Computer Game Technologies

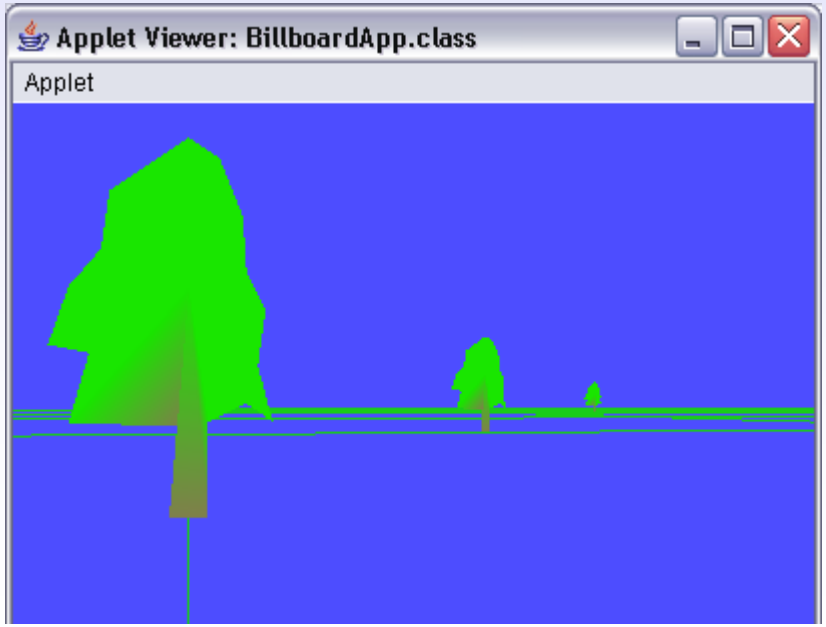
Advanced Animation

Advanced Animation

- Billboarding
- Level of detail (LOD)
- Picking

Billboarding

- Use of 2D images to represent 3D objects
  - Computationally cheap replacement for full 3D models
  - Typically used for complex real-world objects that are in the middle to far distance in the 3D world e.g. trees and other plants



Billboarding (2)

- Images are continually rotated as viewer position changes so that they face the viewer
  - Look 3D as always viewed in a direction parallel to the surface normal
  - Ideal for cylindrically or spherically symmetric objects e.g. water tower, moon
- Also used to keep textual information readable by viewer from any angle

Billboarding in Java 3D

- Billboard behavior object
  - Similar to Interpolator, but does not use an Alpha
  - Animation (rotation) of visible object driven by relative position of viewer
- References a transform group of a visible object
  - TG should be for exclusive use by Billboard
  - Visible object should have a separate TG for positioning (translation) in the world

