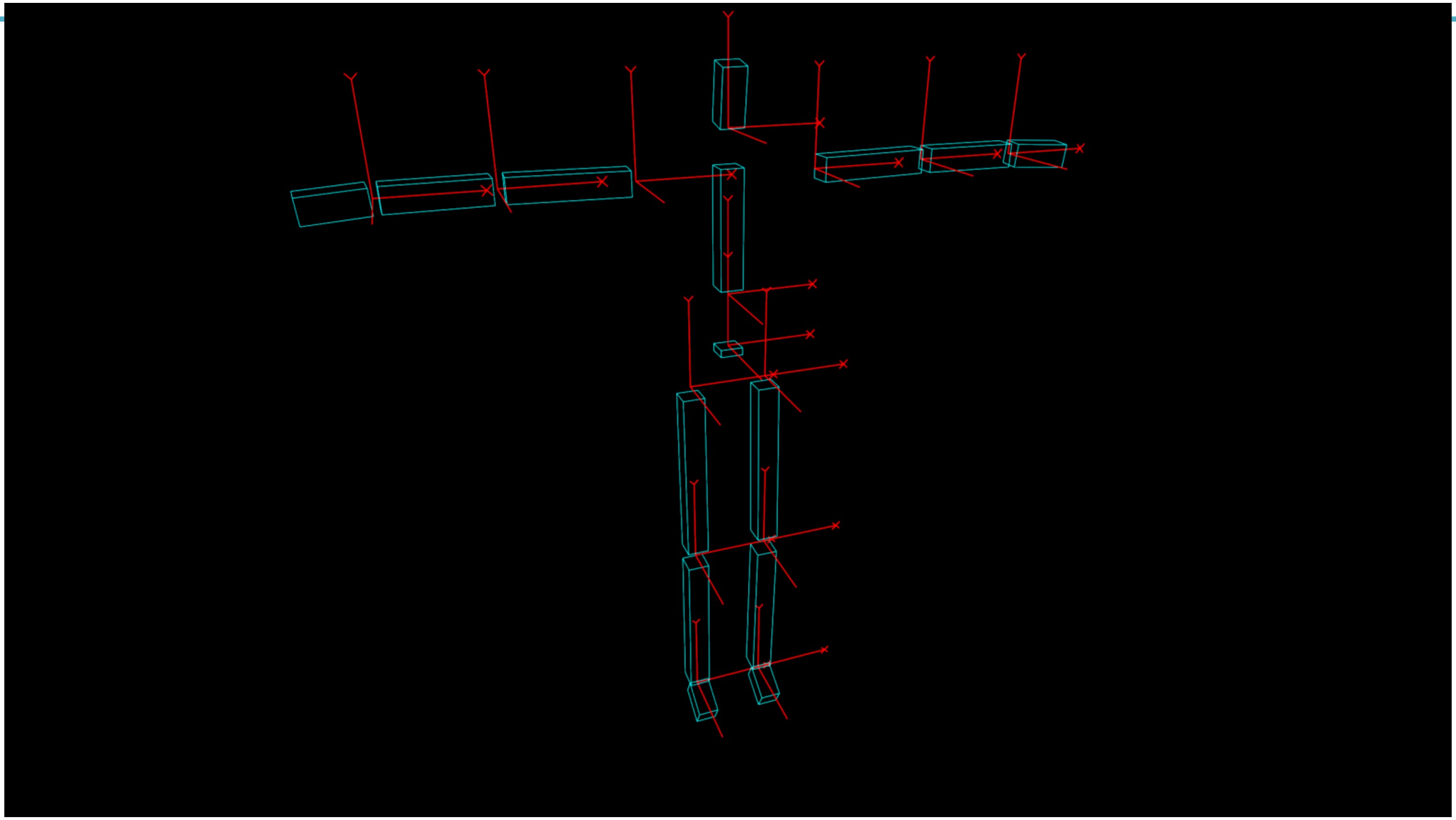
RightFoot

LeftFoot

RightHand

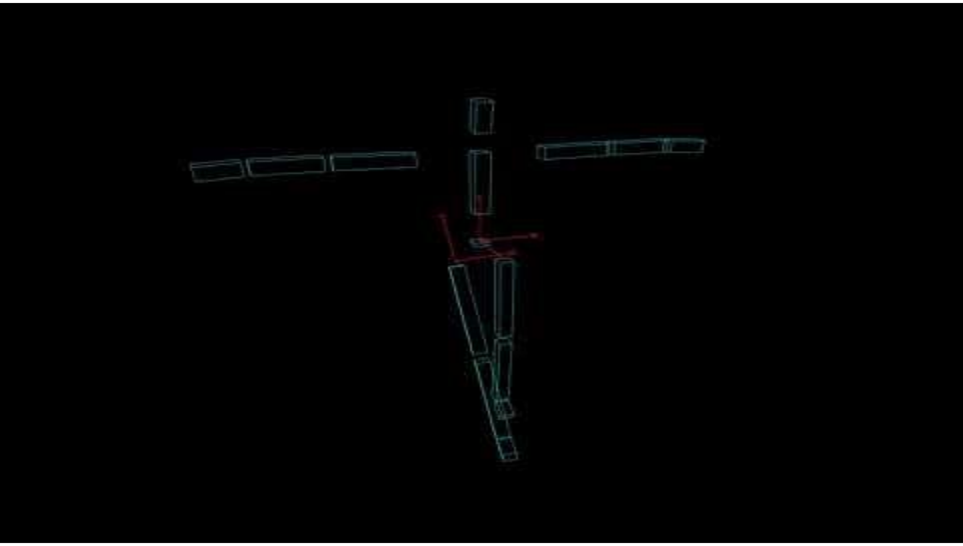
LeftHand

Human Figure - Frames

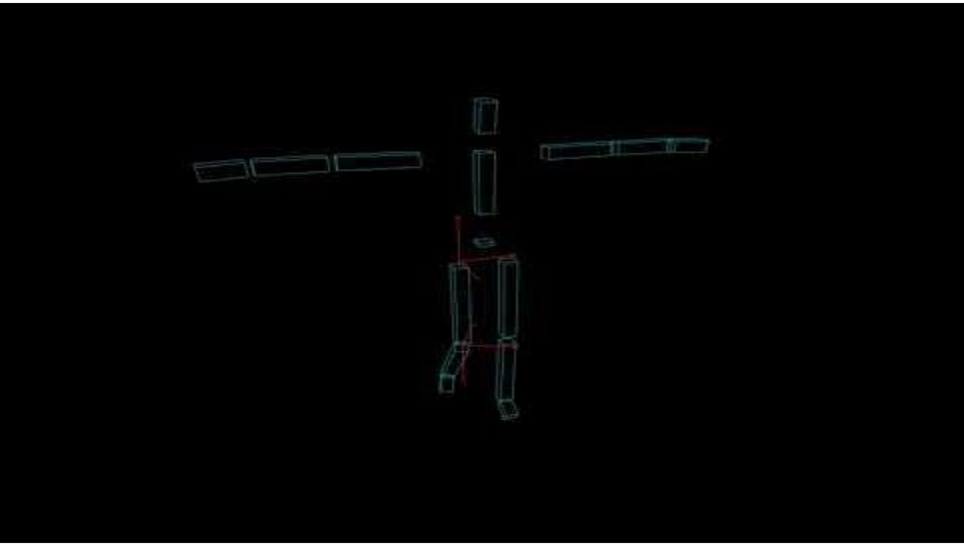


- Each part has its own reference frame (local frame).

