

**Разработка системы визуализации фотореалистичных
трехмерных сцен в реальном времени с использованием GPU**

**Development of a real-time photorealistic three-dimensional scene
visualization system using a GPU**

Иванов Тимофей Андреевич 9-1, Урсова Софья Александровна 9-1,
Амбросовская Дарья Викторовна 9-2, Григорович Вячеслав Дмитриевич 10-5,
Писарев Евгений Александрович 10-5, Синяков Степан Евгеньевич 10-5,
Мосягин Олег Сергеевич 11-5

г. Санкт-Петербург, Государственное бюджетное общеобразовательное учреждение
"Санкт-Петербургский губернаторский физико-математический лицей № 30"








Руководитель: Галинский Виталий Александрович,
преподаватель информатики и программирования физико-математического лицея № 30,
руководитель группы компьютерной графики, зам. директора по ИТ

2019 год

Проблема рендеринга (процесс получения изображения с помощью компьютерной программы) фотореалистичных сцен в реальном времени является одной из самых актуальных проблем компьютерной графики на сегодняшний день. Она является значимой для таких сфер как кинематограф, симуляция некоторых физических явлений, развлекательная сфера и др. Целью проекта является разработка программного обеспечения, позволяющего создавать трехмерные сцены и выводить фотореалистичные изображения с высокой частотой кадров (FPS).

Язык

Для создания трёхмерных сцен авторами был разработан собственный язык программирования. Языковой модуль состоит из трёх подмодулей: сканера, синтаксического анализатора и виртуальной машины. Работа начинается со сканера. Он предназначен для разбиения исходного кода программы на нашем языке на синтаксические единицы – лексемы (токены). Токен – структура данных, хранящая информацию о считанной лексеме. Существует несколько видов токенов: имя, служебное слово, число, оператор, строка, символ.

-  - имя
-  - служебное слово
-  - число
-  - оператор
-  - строка
-  - символ
-  - комментарий (не считывается)

```
while (1 > 0)
{
    a = "hello";
}
/* comment */
```

После идёт синтаксический анализатор. Он создаёт команды для последующего их выполнения виртуальной машиной. Команды состоят из служебных слов (if, else, while, for, switch, case, default, break, continue, return, function, operator) и выражений. Выражение – это дерево операций, выполнение которых возвращает результат этого выражения и совершает побочные действия.

Результатом выражения может быть число, строка, другое выражение или переменная. Побочным действием может быть изменение переменной, создание нового объекта на сцене и т.п.

После препроцессинга начинает работу виртуальная машина. Она выполняет команды, полученные после обработки информации синтаксическим анализатором. В ходе работы

языковой модуль получит данные о сцене и после команды отрисовки передаст их модулю рендеринга.

Язык динамически типизированный, т.е. тип переменной определяется в момент присваивания ей значения. Переменные хранят выражения, в которых могут содержаться функции, строки, числа и ссылки на переменные. У переменных можно создавать поля и после создания обращаться к ним. Это позволяет хранить большие объемы данных в одной структуре.

У функций нет идентификаторов. Чтобы в последствие её использовать, необходимо присвоить функцию в переменную. Для её вызова после имени переменной необходимо написать круглые скобки и передать параметры, если это необходимо.

В языке реализована система контекстов (зона видимости) – все переменные, созданные в контексте функции, не связаны с переменными других контекстов. Из любой зоны видимости можно обратиться к глобальному контексту.

Существует возможность перегрузить операторы для некоторых переменных, создав поле оператора с его новой функцией.

