

Задание #2: Ведьмин Шар

Авторы задания: Фролов В. А., Санжаров В.А.



Рисунок 1. Пример выполненного задания. 20 баллов база. +1 балл за текстуры. +5 за реалистичность сцены. +4 непредусмотренный бонус проверяющего за реалистичную симуляцию огня. Итого 30 баллов.

Аннотация

Цель задания - закрепить на практике основы построения изображений при помощи трассировки лучей. Основной упор задания: понимание фундаментальных принципов построения изображений в кино и мультипликации, позволяющих достигать необходимого уровня гибкости и реалистичности. В процессе выполнения задания студенты сформируют следующие навыки:

- работа с алгоритмами реалистичной компьютерной графики;
- изучение алгоритма трассировки лучей;
- изучение основ Монте-Карло интегрирования;

Обязательная часть (20 баллов)

Необходимо реалистично отобразить т. н. магический кристалл (ведьмин шар) в виде хрустального шара или многогранника (в зависимости от вашего варианта). К визуализации шара предъявляются несколько основных требований:

1. Материал кристалла должен вести себя аналогично стеклу, то есть преломление и отражение должны быть реализованы с **учётом формул Френеля**. Обязательна реализация полного внутреннего отражения.

2. Внутри кристалла должна в тумане либо в темноте проявляться некоторая сцена (в зависимости от вашего варианта). Сцена **должна быть полупрозрачной**, имитируя нечёткие видения предсказательницы. Не допускается резких переходов между материалом шара и сценой внутри него.
3. Кристалл должен стоять на подставке. В минимальном варианте задания --- квадратной или цилиндрической площадке.
4. В сцене **должно быть хотя бы 2 источника света**. Источники должны отбрасывать блики на стекле либо иметь размер и отражаться в нём.
5. Рендеринг **не должен занимать больше 1 минуты**. В случае если вы опасаетесь, что при значительном количестве дополнительно реализованных эффектов скорость упадет ниже допустимого порога, необходимо реализовать прогрессивный (постепенный, уточняющий картинку с каждым проходом) рендеринг.
6. По умолчанию изображение должно рендериться в разрешении 512x512 (для изменения разрешения '-w 1024' в командной строке. Изображение должно быть квадратным).
7. Программа должна сохранять картинку в файл, имя которого формируется следующим образом: 322_fomichëv_v1v2: номер группы, нижнее подчёркивание, фамилия латиницей, нижнее подчёркивание, выбранные варианты исходя из кириллического написания Вашей фамилии.

Внимание! При невыполнении любого требования из базы баллы уполовиниваются! При невыполнении двух требований делиться на 4 и так далее. При этом необходимо помнить что одно из главных требований задания – **реалистичность сцены**. Абстракционизм не приветствуется!

Варианты задания

Ваш вариант задания формируется 2 путями:

1. Вы берёте порядковый номер в алфавите **первой** буквы в вашей фамилии, вычитаете 1 и берёте остаток от деления полученного числа **на 6**. Тем самым Вы получаете порядковый номер варианта формы кристалла.
2. Вы берёте порядковый номер в алфавите **второй** буквы в вашей фамилии, вычитаете 1 и берёте остаток от деления полученного числа **на 9**. Тем самым Вы получаете порядковый номер варианта сцены внутри кристалла.

1. Форма кристалла:

- 0) Шар;
- 1) Куб;
- 2) Четырёх-угольная пирамида;
- 3) Октаэдр;
- 4) Додэкаэдр;

5) Икосаэдр;

2. Сцена внутри кристалла:

- 0) Огонь или огненный шар (проще всего на основе Шума Перлина или Distance Aided Ray Marching <https://habr.com/ru/post/437714/>);
- 1) Разноцветный и/или светящийся дым (проще всего на основе Шума Перлина). В этом варианте можно получить от +2 до +5 баллов за реализацию электрических разрядов внутри кристалла.
- 2) Продолговатые полости либо вкрапления материала внутри стекла, выполненные из материала с другим показателем преломления (рис. 2). В этом единственном варианте можно нарушить общее требование в визуализации сцены внутри. Допустимы резкие переходы, кристалл выключен; Выполняющим этот вариант рекомендуется ознакомиться со следующей статьёй <https://blog.yiningkarlli.com/2019/05/nested-dielectrics.html>



Рисунок 2. Пример шаров с продолговатыми полостями или вкраплениями внутри стекла.

