Fundamentos de Computação Gráfica INF 2608 - Trabalho 2 (Rastreamento de Raios)

Rafael Diniz
Lab. Telemídia, PUC-Rio
rafaeldiniz@telemidia.puc-rio.br

8 de julho de 2013

Sumário

1 Descrição do trabalho

Este trabalho apresenta a implementação do algoritmo de rastramento de raios para o desenho de uma cena com esferas. O código foi feito em código C e para o desenho da cena foram utilizados recursos da biblioteca OpenGL e da biblioteca utilitária GLut.

O programa recebe como entrada um arquivo contendo a descrição do elemento iluminante, das esferas e da câmera e tem como saída a cena renderizada na tela.

2 O algoritmo de Rastreamento de Raios

O algoritmo do Rastreamento de Raios, ou ray tracing, é um algoritmo que consiste do lançamento de raios a partir de uma câmera para cada um dos pixels da tela em direção a um conjunto de objetos 3D, levando-se em consideração as faces dos objetos que estão em primeiro na interseção do raio lançado. Assim que o raio bate num ponto de um objeto, a cor desse ponto é calculado levando-se em conta os direntes tipos de reflexão da luz considerados ou mesmo se o objeto está atrás de outro objeto, oque caracteriza uma sombra.

Os tipos de reflexão de luz considerados são os descritos por Phong: ambiente, difusa e especular.

A componente ambiente é constante em toda a cena.

A componente difusa segue a lei de Lambert, que relaciona a intensidade de luz refletida pela superfície ao coseno do ângulo de incidência da luz à normal à superfície do objeto.

E a componente especular é dada por uma fórmula empírica proposta por Phong que relaciona a intensidade especular a uma potência do coseno do ângulo formado pela direção de reflexão da luz (mesmo ângulo formado pelo vetor que liga o objeto à fonte iluminante com relação à normal do objeto, mas na direção do observador) ao vetor que liga o ponto da superfície ao observador.

3 Implementação

Primeiramente é chamada a função parse_input() que lê o arquivo de entrada e armazena os diferentes elemento da cena: a câmera, a qual somente sua coordenada Z é especificada (X = 0 e Y = 0 são assumidos implicitamente), as luzes, relativa às quais são especificas suas coordenadas X, Y e Z e sua cor R,G,B, e finalmente as esferas, para as quais são definidas suas coordenadas X, Y, Z, sua cor R, G, B e seu raio.

Após essa etapa, a algoritmo de ray tracing é executado a partir da função rayTrace(), que executa os seguintes passos:

- Ordena os objetos pelo valor de profundidade dos mesmos
- Pixel a pixel, da esquerda para direita, de cima para baixo, calcula as coordenadas de mundo do pixel e lança um raio a partir da câmera na direção do pixel.

- Para cada fonte iluminante, fazer a interseção do raio lançado com os objetos que estão mais próximos e caso o objeto não esteja encoberto (sombra) é feita a iluminação (função illuminate()) difusa e especular
- Posteriormente faz-se a iluminação ambiente
- O vértices são renderizados utilizando-se as funções do OpenGL

