Introduction to jMonkeyEngine

What is jMonkeyEngine?

Scene graphs

A jME Application

Coordinate systems

What is jMonkeyEngine?

- jME is a game engine made for developers who want to create 3D games and other visualisation applications following modern technology standards
- Uses Java and is platform independent. Can deploy to windows, mac, linux and android.
- OpenSource, non-profit, New BSD License
- http://www.youtube.com/watch?v=eRC9FDin5dA&feature=player_embedded

What is jMonkeyEngine?

- Has integrated tools to make it easier to create games and applications
  - Physics
  - Special effects (pre/post processing, particles)
  - Terrain-, Vegetation-, Water-systems++
  - Graphical User Interface
  - Networking

Why use a high level API?

- Faster development process
- Not necessary to reinvent the wheel

- Provides abstraction from the low level:
  - *Think Objects*…. Not vertices
  - *Think content*… not rendering process.

What does jME do?

- jME performs rendering optimisation
  - View frustum culling
  - Batching
  - State sorting
- Achieves high performance by rendering via OpenGL
- Uses a modern shader based architecture
- Helps organize your 3D scenes, transformations
- jME is single threaded
- jME is NOT thread safe. Only modify the scenegraph from the rendering thread.

Applications of jME

- Games
- Education
- Scientific visualisation
- Information visualisation
- Geographic Information Systems (GIS)
- Computer-aided design(CAD)
- Animation

Getting started

- Software:
  - Java 6 or later
  - jME3 Beta 1 or later, NOT earlier versions
  - LWJGL for communicating with OpenGL
  - Latest version of graphics drivers
- Hardware:
  - Hardware-accelerated graphics card required
  - Shader support
- Documentation:
  - Website: http://jmonkeyengine.org/
  - Book: jMonkeyEngine 3.0 Beginner's Guide

Development environment

- jME SDK
  - Built on top of Netbeans
  - Aims to be similar to editor environments like the UDK
- Other IDE's
  - Netbeans
  - IntelliJ
  - Eclipse
  - …
  - Text editor + command line

Scene graphs

Scene graphs

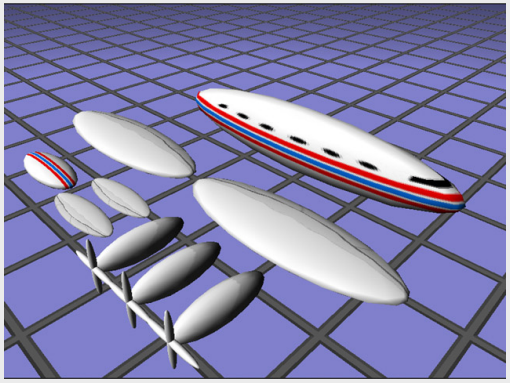
Construction of scene graphs

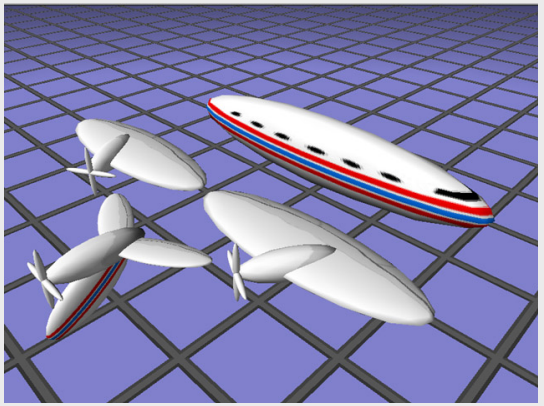
What is a scene graph?

