3D Rotations and Complex Representations

Computer Graphics
CMU 15-462/15-662
Rotations in 3D

- What is a rotation, intuitively?

- *How do you know a rotation when you see it?*
  - length/distance is preserved (no stretching/shearing)
  - orientation is preserved (e.g., text remains readable)
3D Rotations—Degrees of Freedom

- How many numbers do we need to specify a rotation in 3D?
- For instance, we could use rotations around X, Y, Z. But do we need all three?
- Well, to rotate Pittsburgh to another city (say, São Paulo), we have to specify two numbers: latitude & longitude:
- Do we really need both latitude and longitude? Or will one suffice?
- Is that the only rotation from Pittsburgh to São Paulo? (How many more numbers do we need?)

NO: We can keep São Paulo fixed as we rotate the globe.

Hence, we MUST have three degrees of freedom.
Commutativity of Rotations—2D

- In 2D, order of rotations doesn’t matter:

rotate by 40°

rotate by 20°

rotate by 20°

rotate by 40°

Same result! (“2D rotations commute”)
Commutativity of Rotations—3D

- What about in 3D?
- IN-CLASS ACTIVITY:
  - Rotate 90° around Y, then 90° around Z, then 90° around X
  - Rotate 90° around Z, then 90° around Y, then 90° around X
  - (Was there any difference?)

CONCLUSION: bad things can happen if we’re not careful about the order in which we apply rotations!
First things first: how do we get a rotation matrix in 2D? (Don’t just regurgitate the formula!)

Suppose I have a function $S(\theta)$ that for a given angle $\theta$ gives me the point $(x,y)$ around a circle (CCW).

- Right now, I do not care how this function is expressed!*

What’s $e_1$ rotated by $\theta$? $\tilde{e}_1 = S(\theta)$

What’s $e_2$ rotated by $\theta$? $\tilde{e}_2 = S(\theta + \pi/2)$

How about $u := ae_1 + be_2$?

$u := aS(\theta) + bS(\theta + \pi/2)$

What then must the matrix look like?

$$\begin{bmatrix} S(\theta) & S(\theta + \pi/2) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \cos(\theta + \pi/2) \\ \sin(\theta) & \sin(\theta + \pi/2) \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

*I.e., I don’t yet care about sines and cosines and so forth.*
Representing Rotations in 3D—Euler Angles

- How do we express rotations in 3D?
- One idea: we know how to do 2D rotations.
- Why not simply apply rotations around the three axes? (X,Y,Z)
- Scheme is called *Euler angles*
- PROBLEM: “Gimbal Lock” [DEMO]
Gimbal Lock

- When using Euler angles $\theta_x$, $\theta_y$, $\theta_z$, may reach a configuration where there is no way to rotate around one of the three axes!

- Recall rotation matrices around three axes:

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \theta_x & -\sin \theta_x \\ 0 & \sin \theta_x & \cos \theta_x \end{bmatrix} \quad R_y = \begin{bmatrix} \cos \theta_y & 0 & \sin \theta_y \\ 0 & 1 & 0 \\ -\sin \theta_y & 0 & \cos \theta_y \end{bmatrix} \quad R_z = \begin{bmatrix} \cos \theta_z & -\sin \theta_z & 0 \\ \sin \theta_z & \cos \theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Product of these matrices represents rotation by Euler angles:

$$R_x R_y R_z = \begin{bmatrix} \cos \theta_y \cos \theta_z & -\cos \theta_y \sin \theta_z & \sin \theta_y \\ \cos \theta_z \sin \theta_x \sin \theta_y + \cos \theta_x \sin \theta_z & \cos \theta_x \cos \theta_z - \sin \theta_x \sin \theta_y \sin \theta_z & -\cos \theta_y \sin \theta_x \\ -\cos \theta_x \cos \theta_z \sin \theta_y + \sin \theta_x \sin \theta_z & \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_y \sin \theta_z & \cos \theta_x \cos \theta_y \end{bmatrix}$$

- Consider special case $\theta_y = \pi/2$ (so, $\cos \theta_y = 0$, $\sin \theta_y = 1$):

$$\Rightarrow \begin{bmatrix} 0 \\ \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_z & \cos \theta_x \cos \theta_z - \sin \theta_x \sin \theta_z & 0 \\ -\cos \theta_x \cos \theta_z + \sin \theta_x \sin \theta_z & \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_z & 0 \end{bmatrix}$$
Gimbal Lock, continued

- Simplifying matrix from previous slide, we get

\[
\begin{bmatrix}
0 & 0 & 1 \\
\sin(\theta_x + \theta_z) & \cos(\theta_x + \theta_z) & 0 \\
-\cos(\theta_x + \theta_z) & \sin(\theta_x + \theta_z) & 0
\end{bmatrix}
\]

no matter how we adjust $\theta_x, \theta_z$, can only rotate in one plane!

