


P1 CG 2015.1

	Computação Gráfica 1 Prof. Rodrigo de Toledo Apoio: Pedro, Guilherme e Raphael	P1 2015.1 Data: 8/6/2015
---	---	------------------------------------

1) (2 pontos) Vimos em aula, **três** usos para a coordenada homogênea: translação, pontos no infinito (direção) e perspectiva. Explique separadamente para cada um deles, como a coordenada homogênea é usada.

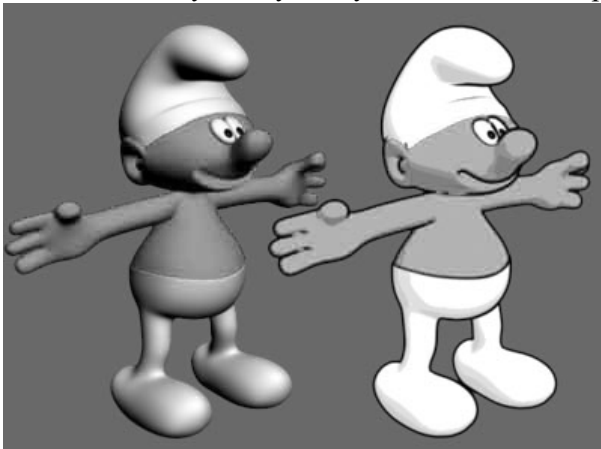
2) (1 ponto) Por que numa imagem de Normal Map em geral a cor predominante é o azul?

3) (2 pontos) Em um determinado momento do Dojo que fizemos em aula o código abaixo funcionava, porém não para todos os casos. Escreva um código para que funcione para quaisquer pontos P1 e P2 válidos.

```
QPoint p1= QPoint(10,9);
QPoint p2= QPoint(1,1);
int a = (p2.ry() - p1.ry())/(p2.rx() - p1.rx());
int b = p1.ry() - (a*p1.rx());
for(int x = p1.rx(); x < p2.rx(); x+=1){
    printPixel(x, ((a*x)+b) );
}
printPixel(p2.rx(),p2.ry());
```

4) (1 ponto) Cite dois exemplos diferentes de *aliasing* e como resolvê-los.

5) (2 pontos, P2 2011.1) Escreva a equação do 2º grau no formato $at^2 + bt + c = 0$ que determina a interseção entre um raio e uma esfera. Considere um raio com origem em P0 passando em P1 (por exemplo, P0 é a câmera e P1 o centro de um pixel); e uma esfera de raio r centrada em O. Dica: separe as componentes x, y, z; para o raio, faça uso de $\Delta x = P1.x - P0.x$, $\Delta y = P1.y - P0.y$, $\Delta z = P1.z - P0.z$; para a esfera use a equação: $(x-O.x)^2 + (y-O.y)^2 + (z-O.z)^2 = r^2$.



6) (2 pontos) O Cartoon Shader tem a intenção de deixar o objeto com aparência de um desenho/cartoon. A imagem ao lado ilustra como é o resultado de um Cartoon Shader (aka Cell Shading). Existem diversas técnicas para criar esse resultado. Esta abaixo é uma técnica simples:

- O sombreadimento que era aplicado de forma contínua, são discretizadas em apenas algumas sombras. (Ao invés de 50 tons de cinza, apenas 4 tons por exemplo).
- Na silhueta do objeto criamos um traço preto (outline), típico de um desenho cartoon.

Modifique o código do fragmento abaixo para aplicar um efeito semelhante ao da imagem.

```
void main()
{
    vec3 normalDirection = normalize(varyingNormalDirection); // Normal do Obj.
    vec3 viewDirection = normalize(_WorldSpaceCameraPos - vec3(position)); // Direção da
Câmera
    vec3 lightDirection = normalize(vec3(_WorldSpaceLightPos0)); // Direção da Luz.

    float cosNormalLight = max(0.0,dot(normalDirection,lightDirection)); // Intensidade
de Luz
    vec3 diffuseReflection = vec3(_LightColor0) * vec3(_Color) * cosNormalLight;
```

```
gl_FragColor = vec4(diffuseReflection, 1.0);
}
```

Gabarito

1)

2) Para simular o efeito de relevo irregular de um objeto, utilizamos um mapa de vetores normais. Este mapa é salvo como uma imagem, portanto convertemos cada vetor normal XYZ em uma cor do sistema RGB. Como geralmente a superfície de um objeto é lisa, seus vetores normais, tem o componente Z com maior valor. Apenas nas irregularidades temos o verde e vermelho mais relevantes. Deste forma, a maioria dos pixels do normalMap terá a cor azul, representando um vetor normal da parte lisa.

