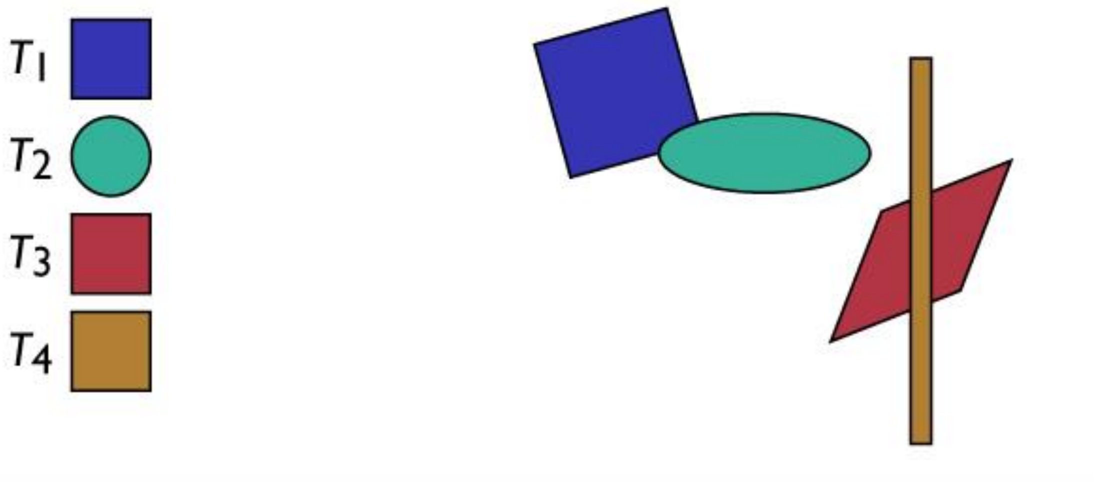
Scene Graphs

COMP557

Paul Kry

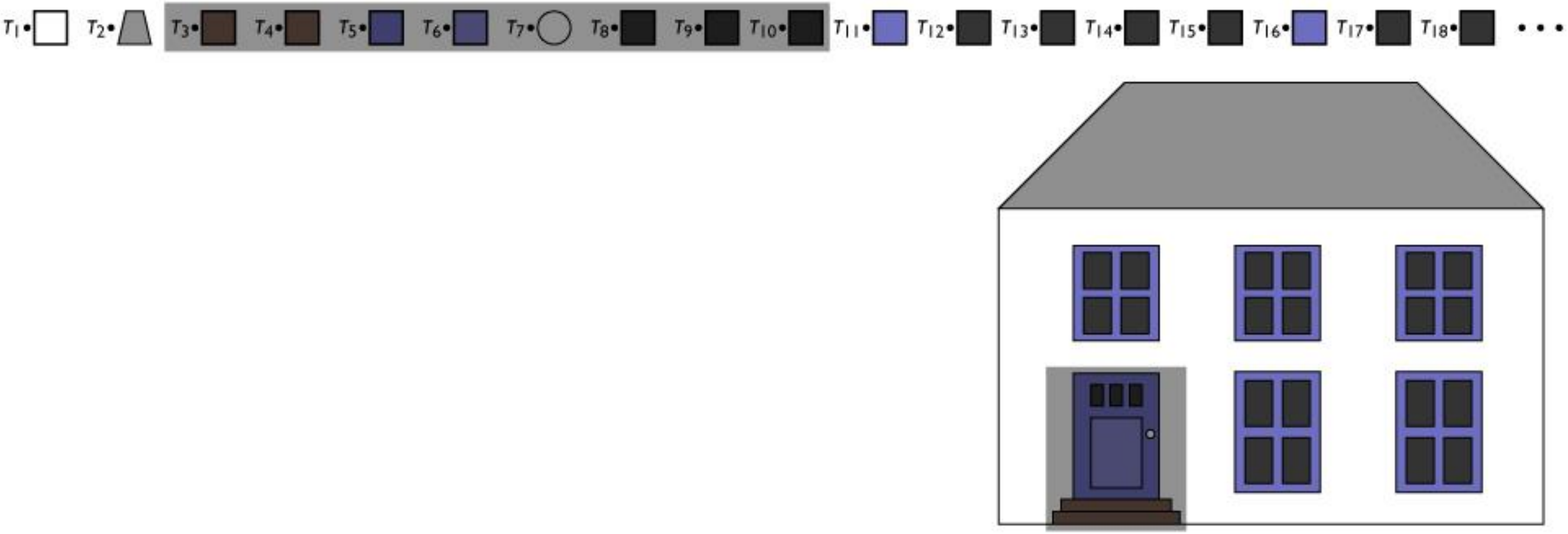
Data structures with transforms

- Representing a drawing ("scene")
- List of objects
- Transform for each object
  - can use minimal primitives: ellipse is transformed circle
  - transform applies to points of object



