2D/3D Geometric Transformations and Scene Graphs

Week 4

Acknowledgement: The course slides are adapted from the slides prepared by Steve Marschner of Cornell University

A little quick math background

- Notation for sets, functions, mappings
- Linear transformations
- Matrices
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
- Geometry of curves in 2D
  - Implicit representation
  - Explicit representation

Implicit representations

- Equation to tell whether we are on the curve

$$\{\mathbf{v} \mid f(\mathbf{v}) = 0\}$$

- Example: line (orthogonal to **u**, distance *k* from **0**)

$$\{\mathbf{v} \mid \mathbf{v} \cdot \mathbf{u} + k = 0\}$$

- Example: circle (center **p**, radius *r*)

$$\{\mathbf{v} \mid (\mathbf{v} - \mathbf{p}) \cdot (\mathbf{v} - \mathbf{p}) + r^2 = 0\}$$

- Always define boundary of region
  - (if *f* is continuous)

Explicit representations

- Also called parametric
- Equation to map domain into plane

$$\{f(t) \,|\, t \in D\}$$

- Example: line (containing **p**, parallel to **u**)

$$\{\mathbf{p} + t\mathbf{u} \,|\, t \in \mathbb{R}\}$$

- Example: circle (center **b**, radius *r*)

$$\{\mathbf{p} + r[\cos t \;\; \sin t]^T \,|\, t \in [0, 2\pi)\}$$

- Like tracing out the path of a particle over time
- Variable *t* is the "parameter"

Transforming geometry

- Move a subset of the plane using a mapping from the plane to itself

$$S \to \{T(\mathbf{v}) \mid \mathbf{v} \in S\}$$

- Parametric representation:

$$\{f(t) \mid t \in D\} \to \{T(f(t)) \mid t \in D\}$$

- Implicit representation:

$$\{\mathbf{v} \mid f(\mathbf{v}) = 0\} \to \{T(\mathbf{v}) \mid f(\mathbf{v}) = 0\}$$
$$= \{\mathbf{v} \mid f(T^{-1}(\mathbf{v})) = 0\}$$

Translation

- Simplest transformation: $T(\mathbf{v}) = \mathbf{v} + \mathbf{u}$
- Inverse: $T^{-1}(\mathbf{v}) = \mathbf{v} - \mathbf{u}$
- Example of transforming circle

Linear transformations

- One way to define a transformation is by matrix multiplication:

$$T(\mathbf{v}) = M\mathbf{v}$$

- Such transformations are *linear*, which is to say:

$$T(a\mathbf{u} + \mathbf{v}) = aT(\mathbf{u}) + T(\mathbf{v})$$

(and in fact all linear transformations can be written this way)

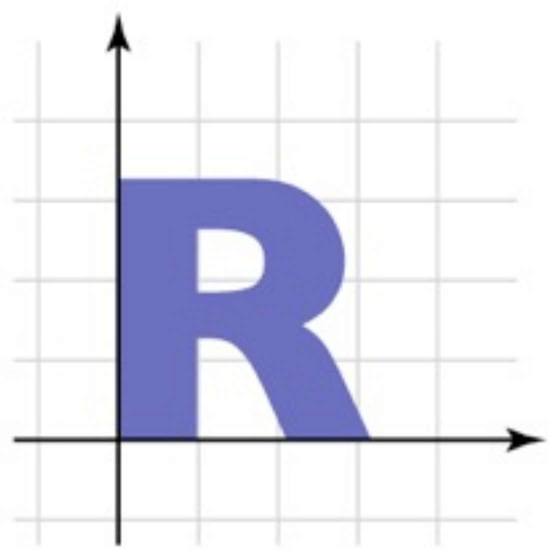
Geometry of 2D linear trans.

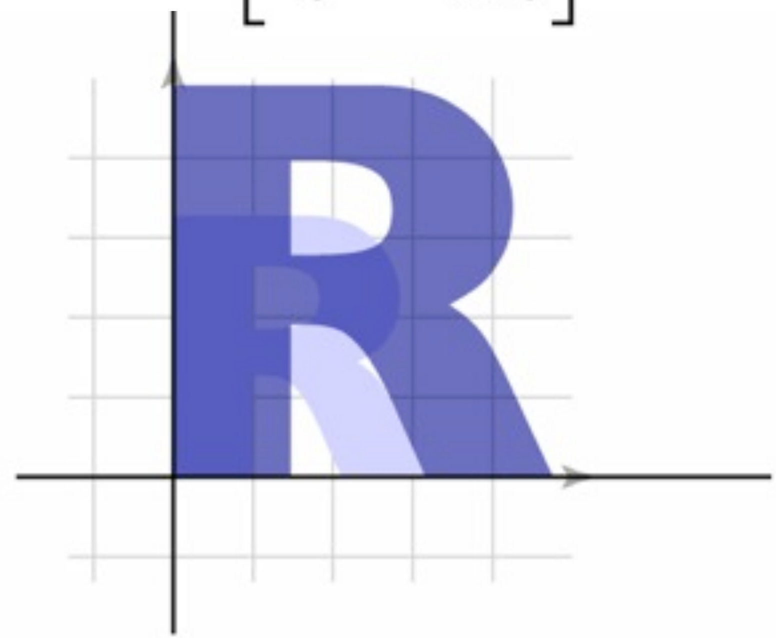
- 2x2 matrices have simple geometric interpretations
  - uniform scale
  - non-uniform scale
  - rotation
  - shear
  - reflection
- Reading off the matrix

Linear transformation gallery

- Uniform scale $\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} sx \\ sy \end{bmatrix}$

$$\begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix}$$

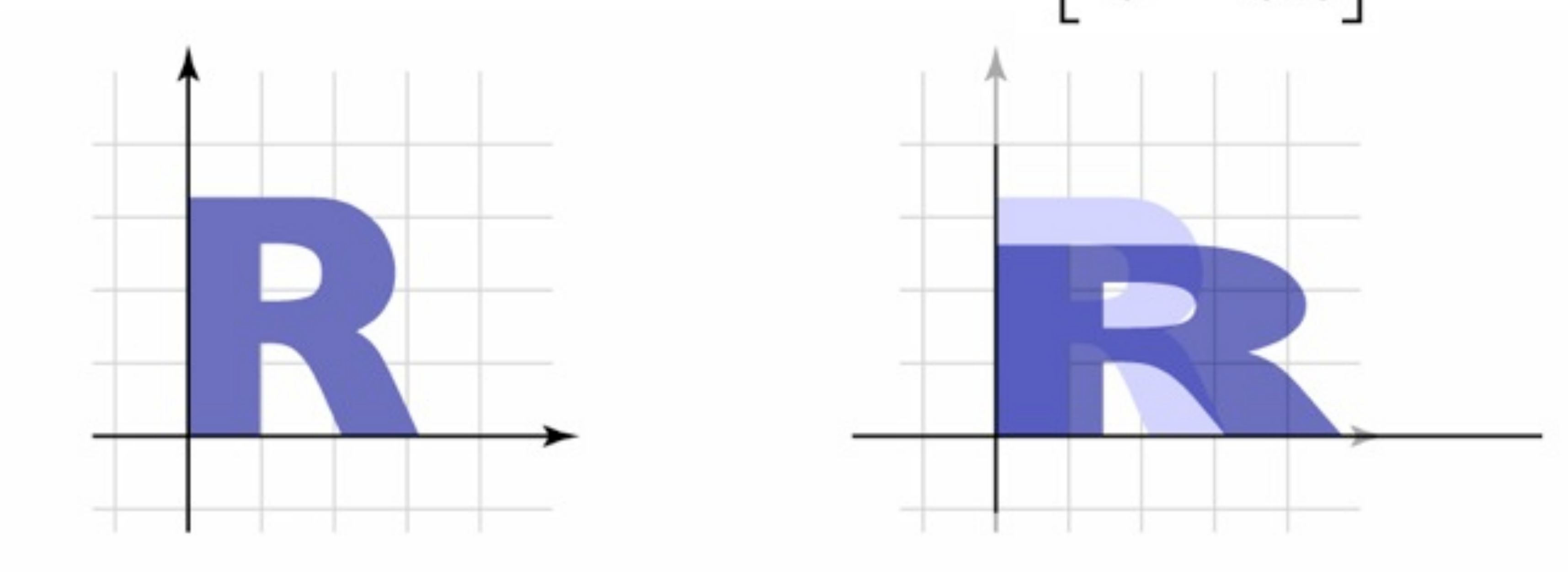




Linear transformation gallery

- Nonuniform scale $$\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$$

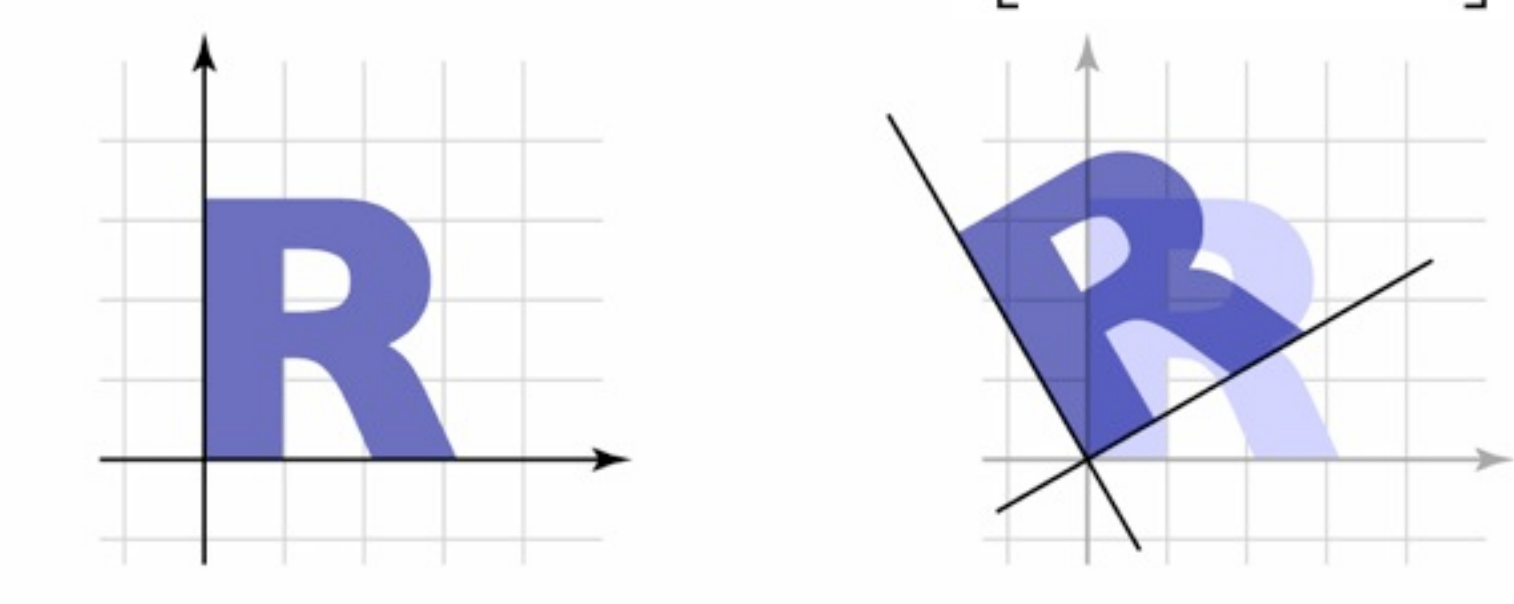
$$\begin{bmatrix} 1.5 & 0 \\ 0 & 0.8 \end{bmatrix}$$



Linear transformation gallery

- Rotation $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$

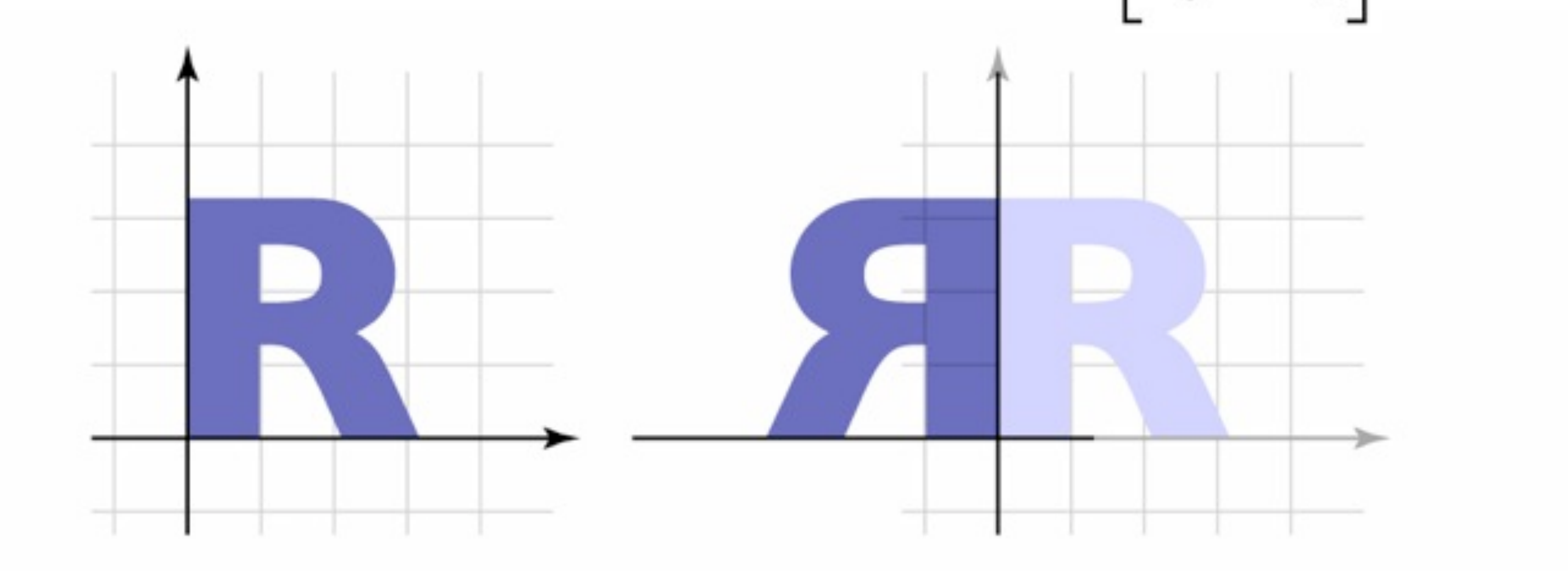
$$\begin{bmatrix} 0.866 & -.05 \\ 0.5 & 0.866 \end{bmatrix}$$



Linear transformation gallery

- Reflection
  - can consider it a special case of nonuniform scale

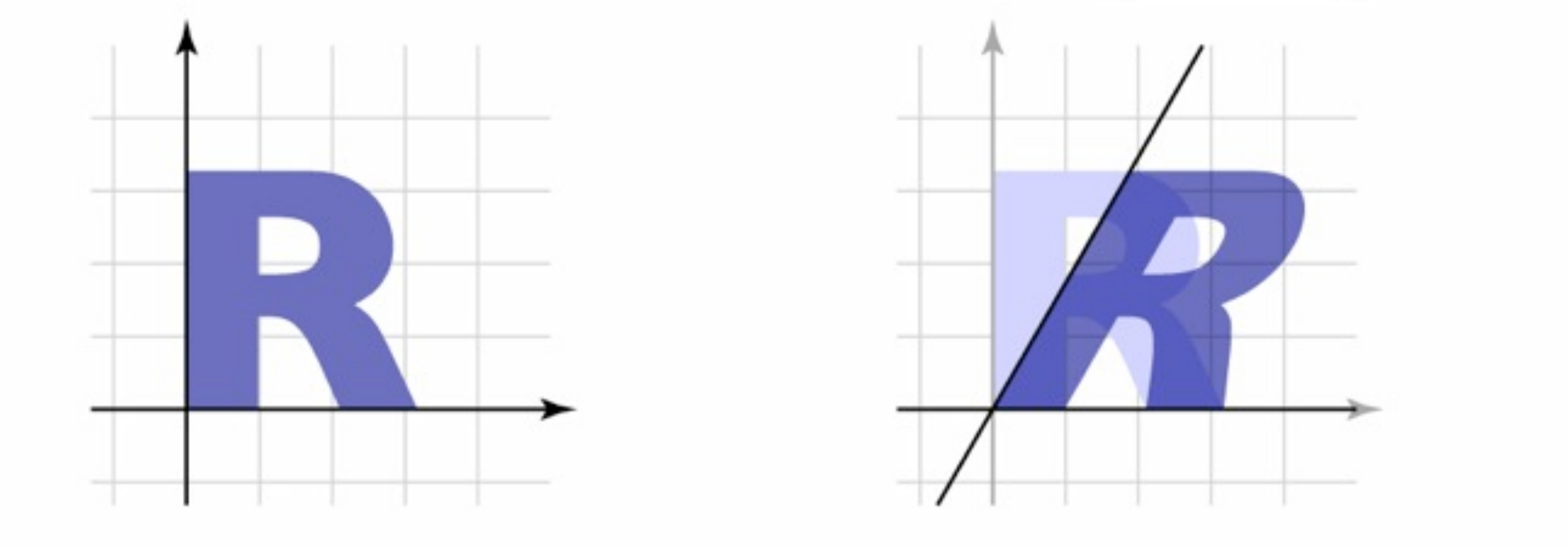
$$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



Linear transformation gallery

- Shear $\begin{bmatrix} 1 & a \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x + ay \\ y \end{bmatrix}$

$$\begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}$$



Composing transformations

- Want to move an object, then move it some more
  - $\mathbf{p} \to T(\mathbf{p}) \to S(T(\mathbf{p})) = (S \circ T)(\mathbf{p})$
- We need to represent *S* o *T* ("S compose T")
  - and would like to use the same representation as for *S* and *T*
- Translation easy
  - $T(\mathbf{p}) = \mathbf{p} + \mathbf{u}_T; S(\mathbf{p}) = \mathbf{p} + \mathbf{u}_S$

$$(S \circ T)(\mathbf{p}) = \mathbf{p} + (\mathbf{u}_T + \mathbf{u}_S)$$

- Translation by $\mathbf{u}_T$ then by $\mathbf{u}_S$ is translation by $\mathbf{u}_T + \mathbf{u}_S$
  - commutative!

