ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»**
**( Н И У « Б е л Г У » )**

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ
КАФЕДРА ОБЩЕЙ МАТЕМАТИКИ

# РАЗРАБОТКА ПРОГРАММНОГО МОДУЛЯ ВИЗУАЛИЗАЦИИ ГЕОМЕТРИЧЕСКИХ ОБЪЕКТОВ С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМОВ ПРОЦЕДУРНЫХ ТЕКСТУР

Выпускная квалификационная работа
обучающегося по направлению подготовки
01.04.01 Математика
Магистерская программа Теория чисел
очной формы обучения, группы 12001732
Зенкевича Дмитрия Александровича

Научный руководитель
д.т.н, профессор
Аверин Г.В.

Рецензент
Директор студии компьютерной
Графики и 3Д моделирования
«Нексус»
Комисов С.А

БЕЛГОРОД 2019

СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

На сегодняшний день индустрия компьютерной графики и 3д моделирования переживает пиковую востребованность и используется почти во всех сферах нашей жизни. Например, архитектура, медицина, образование, кинопроизводство, игры, машиностроение и многое другое. В связи с этим растёт сложность и объем поставляемых задач. В таких условиях становится необходим поиск методов и инструментов для автоматизации и упрощения работы.

В настоящей работе рассматривается создание приложения для создания трёхмерных объектов с помощью процедурных текстуры.

Объектом исследования являются программное обеспечение для создания трёхмерной компьютерной графики.

Предмет исследования – подпрограммы для процедурной генерации трёхмерного объема из двумерного изображения.

Цель работы заключается в создании встраиваемой подпрограммы на базе продукта трёхмерной визуализации для генерации трёхмерного объема из двумерного изображения.

Задачи работы:

1. Анализ существующих на рынке программных комплексов для выполнения поставленных целей;

2. Выбор профессионального свободного и открытого программного обеспечение для создания трёхмерной компьютерной графики, включающее в себя средства моделирования, анимации, рендеринга, постобработки и, компоновки с помощью «узлов»;

3. Разработка встраиваемой подпрограммы;

4. Проектирование интерфейса;

5. Тестирование.

Работа носит практический характер. Ее результаты могут быть использованы в последующих разработках, касающихся генерации трёхмерных объектов с помощью процедурных текстур.

По материалам работы опубликованы 3 статьи, тезисы доклада.

Работа состоит из 3 глав. В первой производится анализ предметной области и выбор необходимого программного обеспечения, необходимого для дальнейших разработок. Результаты работы сформулированы в 3 главе.

Во второй главе проводится описание основных технологий, которые будут использованы для создания программы.

В третьей главе рассматривается процесс создания программного обеспечения для генерации трёхмерных объектов.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Для создания программного продукта, необходимого для автоматизации создания сложных трёхмерных и в частности двумерных объектов необходимо выбрать базовый 3д пакет. На сегодняшний день на рынке представлено большое количество продуктов для создания трёхмерной графики, например: 3DMax, Maya, Cinema4d, Blender. Только Blender обладает открытым исходным кодом и распространяется на бесплатной основе, он был выбран в качестве базовой системы для которой будет разрабатываться программа. Также использовалась программа Substance Designer. Substance Designer – это мощный инструмент, предназначенный для создания текстурных карт любой сложности.

## 1.1 Анализ программного обеспечения и способа визуализации

Blender – это профессиональное свободно распространяющееся и открытое программное обеспечение для создания трёхмерной компьютерной графики, включающее в себя средства моделирования, анимации, рендеринга, постобработки и монтажа видео со звуком, компоновки с помощью «узлов» (Node Compositing), а также для создания интерактивных игр. В настоящее время пользуется наибольшей популярностью среди бесплатных 3D-редакторов в связи с его быстрым и стабильным развитием, которому способствует профессиональная команда разработчиков. Визуализирует 3Д модели с помощью движка рендеринга Cycles. Разберём что такое движок рендеринг и как он работает.

