# Comp3320/6370 Computer Graphics

## Lecture: Transforms I

Mostly based on the book ”Real-time Rendering” and the lecture slides by Tomas Akenine-Möller et al.

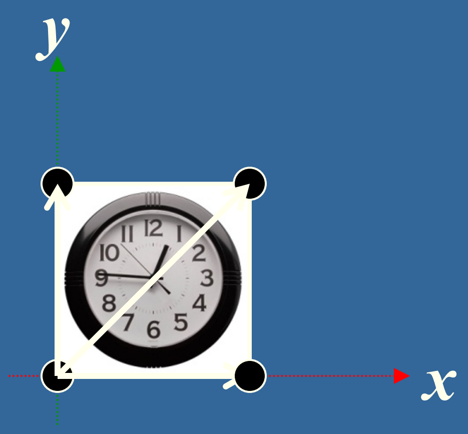
Some slides are from Hill’s book.

# Why transforms?

- We want to be able to **animate** objects and the camera
  - Translations
  - Rotations
  - Shears
  - And more…
- We want to be able to use **projection** transforms

# Motivation for Transforms

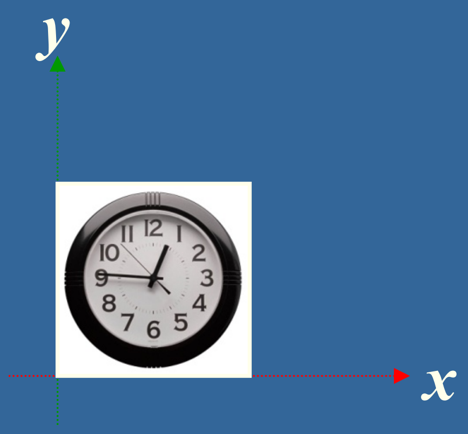
- We can model objects using a bunch of vectors.




(from Hill's book)

# Motivation for Transforms

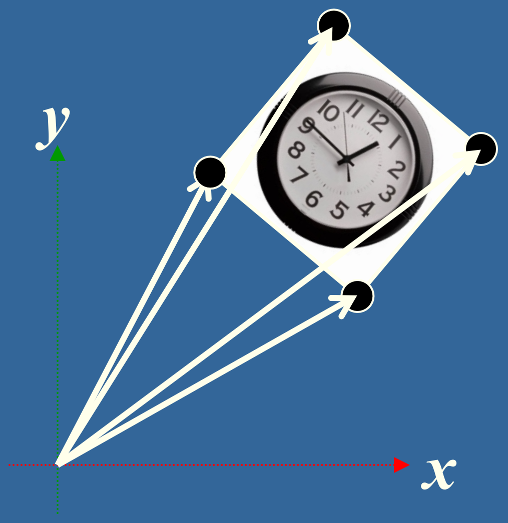
- We can model objects using a bunch of vectors.
- We want to move an object around...




(from Hill's book)

# Motivation for Transforms

- We can model objects using a bunch of vectors.
- We want to move it around... without having to build the object again from scratch.




(from Hill's book)

