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.

Выполнил: Д.В. Коростелев
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2022

Условие

Цель работы:

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

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:

$$r_c(t) = r_c^0 + A_c^r sin(\omega_c^r \cdot t + p_c^r)$$

$$z_c(t) = z_c^0 + A_c^z sin(\omega_c^z \cdot t + p_c^z)$$

$$\varphi_c(t) = \varphi_c^0 + \omega_c^\varphi t$$

$$r_n(t) = r_n^0 + A_n^r sin(\omega_n^r \cdot t + p_n^r)$$

$$z_n(t) = z_n^0 + A_n^z sin(\omega_n^z \cdot t + p_n^z)$$

$$\varphi_n(t) = \varphi_n^0 + \omega_n^\varphi t$$

Требуется реализовать алгоритм обратной трассировки лучей (http://www.ray-tracing.ru/) 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 поддерживать следующие ключи запуска:
--cpu Для расчетов используется только центральный процессор
--gpu Для расчетов задействуется видеокарта
--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.

Вариант 8:

Гексаэдр, Октаэдр, Икосаэдр

Программное и аппаратное обеспечение

- Графический процессор NVIDA GeForce GTX 1050

Графическая память	2 Gb
Разделяемая память на блок	48 Mb
Константная память	64 Mb
Количество регистров на блок	65536
Максимальное количество блоков на процессор	32
Максимальное количество потоков на блок	1024
Количество мультипроцессоров	5

- Процессор Intel Core i7-7700HQ 4x 2.808ГГц

Количество ядер	4
Количество потоков	8
Базовая тактовая частота	2.8 GHz
Максимальная тактовая частота	3.8 GHz
Кеш-память	6 Mb

- Оперативная память DDR4-SODIMM

Объем памяти	8 Gb
Частота	2400 MHz
Форм-фактор	SODIMM
Количество плашек	2

- SSD и HDD накопители

Объем SSD накопителя	128 Gb
Объем HDD накопителя	1 Tb

- Программное обеспечение

Операционная система	Windows 10 Pro
Средство разработки на CUDA (IDE)	Microsoft Visual Studio 2019
Компиляторы	MSVC 2019
Версия CUDA Toolkit	11.4.2
Дополнительный текстовый редактор	Notepad++

Метод решения и описание программы

Из камеры будем пускать лучи на сцену и сначала проверять пересекается ли луч с каким-либо полигоном, если пересечения нет, то закрашиваем точку в черный цвет. Если пересечение с полигоном есть, то нужно проверить – есть ли прямой луч до какого-либо источника света, если прямой луч имеется, то рассчитываем свет по модели Фонга. Если алгоритм на ЦПУ, то мы последовательно для каждого кадра, каждого пикселя и каждого луча считаем цвет пикселя. Если на ГПУ – то каждый поток каждого блока рассчитывает значения цвета пикселя параллельно друг с другом. Также, чтобы избежать эффекта ребристости – используем алгоритм SSAA (реализован как на гпу, так и на цпу).

