IMGD 1001:
3D Art

by
Mark Claypool (claypool@cs.wpi.edu)
Robert W. Lindeman (gogo@wpi.edu)

Outline

- The Pipeline
- Concept Art
- 2D Art
  - Animation, Tiles
- 3D Art (next)
  - Modeling, Texturing, Lighting, Transformations

Polygonal Modeling Basics: Primitives

- *Primitives* are basic shapes
- Most 3d packages have same primitives:
  - Sphere, Cube, Cylinder, Plane
  - Use for “broad strokes”
- Concentrate on primitives within object
  - Ex: human body (ovals for shoulders, cylinders for legs, sphere for head…)
- *Components* are parts that make up primitive
  - Ex: vertices, edges, triangles, faces, elements
  - Similar across all packages but terminology can vary
- *Transformation* allows moving, rotating, scaling object or component

Polygonal Modeling Basics: Normals

- Face normals are at right angle to polygon
  - Tell what direction if facing, how to render, how light will react
- Viewed from other side, is invisible
  - Fine if on inside (say, of solid cube)
- When debugging, pay attention to normals as well as polygons

Polygonal Modeling Basics: Backface Culling

- Toggles display of faces that point away from view
  - When on, see through wireframe
  - When off, looks solid (not drawn)
- Makes look less cluttered

Claypool & Lindeman - WPI, CS & IMGD 5

Modeling Tools

- Certain tools and techniques used 80-90% of the time
- *Line Tool*:
  - Draw outline of object and extrude to get 3-d shape
    - Ex: profile of car. Use line tool. Then, extrude outward to get shape.
- *Extrude*:
  - Take component (often face), duplicating it, pulling pushing or scaling to refine model
    - Ex: take cube. Extrude face outward and smaller
- *Cut*:
  - Subdivides faces and adds new faces
- *Adjust*:
  - The artistic part of modeling. Try to capture form, profile and character by moving vertices
    - "Vertex surgery", part of the technical manipulation

Claypool & Lindeman - WPI, CS & IMGD 6

Based on Chapter 6.2, *Introduction to Game Development*

1 Modeling Technique: Box Modeling

- Done for character, but can apply to other things
- General idea:
  - Start with box, cylinder or other primitive
  - Extrude, Cut, Adjust…
  - Get topology, proportions right
  - Once happy, refine until details complete

Box modeling: Quick Example

- Reference
- Box modeling: extrude, cut, adjust
- Compare to reference
- Shade

Based on Chapter 6.2, *Introduction to Game Development*

Polygons and Limits

- 3d Software renders scene of polygons like game
  - But 3d software slow (*Toy Story* 1 frame / 15 hrs)
  - Game is real time (30 frames / second)
- Need to limit polygons. How spent depends upon world size and where needed.
  - Ex: *Medal of Honor* versus *Soul Caliber 2*. MoH details spread across world, less on avatars. SC2 can have detailed avatars since only 2 in one ring.
- Think of *how many* polygons each item needs. Estimates, educated guesses. Then, make pass. (Tools will often give count)
  - Used wisely, can make detailed scenes with few (Ex: 2.5, page 24)
  - Ch 6.2 assumes 4000 (typical for PS2 street fighting game or hero in 3rd person action game)

Polygon Reduction (1 of 4)

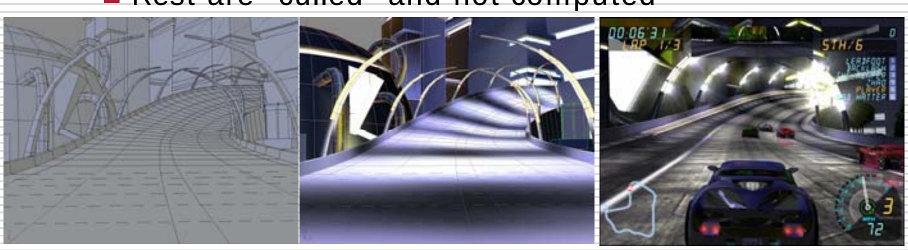
- Being able to model without wasting polygons important → takes practice
- Ask if a player will see face?
  - Ex: oil barrel as cylinder. Will see bottom? Nope, then delete.
- Are all faces necessary? Looks great, yeah, but some can be removed.
  - Ex: 12-sided cylinder still looks “round” with 8 sides? Then do it.
- (Example exercise p30-31)

Polygon Reduction (2 of 4)

- Level-of-detail (LOD) meshes
  - Multiple versions of object, progressively lower levels
- When far away, use low level
  - Assume more objects in Field of View
- When close, use higher level
  - Assume fewer objects in Field of View

