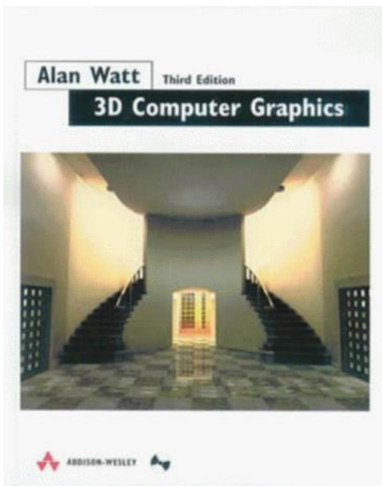
COM3503/4503/6503: 3D Computer Graphics

Lecture 2: Transformations and scene graphs

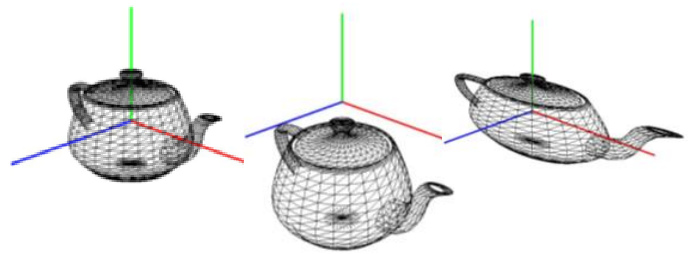


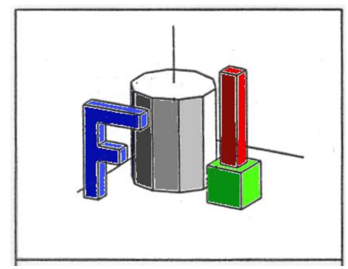
Dr. Steve Maddock
s.maddock@sheffield.ac.uk

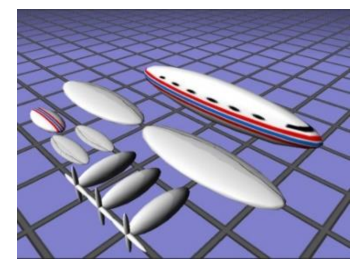
1. Introduction

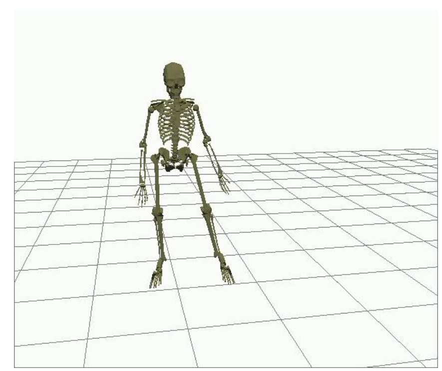
Use transformations to:

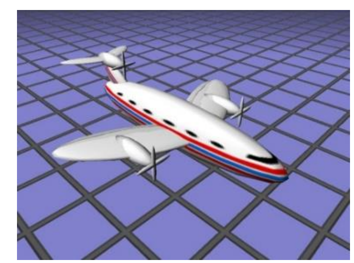
- **Manipulate individual objects**
- **Build scenes**
- **Build complex objects from pieces**
- Control relationship between parts in hierarchical (articulating) objects
- Conversion between coordinate systems



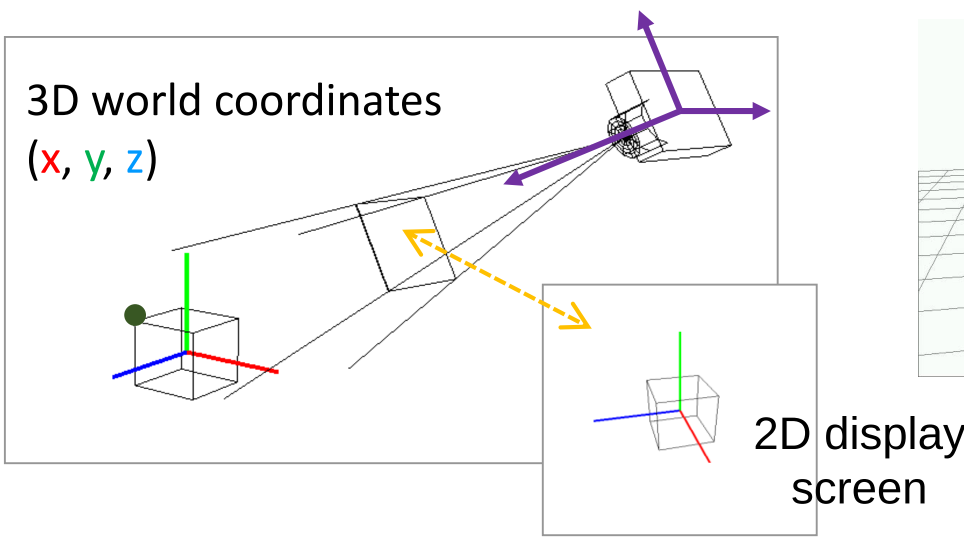








http://viz.aset.psu.edu/jack/java3d/



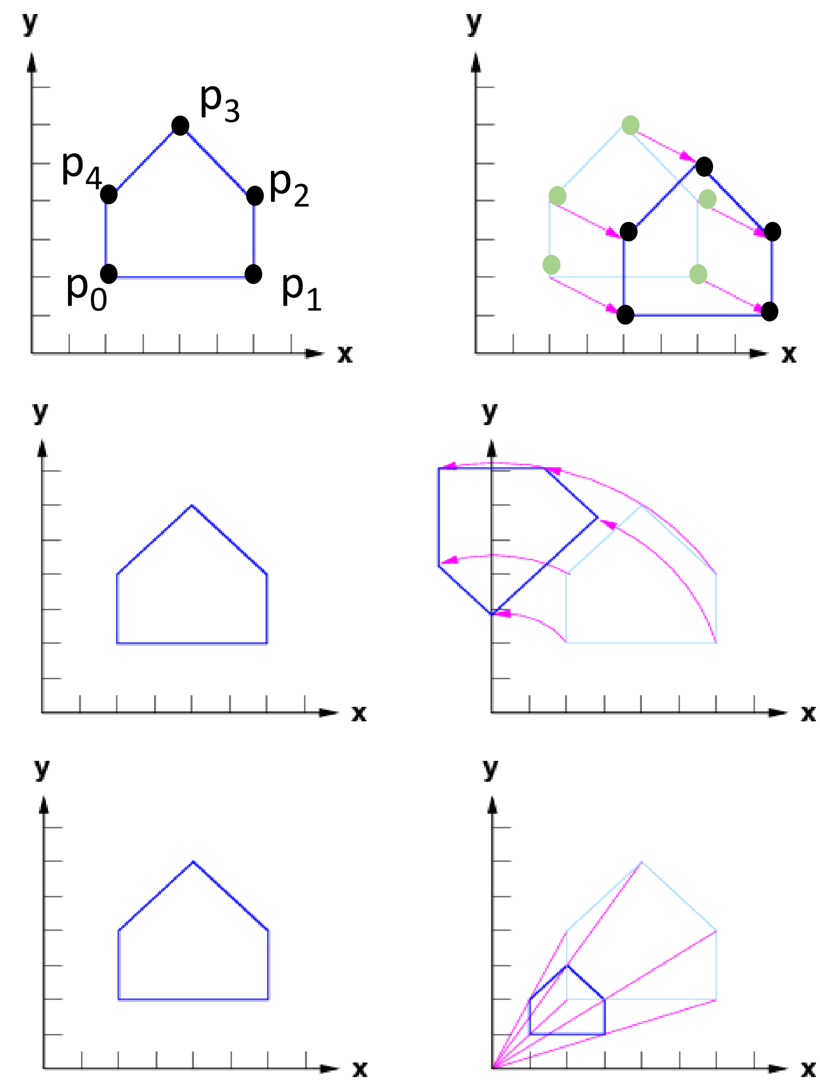
2. Two-dimensional (2D) transformations: summary

- A vertex is represented as a vector
- Transformation is achieved using matrix arithmetic for each vertex

$$\begin{pmatrix} q_x \\ q_y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} p_x \\ p_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

New position	Rotate and/or Scale	Old position	Translate

- Issue: translation is a separate operation



2. Two-dimensional (2D) transformations: summary

Homogeneous coordinates

- (x, y) → (wx, wy, w) for any constant w<>0
- The vertex representation is augmented with an extra '1': (x, y, 1)
- The matrix representation becomes 3x3 for a 2D system

$$M = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad M = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

scale | rotate (anti-clockwise) | translate

- Now, we have

$$q = Mp$$

where

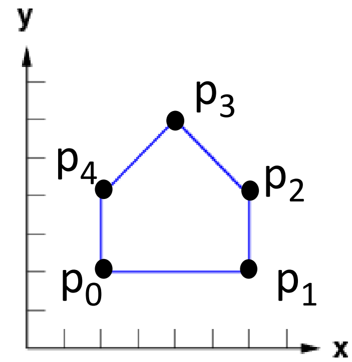
$$M = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix}$$

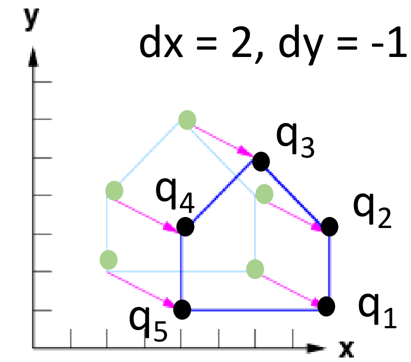
2. Two-dimensional (2D) transformations: summary

- Using vectors for translation:

$$q_i = p_i + T \qquad \text{where } T = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

$$\begin{pmatrix} q_{i,x} \\ q_{i,y} \end{pmatrix} = \begin{pmatrix} p_{i,x} \\ p_{i,y} \end{pmatrix} + \begin{pmatrix} dx \\ dy \end{pmatrix}$$



