1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA

dei25 departamento de engenharia informática 1995 – 2020

Computação Gráfica

André Perrotta (avperrotta@dei.uc.pt)

Hugo Amaro (hamaro@dei.uc.pt)

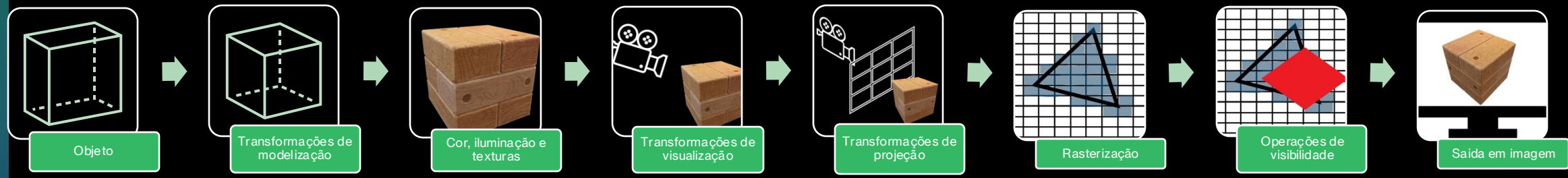
T_08:

Iluminação, efeitos

Objetivos da aula

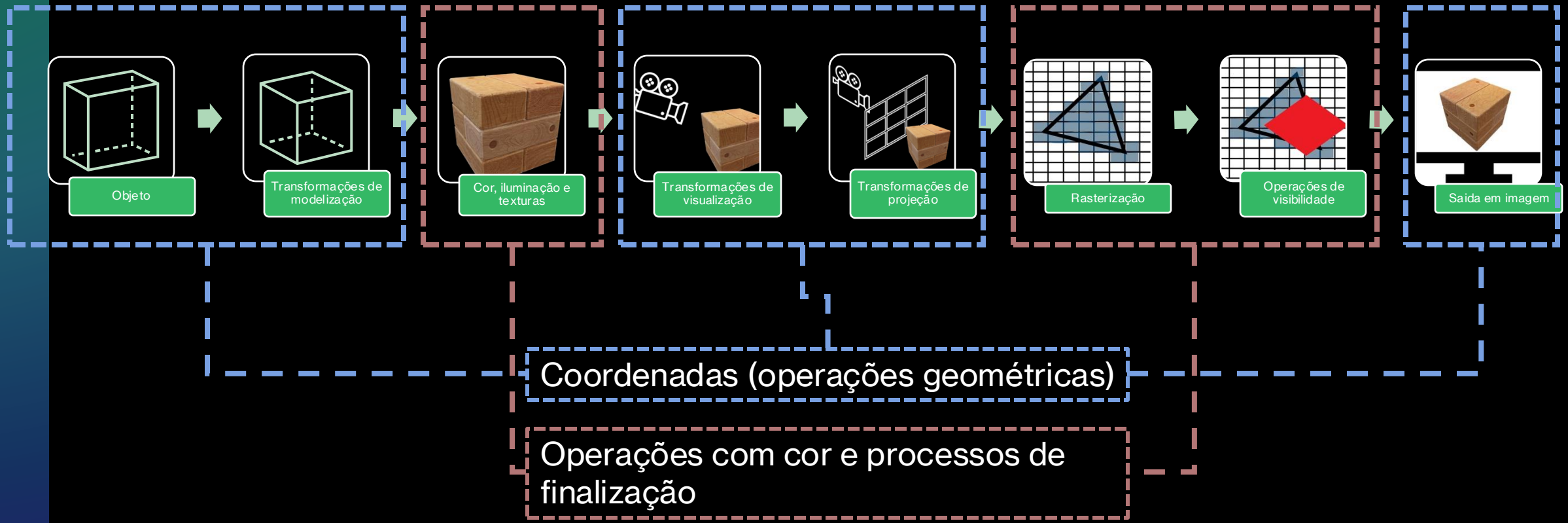
- Introduzir o conceito de luz e cor em CG/OpenGL
- Entender os fundamentos do modelo de Phong
- Entender os fundamentos matemáticos relacionados
- Implementar as luzes básicas em OpenGl/OF

Render pipeline



Pipeline de renderização POLIGONAL

Render pipeline

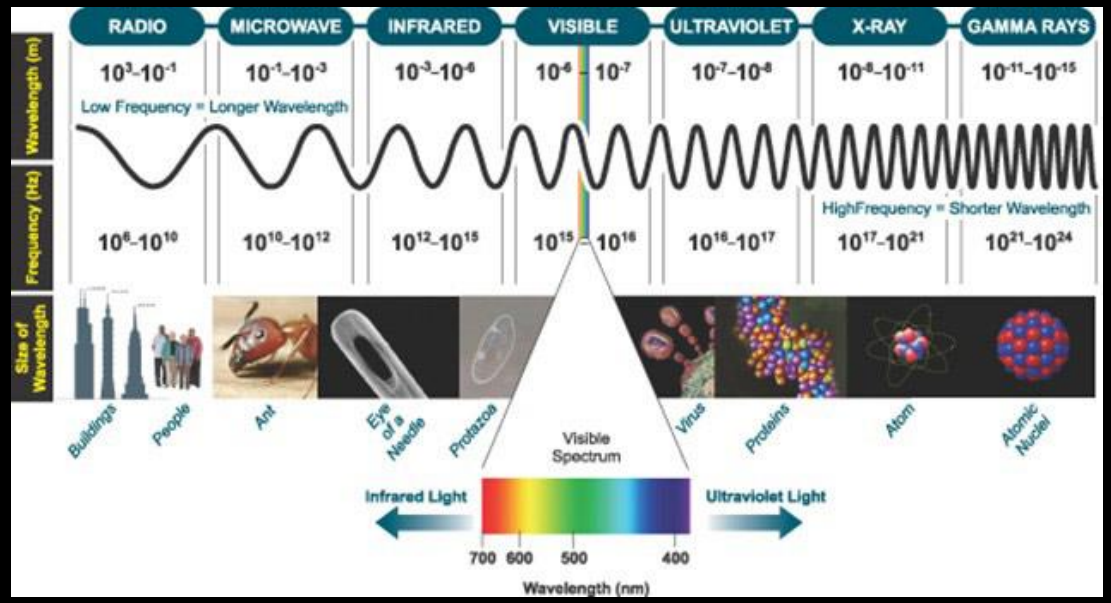


Cor e Iluminação

Cor, iluminação e texturas

Ponto de partida: Cor e Luz?

- O que é a luz?
  - A luz é radiação eletromagnética capaz de ser detectada pelo nosso aparato visual.
- O que é a cor?
  - Cor é uma características dos materiais relativa aos comprimentos de onda que são refletidos (ou emitidos) na forma de radiação eletromagnética, capazes de serem analisados e interpretados pelo nosso aparato visual.
  - A explicação física por trás desses conceitos é bastante complexa, envolvendo diversos tópicos do domínio da física (mecânica quântica, eletromagnetismo, etc.)



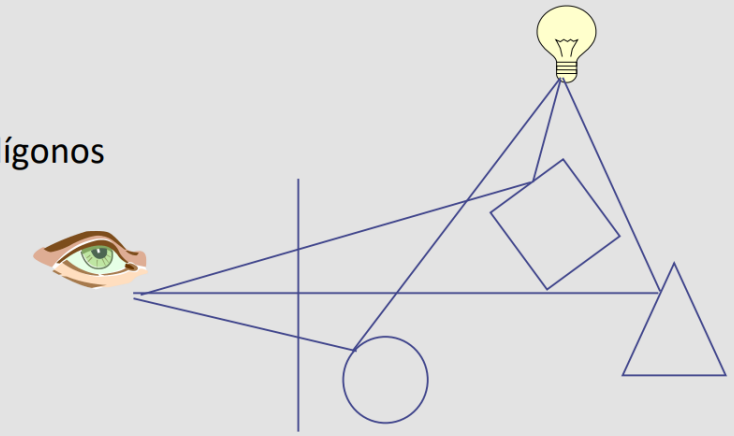
Ponto de partida: Como simular/modelar ?

- Dada a complexidade do fenômeno luz/cor, seria muitíssimo complexo e certamente custoso (tempo de processamento, flexibilidade e usabilidade intuitiva) tentar criar um modelo de iluminação para uso genérico em CG com base nos conceitos e formulação matemática dos fenômenos físicos envolvidos.
- Precisamos de algo simples, intuitivo e, acima de tudo, que possa ser processado em tempo real.

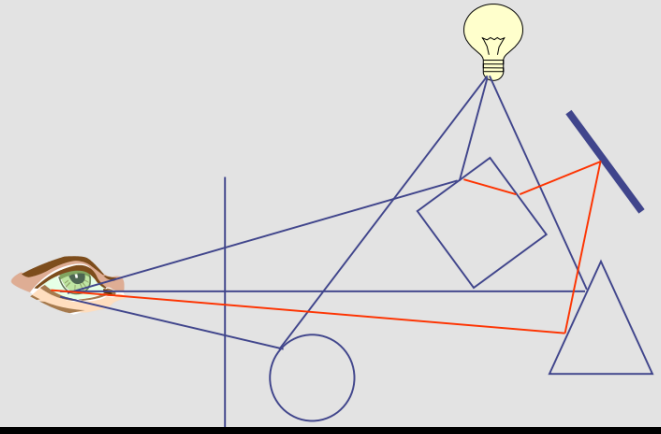
Possíveis modelos (simplificados)

- **Algoritmos de iluminação <u>Locais</u>**
  - Não tem em conta interacções
  - Apenas interpolação (*sombreamento*) de polígonos
  - **Phong**, Ray Casting Simples

- Limitações dos algoritmos locais
  - Sombras
  - Reflexões múltiplas



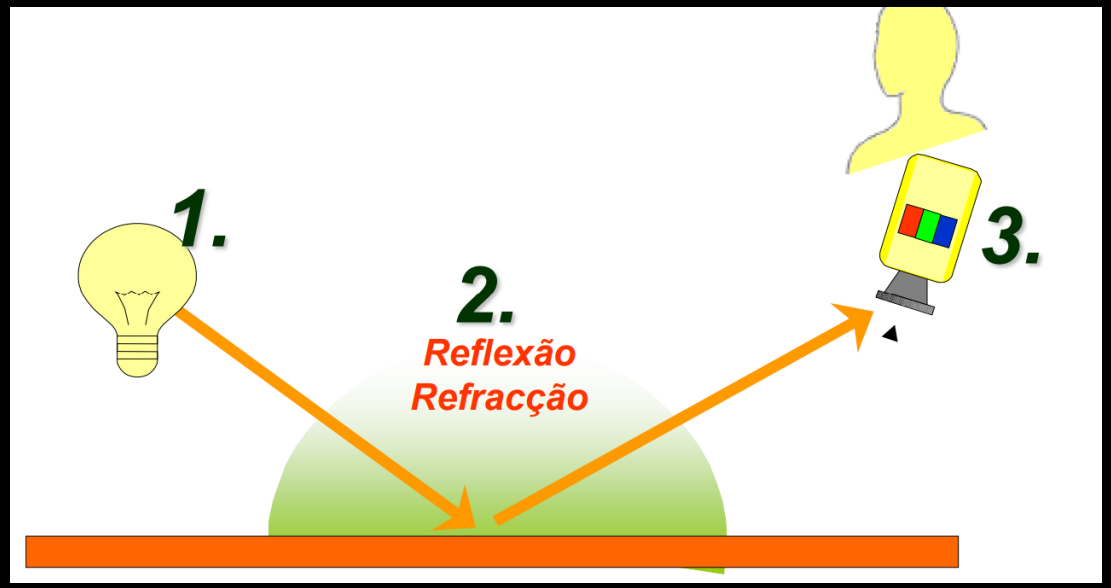
- **Algoritmos de iluminação <u>Globais</u>**
  - Consideram interacções
    - Radiosidade
    - **Ray tracing**
    - Métodos Monte Carlo