- 3) Дом или здание в тумане либо во тьме, но тогда хотя бы некоторые окна должны гореть. В этом варианте приветствуется реализация Г. 3. МГУ (от +2 до +5 баллов за реалистичность);
- 4) Космическое пространство со звёздами (за визуализацию космической станции или корабля можно получить дополнительно от +2 до +5 баллов);
- 5) Вырезка куса сцены из Minecraft (минимально-допустимое разрешение – 256^3); Рекомендуется для повышения реалистичности сцены добавить расчёт Ambient Occlusion.
- 6) Вода в любом виде, допустимы пейзажи с водой.
- 7) Ландшафт при помощи карты высот или полигонального меша (за деревья и другие объекты можно получить от +2 до +5 баллов).

- 8) Фрактальные поверхности (Мандельброт 3D, Фракталы Жулия и др.) при помощи Distant Aided Ray Marching.

Дополнительная часть (20 баллов максимум)

- Резкие тени (+1)
- Мягкие тени (+2-3)
- Анти-алиасинг (+2)
- Ambient Occlusion (+2-4)
- Окружение в виде текстурированной плоскости (+1)
- Имитация микро-рельефа (+2)
- Окружение в виде текстурированного куб-мапа (+2)
- Сцена сделана реалистично, содержит множество интересных объектов и материалов: (+1-5)
- (+1-2) Формулы комбинирования смотреть по той же ссылке <.../distfunctions.htm>.
- Бонус за реалистичность (+1-5 на усмотрении проверяющего)
- Непредусмотренный заданием бонус (+1-5 на усмотрении проверяющего)
- Нелинейный ход лучей, позволяющий добиться специфических искажений пространства по художественному замыслу (+1-3). Такой ход можно реализовать в алгоритме Марширования по лучу (ray marching) если ввести влияние гравитационного поля на направление луча на следующем шаге. Например, при визуализации чёрной дыры в космическом пространстве это необходимый элемент моделирования.
- Спектральные эффекты (от +2 до +3).
- Каустики (от +2 до +3) от кристалла на подставку/подложку.

Для всех используемых текстур обязательно применять билинейную, трилинейную или анизотропную фильтрацию. Оценки за любые текстурированные объекты, без фильтрации будут снижены.

#Разрешается выполнять задание на CPU, на любом языке программирования. Обратите внимание, однако, на правила сдачи.

Сборка и запуск должны быть тривиальными для проверяющего.

Материал для выполнения заданий

Трассировка лучей. Введение.

Алгоритм трассировки лучей в самом общем случае выглядит следующим образом: из виртуального глаза через каждый пиксел изображения испускается луч и находится точка его пересечения с поверхностью сцены (для упрощения изложения мы не рассматриваем объемные эффекты вроде тумана). Лучи, выпущенные из глаза называют первичными. Допустим, первичный луч пересекает некий объект 1 в точке Н1 (рис. 1).

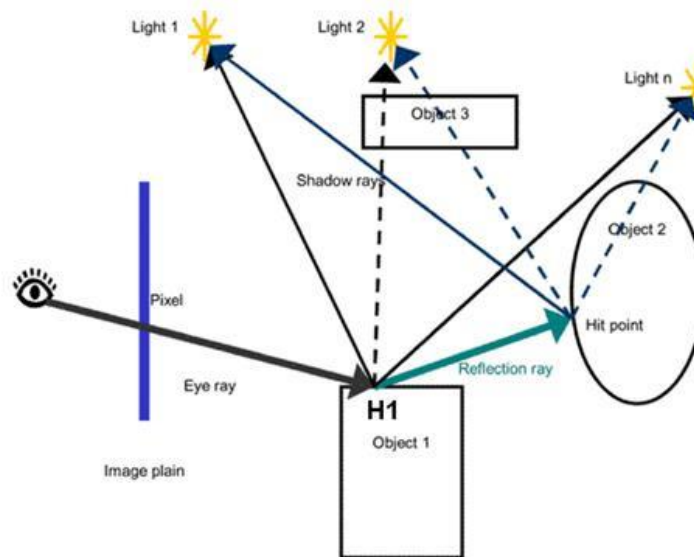


Рисунок 3: Обратная трассировка лучей.