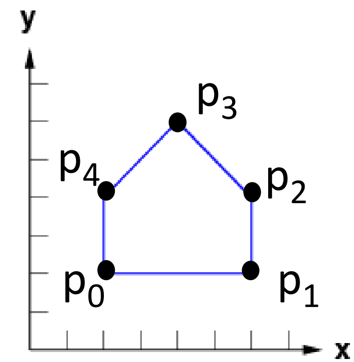


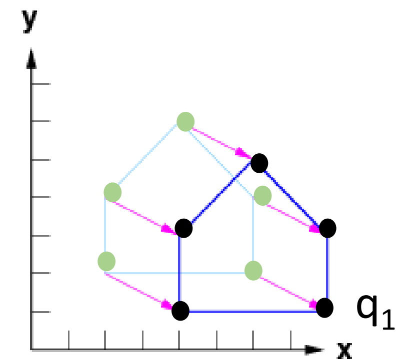
Homogeneous coordinates

- Example, for p_1, (6,2) becomes (6,2,1)

$$q_1 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 6 \\ 2 \\ 1 \end{bmatrix}$$

$$q_1 = \begin{bmatrix} 8 \\ 1 \\ 1 \end{bmatrix}$$



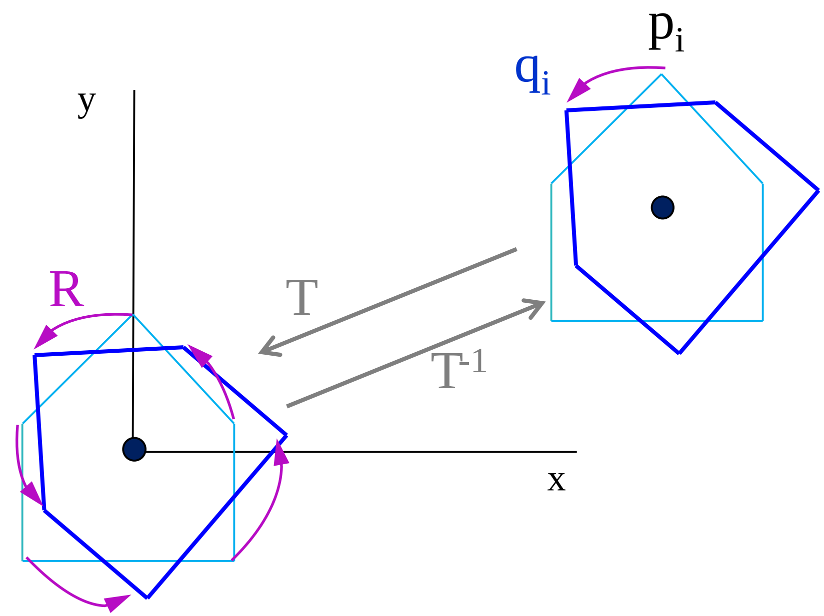


2. Two-dimensional (2D) transformations: summary

- *Rotation about an arbitrary point*
- *General plan*: Translate to world origin (T), rotate (R), and translate back again (T^{-1})

$$q_i = T^{-1}\left(R(T\,p_i)\right)$$

$$q_i = (T^{-1}\,R\,T)\,p_i$$



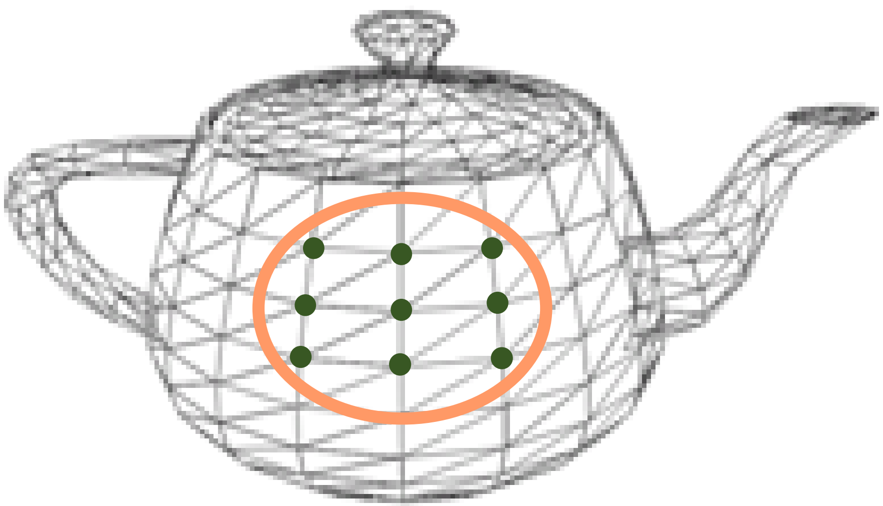
- Combining these:

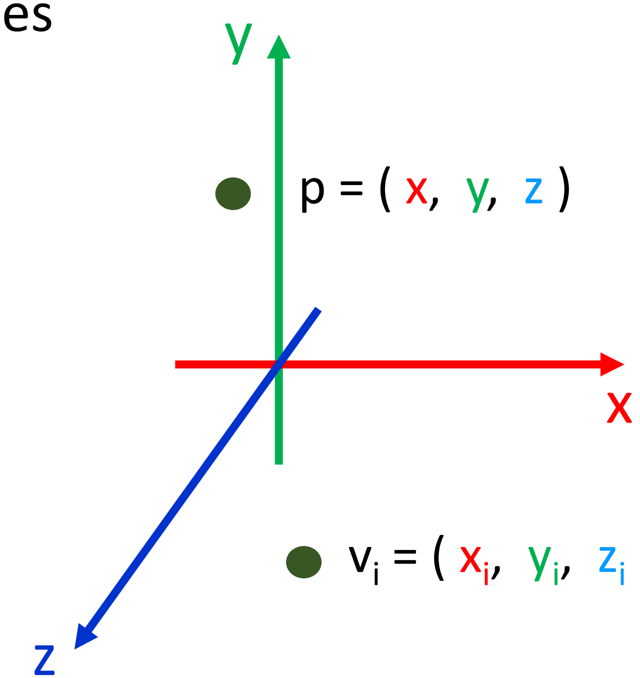
$$M = T^{-1}RT = T^{-1}(RT)$$

$$q_i = M\,p_i$$

3. 3D

- Vertices (points) are connected to make triangles which represent the surface of the object





- In a later lecture we will look at a range of data structures for representing collections of vertices and triangles
- **Today**: how to transform a set of points/vertices

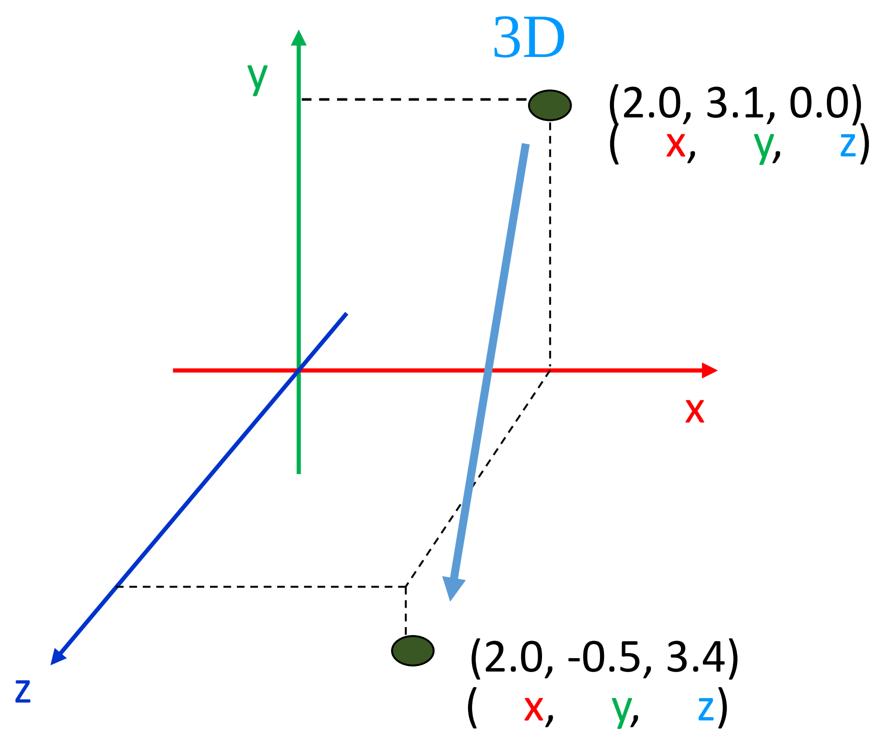
4. Three-dimensional (3D) transformations

- Homogeneous coordinates: (x,y,z) → (wx,wy,wz,w)
- Transformations are now represented as 4x4 matrices
- Example: 3D Translation

$$q_i = M\, p_i$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

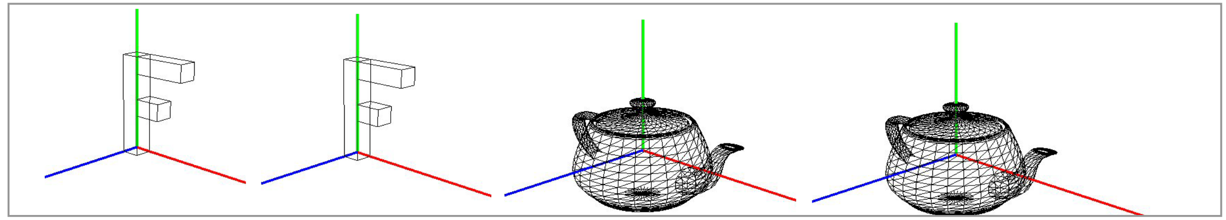
$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -3.6 \\ 0 & 0 & 1 & 3.4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



4. 3D transformations

Identity

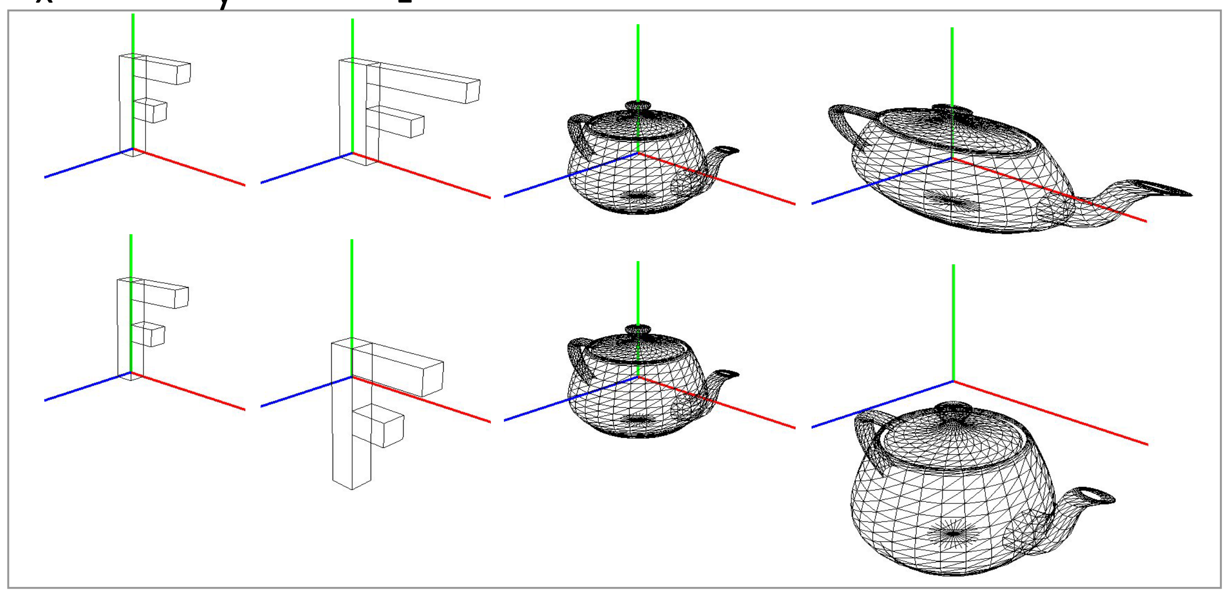
$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