A função callback chamada pelo OpenGL para desenhar a cena é a rayTrace(), onde está localizada a implementação do algoritmo:

```
void rayTrace(void)
{
        struct color_bitmap cbmp;
        struct pixel p;
        float rcolor = 0.0;
        float gcolor = 0.0;
        float bcolor = 0.0;
        order* objects = NULL;
        Vector eye = {0, 0, scene.view_z};
        Ray p_ray;

        glClear(GL_COLOR_BUFFER_BIT);

        /* allocate memory for our local buffer */
        cbmp_init(&cbmp, win_size, win_size);

        /*order the objects*/
        objects = order_objects();

        for(cbmp_start_left(&cbmp, &p);
                !cbmp_at_right_end(&cbmp, &p);
                cbmp_next_col(&p)) {

                for(cbmp_start_top(&cbmp, &p);
                        !cbmp_at_bottom_end(&cbmp, &p);
                        cbmp_next_row(&p)) {

                        /* determine the world coordinate
                         * location of the pixel */
                        Vertex ploc = get_wc_pixel(&world_w, &p,
                        cbmp.width, cbmp.height);

                        /* and create the primary ray */
                        p_ray.origin = eye;
                        p_ray.dir = vector_unit(vector_sub(&ploc, &eye));

                        /* for each object, determine if the
                         * primary ray intersects the object;
                         * keep track of the nearest intersected object */
                        rcolor = 0; gcolor = 0; bcolor = 0;
                        int l;
                        for (l=0; l<scene.no_lights; l++) {
                                color c = perform_intersects(objects, &p_ray,
                                        &scene.lights[0]);
                                /*add ambient lighting*/
```

```
                                rcolor += c.red + ambient_coeff*ambient_inten;
                                gcolor += c.green + ambient_coeff*ambient_inten;
                                bcolor += c.blue + ambient_coeff*ambient_inten;
                        }

                        /* set the color for the pixel in our local buffer */
                        cbmp_set_color(cbmp.red, &p, rcolor);
                        cbmp_set_color(cbmp.green, &p, gcolor);
                        cbmp_set_color(cbmp.blue, &p, bcolor);
                }
        }

        /* Now draw all the pixels */
        glBegin(GL_POINTS);
        for(cbmp_start_left(&cbmp, &p);
                !cbmp_at_right_end(&cbmp, &p);
                cbmp_next_col(&p)) {
                for(cbmp_start_top(&cbmp, &p);
                        !cbmp_at_bottom_end(&cbmp, &p);
                        cbmp_next_row(&p)) {
                        glColor3f(cbmp_get_red(&cbmp, &p),
                                  cbmp_get_green(&cbmp, &p),
                                  cbmp_get_blue(&cbmp, &p));
                        glVertex2i(p.x, p.y);
                }
        }
        cbmp_free(&cbmp);
        glEnd();
        glFlush();
}
```

Outra função relevante do código é a função illuminate:

```
static color illuminate(Light* l, Ray* r, Vertex* p, Vector* n, order* start,
order* current)
{
        Ray ref_ray, shadow_ray;
        float angle, coeff;
        bool light_blocked;
        color res = {0, 0, 0};
        Vertex lightv = {l->x_pos, l->y_pos, l->z_pos};

        shadow_ray.dir = vector_sub(&lightv, p);
        shadow_ray.dir = vector_unit(shadow_ray.dir);
        shadow_ray.origin = *p;
        /*diffuse lighting*/
        /*check if the light ray is blocked*/
        light_blocked = ray_is_blocked(start, &shadow_ray, current);
        if (!light_blocked) {
                angle = vector_dot(n, &shadow_ray.dir);
                coeff = diffuse_coeff * angle;
                res.red = l->red * coeff;
                res.green = l->green * coeff;
```

```
                res.blue = l->blue * coeff;
        }

        /*specular lighting*/
        ref_ray.dir = reflected_vector(&shadow_ray, n);
        ref_ray.origin = *p;
        if (!light_blocked) {
                angle = vector_dot(&r->dir, &ref_ray.dir);
                coeff = absorb_coeff * pow(angle, shine);
                res.red += l->red * coeff;
                res.green += l->green * coeff;
                res.blue += l->blue * coeff;
        }

        return res;
}
```

