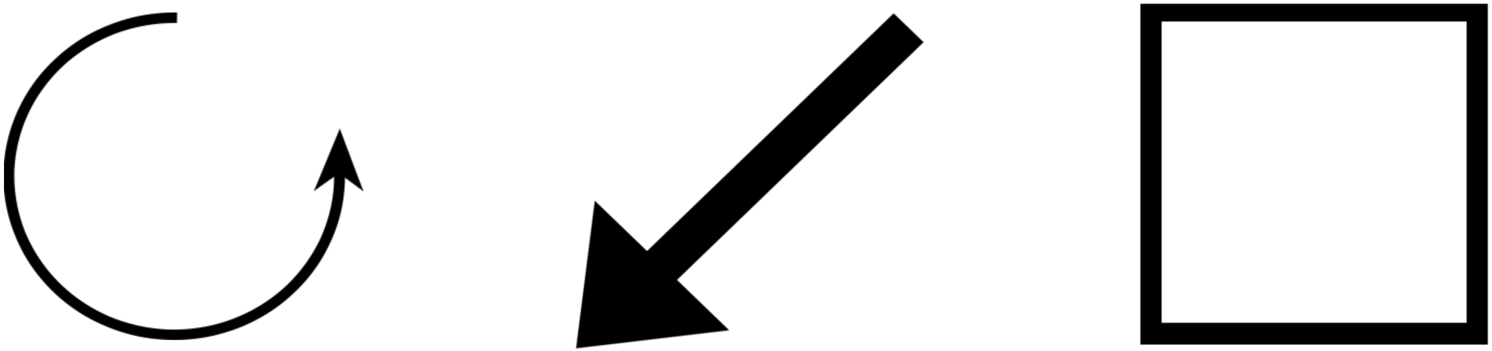


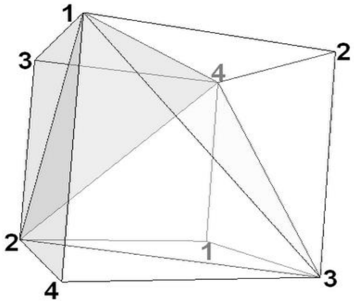
Geometric Transformations

2D and 3D

Andries van Dam

How do we use Geometric Transformations? (1/2)

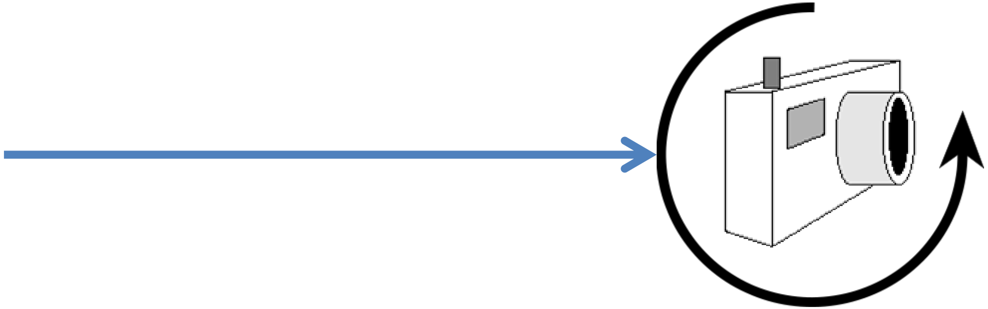
- Objects in a scene are a collection of points…



- These objects have location, orientation, size
- Corresponds to transformations, Translation ($\boldsymbol{T}$), Rotation ($\boldsymbol{R}$), and Scaling ($\boldsymbol{S}$)

How do we use Geometric Transformations? (2/2)

- A scene has a camera/view point from which the scene is viewed
- The camera has some location and some orientation in 3-space ...



- These correspond to Translation and Rotation transformations
- Need other types of viewing transformations as well - learn about them shortly

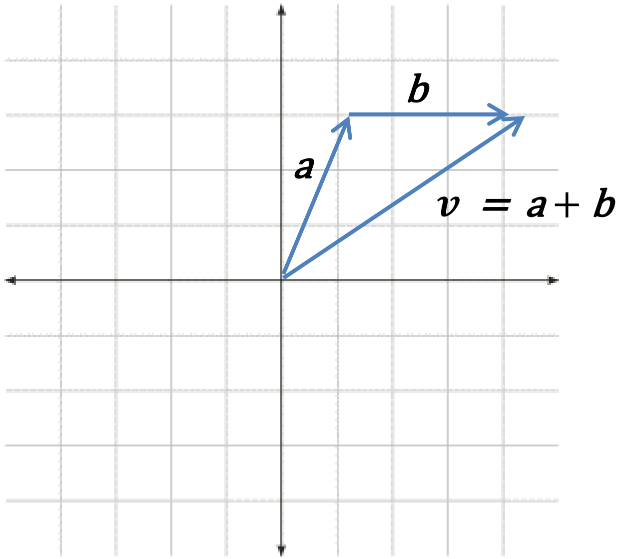
Some Linear Algebra Concepts...

- 3D Coordinate geometry
- Vectors in 2 space and 3 space
- Dot product and cross product – definitions and uses
- Vector and matrix notation and algebra
- Identity Matrix
- Multiplicative associativity
  - E.g. A(BC) = (AB)C
- Matrix transpose and inverse – definition, use, and calculation
- Homogeneous coordinates $(x, y, z, \boldsymbol{w})$

You will need to understand these concepts!

