Project 4: Problem 1

20204898 박소은

# Execution environment

* Colab: GPU T4

# How to compile

## cuda\_ray.cu

!nvcc cuda\_ray.cu -o cuda\_ray

## openmp\_ray.cpp

g++ -fopenmp openmp\_ray.cpp -o openmp\_ray

# How to execute

## cuda\_ray.cu

!./cuda\_ray

## openmp\_ray.cpp

./openmp\_ray.exe [num\_thread] result.ppm

# Entire source code

## cuda\_ray.cu

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <time.h>  #include <math.h>  #define CUDA 0  #define OPENMP 1  #define SPHERES 20  #define rnd( x ) (x \* rand() / RAND\_MAX)  #define INF 2e10f  #define DIM 2048  // Indicates 3D sphere shape: sphere location, radius, and color information  struct Sphere {      float   r,b,g;      float   radius;      float   x,y,z;  };  // kernel function  \_\_global\_\_ void CUDA\_kernel(Sphere\* s, unsigned char\* ptr) {    int x = threadIdx.x + blockIdx.x \* blockDim.x;    int y = threadIdx.y + blockIdx.y \* blockDim.y;    int offset = x + y\*DIM;    float ox = (x - DIM/2);    float oy = (y - DIM/2);    float r=0, g=0, b=0;    float   maxz = -INF;    for(int i=0; i<SPHERES; i++) {      float dx = ox - s[i].x;      float dy = oy - s[i].y;      float t, n;      // hit() function      if (dx \* dx + dy \* dy < s[i].radius \* s[i].radius) {        float dz = sqrtf(s[i].radius \* s[i].radius - dx \* dx - dy \* dy);        n = dz / sqrtf(s[i].radius \* s[i].radius);        t = dz + s[i].z;      } else {        t = -INF;      }      if (t > maxz) {        float fscale = n;        r = s[i].r \* fscale;        g = s[i].g \* fscale;        b = s[i].b \* fscale;        maxz = t;      }    }    ptr[offset\*4 + 0] = (int)(r \* 255);    ptr[offset\*4 + 1] = (int)(g \* 255);    ptr[offset\*4 + 2] = (int)(b \* 255);    ptr[offset\*4 + 3] = 255;  }  // Function to store images in PPM file format  void ppm\_write(unsigned char\* bitmap, int xdim,int ydim, FILE\* fp)  {    int i,x,y;    fprintf(fp,"P3\n");    fprintf(fp,"%d %d\n",xdim, ydim);    fprintf(fp,"255\n");    for (y=0;y<ydim;y++) {      for (x=0;x<xdim;x++) {        i=x+y\*xdim;        fprintf(fp,"%d %d %d ",bitmap[4\*i],bitmap[4\*i+1],bitmap[4\*i+2]);      }      fprintf(fp,"\n");    }    printf("[result.ppm] was generated. \n");  }  int main(int argc, char\* argv[])  {    srand(time(NULL));    FILE\* fp = fopen("result.ppm", "w");    // temp\_s: Sphere used by the CPU    Sphere \*temp\_s = (Sphere\*)malloc( sizeof(Sphere) \* SPHERES );    for (int i=0; i<SPHERES; i++) {      temp\_s[i].r = rnd( 1.0f );      temp\_s[i].g = rnd( 1.0f );      temp\_s[i].b = rnd( 1.0f );      temp\_s[i].x = rnd( 2000.0f ) - 1000;      temp\_s[i].y = rnd( 2000.0f ) - 1000;      temp\_s[i].z = rnd( 2000.0f ) - 1000;      temp\_s[i].radius = rnd( 200.0f ) + 40;    }    // cuda\_s: Sphere used by the GPU    Sphere \*cuda\_s;    cudaMalloc((void\*\*)&cuda\_s, sizeof(Sphere) \* SPHERES);    cudaMemcpy(cuda\_s, temp\_s, sizeof(Sphere) \* SPHERES, cudaMemcpyHostToDevice);    // bitmap: Bitmap used by CPU    unsigned char\* bitmap;    bitmap = (unsigned char\*)malloc(sizeof(unsigned char) \* DIM\*DIM\*4);    // cuda\_bitmap: Bitmap used by GPU    unsigned char\* cuda\_bitmap;    cudaMalloc((void\*\*)&cuda\_bitmap, sizeof(unsigned char) \*DIM\*DIM\*4);    cudaMemcpy(cuda\_bitmap, bitmap, sizeof(unsigned char)\*DIM\*DIM\*4, cudaMemcpyHostToDevice);  // Execution configuration    dim3 gridDims(DIM / 16, DIM / 16);    dim3 blockDims(16, 16);    clock\_t start = clock();    CUDA\_kernel<<<gridDims, blockDims>>>(cuda\_s, cuda\_bitmap);    clock\_t end = clock();    cudaDeviceSynchronize(); // Wait until GPU ends    cudaMemcpy(bitmap, cuda\_bitmap, sizeof(unsigned char)\*DIM\*DIM\*4, cudaMemcpyDeviceToHost); // Copy the result      clock\_t exe\_time = end - start;    double exe\_time\_ms = ((double)exe\_time / CLOCKS\_PER\_SEC) \* 1000.0;    printf("CUDA ray tracing: %f ms \n", exe\_time\_ms);    ppm\_write(bitmap,DIM,DIM,fp); // write ppm file    fclose(fp);    free(bitmap);    free(temp\_s);    free(cuda\_bitmap);    free(cuda\_s);    return 0;  } |