Composing transformations

- Linear transformations also straightforward
  - $$T(\mathbf{p}) = M_T\mathbf{p}; S(\mathbf{p}) = M_S\mathbf{p}$$
    $$(S \circ T)(\mathbf{p}) = M_S M_T \mathbf{p}$$
- Transforming first by M_T then by M_S is the same as transforming by $M_S M_T$
  - only sometimes commutative
    - e.g. rotations & uniform scales
    - e.g. non-uniform scales w/o rotation
  - Note $M_S M_T$, or S o T, is T first, then S

Combining linear with translation

- Need to use both in single framework
- Can represent arbitrary seq. as $T(\mathbf{p}) = M\mathbf{p} + \mathbf{u}$
  - $T(\mathbf{p}) = M_T\mathbf{p} + \mathbf{u}_T$
  - $S(\mathbf{p}) = M_S\mathbf{p} + \mathbf{u}_S$
  - $$\begin{aligned}(S \circ T)(\mathbf{p}) &= M_S(M_T\mathbf{p} + \mathbf{u}_T) + \mathbf{u}_S \\ &= (M_S M_T)\mathbf{p} + (M_S\mathbf{u}_T + \mathbf{u}_S)\end{aligned}$$
  - e. g. $S(T(0)) = S(\mathbf{u}_T)$
- Transforming by M_T and $\mathbf{u}_T$, then by M_S and $\mathbf{u}_S$, is the same as transforming by $M_S M_T$ and $\mathbf{u}_S + M_S\mathbf{u}_T$
  - This will work but is a little awkward

Homogeneous coordinates

- A trick for representing the foregoing more elegantly
- Extra component w for vectors, extra row/column for matrices
  - for affine, can always keep $w = 1$
- Represent linear transformations with dummy extra row and column

$$\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \\ 1 \end{bmatrix}$$

Homogeneous coordinates

- Represent translation using the extra column

$$\begin{bmatrix} 1 & 0 & t \\ 0 & 1 & s \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t \\ y + s \\ 1 \end{bmatrix}$$

Homogeneous coordinates

- Composition just works, by 3x3 matrix multiplication

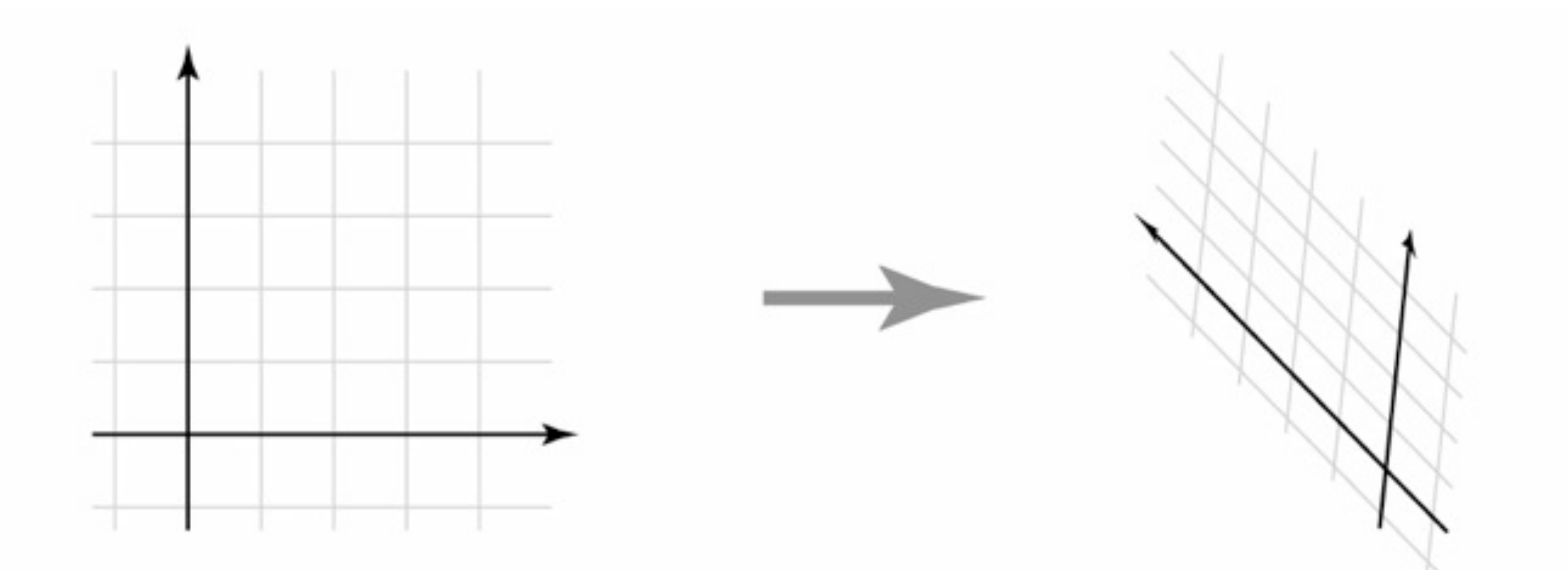
$$\begin{bmatrix} M_S & \mathbf{u}_S \\ 0 & 1 \end{bmatrix} \begin{bmatrix} M_T & \mathbf{u}_T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} (M_S M_T)\mathbf{p} + (M_S \mathbf{u}_T + \mathbf{u}_S) \\ 1 \end{bmatrix}$$

- This is exactly the same as carrying around *M* and **u**
  – but cleaner
  – and generalizes in useful ways as we'll see later

Affine transformations

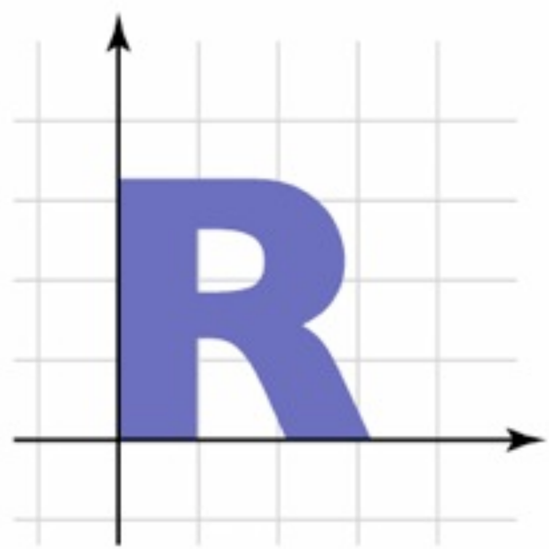
- The set of transformations we have been looking at is known as the "affine" transformations
  - straight lines preserved; parallel lines preserved
  - ratios of lengths along lines preserved (midpoints preserved)

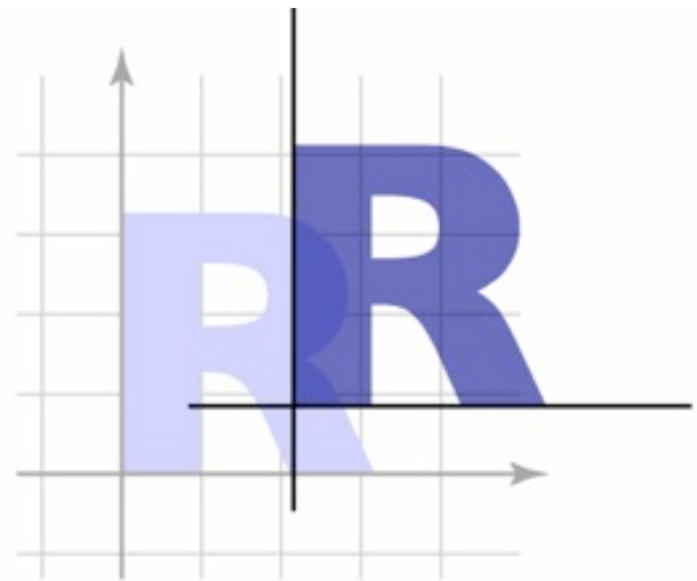


Affine transformation gallery

- Translation

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0 & 2.15 \\ 0 & 1 & 0.85 \\ 0 & 0 & 1 \end{bmatrix}$$

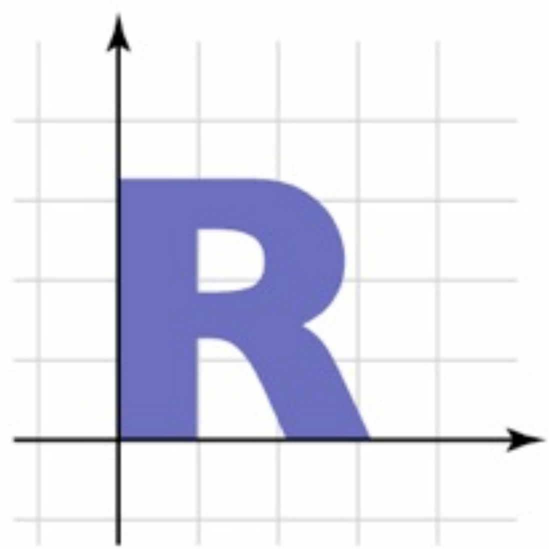


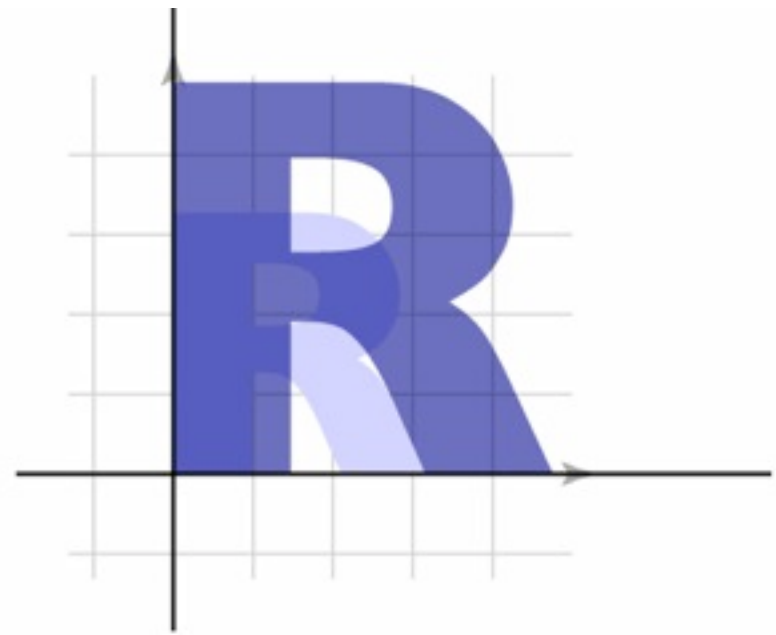


Affine transformation gallery

- Uniform scale

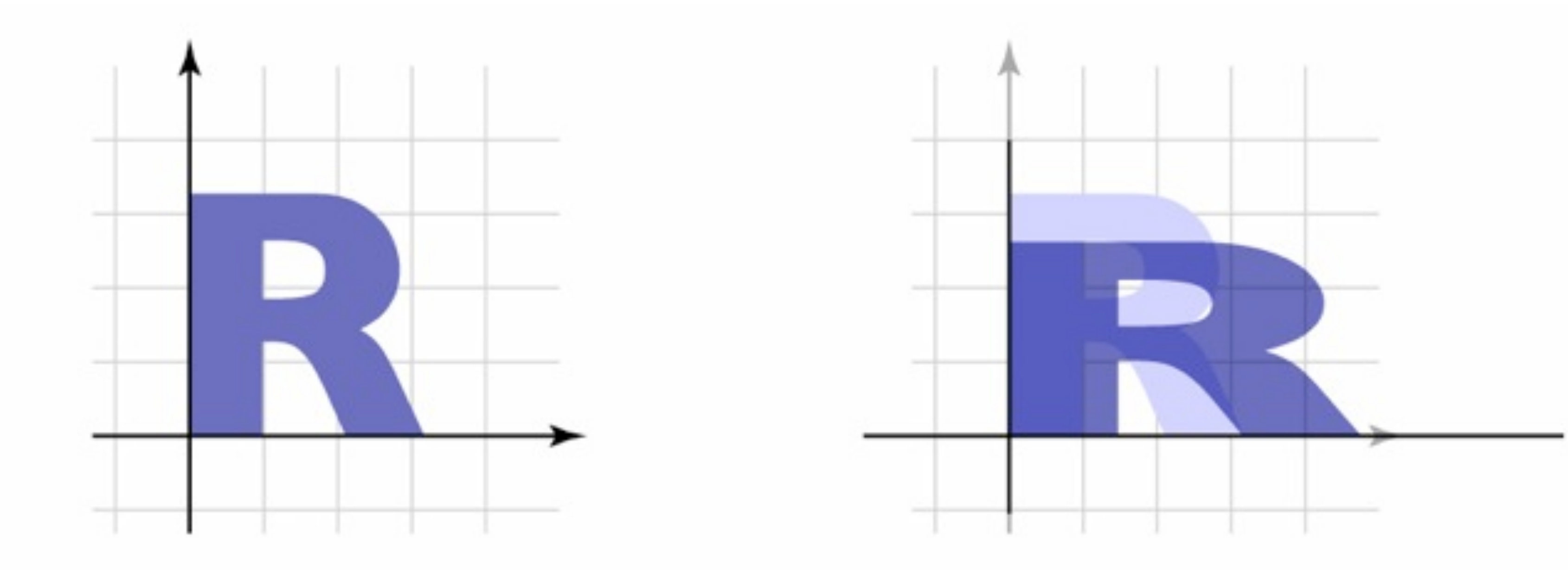
$$\begin{bmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$





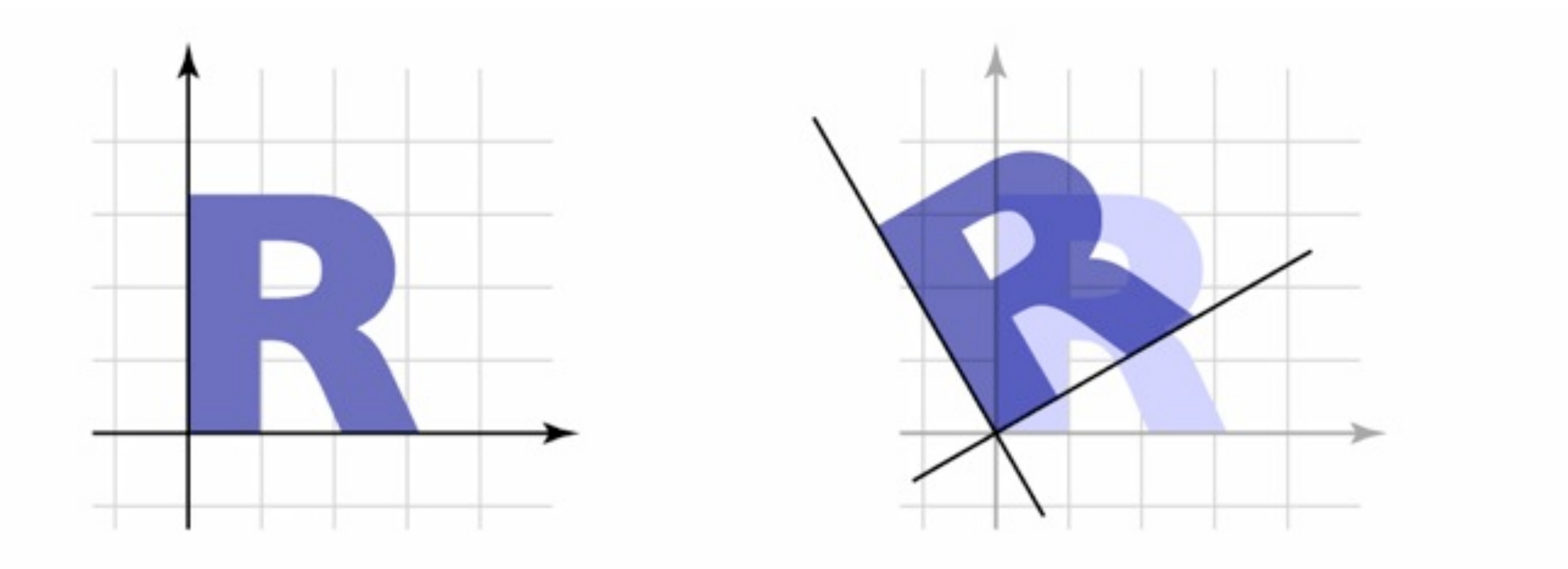
Affine transformation gallery

- Nonuniform scale $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$ $\begin{bmatrix} 1.5 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



