Java2D/Java3D Graphics

Sandro Spina
Computer Graphics and Simulation Group

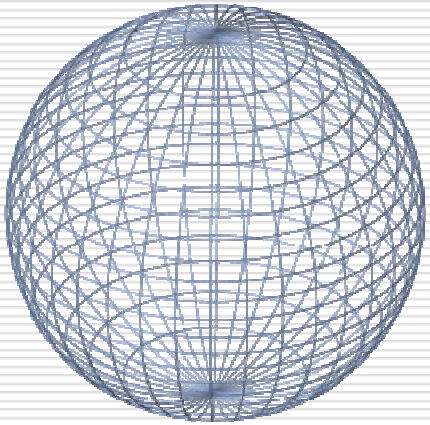
Computer Science Department
University of Malta

Abstraction in Software Engineering

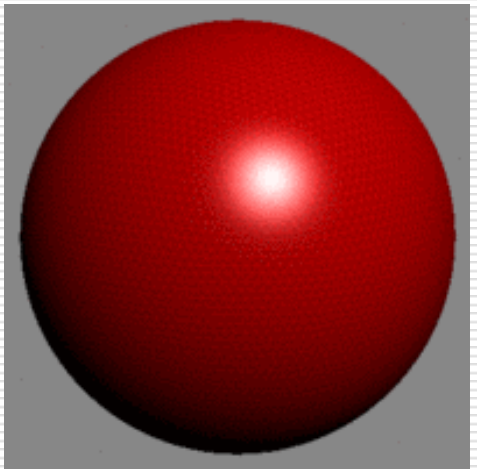
- We shall be looking at how abstraction is essential when working with Computer Graphics
- Java2D and Java3D are APIs which provide this abstraction over OpenGL/DirectX which are providing a simpler abstraction of the underlying GPU.
- Software Engineers should be capable of coming up with adequate abstractions.
- Sun provides an ideal example through it's rendering APIs.

3D Graphics Basic Elements

A Modeler : constructs virtual world models.
Eg Autodesk Maya.

