Principles of Computer Game Design and Implementation

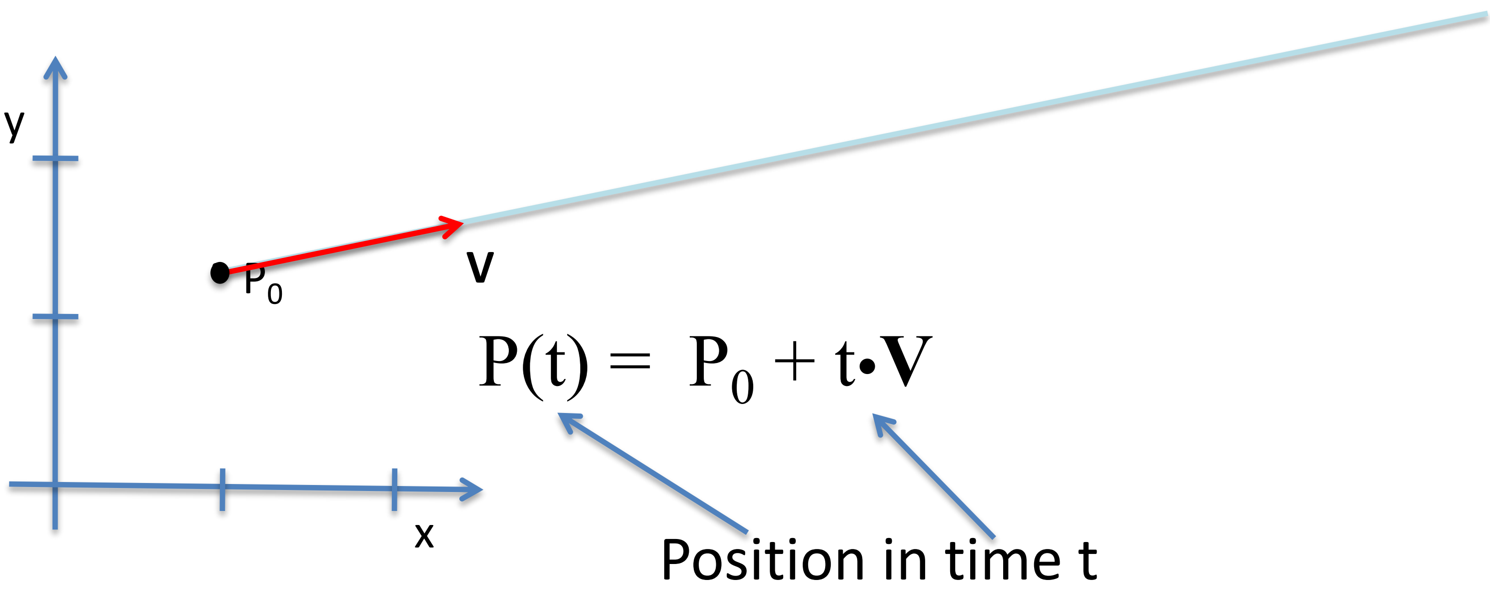
Lecture 7

Movement in Space

- We used vectors to specify the position of an object in space.
- Vectors are also used to specify the *direction of movement*
  - (and other purposes, e.g., lightening, physics, etc.)

Uniform Motion

- An object moves
  - starting from point P_0
  - with a constant speed
  - along a straight line

