Transformations …

Sandro Spina

Computer Graphics and Simulation Group

Computer Science Department

University of Malta

What is a transform?

- A transform is an operation on points and vectors, and converts them in some pre-determined way.
- With transforms we can position, reshape and animate objects(set of points), lights (vectors), and cameras (vectors)
- Using matrices we can also project objects into a plane in different ways.
- Transforms are expressed as matrices.

Linear Transforms

- A *linear transform* is one that preserves vector addition and scalar multiplication.

- $\mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{y}) = \mathbf{f}(\mathbf{x} + \mathbf{y})$ Addition

- $k\mathbf{f}(\mathbf{x}) = \mathbf{f}(k\mathbf{x})$ Multiplication

- For eg. the scaling function $\mathbf{f}(\mathbf{x}) = 5\mathbf{x}$ takes a vector $\mathbf{x}$ and multiplies (all components) by 5. It is linear since the result of adding two vectors then multiplying them will be the same as first multiplying the two vectors then adding them.

Linear Transforms (ii)

- Scaling and rotation are both linear.
- The scaling and rotation transforms, which transform a 3-element vector, can be represented using a 3x3 matrix.
- However a function such as $\mathbf{f}(\mathbf{x}) = \mathbf{x} + (7,3,2)$ is not linear.
- $\mathbf{f}(\mathbf{x}+(7,3,2)) + \mathbf{f}(\mathbf{y}+(7,3,2)) \neq \mathbf{f}((\mathbf{x} + \mathbf{y}) + (7,3,2))$
- However it would be useful if we could combine scaling and rotation with translation ….

Affine Transforms

- In order to combine linear transforms with translation, affine transforms are used.
- An affine transform is a combination of a linear transform followed by a translation transform.
- The main characteristic of an affine transform is that it preserves parallel lines, but not necessarily lengths and angles.
- These transforms will be represented as 4 x 4 matrices using what is referred to as homogenous notation.

Homogenous Notation (i)

- Recall that, a point describes a location in space, whereas a vector describes a direction and has no location.
- Also using 3 x 3 matrices we can perform linear transformations such as rotations, scaling and shearing on coordinates.
- As we've already seen translation is not linear … hence cannot be performed using these matrices.

Homogenous Notation (ii)

- This problem in not important for vectors, since these do not represent a particular location (translation has no meaning), however translation is very important for points (for obvious reasons).

- The use of homogenous notation (as we shall see) allows us to incorporate translation (for points) in our matrices, i.e. using this notation we are able to represent affine transforms.

- We basically need to augment our 3x3 matrices (in the case of 3D) by one dimension to 4 x 4 matrices.

Homogenous Notation (iii)

- A homogenous vector is represented as follows:

  p = (p_x, p_y, p_z, p_w)

- In the case of points p_w is equal 1 and in the case of vectors p_w is equal to 0.

- When p_w is not equal to either 1 or 0, then the actual point (p_x, p_y, p_z) is calculated through homogenization, where all components are divided by p_w

- **P** is then equal to $(p_x/p_w, p_y/p_w, p_z/p_w, 1)$

Homogenous Notation (iv)

- The following represents how a 3 x 3 matrix M is augmented into the homogenous form:

$$M_{4x4} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & 0 \\ m_{10} & m_{11} & m_{12} & 0 \\ m_{20} & m_{21} & m_{22} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Rotation, scaling and shear matrices can replace **M** in this matrix, whereas all translation operations use the additional elements of the augmented matrix. We shall now see how to represent these transformations in the matrix above.

Homogenous Notation (v)

- Using homogenous notation we will be able to carry out more general affine transforms that contain translation such as:
  - Rotation about an axis that does not pass through the origin
  - Scale about a plane that does not pass through the origin
  - Reflection about a plane that does not pass through the origin
- We'll first have a look at the 'standard' transformation about the origin, but keep in mind that we could always first translate the 'centre' of the transformation to the origin, perform the linear transformation then translate back to the original position. Essentially performing the transformation $\mathbf{TRT}^{-1}$.

Orthogonal Matrices (Recall)

- A square matrix **M**, with only real elements, is orthogonal if and only if $\mathbf{M}\mathbf{M}^T = \mathbf{M}^T\mathbf{M} = \mathbf{I}$
- This means that the transpose of **M** is equal to the inverse of **M** , i.e. $\mathbf{M}^{-1} = \mathbf{M}^T$
- The standard basis is orthonormal, since the basis vectors are orthogonal to each other and of length one. Representing this basis as a matrix $\mathbf{E} = (\mathbf{e}_x\ \mathbf{e}_y\ \mathbf{e}_z) = \mathbf{I}$, gives us an orthogonal matrix.

Transforms Classification

- We can classify transforms as being either affine, orthogonal or both.
- *Translation* matrix (moves a point) is affine.
- *Rotation* matrix (rotates a set number of radians about an axis (x,y or z)) is both orthogonal and affine. Since it’s orthogonal than it’s inverse is simple to calculate as M^T
- *Scaling* matrix is affine.
- *Shearing* matrix is affine.