Пример кода на нашем языке:

```
include("stl\\rt\\trg_raytrace.trg");

std::setrndmode("cuda");

Scene = scene();

BuildFence = function( s = 15 ~ "num", gr = 6 ~ "num", a = #vec3(0) ~ "vec3" )
{
    #Scene.Add(#fence(a + #vec3(0, 1, 0), s, 2, gr, #material::blue_plastic()));
    i = 0;
    an = #mth::Degree2Radian(90);
    P1 = #mth::Degree2Radian(360) / gr;
    while (i < gr)
    {
        #Scene.Add(#conus(#vec3(an * s, 1, s), 1, 3, material::red_plastic()));
        an += P1;
        i++;
    }
};

t = texture("chess.g24", 2);
a = material::white_plastic();
a.SetTexture(t, 1);
Scene.Add(tor(vec3(0, 0.75, 1), 0.75, 5, material::green_plastic()));
BuildFence(15, 6);

Scene.Add(model("BIN\\MODELS\\cow.g3dm"), enviroment(0, 1), material::red_plastic()));

Scene.Add(plane(vec3(0, 1, 0), 0, material::silver()));

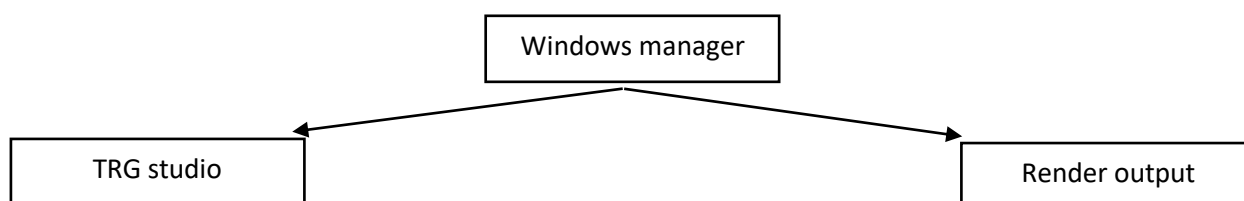
p = dir_light(vec3(-7, 5, 8), vec3(1, 1, 1));
Scene.Add(p);

while (1)
{
    Scene.CamSet(1920, 1080, vec3(30, 10, 30 * mth::sin(i / 180 * 3.14)),
                vec3(0, 0, 0), vec3(0, 1, 0));
    Scene.Render();
    i += 3;
}
```

Пользовательский интерфейс.

Авторами была проделана работа по созданию среды разработки и окна вывода под данный проект. Целью было написать пользовательский интерфейс, в котором было бы можно редактировать файлы, обозревать решение, получать ошибки и выводить полученный результат. В процессе создания было принято решения разработать интерфейс близкий к visual studio, только с теми возможностями, которые необходимы.

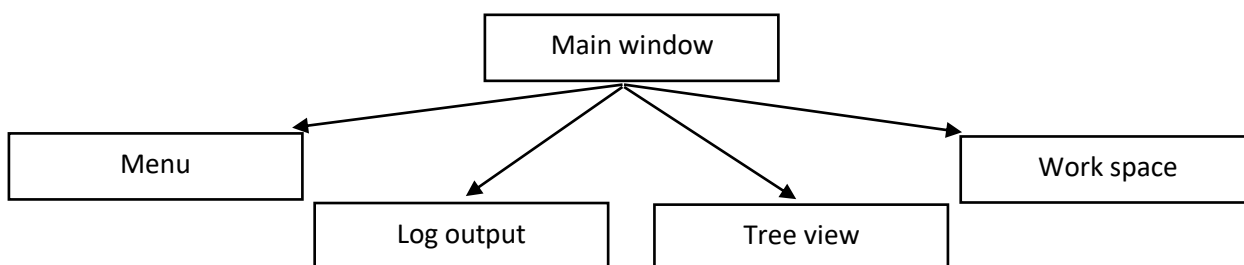
Для создания системы интерфейсов авторами был использован Windows application programming interface. Схема, описывающая эту систему, представлена ниже:



Во главе всей системы стоит Windows manager. Он хранит в себе окно среды разработки (TRG studio) и окно вывода изображения (Render output) и запускает цикл обработки сообщений.

