Content

Building Geometry

Appearance

Lights

Model Loaders

Building Geometry

- A `Geometry` represents a 3D object:
- Mesh*:*
  - *The form or structure of a shape*
- Material*:*
  - *The color, transparency, and shading of a shape*.

Geometry class methods

- Methods on `Geometry` set mesh and material attributes

```
Geometry(String name)
Geometry(String name, Mesh mesh)

public void setMesh(Mesh mesh)
public void setMaterial(Material material)
```

Defining Mesh for Geometry

- Three choices when creating mesh for geometry:

1. Built in shapes (Box, Sphere, etc.)
2. Load 3D models (created in 3ds max, blender, etc.)
3. Create mesh programatically

Coordinate System

- 3D coordinates are given in a *right-handed coordinate system*

  X = left-to-right

  Y = bottom-to-top

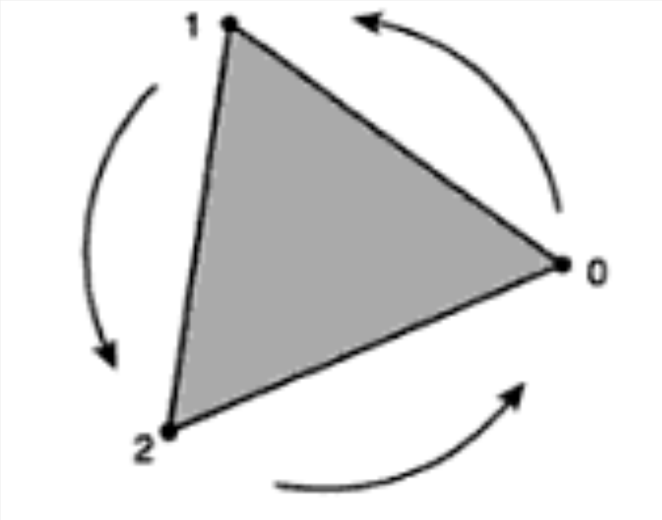
  Z = back-to-front

Coordinate Order

- Polygons have a front and back:
  - By default, only the front side of a polygon is rendered
  - A polygon's winding order determines which side is the front
  - Most polygons only need one side rendered
  - You can turn on double-sided rendering, at a performance cost

Using Coordinate Order

- jME is uses a right-handed coordinate system
  - The front of the polygon is determined by the ordering of the vertices
  - Counterclockwise



Defining Vertices

- A *vertex* describes a polygon and contains:
  - A 3D coordinate
  - A color
  - A texture coordinate
  - A lighting *normal vector*
- Only the 3D coordinate in a vertex is required, the rest are optional

Defining Vertices

- A vertex normal defines surface information for lighting
  - But the coordinate winding order defines the polygon's front and back
- If you want to light your geometry, you must specify vertex lighting normals
  - Lighting normals must be *unit* length

Building Meshes

- jME supports three types of geometric primitives:
  - Points
  - Lines
  - Triangles

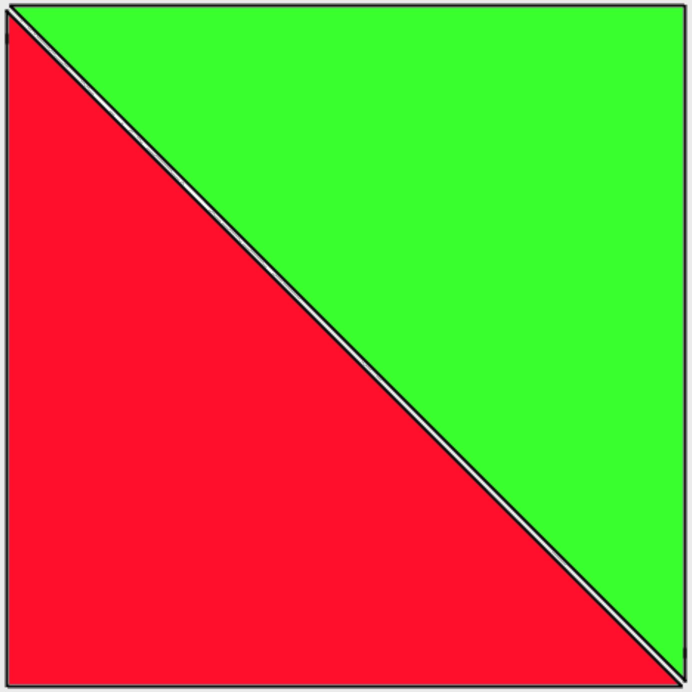
- The Mesh class have several derived subclasses that create specific shapes:
  - Boxes, cylinders, spheres
  - Domes, pyramid, torus
  - Surfaces or curves

Defining vertices

- Non-Indexed
  - Define vertices in singles, pairs or triples to build points, lines, and triangles one at a time.
  - Redundant coordinates, lighting normals, colors, and texture coordinates

- Indexed
  - Indices are used along with the lists of coordinates, lighting normals, color and texture coordinates
  - Indices select which coordinates to use from each list
  - Indices are also used for lighting normals, colors, and texture coordinates
  - For surfaces, the same vertices are reused for adjacent lines and triangles, providing an efficient use of vertex information
  - No redundant coordinates in indexed geometry

Building Meshes



- Non-indexed:

```
Vector3f[] vertices = new Vector3f[]{
         new Vector3f(0, 1, 0), // red triangle
         new Vector3f(0, 0, 0),
         new Vector3f(1, 0, 0),
         new Vector3f(1, 0, 0), // green triangle
         new Vector3f(1, 1, 0),
         new Vector3f(0, 1, 0),
};
```

- Indexed:

```
Vector3f[] vertices = new Vector3f[]{
         new Vector3f(0, 0, 0),
         new Vector3f(1, 0, 0),
         new Vector3f(0, 1, 0),
         new Vector3f(1, 1, 0),
};
```

```
int[] indices = new int[]{
          2, 0, 1, //red tri
          1, 3, 2, //green tri
};
```