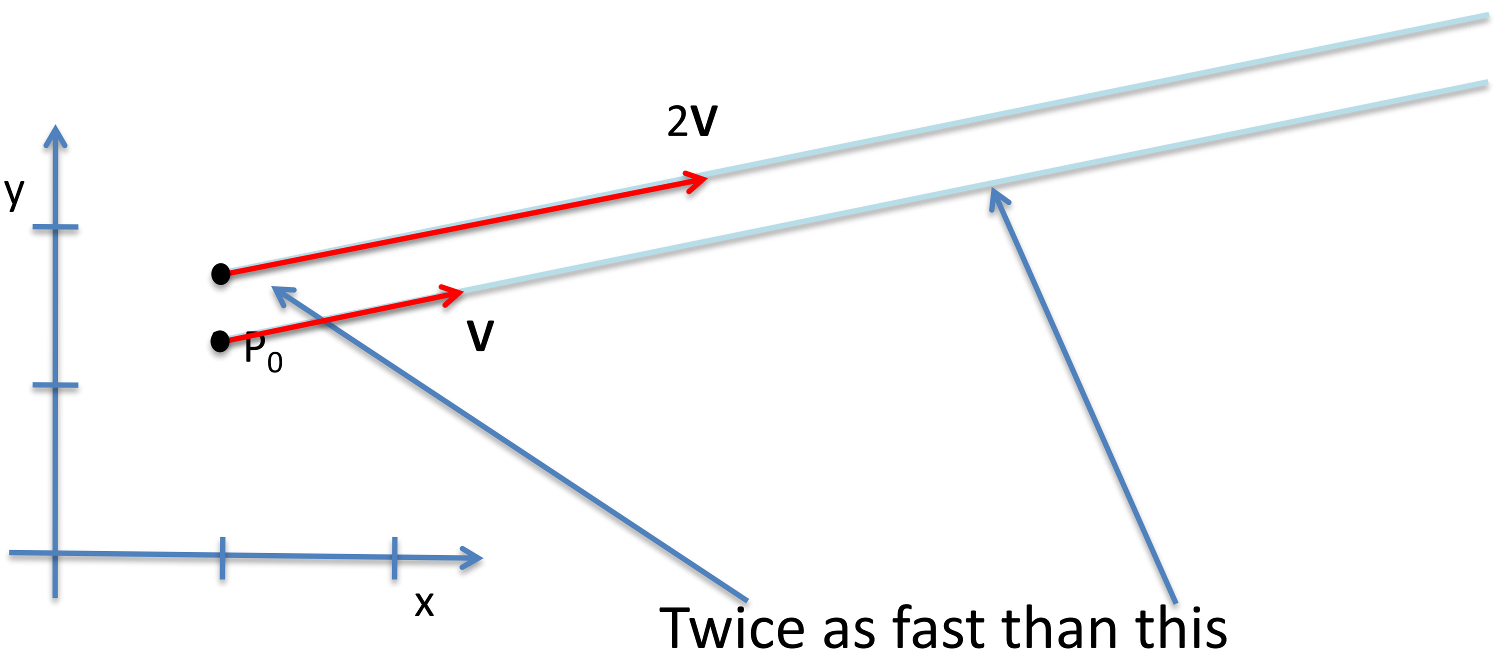


Vector Speed

- Motion equation

$$P(t) = P_0 + t \cdot \mathbf{V}$$

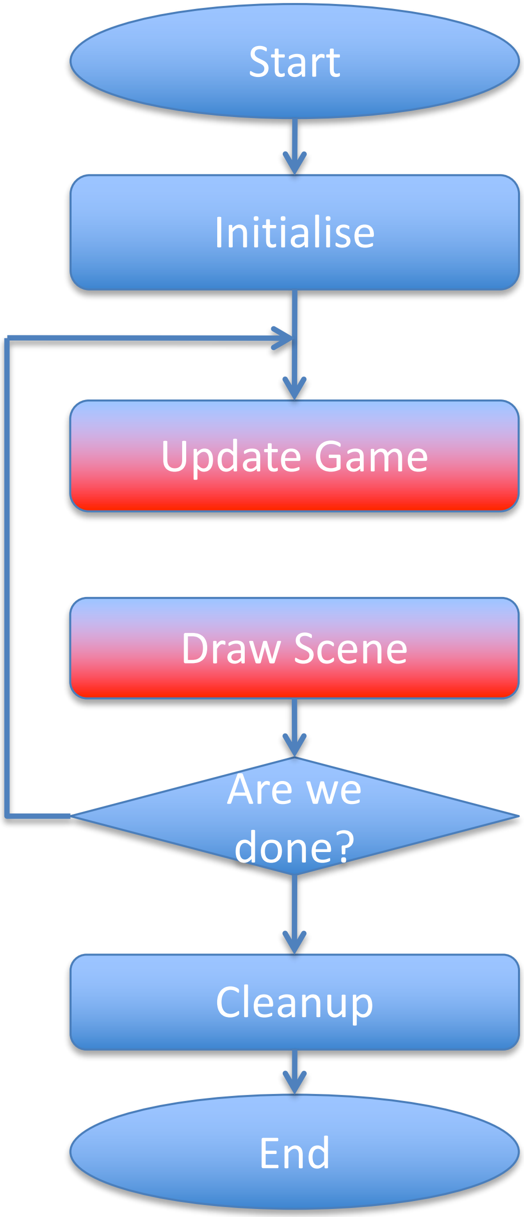
  - **V** specifies *direction* and *speed*



