Wrapping up Spatial Transformations
picking up from last time…

Computer Graphics
CMU 15-462/15-662
Mini HW2 due Monday before class
Q: How can we perform perspective projection* using homogeneous coordinates?

Remember from our pinhole camera model that the basic idea was to “divide by $z$”

So, we can build a matrix that “copies” the $z$ coordinate into the homogeneous coordinate.

Division by the homogeneous coordinate now gives us perspective projection onto the plane $z = 1$

\[(x, y, z) \mapsto (x/z, y/z)\]

\[
\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 1 & 0
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
z \\
1
\end{bmatrix}
= 
\begin{bmatrix}
x \\
y \\
z \\
z
\end{bmatrix}
\]

\[
\Rightarrow 
\begin{bmatrix}
x/z \\
y/z \\
1
\end{bmatrix}
\]

*Assuming a pinhole camera at (0,0,0) looking down the z-axis
Screen Transformation

- One last transformation is needed in the rasterization pipeline: transform from viewing plane to pixel coordinates.

- E.g., suppose we want to draw all points that fall inside the square $[-1,1] \times [-1,1]$ on the $z = 1$ plane, into a $W \times H$ pixel image.

```
(0,0)  (0,0)
\( (1,1) \)
\( (-1,-1) \)
```

"normalized device coordinates"

```
\( (0,0) \)
\( (W,H) \)
```

image space

Q: What transformation(s) would you apply? (Careful: $y$ is now down!)
Scene Graph

- For complex scenes (e.g., more than just a cube!) scene graph can help organize transformations

- Motivation: suppose we want to build a “cube creature” by transforming copies of the unit cube

- Difficult to specify each transformation directly

- Instead, build up transformations of “lower” parts from transformations of “upper” parts
  - E.g., first position the body
  - Then transform upper arm relative to the body
  - Then transform lower arm relative to upper arm
  - ...
Scene Graph (continued)

- Scene graph stores relative transformations in directed graph
- Each edge (+root) stores a linear transformation (e.g., a 4x4 matrix)
- Composition of transformations gets applied to nodes

E.g., $A_1A_0$ gets applied to left upper leg; $A_2A_1A_0$ to left lower leg
- Keep transformations on a stack to reduce redundant multiplication
Scene Graph—Example

Often used to build up complex “rig”:

In general, scene graph also includes other models, lights, cameras, …
Instancing

- What if we want many copies of the same object in a scene?

- Rather than have many copies of the geometry, scene graph, etc., can just put a “pointer” node in our scene graph.

- Like any other node, can specify a different transformation on each incoming edge.

Deussen et al, “Realistic modeling and rendering of plant ecosystems”
Instancing—Example
Order matters when composing transformations!

- First, scale by 1/2, then translate by (3,1).
- Then, translate by (3,1), then scale by 1/2.
How would you perform these transformations?

Remember: always more than one way to do it!
Common task: rotate about a point $x$

Step 1: translate by $-x$

Step 2: rotate

Step 4: translate by $x$

Q: What happens if we just rotate without translating first?
Drawing a Cube Creature

- Let’s put this all together: starting with our 3D cube, we want to make a 2D, perspective-correct image of a “cube creature”

- First we use our scene graph to apply 3D transformations to several copies of our cube

- Then we apply a 3D transformation to position our camera

- Then a perspective projection

- Finally we convert to image coordinates (and rasterize)

- …Easy, right? :-(
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)
  - origin is preserved (otherwise it’s a rotation + translation)
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?
- Try it at home—grab a water bottle!
  - 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 $\mathbf{e}_1$ rotated by $\theta$? $\tilde{\mathbf{e}}_1 = S(\theta)$

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

How about $\mathbf{u} := a\mathbf{e}_1 + b\mathbf{e}_2$?