3D scale

$$M = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$s_x = 2, \; s_y = 1, \; s_z = 1$



3D translation

$$M = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$t_x = 2, \; t_y = 0, \; t_z = 2$

4. 3D transformations

- Rotation – three different matrices, one for rotation about each acis

3D rotation about the x axis

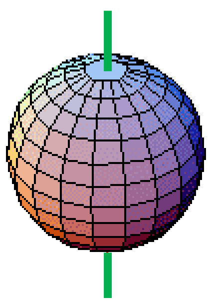
$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

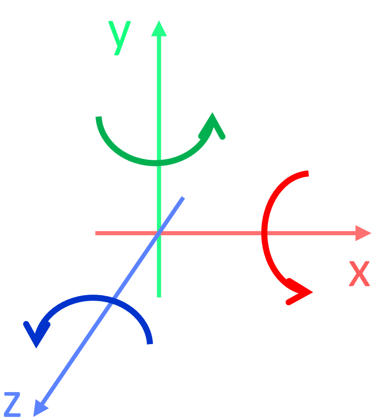
3D rotation about the y axis

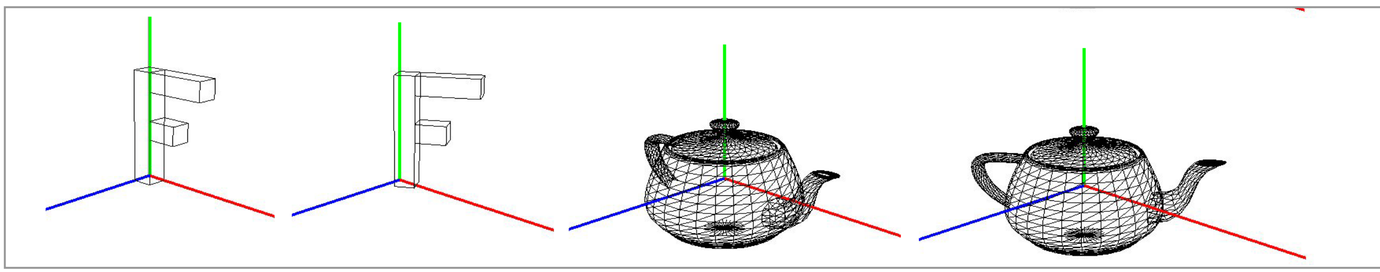
$$R_y = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D rotation about the z axis

$$R_z = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

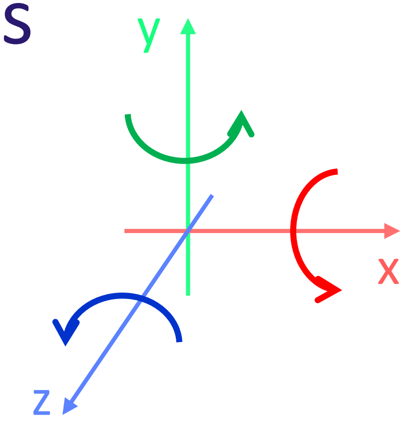




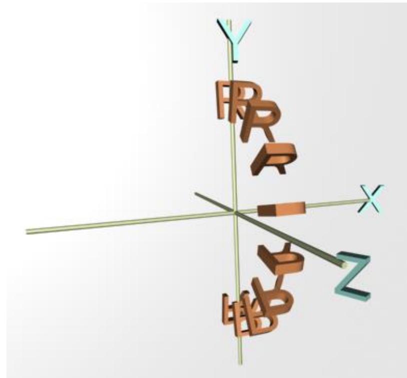


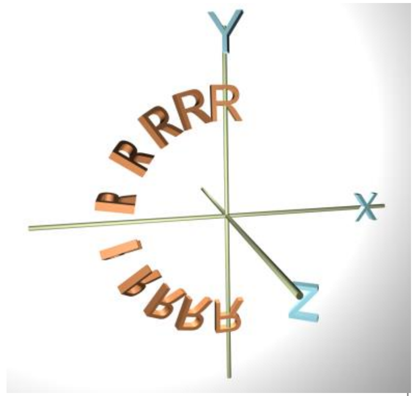
$$\begin{pmatrix} 0.866 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4.1 Representing rotation: Euler angles



- General idea: Specify how much to rotate about each of the X, Y and Z axis (in some decided order)
- Extension: Specify the angle and an axis
  - rotate(angle, x, y, z)
  - Rotate anticlockwise about line between origin and x,y,z
- Issues:
  - Hard to 'visualise' multiple rotations
  - Movement path between orientations is not unique →
- In practice interpolation between rotations is done in quaternion space using Spherical Linear IntERPolation
  - Usage: Euler → quaternions → Euler





5. Composition of transformations

- Concatenate a series of matrices to form a net transformation matrix:

$V' = M_1 V$

$V'' = M_2 V'$

$V'' = M_2 M_1 V$ (M_1 is applied to V, then M_2 is applied)

$V'' = M_c V$ where $M_c = M_2 M_1$

- A general transformation matrix will be of the form:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & t_x \\ a_{21} & a_{22} & a_{23} & t_y \\ a_{31} & a_{32} & a_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

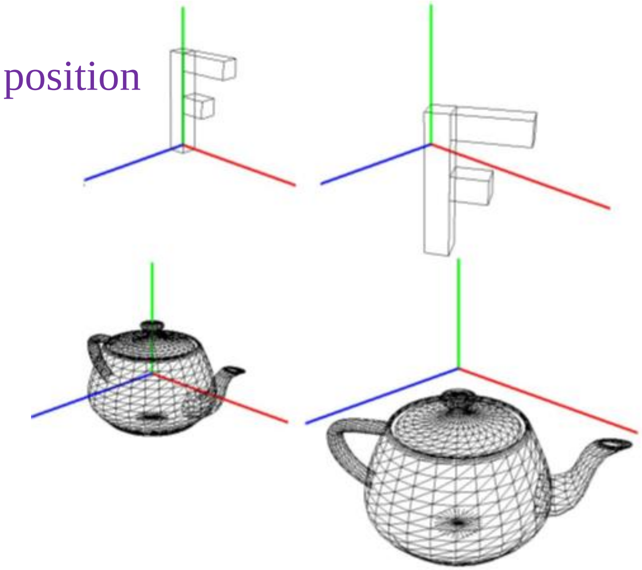
- The 3x3 upper-left sub-matrix A is the net rotation and scaling, while (t_x,t_y,t_z) gives the net translation.

5.1 Order matters…

New vertex position = matrix × old vertex position

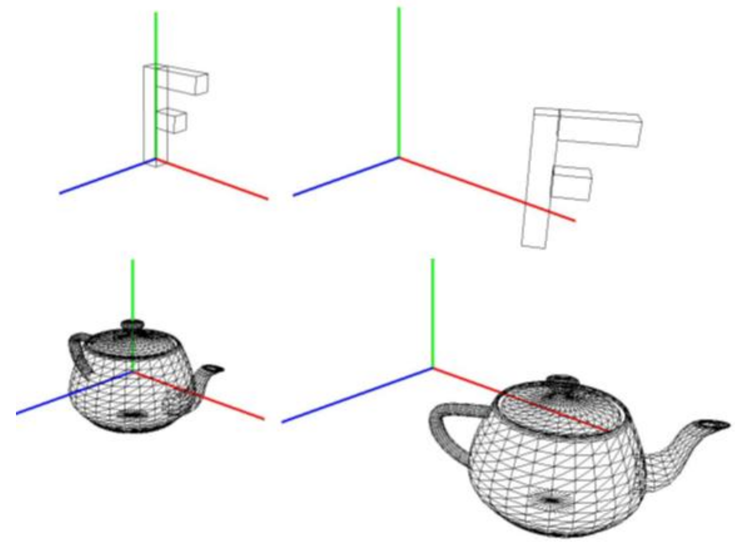
Rotate first $\quad V_i' = M\, V_i$

$$M = \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.866 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.866 & 0 & 0.5 & 2 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



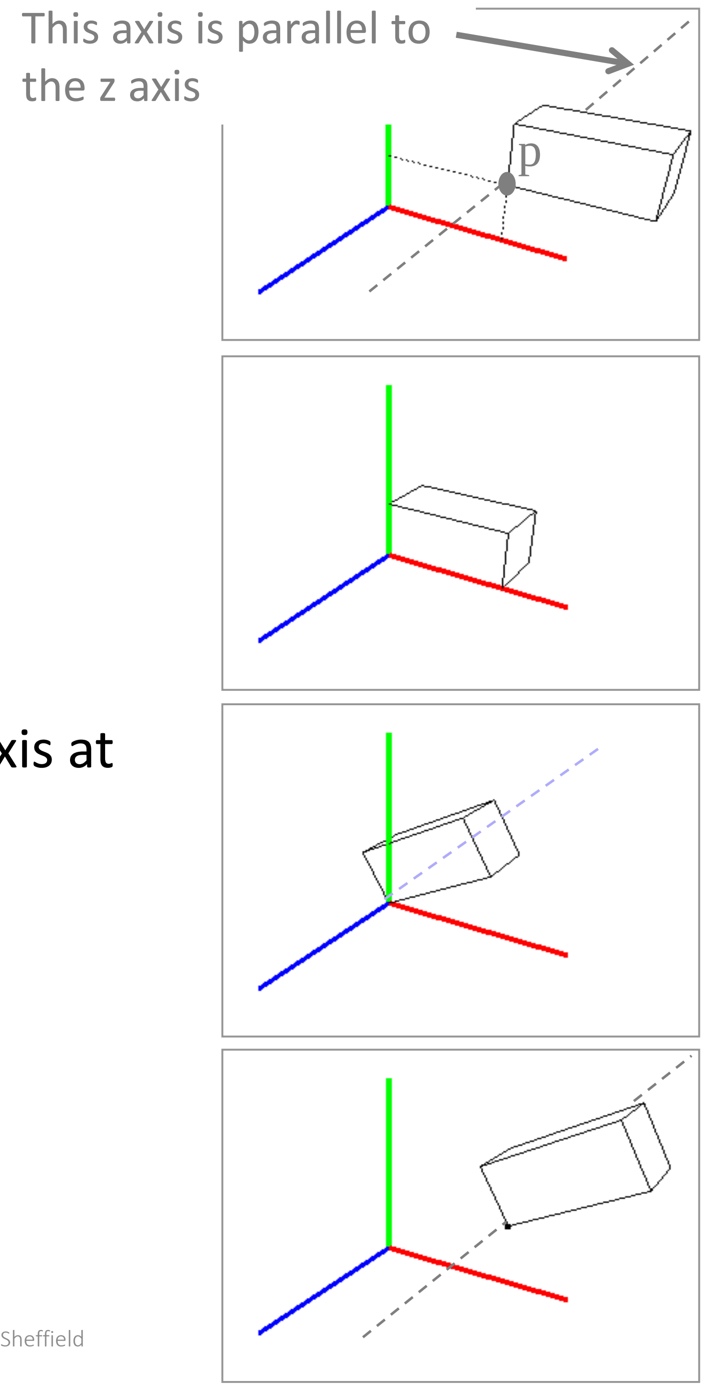
Translate first $\quad V_i' = M\, V_i$

$$M = \begin{pmatrix} 0.866 & 0 & 0.5 & 0 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.866 & 0 & 0.5 & 2.732 \\ 0 & 1 & 0 & 0 \\ -0.5 & 0 & 0.866 & 0.732 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



