How Do We Program Graphics in 3D?

• Much like in 2D but with an extra (Z) dimension
• BUT need to worry about viewer (camera or eye) position
• Realistic 3D determined by lighting
• Ultimately must generate 2D view of 3D scene
2D Graphics Pipeline

- Turns vector-based 2D objects into pixel colours

![Diagram of 2D Graphics Pipeline]

3D Graphics Pipeline

- Turns 3D objects into screen pixels
- 3D objects (usually) defined by vertices
  - Complex object approximated by flat, triangular surfaces defined by 3 vertices
- Colour of each vertex determined
  - Intrinsic colour plus lighting effects
- Non-vertex colours determined by interpolation
  - Shading model
- Objects mapped to 2D viewing window
  - Rasterization
  - Face culling and hidden surface removal
  - Texture mapping

Computer Game Technologies, 2017
3D Graphics Libraries

- Direct X
- OpenGL
- Equivalent to Java2D in the 3D world
- OpenGL graphics pipeline

(Hardware versus Software Fig. 1-2)

Hardware versus Software

- A software 3D renderer implements 3D graphics drawing entirely in software, presenting a final pixel screen buffer to the video card
  - See e.g. DGJ
- 3D graphics video cards support DirectX and OpenGL functions in hardware
- Standard operations on vertices
  - Vectors and matrices
- GPUs are more powerful than CPUs at what they do!
  - NVIDIA GeForce GTX 1080 achieves 8800 Gflops
  - 10-20 Gflops for current CPUs
  - Moving to be more general purpose processors
  - Parallel processing
OpenGL Example

Example 1-1: Chunk of OpenGL Code

```c
#include <whateverYouNeed.h>
main() {
    InitializeAWindowPlease();
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glColor3f (1.0, 1.0, 1.0);
    glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
    glBegin(GL_POLYGON);
    glVertex3f (0.25, 0.25, 0.0);
    glVertex3f (0.75, 0.25, 0.0);
    glVertex3f (0.75, 0.75, 0.0);
    glVertex3f (0.25, 0.75, 0.0);
    glEnd();
    glFlush();
    UpdateTheWindowAndCheckForEvents();
}
```

Java 3D

- Higher level approach
- Based on the concept of a scene graph
- Specifies elements of the 3D world
  - Visible objects
  - Lighting
  - Camera
- Java 3D renderer handles the low level details of drawing a 3D scene
  - Retained mode (scene graph) versus immediate mode
- Built on top of DirectX or OpenGL
  - Java bindings available e.g. JOGL
Java 3D Scene Graph Example

```
Computer Game Technologies, 2017
```

Java 3D Components

- A virtual 3D universe
- Camera (or viewer) position in that universe
- Lights
  - As many as needed
  - Different locations and properties
- Background
- Objects in the 3D world
  - Scenery
  - Game sprites
  - Position and appearance
- Objects can share properties
  - Appearance
  - Transformations

```
Computer Game Technologies, 2017
```
A First Java 3D Example

The Scene Graph

Java 3D Example Code

```
public class HelloJava3Da extends Applet {
    public HelloJava3Da() {
        setLayout(new BorderLayout());
        GraphicsConfiguration config =
            SimpleUniverse.getPreferredConfiguration();

        Canvas3D canvas3D = new Canvas3D(config);
        add("Center", canvas3D);

        BranchGroup scene = createSceneGraph();

        // SimpleUniverse is a Convenience Utility class
        SimpleUniverse simpleU = new SimpleUniverse(canvas3D);

        // This will move the ViewPlatform back a bit so the
        // objects in the scene can be viewed.
        simpleU.getViewingPlatform().setNominalViewingTransform();

        simpleU.addBranchGraph(scene);
    } // end of HelloJava3Da (constructor)
```
Java 3D Example Code (2)

```java
public BranchGroup createSceneGraph() {
    // Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();

    objRoot.addChild(new ColorCube(0.4));

    return objRoot;
} // end of CreateSceneGraph method of HelloJava3Da

// The following allows this to be run as an application
// as well as an applet

public static void main(String[] args) {
    Frame frame = new MainFrame(new HelloJava3Da(), 256, 256);
} // end of main (method of HelloJava3Da)

} // end of class HelloJava3Da
```