Billboarding Scene Graph

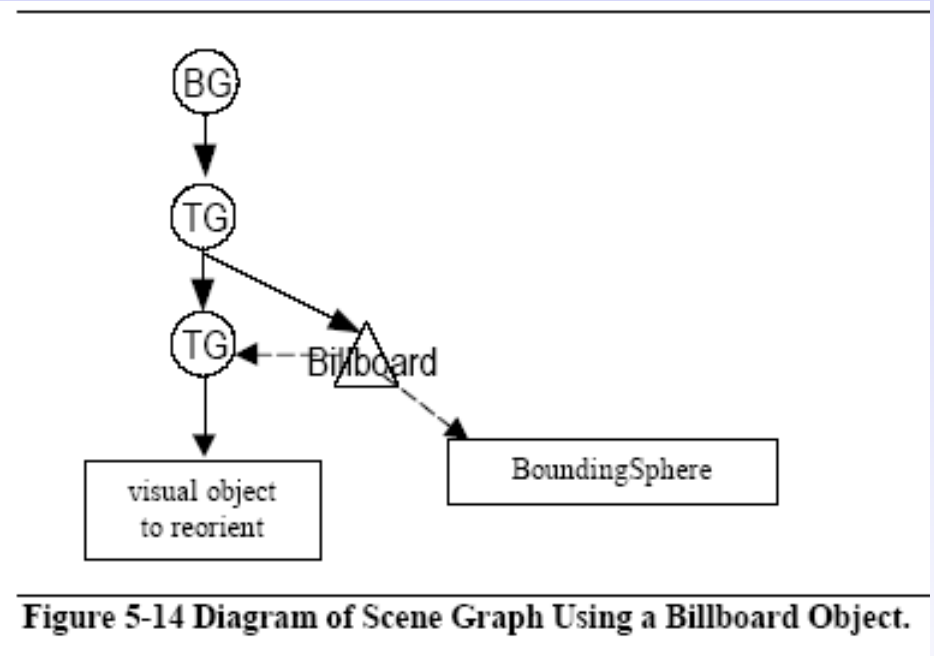


Figure 5-14 Diagram of Scene Graph Using a Billboard Object.

Billboarding Example

```
public BranchGroup createSceneGraph(SimpleUniverse su) {
  BranchGroup objRoot = new BranchGroup();

  Vector3f translate = new Vector3f();
  Transform3D T3D = new Transform3D();
  TransformGroup TGT = new TransformGroup();
  TransformGroup TGR = new TransformGroup();
  Billboard billboard = null;
  BoundingSphere bSphere = new BoundingSphere();

  translate.set(new Point3f(1.0f, 1.0f, 0.0f));
  T3D.setTranslation(translate);
  TGT.set(T3D);

  // set up for billboard behavior
  TGR.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
  billboard = new Billboard(TGR);
  billboard.setSchedulingBounds(bSphere);
```

Billboarding Example (2)

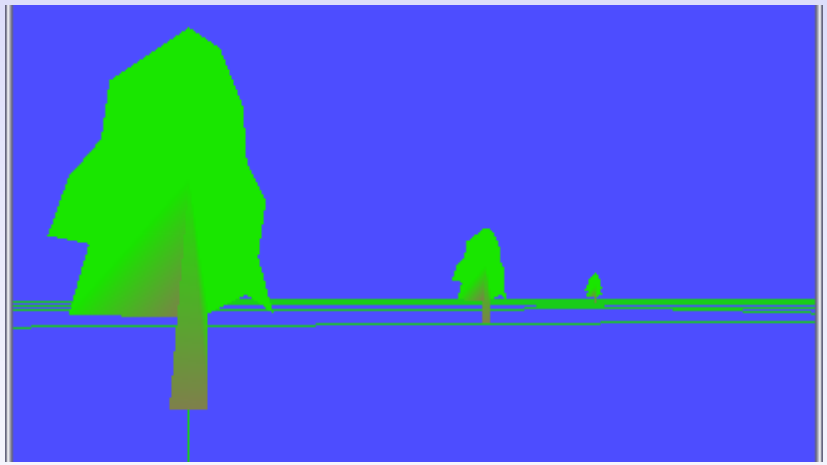
```
  // assemble scene graph
  objRoot.addChild(TGT);
  objRoot.addChild(billboard);
  TGT.addChild(TGR);
  TGR.addChild(createTree());

  // add KeyNavigatorBehavior(vpTrans) code removed;

  return objRoot;
} // end of CreateSceneGraph method of BillboardApp
```

Billboarding in Java 3D (2)

