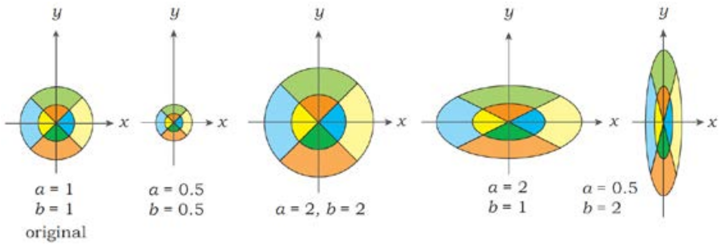
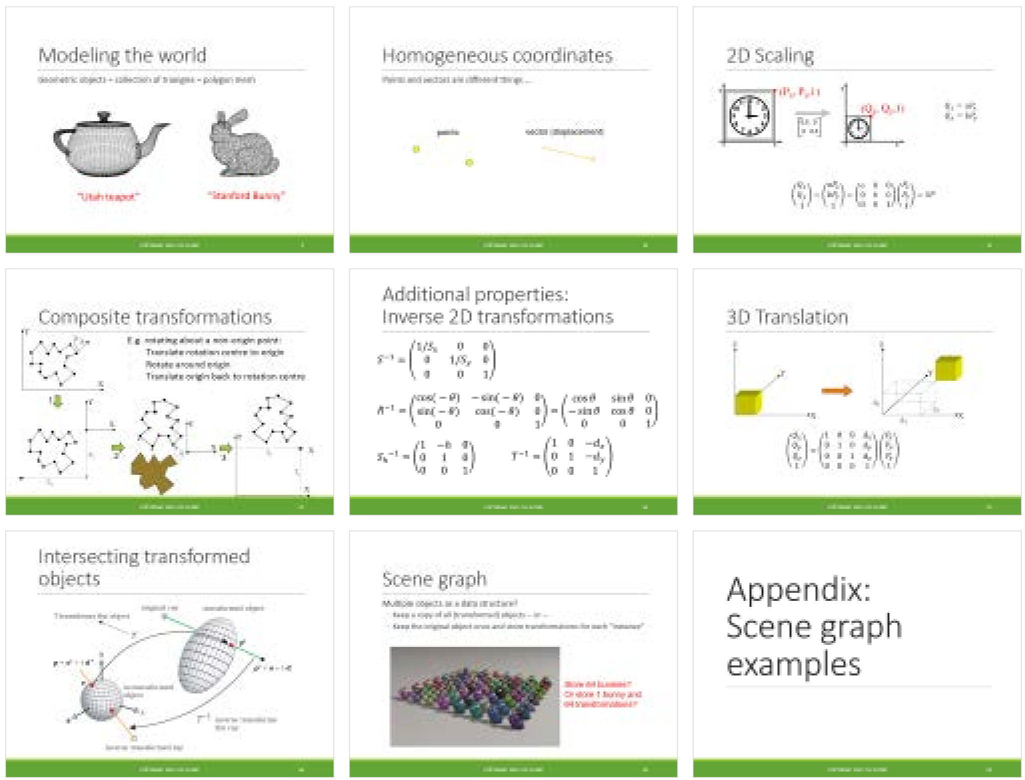
Transformations and Scene Graphs



FUNDAMENTALS OF COMPUTER GRAPHICS

PHILIP DUTRÉ
DEPARTMENT OF COMPUTER SCIENCE

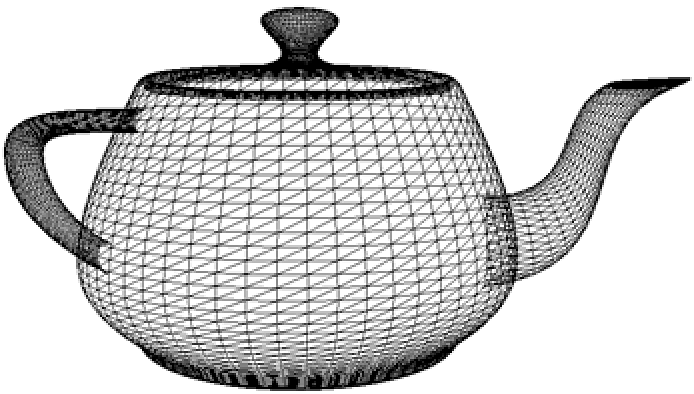
Lecture Overview



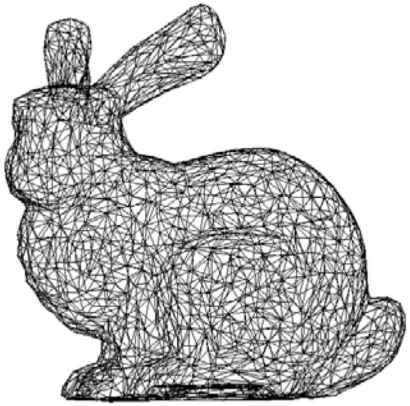
Relevant sections in book: Chapter 19, 20, 21
(Illustrations from *Ray Tracing From The Ground Up*, *Physically-Based Rendering*, *Fundamentals of Computer Graphics*)
(Page numbering might skip some slides due to 'hidden' slides in my presentation.)

Modeling the world

Geometric objects = collection of triangles = polygon mesh



"Utah teapot"



"Stanford Bunny"

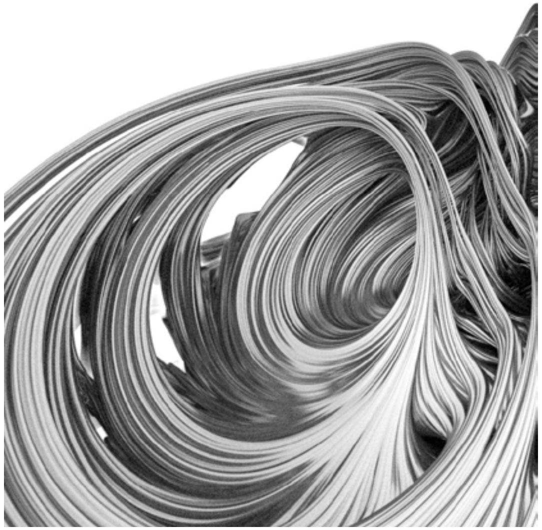
Modeling the world

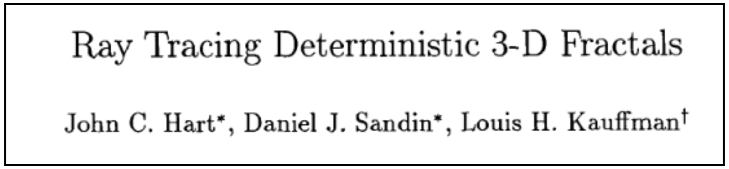
https://youtu.be/wq5Xf4s_0ZU

Modeling the world

Ray tracing is not restricted to triangles

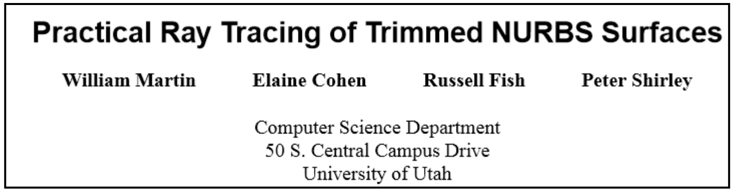
Any object … as long as we can compute an intersection with a ray (… and a normal vector)





Ray Tracing Deterministic 3-D Fractals

John C. Hart*, Daniel J. Sandin*, Louis H. Kauffman†



Practical Ray Tracing of Trimmed NURBS Surfaces

William Martin **Elaine Cohen** **Russell Fish** **Peter Shirley**

Computer Science Department
50 S. Central Campus Drive
University of Utah

Ray Tracing JELL-O® Brand Gelatin

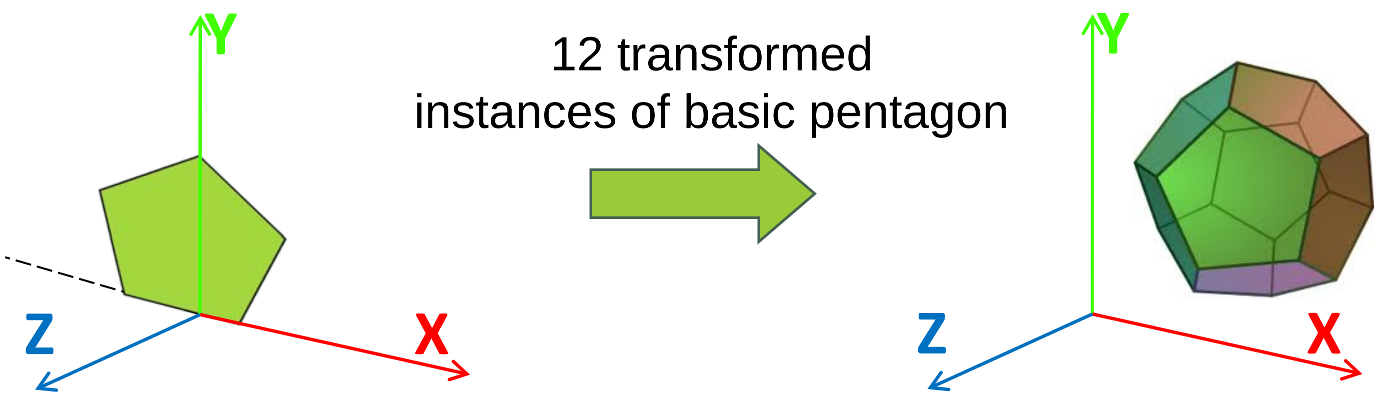
Paul S. Heckbert

Dessert Foods Division
Pixar
San Rafael, CA

Datastructures for modeling

Define objects only once

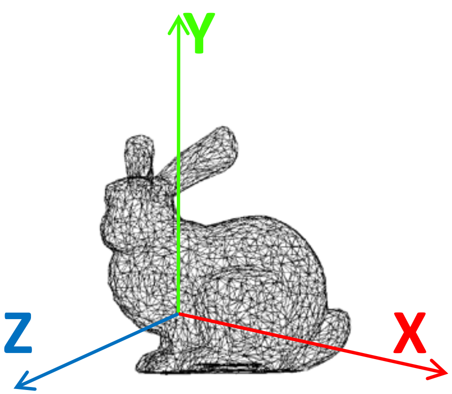
- … then transform to compose bigger objects
- Use translations, rotations, scaling, …



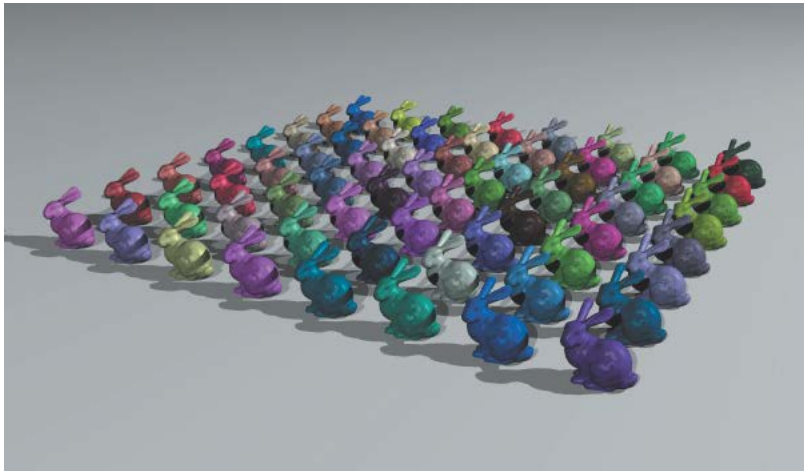
Datastructures for modeling

Define objects only once

- … then transform to compose bigger scenes
- Use translations, rotations, scaling, …



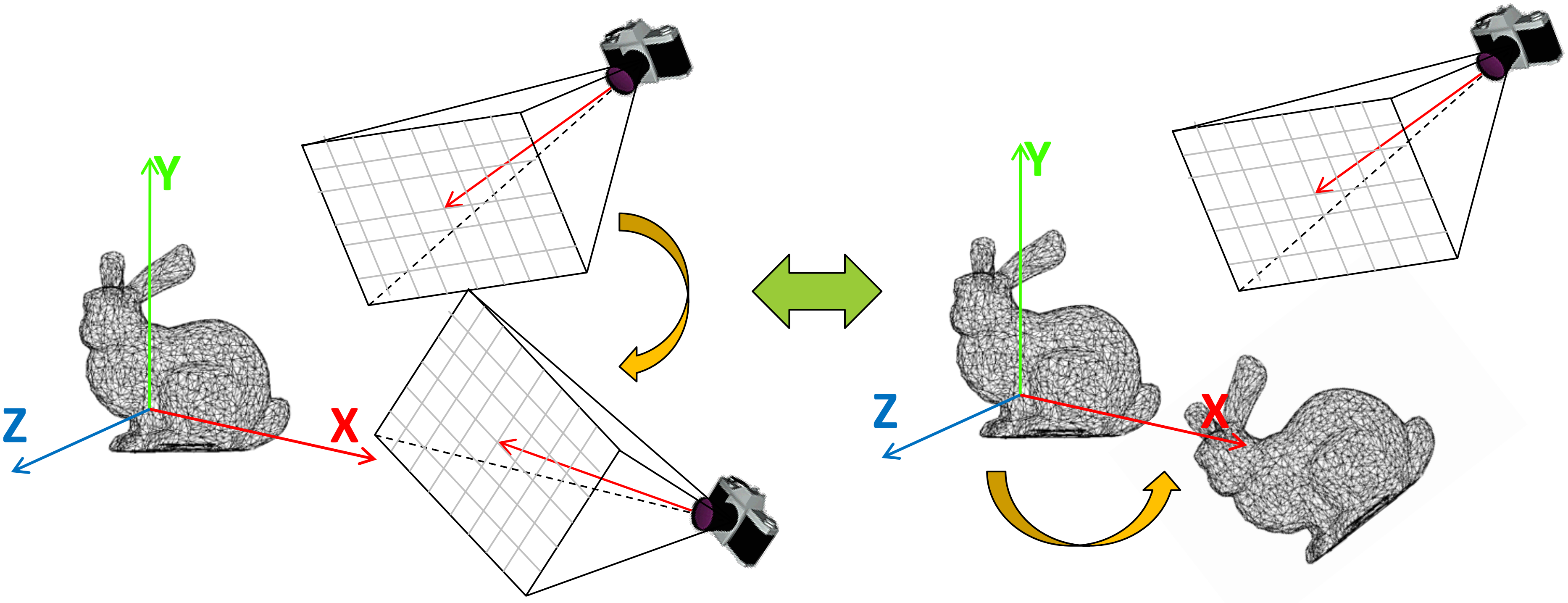
64 transformed instances of bunny



Why are transformations (also) useful?

Move the camera in the 3D scene

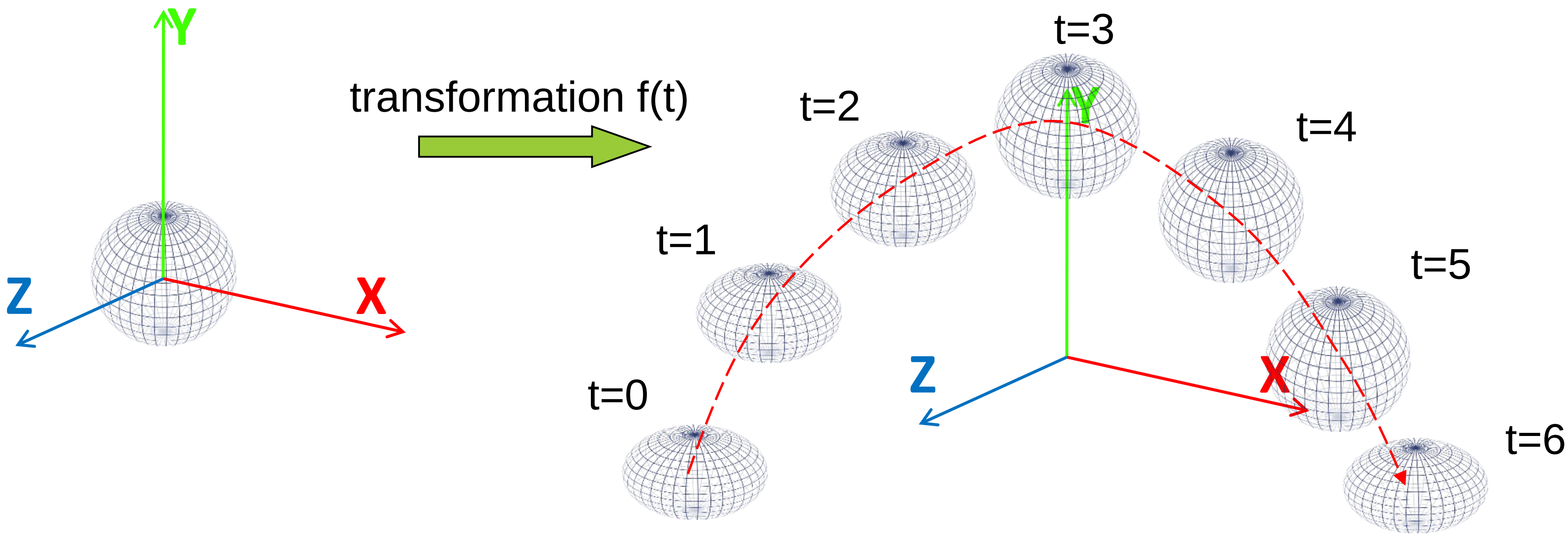
- … equivalent to keeping camera fixed, and apply inverse transformation to the 3D scene



Why are transformations (also) useful?

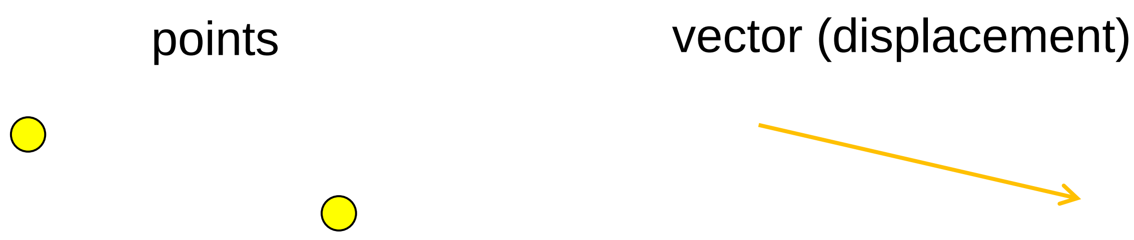
Computer animation

- Translate / rotate / warp object over time
- Transformations can be time-dependent



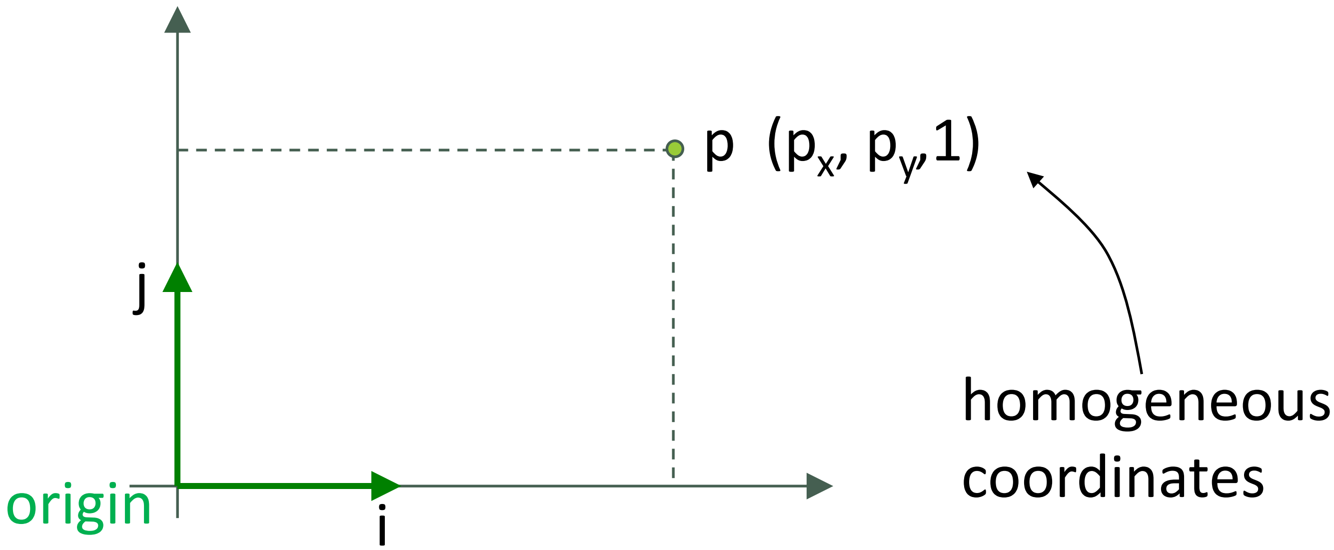
Homogeneous coordinates

Points and vectors are different things …



Homogeneous coordinates

Each point can be expressed in a coordinate system: **p** = p_x**i** + p_y**j** + **origin**