Main Loop

- In a game engine we do not have access to continuous time
- Every iteration *update* the position

$$P = P + \mathbf{V}$$



jMonkeyEngine

- Create two boxes and then…

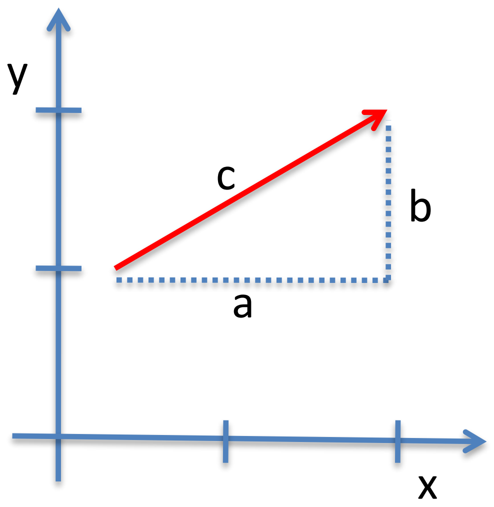
```
public void simpleUpdate(float tpf) {
  b.move(new Vector3f(1,0,0).mult(0.005f));
  c.move(new Vector3f(2,1,0).mult(0.005f));
}
```

Motion Speed

- How to make the objects move in any direction with the same speed?

- Given a vector, we need to be able to keep the direction but make its length 1.

Length of a 2D Vector



Pythagoras theorem

$$c^2 = a^2 + b^2$$

- Given a 2D vector **V**=(x_v,y_v) its length

$$\|\mathbf{V}\| = \sqrt{{x_\mathrm{v}}^2 + {y_\mathrm{v}}^2}$$

E.g. $\mathbf{V} = (2, 7);$ $\|\mathbf{V}\| = \sqrt{2^2 + 7^2}$

A Unit (Direction) Vector

- A vector of length ONE is called a *unit vector*
- One can always *normalise* a vector

$$\mathbf{U} = \frac{1}{\|\mathbf{V}\|} \cdot \mathbf{V}$$

$$\mathbf{V} = (2, 7); \quad \|\mathbf{V}\| = \sqrt{2^2 + 7^2}; \quad \mathbf{U} = \ ?$$

$$\mathbf{U} = \frac{1}{\sqrt{53}} \cdot (2, 7) \approx (0.274, 0.959)$$

Length of a 3D Vector

- Given a 3D vector **V**=(x_v, y_v, z_v) its length

$$\|\mathbf{V}\| = \sqrt{\mathrm{x_v}^{\,2} + \mathrm{y_v}^{\,2} + \mathrm{z_v}^{\,2}}$$

Vector normalisation

$$\mathbf{U} = \frac{1}{\|\mathbf{V}\|} \cdot \mathbf{V}$$

Vector Normalisation

```
Vector3f v = new Vector3f(1,2,3);
float l = v.length();
Vector3f u = v.clone().mult(1/l);

c.move(u.mult(.01f));
```

But then...

```
Vector3f v = new
  Vector3f(1,2,3);
Vector3f u = v.normalize();
float speed = 0.1f; // arbitrary

c.move(u.mult(speed));
```

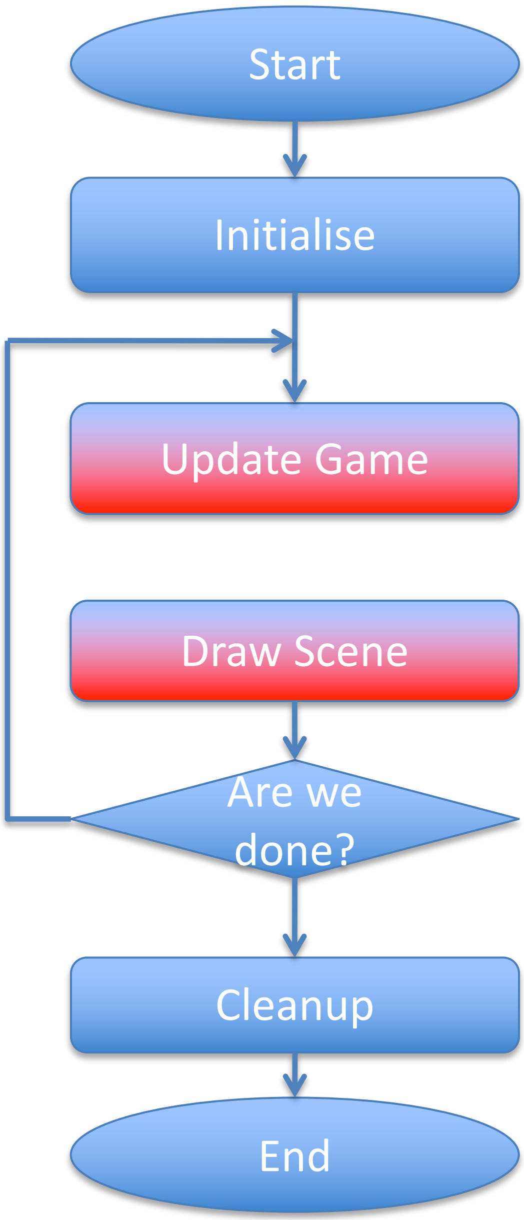
Main Loop

- Every iteration *update* the position

$$P = P + speed \cdot \mathbf{U}$$

- **U** is a unit vector

Different speed on different hardware!



