## BUILDING 3D SHAPES

1. A `Shape3D` leaf node builds a 3D object with:
   - Geometry:
     - The form or structure of a shape
   - Appearance:
     - The coloration, transparency, and shading of a shape
2. The `Shape3D` class extends the `Leaf` class

| *Class Hierarchy* |
|---|
| **java.lang.Object**<br>└ **javax.media.j3d.SceneGraphObject**<br>└ **javax.media.j3d.Node**<br>└ **javax.media.j3d.Leaf**<br>└ **javax.media.j3d.Shape3D** |



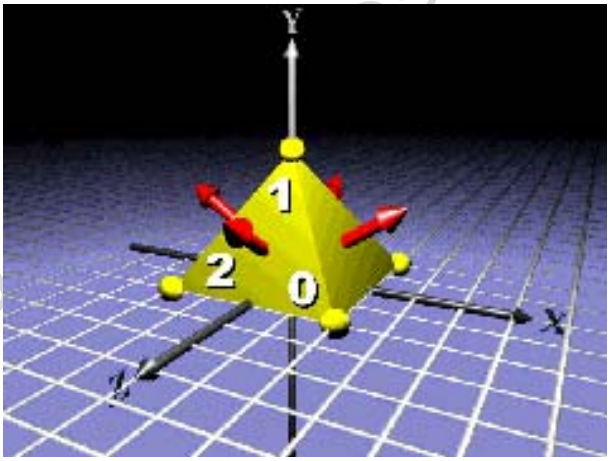


3. `Shape3D` methods to set geometry/appearance attributes

| *Method* |
|---|
| **Shape3D( )** |
| **Shape3D( Geometry geometry, Appearance appearance )** |
| **void setGeometry( Geometry geometry )** |
| **void setAppearance( Appearance appearance )** |

4. Building geometry using coordinates
   - Using a right-handed coordinate system
     - The right-handed coordinate system
       - X = left-to-right
       - Y = bottom-to-top
       - Z = back-to-front
     - Distances are conventionally in meters.

- Building geometry is like a 3D connect-the-dots game
  - Place "dots" at 3D coordinates
  - Connect-the-dots to form 3D shapes
- Pyramid: an example








- Using coordinate order
  - Polygons have a front and back:
    - By default, only the front side of a polygon is rendered
    - A polygons winding order determines which side is the front
  - Use the right-hand rule:
    - Curl your right-hand fingers around the polygon perimeter in the order vertices are given (counter-clockwise)
    - Your thumb sticks out the front of the polygon

5. GeometryArray class hierarchy

- All geometry types are derived from Geometry

*Class Hierarchy*

```
java.lang.Object
  └ javax.media.j3d.SceneGraphObject
    └ javax.media.j3d.NodeComponent
      └ javax.media.j3d.Geometry
        └ javax.media.j3d.GeometryArray
```

6. Building different types of geometry

- There are 14 different geometry array types grouped into:
  - Simple geometry:
    - `PointArray`, `LineArray`, `TriangleArray`, and `QuadArray`

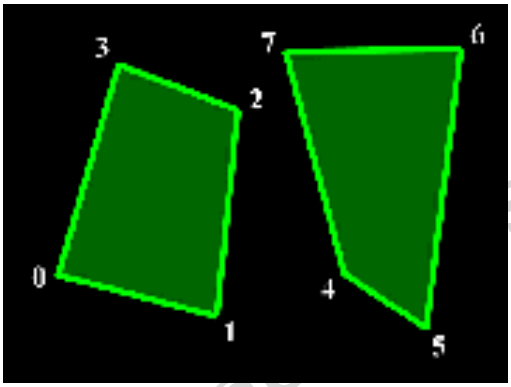


  - Strip geometry:
    - `LineStripArray`, `TriangleStripArray`, and `TriangleFanArray`
  - Indexed simple geometry:
    - `IndexedPointArray`, `IndexedLineArray`, `IndexedTriangleArray`, and `IndexedQuadArray`
  - Indexed stripped geometry:
    - `IndexedLineStripArray`, `IndexedTriangleStripArray`, and `IndexedTriangleFanArray`