Translation Transform (visual)

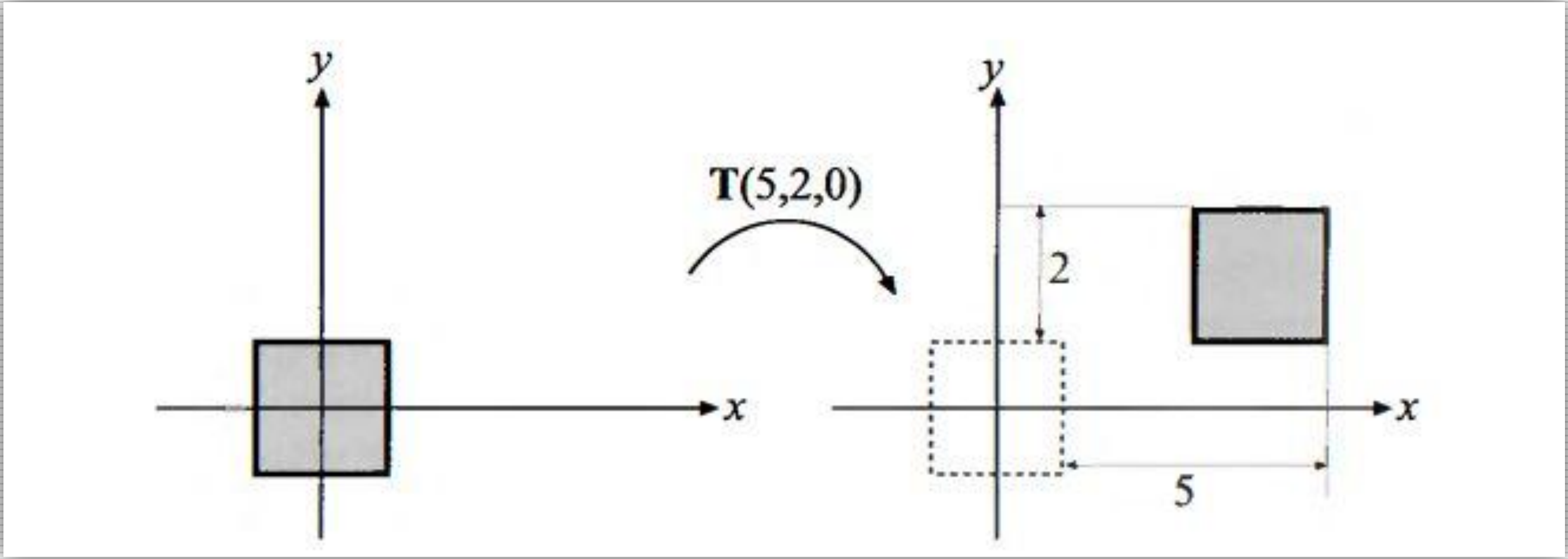


Image from Tomas Akenine-Moeller RTR book .. This chapter is mostly based On the transform chapter in this book.

Translation Transform (ii)

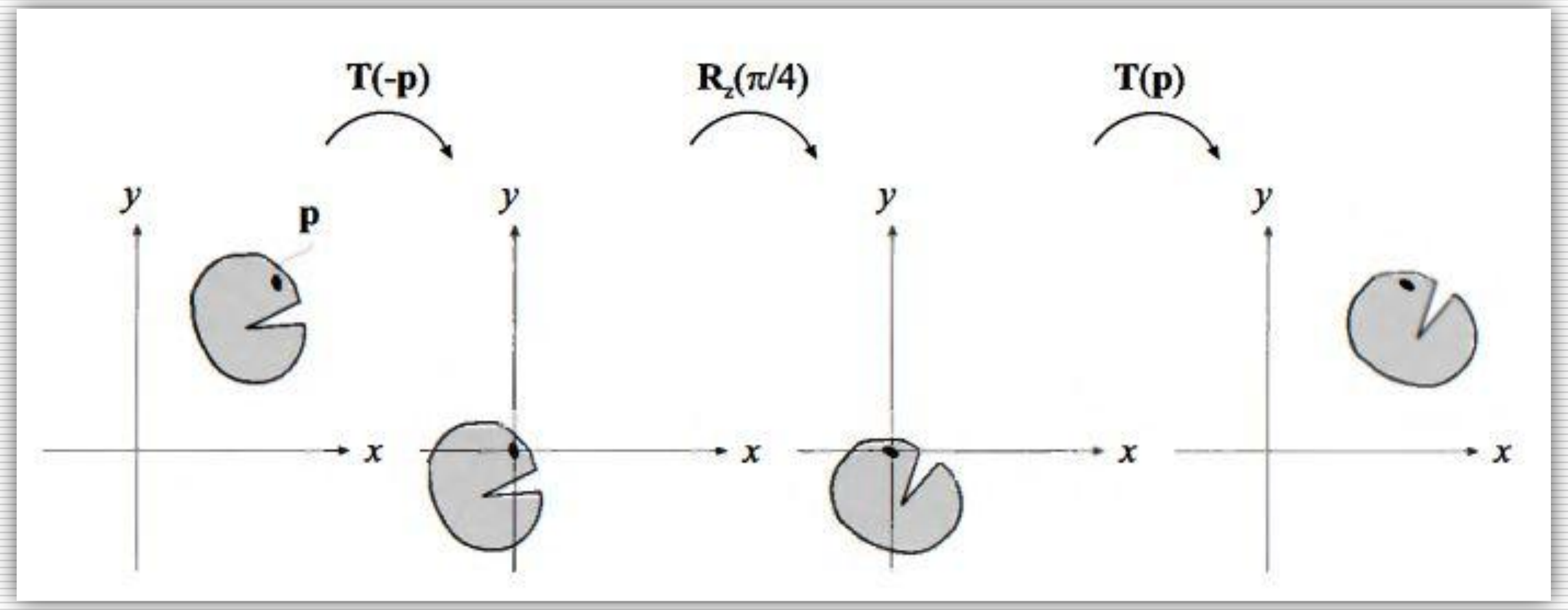
- A translation transform **T**(**t**) represented by the matrix **T**, moves a point from one location to another according to a vector **t** = (t_x, t_y, t_z)

