Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Курсовой проект по курсу «Программирование графических процессоров»

Обратная трассировка лучей (Ray Tracing). Технологии MPI, CUDA и OpenMP.

Студент: Л. Я. Вельтман
Преподаватель: К. Г. Крашенинников
А. Ю. Морозов
Группа: М8О-407Б
Дата:
Оценка:
Подпись:

Москва, 2021

Условие

Цель работы: Совместное использование технологии MPI, технологии CUDA и технологии OpenMP для создание фотореалистической визуализации. Создание видеоролика/анимации.

Вариант 8:
На сцене должны располагаться три тела: Гексаэдр, Октаэдр, Икосаэдр.
Программа должна поддерживать следующие ключи запуска:
−−cpu Для расчетов используется центральный процессор (OpenMP)
−−gpu Для расчетов задействуется видеокарта (CUDA)
−−default В stdout выводится конфигурация входных данных (в формате описанном ранее), при которой получается наиболее красочный результат, после чего программа завершает свою работу.
Запуск программы без аргументов подразумевает запуск с ключом −−gpu.
Подразумевается, что запуск всегда осуществляется на кластере (т.е. всегда используется технология MPI). Учесть возможность наличия нескольких GPU в рамках одной машины кластера.
Входные данные:

1. Количество кадров.
2. Путь к выходным изображениям. В строке содержится спецификатор %d, на место которого должен подставляться номер кадра. Формат изображений соответствует формату описанному в лабораторной работе 2.
3. Разрешение кадра и угол обзора в градусах по горизонтали.
4. Параметры движения камеры $r_c^0, z_c^0, \varphi_c^0, A_c^r, A_c^z, \omega_c^r, \omega_c^z, \omega_c^\varphi, \rho_c^r, \rho_c^z$, и $r_n^0, z_n^0, \varphi_n^0, A_n^r, A_n^z, \omega_n^r, \omega_n^z, \omega_n^\varphi, \rho_n^r, \rho_n^z$.
5. Параметры тел: центр тела, цвет (нормированный), радиус (подразумевается радиус сферы в которую можно было бы вписать тело), коэффициент отражения, коэффициент прозрачности, количество точечных источников света на ребре.
6. Параметры пола: четыре точки, путь к текстуре, оттенок цвета и коэффициент отражения.
7. Количество (не более четырех) и параметры источников света: положение и цвет.

Выходные данные: В процессе работы программа должна выводить в stdout статистику в формате: номер кадра, время на обработку кадра в миллисекундах.

Программное и аппаратное обеспечение:
Device Number: 0
Device name: GeForce GT 545
TotalGlobalMem: 3150381056
Const Mem : 65536
Max shared mem for blocks 49152
Max regs per block 32768
Max thread per block 1024
multiProcessorCount : 3
maxThreadsDim 1024 1024 64
maxGridSize 65535 65535 65535
OS: macOS Catalina version 10.15.5
Text Editor: Sublime Text 3

1 Описание

Метод решения

Для начала строим сцену: задаем координаты вершин тел и добавляем грани в массив сцены, строим из полигонов пол сцены. Далее нужно настроить камеру. Для этого нужно посчитать базис, связанный с камерой, в который нам нужно перейти, посчитать векторы, которые находятся в плоскости Oxy - это наши лучики и переносим построенные лучи в базис с камерой. Затем рендерим пиксель, откуда выпущен луч. Это делается с помощью ray tracing. Главной задачей алгоритма обратной трассировки лучей является нахождение пересечения треугольника сцены и луча. Нужно найти не просто все пересечения, а именно ближайшее к нам пересечение. Для этого мы перебираем все полигоны сцены и ищем пересечение, используя для расчетов оптимизированный вариант формул, представленных ниже.

Пересечение луча и треугольника

• Оптимизированный вариант

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{dot\,(P, E1)} * \begin{bmatrix} dot\,(Q, E2) \\ dot\,(P, T) \\ dot\,(Q, D) \end{bmatrix}$$

$$E1 = v1 - v0$$

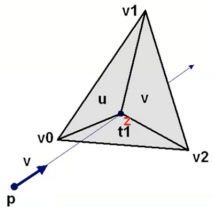
$$E2 = v2 - v0$$

$$T = p - v0$$

$$P = cross\,(D, E2)$$

$$Q = cross\,(T, E1)$$

$$D = v$$



2011

Если пересечение нашлось, мы знаем цвет пикселя, далее применим упрощенные модели глобального освещения, а именно фоновое и диффузное освещение. Вычисляем интенсивность диффузной составляющей как скалярное произведение вектора нормали (вектор, перпендикулярный освещаемой поверхности) и направленного луча света (вектор направления, который является разностью между позицией источника света и позицией точки пересечения луча с ближайшим треугольником сцены). Затем, это значение умножается на цвет источника света, и в результате мы получим компоненту диффузного освещения, которая будет становиться темнее с ростом угла между векторами. Теперь, когда у нас есть фоновый и диффузный компоненты (используем константный коэффициент фонового освещения), мы суммируем их цвета, а затем умножаем результат на цвет объекта, получая таким образом результирующий цвет выходного фрагмента. Если пересечения нет, то просто черный пиксель.

Для выполнения задания курсового проекта в функции main после считывания исходных данных, выделения памяти под массив треугольников на cpu/gpu (в зависимости от ключа) блок кода, отвечающий за рендеринг будет выполняться

каждым процессом параллельно (один процесс - один кадр).

В функции Render происходит рендер на CPU, здесь мы применяем распараллеливание с помощью технологии OpenMP. С помощью директивы #pragma omp parallel создастся параллельный регион для следующего за ней структурированного бока. Директива parallel указывает, что структурированный блок кода должен быть выполнен параллельно в несколько потоков. Таким образом, для каждого из процессов участок кода, отвечающий за обновление значений будет выполняться в многопоточном режиме. Вместо двух циклов теперь реализован один цикл и обновляем итерационных значения i, j в зависимости от k, отвечающей за номер нити.