Ponto de partida: Descrição do Problema

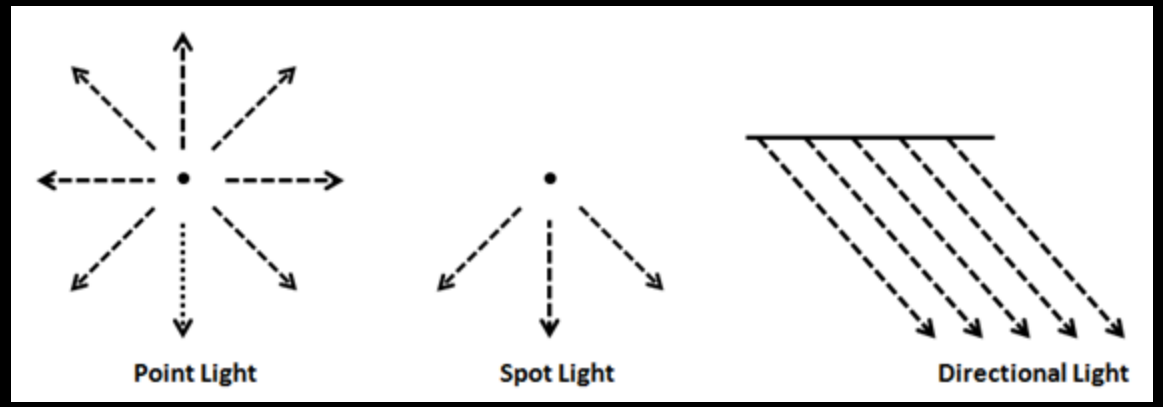
- Queremos um modelo que, numa cena virtual contendo fontes de luz e objetos (vértices!), seja capaz de:
  - Estimar a intensidade de luz resultante em cada vértice
  - Estimar a cor resultante em cada vértice

  - Ou seja:
    - Queremos modelar a INTERAÇÃO entre a luz e um objeto/material
      - Reflexão
      - Refração
      - Absorção
      - Transmissão

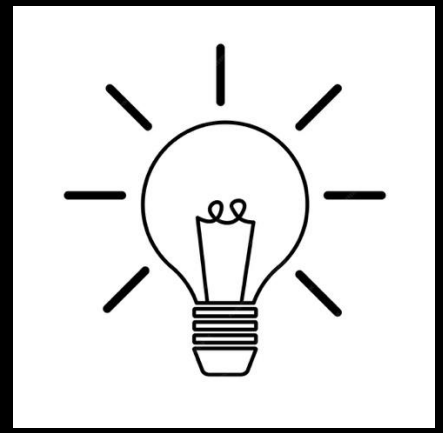


Fontes de luz

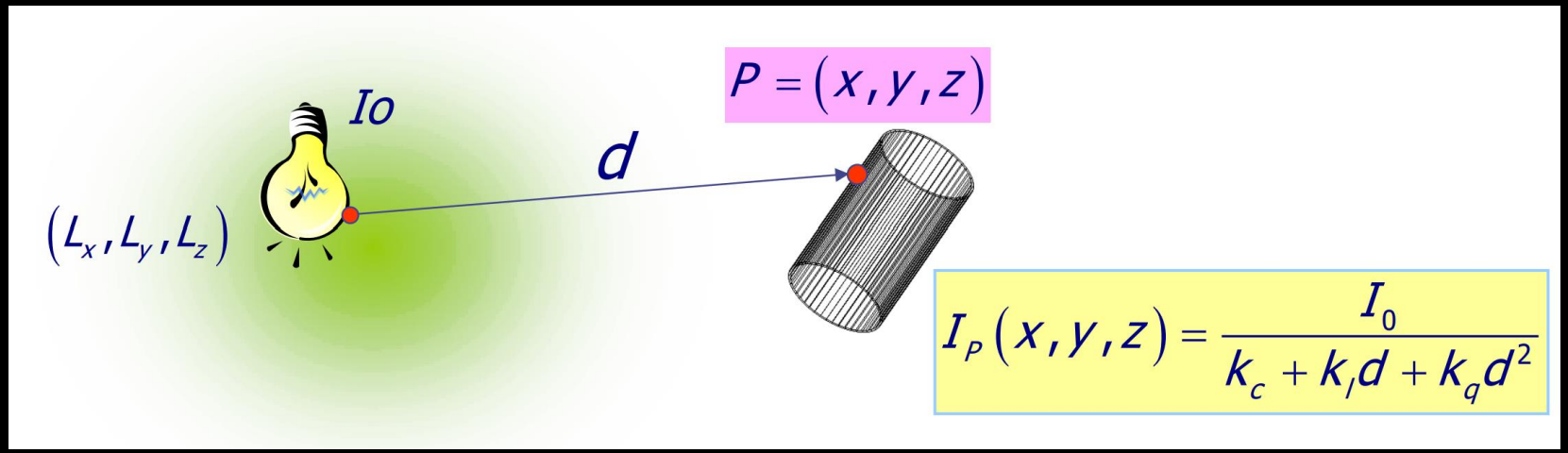
- Para poder descrever tal modelo, precisamos primeiramente definir o que são (ou podem ser) “fontes de luz” no nosso contexto.
- Assim, de forma simplificada podemos descrever 3 tipos de fontes de luz que generalizam as fontes usuais que nos rodeiam:
  - Luz pontual:
    - Ex: Lâmpadas e similares
  - Luz direcional:
    - Ex: Sol, Lua
  - Luz foco:
    - Ex: lanterna

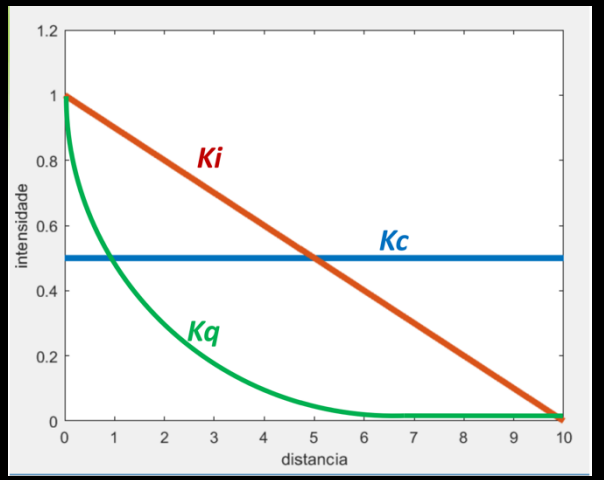


Luz pontual



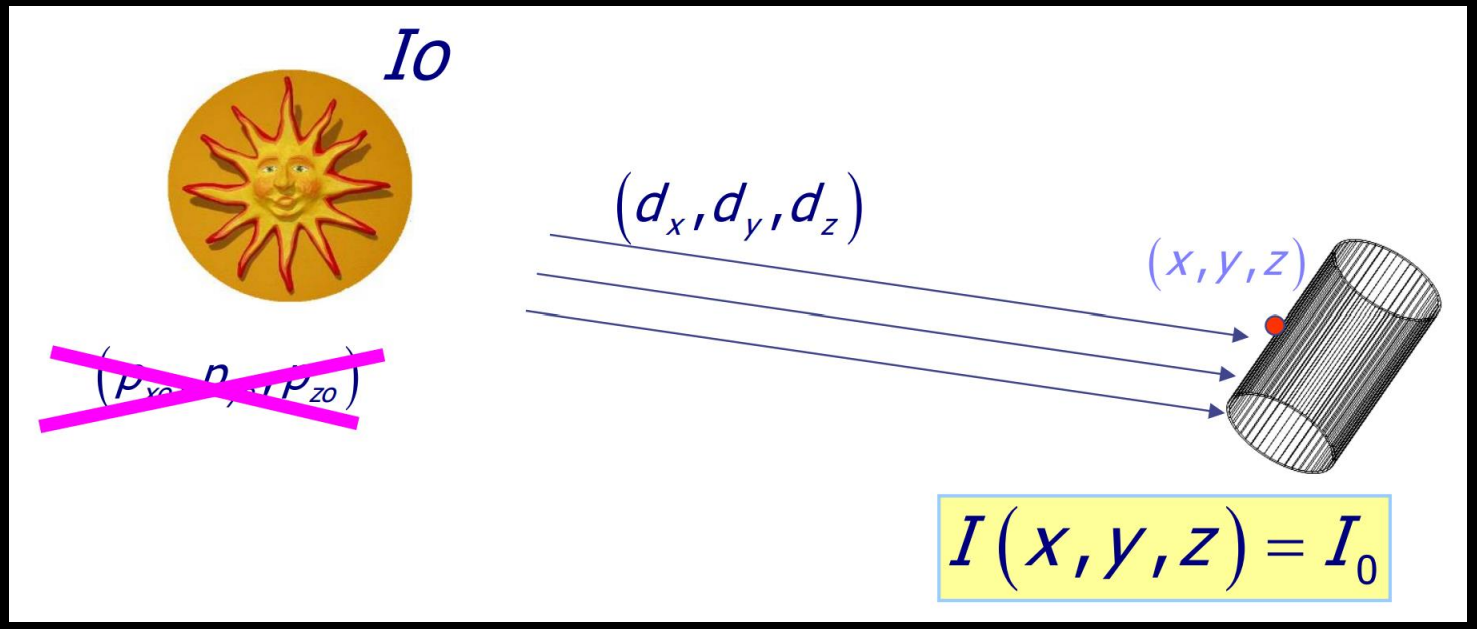
- Irradia em todas as direções
- Possui uma posição
- Está a uma distância finita de todos os vértices da cena
- Possui uma intensidade I_0 e fatores de atenuação (k_c, k_l, k_q)