Affine transformation gallery

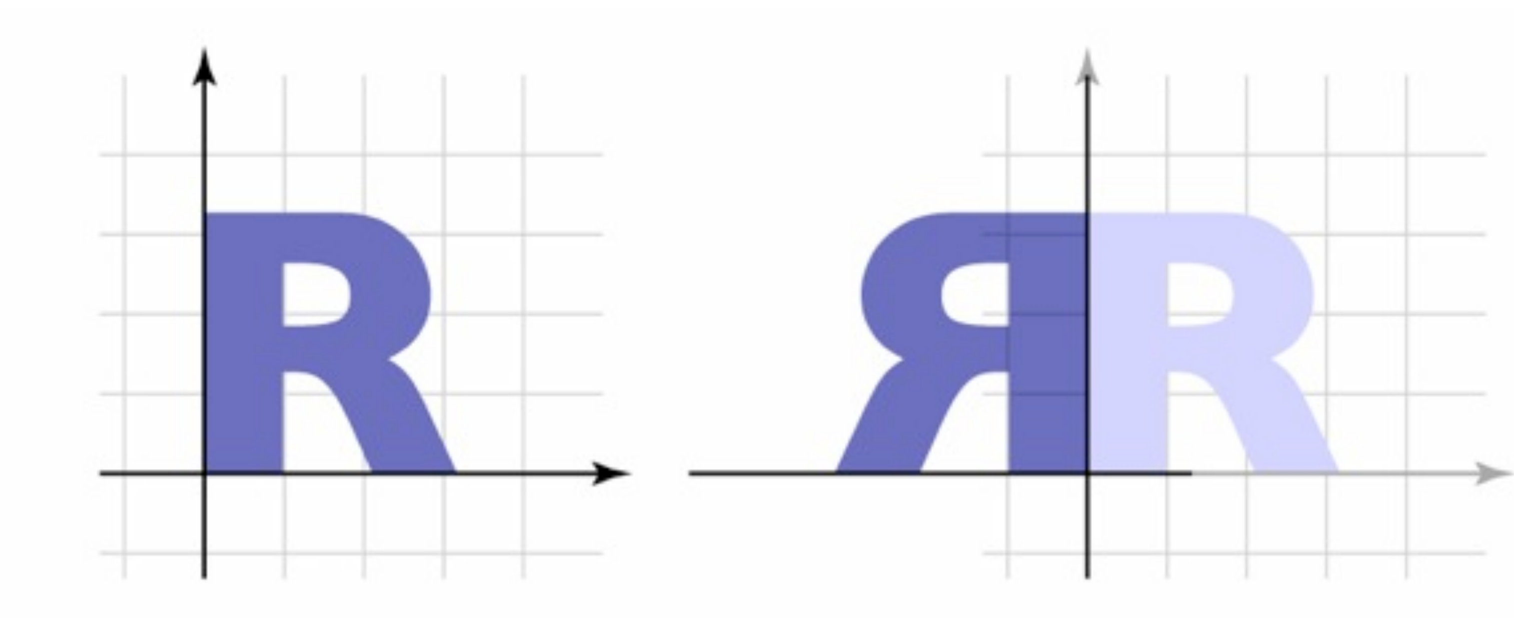
- Rotation $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0.866 & -0.5 & 0 \\ 0.5 & 0.866 & 0 \\ 0 & 0 & 1 \end{bmatrix}$



Affine transformation gallery

- Reflection
  - can consider it a special case of nonuniform scale

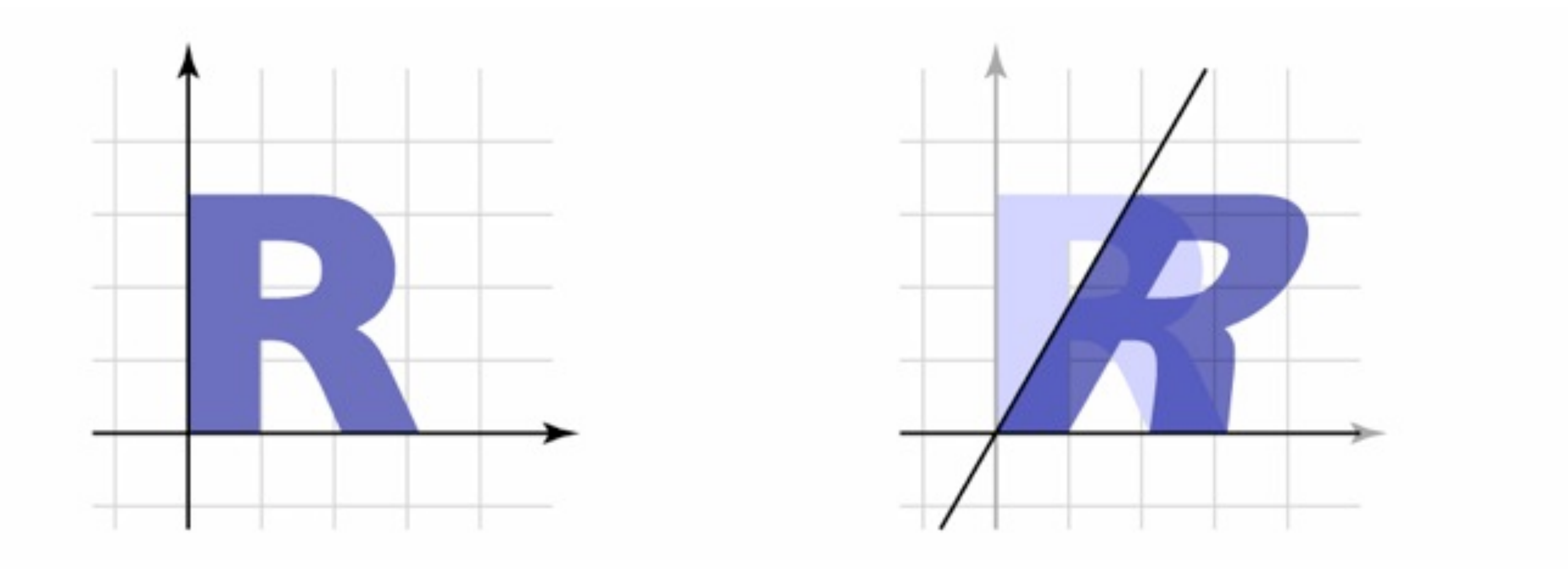
$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Affine transformation gallery

- Shear

$$\begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

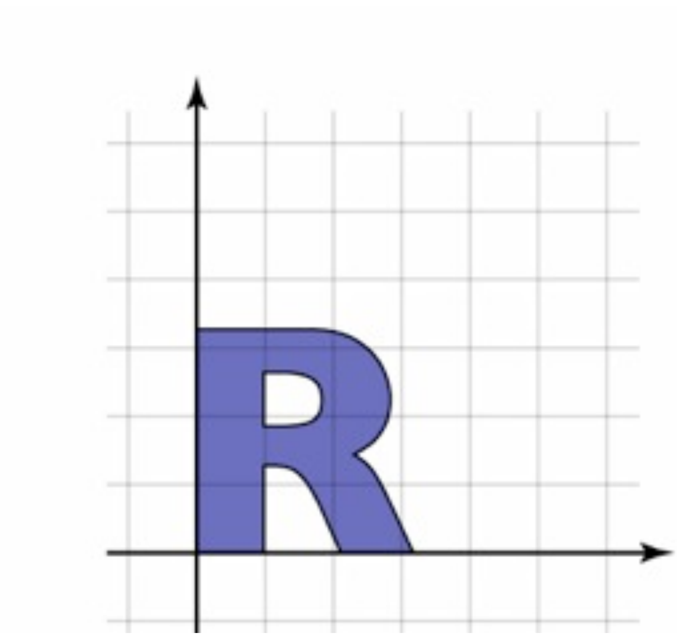


General affine transformations

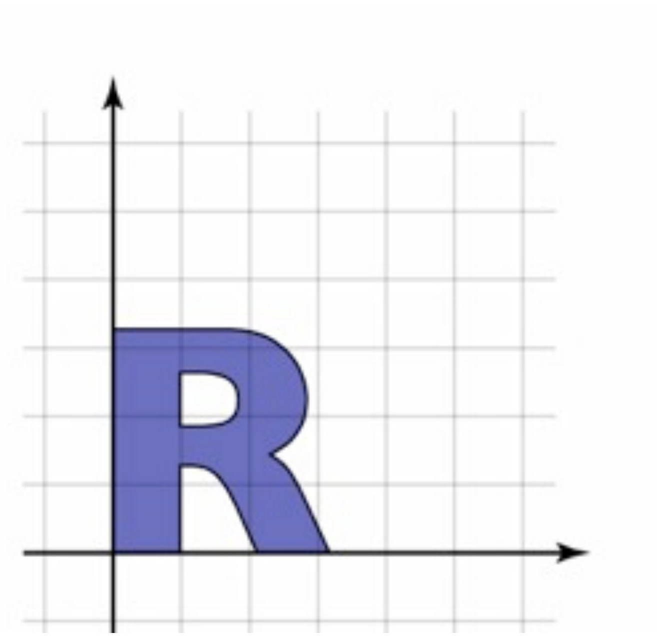
- The previous slides showed "canonical" examples of the types of affine transformations
- Generally, transformations contain elements of multiple types
  - often define them as products of canonical transforms
  - sometimes work with their properties more directly

Composite affine transformations

- In general **not** commutative: order matters!



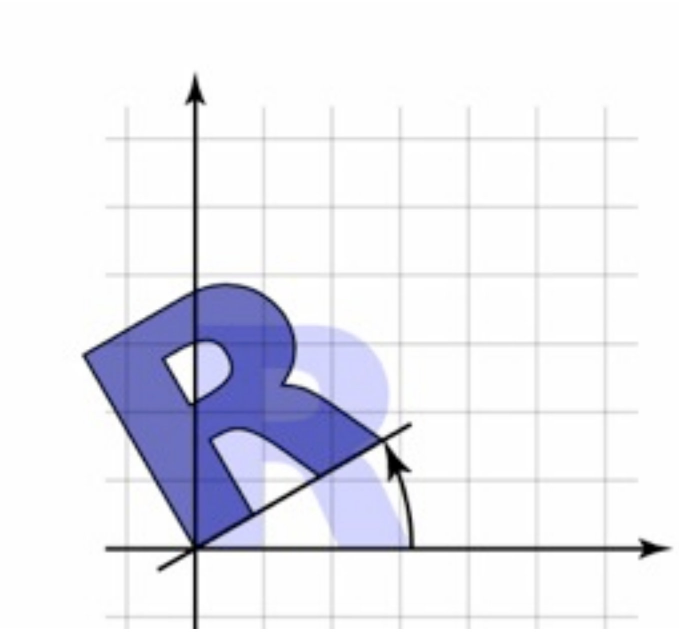
rotate, then translate



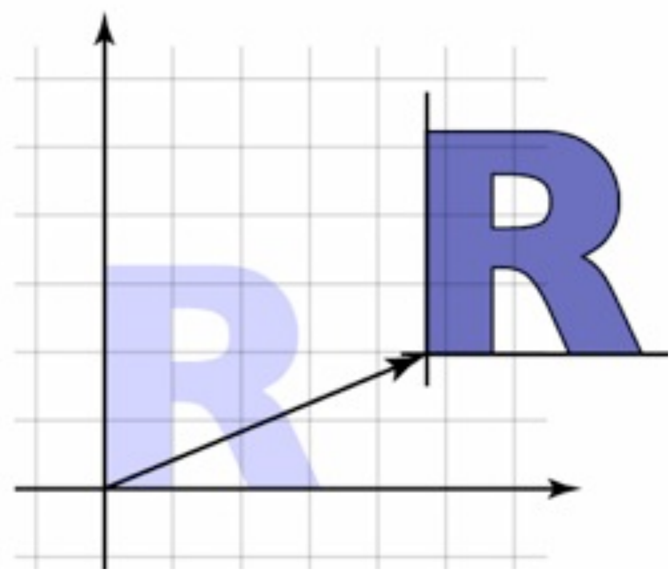
translate, then rotate

Composite affine transformations

- In general **not** commutative: order matters!



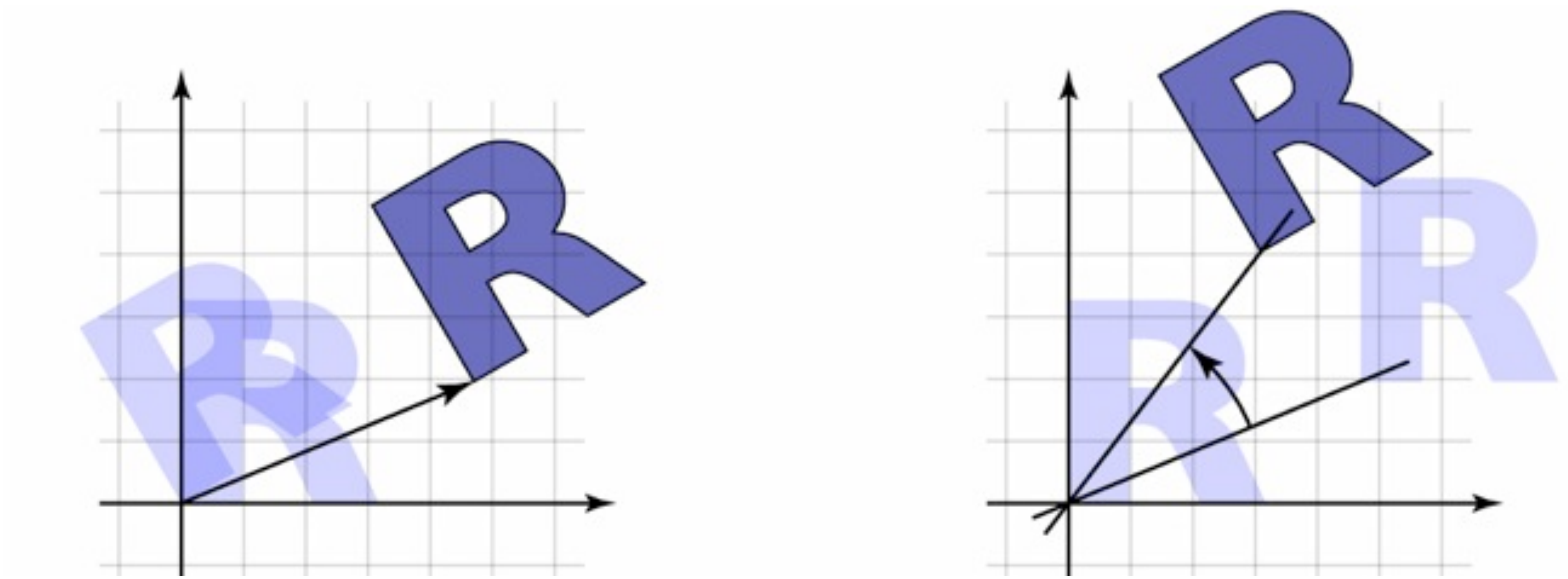
rotate, then translate



translate, then rotate

Composite affine transformations

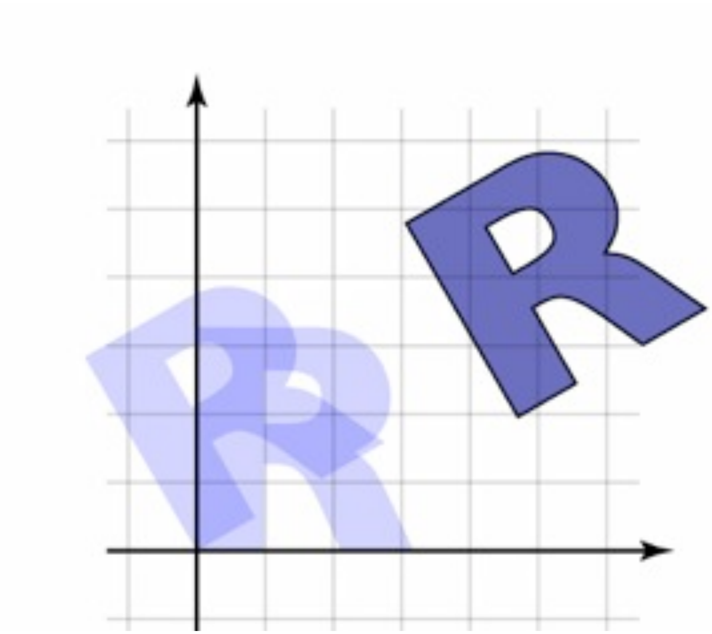
- In general **not** commutative: order matters!