Cycles — это новый движок рендеринга, который впервые появился в Blender версии 2.61 (декабрь 2011). По сегодняшний день он доступен в ка-

честве дополнения для программы наряду со старым встроенным рендером Blender Internal. Cycles создает изображение методом трассировки лучей с упором на интерактивность и простоту использования. Для материалов используется система нодов, с помощью которой можно создавать фотореалистичные материалы любой сложности. Большими преимуществами движка являются возможность быстрого просмотра результата непосредственно в окне 3D-вида, а также рендеринг с помощью графической карты (GPU).

Движок рендеринга Cycles имеет в своем распоряжении 77 нодов для создания материалов. Помнить их все и знать, что каждый делает, достаточно проблематично, если вы не создаете материалы с его помощью ежедневно. А вот с помощью данной книги для вас не составит труда узнать, что делает каждый нод (и не только ноды), и в любой момент освежить в памяти полученные ранее знания.

В компьютерной графике все, что определяет внешний вид поверхности или объема называется шейдером. Но в Cycles шейдерами называется особая группа нодов из меню Shader, которые в большей степени определяют внешний вид объекта. А материалом называют всю группу нодов, которая составляет отдельный материал.

## 1.2 Терминология технологии визуализации

Основным отличием Cycles от нефотореалистичных движков состоит в том, что лучи отражения и преломления не разделяются на различные функции, а вычисляются с помощью одной функции BSDF. В целом это создает небольшие ограничения по настройке материалов, но зато делает этот процесс гораздо более простым, за счет значительно меньшего количества параметров.

- BSDF (bidirectional scattering distribution function) — функция распределения двунаправленного рассеивания. Является наиболее общей функцией. Содержит в себе функции BRDF и BTDF.

- BSSRDF (bidirectional scattering-surface reflectance distribution function) — функция распределения двунаправленного рассеивания поверхности отражения. BSSRDF описывает, как свет перемещается между любыми двумя лучами, которые попали на поверхность. В случае с Cycles на данный момент используется лишь в шейдере Subsurface Scattering.

- BRDF (bidirectional reflectance distribution function) — функция распределения двунаправленного отражения. Представляет собой упрощенную BSSRDF, при условии, что свет отражается от поверхности в точке падения.

- BTDF (bidirectional transmittance distribution function) — функция распределения двунаправленного пропускания. Работает также, как и BRDF с той разницей, что луч выходит с противоположной стороны поверхности.

