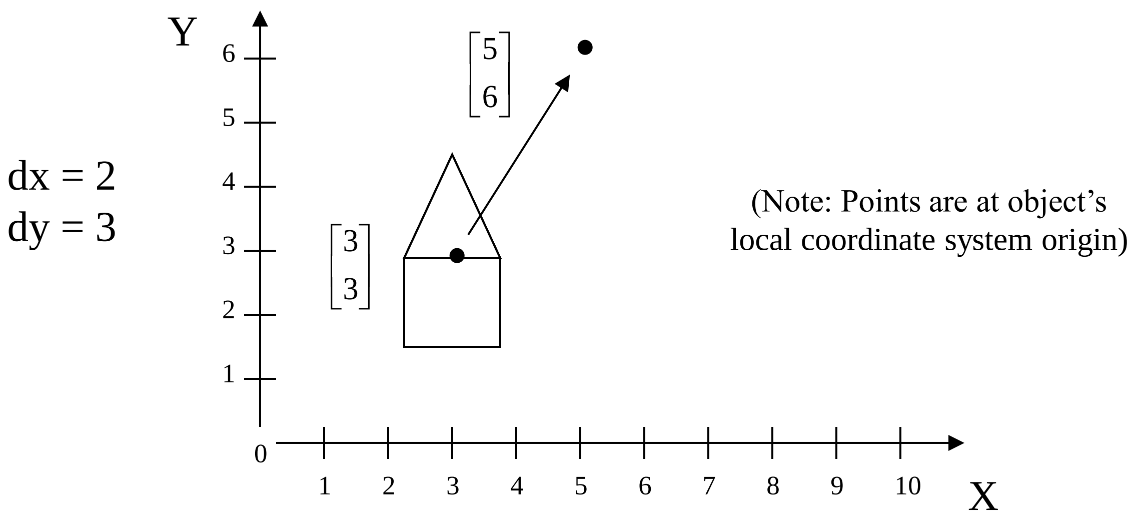
Geometric Transformations

2D Translation



- Component-wise addition of vectors

  $\boldsymbol{v}' = \boldsymbol{v} + \boldsymbol{t}$ where $v = \begin{bmatrix} x \\ y \end{bmatrix},\quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix},\quad t = \begin{bmatrix} dx \\ dy \end{bmatrix}$

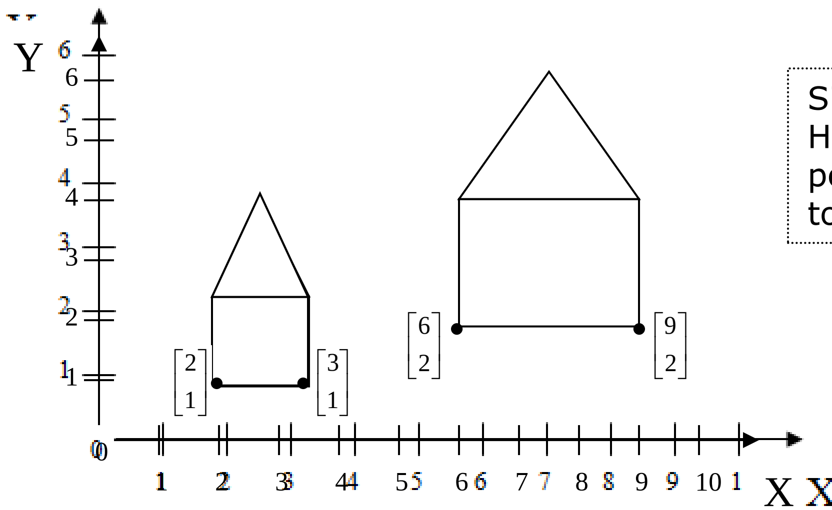
  and $x' = x + dx$

  $y' = y + dy$

  To move polygons: translate vertices (vectors) and redraw lines between them
- Preserves lengths (isometric)
- Preserves angles (conformal)

