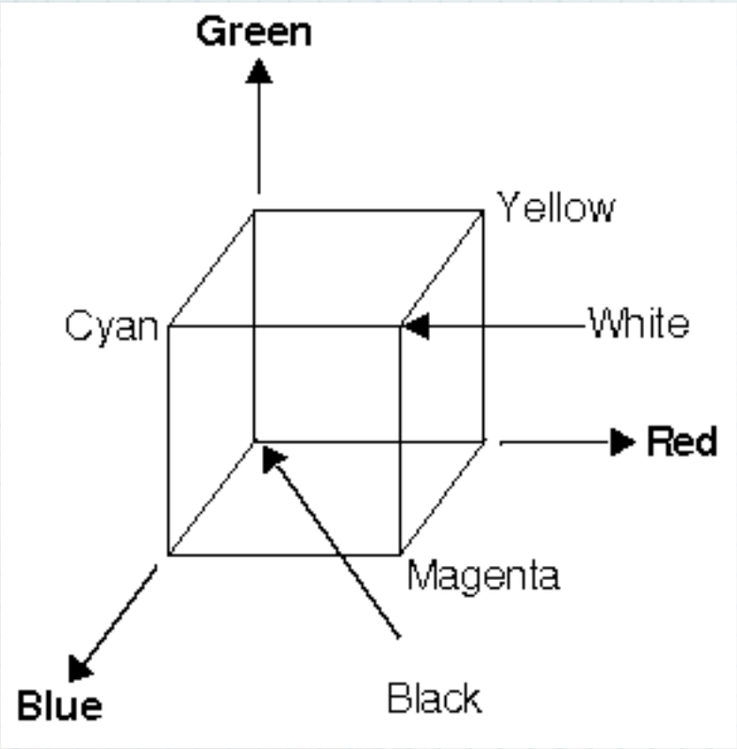
Colors, Textures, Cameras, and Lights

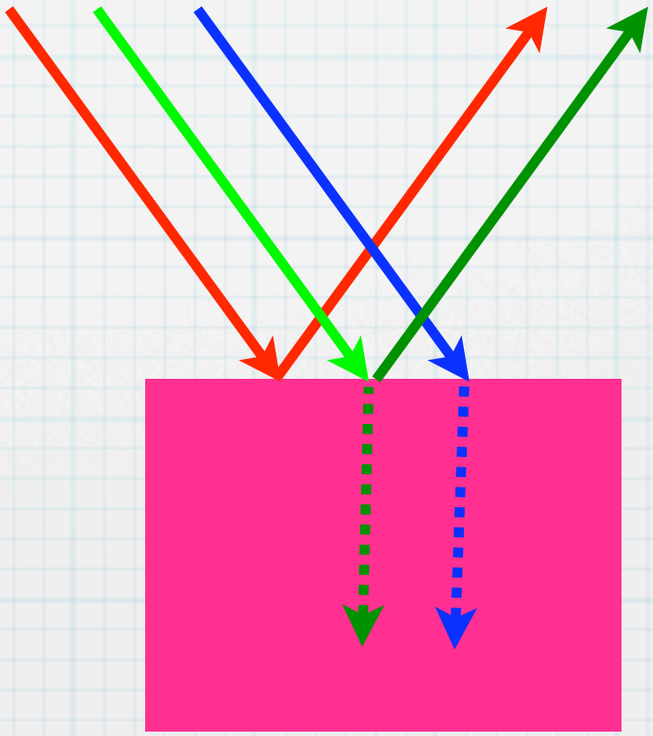
In PLaSM

plasm/psmlib/colors.psm

Luce e Colore

* Spettro elettromagnetico (dip. temperatura)
* Occhio sensibile ad uno spettro dal rosso al violetto
* Simulato da una combinazione "convessa" di tre colori: Rosso, Verde e Blu
* Modello additivo e sottrattivo





(R,G,B)=(1,0,0.5)
(-R,-G,-B)=(0,1,0.5)

Just a Color

CLASS RGBcolor

(r:: isinto:<0,1>; g:: isinto:<0,1>; b:: isinto:<0,1>)

= <r,g,b>;

RED;
Plasm Object of Class rgbcolor and Value < 1 , 0 , 0 >

RED.r;
1

Color Cube

```
DEF COLOR (pol::isPol; col::isRGBColor) =
    pol WITH <'RGBcolor', col.this>;

DEF R_CUBE=CUBOID:<1,1,1> COLOR RED;

R_CUBE;
PolComplex < 3 , 3 > $ < < 'RGBcolor' , < 1 , 0 , 0 > > >
```