- A datastructure containing all data needed to render the scene
- Commonly used in 3D applications and vector based graphics
- jME renders the scene graph automatically to the screen
- A scene graph is a transform hierarchy
- All nodes contains a transform
- Leaf nodes can contain the geometry
- Geometry can only be a leaf

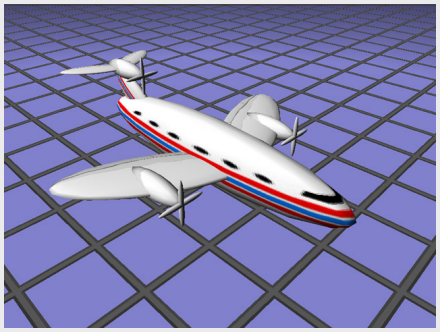
A conceptual scene graph

- Assemble a plane





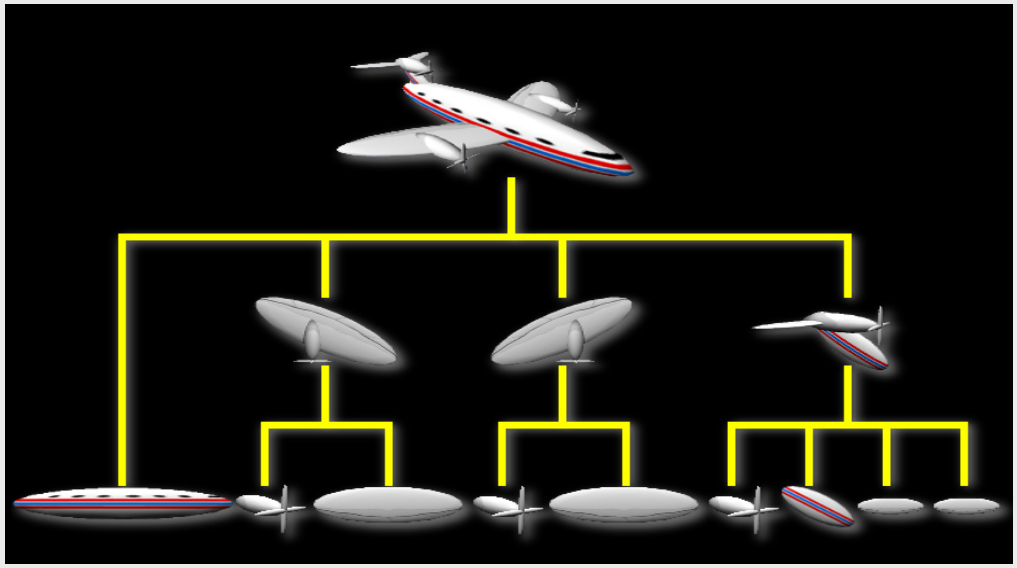
Start with getting an overview of the different parts, group related parts



Assemble the groups to build the plane

Scene graph diagram

- To outline a scene graph can help to clarify a design and ease the development of software
- Better performance with good organisation



Scene graph construction

- Scene graphs are built with the following components.
  - Nodes
  - Geometries (Mesh and Material)
  - (Lights)
  - (Controls)

Scene graph terminology

- A Geometry represents a **visible** 3D object in the scene graph.
- A Node is an **invisible "handle"** for a group of Spatials in the scene graph.
- Geometry and Node inherits from Spatial
- Spatial contains
  - A transform
  - Lights
  - Controls

Scene graph construction

- We create nodes by instantiating jME classes

```
Geometry geom0 = new Geometry( "geom0", mesh0 );
Geometry geom1 = new Geometry( "geom1" );
```

- We modify the nodes by using methods on an instance.

```
geom1.setMesh( mesh1 );
```

- Build groups with nodes

```
Node node = new Node( );
node.attachChild( geom0 );
node.attachChild( geom1 );
```

A jME application

SimpleApplication

- The base for most jME application
- Gives you access to standard game features such as
  - scene graph (rootNode)
  - an asset manager
  - a user interface (guiNode)
  - input manager
  - fly-by camera.

SimpleApplication

- You should inherit from SimpleApplication
- You initialise your data by overriding

```
public void simpleInitApp()
```

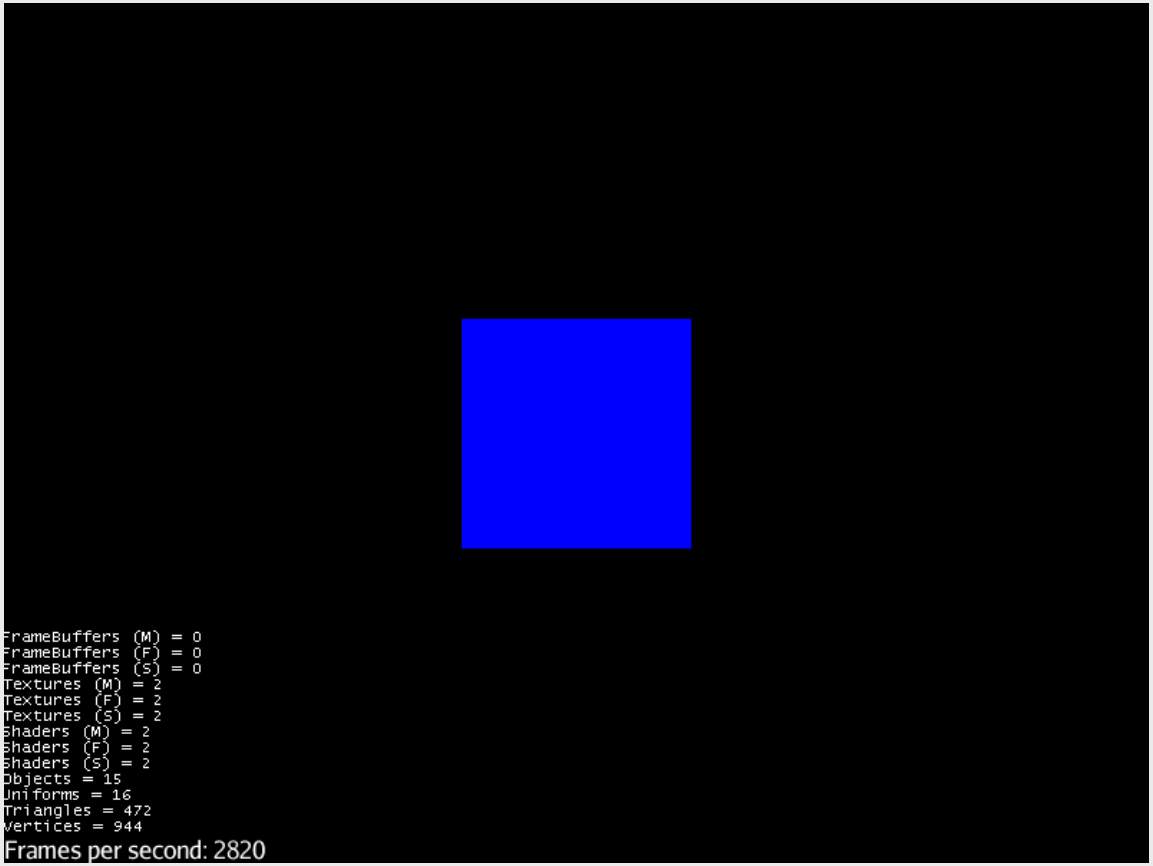
- You have to add your subgraph to the rootNode to make it visible
- Get a callback in the rendering thread by overriding

```
public void simpleUpdate(float tpf)
```

Bypassing SimpleApplication

- It is possible
- You lose functionality
- Only necessary if you have specific requirements
- You can unload everything added by SimpleApplication
- "Simple" means nothing more than necessary