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

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
- “Gimbal Lock”
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, $\cos \theta_y = 0, \sin \theta_y = 1$):

  
  $$\Longrightarrow \begin{bmatrix}
  0 & 0 & 1 \\
  \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_z & \cos \theta_x \cos \theta_z - \sin \theta_x \sin \theta_z & 0 \\
  -\cos \theta_x \cos \theta_z + \sin \theta_x \sin \theta_z & \cos \theta_z \sin \theta_x + \cos \theta_x \sin \theta_z & 0
  \end{bmatrix}$$
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 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! :-)

...we’ll see a much easier way, later on.
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

\[ \imath \triangleq \sqrt{-1} \]

\[ \text{nonsense!} \]

More importantly: obscures geometric meaning.
Imaginary Unit—Geometric Description

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

REAL

\[ \mathbb{R}^2 \]

COMPLEX

\[ \mathbb{C} \]

\[ (a, b) \]

\[ a + bι \]

...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:

  - **scalar multiplication**
  - **vector addition**
  - **complex multiplication**

Complex multiplication:
- angles add
- magnitudes multiply

“POLAR FORM”*:

\[
\begin{align*}
  z_1 &:= (r_1, \theta_1) \\
  z_2 &:= (r_2, \theta_2) \\
  z_1 z_2 &= (r_1 r_2, \theta_1 + \theta_2)
\end{align*}
\]

*Not quite how it really works, but basic idea is right.
Complex Product—Rectangular Form

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

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

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?
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>$u = re^{i\alpha}$</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>$= \cdots$ some 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>
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:
  \[ \mathbb{H} := \text{span}(\{1, i, j, k\}) \]
  \[ q = a + bi + cj + dk \in \mathbb{H} \]

- Quaternion product determined by
  \[ i^2 = j^2 = k^2 = ijk = -1 \]
  together w/ “natural” rules (distributivity, associativity, etc.)

- **WARNING**: product no longer commutes!
  \[ \text{For } q, p \in \mathbb{H}, \quad qp \neq pq \]
  (Why might it make sense that it doesn’t commute?)

“H” is for Hamilton!
Quaternion Product in Components

- Given two quaternions
  
  \[ q = a_1 + b_1 i + c_1 j + d_1 k \]
  
  \[ p = a_2 + b_2 i + c_2 j + d_2 k \]

- Can express their product as

  \[ qp = a_1 a_2 - b_1 b_2 - c_1 c_2 - d_1 d_2 \]

  \[ + (a_1 b_2 + b_1 a_2 + c_1 d_2 - d_1 c_2) i \]

  \[ + (a_1 c_2 - b_1 d_2 + c_1 a_2 + d_1 b_2) j \]

  \[ + (a_1 d_2 + b_1 c_2 - c_1 b_2 + d_1 a_2) k \]

  ...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

\[(\text{scalar, vector}) \in \mathbb{H} \cap \mathbb{R} \cap \mathbb{R}^3\]

- 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 $q$:

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

![Diagram showing rotation](image)
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!

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

Note: the quaternion conjugate is the same as the inverse for a unit quaternion. Can you create an inverse?
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.)
Not Covered: Lie algebras/Lie Groups

- Another super nice/useful perspective on rotations is via “Lie groups” and “Lie algebras”
- More than we have time to cover!
- Many benefits similar to quaternions (easy axis/angle representation, no gimbal lock, …)
- Nice for encoding angles bigger than $2\pi$
- Also very useful for taking averages of rotations
- (Very) short story:
  - exponential map takes you from axis/angle to rotation matrix
  - logarithmic map takes you from rotation matrix to axis/angle
Rotations and Complex Representations—Summary

- Rotations are surprisingly complicated in 3D!
- Today, looked at how complex representations help understand/work with rotations in 3D (& 2D)
- In general, many possible representations:
  - Euler angles
  - axis-angle
  - quaternions
  - Lie group/algebra (not covered)
  - geometric algebra (not covered)
- There’s no “right” or “best” way—the more you know, the more you’ll be able to do!
Next time: Perspective & Texture Mapping