$$T(t) = T(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Note that the multiplication of a point **p** with a matrix transform **T**(**t**), results in a new point **p'** equal to (p_x + t_x, p_y + t_y, p_z + t_z)

- The inverse $\mathbf{T}^{-1}$(**t**) is equal to **T**(-**t**).

Rotation Transform (visual)



Rotation Transform (i)

- A rotation transform **R**(θ) represented by the matrix **R**, rotates a point or a vector by a given angle θ, around a given axis passing through the origin.

- Like translation, rotation is a rigid-body transform, meaning that it preserves the distance between all points transformed and preserves handedness.

- Commonly used rotation matrices include :
  - $\mathbf{R}_x(\theta)$: rotate around the x-axis
  - $\mathbf{R}_y(\theta)$: rotate around the y-axis
  - $\mathbf{R}_z(\theta)$: rotate around the z-axis

Rotation Transform (ii)

$$R_x(\emptyset) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\emptyset) & -\sin(\emptyset) & 0 \\ 0 & \sin(\emptyset) & \cos(\emptyset) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_y(\emptyset) = \begin{pmatrix} \cos(\emptyset) & 0 & \sin(\emptyset) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\emptyset) & 0 & \cos(\emptyset) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$R_z(\emptyset) = \begin{pmatrix} \cos(\emptyset) & -\sin(\emptyset) & 0 & 0 \\ \sin(\emptyset) & \cos(\emptyset) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation Transform (iii)

- All rotation matrices have a determinant of one and are thus orthogonal. This also holds for concatenations of rotation matrices.
- Any arbitrary axis rotation can be represented as a series of transform around the standard basis.
- We shall also describe a way by which we shall be able to rotate points and vectors around an arbitrary axis directly.
- $\mathbf{R}_i^{-1}(\theta) = \mathbf{R}_i(-\theta)$

Rotation Transform (example)

- Assume that we want to rotate an object by θ radians around the z-axis, with the centre of rotation being a certain point **p**. What does this transform look like?

- We first need to translate the point **p** so that it lies on the origin of the coordinate system using the transform **T**(-**p**)
- Once the point is on the origin we can rotate by θ radians along the z-axis using the transform $\mathbf{R}_z(\theta)$.
- Finally we translate everything back by **T**(**p**)

- Therefore the resulting transformation **X** is equal to:
  - $\mathbf{T}(\mathbf{p})\ \mathbf{R}_z(\theta)\ \mathbf{T}(-\mathbf{p})$ pay attention to the order

Scaling Transform (i)

- Suppose I have an object (box) in my scene and I want to increase or decrease its size … I would need to apply a scaling transform to all the points which make up the box.

- A scaling matrix **S**(**s**) = **S**(s_x, s_y, s_z) scales an entity with factors s_x, s_y, and s_z along the x, y and z directions respectively. Scaling might not be uniform …

$$S(s) = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Scaling Transform (ii)

- We can also scale using cell (4,4) in our homogenous notation … and using homogenization.

$$S\,(5{,}5{,}5) = \begin{pmatrix} 5 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad S' = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1/5 \end{pmatrix}$$

- Divisions are usually quite inefficient therefore the first method is usually preferred, unless division is always performed un which case there is no extra cost