Homogeneous coordinates

points: (x y z 1)

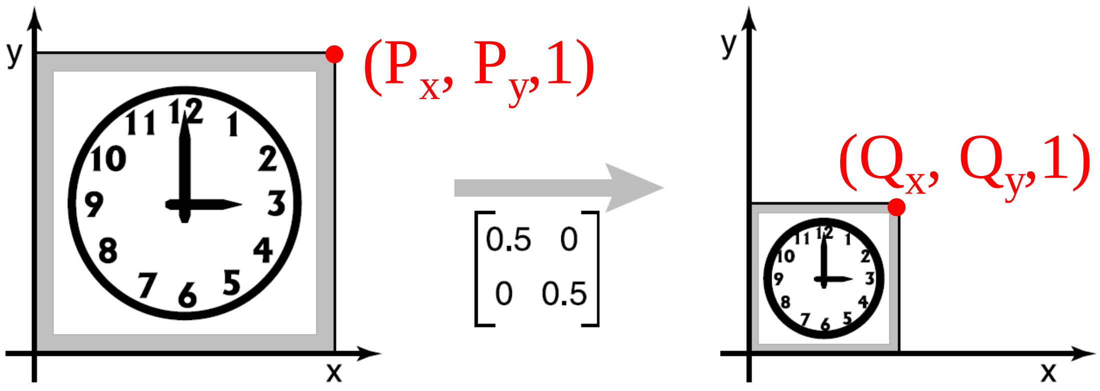
vectors: (x y z 0), “points at infinity”

subtraction: (* * * 1) – (* * * 1) = (* * * 0)

addition: (* * * 1) + (* * * 0) = (* * * 1)

(x y z w) is the same point as (x/w y/w z/w 1)

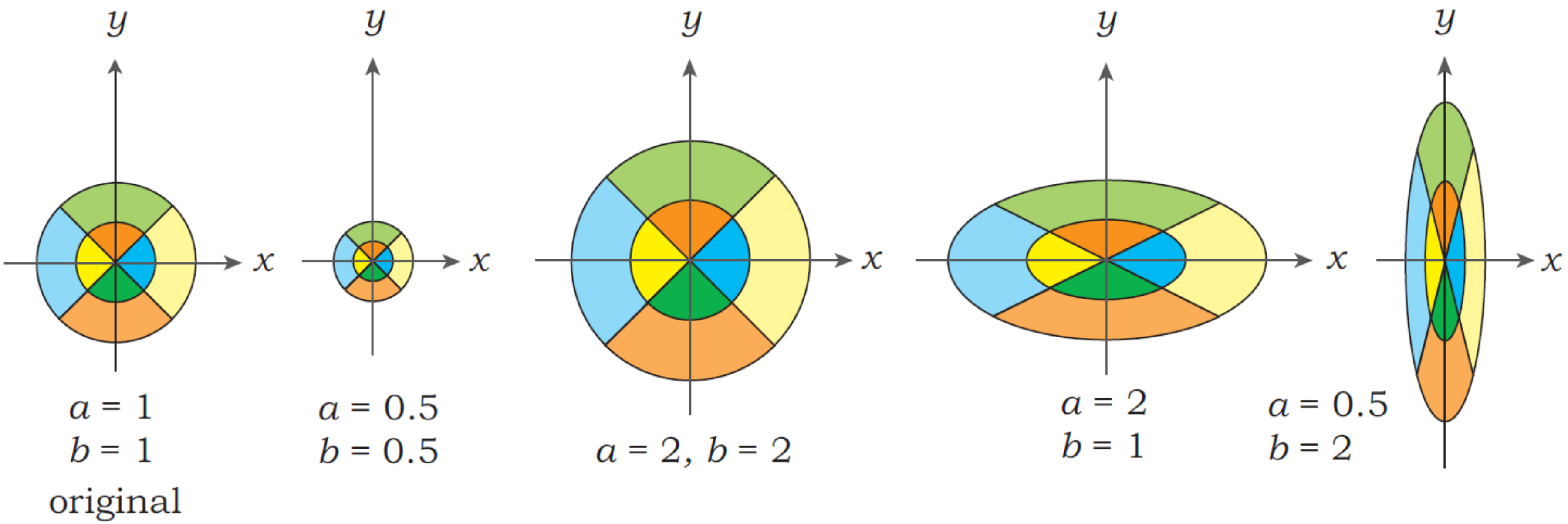
2D Scaling



$$Q_x = aP_x$$
$$Q_y = bP_y$$

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} aP_x \\ bP_y \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = SP$$

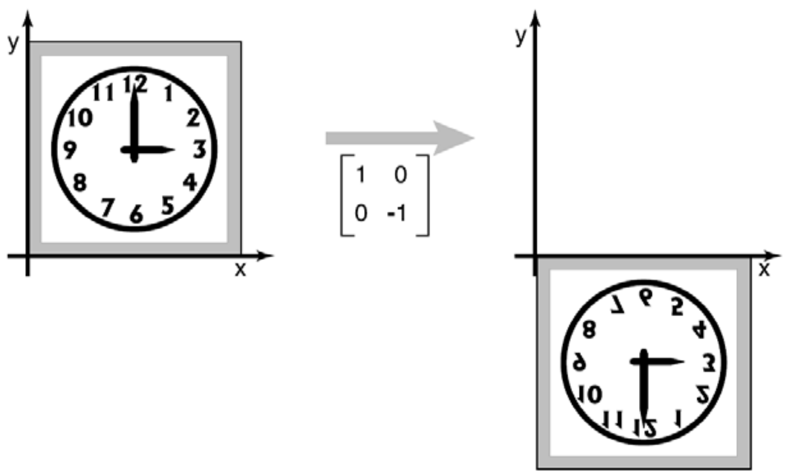
2D Scaling

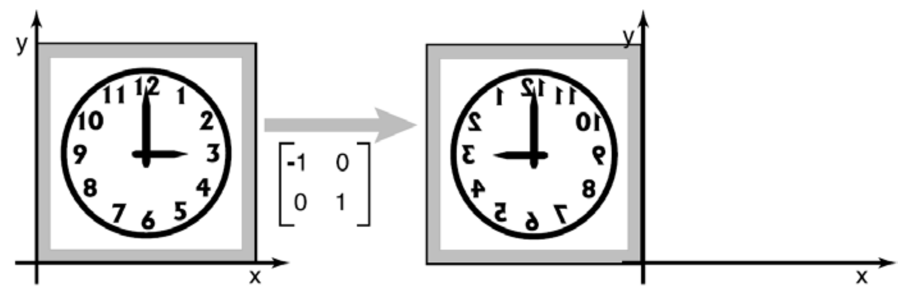


$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} aP_x \\ bP_y \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = SP$$

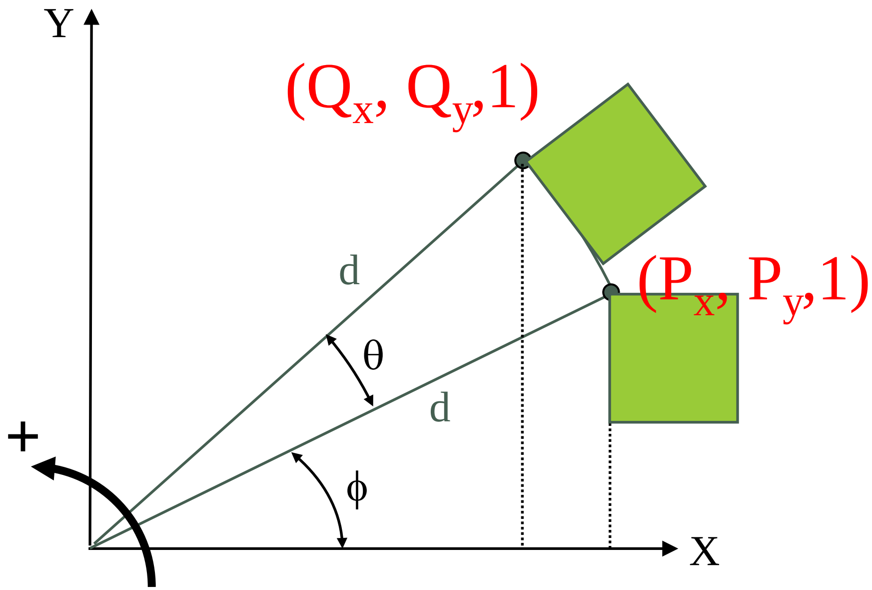
2D Reflection

Special case of scaling





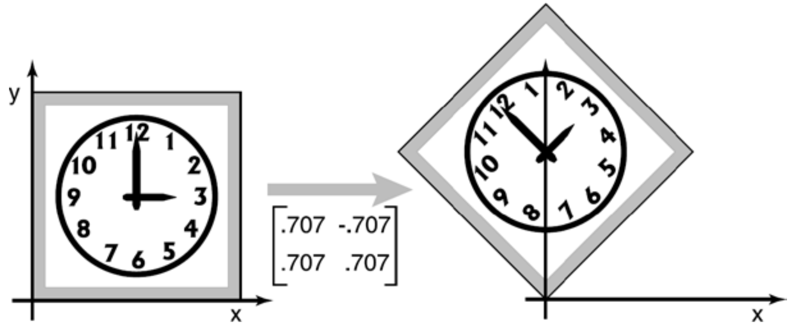
2D Rotation



$$P_x = d\cos\varphi$$
$$P_y = d\sin\varphi$$

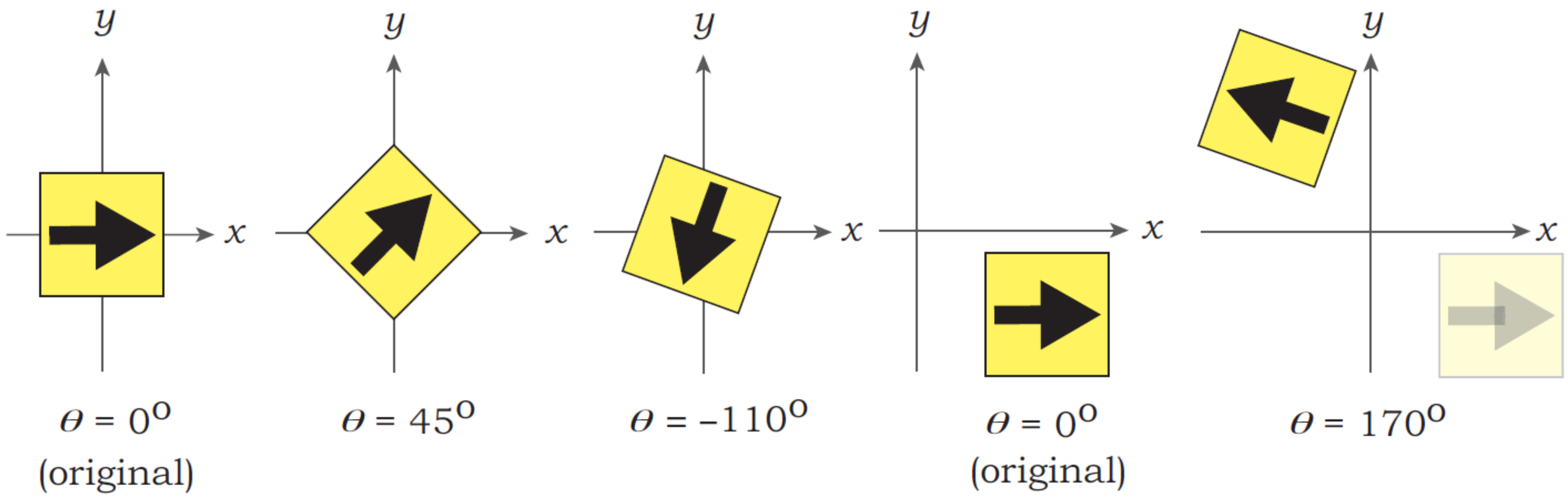
$$Q_x = d\cos(\varphi+\theta)$$
$$Q_y = d\sin(\varphi+\theta)$$

$$\cos(\varphi+\theta) = \cos\varphi\cos\theta - \sin\varphi\sin\theta$$
$$\sin(\varphi+\theta) = \sin\varphi\cos\theta + \cos\varphi\sin\theta$$

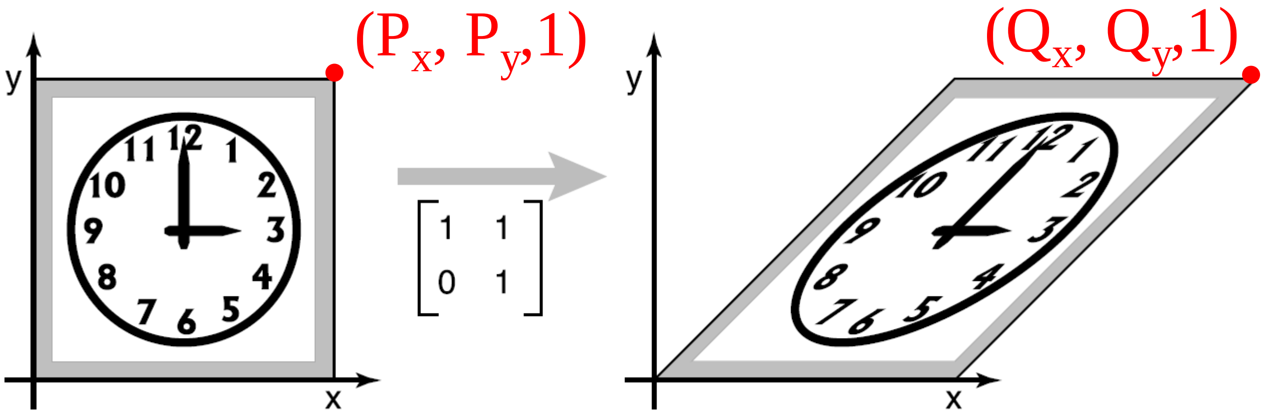


$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = RP$$

2D Rotation


y
x
$\theta = 0^o$
(original)
$\theta = 45^o$
$\theta = -110^o$
$\theta = 0^o$
(original)
$\theta = 170^o$

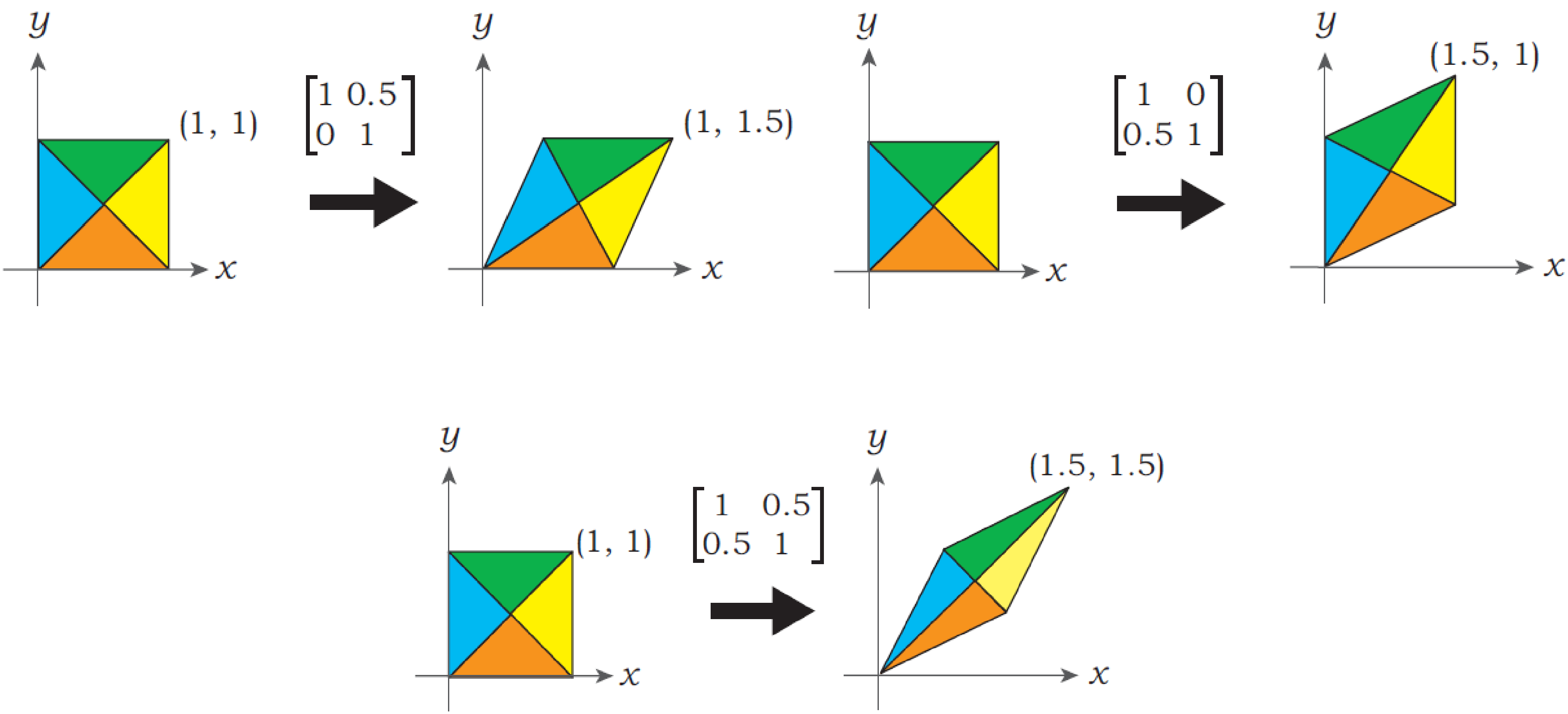
2D Shear



$$Q_x = P_x + hP_y$$
$$Q_y = P_y$$

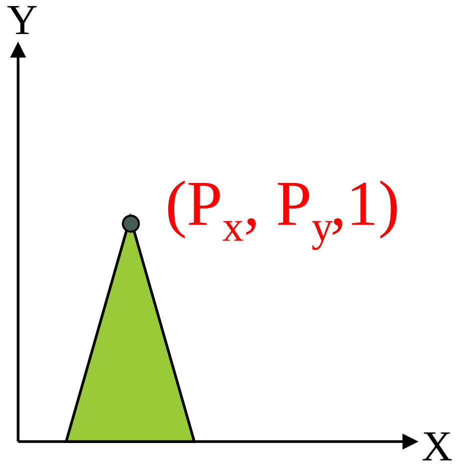
$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} P_x + hP_y \\ P_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = S_h P$$

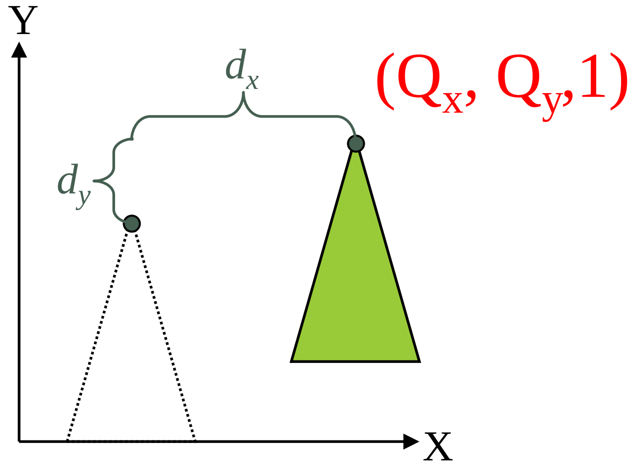
2D Shear



y
x
(1, 1)
$\begin{bmatrix} 1 & 0.5 \\ 0 & 1 \end{bmatrix}$
(1, 1.5)
$\begin{bmatrix} 1 & 0 \\ 0.5 & 1 \end{bmatrix}$
(1.5, 1)
$\begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$
(1.5, 1.5)

