Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа по курсу «Параллельная обработка данных»

Обратная трассировка лучей (Ray Tracing) на GPU.

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

Москва, 2021

Условие

Цель работы: Использование GPU для создание фотореалистической визуализации. Рендеринг полузеркальных и полупрозрачных правильных геометрических тел. Получение эффекта бесконечности. Создание видеоролика.

Вариант 8:
На сцене должны располагаться три тела: Гексаэдр, Октаэдр, Икосаэдр.
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

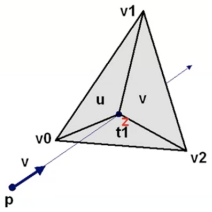
Для начала строим сцену: задаем координаты вершин тел и добавляем грани в массив сцены, строим из полигонов пол сцены. Далее нужно настроить камеру. Для этого нужно посчитать базис, связанный с камерой, в который нам нужно перейти, посчитать векторы, которые находятся в плоскости Oxy - это наши лучики 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.

2 Исходный код

```
#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>

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;
};


__device__ __host__
double dot(vec3 a, vec3 b)
{
    return a.x * b.x + a.y * b.y + a.z * b.z;
```

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

```
96      return {
97                  a.x * v.x + b.x * v.y + c.x * v.z,
98                  a.y * v.x + b.y * v.y + c.y * v.z,
99                  a.z * v.x + b.z * v.y + c.z * v.z
100             };
101 }
102
103
104 void print(vec3 v)
105 {
106     printf("%e %e %e\n", v.x, v.y, v.z);
107 }
108
109
110 __host__ __device__
111 double dmin(double x, double y)
112 {
113     if (x > y) { return y; }
114     return x;
115 }
116
117
118 void BuildStage(Triangle* t, double r1, vec3 o1, uchar4 c1,
119                 double r2, vec3 o2, uchar4 c2,
120                 double r3, vec3 o3, uchar4 c3,
121                 vec3* fv, uchar4 fc)
122 {
123     int st = 0;
124     double p = (1 + sqrt(5)) / 2;
125
126     vec3 icosVertexes[] = {
127         add(o1, norm(vec3{ 0, -1, p})),
128         add(o1, norm(vec3{ 0, 1, p})),
129         add(o1, norm(vec3{-p, 0, 1})),
130         add(o1, norm(vec3{ p, 0, 1})),
131         add(o1, norm(vec3{-1, p, 0})),
132         add(o1, norm(vec3{ 1, p, 0})),
133         add(o1, norm(vec3{ 1, -p, 0})),
134         add(o1, norm(vec3{-1, -p, 0})),
135         add(o1, norm(vec3{-p, 0, -1})),
136         add(o1, norm(vec3{ p, 0, -1})),
137         add(o1, norm(vec3{ 0, -1, -p})),
138         add(o1, norm(vec3{ 0, 1, -p}))
139     };
140
141     t[st++] = Triangle{ icosVertexes[0], icosVertexes[1], icosVertexes[2] , c1 };
142     t[st++] = Triangle{ icosVertexes[1], icosVertexes[0], icosVertexes[3] , c1 };
143     t[st++] = Triangle{ icosVertexes[0], icosVertexes[2], icosVertexes[7] , c1 };
144     t[st++] = Triangle{ icosVertexes[2], icosVertexes[1], icosVertexes[4] , c1 };
```

