JAVA 3D

Corso di Immagini e Multimedialità
Lezione 3.5: JAVA3D – Oggetti 3D

cristian.virgili@uniud.it

ASPETTO E ATTRIBUTI (1)

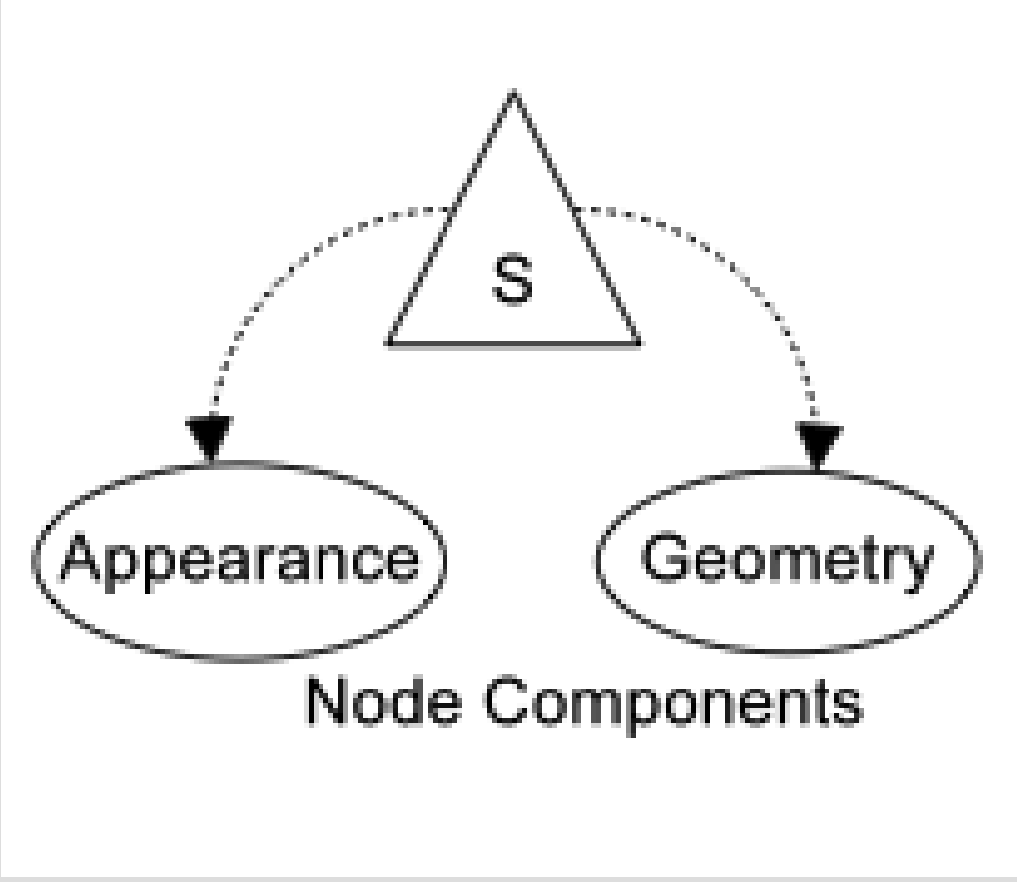
Uno Shape3D può fare riferimento sia ad un oggetto *Geometry* sia ad un oggetto *Appearance*

La *Geometry* specifica le informazioni spaziali e (opzionale) di colore sui vertici

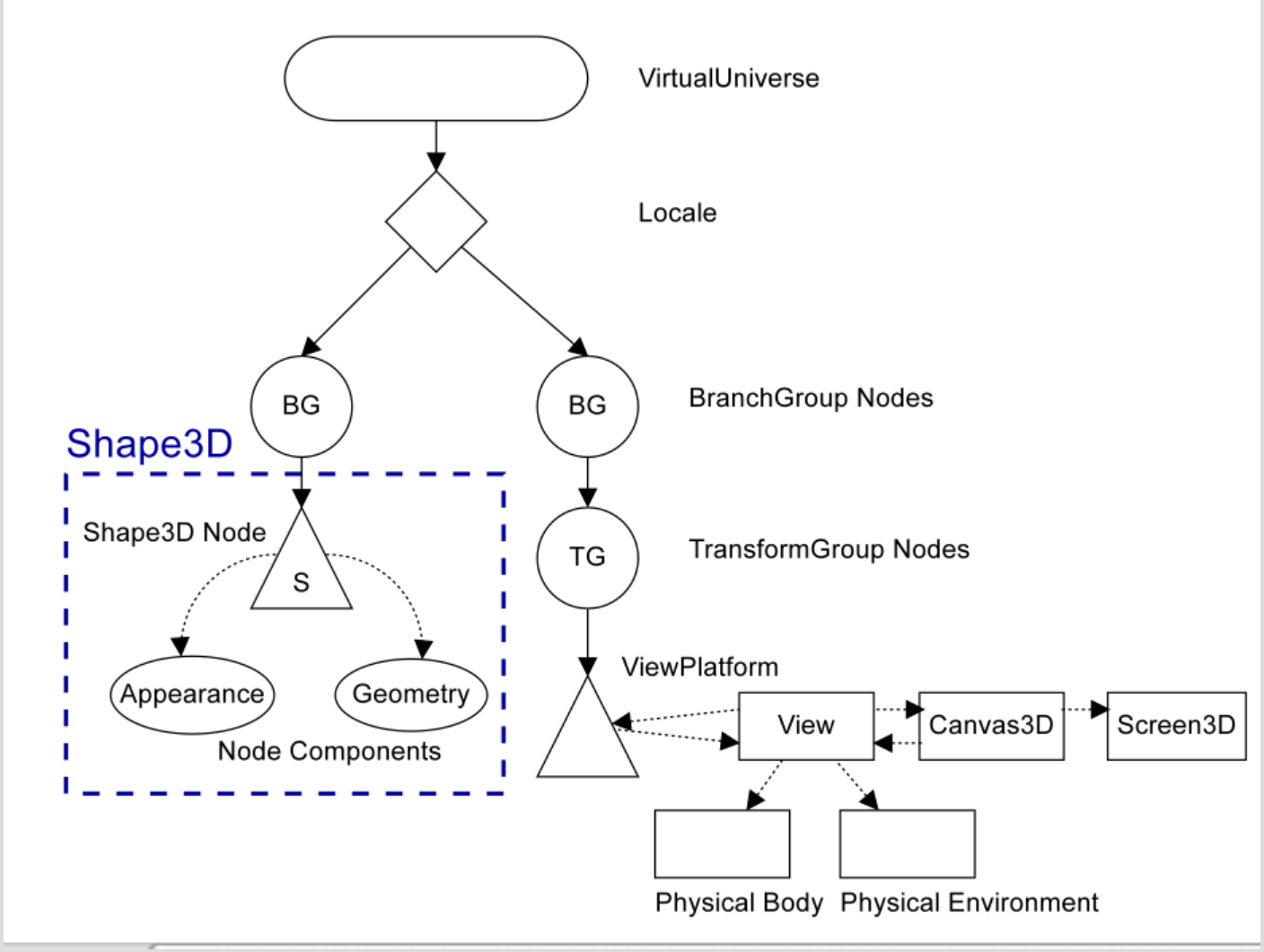
Tuttavia i dati di una *Geometry* sono spesso insufficienti per descrivere l'aspetto di un oggetto:

-è necessario utilizzare un oggetto *Appearance*

APPEARANCE



Shape3D


VirtualUniverse
Locale
BG
BG
BranchGroup Nodes
Shape3D
Shape3D Node
S
TG
TransformGroup Nodes
ViewPlatform
Appearance
Geometry
Node Components
View
Canvas3D
Screen3D
Physical Body
Physical Environment