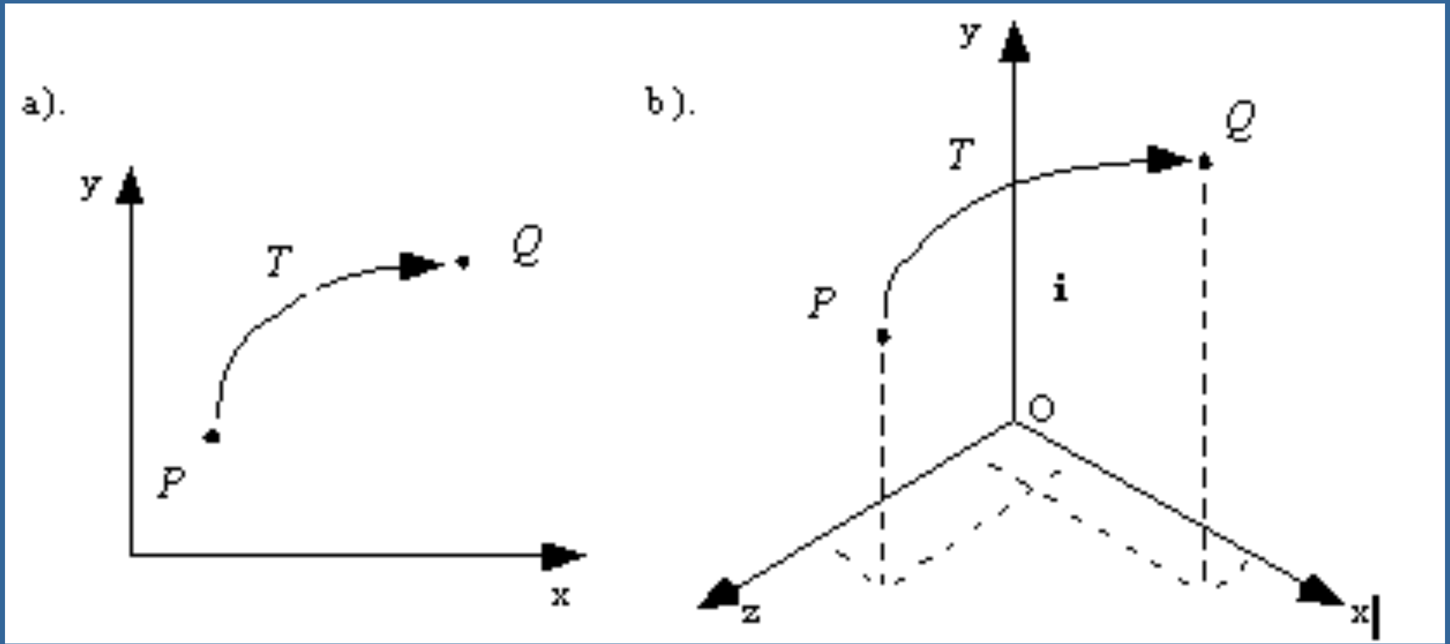
# Transformations

- Transformations change 2D or 3D points and vectors, or change coordinate systems.
  - An object transformation alters the coordinates of each point on the object according to the same rule, leaving the underlying coordinate system fixed.
  - A coordinate transformation defines a new coordinate system in terms of the old one, then represents all of the object's points in this new system.
- Object transformations are easier to understand, so we will do them first.

(from Hill's book)

# Transformations (2)

A (2D or 3D) transformation *T*( ) alters each point, *P* into a new point, *Q*, using a specific formula or algorithm: $Q = T(P)$.




(from Hill's book)

# Transformations (3)

- An arbitrary point *P* in the plane is **mapped** to *Q*.
- *Q* is the **image** of *P* under the mapping *T*.
- We transform an object by transforming each of its points, using the *same* function *T*() for each point.
- The **image** of line *L* under *T*, for instance, consists of the images of *all* the individual points of L.

(from Hill's book)

# Transformations (4)

- Most mappings of interest are **continuous**, so the image of a straight line is still a connected curve of some shape, although it's not necessarily a straight line.

- **Affine transformations**, however, *do* preserve lines: the image under $T$ of a straight line is also a straight line.

(from Hill's book)

# Transformations (5)

- We use an explicit "**coordinate frame**", also called an "**orthonormal basis**", when performing transformations.
- A coordinate frame consists of a point *O*, called the **origin**, and some mutually perpendicular vectors (called **i** and **j** in the 2D case; **i**, **j**, and **k** in the 3D case) that serve as the axes of the coordinate frame.
- Computer graphics uses homogeneous notation in order to distinguish points from vectors. We describe two 2D points by adding a "1" as a third "homogeneous" coordinate (and for vectors we would add a "0" as additional "homogeneous" coordinate.) Here we have two points in 2D:

$$\tilde{P} = \begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}, \tilde{Q} = \begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix}$$

(from Hill's book)

# Transformations (6)

- Recall that this means that point $P$ is at location $= P_x \mathbf{i} + P_y \mathbf{j} + O$, and similarly for $Q$.
- $P_x$ and $P_y$ are the coordinates of $P$.
- To get from the origin to point $P$, move amount $P_x$ along axis $\mathbf{i}$ and amount $P_y$ along axis $\mathbf{j}$.

(from Hill's book)

# Transformations (7)

- Suppose that transformation $T$ operates on any point P to produce point Q:
- $$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = T\left(\begin{pmatrix} P_x \\ P_y \\ 1 \end{pmatrix}\right)$$ or Q = T(P).
- T may be any transformation: e.g.,

$$\begin{pmatrix} Q_x \\ Q_y \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(P_x)e^{-P_x} \\ \dfrac{\ln(P_y)}{1+P_x^2} \\ 1 \end{pmatrix}$$

(from Hill's book)

# Transformations (8)

- To make **affine transformations** we restrict ourselves to much simpler families of functions, those that are *linear* in $P_x$ and $P_y$.
- Affine transformations make it easy to **scale, rotate, and reposition** figures.
- **Successive affine** transformations can be combined into a single overall affine transformation.

(from Hill's book)

# Transformations (9)

- Affine transformations have a **compact matrix** representation.
- Consequence of representing the vectors and points in homogeneous coordinates:
  - The matrix associated with an affine transformation operating on 2D vectors or points (in 2D space) must be a three-by-three matrix.
  - The matrix associated with an affine transformation operating on 3D vectors or points (in 3D space) must be a four-by-four matrix.

(from Hill’s book)

# Transformations (10)

- Affine transformations have a simple form.
- Example in 2D: Because the coordinates of Q are ***linear* combinations** of those of P, the transformed point may be written in the form:

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

(from Hill's book)

# Transformations (11)

- Every affine transformation is **composed of elementary operations**.
- A matrix  may be factored into a product of elementary matrices in various ways. One particular way of factoring the matrix associated with a 2D affine transformation yields

  M = (shear)(scaling)(rotation)(translation)
- That is, any 3 x 3 matrix  that represents a 2D affine transformation can be written as the product of (reading right to left) a translation matrix, a rotation matrix, a scaling matrix, and a shear matrix.

