Advanced Computer Graphics

6 - Affine Matrix, Hierarchical Modeling

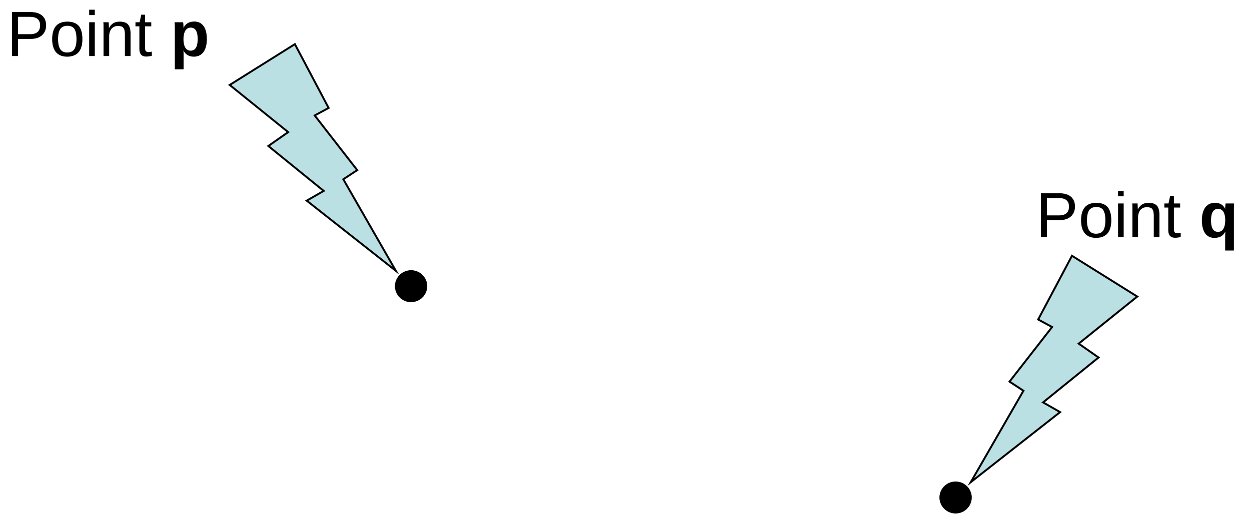
Yoonsang Lee
Fall 2018

Today's Topics

- Affine Geometry: Vectors & Points
- Meanings of an Affine Matrix
- Hierarchical Modeling
  - OpenGL matrix stack

Affine Geometry: Vectors & Points

Points



- What is the “sum” of these two positions ?

