Lab 6

CS 418: Interactive Computer Graphics

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Eric Shaffer

Fog

og is an example of participating media

hese are typically some sort of particulate

catters and otherwise changes light

here are a number of sophisticated ways to render such phenomena
.we’ll use a simple one

or something complex, check out

ttp://www.gdcvault.com/play/1023519/Fast-Flexible-Physically-Based-Volumetric

Distance Fog

Just mix a fragment color with a fog color

The farther the fragment from the viewer, more fog color

...and less original color

Fog in WebGL and GLSL

- We implement fog in the fragment shader
- Compute the distance from fragment to camera

```
float fogCoord = (gl_FragCoord.z/gl_FragCoord.w);
```

- And define a fog color
  - White is a good choice…especially with a white background

```
vec4 fogColor = vec4(1.0, 1.0, 1.0, 1.0);
```

Mixing in Fog

Use linear interpolation to mix fog with the fragment color

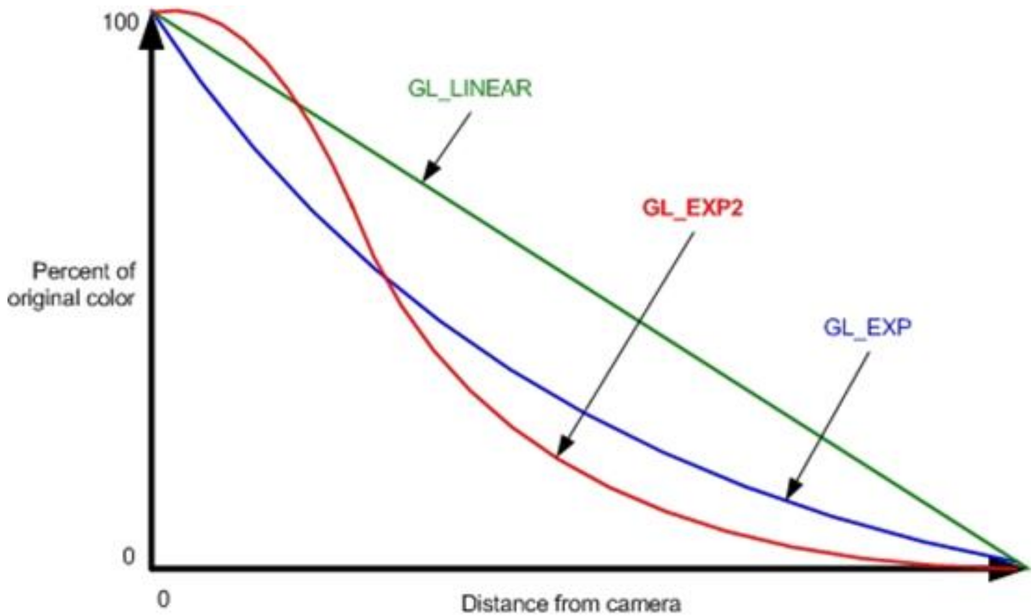
```
gl_FragColor = mix(fogColor, fragColor, fogFactor );
```

- `fragColor` is the color computed by your shading equation
  - Probably Blinn-Phong
- The `mix` function is built-in GLSL function that performs lerp
- But what is `fogFactor`?

Fog Factor

```
const float LOG2 = 1.442695;
float fogDensity = 0.0005
float fogFactor = exp2( -fogDensity * fogDensity * fogCoord * fogCoord * LOG2 );
fogFactor = clamp(fogFactor, 0.0, 1.0);
```

- Fog factor determines how much of the color is fog
  - A value of 1 means all fog in this case…a value of 0 means no fog
- You can see the curve below…we're computing GL_EXP2