Welcome TPF

- simpleUpdate can use a time-per-frame counter

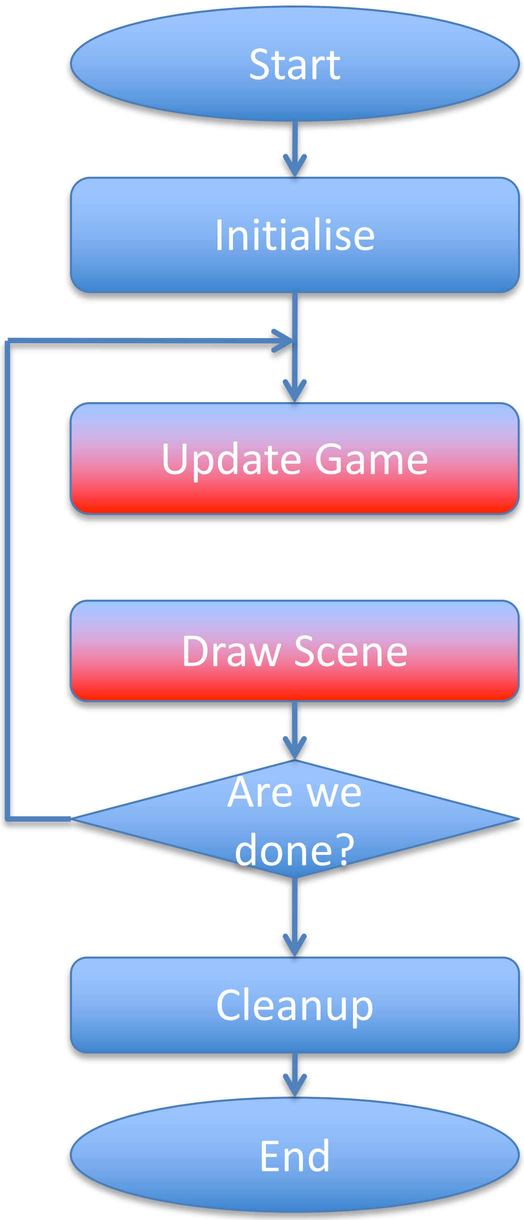
```
c.move(u.mult(tpf));
```

Uniform Motion

- Every iteration *update* the position

$$P = P + speed \cdot tpf \cdot \mathbf{U}$$

- **U** is a unit vector
- *speed* is speed
- *tpf* is time per frame

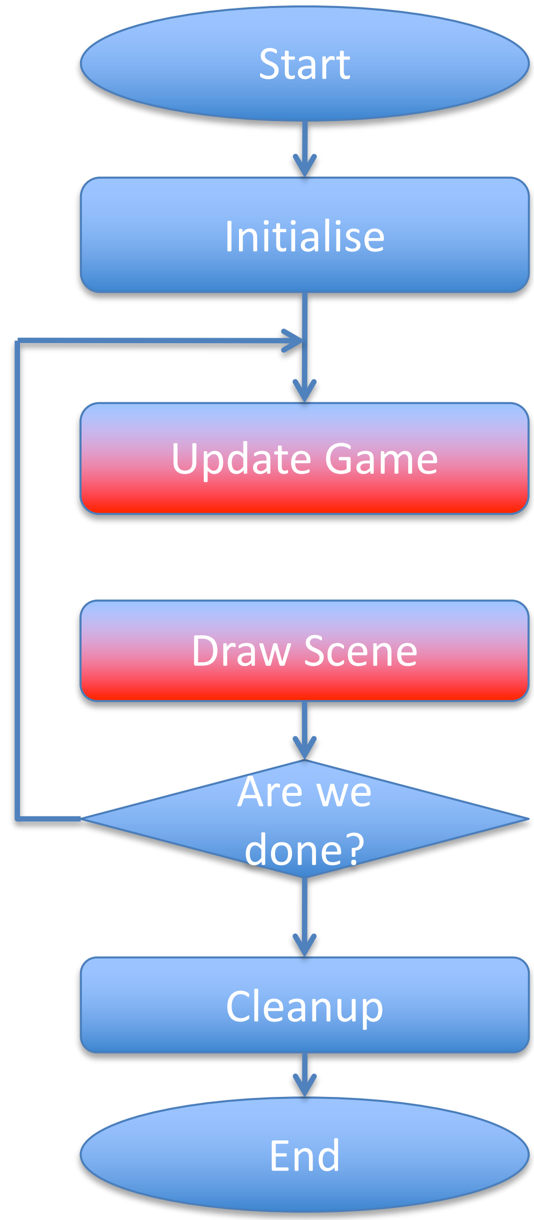


Arbitrary Translation

- Every iteration *update* the position

$$P = P + speed \cdot tpf \cdot \mathbf{U}(t)$$

- **U**(t) - the direction of movement
  - Depends on time!!
- *speed* is speed
- *tpf* is time per frame



Rotation

- Rotating is harder than translating