2 Исходный код

```
// ============================================
// Veltman Lina group 407
// ============================================

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <cuda.h>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>
#include <chrono>
#include <string.h>
#include "mpi.h"
#include <omp.h>

using namespace std::chrono;


#define CSC(call) do { \
  cudaError_t res = call; \
  if (res != cudaSuccess) { \
    fprintf(stderr, "CUDA Error in %s:%d: %s\n", __FILE__, __LINE__, cudaGetErrorString
        (res)); \
    fflush(stderr); \
    exit(0); \
  } \
} while (0) \



typedef unsigned char uchar;

struct vec3
{
  double x;
  double y;
  double z;
};


struct Triangle
{
  vec3 a;
  vec3 b;
  vec3 c;
  uchar4 color;
};
```

```
47
48
49  __device__ __host__
50  double dot(vec3 a, vec3 b)
51  {
52    return a.x * b.x + a.y * b.y + a.z * b.z;
53  }
54
55
56  __device__ __host__
57  vec3 mulc(vec3 a, double c)
58  {
59    return { c * a.x, c * a.y, c * a.z };
60  }
61
62
63  __device__ __host__
64  vec3 prod(vec3 a, vec3 b)
65  {
66    return { a.y * b.z - a.z * b.y, a.z * b.x - a.x * b.z, a.x * b.y - a.y * b.x };
67  }
68
69
70  __device__ __host__
71  vec3 norm(vec3 v)
72  {
73    double l = sqrt(dot(v, v));
74    return { v.x / l, v.y / l, v.z / l };
75  }
76
77
78  __device__ __host__
79  double len(vec3 v)
80  {
81    return sqrt(dot(v, v));
82  }
83
84
85  __device__ __host__
86  vec3 diff(vec3 a, vec3 b)
87  {
88    return { a.x - b.x, a.y - b.y, a.z - b.z };
89  }
90
91
92  __device__ __host__
93  vec3 add(vec3 a, vec3 b)
94  {
95    return { a.x + b.x, a.y + b.y, a.z + b.z };
```

```
96  }
97
98  __device__ __host__
99  vec3 mult(vec3 a, vec3 b, vec3 c, vec3 v)
100 {
101   return {
102         a.x * v.x + b.x * v.y + c.x * v.z,
103         a.y * v.x + b.y * v.y + c.y * v.z,
104         a.z * v.x + b.z * v.y + c.z * v.z
105       };
106 }
107
108
109 void print(vec3 v)
110 {
111   printf("%e %e %e\n", v.x, v.y, v.z);
112 }
113
114
115 __host__ __device__
116 double dmin(double x, double y)
117 {
118   if (x > y) { return y; }
119   return x;
120 }
121
122
123 void BuildStage(Triangle* t, double r1, vec3 o1, uchar4 c1,
124         double r2, vec3 o2, uchar4 c2,
125         double r3, vec3 o3, uchar4 c3,
126         vec3* fv, uchar4 fc)
127 {
128   int st = 0;
129   double p = (1 + sqrt(5)) / 2;
130
131   vec3 icosVertexes[] = {
132     add(o1, norm(vec3{ 0, -1, p})),
133     add(o1, norm(vec3{ 0, 1, p})),
134     add(o1, norm(vec3{-p, 0, 1})),
135     add(o1, norm(vec3{ p, 0, 1})),
136     add(o1, norm(vec3{-1, p, 0})),
137     add(o1, norm(vec3{ 1, p, 0})),
138     add(o1, norm(vec3{ 1, -p, 0})),
139     add(o1, norm(vec3{-1, -p, 0})),
140     add(o1, norm(vec3{-p, 0, -1})),
141     add(o1, norm(vec3{ p, 0, -1})),
142     add(o1, norm(vec3{ 0, -1, -p})),
143     add(o1, norm(vec3{ 0, 1, -p}))
144   };
```

```
145
146   t[st++] = Triangle{ icosVertexes[0], icosVertexes[1], icosVertexes[2] , c1 };
147   t[st++] = Triangle{ icosVertexes[1], icosVertexes[0], icosVertexes[3] , c1 };
148   t[st++] = Triangle{ icosVertexes[0], icosVertexes[2], icosVertexes[7] , c1 };
149   t[st++] = Triangle{ icosVertexes[2], icosVertexes[1], icosVertexes[4] , c1 };
150   t[st++] = Triangle{ icosVertexes[4], icosVertexes[1], icosVertexes[5] , c1 };
151   t[st++] = Triangle{ icosVertexes[6], icosVertexes[0], icosVertexes[7] , c1 };
152   t[st++] = Triangle{ icosVertexes[3], icosVertexes[0], icosVertexes[6] , c1 };
153   t[st++] = Triangle{ icosVertexes[1], icosVertexes[3], icosVertexes[5] , c1 };
154   t[st++] = Triangle{ icosVertexes[4], icosVertexes[5], icosVertexes[11], c1 };
155   t[st++] = Triangle{ icosVertexes[6], icosVertexes[7], icosVertexes[10], c1 };
156   t[st++] = Triangle{ icosVertexes[3], icosVertexes[6], icosVertexes[9] , c1 };
157   t[st++] = Triangle{ icosVertexes[5], icosVertexes[3], icosVertexes[9] , c1 };
158   t[st++] = Triangle{ icosVertexes[7], icosVertexes[2], icosVertexes[8] , c1 };
159   t[st++] = Triangle{ icosVertexes[2], icosVertexes[4], icosVertexes[8] , c1 };
160   t[st++] = Triangle{ icosVertexes[9], icosVertexes[10], icosVertexes[11], c1 };
161   t[st++] = Triangle{ icosVertexes[10], icosVertexes[8], icosVertexes[11], c1 };