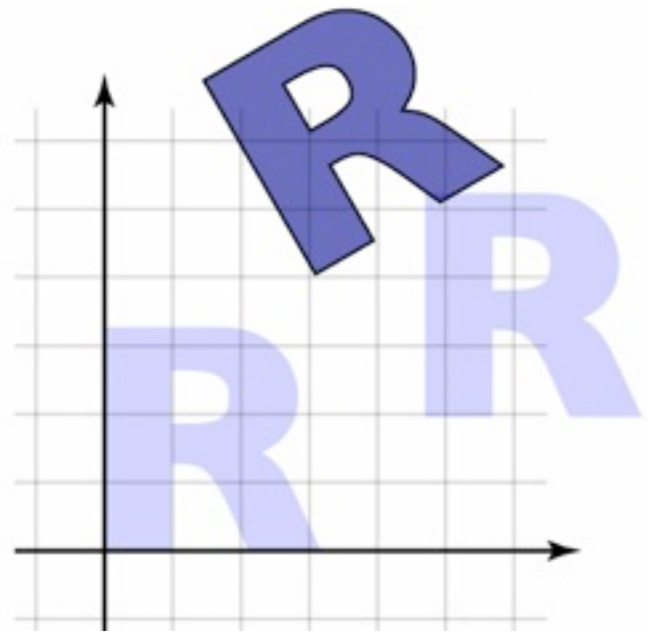
rotate, then translate　　　　translate, then rotate

Composite affine transformations

- In general **not** commutative: order matters!



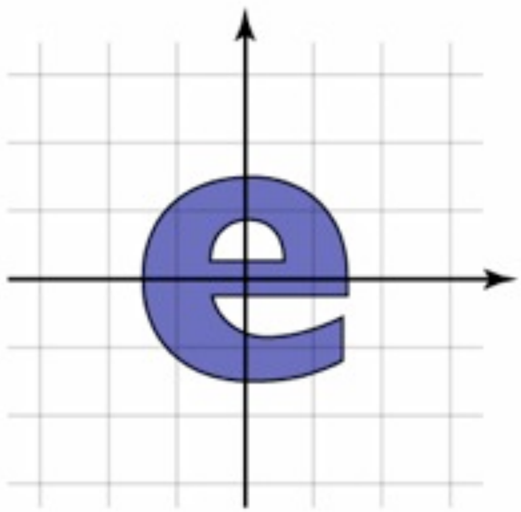
rotate, then translate

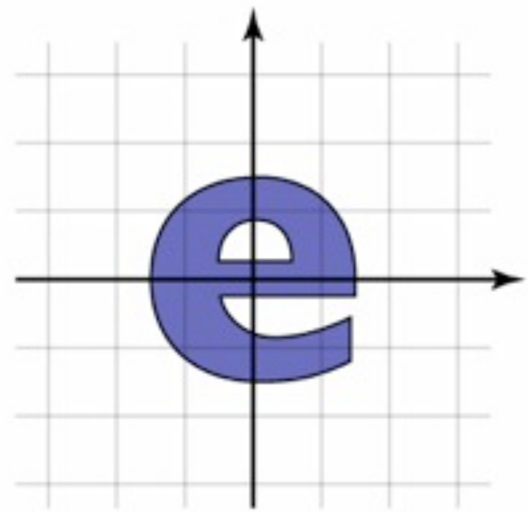


translate, then rotate

Composite affine transformations

- Another example



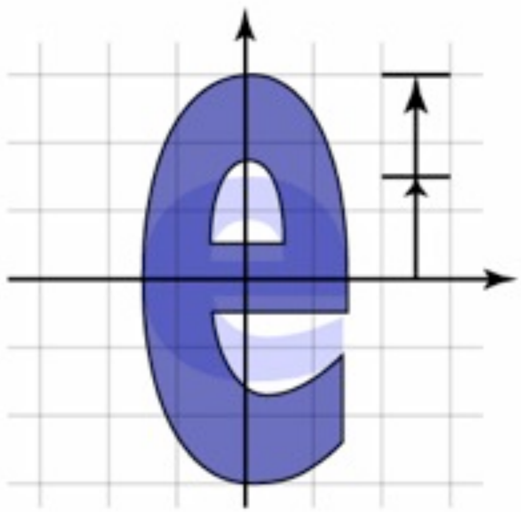


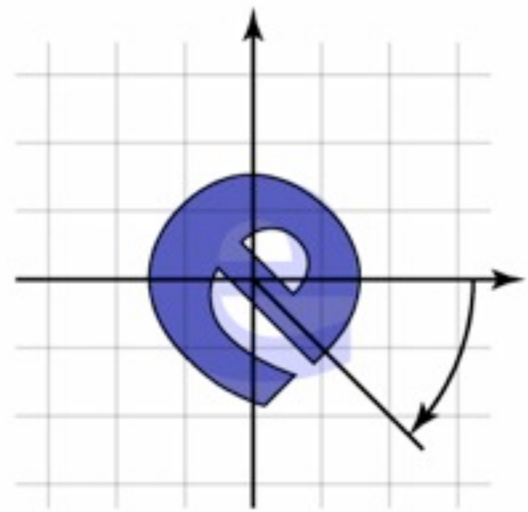
scale, then rotate

rotate, then scale

Composite affine transformations

- Another example



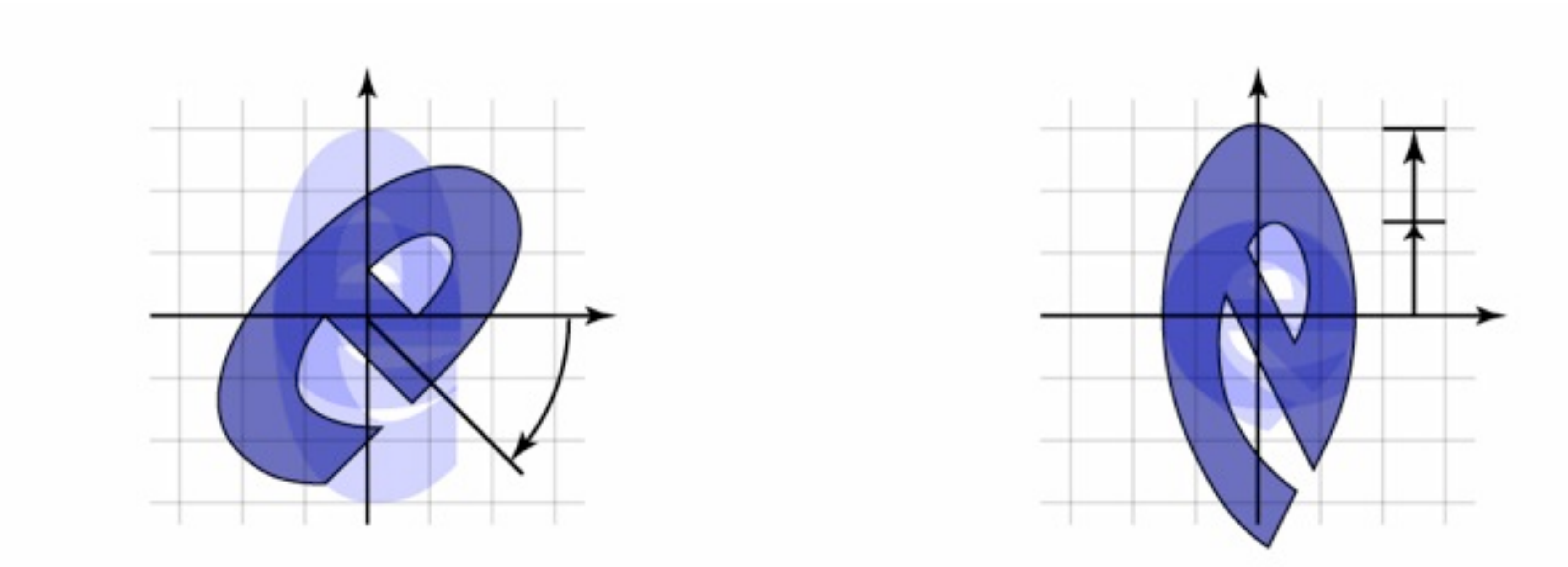


scale, then rotate

rotate, then scale

Composite affine transformations

- Another example

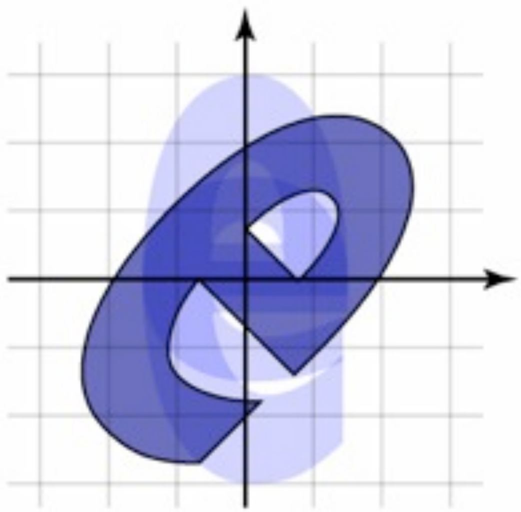


scale, then rotate

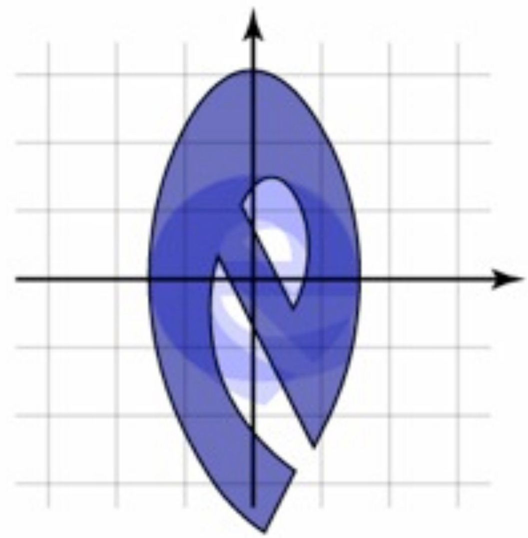
rotate, then scale

Composite affine transformations

- Another example



scale, then rotate



rotate, then scale

Rigid motions

- A transform made up of only translation and rotation is a *rigid motion* or a *rigid body transformation*
- The linear part is an orthonormal matrix

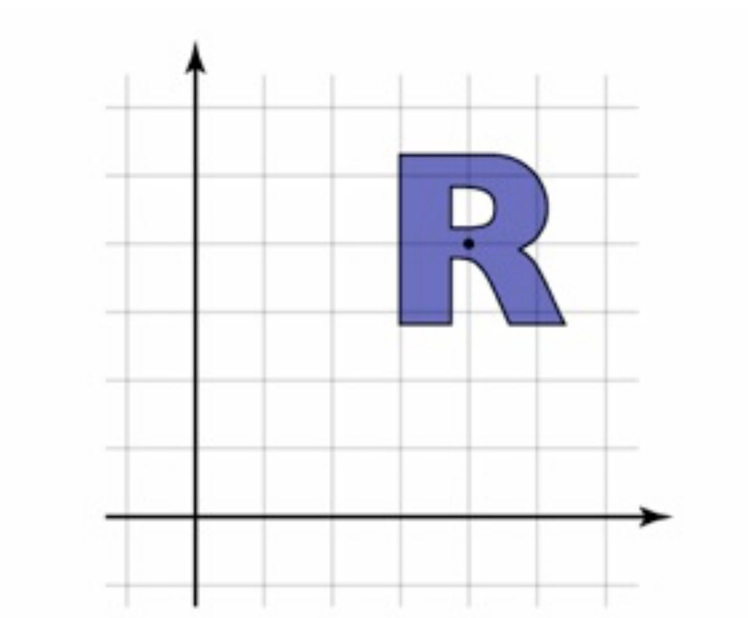
$$R = \begin{bmatrix} Q & \mathbf{u} \\ 0 & 1 \end{bmatrix}$$

- Inverse of orthonormal matrix is transpose
  – so inverse of rigid motion is easy:

$$R^{-1}R = \begin{bmatrix} Q^T & -Q^T\mathbf{u} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} Q & \mathbf{u} \\ 0 & 1 \end{bmatrix}$$

Composing to change axes

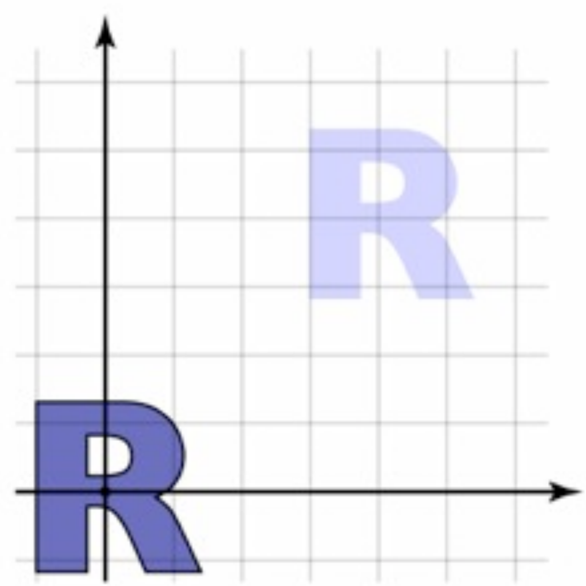
- Want to rotate about a particular point
  - could work out formulas directly…
- Know how to rotate about the origin
  - so translate that point to the origin



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

Composing to change axes

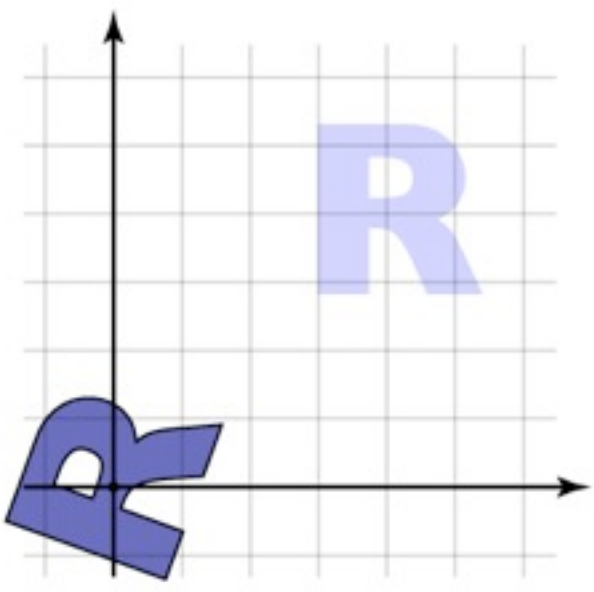
- Want to rotate about a particular point
  - could work out formulas directly…
- Know how to rotate about the origin
  - so translate that point to the origin



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

Composing to change axes

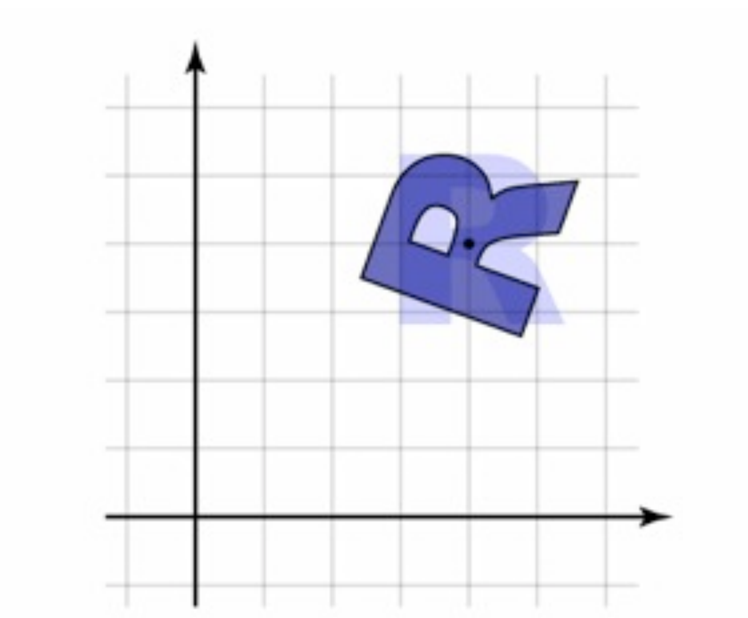
- Want to rotate about a particular point
  - could work out formulas directly…
- Know how to rotate about the origin
  - so translate that point to the origin



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

Composing to change axes

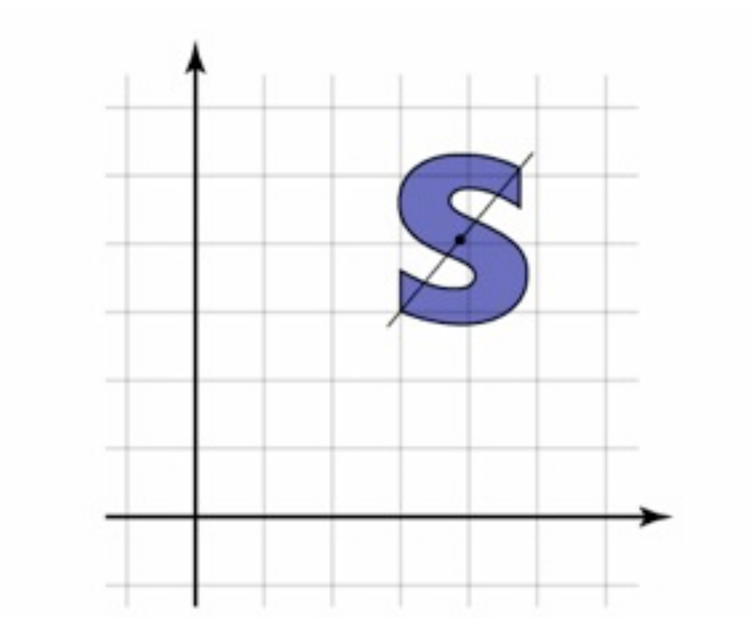
- Want to rotate about a particular point
  - could work out formulas directly…
- Know how to rotate about the origin
  - so translate that point to the origin



