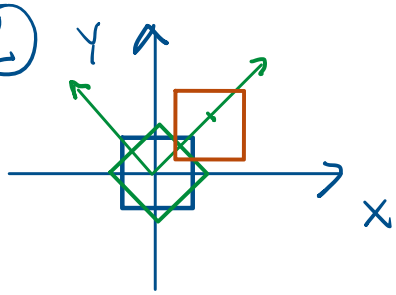
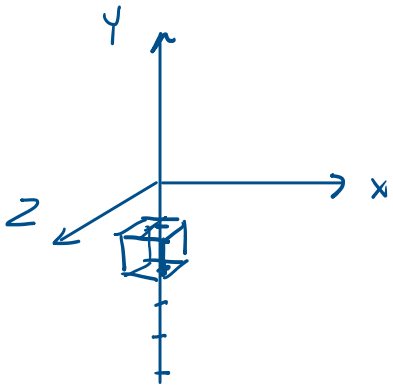
Teste 2012/2013

18 de maio de 2024 00:47

1



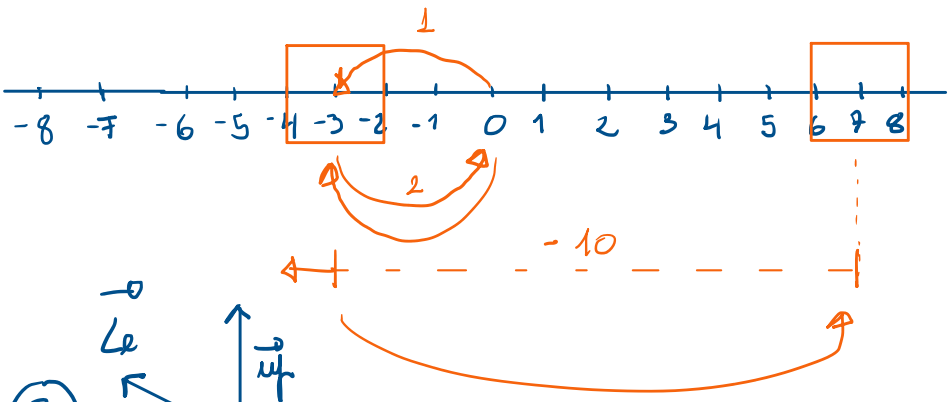
OPÇÃO C



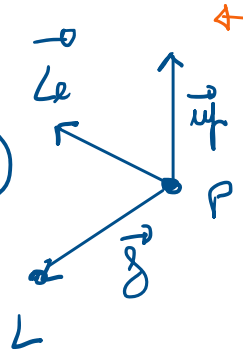
2

2.1	V
2.2	F
2.3	V
2.4	V
2.5	F

Direção em que se movimenta no eixo, depende da direção da câmara



3



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

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

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

4

1º Rotação
2º Translação
3º Scale

5

Double buffering é uma técnica utilizada em computação gráfica para evitar flickering (tremulação) e tearing (rasgamento) ao renderizar imagens. Consiste em usar dois buffers: o buffer traseiro (back buffer), onde a imagem é desenhada, e o buffer frontal (front buffer), que

Double buffering é uma técnica utilizada em computação gráfica para evitar flickering (tremulação) e tearing (rasgamento) ao renderizar imagens. Consiste em usar dois buffers: o buffer traseiro (back buffer), onde a imagem é desenhada, e o buffer frontal (front buffer), que exibe a imagem na tela. Após a renderização completa no back buffer, os conteúdos dos buffers são trocados rapidamente, garantindo que a imagem apresentada ao usuário seja sempre completa e estável, melhorando a qualidade visual de aplicações interativas como jogos e simulações.

- Evitar: FLICKERING, TEARING
- Após a renderização completa do back buffer, os conteúdos são rapidamente trocados para o front buffer
  ⇒ imagem apresentada é sempre completa e estável

(6)

Mipmapping é uma técnica usada em computação gráfica para melhorar a aplicação de texturas em objetos tridimensionais. Consiste em criar e armazenar múltiplas versões de uma textura, cada uma com uma resolução progressivamente menor, formando uma pirâmide de texturas. Durante o render, a versão da textura mais apropriada é escolhida com base na distância do objeto à câmera, reduzindo artefatos visuais como aliasing e melhorando a eficiência do cache de textura.

