CG_LEI_2024_PL10

prof. André Perrotta, prof. Hugo Amaro

Cálculo da cor dos vértices com iluminação em OpenGl atual

obs: modifique os valores das variáveis de acordo com configuração que está a ser utilizada na app do OpenFrameworks

Inicialização do vértice a ser calculado:

- tamanho da tela
- pos (x, y, z, 1)
- vetor normal (x, y, z, 0)

In [1]:
```
import math as m
import numpy as np
```

In [2]:
```
gw = 1024
gh = 768
vertexPos = np.array([-512, -384, 0, 1])
vertexNormal = np.array([0, 0, 1, 0])
```

Inicialização dos parâmetros da luz e material

- posição/direção (x, y, z, 0/1)
- luz I_ambiente
- luz I_difusa
- luz I_specular
- luz atenuação_constante
- luz atenuação_linear
- luz atenuação_quadrática
- luz spot direction
- luz spot_cuttof
- luz spot_coefficient
- material k_ambiente
- material k_difusa
- material k_specular
- material shininess_ns

In [3]:
```
lightPos = np.array([0, 0, 90, 1]);
```

```
Lamb = np.array([1, 1, 1, 0]);
Ldif = np.array([1, 1, 1, 0]);
Lspec = np.array([1, 1, 1, 0]);
atC = 1.;
atLin = 0.0001;
atQuad = 0.00001;

#lightSpotDir = [0 0 0 0];
#spotCutoff = 90;
#spotCoef = 60;

#bronze
matAmb = np.array([0.2125, 0.1275, 0.054, 0]);
matDif = np.array([0.714, 0.4284, 0.18144, 0]);
matSpec = np.array([0.393548, 0.271906, 0.166721, 0]);
ns = 0.2*128;

#custom
#matAmb = np.array([0, 0, 0, 0]);
#matDif = np.array([0, 0, 0, 0]);
#matSpec = np.array([1, 1, 1, 0]);
#customMatCoef = 1
```

Inicialização da matriz modelview

o lookat vai transformar a matriz modelview, para fazer os cálculos precisaremos transformar os vetores e posições que utilizamos com a matriz modelview.

In [4]:
```
mview1 = np.array([(1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, -665.108), (0, 0
mview2 = np.array([(1, 0, 0, 0), (0, 0.514496, 0.857493, 0), (0, -0.85749
mview3 = np.array([(0.882353, 0.470588, 0, 0), (-0.220182, 0.412842, 0.88
mview4 = np.array([(0.882353, -0.470588, 0, 0), (0.220182, 0.412842, 0.88
mview5 = np.array([(1, 0, 0, 0), (0, 0.8, 0.6, 0), (0, -0.6, 0.8, -960),

mview = mview5
print("modelview = \n", mview)
```

```
modelview =
 [[ 1.0e+00  0.0e+00  0.0e+00  0.0e+00]
 [ 0.0e+00  8.0e-01  6.0e-01  0.0e+00]
 [ 0.0e+00 -6.0e-01  8.0e-01 -9.6e+02]
 [ 0.0e+00  0.0e+00  0.0e+00  1.0e+00]]
```

Calcular os vetores necessários e normalizá-los:

- normal do vértice
- vetor luzPos - vérticePos (vlv)
- vetor obsPos - vérticePos (vov)

No caso da luz ser direcional, o vetor (luzPos-vérticePos) é desnecessário, usamos o próprio luzPos

O vetor (obsPos - vérticePos) é calculado de forma diferente para

glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, true/false);

quando **true**:

- obs = [0 0 0 1]

quando **false**, só ficamos com a componente z:

- obs = (vertexPos) * [1 1 0 1]
- vov vai sempre resultar em [0 0 1 0]

mais info sobre isso em https://registry.khronos.org/OpenGL-Refpages/gl2.1/xhtml/glLightModel.xml

In [5]:
```
vertexNormal = np.matmul(mview, vertexNormal)
vertexNormal = vertexNormal/np.linalg.norm(vertexNormal)
print("vertexNormal = ", vertexNormal)

lightPos = np.matmul(mview, lightPos)
vertexPos = np.matmul(mview, vertexPos)

if (lightPos[-1] == 0):
    vlv = lightPos
else:
    vlv = (lightPos - vertexPos)

vlv = vlv/np.linalg.norm(vlv)
print("vlv = ", vlv)

#localViewer
#true
#obs = np.array([0, 0, 0, 1])

#false
obs = vertexPos*np.array([1, 1, 0, 1])

vov = (obs - vertexPos)
vov = vov/np.linalg.norm(vov)
print("vov = ", vov)
```

```
vertexNormal =  [0.  0.6 0.8 0. ]
vlv =  [ 0.79220526  0.55887606 -0.2450885   0.        ]
vov =  [0. 0. 1. 0.]
```