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

```
243
244
245 void Render(Triangle* triangles, vec3 pc, vec3 pv,
246           int w, int h, double angle, uchar4* data,
247           vec3 lightPosition, uchar4 lightColor)
248 {
249     double pi = acos(-1.0);
250     int i, j;
251     double dw = 2.0 / (w - 1.0);
252     double dh = 2.0 / (h - 1.0);
253     double z = 1.0 / tan(angle * pi / 360.0);
254
255     vec3 bz = norm(diff(pv, pc));
256     vec3 bx = norm(prod(bz, { 0.0, 0.0, 1.0 }));
257     vec3 by = norm(prod(bx, bz));
258
259     int kmin;
260     double tmin;
261
262     for (i = 0; i < w; i++)
263         for (j = 0; j < h; j++)
264         {
265             vec3 v = { -1.0 + dw * i, (-1.0 + dh * j) * h / w, z };
266
267             vec3 dir = norm(mult(bx, by, bz, v));
268
269             RayTracing(triangles, pc, dir, &kmin, &tmin);
270             if (kmin != -1)
271             {
272                 double rr = (double)triangles[kmin].color.x / 255.0;
273                 double gg = (double)triangles[kmin].color.y / 255.0;
274                 double bb = (double)triangles[kmin].color.z / 255.0;
275                 double ri = 0.2, gi = 0.2, bi = 0.2;
276
277                 vec3 p = add(pc, mulc(dir, tmin));
278                 vec3 l = diff(lightPosition, p);
279                 vec3 n = prod(diff(triangles[kmin].b, triangles[kmin].a),
280                             diff(triangles[kmin].c, triangles[kmin].a));
281                 double dot_nl = dot(n, l);
282
283                 if (dot_nl > 0)
284                 {
285                     ri += (lightColor.x / 255.0) * dot_nl / (len(n) * len(l));
286                     gi += (lightColor.y / 255.0) * dot_nl / (len(n) * len(l));
287                     bi += (lightColor.z / 255.0) * dot_nl / (len(n) * len(l));
288                 }
289                 data[(h - 1 - j) * w + i].x = (uchar)(255 * dmin(1.0, ri * rr));
290                 data[(h - 1 - j) * w + i].y = (uchar)(255 * dmin(1.0, gi * gg));
291                 data[(h - 1 - j) * w + i].z = (uchar)(255 * dmin(1.0, bi * bb));
```

```
            }
            else
            {
                data[(h - 1 - j) * w + i] = uchar4{ 0, 0, 0, 0 };
            }
        }
}


__global__
void DeviceRender(Triangle* triangles, vec3 pc, vec3 pv,
                  int w, int h, double angle, uchar4* data,
                  vec3 lightPosition, uchar4 lightColor)
{
    double pi = acos(-1.0);
    int i, j;
    double dw = 2.0 / (w - 1.0);
    double dh = 2.0 / (h - 1.0);
    double z = 1.0 / tan(angle * pi / 360.0);

    vec3 bz = norm(diff(pv, pc));
    vec3 bx = norm(prod(bz, { 0.0, 0.0, 1.0 }));
    vec3 by = norm(prod(bx, bz));

    int kmin;
    double tmin;

    int tid = blockIdx.x * blockDim.x + threadIdx.x;
    int ofs = blockDim.x * gridDim.x;

    while (tid < w * h)
    {
        i = tid % w;
        j = tid / w;

        tid += ofs;

        vec3 v = { -1.0 + dw * i, (-1.0 + dh * j) * h / w, z };

        vec3 dir = norm(mult(bx, by, bz, v));

        RayTracing(triangles, pc, dir, &kmin, &tmin);
        if (kmin != -1)
        {
            double rr = (double)triangles[kmin].color.x / 255.0;
            double gg = (double)triangles[kmin].color.y / 255.0;
            double bb = (double)triangles[kmin].color.z / 255.0;
            double ri = 0.2, gi = 0.2, bi = 0.2;

