Rotations

Michael Gleicher
April 2008 – from 2007 notes
Used as notes, not projected as lecture

Why talk about rotations in a Games Class?

- Didn't really get to do it in 559
- Can't let you leave without knowing about Quaternions
- Curious rift between theory and practice
  - Highly mathematical piece adopted early in Games
  - Workable solutions despite (and maybe better than) theory
- Really are useful!
  - Camera control / navigation
  - Rigid body dynamics
  - Articulated figure animation / Skinning

What is a rotation

- A transformation R^n -> R^n, f(x) that a few properties
  - It has a "zero", such that f(0) = 0
  - It preserves "distances" such that |a-b| = |f(a)-f(b)|
  - It preserves "handedness"
- From these properties, you can prove some others
  - It preserves (relative) angles
  - It is a LINEAR Transformation
  - It can be represented as an ortho-normal matrix
  - Rotations have unique inverses
  - The identity is a rotation
- Rotations are important
  - Rigid motions
  - Viewpoint control

Important facts about rotations

- Closed under composition
  - If A&B are rotations, AB is a rotation, as is BA
- Wrap around (not R)
- Non-Commutative AB != BA
- Associative (AB)C = A(BC)

A Detail

- Rotation – is a transformation - relative
- Orientation – is an absolute configuration
- Rotation -> Orientation
  - What happens when you apply a rotation to the identity
- Rotation between two orientations
  - $A^{-1}B$
- Can think of this as local coordinates of the first object

What is the problem with Rotations?

- The set of rotations is the Special Orthogonal Group SO(n)
  - Orthonormal matrices in n dimensions
- Not a convenient representation
- Want a *parameterization*
  - A way to assign "names" to elements of the set
  - Easy to specify members of the set
  - Easy to do operations of interest (in a moment)
- Fundamental theorem of topology
  - Any representation in R^n will have problems
    - it has a different "shape"
  - Singularities, Redundancies, …
  - Hairy Ball Theorem

What might you want to do with Rotations?

- Specify easily
- Represent compactly
- Make sure that you have a rotation (no errors)
- Find the inverse
- Transform points
- Interpolate 2 rotations
- Blend n rotations
- Average n rotations
- Do other linear operations
  - Filter, splines, …
- No singularities (measure distances)

Matrix is a representation for rotations

- Any rotation can be stored as a matrix (1 -> 1)
- Not every matrix is a rotation
  - Can ask about the "closest" rotation matrix to a given on
  - Projection onto a subset
- Where do the axes go
  - Clearly redundant (if know 1 axis in 2d, figure out the other)

Rotations in 2D (not so hard) (toy example)

Basic ideas in 2D

- Matrix (2 vectors)
  - must keep orthonorm
  - Total redundancy
- Angle (distance around circle)
  - Circle as set of rotations
- Point (1 vector)
  - Point on circle
- Velocity
  - Tangent vector (from angle)

Rotations in 3D (2D ideas hard to extend)

Basic ideas in 2D

- Matrix (2 vectors)
  - must keep orthonorm
  - Total redundancy
- Angle (distance around circle)
  - Circle as set of rotations
- Point (1 vector)
  - Point on circle
- Velocity
  - Tangent vector (from angle)

- In 3D
- Matrix (3 vectors)
  - Keep orthonorm is harder
  - Redundancy is less
- Angle (distance?)
  - Hypersphere (2D + 1)
- Point (1 vector – on 4-sphere)
  - Unit Quaternion
- Velocity (tangent to sphere)
  - Exponential coordinates

How do 3x3 rotation matrices do?

	3x3 Matrices	
Specify easily	NO! (ask an artist to type 9 numbers?)	No
Compact	No! (9 numbers, redundancy)	No
Ensure rotation	Hard (Graham-Schmidth Orthogonalization)	No
Compose	Easy (matrix multiply)	Yes
Inverse	Expensive (Matrix inversion)	Sortof
Transform	Easy and Fast	Yes
Interpolate	No! (really hard)	No
Blend / Average	No!	No
Linear Ops	No!	No
No Singularities	Yes, but metrics are hard to find	Sortof

Euler's Theorems

Any rotation can be represented by

- 3 rotations about fixed axes
  - Can be any (almost) any axes, local or fixed coordinates
  - Need to have a consistent convention
  - "Euler Angles"
- A single rotation about an arbitrary axis
  - Some "axis of rotation" – on which points do not move
  - Leads to a 4 number representation (axis angle)

