Graphics Primitives

- 3-dimensional point class
  - `Point3d`

- 3-dimensional (directional) vector class
  - `Vector3d`

- Class of 3-dimensional line segments
  - `LineArray`
Modeling of 3D Objects

- Java3D model of a 3D shape

```
Shape3D
GeometryArray
Appearance
```

- Subclass e.g. *ColorCube*
### Modeling of Surfaces

- Abstract Java3D class for surface geometry

<table>
<thead>
<tr>
<th>GeometryArray</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point3d [] Points</td>
</tr>
<tr>
<td>Vector3d [] Normals</td>
</tr>
<tr>
<td>Color3d [] Colors</td>
</tr>
</tbody>
</table>

- Non-abstract subclasses
  - `TriangleArray`
  - `TriangleStripArray`
  - `QuadArray`
Modeling of Appearance Properties

- Java3D class for appearance properties

<table>
<thead>
<tr>
<th>Appearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Material</td>
</tr>
<tr>
<td>ColoringAttributes</td>
</tr>
<tr>
<td>TransparencyAttributes</td>
</tr>
<tr>
<td>Texture</td>
</tr>
</tbody>
</table>

- **Material** attribute describes reflection properties
- **ColoringAttributes** describe shading properties
Modeling of Light and Reflection

- **Face normals** are needed for
  - visibility
  - reflection properties of two types:
    - diffusive: ambient light reflection, emissive light
    - specular: reflection of directed light

- **Java3D light sources**
  - diffusive light: *AmbientLight*, emissive light from radiating object faces (*Material* attribute)
  - directed light: *DirectionalLight*, *PointLight*
View Port Projections 1

- **Projections** of 3D space onto a 2D plane, e.g.
  - a drawing canvas
  - a film in a camera

are based on straight propagation of light.

- Java3D unifies the treatment of
  - parallel projection
  - perspective projection

using the `View.setProjectionPolicy()` method
View Port Projections 2

- **Parallel projection**
  - Sun rays throw the shadow of objects onto a plane.
  - Coordinate system:
    - projection plane as \((x,y)\)-plane
    - projection direction \(R = (s_1, s_2, 1)\)

- **Perspective projection**
  - Rays of light project space through a camera lens onto a plane
  - Coordinate system:
    - projection plane as \((x,y)\)-plane
    - center of projection on z-axis
Coordinate Systems

- A Java3D scene contains
  - a set of **world coordinates** provided by a `Locale` object
  - a number of **local coordinate systems** provided by a `TransformGroup` object containing a `Transform3D` object.

- Homogeneous coordinates $X=(1,x,y,z)$ allow unified treatment of 3D transformations in a two-step procedure:
  - Multiply $X$ by 4x4 transformation matrix $T$ to obtain $Y$.
  - Scale $Y$ such that its first component equals one.
3D Transformations

- Transform3D class: unified treatment of
  - zoom scaling
    \[\text{setScale}(\text{double})\]
  - translation
    \[\text{setTranslation}(\text{Vector3d})\]
  - rotation
    \[\text{setRotation}(\text{Matrix3d})\]
3D Scenes

- 3D scenes consist of
  - geometrical objects
  - lights
  - cameras (viewpoints).

- Java3D representation uses a **scene graph**:  
  - The scene graph is an unidirectional tree structure rooted in `VirtualUniverse`.
  - Each `BranchGroup` object is a container of subtrees on the scene graph.
  - `TransformGroup` objects provide local coordinate systems.
  - `Behavior` objects control `TransformGroup` behavior.
Sample Scene Graph

VirtualUniverse

BG: BranchGroup
TG: TransformGroup

SimpleUniverse

Locale

View

Behaviour

light branch

object branch

Light1

LightN

Graph1

GraphN

BG

BG

BG

TG

TG

TG

ViewPlatform
Attributes of objects in the scene graph can be accessed only if

- `ALLOW_Attribute_WRITE` or
- `ALLOW_Attribute_READ`

are set before connecting the object to the scene graph.

Optimization of a whole branch of the scene graph at run-time is done by the `BranchGroup.compile()` method.
To connect Java3D to a window:

- Create a **Canvas3D** object.
- Create a **SimpleUniverse** object. *(Locale object attached to VirtualUniverse and view branch)*
- Link them with the **SimpleUniverse.setCanvas3D()** method.
- Add **Canvas3D** to output window.