```

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

```
{
    int deviceSelection = 0;
    if (argc >= 3)
    {
        printf("argc error\n");
        return -1;
    }
    if (argc == 1)
    {
        deviceSelection = 1;
    }
    else if (strcmp(argv[1], "--default") == 0)
    {
        printf("400 \n");
        printf("img_%% d.data \n");
        printf("1240 960 100 \n");
        printf("7.0 3.0 0.0 2.0 1.0 2.0 6.0 1.0 0.0 0.0 \n");
        printf("2.0 0.0 0.0 0.5 0.1 1.0 4.0 1.0 0.0 0.0 \n");
        printf("-2 -2 0 2 200 0 0 \n");
        printf("-2 2 0 2 0 255 0 \n");
        printf("2 0 0 2 0 0 255 \n");
        printf("-4 -4 -1 -4 4 -1 4 -4 -1 4 4 -1 102 62 0 \n");
        return 0;
    }
    else if (strcmp(argv[1], "--gpu") == 0)
    {
        deviceSelection = 1;
    }
    else if (strcmp(argv[1], "--cpu") == 0)
    {
        deviceSelection = 0;
    }

    int n, w, h;
    double a = 100;
    char path[256];

    double r0c, z0c, p0c, arc, azc, wrc, wzc, wpc, prc, pzc,
           r0n, z0n, p0n, arn, azn, wrn, wzn, wpn, prn, pzn;

    double r1 = 2, r2 = 2, r3 = 2;
    vec3 o1 = { -2, -2, 0 };
    vec3 o2 = { -2, 2, 0 };
    vec3 o3 = { 2, 0, 0 };

    int c1x, c1y, c1z;
    uchar4 c1 = { 200, 0, 0 };
    int c2x, c2y, c2z;
    uchar4 c2 = { 0, 255, 0 };
```

```
439     int c3x, c3y, c3z;
440     uchar4 c3 = { 0, 0, 255 };
441
442     vec3 fv[4];
443     fv[0] = { -5, -5, -3 };
444     fv[1] = { -5, 5, -3 };
445     fv[2] = { 5, -5, -3 };
446     fv[3] = { 5, 5, -3 };
447
448     int fcx, fcy, fcz;
449     uchar4 fc = { 102, 62, 0 };
450
451     scanf("%d\n", &n);
452     scanf("%s\n", path);
453     scanf("%d %d %lf\n", &w, &h, &a);
454
455     scanf("%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
456         &r0c, &z0c, &p0c, &arc, &azc, &wrc, &wzc, &wpc, &prc, &pzc);
457     scanf("%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf",
458         &r0n, &z0n, &p0n, &arn, &azn, &wrn, &wzn, &wpn, &prn, &pzn);
459
460     scanf("%lf %lf %lf %lf %d %d %d\n",
461         &o1.x, &o1.y, &o1.z, &r1, &c1x, &c1y, &c1z);
462     scanf("%lf %lf %lf %lf %d %d %d\n",
463         &o2.x, &o2.y, &o2.z, &r2, &c2x, &c2y, &c2z);
464     scanf("%lf %lf %lf %lf %d %d %d\n",
465         &o3.x, &o3.y, &o3.z, &r3, &c3x, &c3y, &c3z);
466     scanf("%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %lf %d %d %d",
467         &fv[0].x, &fv[0].y, &fv[0].z, &fv[1].x, &fv[1].y, &fv[1].z,
468         &fv[2].x, &fv[2].y, &fv[2].z, &fv[3].x, &fv[3].y, &fv[3].z,
469         &fcx, &fcy, &fcz);
470
471     c1.x = c1x;
472     c1.y = c1y;
473     c1.z = c1z;
474
475     c2.x = c2x;
476     c2.y = c2y;
477     c2.z = c2z;
478
479     c3.x = c3x;
480     c3.y = c3y;
481     c3.z = c3z;
482
483     fc.x = fcx;
484     fc.y = fcy;
485     fc.z = fcz;
486
487     char buff[256];
```

```
488     uchar4* data = (uchar4*)malloc(sizeof(uchar4) * w * h);
489
490     vec3 pc, pv;
491
492     Triangle triangles[42];
493
494     BuildStage(triangles, r1, o1, c1, r2, o2, c2, r3, o3, c3, fv, fc);
495
496     double pi = acos(-1);
497
498     double dt = 2 * pi / (double)n;
499
500     vec3 lightPosition = { -2, 0, 4 };
501     uchar4 lightColor = { 255, 255, 255 };
502
503     //float sharedTime = 0;
