Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Computação Gráfica
AD2 - 2° semestre de 2015.

1) Ao programar um jogo, sabe-se que texturas com alta resolução podem trazer problemas quando o objeto em questão está longe da câmera. Qual é este problema? Como corrigir? (1.0 ponto)

São problemas referentes a aliasing: Ocorre a amostragem em um pixel da imagem em uma área grande da textura, correspondente a dois ou mais texels. Podem ocorrer ruídos e pedaços da textura aparecendo e desaparecendo aleatoriamente. Para corrigir é necessário aplicar filtros na textura.

2) Em relação a questão anterior, qual o problema que pode surgir no caso da textura estar em baixa resolução (1.0 ponto).

Neste caso ainda temos o problema de aliasing, porém ocorre uma falta de amostragem correta. Para vários pixels da tela de projeção são mapeados o mesmo pixel da textura, podendo ocorrer o problema de serrilhamento. Para corrigir é necessário interpolar pixels vizinhos da textura. Esta solução no entanto irá gerar texturas "borradas".

3) Explique a diferença que há entre Culling e Clipping (1.0 ponto).

Culling de polígonos consiste em descartar polígonos que não são visíveis para a câmera, seja por razoes de oclusao ou por razoes de estarem fora do câmera frustum. Algoritmos de Frustrum Culling eliminam polígonos fora do view frustrum e algoritmos de Occlusion Culling eliminam polígonos que estão ocultos por outros. Já o Clipping consiste em recortar polígonos ou partes de polígonos que ficaram fora da área delimitadora da tela de projeção.

4) Explique a diferença entre ilumiunação por pixel e iluminação por vértice (1.0 ponto)

Na iluminação por pixel, aplica-se a equação de iluminação (por exemplo, Phong) para cada ponto de superficie de um objeto que corresponde sua projeção num dos pixels da tela. Já na iluminação por vértice aplica-se o cálculo de iluminação para cada vértice, independente de sua projeção na tela. No estágio de rasterização será feita uma interpolação das cores dos vértices, preenchendo o interior do polígono.

5) Escreva em alto nível um programa de Ray-tracing, destacando a sua chamada recursiva (1.0 pontos)

```
Ray_Tracing(VETOR)
Para cada Pixel da Imagem faça
  Objeto_mais_próximo = Nenhum
  Distância_mínima = infinito
```

```
Crie um raio do observador ao pixel
        Para cada objeto da cena faça
        Se o raio tem interseção com este objeto
                Se Distancia_mínima < distancia (câmera até este objeto)
                        Objeto_mais_próximo = este objeto
                Fim se
        Se Objeto_mais_próximo = Nenhum
                Pixel = Cor_de_fundo
        Senão
                Reflexo  = Calcula_reflexo(Objeto_mais_próximo, luz)
                Transmissão = Calcula_transmissão (Objeto_mais_próximo, N)

            Pixel = Phong(Objeto) + Ray_Tracing(Reflexo) + Ray_Tracing(Transmissão)
        Fim Se
  Fim se
Fim faça
Fim faça
```

As chamadas recursivas estão em negrito.

6) Como se faz para resolver o problema de profundidade na rasterização de polígonos? (Expliquem detalhadamente) (1.0 ponto)

Utiliza-se o algoritmo de Z-Buffer. Este algoritmo consiste em criar uma memória de tamanho equivalente ao frame-buffer. Sempre que um pixel referente a um triangulo for pintado no frame-buffer, será escrito no Z-buffer a profundidade do mesmo. Caso já haja uma profundidade escrita anteriormente no Z-Buffer, antes de pintar o pixel será feita uma consulta se este novo pixel possui profundidade maior ou menor. Caso seja maior, permite-se a sua escrita, em cima do anterior. Neste caso atualiza-se o valor do Z-Buffer com o valor deste novo pixel. Caso o valor de profundidade no Z-buffer seja menor, impede-se a escrita do pixel no frame-buffer, pois já há um pixel mais próximo pintado previamente.

7) O Ray tracing é um ótimo método para renderizar funções implícitas. Explique (1.0 ponto).

O Ray tracing, por realizar um cálculo exaustivo de interseções com a geometria da cena, pode usar neste cálculo equações que não sejam necessariamente de polígonos, mas de outras superfícies. Na rasterização isto é praticamente impossível, devido ao pipeline baseado em vértices e interpolação de polígonos.

8) O que é a linguagem CUDA para GPUs? O que é um kernel, dentro deste contexto? (1.0 ponto)

As GPUs se tornaram genéricas o suficiente para poderem executar programas que não são necessariamente etapas do pipeline gráfico. A linguagem CUDA foi criada para poder acessar a GPU de forma mais genérica e poder integrar com programas convencionais. Kernels são funções escritas em CUDA e que serão executadas pela GPU.

9) Escreva o corpo de uma aplicação típica do OpenGL, destacando em comentários alguns dos estágios do pipeline gráfico. (1 ponto)

A seguir a CALLBACK responsável por desenhar um cubo sem o fundo. Aplicando uma textura nas paredes.

```
void desenhar(void)
{
    //Define os parâmetros da câmera que serão usados no estágio de culling
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
            glOrtho(-300.0f, 420.0f, -80.0f, 15.0f, -1.0f, 2000.0f);
    gluLookAt(100.0f, 100.0f, -100.0f,  0.0f, 0.0f, 0.0f,  0.0f, 1.0f,0.0f);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();


    glPushMatrix();
    glScale(2.0f, 2.0f, 2.0f); //Aplica a escala para dobrar o tamanhos do quads. Este comando é processado pelo estágio de processamento de vértice

    glTranslatef(72.0f , 0.0f, 72.0f );  //Desloca os quads para a posição 72, 0, 72. Este comando é processado pelo estágio de processamento de vértice

    glColor3f(1.0f, 1.0f, 1.0f);   //Define cor branca para não afetar a cor da textura.

    glBindTexture(GL_TEXTURE_2D, 1);  //Efetua o bind da textura – a textura é processada no estágio de processamento de pixel

    glBegin(GL_QUADS);
    glTexCoord2f(0.0f, 0.0f);
    glVertex3f(0.0f, 0.0f, 0.0f);

    glTexCoord2f(0.0f, 1.0f);
    glVertex3f(0.0f, fHeight, 0.0f);

    glTexCoord2f(1.0f, 1.0f);
    glVertex3f(fWidth, fHeight,0.0f);


    glTexCoord2f(1.0f, 0.0f);
    glVertex3f(fWidth, 0.0f, 0.0f);//Aplica a textura que é processada no estágio de processamento de pixel


    glutSwapBuffers();  //Troca o framebuvver enviando o buffer construído para a saída de vídeo
}
```

10) Como incluir, no modelo de iluminação Phong, várias fontes de luz? (1 ponto)

Para tratar n fontes de luz no Phong deve-se criar um somatório da componente difusa e especular para cada luz. A componente ambiente deve ser uma só, pois não depende da fonte:

$$I_p = k_a i_a + \sum_{m \in \text{lights}} (k_d(\hat{L}_m \cdot \hat{N}) i_{m,d} + k_s(\hat{R}_m \cdot \hat{V})^{\alpha} i_{m,s}).$$