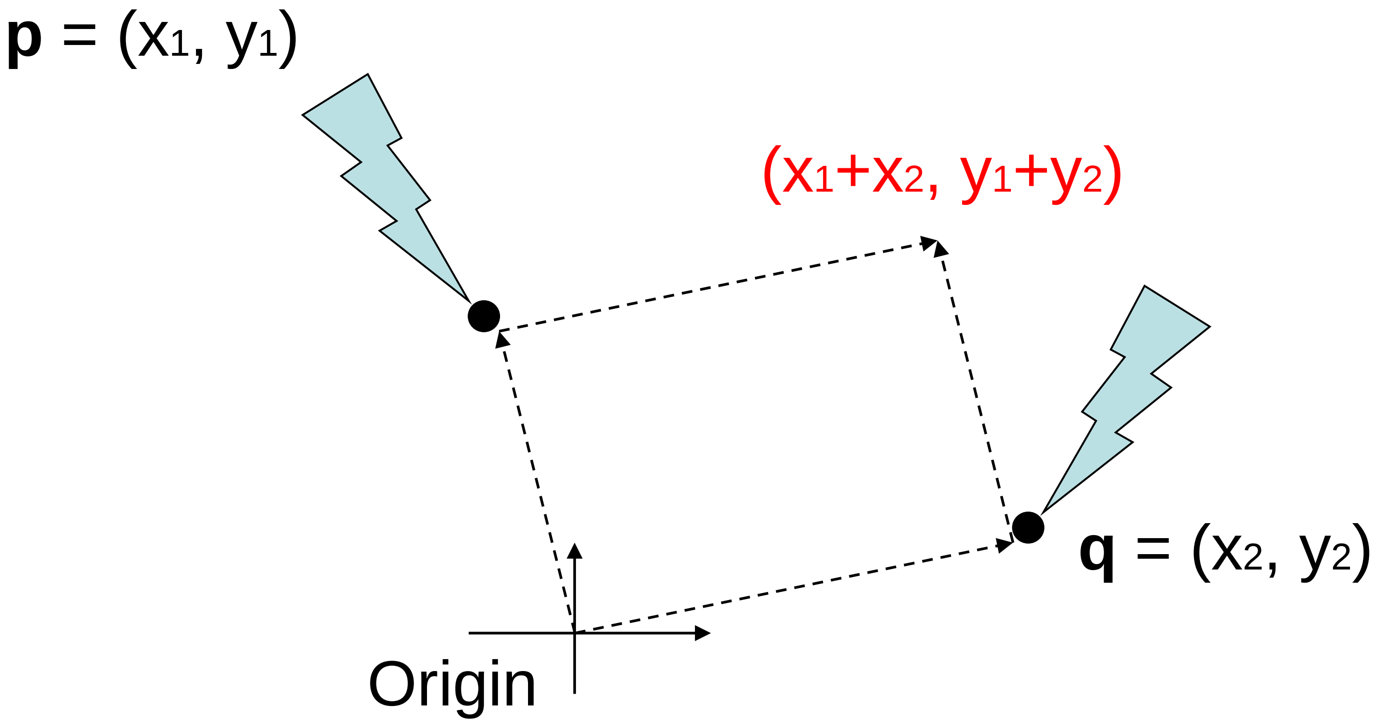
If you assume coordinates, …

$\mathbf{p} = (x_1, y_1)$

$\mathbf{q} = (x_2, y_2)$

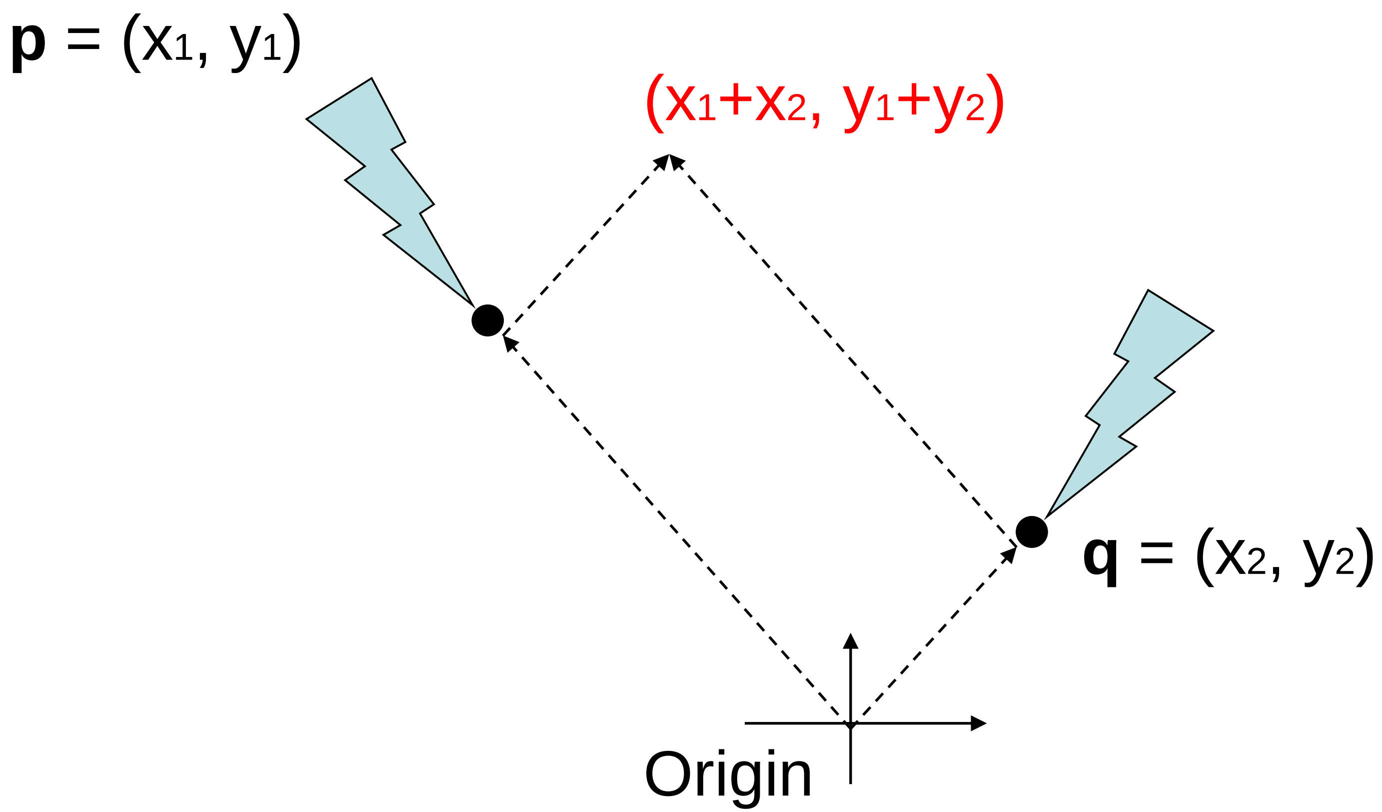
- The sum is (x_1+x_2, y_1+y_2)
  - Is it correct ?
  - Is it geometrically meaningful ?

If you assume coordinates, …



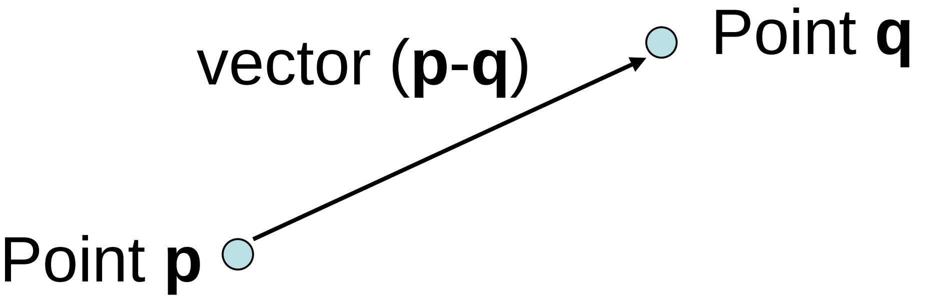
- Vector sum
  - (x_1, y_1) and (x_2, y_2) are considered as vectors from the origin to **p** and **q**, respectively.

If you select a different origin, …



- If you choose a different coordinate frame, you will get a different result

Points and Vectors



- A ***point*** is a position specified with coordinate values.
- A ***vector*** is specified as the difference between two points.
- If an ***origin*** is specified, then a **point** can be represented by a **vector from the origin.**
- But, a point is still not a vector in ***coordinate-free*** concepts.

Points & Vectors are Different!

- Mathematically (and physically),
- *Points* are **locations in space**.
- *Vectors* are **displacements in space**.

- An analogy with time:
- *Times*, (or datetimes) are **locations in time**.
- *Durations* are **displacements in time**.

Vector and Affine Spaces

- ***Vector space***
  - Includes vectors and related operations
  - No points

- ***Affine space***
  - Superset of vector space
  - Includes vectors, points, and related operations

Vector spaces

- A ***vector space*** consists of
  - Set of vectors, together with
  - Two operations: addition of vectors and multiplication of vectors by scalar numbers

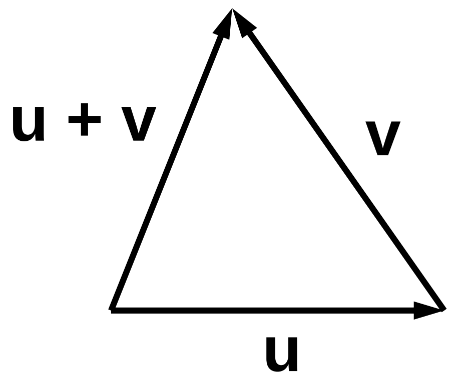
- A ***linear combination*** of vectors is also a vector

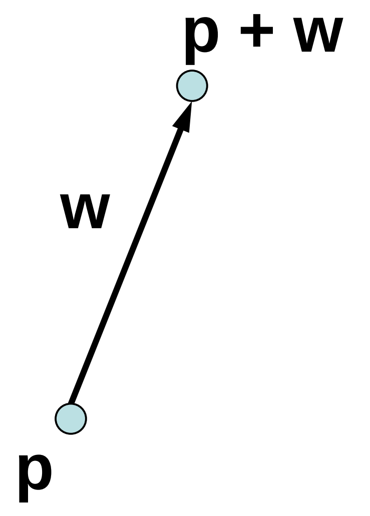
$$\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_N \in V \quad \Rightarrow \quad c_0 \mathbf{u}_0 + c_1 \mathbf{u}_1 + \cdots + c_N \mathbf{u}_N \in V$$