Example

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



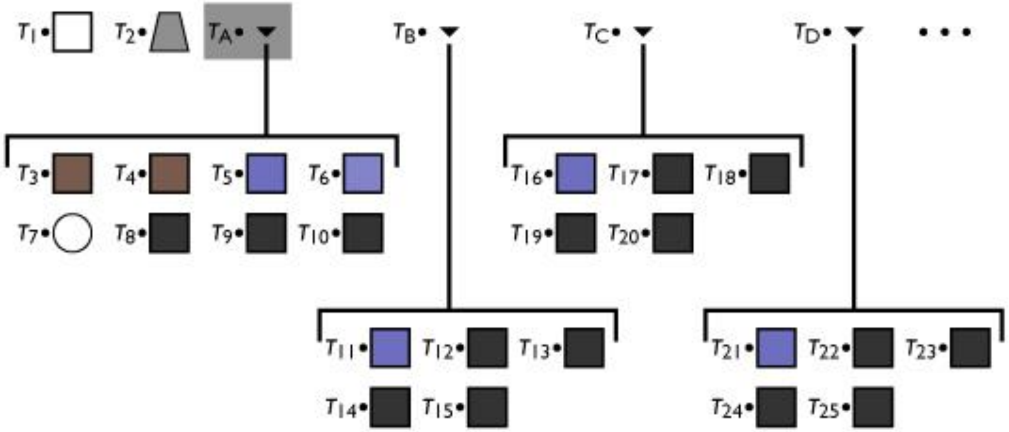
Groups of objects

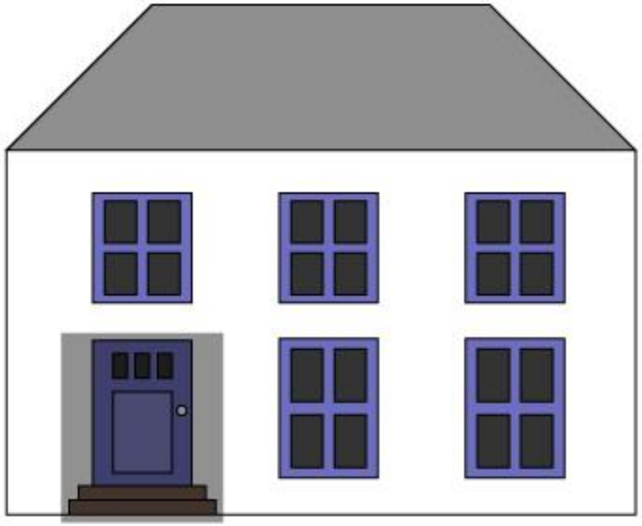
- Treat a set of objects as one
- Introduce new object type: group
  - contains list of references to member objects
- This makes the model into a tree
  - interior nodes = groups
  - leaf nodes = objects
  - edges = membership of object in group

Based on slides by Steve Marschner

Example

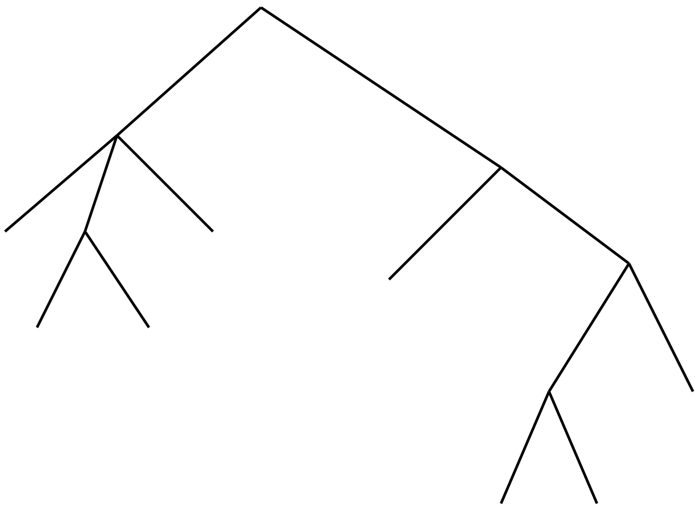
- Add group as a new object type
  - lets the data structure reflect the drawing structure
  - enables high-level editing by changing just one node