Luce e Materiali

* Emissione elettromagnetica (dip. temperatura)
* Quando incide su un materiale può essere assorbita, diffusa, riflessa, ...
* Ogni raggio di luce conta
* Necessaria un' approssimazione:
  Ambient, Diffuse, Specular and Emmissive Color

Diffuse

Luce:

Direzione: $\hat{l}$

Intensità (diffusione): L_d

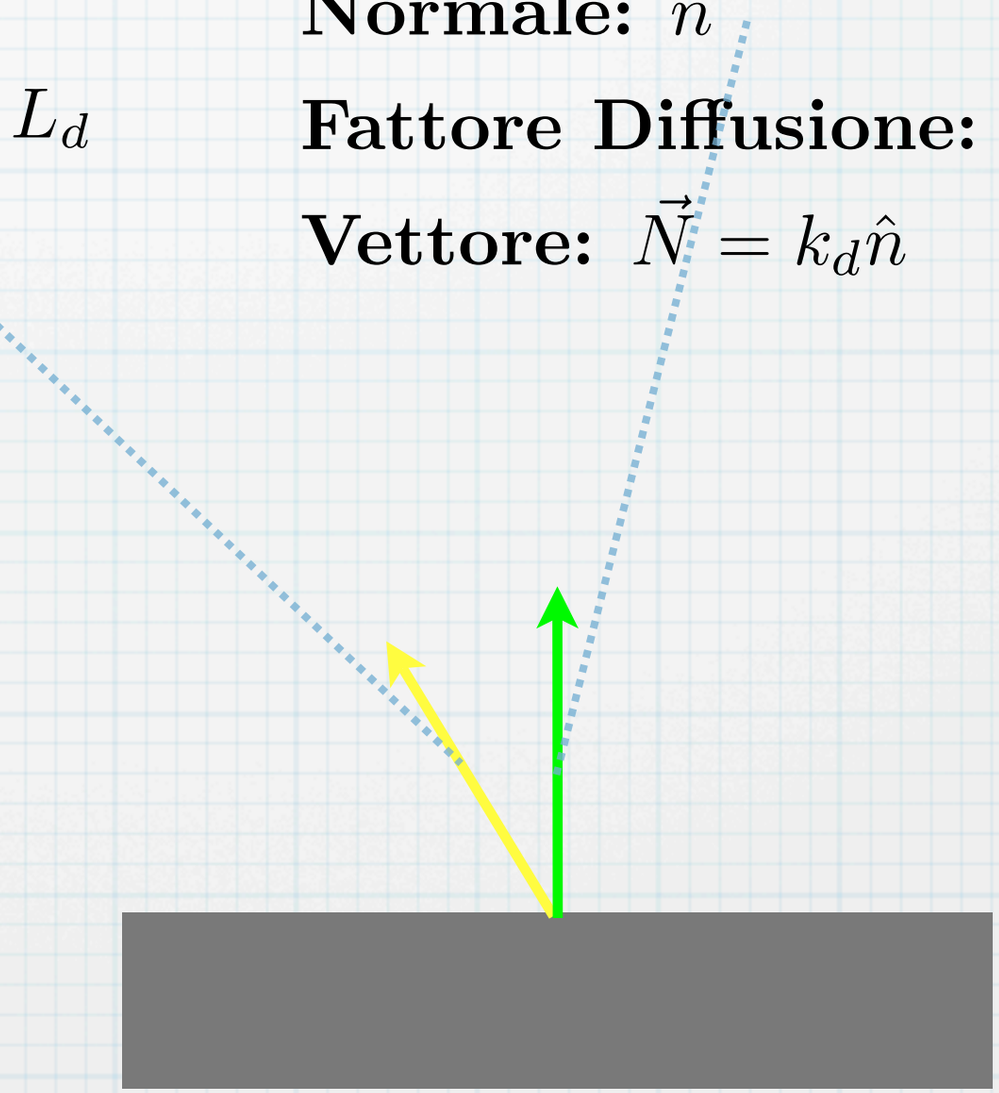
Vettore: $\vec{L_d} = L_d\hat{l}$

Superfice:

Normale: $\hat{n}$

Fattore Diffusione: k_d

Vettore: $\vec{N} = k_d\hat{n}$



Intensità Diffusione:

$$I_d = \vec{L_d} \cdot \vec{N} = L_d k cos\alpha$$

Specular

Luce:

Direzione: $\hat{l}$

Intensità (speculare): L_s

Osservatore:

Direzione: $\hat{v}$

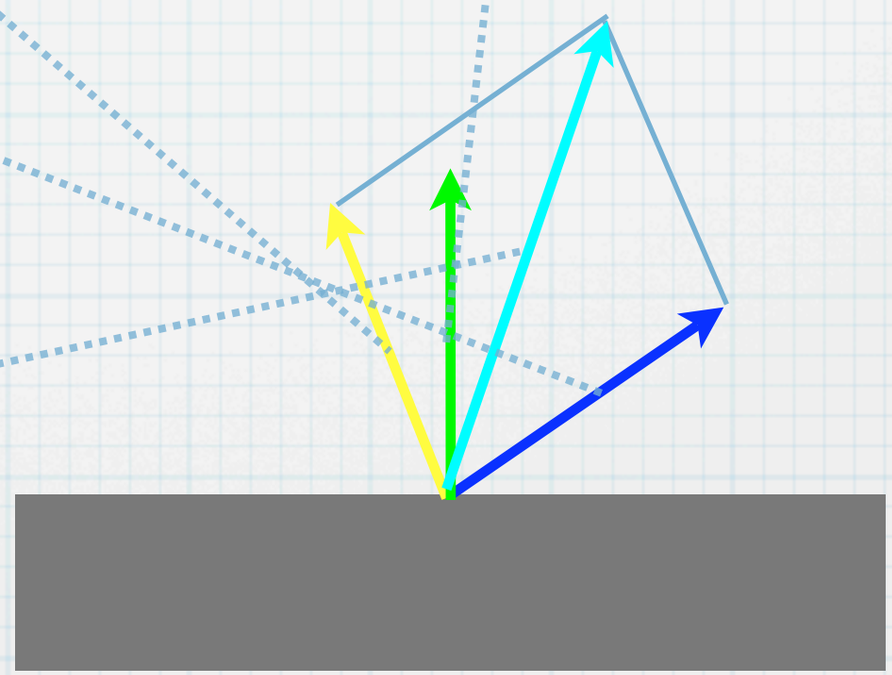
Superfice:

Normale: $\hat{n}$

Fattore Speculare: k_s

Esponente speculare: s

Vettore: $\vec{N} = k_d \hat{n}$



Intensità Speculare:

$$I_s = L_s k_s ((\hat{l} + \hat{v}) \cdot \hat{n})^s$$

Ambient & Emmissive

Luce:

Intensità (ambiente): L_a

Superfice:

Fattore ambiente: k_a

Fattore emmisione: k_e

Intensità:

Ambiente: $I_a = L_a k_a$

Emissione: $I_e = k_e$

Totale

Per ogni luce:

$$I_{luce} = I_a + I_d + I_s$$

Globale:

$$I = I_e + I_a + \sum I_{luce}$$

VRML Material

```
CLASS BaseMaterial (
    diffuse::    isRGBColor;
    specular::   isRGBColor;
    ambient::    isinto:<0,1>;
    emissive::   isRGBColor;
    shininess::  isinto:<0,1>;
    transparency::isinto:<0,1> )
< diffuse, specular, ambient, emissive, shininess, transparency >;
```

CLASS MyMaterial(color::isRGBcolor)

ISA BaseMaterial =

<color,color,0.3,BLACK,0.4,0>;

DEF MY_RED = MyMaterial:Red;

MY_RED;

Plasm Object of Class mymaterial and Value ...

MY_RED.diffuse;

Plasm Object of Class rgbcolor and Value < 1 , 0 , 0 >

```
DEF MATERIAL (pol::isPol; mat::isBaseMaterial) =
    pol WITH <'VRMLmaterial',
    <mat.diffuse.this,
    mat.specular.this,
    mat.ambient,
    mat.emissive.this,
    mat.shininess, mat.transparency> >;
```

```
DEF MR_CUBE=CUBOID:<1,1,1> MATERIAL MY_RED;

PolComplex < 3 , 3 >  $ < < 'VRMLmaterial' , < < 1 , 0 , 0
> , < 1 , 0 , 0 > , 0.3 , < 0 , 0 , 0 > , 0.4 , 0.3 > > >

CLASS SimpleMaterial(color::isRGBcolor) ISA BaseMaterial

CLASS Transparentmaterial
        (color::isRGBColor; transparency::isinto:<0,1>)
        ISA BaseMaterial
```

