Computer Graphics

T4 - Hierarchical Modeling

Yoonsang Lee and Taesoo Kwon

Spring 2019

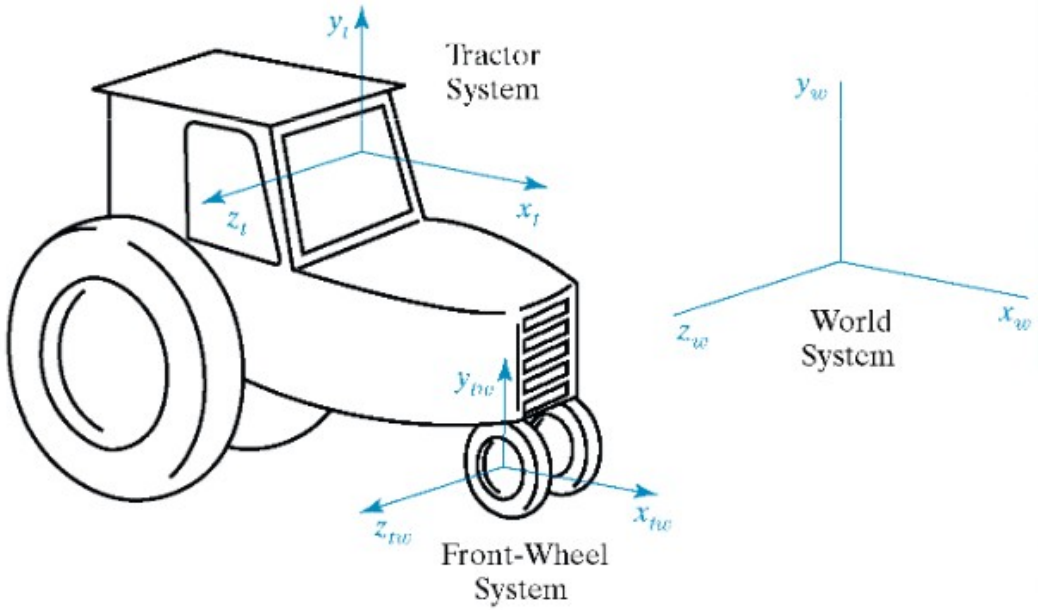
Topics Covered

- Hierarchical Modeling
  - Concept of Hierarchical Modeling
  - OpenGL Matrix Stack

Hierarchical Modeling

Hierarchical Modeling

- A hierarchical model is created by nesting the descriptions of subparts into one another to form a tree organization



