Computer Graphics Programming: Matrices and Transformations

Outline

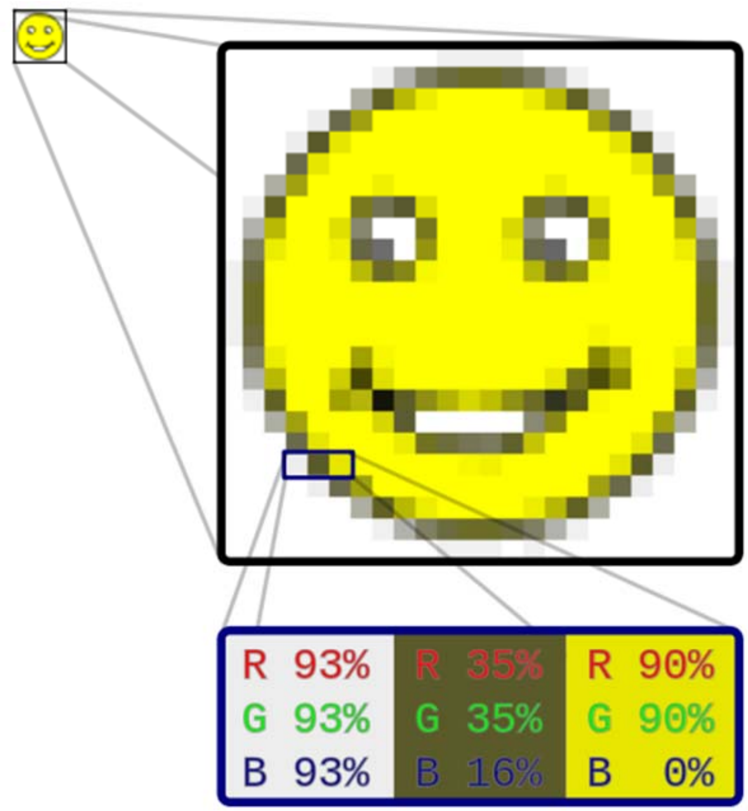
- Computer graphics overview
- Object/Geometry modeling
- 2D modeling transformations and matrices
- 3D modeling transformations and matrices
- Relevant Unity scripting features

Computer Graphics

- Algorithmically generating a 2D image from 3D data (models, textures, lighting)
- Also called rendering

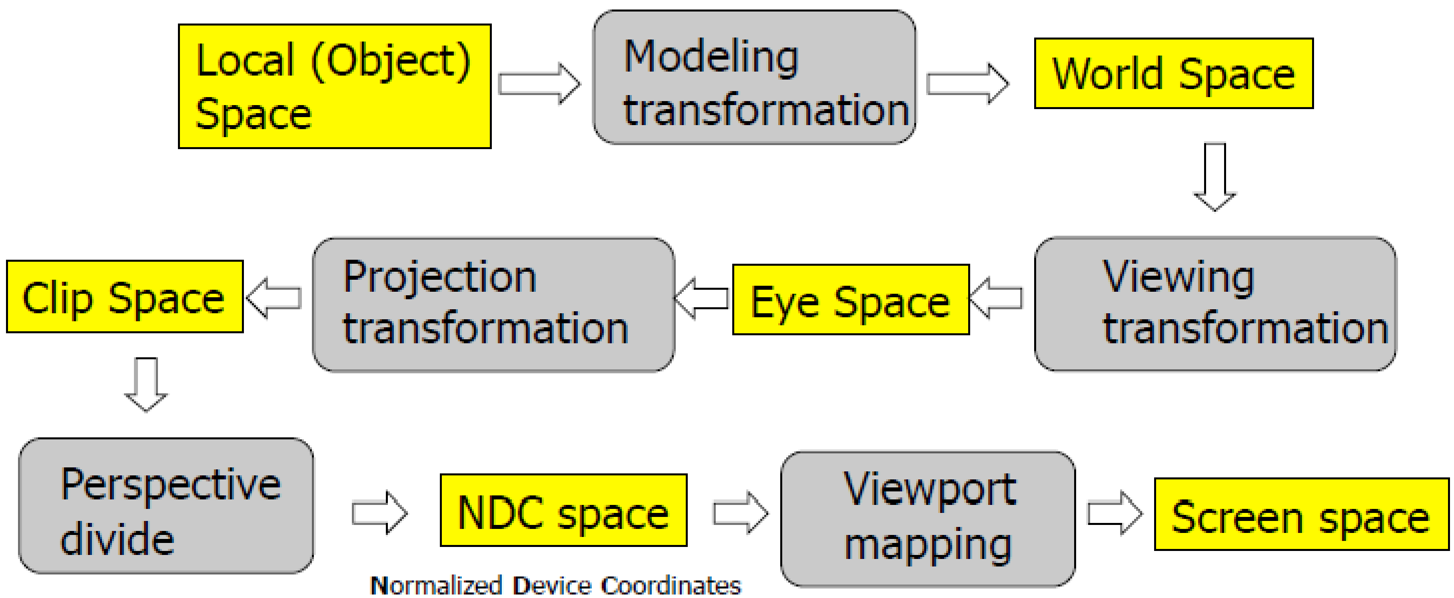
- Raster graphics
  - Array of pixels
  - About 25x25 in the example ->