5.2 Example: Rotate about an arbitrary axis A

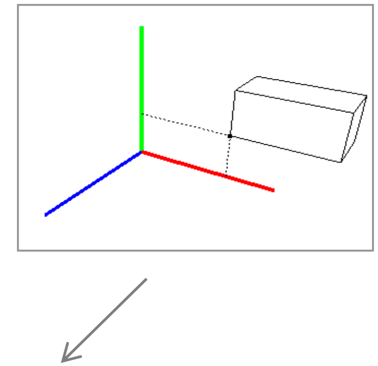
- *General plan*:
  - Translate to origin
  - Align arbitrary axis A to one of x, y or z axes
  - Rotate
  - Inverse of align to axis
  - Translate back again
- Rotate object about an axis parallel to the z axis at point $(t_x, t_y, 0)$:
  - (a) One edge of object is on axis passing through point $p=(t_x, t_y, 0)$
  - (b) Translate object by p to origin, so that object edge is aligned with z axis
  - (c) Rotate about the z axis
  - (d) Translate object back to position p

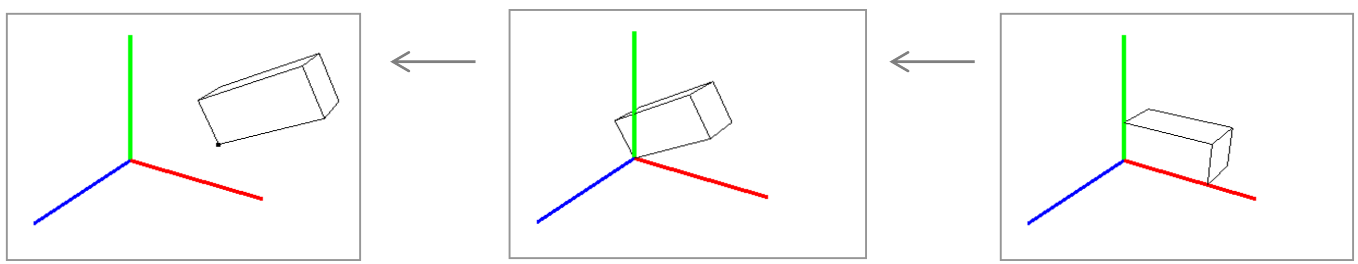


5.3 The net transformation matrix:

$$V_i' = MV_i$$

i = 1..number of vertices





$$M = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3: translate | 2: rotate | 1: translate

5.4 Old vs modern OpenGL

Fixed function pipeline

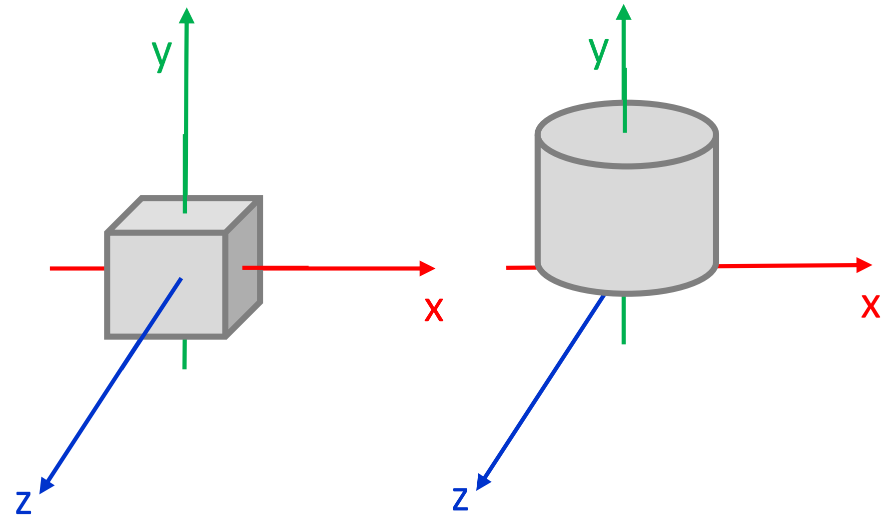
- Commands for each transformation: glRotate[fd](angle, x, y, z)
- These alter the matrix stack which is a persistent variable for the OpenGL context
- Subsequent objects that are drawn are affected by the matrix stack

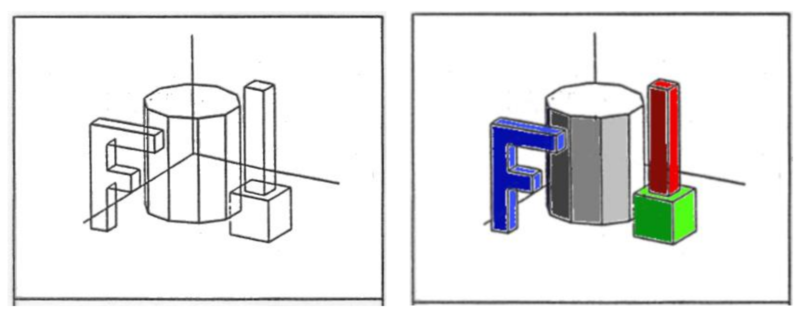
Programmable pipeline

- We need to implement a maths library to do the transformations
- The transformed vertices are then sent to the GPU using relevant buffers
- glm is oft-used, but it is a C++ maths library
- See a later Lab class for a Java-equivalent of this maths library

6. Scene building

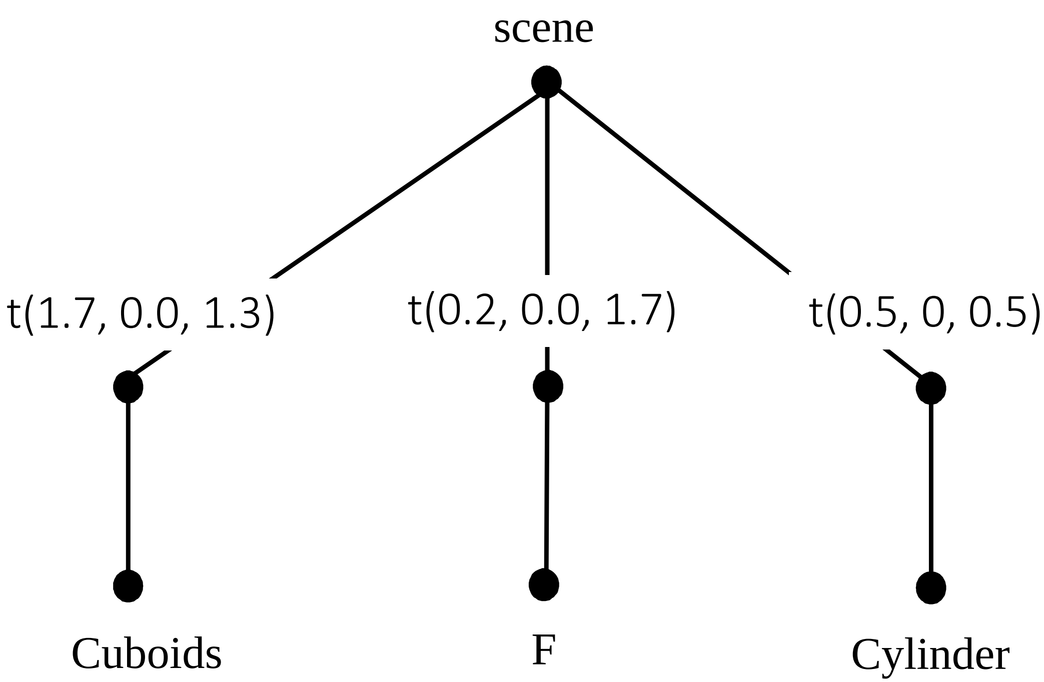
- Make scene from individual objects
- Each object has its own local coordinate system:
  - A cube may be centred at origin
  - For a cylinder, it is more convenient to have a coordinate axis that coincides with its long axis
- Transform object in its local coordinate system, e.g. a scale
- Further transformations place objects in the world coordinate system, e.g. a translation

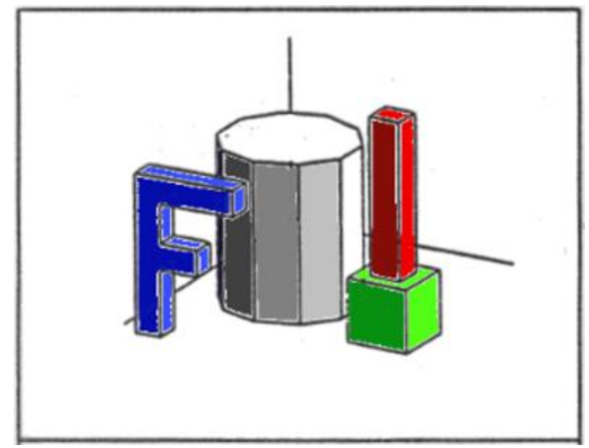




6.1 A scene graph

- We can represent the scene using a scene graph
- Transformations can be represented as explicit nodes in the scene graph



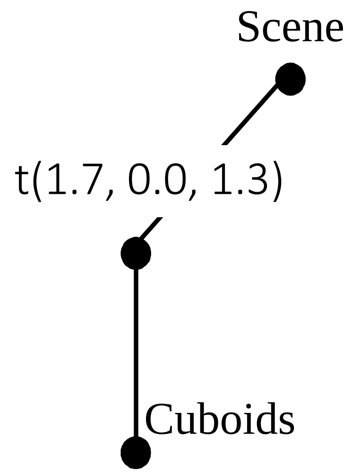


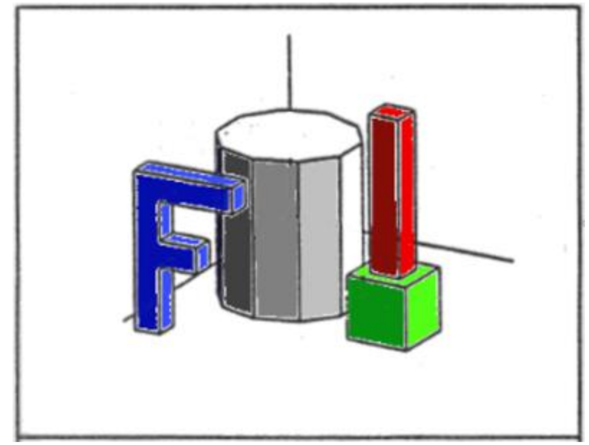
Cuboids:	t(1.7, 0.0, 1.3)
F:	t(0.2, 0.0, 1.7)
Cylinder:	t(0.5, 0, 0.5)