(from Hill's book)

# How implement transforms?

- Matrices!
- Can you really do everything with a matrix?
- Not everything, but a lot!
- We use 3x3 (for 2D homogeneous notation or for normal 3D notation) and 4x4 matrices (for 3D homogeneous notation).

$$\mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \qquad \mathbf{M} = \begin{pmatrix} m_{00} & m_{01} & m_{02} \\ m_{10} & m_{11} & m_{12} \\ m_{20} & m_{21} & m_{22} \end{pmatrix}$$

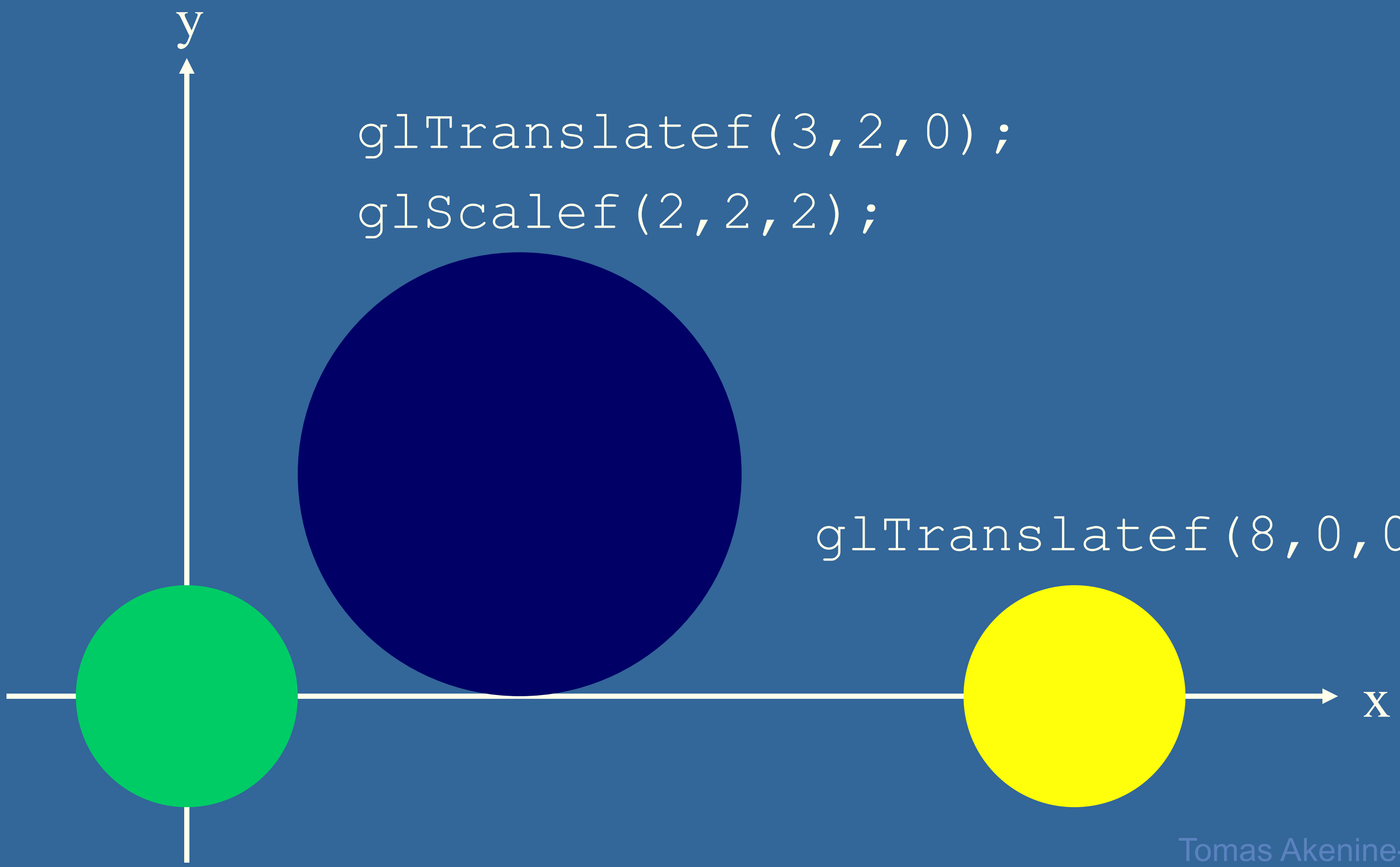
# How do I use transforms practically?