2D translation

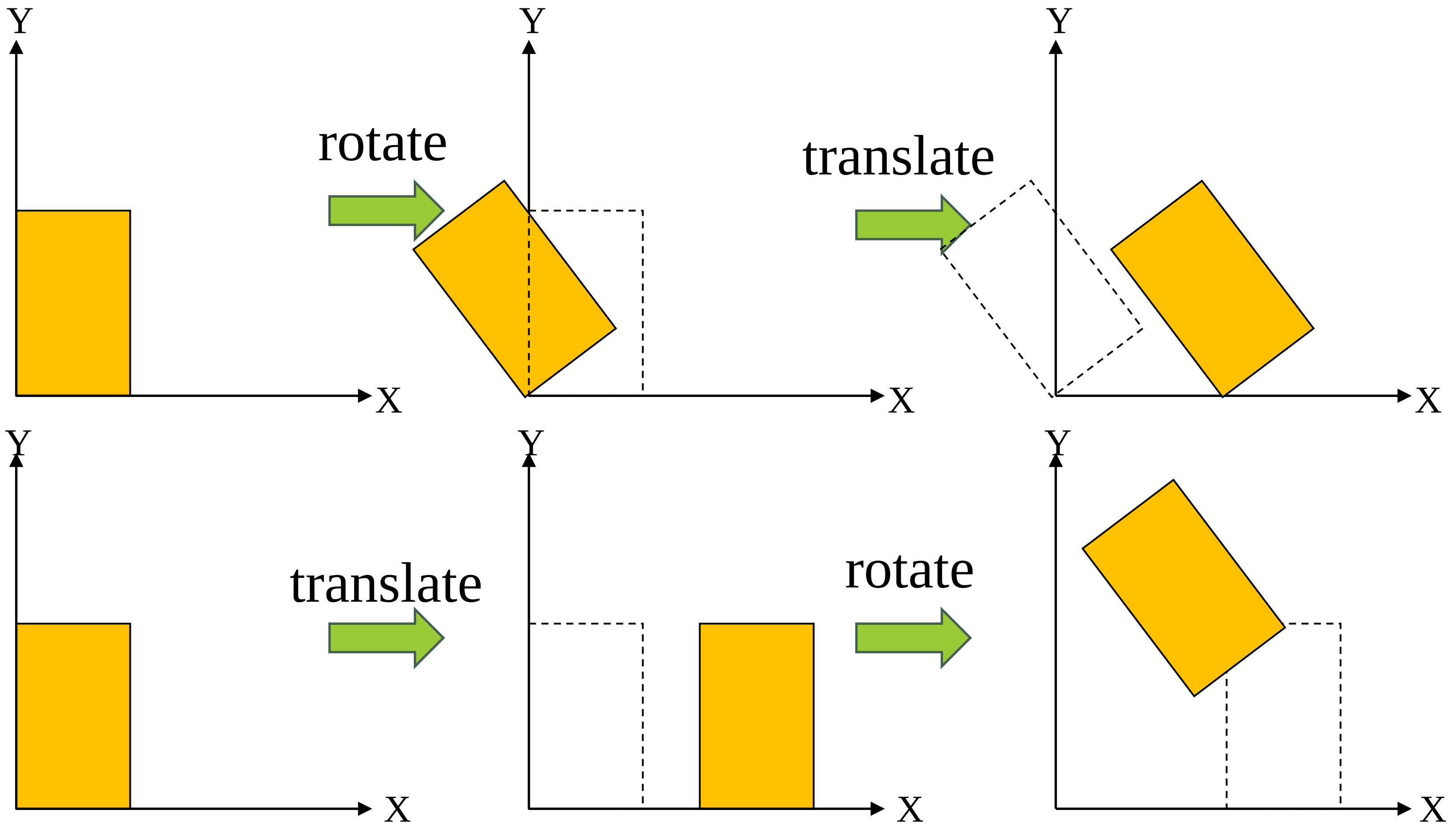




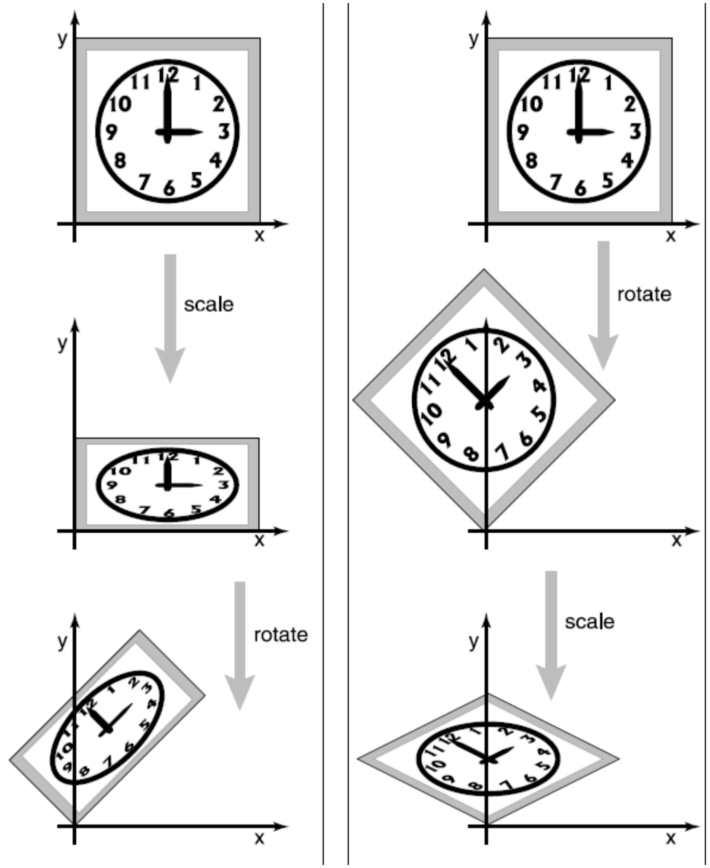
$$Q_x = P_x + d_x$$
$$Q_y = P_y + d_y$$

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} P_x + d_x \\ P_y + d_y \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = TP$$

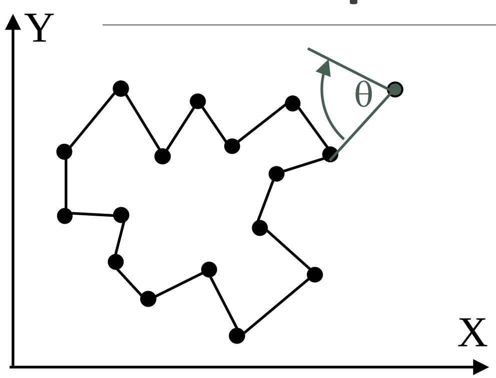
Order of transformations



Order of transformations

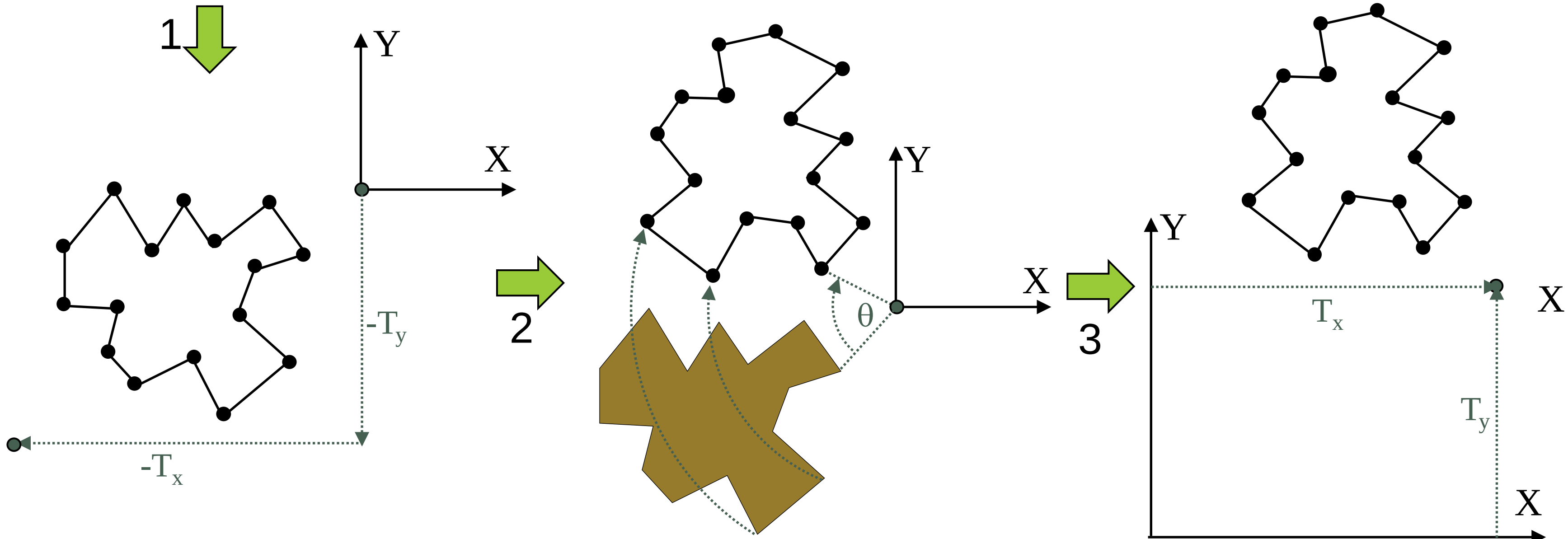


Composite transformations



E.g. rotating about a non-origin point:

1. Translate rotation centre to origin
2. Rotate around origin
3. Translate origin back to rotation centre

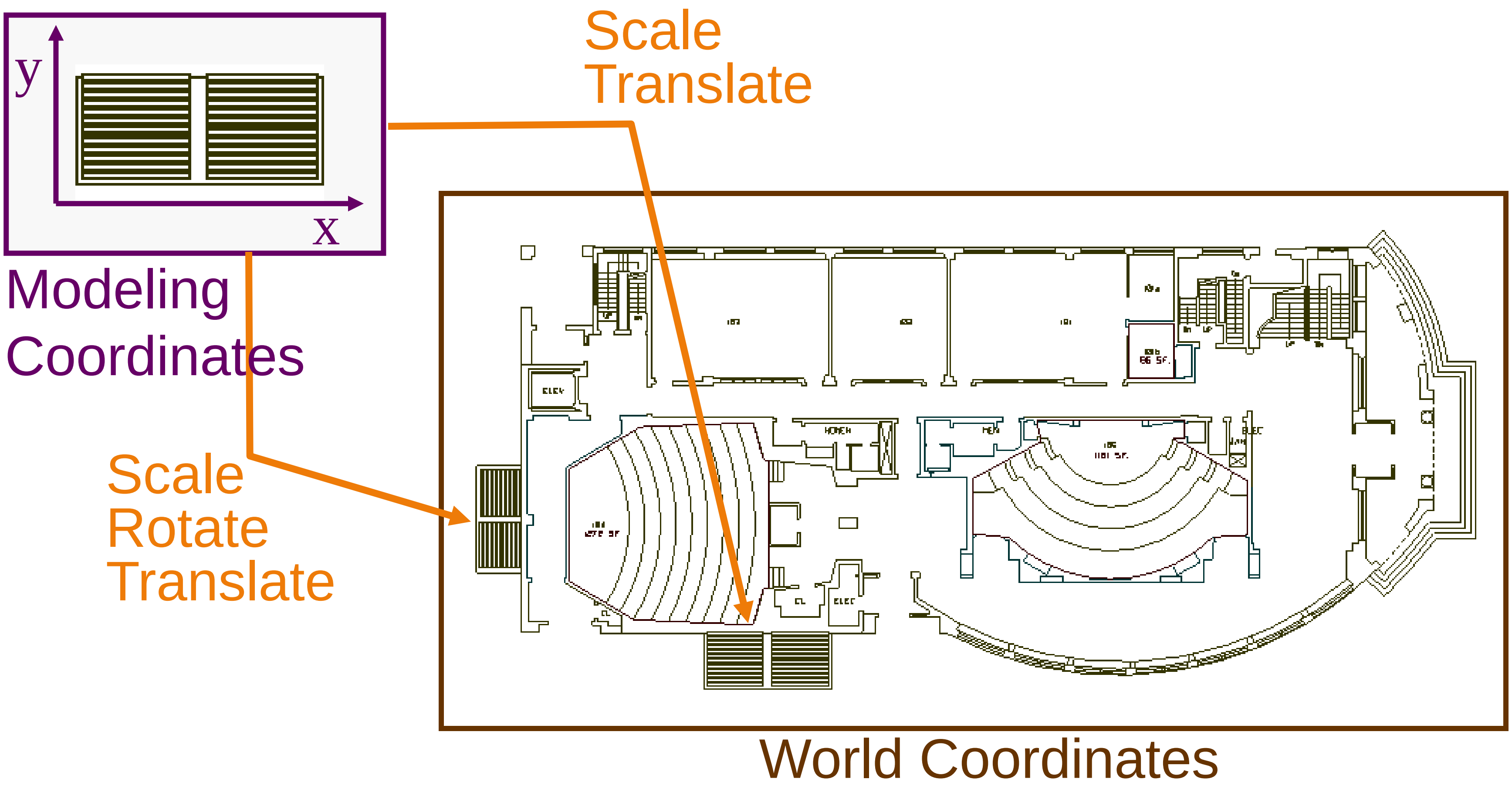


Composite transformations

$$Q = \begin{pmatrix} 1 & 0 & T_x \\ 0 & 1 & T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -T_x \\ 0 & 1 & -T_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = T^{-1}RTP$$

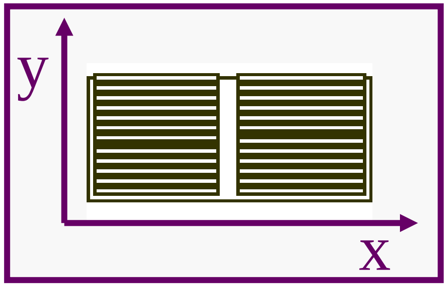
$$T^{-1}RT = \begin{pmatrix} \cos\theta & -\sin\theta & -\cos\theta\, T_x + \sin\theta\, T_y + T_x \\ \sin\theta & \cos\theta & -\sin\theta\, T_x - \cos\theta\, T_y + T_y \\ 0 & 0 & 1 \end{pmatrix}$$

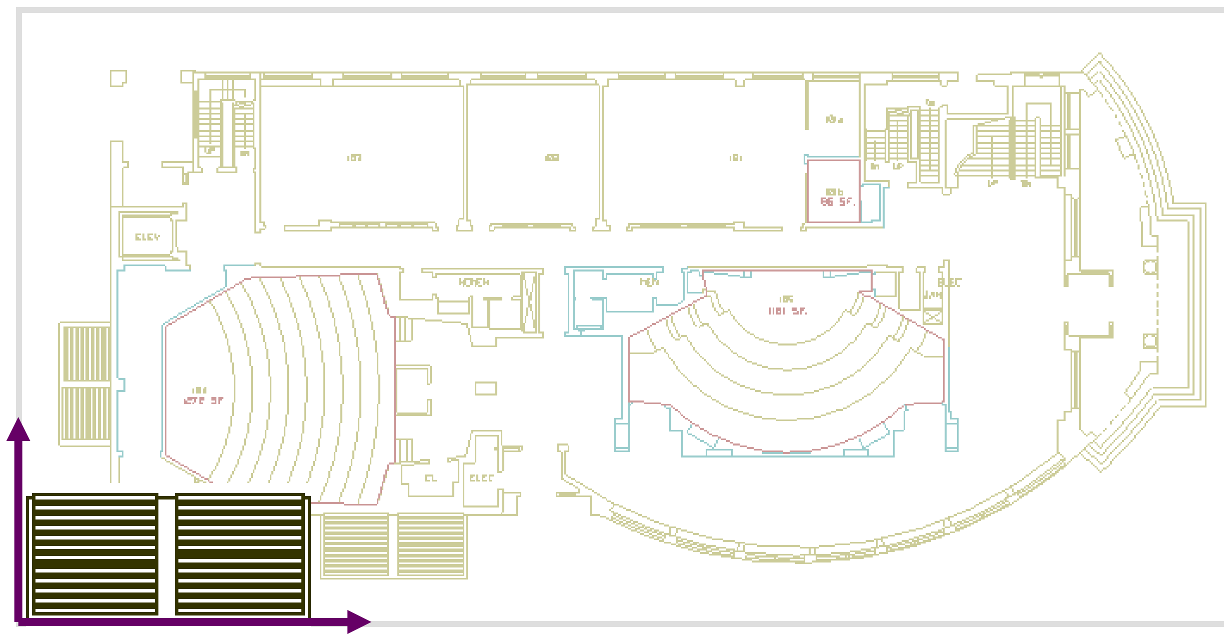
Example



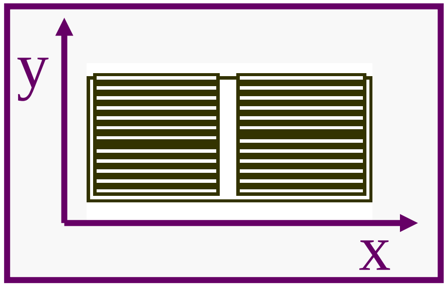
y
x
Modeling Coordinates
Scale
Translate
Scale
Rotate
Translate
World Coordinates