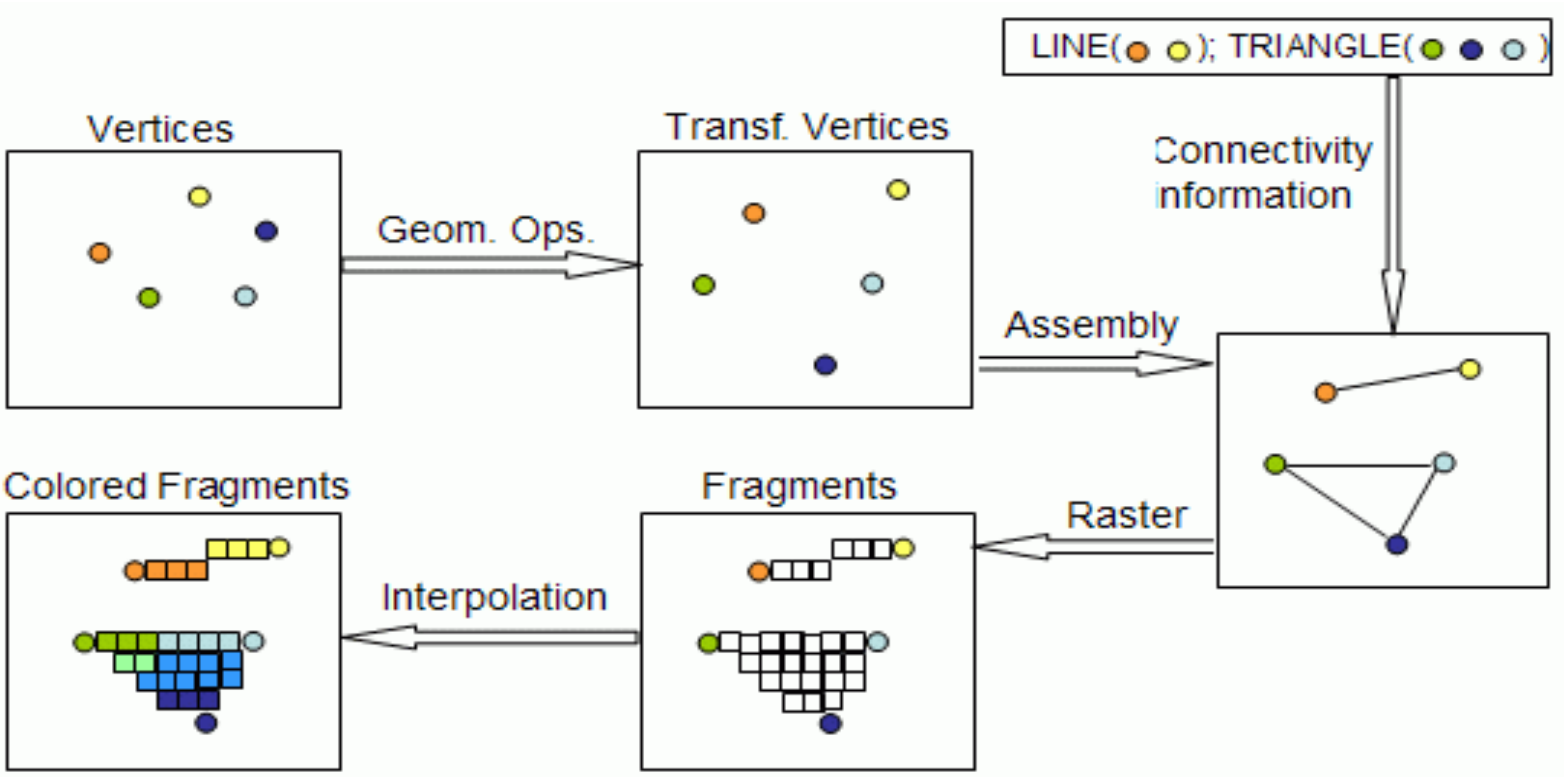


A Renderer : calculates how light interacts with the surfaces of the models in the scene.



The (simplified) Graphics Pipeline

The mechanism that takes a scene description and converts it into something we can see



DirectX and OpenGL

- DirectX and OpenGL are two popular (competing) graphics pipeline models which are today accepted as industry standards.
- DirectX is a proprietary API developed by Microsoft. Current release is DirectX10.1 and is predominantly used in the .NET framework.
- OpenGL is an open standard API. OpenGL operates on a much wider range of hardware platforms and software environments. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl, Java, etc …

Java Graphics APIs – with different levels of abstraction

- Java2D + Java3D : a high-level 3D graphics API. Uses OpenGL internally (or alternatively Direct3D on Windows) . Provides a complete framework (helper classes, etc.) in which to develop 3D applications.

- JOGL : a low-level Java wrapper of the OpenGL graphics API. Makes use of JNI. This is exactly what you'll get (as in function calls) if one is coding in c/c++.

Java2D Rendering Process

- Involves the following steps :
  - Construct the 2D objects
  - Apply transformations to the objects
  - Apply colour and other rendering properties
  - Render the scene on a graphics device

(some of the) Java2D Classes

- *Graphics2D* (abstract class) – used to access the rendering engine. Usually retrieved when using the paintComponent(Graphics g) method. Methods include setColor(..), drawLine(..), draw(Shape s), etc…

- Shape Interface – a geometric object can be rendered by Graphics2D if it implements Shape. Java2D provides a number of built-in shapes including Arc2D, Ellipse2D, Rectangle2D, Line2D …
- Eg. Line2D line = new Line2D(x1,y1,x2,y2)

Java2D Program Structure

- Rendering is event based.
- In Java2D everything is drawn in the paintComponent(Graphics g) method which is invoked when repaint is called.
- Threads can be used as in the Rain example …
- Alternatively the Java2D Timer class can be used as we'll see in the Clock2D example …

A 2D Clock – An example

- Switch to Eclipse …
  - Rain uses the Thread class
  - Clock2D uses the Java2D Timer class

The 3D Rendering Process

- Unlike 2D, rendering a 3D scene is a much more complex process.
- The 3D viewing process typically involves a projective transformation that maps a 3D scene to a 2D plane.
- A number of elements need to be processed including: geometries, materials, lights, shading models, etc…

- Matrix Transformations – Rotation, scaling, shearing, translation.

