Java 3D Lighting

- Java 3D supports the following types of light sources
  - Ambient
  - Directional
  - Point
  - Spot
- Java 3D also supports mechanisms for defining the volumes which lights can affect

General Light Methods

`void setColor(Color3f color)`

Sets the light's current colour

`void setEnable(boolean state)`

Turns the light on or off

`void setInfluencingBounds(Bounds bounds)`

Sets the light's influencing bounds (more later)

Ambient Lights

Light source objects providing the same intensity of light at all locations in all directions

`AmbientLight()`

Constructs and initialises an ambient light source with default values of `lightOn=true` and `colour=(1,1,1)`

`AmbientLight(Color3f colour)`

Constructs and initialises an ambient light with specified colour

`AmbientLight(boolean lightOn, Color3f colour)`

Constructs and initialises an ambient light with given values

Directional Lights

Very distant light sources with a constant direction

`DirectionalLight()`

Constructs and initialises a directional source with default values of `lightOn=true`, `colour=(1, 1, 1)` and `direction=(0, 0, -1)`

`DirectionalLight(Color3f colour, Vector3f direction)`

Constructs and initialises a directional light with specified colour and direction
By default the state is true (on)

`DirectionalLight(boolean lightOn, Color3f colour, Vector3f direction)`

Constructs and initialises a directional light with given values

Point Lights

Light sources at fixed points in space that radiate light equally in all directions away from them

`PointLight()`

Constructs and initialises a point light source with default values of `lightOn=true, colour=(1,1,1), position=(0,0,0)` and `attenuation=(1,0,0)`

`PointLight(Color3f colour, Point3f position, Point3f attenuation)`

Constructs and initialises a point light with given colour, position and attenuation

By default the state is true (on)

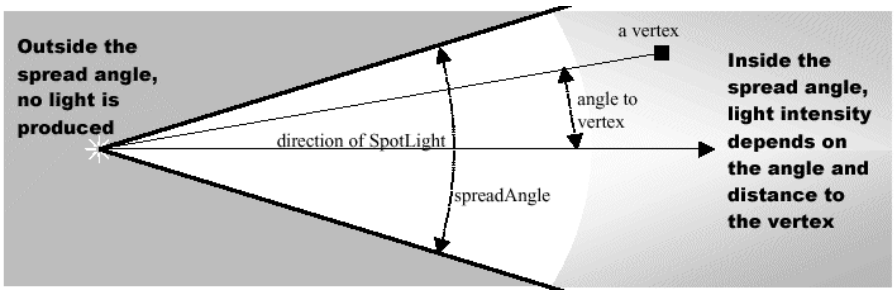
`PointLight(boolean lightOn, Color3f colour, Point3f position, Point3f attenuation)`

Constructs and initialises a point light with given values

Spot Lights

Point light sources with direction, spread angle and concentration added

- The spread angle of a spotlight may cause the light to illuminate part of a visual object
- This is the only light capable of illuminating just a part of a visual object



Spot Lights

`SpotLight()`
Constructs and initialises a spot light source with default values of
`lightOn=true, colour=(1,1,1), position=(0,0,0), attenuation=(1,0,0), direction=(0,0,-1), spreadAngle=PI (180 degrees)` and `concentration=0.0`

```
SpotLight(Color3f colour, Point3f position,
   Point3f attenuation, Vector3f direction,
   float spreadAngle, float concentration)
```
Constructs and initialises a spot light with given values
By default the light is on

Influencing Bounds

- Java 3D insists that all light sources have a boundary of influence
  - Geometry objects must intersect a light's region of influence in order to be affected by it
  - This permits considerable savings in computation time
  - Except in the case of spot lights the whole of any such object is affected by the light during rendering
- There are a number of ways of achieving this

Bounding Primitives

- Java 3D provides 3 Bounds-derived classes for creating boundary regions
  - `BoundingBox()`
    - Cuboid aligned with the main co-ordinate axes
  - `BoundingSphere()`
    - Sphere centred on the origin
  - `BoundingPolytope()`
    - Convex volume defined by 4 or more planes

`BoundingBox()`

```
public BoundingBox()
```

Constructs and initialises a bounding box about the origin
The lower corner is initialised to (-1.0, -1.0, -1.0)
The upper corner is initialised to (1.0, 1.0, 1.0)

```
public BoundingBox(Point3d lower,
                   Point3d upper)
```

Constructs and initialises a bounding box given minimum and maximum dimensions in x, y, z

`lower` is the "small" corner `upper` is the "large" corner

BoundingSphere()

```
public BoundingSphere()
```

Constructs and initialises a bounding sphere with radius = 1 at the origin

```
public BoundingSphere(Point3d centre,
                      double radius)
```

Constructs and initialises a bounding sphere from a centre and radius

BoundingPolytope()

```
public BoundingPolytope()
```

Constructs a bounding polytope and initialises it to a set of 6 planes that define a cube such that -1 <= x,y,z <= 1

The values of the planes are

plane[0] : x <= 1 (1,0,0,-1)
plane[1] : -x <= 1 (-1,0,0,-1)
plane[2] : y <= 1 (0,1,0,-1)
plane[3] : -y <= 1 (0,-1,0,-1)
plane[4] : z <= 1 (0,0,1,-1)
plane[5] : -z <= 1 (0,0,-1,-1)

