CSC 433/533 Computer Graphics

Alon Efrat
Credit: Joshua Levine

Lecture 18 Transformations and Science Graphs

Oct. 30, 2020

Geometric Transformations as Matrices

- Operations to arrange objects in a scene, view them with cameras, and get them on a screen all can be encoded with linear algebra
  - Geometric operations: rotation, translation, scaling, projection, and more…
- These transforms operate differently on points, displacement vectors, and surface normals

2D Linear Transformations

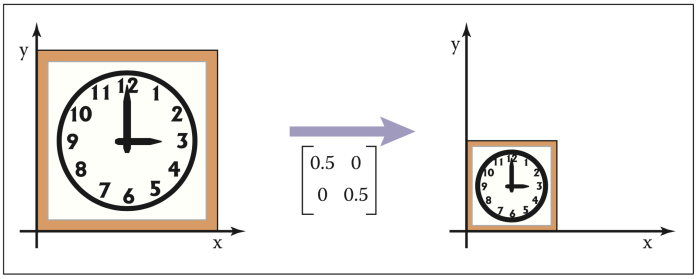
- We can transform points in a 2D coordinate system by multiplying the point (a vector) by a matrix (the transformation):
- e.g. Multiply the matrix A by $\mathbf{x}$ = (x,y), or A$\mathbf{x}$:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

- Such transformations are called *linear* because they satisfy the relationship that A(a$\mathbf{x}_1$+$\mathbf{x}_2$) = aA$\mathbf{x}_1$ + A$\mathbf{x}_2$

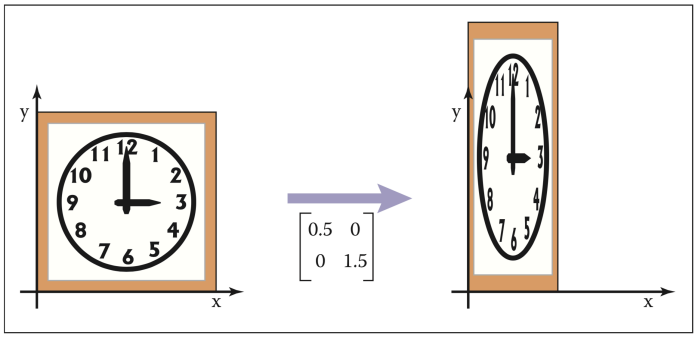
Scaling

- Can be uniform, where $s_x = s_y$. $\begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} s_x x \\ s_y y \end{bmatrix}$



Scaling

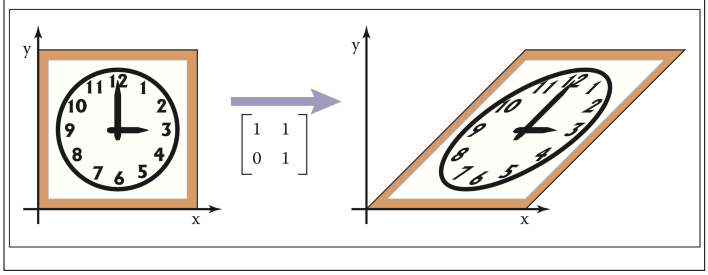
- Can also be nonuniform, where $s_x \neq s_y$.



Shearing

- Horizontal shearing shifts each row based on the y value.

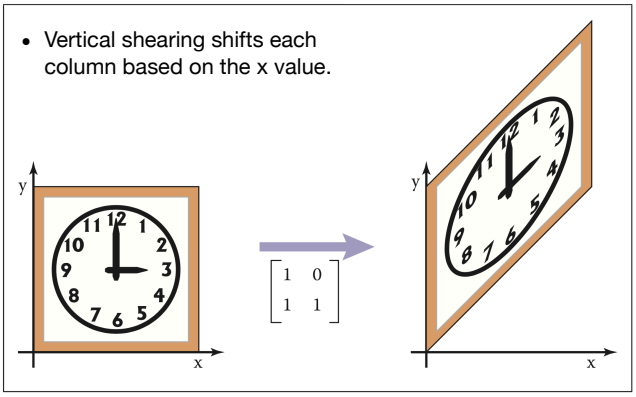
$$\text{shear-x}(s) = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}$$



Shearing

$$\text{shear-y}(s) = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$$

- Vertical shearing shifts each column based on the x value.