Polygon Reduction (3 of 4)

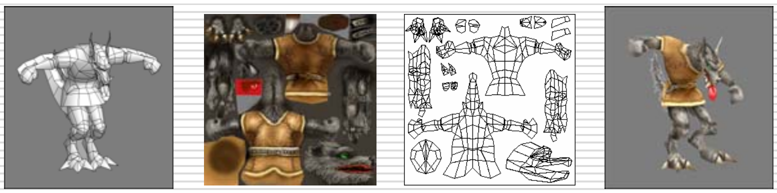
- For entire level (ie- map with environment), entire polygon count matters
  - Impacts amount of memory needed
- But only visible polygons rendered
  - Rest are "culled" and not computed



Images courtesy of WildTangent

Polygon Reduction (4 of 4)

- With low polygon modeling, much of the detail is painted into the texture (next topic!)



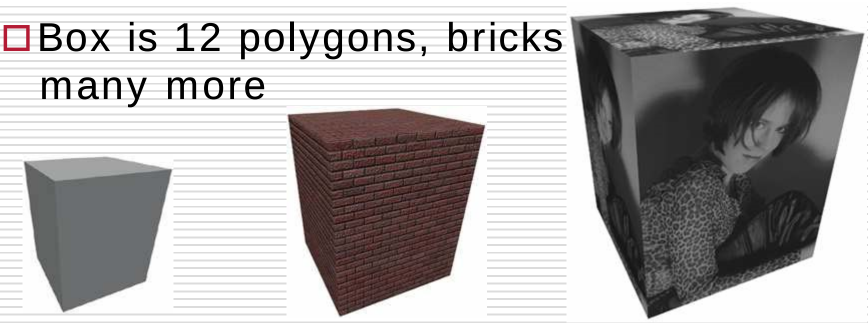
Images courtesy of WildTangent, model and texture by David Johnson.

Texture

- *Shader* – define surface property of object – how shiny, bumpy, how light effects
- *Texture* – bitmap plugged into shader that defines image we want to appear on object

Detail in Texture

- Add depth, lines, etc. without polygons
- Box is 12 polygons, bricks many more

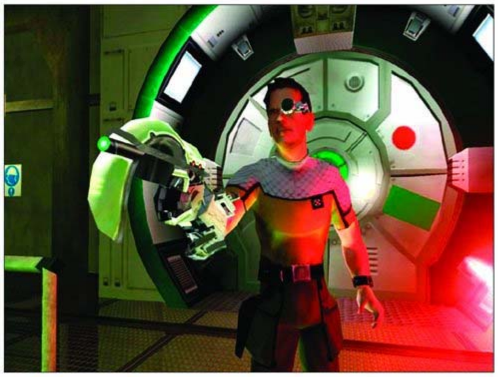


(Taken from http://www.mostert.org/3d/3dpdzscenem.html)

Lighting

- Can conjure feelings, emotions, even change what you are seeing
  - Reveal (or hide) depth
  - (Many books on traditional lighting)
  - AR/ID 3150. LIGHT, VISION AND UNDERSTANDING
- Color, Mood

The Role of Color



RTX Red Rock

Color indicates danger

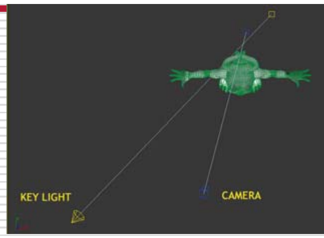
http://www.informit.com/articles/article.asp?p=174370

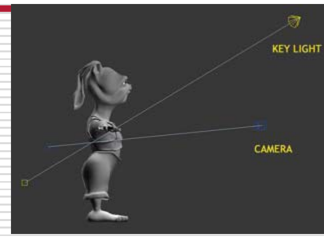
The Role of Lighting

Long shadows not only add to the atmosphere, but also help break up repetition

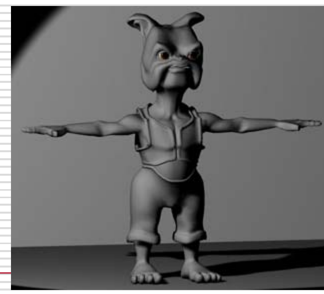
http://www.informit.com/articles/article.asp?p=174370

Lighting Setup (1 of 3)