- Say you have a circle with origin at (0,0,0) and with radius 1 – i.e. the unit circle

- `glTranslatef(8,0,0);`
- `RenderCircle();`
- `glTranslatef(3,2,0);`
- `glScalef(2,2,2);`
- `RenderCircle();`

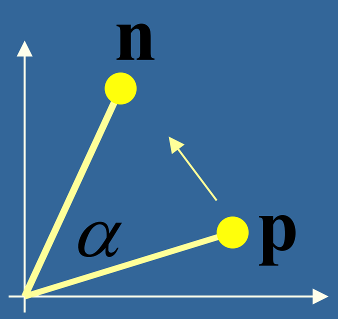
# Cont'd from previous slide
# A simple 2D example

- A circle in **model space**



# Derivation of rotation matrix in 2D

$$\mathbf{n} = \mathbf{R}_z\mathbf{p}?$$




$$\mathbf{p} = re^{i\phi} = r(\cos\phi + i\sin\phi) \quad \text{[rotation is mult by } e^{i\alpha}\text{]}$$

$$\mathbf{n} = e^{i\alpha}p = re^{i\alpha}e^{i\phi} =$$

$$= r[(\cos\alpha + i\sin\alpha)(\cos\phi + i\sin\phi)] =$$

$$= r(\cos\alpha\cos\phi - \sin\alpha\sin\phi) + ir(\cos\alpha\sin\phi + \sin\alpha\cos\phi)$$

$$= re^{i(\alpha+\phi)}$$

$$\mathbf{p} = (p_x, p_y)^T = (r\cos\phi, r\sin\phi)^T$$

$$\mathbf{n} = (n_x, n_y)^T = (r(\cos\alpha\cos\phi - \sin\alpha\sin\phi), r(\cos\alpha\sin\phi + \sin\alpha\cos\phi))^T$$

# Derivation 2D rotation, cont'd

$$\mathbf{p} = (p_x, p_y)^T = (r\cos\phi, r\sin\phi)^T$$

$$\mathbf{n} = (n_x, n_y)^T = [r(\cos\alpha\cos\phi - \sin\alpha\sin\phi),$$

$$r(\cos\alpha\sin\phi + \sin\alpha\cos\phi)]^T$$

$\mathbf{n} = \mathbf{R}_z\mathbf{p}$ what is $\mathbf{R}_z$?

$$\begin{pmatrix} n_x \\ n_y \end{pmatrix} = \underbrace{\begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}}_{\mathbf{R}_z} \begin{pmatrix} p_x \\ p_y \end{pmatrix}$$

# Rotations in 3D (standard notation)

- Same as in 2D for Z-rotations, but with a 3x3 matrix

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \Rightarrow \mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- For X

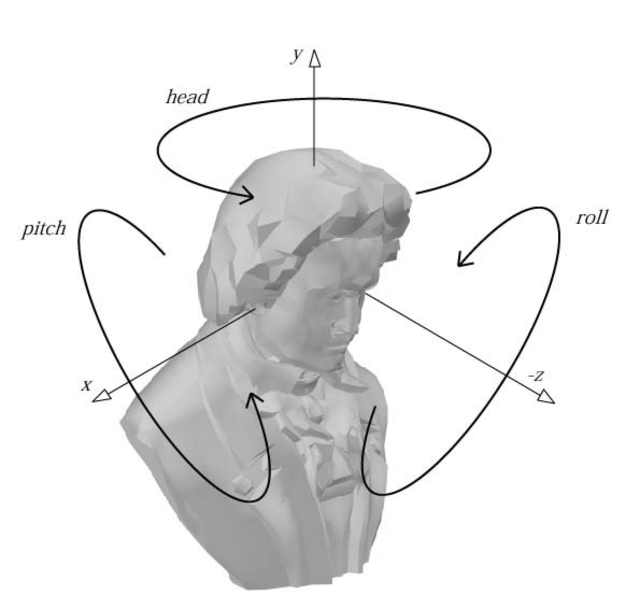
$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix}$$

- For Y

$$\mathbf{R}_y(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

# The Euler Transform

- Assume the view looks down the negative *z*-axis, with up in the *y*-direction, *x* to the right.




$$\mathbf{E}(h,p,r)=\mathbf{R}_z(r)\mathbf{R}_x(p)\mathbf{R}_y(h)$$

- *h*=head, *p*=pitch, *r*=roll are called "Euler angles".
- Gimbal lock can occur – looses one degree of freedom (Example: h=0,p=$\pi$/2, then the *z*-rotation becomes rotation around *y*-axis)

