

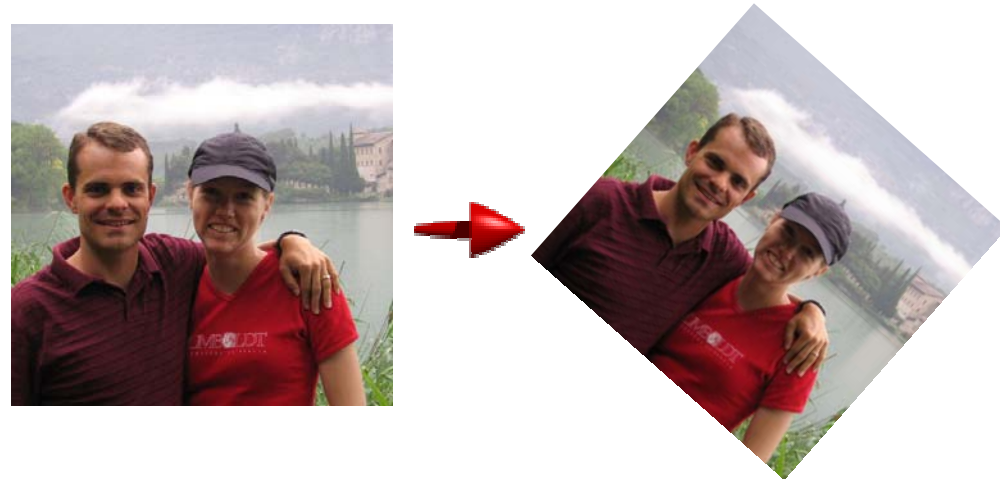
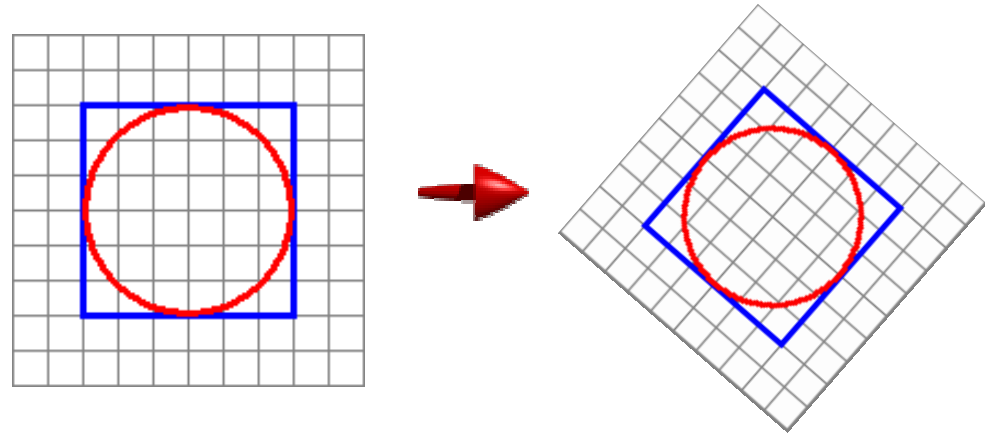
2D Imaging and Transformations

Computer Graphics
COMP 770 (236)
Spring 2007

Instructor: Brandon Lloyd

2D geometric transforms

- Functions for mapping points from one place to another
- Geometric transforms can be applied to
 - drawing primitives (lines, conics, triangles)
 - pixel coordinates of an image



Translation

- Translations have the following form:

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

- ***inverse function***: undoes the translation:

$$\begin{aligned}x &= x' - t_x \\ y &= y' - t_y\end{aligned}$$

- ***identity***: leaves every point unchanged.

$$\begin{aligned}x' &= x + 0 \\ y' &= y + 0\end{aligned}$$

Groups and composition

- Translations:
 1. There exists an inverse mapping for each function
 2. There exists an identity mapping
- Functions with these properties are closed under composition

$$x' = \underbrace{T_1 T_2 T_3 \cdots T_n}_{T'} x$$

- Referred to as an algebraic **group**

2D rotations

- Another group - rotation about the origin:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = R \begin{bmatrix} x \\ y \end{bmatrix}$$

$$R^{-1} = \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

$$R_{\theta=0} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

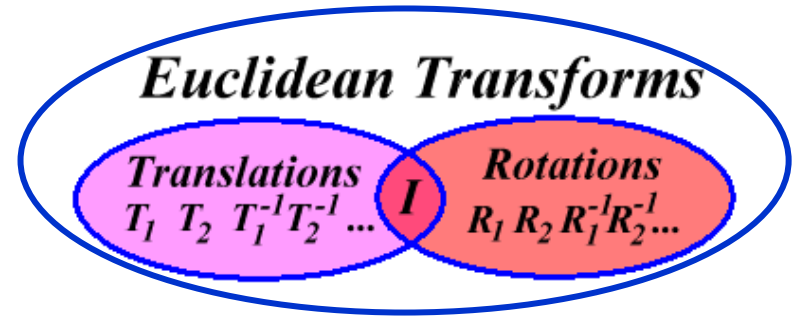
Euclidean transforms

■ Euclidean Group

- translations + rotations

■ Properties:

- Preserve distances
- Preserve angles
- How do you represent these functions?



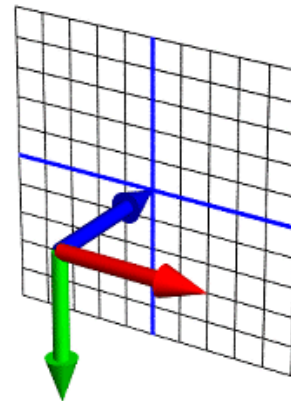
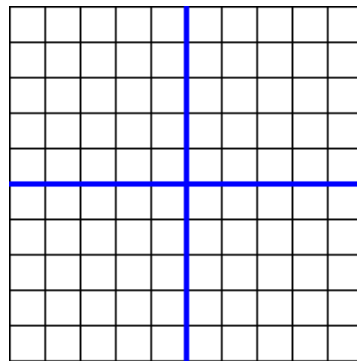
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Problems with this form

- Translation and rotation considered separately
- Inverse transform involves multiple steps
- Order matters between the R and T parts

$$R(T(\bar{x})) \neq T(R(\bar{x}))$$

- *Problem remedied by considering our 2D plane as a subspace within 3D.*



Choosing a Subspace

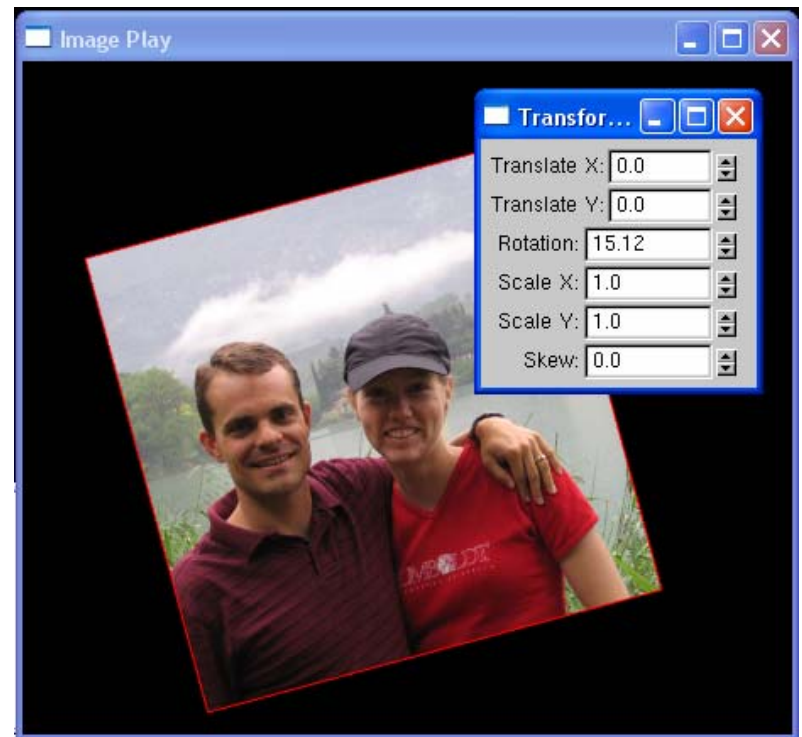
- Can use **any** planar subspace that does not contain the origin
- WLOG assume our 2D space lies on the 3D plane $z = 1$. Now we can express all Euclidean transforms in matrix form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & t_x \\ \sin \theta & \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- This gives a three parameter group of transformations.

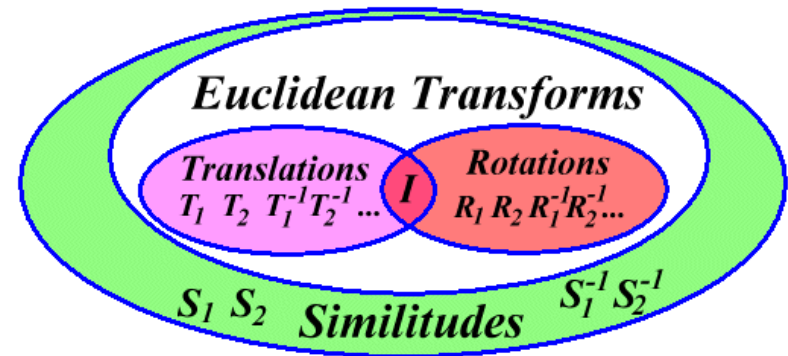
Playing with Euclidean transforms

- In what order are the translation and rotation performed?
- Will this family of transforms always generate points on our chosen 3-D plane? Why?



Similitude transforms

- Similitude Group:
 - 4-parameter superset of Euclidean transforms
 - Also called *similarities*

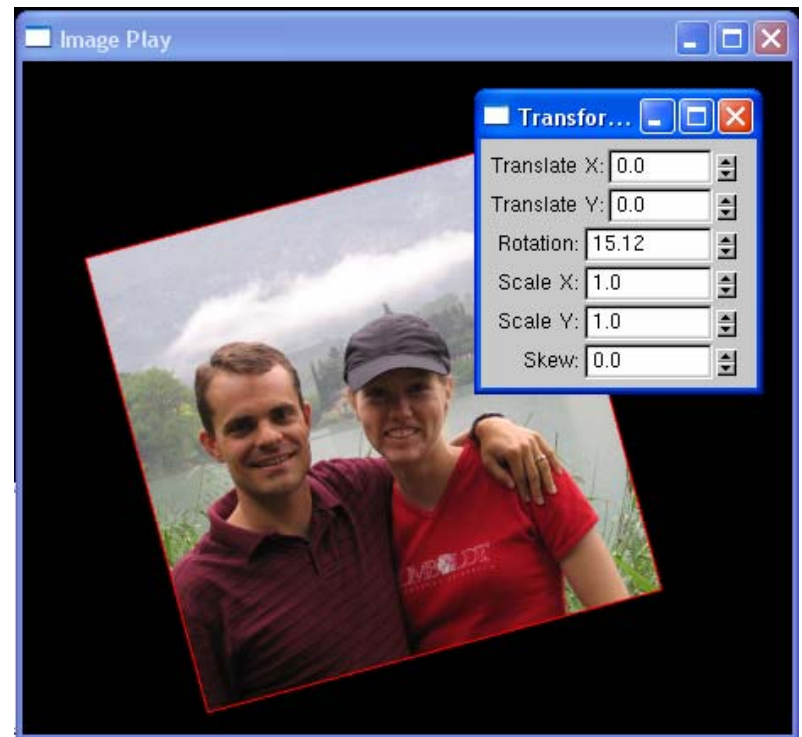


- Properties:
 - Distances between points are changed by a fixed ratio
 - Angles are preserved
 - Maintains a "Similar" shape (similar triangles, circles map to circles, etc.)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \sigma \cos \theta & -\sigma \sin \theta & t_x \\ \pm \sigma \sin \theta & \pm \sigma \cos \theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Playing with Similitude transforms

- Adds reflections
- Scales in x and y must be the same. Why?
- Order?
- Will this family of transforms always generate points on the chosen plane? Why?



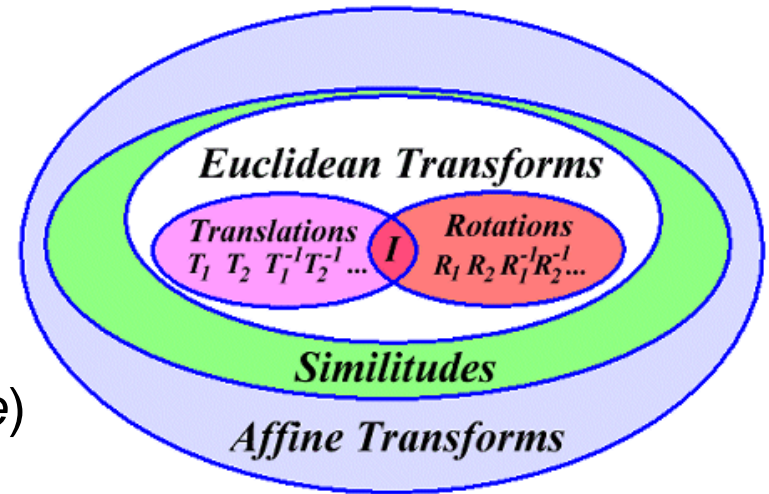
Affine transforms

■ Affine Group

- 6-parameters

■ Properties :

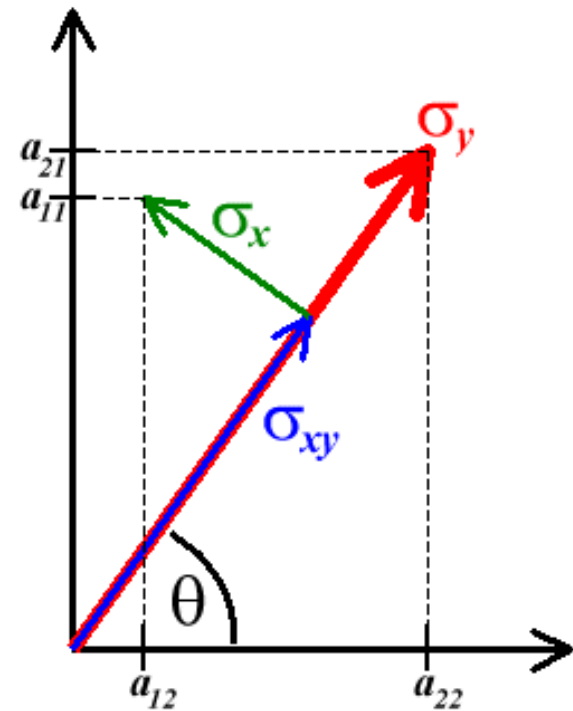
- Preserve our selected plane (sometimes called the *Affine plane*)
- Preserve parallel lines
- **Rigid body** transforms



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \sigma_{xy} \sin \theta + \sigma_x \cos \theta & \sigma_{xy} \cos \theta - \sigma_x \sin \theta \\ \sigma_y \sin \theta & \sigma_y \cos \theta \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \begin{matrix} t_x \\ t_y \\ 1 \end{matrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Affine transforms

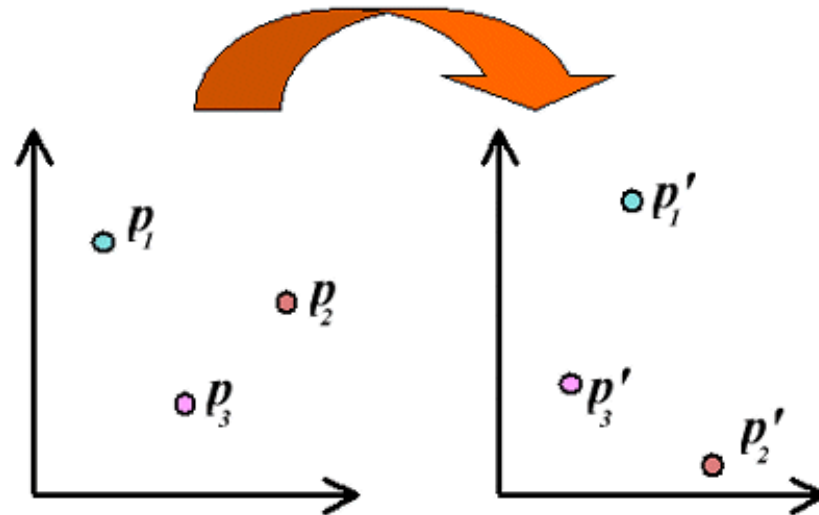
- σ_x scales the x-dimension
- σ_y scales the y-dimension
- σ_{xy} is often called the *skew* parameter