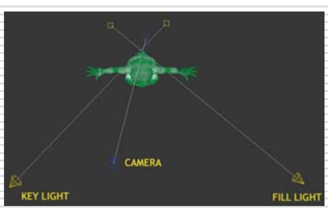


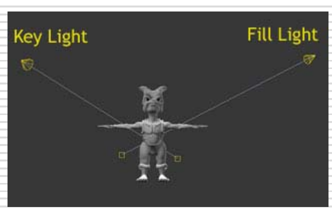
Key light – main source.
The Key light is placed next to the camera, about 35-45 degree angle to the subject. The angle is determined by what kind of mood that you want the scene to have.



http://www.3dtotal.com/team/Tutorials/Jenns3pt_tut/3ptlighting.asp

Lighting Setup (2 of 3)

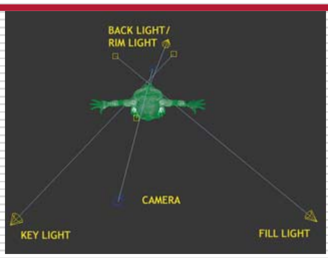


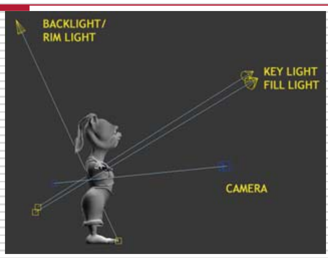


Fill light – Brings out some details out of shadow.
Place the Fill Light at a 90 degree angle from the Key Light, usually slightly higher or lower than the Key Light.

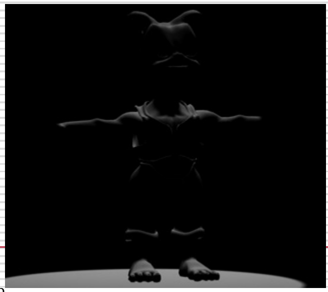
http://www.3dtotal.com/team/Tutorials/Jenns3pt_tut/3ptlighting.asp

Lighting Setup (3 of 3)





Backlight - Highlights edges, pulls away from background. Placed directly opposite the camera and behind the subject.



http://www.3dtotal.com/team/Tutorials/Jenns3pt_tut/3ptlighting.asp

Working with 3D Lights

- *Directional Lights* – used for sunlight or moonlight. Often as key light. Predictable.
- *Ambient Lights* – spread everywhere, equally. Uniform diffuse lights. Precise control over illumination.
- *Spot Lights* – focus beam on single location. Great control.
- *Point Lights* – single point in all directions. Light bulbs, candles, etc.

Based on Chapter 6.6, *Introduction to Game Development*

Example of Working with 3D Lights

Ambient light

Effective Lighting Practices (1 of 2)

Pools of light
– Don't always try to light evenly.
- Gives sense of mystery

Pools of light in Indiana Jones: The Emperor's Tomb

Effective Lighting Practices (2 of 2)

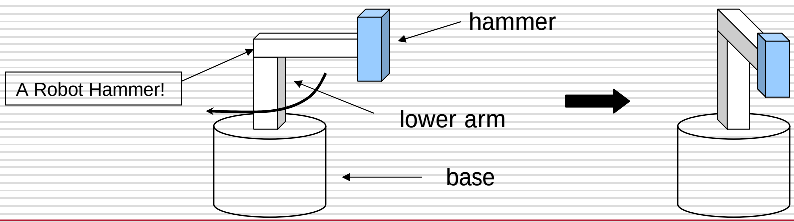
Guide lights –
- Use light to guide the player.
- Helps highlight areas that are accessible and important to the objectives.

Introduction to Transformations

- A *transformation* changes an object's
  - Size (scaling)
  - Position (translation)
  - Orientation (rotation)
- Transform object by applying sequence of matrix multiplications to object vertices

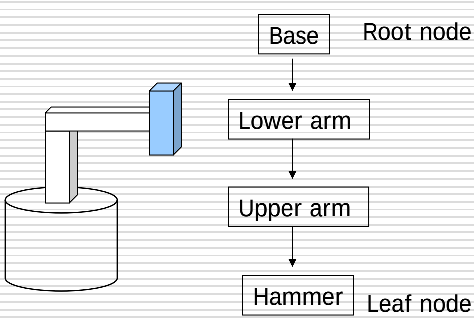
Hierarchical Transformations

- Graphical scenes have object dependencies
- Many small objects
- Attributes (position, orientation, *etc.*) depend on each other



Hierarchical Transformations

- Object dependency description using tree structure

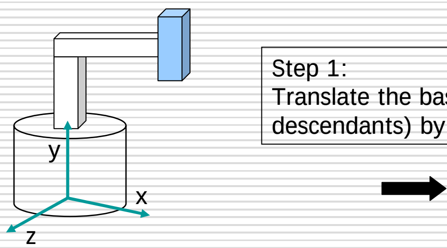


Object position and orientation can be affected by its parent, grand-parent, grand-grand-parent, ... nodes