Example

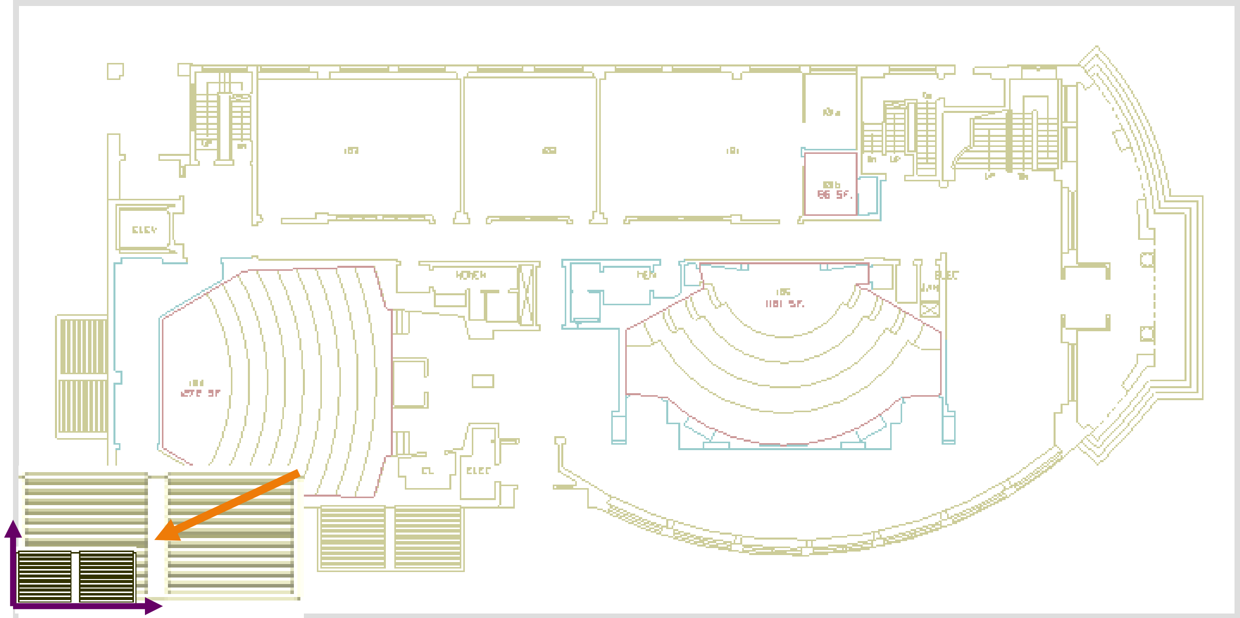




Example

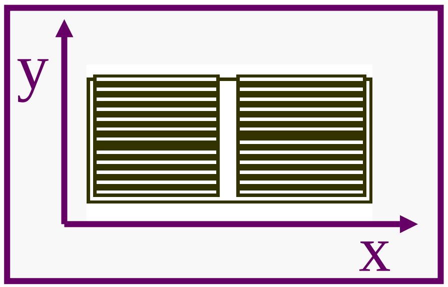


$x' = x * sx$
$y' = y * sy$

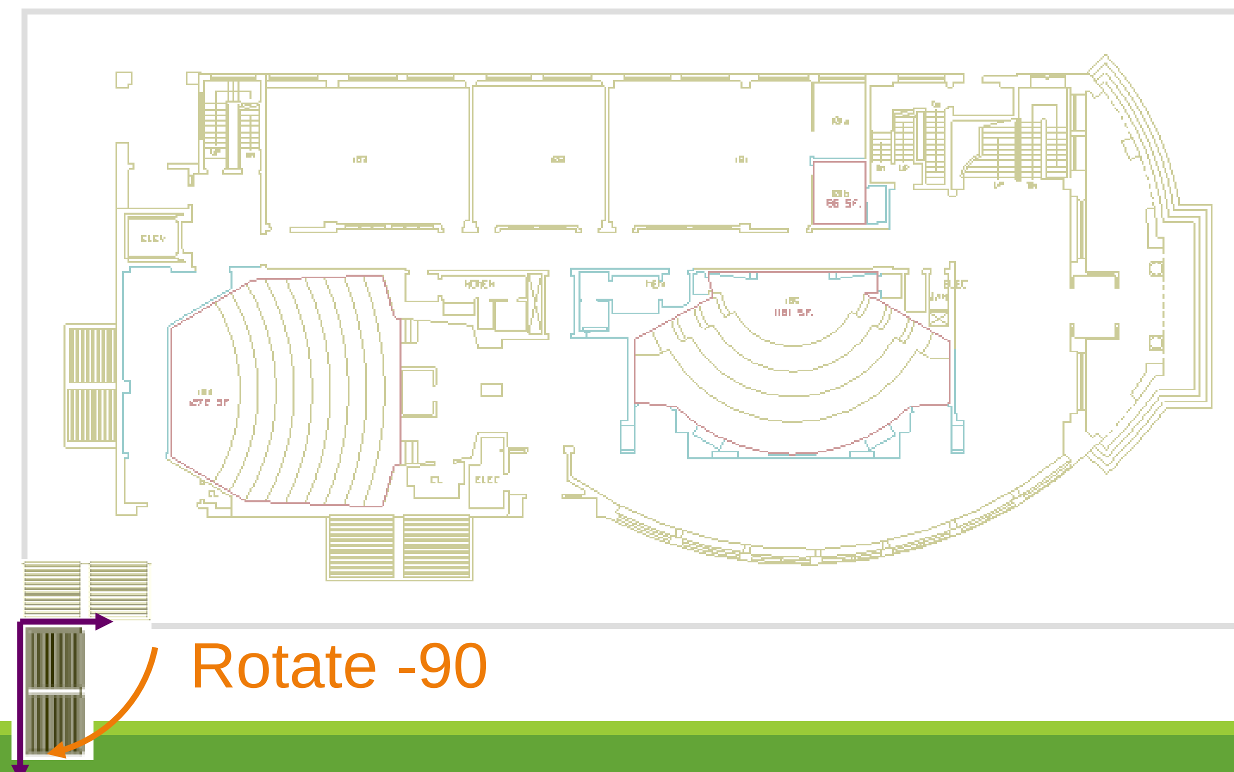


Scale .3,.3

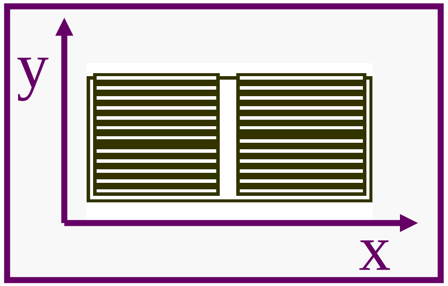
Example



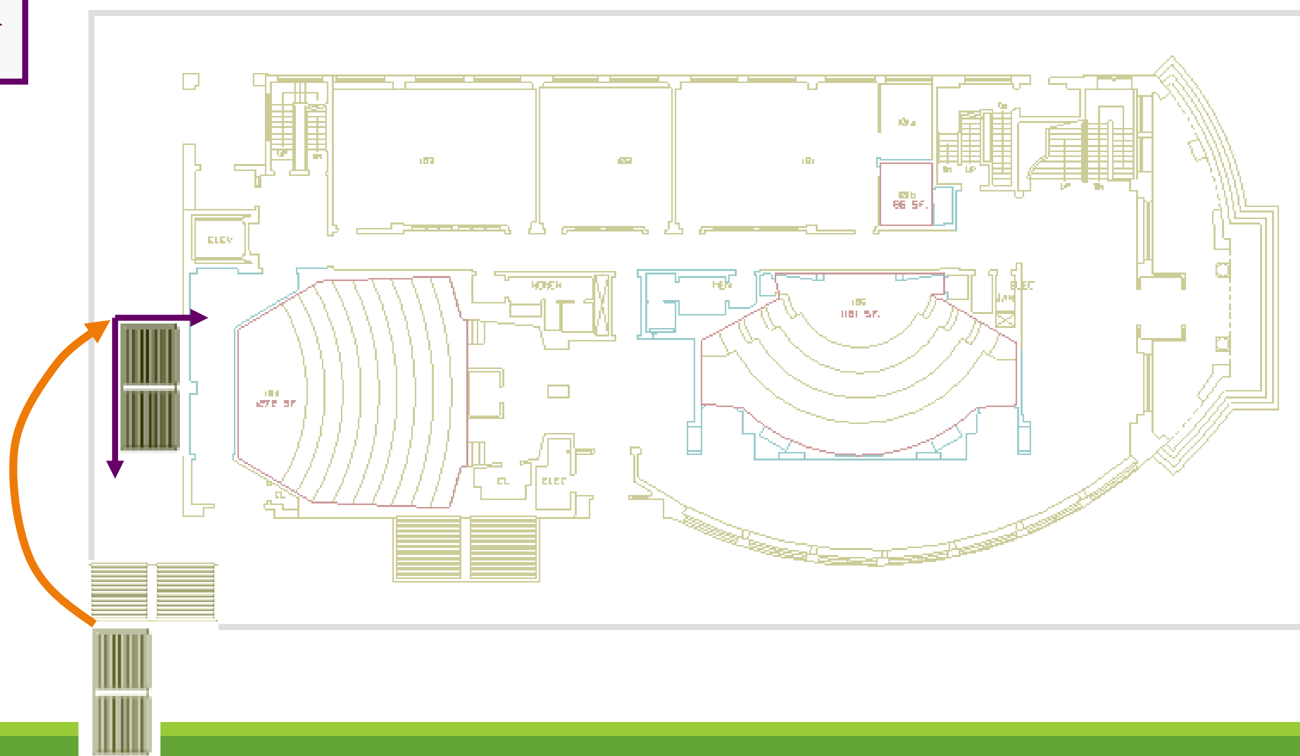
$$x' = (x*sx)*\cos\Theta - (y*sy)*\sin\Theta$$
$$y' = (x*sx)*\sin\Theta + (y*sy)*\cos\Theta$$



Example



$$x' = ((x*sx)*\cos\Theta - (y*sy)*\sin\Theta) + tx$$
$$y' = ((x*sx)*\sin\Theta + (y*sy)*\cos\Theta) + ty$$



Translate 3, 5

Additional properties: Inverse 2D transformations

$$S^{-1} = \begin{pmatrix} 1/S_x & 0 & 0 \\ 0 & 1/S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R^{-1} = \begin{pmatrix} \cos(-\theta) & -\sin(-\theta) & 0 \\ \sin(-\theta) & \cos(-\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

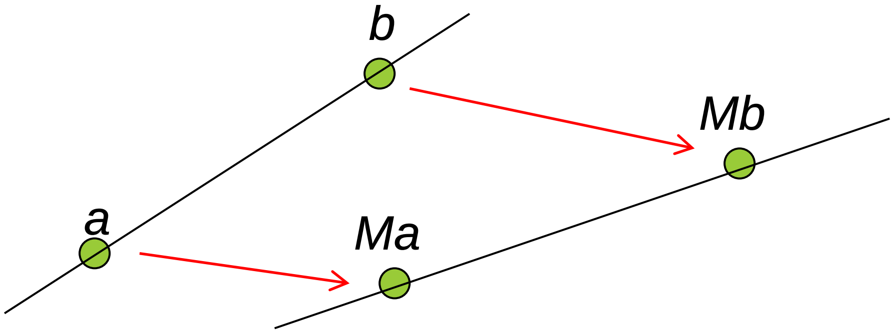
$$S_h{}^{-1} = \begin{pmatrix} 1 & -h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad T^{-1} = \begin{pmatrix} 1 & 0 & -d_x \\ 0 & 1 & -d_y \\ 0 & 0 & 1 \end{pmatrix}$$

Additional properties: Affine Transformations

Coordinates of Q are a linear combination of coordinates of P

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11}P_x + m_{12}P_y + m_{13} \\ m_{21}P_x + m_{22}P_y + m_{23} \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix} = MP$$

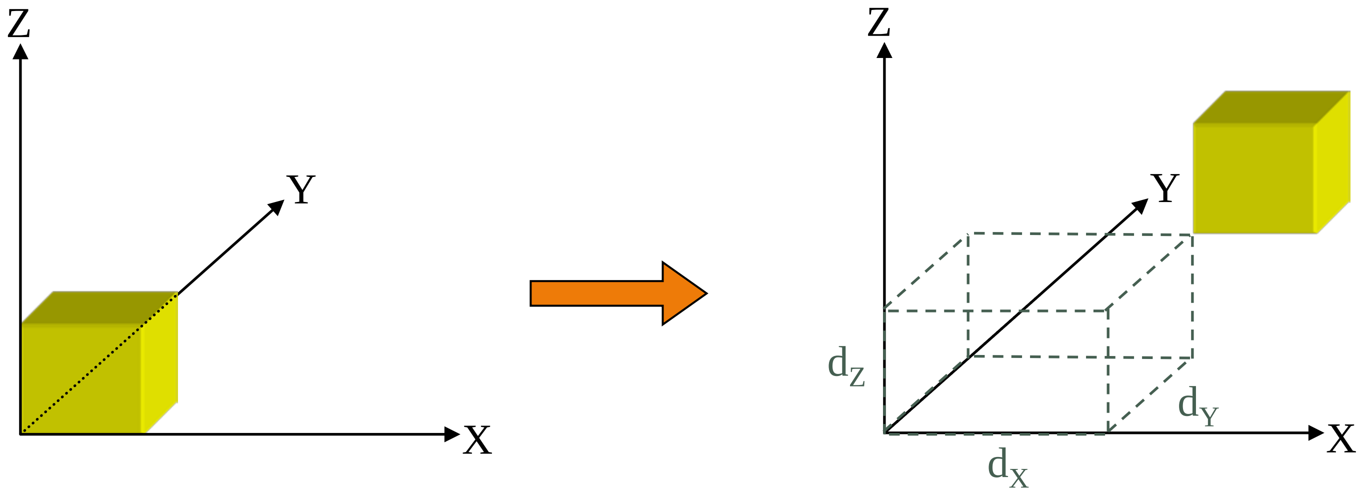
Lines remain lines:



$Mp = Ma + t(Mb - Ma)$ is the transformed line ➔ transforming vertices is sufficient for transforming a triangle

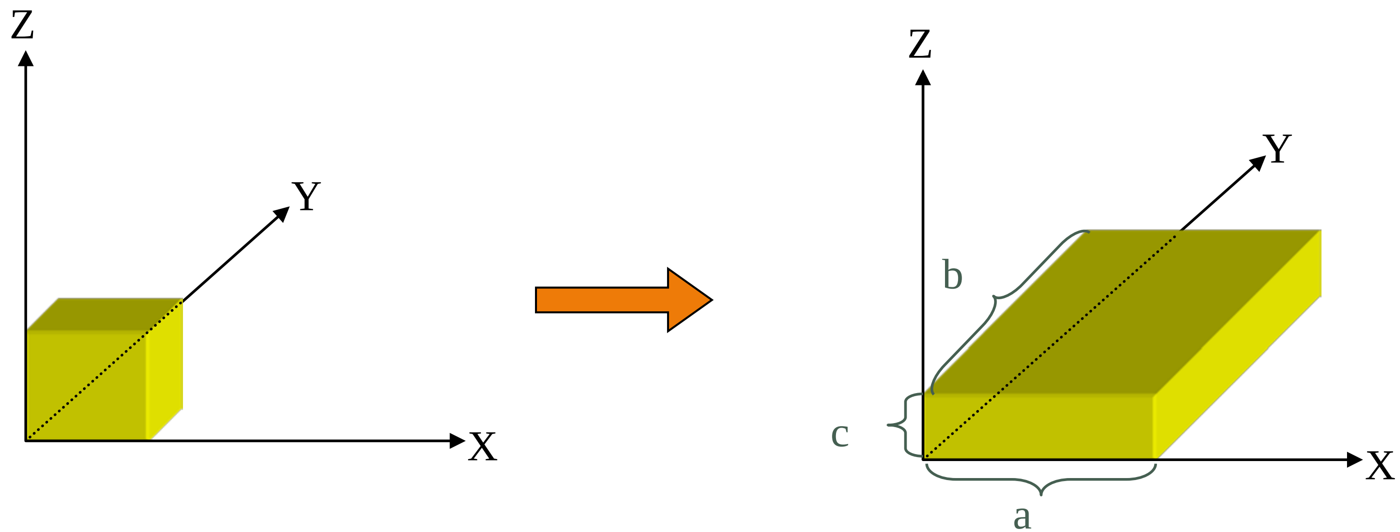
$$Mp = \begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} a_x + t(b_x - a_x) \\ a_y + t(b_y - a_y) \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11}a_x + m_{12}a_y + t(m_{11}(b_x - a_x) + m_{12}(b_y - a_y)) + m_{13} \\ m_{21}a_x + m_{22}a_y + t(m_{21}(b_x - a_x) + m_{22}(b_y - a_y)) + m_{23} \\ 1 \end{pmatrix}$$

3D Translation



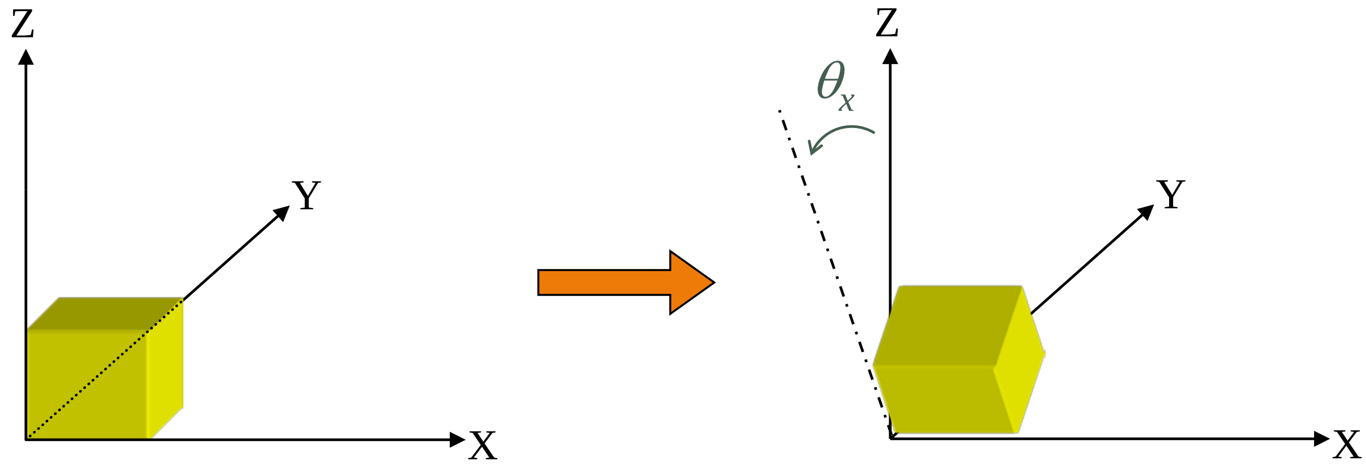
$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3D Scaling



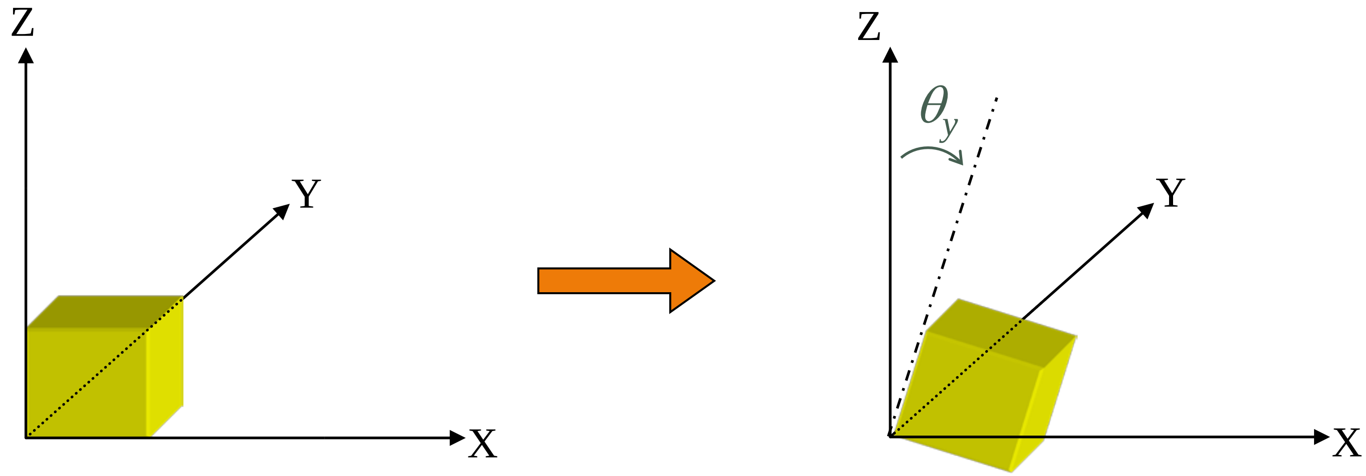
$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3D Rotation around X-axis



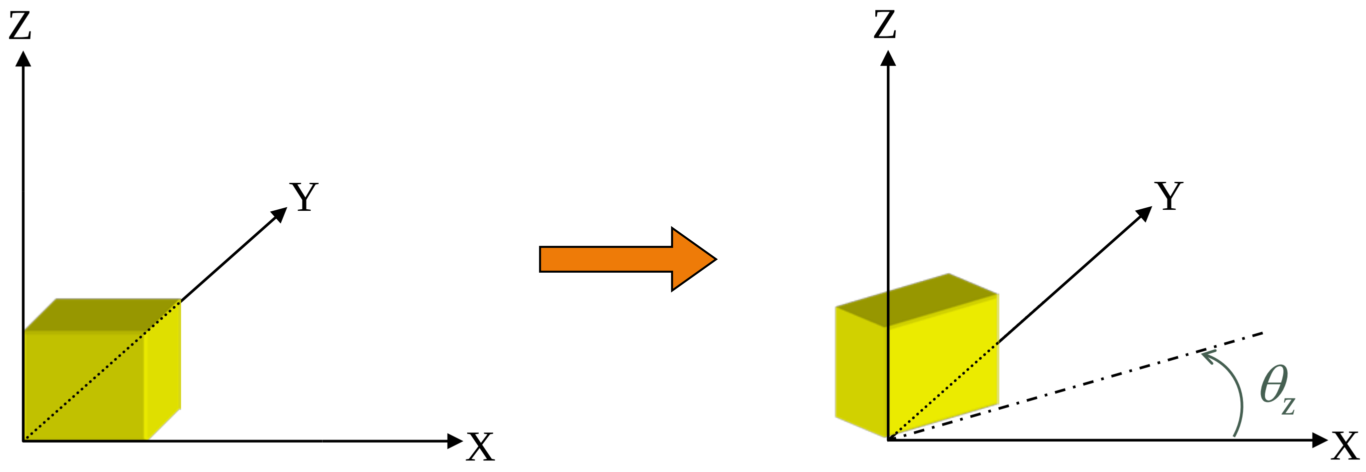
$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3D Rotation around Y-axis



$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_x & 0 & \sin\theta_x & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_x & 0 & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3D Rotation around Z-axis

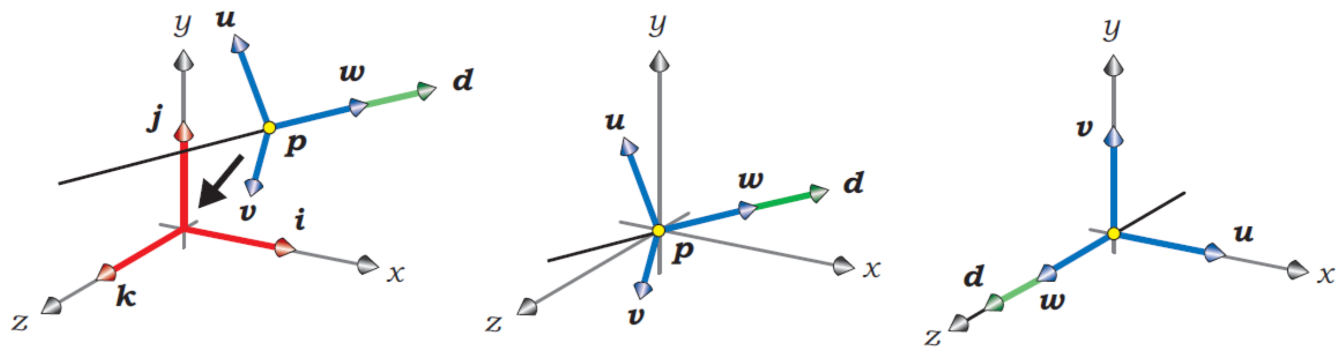


$$\begin{pmatrix} Q_x \\ Q_y \\ Q_z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta_x & -\sin\theta_x & 0 & 0 \\ \sin\theta_x & \cos\theta_x & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}$$