Uma observação é que o sistema de referencia de coordenadas do mapa de normal é a superfície do objeto. Por isso o Z representa a altura.

4)

(bastavam dois pares de problema/solução)

Problema: resolução da textura (em texels) bem maior que sua projeção na tela (em pixels). Solução: Mipmapping, pois reproduz a textura em resoluções mais baixas e a mais adequada é usada na hora da renderização.

Problema: desenho de reta (Brenseham) com efeito de degraus nos pixels. Solução, usar subpixels e aplicar tons intermediários quando parte dos subpixels é acesa.

Problema: contorno de objetos no ray-tracing. Solução: tirar a média de subpixels

5)

O ponto de interseção pode ser representado por

$P_0 + t\Delta$

Substituindo:

$$(P_0x + t\Delta x - O.x)^2 + (P_0y + t\Delta y - O.y)^2 + (P_0z + t\Delta z - O.z)^2 = r^2.$$

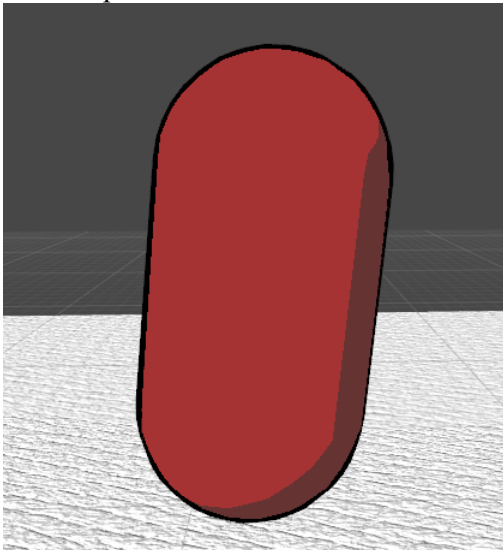
Resposta final:

$$\Delta \bullet \Delta t^2 + 2\Delta \bullet (P_0 - O) t + (P_0 - O) \bullet (P_0 - O) - r^2 = 0$$

6) Existem várias maneiras de resolver. Na correção da questão, não consideraremos errado se o valores estiverem diferentes, apenas a ideia precisa ser parecida. Ou se a ideia for diferente, o resultado precisa ser parecido.

A variável `cosNormalLight` define a intensidade de luz que incide no objeto. Os if's fazem com que essa intensidade só possa ter 4 valores, tornando esse efeito discreto. Você poderia usar outras maneiras de discretizar, qualquer maneira que seja coerente, é considerada correta.

Quando o observador está olhando para a silhoueta de um objeto, o cosseno entre o raio que saiu do observador e a normal do objeto, formam um angulo aproximadamente reto. Então ao termos o cosseno próximo de zero. Pintamos o fragmento de preto.



FRAGMENTO

```
void main()
{
    vec3 normalDirection = normalize(varyingNormalDirection);
    vec3 viewDirection = normalize(_WorldSpaceCameraPos - vec3(position));
```

```
vec3 lightDirection = normalize(vec3(_WorldSpaceLightPos0));

float cosNormalLight = max(0.0,dot(normalDirection,lightDirection));
// Parte que discretiza o efeito de iluminação nas cores. Valia 1 ponto.
if (cosNormalLight > 0.85)
    cosNormalLight = 1.0;
if (cosNormalLight > 0.50)
    cosNormalLight = 0.85;
if (cosNormalLight > 0.25)
    cosNormalLight = 0.50;
if (cosNormalLight < 0.25)
    cosNormalLight = 0.0;
vec3 diffuseReflection = vec3(_LightColor0) * vec3(_Color) * cosNormalLight;

// Parte que cria o outline. Valia 1 ponto.
if (dot(viewDirection,normalDirection) < 0.20)
    gl_FragColor = vec4(0.0,0.0,0.0,1.0); // Black
else
}
```

Publicado por [Google Drive](#) – [Denunciar abuso](#) – 5Atualizado automaticamente a cada minutos