504
505     if (deviceSelection == 0)
506     {
507         double cpuTime;
508
509         for (int k = 0; k < n; ++k)
510         {
511             pc = CoordCameraFromTime(r0c, z0c, p0c, arc, azc, wrc, wzc, wpc, prc, pzc,
                    k * dt);
512             pv = CoordViewPointFromTime(r0n, z0n, p0n, arn, azn, wrn, wzn, wpn, prn,
                    pzn, k * dt);
513
514             auto start = steady_clock::now();
515
516             Render(triangles, pc, pv, w, h, a, data, lightPosition, lightColor);
517
518             auto end = steady_clock::now();
519
520             cpuTime = ((double)duration_cast<microseconds>(end - start).count()) /
                    1000.0;
521
522             sprintf(buff, path, k);
523             printf("%s %e ms\n", buff, cpuTime);
524             //sharedTime += cpuTime;
525             FILE* out = fopen(buff, "wb");
526
527             fwrite(&w, sizeof(int), 1, out);
528             fwrite(&h, sizeof(int), 1, out);
529             fwrite(data, sizeof(uchar4), w * h, out);
530             fclose(out);
531         }
532         //printf("All time: %e ms\n", sharedTime);
533
```

```
534     }
535     else
536     {
537         float deviceTime = 0;
538         cudaEvent_t start, stop;
539         CSC(cudaEventCreate(&start));
540         CSC(cudaEventCreate(&stop));
541 
542         Triangle* deviceTriangles;
543 
544         CSC(cudaMalloc((void**)(&deviceTriangles), 42 * sizeof(Triangle)));
545         CSC(cudaMemcpy(deviceTriangles, triangles, 42 * sizeof(Triangle),
                cudaMemcpyHostToDevice));
546 
547         uchar4* deviceData;
548         CSC(cudaMalloc((void**)(&deviceData), w * h * sizeof(uchar4)));
549 
550         for (int k = 0; k < n; ++k)
551         {
552             pc = CoordCameraFromTime(r0c, z0c, p0c, arc, azc, wrc, wzc, wpc, prc, pzc,
                    k * dt);
553             pv = CoordViewPointFromTime(r0n, z0n, p0n, arn, azn, wrn, wzn, wpn, prn,
                    pzn, k * dt);
554 
555             CSC(cudaEventRecord(start));
556 
557             DeviceRender<<<128, 128>>>(deviceTriangles, pc, pv, w, h, a, deviceData,
                    lightPosition, lightColor);
558             CSC(cudaGetLastError());
559 
560             CSC(cudaEventRecord(stop));
561             CSC(cudaEventSynchronize(stop));
562             CSC(cudaEventElapsedTime(&deviceTime, start, stop));
563 
564             CSC(cudaMemcpy(data, deviceData, w * h * sizeof(uchar4),
                    cudaMemcpyDeviceToHost));
565 
566             sprintf(buff, path, k);
567             printf("%s %e ms\n", buff, deviceTime);
568 
569             //sharedTime += deviceTime;
570 
571             FILE* out = fopen(buff, "wb");
572 
573             fwrite(&w, sizeof(int), 1, out);
574             fwrite(&h, sizeof(int), 1, out);
575             fwrite(data, sizeof(uchar4), w * h, out);
576             fclose(out);
577         }
```

```
578         //printf("All time: %e ms\n", sharedTime);
579 
580         CSC(cudaEventDestroy(start));
581         CSC(cudaEventDestroy(stop));
582         CSC(cudaFree(deviceTriangles));
583         CSC(cudaFree(deviceData));
584     }
585     free(data);
586     return 0;
587 }
```