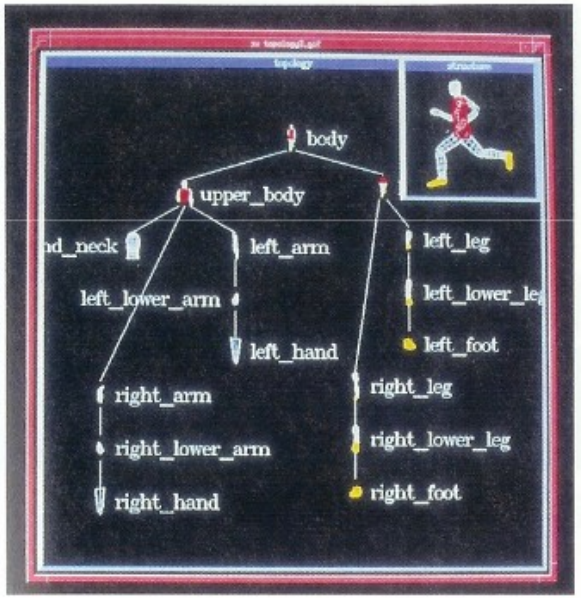
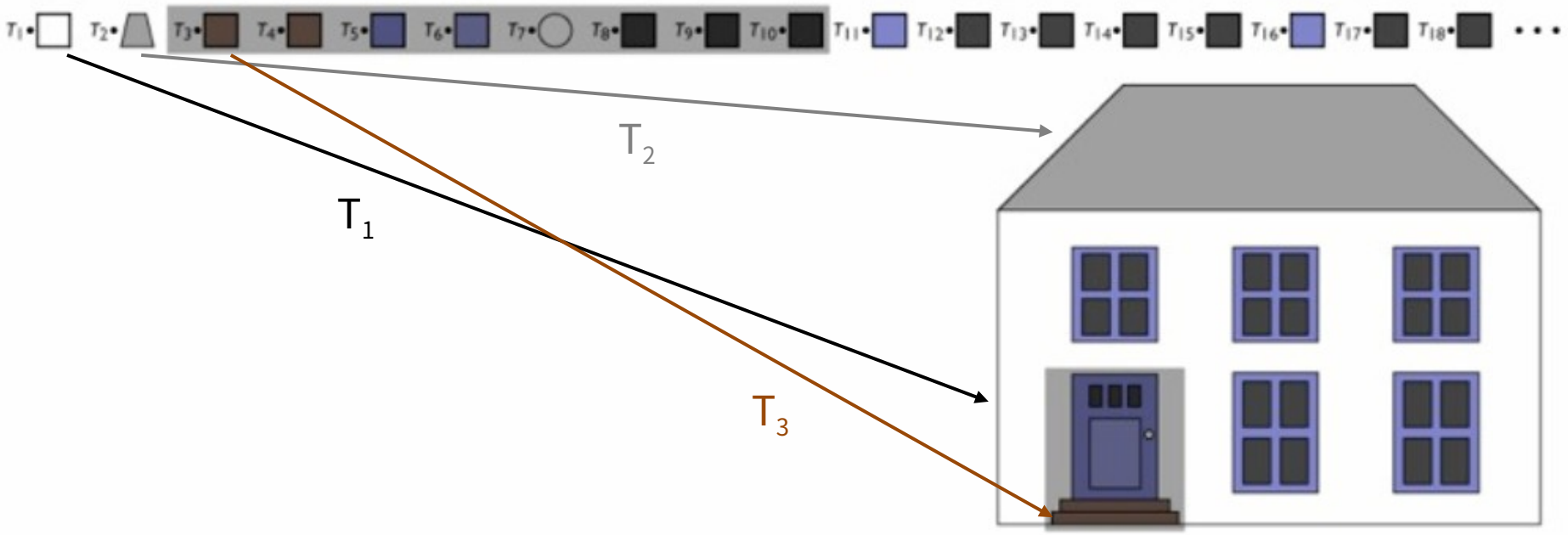


FIGURE 14-4 An object hierarchy generated using the PHIGS Toolkit package developed at the University of Manchester. The displayed object tree is itself a PHIGS structure. (*Courtesy of T. L. J. Howard, J. G. Williams, and W. T. Hewitt, Department of Computer Science, University of Manchester, United Kingdom.*)