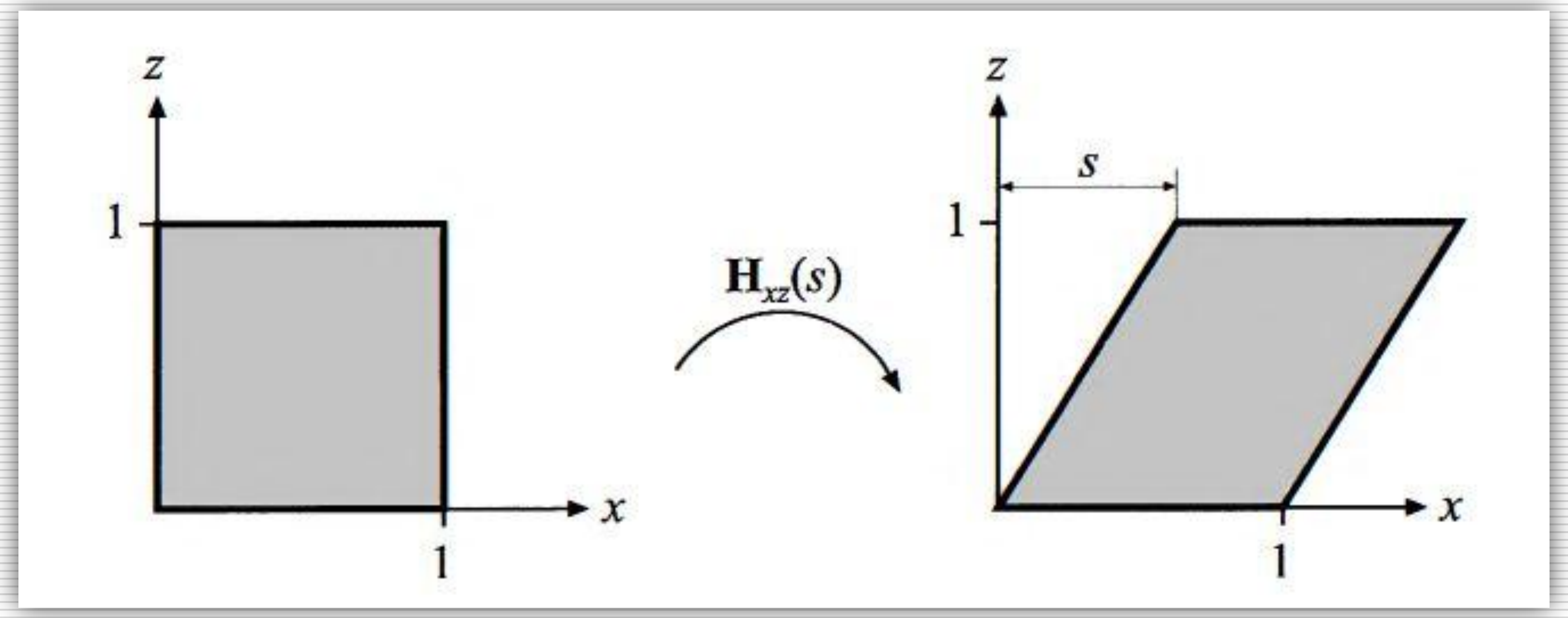
Scaling Transform (Reflection)

- In the case that one of the values in the scaling vector is negative then we are performing a reflection transform (or mirror).

- Which could be problematic since a reflection of points could change from clockwise to counterclockwise ordering of the points in a triangle … (we still need to check this out in later material)

- In any case it is important to detect whether a given matrix reflects in some manner. This can be done by checking whether the determinant of the 3 x 3 matrix is negative. If it is then we have a reflective matrix.

Shearing Transform (visual)



Shearing Transform (i)

- Shearing transforms are mainly used to create distortion.

- There are six basic shearing matrices denotes by :
  - $\mathbf{H}_{xy}(s)$ $\mathbf{H}_{xz}(s)$
  - $\mathbf{H}_{yx}(s)$ $\mathbf{H}_{yz}(s)$
  - $\mathbf{H}_{zx}(s)$ $\mathbf{H}_{zy}(s)$

- The first parameter denotes which coordinate is being changed by the shear matrix, whereas the second denotes the coordinate which does the shearing.

Shearing Transform (ii)

- One of the shearing matrices is shown below:

$$H_{xz}(s) = \begin{pmatrix} 1 & 0 & s & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- *S* is a scalar quantity … the effect of multiplying point **p** with the above is