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

Composing to change axes

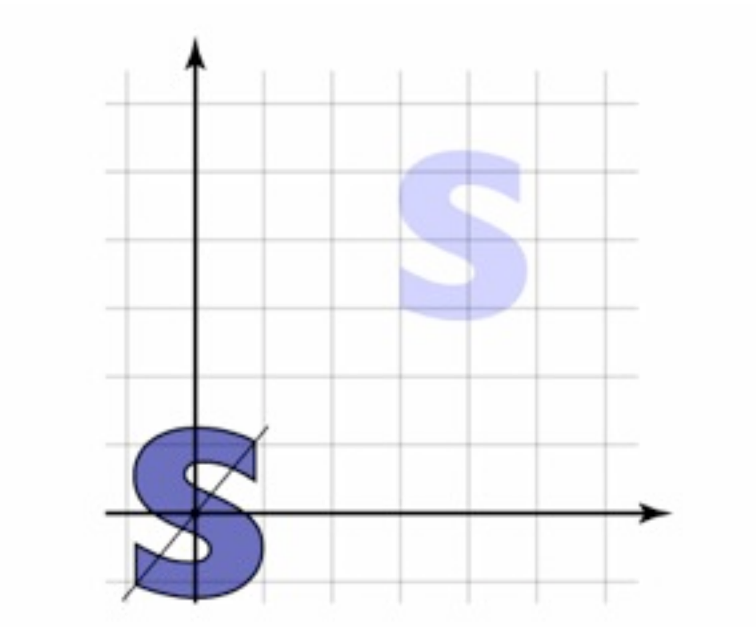
- Want to scale along a particular axis and point
- Know how to scale along the y axis at the origin
  - so translate to the origin and rotate to align axes



$$M = T^{-1}R^{-1}SRT$$

Composing to change axes

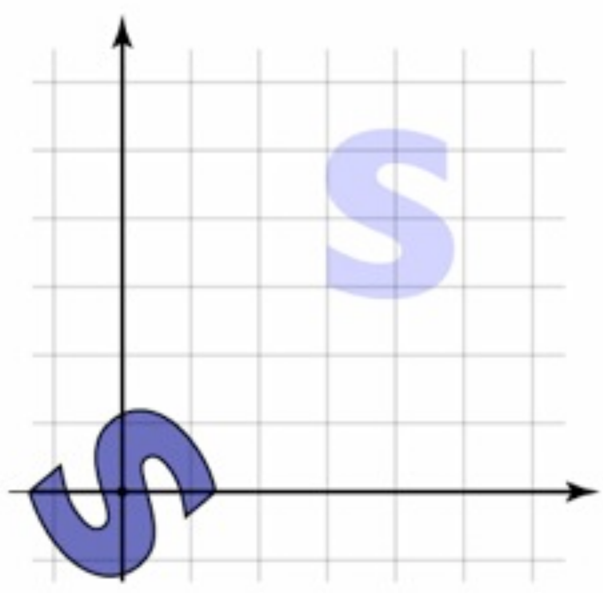
- Want to scale along a particular axis and point
- Know how to scale along the y axis at the origin
  – so translate to the origin and rotate to align axes



$$M = T^{-1} R^{-1} S R T$$

Composing to change axes

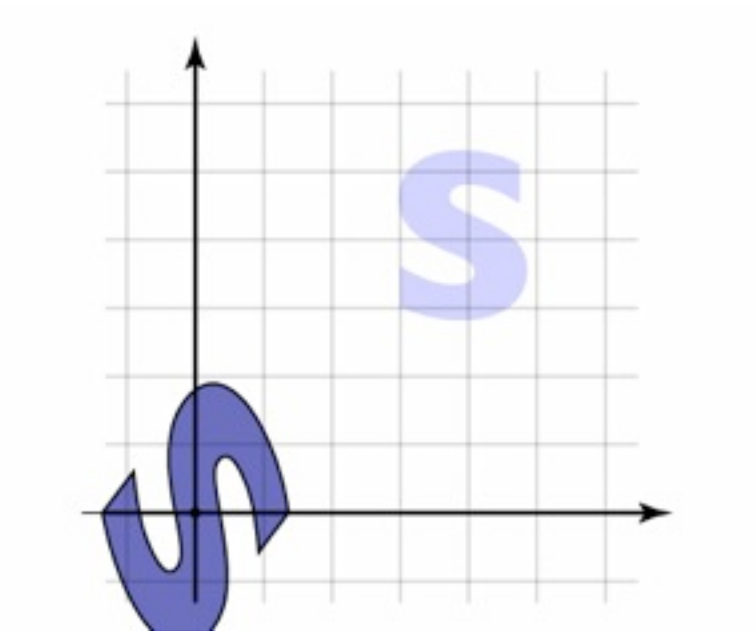
- Want to scale along a particular axis and point
- Know how to scale along the y axis at the origin
  - so translate to the origin and rotate to align axes



$$M = T^{-1}R^{-1}SRT$$

Composing to change axes

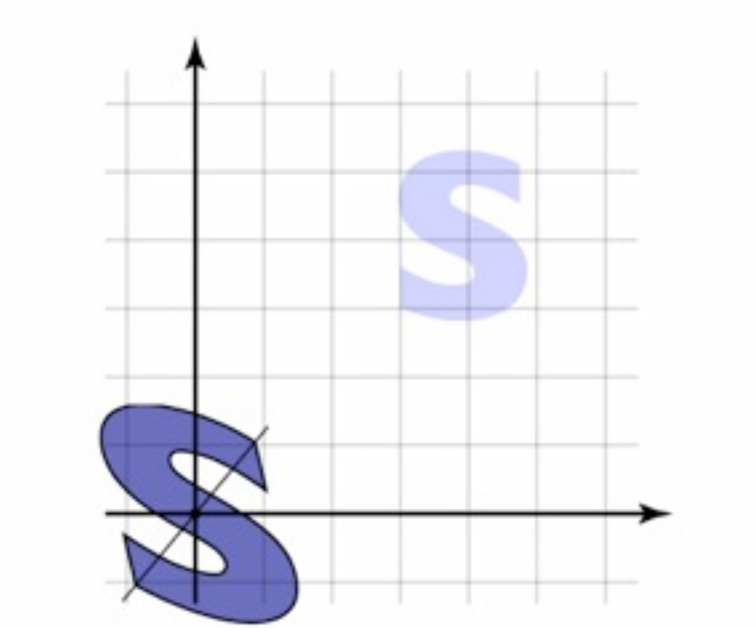
- Want to scale along a particular axis and point
- Know how to scale along the y axis at the origin
  - so translate to the origin and rotate to align axes



$$M = T^{-1}R^{-1}SRT$$

Composing to change axes

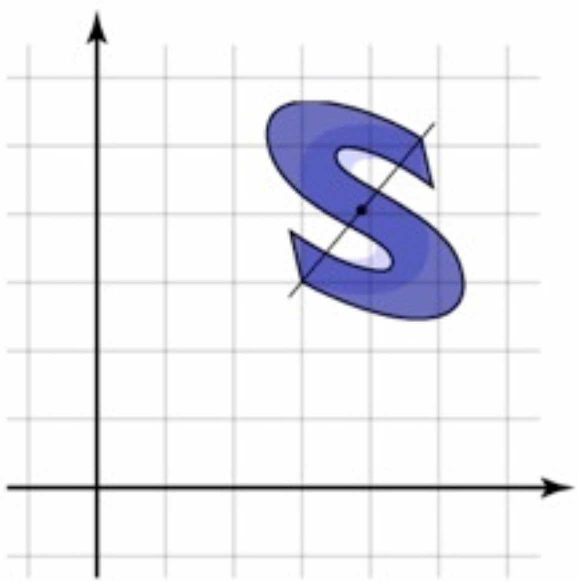
- Want to scale along a particular axis and point
- Know how to scale along the y axis at the origin
  - so translate to the origin and rotate to align axes



$$M = T^{-1}R^{-1}SRT$$

Composing to change axes

- Want to scale along a particular axis and point
- Know how to scale along the y axis at the origin
  - so translate to the origin and rotate to align axes



$$M = T^{-1} R^{-1} S R T$$

Transforming points and vectors

- Recall distinction points vs. vectors
  - vectors are just offsets (differences between points)
  - points have a location
    - represented by vector offset from a fixed origin
- Points and vectors transform differently
  - points respond to translation; vectors do not

$$\mathbf{v} = \mathbf{p} - \mathbf{q}$$

$$T(\mathbf{x}) = M\mathbf{x} + \mathbf{t}$$

$$\begin{aligned} T(\mathbf{p} - \mathbf{q}) &= M\mathbf{p} + \mathbf{t} - (M\mathbf{q} + \mathbf{t}) \\ &= M(\mathbf{p} - \mathbf{q}) + (\mathbf{t} - \mathbf{t}) = M\mathbf{v} \end{aligned}$$

Transforming points and vectors

- Homogeneous coords. let us exclude translation
  - just put 0 rather than 1 in the last place

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

  - and note that subtracting two points cancels the extra coordinate, resulting in a vector!
- Preview: projective transformations
  - what's really going on with this last coordinate?
  - think of R^2 embedded in R^3: all affine xfs. preserve z=1 plane
  - could have other transforms; project back to z=1

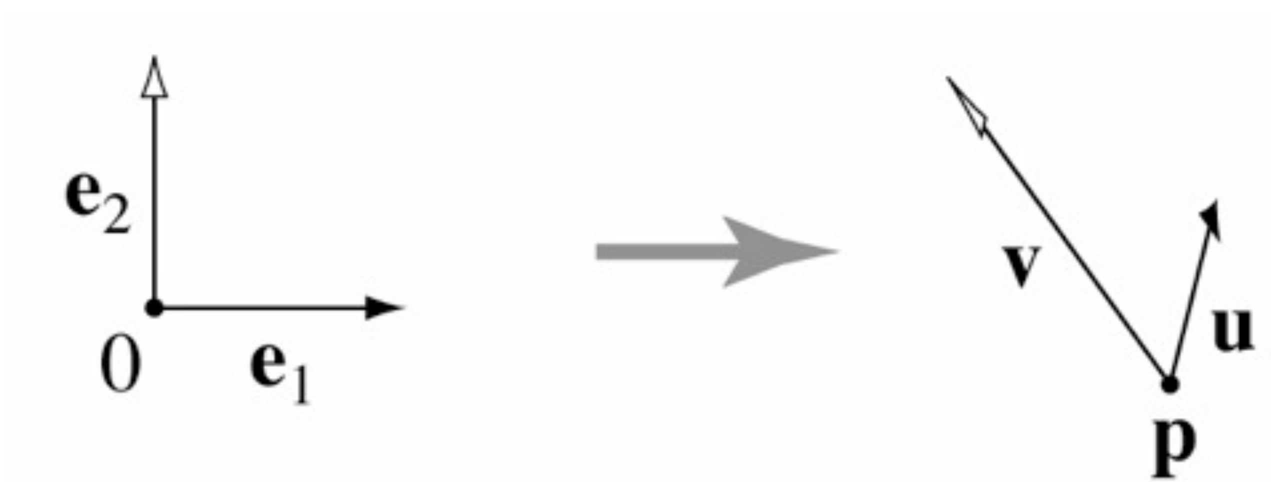
More math background

- Coordinate systems
  - Expressing vectors with respect to bases
  - Linear transformations as changes of basis

Affine change of coordinates

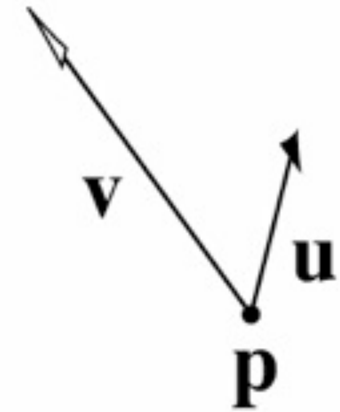
- Six degrees of freedom

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{p} \\ 0 & 0 & 1 \end{bmatrix}$$



Affine change of coordinates

- Coordinate frame: point plus basis
- Interpretation: transformation changes representation of point from one basis to another
- "Frame to canonical" matrix has frame in columns
  - takes points represented in frame
  - represents them in canonical basis
  - e.g. [0 0], [1 0], [0 1]
- Seems backward but bears thinking about

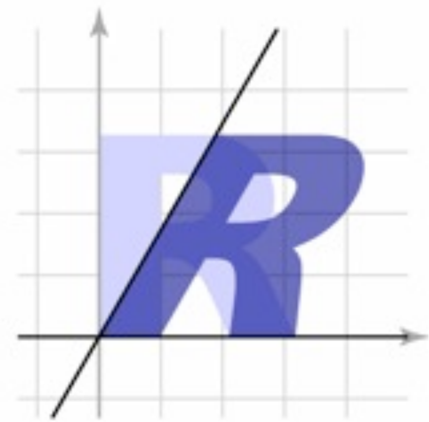


$$\begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{p} \\ 0 & 0 & 1 \end{bmatrix}$$

Affine change of coordinates

- A new way to "read off" the matrix
  - e.g. shear from earlier
  - can look at picture, see effect on basis vectors, write down matrix

$$\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- Also an easy way to construct transforms
  - e. g. scale by 2 across direction (1,2)

Affine change of coordinates

- When we move an object to the origin to apply a transformation, we are really changing coordinates
  - the transformation is easy to express in object's frame
  - so define it there and transform it

$$T_e = F T_F F^{-1}$$

  - T_e is the transformation expressed wrt. $\{e_1, e_2\}$
  - T_F is the transformation expressed in natural frame
  - F is the frame-to-canonical matrix $[u\ v\ p]$
- This is a *similarity transformation*

Coordinate frame summary

- Frame = point plus basis
- Frame matrix (frame-to-canonical) is

$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{p} \\ 0 & 0 & 1 \end{bmatrix}$$