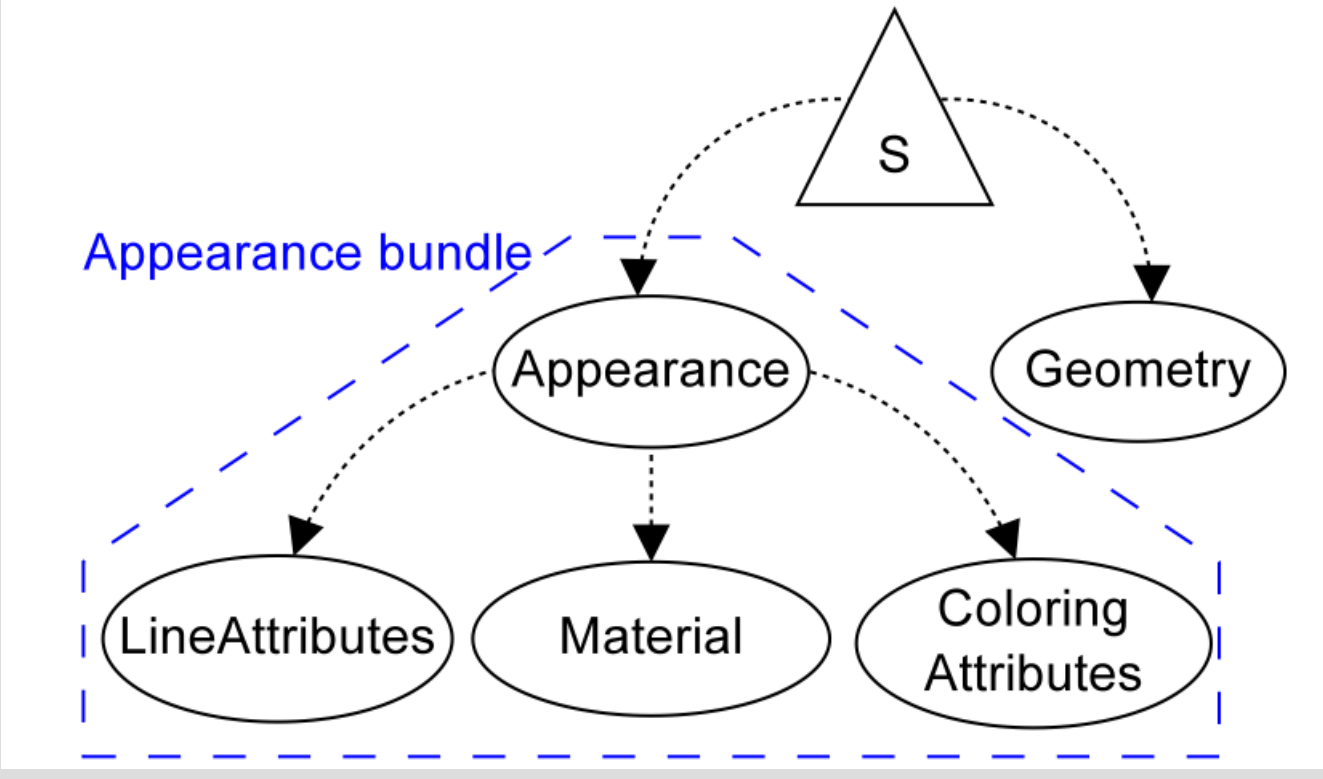
ASPETTO E ATTRIBUTI (2)

Un oggetto *Appearance* non contiene le informazioni sull'aspetto di uno Shape3D ma sa dove trovare tali informazioni

Un oggetto *Appearance* può referenziarsi a molteplici oggetti sottoclassi di *NodeComponent* (detti “attributi”), dove sono memorizzate le informazioni sull'aspetto dello Shape3D

ASPETTO E ATTRIBUTI (3)

L'insieme di *Appearance* e degli attributi a cui fa riferimento è un **appearance bundle**.



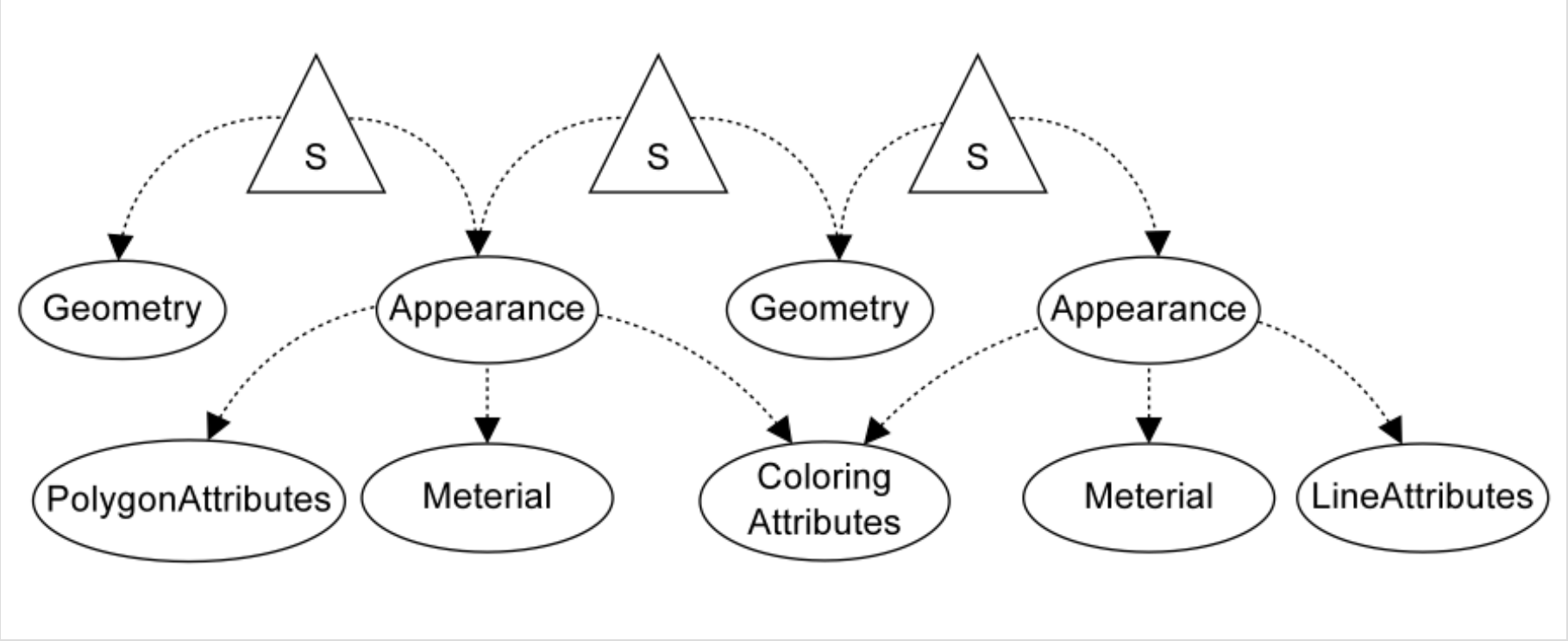
ASPETTO E ATTRIBUTI (4)

Gli oggetti a cui può referenziarsi un *Appearance* sono molteplici, i principali sono:

- void setPointAttributes(PointAttributes pA)
- void setLineAttributes(LineAttributes lA)
- void setPolygonAttributes(PolygonAttributes pA)
- void setColoringAttributes(ColoringAttributes cA)
- void setTransparencyAttributes(TransparencyAttributes tA)
- void setRenderingAttributes(RenderingAttributes rA)

ASPETTO E ATTRIBUTI (5)

Come per *Geometry*, *Appearance* e relativi attributi sono *NodeComponent*, quindi non fanno parte dell'albero. Si possono condividere le strutture dati.



PointAttributes

L'oggetto *PointAttributes* gestisce come devono essere renderizzati i punti

Per default se un vertice è renderizzato come un punto allora viene riempito un solo pixel

È possibile usare il metodo *setPointSize()* per rendere il punto più grande

Per default punti più grandi hanno l'aspetto di un quadrato, a meno di non usare il metodo *setPointAntialiasingEnable()*

LineAttributes

L'oggetto *LineAttributes* gestisce come devono essere renderizzate le linee

L'oggetto *LineAttributes* decide come renderizzare le linee tramite tre parametri

Per default una linea è tracciata continua, di spessore uno e senza antialiasing

Questi attributi possono essere modificati chiamando i metodi *setLinePattern()*, *setLineWidth()* e *setLineAntialiasingEnable()*

PolygonAttributes (1)

L'oggetto *PolygonAttributes* gestisce come devono essere renderizzati i poligoni

L'oggetto *PolygonAttributes* decide come renderizzare i poligoni tramite due parametri:

- come il poligono è rasterizzato (linee, punti, poligoni)
- se alcune sue parti non devono essere renderizzate

PolygonAttributes (2)

Per default un poligono è rasterizzato pieno, ma il metodo *setPolygonMode*() può cambiare tale modalità (attivando il wireframe o solo vertici)

Il metodo *setCullFace*() può essere usato per ridurre il numero di poligoni da renderizzare

Se *setCullFace()* è impostato su **CULL_FRONT** o **CULL_BACK** allora in media la metà dei poligoni non sarà renderizzata
se impostato su **CULL_NONE** vengono renderizzati tutti i poligoni

ColoringAttributes (1)

L'oggetto *ColoringAttributes* controlla come qualsiasi primitiva è colorata

Il metodo *setColor()* imposta un colore che (in alcuni casi) specifica il colore della primitiva

Il metodo *setShadeModel()* determina come interpolare i colori tra le varie primitive

ColoringAttributes (2)

Poiché i colori possono essere definiti anche ad ogni vertice della *Geometry*, ci potrebbe essere un conflitto con il colore del *ColoringAttributes*

In tali situazioni prevale il colore della *Geometry*

Analogamente se sono state attivate delle luci il *ColoringAttributes* viene ignorato

TransparencyAttributes (1)

L'oggetto *TransparencyAttributes* gestisce la trasparenza di qualsiasi primitiva

Il metodo *setTransparency()* definisce il valore di trasparenza (0.0 opaco, 1.0 trasparente)

Il metodo *setTransparencyMode()* abilita la trasparenza ed imposta la tecnica di trasparenza usata (**SCREEN_DOOR, BLENDED** e **NONE**)

TransparencyAttributes (2)

La tecnica **SCREEN_DOOR** seleziona (random) alcuni pixel e li rende totalmente opachi, il resto dei pixel è reso totalmente trasparente
- la percentuale di pixel trasparenti è uguale al valore di trasparenza impostato