Linear Transformations (1/3)

- We represent vectors as ***bold-italic*** letters ($\boldsymbol{v}$) and scalars as just *italicized* letters (c)
- Any vector in plane can be defined as addition of two non-collinear basis vectors in the plane
  - Recall that a basis is a set of vectors with the following two properties:
  - The vectors are linearly independent
  - Any vector in the vector space can be generated by a linear combination of the basis vectors
- Scalar constants can be used to adjust magnitude and direction of resultant vector

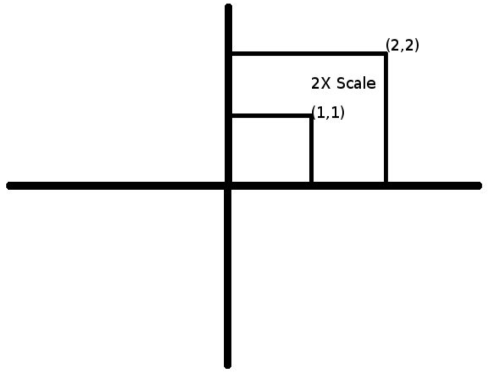


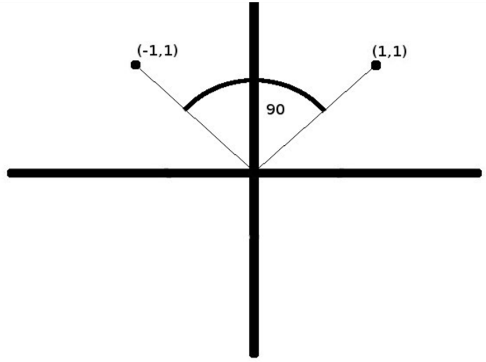
Linear Transformations (2/3)

- Definition of a linear function, $\boldsymbol{f}$:
  - $\boldsymbol{f}(\boldsymbol{v}+\boldsymbol{w})=\boldsymbol{f}(\boldsymbol{v})+\boldsymbol{f}(\boldsymbol{w})$ where domain and co-domain of $\boldsymbol{f}$ are identical
    - function of a vector addition is equivalent to addition of function applied to each of the vectors
  - $\boldsymbol{f}(c\boldsymbol{v})=c\boldsymbol{f}(\boldsymbol{v})$
    - function of a scalar multiplication with a vector is scalar multiplied by function applied to vector
- Both of these properties must be satisfied in order for $\boldsymbol{f}$ to be a linear operator

Linear Transformations (3/3)

- Graphical Use: transformations of points around the origin (**leaves the origin invariant**)
  - These include *Scaling* and *Rotations* (but not translation),
  - *Translation* is not a linear function (moves the origin)
  - Any linear transformation of a point will result in another point in the same coordinate system, transformed about the origin





Linear Transformations as Matrices (1/2)

- Linear Transformations can be represented as non-singular (invertible) matrices
- Let's start with 2D transformations:

$$\boldsymbol{T} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

- The matrix $\boldsymbol{T}$ can also be written as:

$$\begin{pmatrix} \boldsymbol{T}(e1) & \boldsymbol{T}(e2) \end{pmatrix} \text{, where } \boldsymbol{T}(e1) = \begin{bmatrix} a \\ c \end{bmatrix}, \boldsymbol{T}(e2) = \begin{bmatrix} b \\ d \end{bmatrix}$$

- Where $e1$ and $e2$ are the standard unit basis vectors along the x and y vectors:

$$e1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ e2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

- Why is this important? This means we can compute the columns of a transformation matrix one by one by determining how our transformation effects each of the standard unit vectors. Thus $\boldsymbol{T}$ "sends $e1$ to = $\begin{bmatrix} a \\ c \end{bmatrix}$ "
- Use this strategy to derive transformation matrices

Linear Transformations as Matrices (2/2)

- A transformation of an arbitrary column vector $\begin{bmatrix} x \\ y \end{bmatrix}$ has form:

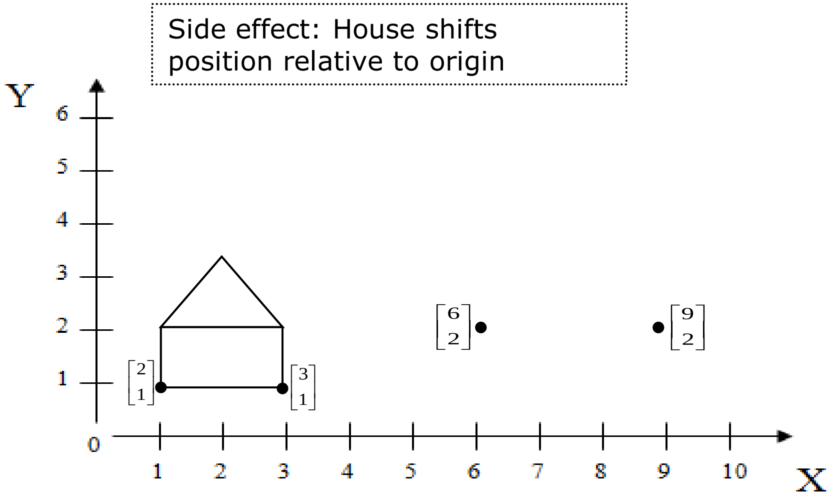
$$\boldsymbol{T}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$$

- Let's substitute $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ for $\begin{bmatrix} x \\ y \end{bmatrix}$: $\boldsymbol{T}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a \\ c \end{bmatrix}$
  - transformation applied to $\begin{bmatrix} 1 \\ 0 \end{bmatrix}$ is 1^{st} column of $\boldsymbol{T}$