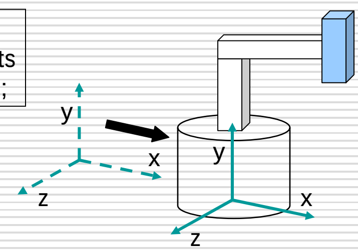
Hierarchical representation is known as **Scene Graph**

Hierarchical Transformations

- Relative Transformations - Specify the transformation for each object relative to its parent

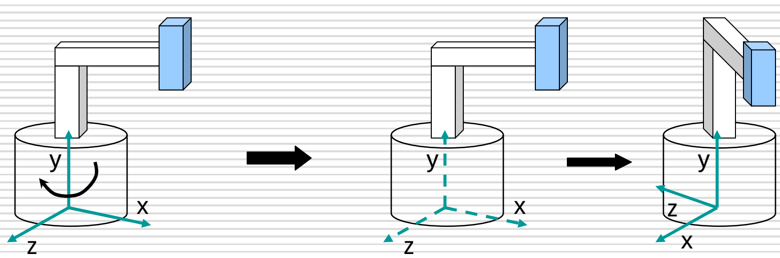


Step 1:
Translate the base (and its descendants) by (5, 0, 0);

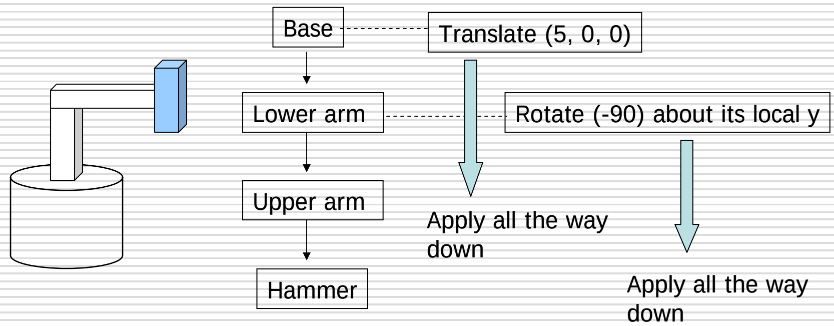


Hierarchical Transformations

Step 2:
Rotate the lower arm and (its descendants) relative to the base's local y axis by -90 degrees



Hierarchical Transformations



Transformation uses Matrices

- All transformations can be performed using matrix/vector multiplication
- Allows pre-multiplication of all matrices
- Note: point (x, y) needs to be represented as (x, y, 1), also called ***homogeneous coordinates***

Point Representation

- We use a column matrix (2x1 matrix) to represent a 2D point

$$\begin{pmatrix} x \\ y \end{pmatrix}$$

- General form of transformation of a point *(x, y)* to *(x', y')* can be written as:

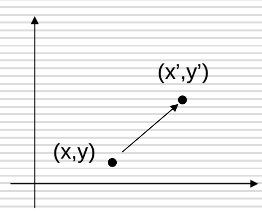
$$x' = ax + by + c \qquad \qquad \begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$
$$y' = dx + ey + f$$

or

Translation

- To reposition a point along a straight line
- Given point *(x, y)* and translation distance (t_x, t_y)
- The new point: *(x', y')*

$x' = x + t_x$
$y' = y + t_y$



or

$P' = P + T$ where $P' = \begin{pmatrix} x' \\ y' \end{pmatrix}$ $P = \begin{pmatrix} x \\ y \end{pmatrix}$ $T = \begin{pmatrix} t_x \\ t_y \end{pmatrix}$

3x3 2D Translation Matrix

$x' = x + t_x$
$y' = y + t_y$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}$$

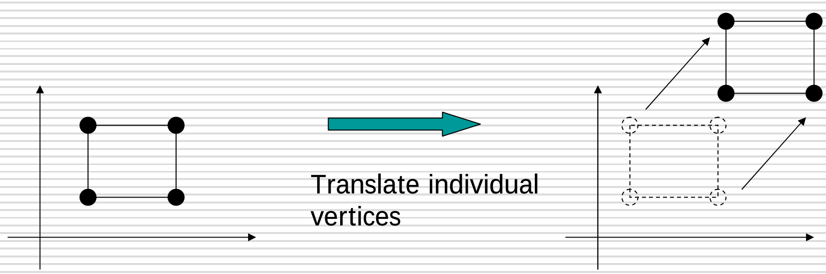
use 3x1 vector

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

Note: it becomes a matrix-vector multiplication

Translation of Objects

- How to translate an object with multiple vertices?