Affine Spaces

- An ***affine space*** consists of
  - Set of points, an associated vector space, and
  - Two operations: the difference between two points and the addition of a vector to a point

Addition



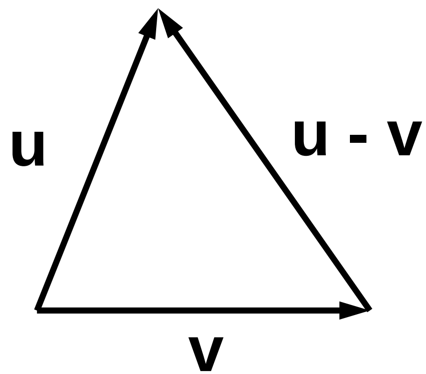


u + **v** is a vector

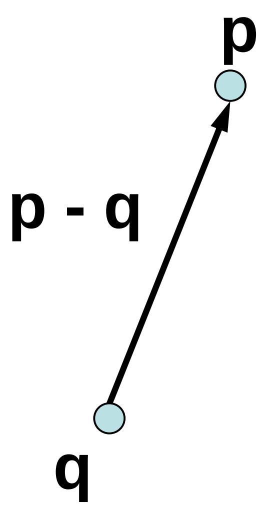
p + **w** is a point

u, v, w : vectors
p, q : points

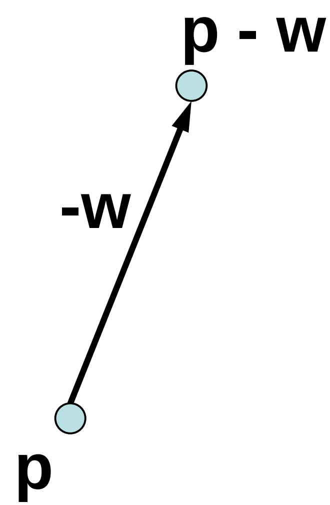
Subtraction



u - **v** is a vector



p - **q** is a vector



p - **w** is a point

u, v, w : vectors
p, q : points

Scalar Multiplication

scalar • vector = vector

1 • point = point

0 • point = vector

c • point = (undefined)   if ($c \neq 0,1$)

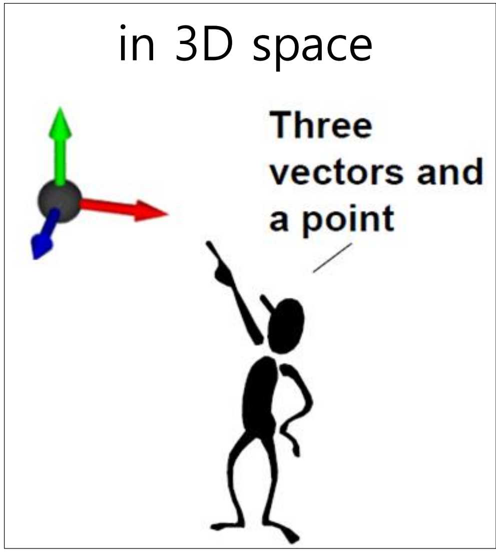
Affine Frame

- A ***frame*** is defined as a set of vectors $\{\mathbf{v}_i \mid i=1, \ldots, N\}$ and a point **o**
  - Set of vectors $\{\mathbf{v}_i\}$ are bases of the associate vector space
  - **o** is an origin of the frame
  - *N* is the dimension of the affine space
  - Any point **p** can be written as

$$\mathbf{p} = \mathbf{o} + c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_N\mathbf{v}_N$$

  - Any vector **v** can be written as

$$\mathbf{v} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_N\mathbf{v}_N$$



Summary

$$
\begin{array}{lll}
\text{point} + \text{point} = \text{undefined} & & \\
\text{point} - \text{point} = \text{vector} & & \\
\text{point} \pm \text{vector} = \text{point} & & \\
\text{vector} \pm \text{vector} = \text{vector} & & \\
\text{scalar} \bullet \text{vector} = \text{vector} & & \\
\text{scalar} \bullet \text{point} = \text{point} & & \text{iff scalar} = 1 \\
\phantom{\text{scalar} \bullet \text{point}} = \text{vector} & & \text{iff scalar} = 0 \\
\phantom{\text{scalar} \bullet \text{point}} = \text{undefined} & & \text{otherwise}
\end{array}
$$

Points & Vectors in Homogeneous Coordinates

- In 3D spaces,
- A **point** is represented: (x, y, z, **1**)
- A **vector** can be represented: (x, y, z, **0**)

$(x_1, y_1, z_1, 1) + (x_2, y_2, z_2, 1) = (x_1+x_2, y_1+y_2, z_1+z_2, 2)$

point ***point*** ***undefined***

$(x_1, y_1, z_1, 1) - (x_2, y_2, z_2, 1) = (x_1-x_2, y_1-y_2, z_1-z_2, 0)$

point ***point*** ***vector***

$(x_1, y_1, z_1, 1) + (x_2, y_2, z_2, 0) = (x_1+x_2, y_1+y_2, z_1+z_2, 1)$

point ***vector*** ***point***

A Consistent Model

- **Behavior of affine frame coordinates is completely consistent with our intuition**
  - **Subtracting two points yields a vector**
  - **Adding a vector to a point produces a point**
  - **If you multiply a vector by a scalar you still get a vector**
  - **Scaling points gives a nonsense 4th coordinate element in most cases**

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ 1 \end{bmatrix} - \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 - b_1 \\ a_2 - b_2 \\ a_3 - b_3 \\ 0 \end{bmatrix} \qquad \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ 1 \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ 0 \end{bmatrix} = \begin{bmatrix} a_1 + v_1 \\ a_2 + v_2 \\ a_3 + v_3 \\ 1 \end{bmatrix}$$