162   t[st++] = Triangle{ icosVertexes[5], icosVertexes[9], icosVertexes[11], c1 };
163   t[st++] = Triangle{ icosVertexes[9], icosVertexes[6], icosVertexes[10], c1 };
164   t[st++] = Triangle{ icosVertexes[7], icosVertexes[8], icosVertexes[10], c1 };
165   t[st++] = Triangle{ icosVertexes[8], icosVertexes[4], icosVertexes[11], c1 };
166
167   vec3 hexVertexes[] = {
168     add(o2, norm(vec3{-1, -1, -1})),
169     add(o2, norm(vec3{-1, 1, -1})),
170     add(o2, norm(vec3{ 1, -1, -1})),
171     add(o2, norm(vec3{ 1, 1, -1})),
172     add(o2, norm(vec3{-1, -1, 1})),
173     add(o2, norm(vec3{-1, 1, 1})),
174     add(o2, norm(vec3{ 1, -1, 1})),
175     add(o2, norm(vec3{ 1, 1, 1})),
176   };
177
178   t[st++] = Triangle{ hexVertexes[5], hexVertexes[4], hexVertexes[7], c2 };
179   t[st++] = Triangle{ hexVertexes[4], hexVertexes[6], hexVertexes[7], c2 };
180   t[st++] = Triangle{ hexVertexes[0], hexVertexes[1], hexVertexes[3], c2 };
181   t[st++] = Triangle{ hexVertexes[2], hexVertexes[0], hexVertexes[3], c2 };
182   t[st++] = Triangle{ hexVertexes[3], hexVertexes[1], hexVertexes[5], c2 };
183   t[st++] = Triangle{ hexVertexes[3], hexVertexes[5], hexVertexes[7], c2 };
184   t[st++] = Triangle{ hexVertexes[0], hexVertexes[2], hexVertexes[4], c2 };
185   t[st++] = Triangle{ hexVertexes[4], hexVertexes[2], hexVertexes[6], c2 };
186   t[st++] = Triangle{ hexVertexes[1], hexVertexes[0], hexVertexes[5], c2 };
187   t[st++] = Triangle{ hexVertexes[0], hexVertexes[4], hexVertexes[5], c2 };
188   t[st++] = Triangle{ hexVertexes[2], hexVertexes[3], hexVertexes[7], c2 };
189   t[st++] = Triangle{ hexVertexes[6], hexVertexes[2], hexVertexes[7], c2 };
190
191   vec3 octVertexes[] = {
192     add(o3, norm(vec3{ 0, 0, -1})),
193     add(o3, norm(vec3{ 0, 0, 1})),
```

```
    add(o3, norm(vec3{-1, 0, 0})),
    add(o3, norm(vec3{ 1, 0, 0})),
    add(o3, norm(vec3{ 0, -1, 0})),
    add(o3, norm(vec3{ 0, 1, 0})),
  };

  t[st++] = Triangle{ octVertexes[0], octVertexes[4], octVertexes[2], c3 };
  t[st++] = Triangle{ octVertexes[0], octVertexes[2], octVertexes[5], c3 };
  t[st++] = Triangle{ octVertexes[0], octVertexes[5], octVertexes[3], c3 };
  t[st++] = Triangle{ octVertexes[0], octVertexes[3], octVertexes[4], c3 };
  t[st++] = Triangle{ octVertexes[1], octVertexes[2], octVertexes[4], c3 };
  t[st++] = Triangle{ octVertexes[1], octVertexes[5], octVertexes[2], c3 };
  t[st++] = Triangle{ octVertexes[1], octVertexes[3], octVertexes[5], c3 };
  t[st++] = Triangle{ octVertexes[1], octVertexes[4], octVertexes[3], c3 };

  t[st++] = Triangle{ fv[0], fv[2], fv[1], fc };
  t[st++] = Triangle{ fv[1], fv[2], fv[3], fc };
}


__host__ __device__
void RayTracing(Triangle* triangles, vec3 pos, vec3 dir, int* i, double* t)
{
  int k, k_min = -1;
  double ts_min = 0;
  for (k = 0; k < 42; ++k)
  {
    vec3 e1 = diff(triangles[k].b, triangles[k].a);
    vec3 e2 = diff(triangles[k].c, triangles[k].a);
    vec3 p = prod(dir, e2);
    double div = dot(p, e1);
    if (fabs(div) < 1e-10)
      continue;
    vec3 t = diff(pos, triangles[k].a);
    double u = dot(p, t) / div;
    if (u < 0.0 || u > 1.0)
      continue;
    vec3 q = prod(t, e1);
    double v = dot(q, dir) / div;
    if (v < 0.0 || v + u > 1.0)
      continue;
    double ts = dot(q, e2) / div;
    if (ts < 0.0)
      continue;
    if (k_min == -1 || ts < ts_min)
    {
      k_min = k;
      ts_min = ts;
    }
```

```
243   }
244   *i = k_min;
245   *t = ts_min;
246 }
247
248
249 void Render(Triangle* triangles, vec3 pc, vec3 pv,
250       int w, int h, double angle, uchar4* data,
251       vec3 lightPosition, uchar4 lightColor)
252 {
253   double dw = 2.0 / (w - 1.0);
254   double dh = 2.0 / (h - 1.0);
255   double z = 1.0 / tan(angle * M_PI / 360.0);
256
257   vec3 bz = norm(diff(pv, pc));
258   vec3 bx = norm(prod(bz, { 0.0, 0.0, 1.0 }));
259   vec3 by = norm(prod(bx, bz));
260
261   int size = w * h;
262
263   int kmin;
264   double tmin;
265
266   #pragma omp parallel
267   {
268     int threadQuantity = omp_get_num_threads();
269     int threadId = omp_get_thread_num();
270
271     for (int k = threadId; k < size; k += threadQuantity)
272     {
273       int i = k % w;
274       int j = k / w;
275