## openmp\_ray.cpp

|  |
| --- |
| #include <stdio.h>  #include <string.h>  #include <stdlib.h>  #include <time.h>  #include <math.h>  #include <omp.h>  #pragma warnings(disable: 4996)  #define \_CRT\_SECURE\_NO\_WARNINGS  #define CUDA 0  #define OPENMP 1  #define SPHERES 20  #define rnd( x ) (x \* rand() / RAND\_MAX)  #define INF 2e10f  #define DIM 2048  struct Sphere {  float r, b, g;  float radius;  float x, y, z;  float hit(float ox, float oy, float\* n) {  float dx = ox - x;  float dy = oy - y;  if (dx \* dx + dy \* dy < radius \* radius) {  float dz = sqrtf(radius \* radius - dx \* dx - dy \* dy);  \*n = dz / sqrtf(radius \* radius);  return dz + z;  }  return -INF;  }  };  void kernel(int x, int y, Sphere\* s, unsigned char\* ptr)  {  int offset = x + y \* DIM;  float ox = (x - DIM / 2);  float oy = (y - DIM / 2);  float r = 0, g = 0, b = 0;  float maxz = -INF;  for (int i = 0; i < SPHERES; i++) {  float n;  float t = s[i].hit(ox, oy, &n);  if (t > maxz) {  float fscale = n;  r = s[i].r \* fscale;  g = s[i].g \* fscale;  b = s[i].b \* fscale;  maxz = t;  }  }  ptr[offset \* 4 + 0] = (int)(r \* 255);  ptr[offset \* 4 + 1] = (int)(g \* 255);  ptr[offset \* 4 + 2] = (int)(b \* 255);  ptr[offset \* 4 + 3] = 255;  }  void ppm\_write(unsigned char\* bitmap, int xdim, int ydim, FILE\* fp)  {  int i, x, y;  fprintf(fp, "P3\n");  fprintf(fp, "%d %d\n", xdim, ydim);  fprintf(fp, "255\n");  for (y = 0; y < ydim; y++) {  for (x = 0; x < xdim; x++) {  i = x + y \* xdim;  fprintf(fp, "%d %d %d ", bitmap[4 \* i], bitmap[4 \* i + 1], bitmap[4 \* i + 2]);  }  fprintf(fp, "\n");  }  printf("[result.ppm] was generated. \n");  }  int main(int argc, char\* argv[])  {  int no\_threads, x, y;  srand(time(NULL));  if (argc != 2) {  printf("> a.out [threadNum]\n");  printf("[threadNum] 1~16: OpenMP using 1~16 threads\n");  exit(0);  }  FILE\* fp = fopen("result.ppm", "w");  no\_threads = atoi(argv[1]);  Sphere\* temp\_s = (Sphere\*)malloc(sizeof(Sphere) \* SPHERES);  for (int i = 0; i < SPHERES; i++) {  temp\_s[i].r = rnd(1.0f);  temp\_s[i].g = rnd(1.0f);  temp\_s[i].b = rnd(1.0f);  temp\_s[i].x = rnd(2000.0f) - 1000;  temp\_s[i].y = rnd(2000.0f) - 1000;  temp\_s[i].z = rnd(2000.0f) - 1000;  temp\_s[i].radius = rnd(200.0f) + 40;  }  unsigned char\* bitmap;  bitmap = (unsigned char\*)malloc(sizeof(unsigned char) \* DIM \* DIM \* 4);  // Set the number of threads  omp\_set\_num\_threads(no\_threads);  clock\_t start = clock();  #pragma omp parallel for default(shared) private(x, y)  for (x = 0; x < DIM; x++) {  for (y = 0; y < DIM; y++) {  kernel(x, y, temp\_s, bitmap);  }  }  clock\_t end = clock();  clock\_t exe\_time = end - start;  double exe\_time\_ms = ((double)exe\_time / CLOCKS\_PER\_SEC) \* 1000.0;  printf("OpenMP (%d threads) ray tracing: %f ms \n", no\_threads, exe\_time\_ms);  ppm\_write(bitmap, DIM, DIM, fp);  fclose(fp);  free(bitmap);  free(temp\_s);  return 0;  } |

# Program output results, Ray-tracing result pictures

## cuda\_ray.cu

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

다채로움, 스크린샷, 구체, 블러이(가) 표시된 사진

자동 생성된 설명

## openmp\_ray.c

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

# Experimental results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Execute 1** | **Execute 2** | **Execute 3** | **Average(ms)** |
| **Original code** | 1547 | 1835 | 1424 | **1602** |
| **OpenMP(1 thread)** | 2140 | 1816 | 2419 | **2125** |
| **OpenMP(2 threads)** | 1476 | 1652 | 1262 | **1463.33333** |
| **OpenMP(4 threads)** | 1086 | 789 | 1009 | **961.333333** |
| **OpenMP(8 threads)** | 699 | 587 | 633 | **639.666667** |
| **OpenMP(12 threads)** | 599 | 542 | 549 | **563.333333** |
| **OpenMP(16 threads)** | 542 | 555 | 530 | **542.333333** |
| **CUDA** | 0.037 | 0.022 | 0.02 | **0.02633333** |

## Screen captures of output result.

* Original code

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

* CUDA

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

* OpenMP

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명