2D Scaling

$$s_x = 3$$
$$s_y = 2$$



Side effect: House shifts position relative to origin

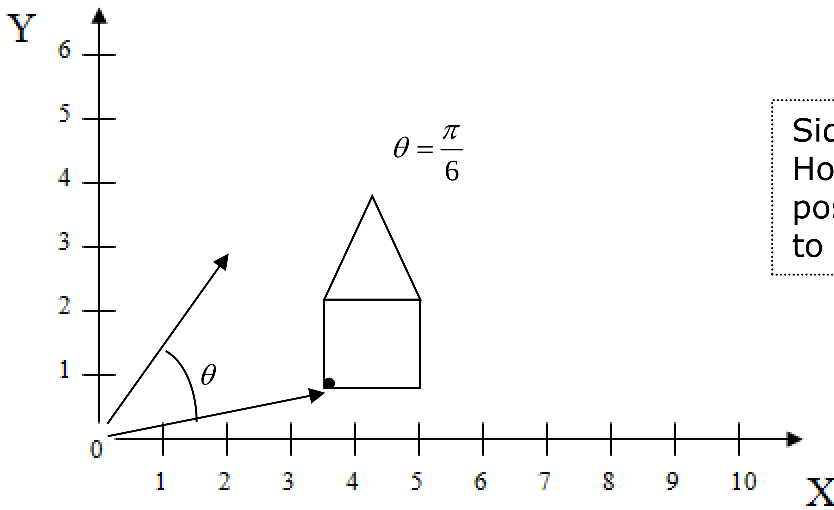
- Component-wise scalar multiplication of vectors

$\boldsymbol{v'} = \boldsymbol{Sv}$ where $v = \begin{bmatrix} x \\ y \end{bmatrix}, \quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

and $S = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}$ $\quad x' = s_x x$, $\quad y' = s_y y$

- Does not preserve lengths
- Does not preserve angles (except when scaling is uniform)

2D Rotation



Side effect: House shifts position relative to origin

- Rotation of vectors through an angle θ

$\boldsymbol{v'} = \boldsymbol{R_\theta}\, \boldsymbol{v}$ where $v = \begin{bmatrix} x \\ y \end{bmatrix}, \quad v' = \begin{bmatrix} x' \\ y' \end{bmatrix}$

and $x' = x\cos\Theta - y\sin\Theta$
$y' = x\sin\Theta + y\cos\Theta$

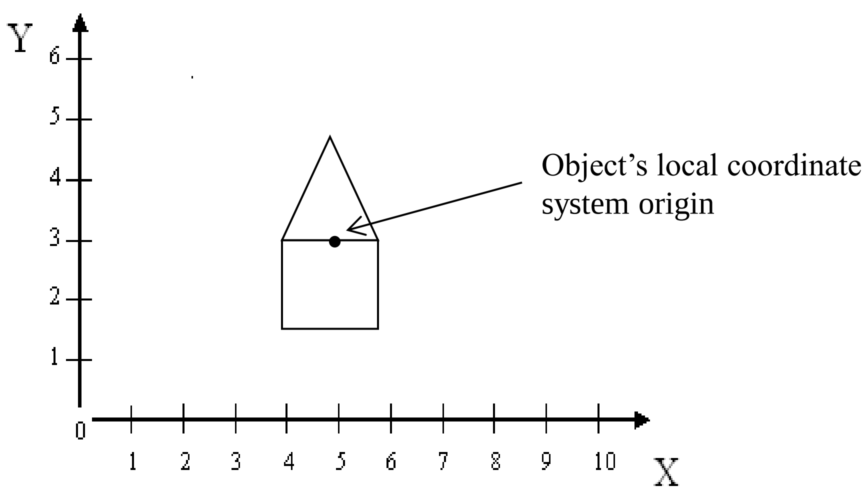
$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

NB: A rotation by 0 angle, i.e. no rotation at all, gives us the identity matrix

- Proof by sine and cosine summation formulas
- Preserves lengths in objects, and angles between parts of objects

2D Rotation and Scale are Relative to Origin

- Suppose object is not centered at origin and we want to scale and rotate it.
- Solution: move to the origin, scale and/or rotate *in its local coordinate system*, then move it back.



- This sequence suggests the need to compose successive transformations…

Homogenous Coordinates

- Translation, scaling and rotation are expressed as:

translation: $v' = v + t$

scale: $v' = Sv$

rotation: $v' = Rv$

- Composition is difficult to express
  - translation is not expressed as a matrix multiplication
- Homogeneous coordinates allows expression of all three transformations as 3x3 matrices for easy composition