276       vec3 v = { -1.0 + dw * i, (-1.0 + dh * j) * h / w, z };
277
278       vec3 dir = norm(mult(bx, by, bz, v));
279
280       RayTracing(triangles, pc, dir, &kmin, &tmin);
281       if (kmin != -1)
282       {
283         double rr = (double)triangles[kmin].color.x / 255.0;
284         double gg = (double)triangles[kmin].color.y / 255.0;
285         double bb = (double)triangles[kmin].color.z / 255.0;
286         double ri = 0.2, gi = 0.2, bi = 0.2;
287
288         vec3 p = add(pc, mulc(dir, tmin));
289         vec3 l = diff(lightPosition, p);
290         vec3 n = prod(diff(triangles[kmin].b, triangles[kmin].a),
291                   diff(triangles[kmin].c, triangles[kmin].a));
```

```
292         double dot_nl = dot(n, l);
293 
294         if (dot_nl > 0)
295         {
296           ri += (lightColor.x / 255.0) * dot_nl / (len(n) * len(l));
297           gi += (lightColor.y / 255.0) * dot_nl / (len(n) * len(l));
298           bi += (lightColor.z / 255.0) * dot_nl / (len(n) * len(l));
299         }
300         data[(h - 1 - j) * w + i].x = (uchar)(255 * dmin(1.0, ri * rr));
301         data[(h - 1 - j) * w + i].y = (uchar)(255 * dmin(1.0, gi * gg));
302         data[(h - 1 - j) * w + i].z = (uchar)(255 * dmin(1.0, bi * bb));
303       }
304       else
305       {
306         data[(h - 1 - j) * w + i] = uchar4{ 0, 0, 0, 0 };
307       }
308     }
309   }
310 }
311 
312 
313 __global__
314 void DeviceRender(Triangle* triangles, vec3 pc, vec3 pv,
315           int w, int h, double angle, uchar4* data,
316           vec3 lightPosition, uchar4 lightColor)
317 {
318   double pi = acos(-1.0);
319   int i, j;
320   double dw = 2.0 / (w - 1.0);
321   double dh = 2.0 / (h - 1.0);
322   double z = 1.0 / tan(angle * pi / 360.0);
323   vec3 bz = norm(diff(pv, pc));
324   vec3 bx = norm(prod(bz, { 0.0, 0.0, 1.0 }));
325   vec3 by = norm(prod(bx, bz));
326 
327   int kmin;
328   double tmin;
329 
330   int tid = blockIdx.x * blockDim.x + threadIdx.x;
331   int ofs = blockDim.x * gridDim.x;
332 
333   while (tid < w * h)
334   {
335     i = tid % w;
336     j = tid / w;
337 
338     tid += ofs;
339     vec3 v = { -1.0 + dw * i, (-1.0 + dh * j) * h / w, z };
340 
```

```
341     vec3 dir = norm(mult(bx, by, bz, v));
342
343     RayTracing(triangles, pc, dir, &kmin, &tmin);
344     if (kmin != -1)
345     {
346       double rr = (double)triangles[kmin].color.x / 255.0;
347       double gg = (double)triangles[kmin].color.y / 255.0;
348       double bb = (double)triangles[kmin].color.z / 255.0;
349       double ri = 0.2, gi = 0.2, bi = 0.2;
350
351       vec3 p = add(pc, mulc(dir, tmin));
352       vec3 l = diff(lightPosition, p);
353
354       vec3 n = prod(diff(triangles[kmin].b, triangles[kmin].a),
355              diff(triangles[kmin].c, triangles[kmin].a));
356       double dot_nl = dot(n, l);
357
358       if (dot_nl > 0)
359       {
360         ri += (lightColor.x / 255.0) * dot_nl / (len(n) * len(l));
361         gi += (lightColor.y / 255.0) * dot_nl / (len(n) * len(l));
362         bi += (lightColor.z / 255.0) * dot_nl / (len(n) * len(l));
363       }
364       data[(h - 1 - j) * w + i].x = (uchar)(255 * dmin(1.0, ri * rr));
365       data[(h - 1 - j) * w + i].y = (uchar)(255 * dmin(1.0, gi * gg));
366       data[(h - 1 - j) * w + i].z = (uchar)(255 * dmin(1.0, bi * bb));
367     }
368     else
369     {
370       data[(h - 1 - j) * w + i] = uchar4{ 0, 0, 0, 0 };
371     }
372   }
373 }
374
375
376 vec3 CoordCameraFromTime(double r0c, double z0c, double p0c,
377             double arc, double azc,
378             double wrc, double wzc, double wpc,
379             double prc, double pzc, double t)
380 {
381   double r = r0c + arc * sin(wrc * t + prc);
382   double z = z0c + azc * sin(wzc * t + pzc);
383   double phi = p0c + wpc * t;
384   return vec3{ r * cos(phi), r * sin(phi), z };
385 };
386
387
388 vec3 CoordViewPointFromTime(double r0n, double z0n, double p0n,
389              double arn, double azn,
```

```
390             double wrn, double wzn, double wpn,
391             double prn, double pzn, double t)
392 {
393   double r = r0n + arn * sin(wrn * t + prn);
394   double z = z0n + azn * sin(wzn * t + pzn);
395   double phi = p0n + wpn * t;
396   return vec3{ r * cos(phi), r * sin(phi), z };