Points & Vectors in Homogeneous Coordinates

- Multiplying affine transformation matrix to a point and a vector

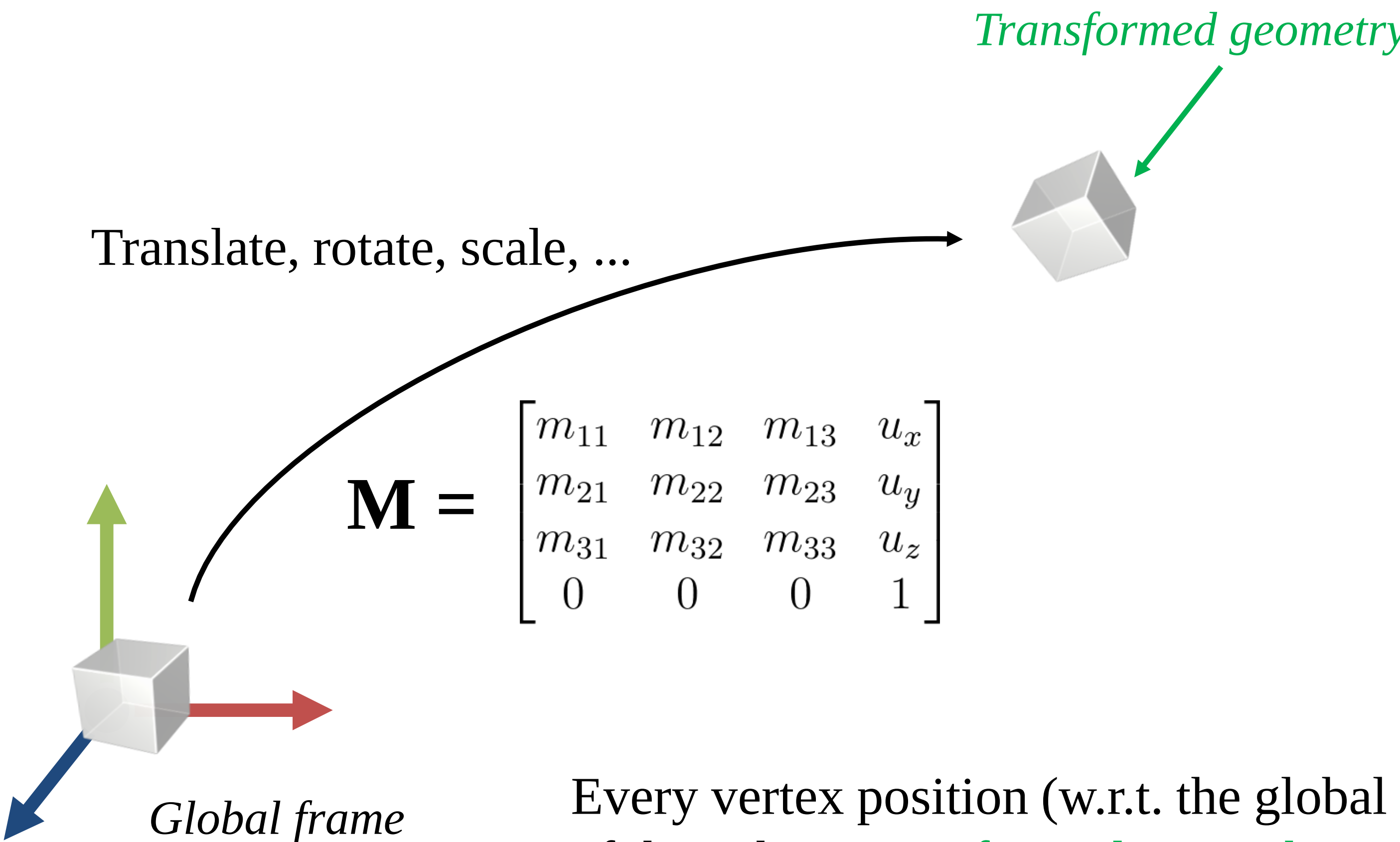
$$\begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ \boxed{1} \end{bmatrix} = \begin{bmatrix} M\mathbf{p} + \mathbf{t} \\ \boxed{1} \end{bmatrix} \qquad \begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \boxed{0} \end{bmatrix} = \begin{bmatrix} M\mathbf{v} \\ \boxed{0} \end{bmatrix}$$

point $\longrightarrow$ point　　　vector $\longrightarrow$ vector

- Note that translation is not applied to a vector!

Meanings of an Affine Matrix

1) A 4x4 Affine Transformation Matrix transforms a Geometry

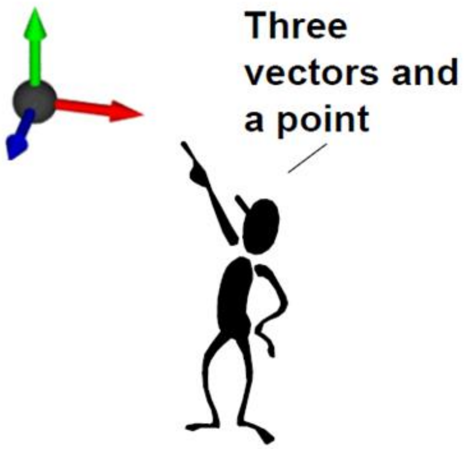


$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Every vertex position (w.r.t. the global frame) of the cube is transformed to another position (w.r.t. the global frame)

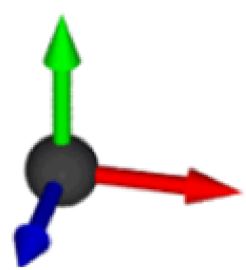
Review: Affine Frame

- An **affine frame** in 3D space is defined by three vectors and one point
  - Three vectors for x, y, z axes
  - One point for origin



Global Frame

- A **global frame** is usually represented by
  - Standard basis vectors for axes : $\hat{\mathbf{e}}_x, \hat{\mathbf{e}}_y, \hat{\mathbf{e}}_z$
  - Origin point : $\mathbf{0}$



Let's transform a global frame

- Apply M to a global frame, that is,
  - Multiply M with the x, y, z axis *vectors* and the origin *point* of the global frame:

x axis *vector*

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ \boxed{0} \end{bmatrix} = \begin{bmatrix} m_{11} \\ m_{21} \\ m_{31} \\ \boxed{0} \end{bmatrix}$$

y axis *vector*

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \\ \boxed{0} \end{bmatrix} = \begin{bmatrix} m_{12} \\ m_{22} \\ m_{32} \\ \boxed{0} \end{bmatrix}$$