- Algorithm tradeoffs:
  - Computation time
  - Memory cost
  - Image quality



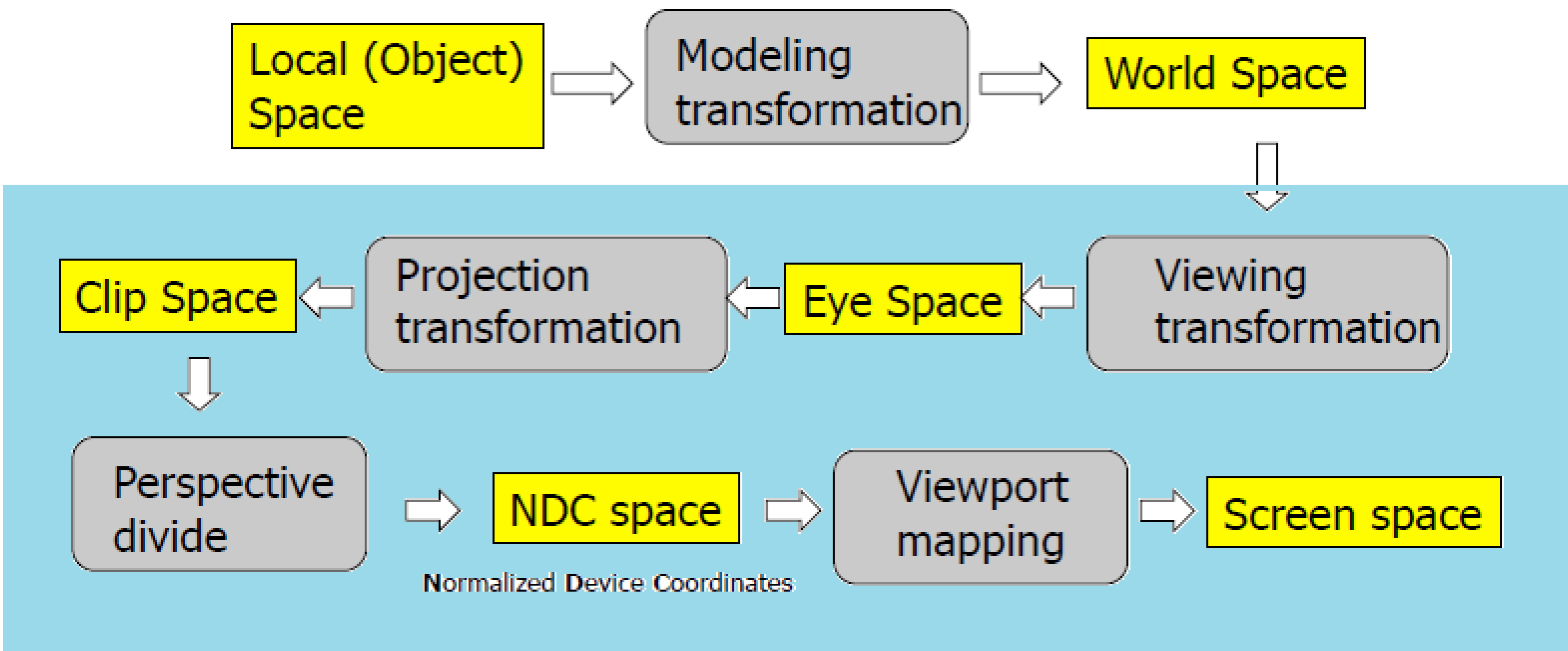
Computer Graphics

- The graphics pipeline is a series of conversions of points into different *coordinate systems* or *spaces*

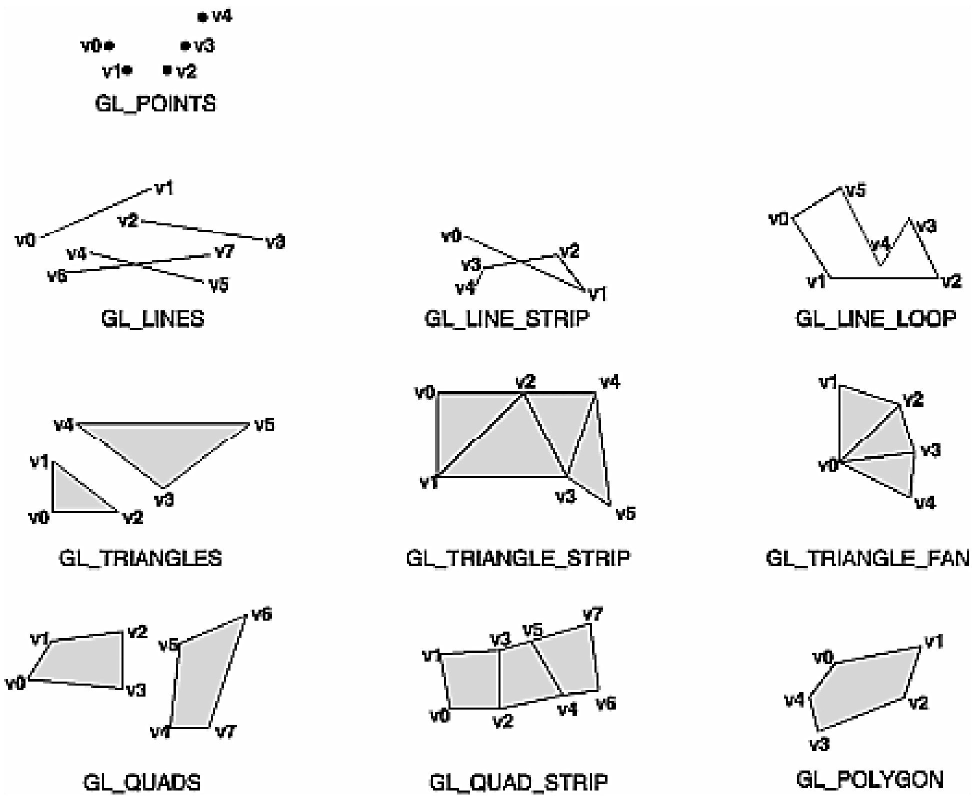


Computer Graphics

- Virtual cameras in Unity will handle everything from the viewing transformation on



OpenGL specifying geometry



Legacy syntax example:

```
glBegin(GL_POLYGON);
   glVertex2f(-0.5, -0.5);
   glVertex2f(-0.5, 0.5);
   glVertex2f(0.5, 0.5);
   glVertex2f(0.5, -0.5);
glEnd();
```

Unity specifying geometry – Mesh class

- Requires two types of values
  - Vertices (specified as an array of 3D points)
  - Triangles (specified as an array of Vector3s whose values are indices in the vertex array)
- Documentation and Example
  - http://docs.unity3d.com/Documentation/Manual/GeneratingMeshGeometryProcedurally.html
  - http://docs.unity3d.com/Documentation/ScriptReference/Mesh.html
- The code on the following slides is attached to a cube game object (rather than an EmptyObject)