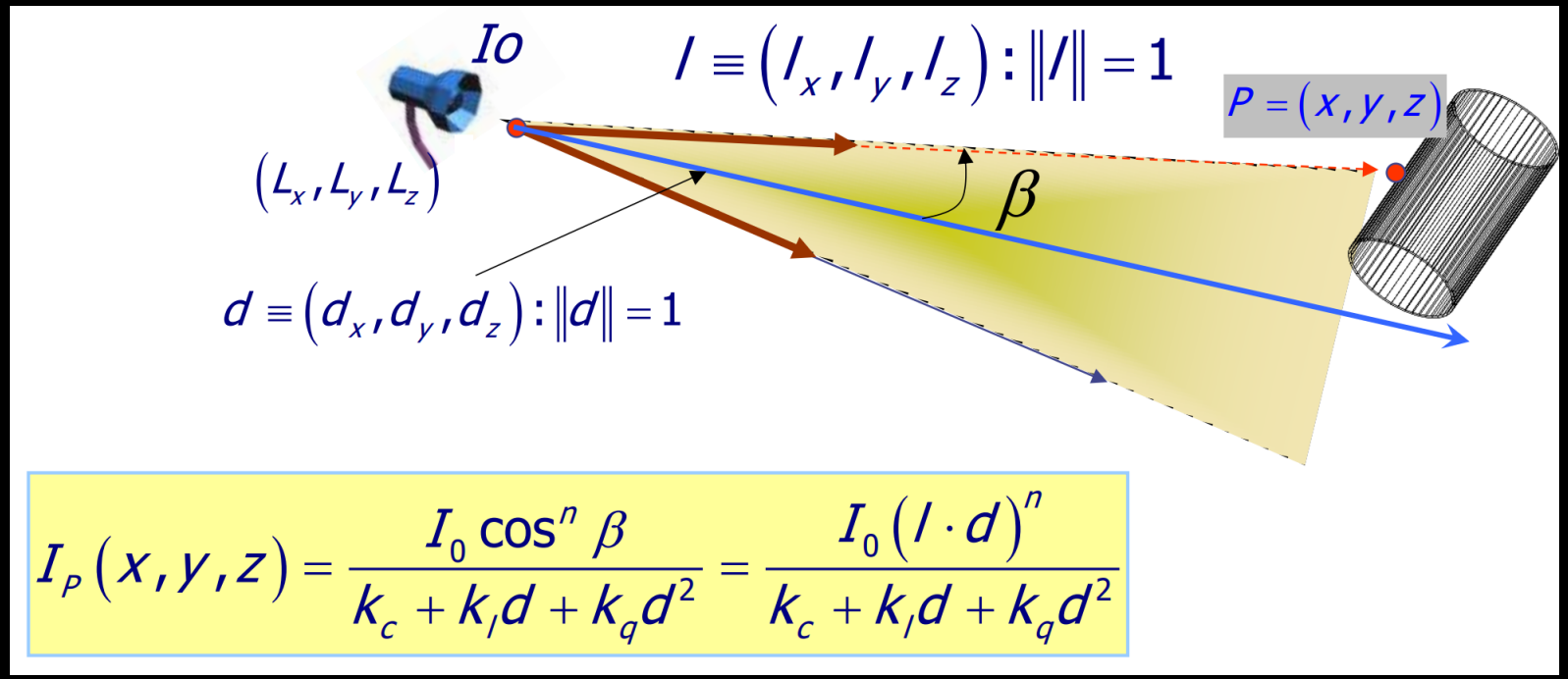
Luz direcional

- Irradia numa única direção
- Atinge todos os vértices da cena com a mesma direção
- Está a uma distância “infinita” de todos os vértices
- Possui uma intensidade I_0



Luz foco

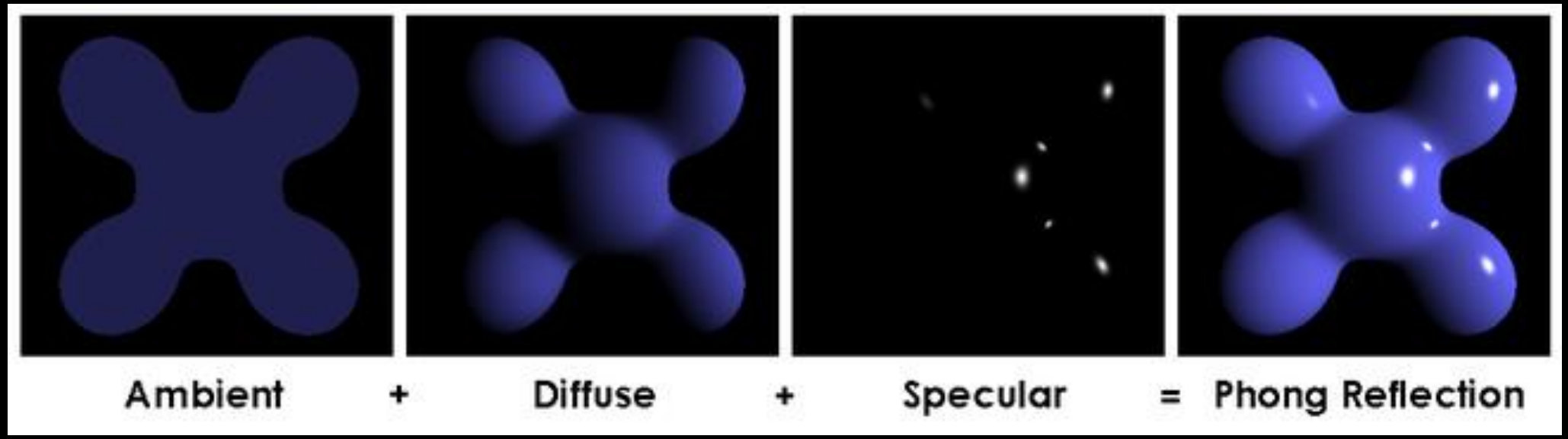
- Irradia num feixe de direções determinado por um ângulo de abertura
- Tem posição e direção (para onde aponta)
- Distância finita aos vértices onde incide
- Atenuação



$$I_P(x,y,z) = \frac{I_0 \cos^n \beta}{k_c + k_l d + k_q d^2} = \frac{I_0 (I \cdot d)^n}{k_c + k_l d + k_q d^2}$$

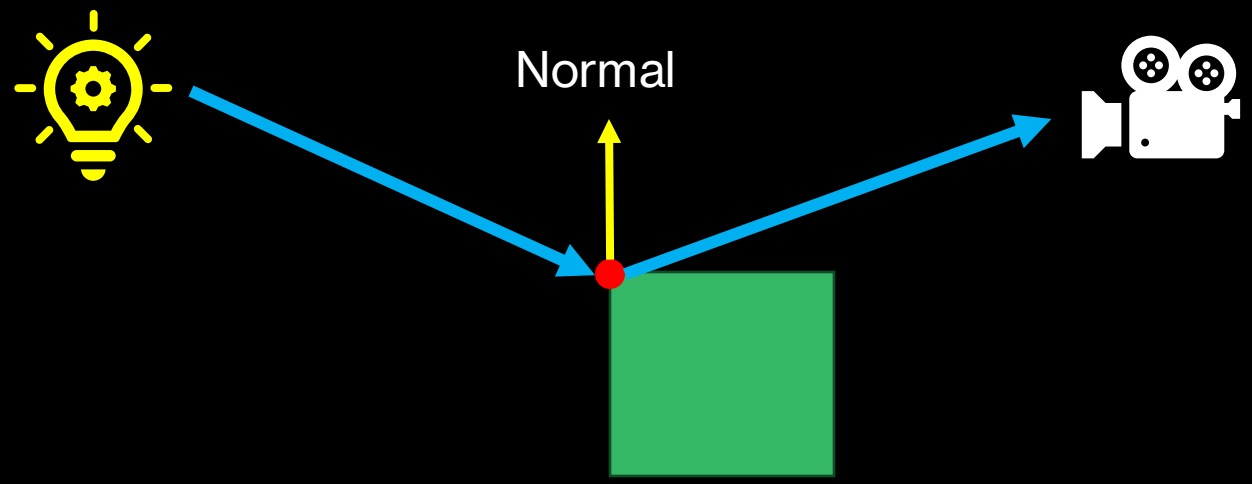
Interação com objetos (vértices)

- Precisamos agora definir um modelo de interação das fontes de luz com os objetos (vértices) da cena.
- Para esta finalidade usamos o Modelo de Phong, que caracteriza a interação de uma fonte de luz com um objeto utilizando 3 componentes:
  - Ambiente, difusa, especular



Interação com objetos

- A interação entre uma fonte de luz e um objeto, será então calculada com base em diversos parâmetros relativos a aspectos das fontes de luz em cena, da definição de material/cor de reflexão dos vértices e da relação geométrica entre a fonte, o vértice e o observador/câmera da cena



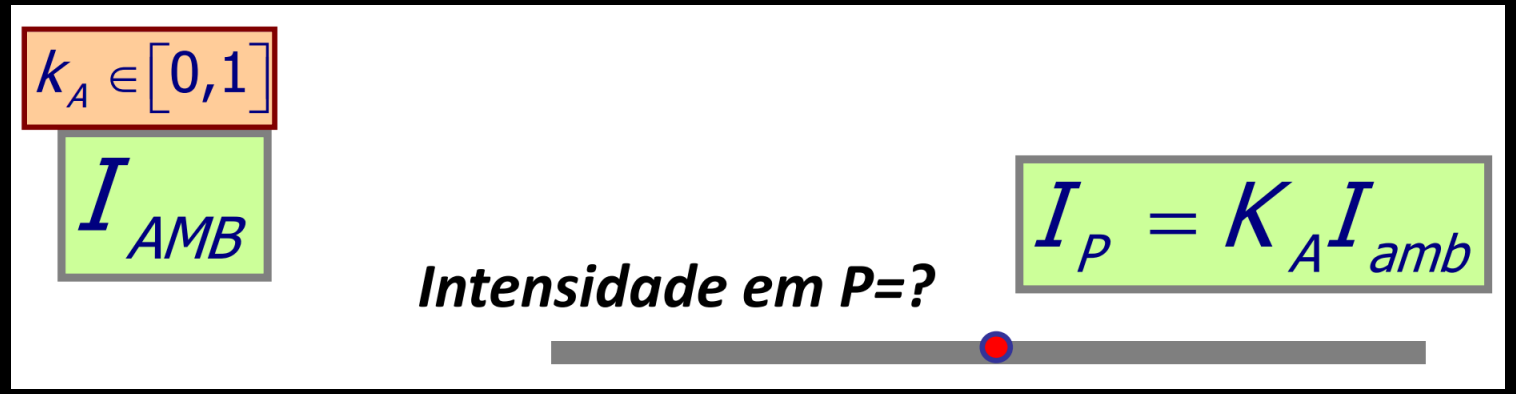
Modelo de Phong

- O modelo de Phong dita que a interação entre uma fonte de luz e um objeto é realizada através de 3 componentes:
  - Ambiente
  - Difusa
  - Especular

- As fontes de luz têm a capacidade de “iluminar” cada uma dessas componentes, e os objetos, ou os materiais definidos para os objetos, têm a capacidade de refletir essas 3 componentes.
- Cada uma das componentes estabelece uma relação específica entre fonte, vértice e observador