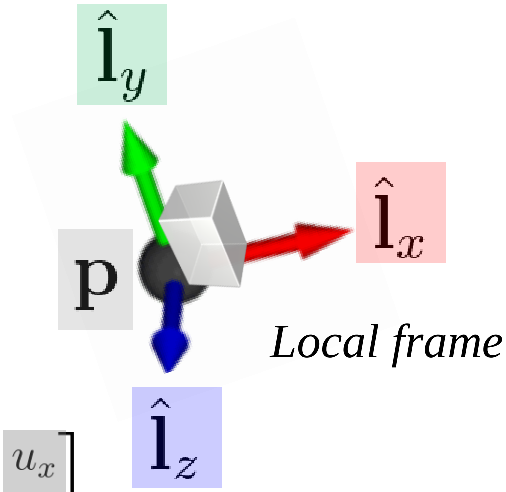
z axis *vector*

$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ \boxed{0} \end{bmatrix} = \begin{bmatrix} m_{13} \\ m_{23} \\ m_{33} \\ \boxed{0} \end{bmatrix}$$

origin *point*

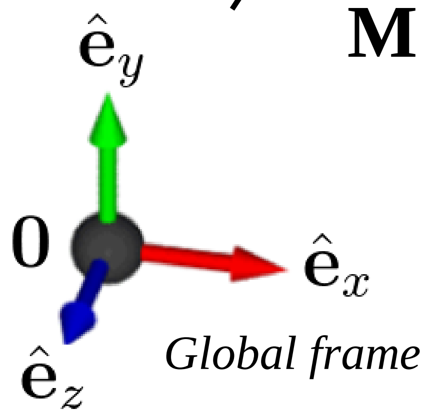
$$\begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \boxed{1} \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \\ u_z \\ \boxed{1} \end{bmatrix}$$

2) A 4x4 Affine Transformation Matrix **defines an Affine Frame w.r.t. Global Frame**



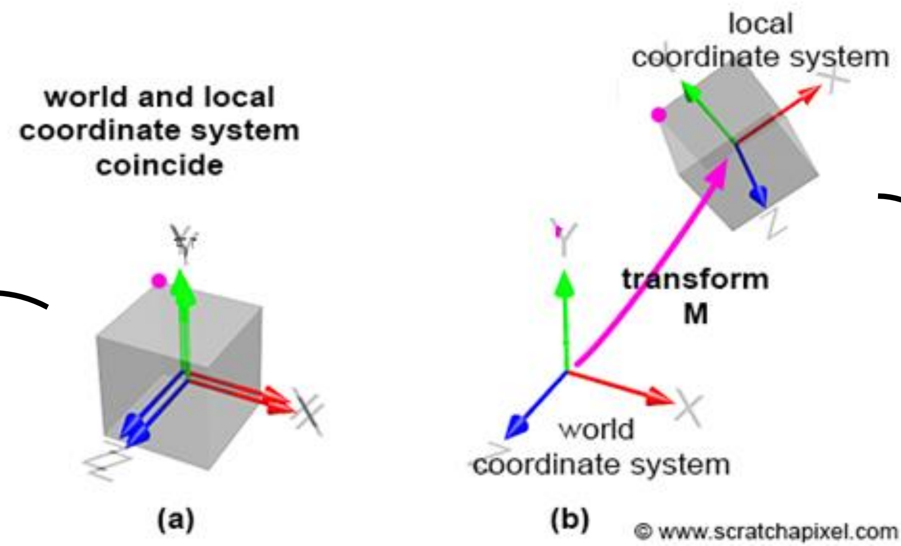
$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x axis *vector* | y axis *vector* | z axis *vector* | origin *point*



: The axis vectors and origin point of the **object's local frame represented in the global frame**

Examples



This local frame
is defined by:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x axis *vector*

y axis *vector*

z axis *vector*

origin *point*

of the local frame ***represented in the global frame***

This local frame
is defined by:

origin *point*

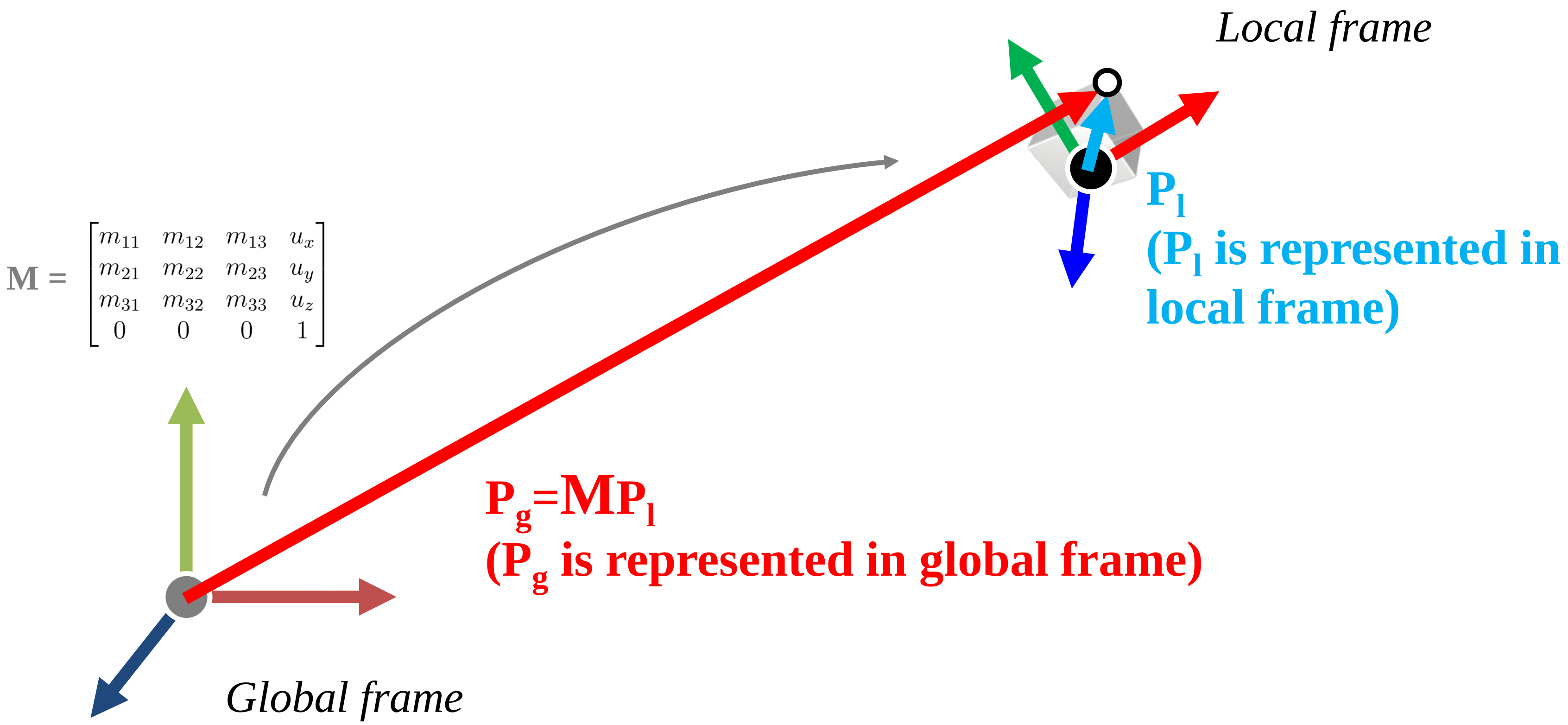
$$M = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_1 \\ m_{21} & m_{22} & m_{23} & u_2 \\ m_{31} & m_{32} & m_{33} & u_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

