OBJECT ILLUMINATION

1. Illumination model

- When exposed to a given distributed light (or a point light placed sufficiently far away), the illumination I at a surface point is determined by $\mathbf{N} \cdot \mathbf{L}$ and $\mathbf{E} \cdot \mathbf{R}$ in the equation

$$I = k_d I_a + \frac{I_p}{d + d_0}[k_d\,(\mathbf{N} \cdot \mathbf{L}) + k_s\,(\mathbf{E} \cdot \mathbf{R})^n]$$

  - I is the illumination intensity at the point
  - k_d, k_s are material constants
  - I_a, I_p are ambient and point light intensities
  - **N, L, E, R** are vectors defining the directions for surface normal, point light, viewer's eye, and reflection.

  - d, d_0 are distance constants.
  - The inner product of two vectors: $\mathbf{a}\cdot\mathbf{b} = |\mathbf{a}||\mathbf{b}|\cos\angle(\mathbf{a}, \mathbf{b})$

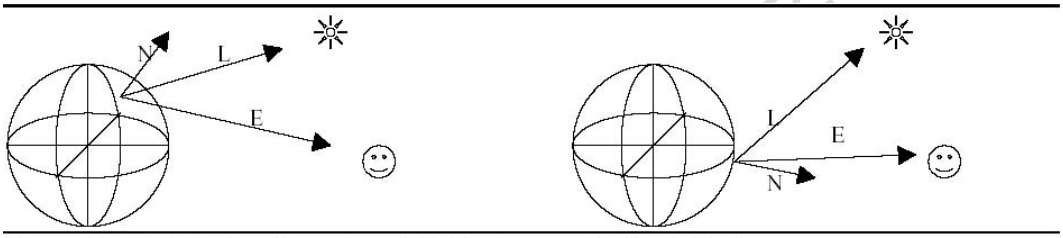


Figure 6-1 Light, Surface Normal, and Eye Vectors used to Shade Vertices.

2. Lighting reflections

- *Ambient* reflection results from ambient light, constant low level light, in a scene.
- *Diffuse* reflection is the normal reflection of a light source from a visual object.

- *Specular* reflections are the highlight reflections of a light source from an object, which occur in certain situations.

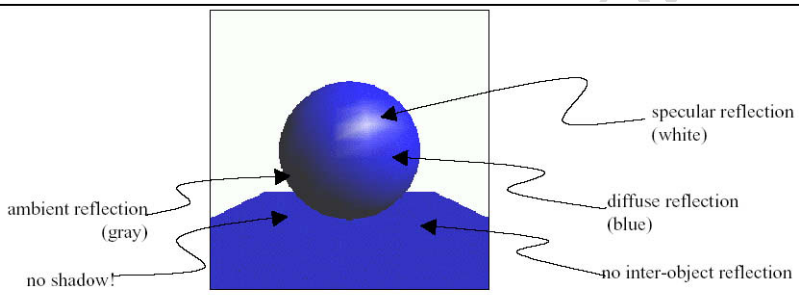


Figure 6-2 Shaded Sphere and Plane

3. Java3D shading model
   - The shade model is specified as one of `SHADE_GOURAUD`, `SHADE_FLAT`, `FASTEST`, `NICEST`.

- In *flat shading*, all pixels for a polygon are assigned the shade value from one vertex of the polygon.
  - A curved surface is represented as a set of constantly shaded plane surfaces.

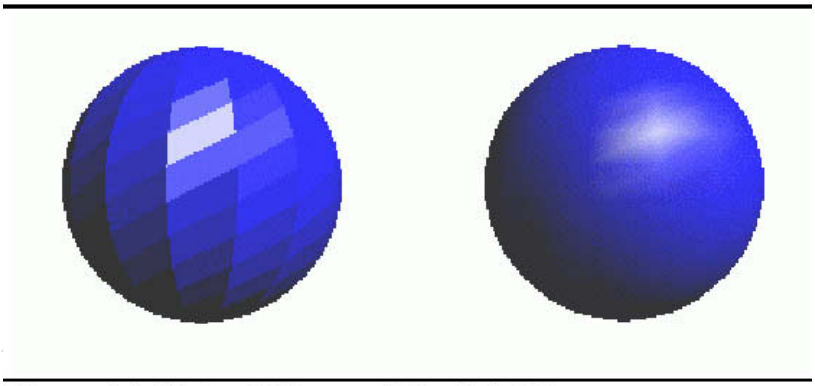


Figure 6-3 Flat and Gouraud Shaded Spheres.