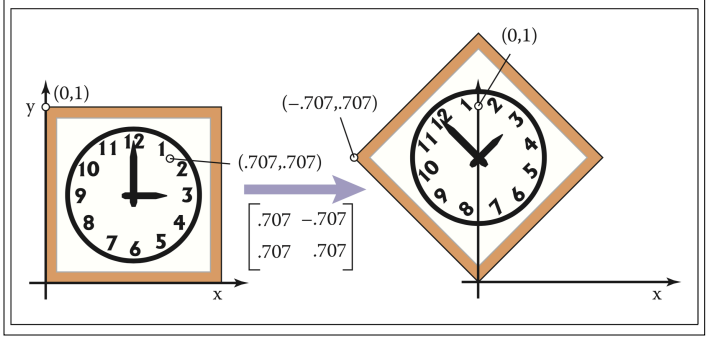


Rotation

$$\text{rotate}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

- Rotate counterclockwise by an angle ϕ about the origin.

$$\begin{bmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}$$

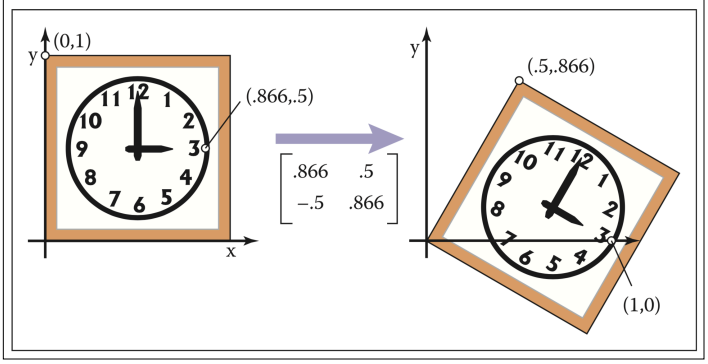


Rotation

$$\text{rotate}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix}$$

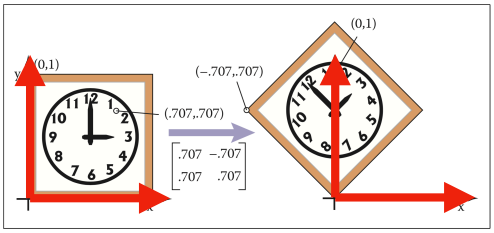
- Clockwise rotations can also be represented by negative angles

$$\begin{bmatrix} \cos\frac{-\pi}{6} & -\sin\frac{-\pi}{6} \\ \sin\frac{-\pi}{6} & \cos\frac{-\pi}{6} \end{bmatrix} = \begin{bmatrix} 0.866 & 0.5 \\ -0.5 & 0.866 \end{bmatrix}$$



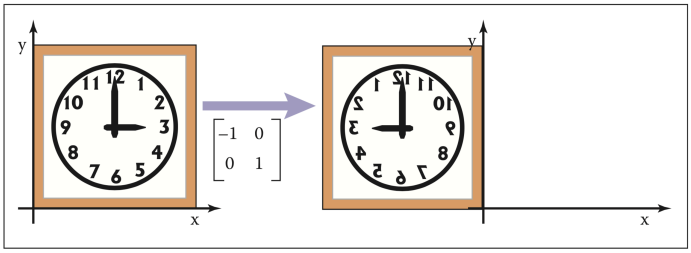
Rotations are Always Orthogonal Matrices

- Recall: An **orthogonal matrix** always has columns and rows that are orthogonal unit vectors
- Implication: Have the effect of rotating the coordinate axes

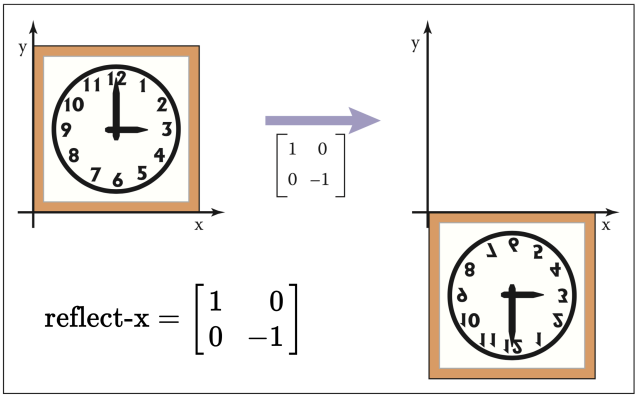


Reflection

$$\text{reflect-y} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$



Reflection



$$\text{reflect-x} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Composition

