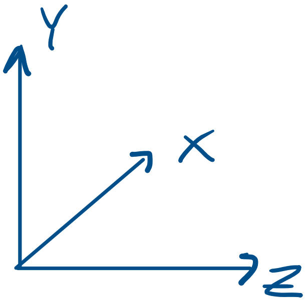
# Teste 20/21

17 de maio de 2024 20:08

1

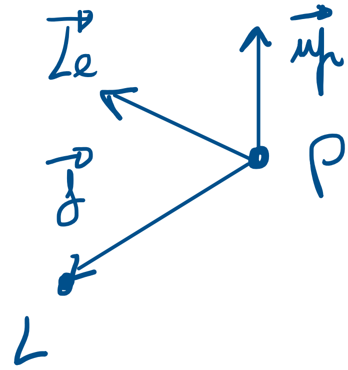
a)




glutLookAt(0, sen(α), cos(α),
0, 0, 0
0, 1, 0)

b) glTranslate(0, sen(α), cos(α))
glRotate(−α, 1, 0, 0)

2




$\vec{F} = L - P$

$\vec{Le} = \vec{f} \times \vec{up}$

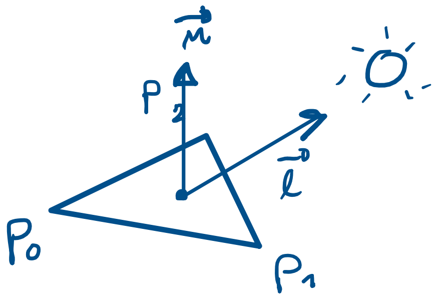
$\vec{upReal} = \vec{Le} \times \vec{f}$