x axis *vector*

y axis *vector*

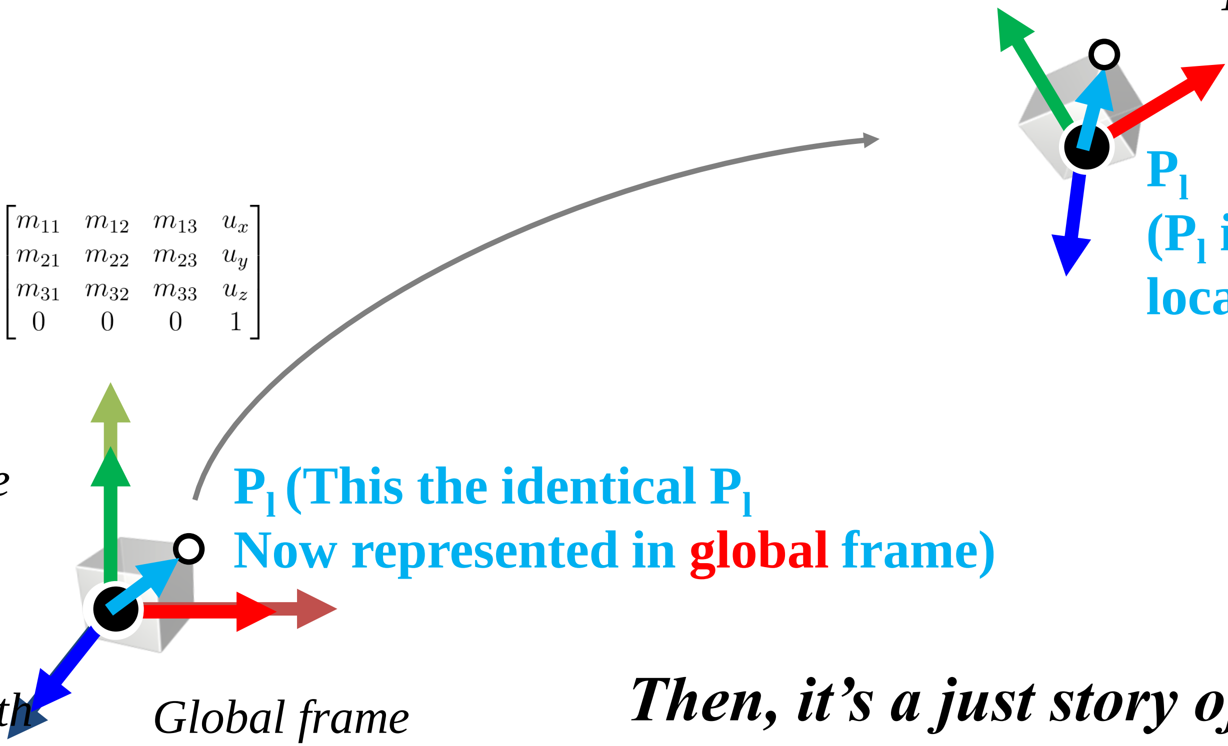
z axis *vector*

3) A 4x4 Affine Transformation Matrix transforms a Point Represented in One Frame to a Point Represented in Another Frame



3) A 4x4 Affine Transformation Matrix transforms a Point Represented in One Frame to a Point Represented in Another Frame Because...

$$\mathbf{M} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & u_x \\ m_{21} & m_{22} & m_{23} & u_y \\ m_{31} & m_{32} & m_{33} & u_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



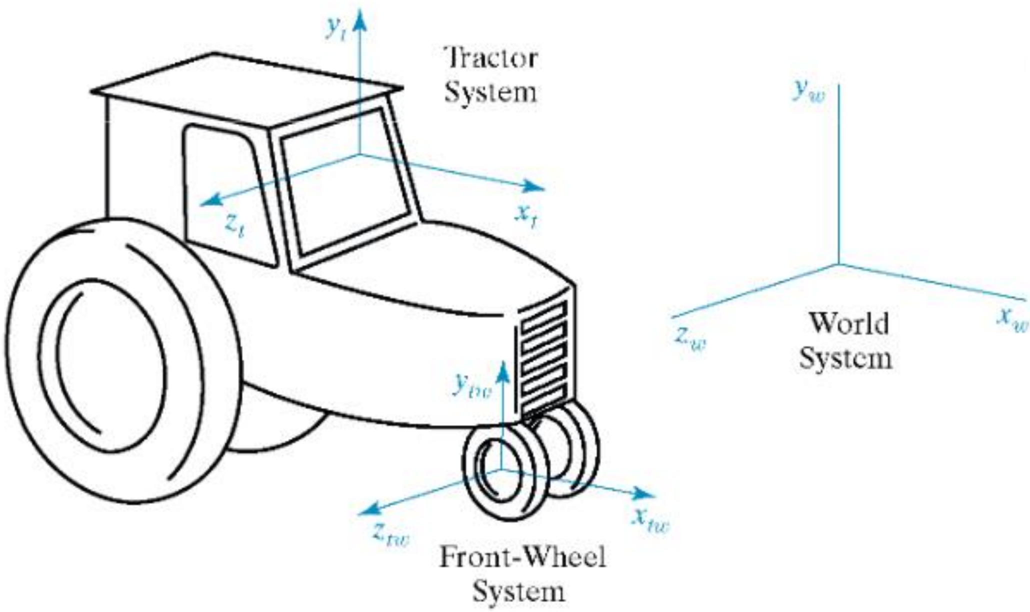
Let's say we have the same cube object and its local frame coincident with the global frame

Then, it's a just story of transforming a geometry!

Hierarchical Modeling

Hierarchical Modeling

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



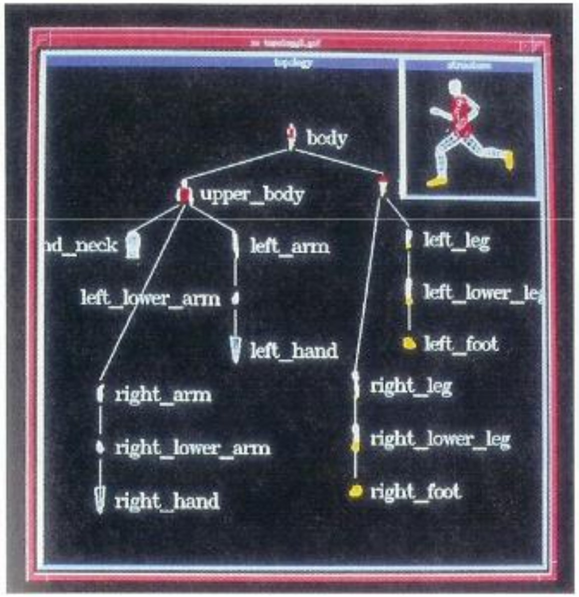
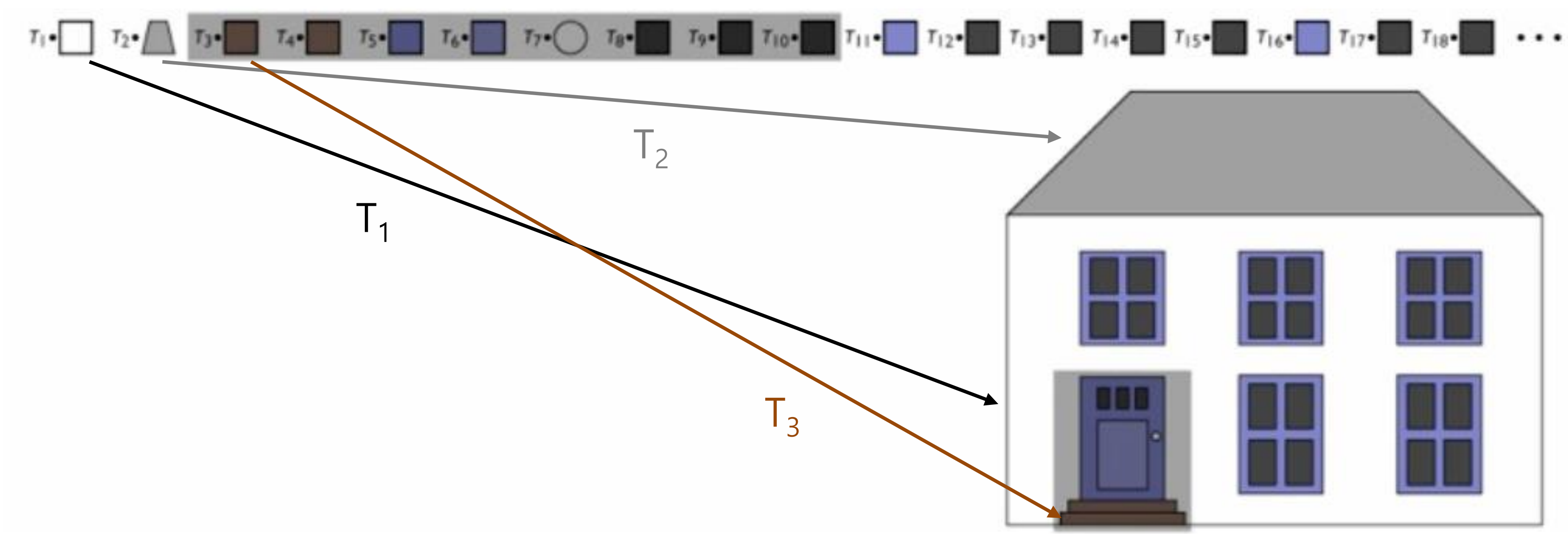


FIGURE 14-4 An object hierarchy generated using the PHIGS Toolkit package developed at the University of Manchester. The displayed object tree is itself a PHIGS structure. (*Courtesy of T. L. J. Howard, J. G. Williams, and W. T. Hewitt, Department of Computer Science, University of Manchester, United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