Mesh pt. 1 – assign vertices

```
Mesh mesh = new Mesh();
gameObject.GetComponent<MeshFilter>().mesh = mesh;

Vector3[] vertices = new Vector3[4];
vertices[0] = new Vector3(0.0f, 0.0f, 0.0f);
vertices[1] = new Vector3(width, 0.0f, 0.0f);
vertices[2] = new Vector3(0.0f, height, 0.0f);
vertices[3] = new Vector3(width, height, 0.0f);
mesh.vertices= vertices;
```

Mesh pt. 2 – assign triangles

```
int[] tri = new int[6];
// Lower left triangle of a quad
tri[0] = 0;
tri[1] = 2;
tri[2] = 1;
// Upper right triangleof a quad
tri[3] = 2;
tri[4] = 3;
tri[5] = 1;
mesh.triangles = tri;
```

More mesh values

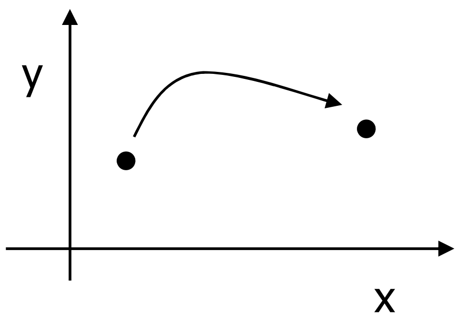
```
// Normal vectors (one per vertex)
Vector3[] normals = new Vector3[4];
// compute normals…
mesh.normals= normals;

// Texture coordinates (one per vertex)
Vector2[] uv = new Vector2[4];
// assign uvs…
mesh.uv= uv;
```

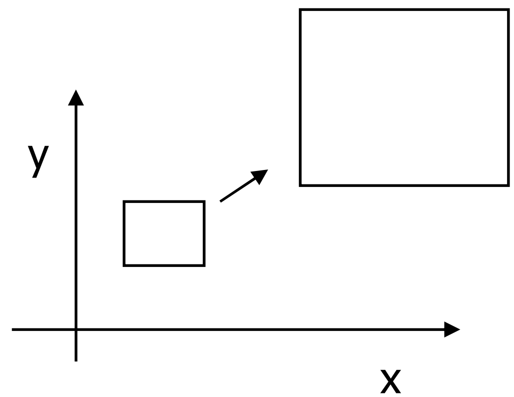
Side note: You can also use mesh.RecalculateNormals(); if you want Unity to try to compute normals for you.

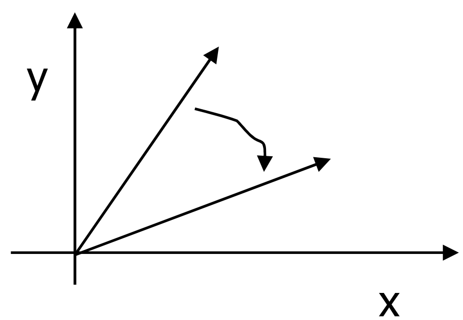
Critical thinking – geometry modeling

- Which of the following statements is true?
  A. Smooth models like spheres are inexpensive to create
  B. A 3D model can be created faster than four hand drawn 2D images of the object from the front, back, and sides
  C. 3D shapes can be constructed out of 2D primitives
  D. All 3D models must be solid volumes

2D Transformations







2D Transformation

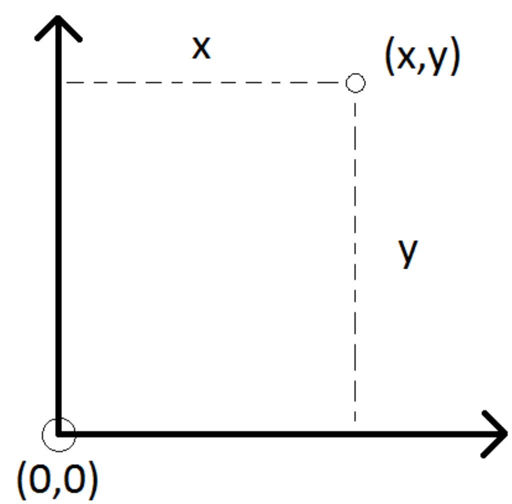
- 2D object
  - Points/Vertices
  - Line segments
  - Vector
- Transformations can change the object’s
  - Position (translation)
  - Size (scaling)
  - Orientation (rotation)
  - Shape (shear)

Point representation

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

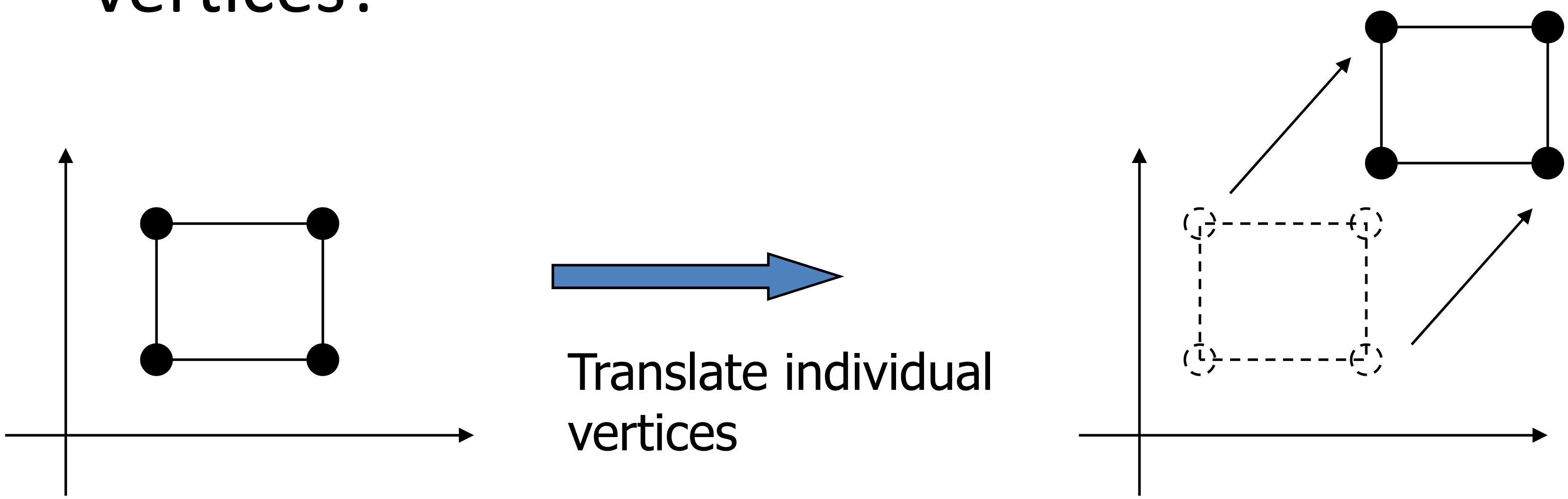
$$p = \begin{bmatrix} x \\ y \end{bmatrix}$$