La tecnica **BLENDED** imposta la trasparenza di ogni pixel al valore di trasparenza impostato

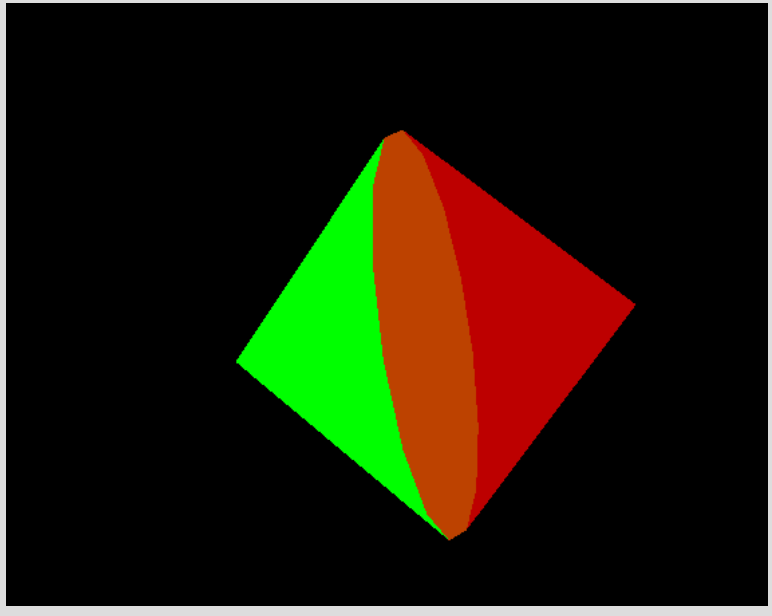
Lo **SCREEN_DOOR** è più veloce del **BLENDED**, ma (ovviamente) al costo di una minore qualità del rendering

ESERCIZI

Colorare un oggetto di rosso

Riprendere l'esempio della trottola e colorare le due parti di due colori diversi (ad esempio rosso e verde)

Rendere una delle due parti trasparente del 50%



ESERCIZI

Riprendere l'esempio del CUBO TAGLIATO e fare in modo che si vedano le parti nascoste.

BOUND e SCOPE (1)

Negli esempi studiati abbiamo visto che per alcuni oggetti è necessario definire un *Bound*

un *Bound* definisce uno spazio (o regione) dove un nodo (una luce, un behavior, ecc.) agisce un *Bound* è definito usando le classi *BoundingSphere*, *BoundingBox* e *BoundingPolytope*

Ad esempio una sorgente di luce deve definire una regione di influenza (*influencing bounds*) e se un oggetto la interseca allora sarà illuminato

BOUND e SCOPE (2)

Lo Scope è una collezione di oggetti *Group* (*BranchGroup*, *TransformGroup*, ecc.) sui quali un certo nodo fa sentire la sua influenza
Bound e *Scope* forniscono una funzionalità simile, ma in modo diverso:
- un **Bound** specifica regioni spaziali
- uno **Scope** specifica raggruppamenti

Bound e *Scope* sono usati da molti oggetti: luci, behavior, nebbia, suoni, ecc.

BOUND

I *Bound* permettono al programmatore di modificare azioni, aspetto, suoni, ecc.

Se opportunamente usati i *Bound* permettono anche di incrementare le performance, riducendo il lavoro svolto dal motore di rendering

Per tale motivo i *Bound* dovrebbero essere scelti più piccoli (e semplici) possibili

BOUNDING SPHERE

La *BoundingSphere* è il modo più semplice per definire una "bounding region"

Come dice il nome stesso la *BoundingSphere* serve per creare bound sferici

Il costruttore di default crea una sfera centrata nell'origine con **raggio 1**

È il *bound* che fornisce le migliori performance di rendering

BOUNDINGBOX

Un *BoundingBox* definisce un bound rettangolare tramite due punti:

- un punto è la combinazione delle coordinate (x,y,z) minime della regione, l'altro punto è la combinazione delle coordinate (x,y,z) massime

- i lati sono paralleli agli assi

- questo vincolo rende il calcolo delle intersezioni tra bound più semplice

Il costruttore di default crea un cubo centrato nell'origine con lato 2

BOUNDING POLYTOPE

Un *BoundingPolytope* definisce un bound poliedrico usando l'intersezione di quattro o più semispazi

- un semispazio è una regione di spazio limitata da un lato da un piano infinito
- la definizione di un *BoundingPolytope* è fatta elencando i piani che creano la regione
-Una volta che un *BoundingPolytope* è stato creato il numero dei suoi piani non può essere cambiato
-Il costruttore di default crea un cubo centrato nell'origine con lato 2

SCOPE

Alcuni nodi permettono di specificare sia un *Bound* sia uno *Scope* ed i due oggetti non sono mutuamente esclusivi

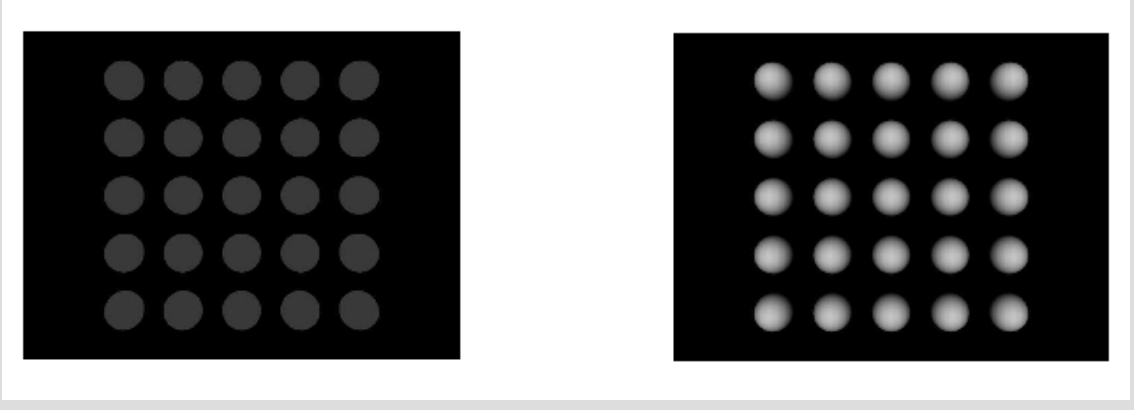
Un nodo per avere effetto su un oggetto, l'oggetto deve essere sia nel *Bound* sia nello *Scope* del nodo

Lo *Scope* è usato per limitare l'effetto di un nodo quando l'uso di un *Bound* è difficile o impossibile

ILLUMINAZIONE

L'illuminazione conferisce maggiore realismo alla scena

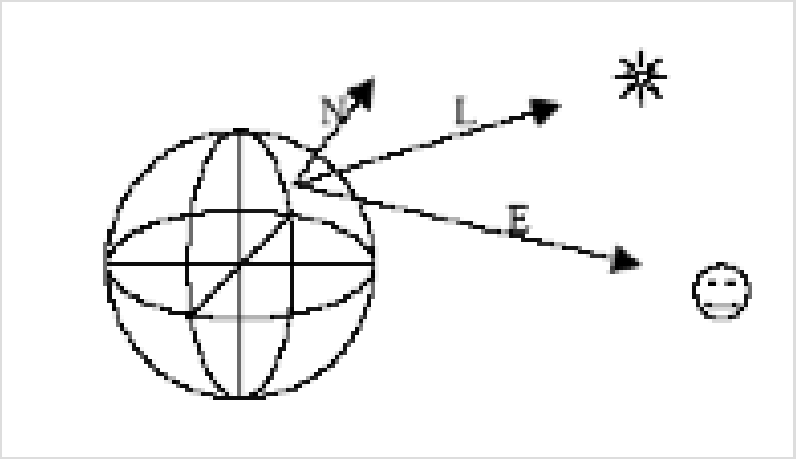
Gli effetti dell'illuminazione sono dati da:
- Proprietà fisiche dell'oggetto;
- Caratteristiche della sorgente luminosa;
- Posizione degli oggetti rispetto alla sorgente;
- Angolo da cui è visto l'oggetto.