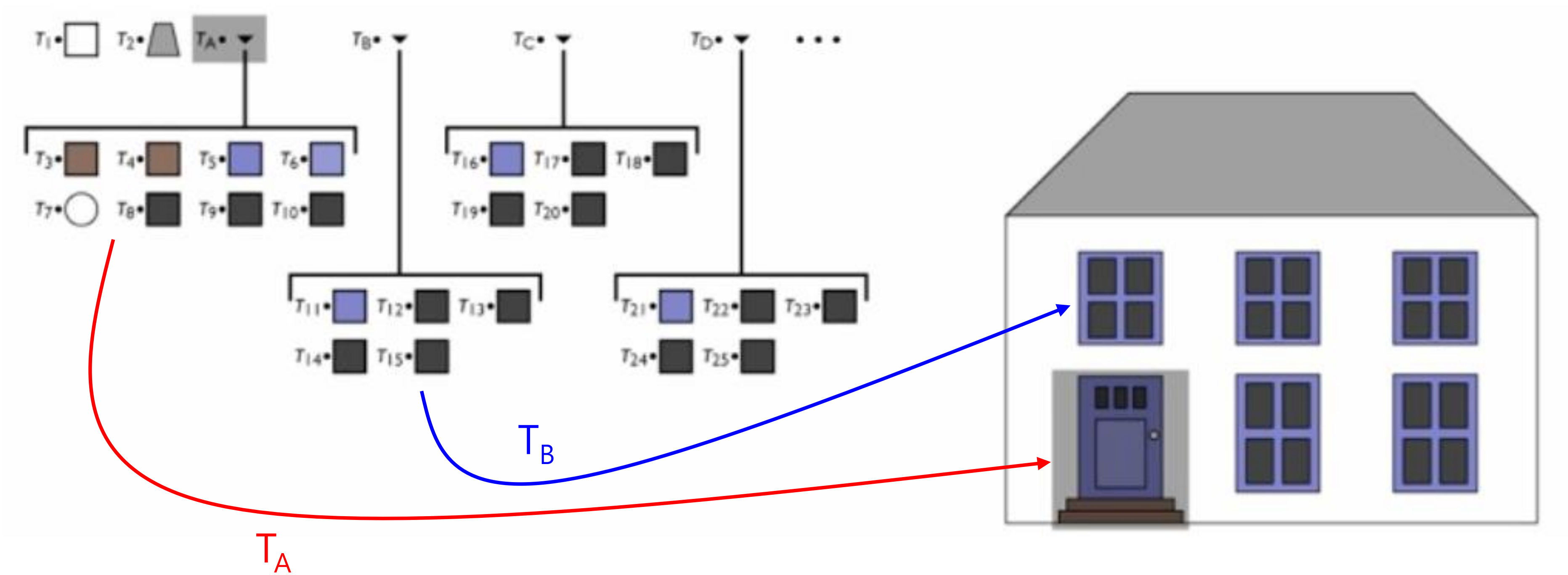
Example

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



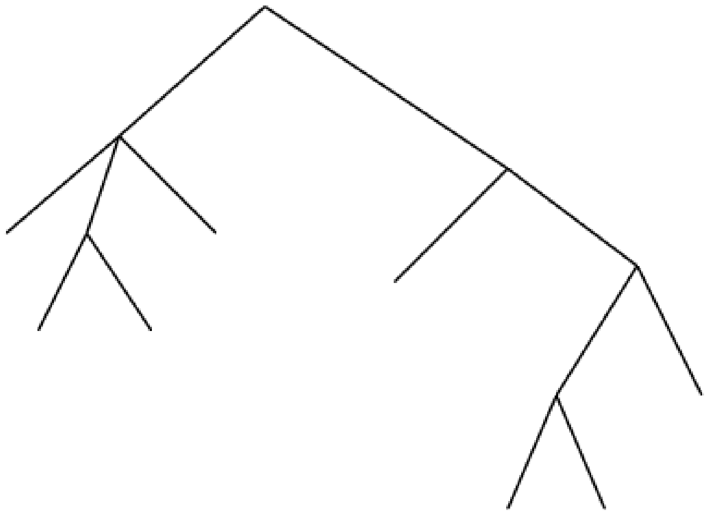
“Grouping”

- Treat 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

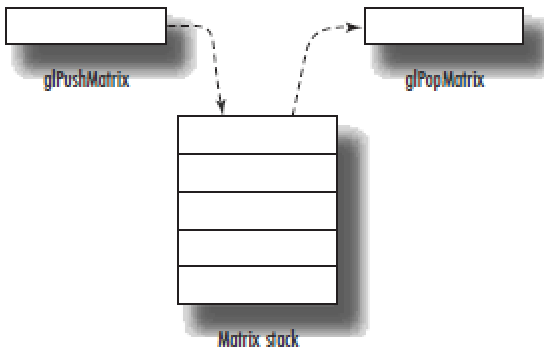
Hierarchical Modeling in OpenGL

- OpenGL provides a useful way of drawing objects in the hierarchical structure (scene graph)

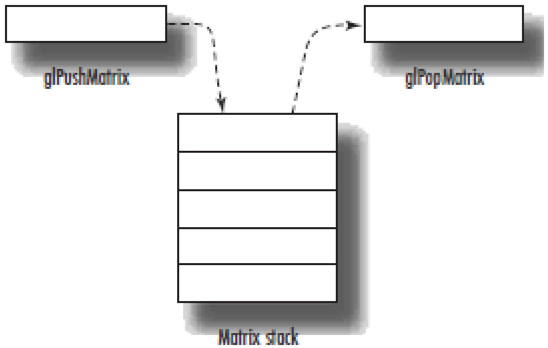
- **-> Matrix stack**

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)

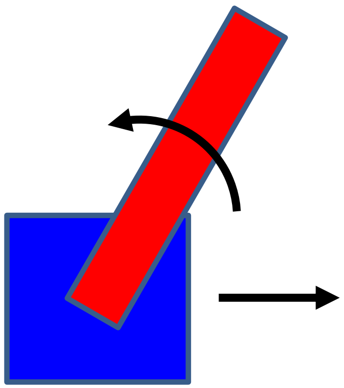


OpenGL Matrix Stack



- 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 **numbers of glPushMatrix() calls and glPopMatrix() calls must be the same.**

A simple example



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) # to draw base
  - **TS**
  - T
  - I
- Draw a box
- glPopMatrix()
  - **T**
  - I
- glPushMatrix()
  - **T**
  - T
  - I
- glRotate(R) # to rotate arm
  - **TR**
  - T
  - I
- glPushMatrix()
  - **TR**
  - TR
  - T
  - I
- glScale(U) # to draw arm
  - **TRU**
  - TR
  - T
  - I
- Draw a box
- glPopMatrix()
  - **TR**
  - T
  - I
- glPopMatrix()
  - **T**
  - I
- glPopMatrix()
  - **I**

[Practice] Matrix Stack (modify lec5 code)

```
def render(camAng, count):
    # edit here

    # blue base transformation
    glPushMatrix()
    glTranslatef(-.5+(count%360)*.003, 0, 0)

    # blue base drawing
    glPushMatrix()
    glScalef(.2, .2, .2)
    glColor3ub(0, 0, 255)
    drawBox()
    glPopMatrix()

    # red arm transformation
    glPushMatrix()
    glRotatef(count%360, 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()
```

```
# modify main() function
def main():
    if not glfw.init():
        return
    window = glfw.create_window(640,640,'Matrix Stack',
None,None)
    if not window:
        glfw.terminate()
        return
    glfw.make_context_current(window)
    glfw.set_key_callback(window, key_callback)
    glfw.swap_interval(1)

    count = 0
    while not glfw.window_should_close(window):
        glfw.poll_events()
        render(gCamAng, count)
        glfw.swap_buffers(window)
        count += 1

    glfw.terminate()
```

Next Time

- No classes: Sep 26, Oct 3

- Oct 10:
  - Rendering Pipeline, Viewing Transformation
  - Projection Transformation, Viewport Transformation

- Assignment 3 (Due date: Oct 9, 23:59)

- Acknowledgement: Some materials come from the lecture slides of
  - Prof. Jehee Lee, SNU, http://mrl.snu.ac.kr/courses/CourseGraphics/index_2017spring.html
  - Prof. Steve Marschner, Cornell Univ., http://www.cs.cornell.edu/courses/cs4620/2014fa/index.shtml
  - Prof. Sung-eui Yoon, KAIST, https://sglab.kaist.ac.kr/~sungeui/CG/