Human Figure - Movement of rhip & rknee



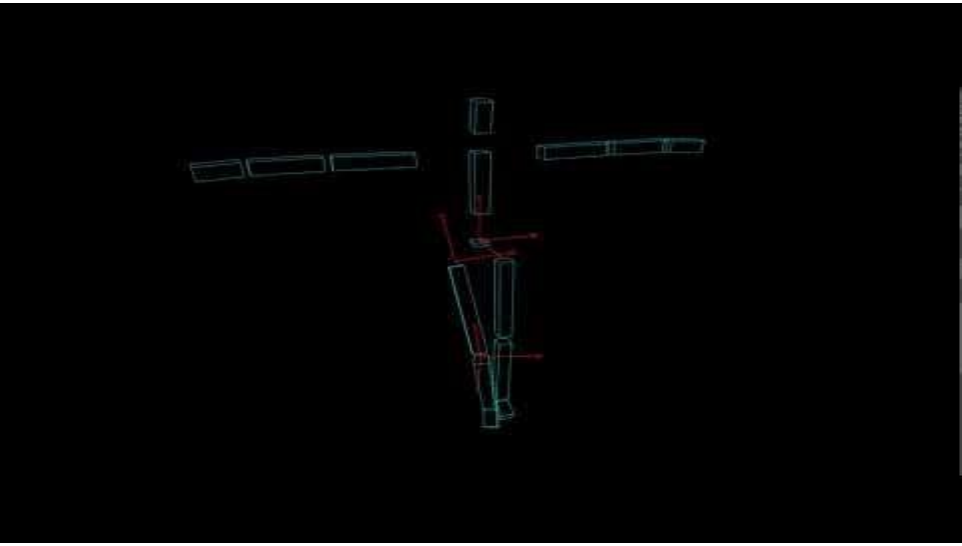
https://youtu.be/Q7lhvMkCSCg



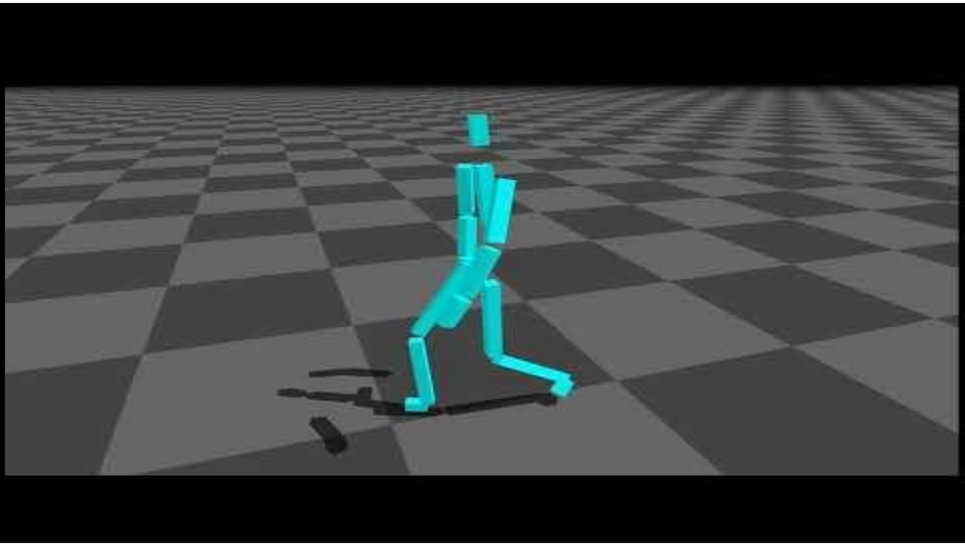
https://youtu.be/Q5R8WGUwpFU

- 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.
- This allows a part to "group" its children together.

Human Figure - Movement of more joints



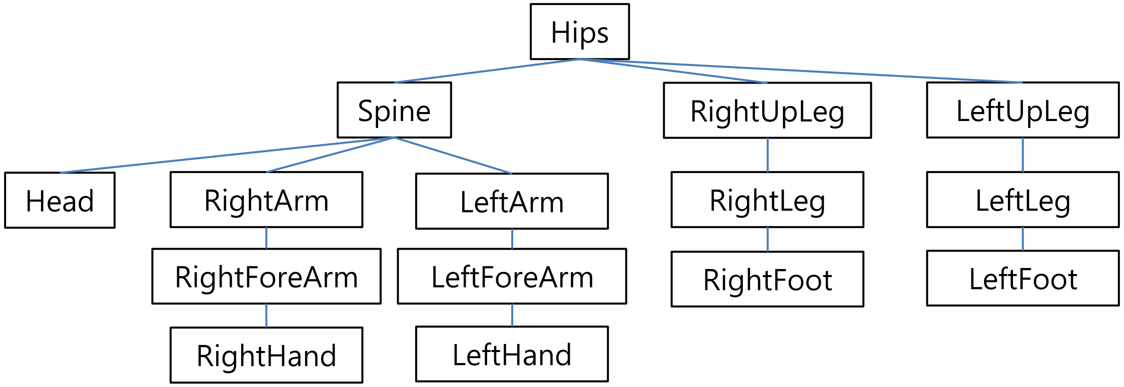
https://youtu.be/9dz8bvVK9zc

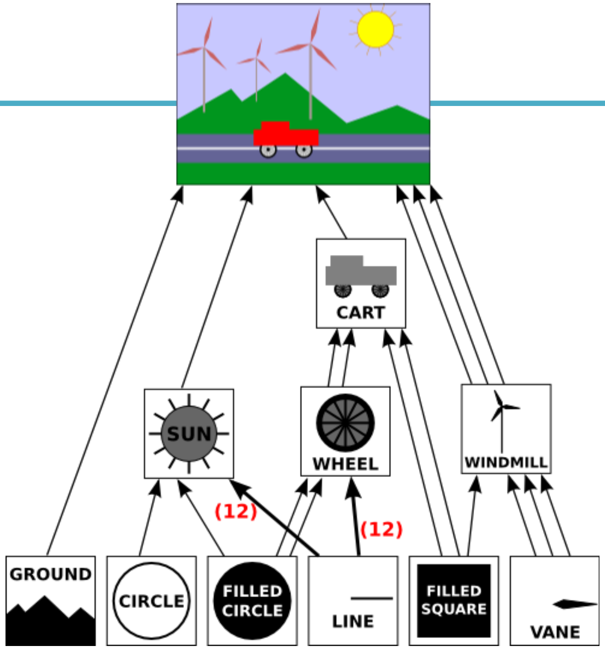


https://youtu.be/PEhyWI8LGBY

- 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.
- This allows a part to "group" its children together.

Hierarchical Model





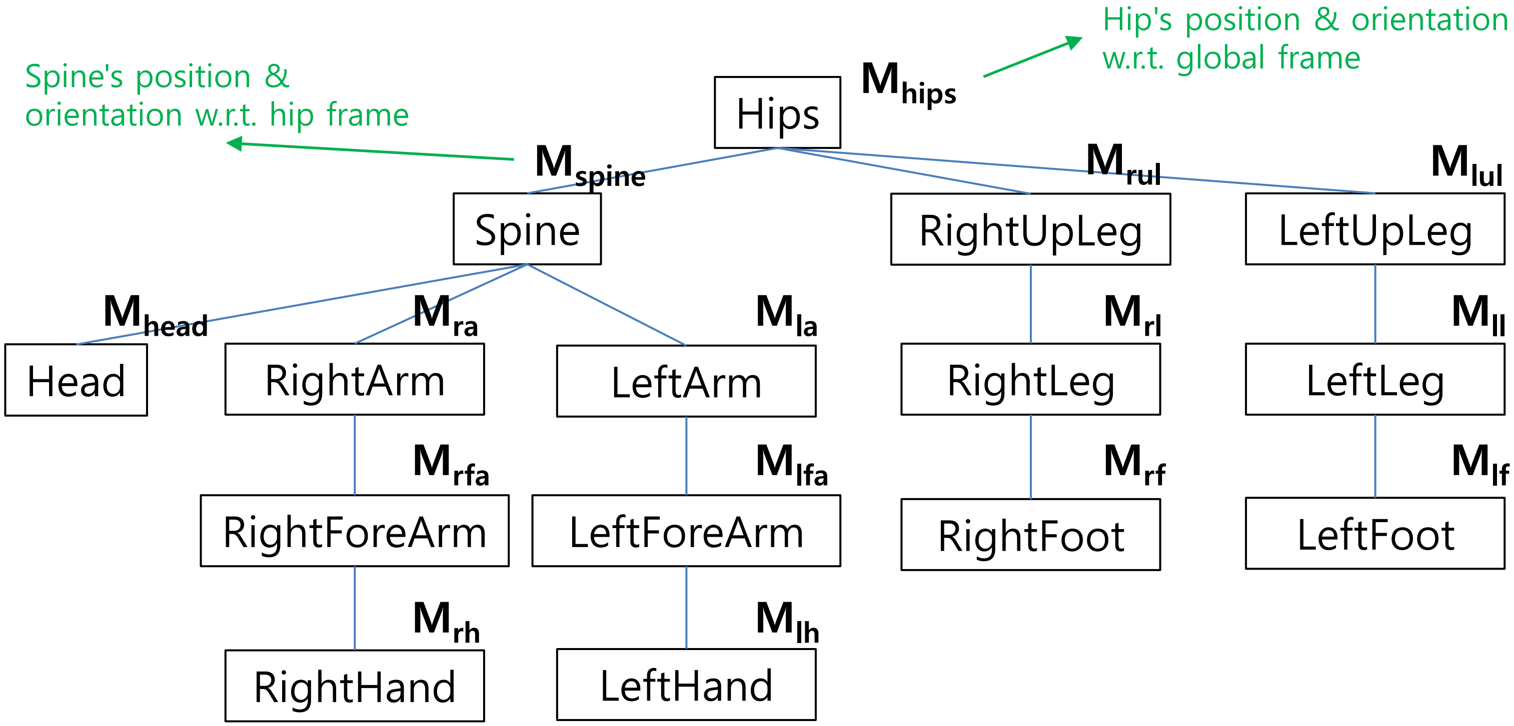
- A hierarchical model is represented by a graph structure.
  - A tree structure is most commonly used.

- *Scene graph*: A graph structure that represents an entire scene.

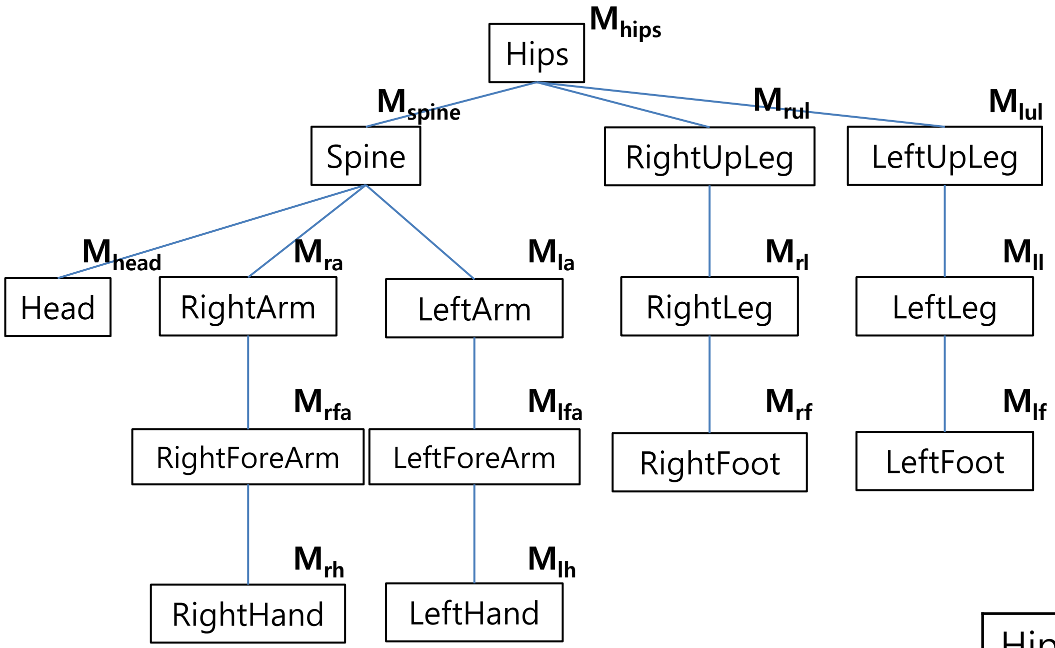
- Each node has its own transformation w.r.t. parent node’s frame.

Rendering Hierarchical Models

- Each node has its own transformation w.r.t. parent node’s frame.
- Using these transformations, a hierarchical model can be rendered by *depth-first traversal*.