Далее необходимо определить для каждого источника освещения, видна ли из него эта точка. Предположим пока, что все источники света точечные. Тогда для каждого точечного источника света, до него испускается теневой луч из точки Н1. Это позволяет сказать, освещается ли данная точка конкретным источником. Если теневой луч находит пересечение с другими объектами, расположенными ближе чем источник света, значит, точка Н1 находится в тени от этого источника и освещать ее не надо. Иначе, считаем освещение по некоторой локальной модели (Фонг, Кук-Торранс и.т.д.). Освещение со всех видимых (из точки Н1) источников света складывается. Далее, если материал объекта 1 имеет отражающие свойства, из точки Н1 испускается отраженный луч и для него вся процедура трассировки рекурсивно повторяется. Аналогичные действия должны быть выполнены, если материал имеет преломляющие свойства.

```

// Алгоритм трассировки лучей
//
float3 RayTrace(const Ray& ray)
{
    float3 color(0,0,0);

    Hit hit = RaySceneIntersection(ray);
    if (!hit.exist)
        return color;

    float3 hit_point = ray.pos + ray.dir*hit.t;

    // затенение, используется локальная модель освещения для источников света
    //
    for(int i=0; i < lights.size(); i++)
        if(Visible(hit_point, lights[i]))
            color += Shade(hit_point, hit.normal);

    if (hit.material.reflection > 0)
    {
        Ray reflRay = reflect(ray, hit);
        color += hit.material.reflection*RayTrace(reflRay);
    }

    if (hit.material.refraction > 0)
    {
        Ray refrRay = refract(ray, hit);
        color += hit.material.refraction*RayTrace(refrRay);
    }

    return color;
}

```

Листинг 1. Алгоритм обратной трассировки лучей.

Поясним фрагмент программы (листинг 1). Луч представлен двумя векторами. Первый вектор – pos – точка испускания луча. Второй – dir – нормализованное направление луча. Цвет – вектор из трех чисел – синий, красный, зеленый. В самом начале функции RayTrace мы считаем пересечение луча со сценой (представленной просто списком объектов пока что) и сохраняем некоторую информацию о пересечении в переменной hit и расстояние до пересечения в переменной hit.t. Далее, если луч промахнулся и пересечения нет, нужно вернуть фоновый цвет (в нашем случае черный). Если пересечение найдено, мы вычисляем точку пересечения hit_point, используя уравнение луча (эквивалентное параметрическому уравнению прямой с условием $t > 0$).

Когда мы вычислили точку пересечения в мировых координатах, приступаем к расчету теней и затенения (shading). Пусть источники лежат в массиве lights. Тогда проходим в цикле по всему массиву и для каждого источника света проверяем (той же трассировкой луча), виден ли источник света из данной точки hit_point. Если виден, прибавляем освещение от данного источника, вычисленное по некоторой локальной модели (например модели Фонга).

***При расчёте освещения по локальной модели внутри функции Shade:
яркость источника нужно делить на квадрат расстояния до него!***

Оставим пока что это правило без объяснения.

После, если у материала объекта, о который ударился луч, есть отражающие или преломляющие свойства, трассируем лучи рекурсивно, умножаем полученный цвет на соответствующий коэффициент отражения или преломления и прибавляем к результирующему цвету. Коэффициенты reflection и refraction могут быть как монохромными так и цветными. Всё зависит от того, какая используется математическая модель для представления материалов.

Теневые лучи можно сделать цветными. Такие лучи используются, если один объект перекрывается другим прозрачным объектом. В таком случае рассчитывается длина пути теневого луча внутри прозрачного объекта, и тень может приобрести какой-либо оттенок исходя из закона Бугера-Ламберта-Бера. Разумеется, тени, рассчитанные таким образом корректны, только если прозрачный объект, отбрасывающий тень, имеет очень близкий к единице коэффициент преломления (считаем что коэффициент преломления воздуха равен 1). Если это не так, то под прозрачным объектом образуется сложная картина, называемая каустиком. Каустики рассчитываются отдельно с помощью метода фотонных карт или других более продвинутых алгоритмов. Типичный пример каустика – солнечный зайчик от стакана воды, когда через него просвечивает солнце.