As vantagens do mipmapping incluem uma renderização mais suave e realista, especialmente para objetos distantes, além de melhorar o desempenho gráfico ao diminuir a carga de processamento e a memória necessária para texturas detalhadas. No entanto, as desvantagens incluem o aumento do uso de memória devido ao armazenamento das múltiplas versões das texturas e a complexidade adicional no processo de criação e gerenciamento dessas mipmaps.

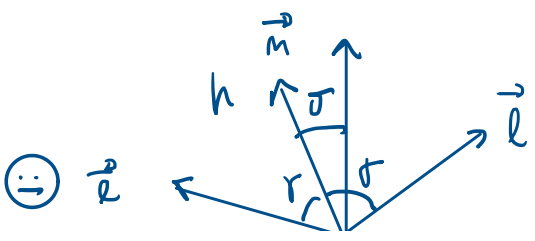
→ Pré-processamento

(7) $I = (I_d \cdot K_d \cdot \cos(\alpha)) \times f_{att}$

$I = I_s \cdot K_s \cos(\sigma)^s$

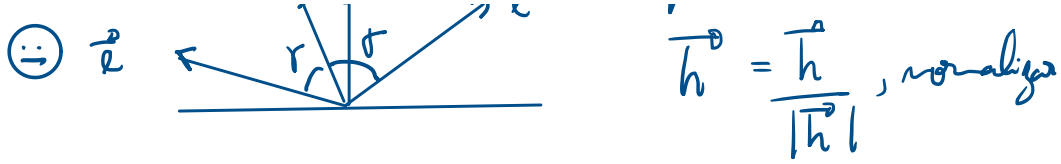
↳ $\frac{1}{d^2}$

↳ shininess



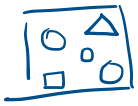
$\vec{h} = \vec{e} + \vec{l}$

$\vec{h}^0 = \frac{\vec{h}}{}$, normaliza



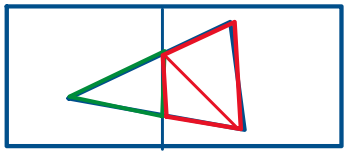
$\vec{h}' = \frac{\vec{h}}{|\vec{h}|}$, normaliza

(8)

1. Bounding Box 

2. Dividir volume delimitador em sub-regioes iguais

3. Distribuir vértices pelas sub-regioes



A: Guardar as duas células como um nodo pai
- subutilização da placa gráfica

B: Partir o triângulo em 3 triângulos, e guardar cada um na célula correspondente
- processar mais vértices

C: Duplicar o triângulo <- mais beneficios em termos de CG
- a maior parte dos triângulos não atravéssam fronteiras das células
- gasta-se mais memória (mas só em termos de índices)
- se e quando for desenhar o segundo, verifica que já foi desenhado outro nessa profundidade e os pixeis não são processados

4. Recursividade

Para cada sub-região que contém vértices, criar um novo nó filho
Repetir o processo de divisão para cada um
Casos de paragem:

Quando parar a divisão?
- quando o número de triângulos for menor do que um determinado valor, t
- quando o volume da célula for menor do que um determinado valor, c
- quando a profundidade da árvore for maior do que um determinado valor, p

(9)

```
Sphere(1); // Sol

Rotate(alpha, 0, 1, 0) // rotação em torno do eixo dos YY

Translate(10, 0, 0); // distância entre o planeta e o sol

Rotate (10, 0, 0, 1); // inclinação orbital

Sphere(0.25); // planeta
```

```
Sphere(1); // Sol

Rotate(alpha, 0, 1, 0) // rotação em torno do eixo dos YY

Translate(10, 0, 0); // distância entre o planeta e o sol

Rotate (10, 0, 0, 1); // inclinação orbital

Sphere(0.25); // planeta

Rotate(beta, 0, 1, 0); //  Nova rotação em torno dos YY para afectar a posição da lua

Translate(1.5, 0, 0); // distância entre o planeta e a lua

Sphere (0.1); // lua
```