Rendering Hierarchical Models



M_{hips} Hips

M_{spine} Spine — M_{head} Head, M_{ra} RightArm, M_{la} LeftArm

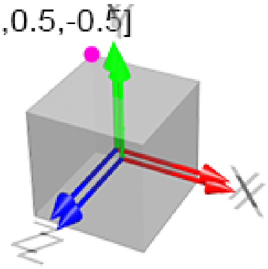
M_{rul} RightUpLeg — M_{rl} RightLeg — M_{rf} RightFoot

M_{lul} LeftUpLeg — M_{ll} LeftLeg — M_{lf} LeftFoot

RightArm — M_{rfa} RightForeArm — M_{rh} RightHand

LeftArm — M_{lfa} LeftForeArm — M_{lh} LeftHand

$\mathbf{p}$= [-0.5,0.5,-0.5]

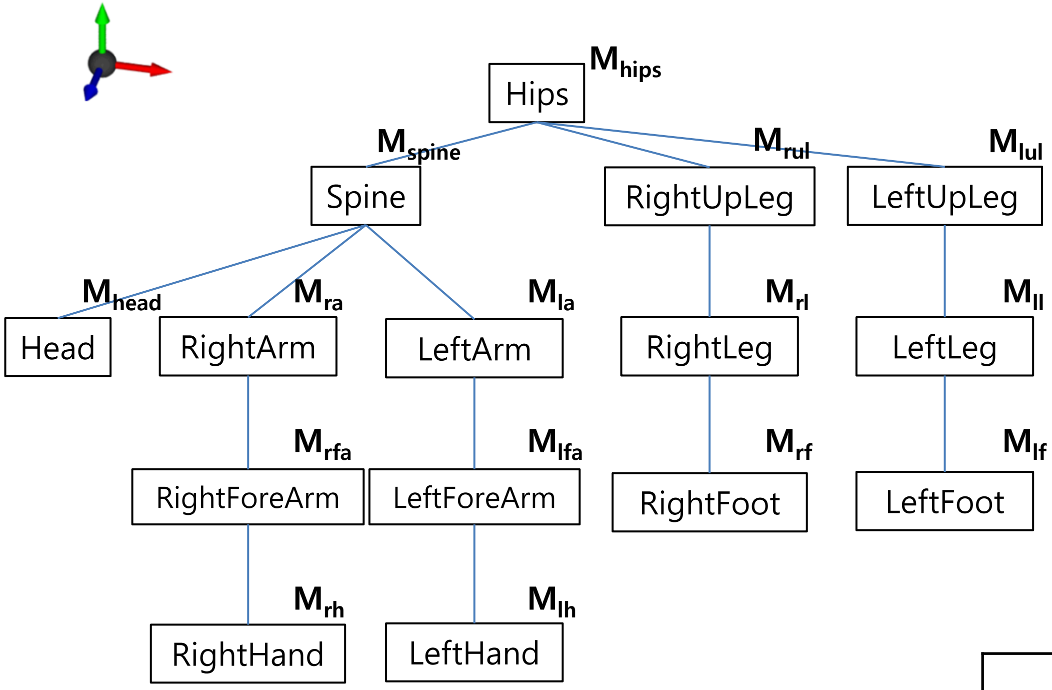


Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is…

Hips	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{p}$
Spine	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{p}$
Head	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{head}\,\mathbf{p}$
RightArm	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{ra}\,\mathbf{p}$
RightForeArm	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{ra}\,\mathbf{M}_{rfa}\,\mathbf{p}$
RightHand	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{ra}\,\mathbf{M}_{rfa}\,\mathbf{M}_{rh}\,\mathbf{p}$
LeftArm	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{la}\,\mathbf{p}$
…	

This can be effectively computed using a *stack*. ←

Rendering Hierarchical Models



Hips M_{hips}

Spine M_{spine}

RightUpLeg M_{rul}

LeftUpLeg M_{lul}

Head M_{head}

RightArm M_{ra}

LeftArm M_{la}

RightLeg M_{rl}

LeftLeg M_{ll}

RightForeArm M_{rfa}

LeftForeArm M_{lfa}

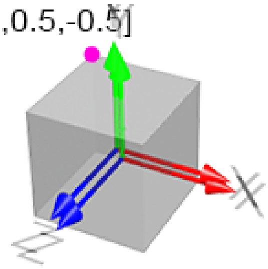
RightFoot M_{rf}

LeftFoot M_{lf}

RightHand M_{rh}

LeftHand M_{lh}

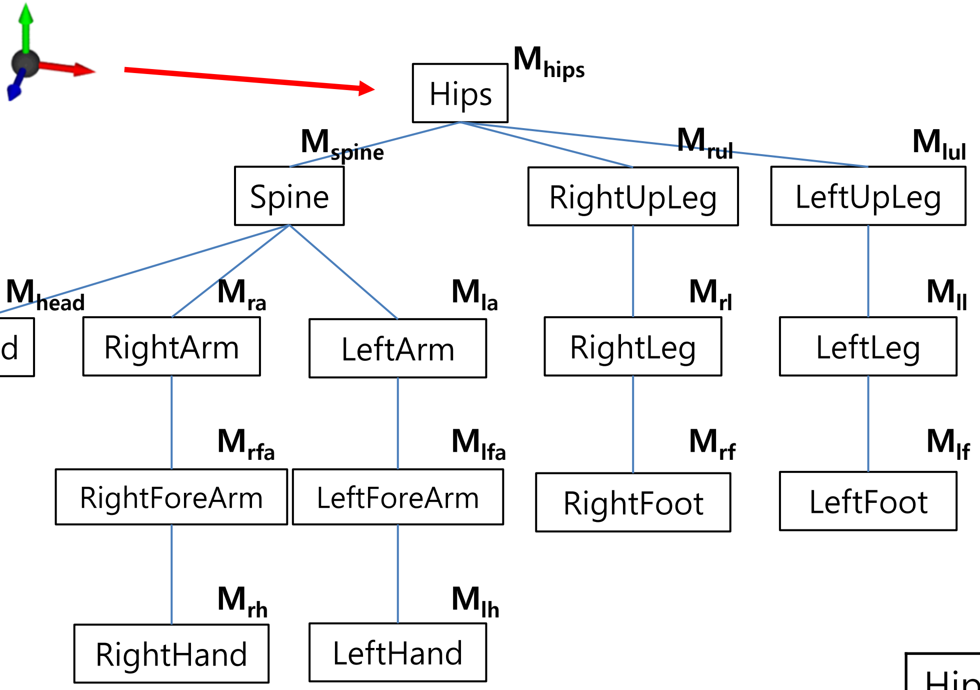
$\mathbf{p}$= [-0.5,0.5,-0.5]



Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is…

|

Rendering Hierarchical Models



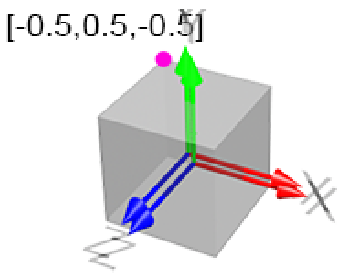
Hips $\mathbf{M}_{hips}$

Spine $\mathbf{M}_{spine}$
RightUpLeg $\mathbf{M}_{rul}$
LeftUpLeg $\mathbf{M}_{lul}$

Head $\mathbf{M}_{head}$
RightArm $\mathbf{M}_{ra}$
LeftArm $\mathbf{M}_{la}$
RightLeg $\mathbf{M}_{rl}$
LeftLeg $\mathbf{M}_{ll}$