- Points are defined with respect to
  - origin (point)
  - coordinate axes (basis vectors)



Translation

- How to translate an object with multiple vertices?

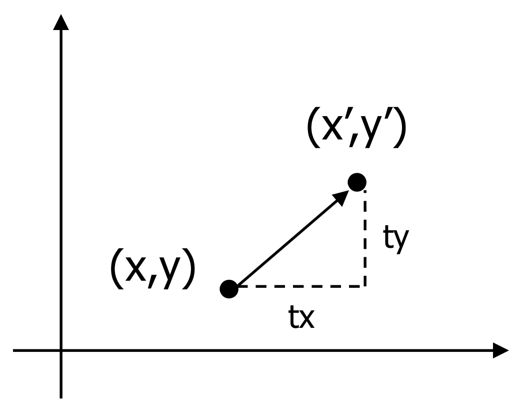


Translation

- Re-position a point along a straight line
- Given a point (x,y), and the translation distance or vector (tx,ty)

The new point: (x′, y′)

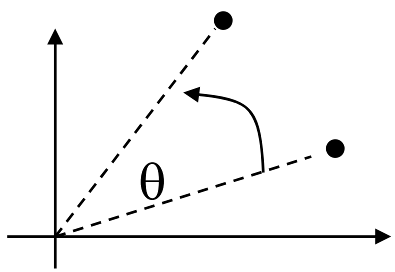
$$x' = x + tx$$
$$y' = y + ty$$



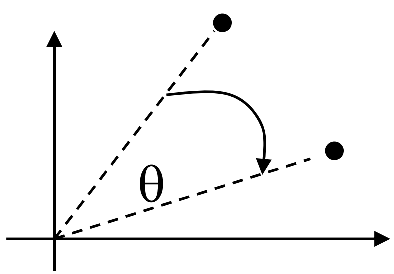
OR p′ = p + t where $p' = \begin{bmatrix} x' \\ y' \end{bmatrix}$ $p = \begin{bmatrix} x \\ y \end{bmatrix}$ $t = \begin{bmatrix} tx \\ ty \end{bmatrix}$

2D Rotation

- Rotate with respect to origin (0,0)



$\theta > 0$: Rotate counter clockwise



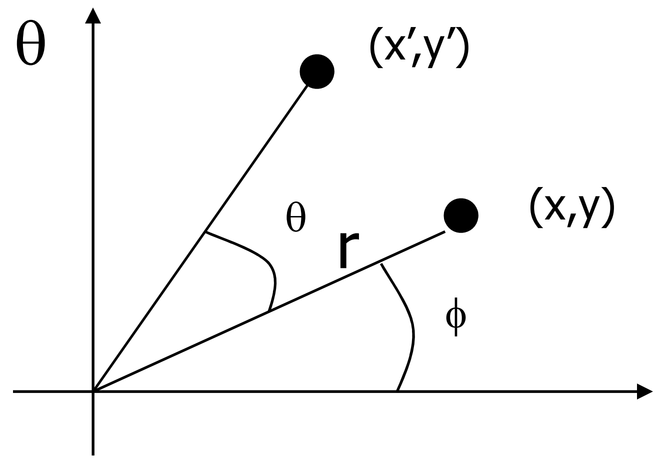
$\theta < 0$: Rotate clockwise

Rotation

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

$\longrightarrow$ (x′, y′)

How to compute (x′, y′) ?



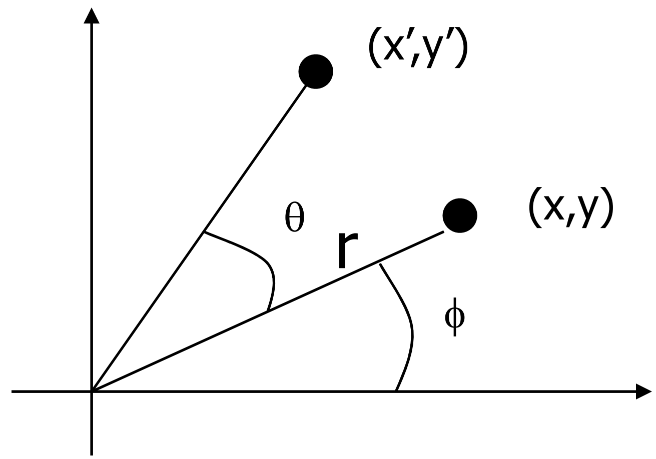
$x = r\cos(\phi)$ $y = r\sin(\phi)$

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

Rotation

$x = r\cos(\phi)$ $y = r\sin(\phi)$

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



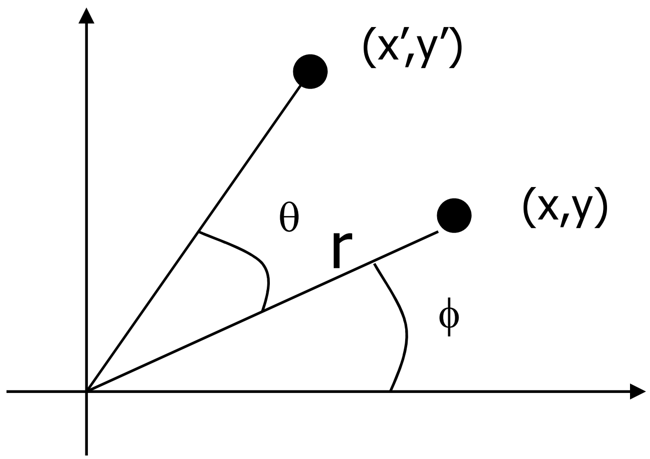
$$x' = r\cos(\phi+\theta)$$
$$= r\cos(\phi)\cos(\theta) - r\sin(\phi)\sin(\theta)$$
$$= x\cos(\theta) - y\sin(\theta)$$