Удаление хвостовой рекурсии

В случае, если отраженный (или преломленный) луч всегда один (то есть не может быть так, что имеется одновременно и отраженный и преломленный луч), то рекурсивный вызов в листинге 1, можно заменить на простой цикл.

```
float3 color(0,0,0);
float3 k(1.0f, 1.0f, 1.0f);

for(int i=0; i < MAX_REFLECTION_DEPTH; i++)
```

```

{
    Hit hit = RaySceneIntersection(ray);
    if (!hit.exist)
        return color;

    float3 hit_point = ray.pos + ray.dir*hit.t;

    // затенение, используется локальная модель освещения для источников света
    //
    for(int i=0; i < lights.size(); i++)
        if(Visible(hit_point, lights[i]))
            color += k*Shade(hit_point, hit.normal);

    if (hit.material.reflection <= 0)
        break;

    ray = reflect(ray, hit);
    k *= hit.material.reflection;
}

```

Листинг 2. Алгоритм обратной трассировки лучей без рекурсии.

Это весьма актуально при реализации на GPU, где рекурсия не поддерживается изначально (хотя может быть смоделирована через стек).

Марширование луча (Ray Marching)

Этот способ позволяет визуализировать объёмные эффекты за счёт накопления цвета вдоль луча (рис. 4). Сначала необходимо найти ближнюю и дальнюю точки пересечения луча и ограничивающей поверхности (в случае задания поверхности кристалла). Не забудьте что кристалл преломляет свет, поэтому направление луча изменится при попадании его в кристалл.

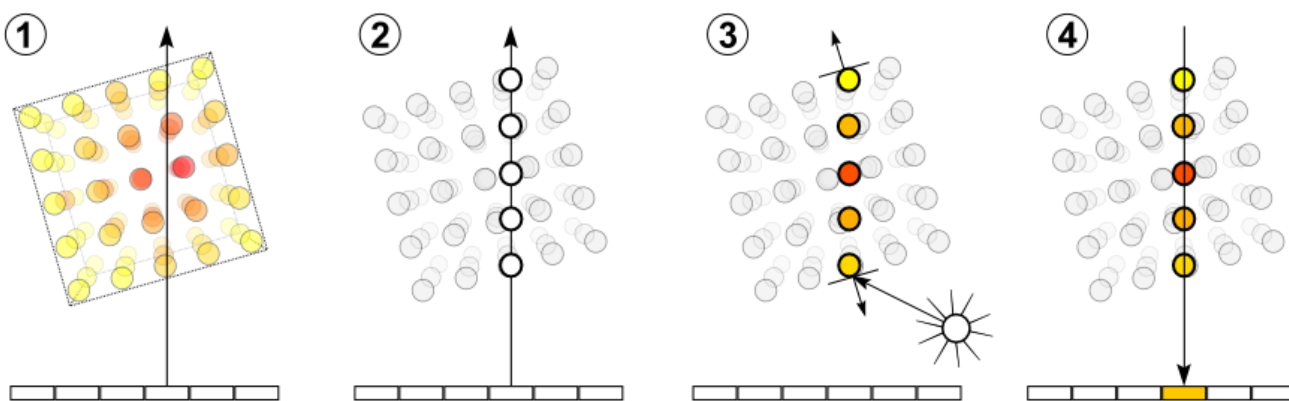


Рисунок 4. Иллюстрация алгоритма марширования по лучу.

После чего необходимо с фиксированным (либо динамически-изменяемым) шагом двигаться вдоль направления луча, учитывая его взаимодействие со средой и накапливая уменьшая на каждом шаге текущую непрозрачность путём последовательного умножения. За счёт этого реализуется экспоненциальный закон затухания света в среде Бугера-Ламберта-Бэра (Beers Law).

Фото-реалистичная визуализация. Погружение.

Однако, трассировка лучей в том виде, в котором она была описана не дает фото-реалистичное изображение. Во-первых, материалы реального мира в силу наличия микрорельефа обычно не отражают свет строго в одном направлении (Ламбертовские поверхности, например, равномерно рассеивают свет по всем направлениям). Во-вторых, источники света далеко не всегда имеют точечные размеры (тени от таких источников света получаются мягкие, с плавным переходом свет-тень).

Далее, изображение хранится в памяти как двумерная матрица пикселей, это бесспорно. Однако, считается, что изображение предметов реального мира это не просто набор пикселей. Авторы [Physically Based Rendering] и [Ray Tracing From The Ground Up] склонны рассматривать изображение как непрерывную двумерную функцию. То, что мы видим на экране – приближение этой функции, ее дискретизованное в заданном разрешении представление.

Трассировка лучей в самом общем понимании - это метод, позволяющий восстановить значения этой функции с помощью точечных сэмплов, то есть значений этой функции в точках. Поэтому для того чтобы получить изображение, лишенное алиасинга, необходимо как минимум делать более чем 1 сэмпл на пиксел.