Building different types of meshes

- There are 8 different ways to represent the vertex data in the mesh:
  - `Points`
  - `Lines`
  - `LineStrip`
  - `LineLoop`
  - `Triangles`
  - `TriangleStrip`
  - `TriangleFan`
  - `(Hybrid)`

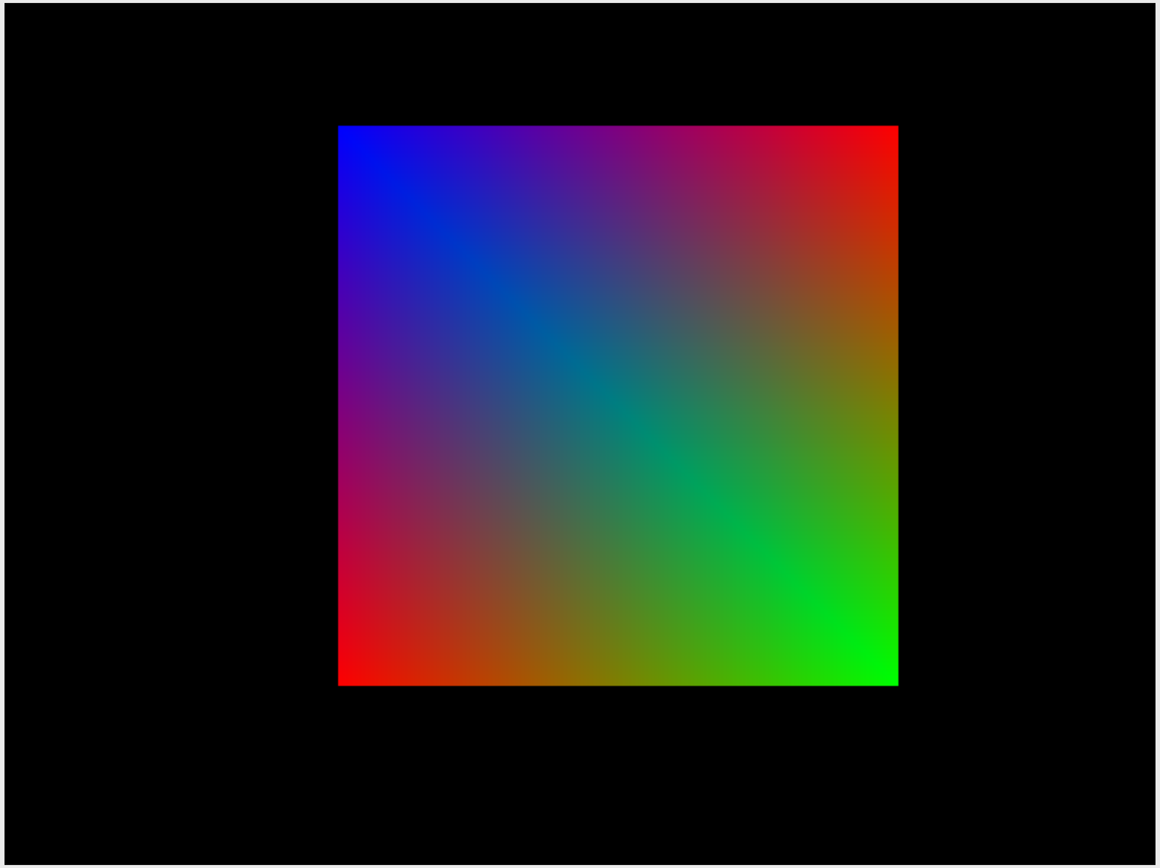
Setting mesh data

- Mesh data is set through native buffers

```
void setBuffer(VertexBuffer.Type type, int components, java.nio.ByteBuffer buf);
void setBuffer(VertexBuffer.Type type, int components, java.nio.FloatBuffer buf);
void setBuffer(VertexBuffer.Type type, int components, java.nio.IntBuffer buf);
```

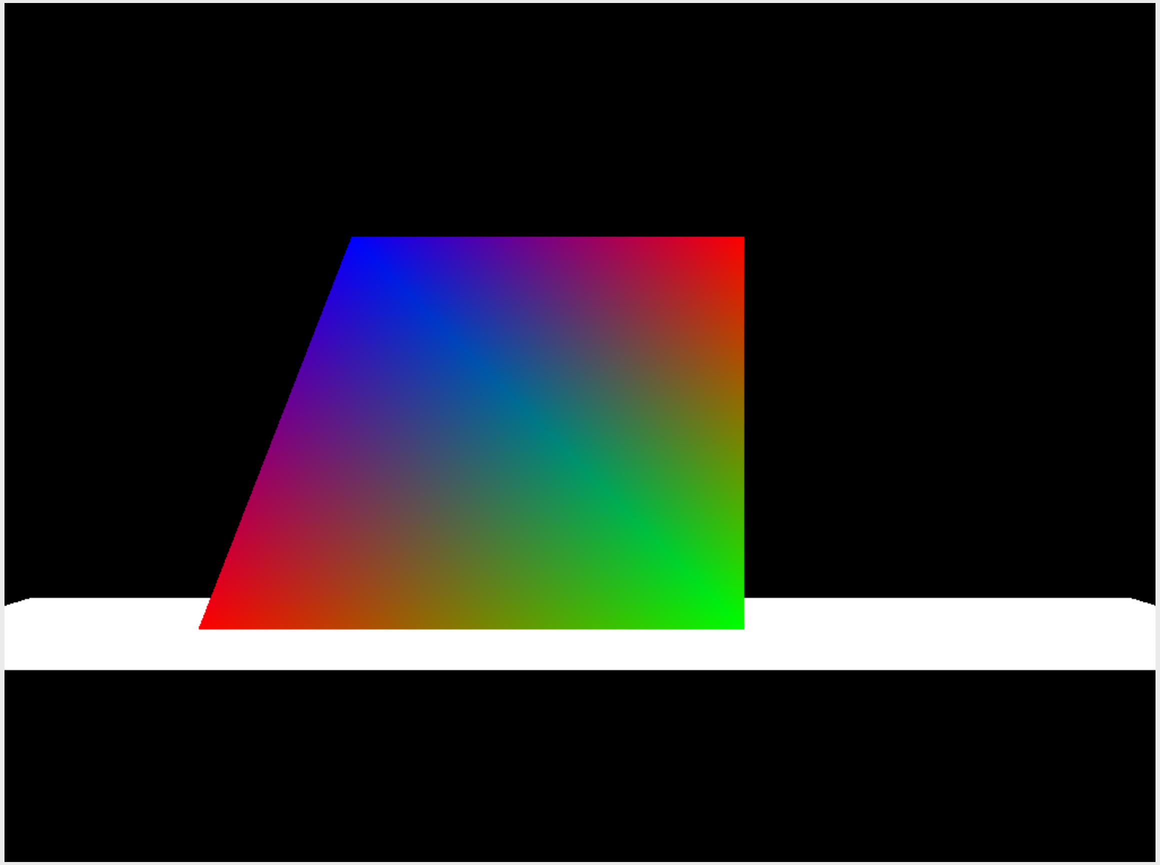
- VertexBuffer types:
  - `Position`
  - `Normal`
  - `Index`
  - `Color`
  - `TexCoord`
  - `+++`