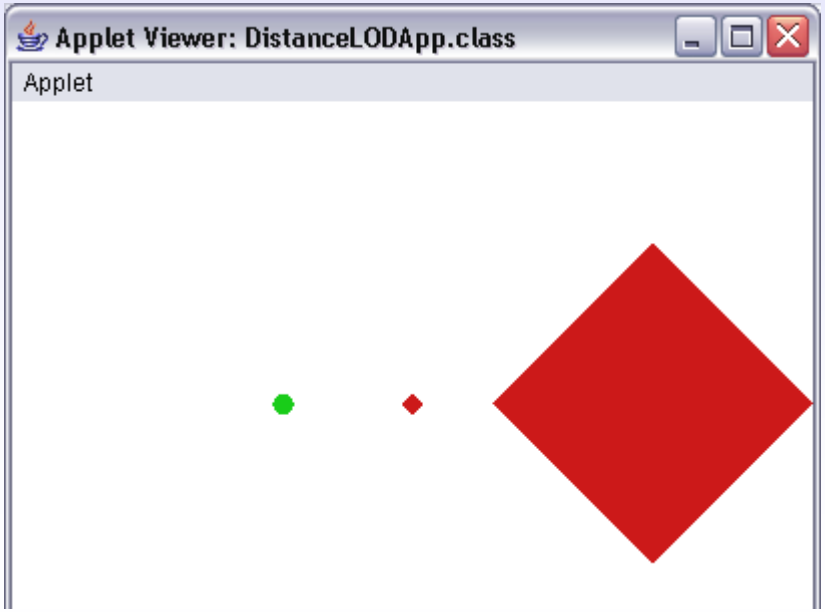
- Default rotation is around an axis
  - Good for cylindrically symmetric objects, such as trees
  - These objects would be revealed as 2D if viewed from above or below

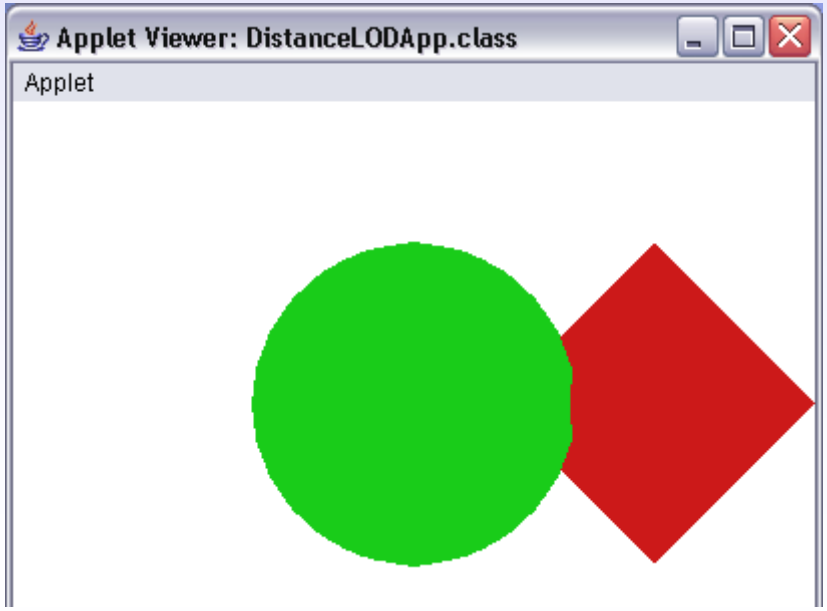


- Can also rotate around a point
  - Object viewed orthogonally from any viewer position
  - Good for spherical objects e.g. planets, moon

Level-of-Detail (LOD)

- Vary amount of detail in visual object depending on distance to the viewer
  - An object that is close to viewer should have good geometrical and appearance detail
  - Objects far away can use simple geometries and appearance





Level-of-Detail (2)

- LOD may also be changed in other circumstances:
  - Based on rendering speed ie reduce LOD to maintain a particular frame rate
  - Based on visual object velocity ie a fast moving object may not need as much geometric detail
  - LOD could be user settable e.g. to match screen resolution, processor speed

LOD in Java 3D

- LOD object
  - Abstract class
  - DistanceLOD extends LOD to provide "switch on distance to viewer"
  - Custom LOD classes can be written
- LOD references one or more Switch objects
- Switch objects allow switching between different visual objects
  - A Switch object is a special group that allows zero, one or more of its children (usually visual objects) to be rendered
- Switching depends on specified criteria
  - E.g. distance to viewer

DistanceLOD

- Create a number of visual objects with varying detail
- Add objects to a Switch object
- Set Switch object as target of DistanceLOD object
- Selection of child of Switch object to be rendered depends on a set of threshold distances
  - Distances specified as an array starting with the maximum distance at which the first child will be used
  - Switch to second child made when this distance exceeded
  - First child is most detailed visual object ie one to be used when close to viewer
  - Subsequent distances must always be greater than previous distance
  - One fewer distances need to be specified than there are visual objects to be rendered