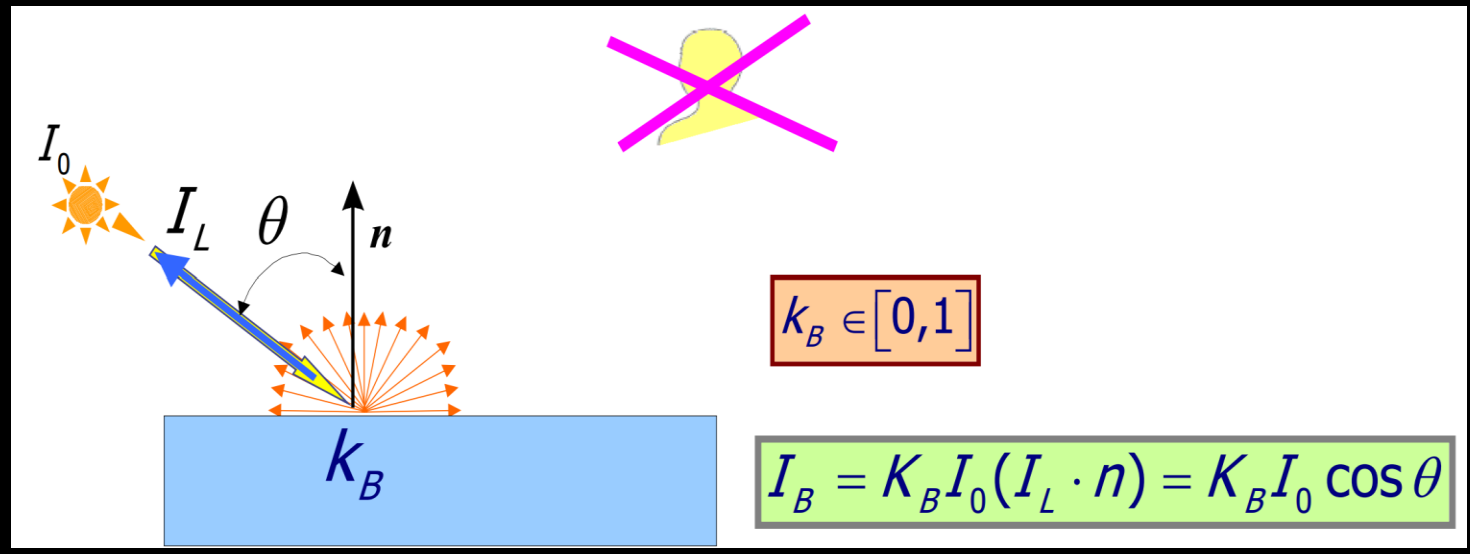
Componente Ambiente

- Representa a luz “indireta”
- Não depende de posição ou ângulo de incidência nos vértices
- Não depende da posição do observador
- É constante
- Caracterizada pelo coeficiente de reflexão ambiente k_A



Componente difusa

- Reflexão Difusa ou de corpo
- Reflexão em todas as direções
- Depende do ângulo de incidência da luz com a normal do vértice incidente
- Não depende do observador
- Coeficiente de reflexão difusa k_B

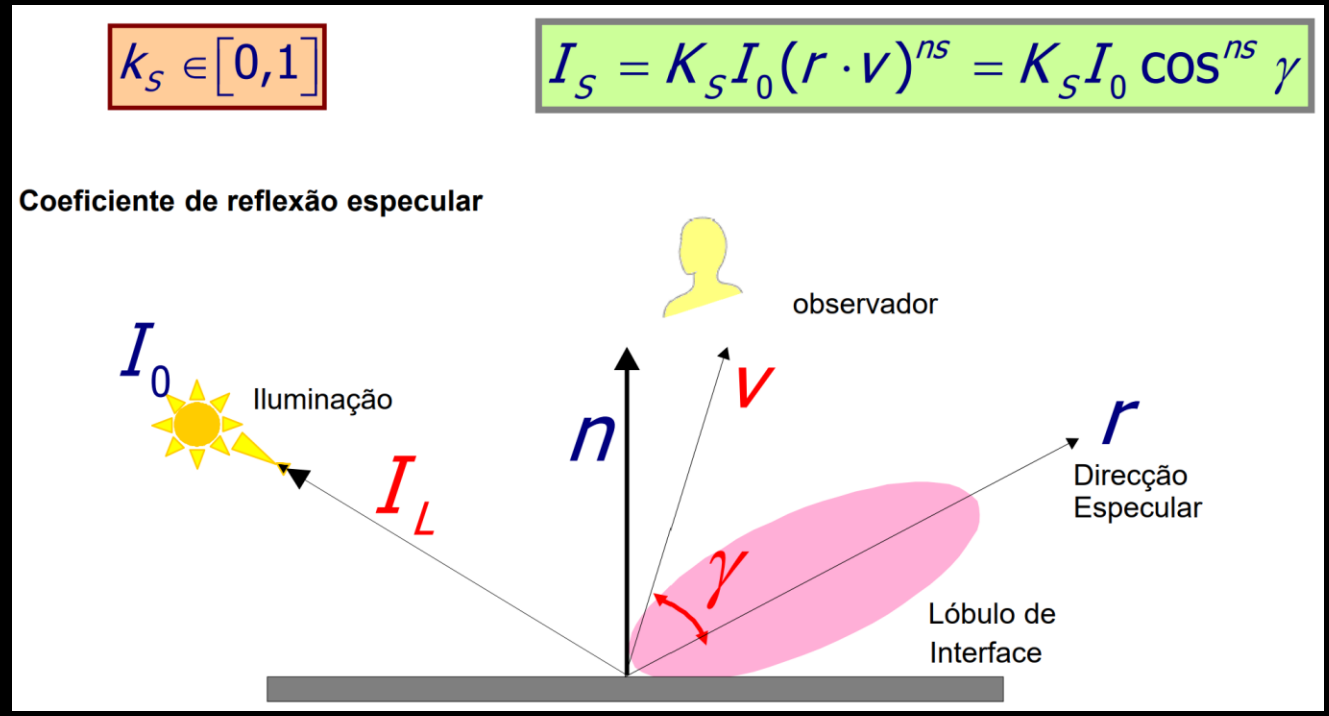


Componente especular

- Reflexão de superfície
- Depende da posição do observador e da normal do vértice incidente.
- É máxima quando há alinhamento da direção especular com a vista do observador.
- Possui um coeficiente de especularidade ns

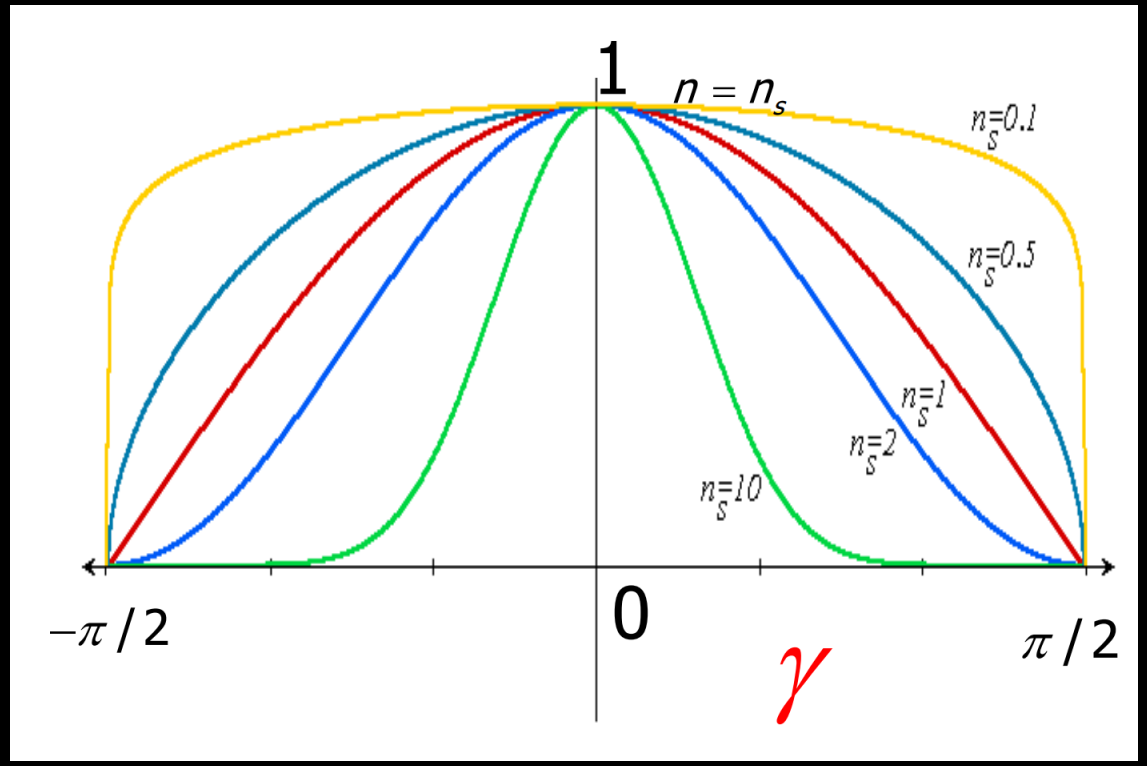
$k_s \in [0,1]$

$I_s = K_s I_0 (r \cdot v)^{ns} = K_s I_0 \cos^{ns} \gamma$



Coeficiente de especularidade

- Quão refletivo é o material



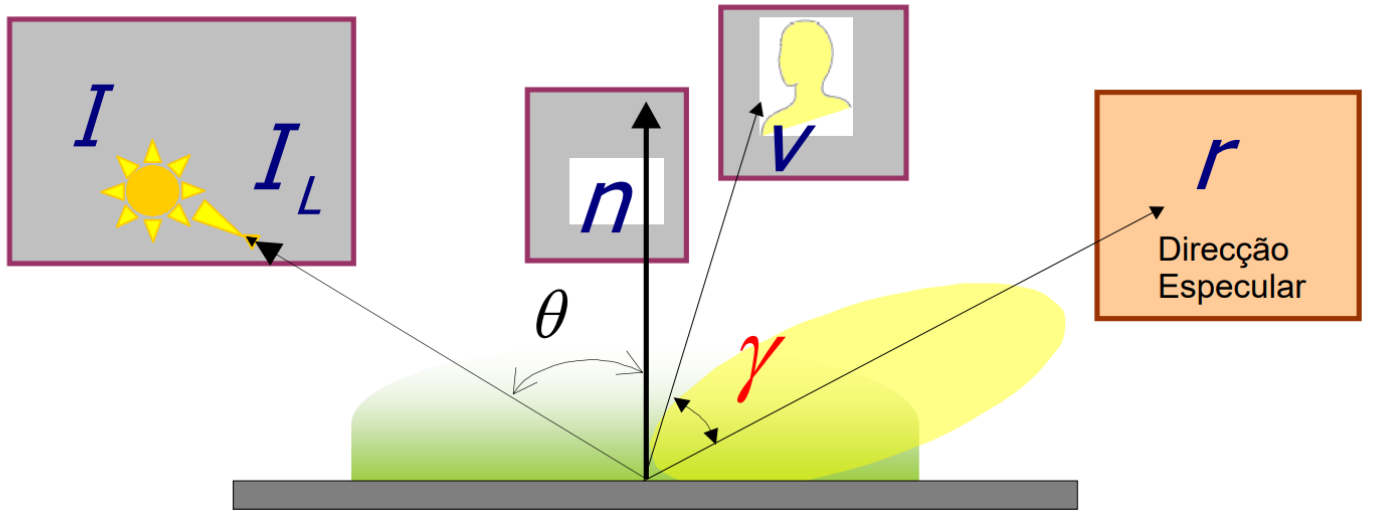
Resumindo

- **Componentes**
  - 1. Iluminação ambiente
  - 2. Emissão
  - **3. Reflexão Difusa**
  - **4. Reflexão especular**