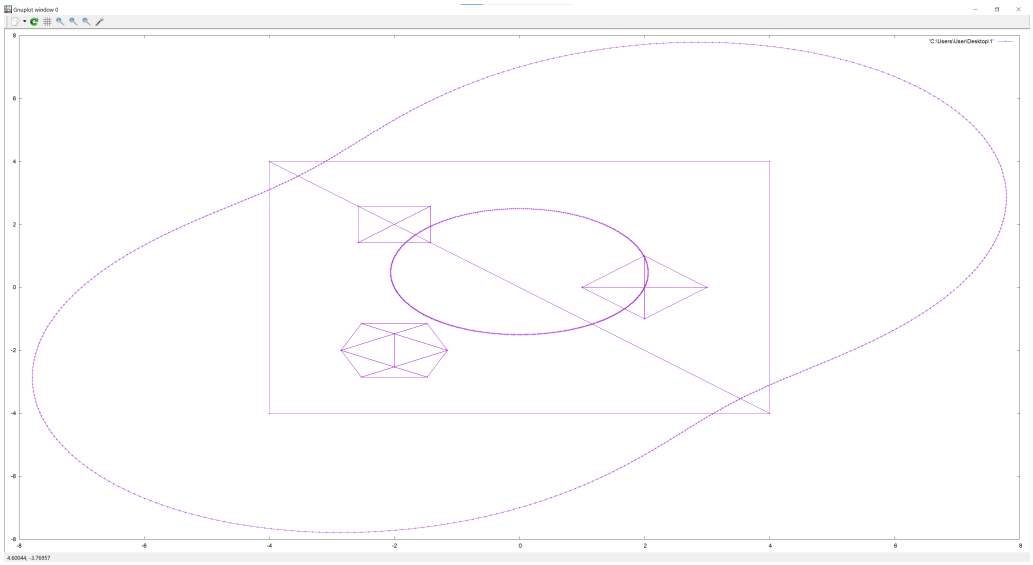
3 Результаты

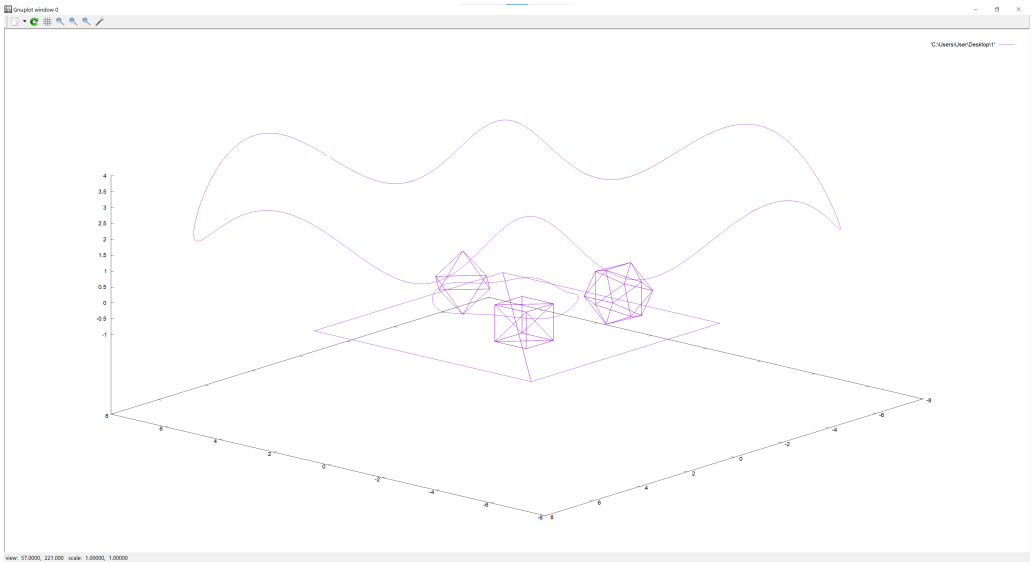
```
500
img_%d.data
1240 960 100
7.0 3.0 0.0 2.0 1.0 2.0 6.0 1.0 0.0 0.0
2.0 0.0 0.0 0.5 0.1 1.0 4.0 1.0 0.0 0.0

-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
```

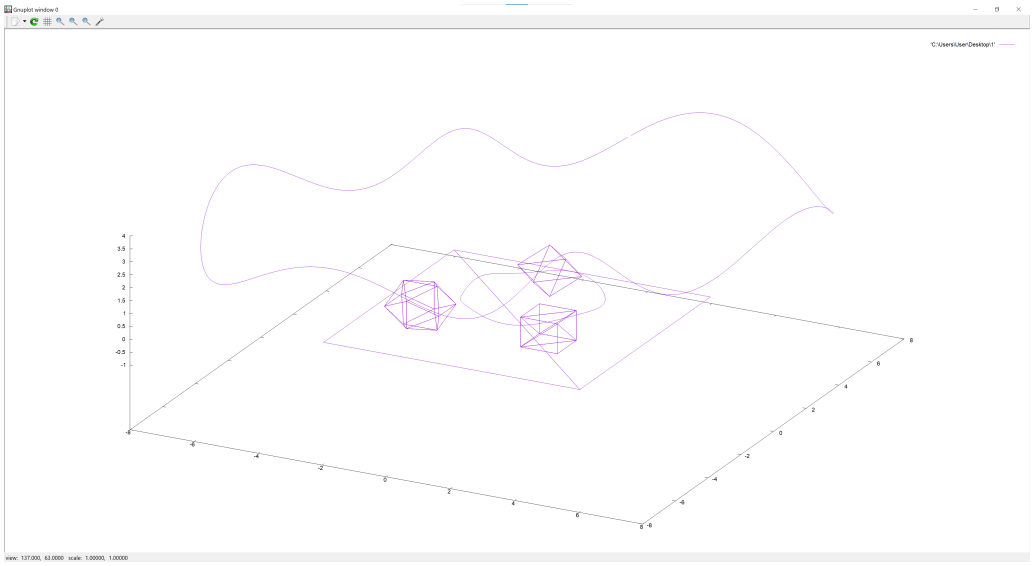
time GPU	time CPU
5.549607e+04 ms	9.520951e+05 ms