- Transformations can be **composed** to perform combinations of transformations.
- For example, one could first rotate with matrix R and then scale with matrix S
- Applied to a point $\mathbf{v_1}$, this would be
  - $\mathbf{v_2} = R\mathbf{v_1}$ to rotate and then
  - $\mathbf{v_3} = S\mathbf{v_2} = SR\mathbf{v_1}$
    $= (SR)\mathbf{v_1} = T\mathbf{v_1}$ where T = SR

Order Matters for Composition

- First rotate, then non-uniform scale
- Rotate:

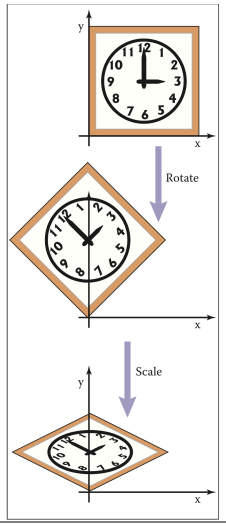
$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}$$

- Scale:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

- Combined:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.707 \\ 0.353 & 0.353 \end{bmatrix}$$



Order Matters for Composition

- First non-uniform scale, then rotate
- Rotate:

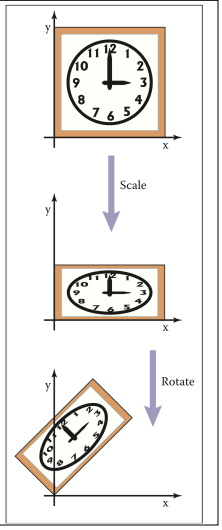
$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix}$$

- Scale:

$$\begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix}$$

- Combined:

$$\begin{bmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0.5 \end{bmatrix} = \begin{bmatrix} 0.707 & -0.353 \\ 0.707 & 0.353 \end{bmatrix}$$



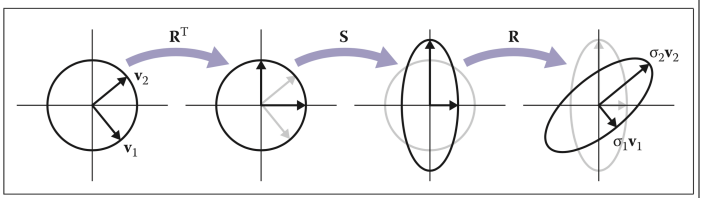
Decomposition

- Recall: a **symmetric matrix** is any matrix M such that $M = M^T$
- Let's consider a special type of composition, of the form RSR^T
  - Motivation: scaling on arbitrary axis, e.g. rotate, then scale, then rotate back
  - Example: $\text{rotate}(-45^\circ)\text{scale}(1.5, 1)\text{rotate}(45^\circ)$

$$\begin{bmatrix} 1.25 & -0.25 \\ -0.25 & 1.25 \end{bmatrix}$$

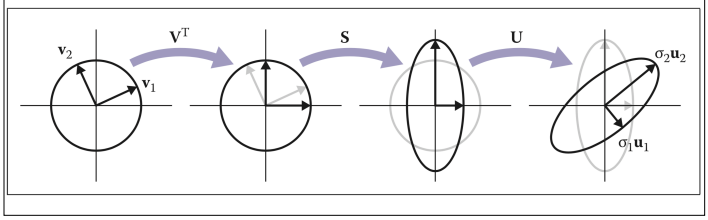
Decomposition

- RSR^T is always a symmetric matrix
  - Why? $(RSR^T)^T = R^{TT}S^TR^T = RSR^T$)
- Any symmetric metric A can be decomposed to $A = RSR^T$
  - All rotations R, happen to be **orthogonal matrices**.
- Any symmetric transformation matrix is a scale in some axis