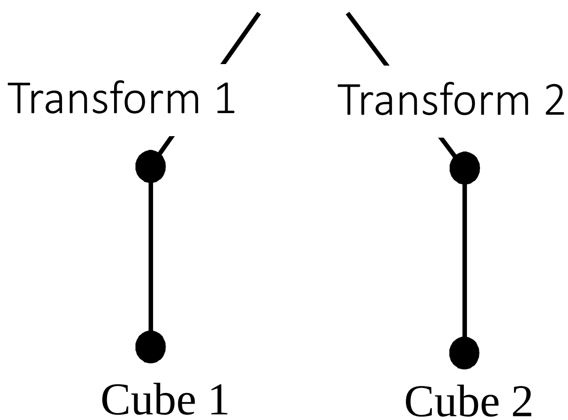
s = scale, r = rotate, t = translate

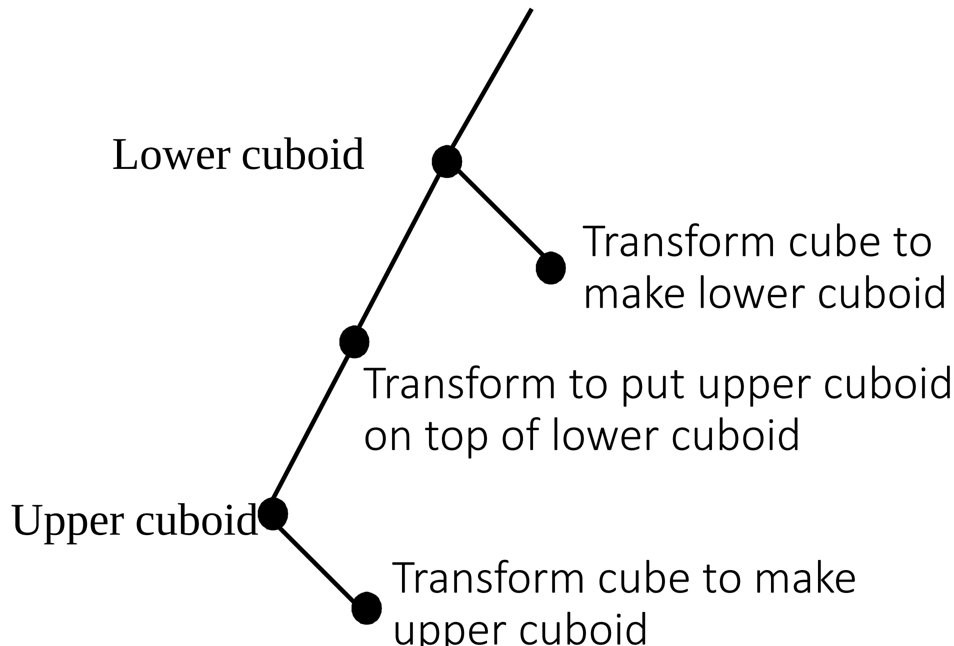
6.2 The stack of cuboids

- Alternative ways to structure the rest of the scene graph

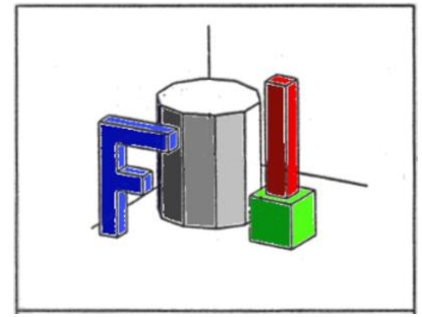








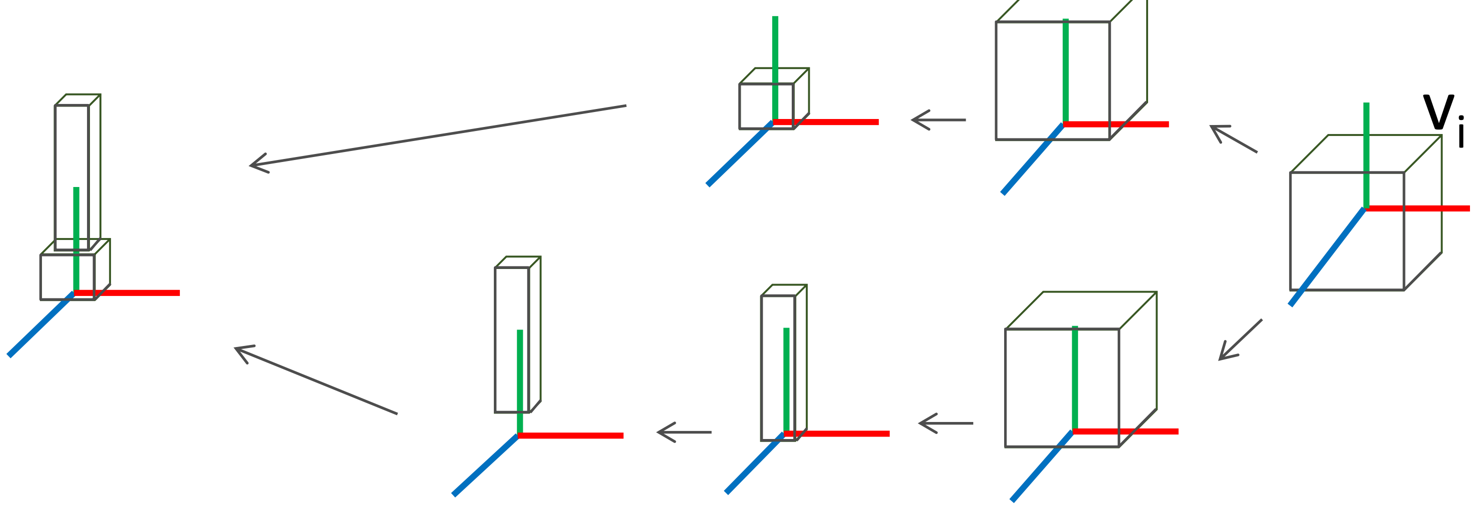
6.3 Transformations



Transform 1

Cuboid1: s(0.3,0.3,0.3) ; t(0, 0.5, 0) ; draw cube 1

Both cuboids



Assume unit cube, vertices v_i, defined with centre at origin

Cuboid2: t(0,0.3,0) ; s(0.2,1.9,0.2) ; t(0, 0.5, 0); draw cube 2

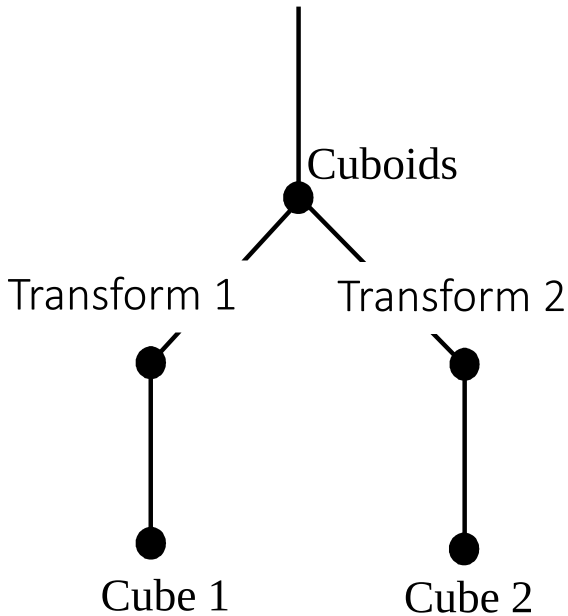
b a

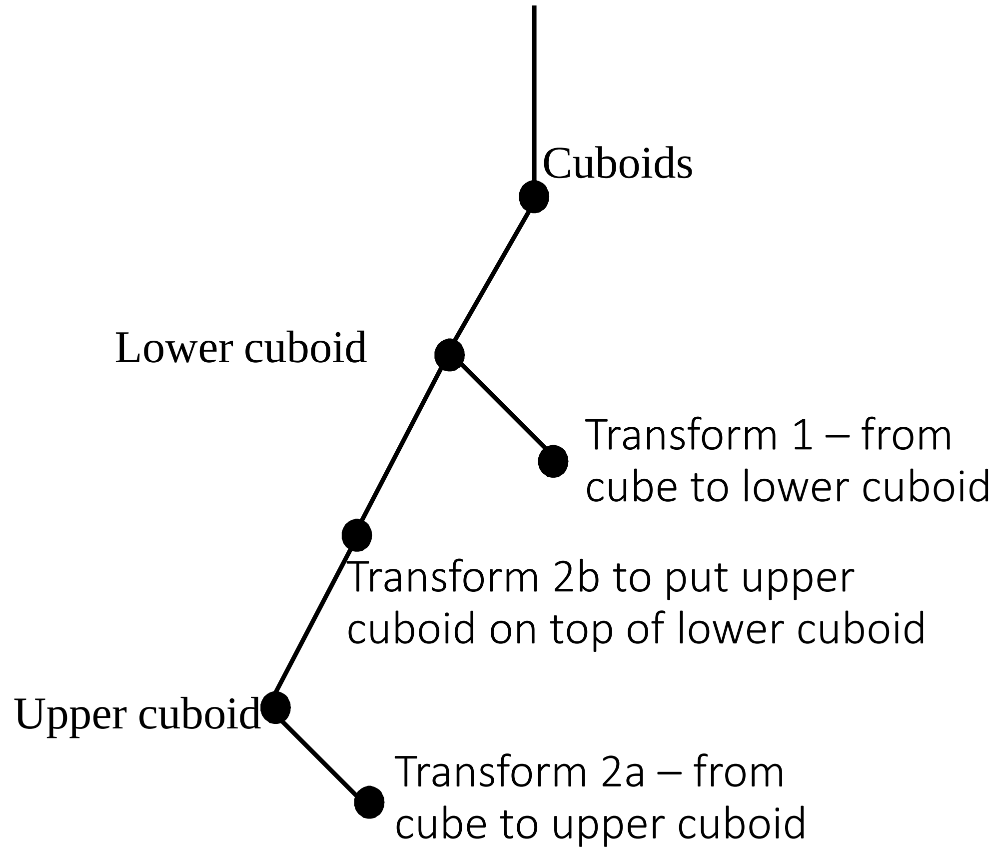
Transform 2

s = scale, r = rotate, t = translate

6.4 Completing the scene graph

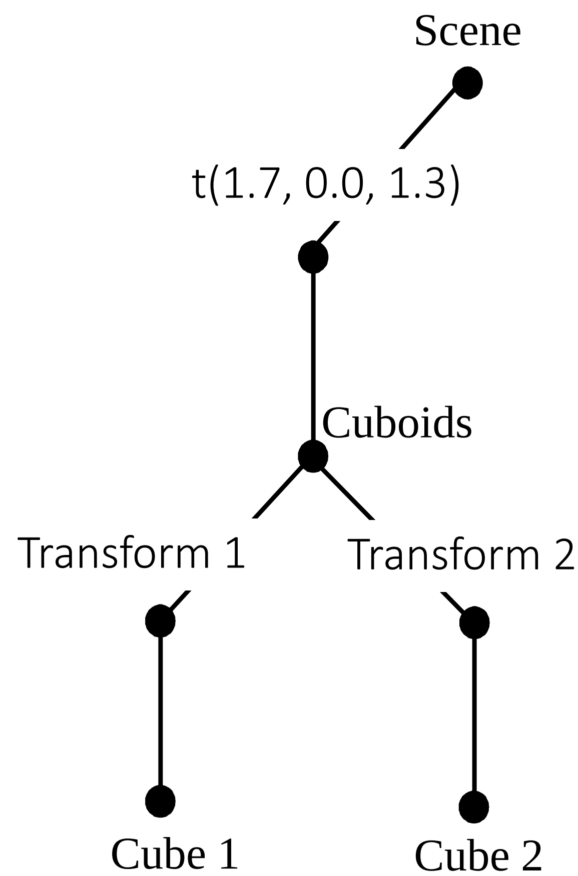
- Alternatives:





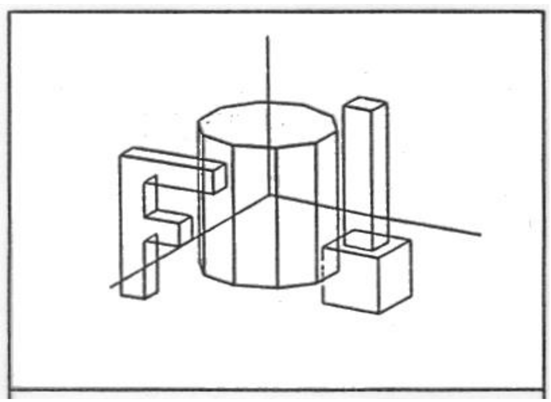
6.4 Compounding transformations

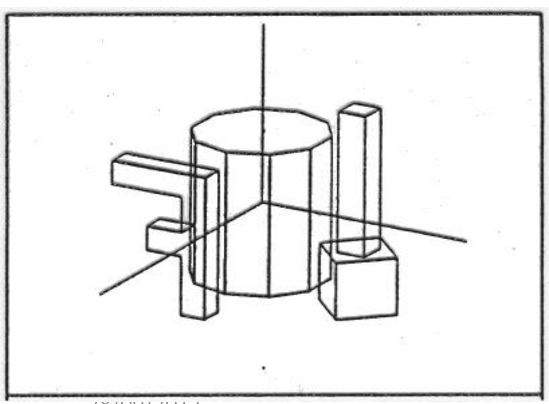
- As we descend the tree the transformations are multiplied together
- In this example:
  - cube 1 is transformed by both t(1.7,0.0,1.3) and by Transform 1
  - cube 2 is transformed by both t(1.7,0.0,1.3) and by Transform 2
- If an extra transform is included after Transform 1, it will only affect cube 1
- A transform above the scene node would affect everything in the scene

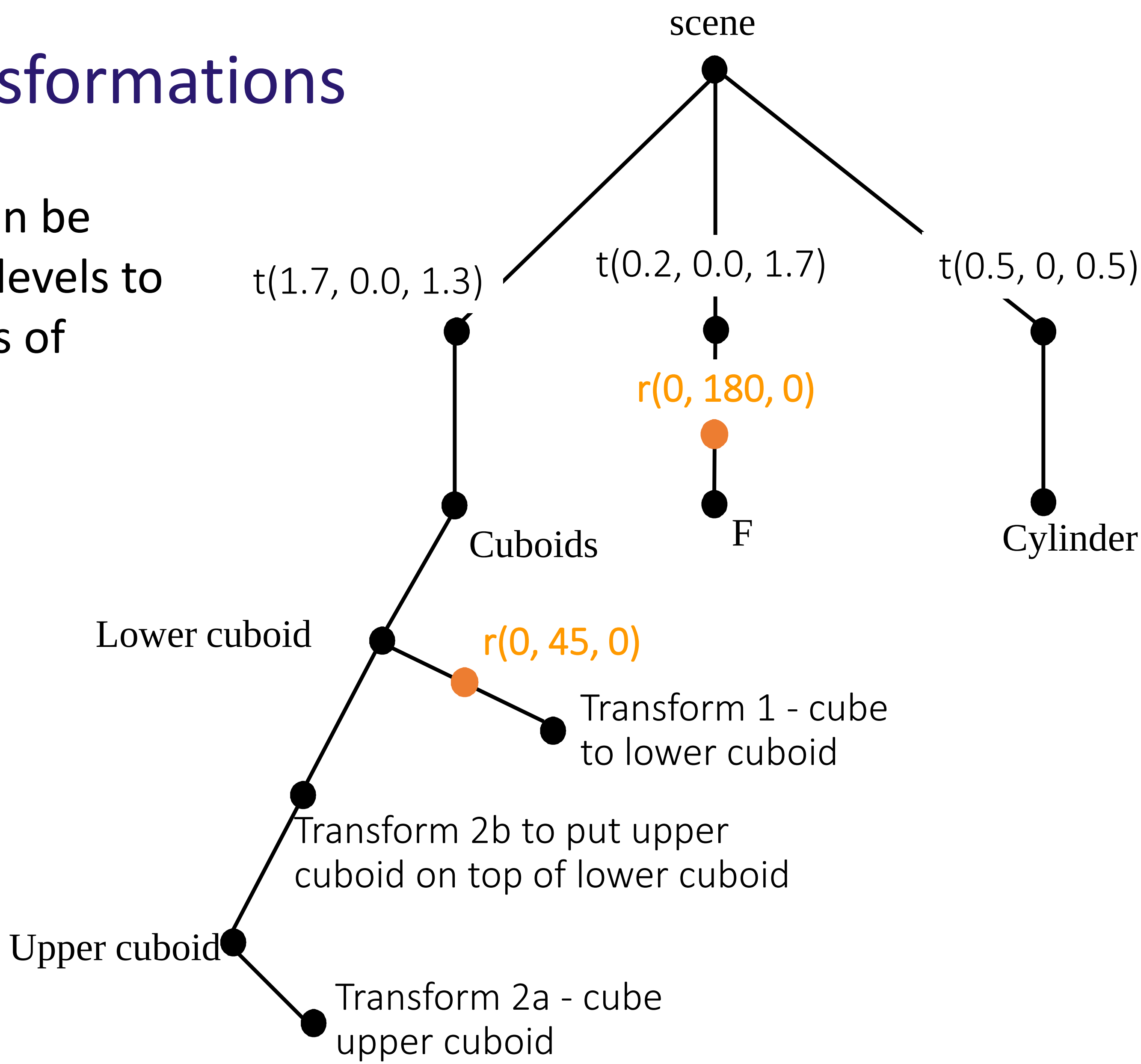


6.5 More transformations

- Transformations can be added at different levels to affect different sets of objects



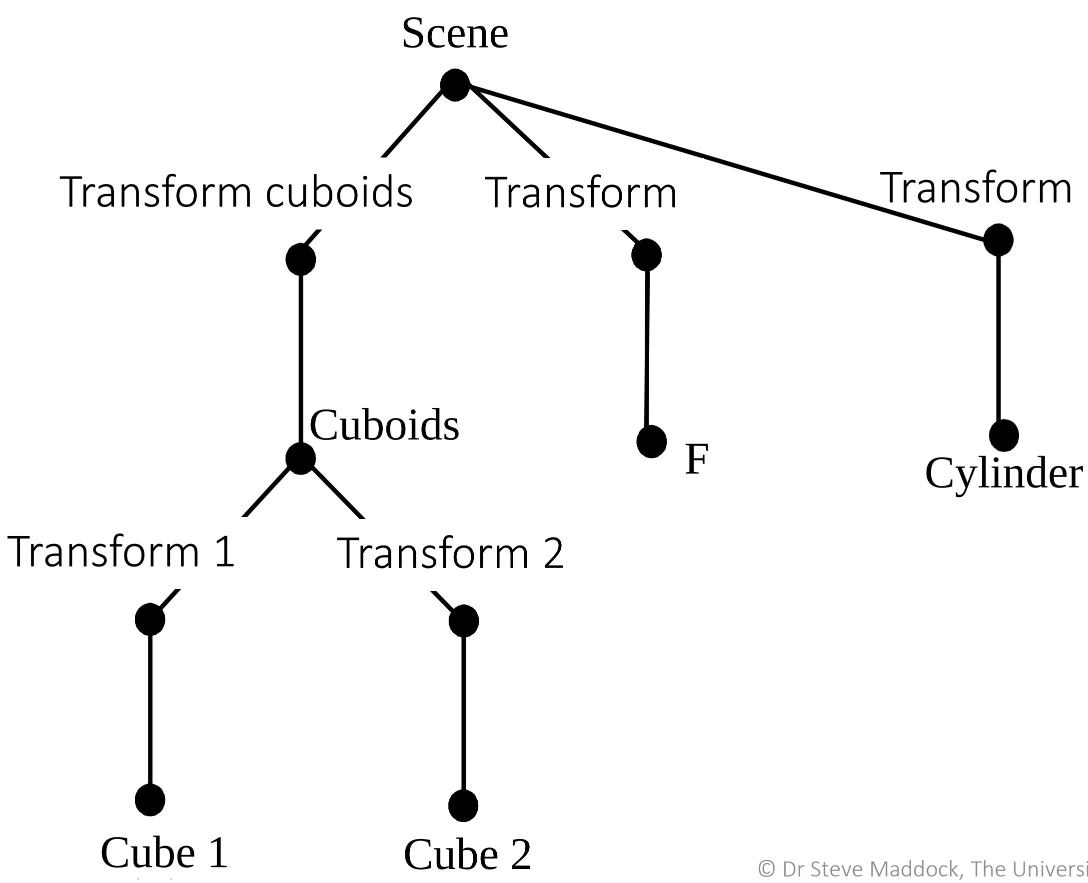




7. Old vs modern OpenGL

Fixed–function pipeline

- The scene graph was often hard-coded or hard-wired




```
glPushMatrix();   // draw scene
  glPushMatrix();
    transformCuboids;
    glPushMatrix();
       transformCube1();
       drawCube1();
    glPopMatrix();
    glPushMatrix();
      transformCube2();
      drawCube2();
    glPopMatrix();
  glPopMatrix();
  glPushMatrix();
    transformF();
    drawF();
  glPopMatrix();
  glPushMatrix();
    transformCylinder();
    drawCylinder();
  glPopMatrix();
glPopMatrix();
```

OLD WAY

7.1 Example 1

OLD WAY

- The following program snippets produce different results

A

```
glLoadIdentity();
glRotated(45, 1,0,0);
glTranslated(x,y,z);
drawObject1();
drawObject2();
```

CTM = R

CTM = RT

$RT(v_{i(o1)})$

$RT(v_{i(o2)})$

B

```
glLoadIdentity();
glRotated(45, 1,0,0);
glPushMatrix();
  glTranslated(x,y,z);
  drawObject1();
glPopMatrix();
drawObject2();
```