Q: What does this matrix do?

- We are now “locked” into a single axis of rotation
- Not a great design for airplane instruments! (but why is it okay for controls?)
Rotation from Axis/Angle

Alternatively, there is a general expression for a matrix that performs a rotation around a given axis $u$ by a given angle $\theta$:

$$\begin{bmatrix}
\cos \theta + u_x^2 (1 - \cos \theta) & u_x u_y (1 - \cos \theta) - u_z \sin \theta & u_x u_z (1 - \cos \theta) + u_y \sin \theta \\
 u_y u_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_y u_z (1 - \cos \theta) - u_x \sin \theta \\
 u_z u_x (1 - \cos \theta) - u_y \sin \theta & u_z u_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2 (1 - \cos \theta)
\end{bmatrix}$$

Just memorize this matrix! :-)

There is an easier way:

rotate(((u, \theta), s)) \equiv ((s \cdot u)u + \cos(\theta)(s - (s \cdot u)u) + \sin(\theta)(u \times s)

part of $s$ parallel to $u$

part of $s$ perpendicular to $u$

what's this?
Complex Analysis—Motivation

- Natural way to encode geometric transformations in 2D
- Simplifies code / notation / debugging / thinking
- *Moderate* reduction in computational cost/bandwidth/storage
- Fluency with complex analysis can lead into deeper/novel solutions to problems...

Truly: no good reason to use 2D vectors instead of complex numbers...
DON’T: Think of these numbers as “complex.”

DO: Imagine we’re simply defining additional operations (like dot and cross).
Imaginary Unit

More importantly: obscures geometric meaning.
Imaginary unit is just a quarter-turn in the counter-clockwise direction.
Complex Numbers

- Complex numbers are then just 2-vectors
- Instead of $e_1, e_1$, use “1” and “ι” to denote the two bases
- Otherwise, behaves exactly like a real 2-dimensional space

...except that we’re also going to get a very useful new notion of the product between two vectors.
Complex Arithmetic

- Same operations as before, plus one more:
  - Vector addition
  - Scalar multiplication
  - Complex multiplication

- Complex multiplication:
  - Angles add
  - Magnitudes multiply

"POLAR FORM"*:

$z_1 := (r_1, \theta_1)$

$z_2 := (r_2, \theta_2)$

$z_1 z_2 = (r_1 r_2, \theta_1 + \theta_2)$

*Not quite how it really works, but basic idea is right.

Have to be more careful here!
Complex Product—Rectangular Form

- Complex product in “rectangular” coordinates \((1, i)\):

\[
\begin{align*}
  z_1 &= (a + bi) \\
  z_2 &= (c + di) \\
  z_1 z_2 &= ac + adi + bci + bd i^2 = (ac - bd) + (ad + bc)i.
\end{align*}
\]

- We used a lot of “rules” here. Can you justify them geometrically?

- Does this product agree with our geometric description (last slide)?
Complex Product—Polar Form

- Perhaps most beautiful identity in math:
  \[ e^{i\pi} + 1 = 0 \]

- Specialization of Euler’s formula:
  \[ e^{i\theta} = \cos(\theta) + i \sin(\theta) \]

- Can use to “implement” complex product:
  \[ z_1 = ae^{i\theta}, \quad z_2 = be^{i\phi} \]
  \[ z_1 z_2 = abe^{i(\theta+\phi)} \]
  (as with real exponentiation, exponents add)

Q: How does this operation differ from our earlier, “fake” polar multiplication?
2D Rotations: Matrices vs. Complex

Suppose we want to rotate a vector \( u \) by an angle \( \theta \), then by an angle \( \phi \).

**REAL / RECTANGULAR**

\[
\begin{align*}
\mathbf{u} &= (x, y) \\
\mathbf{A} &= \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix} \cos \phi & -\sin \phi \\ \sin \phi & \cos \phi \end{bmatrix}
\end{align*}
\]

\[
\mathbf{Au} = \begin{bmatrix} x \cos \theta - y \sin \theta \\ x \sin \theta + y \cos \theta \end{bmatrix}
\]

\[
\mathbf{BAu} = \begin{bmatrix} (x \cos \theta - y \sin \theta) \cos \phi - (x \sin \theta + y \cos \theta) \sin \phi \\ (x \cos \theta - y \sin \theta) \sin \phi + (x \sin \theta + y \cos \theta) \cos \phi \end{bmatrix}
\]

\[= \ldots \text{some trigonometry} \ldots =\]

\[
\mathbf{BAu} = \begin{bmatrix} x \cos(\theta + \phi) - y \sin(\theta + \phi) \\ x \sin(\theta + \phi) + y \cos(\theta + \phi) \end{bmatrix}
\]

**COMPLEX / POLAR**

\[
\begin{align*}
\mathbf{u} &= re^{i\alpha} \\
a &= e^{i\theta} \\
b &= e^{i\phi}
\end{align*}
\]

\[
abu = re^{i(\alpha + \theta + \phi)}.
\]
Pervasive theme in graphics:

Sure, there are often many “equivalent” representations.