Среда разработки

Схема, описывающая среду разработки, представлена ниже:

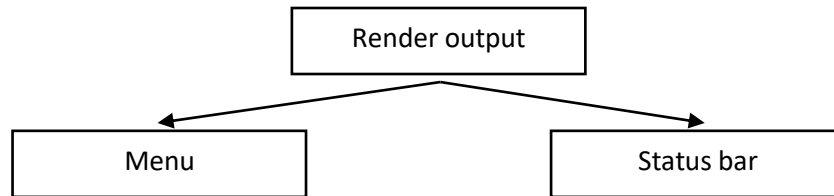


При запуске проекта открывается главное окно среды разработки (Main window). Оно содержит в себе следующие элементы:

- меню. С его помощью можно осуществлять такие действия, как открытия, сохранение файлов, сборка, запуск проектов и т.д. Оно было создано с помощью системы ресурсов – бинарных файлов, добавляемых в исполняемый файл при компоновке программы. Доступ к некоторым элементам меню осуществляется по горячим клавишам.
- окно редактирования (Work space). В нём можно изменять файлы проекта, переключаться по файлам, сохранять их и др. Окно редактирования является диалоговым окном (тоже ресурс WinAPI).
- дерево проекта. Оно позволяет отобразить всю структуру проекта и открыть, закрывать, удалять файлы, в него включающиеся.
- окно журнала ошибок (Log output). Сюда выводятся ошибки, найденные при сборке проекта, написанного на TGL.

Окно вывода.

Схема, описывающая окно вывода, представлена ниже:



При запуске проекта *.tgl из реализованной авторами среды разработки открывается окно вывода (Output window). Оно содержит в себе следующие элементы:

- меню. С его помощью можно осуществлять такие действия, как сохранение изображений. Оно было создано аналогично меню главного окна.
- строка состояния. Отображает позицию мышки в координатах экрана, позицию в текстуре, цвет пикселя, что позволяет получить наиболее точную информации о результате работы выполнения языка, помимо этого также отображается краткая сводка о возможностях элементов при наведении на них.

Рендеринг

Трассировка лучей

Основы алгоритма.

Построение кадра выполняется методом обратной трассировки лучей. Для того чтобы получить цвет пикселя итоговой картинки наша система пускает луч в направлении на заданный пиксель, ищет пересечения этого луча с объектами сцены.

Пересечение с треугольником.

Проверка принадлежности точки пересечения луча и плоскости треугольнику.

Сначала проверяется пересечение луча и плоскости.

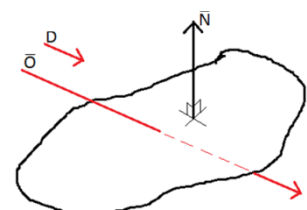
$$\vec{N} = (A, B, C)$$

$$\vec{P} = (x, y, z)$$

$$F(x, y, z) = A \cdot x + B \cdot y + C \cdot z + D = 0$$

$$A \cdot (Ox + Dx \cdot t) + B \cdot (Oy + Dy \cdot t) + C \cdot (Oz + Dz \cdot t) + D = 0$$

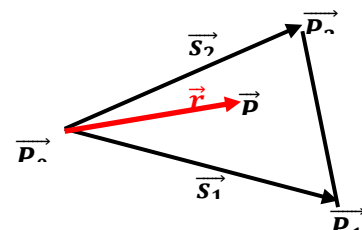
$$t = \frac{-(A \cdot Ox + B \cdot Oy + C \cdot Oz + D)}{A \cdot Dx + B \cdot Dy + C \cdot Dz}, \quad t = \frac{-(\vec{N} \cdot \vec{P} + D)}{\vec{N} \cdot \vec{D}}$$



Если $t < 0$, то пересечений нет.

Если треугольник невырожденный, то \vec{s}_1 и \vec{s}_2 образуют базис

на плоскости (все точки лежат в одной плоскости).