- In *Gouraud shading*, each pixel is shaded with a value derived from trilinear interpolation of the shade value of each vertex of its enclosing polygon.
  - After the intensity values at the vertices of each polygon are determined, all other intensities for the surface are calculated from them.

4. Recipe for Lit Visual Objects

- Light Source specification
  - set bounds, and add to scene graph
- Visual object
  - normals and material properties

5. A simple lights example

```
1.   Appearance createAppearance() {
2.     Appearance appear = new Appearance();
3.     Material material = new Material();
4.     appear.setMaterial(material);
5.
6.     return appear;
7.   }
8.
9.   BranchGroup createScene (){
10.    BranchGroup scene = new BranchGroup();
11.
12.    scene.addChild(new Sphere(0.5f, Sphere.GENERATE_NORMALS,
13.                              createAppearance()));
14.
15.    AmbientLight lightA = new AmbientLight();
16.    lightA.setInfluencingBounds(new BoundingSphere());
17.    scene.addChild(lightA);
18.
19.    return scene;
20.  }
```

Code Fragment 6-1 Creating a Scene with a Lit Sphere.

6. Java 3D provides four types of lights

Class Hierarchy

```
java.lang.Object
└ javax.media.j3d.SceneGraphObject
  └ javax.media.j3d.Node
    └ javax.media.j3d.Leaf
      └ javax.media.j3d.Light
        ├ javax.media.j3d.AmbientLight
        ├ javax.media.j3d.DirectionalLight
        └ javax.media.j3d.PointLight
          └ javax.media.j3d.SpotLight
```

7. Creating ambient lights

- Light rays aim in all directions, flooding an environment and illuminating shapes evenly

8. AmbientLight example code

- Create a light

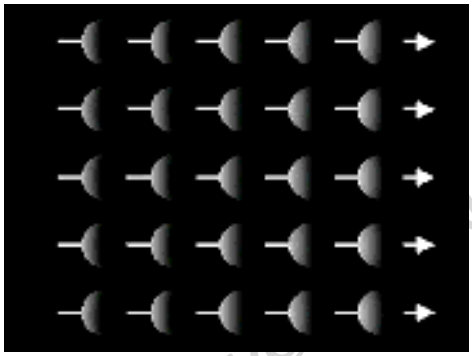
```
AmbientLight myLight = new AmbientLight( );
myLight.setEnable( true );
myLight.setColor( new Color3f( 1.0f, 1.0f, 1.0f ) );
```

- Set its influencing bounds

```
BoundingSphere myBounds
   = new BoundingSphere(new Point3d( ), 1000.0 );
myLight.setInfluencingBounds( myBounds );
```

9. Creating directional lights

- Parallel light rays aiming in a direction (default $-$z-axis).



10. DirectionalLight example code

- Create a light

```
DirectionalLight myLight = new DirectionalLight( );
myLight.setEnable( true );
```

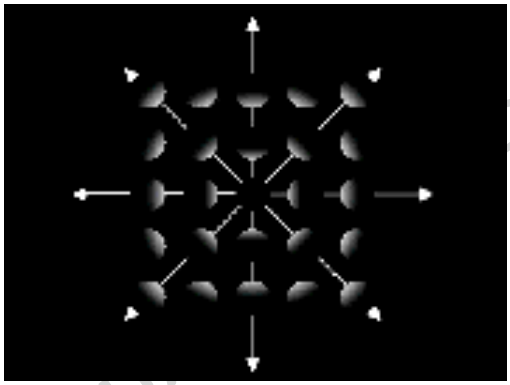
```
myLight.setColor( new Color3f( 1.0f, 1.0f, 1.0f ) );
myLight.setDirection( new Vector3f( 1.0f, 0.0f, 0.0f ));
```

- Set its influencing bounds

```
BoundingSphere myBounds
   = new BoundingSphere(new Point3d( ), 1000.0 );
myLight.setInfluencingBounds( myBounds );
```