$$I_A = K_A I_{amb}$$

$$I_B = K_B I (I_L \cdot n) = K_B I \cos\theta$$

$$I_S = K_S I \, (r \cdot v)^{ns} = K_S I \cos^{ns} \gamma$$

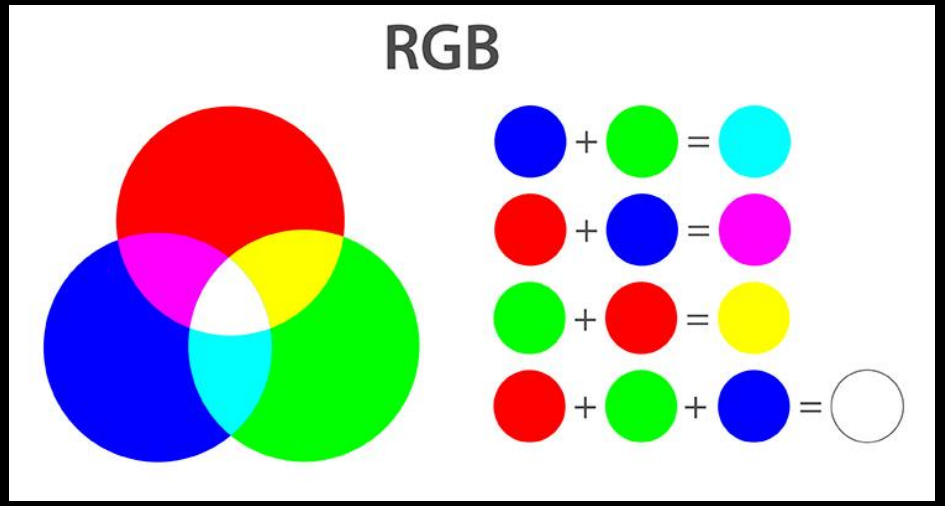


I?, K ?

- O que são os I?
- O que são os K_A, K_B, K_S?

Modelo de cor do OpenGl

- Em OpenGL, utiliza-se o modelo de cor RGB, que é um modelo matemático que discretiza as cores percepcionadas pelos humanos em 3 componentes: Red, Green, Blue.



I: Intensidade das componentes da fonte

- O que são os I?
  - Intensidade da fonte de luz em (R, G, B) para cada uma das componentes
    - (R, G, B)resultante = (R, G, B)ambiente + (R, G, B)difusa + (R, G, B)especular
  - Posso pensar em I como sendo um “vetor cor”, numa base R, G, B. O que permite comparar a intensidade de duas cores através da comparação dos módulos.

K: coeficientes de reflexão

- Os K_A, K_B, K_S são os coeficientes de reflexão para as componentes ambiente, difusa e especular, característicos de um material.
- Determinam a cor e intensidade resultante em cada vértice.
- Estes coeficiente serão então os fatores determinantes para configurar os diferentes materiais.

Materiais

- Então, para configurar um material preciso definir:
- K_A = (R, G, B)ambiente
- K_B = (R, G, B)difusa
- K_S = (R, G, B)especular

- E também o ns = coeficiente de especularidade = shininess = quão refletora é a superfície do material
- ns = (float) [0, 128]

Material: glColorMaterial

- Até então, utilizamos glColor3f(R, G, B) para definir a cor dos objetos.
- Com a iluminação ligada, isso não basta.
- Para poder continuar a utilizar glColor, temos de “dizer” ao openGl que o glColor deve ser utilizado para configurar as componentes de um material:
  - glEnable(GL_COLOR_MATERIAL)
- E então configurar (por exemplo):

```
glColor3f(1, 1., 1.);
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT);
glColorMaterial(GL_FRONT_AND_BACK, GL_DIFFUSE);
glColorMaterial(GL_FRONT_AND_BACK, GL_SPECULAR);
glColorMaterial(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE);
```

Materiais: glMaterial

- Ou, com GL_COLOR_MATERIAL desligado, podemos criar então materiais:

```
GLfloat  bronzeAmb []={ 0.2125 ,0.1275 ,0.054, 1. };
GLfloat  bronzeDif []={   0.714 ,0.4284 ,0.18144, 1. };
GLfloat  bronzeSpec []={ 0.393548 ,0.271906 ,0.166721, 1. };
GLint  bronzeCoef = 0.2 *128;
```

```
glMaterialfv(GL_FRONT, GL_AMBIENT, bronzeAmb);
glMaterialfv(GL_FRONT, GL_DIFFUSE, bronzeDif);
glMaterialfv(GL_FRONT, GL_SPECULAR, bronzeSpec);
glMaterialf(GL_FRONT, GL_SHININESS, bronzeCoef);
```

Cor final em um vértice

- A cor final em um vértice pode ser calculada para cada componente de cor (RGB) separadamente:

$$R_{vertice} = R_{amb_{mat}} R_{amb_{fonte}} + R_{dif_{mat}} R_{dif_{fonte}} cos(\theta) + R_{spec_{mat}} R_{spec_{fonte}} cos(\gamma)^{ns}$$

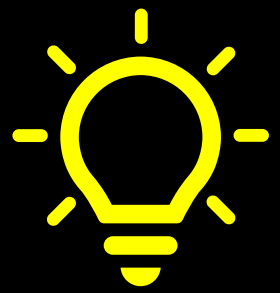
$$G_{vertice} = G_{amb_{mat}} G_{amb_{fonte}} + G_{dif_{mat}} G_{dif_{fonte}} cos(\theta) + G_{spec_{mat}} G_{spec_{fonte}} cos(\gamma)^{ns}$$

$$B_{vertice} = B_{amb_{mat}} B_{amb_{fonte}} + B_{dif_{mat}} B_{dif_{fonte}} cos(\theta) + B_{spec_{mat}} B_{spec_{fonte}} cos(\gamma)^{ns}$$

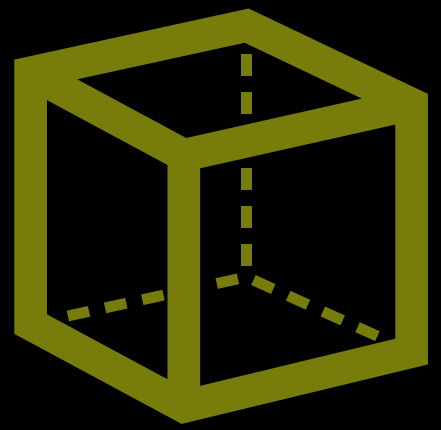
θ = ângulo entre direção da fonte e normal do vértice

γ = ângulo entre direção do observador e reflexão da direção da fonte com a normal do vértice

Como calcular: fonte pontual (sem atenuação)



I_A = (1., 1., 0)



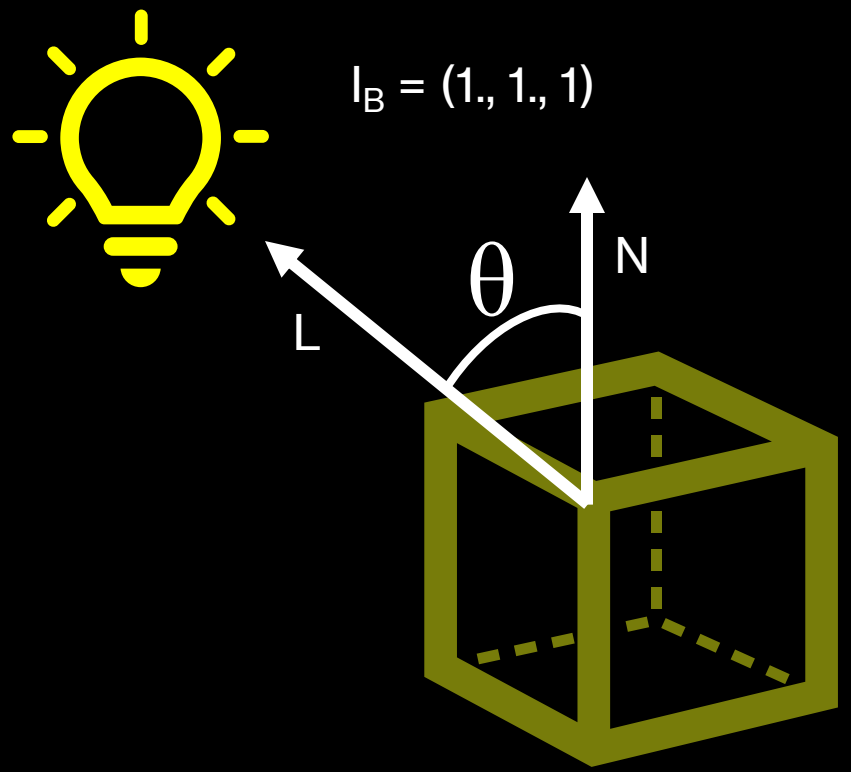
K_A = (0.5, 0.5, 0.5)

$$Cor_A = (1, 1, 0)(0.5, 0.5, 0.5) = (0.5, 0.5, 0)$$

Como calcular: fonte pontual (sem atenuação)

$$\vec{L} = (x_{luz}, y_{luz}, z_{luz}) - (x_{vertex}, y_{vertex}, z_{vertex})$$

$$\cos(\theta) = \frac{\vec{L} \bullet \vec{N}}{|\vec{L}||\vec{N}|}$$



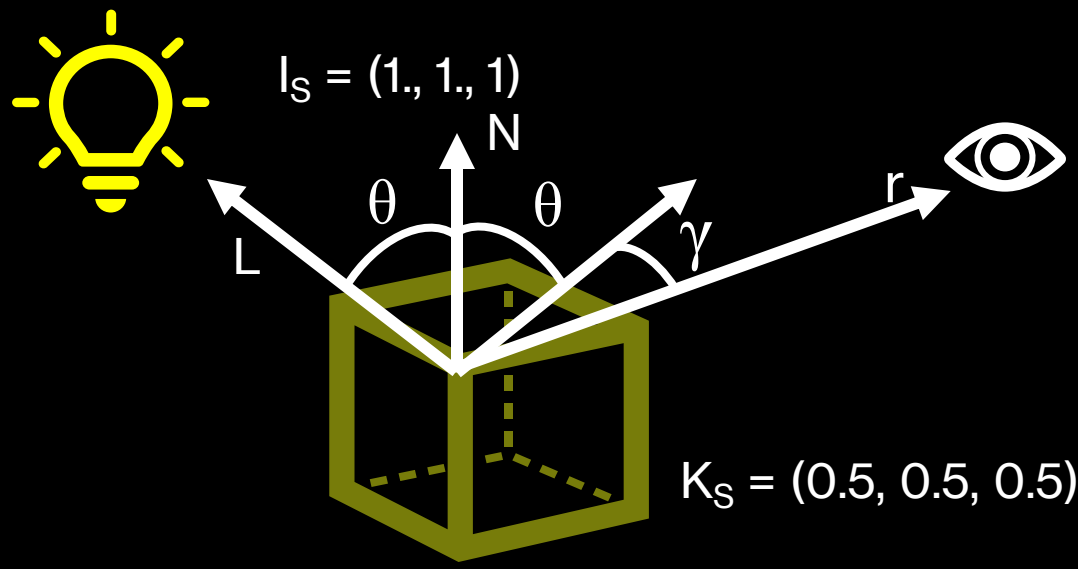
Ex: θ = 60°

Cor_B = (1, 1, 1)(0.5, 0.5, 0.5)cos(60°)
Cor_B = (0.25, 0.25, 0.25)