Mesh Example



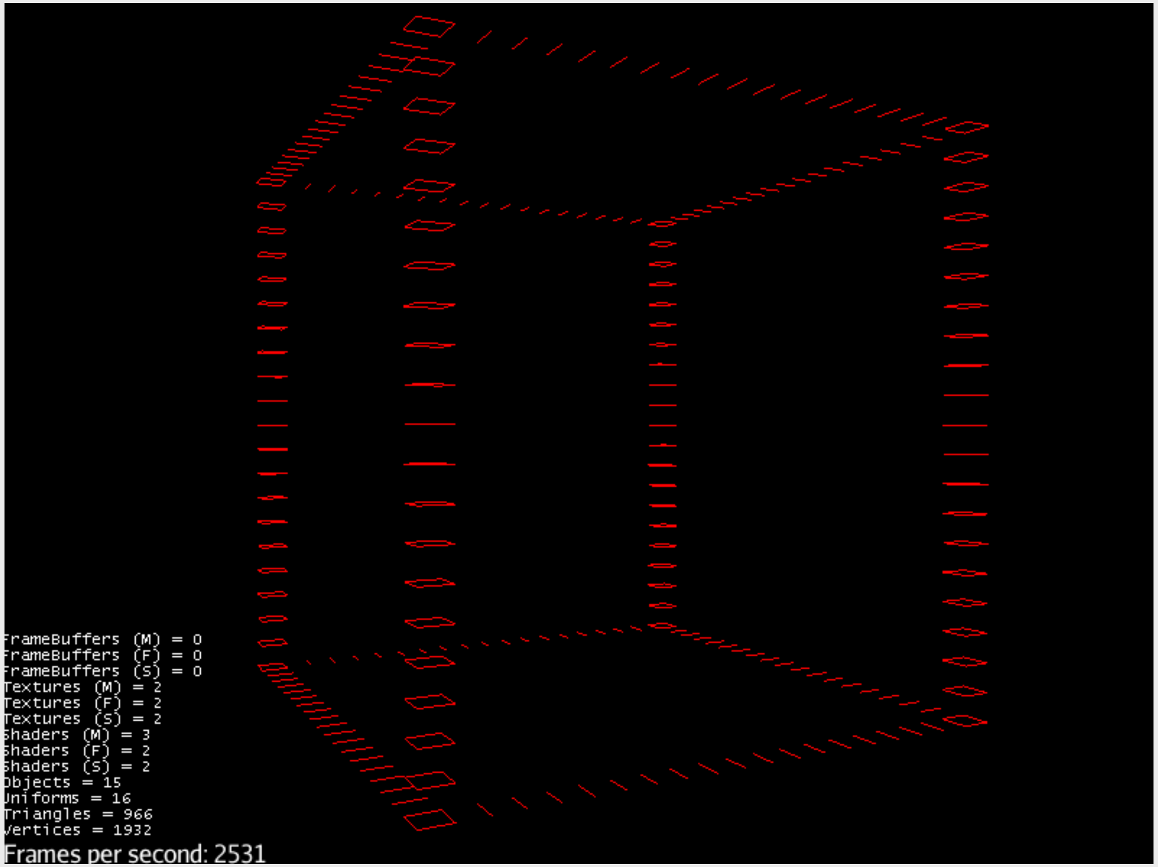
MeshExample.java

Dynamic Mesh Example



MeshExample.java

Render Modes Example



```
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 2
Textures (F) = 2
Textures (S) = 2
Shaders (M) = 3
Shaders (F) = 2
Shaders (S) = 2
Objects = 15
Uniforms = 16
Triangles = 966
Vertices = 1932
Frames per second: 2531
```