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

Rotation

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

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

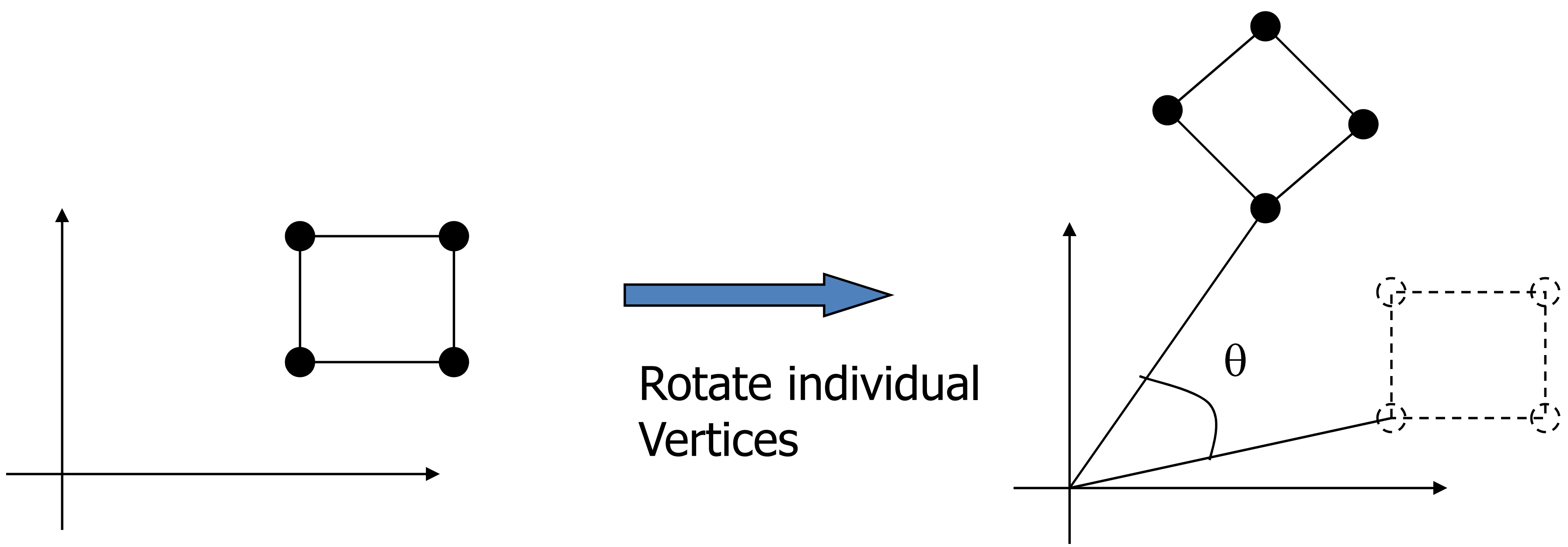


Matrix form:

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

Rotation

- How to rotate an object with multiple vertices?