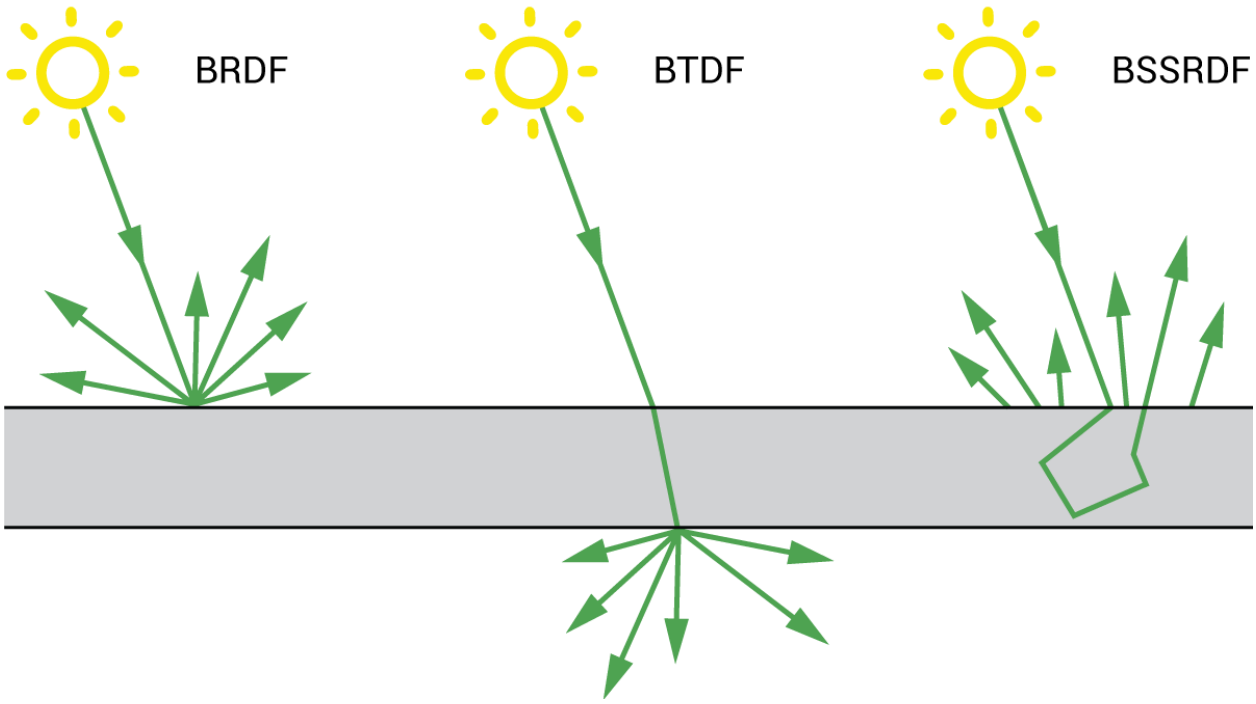


Рис. 1.1 – Поведение луча при использовании главных функций

## 1.3 Средства определяющие внешний вид объекта

Шейдеры определяют то, как будет взаимодействовать свет с мешем. Один или несколько шейдеров могут указать лучу отразиться от поверхности, пройти сквозь нее или поглотиться ею. Исключением является шейдер Emission, который управляет тем, как свет излучается из меша.

Материалы определяют внешний вид мешей. С их помощью можно влиять на внешний вид поверхности (surface), на внутренний объем поверхности (volume) и на смещение поверхности (displacement).

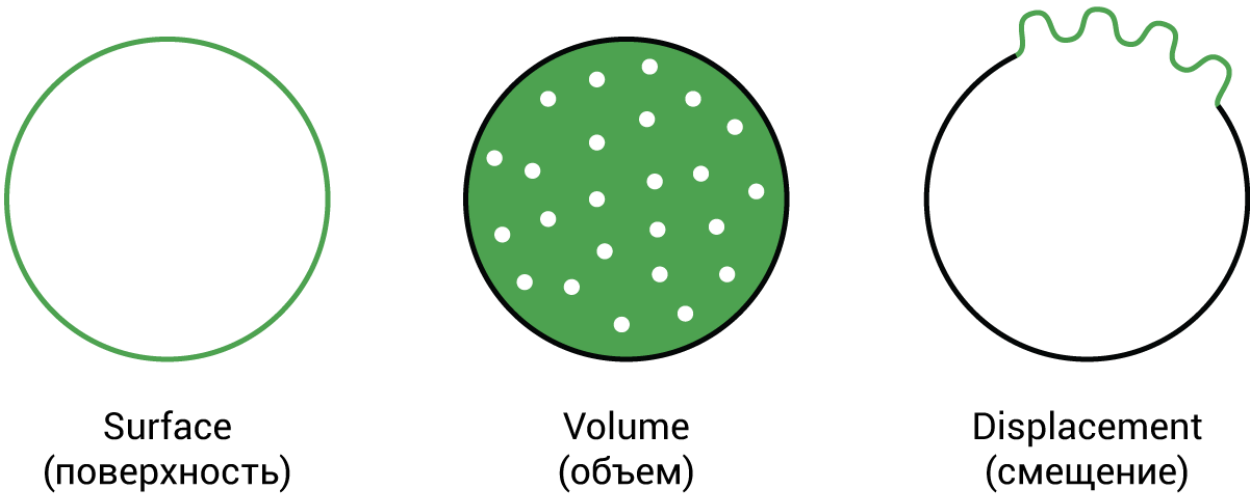


Рис. 1.2 – Части воздействия света

Surface

Поверхностные шейдеры определяют взаимодействие света с объектом на его поверхности. С их помощью определяется дальнейшее направление луча: поглощение, отражение, преломление.

