МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Компьютерные науки и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Курсовая работа №7
по курсу «Параллельная обработка данных»

Обратная трассировка лучей (Ray Tracing) на GPU

Студент: А. А. Садаков
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2023

Условие

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

Задание:

Сцена. Прямоугольная текстурированная поверхность (пол), над которой расположены три платоновых тела. Сверху находятся несколько источников света. На каждом ребре многогранника располагается определенное количество точечных источников света. Грани тел обладают зеркальным и прозрачным эффектом. За счет многократного переотражения лучей внутри тела, возникает эффект бесконечности.

Камера. Камера выполняет облет сцены согласно определенным законам. В цилиндрических координатах (r, φ, z), положение и точка направления камеры в момент времени t определяется следующим образом:

$$
\begin{aligned}
r_c(t) &= r_0 + A_c^r \sin(w_c^r t + p_c^r) \\
z_c(t) &= z_0 + A_c^z \sin(w_c^z t + p_c^z) \\
\phi_c(t) &= \phi_c^0 + w_c^\phi t \\
r_n(t) &= r_0 + A_n^r \sin(w_n^r t + p_n^r) \\
z_n(t) &= z_0 + A_n^z \sin(w_n^z t + p_n^z) \\
\phi_n(t) &= \phi_n^0 + w_n^\phi t
\end{aligned}
$$

где

$t \in [0, 2\pi]$

Требуется реализовать алгоритм обратной трассировки лучей с использованием технологии CUDA. Выполнить покадровый рендеринг сцены. Для устранения эффекта «зубчатости», выполнить сглаживание (например с помощью алгоритма SSAA). Полученный набор кадров склеить в анимацию любым доступным программным обеспечением. Подобрать параметры сцены, камеры и освещения таким образом, чтобы получить наиболее красочный результат. Провести сравнение производительности gpu и cpu (т.е. дополнительно нужно реализовать алгоритм без использования CUDA).

Программа должна принимать на вход следующие параметры:

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 $r_c^0, z_c^0, \phi_c^0, A_c^r, A_c^z, w_c^r, w_c^z, w_c^\phi, p_c^r, p_c^z$ и $r_n^0, z_n^0, \phi_n^0, A_n^r, A_n^z, w_n^r, w_n^z, w_n^\phi, p_n^r, p_n^z$

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.

8. Максимальная глубина рекурсии и квадратный корень из количества лучей на один пиксель (для SSAA).

Вариант 3: Тетраэдр, Гексаэдр, Икосаэдр.

Программное и аппаратное обеспечение

ГРАФИЧЕСКИЙ ПРОЦЕССОР
Имя устройства: NVIDIA GeForce GTX 1650
Compute capability: 7.5
Размер графической памяти: 4242604032
Размер разделяемой памяти: 49152
Размер константной памяти: 65536
Максимальное количество регистров на блок: 65536
Максимальное количество потоков на блок: 1024
Количество мультипроцессоров: 14

ПРОЦЕССОР
Имя устройства: Intel Core I5-10300H
Архитектура: Comet Lake-H
Количество ядер (потоков): 4(8)
Базовая (максимальная) частота: 2,5(4,5)ГГц
Кеш 1-го уровня: 64Кб (на ядро)
Кеш 2-го уровня: 256Кб (на ядро)
Кеш 3-го уровня: 6Мб (всего)

ОПЕРАТИВНАЯ ПАМЯТЬ
Объём: 8Гб
Тип: DDR4
Частота:2933МГц

ЖЁСТКИЙ ДИСК
Имя устройства: Intel SSDPEKNW512G8L
Тип диска: SSD
Объём памяти: 512Гб
Скорость чтения/записи: 1500/1000 Мб/с

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ
OS: Windows 10,Подсистема Ubuntu 20.04
IDE: Visual Studio Code
Компилятор: nvcc

Метод решения

Все фигуры в сцене заданы с помощью треугольников, в качестве основной идеи используется вариант пересечения луча и полигона. Для добавления в работу теней была использована модель освещения Фонга. При вычислении цвета точки выпускаем луч в направлении источника света, если на пути назад луч встретит объект, значит источник является тенью.