ILLUMINAZIONE 2

I parametri fondamentali per gestire l'illuminazione di una superficie sono:

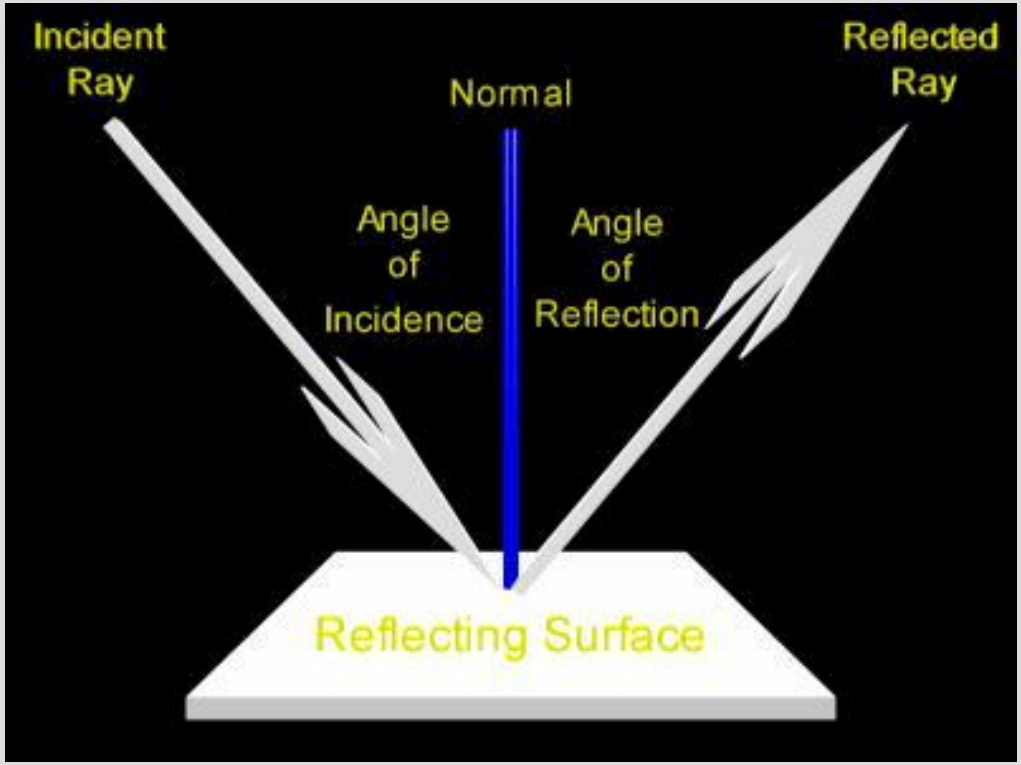
- La normale (N) Linea perpendicolare al punto considerato
- La direzione della luce (L)
- La direzione in cui guarda l'osservatore (E)



NORMALI (1)

Utilizzando le primitive **com.sun.j3d.utils.geometry** di Box, Cone, Cylinder e Sphere, le normali alla superficie vengono generate automaticamente per impostazione predefinita.

Una normale è il vettore perpendicolare ad un piano, di 90 gradi dal piano.



NORMALI (2)

Ad esempio ogni faccia di un cubo avrà quattro *Point3f* con le corrispondenti quattro normali che definiscono la normale alla superficie.

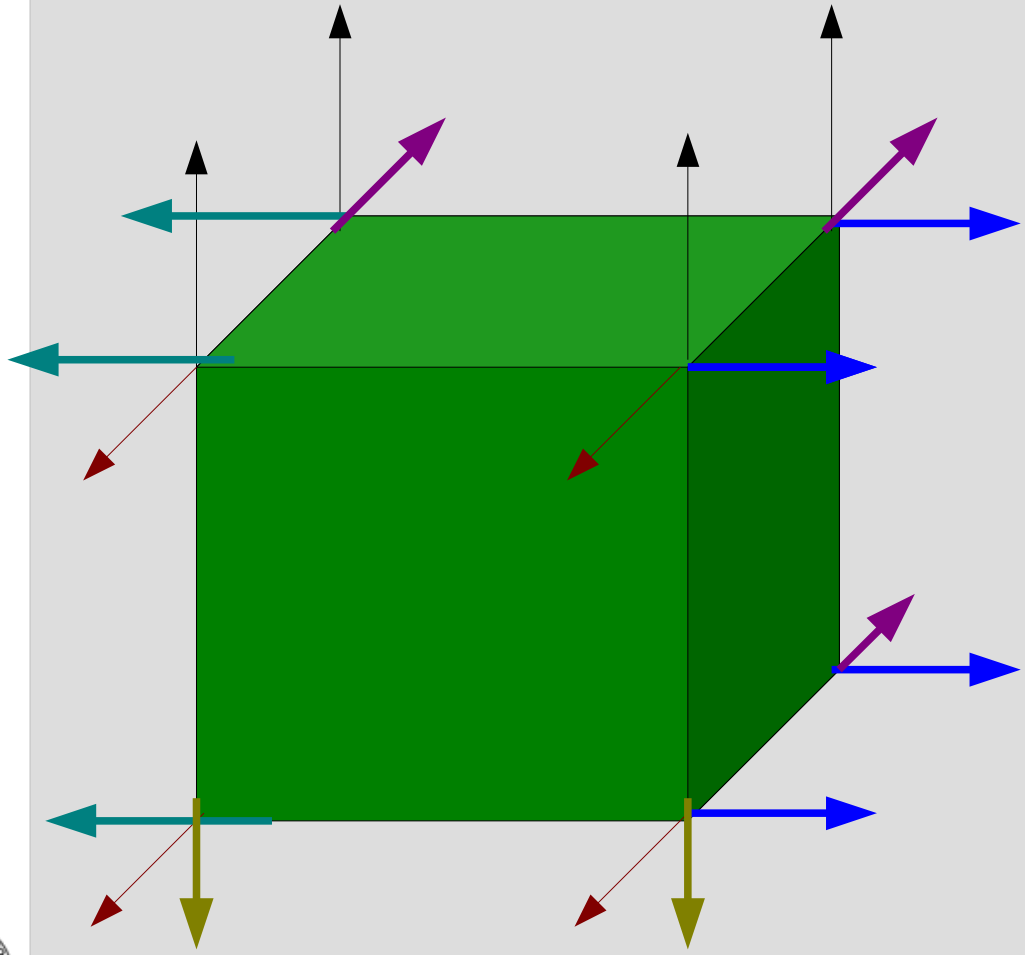
Queste normali vengono quindi utilizzate per valutare l'illuminazione per determinare quale lato di una faccia è illuminata.

Le Normali sono definite utilizzando *Vector3f* che indicano la direzione l'oggetto.

Ad esempio preso *Vector3f* per un *Point3f* (faccia) dato come (0.0f, 0.0f,-1.0f) vorrebbe dire che questa superficie è stata rivolta indietro verso l'origine (0.0f, 0.0f, 0.0f).

NORMALI (3)

Le normali delle facce sono date dalle normali dei suoi vertici



Vector3D(1f,0f,0f)

Vector3D(0f,1f,0f)

Vector3D(0f,0f,1f)

Vector3D(-1f,0f,0f)

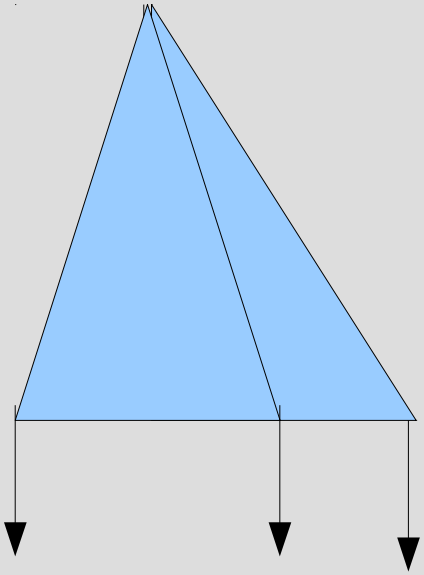
Vector3D(0f,1f,0f)

Vector3D(0f,0f,-1f)

NORMALI (4)

Le normali si possono definire tramite:

- assegnazione (manuale o calcolo)



```
Vector3f [ ]        normals = {
N1, N1, N1,
N2,N2,N2,
....
}
protected  Geometry  createGeometry ( )    {
TriangleArray triangles = new TriangleArray (
faces . length ,
TriangleArray .COORDINATES|GeometryArray.NORMALS);
triangles.setCoordinates(0 , faces ) ;
triangles.setNormals(0 , normals);
Return triangles ;
```

NORMALI (5)

Le normali si possono definire tramite:

- generatore

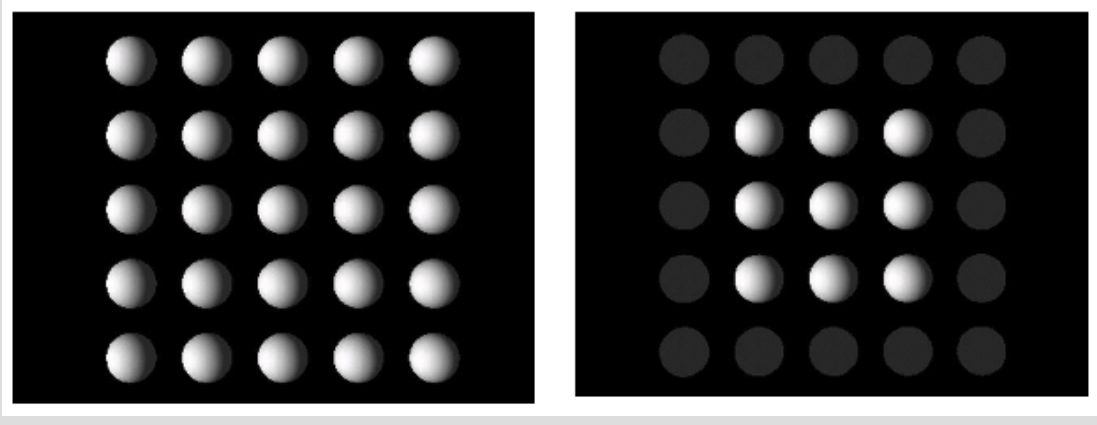
```
Protected Geometry createGeometry ( )   {
   TriangleArray    triangles ;
   triangles = new TriangleArray (faces.length, TriangleArray.COORDINATES) ;
   triangles.setCoordinates(0,faces ) ;
   GeometryInfo gi = new GeometryInfo ( triangles ) ;
   NormalGenerator normalGenerator = new NormalGenerator ( );
   normalGenerator.generateNormals(gi) ;
   return gi.getGeometryArray () ;
}
```

Il NormalGenerator calcola le normali di un oggetto GeometryInfo. Le normali vengono stimate in base a un'analisi delle informazioni degli indici. Se la forma non è indicizzata, il sistema ne creerà una in automatico (approfondite su javadoc).