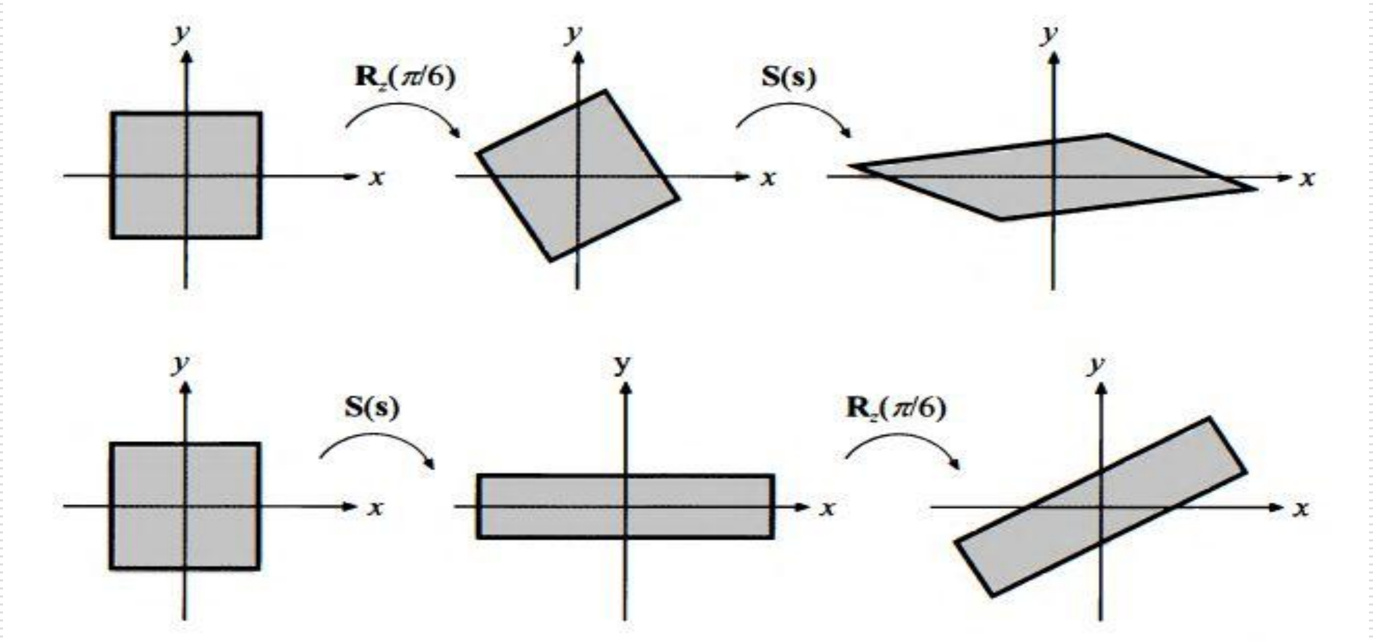
$$(p_x + sp_z, p_y, p_z)$$

Concatenation of Transforms

- Composition of transforms is very important because we usually want to represent a series of transformations in one matrix. Mainly for efficiency reasons.
- However the way (or rather order) we use to compose (concatenate) them together is very important.
- Recall that the multiplication operator on matrices is non-commutative.
- Concatenation (or composition) of transforms is thus said to be order-dependent.

Concatenation of Transforms

- Important to remember is that if we want to first scale a point then rotate it then translate it we need to concatenate the matrices as follows: (T(R(S(p))))

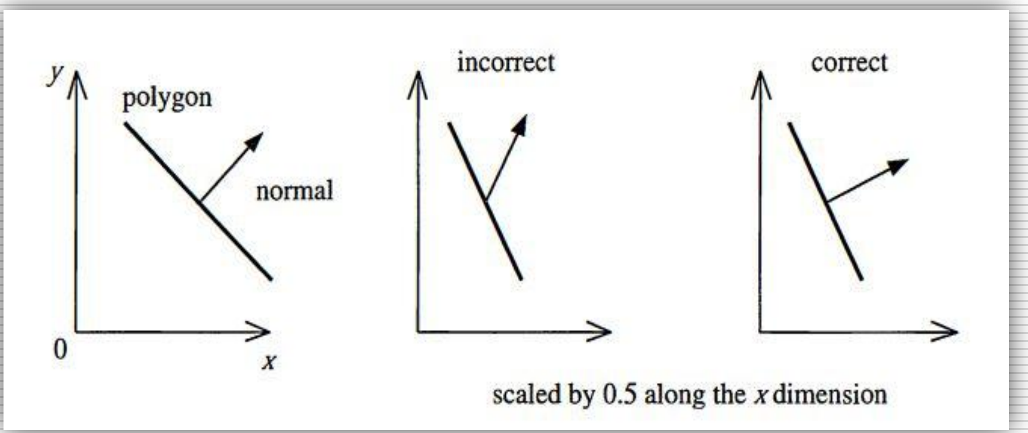


Rigid Body Transform

- A transform consisting only of translations and rotations is referred to as a rigid body.
- This means that the relation between all the points composing an object which is transformed are maintained after the transform.
- The inverse of a rigid body transform $\mathbf{X} = \mathbf{T}(t)\ \mathbf{R}$, is equal to the transform $\mathbf{X}^{-1} = \mathbf{R}^{-1}\ \mathbf{T}(\mathbf{t})^{-1} = \mathbf{R}^{T}\ \mathbf{T}(-\mathbf{t})$
- Rigid body transforms are many times used in physics engines in order to create physically based animations.

Normal Transform

- Matrices are used to transform many entities in CG including points, vectors, geometry, etc
- However attention need to be gmade when transforming the surface normals ... Check what happens in the diagram below.
- The proper method is to multiply by the transpose of the adjoint .. Derivation follows.



