Solução com threads

Dividimos a imagem em oito colunas iguais para cada thread trabalhar em uma coluna.

# Adições de código

## Inicializações

Primeiro foi incluída a biblioteca pthread:

#include <pthread.h>

Após isso foi definido o número de threads que seriam criados:

#define NUM\_THREADS 8

Também foi necessário criar uma struct para a passagem de argumentos usados na função que o thread usa:

struct thread\_data {

int thread\_id;

int inicio;

int fim;

int samples;

int s;

float rcp\_samples;

uchar \*image;

camera c;

}typedef thread\_data;

Declaração da função que cria cada pixel, ela recebe a struct que possui os argumentos:

void \*doWork(void \*threadarg);

Declaração de um array global que contém as informações dos threads:

struct thread\_data thread\_data\_array[NUM\_THREADS];

## Main

Dentro da main, precisamos declarar estas variáveis que serão usadas na criação e no join dos threads:

pthread\_t threads[NUM\_THREADS];

pthread\_attr\_t attr;

long t;

int rc;

void \*status;

A seguir definimos que os threads aceitam join:

// Inicia o objeto attr com os valores padrões

pthread\_attr\_init(&attr);

pthread\_attr\_setdetachstate(&attr, PTHREAD\_CREATE\_JOINABLE);

### Criação dos threads

Dentro do for inicializamos a struct com os argumentos que cada thread usará, dividindo a criação da imagem em oito colunas.

Em seguida, criamos o thread.

for(t = 0; t < NUM\_THREADS; t++) {

printf("Main: criando thread %ld\n", t);

// Inicialização da struct

thread\_data\_array[t].thread\_id = t;

thread\_data\_array[t].inicio = t \* (c.view.width/NUM\_THREADS);

thread\_data\_array[t].fim = thread\_data\_array[t].inicio + c.view.width/NUM\_THREADS;

thread\_data\_array[t].samples = samples;

thread\_data\_array[t].s = s;

thread\_data\_array[t].rcp\_samples = rcp\_samples;

thread\_data\_array[t].image = image;

thread\_data\_array[t].c = c;

// Criação do thread

rc = pthread\_create(&threads[t], &attr, doWork, (void \*) &thread\_data\_array[t]);

if (rc) {

printf("ERROR; return code from pthread\_create() is %d\n", rc);

exit(-1);

}

}

### Join dos threads

Destruição do objeto attr:

pthread\_attr\_destroy(&attr);

Dentro do for fazemos o join dos threads:

for(t = 0; t < NUM\_THREADS; t++) {

rc = pthread\_join(threads[t], &status);

if (rc) {

printf("ERROR; return code from pthread\_join() is %d\n", rc);

exit(-1);

}

printf("Main: join com thread %ld completo. Status: %ld\n",t,(long)status);

}

## Função que o thread executa

void \*doWork(void \*threadarg) {

// Inicialização das variáveis

struct thread\_data \*my\_data;

my\_data = (struct thread\_data \*) threadarg;

long taskid = my\_data->thread\_id;

int inicio = my\_data->inicio;

int fim = my\_data->fim;

int samples = my\_data->samples;

int s = my\_data->s;

float rcp\_samples = my\_data->rcp\_samples;

uchar \*image = my\_data->image;

camera c = my\_data->c;

for(inicio ; inicio < fim ; inicio++) {

for(int j = 0 ; j < c.view.height; j++) {

float r, g, b;

r = g = b = 0.0;

for(s=0; s<samples; s++) {

ray rr = get\_primary\_ray(&c,inicio,j,s);

color col = trace(c,&rr,0);

r += col.r;

g += col.g;

b += col.b;

}

r = r \* rcp\_samples;

g = g \* rcp\_samples;

b = b \* rcp\_samples;

//red green blue color components

image[ 3\* (inicio \* c.view.height + j) + 0] = floatToIntColor(r);

image[ 3\* (inicio \* c.view.height + j) + 1] = floatToIntColor(g);

image[ 3\* (inicio \* c.view.height + j) + 2] = floatToIntColor(b);

}

}

printf("Sou o thread %ld!\n", taskid);

// Termina o thread

pthread\_exit((void\*) taskid);

}