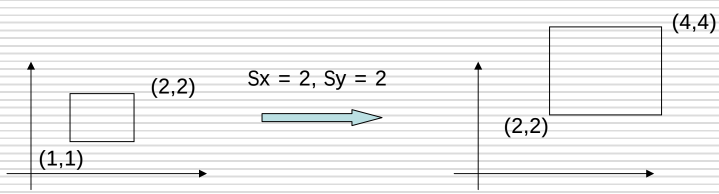


2D Scaling

- Scale: Alter object size by scaling factor (Sx, Sy). *i.e.*,

*x' = x * Sx*
*y' = y * Sy*

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



3x3 2D Scaling Matrix

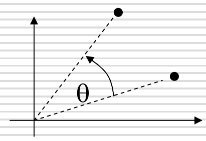
*x' = x * Sx*
*y' = y * Sy*

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} Sx & 0 \\ 0 & Sy \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

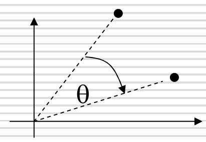
$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

2D Rotation

- Default rotation center is origin *(0,0)*



$\theta > 0$: Rotate counter clockwise

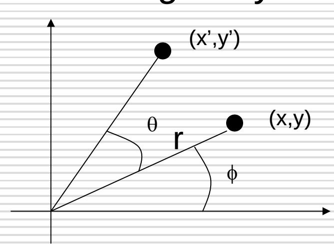


$\theta < 0$: Rotate clockwise

2D Rotation (cont.)

(x,y) → Rotate *about the origin* by θ

→ (x', y')



How to compute (x', y') ?

$x = r*\cos(\phi)$ $\quad$ $x' = r*\cos(\phi + \theta)$

$y = r*\sin(\phi)$ $\quad$ $y' = r*\sin(\phi + \theta)$

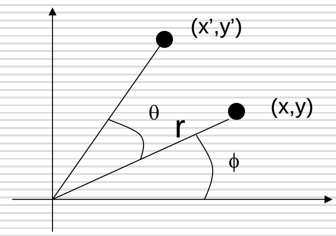
2D Rotation (cont.)

- Using trigonometric identities

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

$$\sin(\theta+\phi)=\sin\theta\cos\phi+\cos\theta\sin\phi$$

$x' = x\cos(\theta) - y\sin(\theta)$
$y' = x\sin(\theta) + y\cos(\theta)$

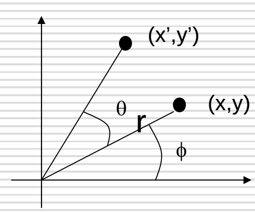


Matrix form?

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

3x3 2D Rotation Matrix

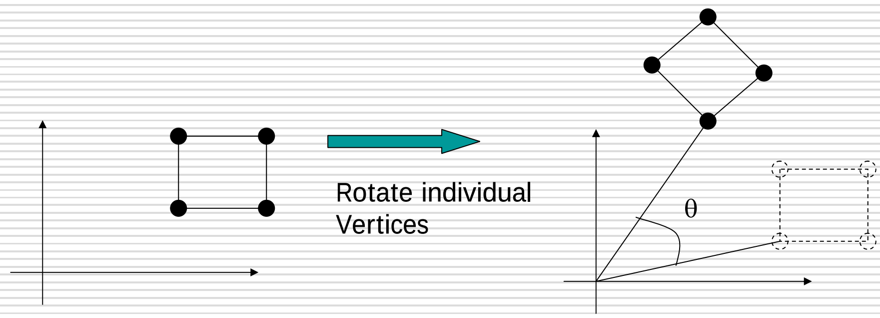
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$



$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

2D Rotation

- How to rotate an object with multiple vertices?



Arbitrary Rotation Center

- To rotate about arbitrary point P = (Px, Py) by θ:
  - Translate object by T(-Px, -Py) so that P coincides with origin
  - Rotate the object by R(θ)
  - Translate object back: T(Px, Py)
- In matrix form
  - T(Px,Py) R(θ) T(-Px,-Py) * P

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & Px \\ 0 & 1 & Py \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & -Px \\ 0 & 1 & -Py \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

- Similar for arbitrary scaling anchor

Composing Transformations

- Composing transformations
  - Applying several transforms in succession to form one overall transformation
- Example
  - **M1 X M2 X M3 X P**

  where M1, M2, M3 are transform matrices applied to P
- Be careful with the order!
- For example
  - Translate by (5, 0), then rotate 60 degrees is NOT same as
  - Rotate by 60 degrees, then translate by (5, 0)