K_B = (0.5, 0.5, 0.5)

Como calcular: fonte pontual (sem atenuação)

Ex: $\theta = 60°$, $\gamma = 30°$, ns = 2



$$I_L = \frac{P_L - P}{\|P_L - P\|}$$

$$v = \frac{Ob - P}{\|Ob - P\|}$$

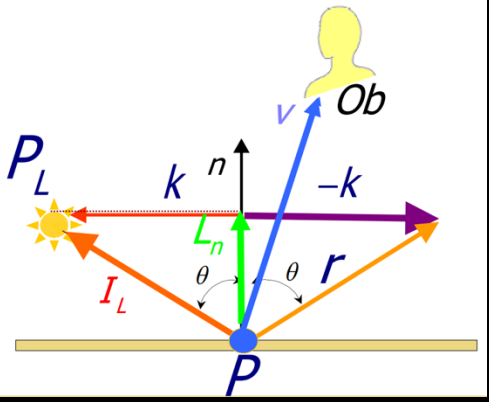
$$L_n = (I_L \cdot n)\, n$$

$$I_L = L_n + k \Leftrightarrow k = I_L - L_n$$

$$r = L_n - k$$

$$r = (I_L \cdot n)\, n - (I_L - (I_L \cdot n)\, n)$$

$$\vec{r} = 2(\vec{I}_L \cdot \vec{n})\, \vec{n} - \vec{I}_L$$



$Cor_S = (1, 1, 1)(0.5, 0.5, 0.5)\cos(\gamma)^{ns}$

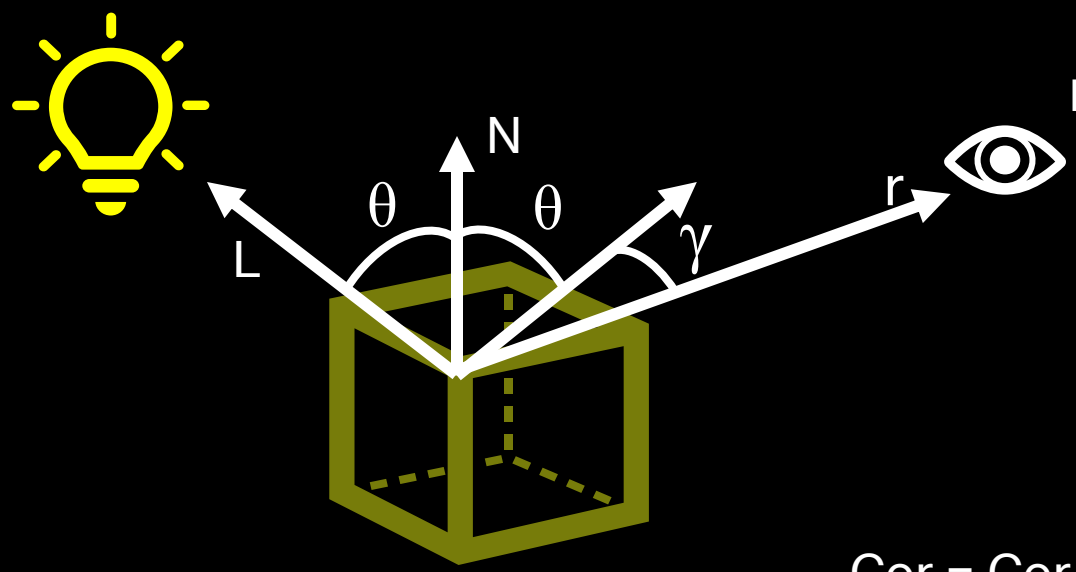
$Cor_S = (0.375, 0.375, 0.375)$

Como calcular: fonte pontual (sem atenuação)

Ex: $\theta = 60°$, $\gamma = 30°$, ns = 2

$I = \{I_A(1., 1., 0), I_B(1., 1., 1), I_S(1., 1., 1)\}$

$K = \{K_A(0.5, 0.5, 0.5), K_B(0.5, 0.5, 0.5), K_S(0.5, 0.5, 0.5)\}$



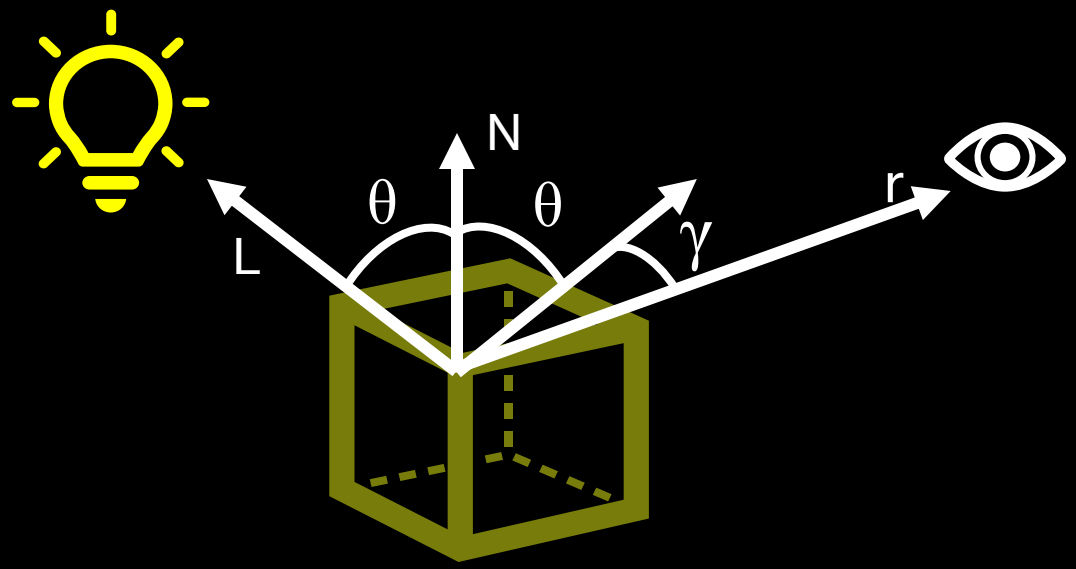
$Cor = Cor_A + Cor_B + Cor_S$

Cor = (0.5, 0.5, 0.)+(0.25, 0.25, 0.25) + (0.375, 0.375, 0.375)

Cor = (1.125, 1.125, 0.625) -> (1, 1, 0.626)

> 1 -> crop

Como calcular: fonte pontual (com atenuação?)



Ex: θ = 60°, γ = 30°, ns = 2

$I = \{I_A(1., 1., 0), I_B(1., 1., 1), I_S(1., 1., 1)\}$

$K = \{K_A(0.5, 0.5, 0.5), K_B(0.5, 0.5, 0.5), K_S(0.5, 0.5, 0.5)\}$

$Cor = (Cor_A + Cor_B + Cor_S)/atenuação$

$Cor = (1, 1, 0.626)/atenuação$

atenuação = $k_c + k_l d + k_q d^2$

k_c =coef. de atenuação constante [0-1]

k_l =coef. de atenuação linear [0-1]

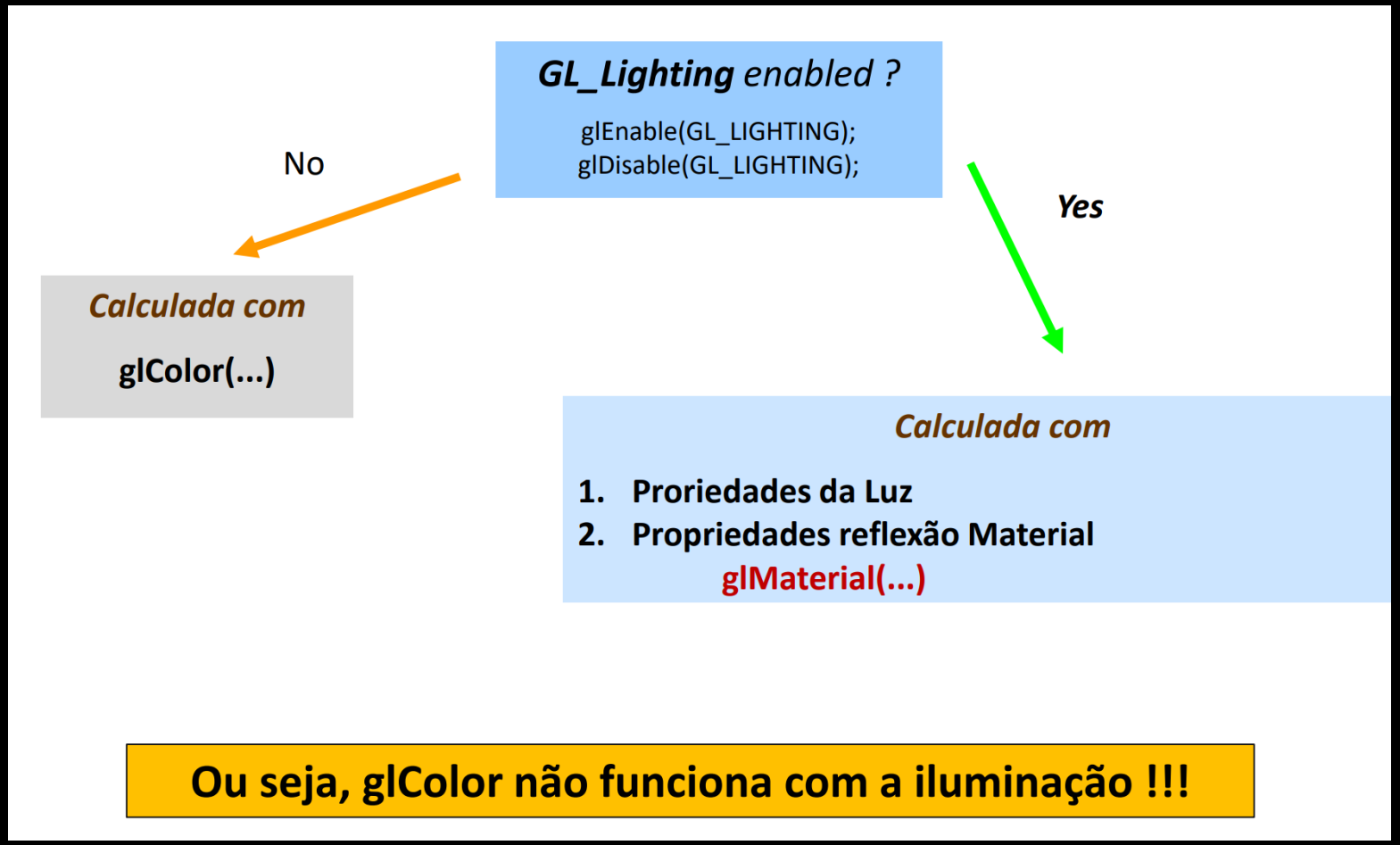
k_q =coef. de atenuação quadrática [0-1]

d = distância entre fonte e vértice

Conhecimento crucial:

- A luz é calculada nos vértices.
- As posições e direções das fontes de luz também são tratadas como vértices, portanto serão transformadas pela matriz ModelView.
  - A definição da posição/direção da fonte de luz deve ser feita após lookat e antes de desenhar os objetos
- O modelo utilizado é o de Phong: 3 componentes:
  - Ambiente, Difusa, Especular
    - Ambiente não depende de posição da fonte ou posição do observador, nem da normal do vértice
    - Difusa depende da posição da fonte e da normal do vértice, mas não da posição do observador
    - Especular depende da posição da fonte, da normal do vértice e da posição do observador

Conhecimento crucial:



Já sabemos calcular a cor resultante da iluminação nos vértices, mas como é que fazemos "entre os vértices", nas faces dos polígonos?

