linear

- we will want to represent the geometry of points in space
- we will often want to perform (rigid) transformations to these objects to position them
  - translate
  - rotate
- or move them in an animation
  - time varying tform
- position or move virtual camera
- we also may use non-rigid tforms to specify shape
  - scale an object
  - squash a sphere into an ellipsoid.

SO....

- so we must understand how to manipulate 3d coordinates and transforms
- we must pay attention to order of tforms
- we must pay attention to the role of the coordinate system w.r.t. which we perform a tform
- we will look at linear and affine transformations
- at end of the day, our code will have vertices with 3d coords and we will use 4 by 4 matrices to describe properly manipulate them
- but to figure out what to code, we need to first do some thinking/paper-pencil work.

Geometric data types

- we describe a point using a coordinate vector
  \[
  \begin{bmatrix}
  x \\
  y \\
  z
  \end{bmatrix}
  \]
- specifies position wrt an agreed upon coordinate system
  - three agreed directions
  - agreed origin
  - if we change agreed upon c.s., we must change the coordinate vector
- so a point is specified with a coordinate system and a coordinate vector

4 geometric data types

- point: \( \vec{p} \)
  - represents place
- vector: \( \vec{v} \)
  - represents motion/offset between points
- coordinate vector: \( \vec{c} \)
- coordinate system \( \vec{s} \)
  - “basis” for vectors
  - “frame” is for points
vectors vs coordinate vectors

- a vector is a geometric entity (motion/offset between points) in a real or virtual 3D world
- a coordinate vector is a set of numbers used to specify a vector given an agreed coordinate system

vector space

- a set
- has an addition operation
- has a scalar multiplication
- some other rules
  - addition is associative and commutative
  - scalar mul must distribute across vector add

\[
\alpha(\vec{v} + \vec{w}) = \alpha \vec{v} + \alpha \vec{w}
\]

example vector spaces

- \(R^3\) (triplets of numbers)
- pairs of elements of \(R^3\), where two pairs are equivalent if they share their offset.
  - in particular the “vectors” are not triplets of numbers, but bags of triplet pairs.
- this class: motion between points in our room.
  - these “vectors” should not be thought of as numerical objects.

coordinate system: basis

- a basis is a minimal set of vectors that we can use to get to all of the vectors using our ops.
  - minimal == linearly independent
- dimension is number of basis elements needed
- for us it will be 3
- so basis can be used to address all of the vectors uniquely using coordinates

\[
\vec{v} = \sum_i c_i \vec{b}_i
\]

- so a basis (a non-numerical object) can be used to represent a vector (non-numerical) with a set of coordinates (a numerical object!).

shorthand

- write this as

\[
\vec{v} = \sum_i c_i \vec{b}_i = \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}
\]

- even shorter

\[
\vec{v} = \vec{B}'c
\]

linear transformation

- a linear tform \(\mathcal{L}\) maps from \(V\) to \(V\)
• satisfies 2 rules

\[ L(\vec{v} + \vec{u}) = L(\vec{v}) + L(\vec{u}) \]
\[ L(\alpha \vec{v}) = \alpha L(\vec{v}) \]

• we will use the notation \( \vec{v} \rightarrow L(\vec{v}) \)
  
  – book uses \( \Rightarrow \)

linear tforms and matrices

• linear transformation can be exactly specified by telling us its effect on the basis vectors.

• linear transforms can be expressed with matrix multiplication

• Linearity implies

\[ \vec{v} \rightarrow L(\vec{v}) = L(\sum_i c_i \vec{b}_i) = \sum_i c_i L(\vec{b}_i) \]

• in our shorthand this is

\[
\begin{bmatrix}
  \vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
c_3
\end{bmatrix}
\rightarrow
\begin{bmatrix}
L(\vec{b}_1) & L(\vec{b}_2) & L(\vec{b}_3)
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
c_3
\end{bmatrix}
\]

• each \( L(\vec{b}_i) \) can ultimately be written as some linear combination of the original basis vectors using numbers \( M_{i,j} \)

\[
\begin{bmatrix}
L(\vec{b}_1) & L(\vec{b}_2) & L(\vec{b}_3)
\end{bmatrix}
=
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
\begin{bmatrix}
M_{1,1} & M_{1,2} & M_{1,3} \\
M_{2,1} & M_{2,2} & M_{2,3} \\
M_{3,1} & M_{3,2} & M_{3,3}
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
c_3
\end{bmatrix}
\]

so

• a linear transform operating on a vector can be expressed as

\[
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
c_3
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
\begin{bmatrix}
M_{1,1} & M_{1,2} & M_{1,3} \\
M_{2,1} & M_{2,2} & M_{2,3} \\
M_{3,1} & M_{3,2} & M_{3,3}
\end{bmatrix}
\begin{bmatrix}
c_1 \\
c_2 \\
c_3
\end{bmatrix}
\]

well defined ops

• vector to vector

\[ \vec{b}'c \rightarrow \vec{b}'Mc \]
  
  – see fig

• basis to basis,

\[
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
\rightarrow
\begin{bmatrix}
L(\vec{b}_1) & L(\vec{b}_2) & L(\vec{b}_3)
\end{bmatrix}
\]

• so this can be written as

\[ \vec{b}' \rightarrow \vec{b}'M \]
  
  – see fig
• coordinate vector to coordinate vector (this is the one we will see in code, but not until then).

\[ \mathbf{c} \rightarrow M\mathbf{c} \]

text:

**identity and inverse**

• the identity matrix \( I \) implements to “do nothing” transform

• an inverse matrix has the property \( MM^{-1} = M^{-1}M = I \)

• not every matrix has an inverse, but nice ones do, and all of our matrices are nice.