Сосредоточимся теперь на том, как получать значения функции изображения в точках. Каждой точке в двумерном пространстве экрана соответствует точка на некоторой поверхности (эту точку можно найти с помощью прослеживания пути луча, выпущенного из виртуального глаза через точку экрана в сцену; назовем ее точкой x). Освещенность точке поверхности x вычисляется при помощи интеграла следующего вида:

$$I(\phi_r, \theta_r) = \int_{\phi_i, \theta_i} L(\phi_i, \theta_i) R(\phi_i, \theta_i, \phi_r, \theta_r) \cos(n, l_{\phi_i, \theta_i}) d\phi_i d\theta_i$$

Формула 1. (ϕ_i, θ_i) – направление на источник. (ϕ_r, θ_r) – направление на наблюдателя.

$L(\phi_i, \theta_i)$ – это функция, описывающая общее освещение, падающее в точку x под всеми возможными углами в пределах полусферы. $R(\phi_i, \theta_i, \phi_r, \theta_r)$ – BRDF (bidirectional reflectance distribution function – двунаправленная функция распределения отражения или ДФО). Эта функция полностью описывает свойства взаимодействия конкретной поверхности со светом.

Фактически, ДФО просто связывает по некоторому закону интенсивность и угол падающего света с интенсивностью и углом отраженного или пропущенного прозрачной поверхностью света (рис. 3).

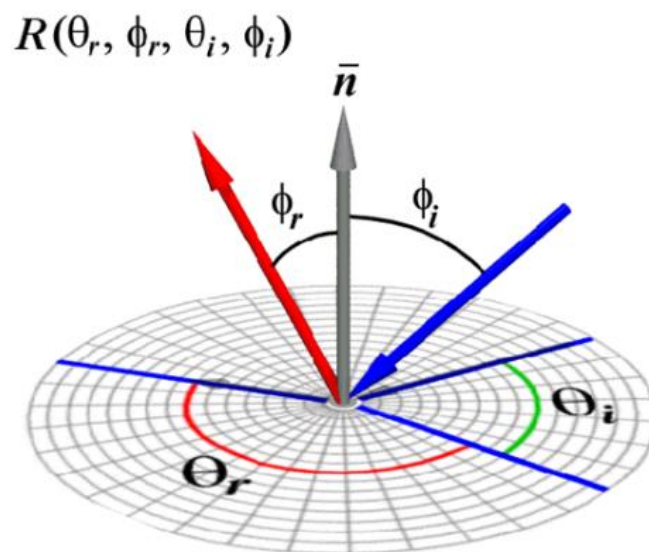


Рисунок 5: BRDF (ДФО).

Вычисляемая интегрированием интенсивность освещения в точке $I(\varphi_r, \theta_r)$ является не числом, а функцией. Другими словами, она дает значения интенсивности света, отражаемой поверхностью под разными углами (собственно пара φ_r, θ_r). Таким образом, выполняя интегрирование по полусфере приходящего (падающего) в точку освещения L с учетом свойств поверхности R, мы получаем новую функцию распределения освещения в пространстве, обусловленную свойствами отражения поверхности (материала) в точке x. Хотя если мы рассматриваем луч, который трассировали через пиксел из виртуального глаза, то направление конечно одно и задано этим лучом.

Обратите внимание что:

Одна из наиболее часто используемых BRDF, представляющая полностью диффузную поверхность – BRDF Ламберта. Это константа! Скалярное произведение нормали и направления на источник зачёркнуто! Это сделано по той причине, что *скалярное произведение* --- это на самом деле *косинус в интеграле*. А не часть BRDF Ламберта, которое представляет из себя константу (при этом идеально-зеркальное отражение всегда рассматривается как специальный случай, в котором этого косинуса нет). К сожалению, похожая ошибка допущена во многих источниках.

```
float3 EvalDiffuseBRDF(float3 l, float3 norm, float3 color)
{
    return color*max(dot(norm,l),0);
}
```

Листинг 4. Вычисление BRDF Ламберта.

Итак, интеграл освещённости (формула 1) берется из тех соображений, что освещенность в точке поверхности складывается из света, падающего на поверхность со всех направлений и отражающегося по какому-то определенному закону (какому именно, устанавливает функция ДФО). Эта формула не более чем математическая модель и она верна не всегда. Например, в случае стекла нужно учитывать свет, приходящий из под поверхности и проводить интегрирование по полной сфере. В случае кожи ситуация еще сложнее, так как необходимо учитывать такой эффект как подповерхностное рассеивание.

