MAC0420/5744 – Introdução à Computação Gráfica
1o Semestre de 2011
Terceiro exercício programa - Ray Tracing

Data de entrega: sábado, 18/06/2011, até 23:55h

Esse exercício pode ser feito por grupos de até 2 alunos de MAC0420 e é individual para os alunos de MAC5744

1. Objetivo

Implementar um programa traçador de raios para obter imagens realistas. A entrada do programa é um arquivo de descrição da cena e a saída é uma imagem no formato PPM. Esse EP não requer o uso do OpenGL, e você tem flexibilidade de alterar um pouco o formato do arquivo de entrada, desde que suporte os seguintes elementos:
Objetos: esferas e triângulos.
Pigmentos: cores sólidas e padrões quadriculados
Modelo de iluminação: ambiente, difusão, especular, atenuação com distância, sombras, reflexão, e refração.
Sugestões de partes opcionais: outros objetos tipo discos circulares, supersampling para anti-aliasing, pigmentos gradientes, interface para visualização usando o GLUT e OpenGL, spotlights, animação, etc.

2. Definição do arquivo de entrada

O arquivo de entrada define, na seguinte ordem:

- as condições de observação;
- as fontes de iluminação;
- uma lista de pigmentos e texturas;
- uma lista dos tipos de acabamento para as superfícies; e
- a descrição dos objetos da cena.

Cada objeto possui além de sua forma geométrica, um pigmento e um tipo de acabamento para a sua superfície. O pigmento pode ser entendido como uma função que associa uma cor básica para cada ponto do objeto, e o acabamento determina como essa cor é modificada de acordo com a luz ambiente, difusa, especular, reflexão e refração.

A entrada foi definida para que seja amigável ao programa e não ao usuário. Todas as coordenadas são fornecidas como valores em (x, y, z) com respeito ao frame de coordenadas padrão. Todas as cores são fornecidas como um vetor RGB de floats no intervalo [0,1].

Linha de comando: A linha de comando define os nomes dos arquivos de entrada, saída, e o tamanho da imagem. Por exemplo:
`ep3 entrada.txt saida.ppm 400 300`
lê um arquivo chamado "entrada.txt"e gera um arquivo chamado "saida.ppm"contendo uma imagem de largura 400 e altura 300 pixels. Você pode utilizar os parâmetros `argc` e `argv` do `main` e a função `atoi` para converter o string "400"para o inteiro 400.

Definição das condições de observação: as 4 primeiras linhas do arquivo de entrada contém informações sobre a câmera e a projeção perspectiva, de forma similar ao **gluLookAt** e **gluPerspective**. Elas definem a posição do olho, o ponto da cena sob observação, o vetor "up"e o campo visual vertical y em graus. Não há nenhum plano de corte "near"ou "far", e você deve assumir que o plano de imagem está localizado a uma unidade de distância do olho e centrado ao redor da direção de observação. O "aspect-ratio"da janela é determinado pela altura e largura da mesma.

Fontes de luz: a linha seguinte define o número de fontes de luz n_L, não mais que 10. As n_L linhas seguinte definem cada uma dessas fontes de luz. Cada linha contém 9 floats: as coordenadas (x, y, z) da fonte de luz, as suas componentes RGB, e os parâmetros (a, b, c) que definem a fórmula de atenuação $a + bd + cd^2$. As fontes de luz são numeradas de 0 a $n_L - 1$. A luz 0 é sempre interpretada como luz ambiente, e embora sua posição e parâmetros de atenuação sejam fornecidos, esses devem ser **ignorados** pelo seu programa.

Pigmentos: a linha seguinte às fontes contém o número de pigmentos n_P, não mais que 20, e é seguida pelas especificações dos n_P pigmentos, numerados de 0 a $n_P - 1$. Cada pigmento pode ser considerado uma função que mapeia um ponto (x,y,z) para uma cor RGB. Os seguintes tipos de pigmentos devem ser considerados:

- **Sólido:** A palavra "SOLI"seguida por um valor RGB. Cada ponto sobre a superfície do objeto tem essa cor.
- **Quadriculado:** Define um padrão quadriculado 3D. É defido pela palavra "QUAD"seguida por dois valores RGB, C_0 e C_1, seguidos por um escalar s que corresponde ao tamanho de cada quadrado do padrão.

  Para determinar a cor de um ponto, divida suas coordenadas (x,y,z) por s e utilize a função `floor()` (definida na biblioteca `math.h`) para cada componente. Se a soma dos três inteiros resultantes for par, então defina a cor do ponto como sendo C_0. Caso contrário, utilize C_1.

Acabamento de superfícies: A linha seguinte define o número de acabamentos n_F, numeradas de 0 a $n_F - 1$. As linhas seguintes definem os n_F tipos de acabamento. Cada acabamento é definido por 7 (sete) números: o coeficiente de ambiente k_a, o coeficiente de difusão k_d, o coeficiente especular k_s, o brilho α, o coeficiente de reflexão k_r, o coeficiente de transmissão k_t e o índice de refração do interior do objeto ior. Você pode assumir que não há intersecção de objetos e que eles estão separados por um vácuo.

Objetos: A linha seguinte define o número de objetosn_O, numeradas de 0 a $n_O - 1$. Cada linha começa com dois inteiros que indicam o pigmento e o acabamento usado na superfície do objeto (um índice que indica n_P e n_F. Esses números são seguidos por:

- Esfera: definida pela palavra "ESFE" seguida pela coordenada (x,y,z) de seu centro e o valor de seu raio r.
- Triângulo: definido pela palavra "TRIA"seguida pelas coordenadas (x,y,z) de seus 3 vértices.

Você pode assumir um máximo de 10 fontes de luz, 50 pigmentos, 50 tipos de acabamento e 200 objetos em uma cena.

3. Saída

A saída do programa deve gerar um arquivo de imagem no formato PPM P6. Certifique-se que ele pode ser lido pelo GIMP.

A primeira linha de um arquivo nesse formato contem o string P6 (seguido por um caracter para pular linha). A segunda linha contem a largura e altura da imagem em pixels, separados por um espaço em branco, e seguidos por um pula-linha. A terceira linha contem o número '255' (que corresponde ao

valor máximo de uma componente de cor de um pixel), seguido por um pula-linha. A seguir você deve armazenar os valores RGB de cada pixel, começando pelo canto superior esquerdo, linha a linha, de cima para baixo, até acabar no canto inferior direito. Para cada componente real de RGB entre [0,1], escale-os para o intervalo [0,255], e armazene esse valor em uma variável do tipo 'unsigned char' para que seja colocada no arquivo de saída.

Para testar, sugiro trabalhar com imagens bem pequenas.

4. Organização geral do programa:

Eu sugiro que você comece seu programa construindo as rotinas para operações básicas com vetores (atribuição, adição, subtração, produtos escalar e vetorial, etc). Se você sabe C++, elas poderiam ser as funções membro da classe vetor. Essa classe poderia ser generalizada para representar também pontos e cores RGB.

Você pode definir um raio como sendo um ponto P mais um vetor u (não se esqueça de normalizá-lo). Você pode também manter na mesma estrutura um valor t_0, para manter a distância ao primeiro objeto. Seria bom também que você criasse um frame de coordenadas, para armazenar toda a informação sobre a posição da câmera, e um método para traçar raios a partir das coordenadas (linha, coluna) de um pixel.

Seu programa deverá ler a linha de comando, decodificá-la, e a seguir carregar o arquivo de entrada contendo a especificação da cena, salvando essa especificação em uma coleção de vetores internos. Não se esqueça de inicializar o seu frame de coordenadas.

Baseado nas coordenadas do olho, do ponto sob observação e da direção do vector up (como fizemos com o simulador de vôo), você pode construir o seu frame de observação. A partir do campo visual vertical y (fov-y) e o aspect ratio, você pode calcular os cantos da janela de observação (Dica: uma simplificação que pode ser útil é escalonar os comprimentos dos vetores x e y do frame de observação para que sejam metade da largura e altura da janela de observação). A partir do tamanho da imagem você pode fazer as interpolações necessárias para determinar os pontos através dos quais cada raio deverá ser traçado.

Seu programa deverá então gerar um raio do olho através do centro de cada pixel da janela de observação. Verifique se esse raio intersecta algum objeto da cena, e salve o objeto mais próximo (com o menor t_0 positivo). Se o raio não interceptar nenhum objeto, então a cor do pixel será a cor default do fundo (eu usei (0.5, 0.5, 0.5)). Caso contrário, calcule o pigmento no ponto de intersecção e a normal àquele ponto. Tenha cuidado, pois a normal depende se o raio atinge a superfície pela frente ou por trás (de dentro do objeto).

A seguir calcule o sombreamento a partir do acabamento do objeto. Primeiro calcule a intensidade ambiente do ponto utilizando a fonte L_0, e a seguir, para cada fonte de luz, determine se o ponto é visível (tome cuidado, pois como o raio deve partir da superfície do objeto, você não vai querer considerar essa mesma superfície como uma possível intersecção. Você pode evitar isso considerando simplesmente que t_0 seja maior que um valor bem pequeno, tipo 0.001). Se a luz for visível, use as equações fornecidas em aula para calcular a intensidade no ponto. Se a componente de refração e/ou reflexão forem não nulas, você deverá traçar mais raios, recursivamente. Você pode assumir um máximo de 4 chamadas recursivas.

5. Dicas finais:

Implemente o seu programa em etapas, de forma que, se você não conseguir terminá-lo, você pode entregar um subconjunto do que foi solicitado. Por exemplo, comece desenhando apenas esferas, com cores sólidas, em imagens pequenas (10x10) e sem refração ou reflexão. A medida que você tiver mais confiança em suas rotinas, escreva as demais.

Uma outra fonte para ajudar você são as mensagens do executável que será colocado a sua disposição em nossa página.

Bom divertimento!