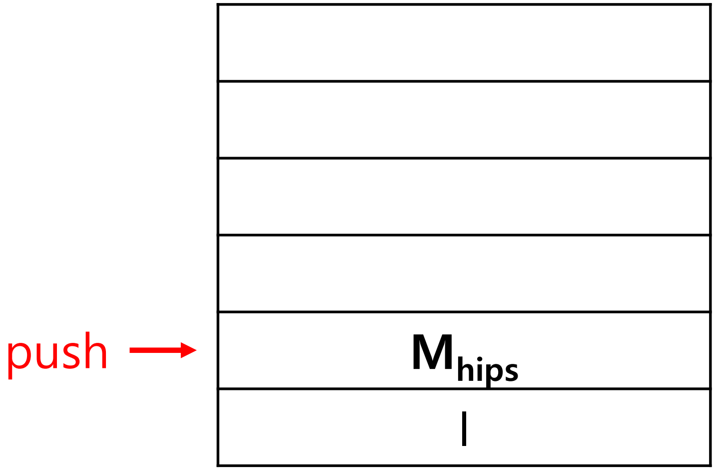
RightForeArm $\mathbf{M}_{rfa}$
LeftForeArm $\mathbf{M}_{lfa}$
RightFoot $\mathbf{M}_{rf}$
LeftFoot $\mathbf{M}_{lf}$

RightHand $\mathbf{M}_{rh}$
LeftHand $\mathbf{M}_{lh}$



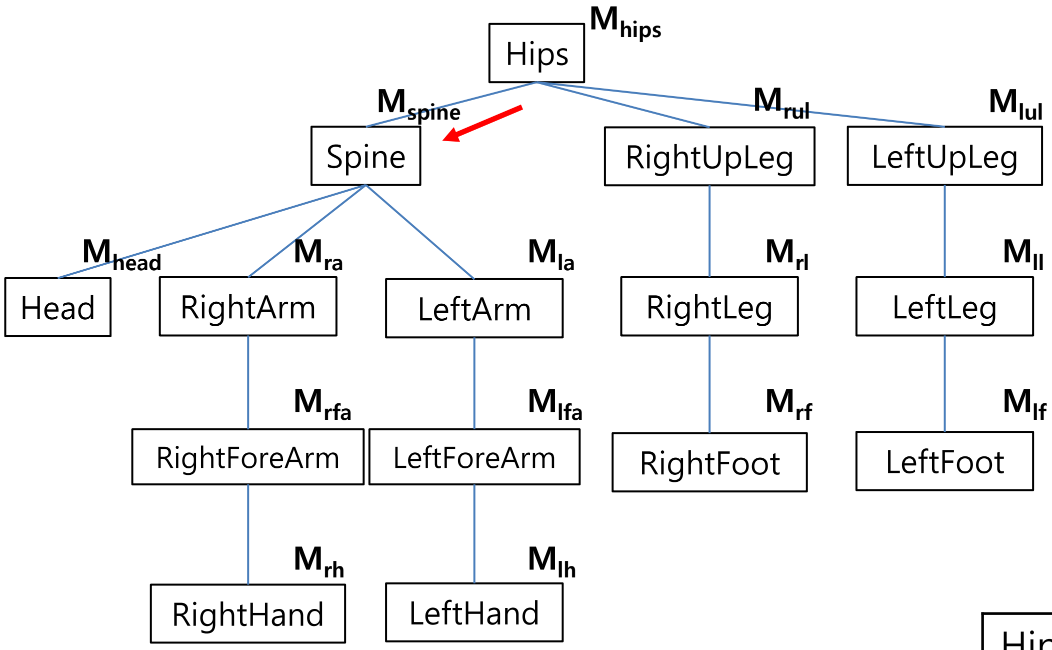
p= [-0.5,0.5,-0.5]

Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is...



Hips	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{p}$

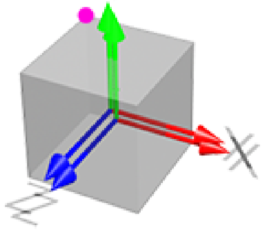
Rendering Hierarchical Models



push →

$M_{hips}\ M_{spine}$
M_{hips}
I

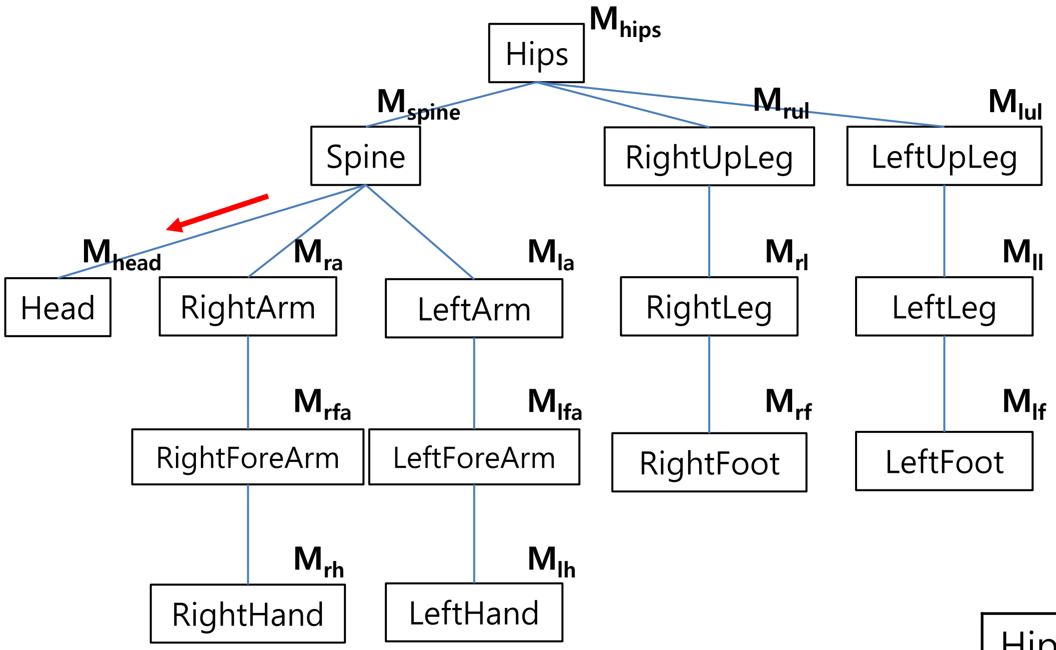
p= [-0.5,0.5,-0.5]



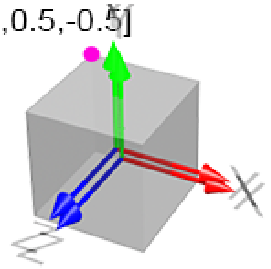
Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is...