$$P_{2d}(x, y) \rightarrow P_h(wx, wy, w), \quad w \neq 0$$

$$P_h(x', y', w), \quad w \neq 0$$

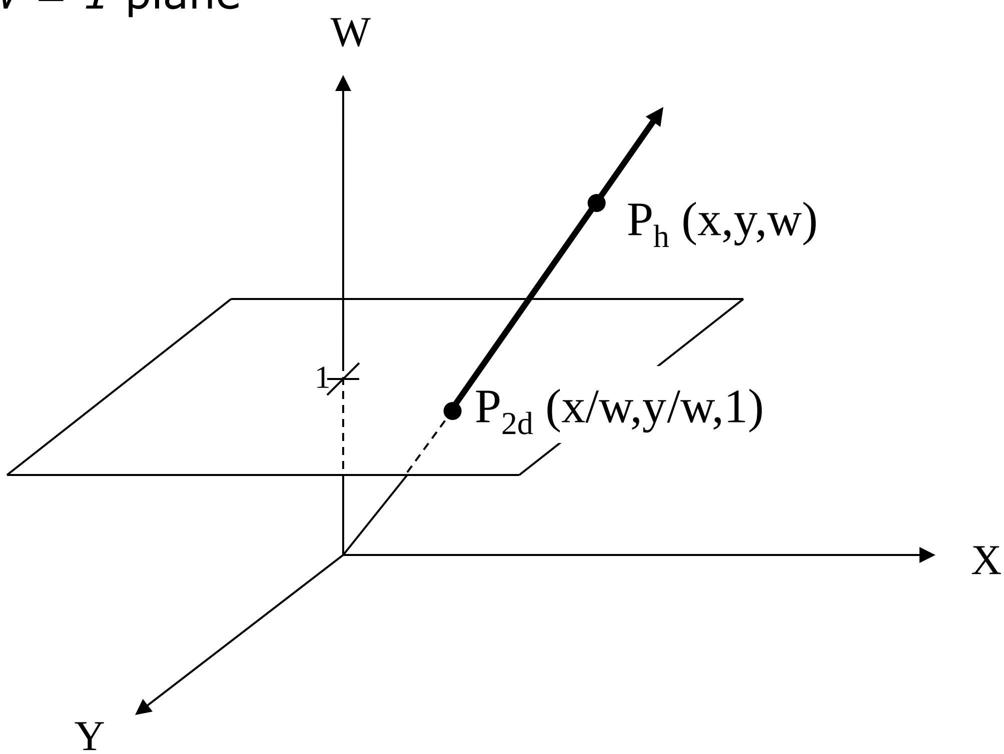
$$P_{2d}(x, y) = P_{2d}\left(\frac{x'}{w}, \frac{y'}{w}\right)$$

- w is 1 for affine transformations in graphics

What is $\begin{bmatrix} x \\ y \\ w \end{bmatrix}$?

- P_{2d} is intersection of line determined by P_h with the

  w = 1 plane



- Infinite number of points correspond to $(x, y, 1)$: they constitute the whole line (tx, ty, tw)

2D Homogeneous Coordinate Transformations (1/2)

- For points written in homogeneous coordinates,

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

translation, scaling and rotation relative to the origin are expressed homogeneously as:

$$T(dx, dy) = \begin{bmatrix} 1 & 0 & dx \\ 0 & 1 & dy \\ 0 & 0 & 1 \end{bmatrix} \qquad v' = T(dx, dy)v$$

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad v' = S(s_x, s_y)v$$

$$R(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad v' = R(\phi)v$$

2D Homogeneous Coordinate Transformations (2/2)

- Consider the rotation matrix:

$$R(\phi) = \left[\begin{array}{cc|c} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ \hline 0 & 0 & 1 \end{array}\right]$$

- The 2 x 2 submatrix columns are:
  - unit vectors (length=1)
  - perpendicular (dot product=0)
  - vectors into which X-axis and Y-axis rotate
- The 2 x 2 submatrix rows are:
  - unit vectors
  - perpendicular
  - vectors that rotate into X-axis and Y-axis
- Preserves lengths and angles of original geometry. Therefore, matrix is a "rigid body" transformation.

Examples

- Translate [1,3] by [7,9]

$$\begin{bmatrix} 1 & 0 & 7 \\ 0 & 1 & 9 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 8 \\ 12 \\ 1 \end{bmatrix}$$

- Scale [2,3] by 5 in the X direction and 10 in the Y direction

$$\begin{bmatrix} 5 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 30 \\ 1 \end{bmatrix}$$

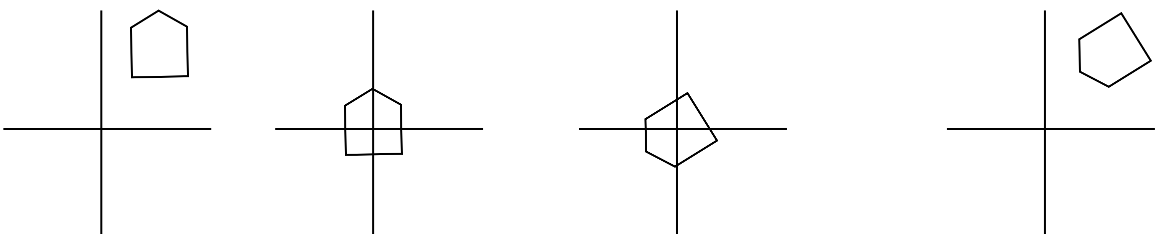
- Rotate [2,2] by 90° ($\pi/2$)

$$\begin{bmatrix} \cos(\pi/2) & -\sin(\pi/2) & 0 \\ \sin(\pi/2) & \cos(\pi/2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 2 \\ 1 \end{bmatrix}$$