- We will look at the maths of it tomorrow

- For now, let’s talk about coding

Quaternions

- We could have studies ***what*** quaternions are

Quaternion is a "thing" that helps rotate objects.

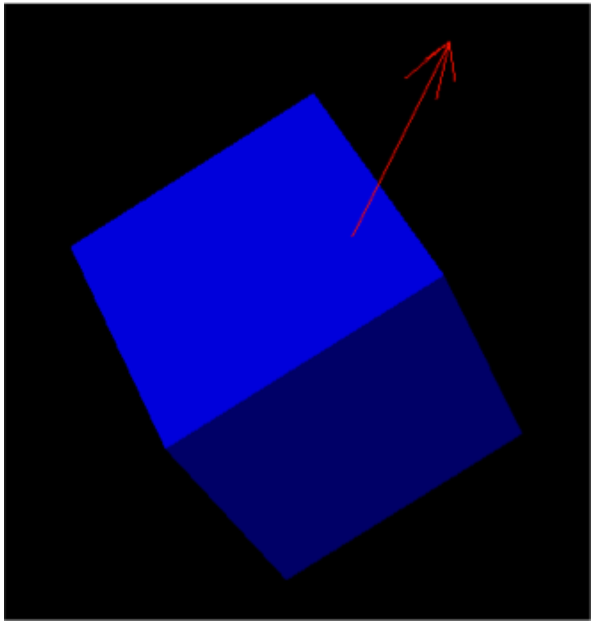
simpleInitApp()

```
…
 Box box = new Box(1, 1, 1);
 b  = new Geometry("Box", box);
 b.setMaterial(mat);
 rootNode.attachChild(this.b);
…
```

Example

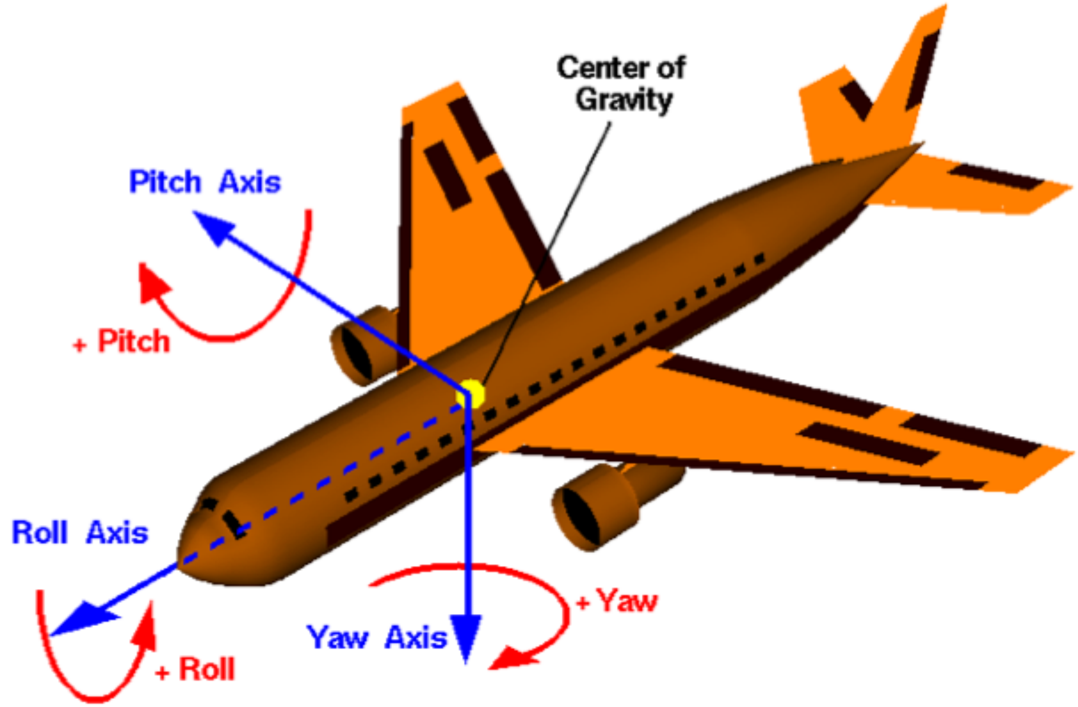
```
…
Vector3f axis =
  new Vector3f(1, 2, 3);
Quaternion quat = new Quaternion();
…

public void simpleUpdate(float tpf) {
  quat.fromAngleAxis(tpf, axis);
  b.rotate(quat);
}
…
```