Hips	$p' = M_{hips}\ p$
Spine	$p' = M_{hips}\ M_{spine}\ p$

Rendering Hierarchical Models

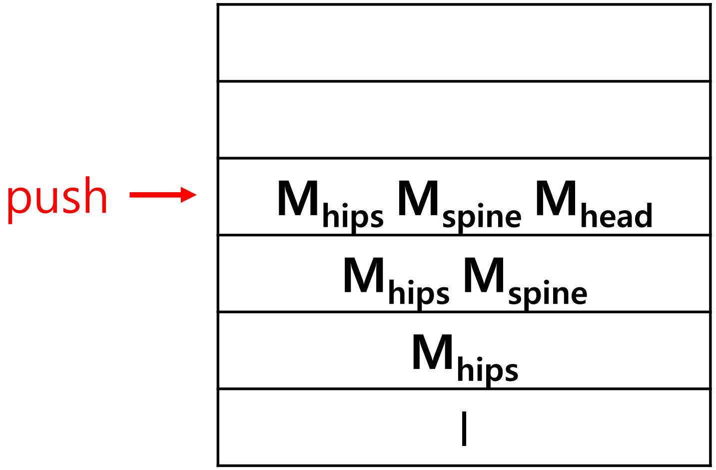


p = [-0.5,0.5,-0.5]



Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is...

Hips	$\mathbf{p'} = \mathbf{M}_{hips}\,\mathbf{p}$
Spine	$\mathbf{p'} = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{p}$
Head	$\mathbf{p'} = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{head}\,\mathbf{p}$



push

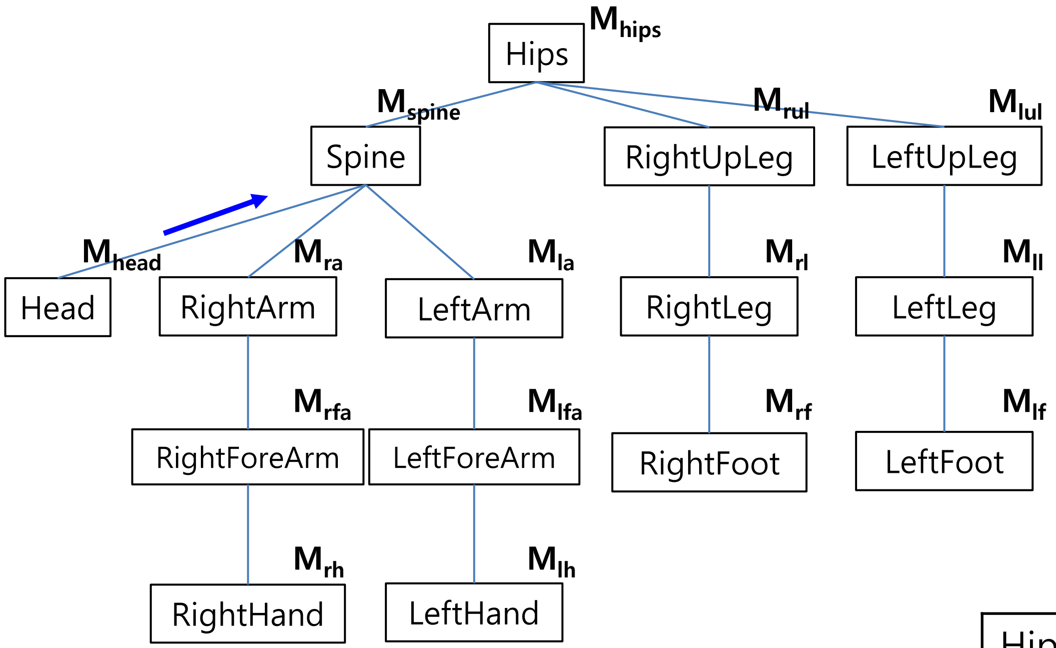
$\mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{head}$

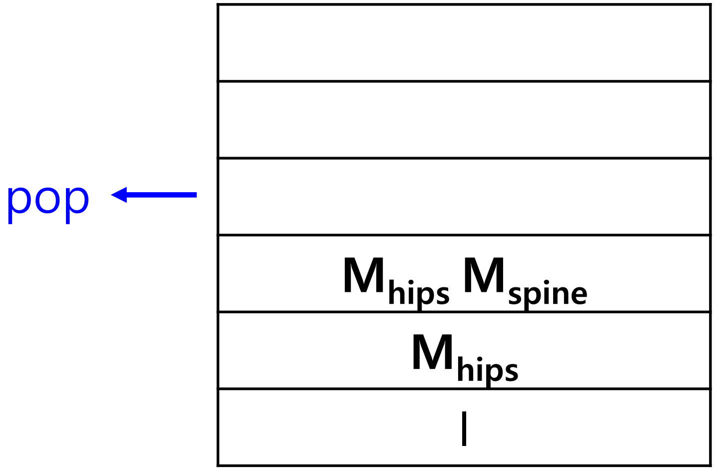
$\mathbf{M}_{hips}\,\mathbf{M}_{spine}$

$\mathbf{M}_{hips}$

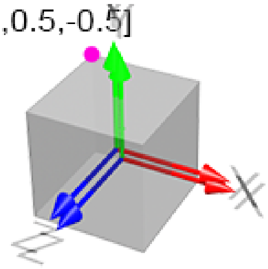
I

Rendering Hierarchical Models





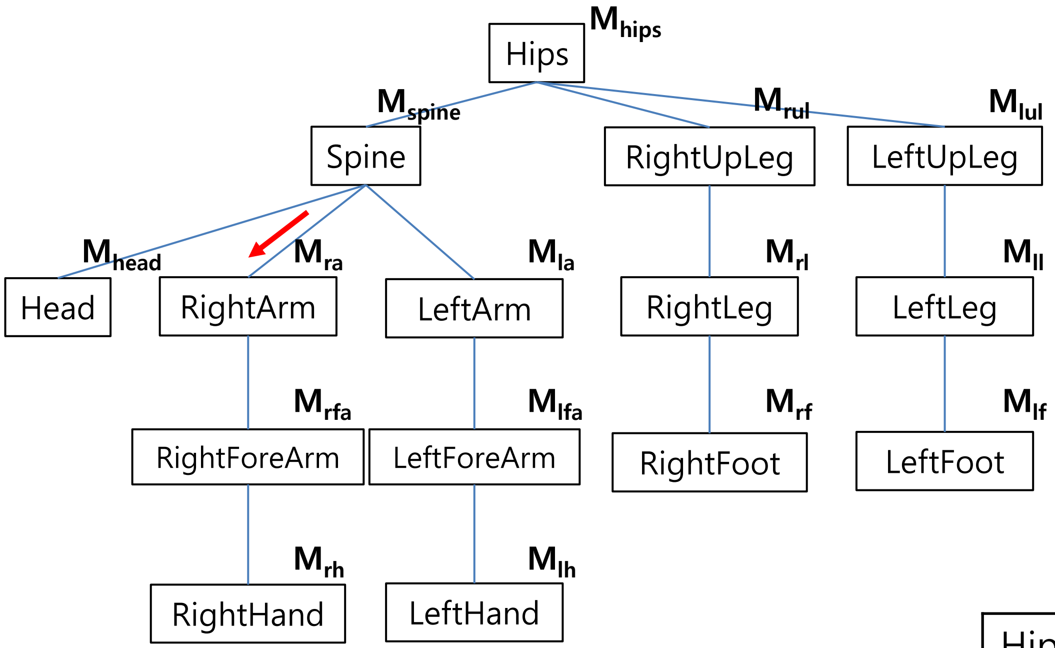
$\mathbf{p}$= [-0.5,0.5,-0.5]



Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is…

Hips	$\mathbf{p'} = \mathbf{M}_{hips}\,\mathbf{p}$
Spine	$\mathbf{p'} = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{p}$
Head	$\mathbf{p'} = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{head}\,\mathbf{p}$

Rendering Hierarchical Models



Hips $\mathbf{M}_{hips}$

Spine $\mathbf{M}_{spine}$

RightUpLeg $\mathbf{M}_{rul}$

LeftUpLeg $\mathbf{M}_{lul}$

Head $\mathbf{M}_{head}$

RightArm $\mathbf{M}_{ra}$

LeftArm $\mathbf{M}_{la}$

RightLeg $\mathbf{M}_{rl}$

LeftLeg $\mathbf{M}_{ll}$

RightForeArm $\mathbf{M}_{rfa}$

LeftForeArm $\mathbf{M}_{lfa}$

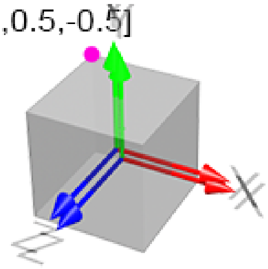
RightFoot $\mathbf{M}_{rf}$

LeftFoot $\mathbf{M}_{lf}$

RightHand $\mathbf{M}_{rh}$

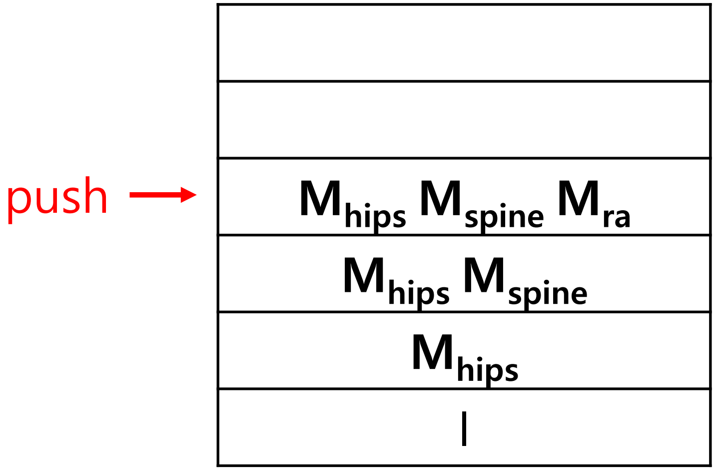
LeftHand $\mathbf{M}_{lh}$

$\mathbf{p}$= [-0.5,0.5,-0.5]



Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is...

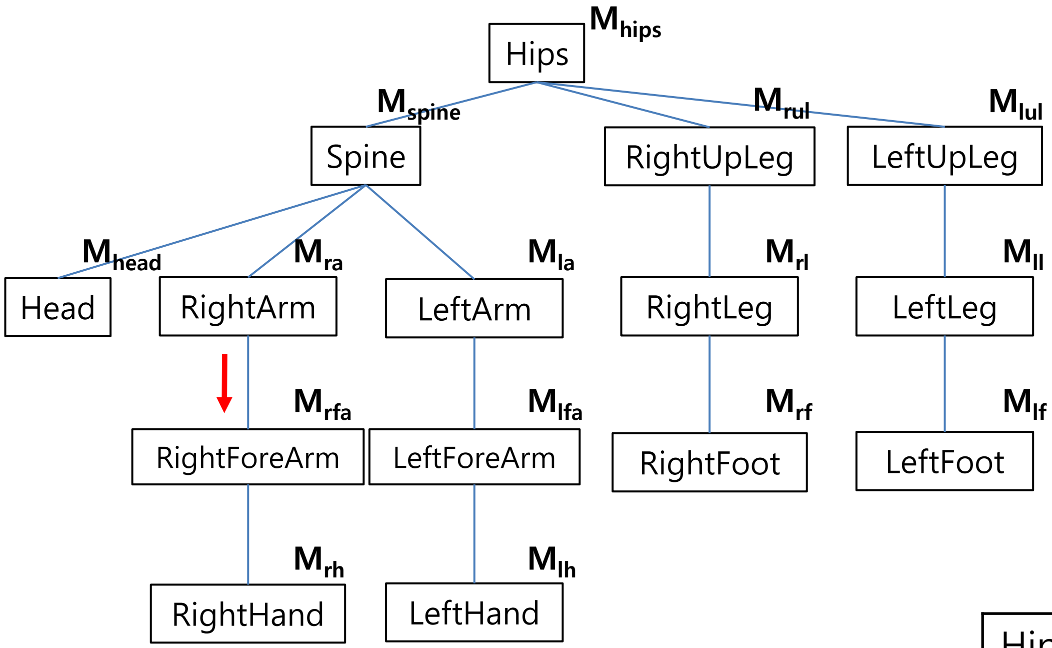
Hips	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{p}$
Spine	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{p}$
Head	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{head}\,\mathbf{p}$
RightArm	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{ra}\,\mathbf{p}$



push →

$\mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{ra}$
$\mathbf{M}_{hips}\,\mathbf{M}_{spine}$
$\mathbf{M}_{hips}$
$\mathbf{I}$