CTM = R

CTM = RT

$RT(v_{o1,i})$

CTM = R

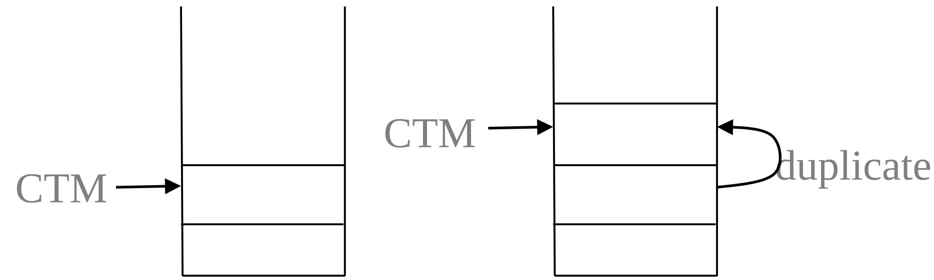
$R(v_{o2,i})$

- A: rotation and translation are applied to both objects
- B: rotation is applied to both objects, but translation is only applied to object 1

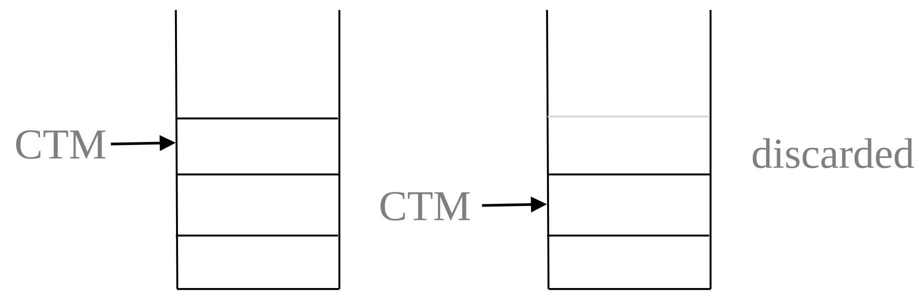
7.2 glPushMatrix() and glPopMatrix() OLD WAY

- CTM – current transformation matrix
  - Concatenation of all matrices from this point to bottom of stack

- void glPushMatrix()
  - ‘remember where you are’
  - Duplicates what is now the second-to-top matrix as the top matrix, which is known as the CTM



- void glPopMatrix()
  - ‘go back to where you were’
  - Pops the top matrix off the stack. Thus the second-to-top matrix becomes the top matrix, i.e. the CTM



7.3 Example 2 OLD WAY

- The following bracketed expressions show which transformations are applied to objects 1, 2, and 3, and in which order they are applied.

(R1(T1(S1(object1)))

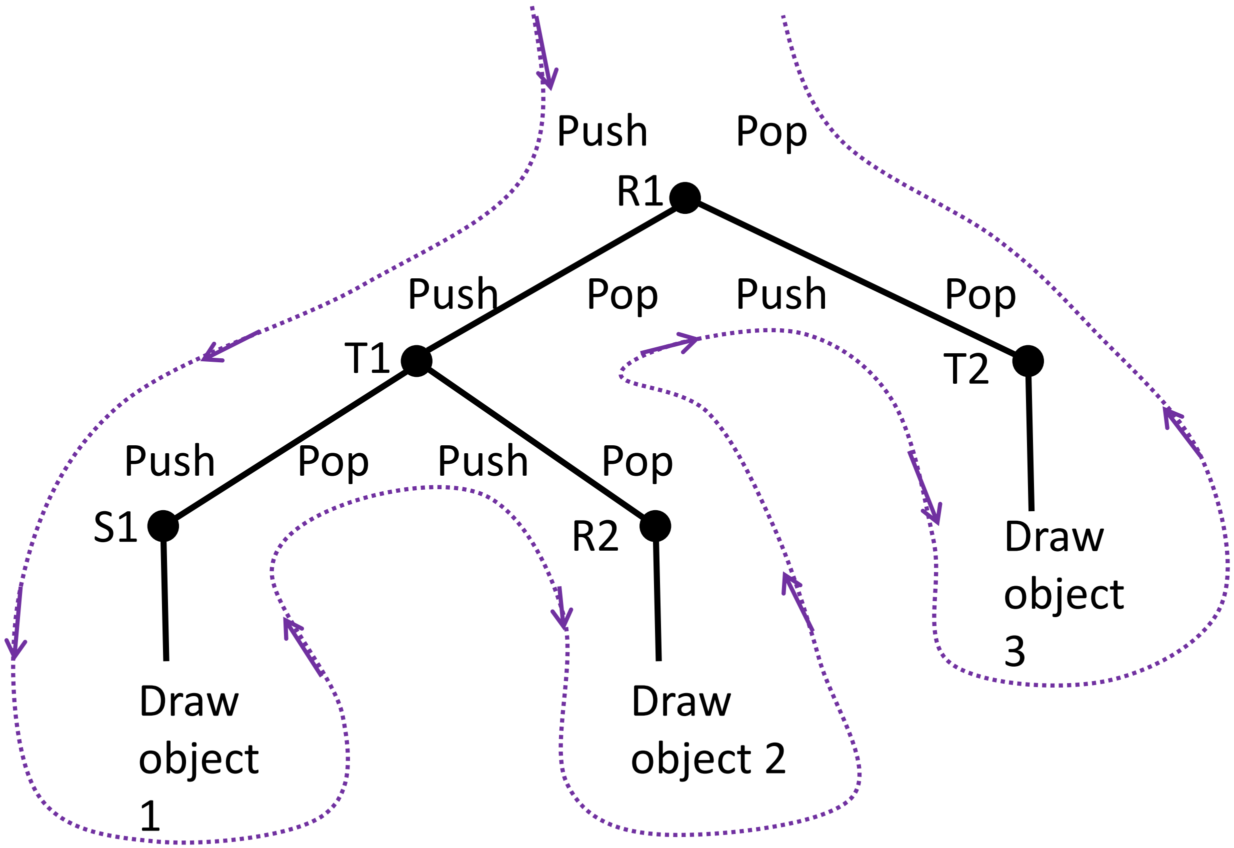
(R1(T1(R2(object2)))

(R1(T2(object3))

```
glLoadIdentity();
glPushMatrix();
  glRotated(45, 1,0,0);            R1
  glPushMatrix();
    glTranslated(x,y,z);           T1
    glPushMatrix();
      glScaled(x1,y1,z1);          S1
      drawObject1();
    glPopMatrix();
    glPushMatrix();
      glRotated(30, 0,1,0);        R2
      drawObject2();
    glPopMatrix();
  glPopMatrix();
  glPushMatrix();
    glTranslated(x,y,z);           T2
    drawObject3();
  glPopMatrix();
glPopMatrix();
```

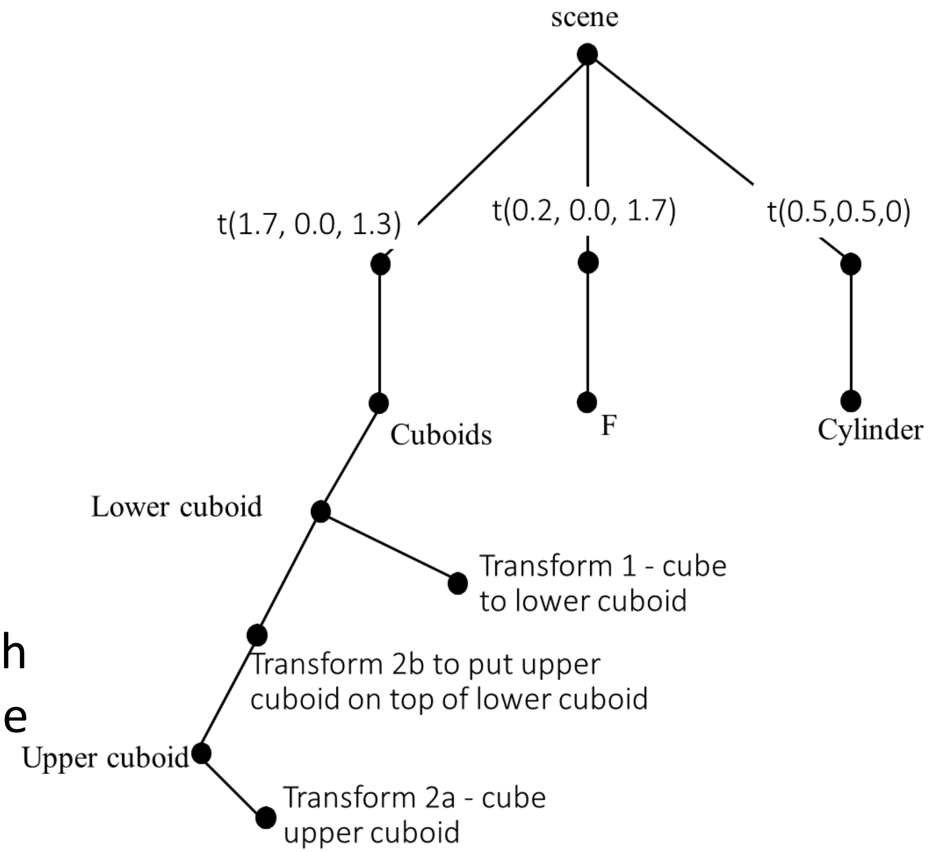
7.3 A Tree for Example 2

- A tree is a visual way to represent the collection of transformations applied in a push…pop hierarchy
  - (R1(T1(S1(object1)))
  - (R1(T1(R2(object2)))
  - (R1(T2(object3))
- This behaviour could be reproduced with methods for transformations and making use of a stack data structure



7.4 Using a scene graph API

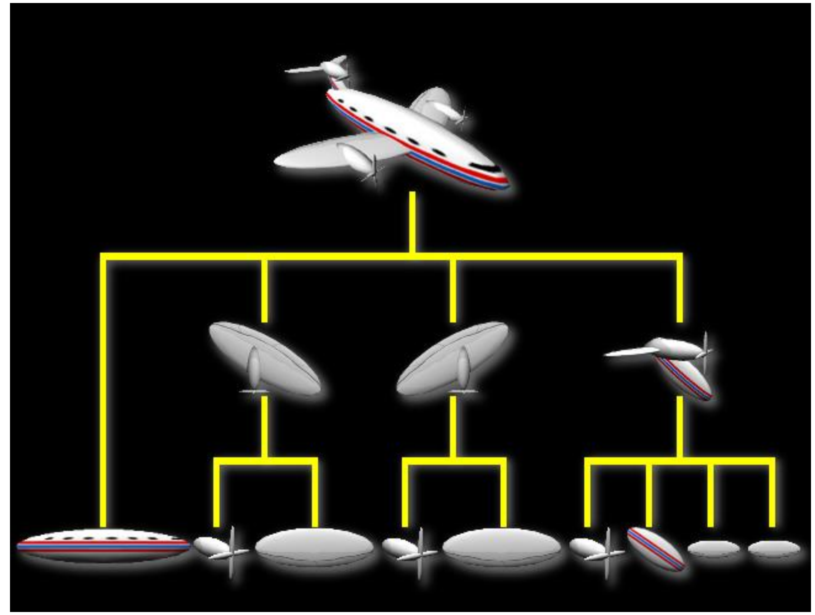
- Instead a scene graph API could be used, e.g. OpenSceneGraph
  - methods to add nodes to the scene graph – a (hidden) data structure
  - nodes can contain children, so a parent-child structure is established
- Traverse scene graph data structure to render the objects
- But: isn't adding nodes to the scene graph is similar to hard-wiring the graph into the code?
- However: the scene graph can be more easily changed whilst a program is running



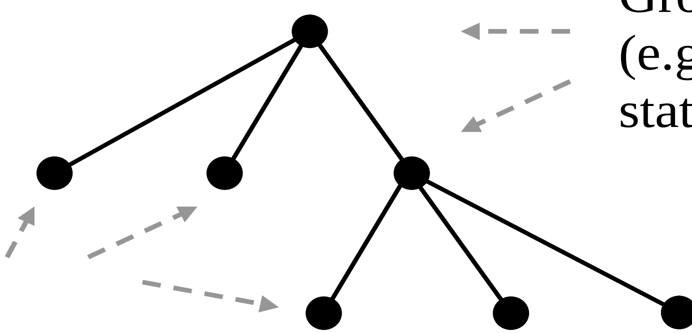
It is a much more flexible approach

8. More scene graph nodes

- Early scene graphs were essentially transform hierarchies
- Parent-child hierarchies – See Next Week's lecture
  - Example: building (walls, floors, windows, interior rooms (desks, chairs))
  - Example: horse (parent) and knight (child).