Bounding Polytopes

- A bounding polytope object defines a volume using the intersection of four or more half-spaces
- The region defined by a bounding polytope is always convex and must be closed
- Each plane in a bounding polytope specifies a half-space defined by the equation:

$$Ax + By + Cz + D \leq 0$$

  where A, B, C, D are the parameters that specify the plane
- The intersection of the set of half-spaces corresponding to the planes defines the volume

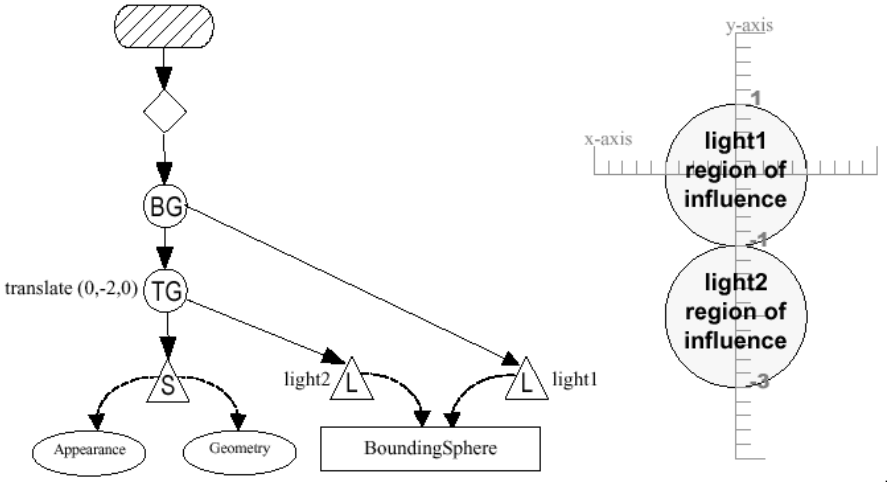
Spherical Influencing Bounds

```
AmbientLight lightA = new
  AmbientLight();

lightA.setInfluencingBounds(new
  BoundingSphere());

scene.addChild(lightA);
```

Bounding Spheres



Bounding Leaves

- If we don’t want a light’s region of influence to follow it around but remain in a particular place in our universe we can use a bounding leaf
- Bounding leaves are leaf nodes of the scene graph in their own right and exist independently of any lights and their locations
- Bounding leaves can be given their own transform groups

BoundingLeaf()

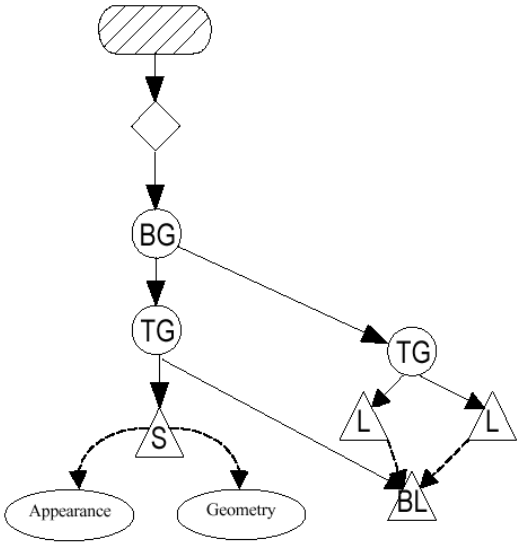
```
BoundingLeaf sphereLeaf =
 new BoundingLeaf(new BoundingSphere());

AmbientLight LA = new AmbientLight();

LA.setInfluencingBoundingLeaf(sphereLeaf);

scene.addchild(sphereLeaf);
scene.addChild(LA);
```

Bounding Leaf



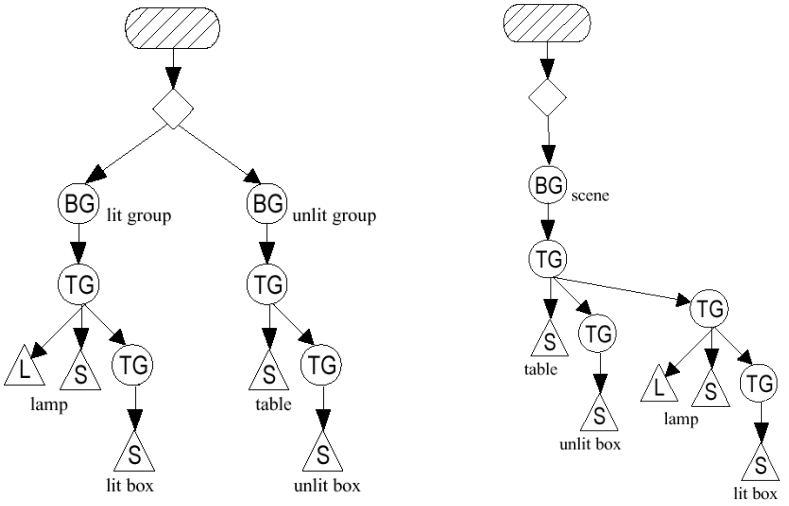
Scope

- With bounding leaves we are getting close to a situation in which lights may be set up such that their region of influence is determined by geometry objects in the scene graph - close but not quite
- Scope allows us to actually do this!
  - E.g. a spotlight can be set to follow a particular object wherever it goes using scope

Scope Lists

- Branches of the scene graph can be added to the scope of a light with `lightA.addScope(selectedBranchGroup)`
- Scoping can lead to the lights driving the scene construction in very strange ways so beware
  - In the diagrams on the next slide what happens to the lamp when the table is moved?

Scope versus Non-Scope Example



Scope/Non-Scope Example
Two boxes and a lamp on a table