Demo



But Then…

```
b.rotate(pitch, yaw, roll);
```



also works

http://upload.wikimedia.org/wikipedia/commons/7/7e/Rollpitchyawplain.png

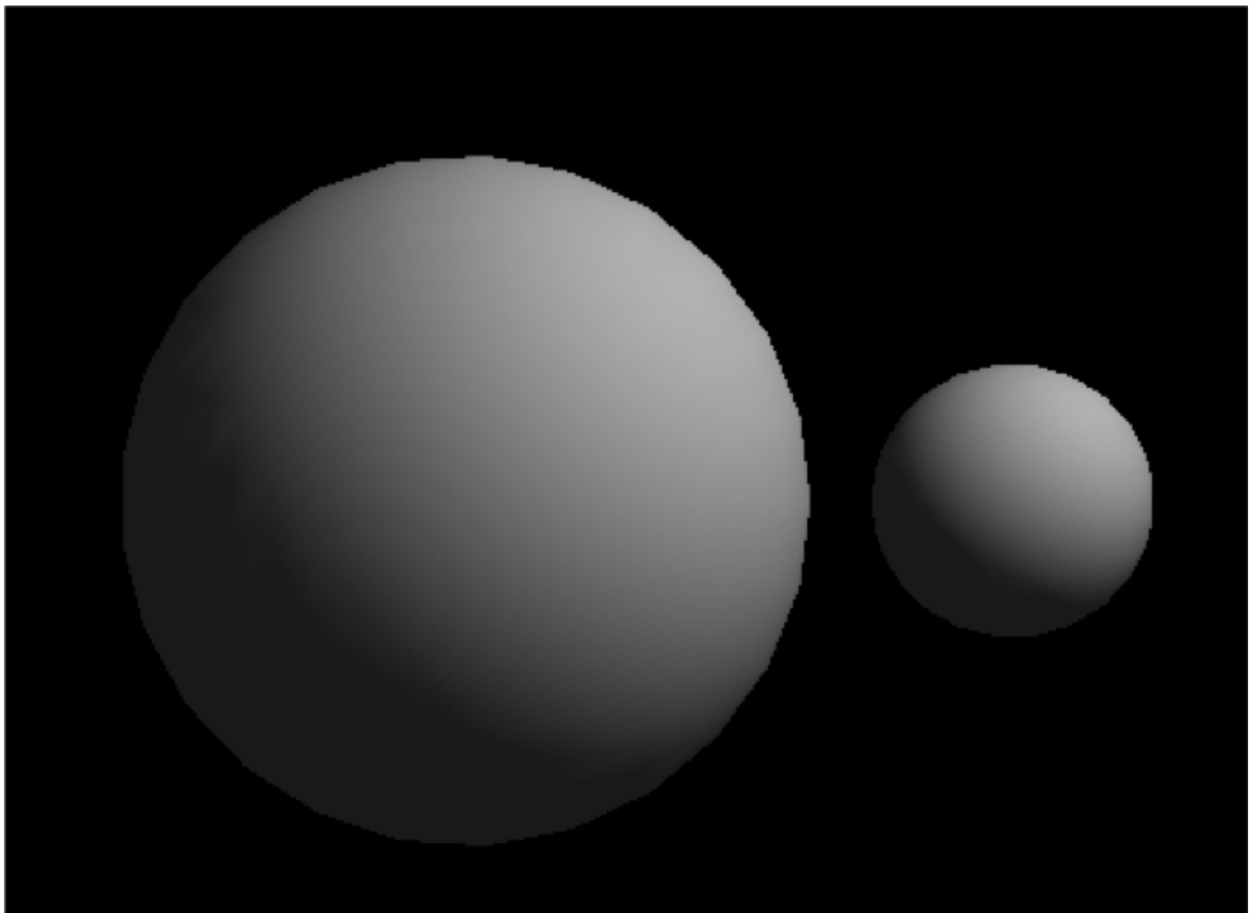
A Simple Example

```
b.rotate(tpf*10*FastMath.DEG_TO_RAD,
         0,
         0);
```