Разложим вектор \vec{r} в базисе (\vec{s}_1, \vec{s}_2) :

$$\vec{r} = \vec{s}_1 \cdot u + \vec{s}_2 \cdot v$$

Если $0 \leq u \leq 1$ и $0 \leq v \leq 1$ и $u + v \leq 1$, то точка \vec{P} лежит в треугольнике
Найдем u :

$$u = \frac{u'}{|\vec{s}_1|}$$

выразим из u'' :

$$u = \frac{u'}{|\vec{s}_1|}$$

$$u' = \frac{u''}{\cos(\vec{s}_1, \vec{s}_1')} = \frac{u''}{\frac{(\vec{s}_1 \cdot \vec{s}_1')}{|\vec{s}_1| \cdot |\vec{s}_1'|}}$$

здесь:

$$u'' = \frac{\vec{r} \cdot \vec{s}_1'}{|\vec{s}_1'|}$$

находим \vec{s}_1' $[\vec{a} \times (\vec{b} \times \vec{c}) = \vec{b} \cdot (\vec{a} \cdot \vec{c}) - \vec{c} \cdot (\vec{a} \cdot \vec{b})]$:

$$\vec{s}_1' = \vec{s}_2 \times (\vec{s}_1 \times \vec{s}_2) = \vec{s}_1 \cdot (\vec{s}_2 \cdot \vec{s}_2) - \vec{s}_2 \cdot (\vec{s}_2 \cdot \vec{s}_1)$$

$$u = \frac{u'}{|\vec{s}_1|} = \frac{\frac{u''}{\frac{(\vec{s}_1 \cdot \vec{s}_1')}{|\vec{s}_1| \cdot |\vec{s}_1'|}}}{|\vec{s}_1|} = \frac{u'' \cdot |\vec{s}_1| \cdot |\vec{s}_1'|}{|\vec{s}_1| \cdot (\vec{s}_1 \cdot \vec{s}_1')} = \frac{u'' \cdot |\vec{s}_1'|}{(\vec{s}_1 \cdot \vec{s}_1')}$$

$$u = \frac{u'' \cdot |\vec{s}_1'|}{(\vec{s}_1 \cdot \vec{s}_1')} = \frac{\vec{r} \cdot \vec{s}_1'}{|\vec{s}_1'|} \cdot \frac{|\vec{s}_1'|}{(\vec{s}_1 \cdot \vec{s}_1')} = \frac{\vec{r} \cdot \vec{s}_1'}{(\vec{s}_1 \cdot \vec{s}_1')} =$$

$$= \frac{((\vec{r} \cdot \vec{s}_1) \cdot (\vec{s}_2 \cdot \vec{s}_2) - (\vec{r} \cdot \vec{s}_2) \cdot (\vec{s}_2 \cdot \vec{s}_1))}{(\vec{s}_1 \cdot \vec{s}_1) \cdot (\vec{s}_2 \cdot \vec{s}_2) - (\vec{s}_1 \cdot \vec{s}_2) \cdot (\vec{s}_2 \cdot \vec{s}_1)} =$$

$$= \frac{((\vec{r} \cdot \vec{s}_1) \cdot (\vec{s}_2 \cdot \vec{s}_2) - (\vec{r} \cdot \vec{s}_2) \cdot (\vec{s}_2 \cdot \vec{s}_1))}{\vec{s}_1^2 \cdot \vec{s}_2^2 - (\vec{s}_1 \cdot \vec{s}_2)^2}$$

$$\vec{r} = \vec{P} - \vec{P}_0:$$

$$u = \vec{P} \cdot \frac{(\vec{s}_1 \cdot \vec{s}_2^2 - \vec{s}_2 \cdot (\vec{s}_1 \cdot \vec{s}_2))}{\vec{s}_1^2 \cdot \vec{s}_2^2 - (\vec{s}_1 \cdot \vec{s}_2)^2} - \vec{P}_0 \cdot \frac{(\vec{s}_1 \cdot \vec{s}_2^2 - \vec{s}_2 \cdot (\vec{s}_1 \cdot \vec{s}_2))}{\vec{s}_1^2 \cdot \vec{s}_2^2 - (\vec{s}_1 \cdot \vec{s}_2)^2}$$