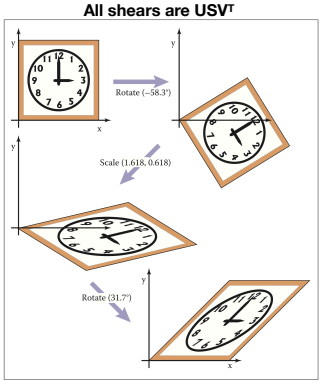
Decomposition

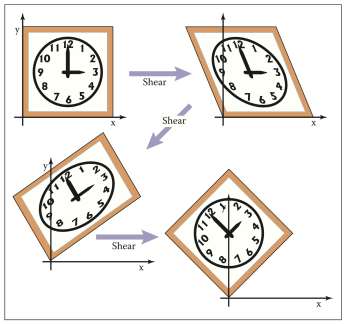
- Any arbitrary matrix can be decomposed to A = USV^T where U and V are orthogonal matrices (but not necessarily rotations)
  - This is called **singular value decomposition**
- This has a similar interpretation, but with different rotations before/after the scale



Some Consequences

All shears are USV^T





Any rotation is three shears (Paeth decomposition)

Inversion

- Interpreting a matrix M as a geometric transformation, how might we undo the operation?
- We can apply the **inverse** transformation, it turns out by applying the inverse matrix, M^{-1}
  - Recall that $MM^{-1} = M^{-1}M = I$, the identity matrix
- This is true for all operations we've discussed, e.g. scale by s is undone by scale of 1/s, rotate by angle ϕ is undone by rotate of angle -ϕ
- **Key point:** the algebraic inverse *is* the geometric inverse

3D Linear Transformations

3D Linear Transformations

- We can transform points in a 3D coordinate system by multiplying the point (a vector) by a matrix (the transformation), just like in 2D!
- The only difference is we will use 3x3 matrices A by **x** = (x,y,z), or A**x**, e.g. for scale and shear:

$$\text{scale}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & s_z \end{bmatrix}$$

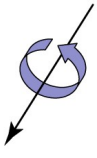
$$\text{shear-x}(d_y, d_z) = \begin{bmatrix} 1 & d_y & d_z \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Rotations in 3D

- In 2D, a rotation is about a point
- In 3D, a rotation is about an axis

convention: positive rotation is CCW

2D



3D

convention: positive rotation is CCW when axis vector is pointing at you

Rotations about 3D Axes

- In 3D, we need to pick an axis to rotate about

$$\text{rotate-z}(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

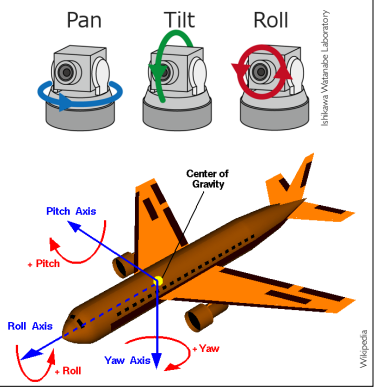
- And we can pick any of the three axes

$$\text{rotate-x}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix}$$

$$\text{rotate-y}(\phi) = \begin{bmatrix} \cos\phi & 0 & \sin\phi \\ 0 & 1 & 0 \\ -\sin\phi & 0 & \cos\phi \end{bmatrix}$$

Building Complex Rotations from Axis-Aligned Rotations

- Rotations about **x**, **y**, **z** are sometimes called **Euler angles**
- Build a combined rotation using matrix composition



Arbitrary Rotations

- To rotate about any axis: we change the coordinate space we are working in, using orthogonal matrices.
- Consider orthogonal matrix R_{uvw}, form by taking three orthogonal vectors **u**, **v**, and **w**:

Property of orthogonal vectors:

$$\mathbf{u}\cdot\mathbf{u} = \mathbf{v}\cdot\mathbf{v} = \mathbf{w}\cdot\mathbf{w} = 1$$
$$\mathbf{u}\cdot\mathbf{v} = \mathbf{v}\cdot\mathbf{w} = \mathbf{w}\cdot\mathbf{u} = 0$$

$$\mathbf{R}_{uvw} = \begin{bmatrix} & \mathbf{u} & \\ & \mathbf{v} & \\ & \mathbf{w} & \end{bmatrix}$$

