points vs vectors

- vector $\vec{v}$, := motion between points in space
  - has the structure of a 3 dimensional linear/vector space.
  - addition and scalar multiplication have meaning
  - zero vector is no motion
  - cannot really translate motion
- point $\tilde{p}$:= a position in space
  - has the structure of a so-called 3D affine space.
  - addition and scalar mul don't make sense
  - zero doesn't make sense
  - subtraction does make sense, gives us a vector

$$\tilde{p} - \tilde{q} = \vec{v}$$

  - Conversely point and vector gives a point

$$\tilde{q} + \vec{v} = \tilde{p}$$

- new ideas points, frames, Cvecs4, affine transforms, affine matrices matrices

frames

- basis is three vectors

$$\vec{v} = \sum_i c_i \vec{b}_i$$

- for affine space we will use a frame
  - start with a chosen origin point $\tilde{o}$,
  - add to it a linear combination of vectors using coordinates c_i to get to any desired point $\tilde{p}$.

$$\tilde{p} = \tilde{o} + \sum_i c_i \vec{b} \quad = \quad \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} = \vec{\mathbf{f}}^t \mathbf{c}$$

4-coordinate vectors

- point is specified with a 4-coordinate vector
  - four numbers
  - last one is always 1
  - .... or 0. (and we get a vector)

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 0 \end{bmatrix} = \vec{v}$$

- the use of Cvec4s will also come in super handy with camera projections.

affine matrices

- a we will call a matrix an "affine matrix" if it is of the form

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- we perform an "affine transformation" on a point by placing an affine matrix between a frame and a 4-coordinate vector, just like with linear transforms

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} \Rightarrow$$
$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

- for short $\vec{\mathbf{f}}^t \mathbf{c} \Rightarrow \vec{\mathbf{f}}^t A \mathbf{c}$
- the data types work iff the 4th row is $[0, 0, 0, 1]$
  - two ways to show.
- similarly, we can apply an affine transform to a frame as $\vec{\mathbf{f}}^t \Rightarrow \vec{\mathbf{f}}^t A$

inverse

- inverse matrix $A^{-1}A = I$
- undoes the affine transform
- it too is an affine transform

building an affine transform from a Linear transform

- suppose i have a 3-by-3 matrix
- let me put it in the upper left of an affine matrix and apply it

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} \Rightarrow$$
$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$
$$= \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1' \\ c_2' \\ c_3' \\ 1 \end{bmatrix}$$

- this has the same effect on the coordinates as a 3by3 matrix multiply
- as if we applied a linear transform on the vectors connecting the origin to the point
- so we can use this to, say rotate a point about the origin
- see fig
- matrix shorthand for an upgraded linear transform

$$L = \begin{bmatrix} l & 0 \\ 0 & 1 \end{bmatrix}$$

translations

- suppose i transform using a matrix of the form:

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 & \tilde{o} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ 1 \end{bmatrix}$$

- we see that its effect on the coordinates is

$$\begin{aligned} c_1 &\Rightarrow c_1 + t_x \\ c_2 &\Rightarrow c_2 + t_y \\ c_3 &\Rightarrow c_3 + t_z \end{aligned}$$

- this is a translation of the point
- For a translation we use the shorthand

$$T = \begin{bmatrix} i & t \\ 0 & 1 \end{bmatrix}$$

together

- any affine matrix can be factored into lin-trans form

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & d \\ 0 & 1 & 0 & h \\ 0 & 0 & 1 & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a & b & c & 0 \\ e & f & g & 0 \\ h & i & j & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- in shorthand

$$\begin{aligned} \begin{bmatrix} l & t \\ 0 & 1 \end{bmatrix} &= \begin{bmatrix} i & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} l & 0 \\ 0 & 1 \end{bmatrix} && (1) \\ A &= TL && (2) \end{aligned}$$

- if the linear part is a rotation, then we call this a rigid body matrix which implements a rigid body transform, (RBT).
  - preserves vector dot products, handedness of triples, and distances between points.

$$A = TR$$

affine transform acting on vector

- if fourth coord of $\mathbf{c}$ is zero, this just transforms a vector to a vector.
  - notice that the fourth column is irrelevant
  - a vector cannot be translated

transforming normals

- we use normals for shading
- how do they transform
- suppose i rotate forward
  - normal gets rotated forward

- suppose squash in the y direction
  - normal gets higher in the y direction (see figure)
- what is the rule?

computing normals

- tangent is vector between two nearby points
- normal is orthogonal to tangent

$$\vec{n} \cdot (\tilde{p_1} - \tilde{p_0}) = 0$$

- in coordinates

$$\begin{bmatrix} nx & ny & nz & * \end{bmatrix} \left(\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} \right) = 0$$

derive

- suppose we transform using A
- lets plug in $A^{-1}A$

$$(\begin{bmatrix} nx & ny & nz & * \end{bmatrix} \mathbf{A^{-1}})(\mathbf{A} \left(\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} - \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} \right)) = 0$$

- define $[x', y', z', 1]^t = A[x, y, z, 1]^t$ to be the coordinates of a transformed point,
- and $[nx', ny', nz', *] = [nx, ny, nz, *]A^{-1}$
  - the $*$ in rhs has no effect since A is an affine matrix.

$$\begin{bmatrix} nx' & ny' & nz' & * \end{bmatrix} \left(\begin{bmatrix} x'_1 \\ y'_1 \\ z'_1 \\ 1 \end{bmatrix} - \begin{bmatrix} x'_0 \\ y'_0 \\ z'_0 \\ 1 \end{bmatrix} \right) = 0$$

- so $[nx', ny', nz']^t$ must be the normal of the tformed geometry

more derive

- Thus, using the shorthand

$$A = \begin{bmatrix} l & t \\ 0 & 1 \end{bmatrix}$$

  we see that

$$\begin{bmatrix} nx' & ny' & nz' \end{bmatrix} = \begin{bmatrix} nx & ny & nz \end{bmatrix} l^{-1}$$

- i.e.,

$$\begin{bmatrix} nx' \\ ny' \\ nz' \end{bmatrix} = l^{-t} \begin{bmatrix} nx \\ ny \\ nz \end{bmatrix}$$

inv transpose

- so inverse transpose/ transpose inverse is the rule
- for rotations, transpose = inverse.
- for scales, transpose = nothing.
- in the code next week, we will send A and l^{-t} to the vertex shader.