Cálculo da componente ambiente:

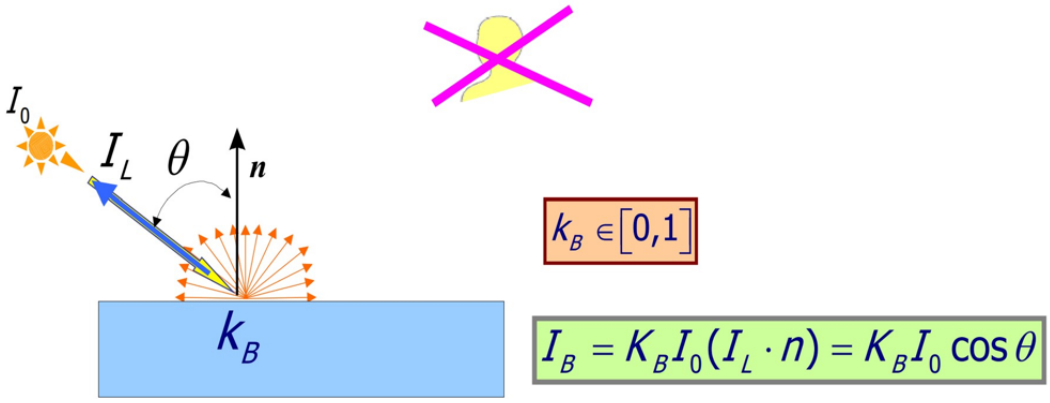
- $I_{amb} = Luz_{amb} \times Material_{amb}$

In [6]:
```
Iamb = Lamb*matAmb
print("Iamb = ", Iamb)
```

```
Iamb =  [0.2125 0.1275 0.054  0.    ]
```

Cálculo da componente difusa:

- $I_{dif} = Luz_{dif} \times Material_{dif} \times cos(\theta)$
- θ = ângulo entre normal e vetor(luzPos - verticePos)
- $\theta \in [0^\circ, 90^\circ]$



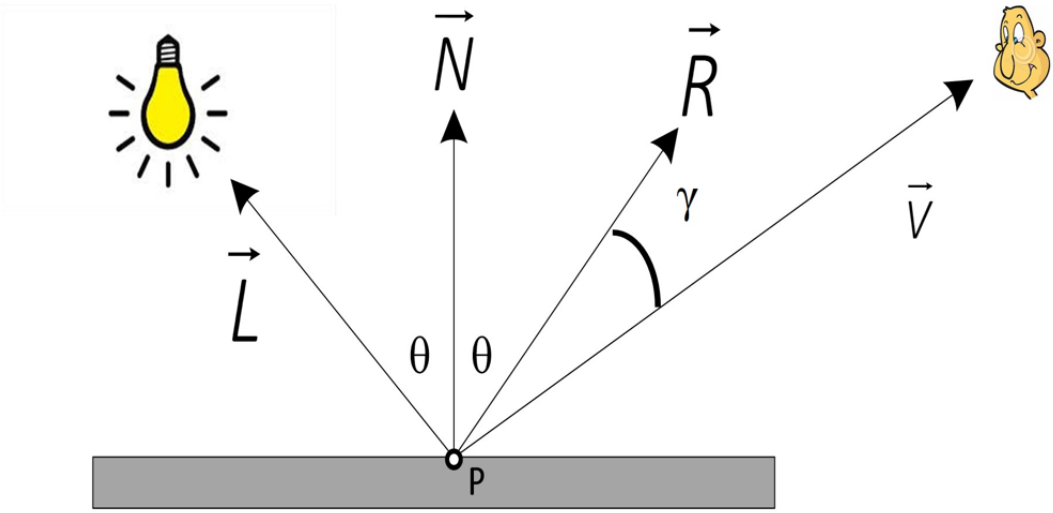
In [7]:
```
cosTheta = np.dot(vertexNormal, vlv);
theta = m.acos(cosTheta);
if (theta > m.pi*0.5):
    Idif = np.array([0, 0, 0, 0])
else:
    Idif = (Ldif*matDif)*cosTheta

print("Idif = ", Idif)
```

```
Idif =  [0.09942795 0.05965677 0.0252664  0.        ]
```