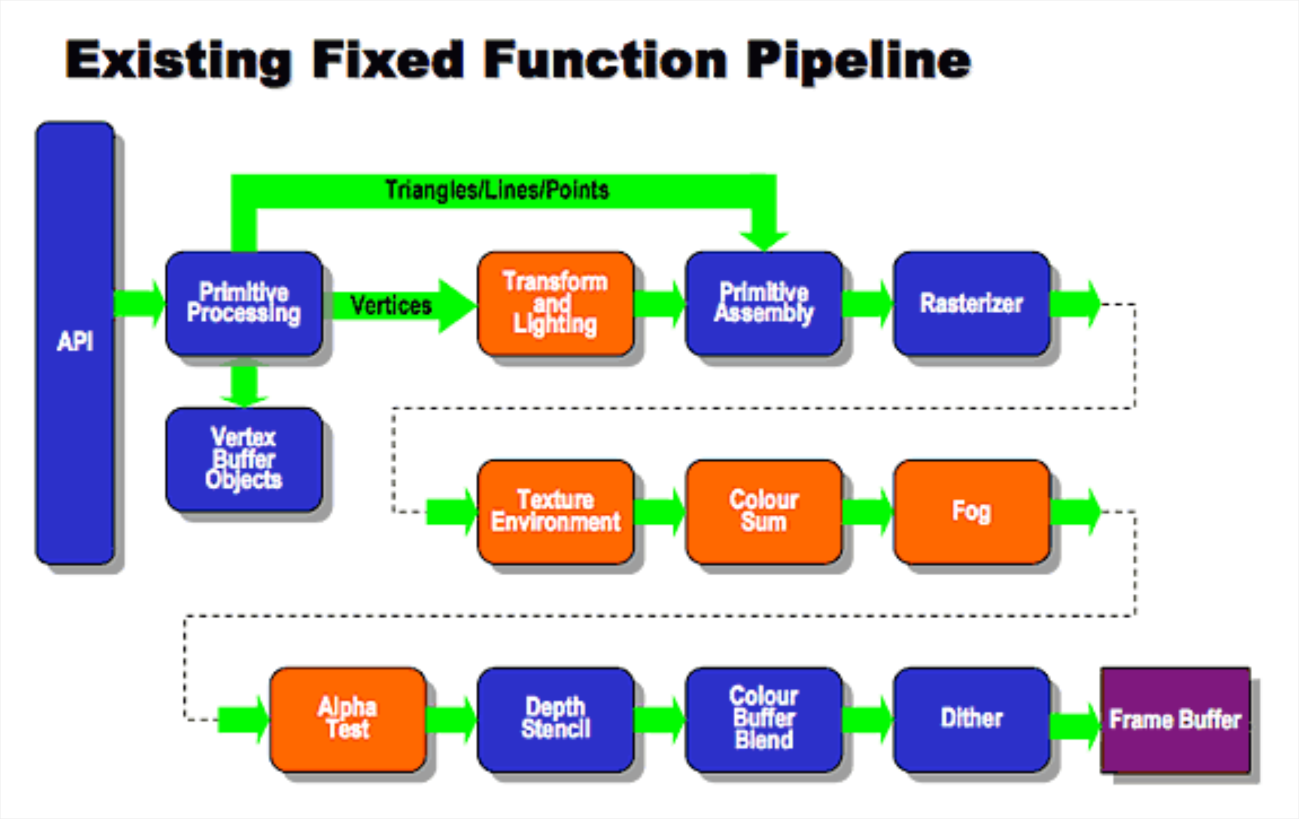
BoxRenderModes.java

Appearance

Appearance

- How to control how jME renders an object?
  - No Fixed Function Pipeline (FFP)
  - jME is fully shader based
  - Features built in shaders that "mimics" FFP
  - You can do almost anything you want