Matrix Compositions: Using Translation

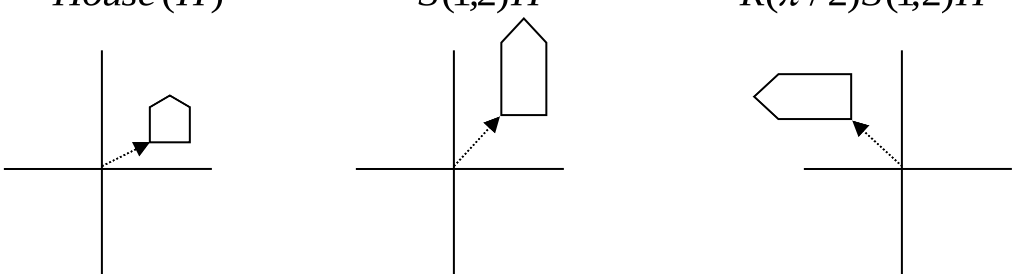
- Avoiding unwanted translation when scaling or rotating an object not centered at origin:
  - translate object to origin, perform scale or rotate, translate back.

$House(H)$ $\quad$ $T(dx,dy)H$ $\quad$ $R(\theta)T(dx,dy)H$ $\quad$ $T(-dx,-dy)R(\theta)T(dx,dy)H$



- How would you scale the house by 2 in “its” y and rotate it through 90° ?

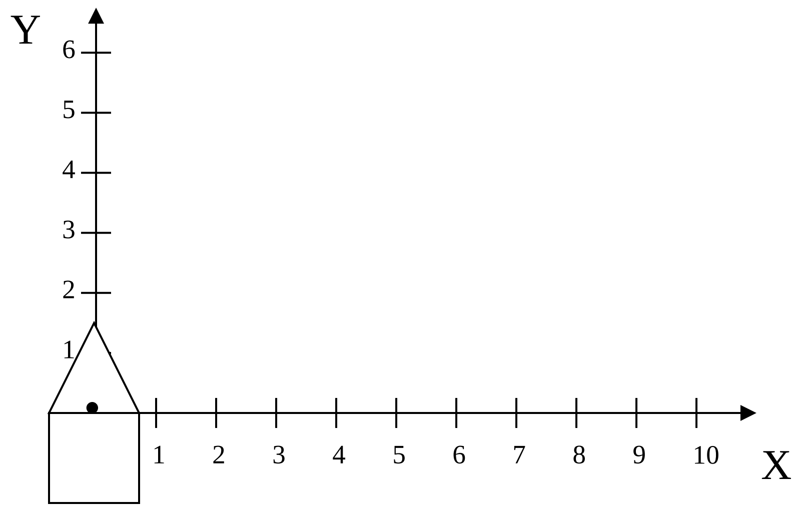
$House(H)$ $\quad$ $S(1,2)H$ $\quad$ $R(\pi/2)S(1,2)H$



- Remember: matrix multiplication is <u>not</u> commutative! Hence order matters! (refer to the Transformation Game at *Demos->Scenegraphs*)