Существуют различные способы вычисления интеграла освещённости. Одни из них более точные, другие менее точные, но более быстрые. Среди точных методов – наиболее популярные

это Монте-Карло трассировка лучей, трассировка путей (прямая, обратная, двунаправленная) и фотонные карты.

Локальные модели освещения. Shade и квадратичное затухание.

Считать интеграл освещенности точно – довольно дорого. Самая простая аппроксимация заключается в том, чтобы рассматривать только прямой свет от источников. Непрямой свет принимается за константу (называемую обычно ambient). Это и называется *локальной моделью освещения*. Расчёт по такой модели обычно называется *затенением (Shade)*. Популярной моделью учёта яркости является квадратичное затухание с добавлением линейной компоненты и константы (<http://in2gpu.com/2014/06/19/lighting-vertex-fragment-shader/>):

$$I = \text{emissive} + \text{ambient} + \text{diffuse} * \text{att} + \text{specular} * \text{att}$$

$$\text{att} = \frac{1}{k_C + k_L * d + k_Q * d^2}$$

d is the distance from light source
to current vertex

k_C is the constant attenuation

k_L is the linear attenuation

k_Q is the quadratic attenuation

Иллюстрация 1: Не физ. корректный способ расчёта локальной освещённости.

Обратите внимание, что формула для переменной **att** в действительности **не корректна** с точки зрения физики (и даже, скорее, с точки зрения банальной математики). Для того чтобы она была корректна, константы k_C и k_L должны быть приравнены **к нулю**, а константа k_Q **к единице**. То есть затухание **строго квадратично**. Отчего так? В воздухе свет не затухает – эта формула вообще не учитывает затухание в среде. Тогда откуда берётся квадрат?

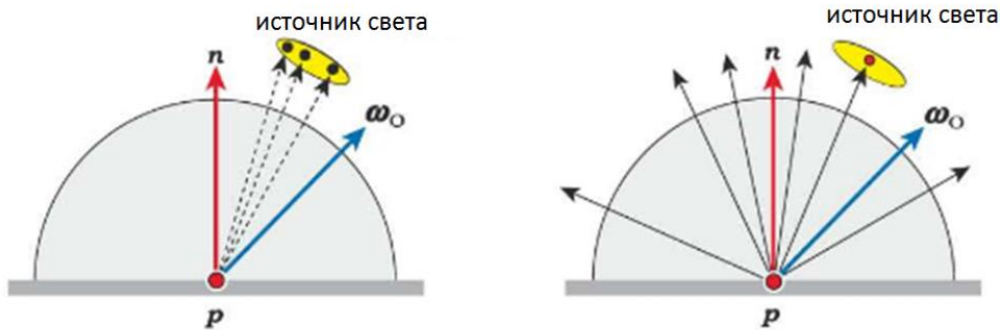


Иллюстрация 2: Расчёт освещённости от площадного источника света.

В действительности он берётся из корректного способа вычисления интеграла освещённости. Предположим, что наш источник света не точечный, а площадный (иллюстрация 2). Если мы будем вычислять интеграл освещённости честно, бросая лучи на полусфере, тогда некоторое небольшое их количество попадёт в источник. А остальные пролетят мимо.

Теперь вопрос: как зависит от расстояния до источника процент лучей, которые будут попадать в источник? Ответ можно получить, если спроецировать источник на полусферу. Тогда мы придём к $1/d^2$. Вот откуда берётся квадратичное затухание. Таким образом, $1/d^2$ --- это чисто геометрический фактор, определяющий какой процент лучей попал бы в источник, если бы мы всё делали правильно. Ведь когда мы считаем освещение по локальной модели, мы все лучи выпускаем точно в источник. Поэтому нам необходим т. н. «регуляризатор» – коэффициент $1/d^2$, который позволяет получить правильный результат, **как если бы** мы считали интеграл освещённости честно. Зато теперь мы можем корректно рассчитывать освещение даже от точечных источников, несмотря на то что вероятность попадания лучом в точку равна нулю.