Normal Transform (Derivation)

- Take N to be the surface normal at a point $\mathbf{P}$
- Take $\mathbf{Q}$ to be a point tangential to point $\mathbf{P}$ ($T = \mathbf{Q} - \mathbf{P}$)
- This means that $N \,.\, T = 0$ i.e. $N^{\mathrm{T}} * T = 0$
- Take M to be the transform matrix applied to $\mathbf{P}$ and $\mathbf{Q}$, i.e. $\mathbf{P'} = \mathrm{M} * \mathbf{P}$ and $\mathbf{Q'} = \mathrm{M} * \mathbf{Q}$
- The new tangent vector $T' = \mathbf{P'} - \mathbf{Q'} = \mathrm{M} * \mathbf{Q} - \mathrm{M} * \mathbf{P}$
- Which is equal to $\mathrm{M} * (\mathbf{Q} - \mathbf{P}) = \mathrm{M} * T$
- Now define N' to be the new normal in the transformed space of P' and Q', thus $(N')^{\mathrm{T}} * T' = 0$
- Define R to be the matrix which transforms the normal N to N'
- Then we have, $0 = (N')^{\mathrm{T}} * T' = (\mathrm{R} * N)^{\mathrm{T}} * T' = (N^{\mathrm{T}} * \mathrm{R}^{\mathrm{T}}) * (\mathrm{M} * T) = N^{\mathrm{T}} * (\mathrm{R}^{\mathrm{T}} * \mathrm{M}) * T$
- Since we know that $N^{\mathrm{T}} * T = 0$ … we have $\mathrm{R}^{\mathrm{T}} * \mathrm{M} = 0$ which leads to $\mathrm{R}^{\mathrm{T}} = \mathrm{M}^{-1}$ and finally to $\mathrm{R} = (\mathrm{M}^{-1})^{\mathrm{T}}$

Euler Transforms (i)

- The Euler transform is essentially an intuitive way of specifying orientation …
- Many times it’s used to specify the direction and orientation a camera is looking at.
- It’s name comes from the Swiss mathematician Leonhard Euler (1707-1783)
- The transform is also used regularly in flight simulators …
- The transform is basically a sequence of rotations.

Euler Transforms (ii)

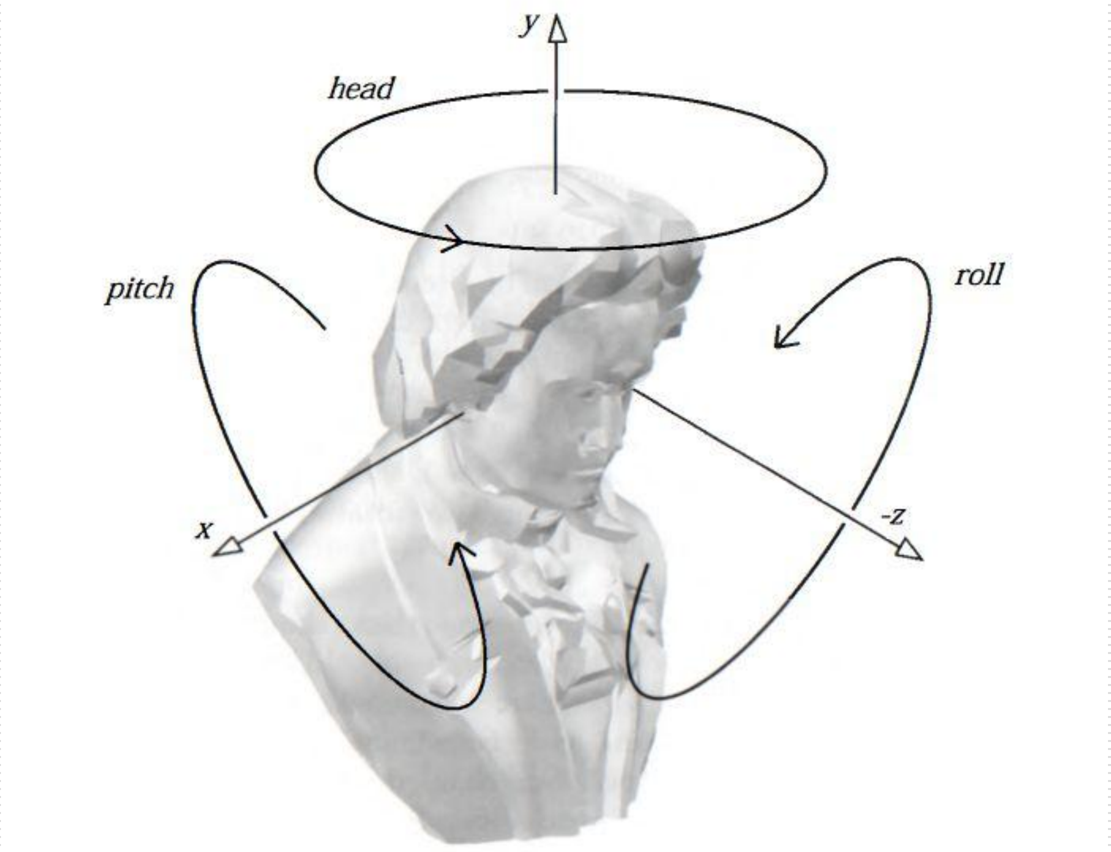
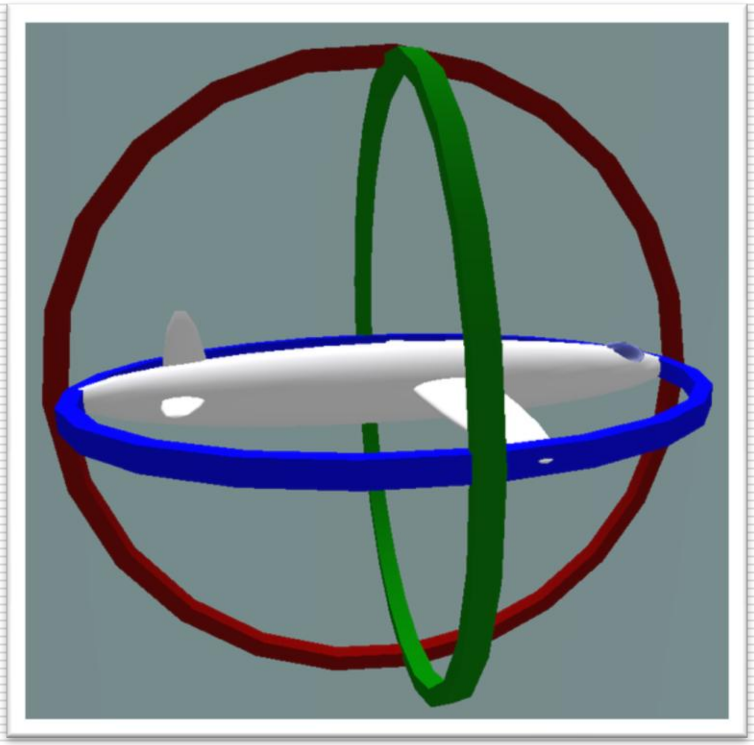