$newP = P + dh \cdot \vec{Le} + dv \cdot \vec{upReal}$

3 $\vec{n} = \vec{v_1} \times \vec{v_2}$ { Regra da mão direita

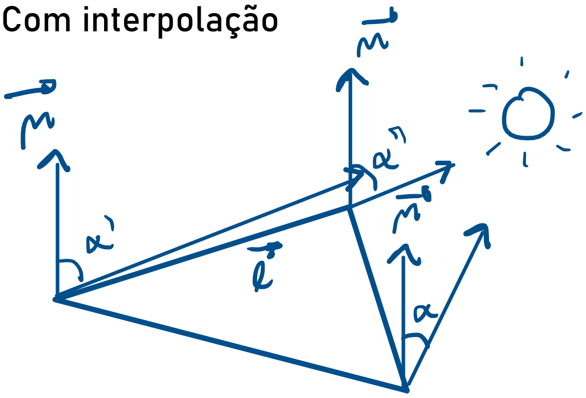
Flat Shading

- Cada triângulo tem uma normal

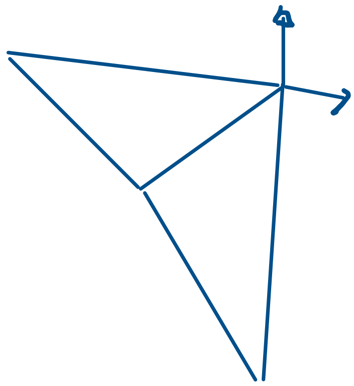



Todo o triângulo fica com a mesma cor →
Descontinuidade de intensidade

Com interpolação



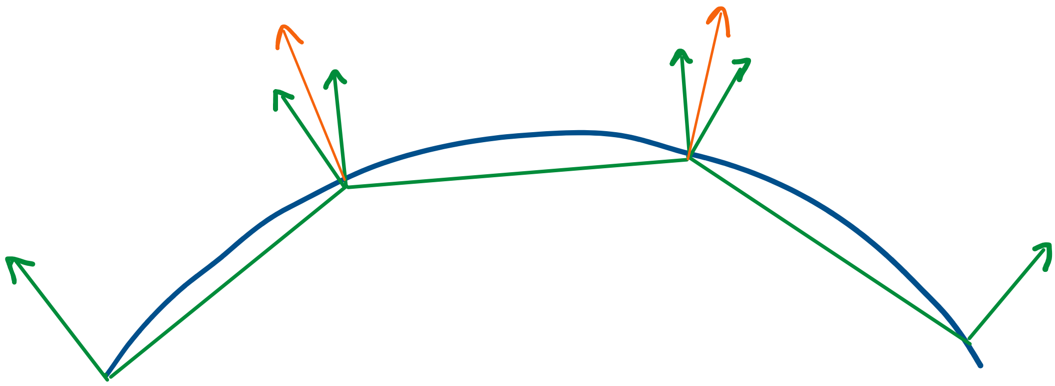

Cálculo da cor para cada vértice
Pontos no interior do triâmgulo
-> cor é calculada por interpolação



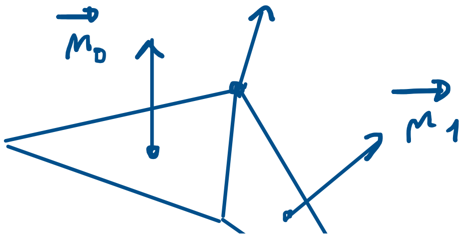
... mas continua a existir descontinuidade ao passar de um triângulo para o outro

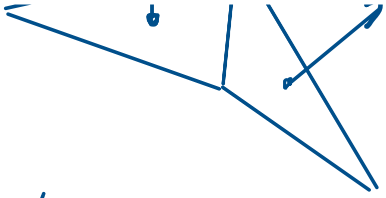
Solução de

## Gouraud



- Uma normal por vértice!

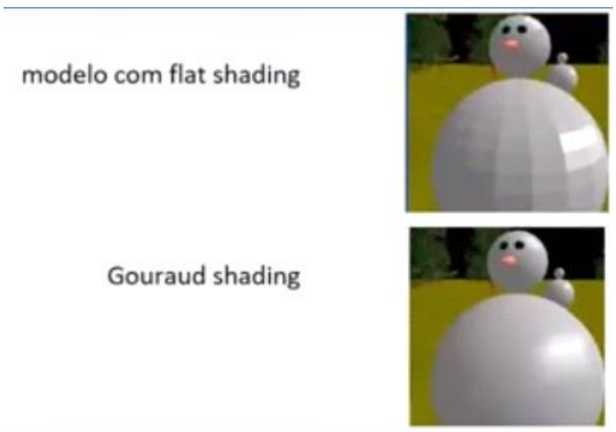




PROBLEMA: Funciona bem com iluminação difusa, mas não funciona muito bem para iluminação especular

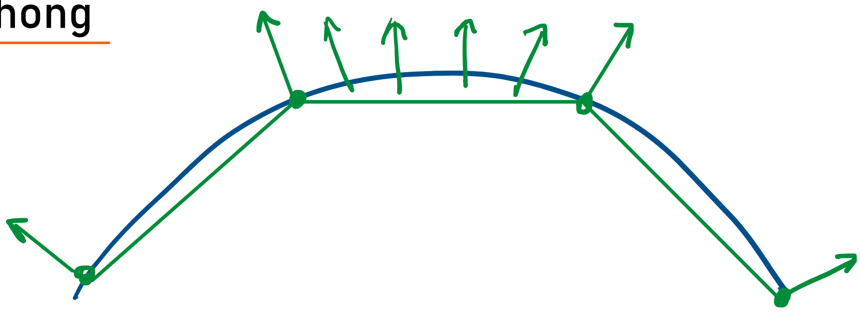
- Cada vértice pode ter uma normal diferente
- Vertices partilhados por vários triângulos têm todos a mesma normal
- A cor dos pontos internos do triângulo é calculada por interpolação da cor dos vértices

Todos estes vértices têm valor especular 0, apesar de existir uma mancha especular dentro do triângulo




Solução de

## Phong



Solução:

- Interpolar normais em vez de cores!