11. Creating point lights

- Light rays emit radially from a point in all directions



12. Using point light attenuation

- Point light rays are attenuated:
  - As distance increases, light brightness decreases

- Attenuation is controlled by three coefficients:
  - constant, linear, and quadratic

$$brightness = \frac{lightIntensity}{constant + linear * distance + quadratic * distance^2}$$

13. PointLight example code

- Create a light

```
PointLight myLight = new PointLight( );
myLight.setEnable( true );
myLight.setColor( new Color3f( 1.0f, 1.0f, 1.0f ) );
myLight.setPosition( new Point3f( 0.0f, 1.0f, 0.0f ) );
myLight.setAttenuation(new Point3f(1.0f, 0.0f, 0.0f));
```


- Set its influencing bounds

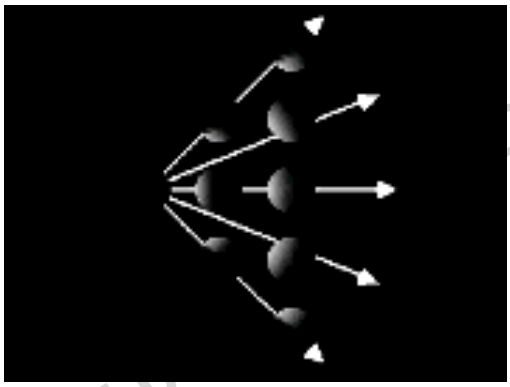
```
BoundingSphere myBounds
   = new BoundingSphere(new Point3d( ), 1000.0 );
myLight.setInfluencingBounds( myBounds );
```

14. Creating spot lights

- Light rays emit radially from a point, within a cone
  - Vary the spread angle to widen, or narrow the cone
    - Spread angle varies from 0.0 to PI/2.0 radians (default PI)
    - A value of PI radians makes the light a PointLight
  - Vary the concentration to focus the spot light
    - Concentrations vary from 0.0 (unfocused) to 128.0 (focused)

– The default is 0.0

- The default aim direction is (0.0, 0.0, -1.0).



15. SpotLight example code

- Create a light

```
SpotLight myLight = new SpotLight( );
```



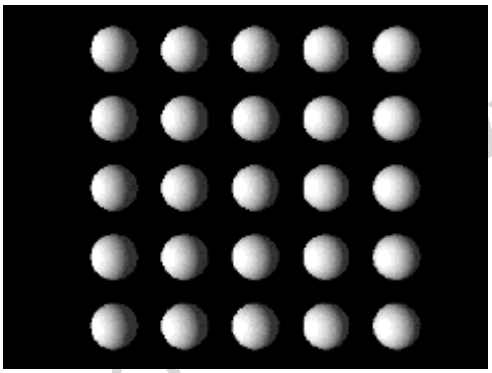
```
myLight.setEnable( true );
myLight.setColor( new Color3f( 1.0f, 1.0f, 1.0f ) );
myLight.setPosition( new Point3f( 0.0f, 1.0f, 0.0f ) );
myLight.setAttenuation(new Point3f(1.0f, 0.0f, 0.0f));
myLight.setDirection(new Vector3f(1.0f, 0.0f, 0.0f ));
myLight.setSpreadAngle( 0.785f ); // 45 degrees
myLight.setConcentration( 3.0f ); // Unfocused
```

- Set its influencing bounds

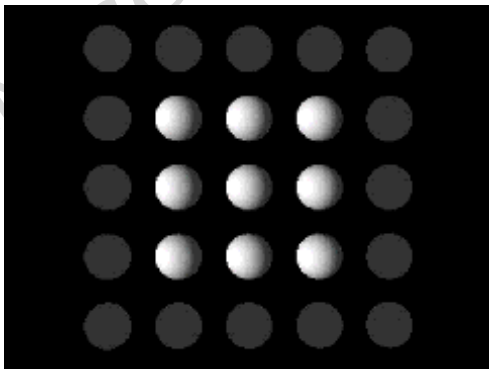
```
BoundingSphere myBounds
   = new BoundingSphere(new Point3d( ), 1000.0 );
myLight.setInfluencingBounds( myBounds );
```