VRML Textures

```
CLASS FullTexture (
    url::        IsString;
    repeatS::    IsBool;
    repeatT::    IsBool;
    center::     IsPoint;
    rotation::   IsReal;
    scale::      IsVect;
    translation:: IsVect ) =
  <url, repeatS, repeatT, center, rotation, scale, translation>;
```

```
CLASS BaseTexture (
    url:: IsString;
    repeatS:: IsBool;
    repeatT:: IsBool )
  ISA FullTexture = < url, repeatS, repeatT, C, R, S, T >
  WHERE C = <0,0>,
      R = 0,
      S = <1,1>,
      T = <0,0>
  END;
```

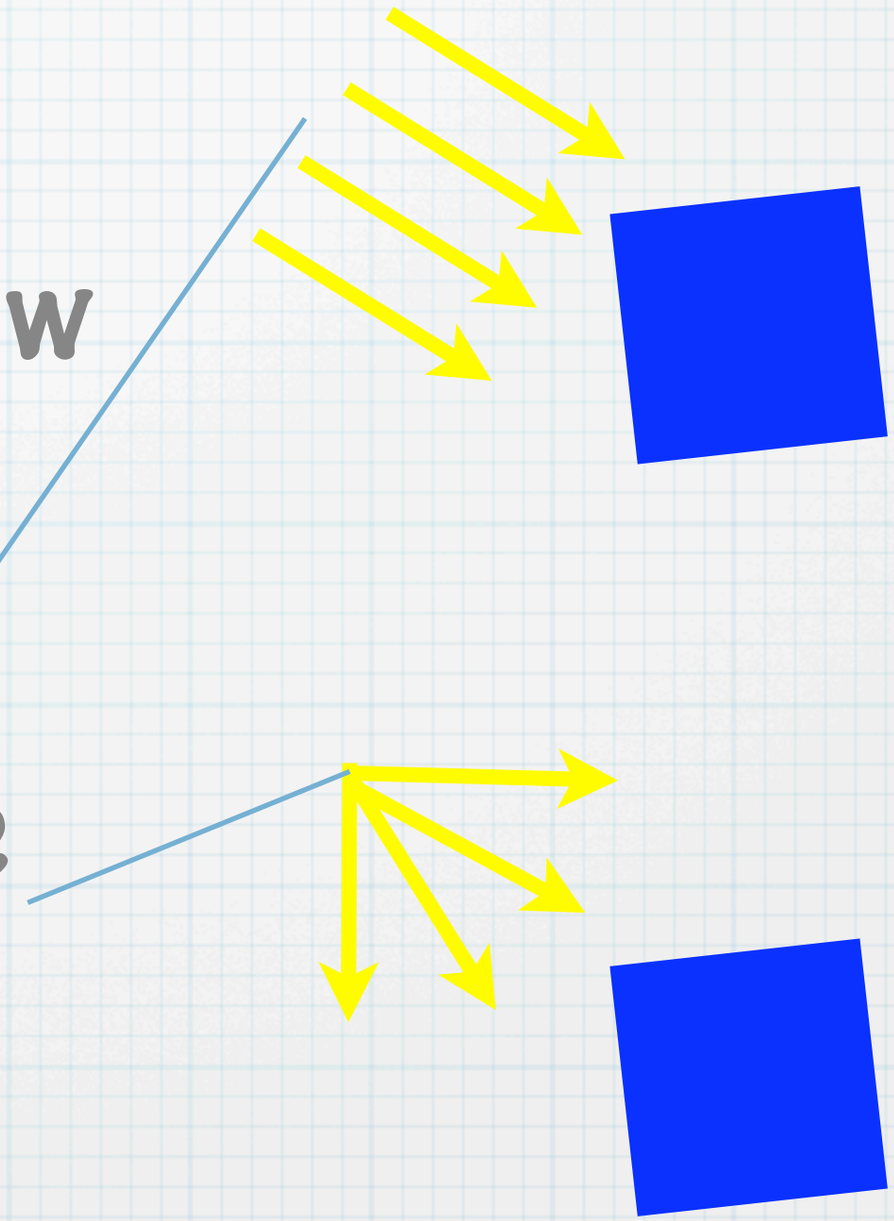
```
CLASS SimpleTexture ( url:: IsString )
  ISA BaseTexture = < url, repeatS, repeatT >
  WHERE
    repeatS = FALSE,
    repeatT = FALSE
  END;
```

```
DEF TEXTURE (pol::isPol; tex::isFullTexture) =
    pol WITH <'VRMLtexture', < .... > >;
```

```
DEF APPEARANCE
(pol::isPol; mat::isBaseMaterial; fulltex::isFullTexture) =
    pol MATERIAL mat TEXTURE fulltex;
```

