3D Transformations and Complex Representations
Quiz 4: Trees and Transformations

Student solutions (beautiful!):

![Tree 1](image1)
![Tree 2](image2)
![Tree 3](image3)

![Tree 4](image4)
![Tree 5](image5)
![Tree 6](image6)
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)
  - lines get mapped to lines (linear)
  - 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°

Why not?
**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!
Representing Rotations—2D

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 $u := a e_1 + b e_2$?

$$u := a S(\theta) + b S(\theta + \pi/2)$$

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., during most of life, NOTHING is gained by thinking in terms of $\sin(\theta)$ and $\cos(\theta)$.}
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”
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! :-)

...we’ll see a different way, later on.
Complex Analysis—Motivation

- Natural way to encode geometric transformations in 2D, 3D
- Simplifies notation / thinking / debugging
- *Moderate* reduction in computational cost/bandwidth/storage
- Fluency with complex analysis can lead into deeper/novel solutions to problems...
**DON’T**: Think of these numbers as “complex.”

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

*A bit of an oversimplification, but go with it for now!*
Imaginary Unit

More importantly: obscures geometric meaning.
Symbol $\imath$ denotes quarter-turn in the counter-clockwise direction.

*Use $\imath$ instead of $i$ to avoid confusion with indices $i$. (\imath in LaTeX)
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 going to define a useful new notion of the product between two vectors.
Complex Arithmetic

- Same operations as before, plus one more:

- 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 really now it works, but useful geometric intuition.
Complex Product—Rectangular Form

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

\[
\begin{align*}
z_1 &= (a + bi) \\
z_2 &= (c + di) \\
z_1z_2 &= ac + adi + bci + bdi^2 = (ac - bd) + (ad + bc)i.
\end{align*}
\]

- 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?

Leonhard Euler (1707–1783)
- Most prolific mathematician of all time
- Opera Omnia—1 vol./yr. starting 1911
- Still going! Now ~75 vols., 25k pages
- 228 papers posthumously
- Many later works while blind
- (Work was also good...)

[source: William Dunham]
### 2D Rotations: Matrices vs. Complex

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

<table>
<thead>
<tr>
<th>REAL / RECTANGULAR</th>
<th>COMPLEX / POLAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u = (x, y)$</td>
<td>$u = r e^{i\alpha}$</td>
</tr>
<tr>
<td>$A = \begin{bmatrix} \cos \theta &amp; -\sin \theta \ \sin \theta &amp; \cos \theta \end{bmatrix}$</td>
<td>$a = e^{i\theta}$</td>
</tr>
<tr>
<td>$B = \begin{bmatrix} \cos \phi &amp; -\sin \phi \ \sin \phi &amp; \cos \phi \end{bmatrix}$</td>
<td>$b = e^{i\phi}$</td>
</tr>
<tr>
<td>$Au = \begin{bmatrix} x \cos \theta - y \sin \theta \ x \sin \theta + y \cos \theta \end{bmatrix}$</td>
<td>$abu = re^{i(\alpha+\theta+\phi)}$.</td>
</tr>
<tr>
<td>$BAu = \begin{bmatrix} (x \cos \theta - y \sin \theta) \cos \phi - (x \sin \theta + y \cos \theta) \sin \phi \ (x \cos \theta - y \sin \theta) \sin \phi + (x \sin \theta + y \cos \theta) \cos \phi \end{bmatrix}$</td>
<td></td>
</tr>
<tr>
<td>$\hspace{2.5cm} = \cdots \text{somne trigonometry} \cdots =$</td>
<td></td>
</tr>
<tr>
<td>$BAu = \begin{bmatrix} x \cos(\theta + \phi) - y \sin(\theta + \phi) \ x \sin(\theta + \phi) + y \cos(\theta + \phi) \end{bmatrix}$.</td>
<td></td>
</tr>
</tbody>
</table>

(...and simplification is not always this obvious.)
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)  

(Not Hamilton)
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:

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

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

- Quaternion product determined by

\[ i^2 = j^2 = k^2 = ijk = -1 \]

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

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

\[ \text{For } q, p \in \mathbb{H}, \quad qp \neq pq \]

(Will understand this a *lot* better when we study transformations.)
Quaternion 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?

\[(x, y, z) \mapsto 0 + xi + yj + zj\]

- 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:

\[(a, u)(b, v) = (ab - u \cdot v, au + bv + u \times v)\]

- 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 $q$:

\[
x \in \text{Im}(\mathbb{H})
\]
\[
q \in \mathbb{H}, \quad |q|^2 = 1
\]
Rotation from Axis/Angle, Revisited

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

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

- Slightly easier to remember (and manipulate) than matrix:

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

- Don’t have time to cover everything, but...
- Quaternions provide some very nice utility/perspective when it comes to rotations:
  - Spherical linear interpolation ("slerp")
  - Hopf fibration / "belt trick"
  - ...

![Diagram of spherical linear interpolation and Hopf fibration](image)
Where else are (hyper-)complex numbers useful in computer graphics?
Complex \#s: Language of Conformal Maps
Useless-But-Beautiful Example: Fractals

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

(Will see exactly how this works later in class.)