или

$$u = (\vec{P} \cdot \vec{U}_1) - u_0$$

здесь:

$$u_0 = (\vec{P}_0 \cdot \vec{U}_1), \vec{U}_1 = \frac{\vec{s}_1 \cdot \vec{s}_2^2 - \vec{s}_2 \cdot (\vec{s}_1 \cdot \vec{s}_2)}{\vec{s}_1^2 \cdot \vec{s}_2^2 - (\vec{s}_1 \cdot \vec{s}_2)^2}$$

Аналогично:

$$v = \vec{P} \cdot \frac{(\vec{s}_2 \cdot \vec{s}_1^2 - \vec{s}_1 \cdot (\vec{s}_1 \cdot \vec{s}_2))}{\vec{s}_1^2 \cdot \vec{s}_2^2 - (\vec{s}_1 \cdot \vec{s}_2)^2} - \vec{P}_0 \cdot \frac{(\vec{s}_2 \cdot \vec{s}_1^2 - \vec{s}_1 \cdot (\vec{s}_1 \cdot \vec{s}_2))}{\vec{s}_1^2 \cdot \vec{s}_2^2 - (\vec{s}_1 \cdot \vec{s}_2)^2}$$

$$v = (\vec{P} \cdot \vec{V}_1) - v_0$$

$$v_0 = (\vec{P}_0 \cdot \vec{V}_1), \vec{V}_1 = \frac{\vec{s}_2 \cdot \vec{s}_1^2 - \vec{s}_1 \cdot (\vec{s}_1 \cdot \vec{s}_2)}{\vec{s}_1^2 \cdot \vec{s}_2^2 - (\vec{s}_1 \cdot \vec{s}_2)^2}$$

Для поиска пересечения луча с треугольником необходимо хранить:

\vec{N}, \vec{D} – для уравнения плоскости;

$\vec{U}_1, u_0, \vec{V}_1, v_0$ – для поиска u и v .

Замечание. Барицентрические координаты:

Выше получили: $\vec{r} = \vec{s}_1 \cdot u + \vec{s}_2 \cdot v$

Подставляем радиус-вектора точек:

$$\vec{P} - \vec{P}_0 = (\vec{P}_1 - \vec{P}_0) \cdot u + (\vec{P}_2 - \vec{P}_0) \cdot v$$

Раскрываем и решаем относительно \vec{P} :

$$\vec{P} = \vec{P}_0 + \vec{P}_1 \cdot u - \vec{P}_0 \cdot u + \vec{P}_2 \cdot v - \vec{P}_0 \cdot v$$

$$\vec{P} = \vec{P}_0 \cdot (1 - u - v) + \vec{P}_1 \cdot u + \vec{P}_2 \cdot v$$

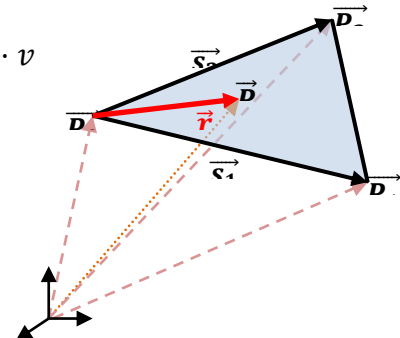
или

$$\vec{P} = \vec{P}_0 \cdot w + \vec{P}_1 \cdot u + \vec{P}_2 \cdot v$$

(w, u, v) — барицентрические координаты

Их можно использовать как интерполирующие коэффициенты для всевозможных атрибутов вершин (нормали, текстурные координаты, цвет и т.п.), заменяя точки $\vec{P}_0, \vec{P}_1, \vec{P}_2$ на соответствующие параметры.