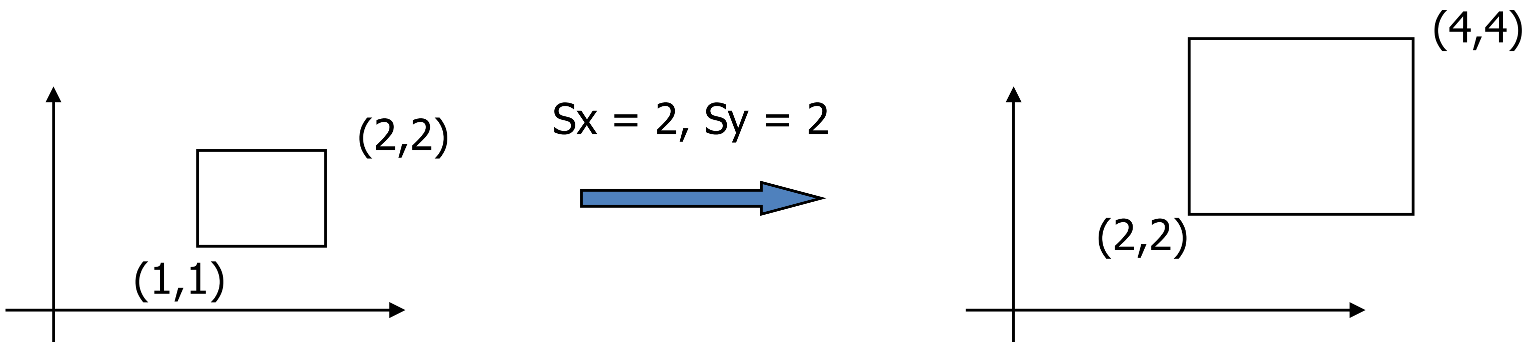


2D Scaling

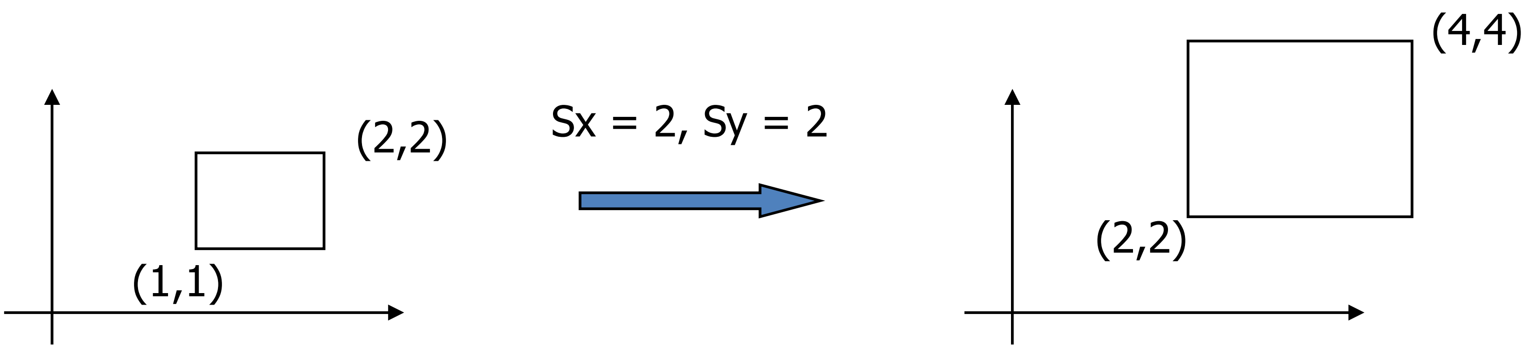
Scale: Alter the size of an object by a scaling factor (Sx, Sy), i.e.

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

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

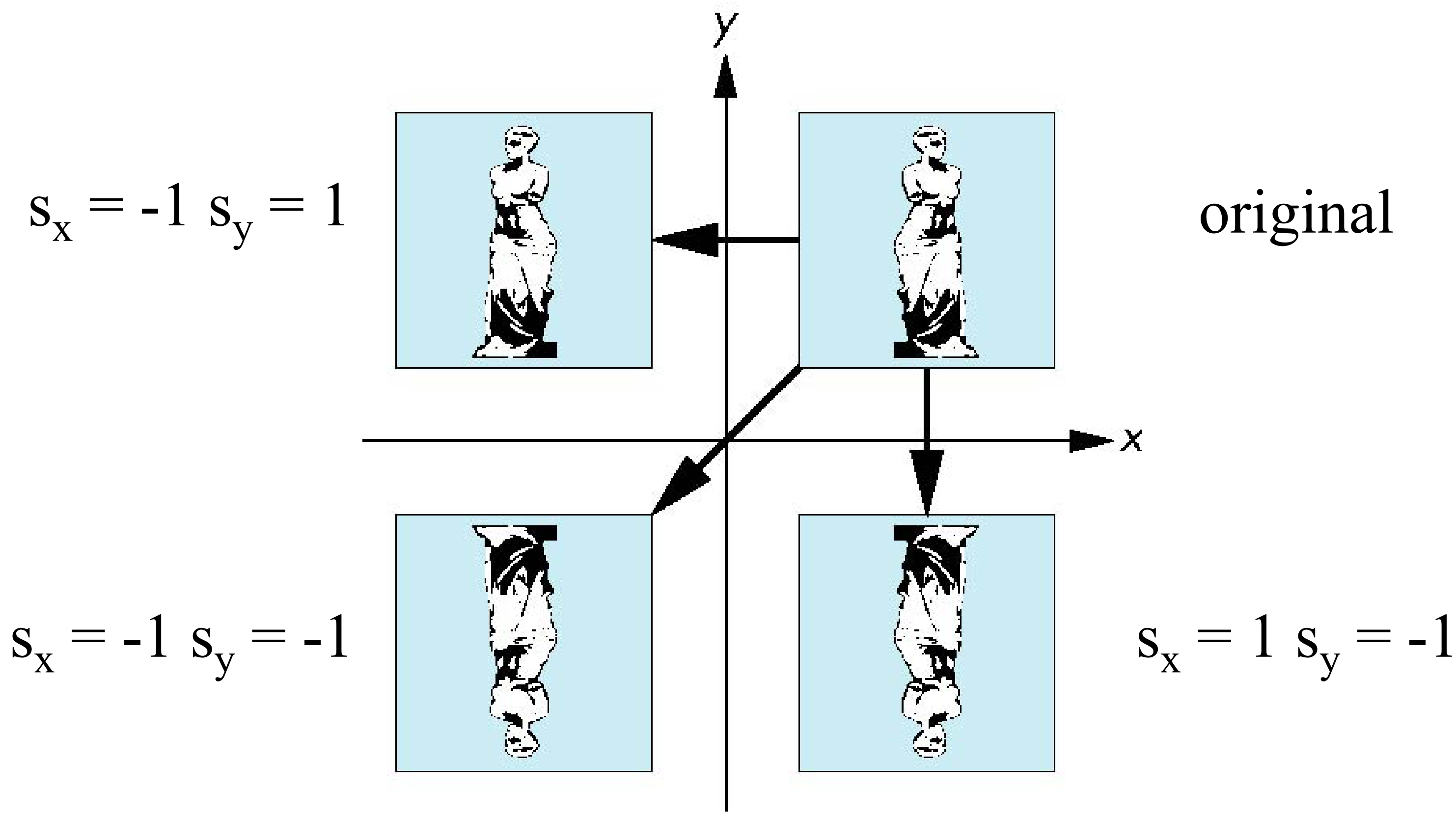


2D Scaling



- Object size has changed, but so has its position!

Scaling special case – Reflection



Put it all together

- Translation: $\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} tx \\ ty \end{vmatrix}$

- Rotation: $\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$

- Scaling: $\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} Sx & 0 \\ 0 & Sy \end{vmatrix} * \begin{vmatrix} x \\ y \end{vmatrix}$

Translation Multiplication Matrix

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} x \\ y \end{vmatrix} + \begin{vmatrix} tx \\ ty \end{vmatrix}$$

Use 3 x 1 vector

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & txx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} y \\ 1 \end{vmatrix}$$

Critical thinking – transformations and matrix multiplication

- Suppose we want to scale an object, then translate it. What should the matrix multiplication look like?