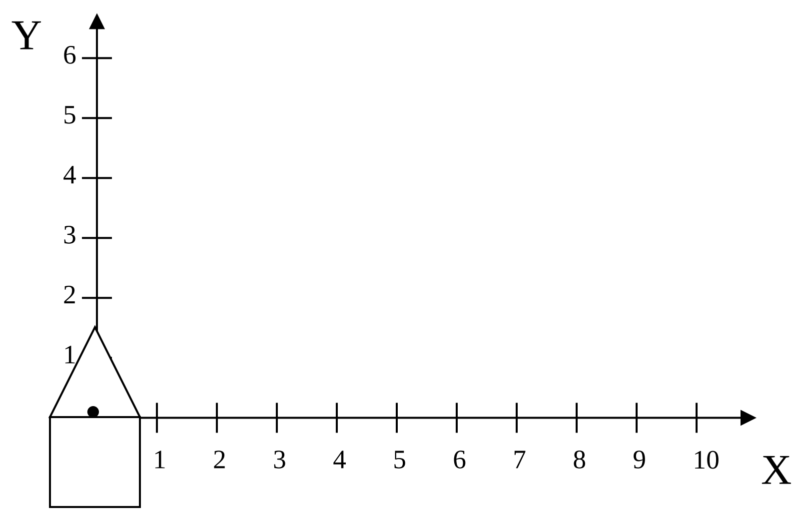
Transformations are NOT Commutative

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



Translation → Rotation

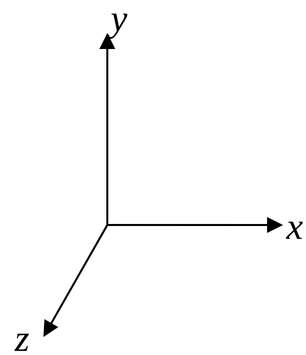
Rotate by 45º then translate by x=6, y=0



Rotation → Translation

3D Basic Transformations (1/2)

(right-handed coordinate system)



- Translation $\begin{bmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{bmatrix}$

- Scaling $\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

3D Basic Transformations (2/2)

(right-handed coordinate system)

- Rotation about X-axis

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation about Y-axis

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Rotation about Z-axis

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Homogeneous Coordinates

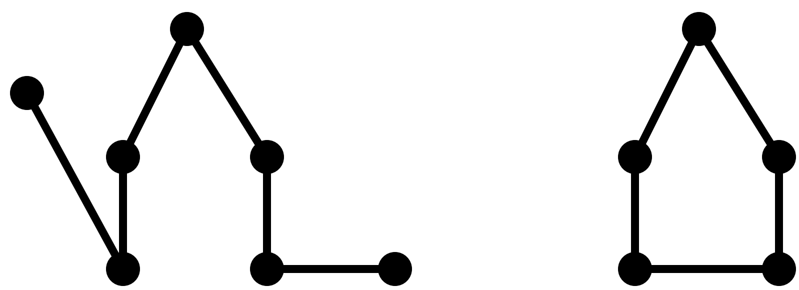
Some uses we'll be seeing later

- Placing sub-objects in parent's coordinate system to construct hierarchical scene graph
  - transforming primitives in own coordinate system
- View volume normalization
  - mapping arbitrary view volume into canonical view volume along z-axis
- Parallel (orthographic, oblique) and perspective projection
- Perspective transformation

2D Object Definition (1/3)

Lines and Polylines

- *Polylines*: lines drawn between ordered points



- Same first and last point make *closed polyline* or *polygon*
- If it does not intersect itself, called *simple polygon*

Polygons

- Ένα πολύγονο (polygon or face = επιφάνεια) ορίζετε σαν μια σειρά από n σημεία (κορυφές, vertices)

$$\left[p_0, p_1, p_2, \ldots, p_{n-1}\right]$$

$$p_i = \left(x_i, y_i, z_i\right)$$

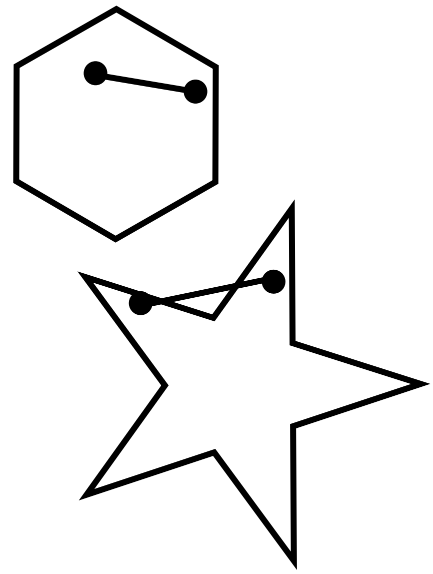
- Τα σημεία πρέπει να βρίσκονται στο ίδιο επίπεδο
- 3 σημεία ορίζουν επίπεδο, αλλά κάποιο τέταρτο δεν είναι κατ’ ανάγκη στο ίδιο επίπεδο

2D Object Definition (3/3)

Convex vs. Concave Polygons

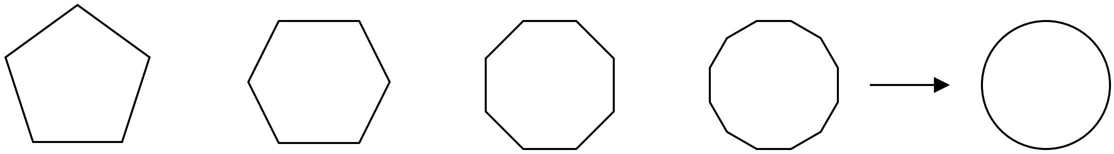
Convex: For every pair of points in the polygon, the line between them is fully contained in the polygon.