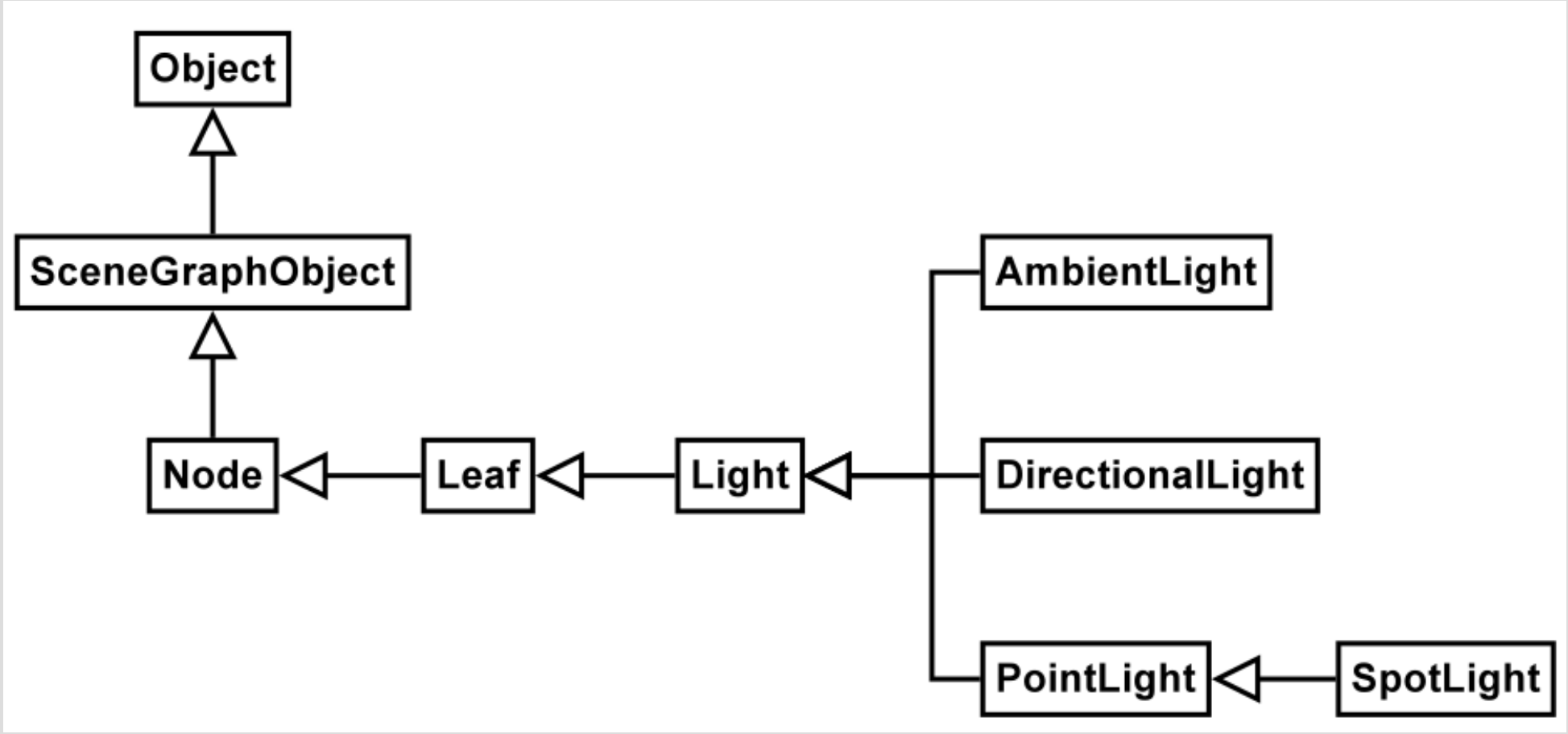
UTILIZZO DELLE LUCI

Per utilizzare sorgenti luminose, servono:

- Un *bound* che indichi la regione in cui la luce ha influenza
- Le normali delle superfici
- Proprietà materiali degli oggetti



Gerarchia di Light



Light (1)

Costruttori di Light:
- *Light()*
- *Light(boolean lightOn, Color3f color)*
- *Light(Color3f color)*

Metodi principali:
- *void setEnable(boolean)*: attiva o disattiva la luce.
- *void setColor(Color3f color)*: imposta il colore della luce.
- *void setInfluencingBounds(Bounds region)*: imposta la regione influenzata dalla luce.

N.B.: la regione d'influenza è svincolata dalla posizione (ma è influenzata dalle trasformazioni).

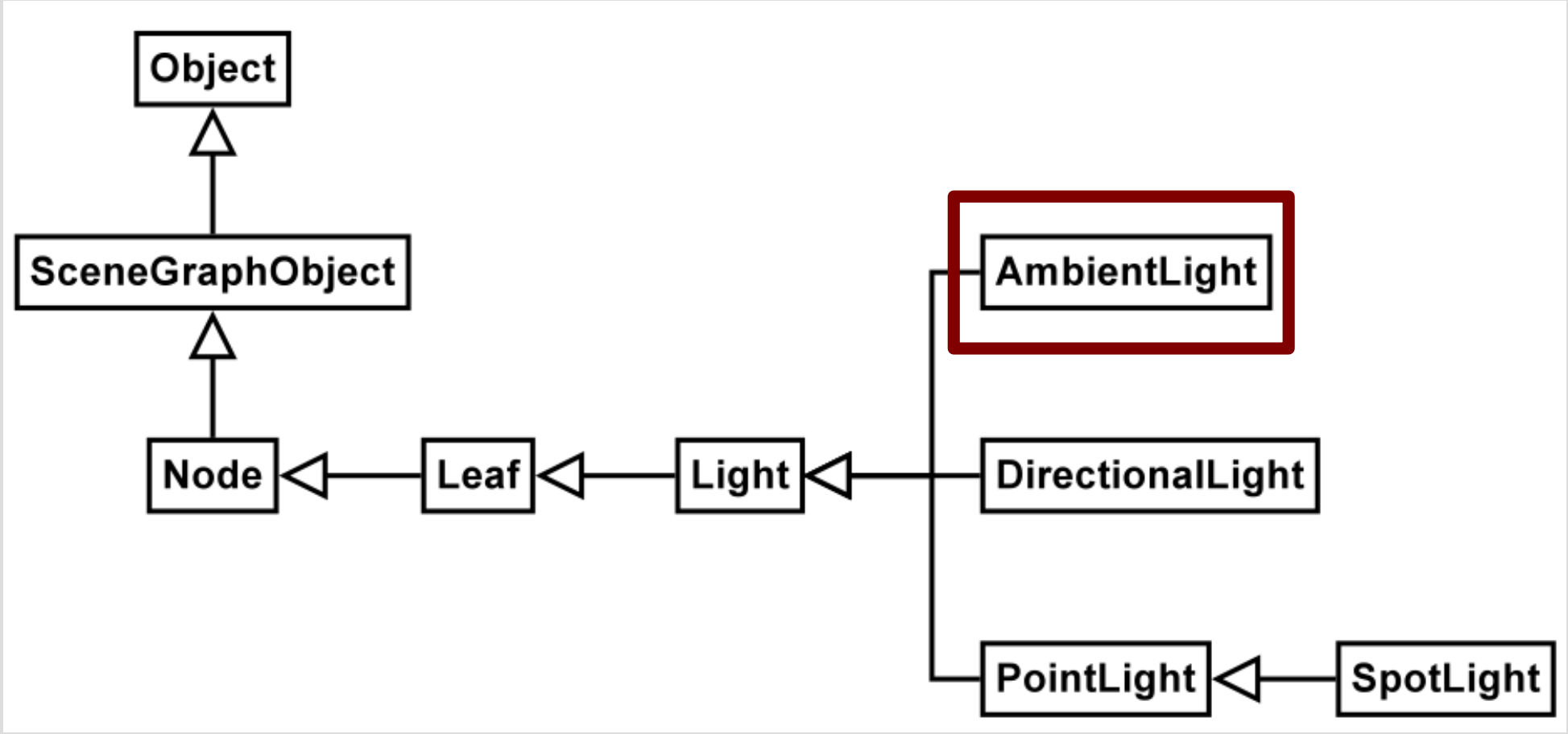
Light (2)

- La luce di default è bianca, accesa, ma priva di zona d'influenza.
- Si deve impostare la zona d'influenza creando degli opportuni oggetti *Bounds*.
- La classe Light è astratta: non possiamo istanziare oggetti di questo tipo.

Esempio di impostazione della zona:

```
Light light ;
...
Bounds bounds = new BoundingSphere(
new Point3d(0.0, 0.0, 0.0), 10.0);
light.setInfluencingBounds(bounds) ;
```

Gerarchia di Light



AmbientLight

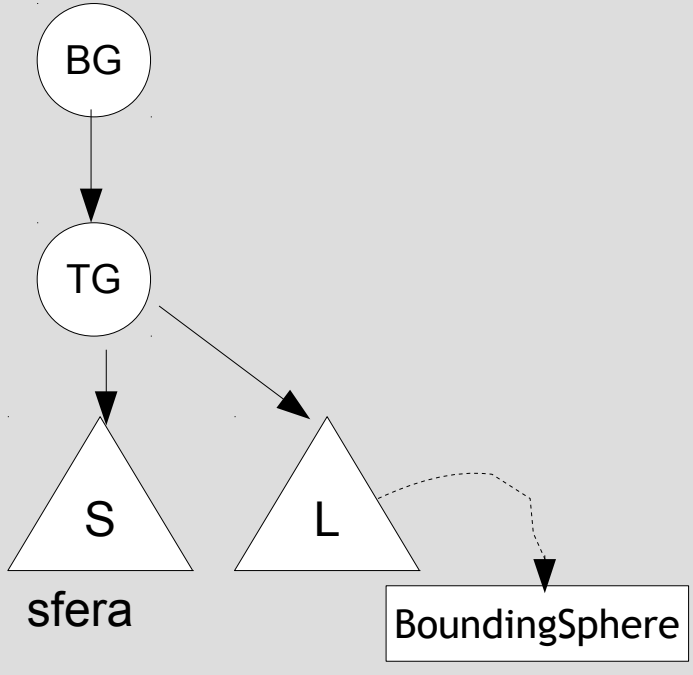
Costruttori di *AmbientLight*:
- *AmbientLight()*
- *AmbientLight(boolean lightOn, Color3f color)*
- *AmbientLight(Color3f color)*

Sorgente di una luce ambientale. La luce ambientale è la luce che sembra provenire da tutte le direzioni.

Esercizio

Inserire una luce ambiente in un mondo con una sfera

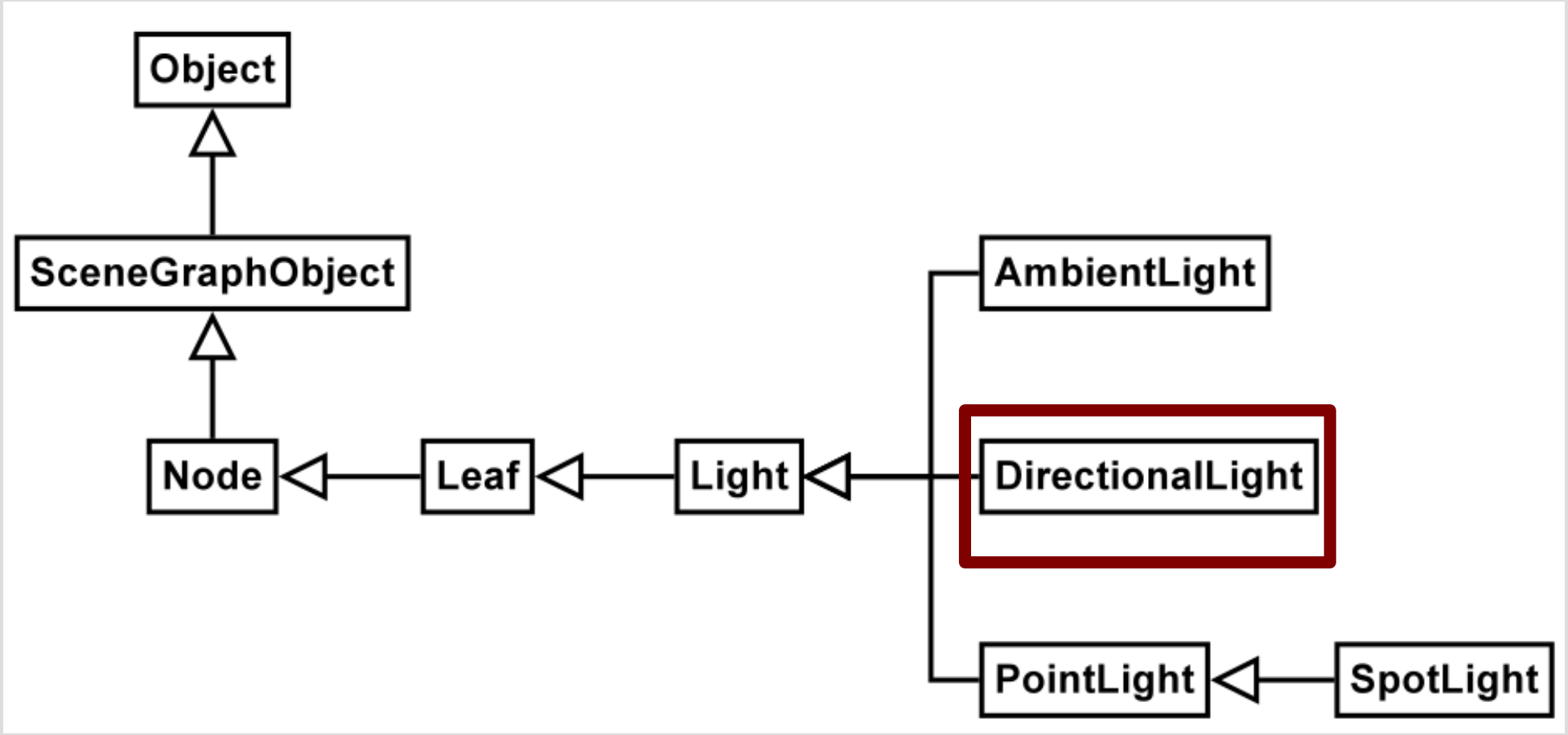
Prova a cambiare il colore della luce



Esercizio

```
new Sphere()

private void ambientLight(){

                /*reazione del bound definisce lo spazio dell'illuminazione mi
                  dice quali sono gli
                  oggetti che posso illuminare*/
                 BoundingSphere bounds = new BoundingSphere(
                                                 new Point3d(0.d,0.d,0.d),10.d);

                 // creazione di una sorgente di luce
                 AmbientLight lightP1 = new AmbientLight();
                      Color3f green = new Color3f(0.0f, 1.0f, 0.0f);
                      lightP1.setColor(green);
                       lightP1.setInfluencingBounds(bounds);
                node.addChild(lightP1); // aggiunta della light al BranchGroup

                }
```

Gerarchia di Light



DirectionalLight (1)

Fornisce illuminazione in un'unica direzione

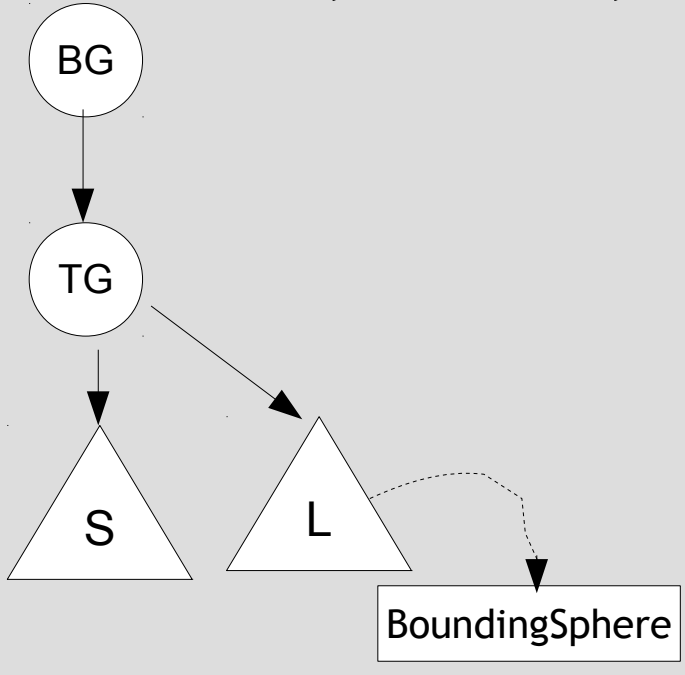
- Non ha posizione ma solo direzione

- L'intensità dei raggi luminosi non varia al variare della distanza tra l'oggetto e la sorgente: tutti i raggi sono paralleli

- Si può usare per simulare sorgenti luminose distanti (ad es. il sole)

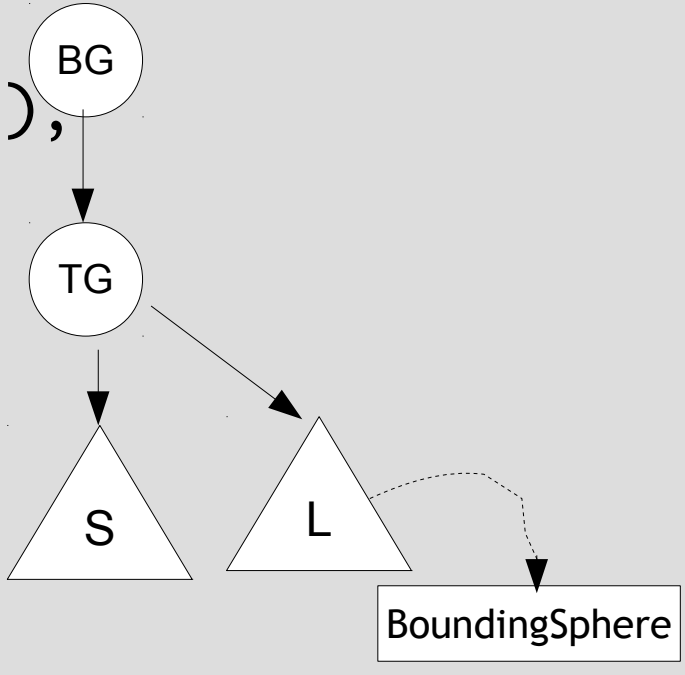
DirectionalLight (2)

```
// aspetto dell'oggetto
Appearance color = new Appearance();
Material material = new Material();
// creazione della Box
Box box = new Box(.5f,.5f,.5f,
Box.GEOMETRY_NOT_SHARED|Box.GENERATE_NORMALS, color);
```



DirectionalLight (3)

```
// creazione del bound
BoundingSphere bounds = new BoundingSphere(new
Point3d(0.0d,0.0d,0.0d),50.0d);
// creazione di una luce direzionale
DirectionalLight lightD1 = new DirectionalLight();

// impostazione del bound
lightD1.setInfluencingBounds(bounds);
// aggiunta al TransformGroup
TG.addChild(lightD1);
TG.addChild(box);
```

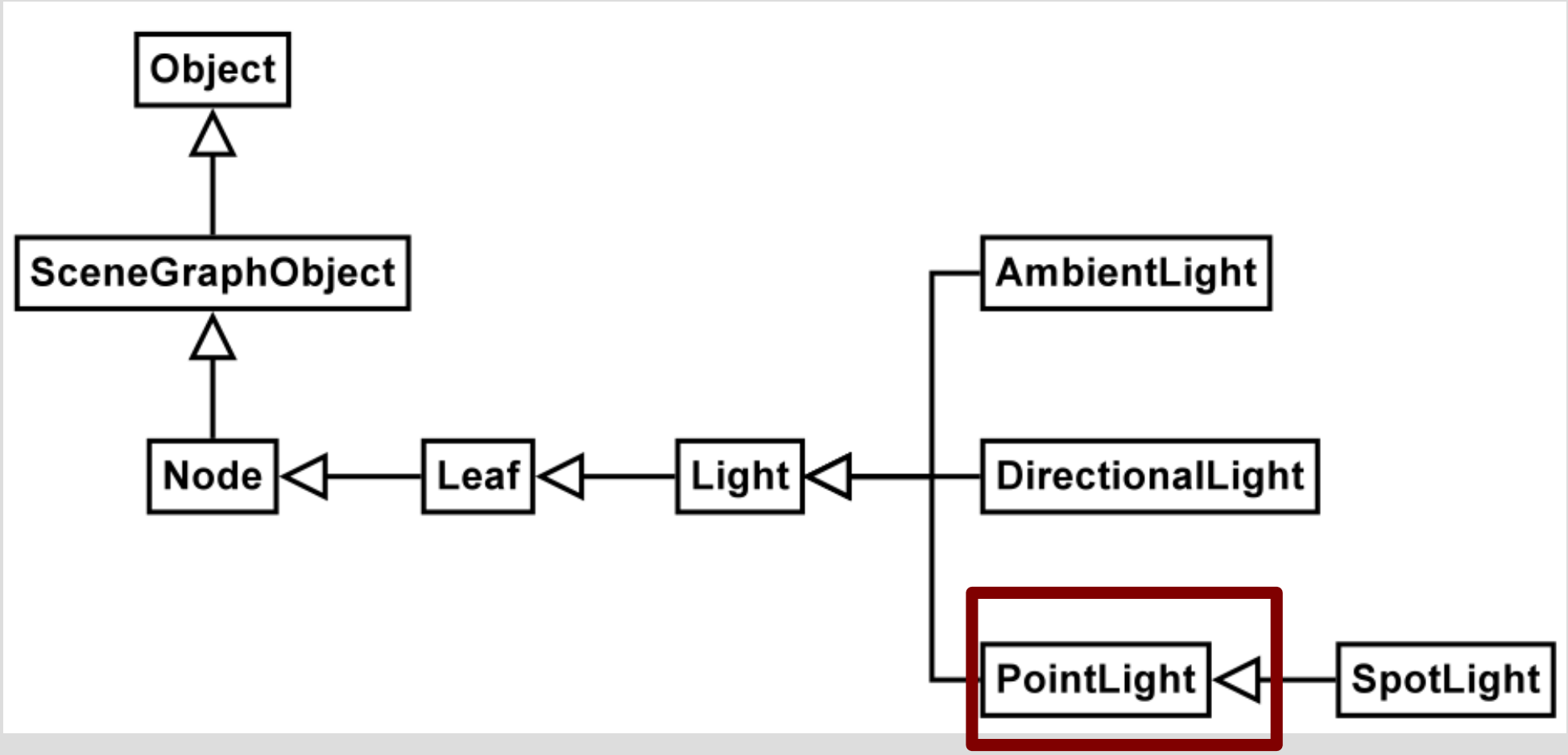


Esercizio

Inserire una luce in un mondo con una sfera

una luce direzionale standard testare la modifica della direzione
(si lavora su setDirection())

Gerarchia di Light



PointLight

- E' un tipo di luce omnidirezionale la cui intensità diminuisce all'aumentare della distanza dalla posizione (funzione di attenuazione)

- Si può usare per simulare sorgenti luminose locali (come lampade)

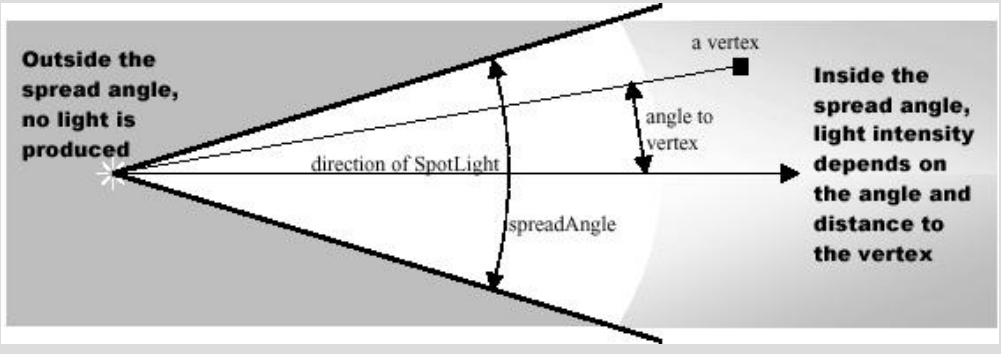
E' possibile definire una propria funzione di attenuazione

SpotLight

E' un sottotipo di PointLight: permette di specificare anche una direzione e un angolo di apertura

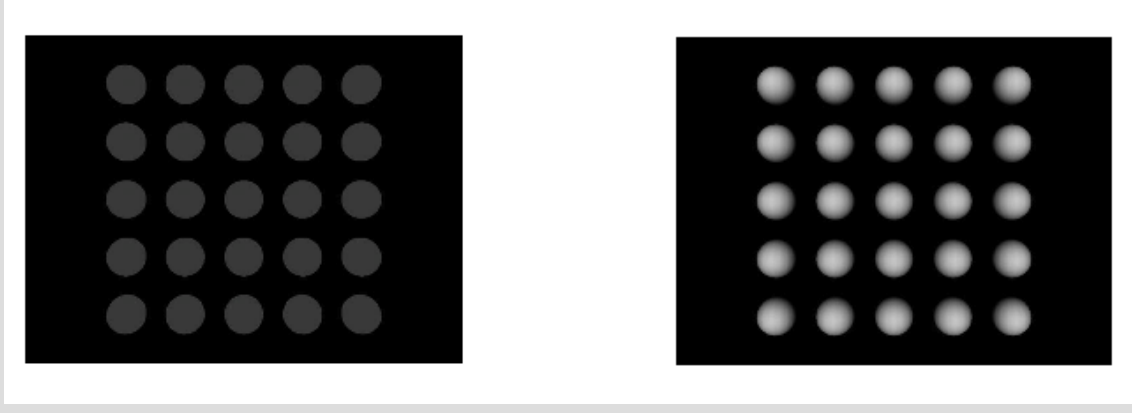
• Produce un cono di raggi luminosi: al di fuori di questo cono non c'è luce