Resumindo:

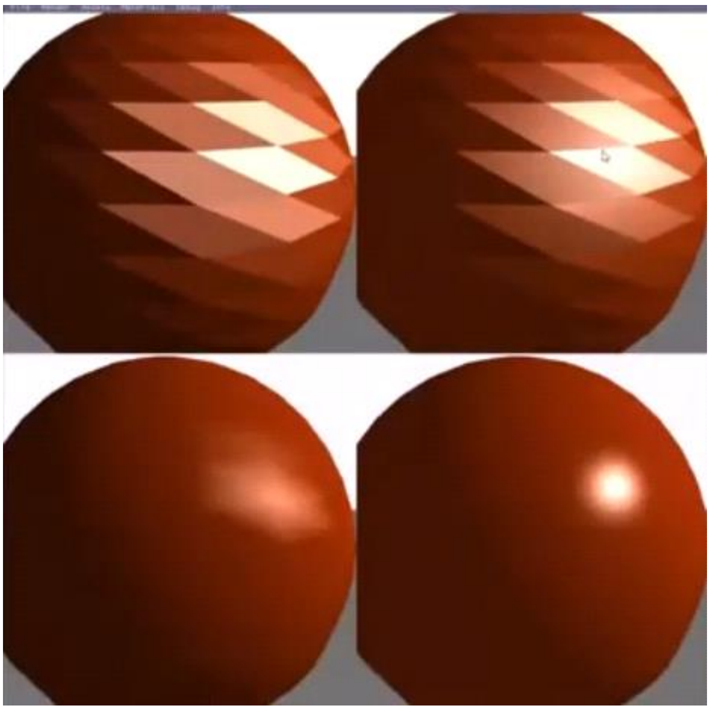
Gouraud
- Para cada vértice calcula-se uma normal
  - Calcular a equação de iluminação por vértice
- Para cada pixel a cor é calculada por interpolação da cor dos vértices

Phong
- Para cada vértice calcula-se uma normal
- Para cada pixel calcular a normal do pixel
  - Interpolação das normais dos vértices
- Calcular a cor usando a normal do pixel

Teoricamente, Phong será mais rápido numa cena em que há mais triângulos do que pixeis (+ número de pixeis limitado ao tamanho da janela).

Flat shading

Flat shading + Interpolação

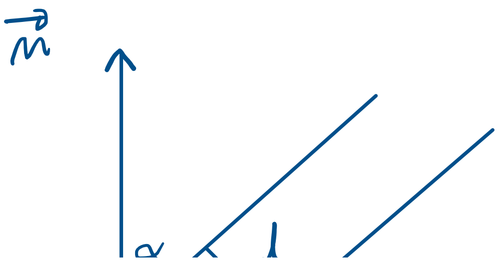


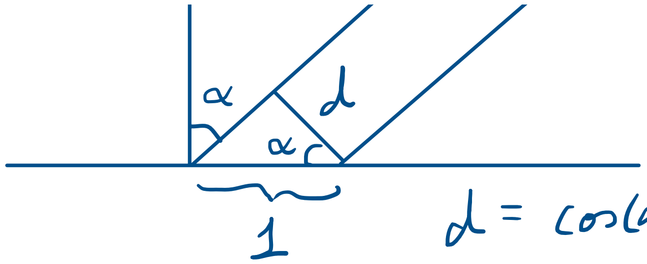
Gouraud

Phong

(4)