- Now substitute $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ for $\begin{bmatrix} x \\ y \end{bmatrix}$: $\boldsymbol{T}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} b \\ d \end{bmatrix}$
  - transformation applied to $\begin{bmatrix} 0 \\ 1 \end{bmatrix}$ is 2^{nd} column of $\boldsymbol{T}$

Scaling in 2D (1/2)

- Scale x by 3, y by 2 ($S_x = 3, S_y = 2$)
  - $\boldsymbol{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ (original vertex); $\boldsymbol{v}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ (new vertex)
- $\boldsymbol{v}' = \boldsymbol{S}\,\boldsymbol{v}$
- Derive $\boldsymbol{S}$ by determining how $e1$ and $e2$ should be transformed
  - $e1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow s_x * e1 = \begin{bmatrix} s_x \\ 0 \end{bmatrix}$ (Scale in X by s_x) , the first column of $\boldsymbol{S}$
  - $e2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow s_y * e2 = \begin{bmatrix} 0 \\ s_y \end{bmatrix}$ (Scale in Y by s_y), the second column of $\boldsymbol{S}$
- Thus we obtain $\boldsymbol{S}$: $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$



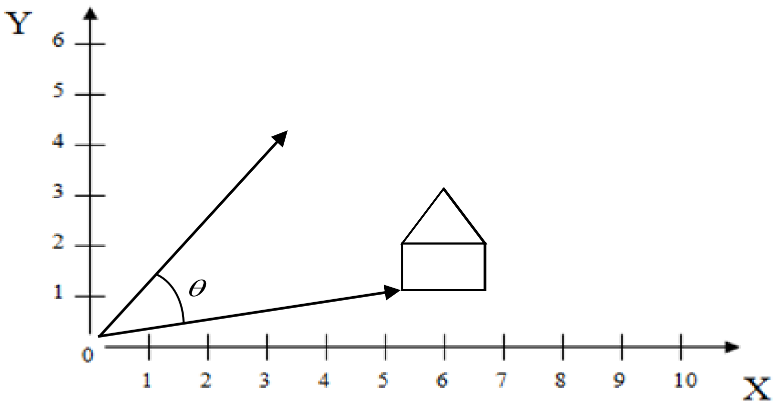
Scaling in 2D (2/2)

- $\boldsymbol{S}$ is a diagonal matrix - can confirm our derivation by simply looking at properties of diagonal matrices:
- $\boldsymbol{Dv} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax \\ by \end{bmatrix} = \boldsymbol{v}'$
  - where $\boldsymbol{D}$ is some diagonal matrix
- i^{th} entry of $\boldsymbol{v}'$ = (i^{th} entry along diagonal of $\boldsymbol{D}$ $*$ i^{th} entry of $\boldsymbol{v}$)
- $\boldsymbol{S}$ multiplies each coordinate of a $\boldsymbol{v}$ by scaling factors (s_x, s_y) specified by the entries along the diagonal, as expected
  - $s_x = a$, $s_y = b$

- Other properties of scaling:
  - does not preserve lengths in objects
  - does not preserve angles between parts of objects (except when scaling is uniform,$s_x = s_y$)
  - if not at origin, translates house relative to origin– often not desired…

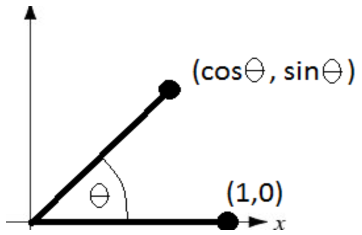
Rotation in 2D (1/2)

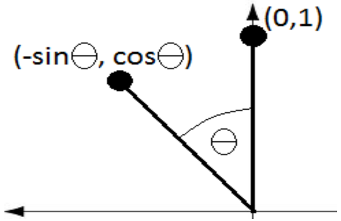
- Rotate by θ about origin
- $\boldsymbol{v}' = \boldsymbol{S}\,\boldsymbol{v}$ where
  - $\boldsymbol{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ (original vertex)
  - $\boldsymbol{v}' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ (new vertex)



- Derive $\boldsymbol{R}_\theta$ by determining how $e1$ and $e2$ should be transformed
  - $e1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \rightarrow \begin{bmatrix} \cos\boldsymbol{\Theta} \\ \sin\boldsymbol{\Theta} \end{bmatrix}$, first column of $\boldsymbol{R}_\theta$
  - $e2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \rightarrow \begin{bmatrix} -\sin\boldsymbol{\Theta} \\ \cos\boldsymbol{\Theta} \end{bmatrix}$, second column of $\boldsymbol{R}_\theta$

- Thus we obtain $\boldsymbol{R}_\theta$: $\begin{bmatrix} \cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$





Rotation in 2D (2/2)

- Let's try matrix-vector multiplication
- $\boldsymbol{R}_\theta * \boldsymbol{v} = \begin{bmatrix} \cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix} = \begin{bmatrix} x' \\ y' \end{bmatrix} = \boldsymbol{v}'$
  - $x' = x\cos\theta - y\sin\theta$
  - $y' = x\sin\theta + y\cos\theta$
- *Other properties of rotation:*
  - preserves lengths in objects, and angles between parts of objects
  - rotation is rigid-body
  - for objects not at the origin, again a translation may be unwanted (i.e., this rotates about origin, not about house's center of rotation)