Turns b at the rate of 10 degrees per second around the X axis

Complex Motion Example

- A moon rotating around a planet



simpleInitApp()

```
Sphere a = new Sphere(100, 100, 1);
earth  = new Geometry("earth", a);
earth.setMaterial(mat);
rootNode.attachChild(earth);

Sphere b = new Sphere(100, 100,
 0.3f);
moon = new Geometry("moon", b);
moon.setMaterial(mat);
moon.setLocalTranslation(3, 0, 0);
```

simpleUpdate()

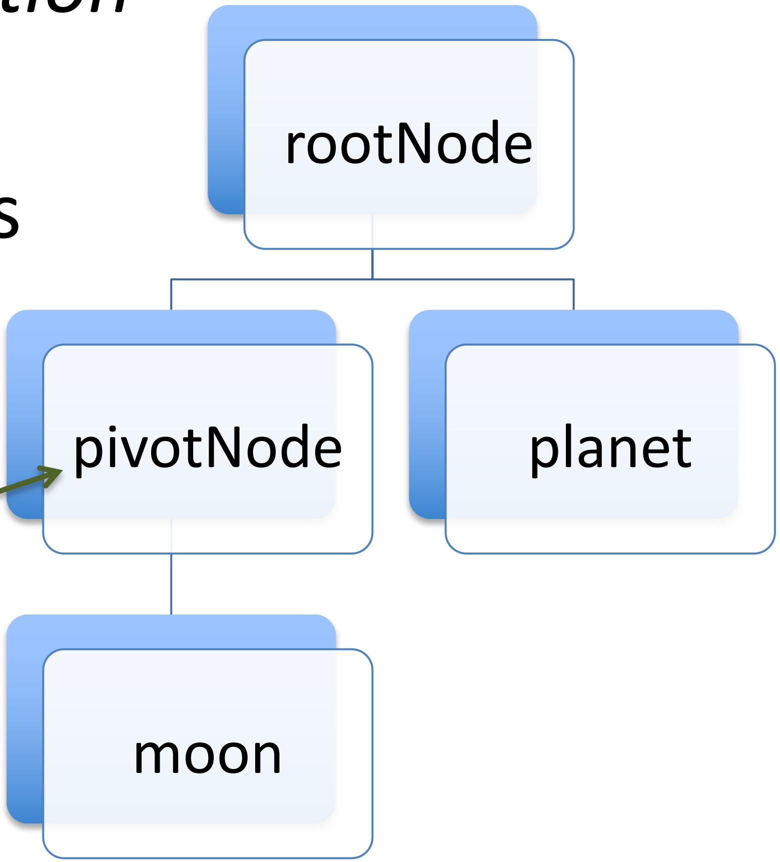
```
public void simpleUpdate(float tpf) {
  quat.fromAngleAxis(tpf, axis);
  moon.rotate(quat);
}
```

Let’s Run It

OOPS!