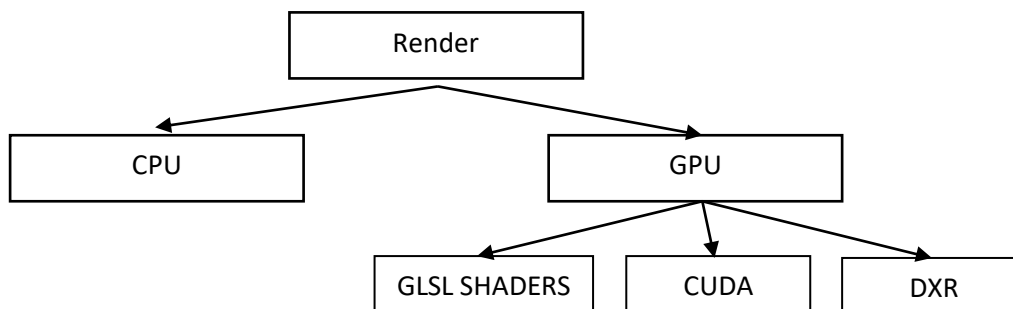
$$\vec{P} = \vec{P}_0 \cdot w + \vec{P}_1 \cdot u + \vec{P}_2 \cdot v$$



Рендер.

Виды рендера.

Авторами реализовано 2 вида рендеринга: CPU и GPU.

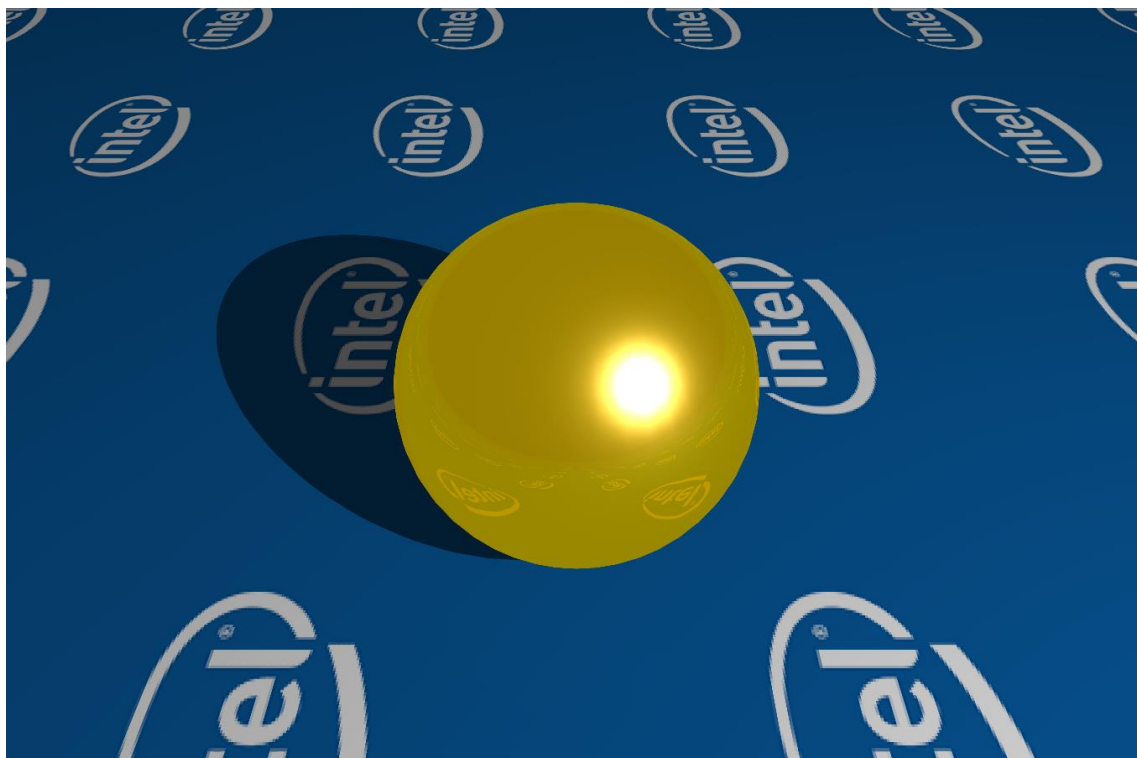


CPU рендер представляет собой оптимизированный алгоритм с использованием многопоточности, что позволяет ускорить просчёт каждого пикселя, так как потоки выполняются параллельно, независимо друг от друга.