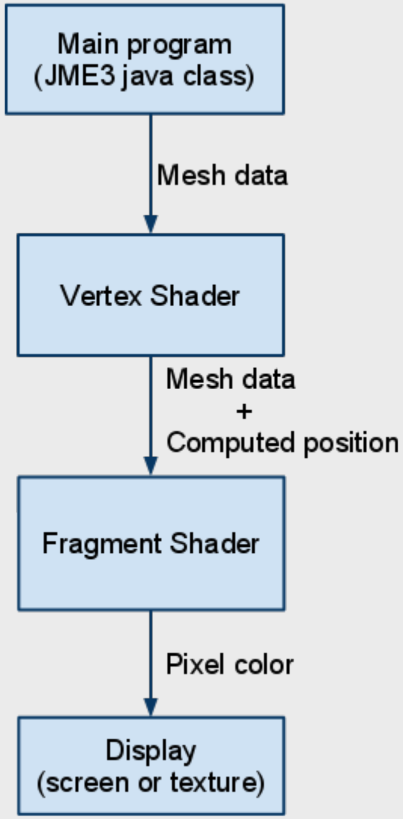
Fixed Function Pipeline



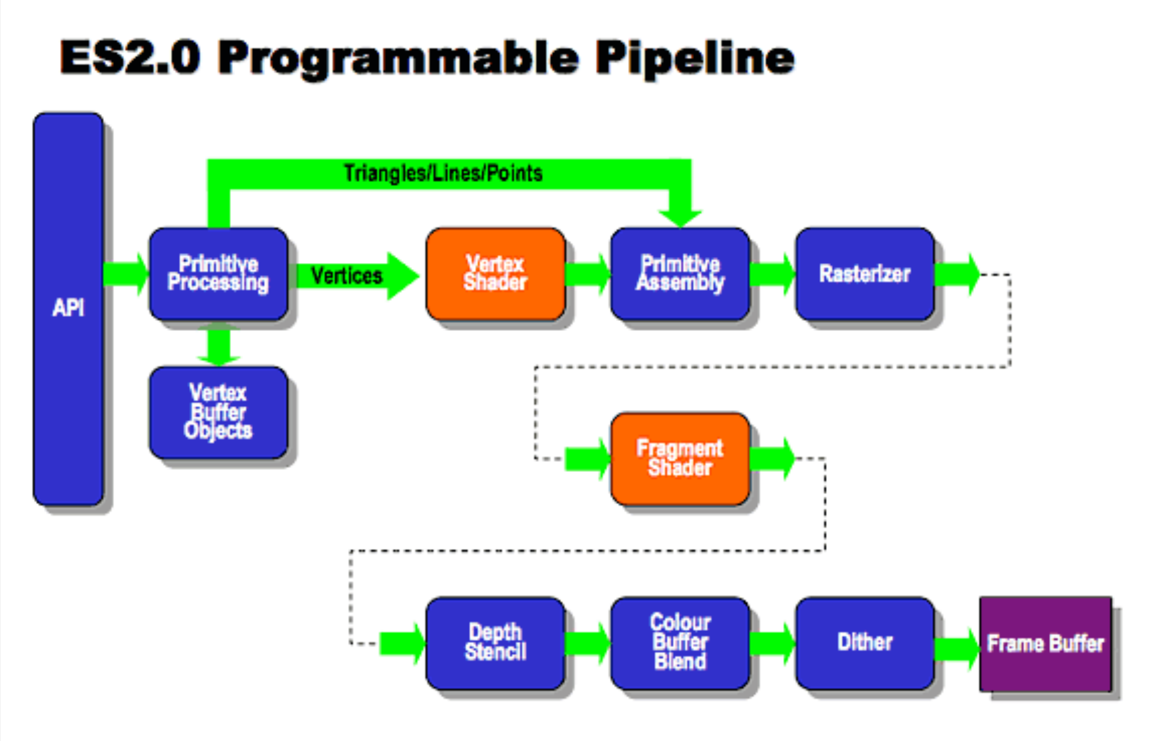
Source: krhonos.org

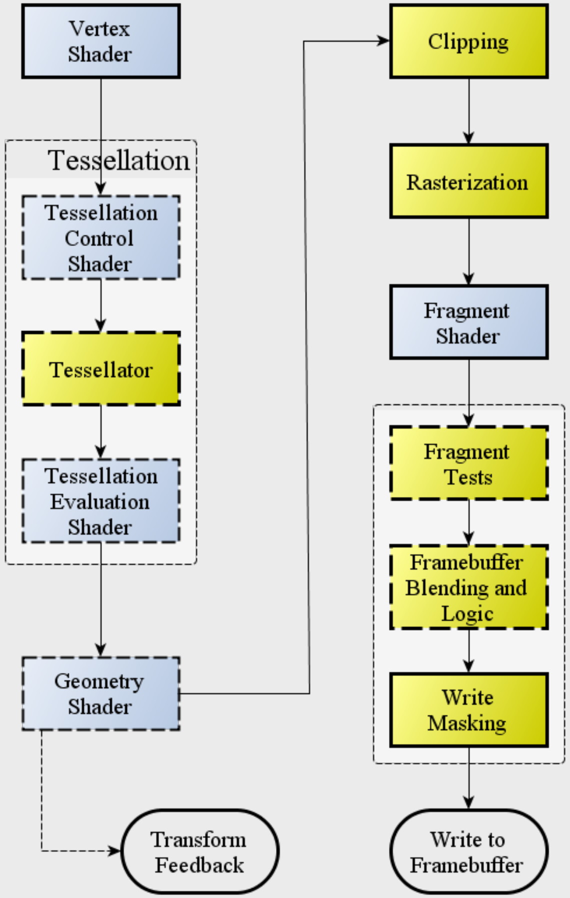
Shaders

- What is a shader?
  - – Vertex Shader
  - – Tesselation Shader
  - – Geometry Shader
  - – Fragment Shader
- GLSL
- Other formats
  - – HLSL, CG



Programmable Pipeline





Sources: krhonos.org and opengl.org

jME and Shaders

- Shaders are encapsulated into Material Definitions
- .j3md – files

- You don’t need to master shaders, a discipline in itself

Materials

- Materials control how jME renders geometry.
- Materials are created/loaded from a Material Definition file (.j3md).
- This means you have to load Material Definitions!
- Rendering specifications are set on the Material object.
- The rendering specifications available in the material depends on the Material Definition.
- All geometry must have a material set!

Material Attributes

- We will focus on two Material Definitons that mimic FFP, in the jME3-core.

`jME3-core.jar/Common/MatDefs/Misc/`**`Unshaded.j3md`**

`jME3-core.jar/Common/MatDefs/Light/`**`Lighting.j3md`**

- More on these shortly!

Material Attributes

- `Lighting Material` controls:
  - Ambient, diffuse and specular colour
  - Shininess
  - Textures
  - +++

Material example code

- Create material for setting shape colours

```
Material mat = new Material(assetManager, "Common/MatDefs/Light/
   Lighting.j3md");
mat.setBoolean("UseMaterialColors", true);
mat.setColor("Ambient", new ColorRGBA(0.3f, 0.3f, 0.3f, 1.0f));
mat.setColor("Diffuse", new ColorRGBA (0.5f, 0.5f, 0.5f, 1.0f));
mat.setColor("GlowColor", new ColorRGBA (0.0f, 0.0f, 0.0f, 0.0f));
mat.setColor("Specular", new ColorRGBA (0.8f, 0.8f, 0.8f, 1.0f));
mat.setFloat("Shininess", 64.0f);
```