Volume

Если луч не отразился и не поглотился на поверхности объекта, он попадает внутрь меша и проходит прямиком к обратной его стороне. В случае наличия шейдера объема будет описано взаимодействие света с объемом. Свет может рассеиваться, поглощаться или излучаться в любой точке объема. Поверхностные шейдеры и шейдеры объема могут быть объединены в одном материале. Это может быть полезно для таких материалов, как стекло, вода или лед, потому как вам необходимо контролировать поведение луча как на поверхности этих объектов, так и внутри них.

Displacement

Сама форма объектов может быть изменена с помощью материалов. Таким образом, текстуры могут быть использованы для того, чтобы сделать поверхность меша более детализированной. В зависимости от настроек, сме-

щение может быть виртуальным (bump), при котором поверхность остается неизменной, а изменяется лишь направление нормалей, чтобы создать впечатление смещения. Также смещение может быть реальным (как при использовании модификатора displacement), или комбинированным: bump + displacement.

## 1.4 Контроль количества отскоков луча

Максимальное количество отскоков луча света можно контролировать. В идеале их должно быть очень много, но на практике мы не можем дожидаться рендеринга сцены бесконечно, поэтому приходится добиваться приемлемых результатов с ограниченным числом отскоков луча. Мы можем регулировать индивидуально количество отскоков для различных типов лучей: diffuse, glossy и transmission.

При установке минимального количества отскоков луча ниже максимального значения, включается вероятностное прекращение трассировки луча. В таком случае луч прекратит свое существование при превышении минимального количества отскоков с определенной вероятностью, что даст более быстрый и в то же время более шумный рендер.

Основными источниками шума в сцене являются отражающая и преломляющая каустики. Их вы также можете отключить по желанию, в ущерб реалистичности изображения.

### 1.4.1 Сохранение энергии

Cycles создан с учетом физических свойств объектов (к сожалению, не всех). Благодаря этому, значительно проще добиваться реалистичных резуль-

татов и сбалансированного освещения. Но есть несколько моментов, которые вы всегда должны держать в голове. Для того, чтобы материал выглядел реалистично с использованием глобального освещения, он должен следовать правилу сохранения энергии. Это правило гласит, что объект не может отразить больше света, чем на него попало.

В Cycles это правило не является строгим и соответственно вы можете его нарушить. При смешивании шейдеров с помощью шейдера Mix, выходные значения всегда будут в диапазоне от 0 до 1, что соответствует данному правилу. А вот при использовании шейдера Add, мы создаем больше отраженного света, чем количество света, которое первоначально попало на объект. Таким образом, вместо глянцевой поверхности, можем получить излучающую, что будет абсолютно некорректно, с точки зрения физики. Но создан шейдер Add не только для того, чтобы нарушать физические правила, а также для создания полупрозрачных материалов, волос, объемов и много другого.

## 1.5 Работа трассировщика пути

Для каждого пикселя изображения создается луч. Все лучи исходят из камеры и до столкновения с чем-либо являются лучами камеры (camera ray). Если луч попадет на источник света (emission) или шейдер holdout, то он будет удален из сцены, а пиксель получит цвет источника света или станет прозрачным, в случае с holdout. Во всех остальных случаях поведение луча зависит от типа шейдера, назначенного поверхностям, об которые он ударяется. Самым простым примером будет идеальное зеркало (sharp glossy). В данном случае луч будет отражаться согласно простейшему правилу, то есть под тем же углом, под которым и попал на поверхность. После этого луч уже продолжит свой путь как глянцевый луч (glossy ray). Допустим, далее он ударит-

ся о диффузную поверхность (diffuse). От данной поверхности луч отразится в случайном направлении. Если к этому моменту времени луч не достигнет максимального количества отскоков для него, то в конечном счете луч попадет на источник света. На этом его путь будет прекращен и пиксель получит цвет в зависимости от того, какой путь преодолел луч. Данный процесс повторяется такое количество раз, сколько сэмплов вы установите на вкладке рендера. В конце принимается среднее значение всех полученных цветов пикселя.