...But why not choose the one that makes life easiest*?

*Or most efficient, or most accurate...
Quaternions

- TLDR: Kind of like complex numbers but for 3D rotations
- Weird situation: can’t do 3D rotations w/ only 3 components!

William Rowan Hamilton
(1805-1865)
Quaternions in Coordinates

- Hamilton’s insight: in order to do 3D rotations in a way that mimics complex numbers for 2D, actually need FOUR coords.
- One real, three imaginary:

  $\mathbb{H} := \text{span}\{1, i, j, k\}$

  $q = a + bi + cj + dk \in \mathbb{H}$

  "H" is for Hamilton!

- Quaternion product determined by

  $i^2 = j^2 = k^2 = ijk = -1$

  together w/ “natural” rules (distributivity, associativity, etc.)

- **WARNING**: product no longer commutes!

  For $q, p \in \mathbb{H}$, $qp \neq pq$

  (Why might it make sense that it doesn’t commute?)
Quatnetion Product in Components

- Given two quaternions
  
  \[ q = a_1 + b_1i + c_1j + d_1k \]
  
  \[ p = a_2 + b_2i + c_2j + d_2k \]

- Can express their product as

  \[
  qp = a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2
  + (a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)i
  + (a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)j
  + (a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)k
  \]

  ...fortunately there is a (much) nicer expression.
Quaternions—Scalar + Vector Form

- If we have four components, how do we talk about pts in 3D?
- Natural idea: we have three imaginary parts—why not use these to encode 3D vectors?
  \[(x, y, z) \mapsto 0 + xi + yj + zk\]
- Alternatively, can think of a quaternion as a pair
  \[(\text{scalar, vector}) \in \mathbb{H} \quad \cap \quad \mathbb{R} \quad \cap \quad \mathbb{R}^3\]
- Quaternion product then has simple(r) form:
  \[(a, u)(b, v) = (ab - u \cdot v, av + bu + u \times v)\]
- For vectors in R3, gets even simpler:
  \[uv = u \times v - u \cdot v\]
3D Transformations via Quaternions

- Main use for quaternions in graphics? *Rotations.*
- Consider vector $x$ ("pure imaginary") and *unit* quaternion $q$:

$$x \in \text{Im}(\mathbb{H})$$
$$q \in \mathbb{H}, \quad |q|^2 = 1$$

always expresses some rotation
Rotation from Axis/Angle, Revisited

- Given axis $u$, angle $\theta$, quaternion $q$ representing rotation is

$$q = \cos(\theta/2) + \sin(\theta/2)u$$

- Much easier to remember (and manipulate) than matrix!

$$\begin{bmatrix}
\cos \theta + u_x^2 (1 - \cos \theta) & u_xu_y (1 - \cos \theta) - u_z \sin \theta & u_xu_z (1 - \cos \theta) + u_y \sin \theta \\
u_yu_x (1 - \cos \theta) + u_z \sin \theta & \cos \theta + u_y^2 (1 - \cos \theta) & u_yu_z (1 - \cos \theta) - u_x \sin \theta \\
u_zu_x (1 - \cos \theta) - u_y \sin \theta & u_zu_y (1 - \cos \theta) + u_x \sin \theta & \cos \theta + u_z^2 (1 - \cos \theta)
\end{bmatrix}$$
Interpolating Rotations

- Suppose we want to smoothly interpolate between two rotations (e.g., orientations of an airplane)
- Interpolating Euler angles can yield strange-looking paths, non-uniform rotation speed, ...
- Simple solution* w/ quaternions: “SLERP” (spherical linear interpolation):

\[
\text{Slerp}(q_0, q_1, t) = q_0(q_0^{-1}q_1)^t, \quad t \in [0, 1]
\]

*Shoemake 1985, “Animating Rotation with Quaternion Curves”
Where else are (hyper-)complex numbers useful in computer graphics?
Generating Coordinates for Texture Maps

Complex numbers are natural language for angle-preserving ("conformal") maps

Preserving angles in texture well-tuned to human perception...
Useless-But-Beautiful Example: Fractals

- Defined in terms of iteration on (hyper)complex numbers:

(Will see exactly how this works later in class.)
Next time: Perspective & Texture Mapping