- Set the material to the Geometry.

```
geom.setMaterial(mat);
```

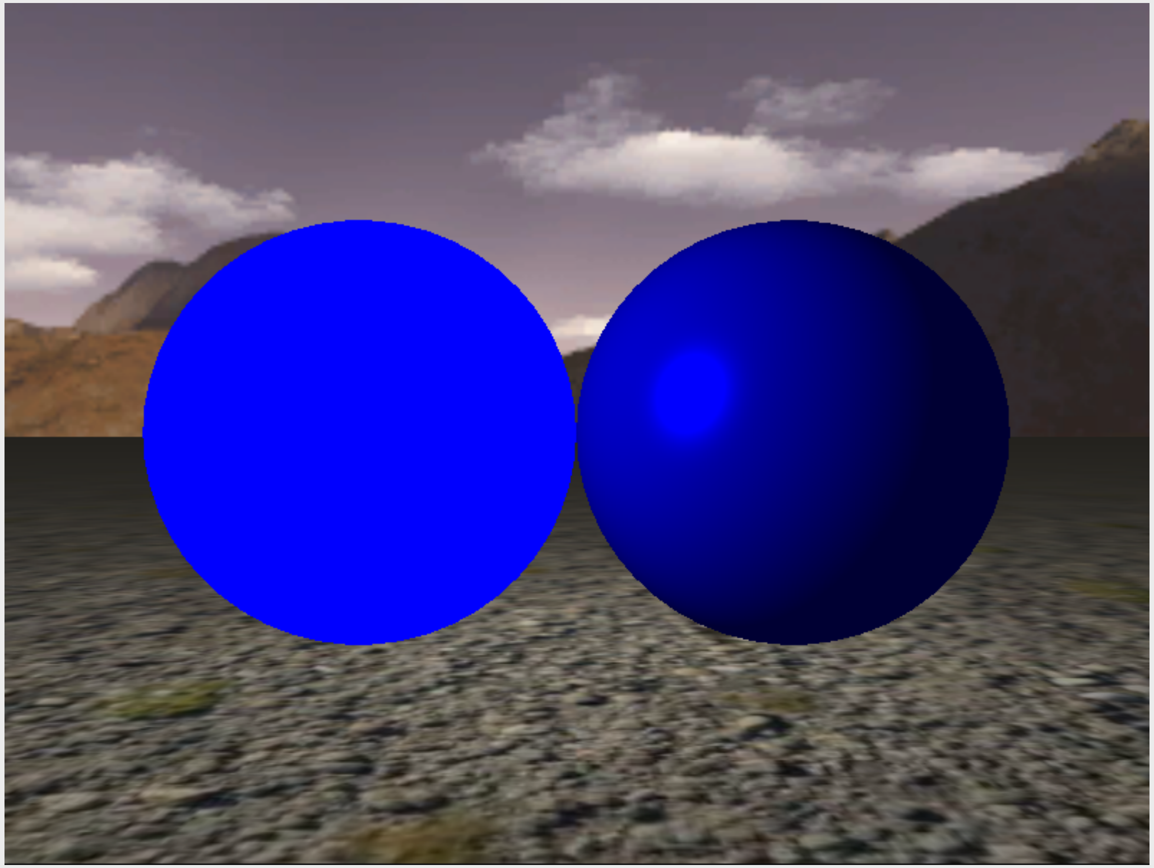
Material Attributes

- Material definitions are located in `jME3-core.jar`

```
"Common/MatDefs/*"
```

- Material editor in the jME SDK
- Overview over the Material Definition Properties on the wiki: http://jmonkeyengine.org/wiki/doku.php/jme3:advanced:materials_overview

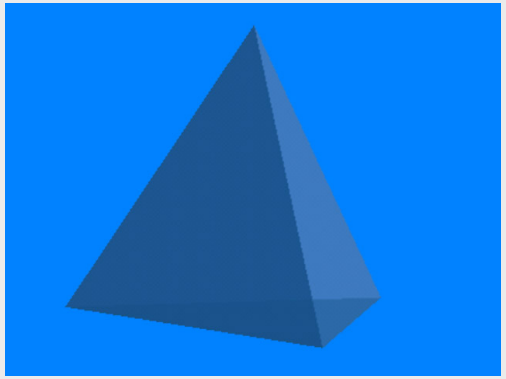
Unshaded and Lighting material example

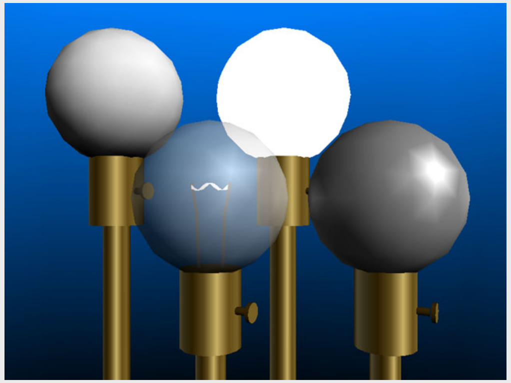


MaterialDifference.java

Transparency

- `Transparency` controls
  - The amount of transparency depends on alpha value
  - Alpa value [0.0f, 1.0f]
  - Transparency modes