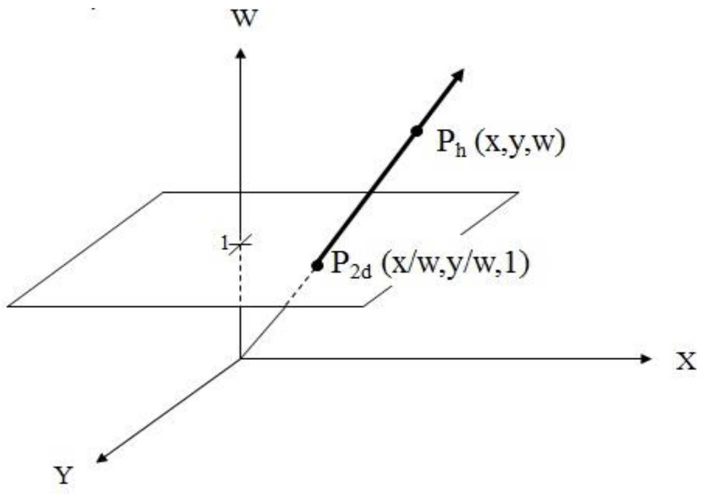
What about translation?

- Translation not a linear transformation (not centered about origin)
- Can't be represented as a 2x2 invertible matrix …
- **Question:** Is there another solution?
- **Answer:** Yes, $\boldsymbol{v}' = \boldsymbol{v} + \boldsymbol{t}$, where $\boldsymbol{t} = \begin{bmatrix} dx \\ dy \end{bmatrix}$
- Addition for translation – this is inconsistent

- If we could treat all transformations in a consistent manner, i.e., with matrix representation, then could combine transformations by composing their matrices
- Let's try using a Matrix again
- How? **Homogeneous Coordinates:** add an additional dimension, the w-axis, and an extra coordinate, the w-component
  - thus 2D -> 3D (effectively the hyperspace for embedding 2D space)

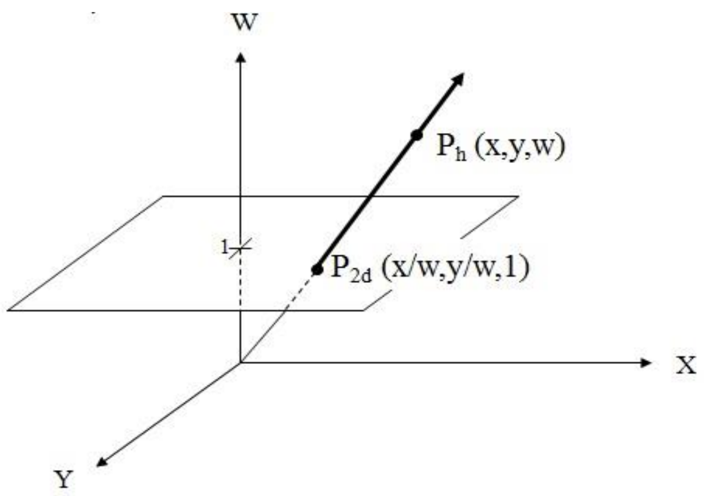
Homogeneous Coordinates (1/3)

- Allows expression of all three 2D transformations as 3x3 matrices
- We start with the point P_{2d} on the xy plane and apply a mapping to bring it to the w-plane in hyperspace
  - $P_{2d}(x, y) \rightarrow P_h(wx, wy, w),\ \ w \neq 0$
- The resulting (x', y') coordinates in our new point P_h are different from the original (x, y) , where $x' = wx$, $y' = wy$
  - $P_h(x', y', w),\ \ w \neq 0$



Homogeneous Coordinates (2/3)

- Once we have this point we can apply a homogenized version of our transformation matrices (next slides) to it to get a new point in hyperspace
- Finally, want to obtain resulting point in 2D-space again so perform a reverse of previous mapping (divide all entries by w)
- This converts our point in hyperspace to a corresponding point in 2D space
  - $P_{2d}(x, y) = P_{2d}\left(\frac{x'}{w}, \frac{y'}{w}\right)$
- The vertex $\boldsymbol{v} = \begin{bmatrix} x \\ y \end{bmatrix}$ is now represented as $\boldsymbol{v} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$



Homogeneous Coordinates (3/3)

- Make transformations map points in hyperplane to another point in hyperplane. Transformations applied to a point in the hyperplane will always yield a result also in the same hyperplane (mathematical closure)
- Transformation $\boldsymbol{T}$ applied to $\boldsymbol{v} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ maps to $\boldsymbol{v}' = \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}$
- How do we apply this to our transformation matrices?
- For linear transformations, maintain 2x2 sub-matrix, expand the matrix as follows, where for 2D transformations, the upper left submatrix is the embedding of either the scale or the rotation matrix derived earlier:
  - $\begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Back to Translation

- Our translation matrix ($\boldsymbol{T}$) can now be represented by embedding the translation vector in the rightcolumn at the top as:
- $$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$$
- Try it - multiply it by our homogenized vertex $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$
- $\boldsymbol{T}\ \boldsymbol{v} = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + dx \\ y + dy \\ 1 \end{bmatrix} = \boldsymbol{v}'$
- Coordinates have been translated, $\boldsymbol{v}'$ still homogeneous

Transformations Homogenized

- Translation uses a 3x3 Matrix, but Scaling and Rotation are 2x2 Matrices
- Let's homogenize! Doesn't affect linearity property of scaling and rotation
- Our new transformation matrices look like this…

Transformation	Matrix
Scaling	$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Rotation	$\begin{bmatrix} \cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Translation	$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix}$

- Note: These 3 transformations are called affine transformations

Examples

- Scaling: *Scale by 15 in the x direction, 17 in the y*

$$\begin{bmatrix} 15 & 0 & 0 \\ 0 & 17 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Rotation: *Rotate by 123°*

