Computação Gráfica

Exame 21/07/007

Duração: 2 horas

Parte I - OpenGL

1. O OpenGL fornece vários mecanismos de aceleração, nomeadamente as Display Lists e os Vertex Buffer Objects. Compare este dois mecanismos.

2. O OpenGL utiliza um modelo de iluminação local. Faça uma breve análise comparativa, em termos de qualidade dos resultados e desempenho, deste tipo de modelos vs. modelos de iluminação global.

3. Apresente e descreva a equação de iluminação utilizada pelo OpenGL considerando uma luz direccional com as componentes ambiente, difusa e especular.

4. Os dois segmentos de código apresentados em seguida geram situações distintas de iluminação. Descreva as diferenças do ponto de vista do resultado entre as duas opções apresentadas. Justifique a resposta dada.

   a)
   ```
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
   gluLookAt(x,y,z, 0.0,0.0,0.0, 0.0f,1.0f,0.0f);
   ```

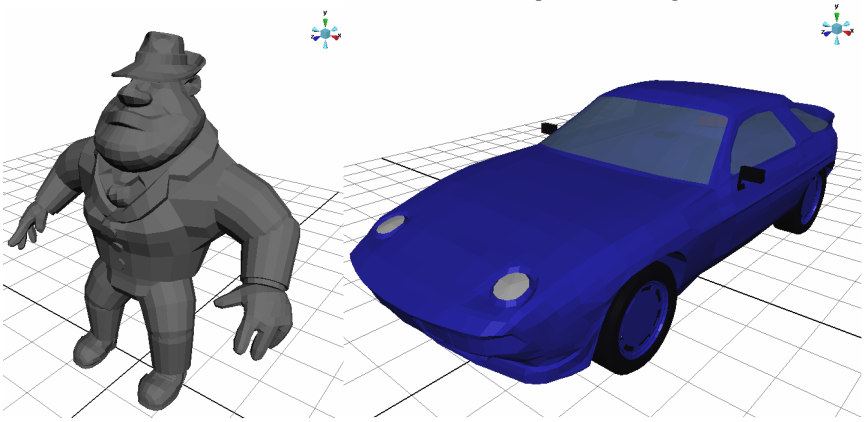
   b)
   ```
   gluLookAt(x,y,z, 0.0,0.0,0.0, 0.0f,1.0f,0.0f);
   glLightfv(GL_LIGHT0, GL_POSITION, pos);
   ```

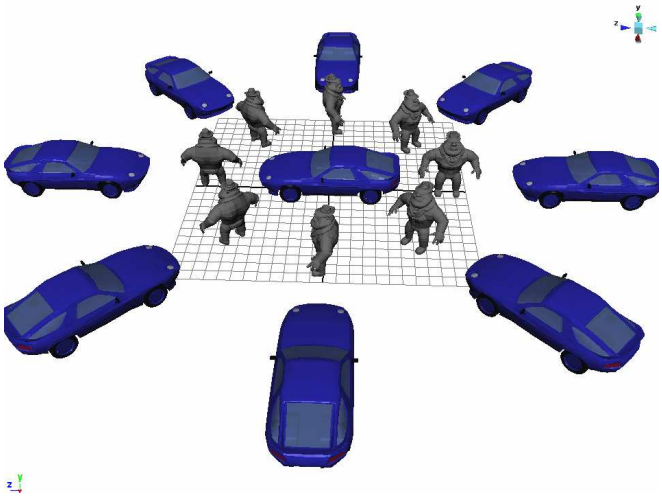
5. Descreva o conceito de *double buffering.*

6. Dados os três vértices de um triângulo, v0, v1 e v2, indique a sequência de passos para efectuar a operação para o cálculo do vector perpendicular à superfície do triangulo, apresentando as fórmulas necessárias.

7. Considere a biblioteca gLCC que contem primitivas gráficas para Al Capones e Porsches como se ilustra nas figuras. Escreva uma função em C que permita construir em OpenGL um modelo semelhante ao apresentado na figura com a cena da página seguinte.

   Como referência, em termos de medidas, o lado dos quadrados da grelha vale uma unidade.





Parte II – Shaders

Vertex Shader	**Fragment shader**

void main() {
...
gl_Position = ftransform();
}
	void main () {
	vec3 color;
	...
	gl_FragColor = color;
	}

1. Considere que se pretende simular nevoeiro (fog) por *pixel* presente no OpenGL através de *shaders*. Considerando *z* a distância do ponto à câmara, *start* e *end* como as distâncias ao *near* e *far* plane respectivamente, o factor do nevoeiro, *f*, é calculada da seguinte forma:

$$f = \frac{end - z}{end - start}$$

A cor do fragmento é então calculada como sendo uma combinação linear entre a cor do nevoeiro (Cf) e a cor original do fragmento (Cr):

$$Cfinal = f*Cr + (1-f)*Cf$$

Escreva o *shader* seguindo a especificação acima apresentada, e indique quais devem ser as variáveis *uniform*.

2. Pretende-se criar um *shader* que permita emular a funcionalidade dos *clip planes* do OpenGL, ou seja, para cada *pixel* é determinado de que lado do plano é que este se encontra e caso o resultado seja negativo o *pixel* é descartado. Escreva o código dos *shaders* assumindo que a informação do plano é acessível através de uma variável *uniform*. Considere ainda que o espaço em que o plano actua é o espaço global.

3. Através da programação de *shaders*, é possível calcular a iluminação por *pixel*, enquanto que a funcionalidade fixa só efectua este cálculo por vértice. Descreva detalhadamente, de um ponto de vista do resultado visível, qual a diferença entre a iluminação por *pixel* e por vértice.

4. Descreva o trabalho realizado entre o processador de vértices e o processador de fragmentos.

5. Descreva as responsabilidades e limitações de um programa a correr num processador de fragmentos.