Concave: Not convex: some two points in the polygon are joined by a line not fully contained in the polygon.

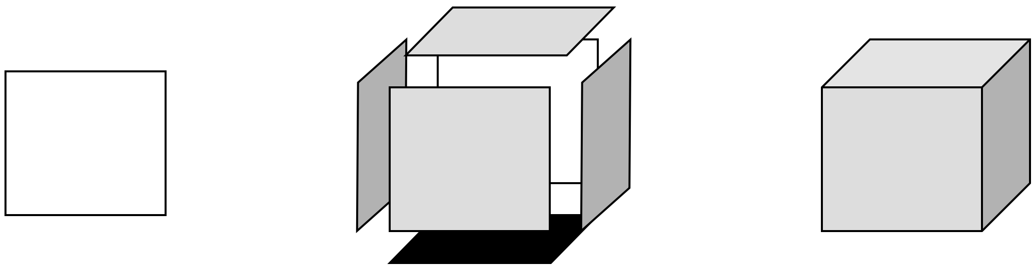


Circle as polygon

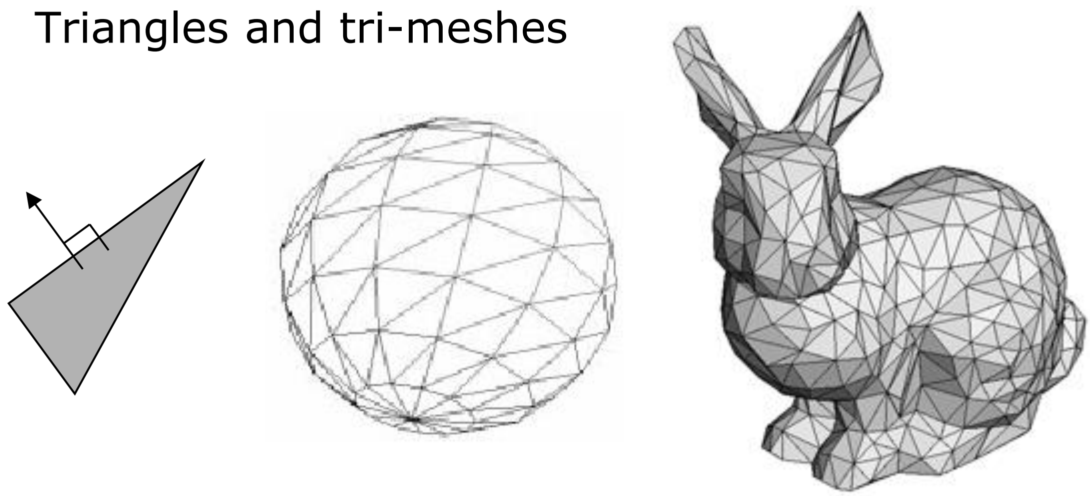
- A circle can be approximated by a polygon with many sides (>15)