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
  - leaf nodes are identified with objects in the scene



Concatenation and hierarchy

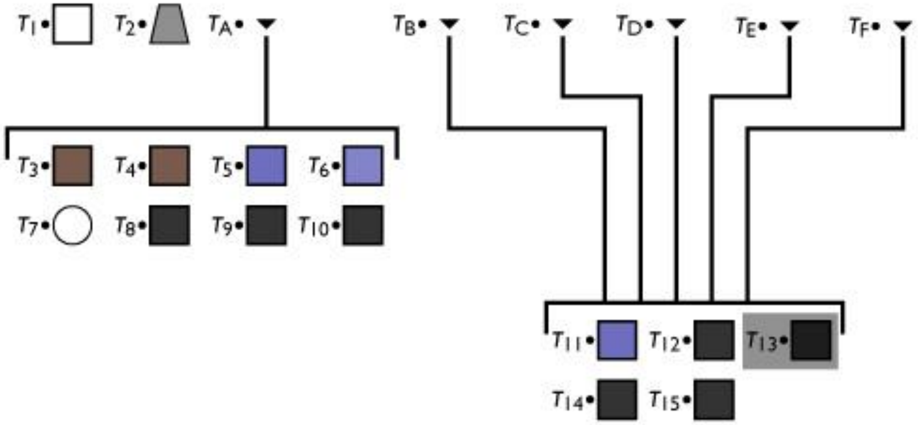
- Transforms associated with nodes or edges
- Each transform applies to all geometry below it
  - want group transform to transform each member
  - members already transformed—concatenate
- Frame transform for object is product of all matrices along path from root
  - each object's transform describes relationship between its local coordinates and its group's coordinates
  - frame-to-canonical transform is the result of repeatedly changing coordinates from group to containing group

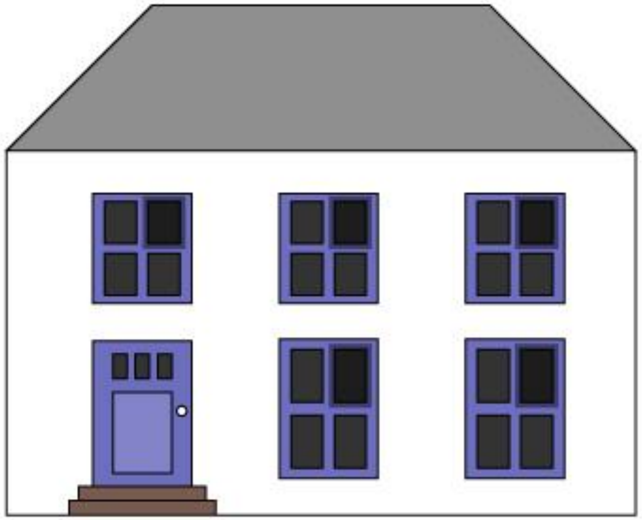
Instances

- Simple idea: allow an object to be a member of more than one group at once
  - transform different in each case
  - leads to linked copies
  - single editing operation changes all instances

Example

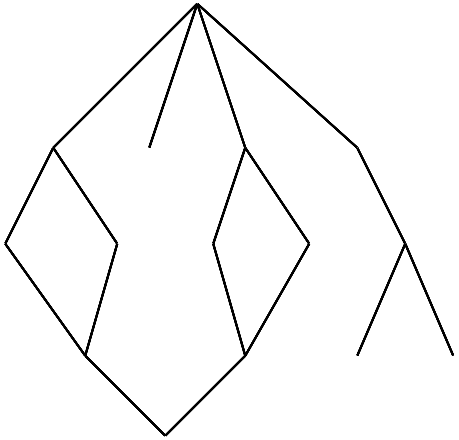
- Allow multiple references to nodes
  - reflects more of drawing structure
  - allows editing of repeated parts in one operation