16. Using light influencing bounds

- A light's illumination is bounded to a region of influence
  - Shapes within the region may be lit by the light



Large bounds



Small bounds

- Light bounding:

  - Enables controlled lighting in large scenes
  - Avoids over-lighting a scene when using multiple lights
  - Saves lighting computation time

17. Creating influencing bounds

- A light region of influence is a bounded volume:
  - Sphere, box, polytope, or combination using Bounds
  - To make a global light, use a huge bounding sphere
- By default, lights have no influencing bounds and illuminate nothing!
  - Common error: forgetting to set influencing bounds

18. Anchoring influencing bounds

- A light bounding volume can be relative to:

- The light's coordinate system
- Volume centered on light
- As light moves, so does volume

- A Bounding leaf's coordinate system
  - Volume centered on a leaf node elsewhere in scene graph
  - As that leaf node moves, so does volume
  - If light moves, volume does not

19. Influencing bounds example code

- Set bounds relative to the lights coordinate system

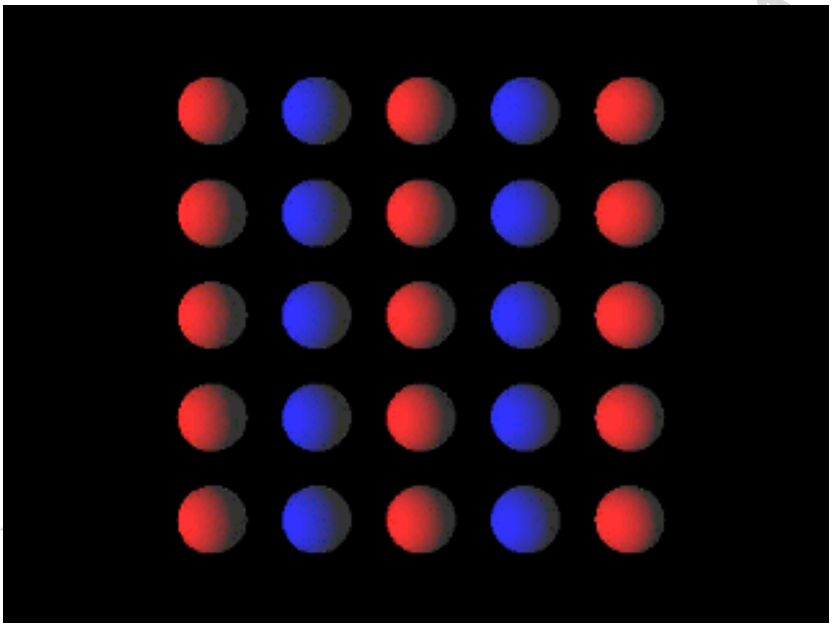
```
PointLight myLight = new PointLight( );
myLight.setInfluencingBounds( myBounds );
```


- Or relative to a bounding leafs coordinate system

```
TransformGroup myGroup = new TransformGroup( );
BoundingLeaf myLeaf = new BoundingLeaf( myBounds );
myGroup.addChild( myLeaf );
.  .  .
PointLight myLight = new PointLight( );
myLight.setInfluencingBoundingLeaf( myLeaf );
```

20. Scoping lights

- A lights illumination may be scoped to one or more groups of shapes
  - Shapes within the influencing bounds and within those groups are lit.



- By default, lights have universal scope and illuminate everything within their influencing bounds

21. Scoping example code

- Build a group of shapes

```
TransformGroup myLightable = new TransformGroup( );
Shape3D myShape = new Shape3D( myGeom, myAppear );
myLightable.addChild( myShape );
```

- Create a light and add the group to its scope list

```
DirectionalLight myLight = new DirectionalLight( );
myLight.addScope( myLightable );
```

22. Hints on Using Lights

- Use as few light sources as you can.
- Directional light sources are preferred since the computation required in rendering is significantly less than for point and spot lights.
- Point light sources are rarely used due to the high computational complexity.
- Including a single Ambient light source with a large region of influence is normal.
- The time required to include the Ambient light is small compared to other light sources.
- For objects, the more vertices, the smoother the lighting effect and the longer it will take to render.