3D Rotation around arbitrary axis

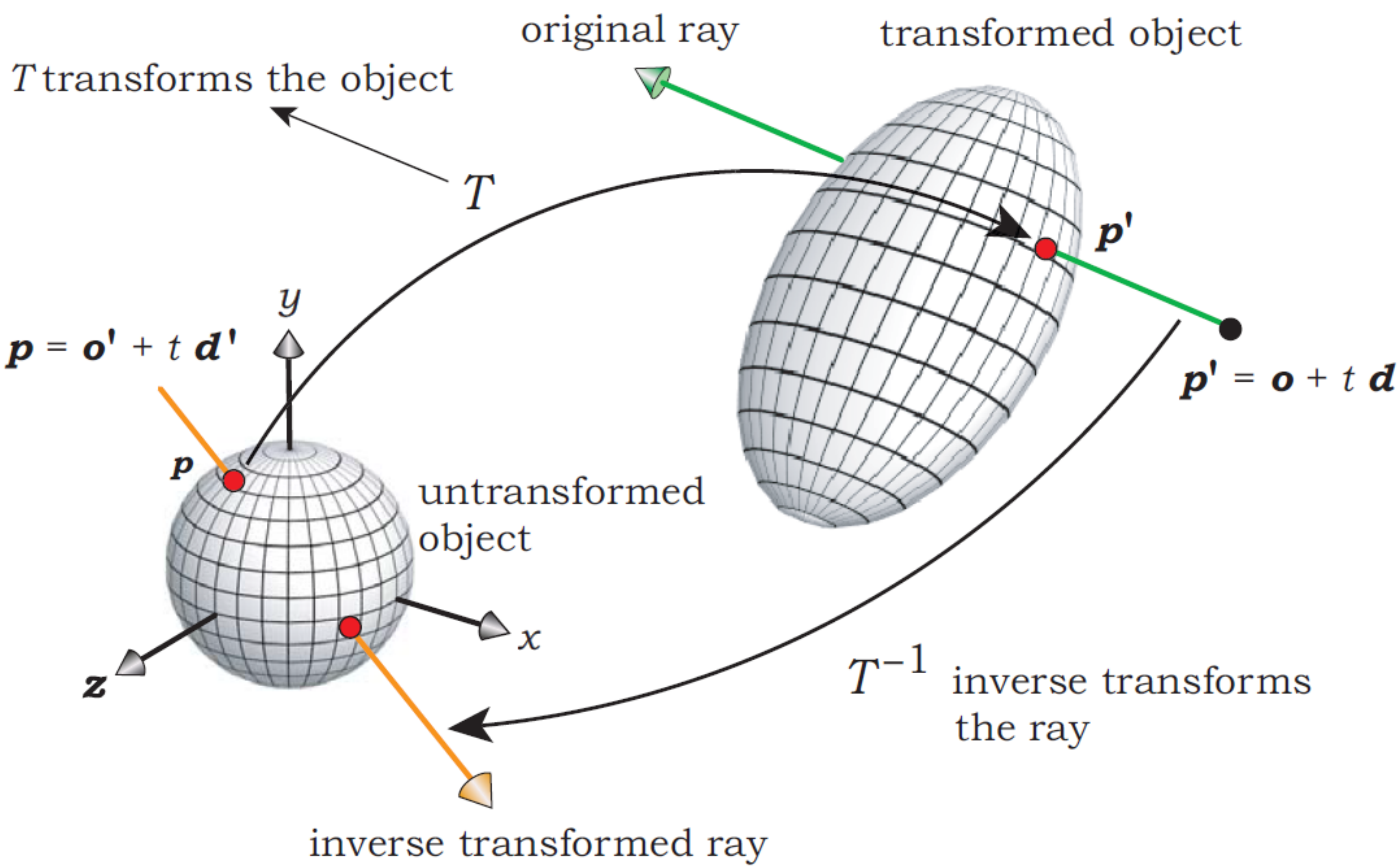
Composite transformation:

- Place axis aligned with coordinate axis
  - (translation + 2 rotations)
- Rotate around coordinate axis
- Inverse transformation
  - (2 inverse rotations + inverse translation)

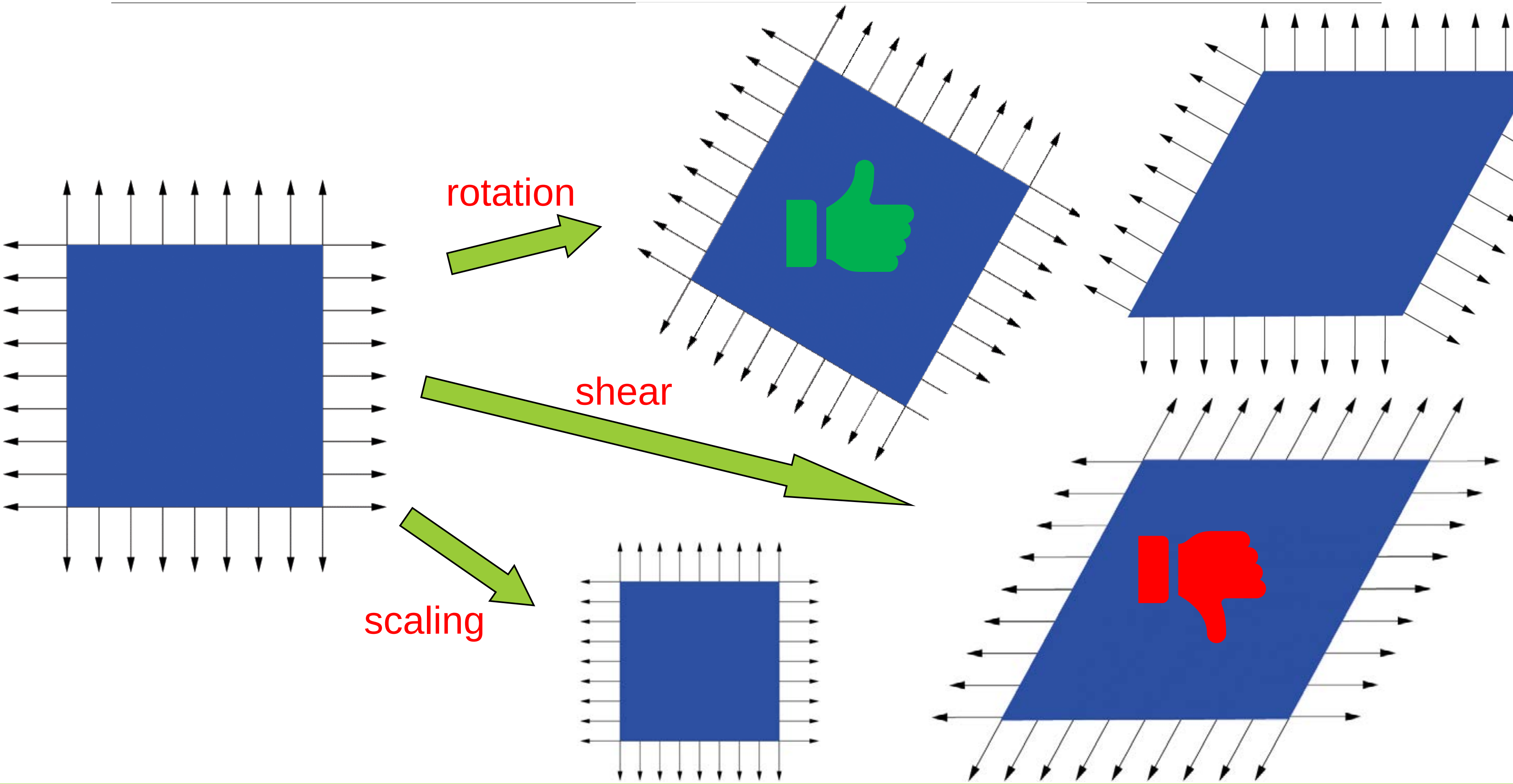


(a) (b) (c)

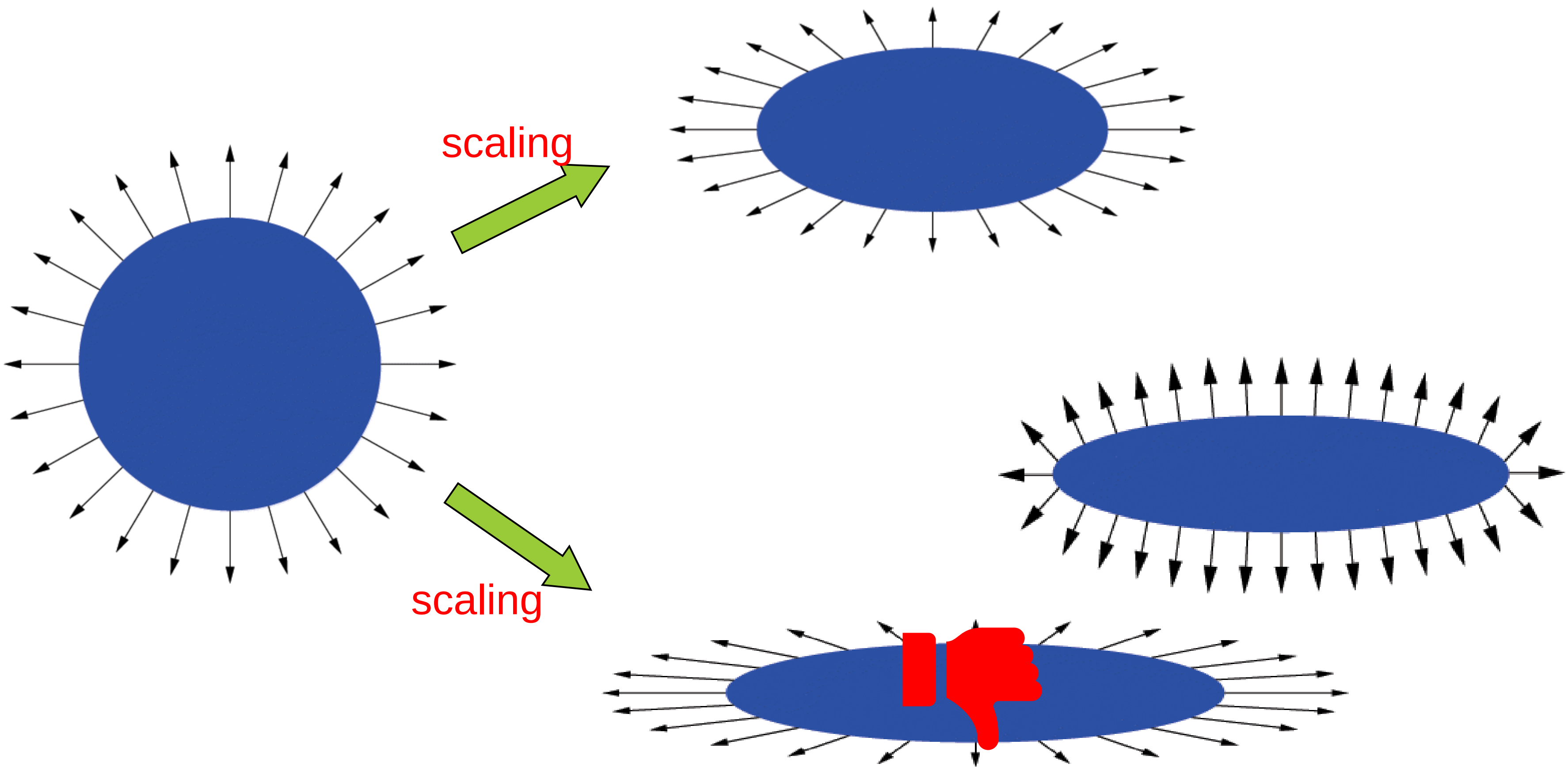
Intersecting transformed objects



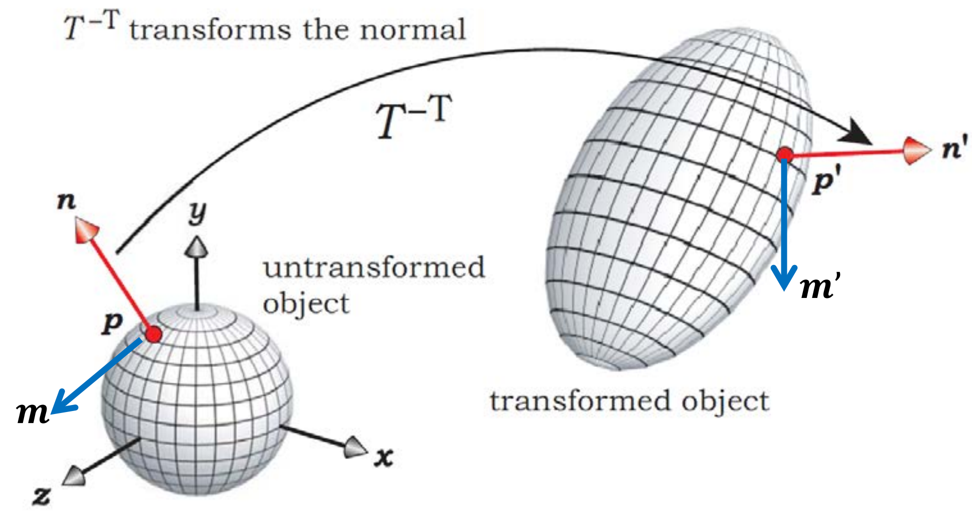
Applying transformations to normal vectors



Applying transformations to normal vectors



Transformation of normal vectors



$$n^T m = 0$$
$$n^T \mathrm{Im} = 0$$
$$n^T T^{-1} T m = 0$$
$$(n^T T^{-1})(Tm) = 0$$

We know: $n'^T (Tm) = 0$

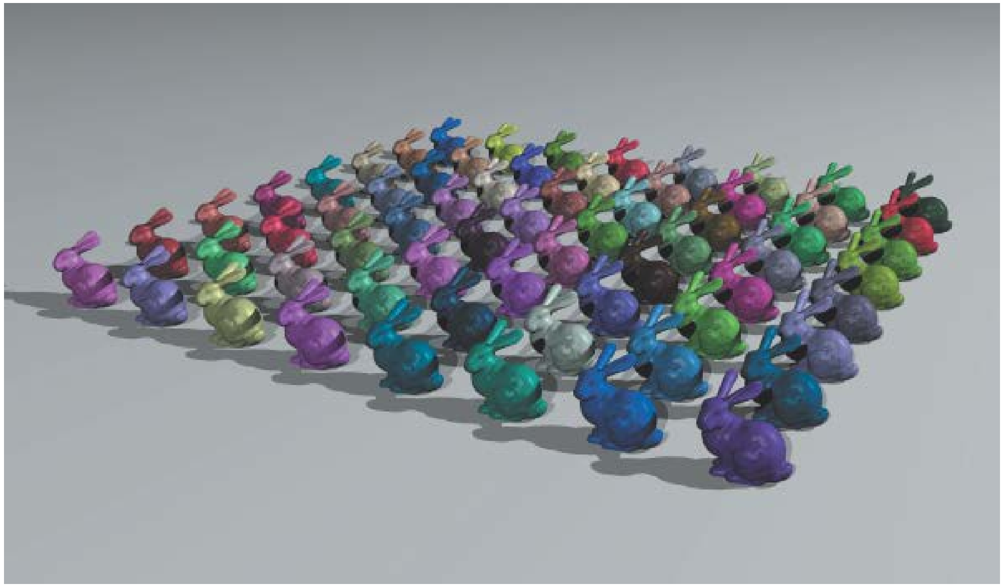
So: $n'^T = (n^T T^{-1})$

$$n' = (n^T T^{-1})^T$$
$$n' = (T^{-1})^T n$$

Scene graph

Multiple objects as a data structure?

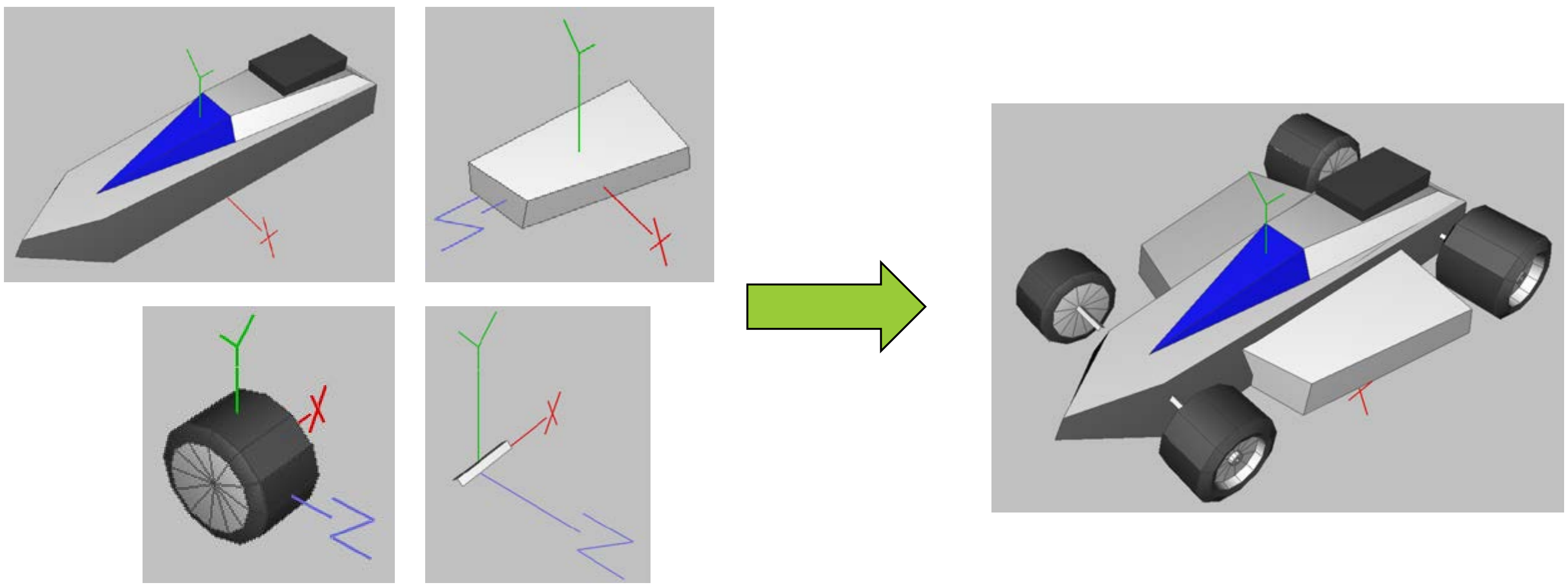
- Keep a copy of all (transformed) objects -- or --
- Keep the original object once and store transformations for each “instance”



Store 64 bunnies?
Or store 1 bunny and
64 transformations?