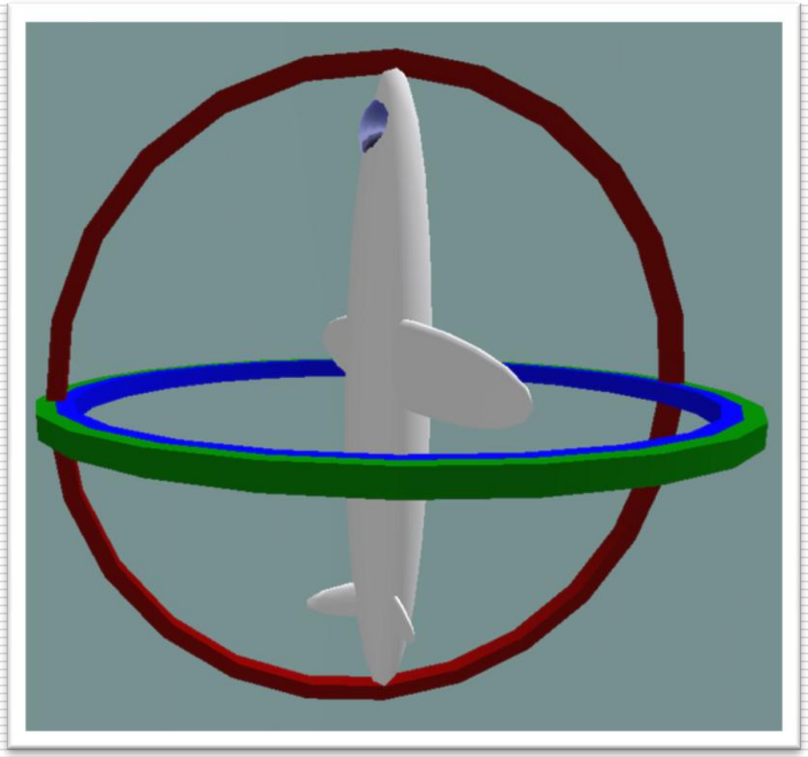
Figure 4.6. Depicting the way, in terms of the Euler transform, you turn your *head*, *pitch*, and *roll*. The default view direction is shown, looking along the negative z-axis with the head oriented along the y-axis.

Euler Transforms (i)

- $\mathbf{E}(h, p, r) = \mathbf{R}_z(r)\, \mathbf{R}_x(p)\, \mathbf{R}_y(h)$
- **E** is a concatenation of rotations along the basis axes. This makes **E** orthogonal.
- Therefore the $\mathbf{E}^{-1} = \mathbf{E}^T$
- h, p and r represent the amount of rotation around their respective axes.
- This transform is available in many Graphics APIs.