Arbitrary Rotations

- What happens when we apply R_{uvw} to any of the basis vectors, e.g.:

$$\mathbf{R}_{uvw}\mathbf{u} = \begin{bmatrix} \mathbf{u}\cdot\mathbf{u} \\ \mathbf{v}\cdot\mathbf{u} \\ \mathbf{w}\cdot\mathbf{u} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = \mathbf{x}$$

- But this means that if we apply R_{uvw}^T to the Cartesian coordinate vectors, e.g.:

$$\mathbf{R}_{uvw}^{\mathrm{T}}\mathbf{y} = \begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} x_v \\ y_v \\ z_v \end{bmatrix} = \mathbf{v}$$

Arbitrary Rotations

- This means that if we want to rotation around an arbitrary axis, we need only to use a change of coordinates
- E.g. to rotate around a direction **w**, we
  - Compute orthogonal directions **u**, **v**, and **w**
  - Change the **uvw** axes to be **xyz** (R_{uvw})
  - Apply a rotate-z()
  - Finally, change the axes back to **uvw** (R_{uvw}^T)

$$\begin{bmatrix} x_u & x_v & x_w \\ y_u & y_v & y_w \\ z_u & z_v & z_w \end{bmatrix} \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & y_u & z_u \\ x_v & y_v & z_v \\ x_w & y_w & z_w \end{bmatrix}$$

R_{uvw}^T rotate-z() R_{uvw}

Transformations for Shapes in Computer Graphics

- These transformations also apply to transforming geometric objects
- Parametric forms:
  - We can transform the points directly, e.g. M(**p**(t)) to transform the parametric positions **p**(t)
- Implicit forms:
  - We invert the transform and test the predicate, e.g. if $f(M^{-1}(\mathbf{p})) = 0$ then **p** is on the transformed implicit shape

Affine Transformations

Translation

- Using the mathematics we've described so far, what about moving objects in space.
- Problem (in 2D), we have:

$$\begin{aligned} x' &= m_{11}x + m_{12}y \\ y' &= m_{21}x + m_{22}y \end{aligned}$$

- But we want:

$$\begin{aligned} x' &= m_{11}x + m_{12}y + x_t \\ y' &= m_{21}x + m_{22}y + y_t \end{aligned}$$

Homogeneous Coordinates

- To put this into one system of linear equations, we **promote** (increase the dimensionality) by adding a component w = 1 for vectors

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & x_t \\ m_{21} & m_{22} & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11}x + m_{12}y + x_t \\ m_{21}x + m_{22}y + y_t \\ 1 \end{bmatrix}$$

- Implements a linear transformation followed by a translation (x_t,y_t)
- These transformations are called **affine transformations**:
  - Like linear transformations, they keep straight lines straight and parallel lines parallel, but they do not preserve the origin

Homogeneous Coordinates

- We promote all points (x,y) to (x,y,w=1), and similarly in 3D we promote (x,y,z) to (x,y,z,w=1)
- These new coordinates are called **homogeneous coordinates**
- Can be thought of as a clever bookkeeping scheme, but also have a geometric interpretation, compare the following matrix with a standard shear:

$$\begin{bmatrix} 1 & 0 & x_t \\ 0 & 1 & y_t \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x + x_t z \\ y + y_t z \\ z \end{bmatrix}$$

Homogeneous Coordinates

- Composition works just as before, but using 3x3 multiplication instead of 2x2
- This approach is easier than keeping the linear transform and the translate separately stored

$$\begin{bmatrix} M_S & \mathbf{u}_S \\ 0 & 1 \end{bmatrix} \begin{bmatrix} M_T & \mathbf{u}_T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} (M_S M_T)\mathbf{p} + (M_S \mathbf{u}_T + \mathbf{u}_S) \\ 1 \end{bmatrix}$$

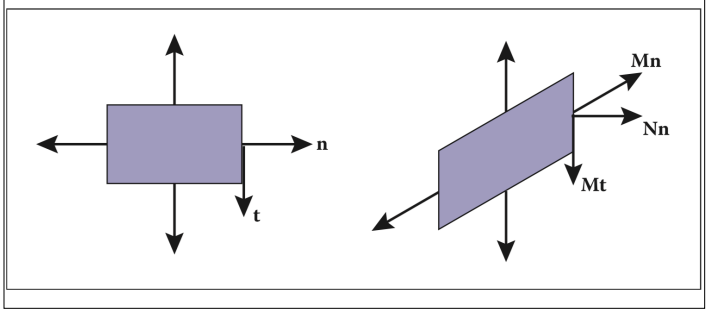
Transforming Points vs. Transforming Vectors