DistanceLOD Example

```
public BranchGroup createSceneGraph() {
  BranchGroup objRoot = new BranchGroup();
  BoundingSphere bounds = new BoundingSphere();

  // create target TransformGroup with Capabilities
  TransformGroup objMove = new TransformGroup();

  // create DistanceLOD target object
  Switch targetSwitch = new Switch();
  targetSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE);

  // add visual objects to the target switch
  targetSwitch.addChild(new Sphere(.40f, 0, 25));
  targetSwitch.addChild(new Sphere(.40f, 0, 15));
  targetSwitch.addChild(new Sphere(.40f, 0, 10));
  targetSwitch.addChild(new Sphere(.40f, 0, 4));
```

DistanceLOD Example (2)

```
  // create DistanceLOD object
  float[] distances = { 5.0f, 10.0f, 20.0f};
  DistanceLOD dLOD = new DistanceLOD(distances, new Point3f());
  dLOD.addSwitch(targetSwitch);
  dLOD.setSchedulingBounds(bounds);

  // assemble scene graph
  objRoot.addChild(objMove);
  // make the bounds move with object
  objMove.addChild(dLOD);
  // must add switch to scene graph
  objMove.addChild(targetSwitch);

  return objRoot;
} // end of CreateSceneGraph method of DistanceLODApp
```

DistanceLOD Example (3)

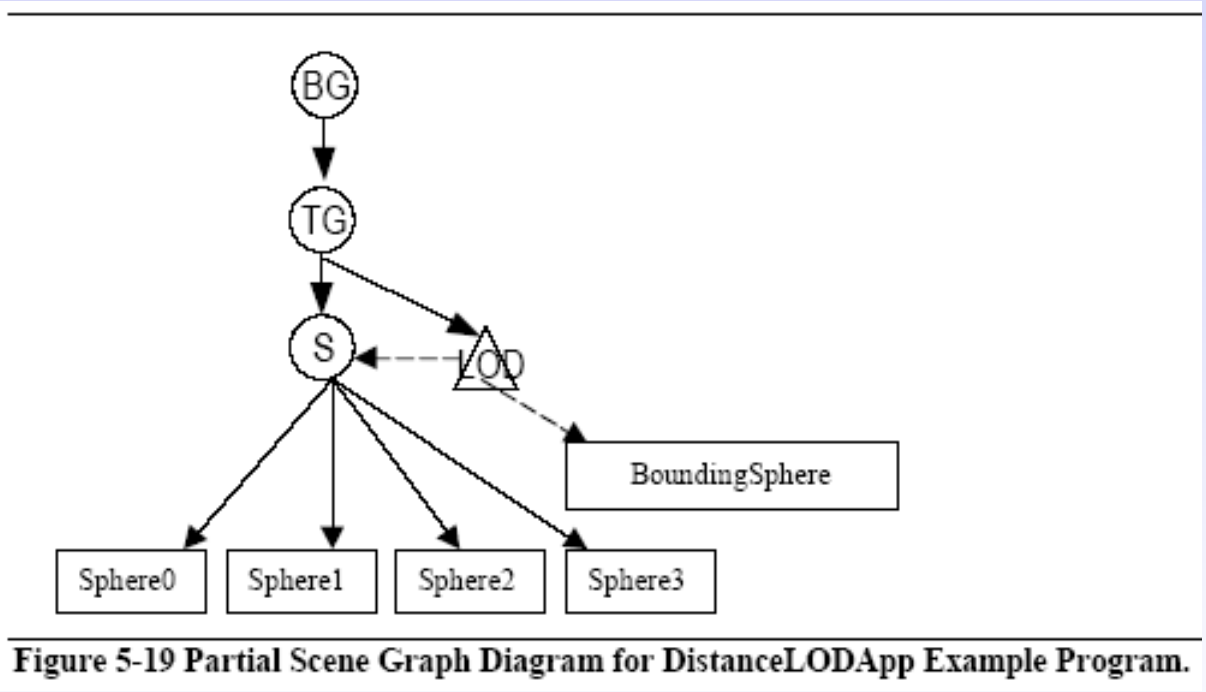


Figure 5-19 Partial Scene Graph Diagram for DistanceLODApp Example Program.