Rendering Hierarchical Models



Hips $\mathbf{M}_{hips}$

Spine $\mathbf{M}_{spine}$

RightUpLeg $\mathbf{M}_{rul}$

LeftUpLeg $\mathbf{M}_{lul}$

Head $\mathbf{M}_{head}$

RightArm $\mathbf{M}_{ra}$

LeftArm $\mathbf{M}_{la}$

RightLeg $\mathbf{M}_{rl}$

LeftLeg $\mathbf{M}_{ll}$

RightForeArm $\mathbf{M}_{rfa}$

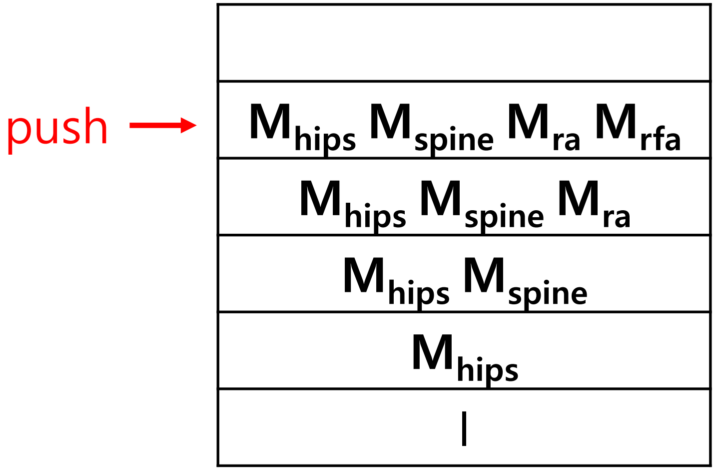
LeftForeArm $\mathbf{M}_{lfa}$

RightFoot $\mathbf{M}_{rf}$

LeftFoot $\mathbf{M}_{lf}$

RightHand $\mathbf{M}_{rh}$

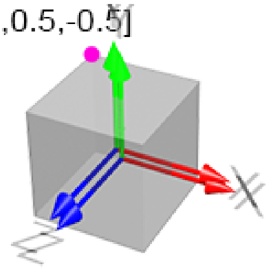
LeftHand $\mathbf{M}_{lh}$



push →

$\mathbf{M}_{hips}\ \mathbf{M}_{spine}\ \mathbf{M}_{ra}\ \mathbf{M}_{rfa}$
$\mathbf{M}_{hips}\ \mathbf{M}_{spine}\ \mathbf{M}_{ra}$
$\mathbf{M}_{hips}\ \mathbf{M}_{spine}$
$\mathbf{M}_{hips}$
I

$\mathbf{p}$= [-0.5,0.5,-0.5]



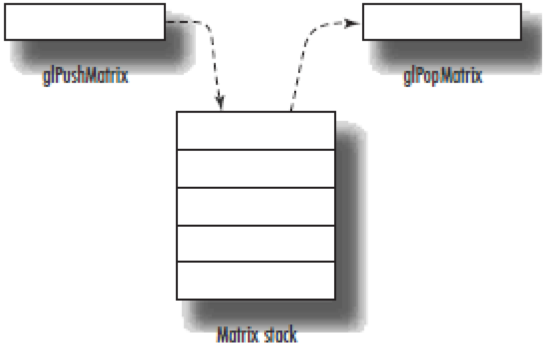
Let's say each part is rendered as a unit box above (without scaling), its vertex position **p'** w.r.t. global frame is...

Hips	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{p}$
Spine	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{p}$
Head	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{head}\,\mathbf{p}$
RightArm	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{ra}\,\mathbf{p}$
RightForeArm	$\mathbf{p}' = \mathbf{M}_{hips}\,\mathbf{M}_{spine}\,\mathbf{M}_{ra}\,\mathbf{M}_{rfa}\,\mathbf{p}$
...	

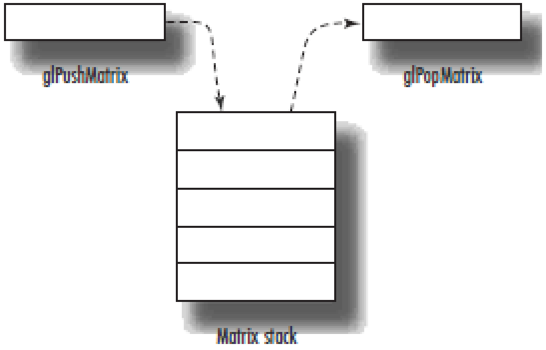
OpenGL Matrix Stack

- Legacy OpenGL provides a stack for this purpose.

- You can **save** the **current transformation matrix** and then **restore** it after some objects have been drawn.



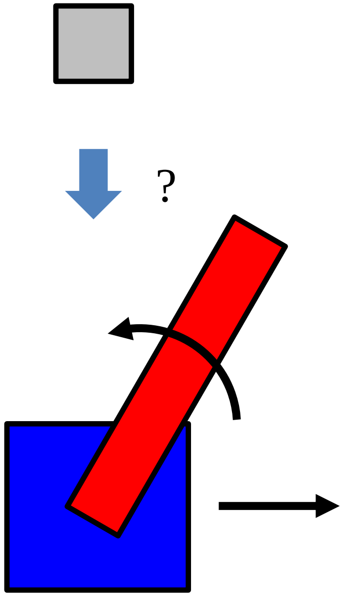
OpenGL Matrix Stack



- 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

drawBox(): draw a unit box



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) # scaling for drawing

TS
T
I

drawBox() **p'=TSp**

glPopMatrix()

T
I

glPushMatrix()

T
T
I

glRotate(R) # to rotate arm

TR
T
I

glPushMatrix()

TR
TR
T
I

glScale(U) # scaling for drawing

TRU
TR
T
I

drawBox() **p'=TRUp**

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 #3

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

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

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

OpenGL Matrix Stack Types

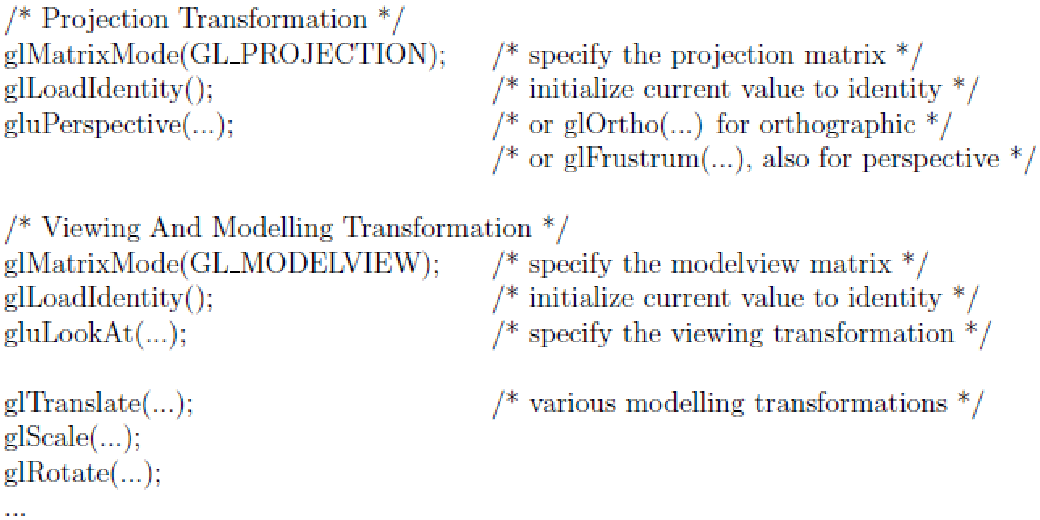
- Actually, OpenGL maintains four different types of matrix stacks:

- **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 glOrtho(...) for orthographic */
                                /* or glFrustrum(...), 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(), glOrtho(), …) 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()
```

Next Time

- Next week: Midterm exam (Apr 27)
- No lab next Monday.

- **No lab & lecture in the week after next.**
- Lab for this lecture (May 9):
  - Lab assignment 8
- Next lecture (May 11):
  - 9 - Orientation & Rotation

- **Class Assignment #2**
  - **Due: 23:59, May 17, 2022**

- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Andy van Dam, Brown Univ., http://cs.brown.edu/courses/csci1230/lectures.shtml
  - Prof. JungHyun Han, Korea Univ., http://media.korea.ac.kr/book/
  - 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. Kayvon Fatahalian and Keenan Crane, CMU, http://15462.courses.cs.cmu.edu/fall2015/