Hello World



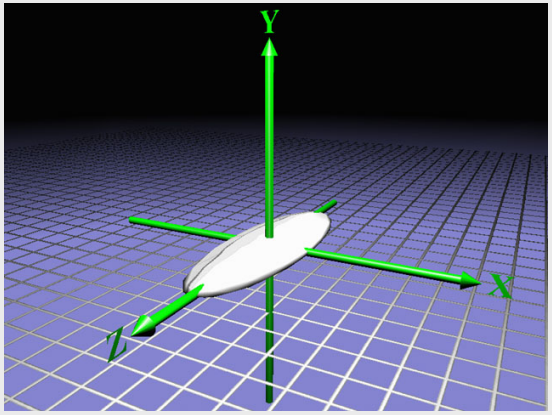
Hello3D.java

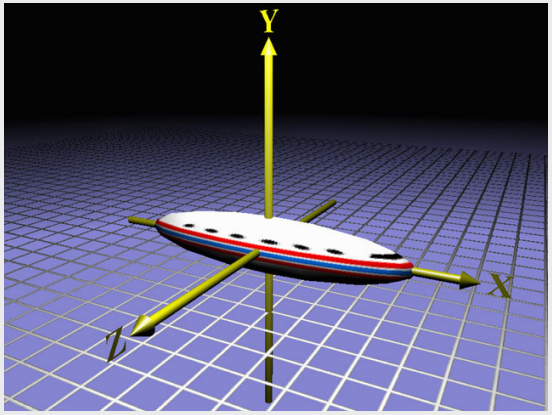
Coordinate systems

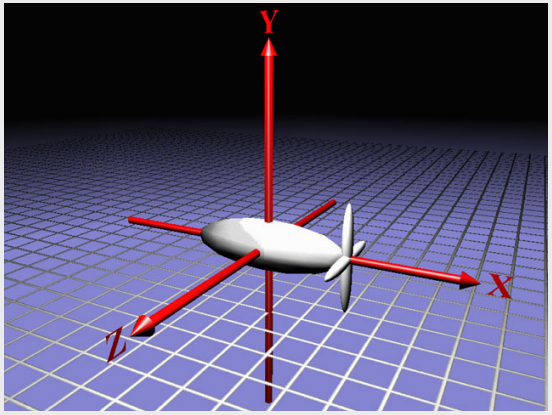
- All spatials share a common *world coordinate system*
- A Spatial creates a new local coordinate system. This is **relative** to the parent
  - Translation (position) sets the relative position
  - Rotation sets the relative rotation
  - Scale sets the relative size
- If you transform the parent system, all the children moves with it