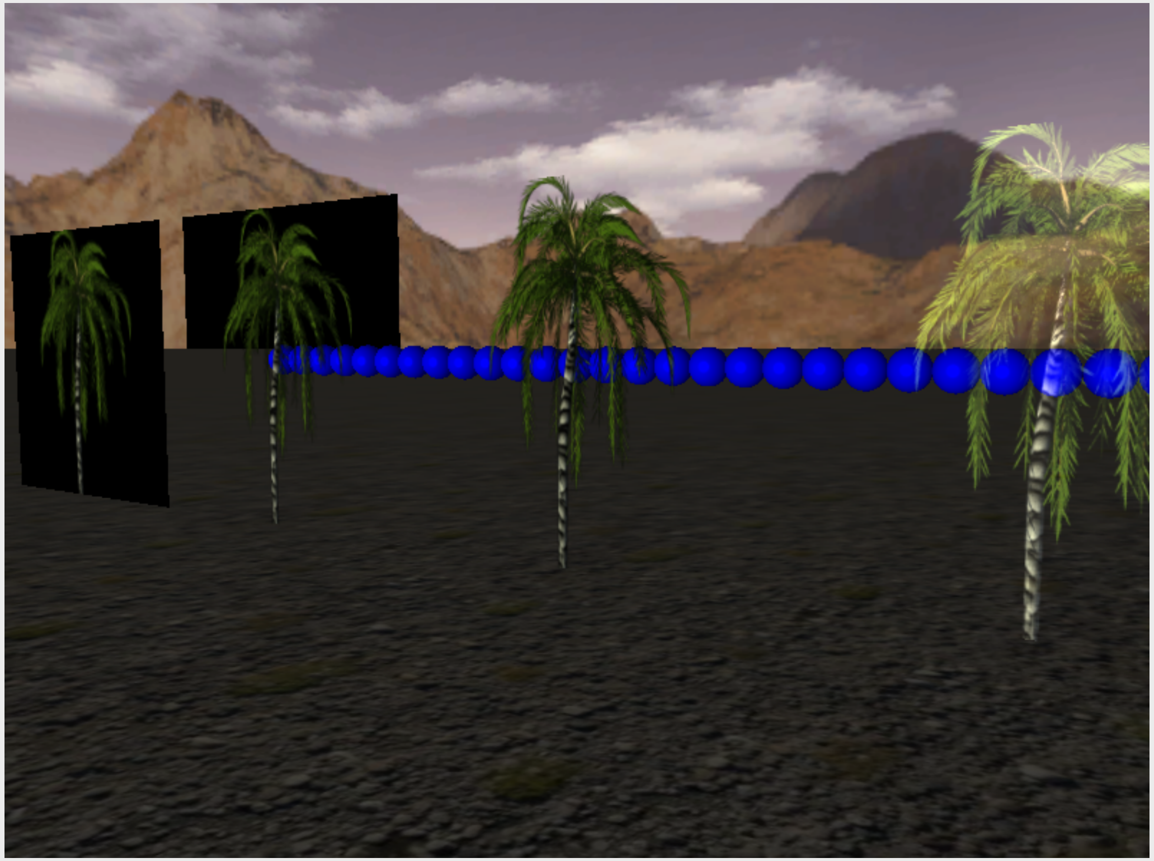
Transparency (blend) Modes

source = value from fragment shader

destination = value from framebuffer

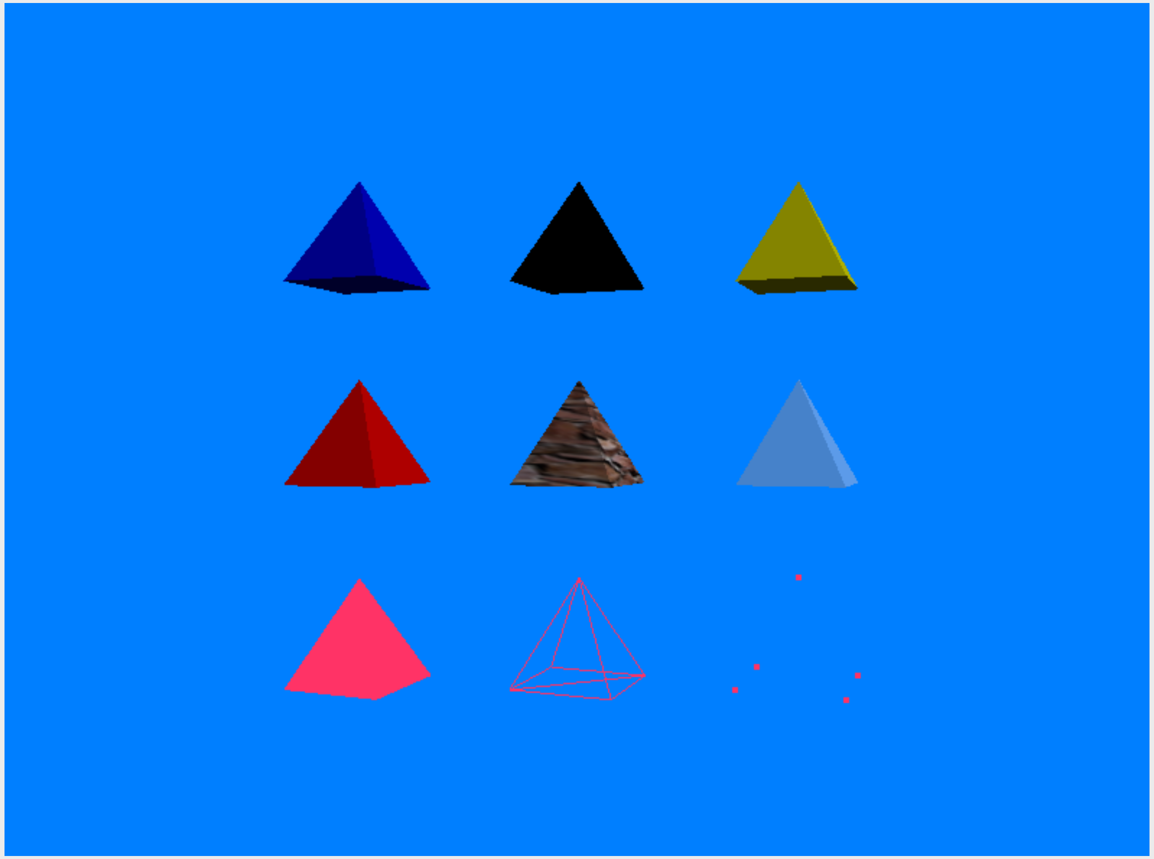
- Opaque (no blend mode)
- Alpha (Result = Source Alpha * Source Color + (1 - Source Alpha) * Dest Color)
- Additive (Result = Source Color + Destination Color)
- Alpha additive (Result = (Source Alpha * Source Color) + Dest Color)
- Modulate (Result = Source Color * Dest Color)
- ModulateX2 (Result = 2 * Source Color * Dest Color)
- PremultAlpha (Result = Source Color + (Dest Color * (1 - Source Alpha)))
- Color (Result = Source Color + (1 - Source Color) * Dest Color)

Transparency example



TransparencyExample.java

Different Materials Example



MaterialExample.java

Lights

Setting lights in a scene

Lights in jME

- jME offers 4 different light types for lighting the scene.
  - Ambient light
  - Directional light
  - Point light
  - Spot light
- Or you can write your own equation in a shader

Light methods

- There are some methods that are common for all light-types
  - setEnable(boolean OnOff), turn lights on off
  - Color, setColor
  - Volume and scope which controls which shapes that will be lit up.

Ambient Light

- General brightness/color of the objects

```
AmbientLight al = new AmbientLight();
al.setColor(ColorRGBA.White.mult(0.3f));
rootNode.addLight(al);
```

Directional Light

- Light in a direction, infinitely far away (the sun)

```
DirectionalLight sun = new DirectionalLight();
sun.setColor(ColorRGBA.White);
sun.setDirection(new Vector3f(-0.5f, -0.5f,
-0.5f).normalizeLocal());
rootNode.addLight(sun);
```

Point Light

- All directions, decreasing intensity (almost like a "light bulb")

```
PointLight lamp_light = new PointLight();
lamp_light.setColor(ColorRGBA.Yellow);
lamp_light.setRadius(4f);
lamp_light.setPosition(new Vector3f(0, 1, 0));
rootNode.addLight(lamp_light);
```