The Scene Graph (with instances)

- With instances, there is no more tree
  - an object that is instanced multiple times has more than one parent
- Transform tree becomes DAG
  - **d**irected **a**cyclic **g**raph
  - group is not allowed to contain itself, even indirectly
- Transforms still accumulate along path from root
  - now *paths* from root to leaves are identified with scene objects



Implementing a hierarchy

- Object-oriented language is convenient
  - define shapes and groups as derived from single class

```
abstract class Shape {
    void draw();
}
```

```
class Square extends Shape {
    void draw() {
        // draw unit square
    }
}
```

```
class Circle extends Shape {
    void draw() {
        // draw unit circle
    }
}
```

Implementing traversal

- Pass a transform down the hierarchy
  - before drawing, concatenate

```
abstract class Shape {
    void draw(Transform t_c);
}
```

```
class Circle extends Shape {
    void draw(Transform t_c) {
        // draw t_c * unit circle
    }
}
```

```
class Square extends Shape {
    void draw(Transform t_c) {
        // draw t_c * unit square
    }
}
```

```
class Group extends Shape {
    Transform t;
    ShapeList members;
    void draw(Transform t_c) {
        for (m in members) {
            m.draw(t_c * t);
        }
    }
}
```

Implementing traversal

In OpenGL, transforms are kept on *matrix stack*, with push and pop matrix commands allowing us to easily store copies before we make modifications in different sub-trees.

```
abstract class DAGNode {
  List<DAGNode> children;
  void display( GLAutoDrawable d) {
    for ( DAGNode n : children ) {
      n.display(d);
    }
  }
}
```

```
class Cube extends DAGNode {
  void display( GLAutoDrawable d ) {
    glut.glutSolidCube(1);
    super.display(d);
  }
}
```

```
class RotateX extends DAGNode {
  double theta;
  void display( GLAutoDrawable d ) {
    GL2 gl = drawable.getGL().getGL2();
    gl.glPushMatrix();
      gl.glRotate( theta, 1, 0, 0 );
      super.display(d);
    gl.glPopMatrix();
  }
}
```

Basic Scene Graph operations

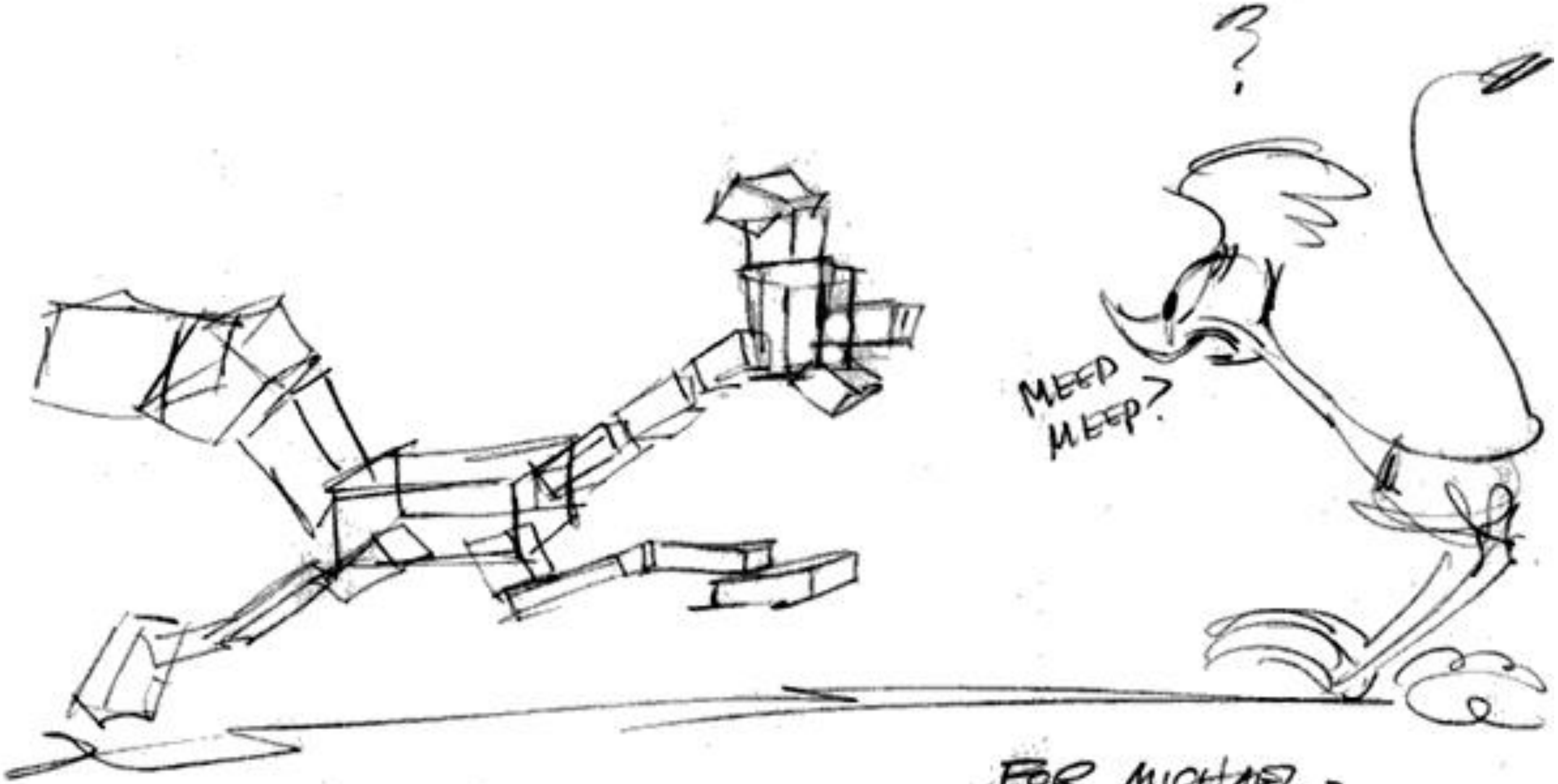
- Editing a transformation
  - good to present usable UI
- Getting transform of object in canonical (world) frame
  - traverse path from root to leaf
- Grouping and ungrouping
  - can do these operations without moving anything
  - group: insert identity node
  - ungroup: remove node, push transform to children
- Reparenting
  - move node from one parent to another
  - can do without altering position