Scene graph

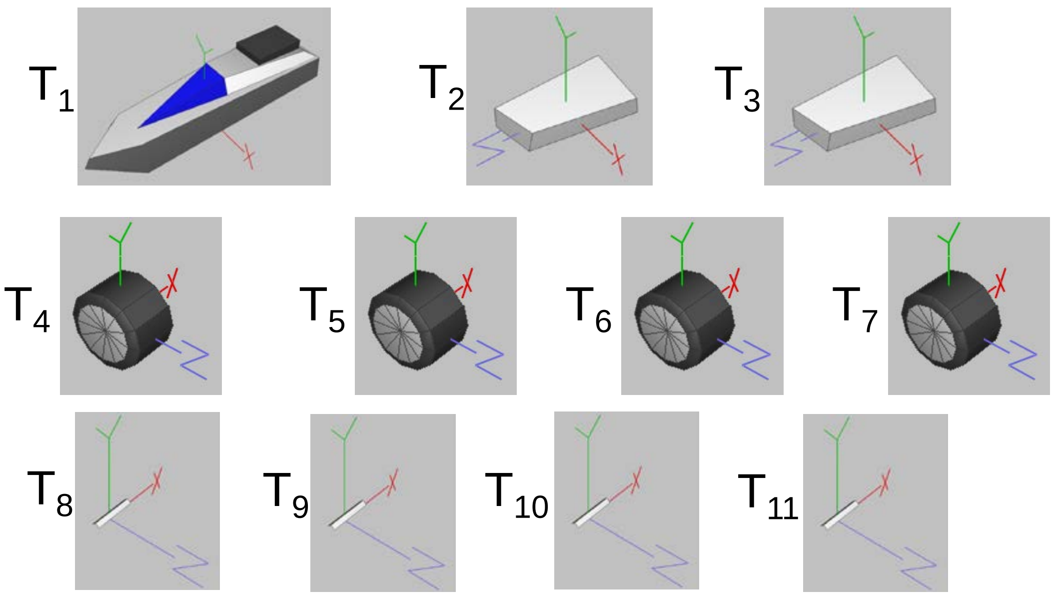
3D Objects are hierarchically modeled

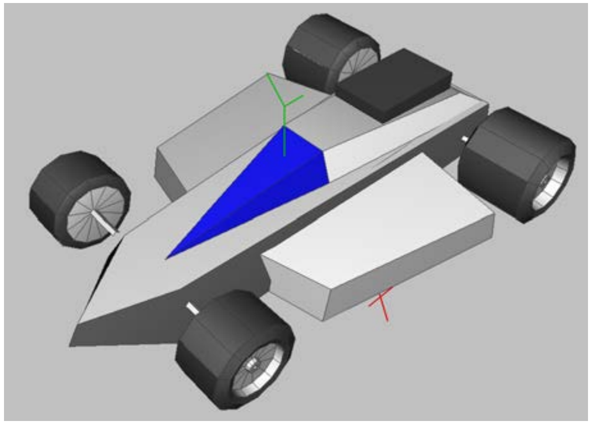


Scene graph

Option 1:

- Store copy of each component object + associated transformation
- Editing the shape is difficult …

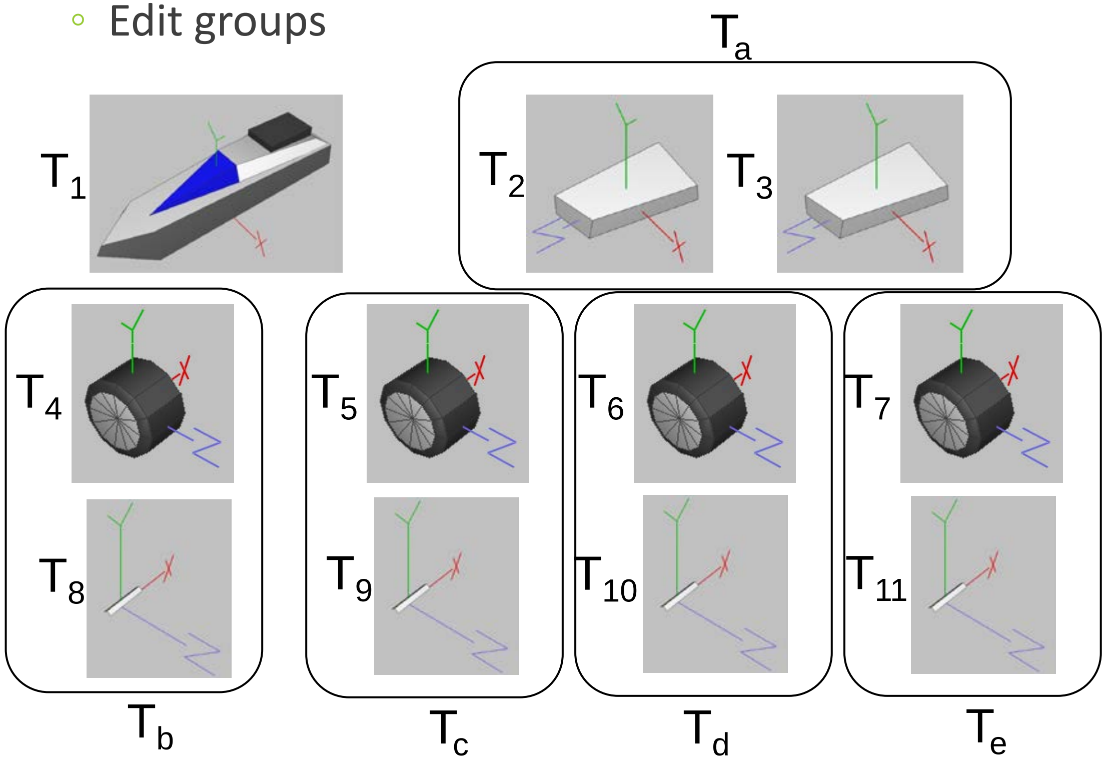


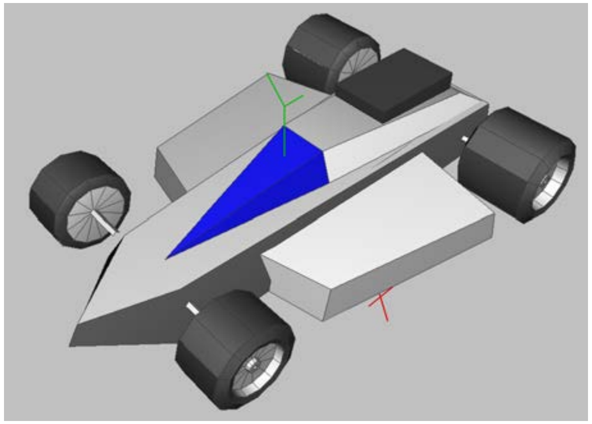


Scene graph

Option 2:

- Group objects together
- Edit groups

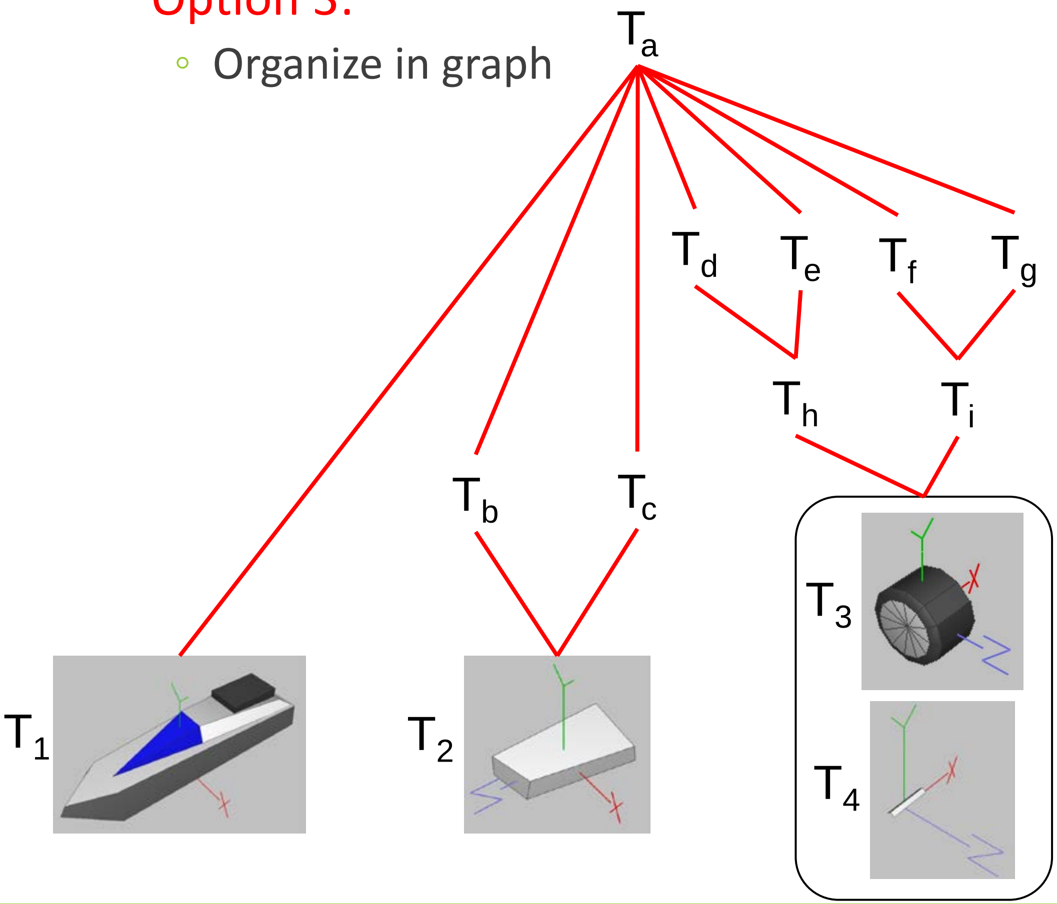


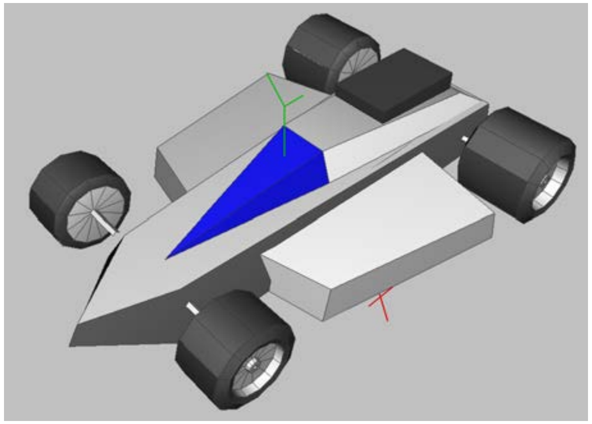


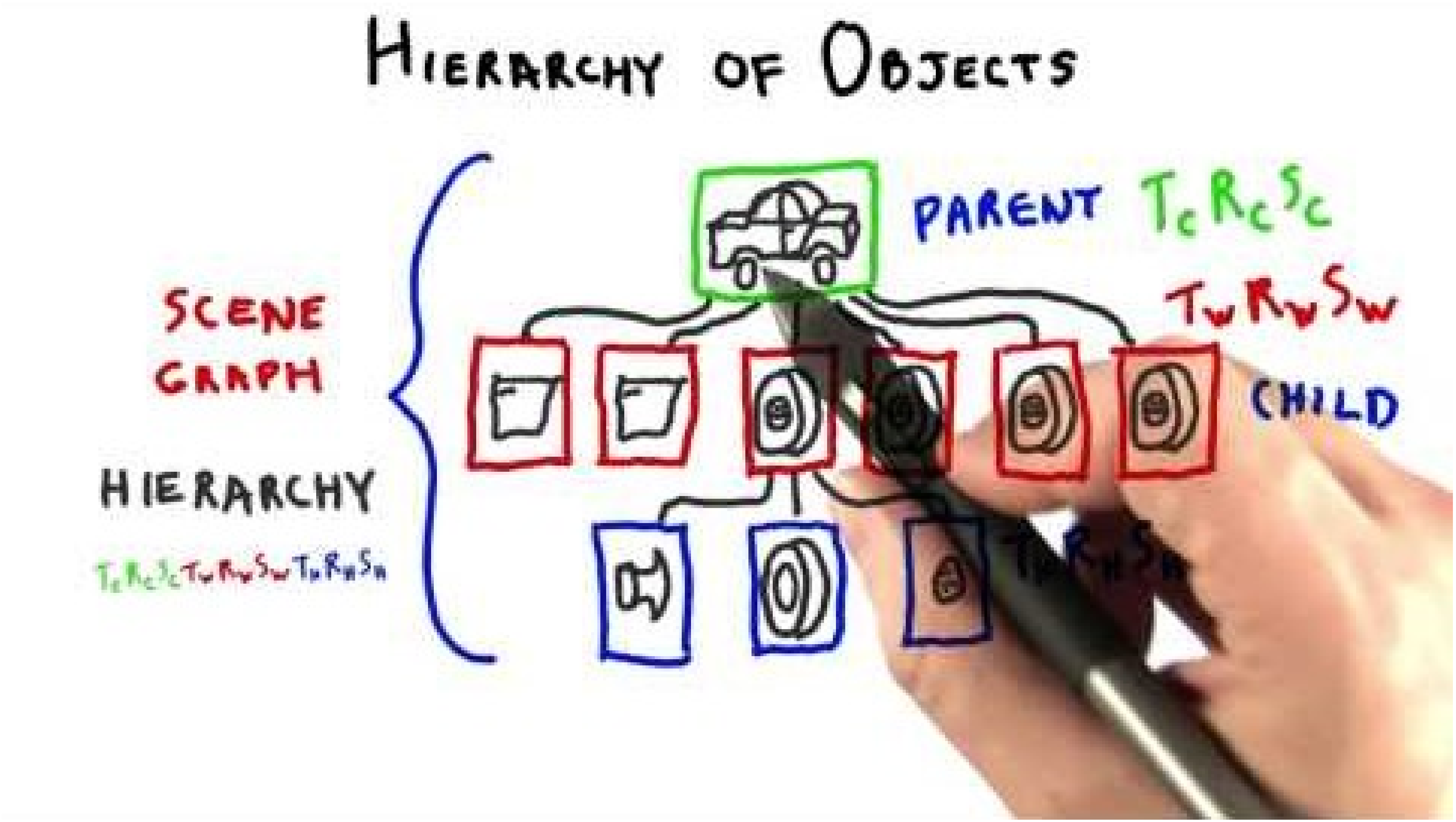
Scene graph

Option 3:

- Organize in graph







https://www.youtube.com/watch?v=rXoGR5pobG4

Scene graph and ray tracing

Scene graph is used for modeling, storing, manipulating … the scene.

How to raytrace a scene graph? (option 1)

- Before ray tracing is started:
  - traverse scene graph, apply (composite) transformations to each geometric object, place all transformed objects in 3D world, …
- During ray tracing:
  - trace rays in fully instantiated 3D world and intersect rays with transformed objects

Scene graph and ray tracing

Scene graph is used for modeling, storing, manipulating … the scene.

How to raytrace a scene graph? (option 2)

- During ray tracing:
  - traverse the scene graph for each ray, apply inverse transformations to ray
  - intersect transformed rays with original, non-transformed objects

Scene graph and ray tracing

GENERATE ENTIRE WORLD

- Entire world in memory
- Difficult to intersect or represent some types of object (e.g. transformed torus)
- easier to build acceleration structures (see next lectures)

INVERSE-TRANSFORM THE RAY

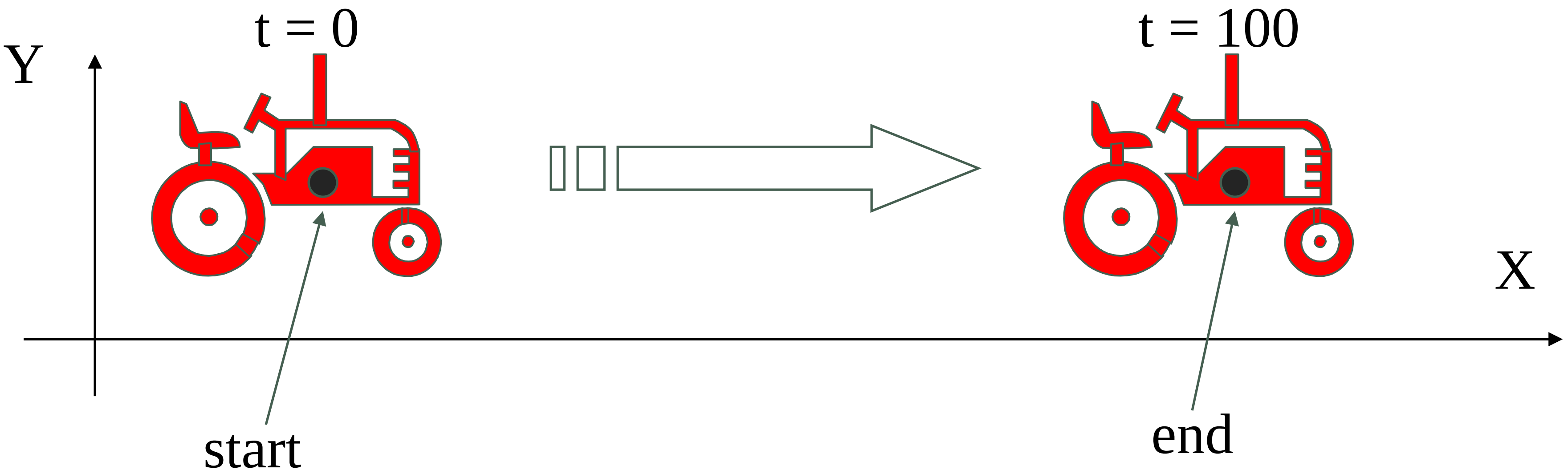
- No need to keep all instances of objects in memory
- Easy to intersect most objects (e.g. transformed torus)
- more difficult to build acceleration structures (see next lectures)

