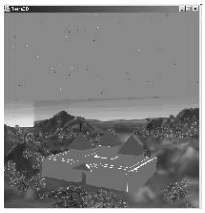
Computer Engineering

240-302, Computer Engineering Lab IV (Software)

Introduction to Java 3D (2)

Dr. Andrew Davison
Room 101, CoE
dandrew@ratree.psu.ac.th



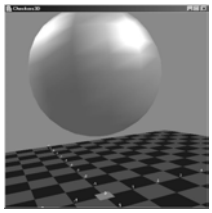
Computer Engineering

Contents

- 1. Checkers3D Again
- 2. The Floor
- 3. Viewer Positioning and Movement
- 4. Loader Classes
- 5. Behavior Objects
- 6. Animation
- 7. Textures
- 8. Sound

Computer Engineering

1. Checkers3D Again



- The scene consists of
  - a dark green and blue tiled surface (and red center)
  - labels along the X and Z axes
  - a blue background
  - a floating sphere lit from two different directions
  - the user (viewer) can move through the scene by moving the mouse

Computer Engineering

Scene Graph for Checkers3D

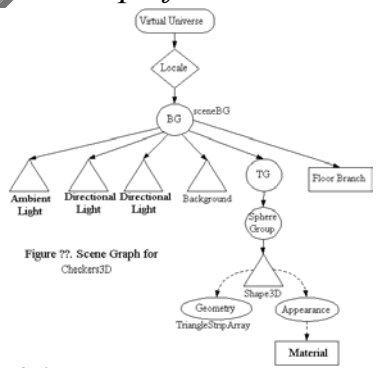


Figure ??. Scene Graph for Checkers3D

view branch not shown

Computer Engineering

WrapChecker3D Constructor

```
private SimpleUniverse su;
private BranchGroup sceneBG;  // for content branch
private BoundingSphere bounds;
        :

public WrapCheckers3D()
{ setLayout( new BorderLayout() );
  setOpaque( false );
  setPreferredSize( new Dimension(PWIDTH, PHEIGHT));

  GraphicsConfiguration config =
        SimpleUniverse.getPreferredConfiguration();
  Canvas3D canvas3D = new Canvas3D(config);
  add("Center", canvas3D);
  canvas3D.setFocusable(true);
  canvas3D.requestFocus();

  su = new SimpleUniverse(canvas3D);
        :
```


Computer Engineering

```
createSceneGraph();
initUserPosition();      // set user's viewpoint

orbitControls(canvas3D);
      // controls for moving the viewpoint

su.addBranchGraph( sceneBG );

} // end of WrapCheckers3D()
```

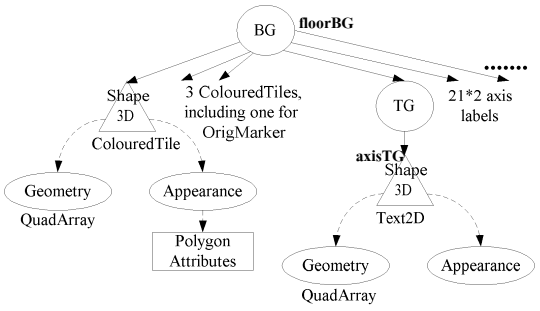
createSceneGraph()

```
private void createSceneGraph()
// initilise the scene below sceneBG
{
  sceneBG = new BranchGroup();
  bounds = new BoundingSphere(new Point3d(0,0,0),
                                    BOUNDSIZE);

  lightScene();         // add the light
  addBackground();      // add the sky
  sceneBG.addChild( new CheckerFloor().getBG() );
        // add the BranchGroup for the floor

  floatingSphere();     // add the floating sphere

  sceneBG.compile();   // fix the scene
} // end of createSceneGraph()
```


2. The Floor

- The floor is made of tiles created with the our `ColouredTiles` class, and axis labels made with the `Text2D` utility class.

Floor Branch of the Scene Graph



CheckerFloor Constructor

```
// constants for various colours
   :
private BranchGroup floorBG;


public CheckerFloor()
// create tiles, add origin marker,
// then the axes labels
{
  ArrayList blueCoords = new ArrayList()
  ArrayList greenCoords = new ArrayList();
  floorBG = new BranchGroup();
            :
```

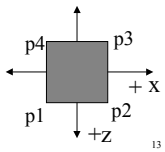


```
boolean isBlue;
for(int z = -FLOOR_LEN/2;
          z <= (FLOOR_LEN/2)-1; z++) {
  isBlue = (z%2 == 0)? true : false;
     // set colour for new row
  for(int x = -FLOOR_LEN/2;
           x <= (FLOOR_LEN/2)-1; x++) {
    if (isBlue)
      createCoords(x, z, blueCoords);
    else
      createCoords(x, z, greenCoords);
    isBlue = !isBlue;
  }
}
     :
```