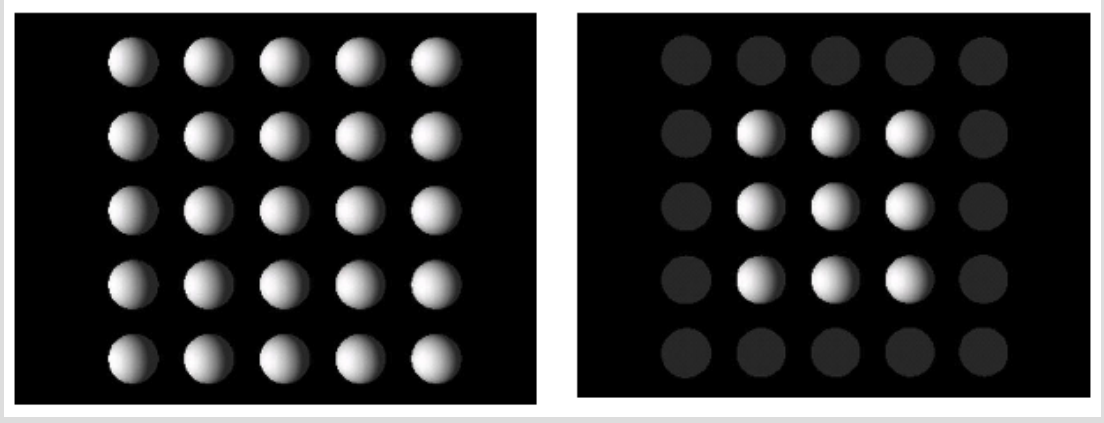
• E' possibile definire un coefficiente di attenuazione: l'intensità è maggiore nel centro del cono ed è minore nei margini



Esercizio
Relazione Finale 3.7

Testare le luci su una matrice 5x5 di sfere



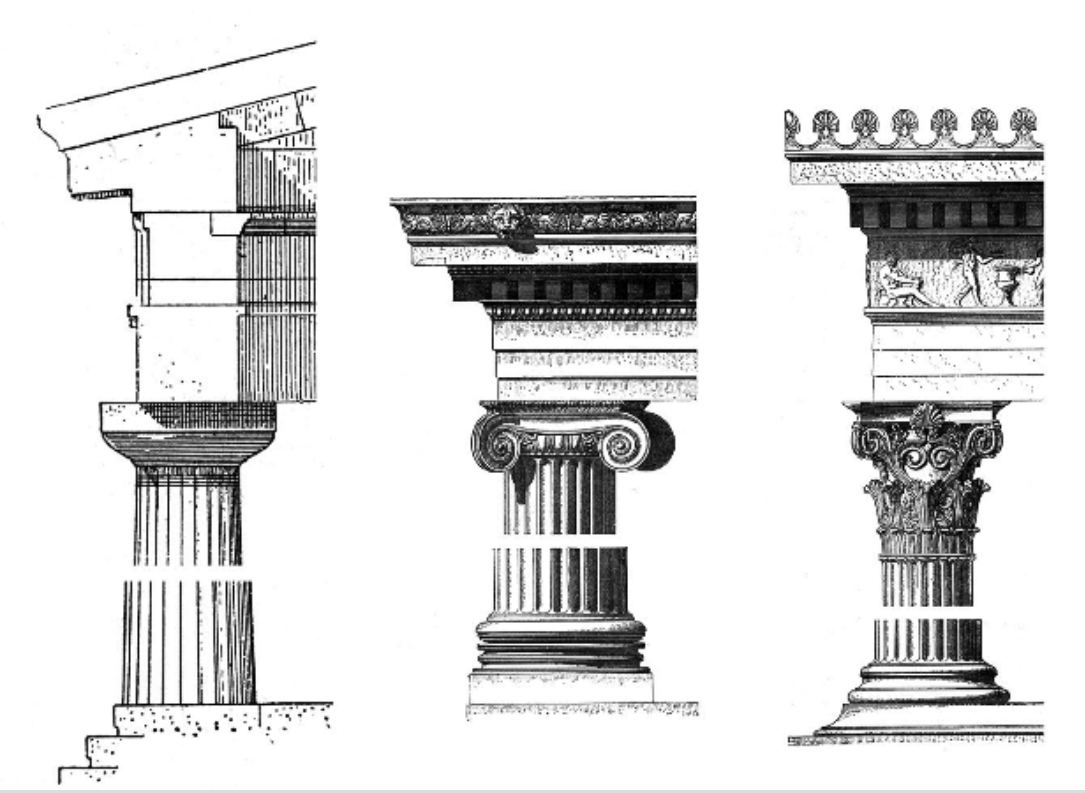


L'oggetto Material definisce la appearance di un oggetto illuminato. Se l'oggetto material in una Appearance è null, gli oggetti che utilizzano quella appearance non saranno illuminati.

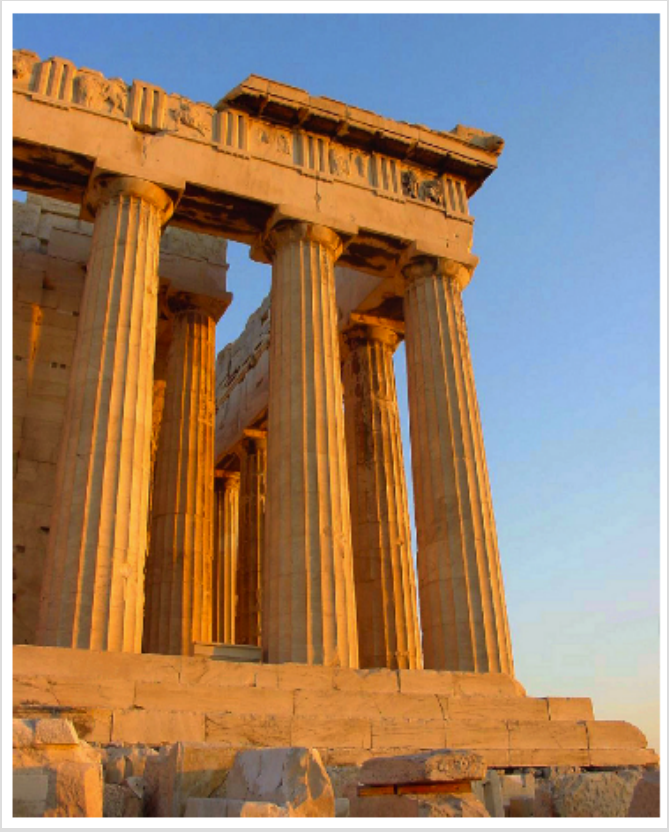
Proprietà della classe Material:

- **Ambient color** - the ambient RGB color reflected off the surface of the material. The range of values is 0.0 to 1.0. The default ambient color is (0.2, 0.2, 0.2).
- **Diffuse color** - the RGB color of the material when illuminated. The range of values is 0.0 to 1.0. The default diffuse color is (1.0, 1.0, 1.0).
- **Specular color** - the RGB specular color of the material (highlights). The range of values is 0.0 to 1.0. The default specular color is (1.0, 1.0, 1.0).
- **Emissive color** - the RGB color of the light the material emits, if any. The range of values is 0.0 to 1.0. The default emissive color is (0.0, 0.0, 0.0).
- **Shininess** - the material's shininess, in the range [1.0, 128.0] with 1.0 being not shiny and 128.0 being very shiny. Values outside this range are clamped. The default value for the material's shininess is 64.
- **Color target** - the material color target for per-vertex colors, one of: AMBIENT, EMISSIVE, DIFFUSE, SPECULAR, or AMBIENT_AND_DIFFUSE. The default target is DIFFUSE.
- La classe Material permette di abilitare o meno la luce.

COLONNA

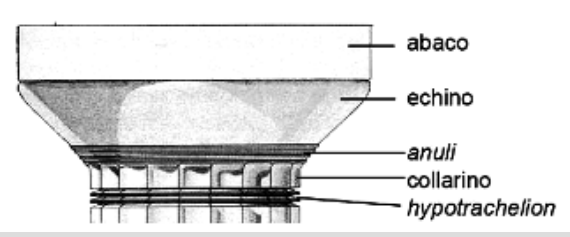


COLONNA DORICA



Cristian Virgili

COLONNA DORICA (1)



Una colonna di ordine Dorico greca è composta da un fusto e un capitello.
- Non ha base, ma poggia direttamente sullo stilobate.

-Il fusto presenta delle scanalature poco profonde unite a spigolo vivo.

- Il numero di scanalature varia tra 16 e 24 (generalmente 20).

- Il fusto ha diametro costante fino ad 1/3 d'altezza e poi si restringe leggermente (entasi).

- Il rapporto tra diametro della base e altezza della colonna è generalmente 1:4.5.

ESERCIZIO
Relazione Finale 3.8

Realizzare la raffigurazione 3D di una **colonna** composta da:

- Fusto: basato su un cilindro (trascurando l'entasi, si può usare la *Primitive* apposita).

- Echino: un tronco di cono (si può ricavare dall'esempio di cilindro già fornito variando i raggi superiore ed inferiore).

- Abaco: un parallelepipedo.

Fornire il tutto di un aspetto opportuno in modo da avere colore coerente con la pietra.

Il risultato deve essere incapsulato in una classe riutilizzabile.

È possibile scomporre in ulteriori classi e si consiglia di procedere in modo da poter collaudare i risultati ad ogni passo (primitive, geometrie, gruppi, aspetto, materiale)

cristian.virgili@uniud.it