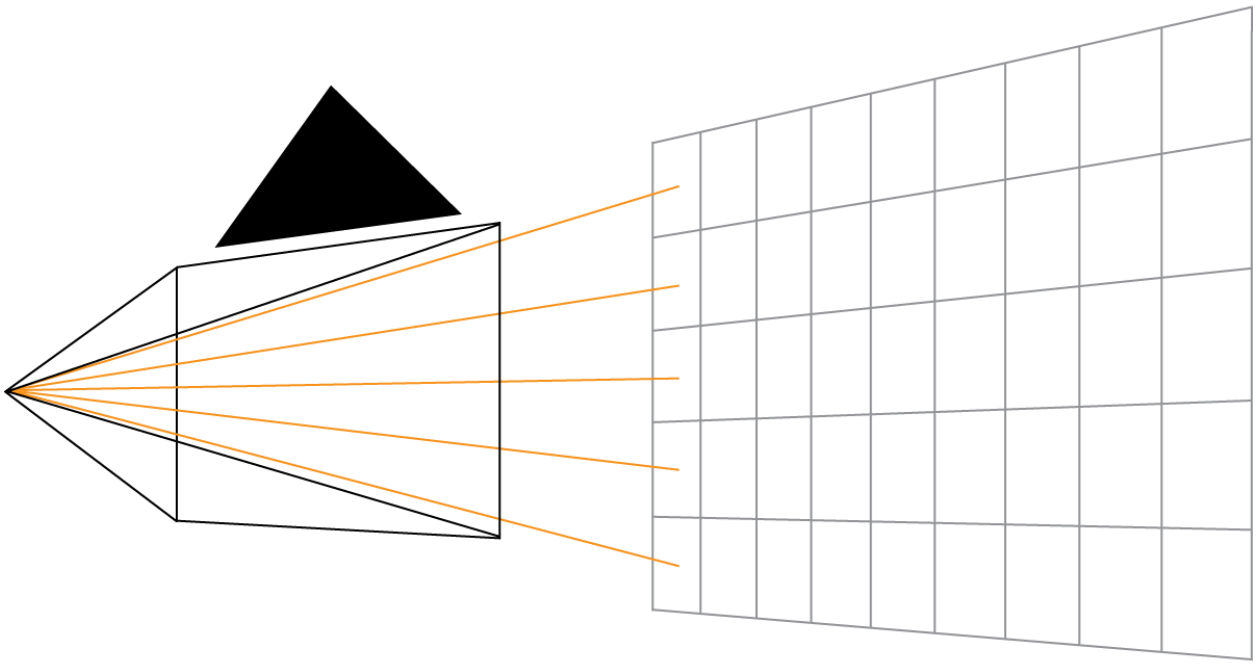

Рис. 1.3 – Установка сэмлов пиксельного изображения

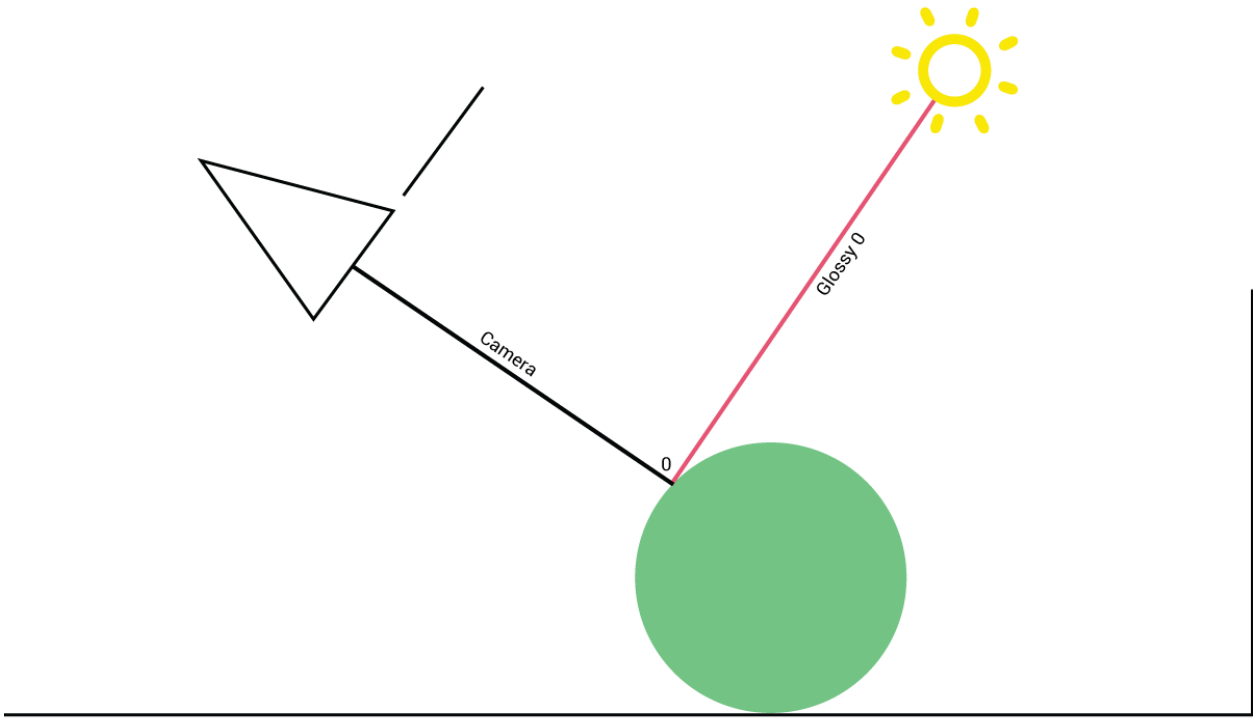


Рис. 1.4 – Луч, вышедший из камеры на глянцевую поверхность

На рисунке 1.4 показан очень простой пример. Луч, вышедший из камеры, попадает на глянцевую поверхность и отражается на источник света. Путь луча после отскока окрашен в красный цвет и называется Glossy 0. Это означает, что луч поменял свой тип, с луча камеры (camera ray) на луч отражения (reflaction). 0 означает количество отскоков, проделанных лучом (Cycles начинает считать с 0). После отскока от поверхности, луч изменяет свой тип, в зависимости от типа поверхности, на которую он попал. В данном случае для поверхности назначен шейдер Glossy. Но разные лучи, вышедшие из камеры, могут отскакивать в различных направлениях и попасть на пол, который на рисунке 1.5 имеет диффузную поверхность. В таком случае луч, отразившийся от пола, станет диффузным лучом (diffuse ray).