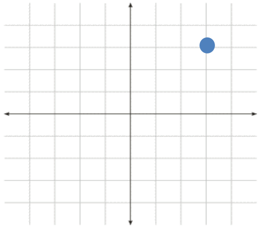
$$\begin{bmatrix} \cos(123) & -\sin(123) & 0 \\ \sin(123) & \cos(123) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

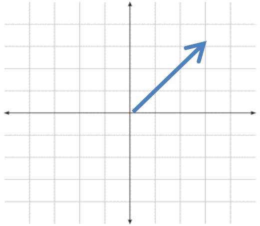
- Translation: *Translate by -16 in the x, +18 in the y*

$$\begin{bmatrix} 1 & 0 & -16 \\ 0 & 1 & 18 \\ 0 & 0 & 1 \end{bmatrix}$$

Before we continue! Vectors vs. Points

- Up until now, we've only used the notion of a point in our 2D space
- We now present a distinction between points and vectors





- We used **Homogeneous coordinates** to more conveniently represent translation; hence points are represented as $(x, y, 1)^T$
- A vector can be rotated/scaled, not translated (always starts at origin), don't use the Homogeneous coordinate, $(x, y, 0)^T$
- For now, let's focus on just our points (typically vertices)

Inverses

- How do we find the inverse of a transformation?
- Take the inverse of the transformation matrix (thanks to homogenization, they're all invertible!):

Transformation	Matrix Inverse	Does it make sense?
Scaling	$\begin{bmatrix} 1/s_x & 0 & 0 \\ 0 & 1/s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	If you scale something by factor X, the inverse is scaling by 1/X
Rotation	$\begin{bmatrix} \cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$	Not so obvious, but can use math! Rotation Matrix is orthonormal, so inverse should just be the transpose
Translation	$\begin{bmatrix} 1 & 0 & -dx \\ 0 & 1 & -dy \\ 0 & 0 & 1 \end{bmatrix}$	If you translate by X, the inverse is translating by -X

Composition of Transformations (2D) (1/2)

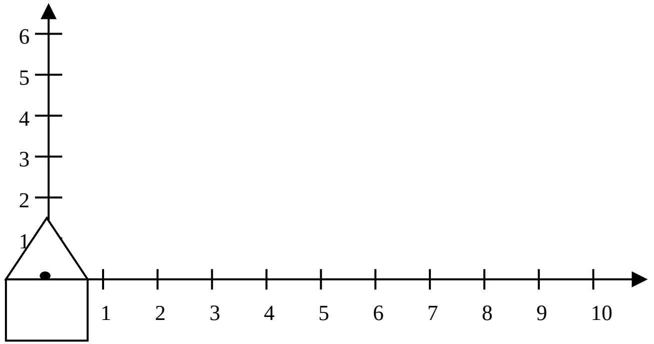
- We now have a number of tools at our disposal, we can combine them!
- An object in a scene uses many transformations in sequence, how do we represent this in terms of functions?
- Transformation is a function; by associativity we can compose functions: $(f \circ g)(i)$
- This is the same as first applying g to some input i and then applying f: $(f(g(i)))$
- Consider our functions f and g as matrices (M_1 and M_2 respectively) and our input as a vector ($\boldsymbol{v}$)
- Our composition is equivalent to $M_1 M_2 v$

Composition of Transformations (2D) (2/2)

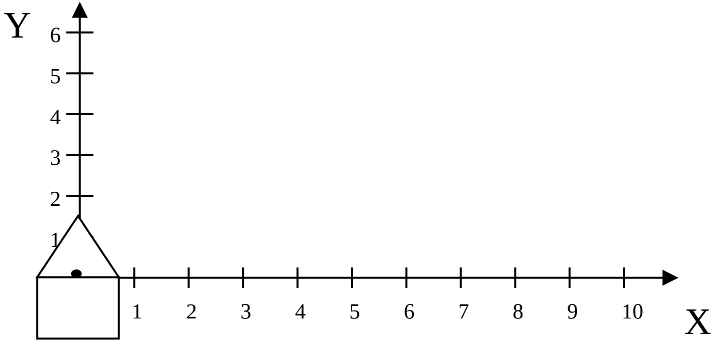
- We can now form compositions of transformation matrices to form a more complex transformation
- For example, $\boldsymbol{TRSv}$, which scales point, then rotates, then translates:
- $$\begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathrm{s}_x & 0 & 0 \\ 0 & \mathrm{s}_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$
- Note that we apply the matrices in sequence right to left, but practically, given associativity, we can compose them and apply the composite to all the vertices in, say, a mesh.
- **Important: Order Matters!**
- Matrix Multiplication is **not commutative.**
- Be sure to check out the Transformation Game at http://www.cs.brown.edu/exploratories/freeSoftware/repository/edu/brown/cs/exploratories/applets/transformationGame/transformation_game_guide.html
- Let's see an example…

