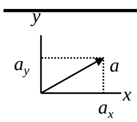
2D Transformations

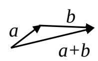
Uses of Transformations

- **Modeling: position and resize parts of a complex model;**
- **Viewing: define and position the virtual camera**
- **Animation: define how objects move/change with time**

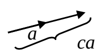
Review of vectors



Addition:



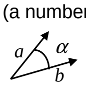
Multiplication by a number c:



Scalar (dot) product (a number):

2D: $(a,b) = a_x b_x + a_y b_y$

3D: $(a,b) = a_x b_x + a_y b_y + a_z b_z$



$(a,b) = |a||b|\cos\alpha \qquad (a,b) = (b,a)$

$(a, k_1 b + k_2 c) = k_1 (a,b) + k_2 (a,c)$

Cross product (a vector):

3D: $a \times b = \det \begin{pmatrix} \mathbf{e_x} & \mathbf{e_y} & \mathbf{e_z} \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{pmatrix}$

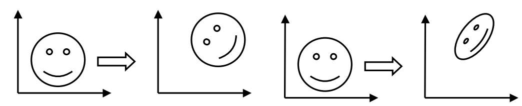
$|a \times b| = |a||b| \sin\alpha$

$a \times b = -b \times a$

$a \times (k_1 b + k_2 c) = k_1 a \times b + k_2 a \times c$

Rigid transformations

- **distance between points is preserved**



Any rigid transformation is a combination of a rotation and translation

Translation by vector t:

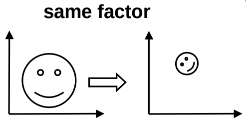
$$x' = x + t_x$$
$$y' = y + t_y$$

Rotation by angle θ:

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

Similarity trasformations

- **all distances between points change by the same factor**

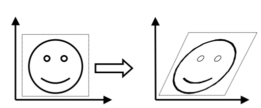


Composition of rotation, translation, and uniform scale

$$x' = sx$$
$$y' = sy$$

Affine transformations

- **map parrallel lines to parallel lines**
- **can be decomposed into rotation, translation, nonuniform scale and skew:**



$$x' = x + sy$$
$$y' = y$$

- **6 parameters**

Matrices and transformations

- **Rotation, scale, skew can be written as 2D matrix operations, but not translation!**

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

nonuniform scale: $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

skew: $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

general linear transform (combination of rotation, scale and skew) $$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Matrices and transformations

- **matrices are in one to one correspondence with linear transformations, i.e. functions such that**

$$F(k_1 a + k_2 b) = k_1 F(a) + k_2 F(b)$$

- **Affine transformations can be written in general form as**

$$f(a) = Ma + t$$

- **compositons of transformations correspond to matrix multiplication**
- **matrix multiplication is not commutative**

Homogeneous coordinates

- **replace 2d points with 3d points, last coordinate 1**
- **for a 3d point (x,y,w) the corrsponding 2d point is (x/w,y/w) if w is not zero**
- **3d points (x,y,0) correspond to directions (x,y), if x or y are not zero**
- **(0,0,0) does not correspond to a 2d point**
- **Geometric construction: 3d points are mapped to 2d points by projection to the plane z =1**

Matrices of basic transformations

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

$$\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$$ translation

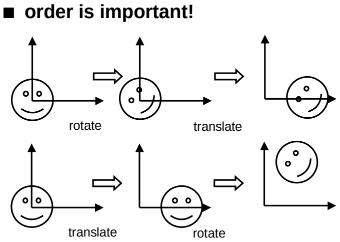
$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ scaling

$$\begin{bmatrix} 1 & s & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$ skew

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{14} \\ 0 & 0 & 1 \end{bmatrix}$$ general affine transform

Compositions of transformations

- **composition of transformations = product of matrices**
- **order is important!**



Line through two points

Equation of a line through 2 points in homogeneous coordinates:

for any point p **on the line, the vectors** $p - p_1, p - p_2$

are parallel, and their cross product is zero:

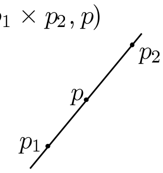
$$(p - p_1) \times (p - p_2) = 0$$

$$((p - p_1) \times (p - p_2), p) = (p_1 \times p_2, p)$$

Line through points $p_1,\ p_2$

$$(p_1 \times p_2, p) = (n, p) = 0$$

homogeneous points



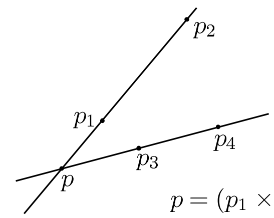
Intersection of two lines

- Given two lines $(n_1, p) = 0, \quad (n_2, p) = 0$ find intersection point p satisfying both line equations
- think about $n_1,\ n_2,\ p$ as 3d vectors!
- equivalent formulation: find vector orthogonal to both $n_1,\ n_2$
- cross product has this property:

$$p = n_1 \times n_2$$

is the intersection point in homogeneous coordinates

Intersection of two lines

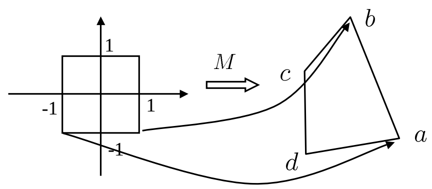


$$p = (p_1 \times p_2) \times (p_3 \times p_4)$$

intersection of two lines through two pairs of points

Mapping rectangles to quads

Problem: map a square $[-1, 1] \times [-1, 1]$ to an arbitrary quadrilateral with vertices $a,\ b,\ c,\ d$ using a linear map M



Mapping rectangles to quads

Idea: look at images of homogeneous points $[1, 0, 0]$ $[0, 1, 0]$ $[0, 0, 1]$

they are columns of the matrix M

What are points with last component zero?