Using the coordinate system

- Every part is built into their own local coordinate system

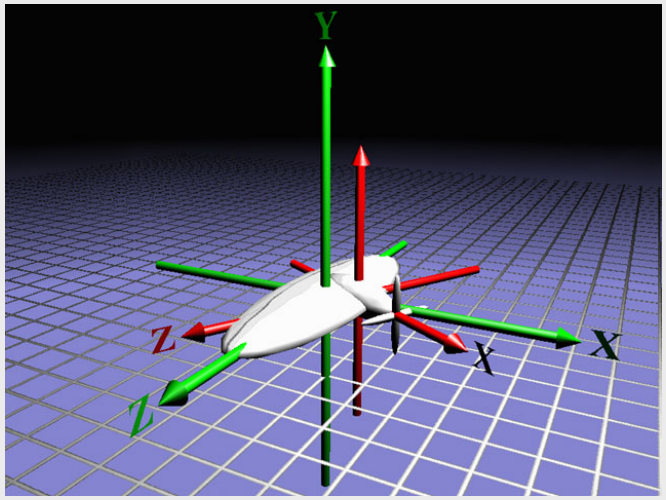


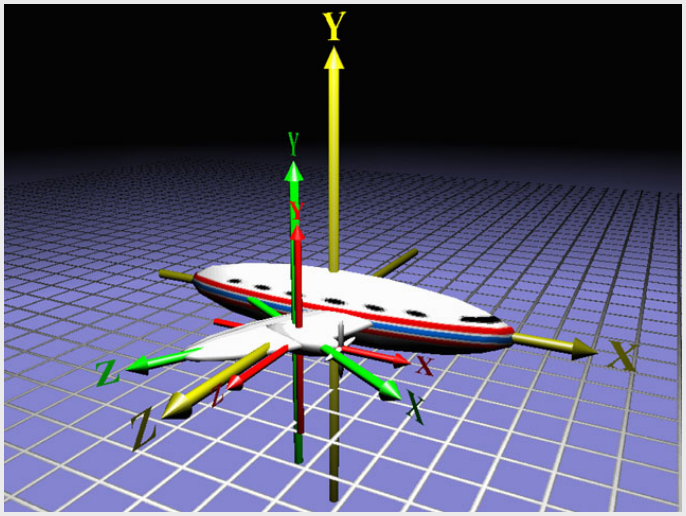




Using the coordinate system

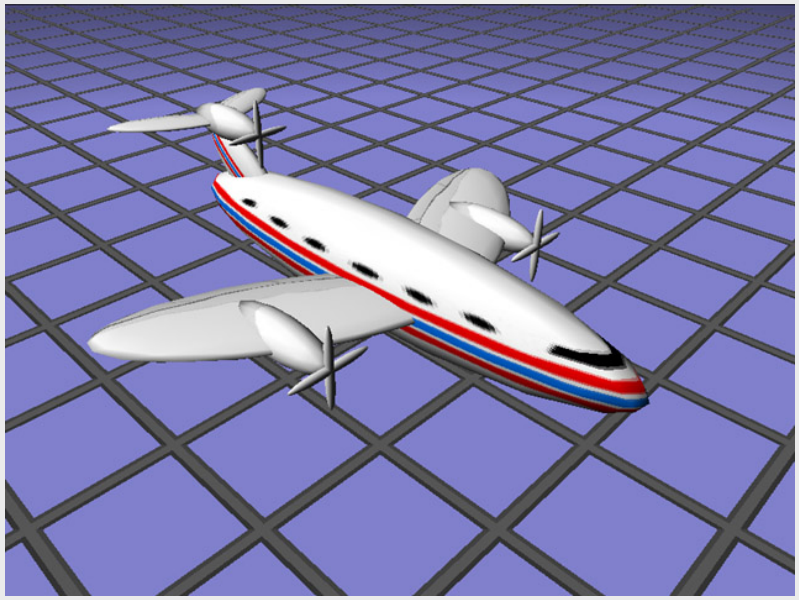
- When these parts are assembled, this transposes the childrens shapes into the parents coordinate system