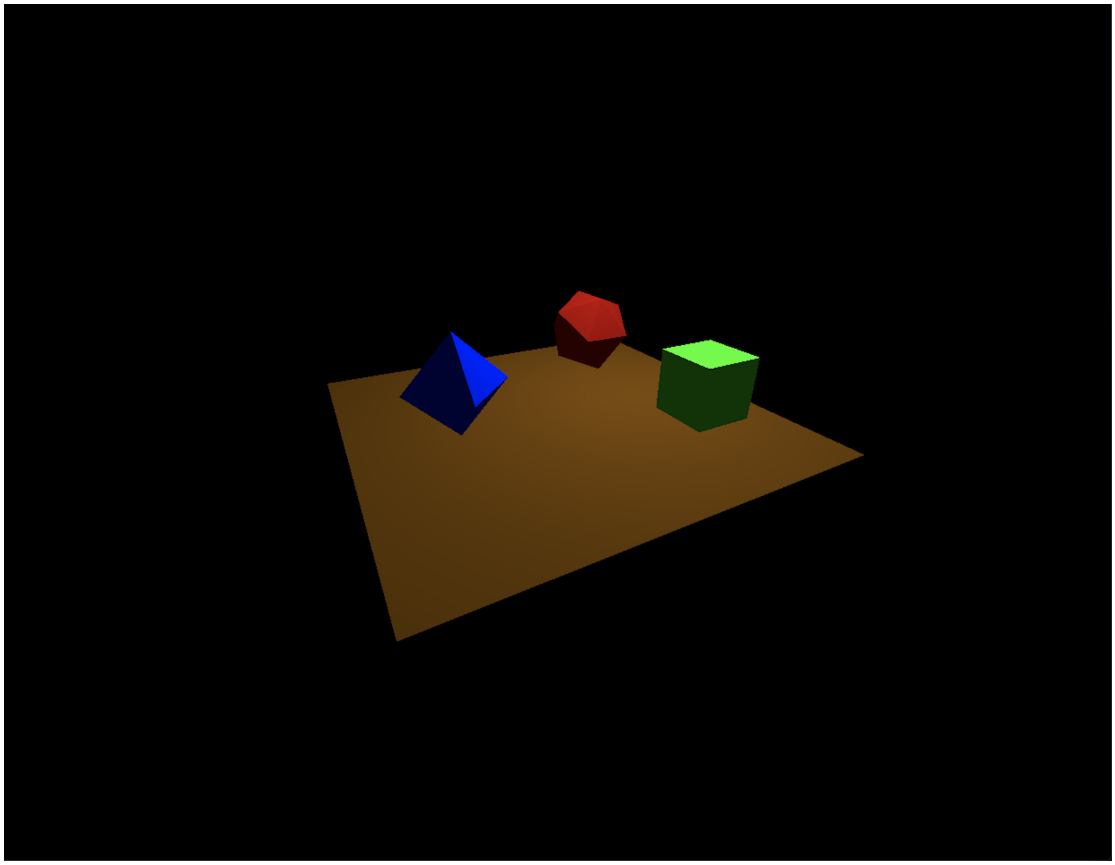
Для более подробного сравнения приведу результаты первых 10 и последних 10 кадров.