Spot Light

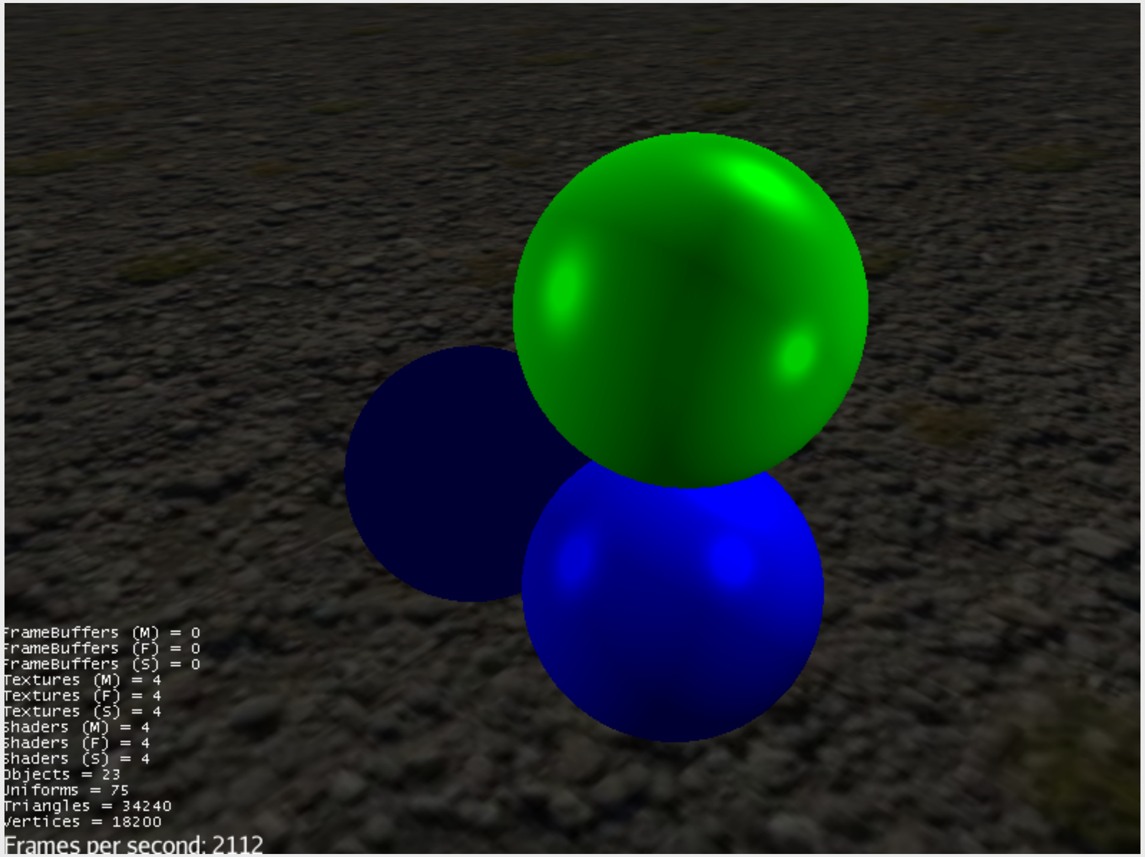
- Direction, position, and two angles (flashlight)

```
SpotLight spot = new SpotLight();
spot.setSpotRange(100f);
spot.setSpotInnerAngle(15f * FastMath.DEG_TO_RAD
spot.setSpotOuterAngle(35f * FastMath.DEG_TO_RAD);
spot.setColor(ColorRGBA.White);
spot.setPosition(cam.getLocation());
spot.setDirection(cam.getDirection());
rootNode.addLight(spot);
```

Lights and Scope

- Every Spatial has a list of lights
- The influence of lights are limited to the subgraph of the Spatial
- Add lights that should influence whole scene directly to the root
- Add lights that only influence parts at the topmost Spatial

Light example



```
FrameBuffers (M) = 0
FrameBuffers (F) = 0
FrameBuffers (S) = 0
Textures (M) = 4
Textures (F) = 4
Textures (S) = 4
Shaders (M) = 4
Shaders (F) = 4
Shaders (S) = 4
Objects = 23
Uniforms = 75
Triangles = 34240
Vertices = 18200
Frames per second: 2112
```

LightExample.java

Model Loaders

Use of loaders

Loaders

- Oficially there only exists loaders for some file formats
  - Ogre DotScene (animated objects, scenes)
  - Ogre Mesh XML
  - Wavefront OBJ (static objects, scenes)
- Other unofficial loaders exist (might not be up to date)
  - COLLADA
  - MD5
- jME want to focus officially supported loaders to only a few
- We will use Ogre DotScene

Ogre DotScene

- Standardized XML file format
- Describes a scene
  - Meshes
  - Materials
  - Lights
  - Level of detail
- Animation

Ogre DotScene

- Meshes are exported as `.mesh.xml`
- Materials as `.material`
- Animations as `.skeleton.xml`
- Scenes as `.scene`

- The .scene file "binds things together"

  For example: `Mesh <-> Material`

Converting models to Ogre DotScene

- Blender 2.62 (free) or Maya
- Import model, any format the editor supports
- Export model as Ogre DotScene
- See guide for installing and setting up Blender with export script correctly
- Why doesn’t the loaded model work?

Using the Ogre DotScene Loader

- Extracts jME spatials from the scene file
  - Geometry
  - Lights
  - Skeleton
  - Animations
- Traverse the loaded graph to access named objects and manipulate them
- Add to scene graph
- Topmost node in loaded subgraph is usually a node

“Debugging” loaded models

```
Spatial model = assetManager.loadModel("models/standing_man.scene");

model.depthFirstTraversal(new SceneGraphVisitor() {
  @Override
  public void visit(Spatial spatial) {
    if (spatial instanceof Geometry) {
      // turn off face culling.
      ((Geometry)spatial).getMaterial().getAdditionalRenderState().
                         setFaceCullMode(RenderState.FaceCullMode.Off);
    }
  }
});

rootNode.attachChild(model);
```

jME3 specific formats

- Binary 3D model or scene (.j3o)
- Optimized format
- Convert them using the jME SDK
  - (you don't have to do this)
- Use this for release builds
- Load models during development

Loader Example

LoaderExample.java