Not commutative

Translate by x=6, y=0 then rotate by 45º

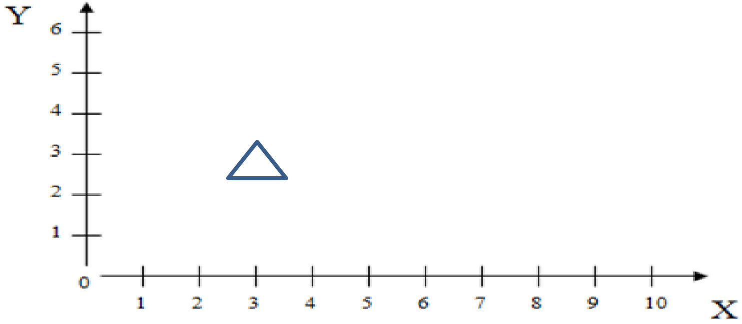


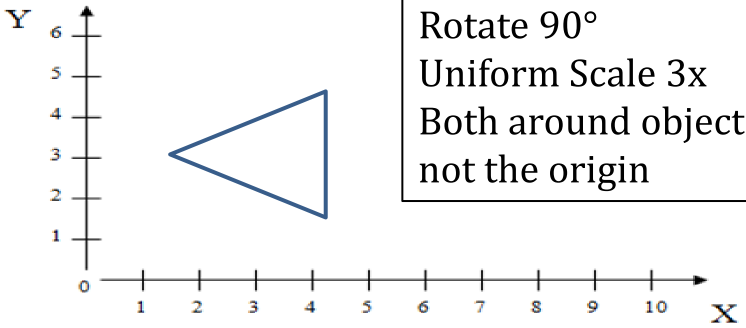
Rotate by 45º then translate by x=6, y=0



Composition (an example) (2D) (1/2)

- Start: Goal:

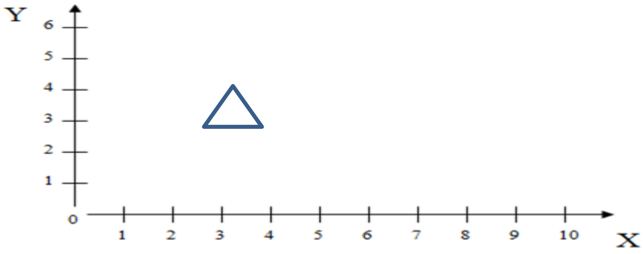




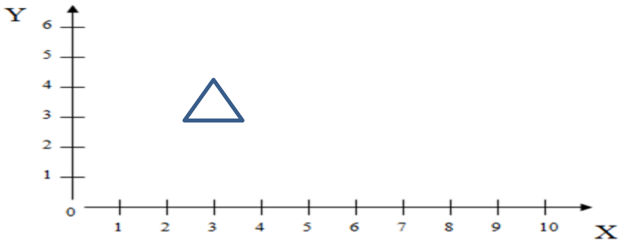
Rotate 90°
Uniform Scale 3x
Both around object's center, not the origin

- Important concept: Make the problem simpler
- Translate object to origin first, scale , rotate, and translate back:
  - $\boldsymbol{T^{-1}RST} = \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 90 & -sin90 & 0 \\ sin90 & cos90 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$
- Apply to all vertices

Composition (an example) (2D) (2/2)



- $\boldsymbol{T^{-1}RST}$
- But what if we mixed up the order? Let's try $\boldsymbol{RT^{-1}ST}$
  - $\begin{bmatrix} \cos 90 & -sin90 & 0 \\ sin90 & cos90 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$
- Oops! We managed to scale it properly but when we rotated it we rotated the object about the origin, not its own center, shifting its position…Order Matters!



Inverses Revisited

- What is the inverse of a sequence of transformations?

$$(\boldsymbol{M_1 M_2 \dots M_n})^{-1} = \boldsymbol{M_n^{-1} M_{n-1}^{-1} \dots M_1}$$

- Inverse of a sequence of transformations is the composition of the inverses of each transformation in reverse order

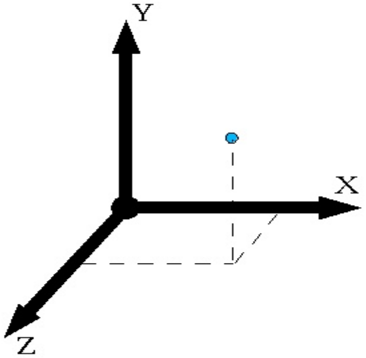
- Say we want to do the opposite transform of the example on Slide 26, what will our sequence look like?

$$\left(\boldsymbol{T^{-1}RST}\right)^{-1} = \boldsymbol{T^{-1}S^{-1}R^{-1}T}$$

- $\begin{bmatrix} 1 & 0 & 3 \\ 0 & 1 & 3 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 90 & sin90 & 0 \\ -sin90 & cos90 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -3 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{bmatrix}$
- We still translate to origin first, then translate back at the end!

Dimension++ (3D!)

- How should we treat geometric transformations in 3D?
- Just add one more coordinate/axis!



- A point is represented as $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$
- A matrix for a linear transformation $\boldsymbol{T}$ can be represented as $\begin{bmatrix} \boldsymbol{T}(e1) & \boldsymbol{T}(e2) & \boldsymbol{T}(e3) \end{bmatrix}$ where $e3$ corresponds to the z-coordinate, $\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$
- But remember to use homogeneous coordinates! Thus embed the scale and rotation matrices upper left submatrix, translation vector upper right column

Transformations in 3D

Transformation	Matrix	Comments
Scaling	$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Looks just like the 2D version right? We just added an s_z term.
Rotation	See next slide	This one's more complicated. In 2D there is only one axis of rotation. In 3D there are infinitely many, thus the matrix has to take into account all possible axes. See next slide…
Translation	$\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$	Similar to the 2D version, we just have one more term, dz, representing change in the z axis

Rodrigues's Formula…

- Rotation by angle θ around vector $\boldsymbol{u} = \begin{bmatrix} \boldsymbol{u_x} \\ \boldsymbol{u_y} \\ \boldsymbol{u_z} \end{bmatrix}$