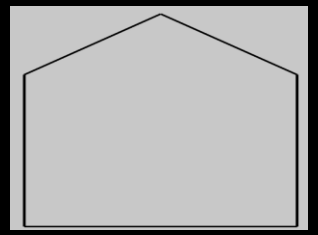
Problema

- Como é calculada a cor dos pixels na face de um polígono?
  - **“Shade model”**
- Duas hipóteses:
  1. FLAT SHADING: Todos os pixels contidos dentro de um polígono são pintados com a mesma cor de um dos vértices
  2. SMOOTH SHADING: Os pixels são pintados com cores calculadas pela interpolação das cores dos vértices, utilizando como parâmetros sua posição e/ou normal.

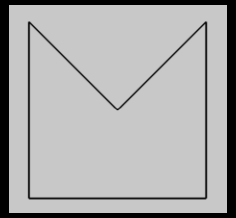
Polígono?

- Neste contexto específico (polygon shading), quando falamos de um polígono estamos sempre a falar de triângulos.
- Quando o OpenGl prepara os vértices para rasterização, todos os objetos geométricos, cujos vértices estão definidos com uma das primitivas em glBegin(PRIMITIVA) – glEnd(), são transformados em pontos, linhas ou triângulos.
- É por esse motivo que o OpenGl não desenha corretamente polígonos não convexos*
- *um polígono é convexo se uma linha que liga dois pontos quaisquer em seu interior também estiver toda em seu interior.

Convexo



Não convexo



Flat shading

- Quando definimos escolhemos utilizar Flat Shading, todos os pixels e vértices da geometria a ser desenhada vão ser coloridos com a cor do “provoking vertex”, que por defeito é o primeiro vértice definido após a chamada glBegin().
- Essa cor pode ser especificada pelo glColor, mas também como resultado do cálculo realizado na etapa de iluminação.

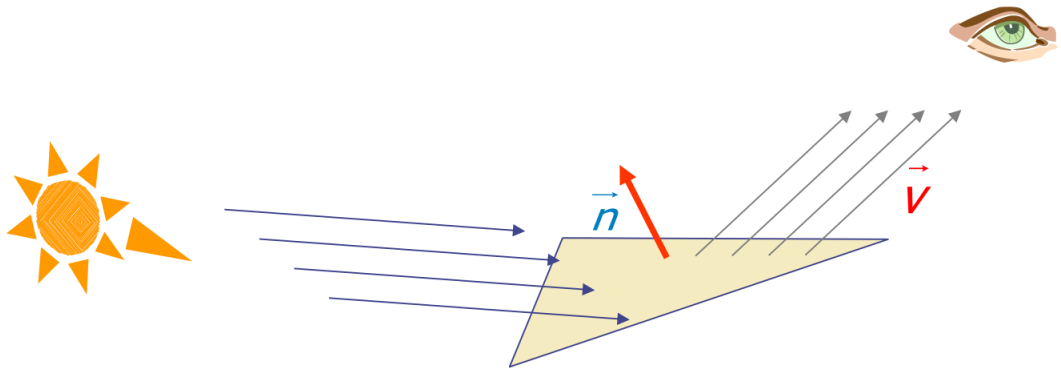
Flat shading

- **Hipóteses do modelo:**
  - A fonte de **iluminação** encontra-se no **infinito** pelo que em cada polígono

$$I_L \cdot n = k_1$$ **Constante**

  - O **observador** encontra-se no **infinito** pelo que em cada polígono

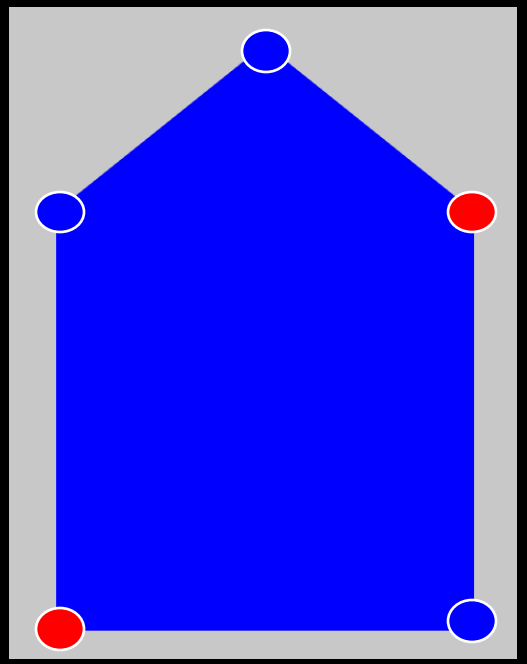
$$v \cdot n = k_2$$ **Constante**



Flat shading

- Com flat shading, o código a seguir, que define uma cor para cada vértice da geometria, irá resultar num objeto todo colorido com a cor do 1º vértice:

```
glShadeModel(GL_FLAT);
glBegin(GL_POLYGON);
glColor3f(0, 0, 1);
glVertex2f(-0.5, 0.5);
glColor3f(1, 0, 0);
glVertex2f(-0.5, -0.5);
glColor3f(0, 0, 1);
glVertex2f(0.5, -0.5);
glColor3f(1, 0, 0);
glVertex2f(0.5, 0.5);
glColor3f(0, 0, 1);
glVertex2f(0, 0.9);
glEnd();
```



Smooth shading

- Quando optamos por usar Smooth Shading para colorir os polígonos, os pixels interiores serão coloridos com valores interpolados com base na cor dos vértices.
- O algoritmo de interpolação pode variar, dependendo do contexto, aplicação e harware.
- Em Computação Gráfica os algoritmos mais utilizados são:
  - Gouraud Shading (utilizado no OpenGl)
  - Phong Shading (utilizado em ray tracing)

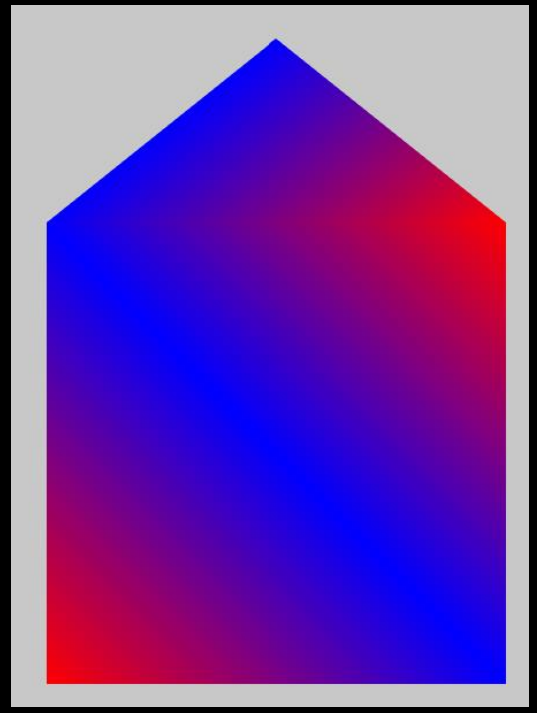
Gouraud Shading

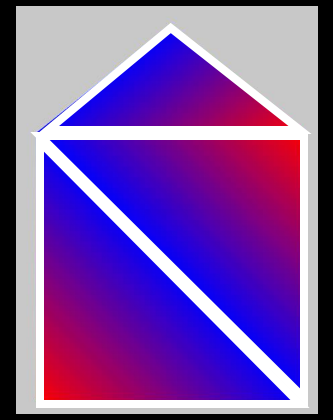
- O algoritmo de Gouraud Shading calcula a cor dos pixels interiores com base apenas na cor dos vértices do polígono.
- O processe é realizado em duas etapas:
  1. Polígono é divido em triângulos
  2. A interpolação acontece com base na cor dos vertices do triângulo

Gouraud Shading

- O código a seguir demonstra o resultado do smooth shading (Gouraud shading) num polígono convexo onde cada vértice está definido com uma cor

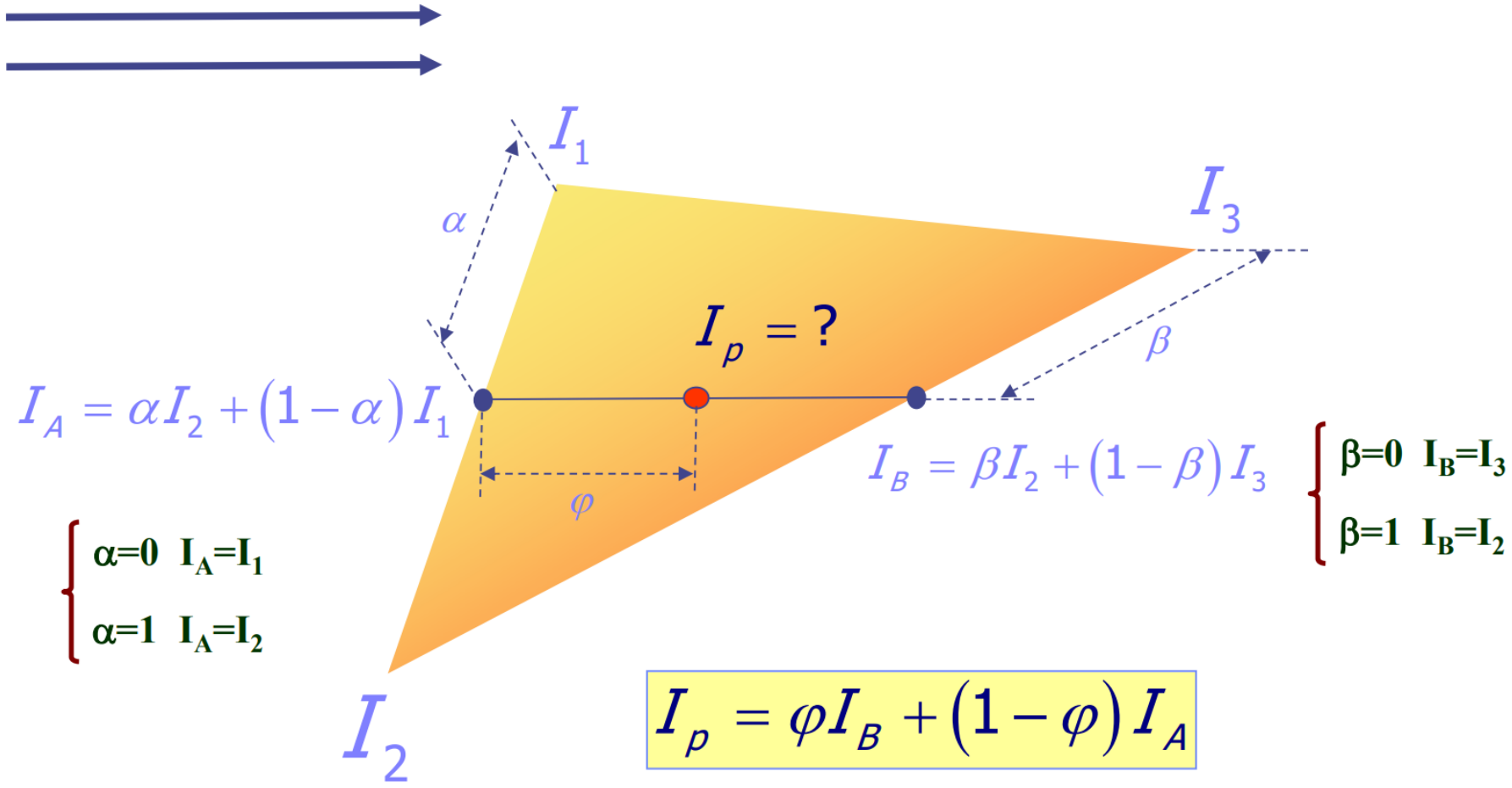
```
glShadeModel(GL_SMOOTH);
glBegin(GL_POLYGON);
glColor3f(0, 0, 1);
glVertex2f(-0.5, 0.5);
glColor3f(1, 0, 0);
glVertex2f(-0.5, -0.5);
glColor3f(0, 0, 1);
glVertex2f(0.5, -0.5);
glColor3f(1, 0, 0);
glVertex2f(0.5, 0.5);
glColor3f(0, 0, 1);
glVertex2f(0, 0.9);
glEnd();
```