Mapping parameters
to matrix elements

Determining affine transforms

- Affine transform uniquely determined by three corresponding points (non-collinear)



Solution method

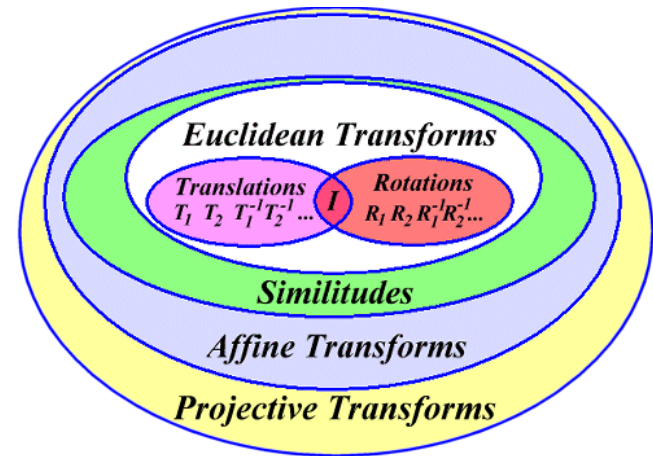
- We know coordinates before and after transform
- We want matrix entries
- 6 equations with 6 unknowns

$$\underbrace{\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \end{bmatrix}}_{\mathbf{x}'} = \underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix}}_{\mathbf{a}}$$

$$\mathbf{a} = \mathbf{X}^{-1} \mathbf{x}$$

Projective transforms

- Projective group:
 - 8 parameters
- Properties:
 - Most general 2D transform we can represent with a matrix
- Causes points to not lie on the plane. We need to deal with



$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Projective space

■ *Projective space:*

- the mapping of points from an N -D space to an M -D subspace ($M < N$)

■ Mapping points (x,y,w) in \mathbf{P}^2 back to plane in \mathbf{R}^2 :

- Intersect the line from $(0,0,0)$ to (x,y,w) with the $w=1$
- Divide by w . Gives $(x/w, y/w, 1)$

■ Origin cannot be uniquely identified

- Disallowed.
- This is why we selected a plane that did not contain the origin.

Projective transforms

- Transformed points defined to within a non-zero scale factor.
- Applying non-zero scale to transform gives the same result:

$$\alpha \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & p_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- We can choose α so that one of the parameters of our matrix is 1 (i.e. $p_{33} = 1$).

Determining projective transforms

Projective transform:

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} \\ p_{21} & p_{22} & p_{23} \\ p_{31} & p_{32} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Can be expressed as a linear rational equation:

$$x' = \frac{p_{11}x + p_{12}y + p_{13}}{p_{31}x + p_{32}y + 1} \quad y' = \frac{p_{21}x + p_{22}y + p_{23}}{p_{31}x + p_{32}y + 1}$$

Rearranging terms gives a linear expression in the coefficients:

$$\begin{aligned} x' &= p_{11}x + p_{12}y + p_{13} - p_{31}xx' - p_{32}yx' \\ y' &= p_{21}x + p_{22}y + p_{23} - p_{31}xy' - p_{32}yy' \end{aligned}$$

Determining projective transforms

- Uniquely defined by the mapping of four points

$$\begin{bmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -x'_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y'_3x_3 & -y'_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4x_4 & -y'_4y_4 \end{bmatrix} \begin{bmatrix} p_{11} \\ p_{12} \\ p_{13} \\ p_{21} \\ p_{22} \\ p_{23} \\ p_{31} \\ p_{32} \end{bmatrix}$$

Projective example



OpenGL imaging

- `glDrawPixels()` – Writes an array of pixels to the framebuffer
- `glReadPixels()` – Reads an region of the framebuffer into an array of pixels in main memory
- `glCopyPixels()` – Copies a region from one part of the framebuffer to another
- `glRasterPos*()` – Sets the current drawing position for `glDrawPixels()` and destination position of `glCopyPixels()`

OpenGL Setup

- Map world coordinates to screen coordinates
 - Typical when working with pixel level imaging
 - Use gluOrtho2D

```
glViewport(0, 0, width, height);  
glClearColor(0, 0, 0, 1);  
glMatrixMode(GL_PROJECTION);  
glLoadIdentity();  
gluOrtho2D(0, width, 0, height);  
glMatrixMode(GL_MODELVIEW);  
glLoadIdentity();
```