7. Building a QuadArray

- A `QuadArray` builds quadrilaterals
  - Between each quadruple of vertices


XYUAN@UWINDSOR.CA




- Rendering may be controlled by shape appearance attributes

• Steps for construction

(1) Create lists of 3D coordinates and normals for the vertices





(2) Create a QuadArray and set the vertex coordinates and normals

(3) Assemble the shape

"`Shape3D myShape = new Shape3D( myQuads, myAppear );`"

## TRANSFORMING SHAPES

1. The default location of 3D shapes
   - By default, all shapes are built within a shared *world coordinate system*
   - A `TransformGroup` builds a new coordinate system for its children, relative to its parent
     - *Translate* to change relative position
     - *Rotate* to change relative orientation
     - *Scale* to change relative size
     - Use in combination
   - Transforms can be arbitrarily nested to include one `TransformGroup` within another





   - A 3D transform must be *affine*
     - No perspective-like homogeneous division, such as for hyperbolic spaces
   - A 3D transform must be congruent if used in a `TransformGroup` above a `ViewPlatform`
     - No non-uniform scaling of the viewpoint

2. 3D transforms are internally represented as a 4×4 matrix
   - A 3D point $\mathbf{P}$ is transformed to $\mathbf{P}'$ by $\tau$:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

- Translation along three axes

$$\begin{aligned}\mathbf{T}_{xyz}(\sigma,\delta,\zeta) &= \mathbf{T}_x(\sigma)\mathbf{T}_y(\delta)\mathbf{T}_z(\zeta)\\ &= \begin{bmatrix} 1 & 0 & 0 & \sigma \\ 0 & 1 & 0 & \delta \\ 0 & 0 & 1 & \zeta \\ 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

- Scaling on three axes

$$\begin{aligned}\mathbf{S}_{xyz}(s_1,s_2,s_3) &= \mathbf{S}_x(s_1)\mathbf{S}_y(s_2)\mathbf{S}_z(s_3)\\ &= \begin{bmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_2 & 0 & 0 \\ 0 & 0 & s_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}\end{aligned}$$

  - Uniform scaling: $s_1 = s_2 = s_3$





- Rotation around three axes or a unit vector
  - rotate around $\mathbf{x}$ for $\alpha$

$$\mathbf{R}_{\mathbf{x}}(\alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - rotate around $\mathbf{y}$ for $\beta$

$$\mathbf{R}_{\mathbf{y}}(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- rotate around $\mathbf{z}$ for $\gamma$

$$\mathbf{R_z}(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- rotate around a *unit* vector $\vec{\mathbf{V}}_k$ for $\theta$

$$\mathbf{R}(\tilde{\mathbf{V}}_\mathbf{k}, \theta) = \begin{bmatrix} k_x k_x v\theta + c\theta & k_x k_y v\theta - k_z s\theta & k_x k_z v\theta + k_y s\theta & 0 \\ k_x k_y v\theta + k_z s\theta & k_y k_y v\theta + c\theta & k_y k_z v\theta - k_x s\theta & 0 \\ k_x k_z v\theta - k_y s\theta & k_y k_z v\theta + k_x s\theta & k_z k_z v\theta + c\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$





where,

$$\begin{aligned} c\theta &= \cos\theta \\ s\theta &= \sin\theta \\ v\theta &= 1 - \cos\theta \end{aligned}$$

and

$$\vec{\mathbf{V}}_k = [k_x \;\; k_y \;\; k_z], \qquad \sqrt{k_x^2 + k_y^2 + k_z^2} = 1$$

3. An example of `TransformGroup`

(1) Build a shape, as before

```
Shape3D myShape = new Shape3D( myGeom, myAppear );
```

(2) Create a 3D transform for a Z-axis rotation by 30 degrees (0.52 radians)

```
Transform3D myTrans3D = new Transform3D( );
myTrans3D.rotZ( 0.52 ); // 30 degrees
```

(3) Create a transform group, set the transform, and add the shape

```
TransformGroup myGroup = new TransformGroup( );
myGroup.setTransform( myTrans3D );
myGroup.addChild( myShape );
```