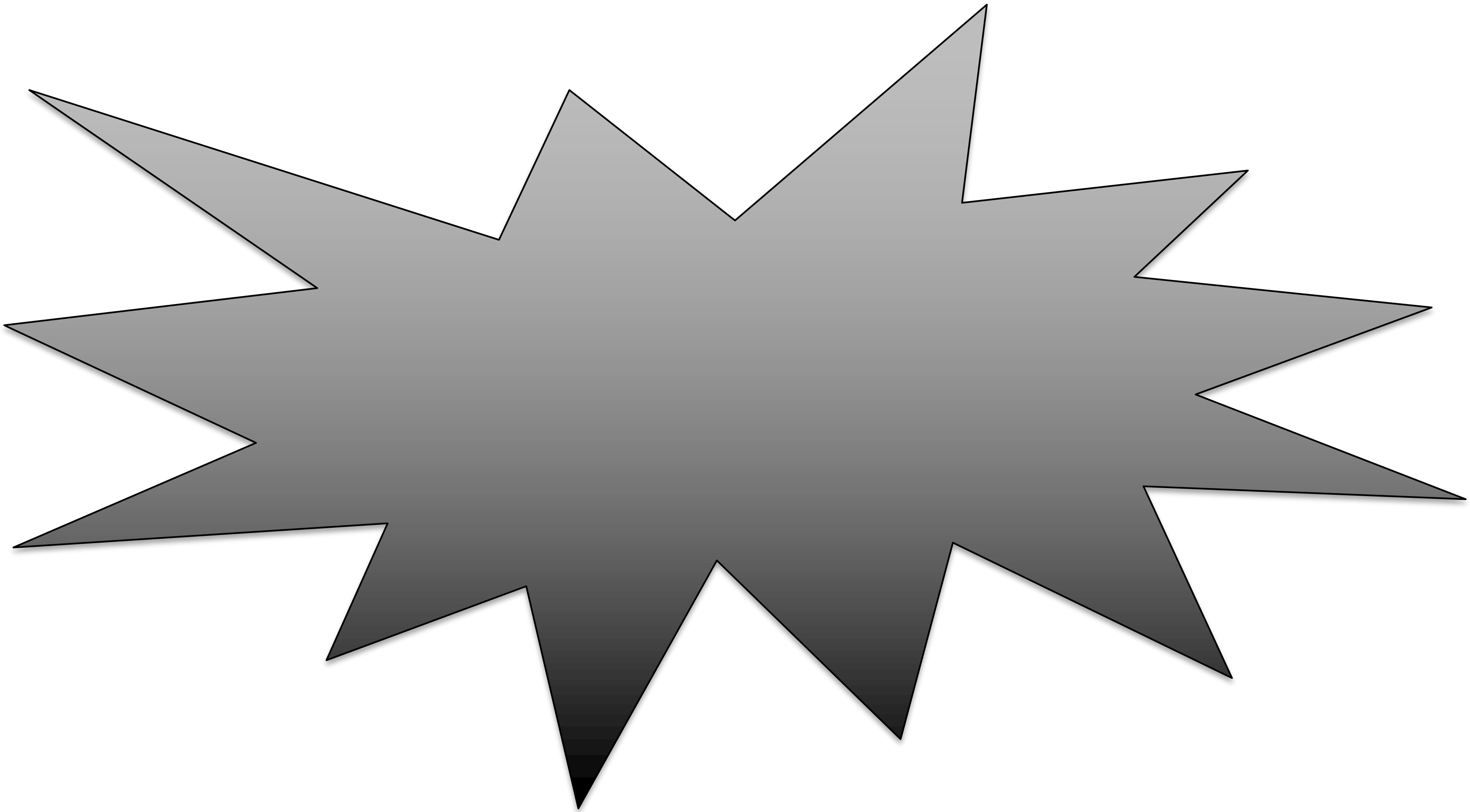
What Went Wrong

- In jME *rotation* and *translation* are independent
- The moon rotates about it's centre
- Scene graph to the rescue!



The pivotNode is the centre of rotation

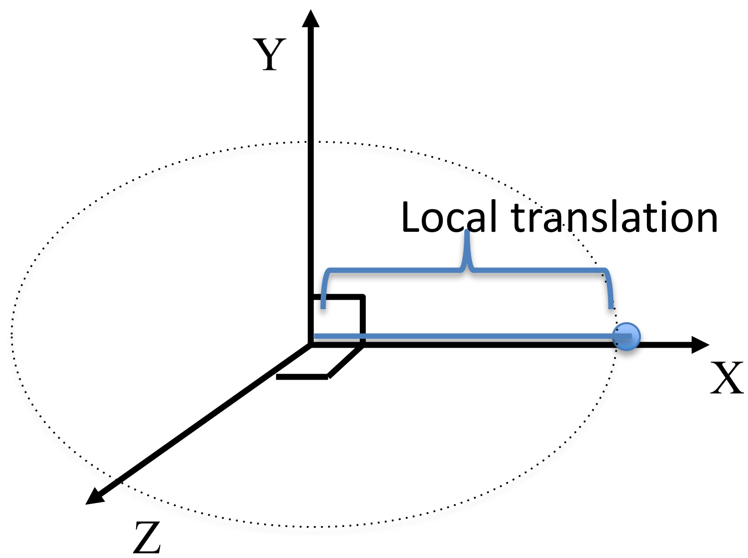
Demo



Code Snippet

```
private Node pivotNode = new Node(“PN");
…
public void simpleInitApp() {
…
  pivotNode.attachChild(moon);
…
}

public void simpleUpdate(float tpf) {
  quat.fromAngleAxis(tpf, axis);
  pivotNode.rotate(quat);
}
```

Pivot Node Explained



Pivot Points

- While it is possible to specify the exact position of a geometry, it is often much simpler to introduce a series of transformations associated with internal nodes of a scene graph.