Displaying an image

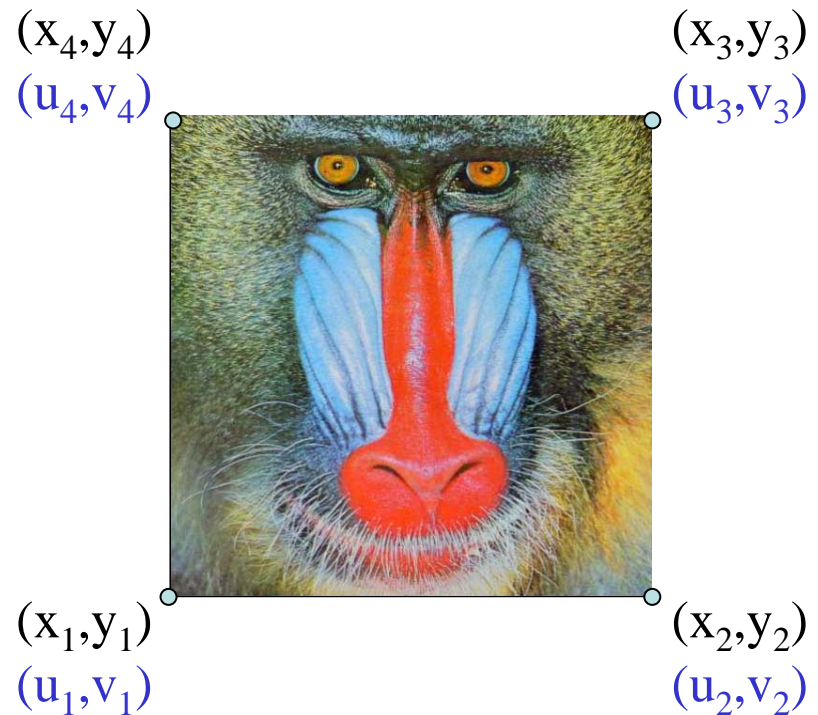
- Code to display a list of images:

```
glClearColor(0, 0, 0, 1)
glClear(GL_COLOR_BUFFER_BIT)
for im in imageList:
    glRasterPos2i(im.x,im.y);
    glDrawPixels(im.width, im.height,
                 GL_RGB, GL_UNSIGNED_BYTE,
                 im.data)

glFlush()
```

Texture mapping

- Imaging functions work directly with pixels
 - Raster position modified by matrix stack but pixels are not.
- Textures “attach” an image to geometry
 - Specify texture coordinates at vertices
 - What kind of transform is applied to the image by this mapping



How to set up a texture map

Controls the interpolation used when texturing

```
glBindTexture(GL_TEXTURE_2D, self.texID)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST)
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST)
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE)
```

```
if (self.channels == 3):
```

```
    glTexImage2D(GL_TEXTURE_2D, 0, 3, self.width, self.height,
                0, GL_RGB, GL_UNSIGNED_BYTE, self.data)
```

```
else:
```

```
    glTexImage2D(GL_TEXTURE_2D, 0, 4, self.width, self.height,
                0, GL_RGBA, GL_UNSIGNED_BYTE, self.data)
```

Associates an integer ID to the texture

Associate the image data with the textures

Makes the texture a “label” texture. Geometry gets color directly from the texture

Applying the texture map

```
glClear(GL_COLOR_BUFFER_BIT)
glEnable(GL_TEXTURE_2D)
for im in imageList:
    glPushMatrix()
    glTranslated(im.tx, im.ty, 0)
    glScaled(im.sx, im.sy, 1)
    glRotated(im.theta, 0, 0, 1)
    glBindTexture(GL_TEXTURE_2D, im.texID)
    glBegin(GL_POLYGON)
        glTexCoord2d(0,1); glVertex2d(0, 0);
        glTexCoord2d(1,1); glVertex2d(w, 0);
        glTexCoord2d(1,0); glVertex2d(w, h);
        glTexCoord2d(0,0); glVertex2d(0, h);
    glEnd()
    glPopMatrix()
glFlush()
glutSwapBuffers()
```

Set up transform for geometry

Select current texture

Associate texture coordinates with vertices

Next time

- 3D transformations