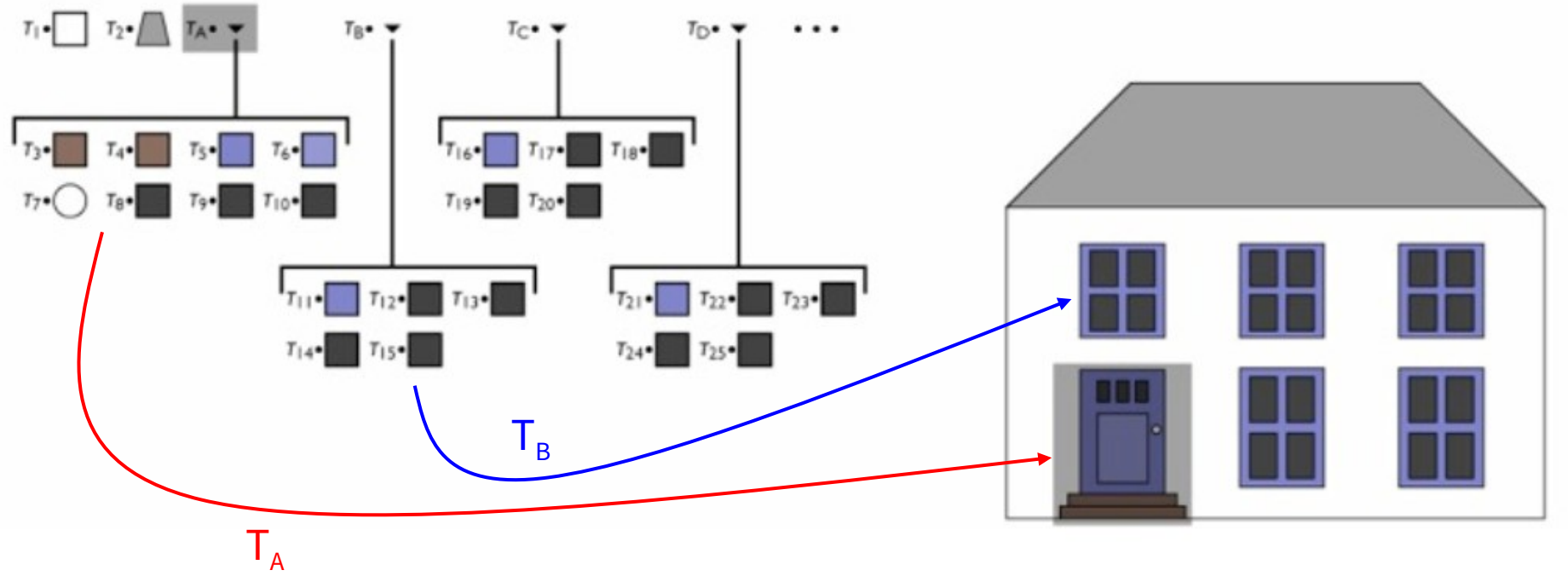
Example

- Can represent drawing with flat list
  - but editing operations require updating many transforms



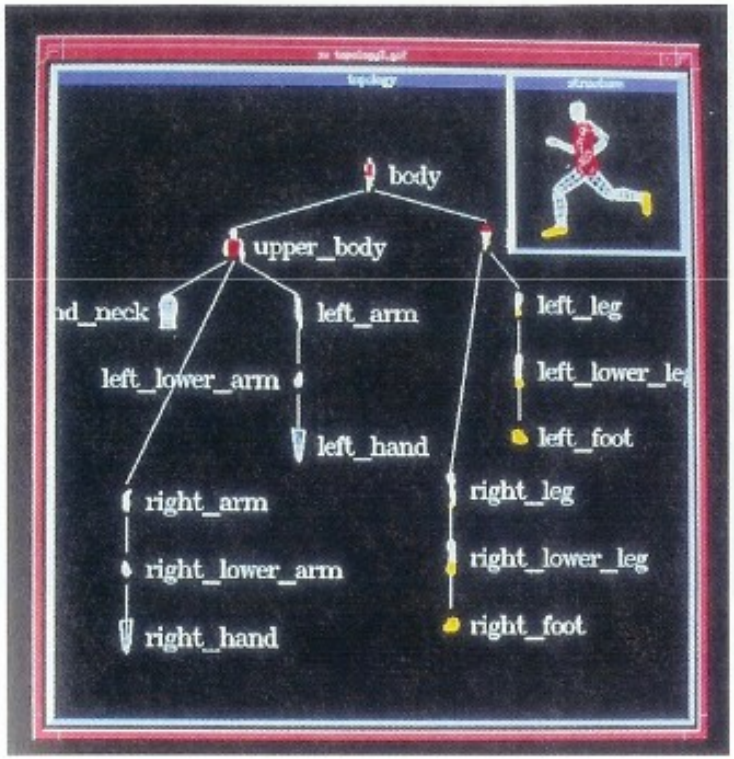
“Grouping”

- Treats a set of objects as one
  - lets the data structure reflect the drawing structure
  - enables high-level editing by changing just one node



The Scene Graph (tree)