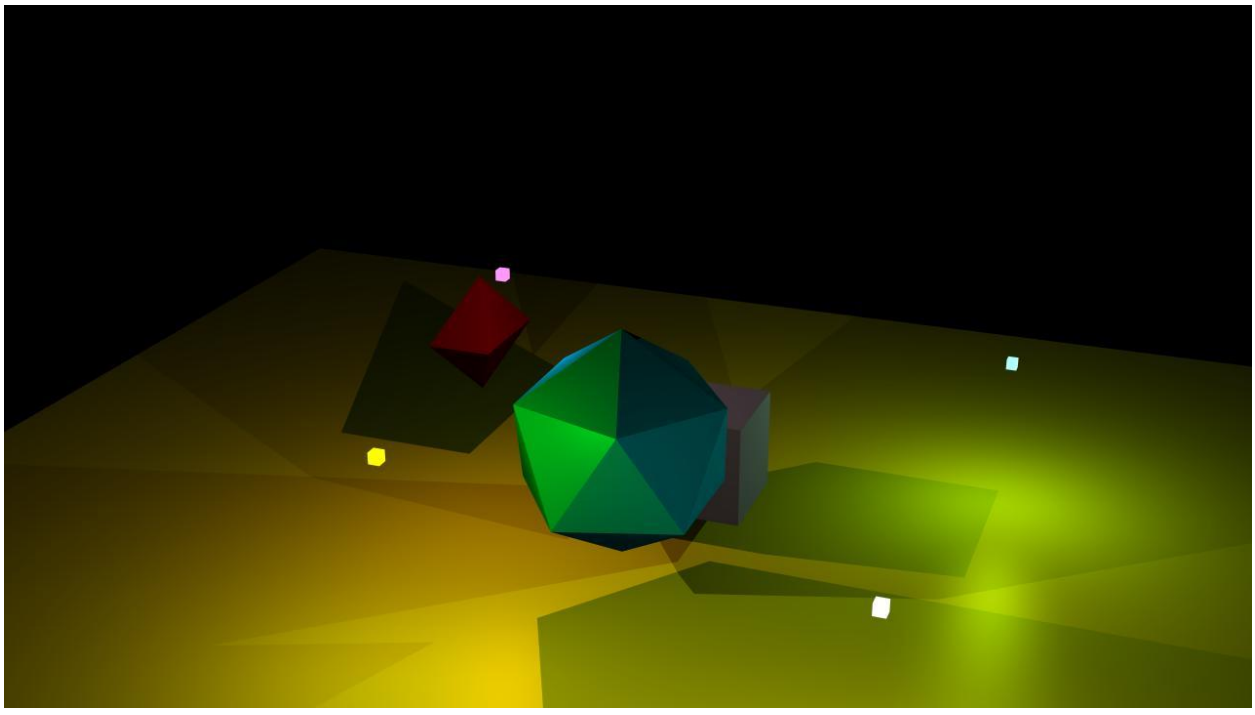
Во входных параметрах можно задавать положение камеры, характер ее движения, точку куда она смотрит и как перемещается со временем. Можно указывать положение фигур, их размер, цвет, характеристики цвета стороны при попадании света на полигон, а также можно задать положение и цвет пола на сцене.

Бенчмарки [время на гпу(время на цпу)]

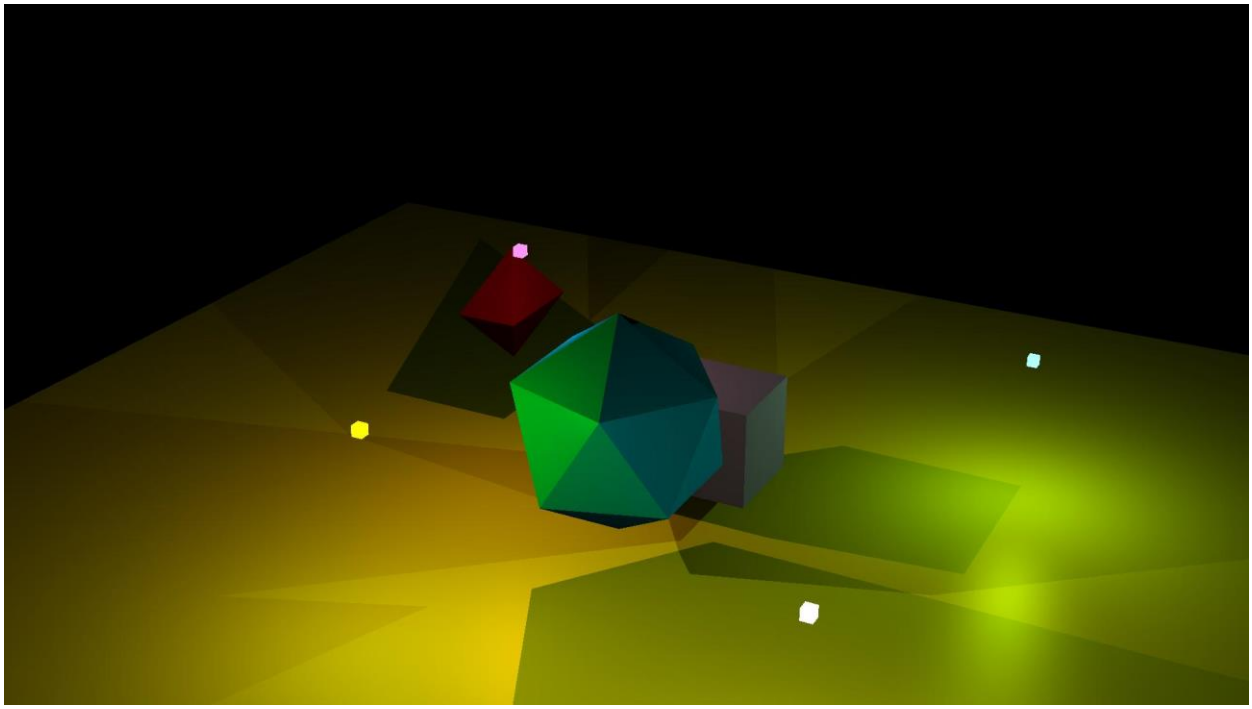
Кол-во источников света / Размерн-ть одного кадра	2	3	4
720x360	~0.1сек (~4 сек)	~0.2 сек (~5.2 сек)	~0.2 сек (~7 сек)
1000x500	~0.5сек (~10 сек)	~0.7 сек (~13 сек)	~0.9 сек (~16 сек)
1280x720	~1.2 сек (~30 сек)	~1.5 сек (~16 сек)	~2.3 сек (~20 сек)
1920x1080	~2 сек (~1.2 мин)	~2.5сек (~3 мин)	~3 сек (9 мин)
1280x720 (SSAA 2)	~4 сек (-)	~6 сек (-)	~10 сек (-)
1920x1080 (SSAA 2)	~7 сек (-)	~11 сек (-)	~17 сек (-)
1280x720 (SSAA 4)	~30 сек (-)	~44 сек (-)	~1 мин (-)