Рендеринг на GPU может проходить одним из трёх способов: при помощи CUDA, GLSL шейдеров и DXR.

Модификаторы.

Для улучшения визуальной составляющей авторы добавили сглаживание и текстурирование. Для сглаживания было проведено исследование и был выбран алгоритм MFAA в связи с его оптимизированностью и скоростью работы. Реализованы алгоритмы anti-aliasing'a (сглаживания) supersampling и jittering.



Освещение.

Глобальное освещение.

Если луч пересек треугольник, начинается процесс расчета цвета данной точки треугольника. Реализована система освещения BRDF, о которой будет рассказано в следующем абзаце. Для расчета цвета используются материалы (набор коэффициентов, необходимых для расчета цвета), считаются тени и накладываются текстуры. В проекте создана библиотека материалов. Затенение вычисляется, учитывая перекрытие точки объекта от источников света другими объектами. При наложении текстур на объекты используется интерполяция текстурных координат вершин треугольника по барицентрическим координатам. В программе реализовано отражение и преломление: к цвету данной точки прибавляется результат данного алгоритма для отраженного и преломленного луча.

BRDF (Bidirectional reflectance distribution function)

Система освещения BRDF является физически корректной, благодаря чему изображение получается реалистичным. В качестве одного из ключевых параметров объекта является его гладкость. При помощи теории микрофасетов, учитывая этот параметр и металлические свойства объекта (поддерживается металл и диэлектрик), осуществляется распределение светового потока на рассеянное (diffuse) и зеркальное (specular) освещение.

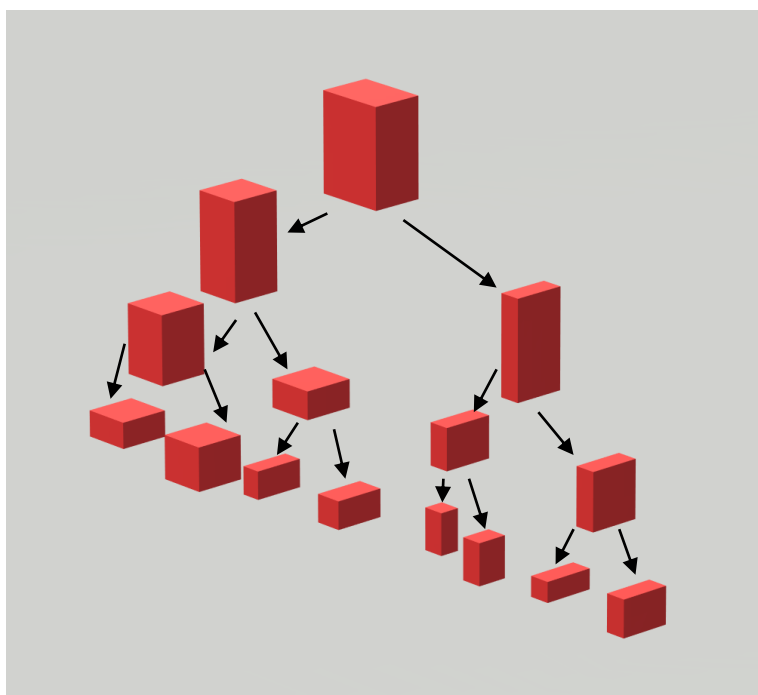
Оптимизация

Регулярная сетка (voxels).