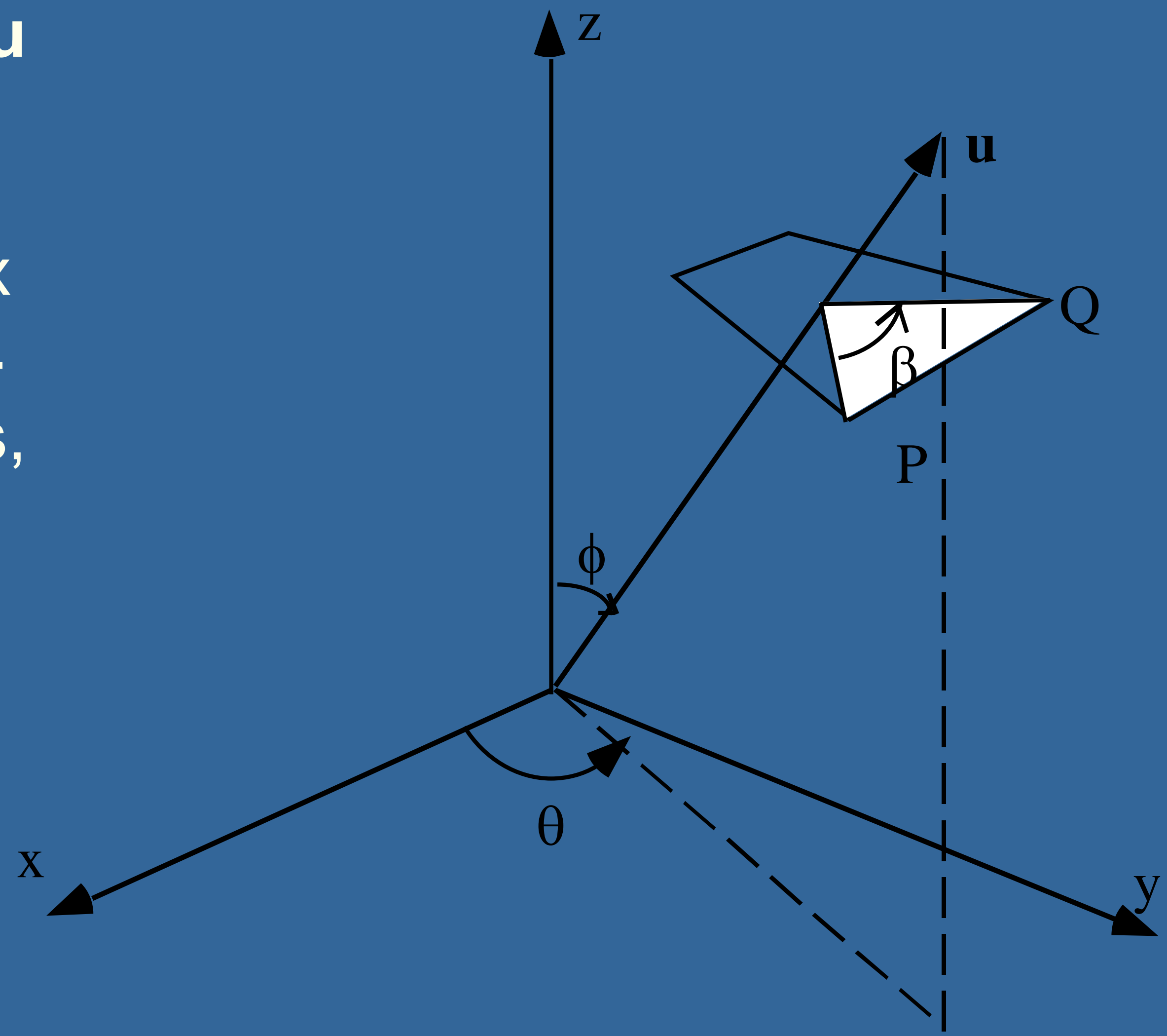
# Building Rotations

- ***Euler's Theorem: Any rotation (or sequence of rotations) about a point is equivalent to a single rotation about some coordinate axis through that point.***
- *Any* 3D rotation around an axis (passing through the origin) can be obtained from the product of five matrices for the **appropriate** choice of Euler angles; we shall see a method to construct the matrices.
- This implies that three values are required (and only three) to completely specify a rotation!

(from Hill's book)

# Rotating about an arbitrary axis

- We wish to rotate around axis **u** to make P coincide with Q.
- **u** can have any direction; it appears difficult to find a matrix that represents such a rotation.
- But it can be found in two ways, a classic way and a constructive way.




(from Hill's book)

# Rotating about an Arbitrary Axis (the classic way)

- ***The classic way:*** Decompose the required rotation into a sequence of known steps:
  - Perform two rotations so that **u** becomes aligned with the *z*-axis.
  - Do a *z*-roll through angle $\beta$.
  - Undo the two alignment rotations to restore **u** to its original direction.

- $R_{\mathbf{u}}(\beta) = R_z(\theta)\, R_y(\Phi)\, R_z(\beta)\, R_y(-\Phi)\, R_z(-\theta)$ is the desired rotation.

- (The constructive way will be addressed in one of the following lectures).

(from Hill's book)

# Rotating about an arbitrary axis

- Open-GL provides a rotation about an arbitrary axis:

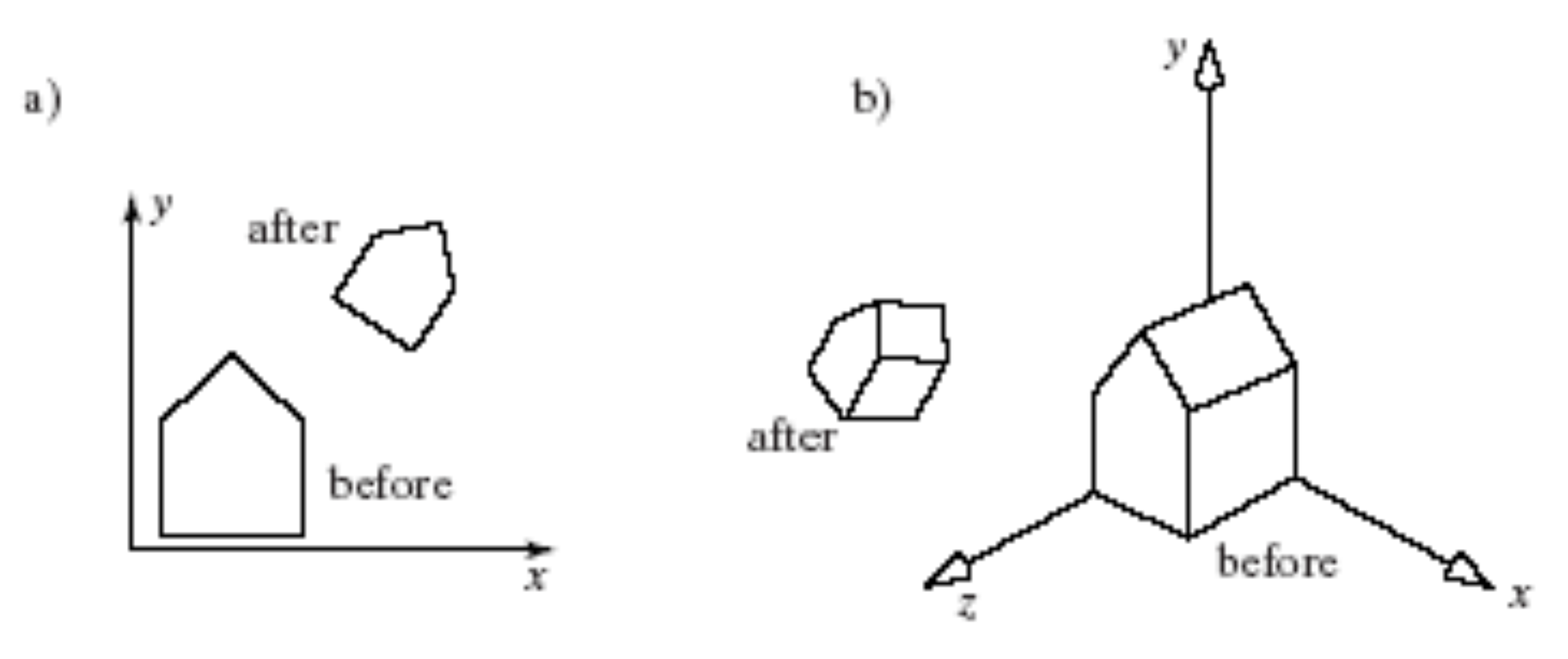
  ```
  glRotated (beta, ux, uy, uz);
  ```

- beta is the angle of rotation.
- ux, uy, uz are the components of a vector u normal to the plane containing P and Q.

(from Hill's book)

# Combinations of (Affine) Transformations

The house has been scaled, rotated and translated, in both 2D and 3D.




(from Hill's book)

# Translations must be simple?

Translation

$$\begin{pmatrix} ? & ? & ? \\ ? & ? & ? \\ ? & ? & ? \end{pmatrix} \mathbf{p} = \mathbf{p} + \mathbf{t}$$

Rotation

$$\mathbf{Rp} = \mathbf{n}$$

- Rotation is matrix mult, translation is add
- Would be nice if we could only use matrix multiplications…
- Turn to **homogeneous coordinates (!)**
- Add a new component to each vector

# Homogeneous notation for 3D space

- A point: $\mathbf{p} = \begin{pmatrix} p_x & p_y & p_z & 1 \end{pmatrix}^T$
- Translation becomes:

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\mathbf{T}(\mathbf{t})} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix} = \begin{pmatrix} p_x + t_x \\ p_y + t_y \\ p_z + t_z \\ 1 \end{pmatrix}$$

- A vector (direction): $\mathbf{d} = \begin{pmatrix} d_x & d_y & d_z & 0 \end{pmatrix}^T$
- Translation of vector: $\mathbf{Td} = \mathbf{d}$
- Also allows for projections (later)

# Rotations in 4x4 form

- Just add a row at the bottom, and a column at the right:

$$\mathbf{R}_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Similar for X and Y
- det( R )=1 (for 3x3 matrices)
- trace( R )=1+2cos(alpha)  (for any axis,3x3)

# 3D Rotations in homogeneous notation

- *z*-roll: the *x*-axis rotates to the *y*-axis.
- *x*-roll: the *y*-axis rotates to the *z*-axis.
- *y*-roll: the *z*-axis rotates to the *x*-axis.