Результаты

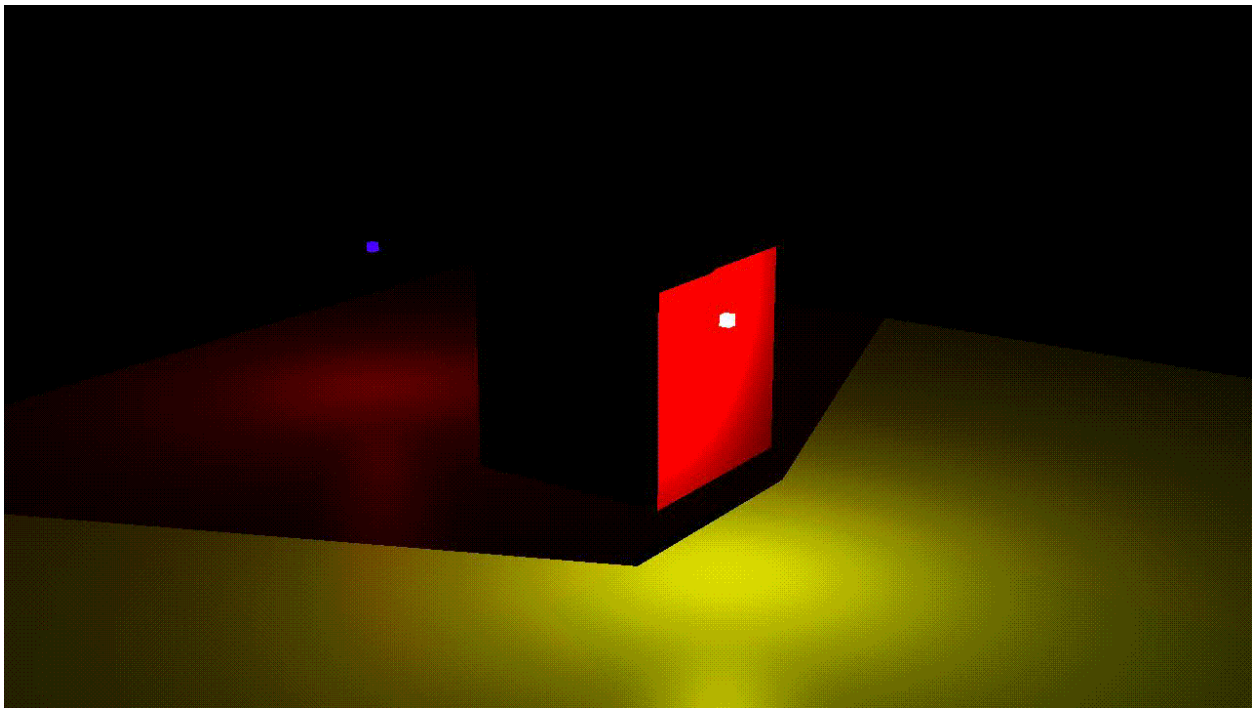
SSAAx2



SSAAx4



SSAA x1



Выводы

Курсовая работа выдалась крайне интересной, познавательной и презентабельной, однако в ходе ее выполнения столкнулся с рядом трудностей, связанных с расчетом цвета по Фонгу, но стоит отметить, что в итоге после дебага программы на цпу распараллеливание не заняло много времени и позволило значительно увеличить скорость работы программы – прирост в производительности оказался колоссальным, так как HD сцена с SSAA 2 состоящая из 60 кадров рендерится в считанные секунду, в то время, как на ЦПУ это время доходит до нескольких десятков минут.