$[1, 0, 0]$ "intersection" of horizontal sides (point at infinity for direction [1,0]

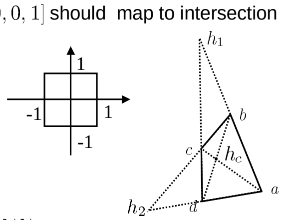
$[0, 1, 0]$ "intersection" of vertical sides (point at infinity for direction [0,1]

$[0, 0, 1]$ intersection of diagonals

Mapping rectangles to quads

Intersections of lines map to intersections of images of lines:

$[1, 0, 0]$ should map to intersection of (a, b) (c, d)
$[0, 1, 0]$ should map to intersection of (b, c) (a, d)
$[0, 0, 1]$ should map to intersection of (a, c) (b, d)



Mapping rectangles to quads

Using formulas for intersections of lines, we get

$$h^1 = (a \times b) \times (c \times d)$$
$$h^2 = (a \times d) \times (b \times c)$$
$$h^c = (a \times c) \times (b \times d)$$

Finally we obtain the transform:

$$M = [h_1 h_2 h_c] = \begin{bmatrix} h_1^x & h_2^x & h_c^x \\ h_1^y & h_2^y & h_c^y \\ h_1^w & h_2^w & h_c^w \end{bmatrix}$$

Note the last line is not $[0, 0, 1]$, this is not an affine transform!

Texture Mapping

- Put image on a quadrilateral (can be thought of as rectangle viewed in perspective)
- Recall scan conversion: we traverse pixels along scan lines intersecting the quadrilateral
- to determine color, need the ***inverse*** of M : for a pixel p need to determine the point q on the image which is mapped to p by M

Let $M^{-1} = \begin{bmatrix} A & B & C \\ D & E & F \\ I & J & K \end{bmatrix}$

Texture Mapping

For pixel $p = [x, y, 1]$ **homogeneous texture coordinates are** $M^{-1}p = [s, t, r]$

Nonhomogeneous (i.e. after division) coordinates are given by

$$s = \frac{Ax + By + C}{Ix + Jy + K} \qquad t = \frac{Dx + Ey + F}{Ix + Jy + K}$$

Texture Mapping

Algorithm (without antialiasing)

- do polygon scan conversion;
- for each pixel determine texture coordinates
- round to the nearest integers (I,j)
- use the color of the pixel (I,j) in the texture image

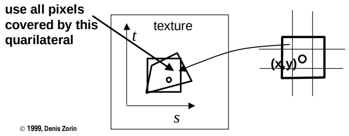
Incremental Mapping

The standard trick: compute numerator and denominator for the inverse map incrementally

```
sn = A*xl + B*ybot + C;
tn = D*xl + E*ybot + F;
den = I*xl + J*ybot + K;
for( y = ybot; y < ytop; y++ ){
   float snl = sn, tnl = tn , denl = den;
   for( x = xleft; x < xright; x++ ){
      Color rgb = Tex( snl/denl, tnl/denl );
      SetPixel( x, y, rgb );
      snl += A; tnl += D; denl += I;
   }
   sn += B + dxdy*A;
   tn += E + dxdy*D;
   den += J + dxdy*I;
}
```

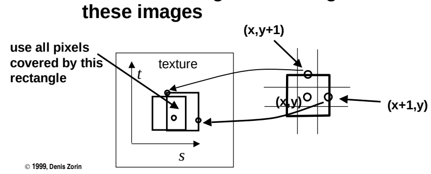
Antialiasing

- Aliasing for texture mapping is similar to the aliasing that can be observed when scaling an image, but the scale factors vary
- the ideal correct approach is to take a filter centered in the quadrilateral at a pixel (x,y), and map it back to the texture using the inverse mapping. To compute a pixel, sum with weights given by the filter the covered texture pixels.



Antialiasing

- Doing the right thing is too complicated: need to compute the image of a square under the image map, find all pixels covered by it etc.
- approximation: look at inverse images of the centers of adjacent pixels; use the coordinate- aligned rectangle containing these images



Antialiasing

- for each pixel (x,y), compute inverse mapping for three points: (x,y), (x+1,y) and (x,y+1)
- for these three points in the texture image find the minimal coordinate-aligned rectangle containing all three points
- the two dimensions of the rectangle give us two scaling factors to use for filtering
- Use the resizing algorithm from the previous homework to compute pixels; scaling factors change from pixel to pixel

Scale factors

A function that estimates scaling factors:

```
EstimateScaleFactor( snl, tnl, denl ){
   s = snl/denl;
   t = tnl/denl;
   sdx = (snl + A) / (denl + I) - s;
   tdx = (tnl + D) / (denl + I) - t;
   sdy = (snl + B) / (denl + J) - s;
   tdy = (tnl + E) / (denl + J) - t;

   scaleS = max(abs(sdx + sdy), abs(sdx), abs(sdy));
   scaleT = max(abs(tdx + tdy), abs(tdx), abs(tdy));
}
```

Scaling factor a_s in the direction s is 1/scaleS, in the direction t, 1/scaleT