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\vartheta & -\sin\vartheta & 0 \\ 0 & \sin\vartheta & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, R_y = \begin{pmatrix} \cos\vartheta & 0 & \sin\vartheta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\vartheta & 0 & \cos\vartheta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$R_z = \begin{pmatrix} \cos\vartheta & -\sin\vartheta & 0 & 0 \\ \sin\vartheta & \cos\vartheta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(from Hill's book)

# Rotations

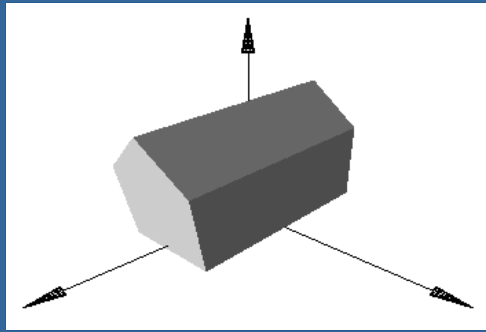
- Note that 12 of the terms in each matrix are the **zeros** and **ones** of the identity matrix.
- They guarantee that the corresponding coordinate of the point being transformed will not be altered.
- The *cos* and *sin* terms always appear in a rectangular pattern in the other rows and columns.

(from Hill's book)

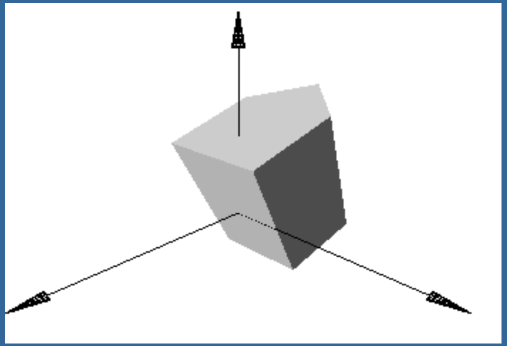
# Example

A barn in its original orientation, and after a -70° *x*-roll, a 30° *y*-roll, and a -90° *z*-roll.

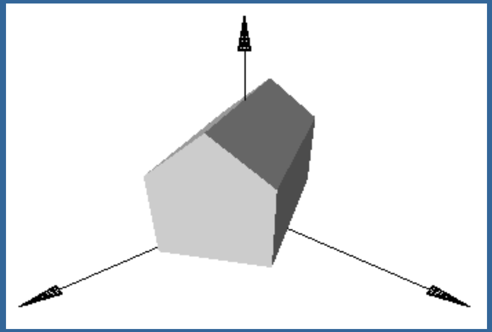
**a) the barn**



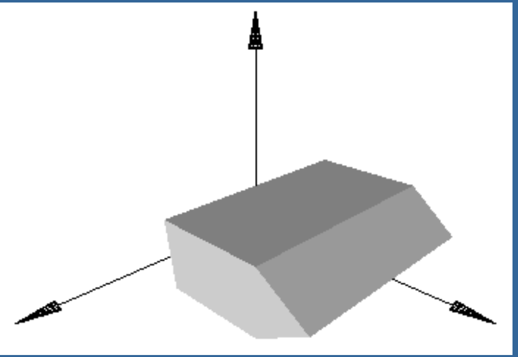
**b) $-70^0$ x-roll**



**c) $30^0$ y-roll**



**d) $-90^0$ z-roll**



(from Hill's book)

# Exercise 11 (Basic Rotations in 3D)

Describe the 3D rotation matrices around the $x$, $y$, and $z$ axis for 90, 180, 270, and 360 degrees (in standard and in homogeneous notation).

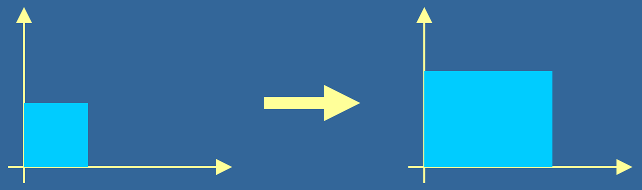
# Exercise 14 (Homogeneous coordinates)

Below several points (or vectors ?) are given in homogeneous coordinates. Determine for each of them the corresponding non-homogeneous 3D Cartesian coordinates.

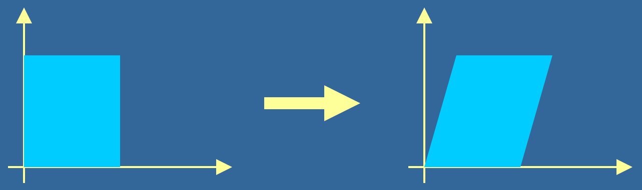
a) (3, 6, 5, 1)
b) (2, 4, 6, 4)
c) (0, 0, 2, 0.25)
d) (0, 0, 0, 1)
e) (1, 0, 0, 0)

# More basic transforms

- Scaling



- Shear



- Rigid-body: rotation then translation

$$\mathbf{X} = \mathbf{T}\mathbf{R}$$

- Concatenation of matrices
  - Not commutative, i.e., $\mathbf{R}\mathbf{T} \neq \mathbf{T}\mathbf{R}$
  - In $\mathbf{X} = \mathbf{T}\mathbf{R}$ , the rotation is done first

# 3D Transformations

We use **coordinate frames**, and suppose that we have an origin $O$ and three mutually perpendicular axes in the directions **i**, **j**, and **k**.