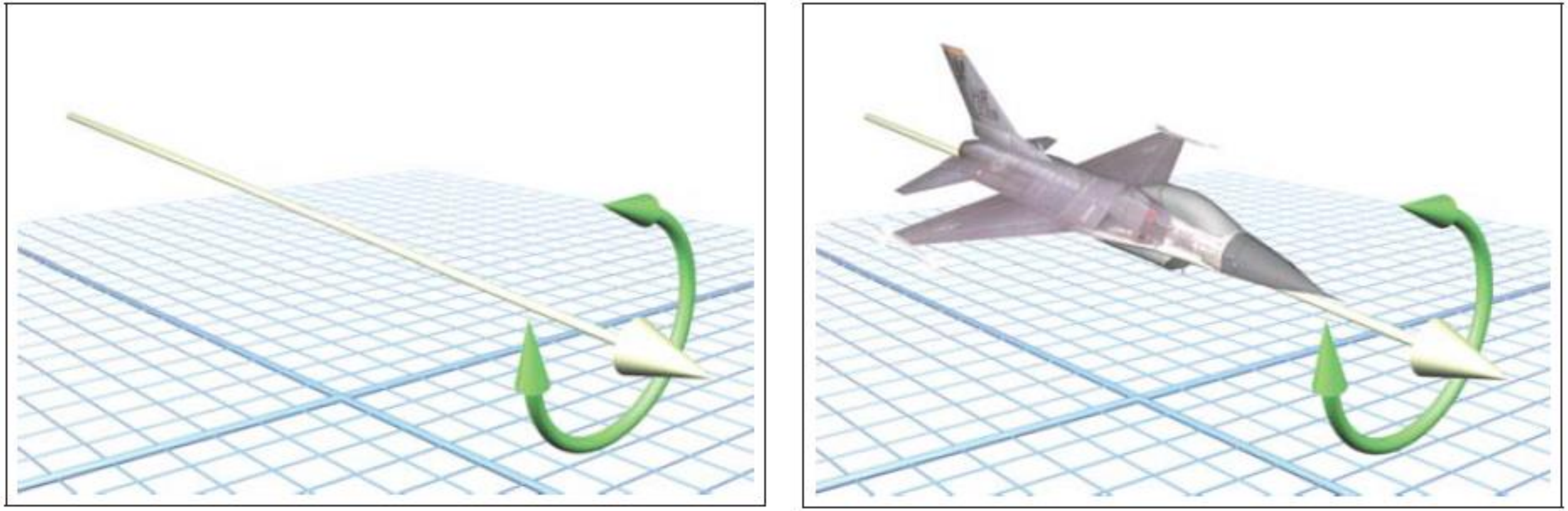


Fog Factor Debugging

- In case you have trouble,
  try just implementing the linear curve for the fogFactor…see if that works.

- Can also render your fogCoord values
  - Compute z=fogCoord/farClipDistance;
  - Set gl_FragColor to (z,z,z, 1.0)
  - See if the image makes sense

Representing Orientations



- Rotating a vector around itself does not change it
- Rotating an object around its principal direction changes its orientation
- Orientations require more information to represent than a vector

Representing Orientations

No simple means of representing a 3D orientation

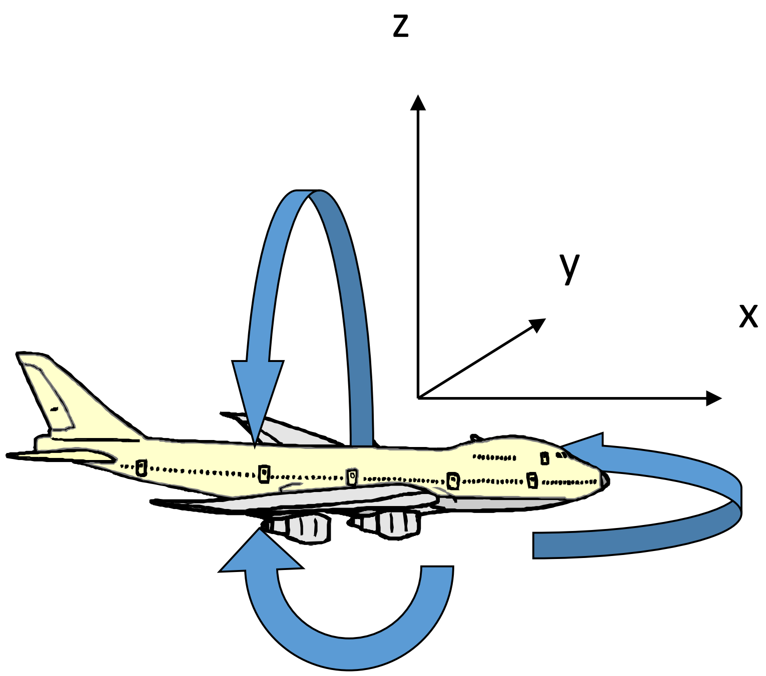
- Unlike position and Cartesian coordinates

There are several popular options:

- Euler angles
- Rotation vectors (axis/angle)
- 3x3 matrices
- Quaternions

Euler Angles

- Airplane orientation
  - Roll
    - rotation about x
    - Turn wheel
  - Pitch
    - rotation about y
    - Push/pull wheel
  - Yaw
    - rotation about z
    - Rudder (foot pedals)
- Airplane orientation
  - Rx(roll) Ry(pitch) Rz(yaw)



Quaternions

- Developed by Sir William Rowan Hamilton [1843]
- Quaternions are 4-D numbers
- With one real axis
- And three imaginary axes: **i**,**j**,**k**
- Imaginary-style multiplication rules

$$\mathbf{q} = q_0 + iq_1 + jq_2 + kq_3$$

Hamilton Math Inst.,
Trinity College

Quaternions

- Introduced to Computer Graphics by Shoemake [1985]
- Given an angle and axis, easy to convert to and from quaternion
  - Euler angle conversion to and from arbitrary axis and angle difficult
- Quaternions allow stable and constant interpolation of orientations
  - Cannot be done easily with Euler angles