- Move points to and from frame by multiplying with *F*

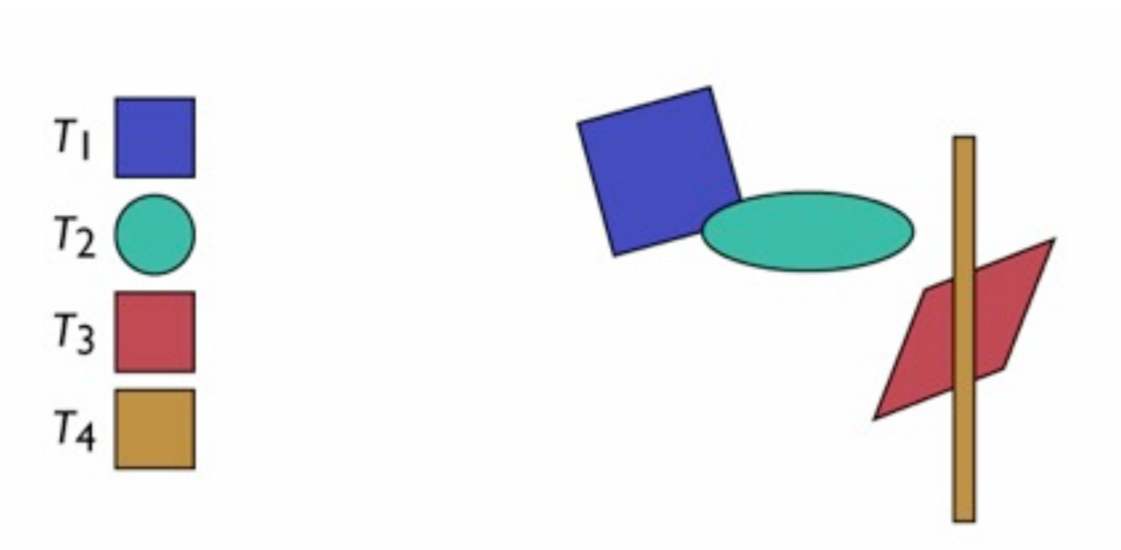
$$p_e = F p_F \quad p_F = F^{-1} p_e$$

- Move transformations using similarity transforms

$$T_e = F T_F F^{-1} \quad T_F = F^{-1} T_e F$$

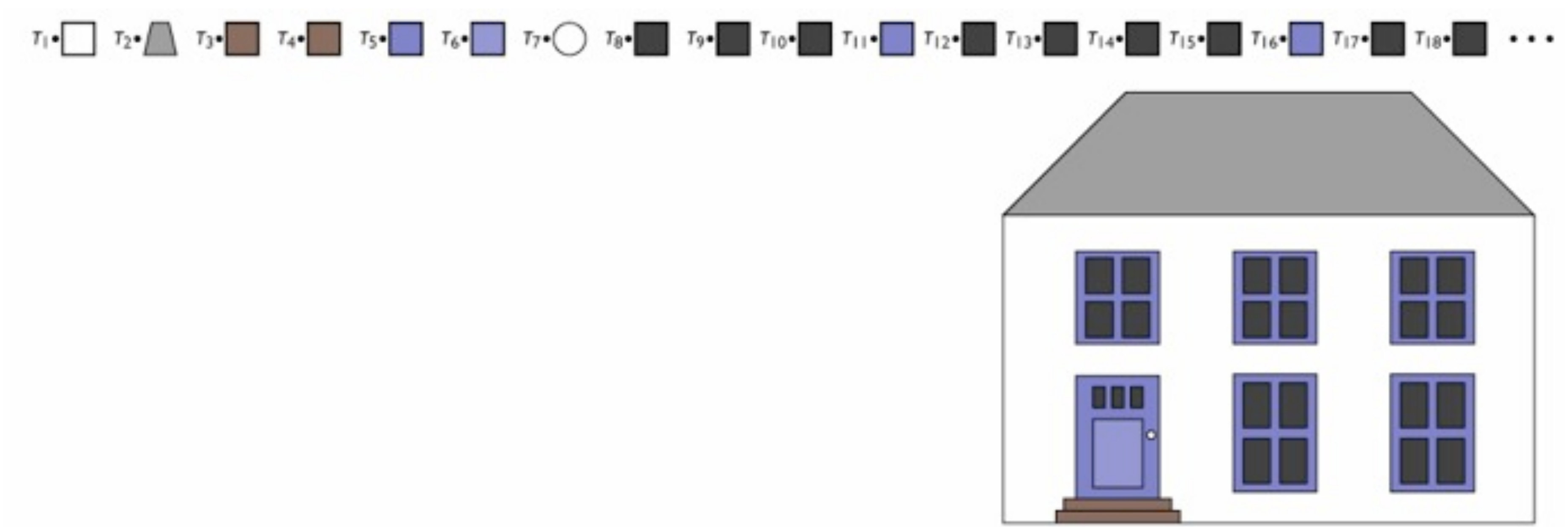
Data structures with transforms

- 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



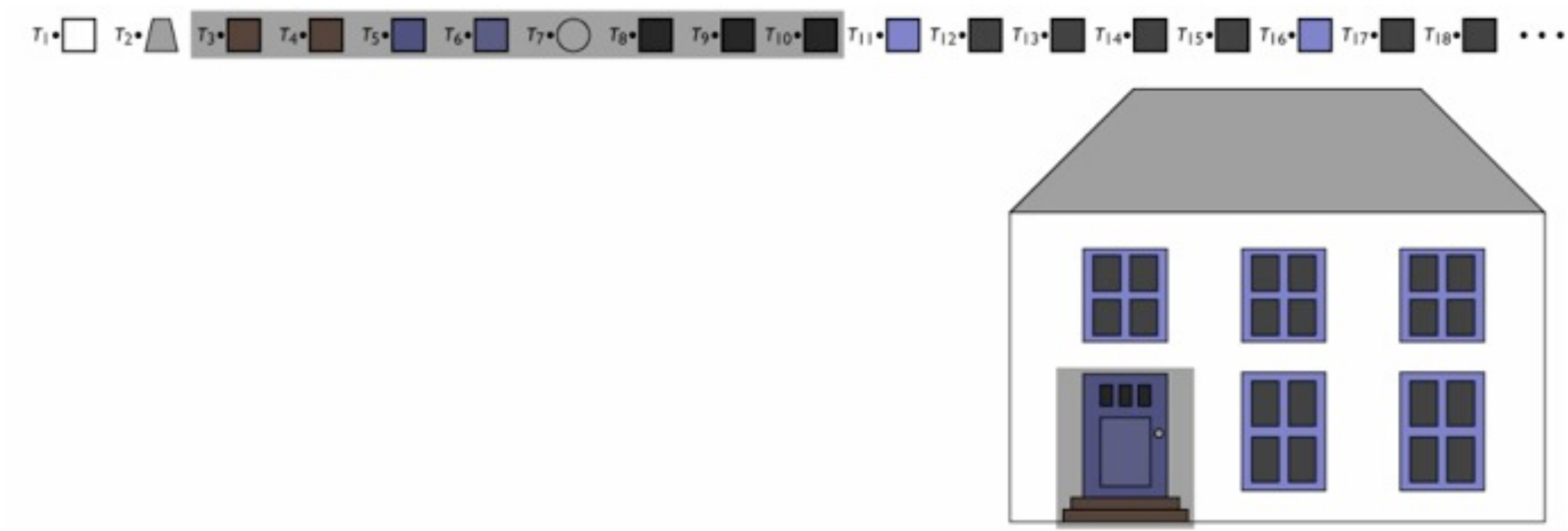
Example

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



Example

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

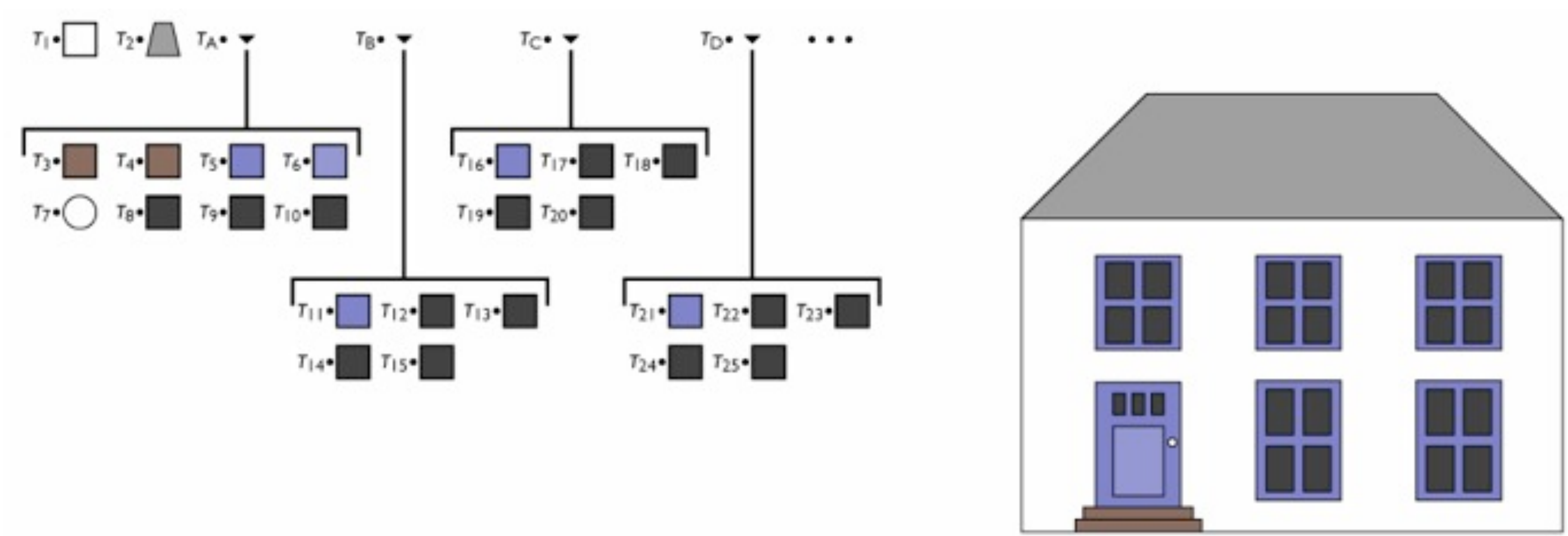


Groups of objects

- 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

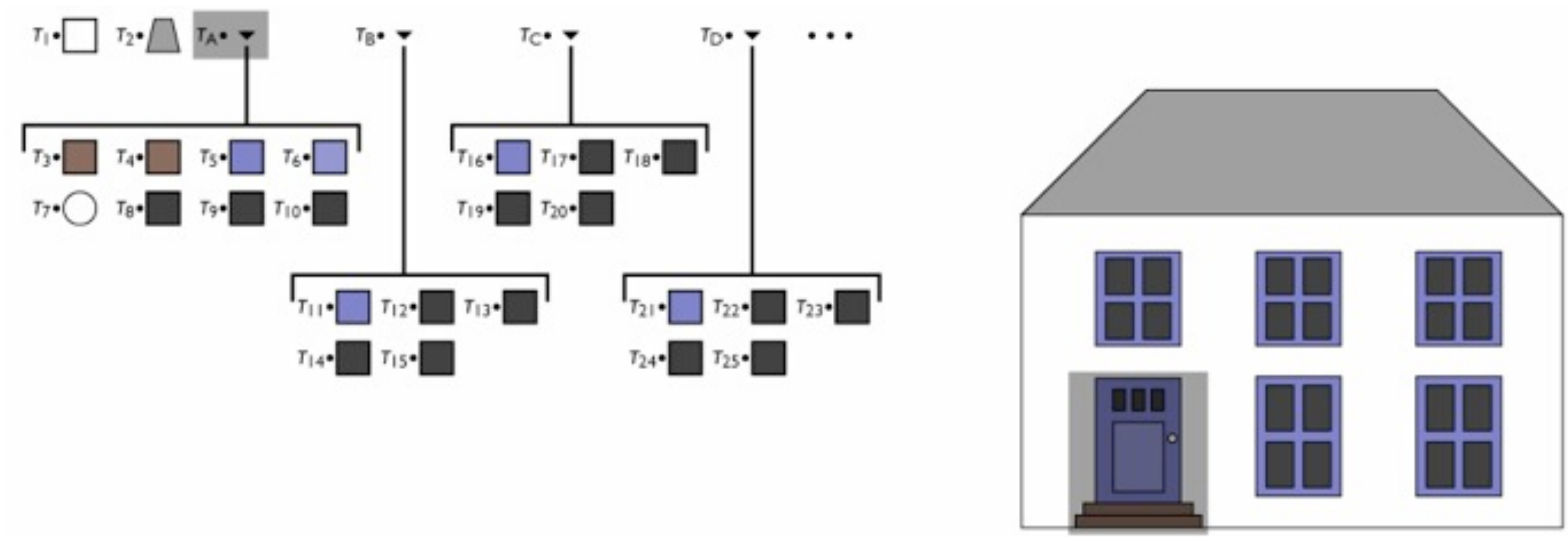
Example

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



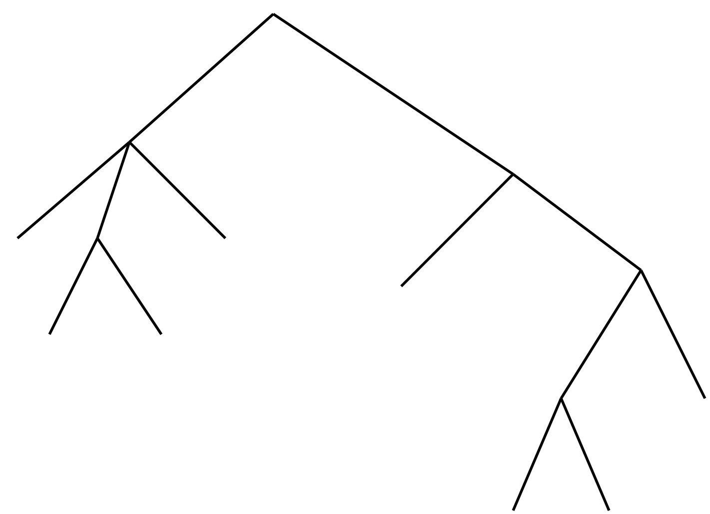
Example

- 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

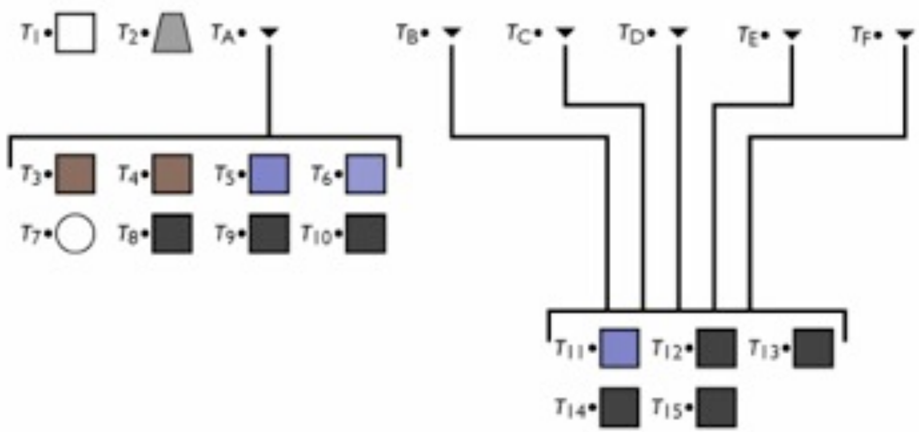
- Transforms associated with nodes or edges
- Each transform applies to all geometry below it
  - want group transform to transform each member
  - members already transformed—concatenate
- Frame transform for object is product of all matrices along path from root
  - each object's transform describes 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

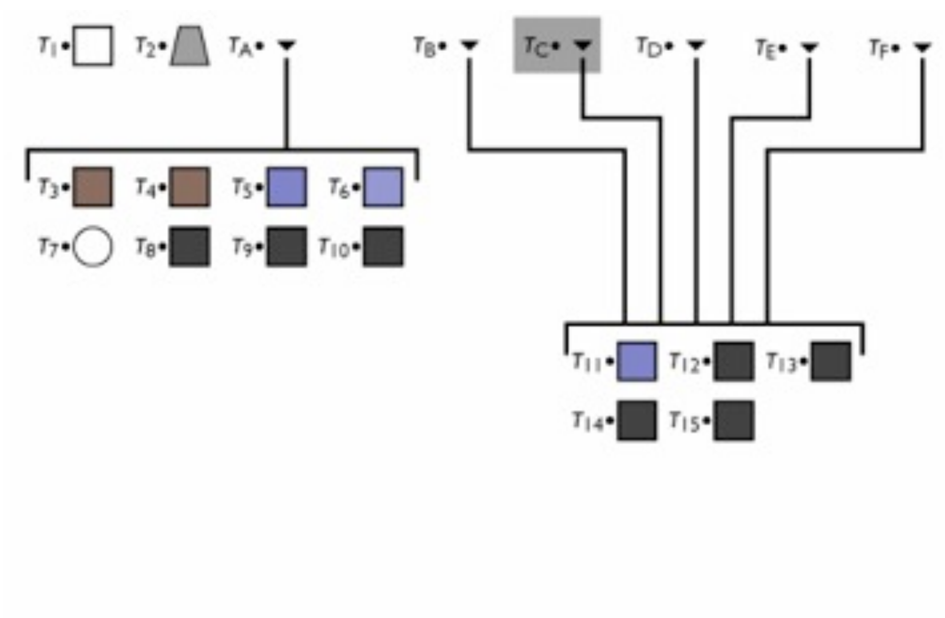
Example

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



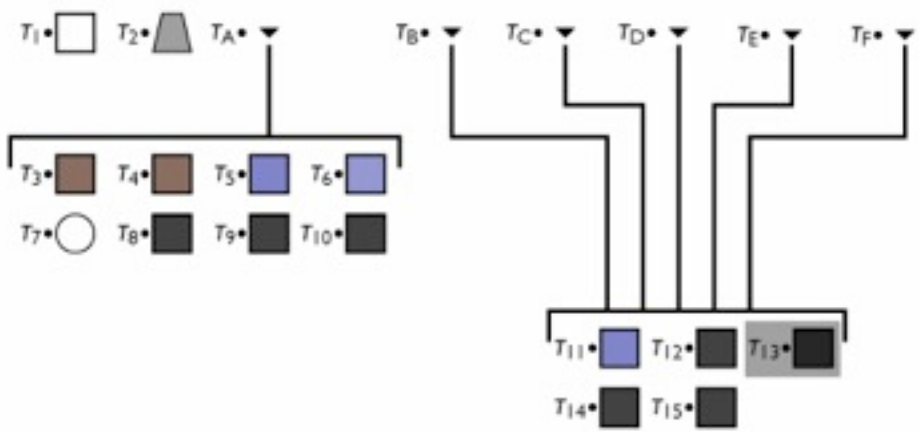
Example

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



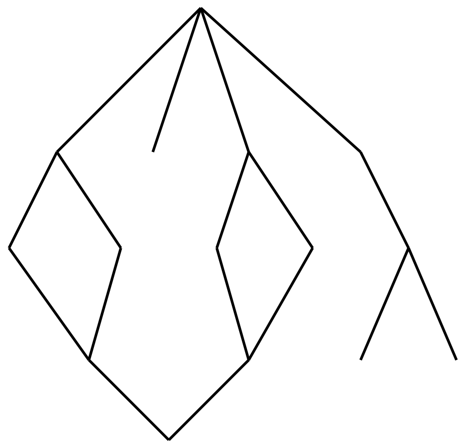
Example

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

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

Implementing traversal

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

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

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

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

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

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

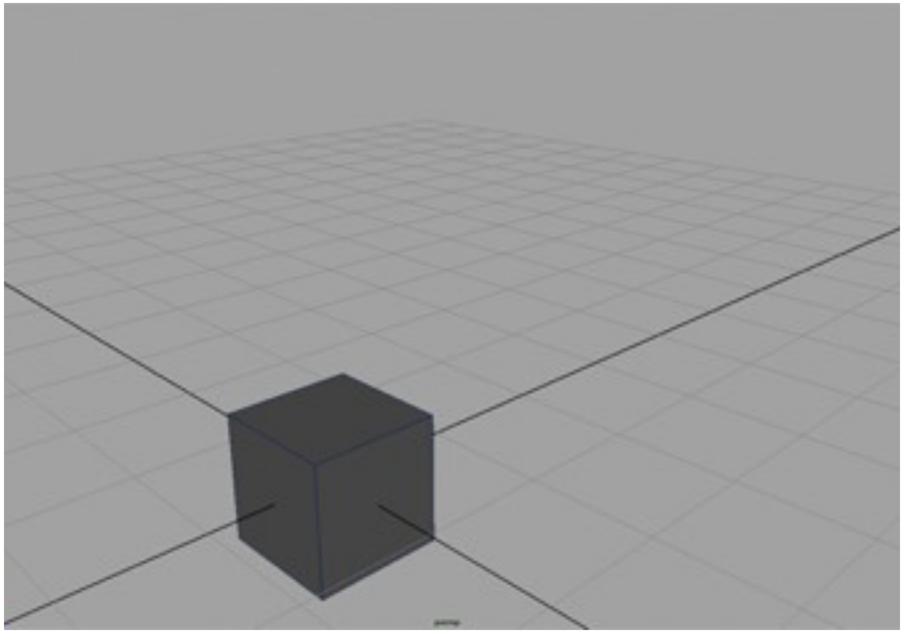
- Objects have properties besides shape
  - color, shading parameters
  - approximation parameters (e.g. precision of subdividing curved surfaces into triangles)
  - behavior in response to user input
  - …
- Setting properties for entire groups is useful
  - paint entire window green
- Many systems 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?