Gouraud Shading

- Operação é realizada durante a rasterização, sendo os pixels são preenchidos da ***esquerda para a direita de cima para baixo***

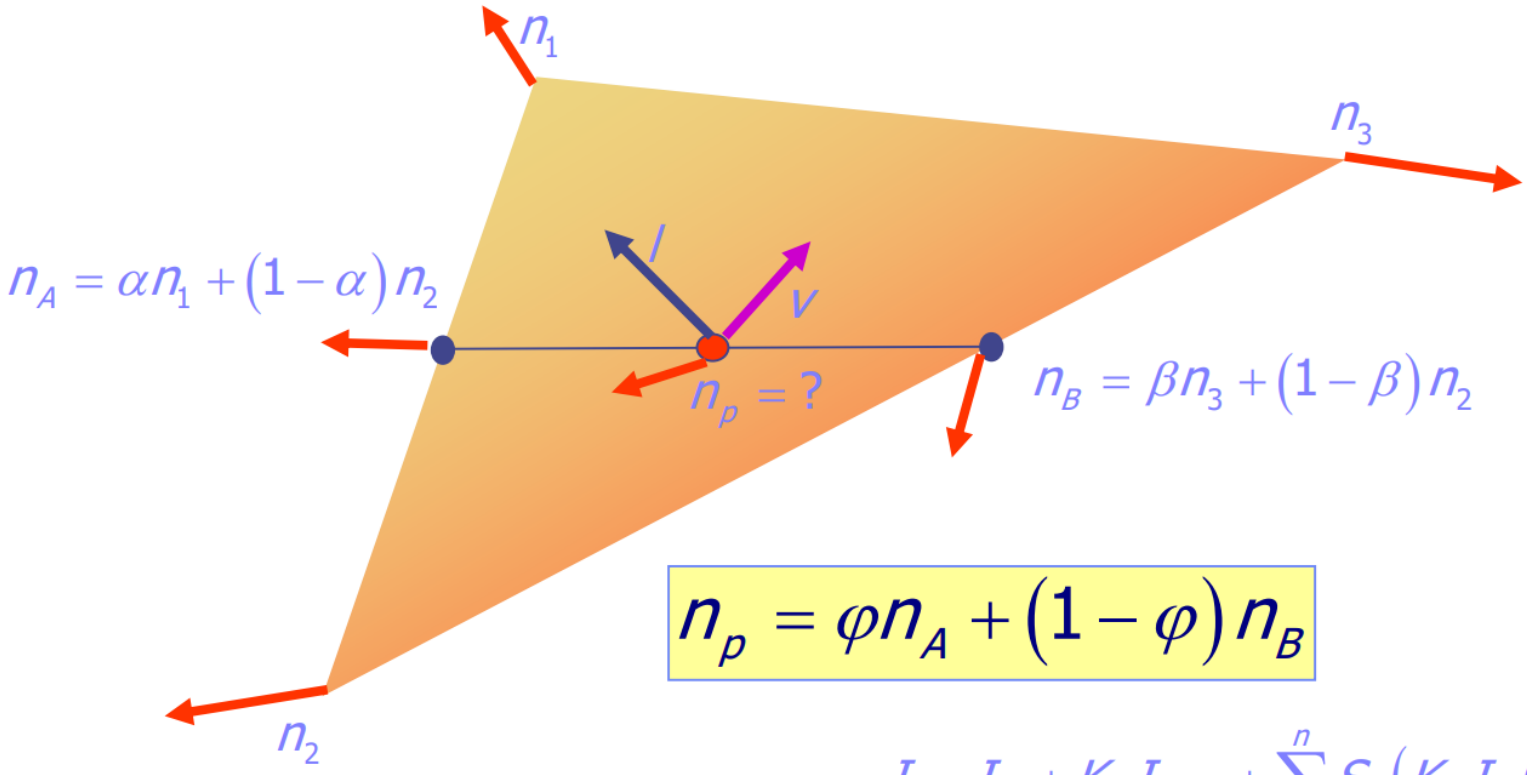


Phong shading

- O modelo de interpolação Phong Shading é mais sofisticado. A determinação da cor interior de cada triângulo é realizada com base nas normais dos vértices e no re-processamento da iluminação para cada ponto.
- Esta estratégia resulta em efeitos de luz mais realistas, mas o cálculo exaustivo pixel-a-pixel da iluminação o torna impraticável para aplicações em tempo-real.

Phong Shading

- Ao contrário de Gouraud no método de Phong o que é interpolado é a normal e não a cor.
- A cor é ***determinada*** a partir da normal interpolada.

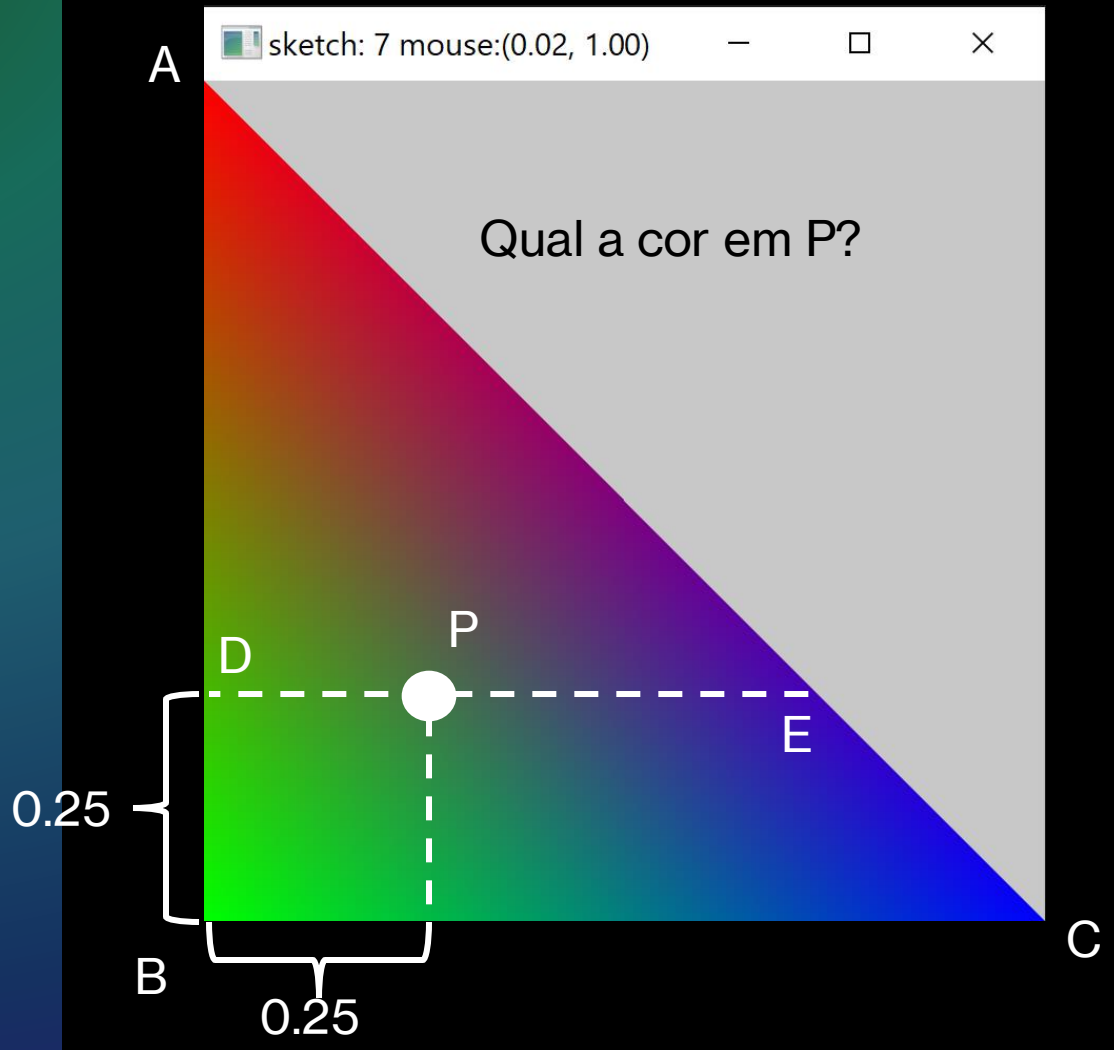


$$n_p = \varphi n_A + (1-\varphi) n_B$$

$$I = I_E + K_A I_{Amb} + \sum_{i=1}^{n} S_i \left(K_B I_{L_i} (l_i \cdot n_p) + K_S I_{Li} (r_i \cdot v)^n \right)$$

Shading models

Exemplo: Gouraud Shading



$\overline{AB} = \overline{BC} = 1, \overline{AC} = \sqrt{2}$

$\overline{DE} = \frac{3}{4}$

$\overline{EC} = \frac{\sqrt{2}}{4}$

normalizamos os segmentos

$\overline{DP} = \frac{\frac{1}{4}}{\frac{3}{4}} = \frac{1}{3}$

$\overline{(EC)} = \frac{\frac{\sqrt{2}}{4}}{\sqrt{2}} = \frac{1}{4}$

$I_D = \frac{1}{4}I_A + \frac{3}{4}I_B = \frac{1}{4}(1,0,0) + \frac{3}{4}(0,1,0) = (\frac{1}{4}, \frac{3}{4}, 0)$

$I_E = \frac{1}{4}I_A + \frac{3}{4}I_C = (\frac{1}{4}, 0, 0) + (0, 0, \frac{3}{4}) = (\frac{1}{4}, 0, \frac{3}{4})$

$I_P = \frac{2}{3}I_D + \frac{1}{3}I_E = (\frac{1}{6}, \frac{1}{2}, 0) + (\frac{1}{12}, 0, \frac{1}{4}) = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4}) = (0.25, 0.5, 0.25)$

Créditos

- Muitos slides e imagens foram “emprestados” do ótimo material do prof. Jorge Henriques ☺