➔ Often a hybrid approach is used

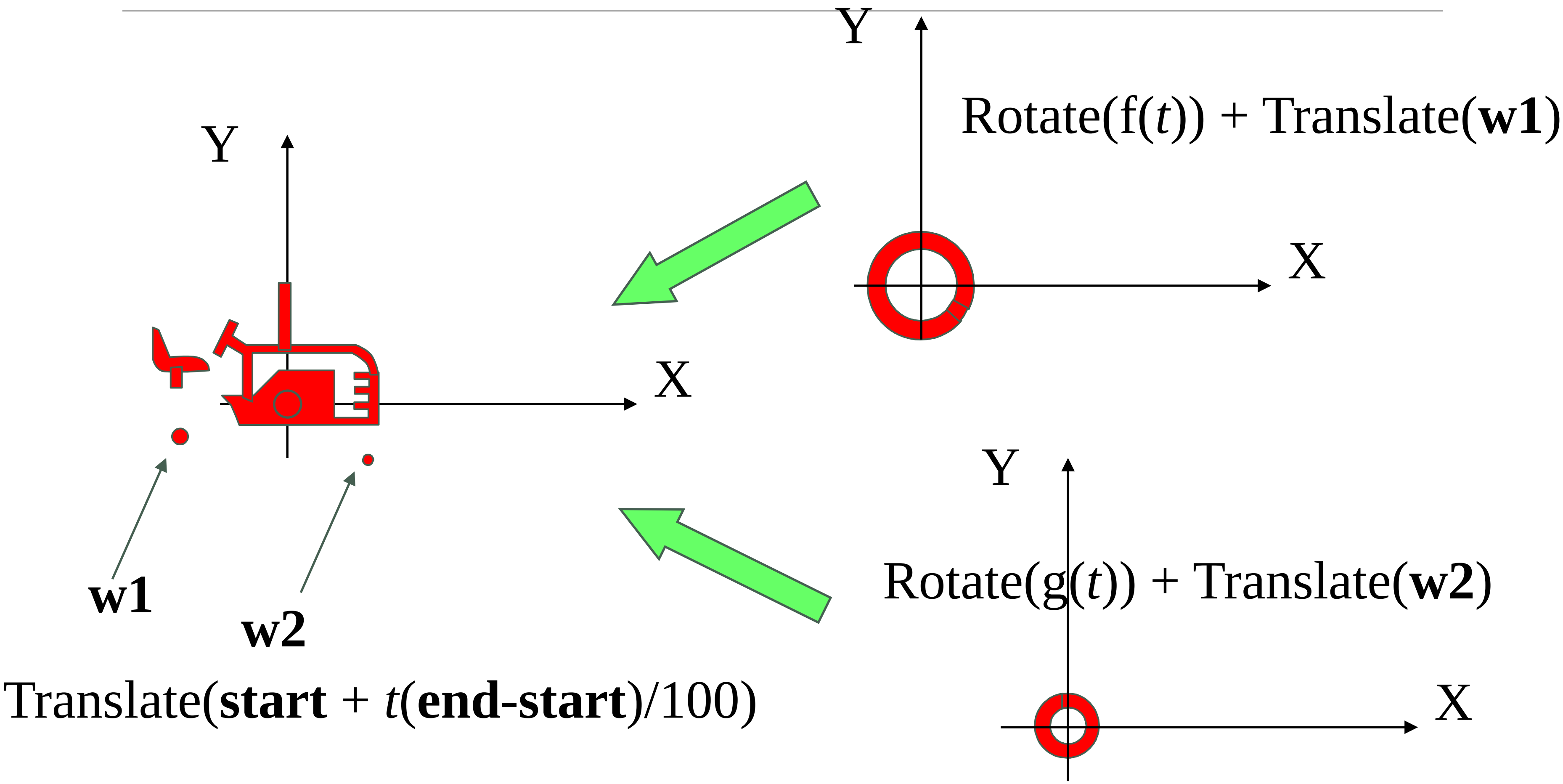
Appendix: Scene graph examples

Scene graph: moving tractor

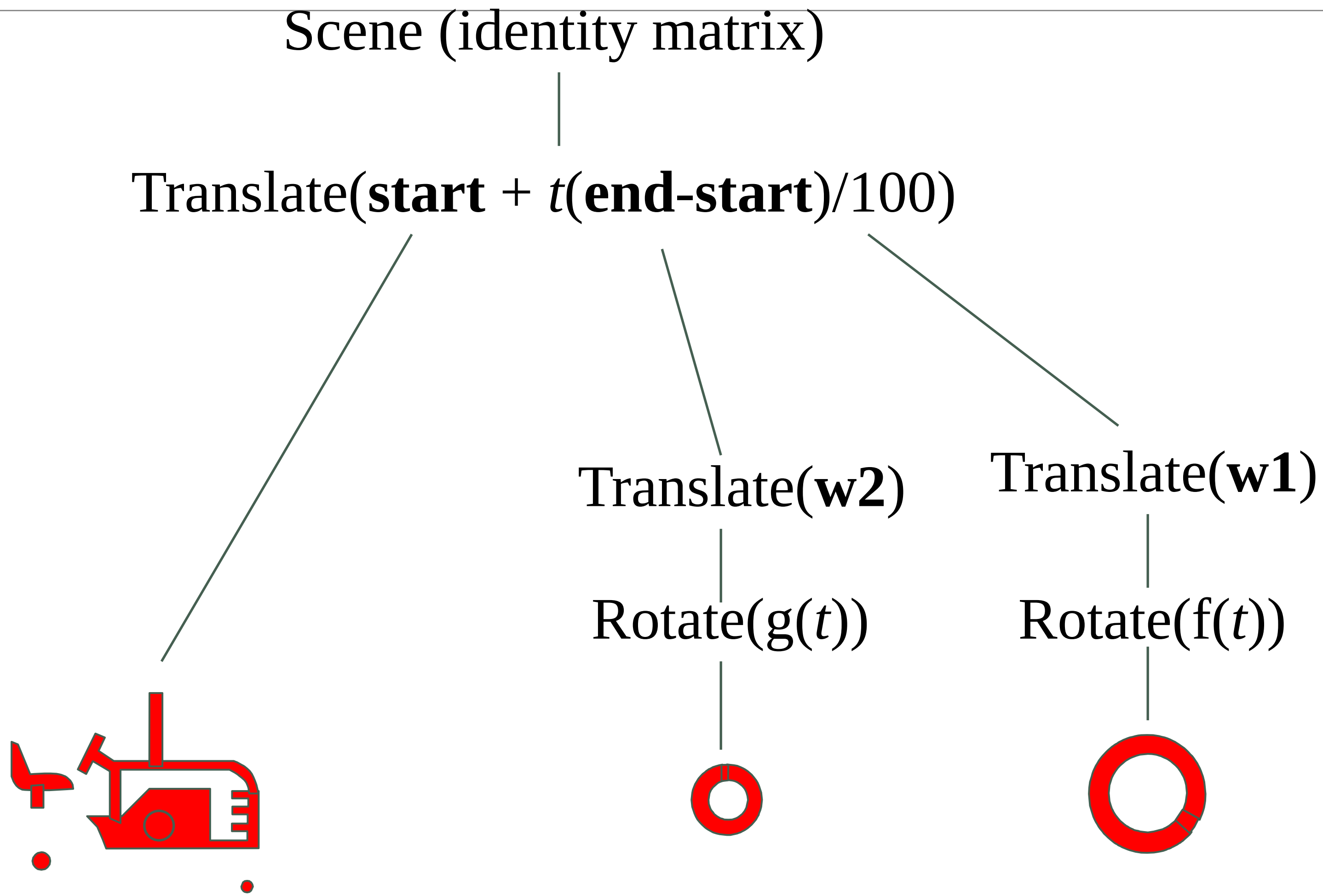
Model a vehicle with rotating wheels …



Scene graph: moving tractor



Scene graph: moving tractor

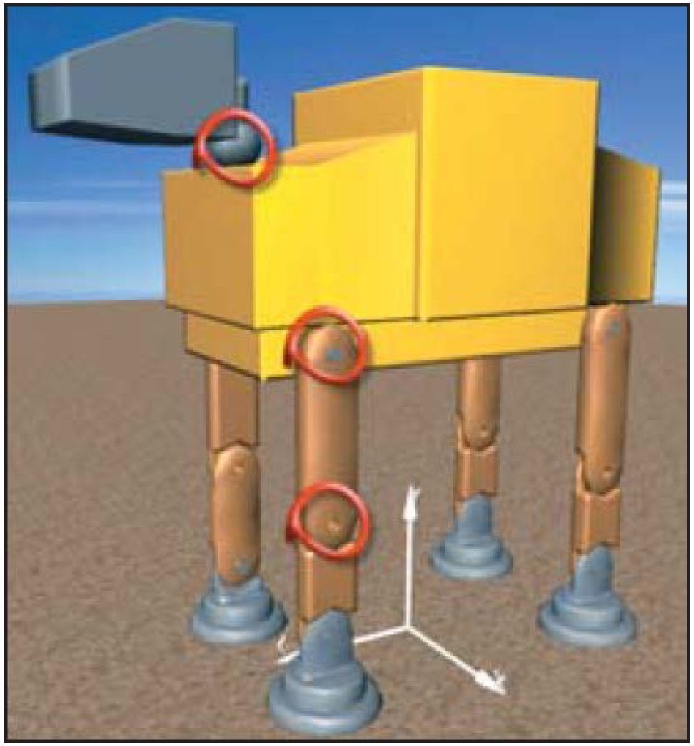


Scene graph: camel caravan

(Computer Graphics: Principles and practice, Hughes et al.)

Camel model

- 3 (rotation) joints: neck, hip, knee



Scene graph: camel caravan

(Computer Graphics: Principles and practice, Hughes et al.)

Model of a single leg

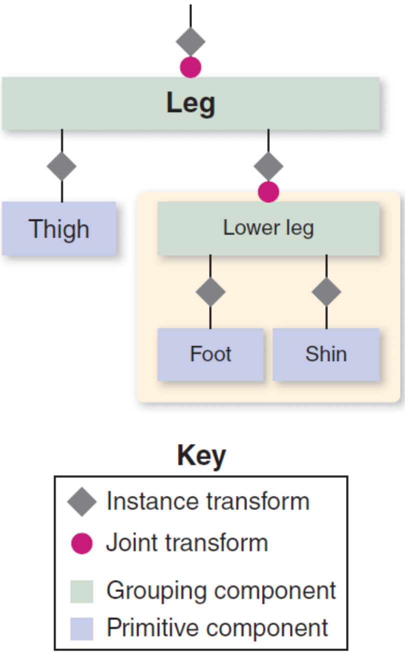


Figure 6.41: Scene graph of the camel-leg model. Here, and below, we use a beige background to highlight a portion of the graph that is being used as a component or submodel.

Scene graph: camel caravan

(Computer Graphics: Principles and practice, Hughes et al.)

Model of a single leg

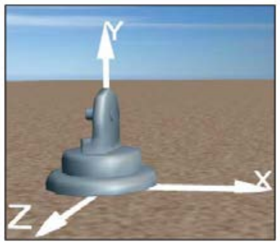


Figure 6.42: Rendering of the foot model, at its canonical position at the origin.

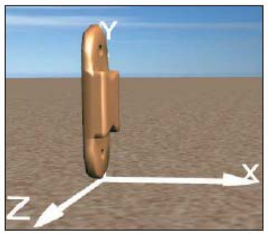


Figure 6.43: Rendering of the shin model, at its canonical position at the origin.

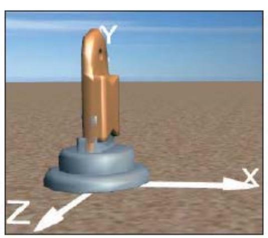


Figure 6.44: Rendering of a first draft of a lower-leg model, constructed by composing the two subcomponents without moving them from their canonical positions at the origin of the coordinate system.

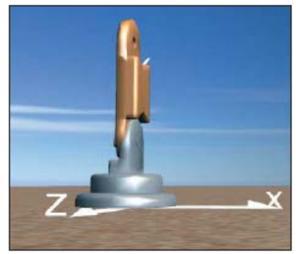


Figure 6.45: Rendering of the lower-leg model, now corrected via application of a modeling transformation on the shin subcomponent.

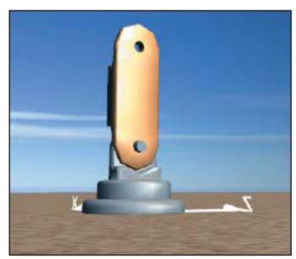


Figure 6.46: Rendering of the lower-leg model from a second point of view.

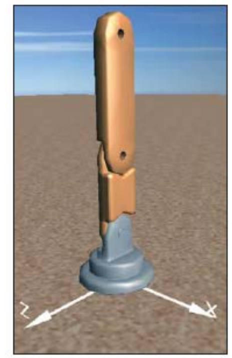


Figure 6.47: Rendering of the complete leg model.

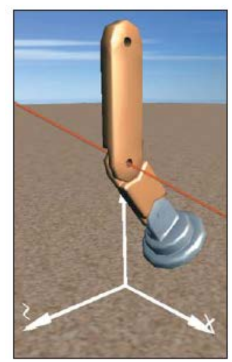


Figure 6.48: Result of specifying a 37° rotation at the knee joint, annotated with a red line through the joint, parallel to the x-axis, showing the axis of rotation.

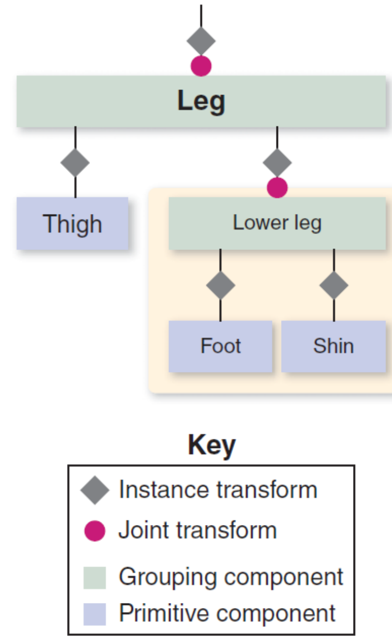


Figure 6.41: Scene graph of the camel-leg model. Here, and below, we use a beige background to highlight a portion of the graph that is being used as a component or submodel.

Scene graph: camel caravan

(Computer Graphics: Principles and practice, Hughes et al.)

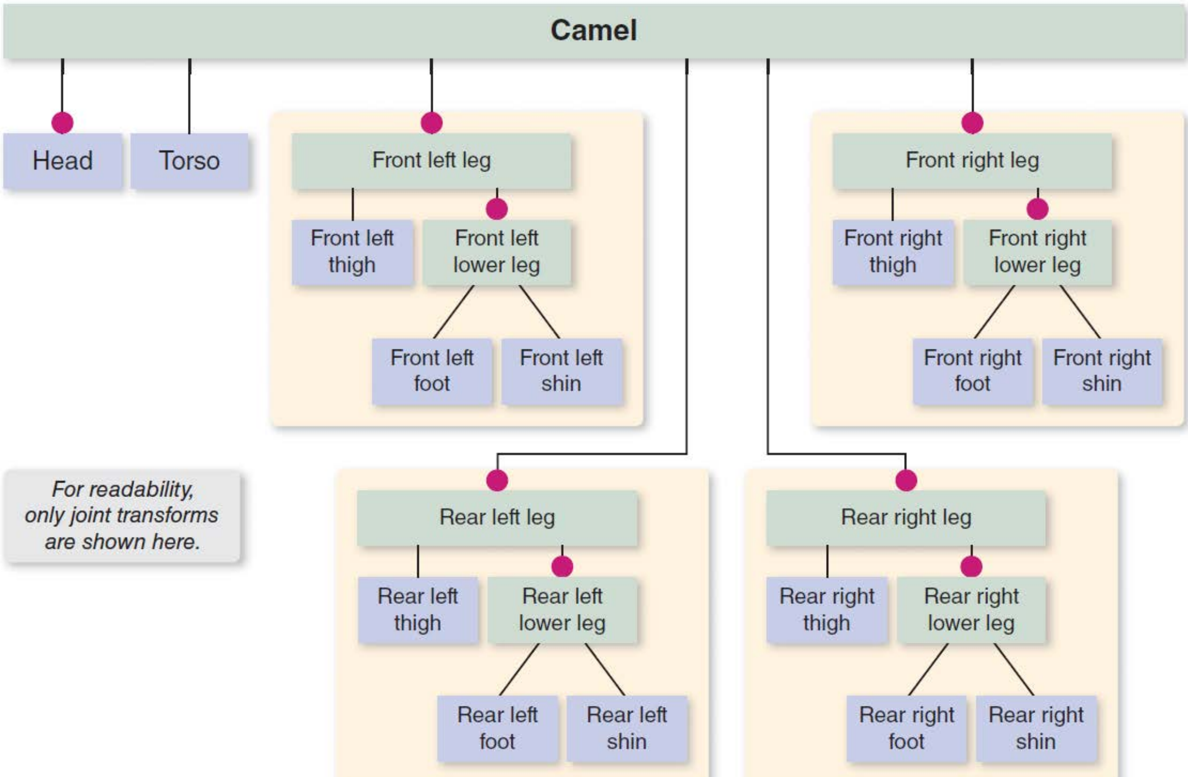


Figure 6.49: Scene graph of a camel constructed without reusable components, allowing individual control of each joint.

Scene graph: camel caravan

(Computer Graphics: Principles and practice, Hughes et al.)

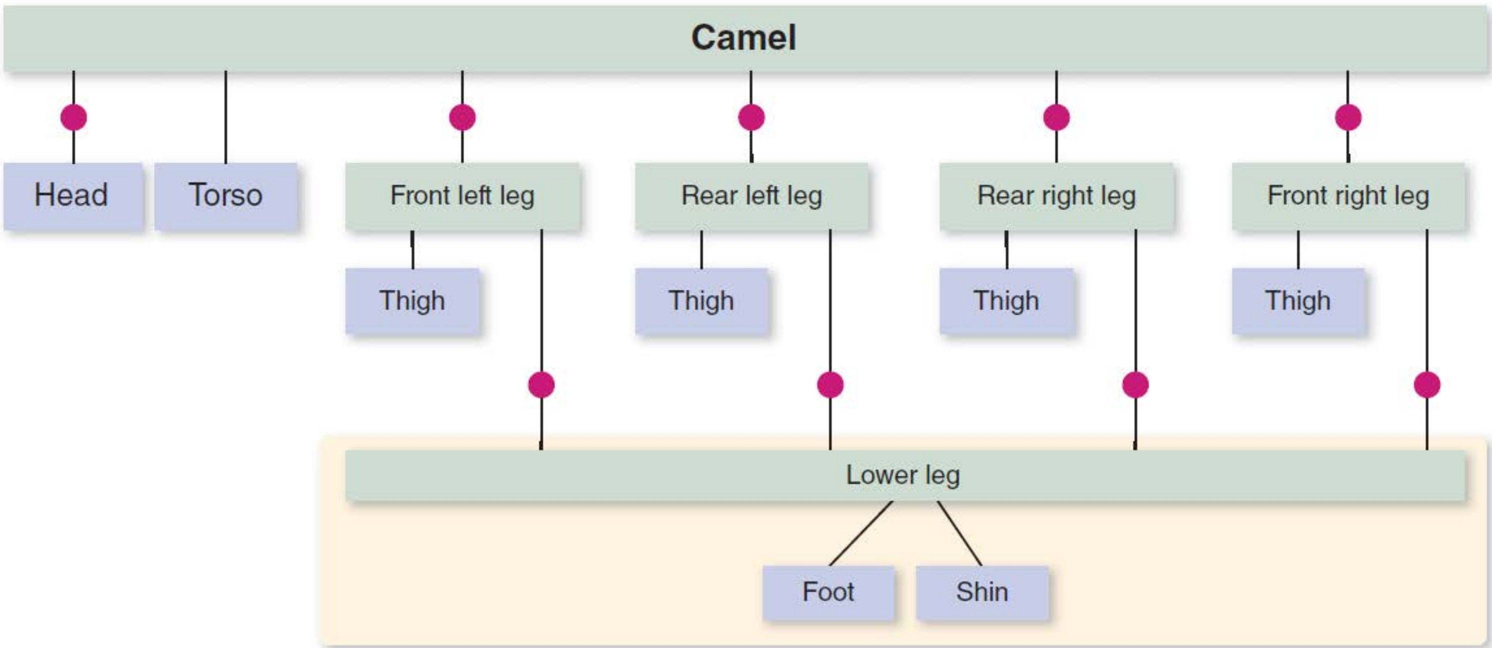


Figure 6.50: Reducing the storage cost by reusing a lower-leg submodel, with no loss of flexibility in joint control.

Scene graph: camel caravan

(Computer Graphics: Principles and practice, Hughes et al.)

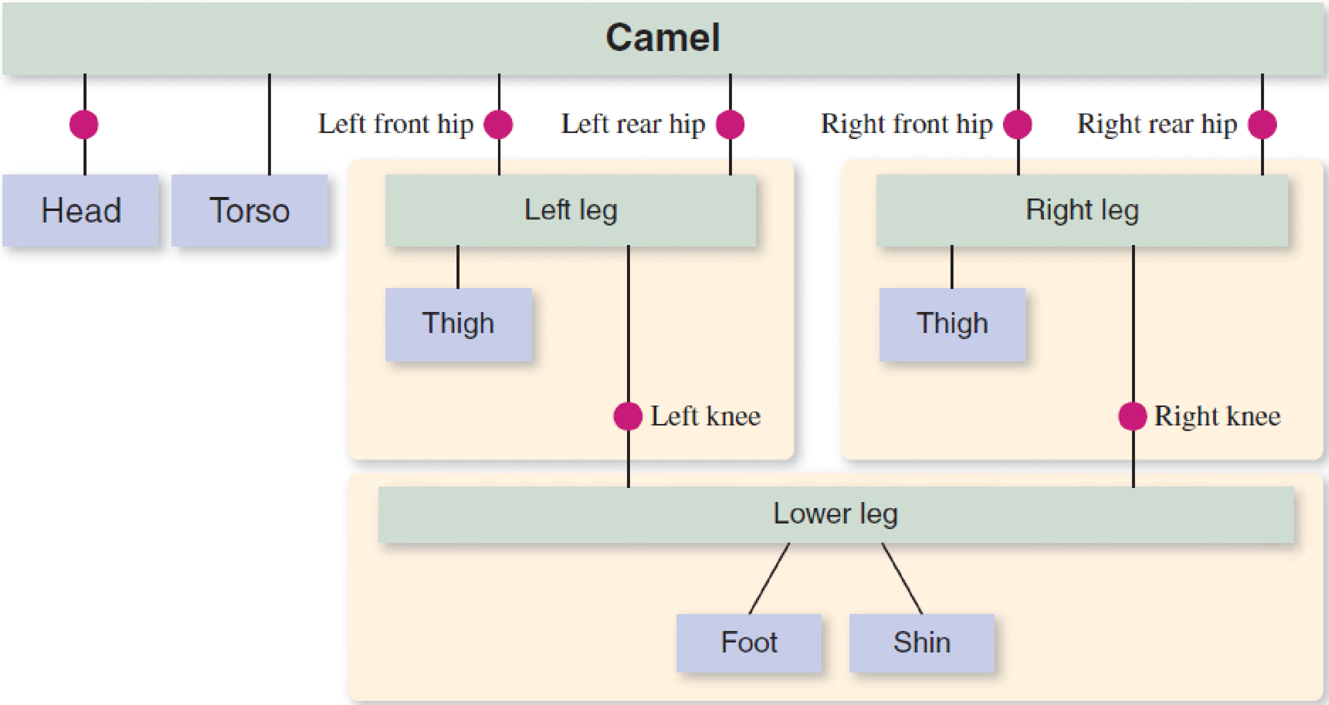


Figure 6.51: Reducing the storage cost by reusing a model for the left-side legs and a separate model for the right-side legs, with great loss of flexibility in joint control.

Scene graph: camel caravan

(Computer Graphics: Principles and practice, Hughes et al.)