Unit Quaternions

- For convenience, we will use only unit length quaternions

$$|\mathbf{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} = 1$$

- These correspond to the set of 4D vectors
- They form the ‘surface’ of a 4D hypersphere of radius 1

Quaternions as Rotations

- A quaternion can represent a rotation by angle θ around a unit vector **a**:

$$\mathbf{q} = \begin{bmatrix} \cos\frac{\theta}{2} & a_x \sin\frac{\theta}{2} & a_y \sin\frac{\theta}{2} & a_z \sin\frac{\theta}{2} \end{bmatrix}$$

$$or$$

$$\mathbf{q} = \left\langle \cos\frac{\theta}{2}, \mathbf{a}\sin\frac{\theta}{2} \right\rangle$$

- If **a** is unit length, then **q** will be also

Rotation using Quaternions

- Let $q = \cos(\theta/2) + \sin(\theta/2)\,\mathbf{u}$ be a unit quaternion: $|q| = |\mathbf{u}| = 1$
- Let point $\mathbf{p} = (x,y,z) = x\,\mathbf{i} + y\,\mathbf{j} + z\,\mathbf{k}$
- Then the product $q\,\mathbf{p}\,q^{-1}$ rotates the point $\mathbf{p}$ about axis $\mathbf{u}$ by angle θ
- Inverse of a unit quaternion is its ***conjugate***

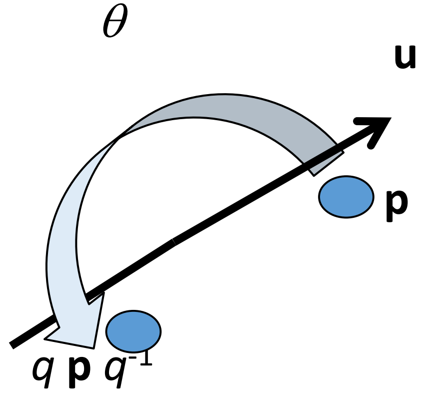
  (negate the imaginary part)

  $$\begin{aligned} q^{-1} &= (\cos(\theta/2) + \sin(\theta/2)\,\mathbf{u})^{-1} \\ &= \cos(-\theta/2) + \sin(-\theta/2)\,\mathbf{u} \\ &= \cos(\theta/2) - \sin(\theta/2)\,\mathbf{u} \end{aligned}$$

- Composition of rotations $q_{12} = q_1\,q_2 \neq q_2\,q_1$

We haven't talked about how to multiply quaternions yet, but don't worry about that for now…

$$q = \cos\frac{\theta}{2} + \mathbf{u}\sin\frac{\theta}{2}$$



Quaternion to Matrix

Again, why do we want to be able to do this?

- To convert a quaternion to a rotation matrix:

$$\begin{bmatrix} 1-2q_2^2-2q_3^2 & 2q_1q_2+2q_0q_3 & 2q_1q_3-2q_0q_2 \\ 2q_1q_2-2q_0q_3 & 1-2q_1^2-2q_3^2 & 2q_2q_3+2q_0q_1 \\ 2q_1q_3+2q_0q_2 & 2q_2q_3-2q_0q_1 & 1-2q_1^2-2q_2^2 \end{bmatrix}$$

Quaternion Multiplication

- We can perform multiplication on quaternions
  - we expand them into their complex number form

$$\mathbf{q} = q_0 + iq_1 + jq_2 + kq_3$$

- If **q** represents a rotation and **q'** represents a rotation, **qq'** represents **q** rotated by **q'**
- This follows very similar rules as matrix multiplication (I.e., non-commutative)

$$\mathbf{q}\mathbf{q}' = \left(q_0 + iq_1 + jq_2 + kq_3\right)\left(q_0' + iq_1' + jq_2' + kq_3'\right)$$

$$= \left\langle ss' - \mathbf{v} \cdot \mathbf{v}', s\mathbf{v}' + s'\mathbf{v} + \mathbf{v} \times \mathbf{v}' \right\rangle$$

Quaternion Multiplication

- Two unit quaternions multiplied together results in another unit quaternion
- This corresponds to the same property of complex numbers
- Remember(?) multiplication by complex numbers is like a rotation in the complex plane
- Quaternions extend the planar rotations of complex numbers to 3D rotations in space

Flight: Orientation

- Lots of ways to implement flight…
- MP requires the use of quaternions
- One option to change orientation:
  - Set up an initial view using mat4.lookat
  - Keep a quaternion that records current orientation
  - Each frame:
    - Capture key presses as Euler angles
    - Construct a temporary quaternion based on the Euler angles.
      - **quat.fromEuler in glMatrix library…get current release!**
    - Update the orientation quaternion using the temp
      - How?
    - Update the view matrix using the orientation quaternion
      - How?

Flight: Moving Forward

- Keep a user set speed factor
    - User can adjust
- Move in the direction of the current orientation
    - By speedFactor*directionVector
    - You’ll need to experiment to find appropriate speedFactor
- What’s the current directionVector?
    - Think about how you could compute it….