Switch Object

- A Switch object controls which of its children should be rendered
  - Zero, one or more
- Children are sub scene graph branches
- Very useful for quickly adding or removing visual objects from the visual scene
  - E.g. as the result of a collision, enemy being killed etc
- Children added using addChild() method
- Children selected for rendering using setWhichChild(int Child) method
  - Numerical index depends on order children added to Switch
  - Switch.CHILD_ALL to render all children
  - Switch.CHILD_NONE to make all children invisible

Switch Example

From Davison's Tour3D

```
public Sprite3D(String fnm, Obstacles obs)
  {
    :::
    // create switch for visibility
    visSwitch = new Switch();
    visSwitch.setCapability(Switch.ALLOW_SWITCH_WRITE);
    // add visual object to switch
    visSwitch.addChild( visObj.getTG() );
    // make visible
    visSwitch.setWhichChild( Switch.CHILD_ALL );

    // create a new transform group for the object
    objectTG = new TransformGroup();
 objectTG.setCapability(TransformGroup.ALLOW_TRANSFORM_READ);
 objectTG.setCapability(TransformGroup.ALLOW_TRANSFORM_WRITE);
    objectTG.addChild( visSwitch );
}  // end of Sprite3D()
```

Switch Example (contd)

```
  public void setActive(boolean b)
  // Activity changes affect the sprite's visibility
  { isActive = b;
    if (!isActive) // make invisible
       visSwitch.setWhichChild( Switch.CHILD_NONE );
    else if (isActive) // make visible
       visSwitch.setWhichChild( Switch.CHILD_ALL );
} // end of setActive()
```

Picking

- User places mouse pointer over visual object and presses mouse button
- A ray is projected into the visual world from the mouse pointer position
- Closest object to image plate that intersects the ray is "picked"

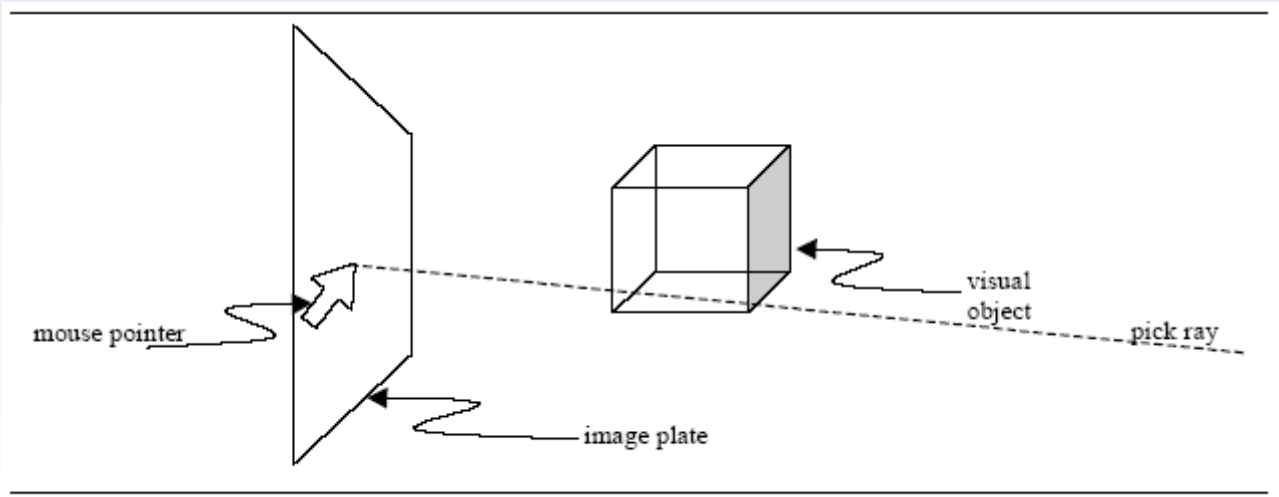


Figure 4-11 Projection of PickRay in the Virtual World

Picking (2)