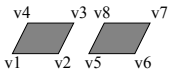
```
  floorBG.addChild(
      new ColouredTiles(blueCoords, blue) );
  floorBG.addChild(
      new ColouredTiles(greenCoords, green) );

  addOriginMarker();
  labelAxes();
}  // end of CheckerFloor()


public BranchGroup getBG()
{  return floorBG;  }
```

```
private void createCoords(int x, int z,
                          ArrayList coords)
// Coords for a single blue or green square,
// its left hand corner at (x,0,z)
{
  // points created in counter-clockwise order
  Point3f p1 = new Point3f(x, 0.0f, z+1.0f);
  Point3f p2 = new Point3f(x+1.0f,0.0f,z+1.0f);
  Point3f p3 = new Point3f(x+1.0f, 0.0f, z);
  Point3f p4 = new Point3f(x, 0.0f, z);

  coords.add(p1); coords.add(p2);
  coords.add(p3); coords.add(p4);
}  // end of createCoords()
```



ColouredTiles Class

- The `ColouredTiles` class extends `Shape3D`, and defines the geometry and appearance of tiles with the same colour.
- The geometry uses a `QuadArray` to represent the tiles as a series of quadlilaterals.



QuadArray Creation

- The constructor is:

```
QuadArray(int vertexCount, int vertexFormat);
```

- In `ColouredTiles`, the `QuadArray` plane is created using:

```
plane = new QuadArray( coords.size(),
                GeometryArray.COORDINATES |
                 GeometryArray.COLOR_3 );
```


Filling the QuadArray

```
// coordinate data
int numPoints = coords.size();
Point3f[] points = new Point3f[numPoints];
coords.toArray( points );
        // ArrayList-->array
plane.setCoordinates(0, points);

// colour data
Color3f cols[] = new Color3f[numPoints];
for(int i=0; i < numPoints; i++)
  cols[i] = col;
plane.setColors(0, cols);
```


Issues

- Counter-clockwise specification of the vertices for each quad
  - makes the top of the quad its 'front'
- Ensure that each quad is a convex, planar polygon.
- Normals or no normals?

Unreflecting Colour

- You can specify a shape's colour in three ways:
  - in the shape's material
    - when the scene is illuminated
  - in the shape's colouring attributes
    - used when the shape is unreflecting
  - in the vertices of the shape's geometry
    - also for unreflecting shapes (used here)

The Axes

- Each axis value is a `Text2D` object, which specifies the string, colour, font, point size, and font style:

```
Text2D message =
  new Text2D("...", white, "SansSerif",
              36, Font.BOLD );
       // 36 point bold Sans Serif
```

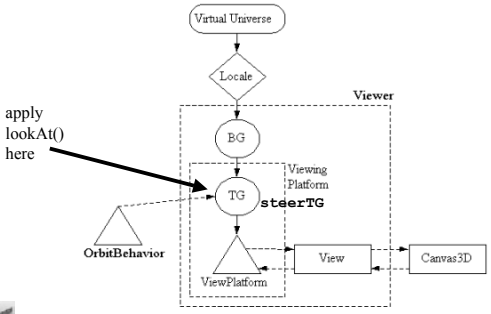

Positioning an Axis Value

```
TransformGroup axisTG = new TransformGroup();
Transform3D t3d = new Transform3D();
t3d.setTranslation( vertex );
   // vertex is the position for the label
axisTG.setTransform(t3d);

axisTG.addChild( message );
```


3. Viewer Positioning

- A simple way of positioning the viewer (the camera) is with the `lookAt()` method. It requires:
  - the viewer's intended position;
  - the point which he is looking at;
  - a vector specifying the upward direction.

View Branch for Checkers3D



Code Fragment

- The view branch is created by the `SimpleUniverse` utility class:

```
su = new SimpleUniverse(canvas3D);
```

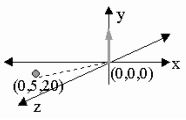
- Access the `TransformGroup` node for the `ViewPlatform`:

```
ViewingPlatform vp = su.getViewingPlatform();
TransformGroup steerTG =
       vp.getViewPlatformTransform();
```

 continued

- Apply `lookAt()` positioning (inverted since the position is relative to the viewer):

```
Transform3D t3d = new Transform3D();
steerTG.getTransform(t3d);

t3d.lookAt(  new Point3d(0,5,20),
             new Point3d(0,0,0),
             new Vector3d(0,1,0) );
t3d.invert();

steerTG.setTransform(t3d);
```



Viewer Movement

- The `OrbitBehavior` class allows a range of moves, rotations, and zooming of the viewer position:

```
OrbitBehavior orbit =
 new OrbitBehavior(c,OrbitBehavior.REVERSE_ALL);
orbit.setSchedulingBounds(bounds);

ViewingPlatform vp = su.getViewingPlatform();
vp.setViewPlatformBehavior(orbit);
```


4. Loader Classes

- `Loader`
  - specifies the elements that should be loaded from a file written in a given 3D format
- `Scene`
  - extracts Java 3D scene graph information from the loaded file

Loader Subclasses

- `Lw3dLoader`
  - for Lightwave 3D scene files
- `ObjectFile`
  - for Wavefront .obj files
- `LoaderBase`
  - implements the `Loader` interface in a generic way to encourage the building of loaders for other 3D formats through subclassing

Other Loaders

- A list of loaders for different file formats:

  `http://www.j3d.org/utilities/loaders.html`
- NCSA Portfolio
  - supports a wide range of formats
  - oldish, currently unsupported, simple
- See chapter 9 of my online book:

  `http://fivedots.coe.psu.ac.th/~ad/jg/ch9/`

ObjLoad Demo

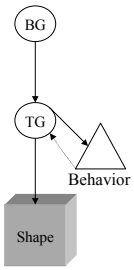
- The `ObjLoad.java` example in the Java 3D demo collection shows how to load a .obj file
  - in `<JAVA HOME>\demo\java3d\ObjLoad`
  - `java ObjLoad galleon.obj`

Loader Example

```
// OBJ Loader classes
import com.sun.j3d.loaders.objectfile.ObjectFile;
import com.sun.j3d.loaders.Scene;
    :

// load OBJ file
ObjectFile of = new ObjectFile(ObjectFile.RESIZE);
Scene scene = null;
try {
  scene = of.load("galleon.obj");
}
catch (Exception e) {
  System.err.println(e);
  System.exit(1);
}
// add loaded model to scene
BranchGroup modelBG = scene.getSceneGroup();
sceneBG.addChild( modelBG );
```

5. Behavior Objects

- Both interaction and animation are specified with `Behavior` objects
- A `Behavior` object changes the scene graph in response to events
  - key presses, mouse moves, object collisions, passage of time, etc.



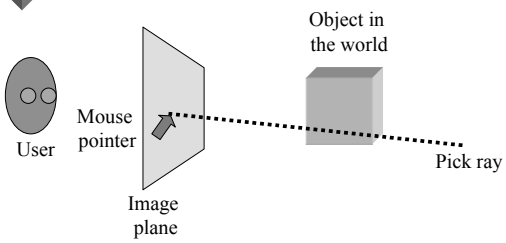
Some Behavior Classes

- `KeyNavigatorBehavior`
- `MouseBehavior`
- User-defined `Behavior` classes
  - triggered by `WakeupCondition` objects
- `PickMouseBehavior`
- `Interpolator`

WakeupConditions

- mouse, keyboard input
- collision
- time and frame change
- object movement
- camera movement
- sensor activity (input devices)

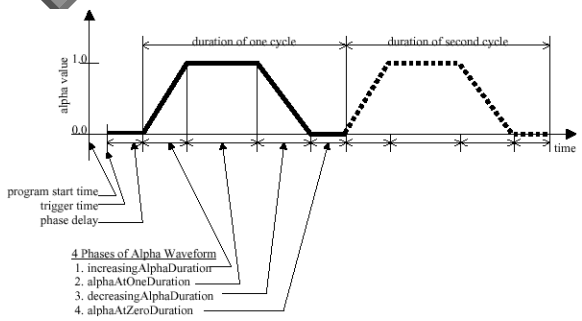
Picking



6. Animation

- Animation is implemented using time-based interpolation (or morphing)
  - time is specified using the `Alpha` class

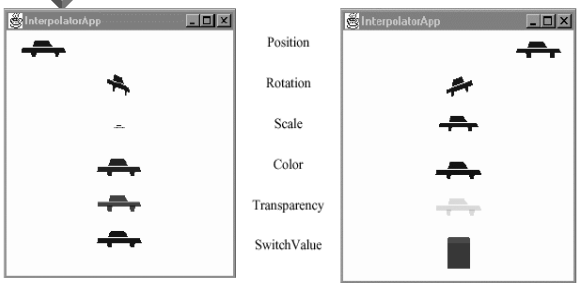
Alpha



Interpolator Behavior

- Position, Rotation, Scale
  - `PositionInterpolator, RotationInterpolator, ScaleInterpolator`
- Color and Transparency
  - `ColorInterpolator`
  - `TransparencyInterpolator`
- Object
  - `SwitchValueInterplator`

Interpolator Examples

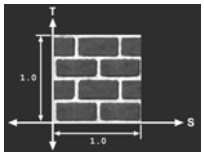


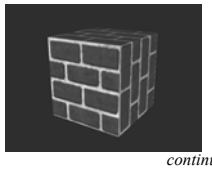
Morphing

- Morphing is the process of gradually deforming a shape into another one.
- For example:
  - changing a pyramid into a cube
  - changing a cat into a dog
  - making a hand grasp
- See the `Morph` class

7. Textures

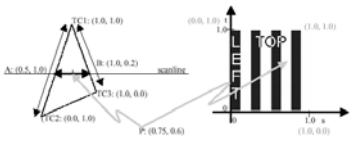
- Create the illusion of detail by wraping a complex image (texture) around a simple geometry
  - realism is increased without the need for a complex shape





continued

- 2D and 3D textures are supported
- Textures an be attached to shapes in a variety of predefined ways, or linked to specific coordinates in the shape.



8. Sound

- Sounds are obtained from audio files
  - typically ".au" or ".wav"
- Possible to modify the volume, left/right balance, and intramural delay of the audio emission
  - but... this is a buggy part of Java 3D

continued

- PointSound
  - radiate sound uniformly in all direction
- ConeSound
  - extend PointSound to create a sound that can be directed along a 3D vector
- BackgroundSound
  - similar to PointSound but does not have a location in space