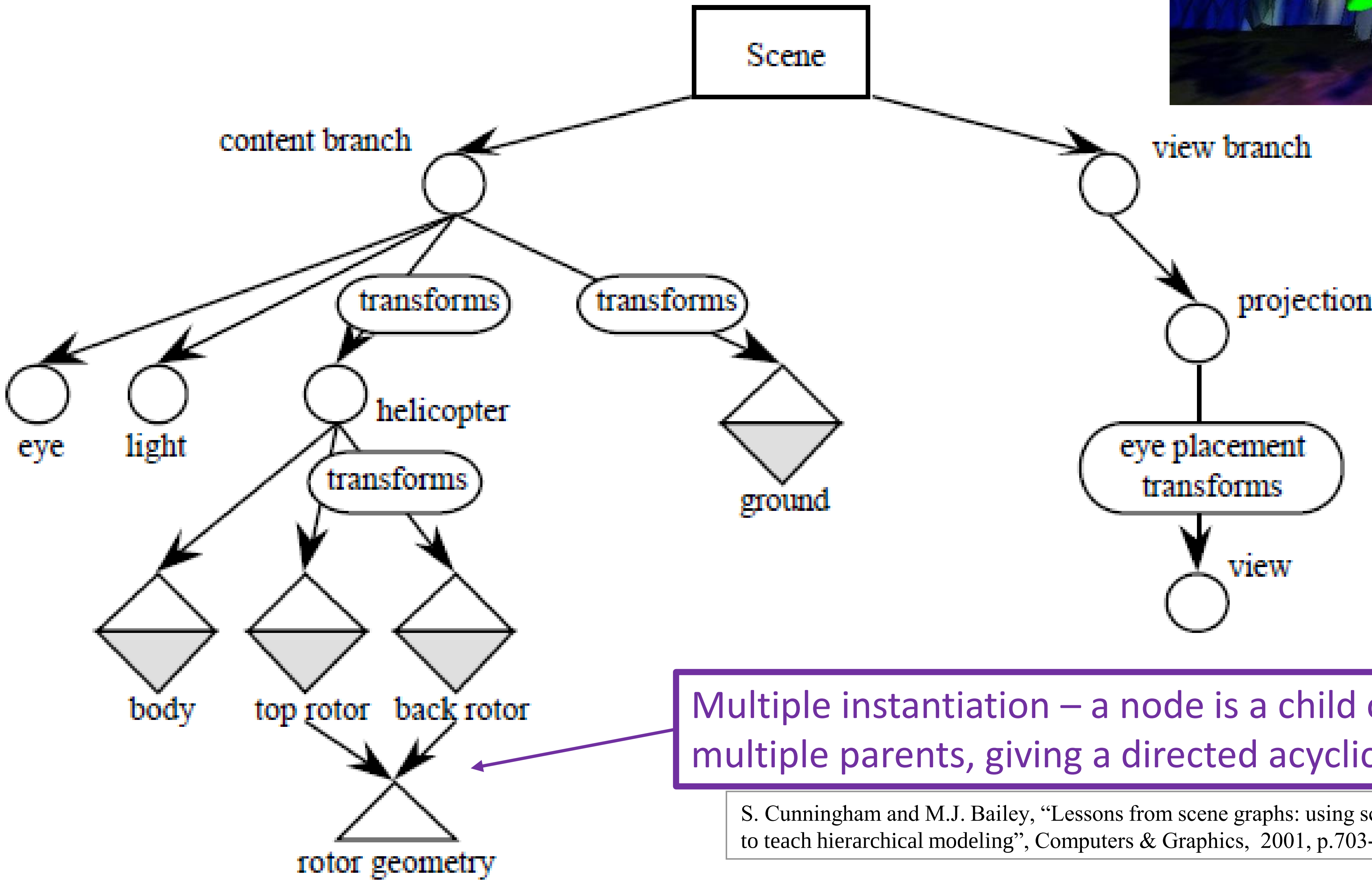
- Later, other kinds of nodes were added



Group nodes
(e.g. transformation, switch, state, level of detail,

Leaf nodes
(e.g. geometry, lights)

8.1 Example



Multiple instantiation – a node is a child of multiple parents, giving a directed acyclic graph

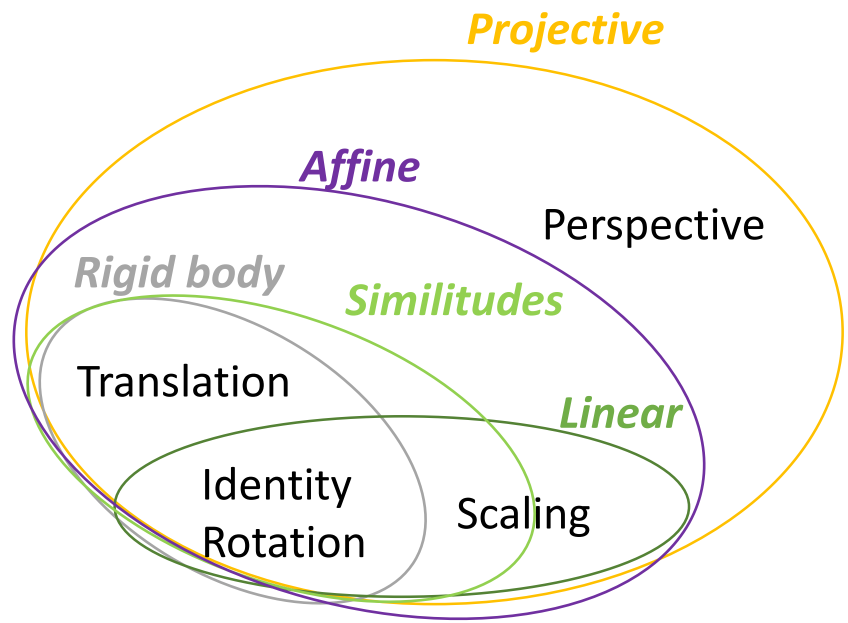
S. Cunningham and M.J. Bailey, "Lessons from scene graphs: using scene graphs to teach hierarchical modeling", Computers & Graphics, 2001, p.703-711

9. Summary

- Rotation and Translation are the ‘rigid-body transformations’
  - Do not change lengths or angles, so a body does not deform when transformed
- Standard rotation matrices rotate around relevant x, y or z axis
  - Rotation about arbitrary axis: Translate to origin, align axes, rotate, inverse align axes, and translate back again
- Use transformations to:
  - Manipulate individual objects, Build scenes, Build complex objects from pieces
  - *Coming soon:* Control relationships between parts in complex hierarchical objects
- A scene graph is used to represent a complex scene
  - Scene graphs have been extended to include other kinds of nodes besides transformations, e.g. switch nodes
  - In commercial systems, a scene graph API is used
  - Parallelism possibilities: multiple processes each traversing the scene graph

Appendix A. Transformation families

- **Rigid Body/Euclidean**
  - Preserve: lengths, angles
- **Similitudes/similarity**
  - (only isotropic/uniform scaling)
  - Preserve: angles, length ratios of a line
- **Linear**
  - (also includes reflection and shear)
  - Preserve: linear combination
- **Affine**
  - Preserve: parallel lines, length ratios of a line
- **Projective**
  - Preserve: lines – lines remain lines (planes remain planes in 3D)

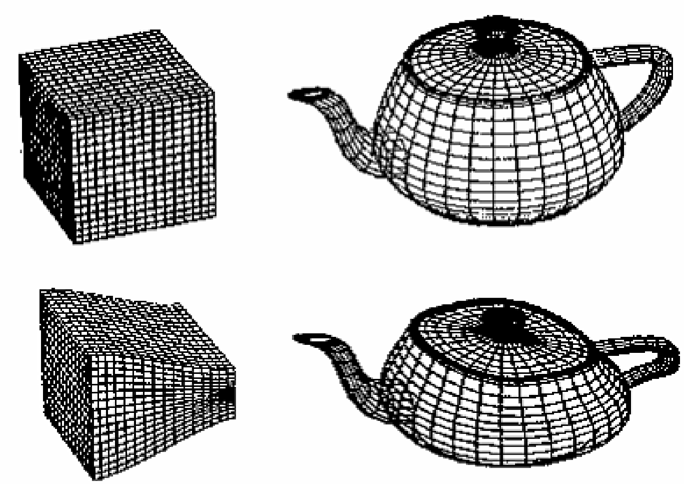


Why? Each family is closed under concatenation, so an affine transformation followed by an affine transformation is still an affine transformation

Slide based on diagrams from graphics courses by Durand, MIT and Jamin, Michigan

Appendix B: Advanced - Structure deforming transformations (Barr, 84)

- vertex V = (x,y,z);
- V' = (x',y',z') = (f(x), f(y), f(z))
- Vi' = f(Vi), for i = 1..n
- Choose a taper axis (e.g. z) and differentially scale one or two of the other two components (x and y).
- Example: global taper in y along the x axis:
  - xi' = xi
  - yi' = ryi
  - zi' = zi
- where
- r = f(xi) = (max(xi) – xi)/ (max(xi)
- Thus, as x increases, y decreases



Barr, A. H., Global and Local Deformations of Solid Primitives, Proceedings of SIGGRAPH '84, Computer Graphics 18, 3 (July 1984), 21-30

Twisting (differential rotation)

- Example: twist an object about its y axis:

$$x' = x \cos\theta + z \sin\theta$$

$$y' = y$$

$$z' = -x \sin\theta + z \cos\theta$$

where $\theta = f(y)$