CS 424/524
U1: Undergraduates Team 1
Jennifer Hill, Ryan Kane, Dorothy Kirlew,
Donald Shaner, Sean Weber

JAVA 3D MANIPULATOR PROGRAMMER'S GUIDE

Version 1.0 December 8, 2011

Java Files Included in the 3D Manipulator Package

S H A P E S

- aCylinder.java: Cylinder class
- aSphere.java: Sphere class
- HexagonalPrism.java: Hexagonal Prism class
- Pyramid.java: Pyramid class
- RectangularPrism.java: Rectangular Prism class
- TriangularPrism.java: Triangular Prism class

G U I P A N E L

- GUI_3D.java: Main GUI class
- CurrentShapesPanel.java - Current shapes panel
- ResizePanel.java - Resize panel
- RotatePanel.java - Rotate panel
- AestheticsPanel.java - Aesthetics panel

M I S C E L L A N E O U S

- SwingTest.java: Creates the canvas and viewing platform
- Logger.java: Contains the logger's read and write functionality
- Colors.java: Designates colors used by the Aesthetics panel
- CustomMouseTranslate.java: Allows the canvas panning

Adding New Shapes

Each shape supported by the 3D Manipulator program is contained in its own specific class file that extends the Shape3D class provided by the Java3D API.

Polyhedral shapes are constructed by declaring a set of Point3f objects to represent each of the shape's vertices. A GeometryArray is created in order to construct each of the shape's faces, and each face is built using the coordinates of the declared vertices. Each shape should be centered around the origin with an initial height, width, and depth of 2 units. Every face of the shape should have its color, unique from the other face colors of that shape.

A variable should be created in *SwingTest.java* called `[shape]Count` which will keep track of how many of that type of shape have been created. This variable should be initiated with a value of 0 and should be incremented by 1 every time a new copy of that shape is created on the canvas. This count variable is concatenated onto the end of the user data string for that shape so that each copy of that shape will be numbered, beginning with 0.

Every shape node should be contained within its own individual TransformGroup called `createRotator()`, which is responsible for creating the rotation interpolator that allows the shape to be manipulated. A second TransformGroup, `mouseTG()`, is also contained within `createRotator()` and holds an object of type `CustomMouseTranslate`, the class which allows every shape to be panned around the canvas.

Polyhedral shapes (i.e.: shapes that have vertices) should also contain a method which establishes the "wireframe" version of the shape. This method should be named `[shape]Edges()` is attached to the colored faces of the shape and is included in the `createRotator()` TransformGroup. Thus, all manipulations made to the shape (scaling, rotating, translating, etc.) are applied to both the solid faces and to the edges simultaneously, making the faces and edges appear to be one single shape.

SwingTest.java should then include a method called `create[Shape]()` to create the new shape. This method should create a new BranchGroup to contain the shape and all of its TransformGroups. This BranchGroup should be declared as detachable so that the shape can be deleted from the canvas. The method should increment the variable `[shape]Count` by 1 every time it is called.

See *Appendix A* for annotated snippets of an example shape class.