- Using homogeneous coordinates, we can differentiate between points and vectors
- Recall:
  - Vectors are just offsets (differences between points), and thus should be not affected by translation
  - Whereas points are represented by vectors offset from the origin
- Rule: vectors have w=0, whereas points have w=1:

$$\begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} M\mathbf{p} + \mathbf{t} \\ 1 \end{bmatrix} \quad \begin{bmatrix} M & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ 0 \end{bmatrix} = \begin{bmatrix} M\mathbf{v} \\ 0 \end{bmatrix}$$

Transforming Normals

- While differences between points transform OK, normals do not necessarily behave the same



Transforming Normals

- The problem is that the orthogonality constraint, that normals always point orthogonal to the surface, is not always preserved.
- One can solve for the correct transformation by observing that tangent vectors, $\mathbf{t}$, transform correctly and $\mathbf{t} \cdot \mathbf{n} = 0$.

$$\mathbf{n}^{\mathrm{T}}\mathbf{t} = \mathbf{0}$$

$$\mathbf{n}^{\mathrm{T}}\mathbf{t} = \mathbf{n}^{\mathrm{T}}\mathbf{I}\mathbf{t} = \mathbf{n}^{\mathrm{T}}\mathbf{M}^{-1}\mathbf{M}\mathbf{t} = 0$$

- So, we can transform normals using the inverse matrix

$$\left(\mathbf{M}^{-1}\right)^{\mathrm{T}}\mathbf{n}$$

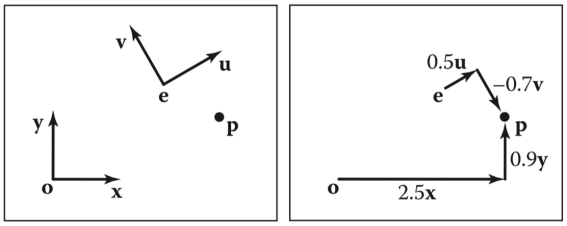
Coordinate Transformations

Coordinate Systems

- Points in space can be represented using an origin position and a set of orthogonal basis vectors:

$$\mathbf{p} = (x_p, y_p) \equiv \mathbf{0} + x_p\mathbf{x} + y_p\mathbf{y} \qquad \mathbf{p} = (u_p, v_p) \equiv \mathbf{e} + u_p\mathbf{u} + v_p\mathbf{v}$$

- Any point can be described in either coordinate system



Matrices for Converting Coordinate Systems

- Using homogenous coordinates and affine transformations, we can convert between coordinate systems:

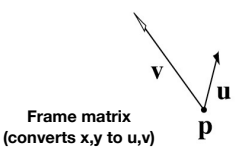
$$\begin{bmatrix} x_p \\ y_p \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x_e \\ 0 & 1 & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_u & x_v & 0 \\ y_u & y_v & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} x_u & x_v & x_e \\ y_u & y_v & y_e \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix}$$

- More generally, any arbitrary coordinate system transform:

$$\mathbf{P}_{uv} = \begin{bmatrix} \mathbf{x}_{uv} & \mathbf{y}_{uv} & \mathbf{o}_{uv} \\ 0 & 0 & 1 \end{bmatrix} \mathbf{p}_{xy}$$

Affine Change of Coordinates

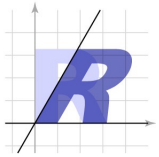
- It turns out this works even if u, v are not orthogonal.
- It also provides another way to interpret and construct transformation matrices



Frame matrix (converts x,y to u,v)

$$\begin{bmatrix} \mathbf{u} & \mathbf{v} & \mathbf{p} \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0.5 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Lec19 Required Reading

- FOCG, Ch. 7

Reminder: Assignment 05

Assigned: Wednesday, Oct. 30
Written Due: Monday, Nov. 11, 4:59:59 pm
Program Due: Wednesday, Nov. 13, 4:59:59 pm