A point $P$ in this frame is given by $P = O + P_x\mathbf{i} + P_y\mathbf{j} + P_z\mathbf{k}$,

and a vector V by $V_x\mathbf{i} + V_y\mathbf{j} + V_z\mathbf{k}$.

$$P = \begin{pmatrix} P_x \\ P_y \\ P_z \\ 1 \end{pmatrix}, V = \begin{pmatrix} V_x \\ V_y \\ V_z \\ 0 \end{pmatrix}$$

# 3-D Transformations

The matrix representing a general 3D transformation in homogeneous notation is 4 x 4.

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

The fourth row of the matrix is a string of zeroes followed a lone one.

(from Hill's book)

# Translation and Scaling

Translation and scaling transformation matrices are given below. The values $s_x$, $s_y$, and $s_z$ cause scaling about the origin of the corresponding coordinates.

$$T = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, S = \begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(from Hill's book)

# Shear

- The shear matrix is given below.
  - a: y along x; b: z along x; c: x along y; d: z along y; e: x along z; f: y along z
- Usually only one of {a,…,f} is non-zero.

$$H = \begin{pmatrix} 1 & a & b & 0 \\ c & 1 & d & 0 \\ e & f & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

(from Hill’s book)

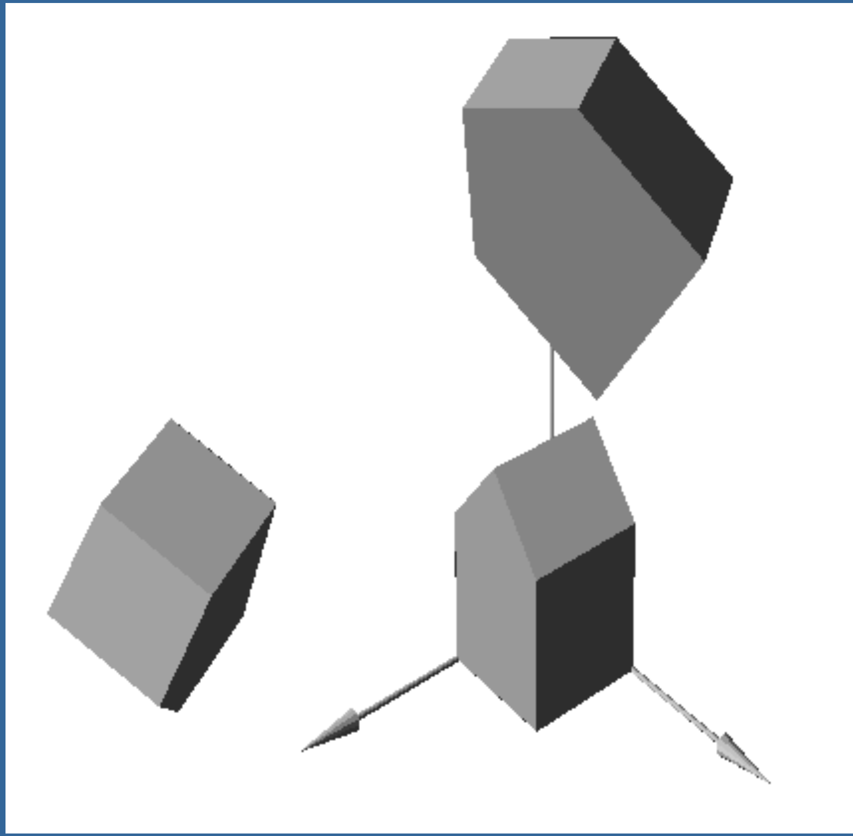
# Composing 3D (Affine) Transformations

- 3D affine transformations can be composed, and the result is another 3D affine transformation.
- The matrix of the **overall transformation** is the product of the individual matrices $M_1$ and $M_2$ that perform the two transformations: $\mathbf{M} = M_2 M_1$ (executed from right to left)
- Any number of affine transformations can be composed in this way, and a single matrix results that represents the overall transform(ation).

(from Hill's book)

# Example

- A barn is first transformed using some $M_1$, and the transformed barn is again transformed using $M_2$. The result is the same as the barn transformed once using $M_2M_1$.



(from Hill's book)

# Building Rotations

- Two 2D rotations combine to make a rotation given by the sum of the rotation angles, and the matrices commute.
- In 3D the situation is more **complicated**, because rotations can be about different axes. Here the order in which two rotations about different axes are performed *does* matter: **3D rotation matrices do not commute**.

**Homework:**

Is the set of matrices together with “+” or “.” a group? Any differences for 2D or 3D?

# Literature

- Tomas Akenine-Möller, Eric Haines, Naty Hoffman: **Real-time rendering**. AK Peters, 3rd edition, 2008, ISBN 978-156881-424-7.

  [This is an excellent but also slightly more advanced graphics book. Please check out the associated webpage at http://www.realtimerendering.com/ . It is a source of up to date information and additional links.].
- Tomas Akenine-Möller, Eric Haines, Naty Hoffman, Angelo Pesce, Sebastien Hillaire, Michal Iwanicki: **Real-time rendering**. AK Peters/CRC Press, 4th edition, 2018, ISBN 9781138627000.
- F.S. Hill: **Computer graphics using OpenGL**. Prentice-Hall, 3rd edition, 2006.