Для добавления отражений и прозрачности используется примерно одинаковая логика для получения отражения/прозрачности нужно возвращать при встрече луча с объектом кроме цвета треугольника умноженного на коэффициент отражения/преломления, нужно ещё добавить цвет отраженного луча, который получается созданием ещё одного луча, но с направлением отражения, само отражение вычисляется через нормаль к треугольнику

и направление базового луча.

Реализация алгоритма SSAA была взята из второй лабораторной.

Описание программы

Отрисовка на CPU:

```
void Scene::renderCPU () {
    double dw = 2.0 / (pointsWidth − 1);
    double dh = 2.0 / (pointsHeight − 1);
    double z = 1.0 / std::tan(angle ∗ std::acos(−1.0) / 360.0);
    Vec3 bz = normalize(pv − pc);
    Vec3 bx = normalize(prod(bz, createVec3(0.0, 0.0, 1.0)));
    Vec3 by = prod(bx, bz);
    for (int i = 0; i < pointsWidth; i++) {
        for (int j = 0; j < pointsHeight; j++) {
            Vec3 a = createVec3(−1.0 + dw ∗ i, (−1.0 + dh ∗ j) ∗
                pointsHeight / pointsWidth, z);
            Vec3 dir = normalize(multiple(bx, by, bz, a));
            points[(pointsHeight − 1 − j) ∗ pointsWidth + i] = ray(pc, dir,
                triangles.data(), lightSource, lightShade, triangles.size(), true
                , step, texture.data(), true);
        }
    }
}

void Scene::smoothingCPU () {
    int multiplier2 = multiplier ∗ multiplier;
    for (int y = 0; y < pointsHeight; y++) {
        for (int x = 0; x < pointsWidth; x++) {
            Vec3 mid = createVec3(0, 0, 0);
            for (int j = 0; j < multiplier; j++) {
                for (int i = 0; i < multiplier; i++) {
                    mid = mid + uchar4ToVec3(smooth[i + j ∗ pointsWidth
                        ∗ multiplier + x ∗ multiplier + y ∗ pointsWidth ∗
                        multiplier2]);
                }
            }
            points[x + pointsWidth ∗ y] = Vec3ToUchar4(mid / (multiplier2)
```

```
                    );
            }
        }
}
```