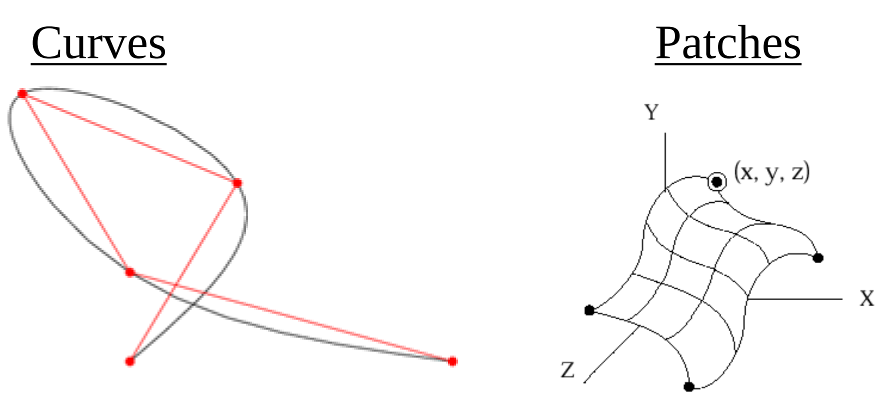
Building 3D Primitives



- Triangles and tri-meshes

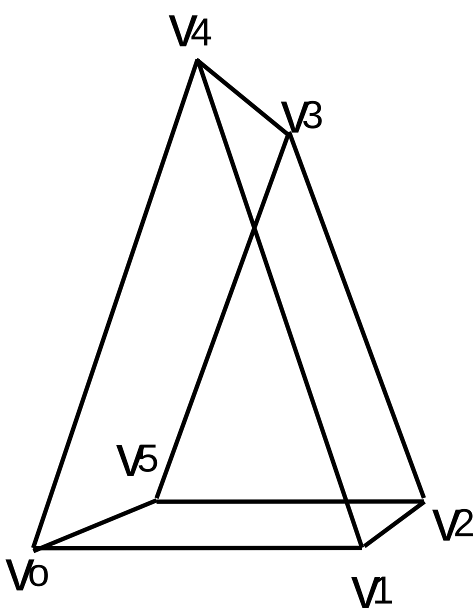


- Often parametric polynomials, called splines



Boundaries are Polynomial curves In 3D

Παράδειγμα Πολύεδρου



- $F0=v_0v_1v_4$
- $F1=v_5v_3v_2$
- $F2=v_1v_2v_3v_4$
- $F3=v_0v_4v_3v_5$
- $F4=v_0v_5v_2v_1$

- V=6,F=5, E=9
- V-E+F=2

Αναπαράσταση πολυέδρων (1)

- Εξαντλητικά (πίνακας από σειρές κορυφών)
  - faces[0] = (x_0,y_0,z_0), (x_1,y_1,z_1), (x_4,y_4,z_4)
  - faces[1] = (x_5,y_5,z_5), (x_3,y_3,z_3), (x_2,y_2,z_2)
  - κτλ ....

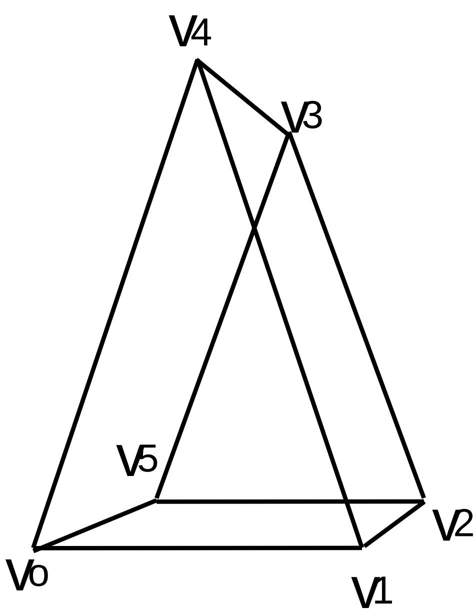
- Πολύ σπάταλο αφού η κάθε κορυφή παρουσιάζεται (τουλάχιστον) 3 φορές στην λίστα
  - Παρόλα αυτά χρησιμοποιείται αρκετά!

Αναπαράσταση πολυέδρων (2)

Indexed Face set

- Πίνακας κορυφών (Vertex array)
  - vertices[0] = (x_0, y_0, z_0)
  - vertices[1] = (x_1, y_1, z_1)
  - κτλ …
- Πίνακας πολυγώνων (Face array) – λίστα από δείκτες στον πίνακα κορυφών
  - faces[0] = 0, 1, 4
  - faces[1] = 5, 3, 2
  - κτλ ...

Η σειρά των κορυφών είναι σημαντική



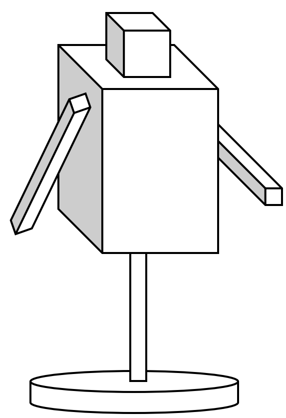
- Το πολύγονο $\{v_0,v_1,v_4\}$ δεν είναι το ίδιο με το $\{v_0,v_4,v_1\}$
- Η κάθετος των δύο δείχνει προς την αντίθετη κατεύθυνση
- Συνήθως το κάθε πολύγωνο είναι ορατό μόνο από το θετικό του half-space
- Αυτό είναι γνωστό ως απόκρυψη πίσω επιφανειών (back-face culling)

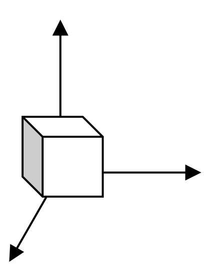
Transformations in Scene Graphs (modeling transformations)

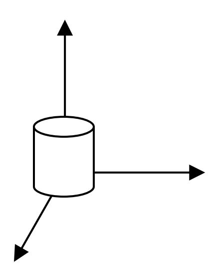
- 3D scenes are often stored in a directed acyclic graph (DAG) called a *scene graph*
  - Open Scene Graph (used in the Cave)
  - Sun’s Java3D™
  - X3D ™ (VRML ™ was a precursor to X3D)

- Typical scene graph format:
  - **objects** (cubes, sphere, cone, polyhedra etc.)
    - stored as nodes (default: unit size at origin)
  - **attributes** (color, texture map, etc.) and **transformations** are also nodes in scene graph (labeled edges on slide 2 are an abstraction)