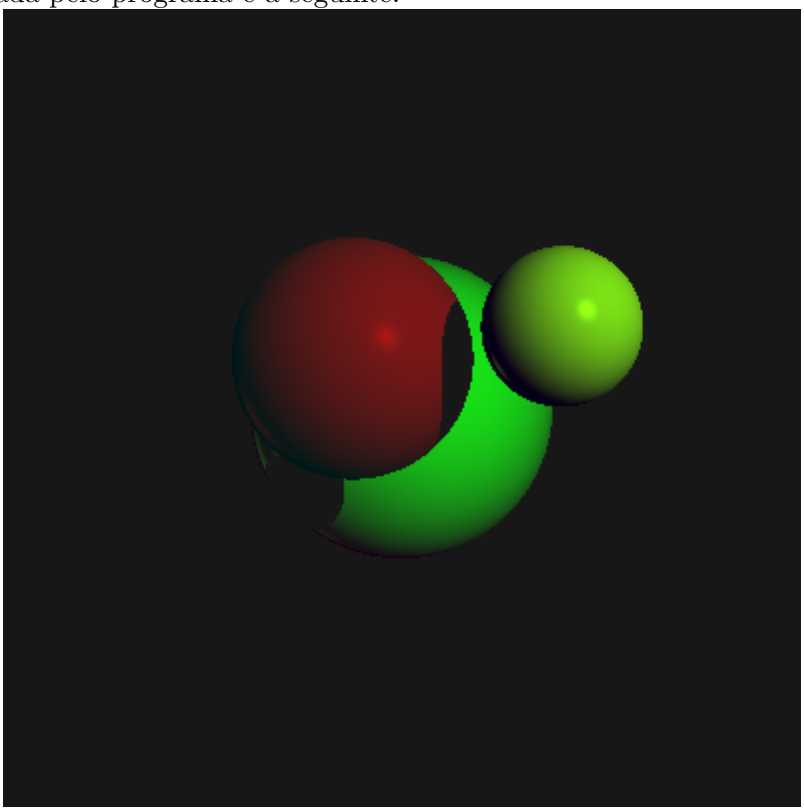
4 Resultados obtidos

Uma cena escolhida como exemplo é a seguinte:

```
VIEW 3
LIGHT 3 2 3 0.5 1.0 0.5
SPHERE 0 0 -5 1.0 0.0 1.0 0.0
SPHERE -0.2 0.2 -2 0.5 1.0 0.0 0.0
SPHERE 0.6 0.3 -1.5 0.3 1.0 1.0 0.0
```

Oque nos indica que a câmera tem posição (0,0,3). Iluminante na posição (3,2,3) e cor (0.5,1.0,0.5) e as esferas nas posições (0,0,-5), (-0.2,0.2,-2) e (0.6,0.3,-1.5) com as cores (0.0,1.0,0.0), (1.0,0.0,0.0) e (1.0,1.0,0.0), e raios 1.0, 0.5 e 0.3.

A tela renderizada pelo programa é a seguinte:



Vemos que o resultado obtido está dentro do esperado, com as reflexões especular e difusa claramente visíveis, as sombras evidentes, e a hierarquia de profundidade dos objetos respeitada.

5 Código Fonte

O código fonte do trabalho deverá estar anexo, sendo que a lista dos arquivos em código C é:

- color_bitmap.c: código C. Contém a implementação das funções auxiliares para tratar com o bitmap.
- color_bitmap.h: código C. Cabeçalho do color_bitmap.c.
- primitives.c: código C. Contém a tranformada para Coordenadas do Mundo (World Coordinates), e operações com vetores.
- primitives.h: código C. Cabeçalho do primitives.c.
- ray_trace.c: códido C. Contém todas as operações do algoritmo de ray tracing.
- ray_trace.h: código C. Cabeçalho do ray_trace.c.
- read_input.c: código C. Contém a função de leitura dos objetos 3D da cena a partir do arquivo de entrada com as descrições.
- sphere.c: código C. Contém as funções para calcular a interseção e normal da esfera.
- sphere.h: código C. Cabeçalho do sphere.c.
- Makefile: arquivo Makefile ajustado pra utilizar o compilador GCC. Deve ter suas variáveis CC e LDFLAGS alteradas em caso de outro compilador.

6 Instruções de compilação e execução

```
Para compilar:
$ make

Para executar:
$ ./t2 cena.txt
```