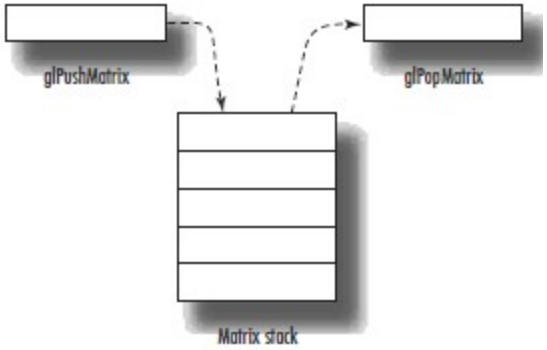
- A name given to various kinds of graph structures (nodes connected together) used to represent scenes
- Simplest form: tree
  - just saw this
  - every node has one parent



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

OpenGL Matrix Stack

- A *stack* for transformation matrices
  - Last In First Outs

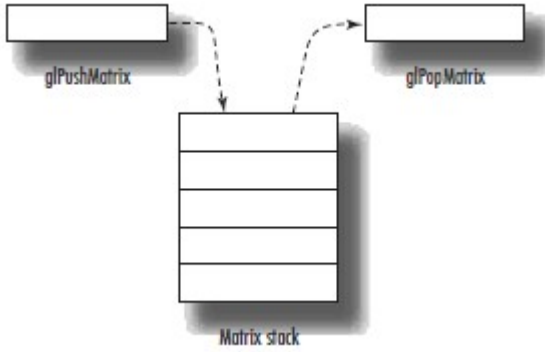


- You can **save** the current transformation matrix and then **restore** it after some objects have been drawn

- Useful for traversing hierarchical data structures (i.e. scene graph or tree)

OpenGL Matrix Stack

- glPushMatrix()
  - Pushes **the current matrix** onto the stack.

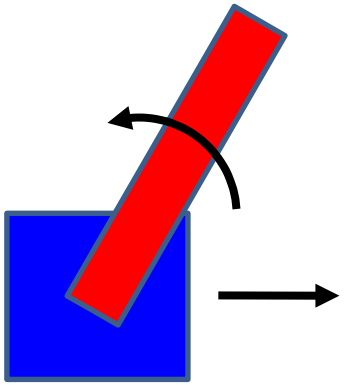


- glPopMatrix()
  - Pops the matrix off the stack.

- The **current matrix** is the matrix **on the top of the stack!**

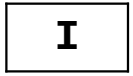
- Keep in mind that the

A simple example

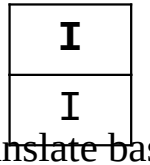


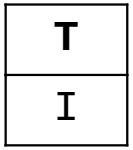
Bold text is the **current transformation matrix** (the one at the top of the matrix stack)

- Start with identity matrix

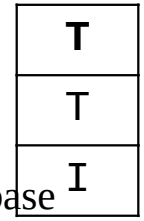


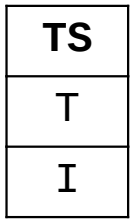
- glPushMatrix()
- glTranslate(T) # to translate base



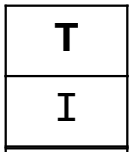


- glPushMatrix()
- glScale(S) # to draw base
- Draw a box

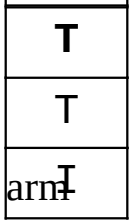


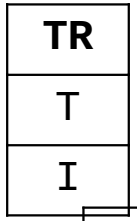


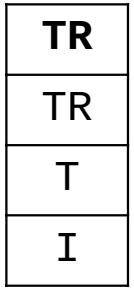
- glPopMatrix()

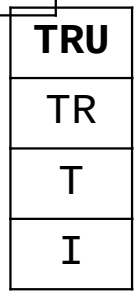


- glPushMatrix()
- glRotate(R) # to rotate arm

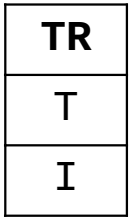




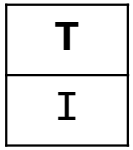




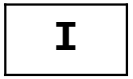
- glPushMatrix()
- glScale(U) # to draw arm
- Draw a box



- glPopMatrix()



- glPopMatrix()



- glPopMatrix()

[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()
```

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 been used so far)
- **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)

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()
```