**matrices for change of basis**

• we just saw as an intermediate result an expression of the form

\[
\begin{bmatrix}
\vec{a}_1 & \vec{a}_2 & \vec{a}_3 \\
\vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
= 
\begin{bmatrix}
M_{1,1} & M_{1,2} & M_{1,3} \\
M_{2,1} & M_{2,2} & M_{2,3} \\
M_{3,1} & M_{3,2} & M_{3,3}
\end{bmatrix}
\]

• or in shorthand

\[
\vec{a}^t = B^tM
\]

\[
\vec{a}^tM^{-1} = B^t
\]

• this is not a transformation.

• we have used a matrix to express one named basis with respect to another.

• this will be useful too.

• we can also use this to have different expressions for the same vector

\[
\vec{v} = B^t\mathbf{c} = \vec{a}^tM^{-1}\mathbf{c}
\]

• ex 2.1 and 2.2

**dot**

• our vectors come equipped with a (bilinear) *dot product* operation

\[
\vec{v} \cdot \vec{w}
\]

• allows us to define the squared length (also called squared norm)

\[
\|\vec{v}\|^2 := \vec{v} \cdot \vec{v}
\]

• The dot product is related to the angle \( \theta \in [0..\pi] \) between two vectors

\[
\cos(\theta) = \frac{\vec{v} \cdot \vec{w}}{\|\vec{v}\| \|\vec{w}\|}
\]

**ortho**

• 2 vectors are *orthogonal* if \( \vec{v} \cdot \vec{w} = 0 \).

• orthonormal basis

• right handed (cyclically-ordered) basis
• dot product in orthonormal basis

\[ \vec{b}'c \cdot \vec{b}'d = (\sum_i c_i \vec{b}_i) \cdot (\sum_j d_j \vec{b}_j) = \sum_i \sum_j c_i d_j (\vec{b}_i \cdot \vec{b}_j) = \sum_i c_i d_i \]

cross

• the output is the vector

\[ \vec{v} \times \vec{w} := ||v|| \, ||w|| \sin(\theta) \, \vec{n} \]

• in a r.h. o.n. basis, the coordinates of \((\vec{b}'c) \times (\vec{b}'d)\) are

\[
\begin{bmatrix}
c_2d_3 - c_3d_2 \\
c_3d_1 - c_1d_3 \\
c_1d_2 - c_2d_1
\end{bmatrix}
\]

rotations

• transform that preserves dot product between vector pairs

• preserves right handedness between ordered vector triples

• so maps r.h.o.n. basis to another

• in 3d, every rotation fixes an axis, and rotates some angles r.h. about that axis.

comments

• rotations about different axes do not commute

• composition of two rots about two axes is a rotation about some third axis.

  – though the identity of this third axis is not obvious.

2D rotations

• rotate by \(\theta\) degrees counter clockwise about the origin

\[
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2
\end{bmatrix}
\begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
\]

• and we can rotate the basis as

\[
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2
\end{bmatrix}
\begin{bmatrix}
\cos \theta & -\sin \theta \\
\sin \theta & \cos \theta
\end{bmatrix}
\]

3d rotations

• rotate a point by \(\theta\) degrees around the z axis of the basis

\[
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\rightarrow
\begin{bmatrix}
\vec{b}_1 & \vec{b}_2 & \vec{b}_3
\end{bmatrix}
\begin{bmatrix}
c & -s & 0 \\
s & c & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z
\end{bmatrix}
\]

where $c \equiv \cos \theta$, and $s \equiv \sin \theta$.

- fixes points on $z$ axis
- for points in $z = k$ plane, it is like a 2D rotation
- basis is important ($z$ direction)

**more 3d rotations**

- around $x$ axis
  $$\begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \Rightarrow \begin{bmatrix} \vec{b}_1 & \vec{b}_2 & \vec{b}_3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & -s \\ 0 & s & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

- forward rotation around the $y$ axis
  $$\begin{bmatrix} c & 0 & s \\ 0 & 1 & 0 \\ -s & 0 & c \end{bmatrix}$$

**arbitrary rotation**

- can get any rotation by applying one $x,y,z$
- can get any rotation by applying one $x,y,x$
  - called Euler angles
  - visualize with set of gimbals
- one can specify rotation with unit vector axis $\hat{k} := [k_x, k_y, k_z]^t$ and $\theta$ using matrix
  $$\begin{bmatrix} k_x^2v + c & k_x k_y v - k_z s & k_x k_z v + k_y s \\ k_y k_x v + k_z s & k_y^2 v + c & k_y k_z v - k_x s \\ k_z k_x v - k_y s & k_z k_y v + k_x s & k_z^2 v + c \end{bmatrix}$$
  - where $v \equiv 1 - c$

**other linear transforms**

- uniform scales (common)
  $$\begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & a \end{bmatrix}$$

- non-uniform scales (used for modeling)
  $$\begin{bmatrix} a_x & 0 & 0 \\ 0 & a_y & 0 \\ 0 & 0 & a_z \end{bmatrix}$$

- shears (rare)
  $$\begin{bmatrix} 1 & b & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- ex 2.3, 2.4