Попытка добавить константный или линейный члены затухания связаны с желанием разработчиков получить глобальную освещённость бесплатно, путём некой очень простой аппроксимации. Либо с попыткой моделирования некоторого сфокусированного или направленного источника, у которого тем не менее хочется сделать затухание по той или иной причине. Вы **можете использовать любые локальные модели освещения**, если они на ваш взгляд повышают реалистичность изображения. В задании вы ничем не ограничены.

Аппроксимация Ambient Occlusion для расчета вторичной освещенности

Как мы уже отмечали, считать интеграл освещенности точно – довольно дорого. Однако, существует ряд аппроксимаций, позволяющих примерно оценить его значение, не выполняя честную трассировку лучей по всем направлениям. Во-первых, следует разделить не прямое освещение на “резко-меняющееся” и “плавно-меняющееся” – соответственно высокочастотную компоненту и низкочастотную компоненты. Высокочастотную составляющую считают, как и раньше, трассировкой лучей. Однако, для нее количество лучей обычно небольшое (если только не рассматриваются мягкие тени и glossy reflections). Например, вам нужен всего 1 отраженный луч чтобы посчитать зеркальные отражения на поверхности.

Для низкочастотной компоненты, обуславливающей диффузные пере-отражения света (которую дольше всего считать), можно применить аппроксимацию, позволяющую получить очень похожий на правду, но не совсем точный результат.

Идея **Ambient Occlusion** похожа на трассировку лучей по всем направлениям, но более простая в вычислительном плане. Мы предполагаем, что вторичный, диффузно переотраженный свет *относительно равномерно распределен в пространстве*. Такое освещение можно заменить неким источником, называемым Environment Light. Он может быть задан панорамой, куб-мапом, или в простейшем случае просто – числом (3 числами – для красного, синего и зеленого). То есть мы говорим, пусть весь наш вторичный свет от относительно-далеко стоящих поверхностей задан некой константой `g_environmentLightColor`.

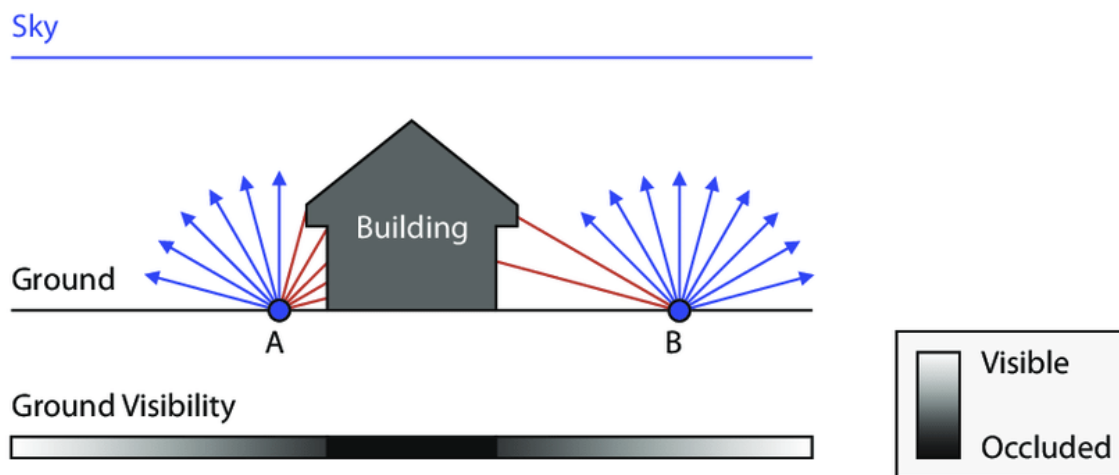


Рисунок 6: Иллюстрация аппроксимации Ambient Occlusion.

Тогда, в той точке, где мы хотим посчитать освещение, мы испускаем лучи по всем направлениям, но не трассируем их в сцену, а лишь пытаемся вычислить, какой процент нашего *Environment Light* закрыт близлежащими поверхностями.

Ссылки

- <http://www.iquilezles.org/www/material/nvscene2008/rwwtt.pdf>
- <http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm>
- <http://www.iquilezles.org/www/articles/terrainmarching/terrainmarching.htm>
- <http://www.subblue.com/projects/mandelbulb>
- <http://www.raytracegroundup.com/>
- <http://www.pbrt.org/>
- <http://ray-tracing.ru>
- <http://code.google.com/p/gl33lessons/>
- <https://habr.com/ru/post/437714/> (взрыв, 180 строк кода на C++)
- <https://blog.yiningkarlli.com/2019/05/nested-dielectrics.html>