Cálculo da componente especular

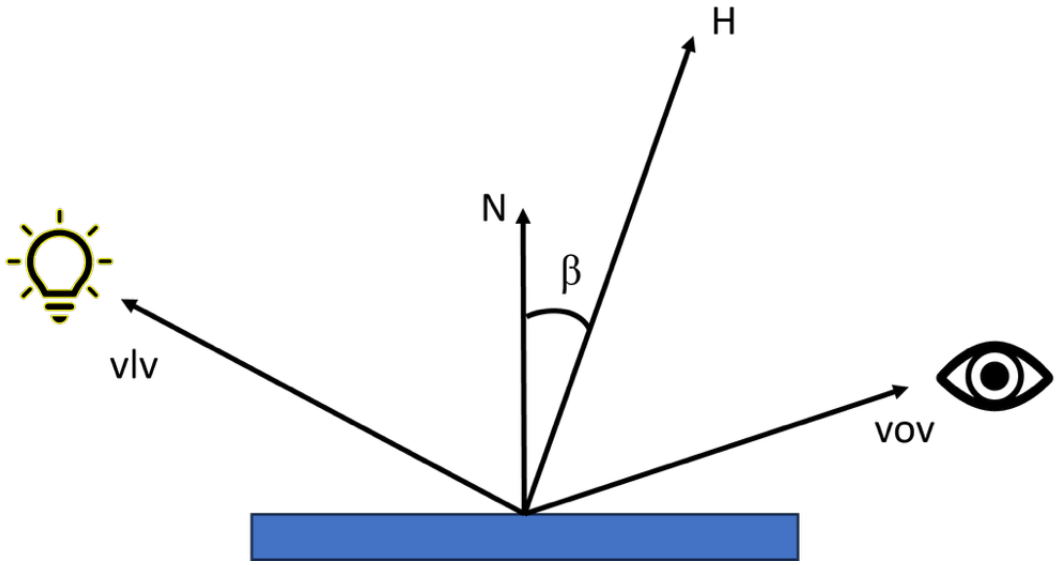
O modelo de Phong define o cálculo da componente especular como:

- $Ispec = Luz_{spec} \times Material_{spec} \times \cos(\gamma)^{ns}$
- γ é o ângulo entre o vetor saída da luz com a normal, e o observador
- $\gamma \in [0^\circ, 90^\circ]$



Contudo, a versão autal do OpenGl, utilizada no OpenFrameworks, utiliza uma variação do modelo de Phong para o cálculo da componente especular:

- $Ispec = Luz_{spec} \times Material_{spec} \times \cos(\beta)^{ns}$
- β é o ângulo entre o vetor **halfway** do vetor (luz-pos) e (obs - pos), e a normal
- $\beta \in [0^\circ, 90^\circ]$



$$\vec{H} = \frac{\vec{vlv} + \vec{vov}}{|\vec{vlv} + \vec{vov}|}$$

In [8]:
```
H = (vlv + vov)
H = H/np.linalg.norm(H)
cosBeta = np.dot(vertexNormal, H)
beta = m.acos(cosBeta)
if (beta > m.pi*0.5 or beta < 0):
    Ispec = np.array([0, 0, 0, 0])
else:
    Ispec = (Lspec*matSpec)*(cosBeta**ns)

print("Ispec = ", Ispec)
```

```
Ispec =  [0.00040551 0.00028017 0.00017179 0.        ]
```

Calculo da atenuação:

$at = at_{constante} + d \times at_{linear} + d^2 \times at_{quadrática}$

só vale para luz pontual ou foco!

In [9]:
```
d = np.linalg.norm(lightPos - vertexPos);

if lightPos[-1] == 0:
    at = 1
else:
    at = atC + atLin*d + atQuad*d*d
```

Calculo final:

- $I_{final} = (I_{amb} + I_{dif} + I_{spec})/at$

In [10]:
```
I = (Iamb + Idif + Ispec)/at

print("I = ", I)
```

```
I =  [0.05958709 0.03575929 0.01515525 0.        ]
```

In [ ]:

In [11]:
```
! jupyter nbconvert --to html vertex_color_calculator.ipynb
```

```
[NbConvertApp] Converting notebook vertex_color_calculator.ipynb to html
[NbConvertApp] WARNING | Alternative text is missing on 3 image(s).
[NbConvertApp] Writing 310972 bytes to vertex_color_calculator.html
```

In [ ]: