Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
Disciplina: Computação Gráfica
AD2 - 2° semestre de 2009.

1) Explique o que é a matriz de transformação 4x4 $\mathbf{M_{obj}}$, em coordenadas homogêneas, que realiza a mudança do sistema de coordenadas local para o sistema de coordenadas da câmera (Dica: a coordenada local primeiro precisa ser transformada para coordenada de mundo para depois realizar a transformação para coordenada de câmera) (1.0 ponto). `

A matriz M_{obj} que faz a mudança de coordenadas do sistema do mundo para o sistema da câmera é dada pelo produto das matrizes R_{ew} e T_{ew}.

$$M_{obj} = R_{ew} \circ T_{ew} = \begin{bmatrix} x_{ex} & x_{ey} & x_{ez} & 0 \\ y_{ex} & y_{ey} & y_{ez} & 0 \\ z_{ex} & z_{ex} & z_{ex} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -eye_x \\ 0 & 1 & 0 & -eye_y \\ 0 & 0 & 1 & -eye_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

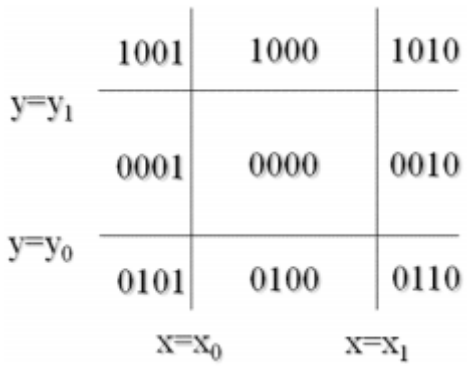
onde T_{ew} posiciona os sistema que define o referencial da câmera na origem e R_{ew} alinha os eixos do sistema da câmera com os eixos canônicos.

2) Descreva o algoritmo de *Sutherland* e *Hodgman* para recorte de polígonos. Considere que a janela de recorte é dada por uma região retangular (1.0 ponto).

O algortimo de *Sutherland* e *Hodgman* é semelhante ao algoritmo de *Cohen-Sutherland*. O polígono é recortado sucessivamente contra todos os lados da figura de recorte, ou seja, a janela de recorte retangular. O algoritmo trabalha sobre uma lista circular de vértices, sendo que os vértices, e também as arestas que os conectam, são processados em seqüência e classificados contra o lado corrente do polígono de recorte.

```
Criar um lista circular le contendo os vértices do polígono.

PARA (cada lado l da janela)

    PARA (cada vértice v de le)

        SE  (v está dentro da janela de recorte) ENTÃO
            Copiar v para a lista de saída ls
        FIM_SE

        SE (a aresta formada por v  e o sucessor v' de v intersecta l) ENTÃO
            Calcular ponto de interseção v".
            Copiar v" para a lista de saída ls.
        FIM_SE

    FIM_PARA

    le ← ls

FIM_PARA
```

3) Escreva um algoritmo que determina a classificação de um ponto, através de um código de 4 bits, com relação às nove regiões determinadas por uma janela retangular (1.0 ponto).




```
unsigned char code(double x, double y,
            double xmin, double xmax, double ymin, double ymax)
{
  unsigned char code=0;

  if (y > ymax) code += 8;
  if (y < ymin) code += 4;
```

```
  if (x > xmax) code += 2;
  if (x < xmin) code += 1;

  return code;
}
```

4) Como funciona a equação de iluminação Phong quando há mais de uma fonte de luz? (1.0 ponto)

A equação de Phong é dada por

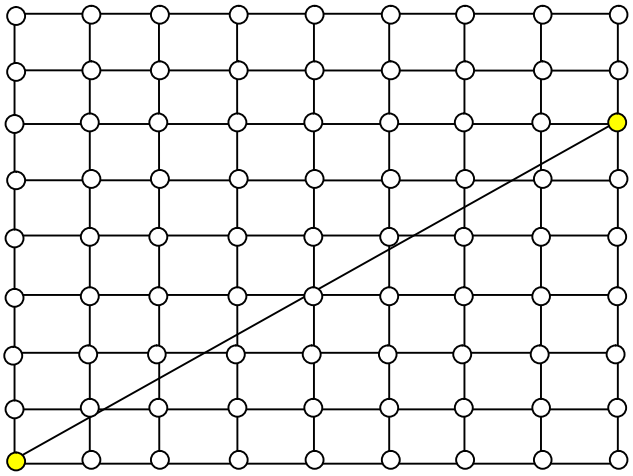
$$I = K_a.I_a + \sum_{i=1}^{n} k_d I_i < N, L > + K_e I_i < O, R >)$$