- Calculating intersection of ray with visual objects is analogous to collision detection
- Default calculates intersections with actual object geometries
  - Potentially computationally very expensive
- Can choose to calculate intersections with simple bounding regions
  - Boxes, spheres
  - As done for collision detection

Result of Picking

- Picked object likely to be processed in some way
  - Moved or rotated
  - Change appearance (colour) or geometry
  - Make invisible
- These effects are applied to different points of the scene graph
  - Transform groups
  - Object Geometry or Appearance nodes
- So what should be returned by the "picking" operation?
  - Scene graph path to visual object

Result of Picking (2)

- For efficiency "ENABLE_PICK_REPORTING" capability must be explicitly set for any group node that is required to be returned in the scene graph path
- Also, leaf nodes can be set to be unpickable if not needed, to limit intersection calculations

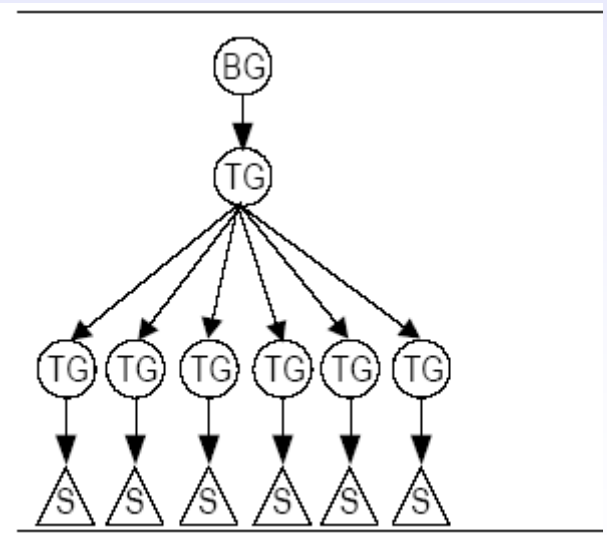


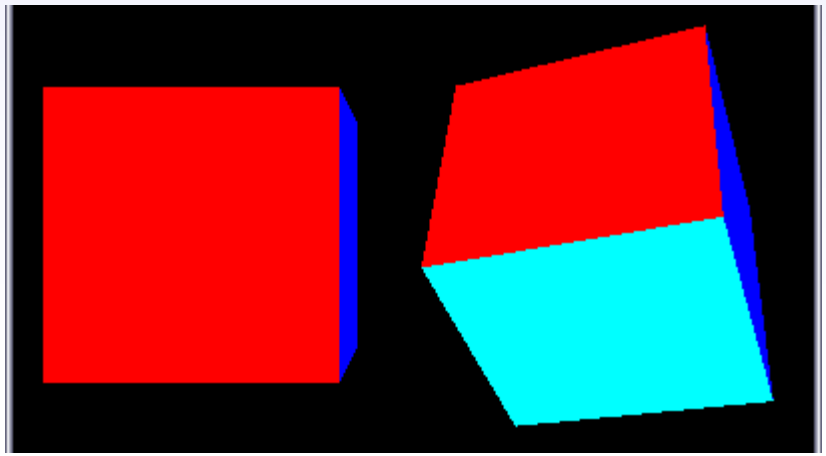
Figure 4-12 Scene Graph Diagram for a Cube

Pick Utility Classes

- Utility behaviour classes provided for object rotation, translation and zooming

```
PickRotateBehavior beh = new PickRotateBehavior(root,
    canvas, bounds);
Root.addChild(beh);
```

  - Picking operations on objects below root node
  - Canvas is where the mouse pointer is (Canvas3D)
  - Bounds is scheduling bounds for behavior



The End