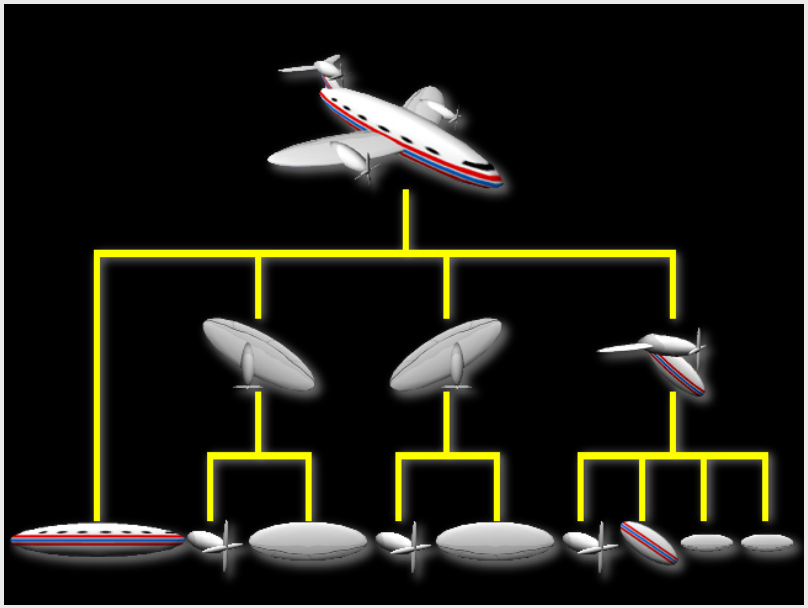




Using the coordinate system

- And so on, until we have built the plane





Transformations

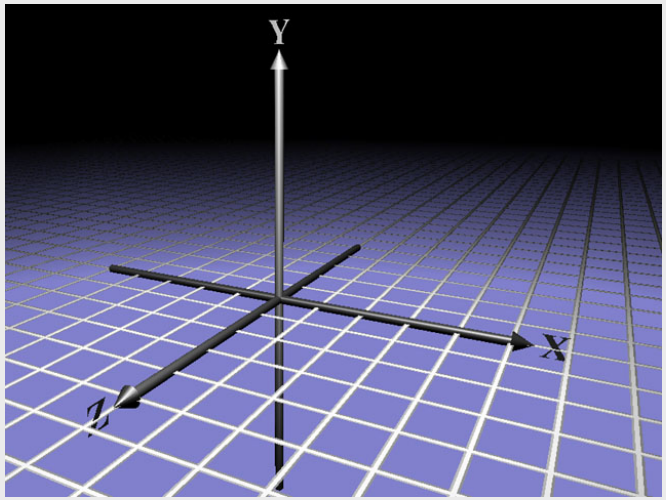
- Every spatial has a *Transform* component
- The Transform represents the *translation*, *rotation* and *scale* of the spatial

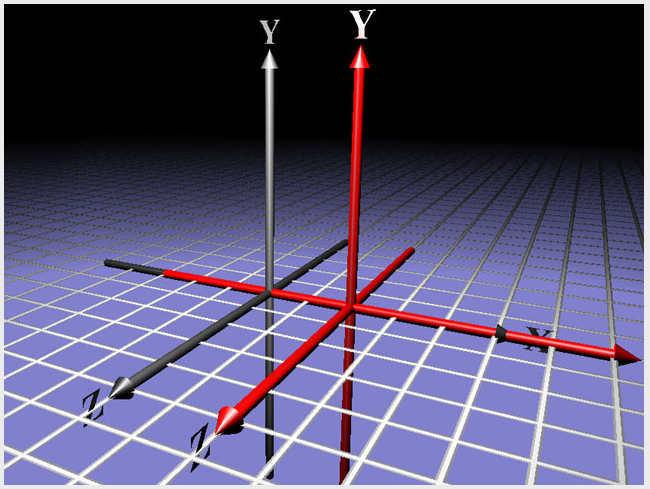
Identity

- By using the method loadIdentity(), the transform is set to Identity
  - No translation in X, Y or Z
  - No rotation
  - A scale factor of 1 on X, Y and Z

Positioning in a coordinate system

- A vector moves the coordinate system
  - Right-hand coordinate system
  - A Vector3f holds the X,Y and Z distance