The Java3D Package

- Java3D caters for the needs described in the previous slide.
- javax.media.j3d.*; -- Main Package
- com.sun.j3d.*; --Utility Classes

- Canvas3D
- Shape3D
- Transform3D

Primitive Geometry (i)

- The geometries of complex objects are built from sets of simple objects (primitives) such as triangles.
- Point* classes : geometric points
- Color* classes : color representations
- Vector* classes : geometric vectors

Primitive Geometry (ii)

- Dodecahedron
  - 20 vertices and 12 pentagon faces
  - First define the vertices using a Point3d[] array
  - Then define the indices which compose the faces. Size of array is equal to 12 * 5 (obviously there are shared vertices
  - Then define the stripCounts
    - Int[] stripCounts = {5,5,5,5,5,5,5,5,5,5,5,5}
  - Check example …

Transformations

- Javax.vecmath package contains matrix classes representing 3x3, 4x4 and general matrices.

- Transform3D class represents geometric transformations which internally maintains a 4x4 double matrix for the transform.

- Provides methods for translation, scaling, reflection and rotation of the matrix. Rotation is notoriously the most complex since a general 3D rotation has an axis of rotation that can be any line in the virtual space.

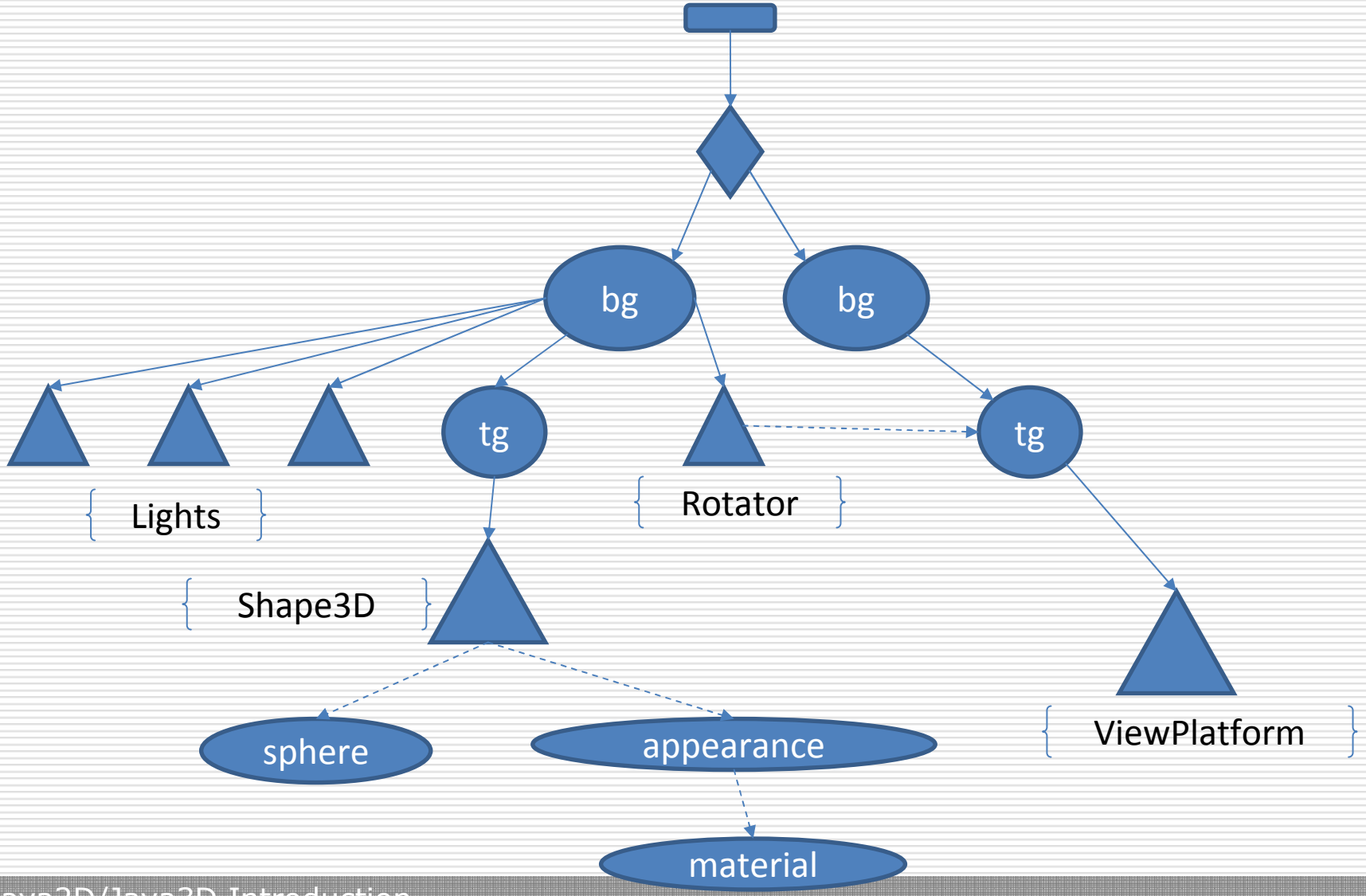
Java 3D Scene Graphs (i)

- Used to organise the various elements in the 3D rendering.
- A scene graph is essentially a virtual universe which describes the relations between its different elements.
- The scene graph enables programmers to specify complex graphics structures and actions in a uniform manner.
- Formally, it is a tree-like structure known as DAG(directed acyclic graph).

Java 3D Scene Graphs (ii)

- The DAG is composed of Node (classes) with NodeComponent (classes)
  - VirtualUniverse and Locale
  - GroupNodes
    - BranchGroup (root a a branch of a scene graph)
    - SharedGroup (used to explicitly share branches)
    - TransformGroup (represents geometric transformations applied to all children)
    - Etc…
  - Leaf Nodes
    - Behaviour
    - Light
    - Shape3D
    - Sound
    - Background
    - Etc…
- NodeComponent eg. Appearance, Texture, ColoringAttributes, etc…

Java 3D Scene Graphs (view rotation)



The structure of a 3D Program

- To write a Java3D program is essentially to assemble a scene graph!

- The scene graph is a complete specification of all the graphics objects and their attributes. It is also linked to the AWT components for displaying rendered images.

Transformations in Scene Graphs

- A TransformGroup object defines a scene-graph group node that represents a specific transformation (Transform3D object).

- The transformation defined by the TransformGroup node is applied to all of its child nodes.

Lighting (Classes)

- AmbientLight() – uniform in all directions and locations
- DirectionalLight() – emits parallel light rays (from infinity)
- PointLight() – has a specific location and emits light rays in all directions.
- SpotLight() – emits light rays in a cone-shaped region.
- All can emit different colours - check example code.

Texturing

- Texture mapping is a method that utilizes images in graphics rendering.
- It can provide a great deal of model details with efficiency.
- Java3D includes classes (NodeComponents) to represent textures which are applied to Shape3D objects.

Behaviour (Abstract Class)

- Java3D provides a general unified approach to implement both animation and interaction.

- Abstract Methods
  - initialize() : invoked when a Behaviour object becomes live
  - processStimulus() : invoked by Java3D under certain wakeup conditions (WakeupCondition class hierarchy, eg. WakeOnElapsedTime(long ms))

- void wakeupOn(WakeupCondition wakeup)

Animations in Scene Graphs

- To produce an animated effect, the rendered scene must change dynamically with time.
- Java3D provides support for incorporation of animation into a scene graph through the Behaviour class.
- More specifically through a family of behaviours known as Interpolators.

The Alpha and Interpolator classes

- An Alpha object defines a function of time that produces values between 0.0 and 1.0.
- The Alpha objects provide inputs to the animation class known as the Interpolator.
- A Java3D Alpha object includes the following parameters:
  - LoopCount : -1 indicates an infinte number of loops
  - increasingAlphaDuration : The time in milliseconds for the alpha value to increase from 0.0 to 1.0
  - etc…

Conclusions

- Levels of abstraction
- Infer ease of use – user friendliness
- But also need to be complete !!

- This was a quick introduction to Java{2|3}D.
- We've mentioned some of the core classes
- But … various others are included in the API

- If you are interested in Java and Graphics I would recommend you also check JOGL.