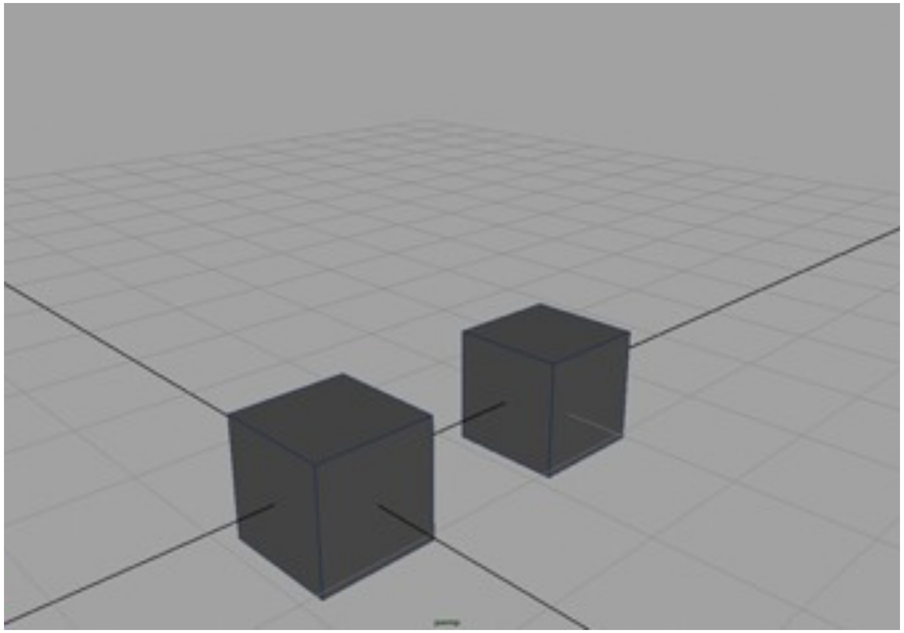
Translation

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



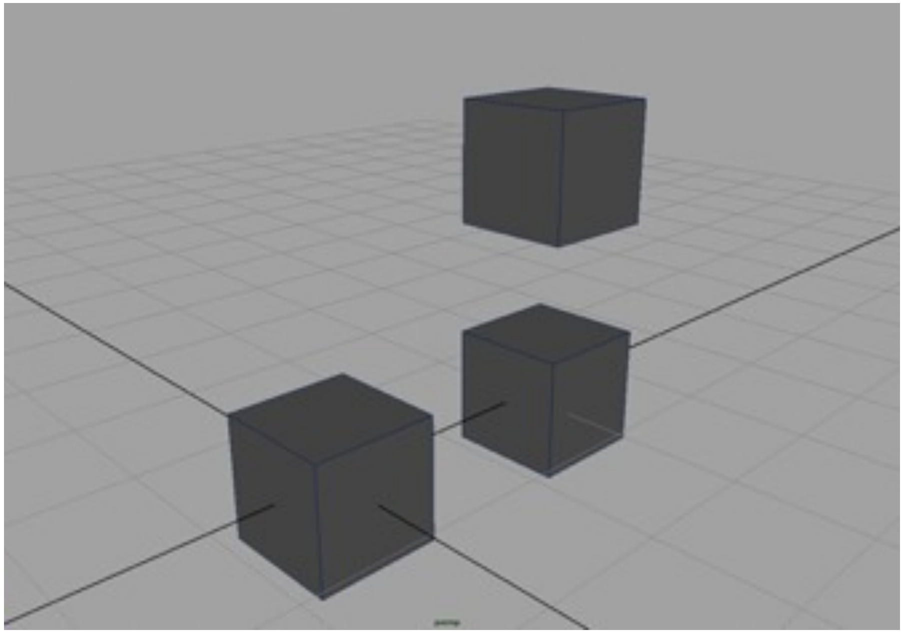
Translation

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



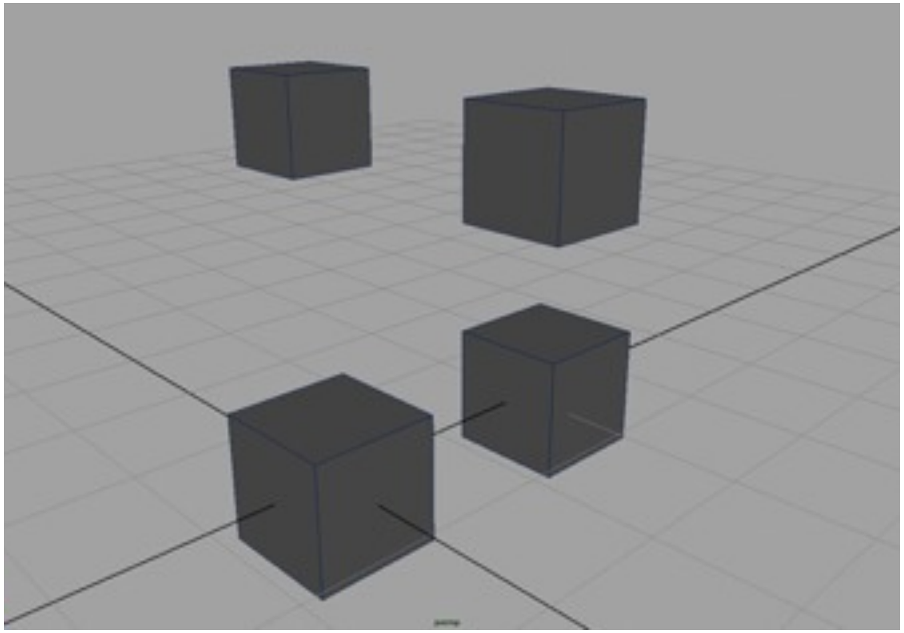
Translation

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



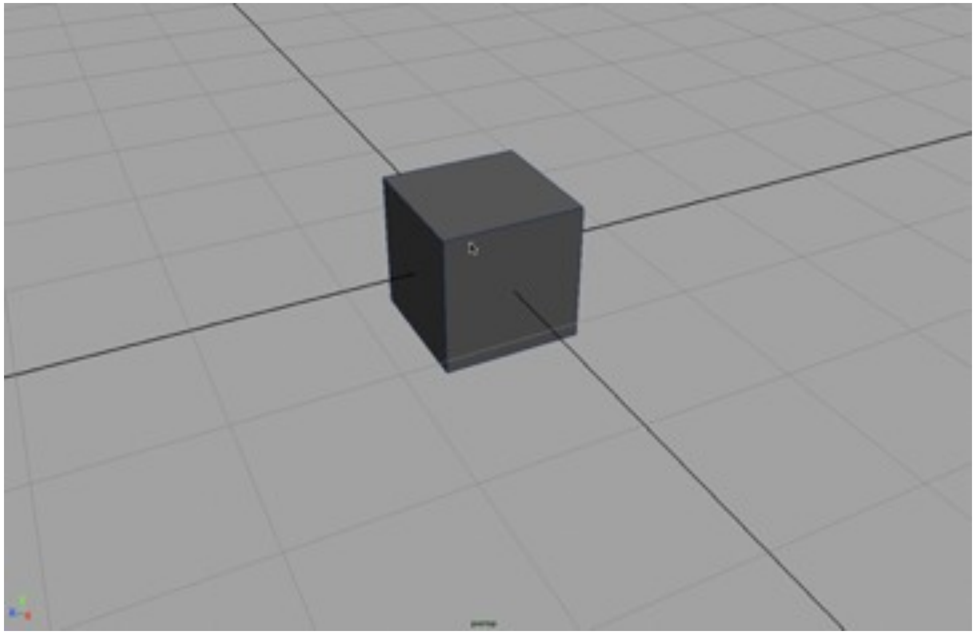
Translation

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



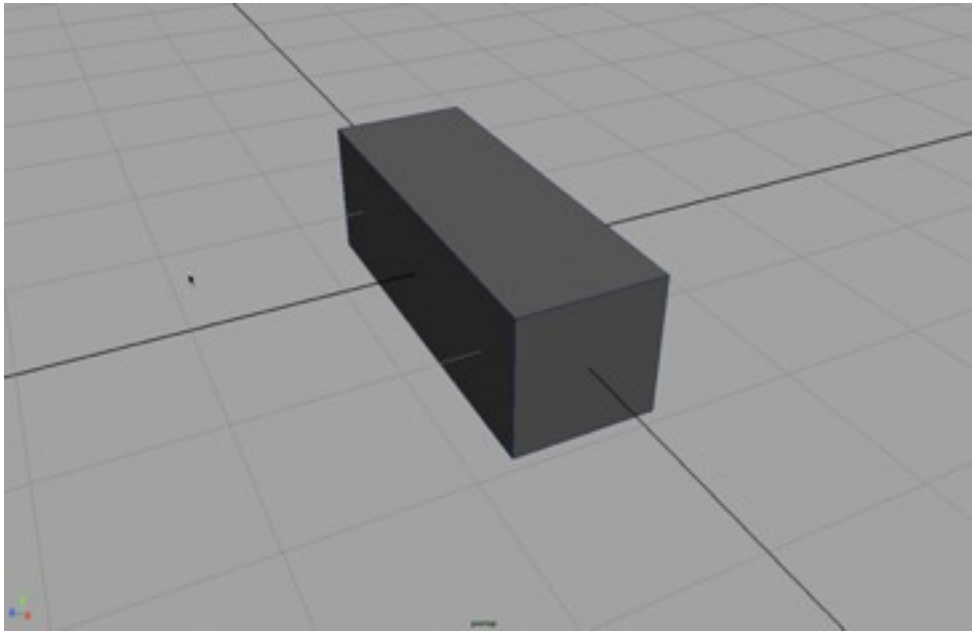
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



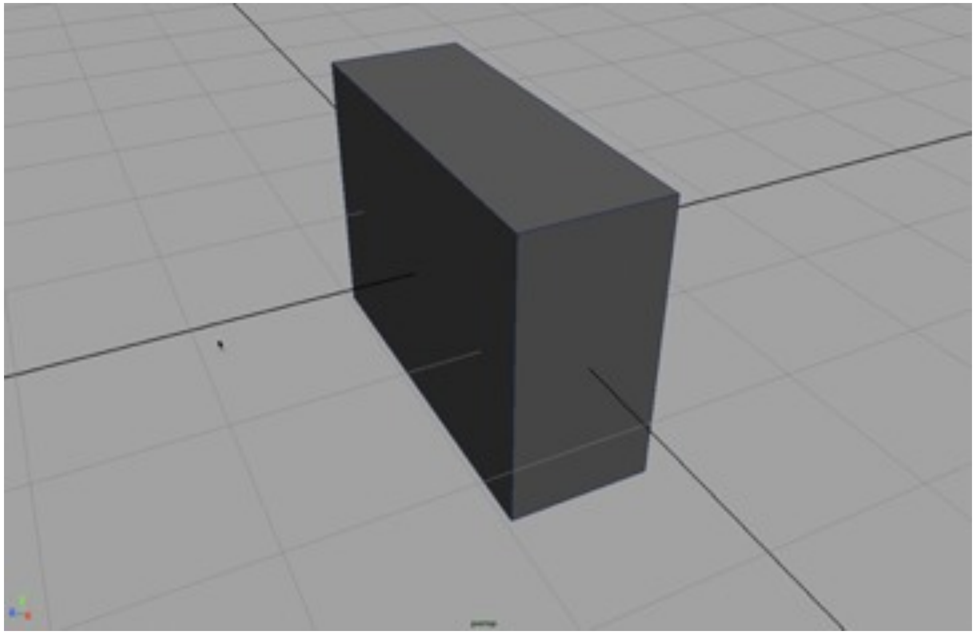
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



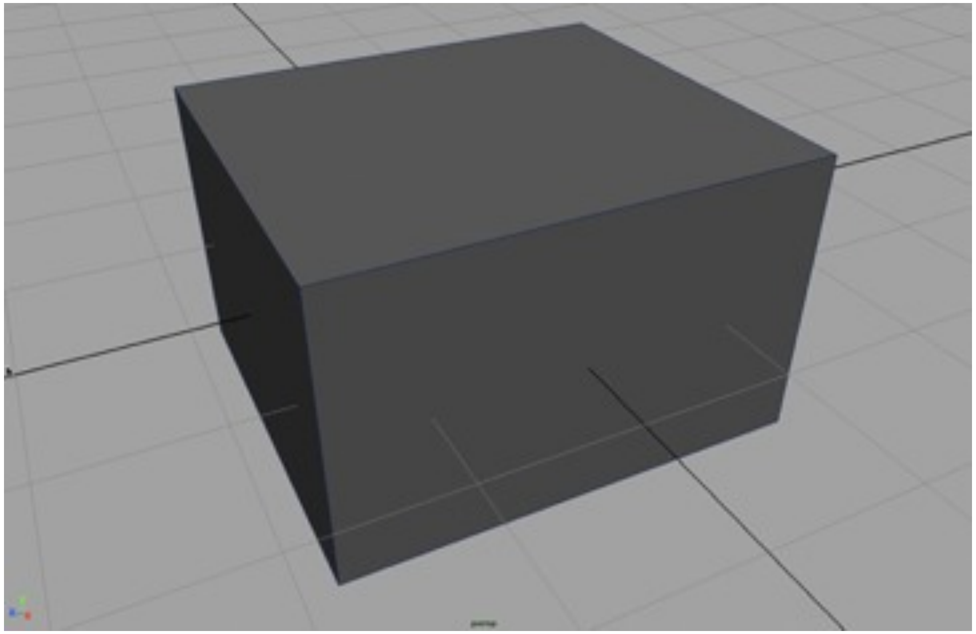
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



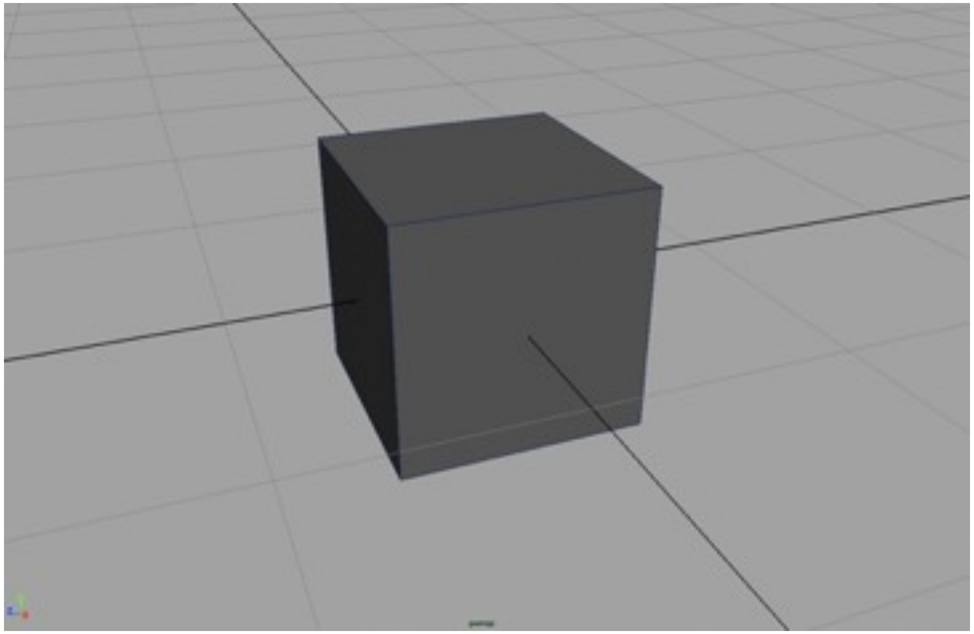
Scaling

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



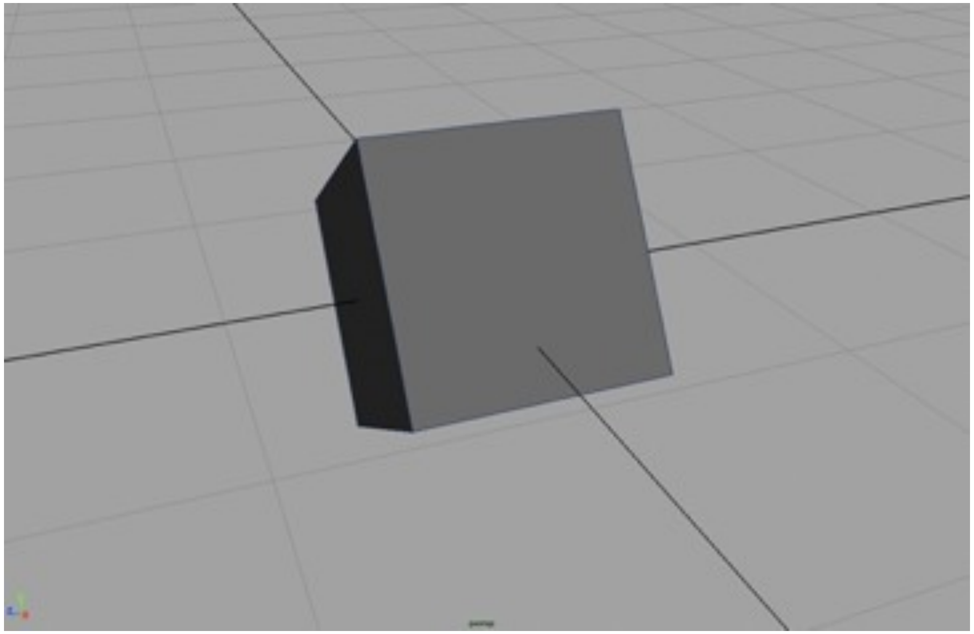
Rotation about z axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