Adding more than geometry

- Objects have properties besides shape
  - color, shading parameters
  - approximation parameters (e.g. precision of subdividing curved surfaces into triangles)
  - behavior in response to user input
  - …
- Setting properties for entire groups is useful
  - paint entire window green
- Many systems include some kind of property nodes
  - in traversal they are read as, e.g., “set current color”

Scene Graph variations

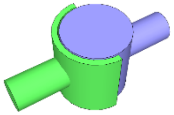
- Where transforms go
  - in every node?
  - on edges?
  - in group nodes only?
  - in special Transform nodes?
- Tree vs DAG
- Nodes for cameras and lights?

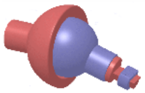


FOR MICHAEL -
WITH ADMIRATION
Chuck Jones
1985

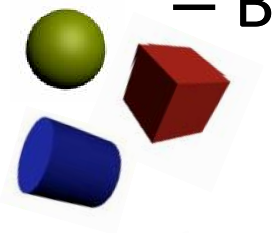
Characters Design, Posing, and Key Frame Animation

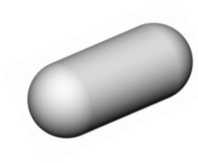
- Joints and bones, it is useful to separate joint transforms from geometry (and geometry transforms)
  - Joint types:





    - Hinge / Rotary, a given axis about a given point
    - Ball / Spherical, XYZ Euler, about a given point
    - Free, e.g., both translation and XYZ Euler angles
  - Bones / geometry:





    - Cube, Sphere, Cylinder, Cone, etc… (see glut methods)
    - Translated and scaled with respect to parent node!
    - Combine shapes (e.g., capsules), different colours, etc.!

Characters Design, Posing, and Key Frame Animation

- What should be the root?
  - A foot? Hips? Torso? Head?
- Want to expose parameters necessary to pose
  - Typically only want to edit joint angles, while geometry may remain fixed (e.g., bone lengths)
- Want meaningful poses produced by interpolation of parameters (gimbal lock?)
- Many options for organizing DAG and classes!

Discuss!

Review and more information

- Textbook
  - 12.2 - Scene graphs
- Not in textbook
  - Directed acyclic graphs and instances