В проекте реализован алгоритм регулярной сетки. Ее особенность регулярной сетки в том, что всё пространство разбивается кубы одинакового размера, называемые вокселями. В каждом вокселе хранятся ссылки на треугольники, которые в нём полностью или частично содержатся. При поиске пересечений используется аналог алгоритма Брезенхема, для поиска пересечения не со всеми треугольниками сцены, а только с теми, которые находятся внутри вокселя, пересечённого данным лучём. Это позволяет во много раз ускорить вычисления за счёт уменьшения количества объектов, с которыми необходимо вычислять пересечения. Особенность сетки с подразбиением в том, что всё пространство разбивается на кубы, которые в свою очередь тоже разбиваются на кубы меньшего размера. Это позволяет быстрее пересекать луч с некоторыми объектами, проверяя только треугольники, принадлежащие данной части куба, что существенно ускоряет процесс трассировки лучей.

Двоичное дерево треугольников (kd-tree).

Для оптимизации отрисовки объектов, состоящих из большого количества треугольников, для них строится двоичное дерево ограничивающих параллелепипедов. Суть этого дерева в том, что сначала производится алгоритм построения дерева, а потом в пересечении с лучом нуждаются не все треугольники сцены. Дерево строится следующим образом: для всех объектов на сцене считается ограничивающий прямоугольный параллелепипед. Потом считается самая длинная координата этого параллелепипеда, и он делится на две части, в каждой из которых примерно одинаковое количество треугольников. Потом каждая для каждой из полученных половин прямоугольного параллелепипеда повторяется та же операция. Алгоритм продолжается до тех пор, пока в каждом из параллелепипедов не остаётся константной количество треугольников (обычно четыре). В результате все треугольники сцены находятся в дереве, с которым можно быстрее найти пересечение луча, чем со всеми треугольниками сцены самими по себе. Вместо простого перебора треугольников луч сначала пересекается с основным прямоугольным параллелепипедом. Если он не пересекается, то и искать пересечения с треугольниками нет смысла, потому что пересечений нет. Если пересёкся, то далее луч пересекается с каждой из половинок этого параллелепипеда аналогичным образом. Таким образом, возникает возможность не пересекать луч с большей частью треугольников сцены, что позволяет значительно оптимизировать трассировку лучей. Например, на сцене из 12000 треугольников дерево ускорило пересечение в более чем 23 раза.



Вычисления на видеокарте

CUDA.

Для переноса вычислений на GPU используется CUDA API. На современных видеокартах количество ядер превышает тысячу. В процессорах их на порядки меньше. Алгоритм обратной трассировки лучей легко распараллеливается на видеокарте, так как цвет каждого пикселя считается независимо от всех остальных. Благодаря этому кадр считается гораздо быстрее.

Расчёт отражений и преломлений на графическом процессоре был проблематичен, так как рекурсивные вычисления в значительной степени замедляют процесс построения кадра. Эту проблему мы решили заменой рекурсии на цикл с сохранением значений в очередь. Аналогично мы поступили с обходом дерева при поиске пересечений со сценой.

Вывод

В результате работы над проектом нам удалось написать программу, позволяющую создавать фотореалистичные изображения в реальном времени на нашем языке программирования. В ходе исследования нами были изучены множество алгоритмов, связанных с обратной трассировкой лучей, и методы их реализации. В будущем авторы планируют добавить новые способы оптимизации вычислений, распределенные вычисления по локальной сети, улучшить физическое взаимодействие объектов и распространить данный проект.

Литература

- Matt Pharr, Wenzel Jacob, Greg Humphreys. "Physically based rendering from theory to Implementation", 3rd ed., Morgan Kaufmann, Elsevier, Book Aid International, 2017.
- Tomas Akenine Moller, Eric Haines, Naty Hoffman, Angelo Pesce, Michal Iwanicki, Sebastien Hillaire. "Real-time rendering", 4th ed., CRC Press, Taylor & Francis Group, A K Peters Book, 2018.
- David Wolff, "OpenGL Shading Language Cookbook", Packt Publishing, 2018.
- А. В. Боресков, "Параллельные вычисления на GPU. Архитектура и программная модель CUDA: учеб. пособие", МюЖ Издательство Московского университета, 2012.