o Iluminação Difusa



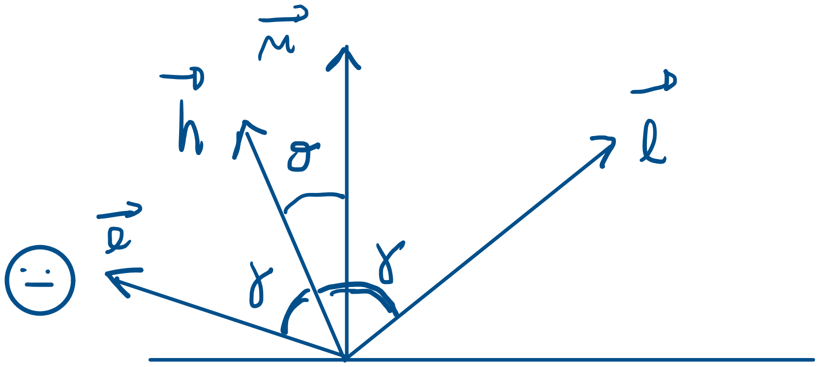



$d = \cos(\alpha)$

$$I_d = \left(L_d \cdot K_d \cdot \cos(\alpha)\right) \times f_{att}$$

Atenuação: $\frac{1}{d^2}$

- Iluminação Especular



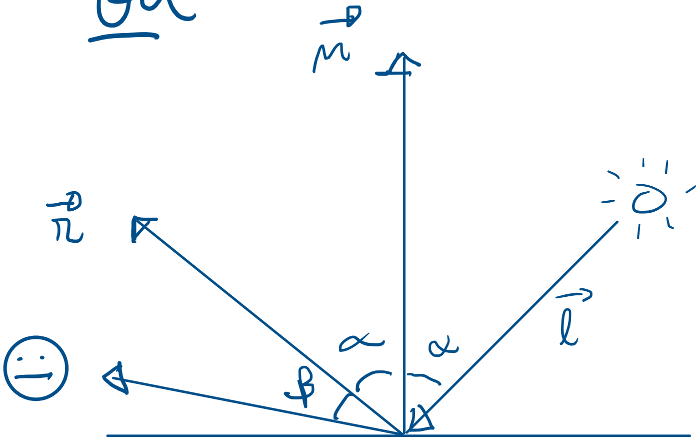

$$h = \frac{\vec{e} + \vec{l}}{|\vec{e} + \vec{l}|}$$

$$I_s = L_s K_s \cdot \cos^s(\theta), \quad s \in [0, 128]$$

↓

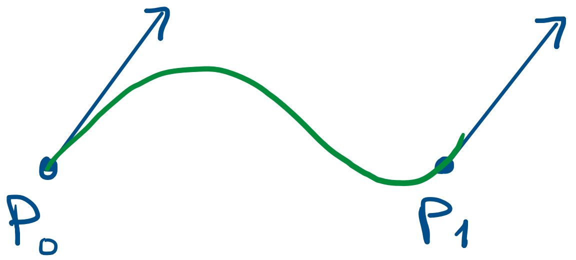
Shininess

<u>ou</u>



## 5 Curvas de Hermite X

É de Bezier afinal




$$f(t) = at^3 + bt^2 + ct + d$$

$$f'(t) = 3at^2 + 2bt + c$$

$$P(0) = d \quad ; \quad P'(0) = c$$

$$P(1) = a + b + c + d \ ;$$

$$P'(1) = 3a + 2b + c$$

1. $Q_1$ tem de ser igual a $P_4$
2. $Q_1 - Q_2$ e $P_3 - P_4$ têm de fazer um segmento de reta contínuo
3. $Q_1 - Q_2$ e $P_3 - P_4$ têm de ser o mesmo trabalho

$$P(t) = (1-t)^3 P_1 + 3(1-t)^2 t P_2 + 3(1-t)t^2 P_3 + t^3 P_4$$

$$Q(t) = (1-t)^3 Q_1 + 3(1-t)^2 t Q_2 + 3(1-t)t^2 Q_3 + t^3 Q_4$$

$$P'(t) = -3(1-t)^2 P_1 + 3[(1-t)^2 - 2t(1-t)]P_2 + 3[2t(1-t) - t^2]P_3 + 3t^2 P_4$$