397 };
398 
399 
400 // Sorry for that piece of ...code :)
401 void MpiReader(int rank, int& input)
402 {
403     if (!rank)
404     {
405         std::cin >> input;
406     }
407 
408     MPI_Bcast(&input, 1, MPI_INT, 0, MPI_COMM_WORLD);
409 }
410 
411 
412 void MpiReader(int rank, double& input)
413 {
414     if (!rank)
415     {
416         std::cin >> input;
417     }
418 
419     MPI_Bcast(&input, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
420 }
421 
422 
423 void MpiReader(int rank, std::string& input)
424 {
425     int sizeFilename;
426 
427     if (!rank)
428     {
429         std::cin >> input;
430         sizeFilename = input.size();
431     }
432 
433     MPI_Bcast(&sizeFilename, 1, MPI_INT, 0, MPI_COMM_WORLD);
434     input.resize(sizeFilename);
435     MPI_Bcast(const_cast<char*>(input.c_str()), sizeFilename, MPI_CHAR, 0,
        MPI_COMM_WORLD);
436 }
437 
```

```
438
439 int main(int argc, char* argv[])
440 {
441   int deviceSelection = 0;
442   if (argc >= 3)
443   {
444     printf("argc error\n");
445     return -1;
446   }
447   if (argc == 1)
448   {
449     deviceSelection = 1;
450   }
451   else if (strcmp(argv[1], "--default") == 0)
452   {
453     printf("400 \n");
454     printf("img_%% d.data \n");
455     printf("1240 960 100 \n");
456     printf("7.0 3.0 0.0 2.0 1.0 2.0 6.0 1.0 0.0 0.0 \n");
457     printf("2.0 0.0 0.0 0.5 0.1 1.0 4.0 1.0 0.0 0.0 \n");
458     printf("-2 -2 0 2 200 0 0 \n");
459     printf("-2 2 0 2 0 255 0 \n");
460     printf("2 0 0 2 0 0 255 \n");
461     printf("-4 -4 -1 -4 4 -1 4 -4 -1 4 4 -1 102 62 0 \n");
462     return 0;
463   }
464   else if (strcmp(argv[1], "--gpu") == 0)
465   {
466     deviceSelection = 1;
467   }
468   else if (strcmp(argv[1], "--cpu") == 0)
469   {
470     deviceSelection = 0;
471   }
472
473   int procRank, numberOfProcs;
474     MPI_Init(&argc, &argv);
475     MPI_Comm_size(MPI_COMM_WORLD, &numberOfProcs);
476     MPI_Comm_rank(MPI_COMM_WORLD, &procRank);
477   int deviceCnt;
478     cudaGetDeviceCount(&deviceCnt);
479     cudaSetDevice(procRank % deviceCnt);
480
481
482   int n, w, h;
483   double a = 100;
484   std::string path;
485
486   double r0c, z0c, p0c, arc, azc, wrc, wzc, wpc, prc, pzc,
```

```
487         r0n, z0n, p0n, arn, azn, wrn, wzn, wpn, prn, pzn;
488
489     double r1 = 2, r2 = 2, r3 = 2;
490     vec3 o1 = { -2, -2, 0 };
491     vec3 o2 = { -2, 2, 0 };
492     vec3 o3 = { 2, 0, 0 };
493
494     int c1x, c1y, c1z;
495     uchar4 c1 = { 200, 0, 0 };
496     int c2x, c2y, c2z;
497     uchar4 c2 = { 0, 255, 0 };
498     int c3x, c3y, c3z;
499     uchar4 c3 = { 0, 0, 255 };
500
501     vec3 fv[4];
502     fv[0] = { -5, -5, -3 };
503     fv[1] = { -5, 5, -3 };
504     fv[2] = { 5, -5, -3 };
505     fv[3] = { 5, 5, -3 };
506
507     int fcx, fcy, fcz;
508     uchar4 fc = { 102, 62, 0 };
509
510     // Wonderful work
511     MpiReader(procRank, n);
512     MpiReader(procRank, path);
513
514     MpiReader(procRank, w); MpiReader(procRank, h); MpiReader(procRank, a);
515
516     MpiReader(procRank, r0c); MpiReader(procRank, z0c); MpiReader(procRank, p0c);
            MpiReader(procRank, arc); MpiReader(procRank, azc); MpiReader(procRank, wrc);
            MpiReader(procRank, wzc); MpiReader(procRank, wpc); MpiReader(procRank, prc);
            MpiReader(procRank, pzc);
517     MpiReader(procRank, r0n); MpiReader(procRank, z0n); MpiReader(procRank, p0n);
            MpiReader(procRank, arn); MpiReader(procRank, azn); MpiReader(procRank, wrn);
            MpiReader(procRank, wzn); MpiReader(procRank, wpn); MpiReader(procRank, prn);
            MpiReader(procRank, pzn);
518
519     MpiReader(procRank, o1.x); MpiReader(procRank, o1.y); MpiReader(procRank, o1.z);
            MpiReader(procRank, r1); MpiReader(procRank, c1x); MpiReader(procRank, c1y);
            MpiReader(procRank, c1z);
520     MpiReader(procRank, o2.x); MpiReader(procRank, o2.y); MpiReader(procRank, o2.z);
            MpiReader(procRank, r2); MpiReader(procRank, c2x); MpiReader(procRank, c2y);
            MpiReader(procRank, c2z);
521     MpiReader(procRank, o3.x); MpiReader(procRank, o3.y); MpiReader(procRank, o3.z);
            MpiReader(procRank, r3); MpiReader(procRank, c3x); MpiReader(procRank, c3y);