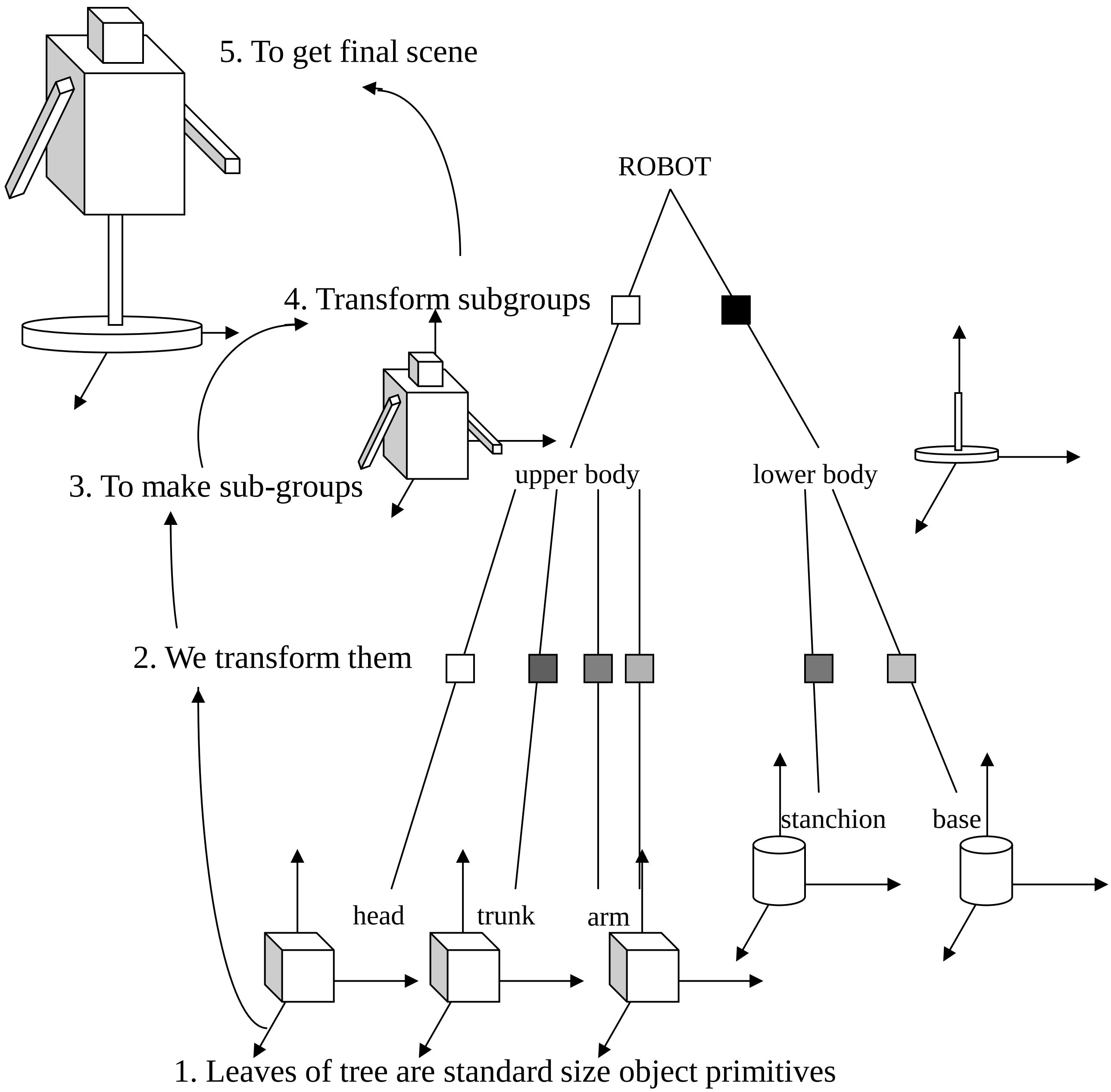
Transformations in Scene Graphs







Transformations in Scene Graphs



Transformations in Scene Graphs

Transformations in Scene Graphs

- Transformations affect all child nodes
- Complex geometry can be reused

  instances of a mesh can have different transformations applied to them (e.g. book is used twice– once under t4 and once under t5)
- Local Transformation Matrix

  Position with respect to parent

Composing Transformations in a Scene Graph

- To determine final composite transformation matrix (CTM) for object node:
  - compose all parent transformations during prefix graph traversal
  - exact detail of how this is done varies from package to package, so be careful