  Note: This is an arbitrary **unit** vector $\boldsymbol{u}$ in xyz space

- Here's a not so friendly rotation matrix:

$$R = \begin{bmatrix} \cos\theta + u_x^2(1-\cos\theta) & u_x u_y(1-\cos\theta) - u_z\sin\theta & u_x u_z(1-\cos\theta) + u_y\sin\theta \\ u_y u_x(1-\cos\theta) + u_z\sin\theta & \cos\theta + u_y^2(1-\cos\theta) & u_y u_z(1-\cos\theta) - u_x\sin\theta \\ u_z u_x(1-\cos\theta) - u_y\sin\theta & u_z u_y(1-\cos\theta) + u_x\sin\theta & \cos\theta + u_z^2(1-\cos\theta) \end{bmatrix}.$$

- This is called the coordinate form of Rodrigues's formula
- Let's try a different way…

Rotating axis by axis (1/2)

- Every rotation can be represented as the composition of 3 different angles of **counter-clockwise** rotation around 3 axes, namely
  - x-axis in the yz plane by ψ
  - y-axis in the xz plane by θ
  - z-axis in the xy plane by ϕ
- Also known as Euler angles, makes problem of rotation much easier

$R_{xy}(\phi)$	$R_{yz}(\psi)$	$R_{xz}(\theta)$
$\begin{bmatrix} cos\phi & -sin\phi & 0 & 0 \\ sin\phi & cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\psi & -sin\psi & 0 \\ 0 & sin\psi & cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- $\boldsymbol{R_{yz}}$: rotation around x axis, $\boldsymbol{R_{xz}}$: rotation about y axis, $\boldsymbol{R_{xy}}$: rotation about z axis
- Note these differ only in where the 3x3 submatrix is embedded in the homogeneous matrix
- You can compose these matrices to form a composite rotation matrix

Rotating axis by axis (2/2)

- It would still be difficult to find the 3 angles to rotate by, given arbitrary axis $\boldsymbol{u}$ and specified angle ψ
- Solution? Make the problem easier by mapping $\boldsymbol{u}$ to one of the principal axes

- **Step 1:** Find a θ to rotate around y axis to put $\boldsymbol{u}$ in the xy plane

- **Step 2:** Then find a $\boldsymbol{\phi}$ to rotate around the z axis to align $\boldsymbol{u}$ with the x axis

- **Step 3:** Rotate by ψ around x axis = coincident $\boldsymbol{u}$ axis

- **Step 4:** Finally, undo the alignment rotations (inverse)

- Rotation Matrix: $\boldsymbol{M} = \boldsymbol{R}_{xz}^{-1}(\boldsymbol{\theta})\boldsymbol{R}_{xy}^{-1}(\boldsymbol{\phi})\boldsymbol{R}_{yz}(\boldsymbol{\psi})\boldsymbol{R}_{xy}(\boldsymbol{\phi})\boldsymbol{R}_{xz}(\boldsymbol{\theta})$

Inverses and Composition in 3D!

- Inverses are once again parallel to their 2D versions…

Transformation	Matrix Inverse
Scaling	$\begin{bmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Rotation	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\psi & sin\psi & 0 \\ 0 & -sin\psi & cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\phi & sin\phi & 0 & 0 \\ -sin\phi & cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} cos\theta & 0 & -sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Translation	$\begin{bmatrix} 1 & 0 & 0 & -dx \\ 0 & 1 & 0 & -dy \\ 0 & 0 & 1 & -dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Composition works exactly the same way…

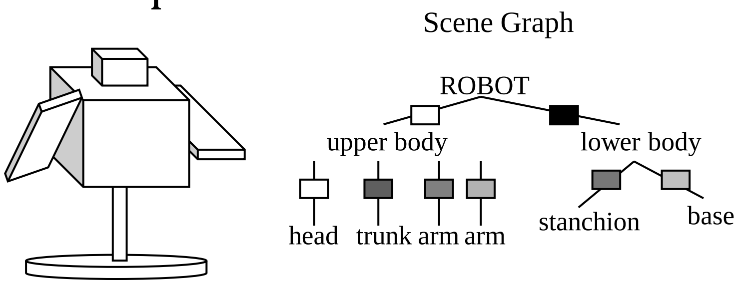
Example in 3D!

- Let's take some 3D object, say a cube, centered at (2,2,2)
- Rotate in object's space by 30° around x axis, 60° around y and 90° around z
- Scale in object space by 1 in the x, 2 in the y, 3 in the z
- Translate by (2,2,4) in world space
- Transformation Sequence: $\boldsymbol{T}\boldsymbol{T}_{\mathbf{0}}^{-\mathbf{1}}\boldsymbol{S}_{\boldsymbol{xy}}\boldsymbol{R}_{\boldsymbol{xy}}\boldsymbol{R}_{\boldsymbol{xz}}\boldsymbol{R}_{\boldsymbol{yz}}\boldsymbol{T}_{\mathbf{0}}$, where $\boldsymbol{T}_{\mathbf{0}}$ translates to (0,0)
- $$\begin{bmatrix}1 & 0 & 0 & 2\\ 0 & 1 & 0 & 2\\ 0 & 0 & 1 & 4\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 2\\ 0 & 1 & 0 & 2\\ 0 & 0 & 1 & 2\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 2 & 0 & 0\\ 0 & 0 & 3 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}cos90 & sin90 & 0 & 0\\ -sin90 & cos90 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$$\begin{bmatrix}cos60 & 0 & sin60 & 0\\ 0 & 1 & 0 & 0\\ -sin60 & 0 & cos60 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & cos30 & sin30 & 0\\ 0 & -sin30 & cos30 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0 & -2\\ 0 & 1 & 0 & -2\\ 0 & 0 & 1 & -2\\ 0 & 0 & 0 & 1\end{bmatrix}$$