Отрисовка на GPU:

```
__global__ void renderGpuKernel (Vec3 pc, Vec3 pv, Triangle *triangles,
    uchar4 *points, int width, int height, double angle, Vec3 lightSource,
    Vec3 lightShade, int n, int step, uchar4 *texture) {
    int idx = blockDim.x * blockIdx.x + threadIdx.x;
    int idy = blockDim.y * blockIdx.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x;
    int offsety = blockDim.y * gridDim.y;

    double dw = 2.0 / (width - 1);
    double dh = 2.0 / (height - 1);
    double z = 1.0 / std::tan(angle * std::acos(-1.0) / 360.0);
    Vec3 bz = normalize(pv - pc);
    Vec3 bx = normalize(prod(bz, {0.0, 0.0, 1.0}));
    Vec3 by = prod(bx, bz);
    for (int i = idx; i < width; i += offsetx) {
        for (int j = idy; j < height; j += offsety) {
            Vec3 a = {-1.0 + dw * i, (-1.0 + dh * j) * height / width, z};
            Vec3 dir = normalize(multiple(bx, by, bz, a));
            points[(height - 1 - j) * width + i] = ray(pc, dir, triangles,
                lightSource, lightShade, n, true, step, texture, false); //
                меняем
                индексациючтобынеполучитьперевернутоеизображение
        }
    }
}

__global__ void smoothingGpuKernel (uchar4 *points, uchar4 *smoothPoints
    , int width, int height, int multiplier) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int idy = blockIdx.y * blockDim.y + threadIdx.y;
    int offsetx = blockDim.x * gridDim.x;
    int offsety = blockDim.y * gridDim.y;
```

```
27
28      int mult = multiplier * multiplier;
29      for (int y = idy; y < height; y += offsety) {
30          for (int x = idx; x < width; x += offsetx) {
31              Vec3 mid = createVec3(0, 0, 0);
32              for (int j = 0; j < multiplier; j++) {
33                  for (int i = 0; i < multiplier; i++) {
34                      mid = mid + uchar4ToVec3(smoothPoints[i + j * width
                            * multiplier + x * multiplier + y * width * mult]);
35                  }
36              }
37              points[x + width * y] = Vec3ToUchar4(mid / (mult));
38          }
39      }
40  }
41
42  void Scene::drawGPU () {
43      uchar4 *gpuTexture;
44      gpuErrorCheck(cudaMalloc(&gpuTexture, textureWidth * textureHeight
            * sizeof(uchar4)));
45      uchar4 *gpuPoints;
46      gpuErrorCheck(cudaMalloc(&gpuPoints, pointsWidth * pointsHeight *
            sizeof(uchar4)));
47      uchar4 *gpuSmooth;
48      gpuErrorCheck(cudaMalloc(&gpuSmooth, smoothWidth * smoothHeight
            * sizeof(uchar4)));
49      Triangle *gpuFigures;
50      gpuErrorCheck(cudaMalloc(&gpuFigures, triangles.size() * sizeof(
            Triangle)));
51      gpuErrorCheck(cudaMemcpy(gpuFigures, triangles.data(), triangles.size()
            * sizeof(Triangle), cudaMemcpyHostToDevice));
52      gpuErrorCheck(cudaMemcpy(gpuTexture, texture.data(), textureWidth *
            textureHeight * sizeof(uchar4), cudaMemcpyHostToDevice));
53      renderGpuKernel<<<1, 256>>>(pc, pv, gpuFigures, gpuSmooth,
            smoothWidth, smoothHeight, angle, lightSource, lightShade, triangles.
            size(), step, gpuTexture);
54      gpuErrorCheck(cudaGetLastError());
55      smoothingGpuKernel<<<256, 256>>>(gpuPoints, gpuSmooth,
```

```
            pointsWidth, pointsHeight, multiplier);
56      gpuErrorCheck(cudaGetLastError());
57      points.resize(pointsWidth * pointsHeight);
58      gpuErrorCheck(cudaMemcpy(&points[0], gpuPoints, pointsWidth *
            pointsHeight * sizeof(uchar4), cudaMemcpyDeviceToHost));
59      gpuErrorCheck(cudaFree(gpuPoints));
60      gpuErrorCheck(cudaFree(gpuSmooth));
61      gpuErrorCheck(cudaFree(gpuFigures));
62      gpuErrorCheck(cudaFree(gpuTexture));
63  }
```

Результаты

Замеры времени работы программы с различными конфигурациями (время указано в микросекундах).

Конфигу-рации / Чис-ло лучей	GPU<256, 256>	CPU
307 200	2 847	27 005
2 764 800	17 480	236 277

Кадры:

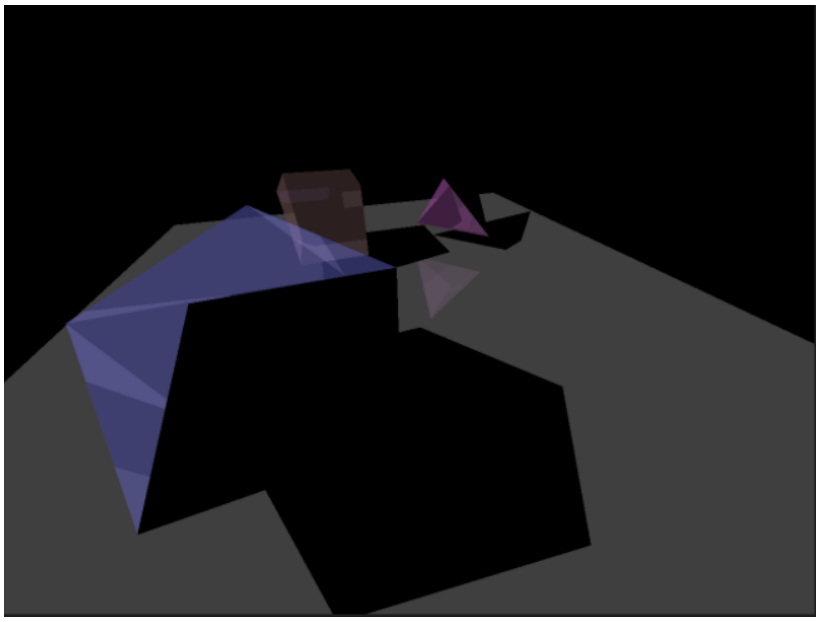

Рис. 1: Кадр 1

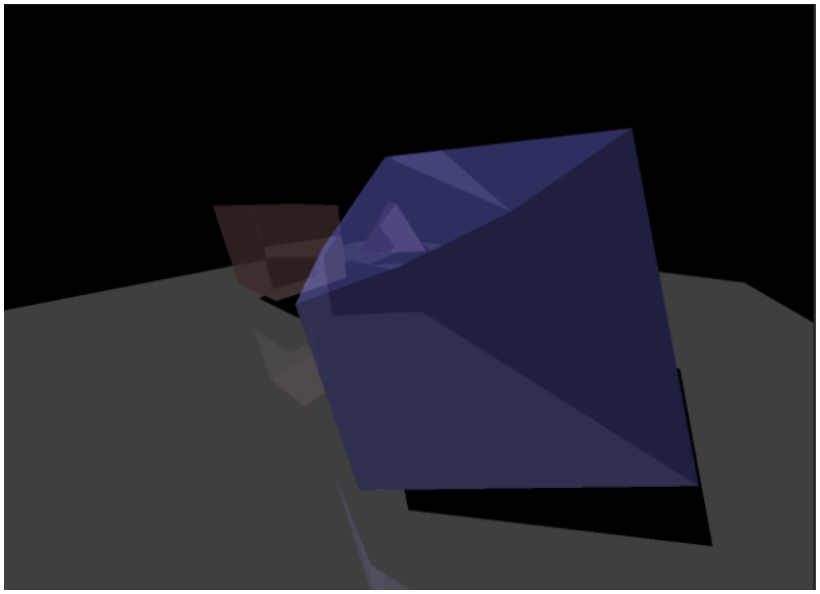

Рис. 2: Кадр 2

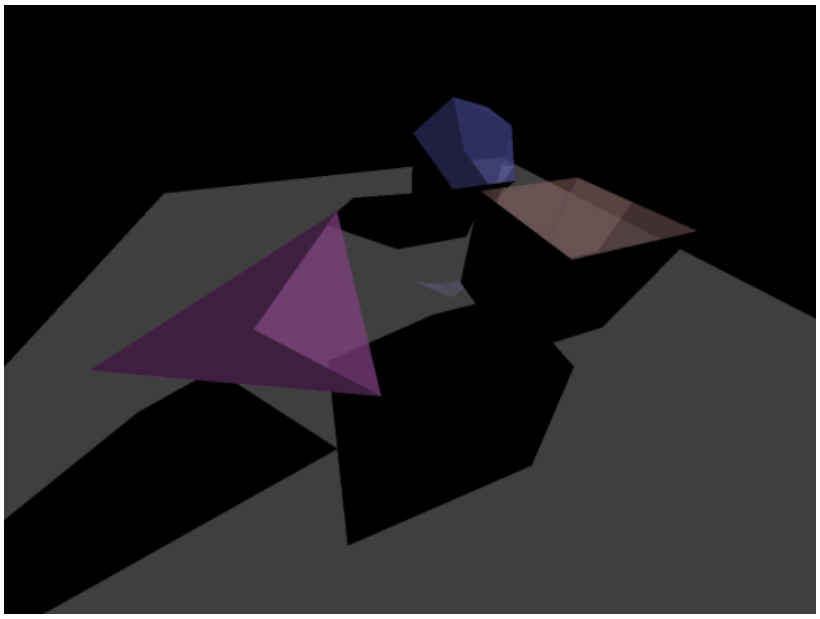

Рис. 3: Кадр 3

Выводы

Суть технологии трассировки лучей — просчет поведения луча света при преломлении и отражении от моделируемого объекта. При этом в расчет берутся как интенсивность виртуального луча (освещенность), так и его взаимодействие с остальными объектами, другими лучами и дополнительными источниками света. В результате этого мы можем наблюдать изобра-

жения, максимально приближённые к тому, что мы привык видеть в реальной жизни. Минусом же этой технологии является высокое требование к мощности ЭВМ.