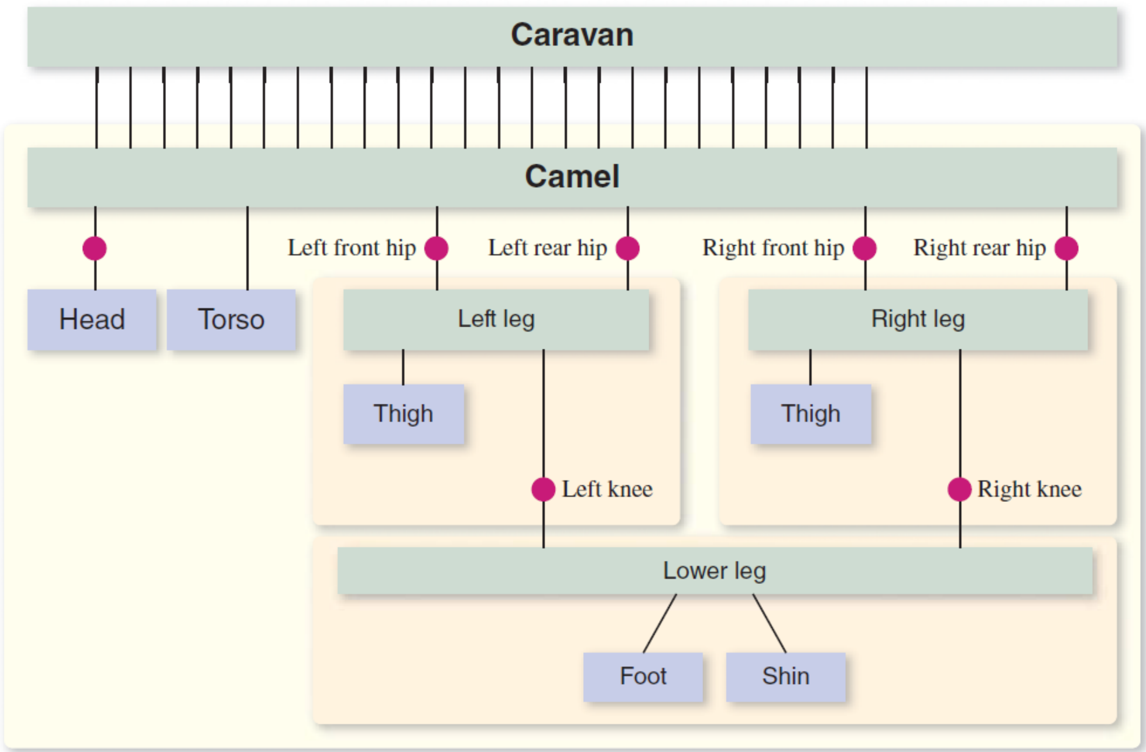
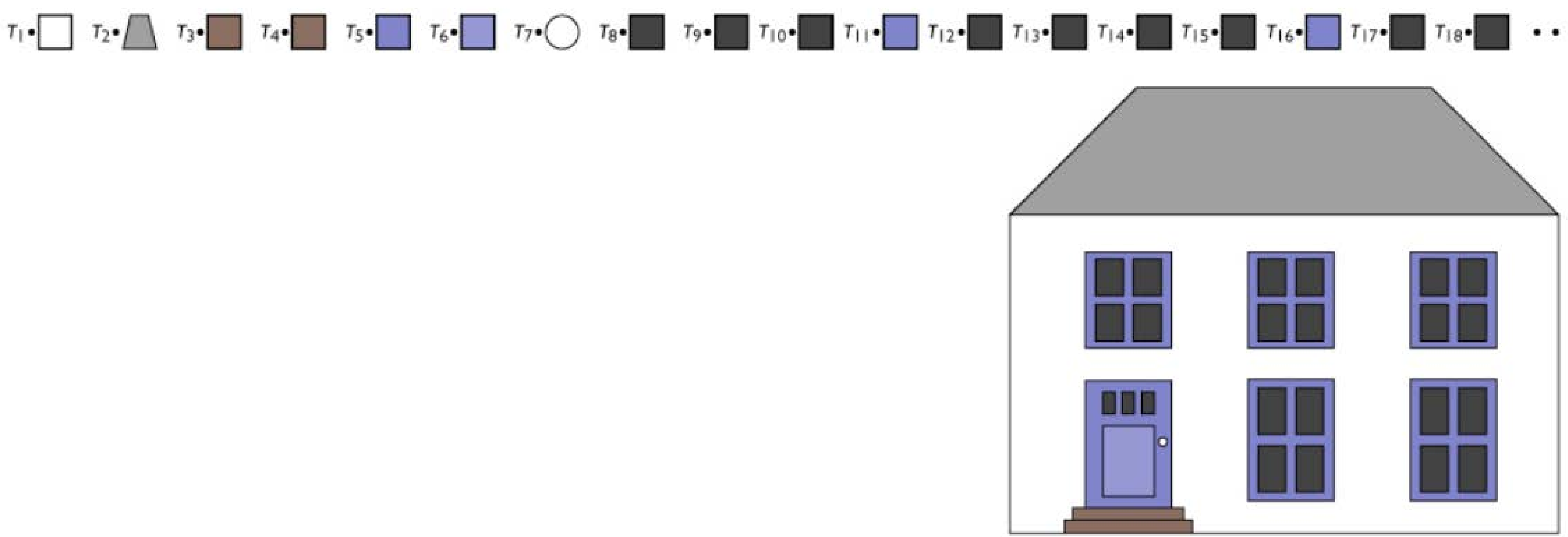


Figure 6.52: Modeling a caravan by reusing a single camel model, a highly scalable approach at the cost of excessive synchronized leg movement.

Scene graph: 2D House

Modeling a scene: many different transformations on different objects

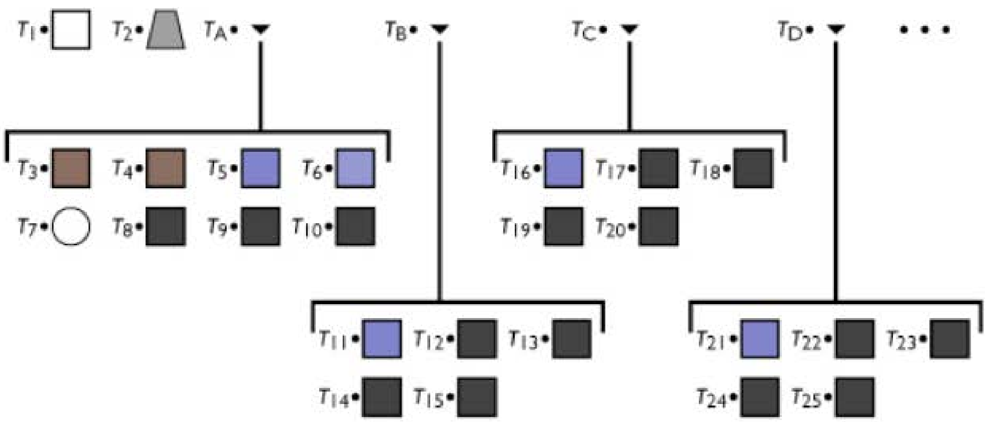
Editing is difficult



Scene graph: 2D House

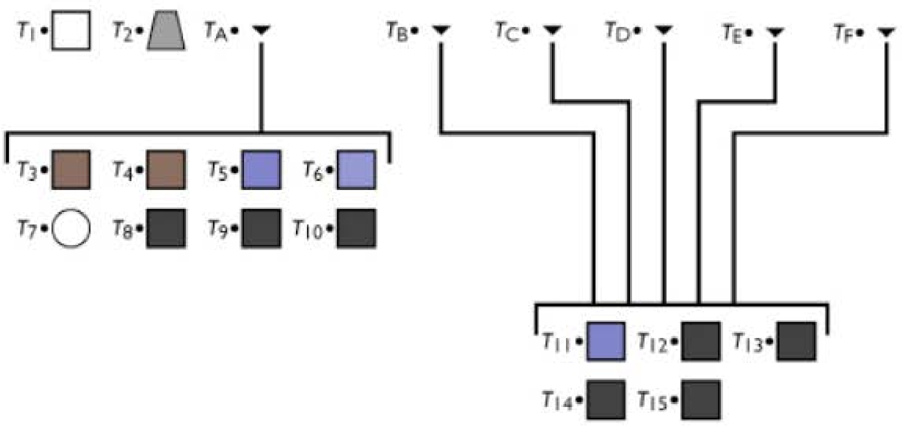
Group objects together

Edit groups



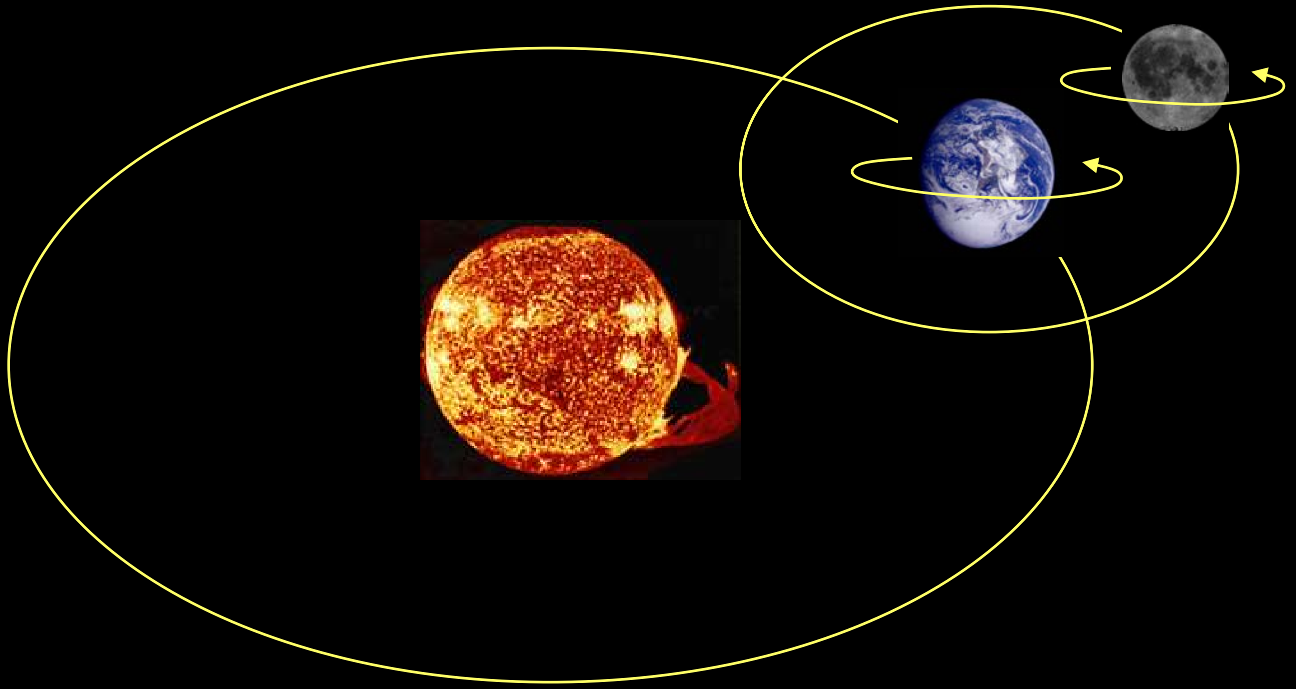
Scene graph: 2D House

Group objects, traverse graph to process scene

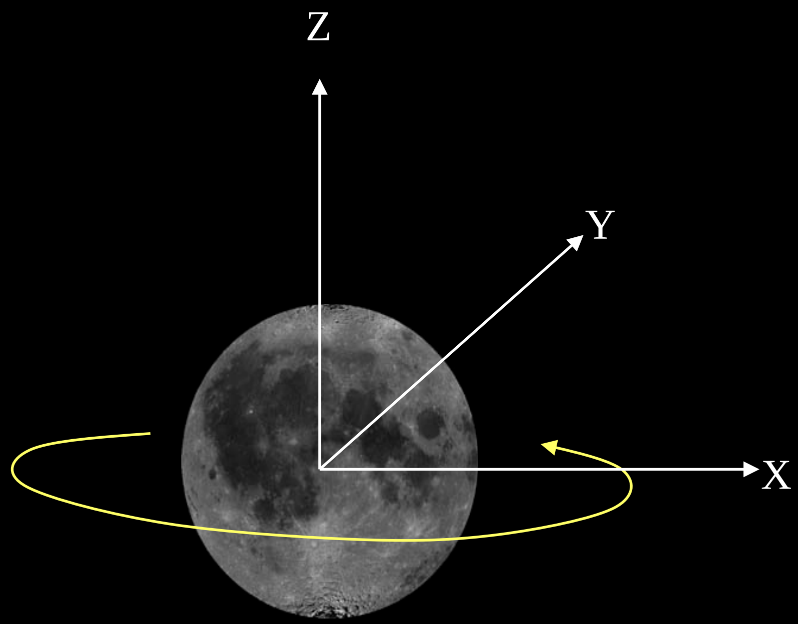


Scene graph: Sun-Earth-Moon

- Rotating planets …

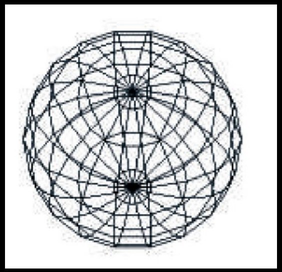


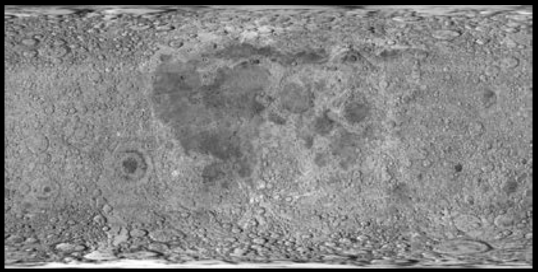
Scene graph: Sun-Earth-Moon



Scale: S(moon_size)

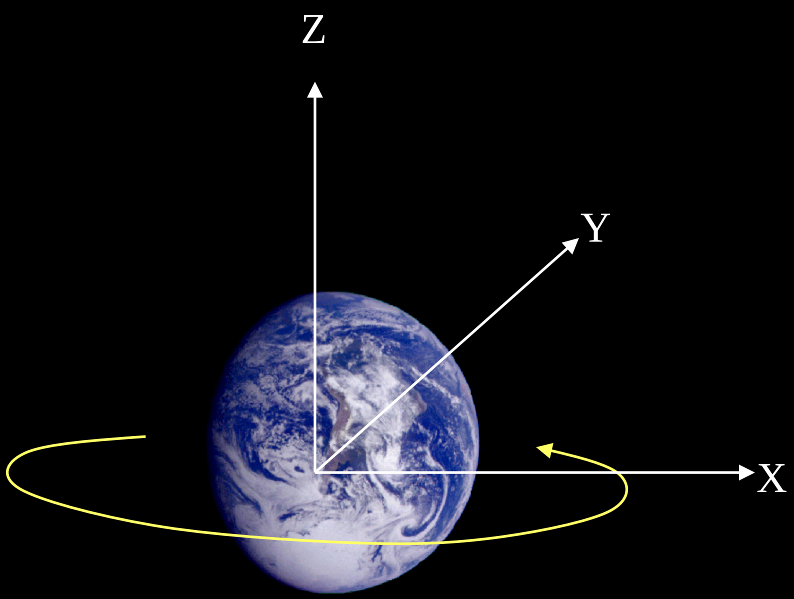
Rotate: R(moon_rotation)





Geometry + texture (later …)

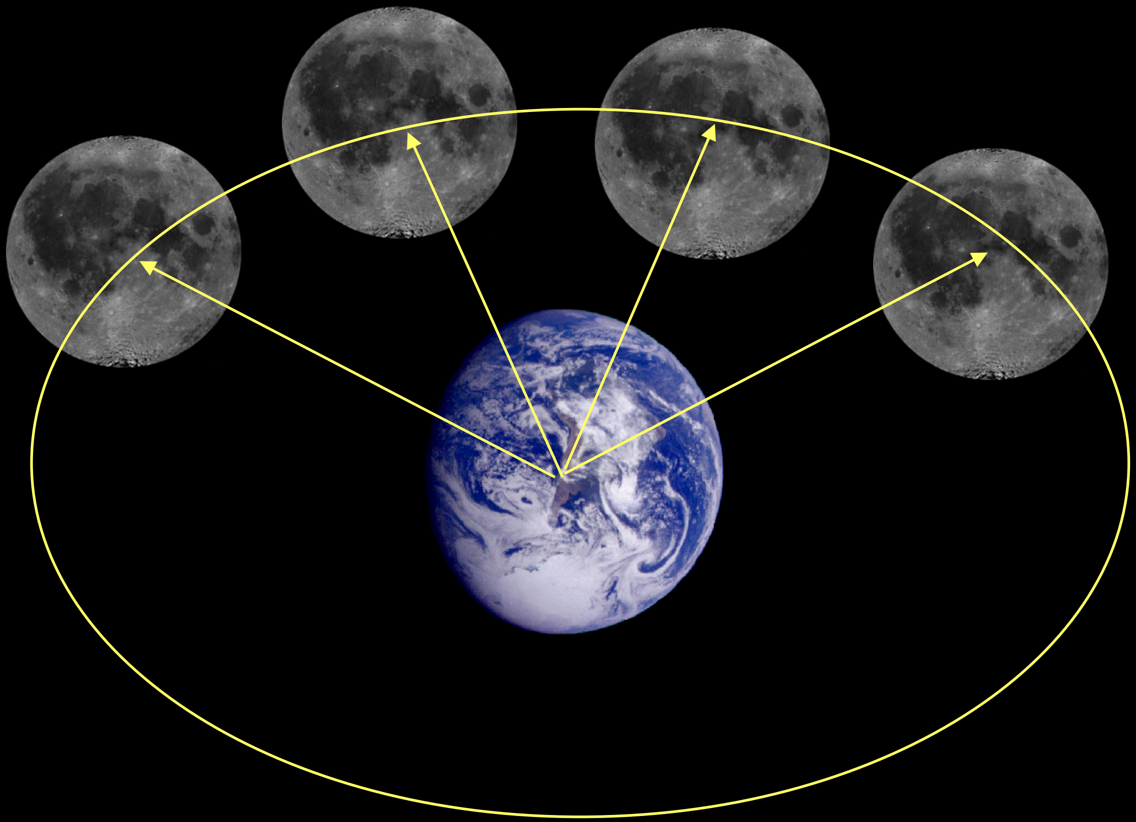
Scene graph: Sun-Earth-Moon



Scale: S(earth_size)

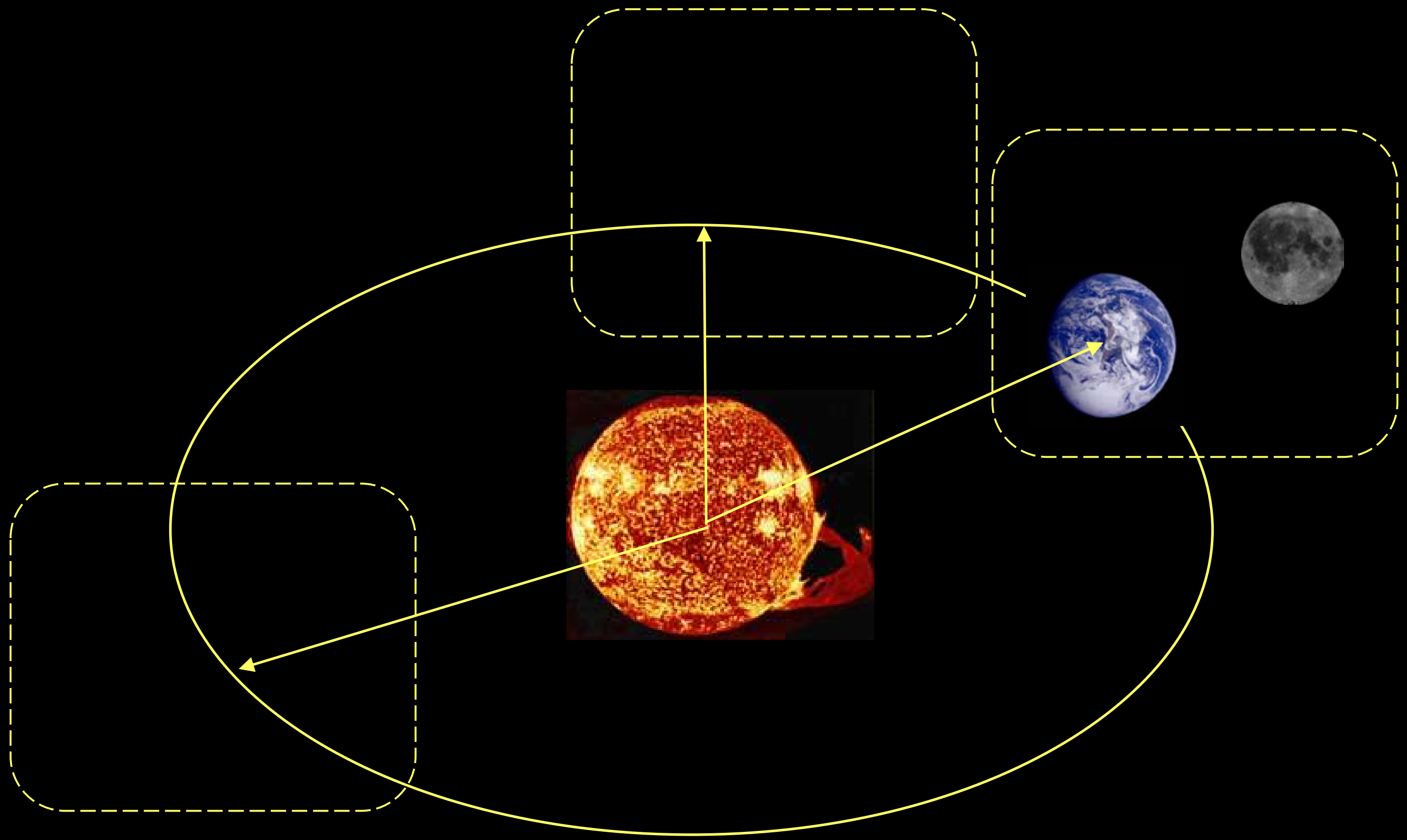
Rotate: R(earth_rotation)

Scene graph: Sun-Earth-Moon



Translate T(moon-earth)

Scene graph: Sun-Earth-Moon



Scale S(sun_size) Translation T(earth-sun)

Scene graph: Sun-Earth-Moon