Euler Angles

- Different conventions are used
  - XYZ – graphics
  - ZYX – animation (human figures)
  - XZX – physics
  - Roll, Pitch, Yaw (e.g. local) – flying
- Very compact
- R^3 -> rotations (so can give sliders to artists)
- Perilous
  - Meanings of later transforms depend on earlier ones (not so easy)
  - Singularities (some nearby transforms may be far away)
  - Can't actually do arithmetic on them

Scorecard

	Euler Angles	3x3 matrices
Specify easily	Sortof (false sense of security)	No
Compact	Yes (3 numbers)	No
Ensure rotation	Yes (any numbers are a rotation)	No
Compose	No	Yes
Inverse	Sortof (a trivial inverse is one of many)	Sortof
Transform	No (need to form matrices)	Yes
Interpolate	No (interpolating numbers gives weird things)	No
Blend / Average	No (false sense of security)	No
Linear Ops	No (false sense of security)	No
No Singularities	No	Sortof

Axis Angle

- Works out well –but
  - How do we do arithmetic on them?
  - Redundancy (many ways to describe vector)
- Note: scalar part (angle), vector part (axis)
- Use a Quaternion (4D complex number)
- Not quite enough…
- Use a UNIT Quaternion
  - Quaternion with unit magnitude
  - Very specific encoding

Unit Quaternions

- Encode a rotation as:
  - Cos(A/2), V sin(A/2)
  - V is the unit vector, A is the angle
- Note the factor of two – creates a redundancy
  - Antipodal equivalence
  - Q and –Q are the same quaternion – need to be careful of this
- We have embedded SO(3) into S(3)!
- Some easy operations
  - Invert by negating vector
  - Multiplication is complex number multiplication
  - Transform a 3D point by qPq^{-1}
  - Quaternions compose by multiplication

Interpolation

- Goal: "nice" paths between orientations
- Great circle routes
  - Points follow geodesics on spheres (circles)
  - "Smoothest" and "shortest" possible paths
- Constant velocity (magnitude / speed) along route
- SLERP – spherical linear interpolation
- Easy if have a single axis
  - Interpolate the angle linearly
- So put into local coordinates of the first, and interpolate
- Kindof expensive

Normalized LERP

- SLERP is great, but expensive
- Notice: if you scale a vector, it's the same direction
  - aV (linearly interpolate a) – the "axis of rotation" is still V
- Linearly interpolate the quaternions
- Renormalize (to make unit quaternions)
- Traces the same great circle route
  - But not at the same velocity
- But is VERY cheap and easy to compute

More than 2 Quaternions

- SLERP does not associate
  - (A->B)->C is not A->(B->C)
- How to average/blend N Quaternions?
- Mathematically right answers are hard
  - Need to understand logarithms
- Normalized LERP works "well enough"
  - Not constant velocity (but this is a small effect)
  - Does associate
- Use exponential coordinates otherwise

Scorecard

	Quaternions	Euler Angles	3x3 matrices
Specify easily	No (but use Euler UI)	Sortof	No
Compact	Yes (4 numbers)	Yes	No
Ensure rotation	Yes (renormalize is easy)	Yes	No
Compose	Yes (very fast quaternion multiply)	No	Yes
Inverse	Yes (very fast)	Sortof	Sortof
Transform	Yes (very fast, about the same at mmult)	No	Yes
Interpolate	Yes (SLERP or NLERP)	No	No
Blend / Average	Yes (NLERP or spherical averages)	No	No
Linear Ops	Yes (NLERP or log maps)	No	No
No Singularities	Yes (easy to avoid antipode problems)	No	Sortof

The Verdict?

- Use Euler Angles ! (?)
  - If really a 1 or 2 d.o.f. problem
  - Generality of 3D rotations aren't too big of a deal
- Use Quaternions if really doing 3D
- Convert Quaternions to Matrices for Hardware

Matrix Exponentials
A curious diversion

- Transformations compose by multiplication, not by addition
- What's "half" of a transformation?
  - M = H+H (no!)
  - M = H H (multiply) H = sqrt(M)
- How to get from A to B in S steps?
  - $(A^{-1}B)$ is the transformation from A to B
  - $A (A^{-1}B)^t$ - t goes from 0->t
  - Take logs : $\ln A + t \ln A^{-1} + t \ln B \rightarrow (1-t) \ln A + t \ln B$
  - Linearly interpolate the matrix logarithms
  - Except that matrix multiplication doesn't commute