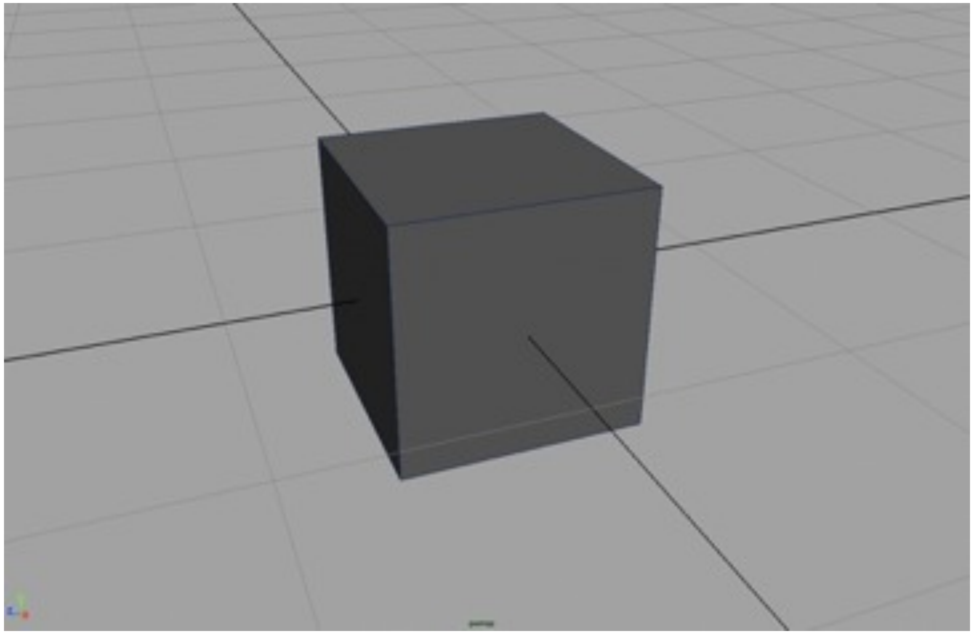
Rotation about z axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



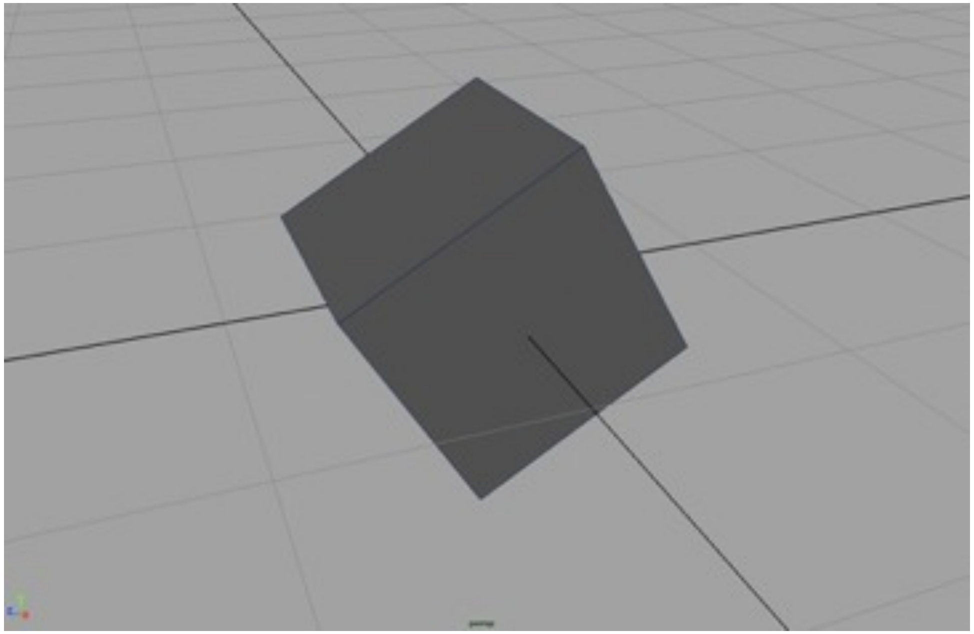
Rotation about *x* axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



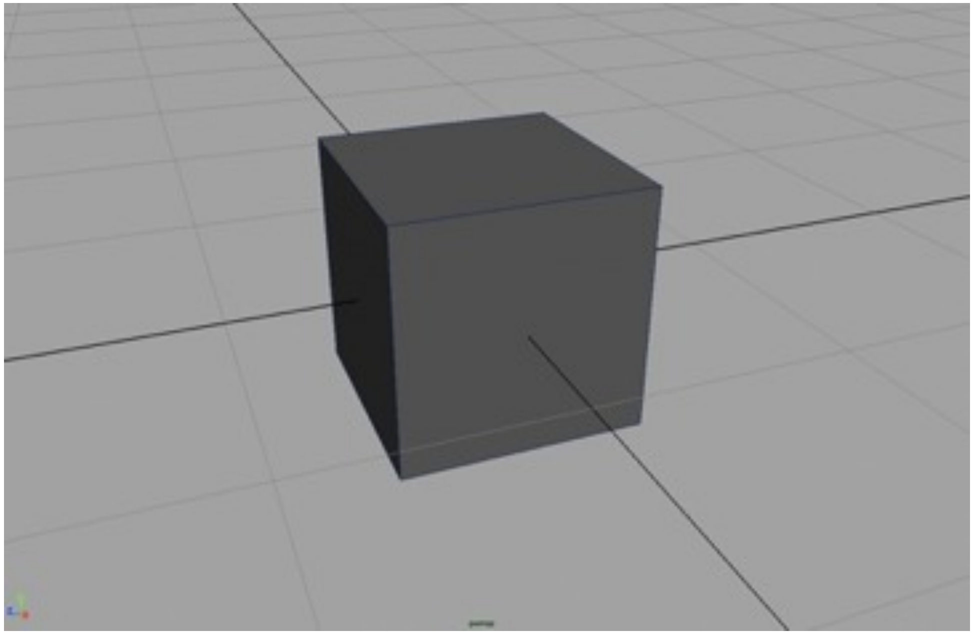
Rotation about *x* axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



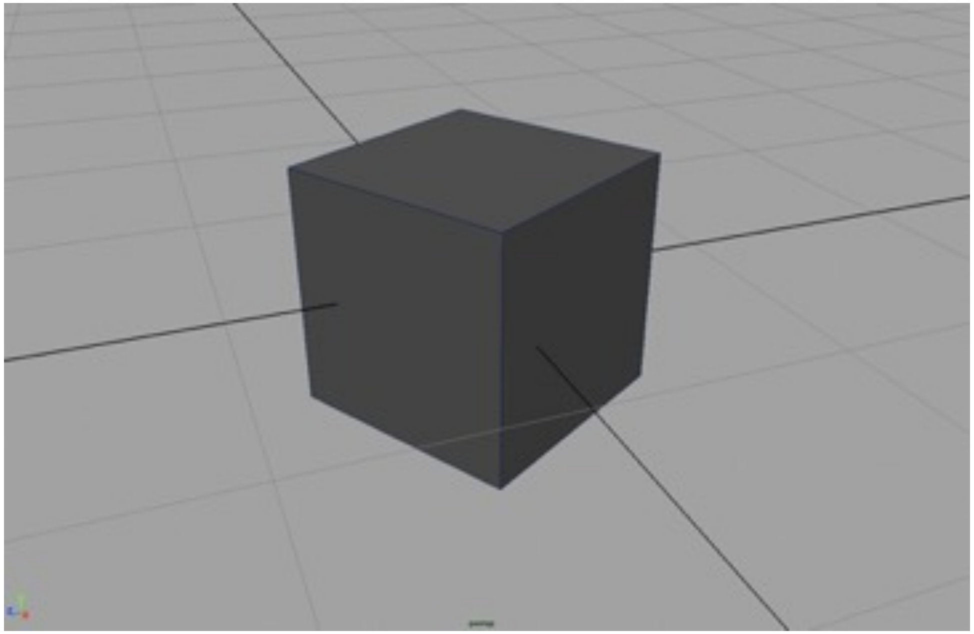
Rotation about y axis

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



Rotation about y axis

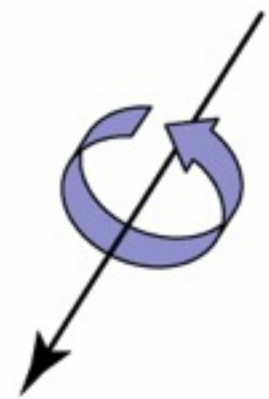
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$



General rotations

- A rotation in 2D is around a point
- A rotation in 3D is around an axis
  - so 3D rotation is w.r.t a line, not just a point
  - there are many more 3D rotations than 2D
    - a 3D space around a given point, not just 1D

2D



3D

Specifying rotations

- In 2D, a rotation just has an angle
  - if it's about a particular center, it's a point and angle
- In 3D, specifying a rotation is more complex
  - basic rotation about origin: unit vector (axis) and angle
    - convention: positive rotation is CCW when vector is pointing at you
  - about different center: point (center), unit vector, and angle
    - this is redundant: think of a second point on the same axis...
- Alternative: Euler angles
  - stack up three coord axis rotations

Coming up with the matrix

- Showed matrices for coordinate axis rotations
  - but what if we want rotation about some random axis?
- Compute by composing elementary transforms
  - transform rotation axis to align with *x* axis
  - apply rotation
  - inverse transform back into position
- Just as in 2D this can be interpreted as a similarity transform

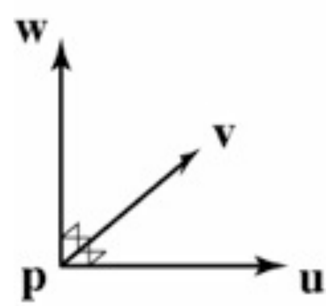
Building general rotations

- Using elementary transforms you need three
  - translate axis to pass through origin
  - rotate about y to get into x-y plane
  - rotate about z to align with x axis
- Alternative: construct frame and change coordinates
  - choose p, u, v, w to be orthonormal frame with p and u matching the rotation axis
  - apply similarity transform $T = F\, R_x(\theta)\, F^{-1}$

Orthonormal frames in 3D

- Useful tools for constructing transformations
- Recall rigid motions
  - affine transforms with pure rotation
  - columns (and rows) form right handed ONB
    - that is, an **o**rtho**n**ormal **b**asis

$$F = \begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{w} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



Building 3D frames

- Given a vector **a** and a secondary vector **b**
  - The **u** axis should be parallel to **a**; the **u**–**v** plane should contain **b**
    - **u** = **u** / ||**u**||
    - **w** = **u** x **b**; **w** = **w** / ||w||
    - **v** = **w** x **u**
- Given just a vector **a**
  - The **u** axis should be parallel to **a**; don't care about orientation about that axis
    - Same process but choose arbitrary **b** first
    - Good choice is not near **a**: e.g. set smallest entry to 1

Building general rotations

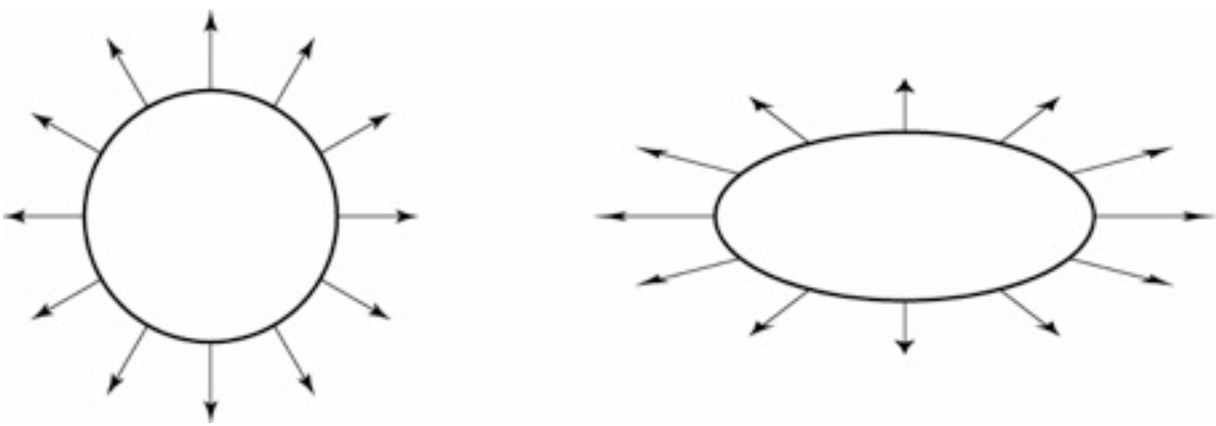
- Alternative: construct frame and change coordinates
  - choose *p*, *u*, *v*, *w* to be orthonormal frame with *p* and *u* matching the rotation axis
  - apply similarity transform $T = F R_x(\theta) F^{-1}$
  - interpretation: move to *x* axis, rotate, move back
  - interpretation: rewrite *u*-axis rotation in new coordinates
  - (each is equally valid)

Building transforms from points

- Recall2D affine transformation has 6 degrees of freedom (DOFs)
  - this is the number of "knobs" we have to set to define one
- Therefore 6 constraints suffice to define the transformation
  - handy kind of constraint: point **p** maps to point **q** (2 constraints at once)
  - three point constraints add up to constrain all 6 DOFs (i.e. can map any triangle to any other triangle)
- 3D affine transformation has 12 degrees of freedom
  - count them by looking at the matrix entries we're allowed to change
- Therefore 12 constraints suffice to define the transformation
  - in 3D, this is 4 point constraints (i.e. can map any tetrahedron to any other tetrahedron)

Transforming normal vectors

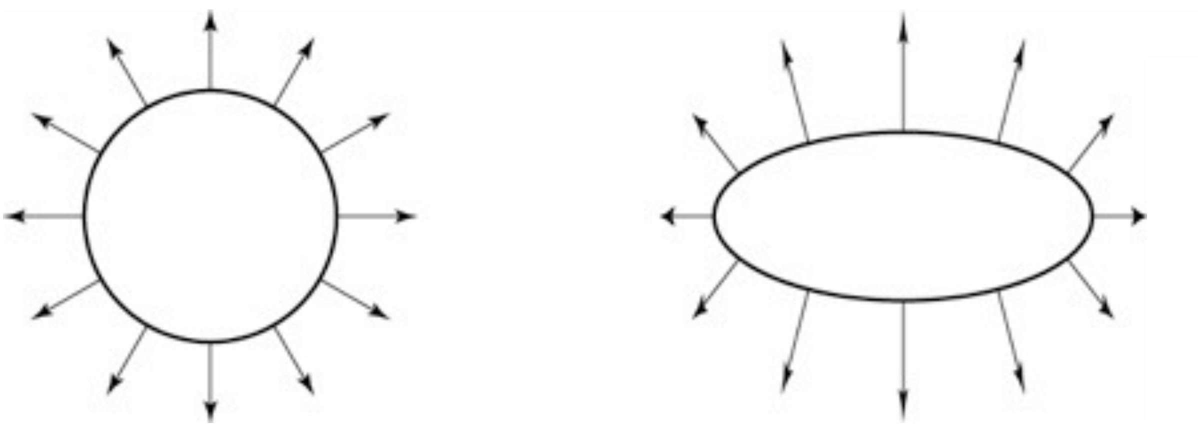
- Transforming surface normals
  - differences of points (and therefore tangents) transform OK
  - normals do not



have: $\mathbf{t} \cdot \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$
want: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T X \mathbf{n} = 0$
so set $X = (M^T)^{-1}$
then: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T (M^T)^{-1} \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$

Transforming normal vectors

- Transforming surface normals
  - differences of points (and therefore tangents) transform OK
  - normals do not



have: $\mathbf{t} \cdot \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$
want: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T X \mathbf{n} = 0$
so set $X = (M^T)^{-1}$
then: $M\mathbf{t} \cdot X\mathbf{n} = \mathbf{t}^T M^T (M^T)^{-1} \mathbf{n} = \mathbf{t}^T \mathbf{n} = 0$