Geometria luce

* Position=(x,y,z,w)
* Coordinata omogenea w
* w=0: Luce Direzionale (all'infinito)
* w!=0: Luce puntiforme
* Posizione e direzione? Spotlight



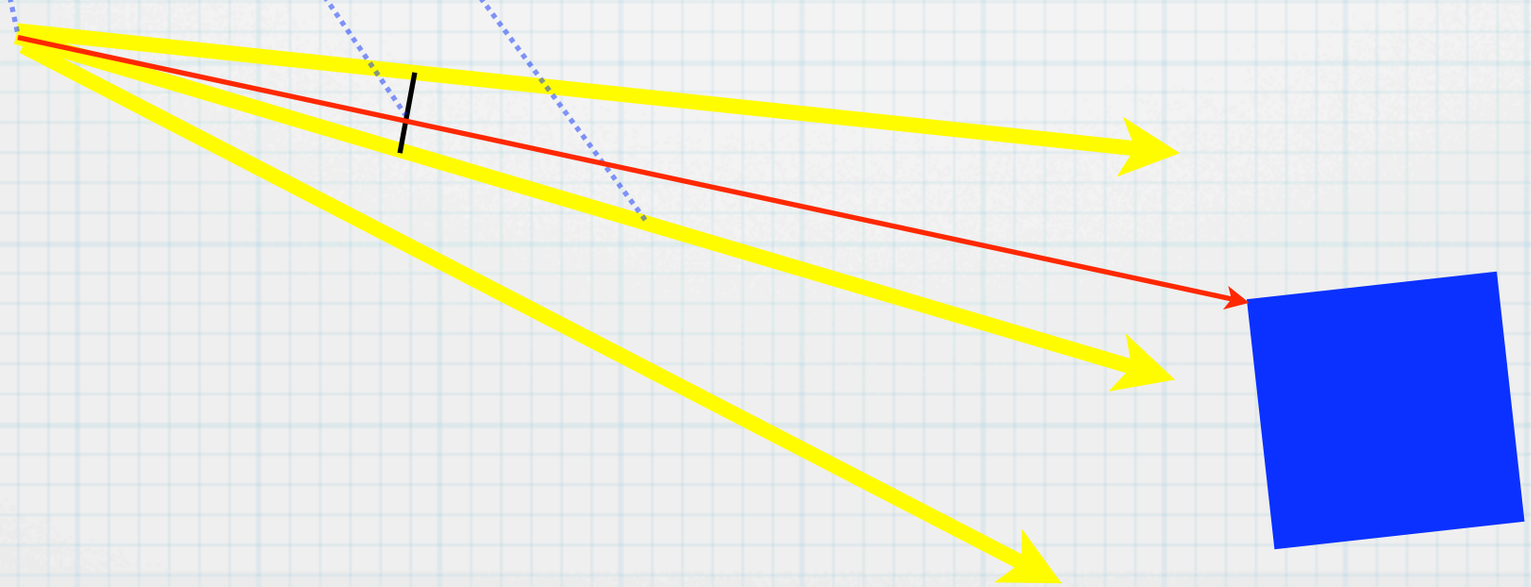
* GL_POSITION è un punto proprio (w!=0)
* GL_SPOT_DIRECTION è la direzione
* GL_SPOT_CUTOFF è l'angolo di apertura
* GL_SPOT_EXPONENT fattore attenuazione angolo

Direzione spotlight (SPOT_DIRECTION): $\hat{l}$

Versore luce_vertice: $\hat{v} = \frac{V-\vec{POSITION}}{|V-\vec{POSITION}|}$

Esponente: e

$S = (\hat{v} \cdot \hat{l})^e$



Attenuation

* La radiazione eletromagnetica si attenua proporzionalmente a $1/r^2$
* Esperienza comune ci indica che non è apparentemente vero:
  * Fonte non puntiforme
  * Correzione del cervello

Attenuazione

Distanza vertice posizione luce: d

GL_CONSTANT_ATTENUATION k_c

GL_LINEAR_ATTENUATION k_l

GL_QUADRATIC_ATTENUATION k_q

$$A = \frac{1}{k_c + k_d + k_q}$$

Calcolo complessivo: Luci, Spotlight, attenuazione

$$I = I = I_e + I_a + \sum ASI_{luce}$$

$$I = k_e + L_a k_a + \sum_{luce} \left(L_a k_a + L_d k_d (\hat{l} \cdot \hat{n}) + L_s k_s \left((\hat{l} + \hat{v}) \cdot \hat{n} \right) \right) \frac{1}{k_c + k_d + k_q} (\hat{v} \cdot \hat{l})^e$$