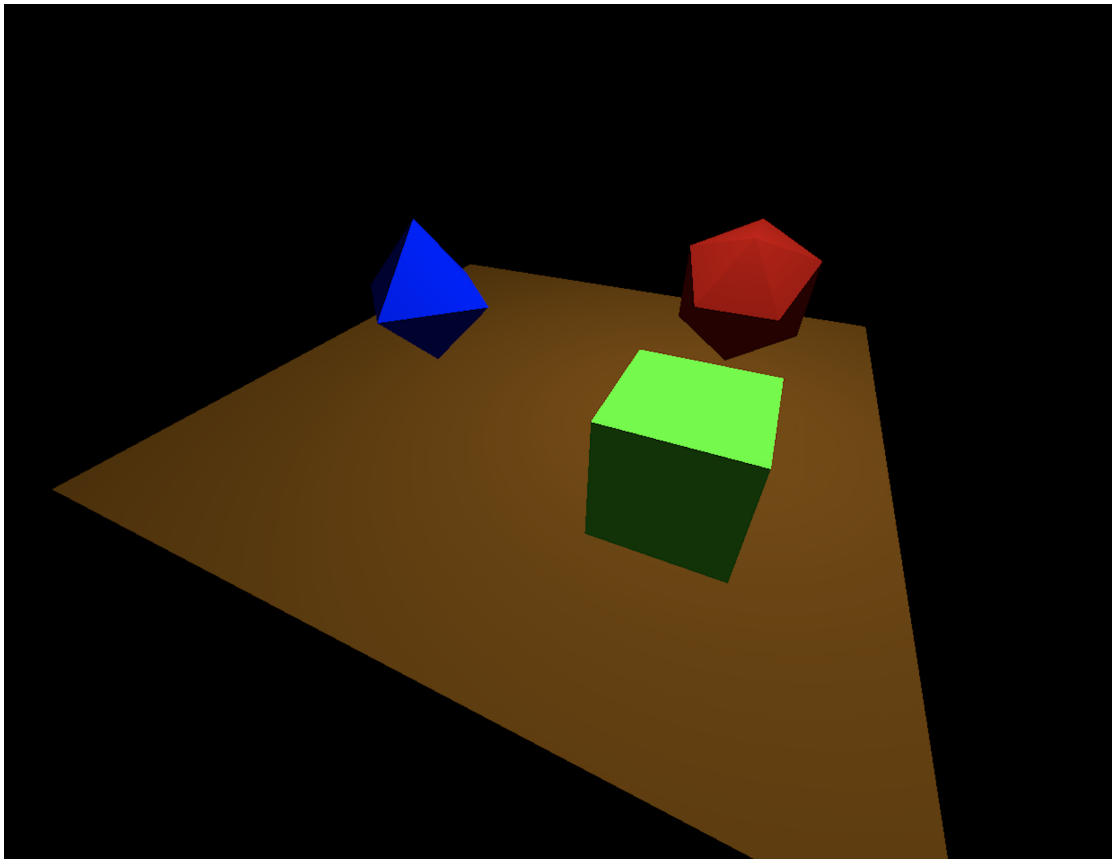
time GPU	time CPU
img_0.data 1.101275e+02 ms	img_0.data 1.954221e+03 ms
img_1.data 1.100945e+02 ms	img_1.data 1.924304e+03 ms
img_2.data 1.100306e+02 ms	img_2.data 1.923791e+03 ms
img_3.data 1.100010e+02 ms	img_3.data 1.869273e+03 ms
img_4.data 1.099160e+02 ms	img_4.data 1.863647e+03 ms
img_5.data 1.098635e+02 ms	img_5.data 1.863423e+03 ms
img_6.data 1.098893e+02 ms	img_6.data 1.861964e+03 ms
img_7.data 1.098086e+02 ms	img_7.data 1.862511e+03 ms
img_8.data 1.097993e+02 ms	img_8.data 1.906986e+03 ms
img_9.data 1.096939e+02 ms	img_9.data 1.861215e+03 ms
img_490.data 1.108853e+02 ms	img_490.data 1.874304e+03 ms
img_491.data 1.108295e+02 ms	img_491.data 1.873343e+03 ms
img_492.data 1.107673e+02 ms	img_492.data 1.872002e+03 ms
img_493.data 1.106642e+02 ms	img_493.data 1.876053e+03 ms
img_494.data 1.105577e+02 ms	img_494.data 1.870852e+03 ms
img_495.data 1.104750e+02 ms	img_495.data 1.871667e+03 ms
img_496.data 1.104352e+02 ms	img_496.data 1.873681e+03 ms
img_497.data 1.103198e+02 ms	img_497.data 1.873698e+03 ms
img_498.data 1.102551e+02 ms	img_498.data 1.877903e+03 ms
img_499.data 1.101609e+02 ms	img_499.data 1.879259e+03 ms