A. p' = Scale * Translate * p

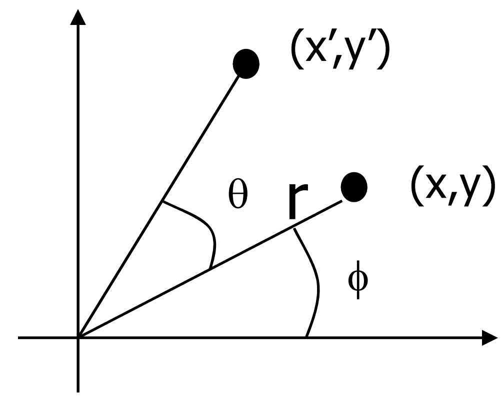
B. p' = Translate * Scale * p

C. p' = p * Scale * Translate

D. Any of these is correct

3x3 2D Rotation Matrix

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



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

3x3 2D Scaling Matrix

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

⇩

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

3x3 2D Matrix representations

- Translation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- Rotation:

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

- Scaling:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} sx & 0 & 0 \\ 0 & sy & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Linear Transformations

- A *linear* transformation can be written as:

x' = ax + by + c

OR

y' = dx + ey + f

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Why use 3x3 matrices?

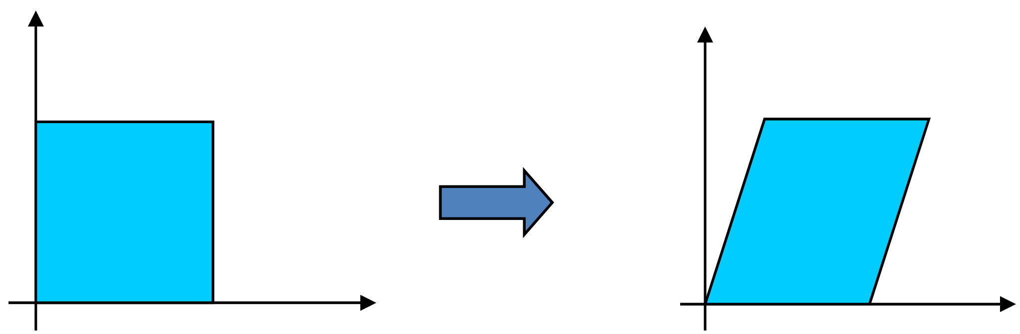
- So that we can perform all transformations using matrix/vector multiplications
- This allows us to *pre-multiply* all the matrices together
- The point (x,y) is represented using Homogeneous Coordinates (x,y,1)

Matrix concatenation

- Examine the computational cost of using four matrices ABCD to transform one or more points (i.e. p' = ABCDp)

- We could: apply one at a time
  - p' = D * p
  - p'' = C * p'
  - …
  - 4x4 * 4x1 for each transformation for each point
- Or we could: concatenate (pre-multiply matrices)
  - M=A*B*C*D
  - p' = M * p
  - 4x4 * 4x4 for each transformation
  - 4x4 * 4x1 for each point

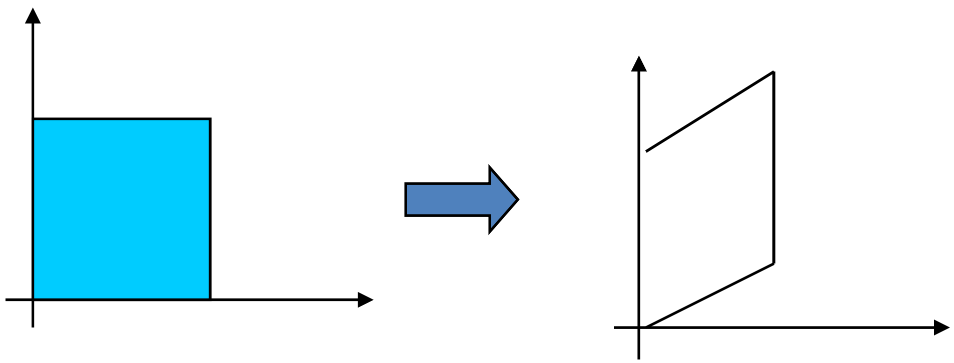
Shearing



- Y coordinates are unaffected, but x coordinates are translated linearly with y
- That is:
  - y' = y
  - x' = x + y * h

$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & h & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Shearing in y



$$\begin{vmatrix} x' \\ y' \\ 1 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 \\ g & 1 & 0 \\ 0 & 0 & 1 \end{vmatrix} * \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

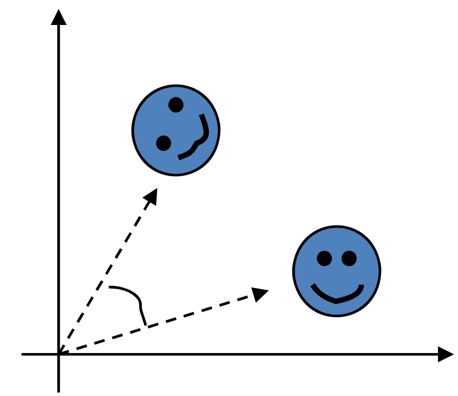
Interesting Facts:

- Any 2D rotation can be built using three shear transformations.
- Shearing will not change the area of the object
- Any 2D shearing can be done by a rotation, followed by a scaling, and followed by a rotation

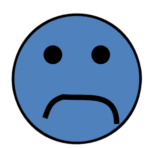
Local Rotation

- The standard rotation matrix is used to rotate about the origin (0,0)

$$\begin{matrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{matrix}$$

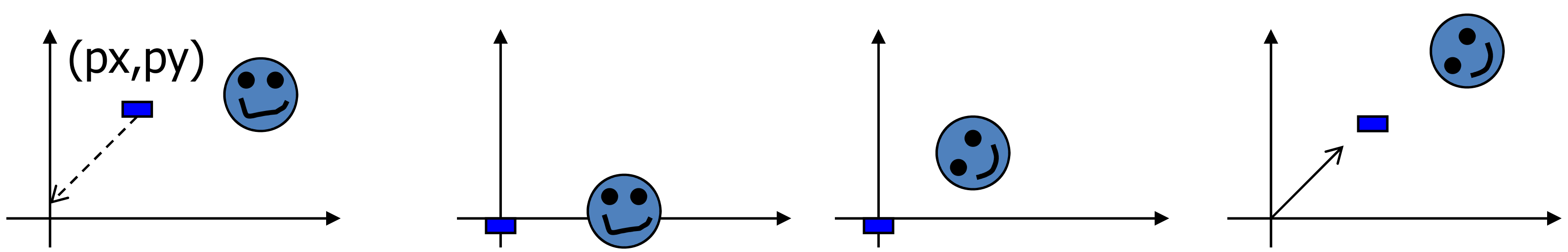


- What if I want to rotate about an arbitrary center?