            MpiReader(procRank, c3z);
522
523     MpiReader(procRank, fv[0].x); MpiReader(procRank, fv[0].y); MpiReader(procRank, fv
```

```
    [0].z); MpiReader(procRank, fv[1].x); MpiReader(procRank, fv[1].y); MpiReader(
    procRank, fv[1].z);
MpiReader(procRank, fv[2].x); MpiReader(procRank, fv[2].y); MpiReader(procRank, fv
    [2].z); MpiReader(procRank, fv[3].x); MpiReader(procRank, fv[3].y); MpiReader(
    procRank, fv[3].z);

MpiReader(procRank, fcx); MpiReader(procRank, fcy); MpiReader(procRank, fcz);


c1.x = c1x;
c1.y = c1y;
c1.z = c1z;

c2.x = c2x;
c2.y = c2y;
c2.z = c2z;

c3.x = c3x;
c3.y = c3y;
c3.z = c3z;

fc.x = fcx;
fc.y = fcy;
fc.z = fcz;

char buff[256];

uchar4* data = (uchar4*)malloc(sizeof(uchar4) * w * h);

vec3 pc, pv;

Triangle triangles[42];

BuildStage(triangles, r1, o1, c1, r2, o2, c2, r3, o3, c3, fv, fc);

double dt = 2 * M_PI / (double)n;

vec3 lightPosition = { -2, 0, 4 };
uchar4 lightColor = { 255, 255, 255 };

//float sharedTime = 0;

double timeStart;
  if (!procRank)
  {
      timeStart = MPI_Wtime();
  }

if (deviceSelection == 0)
```

```
569 {
570   double cpuTime;
571   for (int k = procRank; k < n; k += numberOfProcs)
572   {
573     pc = CoordCameraFromTime(r0c, z0c, p0c, arc, azc, wrc, wzc, wpc, prc, pzc, k * dt
          );
574     pv = CoordViewPointFromTime(r0n, z0n, p0n, arn, azn, wrn, wzn, wpn, prn, pzn, k *
           dt);
575
576     auto start = steady_clock::now();
577
578     Render(triangles, pc, pv, w, h, a, data, lightPosition, lightColor);
579
580     auto end = steady_clock::now();
581
582     cpuTime = ((double)duration_cast<microseconds>(end - start).count()) / 1000.0;
583
584     sprintf(buff, path.c_str(), k);
585
586     printf("%d: %s %e ms\n", k, buff, cpuTime);
587
588     //sharedTime += cpuTime;
589     FILE* out = fopen(buff, "wb");
590
591     fwrite(&w, sizeof(int), 1, out);
592     fwrite(&h, sizeof(int), 1, out);
593     fwrite(data, sizeof(uchar4), w * h, out);
594     fclose(out);
595   }
596   //printf("All time: %e ms\n", sharedTime);
597
598 }
599 else
600 {
601   float deviceTime = 0;
602
603   cudaEvent_t start, stop;
604   CSC(cudaEventCreate(&start));
605   CSC(cudaEventCreate(&stop));
606
607   Triangle* deviceTriangles;
608
609   CSC(cudaMalloc((void**)(&deviceTriangles), 42 * sizeof(Triangle)));
610   CSC(cudaMemcpy(deviceTriangles, triangles, 42 * sizeof(Triangle),
          cudaMemcpyHostToDevice));
611
612   uchar4* deviceData;
613   CSC(cudaMalloc((void**)(&deviceData), w * h * sizeof(uchar4)));
614
```

```
615     for (int k = procRank; k < n; k += numberOfProcs)
616     {
617       pc = CoordCameraFromTime(r0c, z0c, p0c, arc, azc, wrc, wzc, wpc, prc, pzc, k * dt
            );
618       pv = CoordViewPointFromTime(r0n, z0n, p0n, arn, azn, wrn, wzn, wpn, prn, pzn, k *
             dt);
619
620       CSC(cudaEventRecord(start));
621
622       DeviceRender<<<128, 128>>>(deviceTriangles, pc, pv, w, h, a, deviceData,
            lightPosition, lightColor);
623       CSC(cudaGetLastError());
624
625       CSC(cudaEventRecord(stop));
626       CSC(cudaEventSynchronize(stop));
627       CSC(cudaEventElapsedTime(&deviceTime, start, stop));
628
629       CSC(cudaMemcpy(data, deviceData, w * h * sizeof(uchar4), cudaMemcpyDeviceToHost))
            ;
630
631       sprintf(buff, path.c_str(), k);
632       printf("%d: %s %e ms\n", k, buff, deviceTime);
633
634       //sharedTime += deviceTime;
635
636       FILE* out = fopen(buff, "wb");
637
638       fwrite(&w, sizeof(int), 1, out);
639       fwrite(&h, sizeof(int), 1, out);
640       fwrite(data, sizeof(uchar4), w * h, out);
641       fclose(out);
642     }
643     //printf("All time: %e ms\n", sharedTime);
644
645     CSC(cudaEventDestroy(start));
646     CSC(cudaEventDestroy(stop));
647     CSC(cudaFree(deviceTriangles));
648     CSC(cudaFree(deviceData));
649   }
650   free(data);
651
652   MPI_Barrier(MPI_COMM_WORLD);
653   MPI_Finalize();
654
655   double timeEnd;
656     if (!procRank)
657     {
658         timeEnd = MPI_Wtime();
659         std::cout << "TIME: ";
```

```
660         std::cout << (timeEnd - timeStart) * 1000.0 << "ms" << std::endl;
661     }
662
663   return 0;
664 }
```