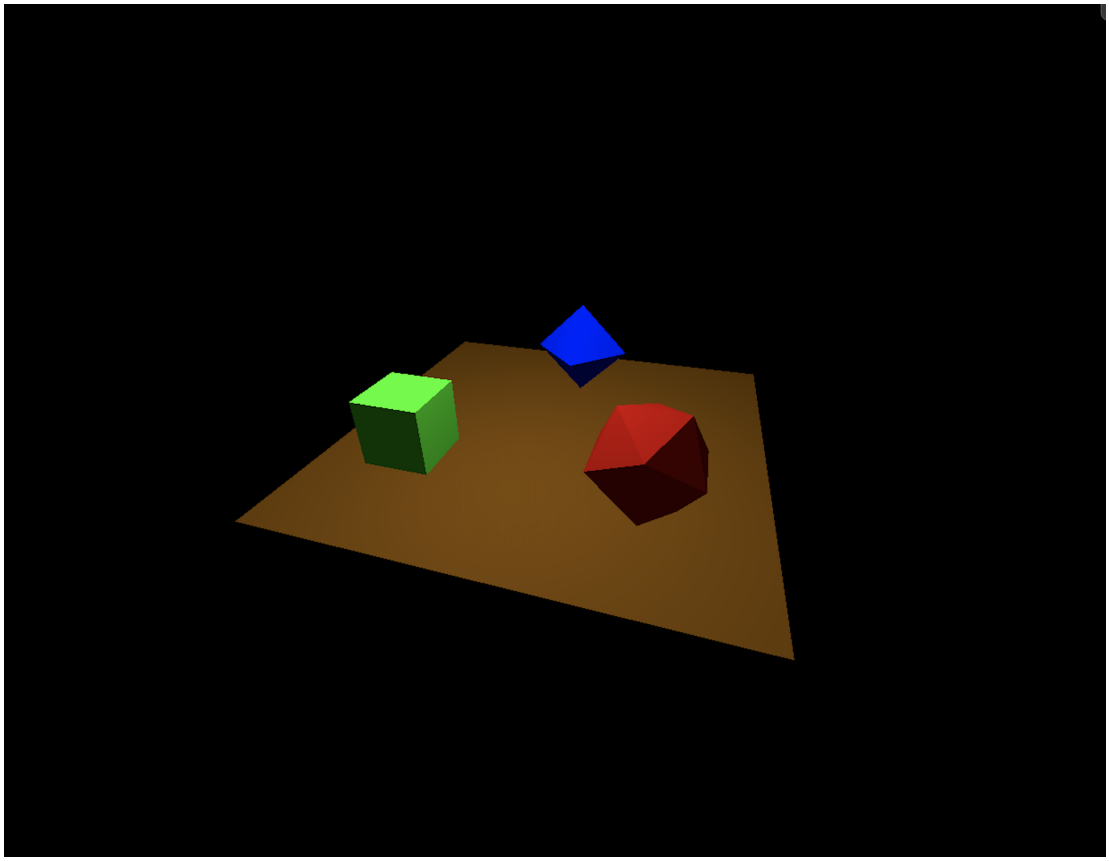

Gnuplot window 0
'C:\Users\User\Desktop\1'
8
6
4
2
0
-2
-4
-6
-8
-8
-6
-4
-2
0
2
4
6
8
4.60044, -1.76957


Gnuplot window 0
'C:\Users\User\Desktop\1'
4
3.5
3
2.5
2
1.5
1
0.5
0
-0.5
-1
8
6
4
2
0
-2
-4
-6
-8
-8
-6
-4
-2
0
2
4
6
8
view: 57.0000, 221.000 scale: 1.00000, 1.00000


Gnuplot window 0
'C:\Users\User\Desktop\1'

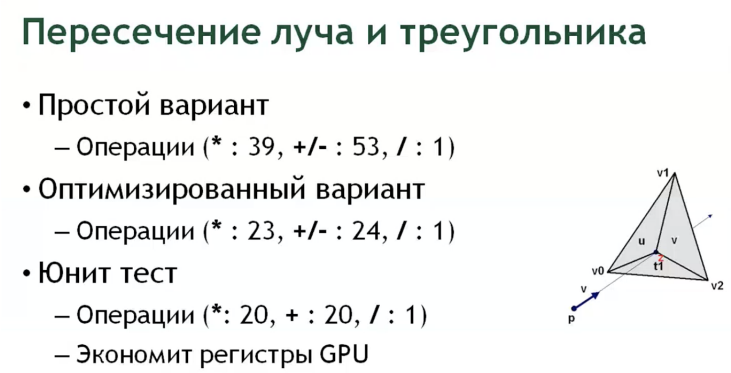






4 Выводы

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.



В ходе тестирования я выяснила, что версия для gpu работает быстрее cpu версии. 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.