$$Q'(t) = -3(1-t)^2 Q_1 + 3[(1-t)^2 - 2t(1-t)]Q_2 + 3[2t(1-t) - t^2]Q_3 + 3t^2 Q_4$$

$$P'(1) = 3(P_4 - P_3)$$

$$Q'(0) = 3(Q_2 - Q_1)$$

$$P_4 = Q_1$$

$$\Rightarrow 3(P_4 - P_3) = 3(Q_2 - Q_1)$$

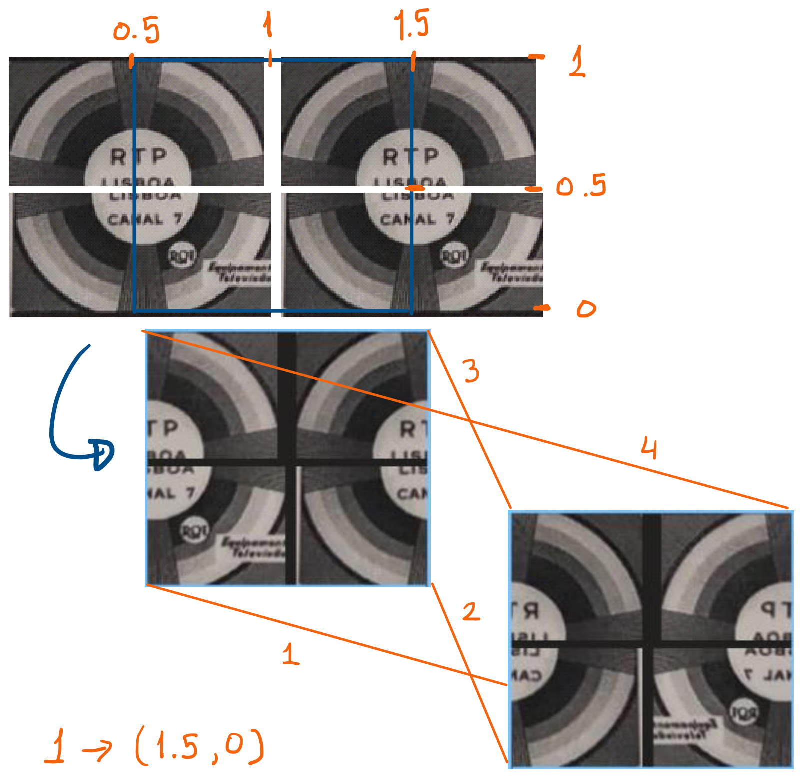
$$\Leftrightarrow P_4 - P_3 = Q_2 - Q_1$$

$$\Leftrightarrow Q_2 = P_4 - P_3 + Q_1$$

$$\Leftrightarrow Q_2 = 2P_4 - P_3$$

R.: Opção B

6




$1 \rightarrow (1.5, 0)$

$2 \rightarrow (0.5, 0)$

$3 \rightarrow (0.5, 1)$

$4 \rightarrow (1.5, 1)$

3 → (0.5, 1)
4 → (1.5, 1)

7 Ver exercício 7 do teste de 21/22

7

A KD-trees surgem como uma forma de remover o grau de liberdade da orientação dividindo a imagem em planos perpendiculares ao eixo x/y e vice-versa na próxima iteração, terminando com uma árvore binária que facilita a definição dos objetos a renderizar com o frustum.

Critérios de Paragem:

- nº de polígonos chegar a um limite
- profundidade da árvore ser muito grande
- a célula ser demasiado pequena

8

```
void renderScene (void) {
    int r = 15;
    (...)
    for (int i = 0; i < n; i++) {
        glPushMatrix ();
        glRotate (45 * i, 0, 1, 0);
        glTranslate (r, 1, 0);
        glTeapot(2);
        glPopMatrix();
    }
    (...)
}
```