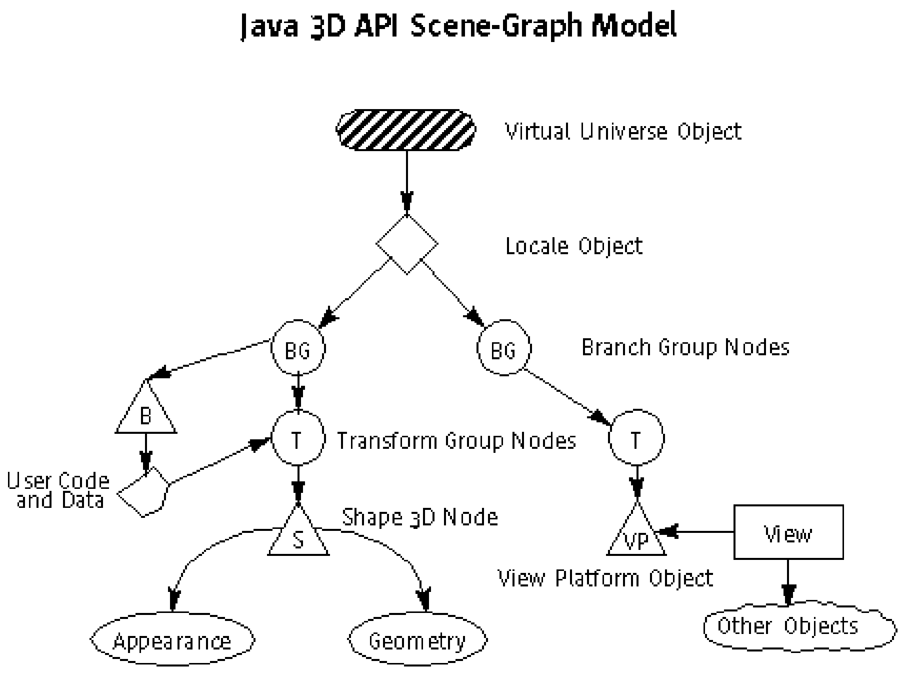
Adding New Functionality

The *GUI_3D.java* file contains all of the information for the graphical user interface. The interface is independent of the actual rendering program; a developer can, if desired, create their own interface to accompany the renderer. All mouse and mouse motion listener actions should be implemented in the user interface (*GUI_3D*).

SwingTest.java sets up the canvas and the viewing dimensions, and includes all of the shape creation methods. Any functionality that manipulates the canvas or its objects in any way should be set within *SwingTest* (and then, if desired, called in *GUI_3D* to allow for button functionality).

In *SwingTest*, the Node variable `shapeClicked` is updated every time a shape is selected by the user with a mouse click. Any manipulations to be done to the user's selected shape should be applied to this `shapeClicked` variable. All functionality pertaining to the aesthetics of the shape, such as the size and colors, must be initially declared within the individual shape's class.

Every shape and all features and functions that belong to it are contained within that shape's BranchGroup. Any functionality that seeks to transform the shape in some way should be included in a TransformGroup which should then be added to that shape's BranchGroup. Multiple TransformGroups should be contained within one primary parent TransformGroup object, which should include the shape and shapeEdge nodes as children. And each shape node is the parent of both an Appearance object and a Geometry object. A diagram of the structure of the Java3D scene graph is included below.



Appendix A: Sample Shape Class

```
public class Pyramid extends Shape3D {
      //private variables...

      /**** vertices ****/
      Point3f frontL = new Point3f(-1.0f, -1.0f, 1.0f);
      //...

    public Pyramid() {
      pyramidGeometry = new TriangleArray(18, TriangleArray.COORDINATES |
                                                              COLOR_3);

      /**** creates faces from vertices ****/
      face(pyramidGeometry, 0, frontR, top, frontL, Colors.ORANGE);
      //continue for remaining faces...

      this.setGeometry(pyramidGeometry);  //copies the geometry to the shape
      this.setAppearance(new Appearance());
      this.setUserData( "Pyramid".concat(Integer.toString(pyramidCount)) );

      //increment shape count
    }

      /**** creates all the faces ****/
    private void face(TriangleArray pyramidGeometry, int index, Point3f
      coordinate1, Point3f coordinate2, Point3f coordinate3, Color3f color) {
      pyramidGeometry.setCoordinate(index, vertexCoord);

      //set coordinates for index+1 and index+2...

      pyramidGeometry.setColor(index, color);

      //set colors for index+1 and index+2...

    }

      /**** creates all the edges ****/
    private void edge(TriangleArray pyramidEdgeGeometry, int index, Point3f
      coordinate1, Point3f coordinate2, Point3f coordinate3, Color3f color) {
      pyramidEdgeGeometry.setCoordinate(index, vertexCoord);
      //set coordinates for index+1 and index+2...
      pyramidEdgeGeometry.setColor(index, color);
      //set colors for index+1 and index+2...
    }


      /**** rotator TransformGroup ****/
      TransformGroup createRotator() {
      Transform3D yAxis = new Transform3D();

      TransformGroup spin = new TransformGroup(yAxis);
```

```
  spin.addChild(this); //add pyramid shape to the spin TG
  spin.addChild(pyramidEdges());

  rotationAlpha = new Alpha
              (0, Alpha.INCREASING_ENABLE, 0, 0, 4000, 0, 0, 0, 0, 0);
  rotator = new RotationInterpolator
              (rotationAlpha, spin, yAxis, 0.0f, (float) Math.PI*2.0f);

  spin.addChild(rotator);  //adds rotation interpolator

  /**** creates TransformGroup for canvas panning ****/
  TransformGroup mouseTG = new TransformGroup();
  myMouseTranslate = new CustomMouseTranslate();
  mouseTG.addChild(myMouseTranslate);
}
```