Tutor lightmaterial (2)

VRML Lights

```
CLASS GenericLight (
    type::          isinto:<0,2>;
    % 0 = PointLight,  1 = DirectionalLight,  2 = SpotLight  %
    appearance::   isGenericLightAppearance;
    geometry::     isGenericLightGeometry ) =
  <type, appearance, geometry>;
```

```
CLASS GenericLightAppearance (
    color::       OR~[isRGBColor, isNull] ;
    intensity::   OR~[isReal, isNull];
    ambient::     OR~[isReal, isNull];
    ison::        OR~[isBool, isNull]) =
  <lcolor, lintensity, lambient, lison>
```

```
WHERE
  lcolor     = IF:<isNull,  K:WHITE, ID>:color,
  lintensity = IF:<isNull,  K:1,     ID>:intensity,
  lambient   = IF:<isNull,  K:0,     ID>:ambient,
  lison      = IF:<isNull,  K:TRUE,  ID>:ison
END;
```

```
CLASS GenericLightGeometry (
    location::     OR~[isVect, isNull];
    direction::    OR~[isVect, isNull];
    attenuation::  OR~[isVect, isNull];
    radius::       OR~[isReal, isNull];
    beamWidth::    OR~[isReal, isNull];
    cutOffAngle::  OR~[isReal, isNull] ) =
  <llocation, ldirection, lattenuation, lradius, lbeamWidth, lcutOffAngle>
```

```
WHERE
    llocation    = IF:<isNull, K:<0,0,0>,   ID>:location,
    ldirection   = IF:<isNull, K:<0,0,-:1>, ID>:direction,
    lattenuation = IF:<isNull, K:<1,0,0>,   ID>:attenuation,
    lradius      = IF:<isNull, K:100,       ID>:radius,
    lbeamWidth   = IF:<isNull, K:(PI/2),    ID>:beamWidth,
    lcutOffAngle = IF:<isNull, K:(PI/4),    ID>:cutOffAngle
  END;
```

```
CLASS BasePointLight (
    pointAppearance::   isSeq;
    pointGeometry::     isSeq  )
  ISA GenericLight = < type, appearance, geometry >
  WHERE
    type = 0,
    appearance = GenericLightAppearance:pointAppearance,
    geometry =
    GenericLightGeometry:<S1:pointGeometry,<>,S2:pointGeometry,S3:pointGeometry,<>,<>>
  END;
```

```
CLASS BaseDirLight (
    dirAppearance::   isSeq;
    dirGeometry::    isSeq  )
  ISA GenericLight = < type, appearance, geometry >
  WHERE
    type = 1,
    appearance = GenericLightAppearance:dirAppearance,
    geometry = GenericLightGeometry:<<>,S1:dirGeometry,<>,<>,<>,<>>
  END;
```

```
CLASS BaseSpotLight (
    spotAppearance::   isSeq;
    spotGeometry::     isSeq  )
  ISA GenericLight = < type, appearance, geometry >
  WHERE
    type = 2,
    appearance = GenericLightAppearance:spotAppearance,
    geometry = GenericLightGeometry:spotGeometry
  END;
```

```
DEF LIGHT (pol::ispol; light::isGenericLight) =
        (copy:pol) with <'VRMLlight',
    <light.type,
    <light.appearance.lcolor.this,
    light.appearance.lintensity,
    light.appearance.lambient, light.appearance.lison>,
    light.geometry.this>>;
```

VRML Cameras

CLASS BaseCamera (

position:: OR~[isVect, isNull];

orientation:: OR~[isVect, isNull];

fieldOfView:: OR~[isReal, isNull];

description:: isString) =

<cposition, corientation, cfieldOfView, description>

```
WHERE
  cposition    = IF:<isNull, K:<0,0,10>,  ID>:position,
  corientation = IF:<isNull, K:<0,0,1,0>, ID>:orientation,
  cfieldOfView = IF:<isNull, K:(PI/4),    ID>:fieldOfView
END;
```

```
CLASS SimpleCamera (
    position::     OR~[isVect, isNull];
    description::  isString )
    ISA BaseCamera = <position, <>, <>, description>;

DEF CAMERA (pol::ispol; camera::isBaseCamera) =
    (copy:pol) with <'VRMLcamera', camera.this>;
```