Translation example code

- Build the geometry

```
Geometry geom = new Geometry("geom", mesh);
```

- To move the geometry +1.0f in the x-direction we need a Vector3f

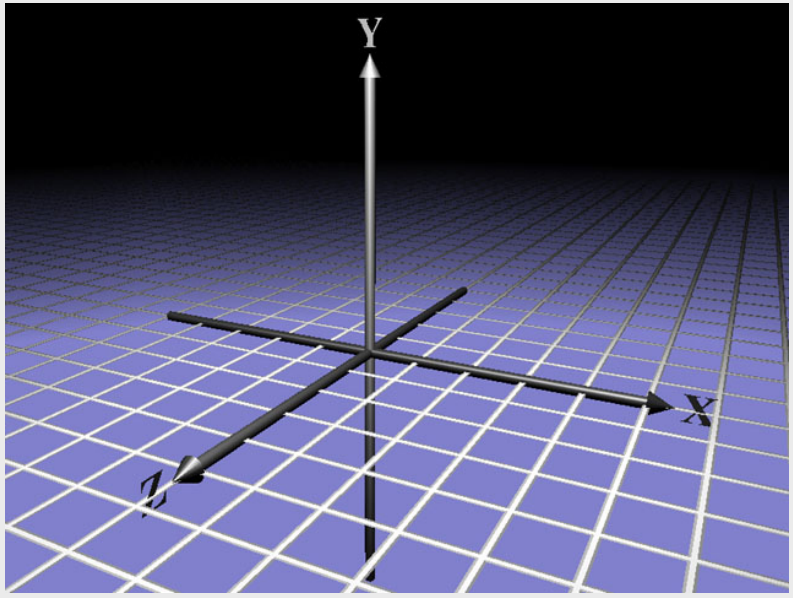
```
Vector3f trans = new Vector3f(1.0f, 0.0f,0.0f);
```

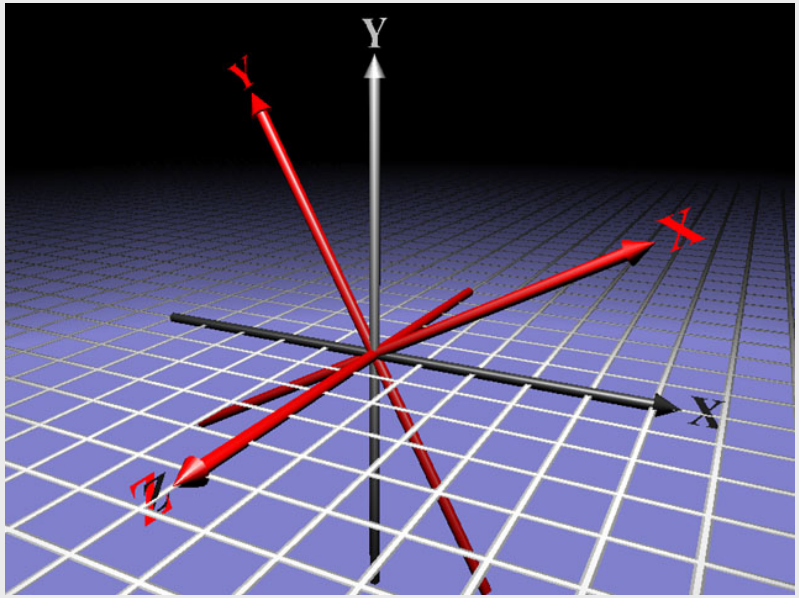
- This translation must be applied to the geometry

```
geom.setLocalTranslation(trans);
```

Rotate a coordinate system

- Rotate around x,y or z and an axis
- Rotate around axis





Rotation, simple example

- Create the geometry

```
Geometry geom = new Geometry("geom", mesh);
```

- Develop a 3D Transform for rotation around y-axis 45 degrees.

```
Quaternion quat = new Quaternion( );
quat.fromAngleNormalAxis((float)Math.PI/4,
   Vector3f.UNIT_Y);
```

- Set the rotation to the geometry

```
geom.setLocalRotation(quat);
```

Scaling a coordinate system

- By scaling we increase or decrease the size of a coordinate system and the shapes to the coordinate system
  - Normal scale is 1.0f
  - To scale equally much in x, y and z we can scale with a simple scale factor

    ```
    void setLocalScale ( float scale );
    ```
  - Or we can use individually scaling factors for each axis

    ```
    void setLocalScale (Vector3f scale);
    ```

Scaling, example code

- Create the geometry

```
Geometry geom = new Geometry("geom", mesh);
```

- Create a Vector3f to scale with different values in the x.y and z axis

```
Vector3f scale = new Vector3f(1.3f, 0.5f, 1.0f);
```