Transformations and the scene graph (1/4)

- Objects can be complex:

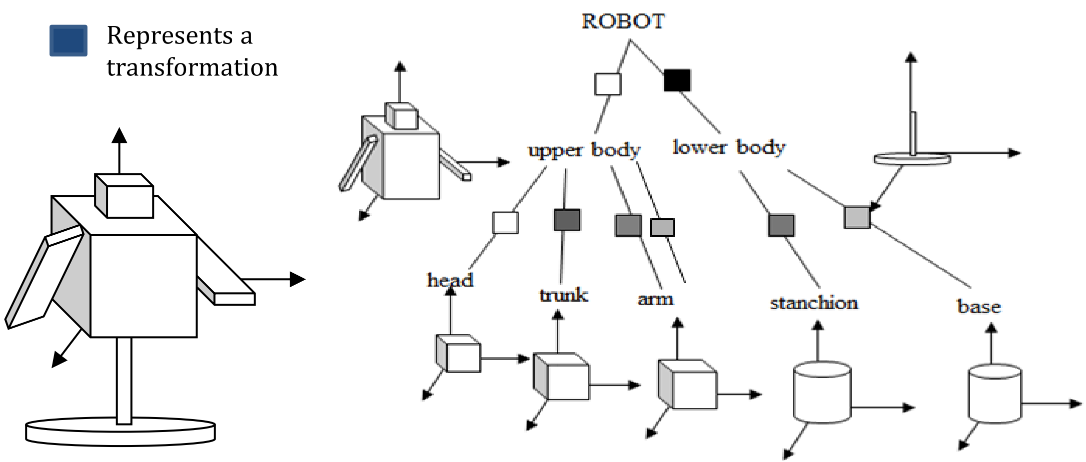


- 3D scenes are often stored in a directed acyclic graph (DAG) called a **scene graph**
  - WPF
  - Open Scene Graph
  - Sun's Java3D™
  - X3D ™ (VRML ™ was a precursor to X3D)
- Typical scene graph format:
  - **objects** (cubes, sphere, cone, polyhedra etc.): stored as nodes (default: unit size at origin)
  - **attributes** (color, texture map, etc.) stored as separate nodes
  - **transformations** are also nodes

Transformations and the scene graph (2/4)

- For your assignments use simplified format:
  - Attributes stored as a components of each object node (no separate attribute node)
  - Transform node affects its subtree
  - Only leaf nodes are graphical objects
  - All internal nodes that are not transform nodes are group nodes

Represents a transformation



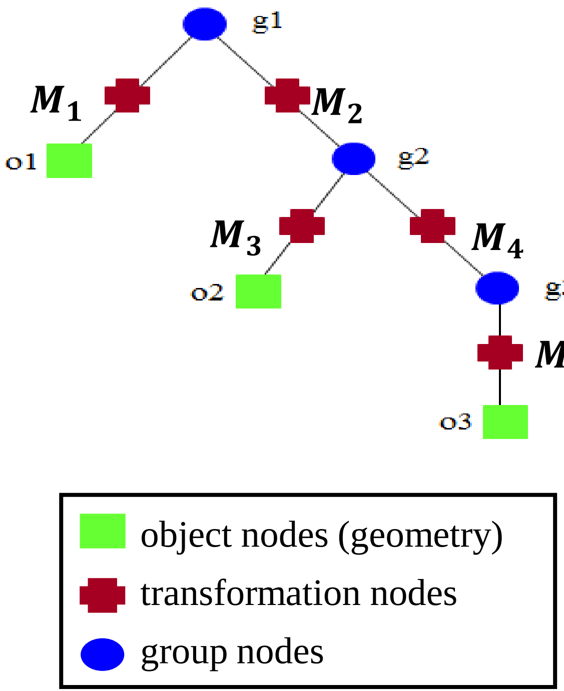
Step 1: Various transformations are applied to each of the leaves (object primitives—head, base, etc.)
Step 2: Transformations are then applied to groups of these objects as a whole (upper body, lower body)

Together this hierarchy of transformations forms the “robot” scene as a whole

Transformations and the scene graph (3/4)

- Notion of a cumulative transformation matrix that builds as you move up the tree ($\boldsymbol{CTM}$), appending higher level transformation matrices to the front of your sequence

- Example:
  - For o1, $\boldsymbol{CTM} = \boldsymbol{M_1}$
  - For o2, $\boldsymbol{CTM} = \boldsymbol{M_2}\,\boldsymbol{M_3}$
  - For o3, $\boldsymbol{CTM} = \boldsymbol{M_2}\,\boldsymbol{M_4}\boldsymbol{M_5}$
  - For a vertex v in o3, position in world (root) coordinate system is:
  - $\boldsymbol{CTM\,v} = (\boldsymbol{M_2 M_4 M_5})\,\boldsymbol{v}$



Transformations and the scene graph (4/4)

- You can reuse groups of objects (sub-trees) if they have been defined
- Group 3 has been used twice here
- Transformations defined within **group 3** itself are the same
- Different $\boldsymbol{CTM}$s for each use of **group 3** as a whole
- $\boldsymbol{T_0 T_1}$ $vs.$ $\boldsymbol{T_0 T_2 T_4}$