Gimbal Lock …

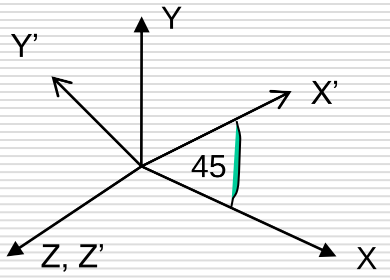




Gimbal Lock … example

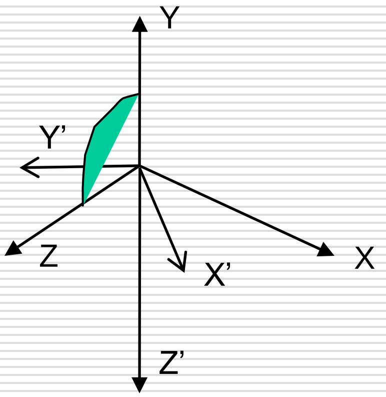
- Let's take an example where two axial systems are mutually aligned.

- After a roll (z-axis) of 45deg we get the alignment shown here …



- After a pitch (x-axis) of 90deg we get the alignment shown here …

- Now if we carry out a yaw of 45deg (y-axis) we would align back X' and X thus counteracting the effect of the original yaw !!



Rotation *(with respect to an arbitrary vector)*

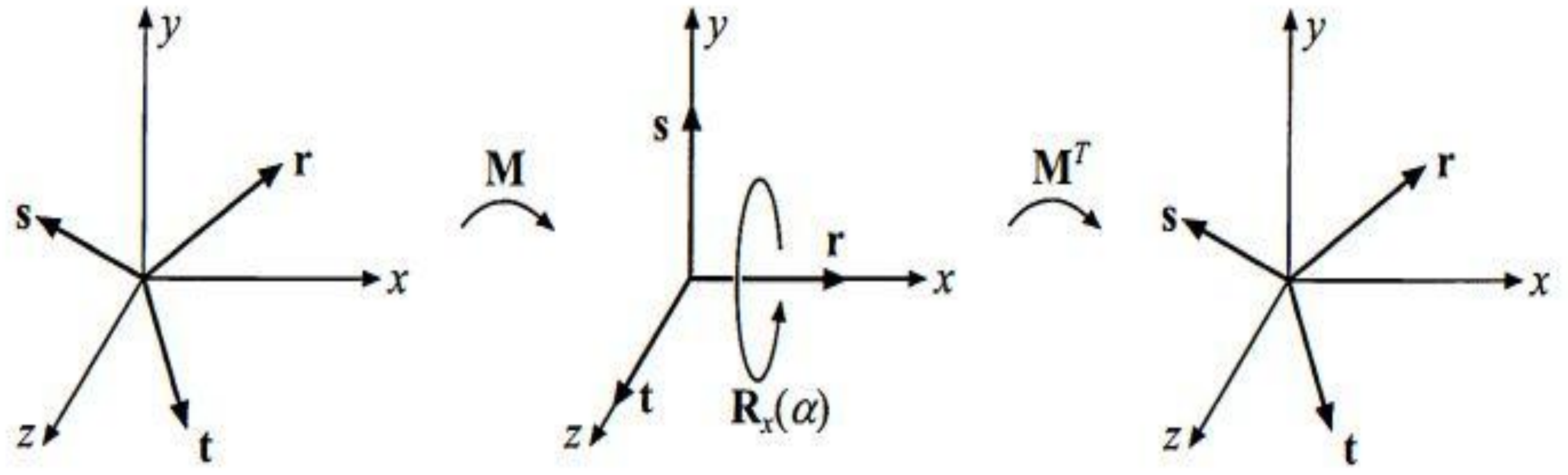


Figure 4.7. Rotation about an arbitrary axis, $\mathbf{r}$, is accomplished by finding an orthonormal basis formed by $\mathbf{r}$, $\mathbf{s}$, and $\mathbf{t}$. We then align this basis with the standard basis so that $\mathbf{r}$ is aligned with the x-axis. The rotation around the x-axis is performed there, and finally we transform back.

Change of Basis

- Suppose we want to express a vector **v** in a basis which is different from the one it is currently defined in (such as for example the standard x,y,z basis)

- Let's say we have vector **v**, described by the coordinate axes $\mathbf{e}_x$, $\mathbf{e}_y$, $\mathbf{e}_z$ and we also have another coordinate system described by the arbitrary basis vectors $\mathbf{f}_x$, $\mathbf{f}_y$, $\mathbf{f}_z$.

- If **w** is **v** expressed in basis **F** then we have:
  - $\mathbf{F}\mathbf{w} = (\mathbf{f}_x, \mathbf{f}_y, \mathbf{f}_z)\ \mathbf{w} = \mathbf{v}$
  - $\mathbf{w} = \mathbf{F}^{-1}\ \mathbf{v}$
  - If **F** is orthogonal then $\mathbf{w} = \mathbf{F}^{T}\ \mathbf{v}$