- Set the local scale for the geometry

```
geom.setLocalScale(scale);
```

Modification of parts of transform

- Modification of parts of an existing transfom
  - The other parts of the transform is untouched
  - Is used to combine translation, rotation and scaling

```
void setTranslation(float x, float y, float z);
void setTranslation(Vector3f trans);
void setRotation(Quaternion quat);
void setScale(float scale);
void setScale(Vector3f scale);
```

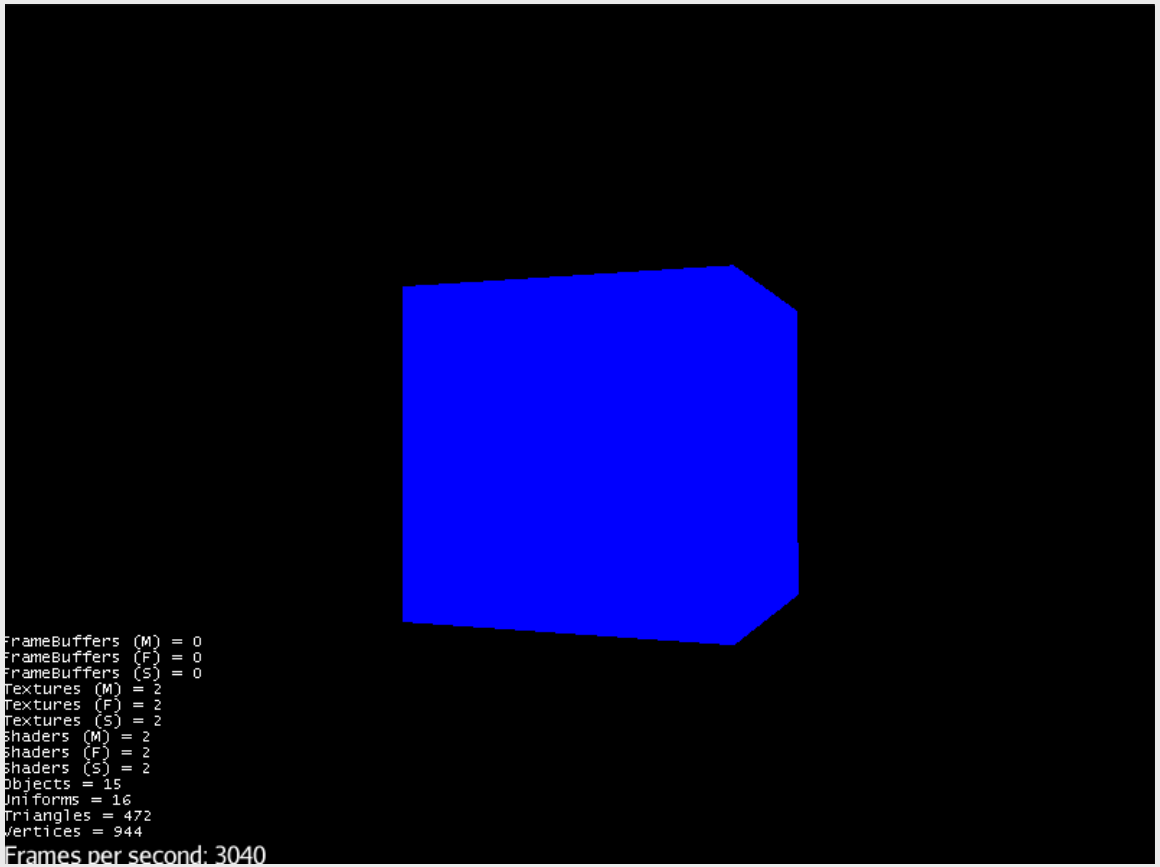
Tranform points

- It is possible to transform points from one coordinate system to another

```
    Vector3f transformVector(Vector3f in,
Vector3f store)
```

- jME uses Vector3f to represent both points and vectors.

Hello Rotation



HelloRotation.java