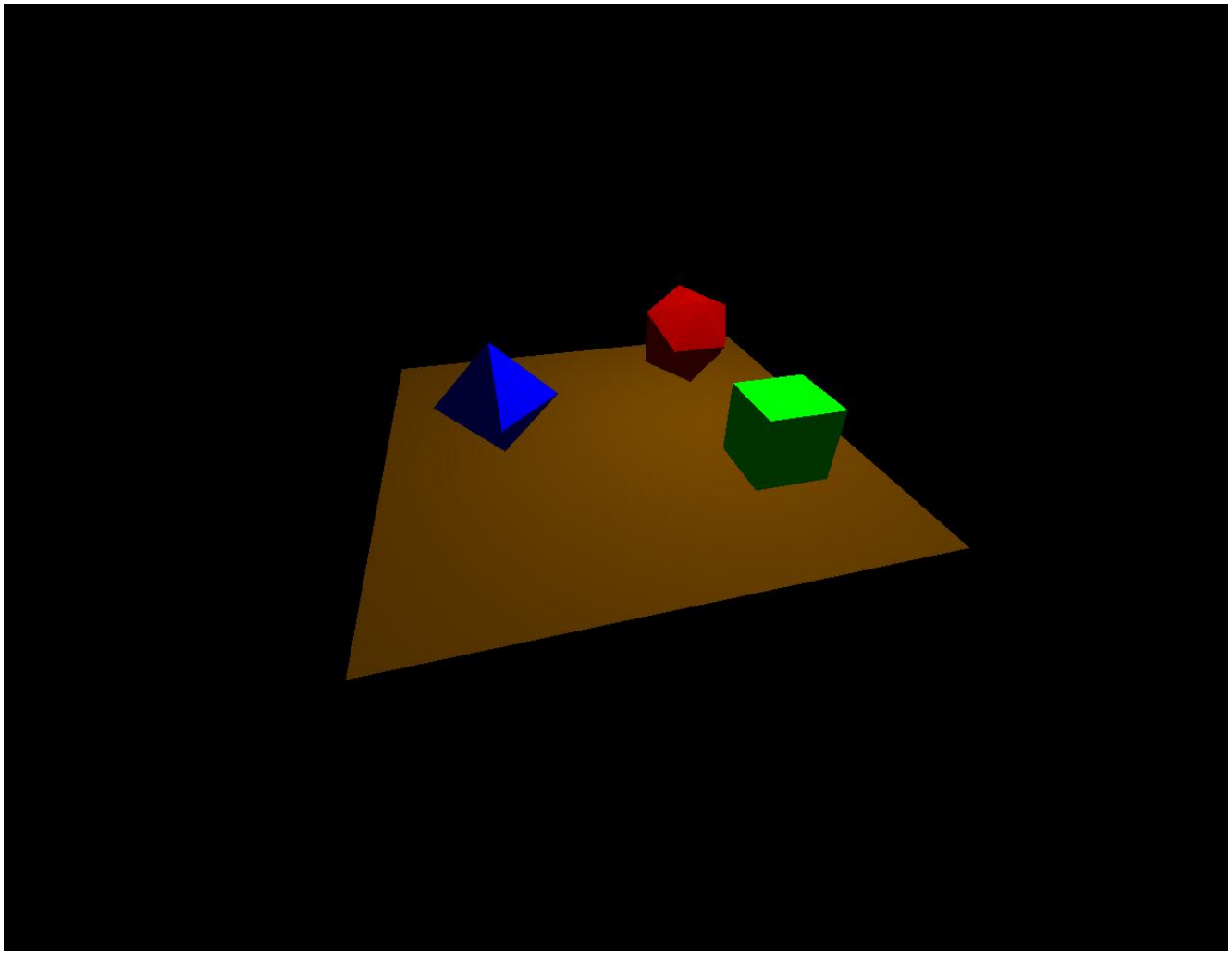
3 Результаты

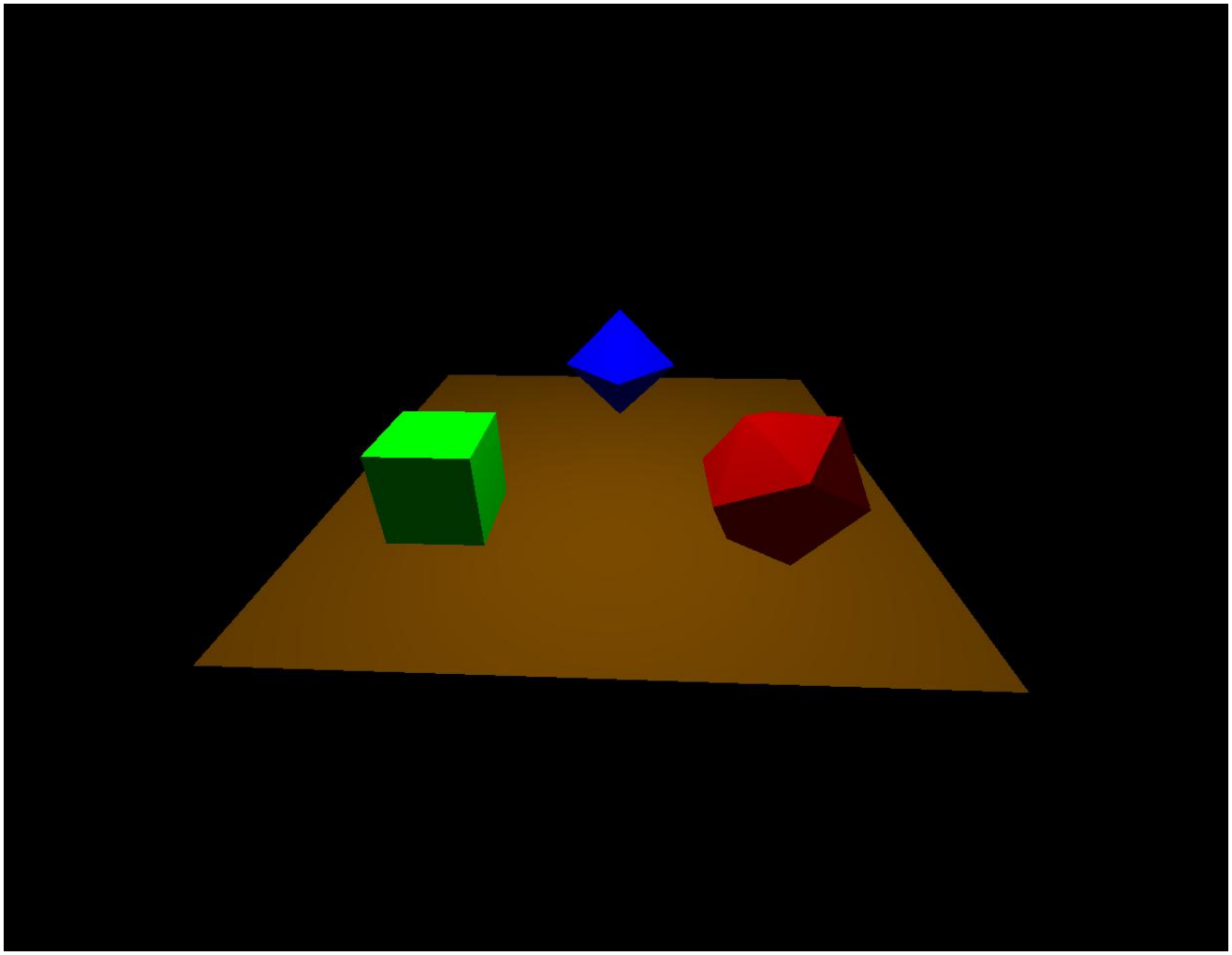
```
500
img_%d.data
1240 960 100
7.0 3.0 0.0 2.0 1.0 2.0 6.0 1.0 0.0 0.0
2.0 0.0 0.0 0.5 0.1 1.0 4.0 1.0 0.0 0.0

-2 -2  0  2 200 0 0
-2  2  0  2 0 255 0
2  0  0  2 0 0 255

-4 -4 -1 -4 4 -1 4 -4 -1 4 4 -1 102 62 0
```