A First Java 3D Example (again)

The Scene Graph

(Java 3D Tutorial Fig. 1-11)
Java 3D Example: Modification 1

```java
public BranchGroup createSceneGraph() {
    // Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();

    // rotate object has composited transformation matrix
    Transform3D rotate = new Transform3D();
    Transform3D tempRotate = new Transform3D();
    rotate.rotX(Math.PI/4.0d);
    tempRotate.rotY(Math.PI/5.0d);
    rotate.mul(tempRotate);
    TransformGroup objRotate = new TransformGroup(rotate);
    objRoot.addChild(objRotate);
    objRotate.addChild(new ColorCube(0.4));

    // Let Java 3D perform optimizations on this scene graph.
    objRoot.compile();
    return objRoot;
} // end of CreateSceneGraph method of HelloJava3D
```

Java 3D Example: Modification 1 (2)

(Java 3D Tutorial Fig. 1-14)
public BranchGroup createSceneGraph() {
    // Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();

    // rotate object has composited transformation matrix
    Transform3D rotate = new Transform3D();
    Transform3D tempRotate = new Transform3D();
    rotate.rotX(Math.PI/4.0d);
    tempRotate.rotY(Math.PI/5.0d);
    tempRotate.mul(rotate);

    TransformGroup objRotate = new TransformGroup(tempRotate);
    objRoot.addChild(objRotate);
    objRotate.addChild(new ColorCube(0.4));

    objRoot.compile();

    return objRoot;
} // end of CreateSceneGraph method of HelloJava3Dbalt

public BranchGroup createSceneGraph() {
    // Create the root of the branch graph
    BranchGroup objRoot = new BranchGroup();

    // Create the transform group node and initialize it to
    // the identity. Add it to the root of the subgraph.
    TransformGroup objSpin = new TransformGroup();
    objSpin.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objRoot.addChild(objSpin);

    // Create a simple shape leaf node, add it to
    // the scene graph.
    // ColorCube is a Convenience Utility class
    objSpin.addChild(new ColorCube(0.4));

    return objRoot;
} // end of CreateSceneGraph method of HelloJava3Dbalt
Java 3D Example: Modification 2 (2)

// Create a new Behavior object that will perform the desired
// operation on the specified transform object and add it into
// the scene graph.
Alpha rotationAlpha = new Alpha(-1, 4000);

RotationInterpolator rotator =
    new RotationInterpolator(rotationAlpha, objSpin);

// a bounding sphere specifies a region a behavior is active
// create a sphere centered at the origin with radius of 100
BoundingSphere bounds = new BoundingSphere();
rotator.setSchedulingBounds(bounds);
objSpin.addChild(rotator);

return objRoot;
} // end of CreateSceneGraph method

Java 3D Example: Modification 2 (3)

• **Capability** to change
  transform dynamically
  • ALLOW_TRANSFORM_WRITE
• Alpha object counts time
  • Loop continuously with period
    of 4 seconds
• Rotation interpolator
  behaviour linearly updates
  rotation for 360degs
• Scheduling bounds specify
  when behaviour is active

(Java 3D Tutorial Fig. 1-18)
The Simple Universe

- Utility class that provides a virtual 3D universe
- Canvas3D is the place everything is drawn to

![Diagram: The Simple Universe Diagram]

The Java 3D Rendering Loop

- The rendering loop is intrinsic to Java 3D
- Renderer starts running in an infinite loop when an instance of View becomes live in the virtual universe
  - E.g. on creation of a SimpleUniverse
- Renderer executes the following loop:

```java
while(true) {
    Process input
    If (request to exit) break
    Perform Behaviors
    Traverse the scene graph
    and render visual objects
}
Cleanup and exit
```

Figure 1-10 Conceptual Renderer Process
The End