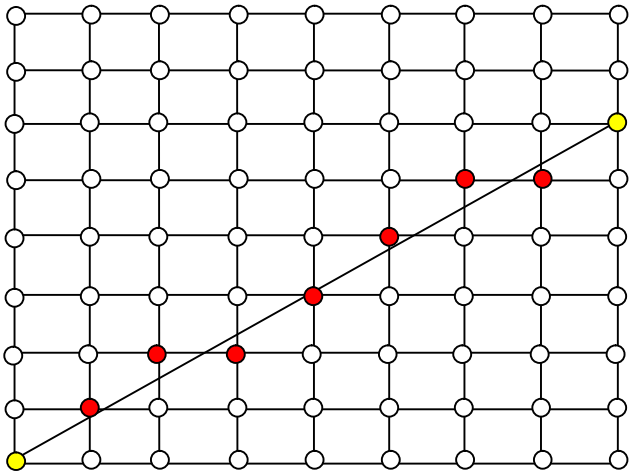
onde K_a, K_d e K_e são as constantes do material, descrevendo como reagem a cada um dos componentes de iluminação, sendo Ia o coeficiente de iluminação ambiente. O somatório significa que será efetuado um cálculo de difuso e especular para cada uma das fontes de luz, de um total de n. I_i é a intensidade de iluminação da fonte de luz, N é a normal do ponto que se está calculando, O é o vetor do observador ao ponto da superfície em questão e R é o vetor de reflexo no ponto da superfície.

5) Qual o algoritmo ou método usado para tratar iluminação ambiente de forma ideal? (1.0 ponto)

Há várias maneiras de resolver o problema, sendo algumas abordagens mais simples e outras mais complexas. Uma delas seria utilizar juntamente com o modelo de iluminação Phong o algoritmo de Radiosidade, próprio para tratar iluminação ambiente com precisão. Outra possibilidade, muito usada pelos artistas 3D, consiste em utilizar várias luzes auxiliares, com intensidades pequenas e espalhadas pela cena. O problema come esta última abordagem é que o número de fontes de luz aumenta o custo computacional do pipeline gráfico tornado o difícil de ser aplicado em tempo real.

6) Considerando o algoritmo do ponto-médio para rasterização de retas, determine quais *pixels* devem ser acessos no reticulado abaixo. Considere os *pixels* como sendo os elementos na interseção das retas do reticulado conforme ilustrado na figura abaixo (1.0 ponto).





7) Que estratégia é usada para que ao renderizar uma imagem no back-buffer os polígonos que estão atrás de outros não se sobreponham aos que estão na frente? (Explique em detalhes) (1 ponto).

O Algoritmo *z-buffer* é o algoritmo mais utilizado para solucionar o problema de remoção de superfícies escondidas. O algoritmo armazena para cada *pixel* a distância existente entre o observador e a superfície mais próxima, a qual é a superfície visível. O *Z-Buffer* cria um *array* bidimensional contendo informação sobre a profundidade de cada *pixel*.

Durante o processo de rasterização, são gerados fragmentos que são candidatos a se tornarem um pixel caso, passem por todos os testes que fazem parte do pipeline gráfico e venham efetivamente a gerar informações no *framebuffer*. O teste de profundidade compara a coordenada de profundidade do fragmento com a coordenada de profundidade armazenada na posição correspondente *no array bidimensional* (o Z-buffer). Caso a

coordenada de profundidade do fragmento seja menor (mais próxima da tela de projeção) que a coordenada já armazenada, escreve-se a informação de cor do fragmento, na posição correspondente no buffer de cores, ao mesmo tempo em que a coordenada de profundidade (o valor de z) no Z-buffer é atualizada com o valor da profundidade do fragmento.

Abaixo descrevemos sucintamente os principais passos do algoritmo Z-buffering.

1. Criar e inicializar com a cor de fundo um *array* bidimensional (buffer de cor) que conterá as informações de cor de cada *pixel* da tela;

2. Inicializar o *array* de profundidades (Z-buffer) com o valor da profundidade máxima;

3. Calcular a coordenada z de cada fragmento gerado para os polígonos rasterizados;

4. Para cada fragmento, testar se a sua profundidade z é menor do que o valor armazenado na posição correspondente no z-buffer. Em caso positivo, atualizar o valor de profundidade no z-buffer e a armazenar a informação de cor no buffer de cores.

8) O que é mapeamento de textura? (1 ponto)

Em Computação Gráfica, mapeamento de textura consiste na técnica que toma uma superfície e altera sua aparência em cada ponto, usando uma imagem, função ou outra fonte de dados. Dados dois objetos $O_1=(U,f)$ e $O_2=(V,g)$, um mapeamento de O_1 em O_2 é uma transformação $T:V\rightarrow U$.

Figura 1. Mapeamento de textura.

O objeto $O_1=(U,f)$ é denominado ***objeto fonte*** e o objeto $O_2=(V,g)$ é denominado ***objeto alvo.*** O mapeamento T define uma nova função de atributos g* no objeto mapeado O_2, onde $g^*=f\circ T$.

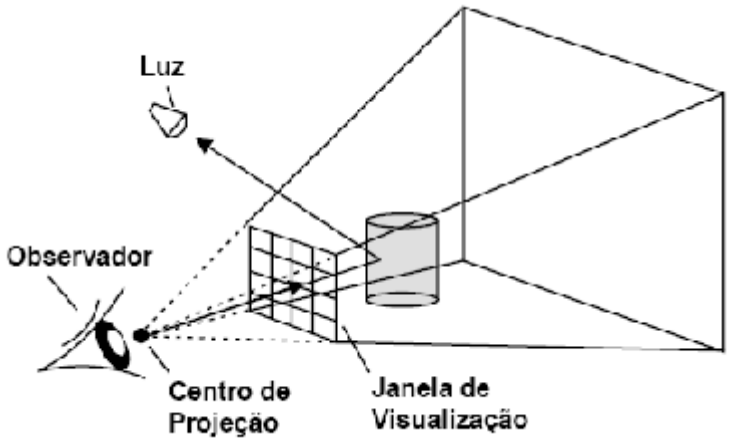
No mapeamento de texturas, considerando, por exemplo, o caso bidimensional, desejamos determinar as coordenadas de textura (u,v) de U para cada ponto no suporte geométrico V de O_2. Logo, a transformação é de V para U.

Existem inúmeras formas de se realizar o mapeamento o que dá origem a diversas técnicas de mapeamento.

9) Descreva o algoritmo de traçado de raios (*Raytracing*), mostrando onde ocorre a recursão (1 ponto) ?

O algoritmo do *Raytracing* simula a geometria ótica envolvida no trajeto dos raios de luz que viajam pelo espaço do cenário (algoritmo de iluminação global), embora o modelo funcione ao contrário do modelo físico, pois neste o raio de luz se origina no objeto e viaja até nossos olhos. Na técnica de *Raytracing*, é suposto que o raio se originou em nossos olhos e atinge o objeto. A inversão não altera a geometria ótica envolvida. Descrevemos abaixo seu funcionamento:

1. Trace um "raio" a partir do observador até a cena a ser representada através de um *pixel* da tela;
2. Determine qual é o primeiro objeto a interceptar esse raio;
3. Calcule a cor da superfície do objeto no ponto de interseção baseado nas características do objeto e na luz;
4. Se a superfície do objeto for reflexiva, calcule um novo raio a partir do ponto de interseção e na "direção de reflexão";
5. Se a superfície do objeto for transparente, calcule um novo raio (refratado) a partir do ponto de interseção.
6. Considere a cor de todos os objetos interceptados pelo raio até sair da cena ou atingir uma fonte de luz, e use esse valor para determinar a cor do *pixel* e se há sombras.



O processo recursivo encontra-se nos passos 4 e 5 uma vez que o algoritmo deve ser repetido para os raios refletidos e refratados e suas contribuições de cor acumuladas no ponto de refração e reflexão.

10) Porque é necessário usar-se um back-buffer e não diretamente o front-buffer no processo de rendering? (1 ponto)

Não se deve gerar a imagem diretamente para o *Front-Buffer* no caso de cenas dinâmicas, por exemplo, uma animação, uma vez que isso pode causar o efeito de *flickering*, devido ao não sincronismo entre a cena, e o conteúdo do buffer que será exibido no dispositivo. Uma idéia é utilizar um buffer adicional denominado *Back-Buffer*. Durante a renderização de um quadro de uma cena, a imagem é construída no *Back-buffer*, que funciona como uma região de rascunho, até que seja completada e então, através de uma chamada a uma rotina de troca de buffers (glSwapBuffers(), por exemplo), o seu conteúdo é transferido para o *Front-Buffer* que, por sua vez, alimenta o dispositivo de exibição com suas informações. Desta forma a imagem da cena é construída de modo consistente no *Back-buffer*, antes que seja copiada para o *Front-Buffer*.