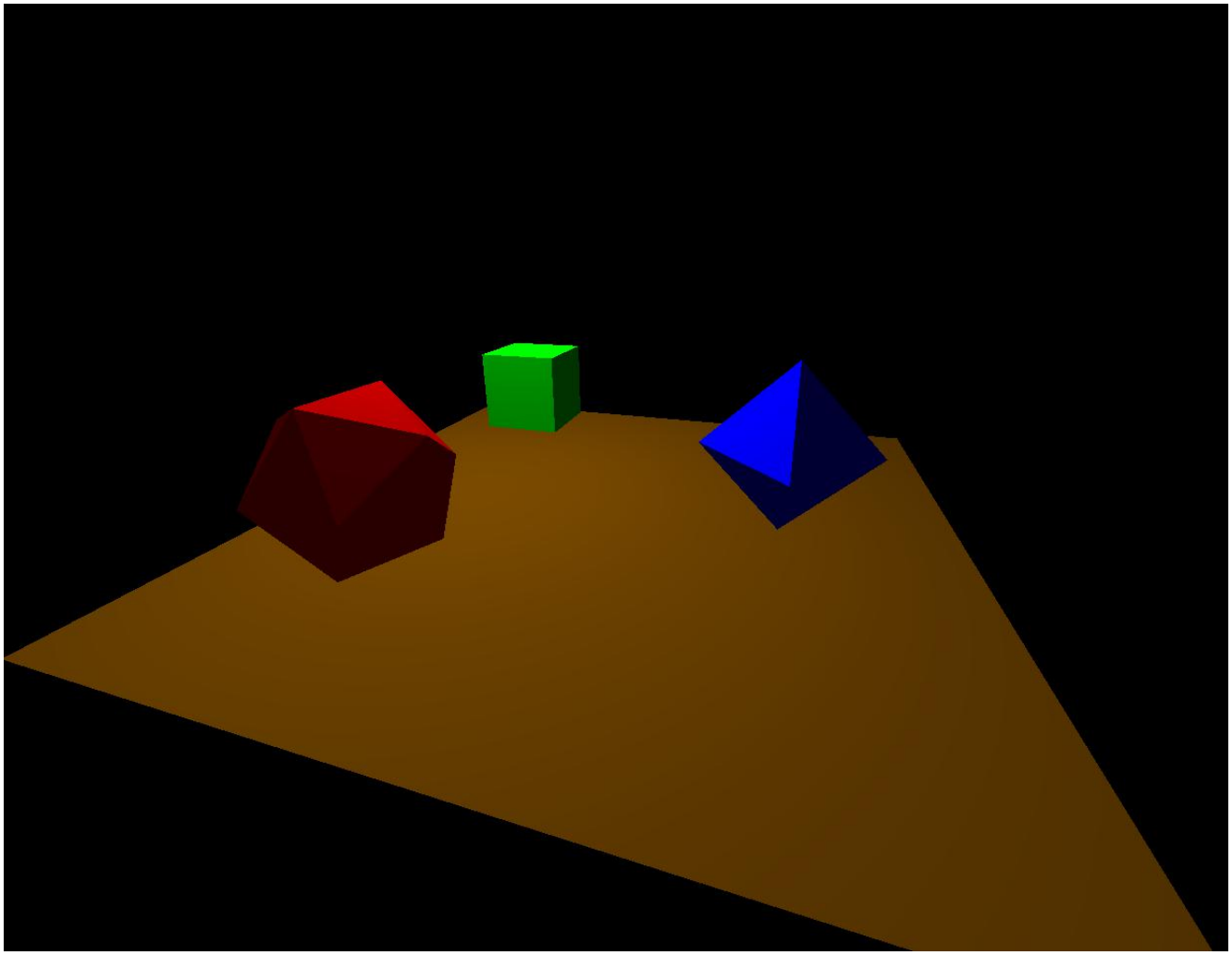
time CPU	time GPU	time MPI, OpenMP (CPU)	time MPI (GPU)
1283204 ms	55496.87 ms	246666 ms	84884.3 ms

Для более подробного сравнения приведу результаты первых 10 и последних 10 кадров (в миллисекундах).

time CPU	time GPU	time MPI, OpenMP (CPU)	time MPI (GPU)
img_0.data 4.105809e+03	1.101130e+02	4.899750e+02	1.091575e+02
img_1.data 3.737724e+03	1.100279e+02	4.785770e+02	1.091557e+02
img_2.data 4.201106e+03	1.099775e+02	5.078030e+02	1.091157e+02
img_3.data 3.807018e+03	1.099685e+02	4.456750e+02	1.096417e+02
img_4.data 2.088893e+03	1.099598e+02	4.676580e+02	1.096591e+02
img_5.data 2.208727e+03	1.099135e+02	4.446080e+02	1.093280e+02
img_6.data 2.647449e+03	1.098651e+02	4.568420e+02	1.092600e+02
img_7.data 2.744433e+03	1.097858e+02	4.459450e+02	1.092985e+02
img_8.data 3.871521e+03	1.097406e+02	4.444950e+02	1.091966e+02
img_9.data 3.832563e+03	1.097084e+02	4.596410e+02	1.088182e+02
img_490.data 1.894141e+03	1.108893e+02	4.635920e+02	1.101043e+02
img_491.data 1.929160e+03	1.108242e+02	4.530300e+02	1.098871e+02
img_492.data 1.915902e+03	1.107481e+02	4.550840e+02	1.108645e+02
img_493.data 1.935033e+03	1.106369e+02	4.630260e+02	1.100657e+02
img_494.data 1.931923e+03	1.105878e+02	4.522850e+02	1.102672e+02
img_495.data 1.910946e+03	1.104663e+02	5.069480e+02	1.101924e+02
img_496.data 1.908709e+03	1.104069e+02	4.540410e+02	1.094623e+02
img_497.data 1.897130e+03	1.103351e+02	4.799600e+02	1.094127e+02
img_498.data 1.899476e+03	1.102800e+02	4.509210e+02	1.111337e+02
img_499.data 1.887473e+03	1.101694e+02	4.657620e+02	1.092396e+02





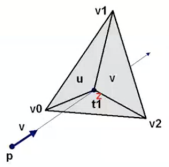


4 Выводы

Во время выполнения курсовой работы я познакомилась с технологией ray tracing. Ray tracing - это по сути обратная проекция пикселей изображения. Эта технология намного сложнее и тяжелее, чем растеризация, но изображение в итоге получается намного реалистичнее. Основной задачей ray tracing является пересечение луча и треугольника. Существует несколько подходов в решении этой задачи: наивный, оптимизированный и unit-тестирование. В данной работе применялся оптимизированный вариант.

Пересечение луча и треугольника

- Простой вариант
  - Операции (* : 39, +/- : 53, / : 1)
- Оптимизированный вариант
  - Операции (* : 23, +/- : 24, / : 1)
- Юнит тест
  - Операции (*: 20, + : 20, / : 1)
  - Экономит регистры GPU



В данном проекте была добавлена технологии MPI и OpenMP.
Message Passing Interface (MPI) — программный интерфейс для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу.
OpenMP — открытый стандарт для распараллеливания программ. Даёт описание совокупности директив компилятора, библиотечных процедур и переменных окружения, которые предназначены для программирования многопоточных приложений на многопроцессорных системах с общей памятью. Таким образом, в этой работе была реализована параллельная обработка на CPU с использованием технологии OpenMP в связке с MPI.

В ходе тестирования я выяснила, что версия для gpu работает быстрее cpu версии. А благодаря совместному использованию технологий OpenMP и MPI версия на CPU стала работать быстрее обычной в 5 раз. Трассировка лучей применяется в компьютерных играх. Turing от Nvidia стала первой архитектурой (лето 2018), позволяющей проводить трассировку лучей в реальном времени на GPU. Другие области применения трассировки лучей - это аурализация и высокочастотные технологии.