Arbitrary Rotation Center

- To rotate about an arbitrary point P (px,py) by θ:
  - Translate the object so that P will coincide with the origin: T(-px, -py)
  - Rotate the object: R(θ)
  - Translate the object back: T(px, py)



Arbitrary Rotation Center

- Translate the object so that P will coincide with the origin: T(-px, -py)
- Rotate the object: R(q)
- Translate the object back: T(px,py)

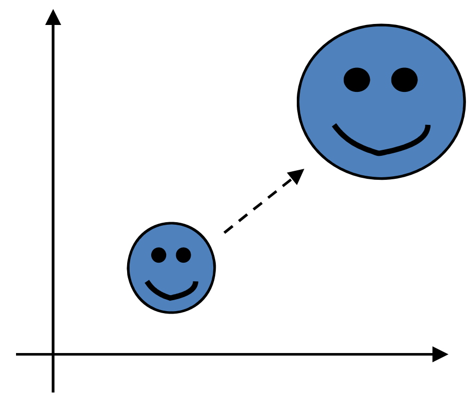
- As a matrix multiplication
  - p' = T[px,py] * R[q] * T[-px, -py] * P

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

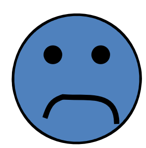
Local scaling

- The standard scaling matrix will only anchor at (0,0)

$$\begin{matrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{matrix}$$

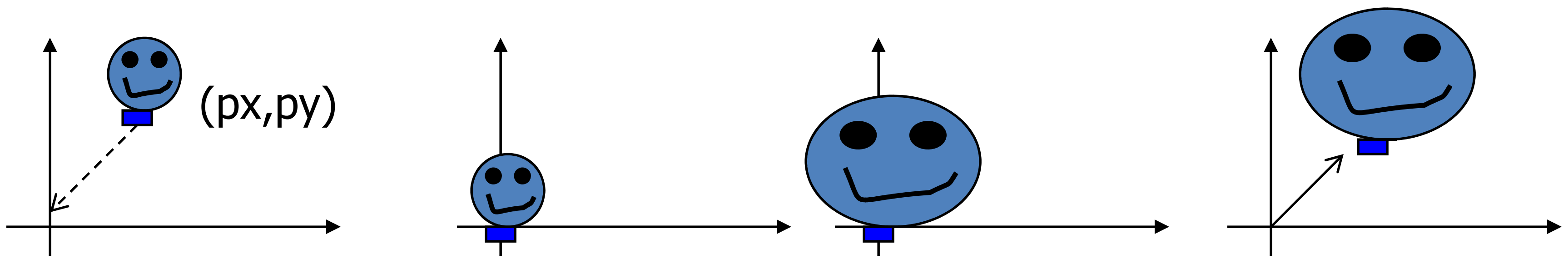


- What if I want to scale about an arbitrary pivot point?



Arbitrary Scaling Pivot

- To scale about an arbitrary pivot point P (px,py):
  - Translate the object so that P will coincide with the origin: T(-px, -py)
  - Scale the object: S(sx, sy)
  - Translate the object back: T(px,py)



Moving to 3D

- Translation and Scaling are very similar, just include z dimension
- Rotation is more complex

3D Translation

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Rotations – rotation about primary axes

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

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

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

3D Scaling

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Vectors and Matrices in Unity

- Vector2 (reference page)
- Vector3 (reference page)
- Vector4 (reference page)

- Matrix4x4 (reference page)

Vector3

- Data members
  - x,y,z floats
- Operations/Operators
  - set(x,y,z)
  - +,- vector-vector operations
  - *,/ vector-scalar operations
  - == comparison (has some flexibility to handle nearly equal values)
  - Normalize, Distance, Dot

Code example with Vector3

In a script attached to a GameObject:

```
Vector3 temp;
temp = new Vector3(3,5,8);
transform.position = temp;
```

Matrix4x4

- this [int row, int column]
- this [int index]
  - index: row+column*4
- GetColumn, GetRow
- SetColumn, SetRow
- * operator

- Note: Unity does not store a modeling transformation matrix for each object

Transformations in Unity

- transform (reference)
  - Position, rotation, and scale of an object
- Methods
  - Translate
  - Rotate
- Data
  - position
  - rotation

transform.Translate

- **function Translate (**

  translation : Vector3,

relativeTo: Space = Space.Self)

- translation vector – tx,ty,tz
- Space.Self – local coordinate system
- Space.World – world coordinate system

transform.Rotate

- **function Rotate (**

eulerAngles: Vector3,

relativeTo: Space = Space.Self)

- Applies a rotation

  eulerAngles.zdegrees around the z axis,
  eulerAngles.x degrees around the x axis, and
  eulerAngles.ydegrees around the y axis
  (in that order).

transform.Rotate

- **function Rotate (**

eulerAngles: Vector3,
relativeTo: Space = Space.Self)

- Space.Self – rotate about local coordinate frame (center of prebuilt GameObjects, could be anywhere for an artist made model)
- Space.World – rotate about world coordinate frame (origin (0,0,0))

On your own activity with transform.Rotate

- In your script for lab1, in Update() add the statement

transform.Rotate(0,1,0);

- Run the animation and hold down the ‘a’ key, is the result what you were expecting?
- Try it again with

transform.Rotate(0,1,0, Space.World);

- Also experiment with using Space.World in a call to Translate