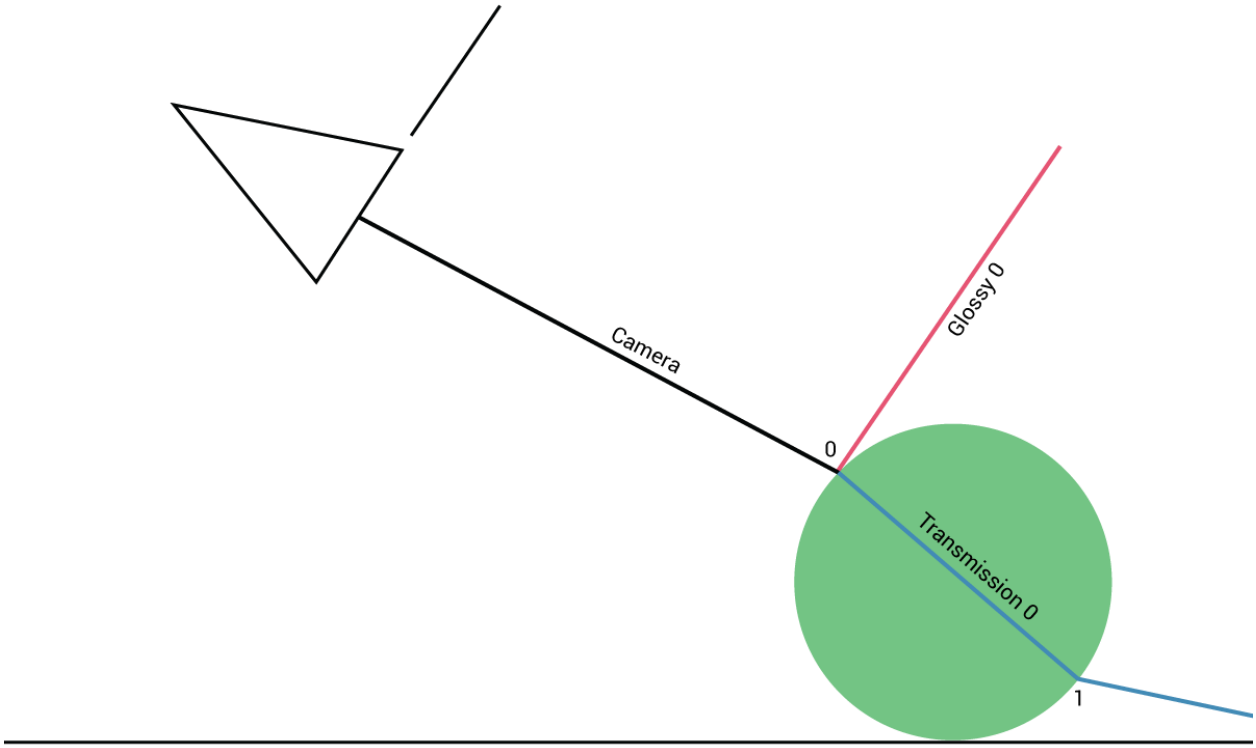


Рис. 1.5 – Луч, вышедший из камеры на стеклянную поверхность

Попадая на стеклянную поверхность, луч решает по какому пути ему пойти дальше.

Решить отразиться ему от поверхности либо пройти сквозь нее, луч может на основе фактора Френеля. Первый пример мы уже разобрали (отражение луча от поверхности), теперь давайте рассмотрим путь сквозь объект. Попав внутрь и выйдя из объекта, луч будет иметь тип Transmission до тех пор, пока не попадет на пол. Так как пол имеет диффузный шейдер, луч соответственно станет диффузным, и продолжит отскакивать от поверхности до попадания на источник света. Весь путь данного луча состоит из 3-х отскоков, но он имеет лишь один отскок transmission и один diffuse. Если бы вы установили максимальное количество отскоков менее 3-х, то луч прекратил бы свое существование при ударе о стену, и не дошел бы до источника света.

Каждый луч, вышедший из камеры, возвращает значение интенсивности своего пикселя. Этот результат называют сэмплом (sample). Чтобы опре-

делить окончательный цвет пикселя, берется среднее значение всех сэмплов, полученных для него.

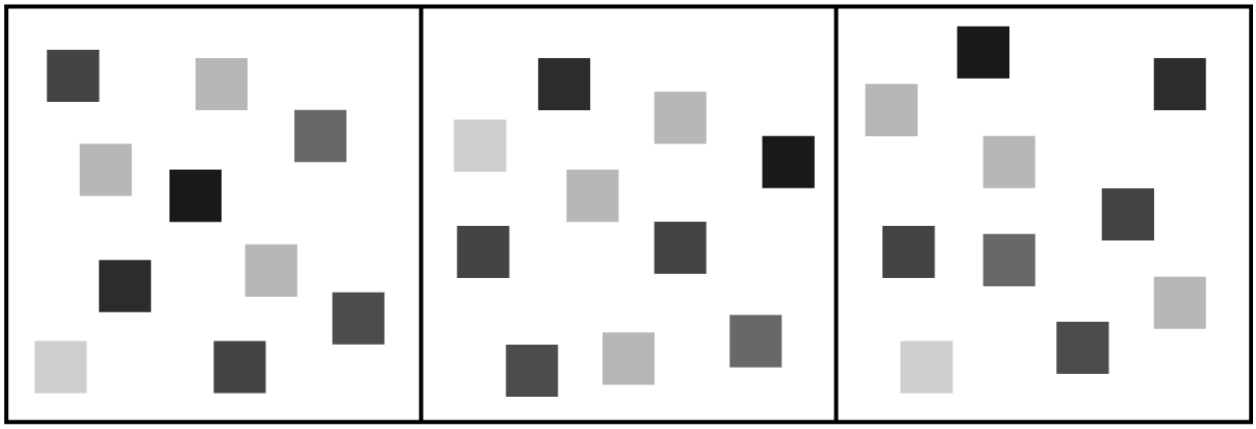

Рис. 1.6 – Три пикселя с хранящимися значениями интенсивности

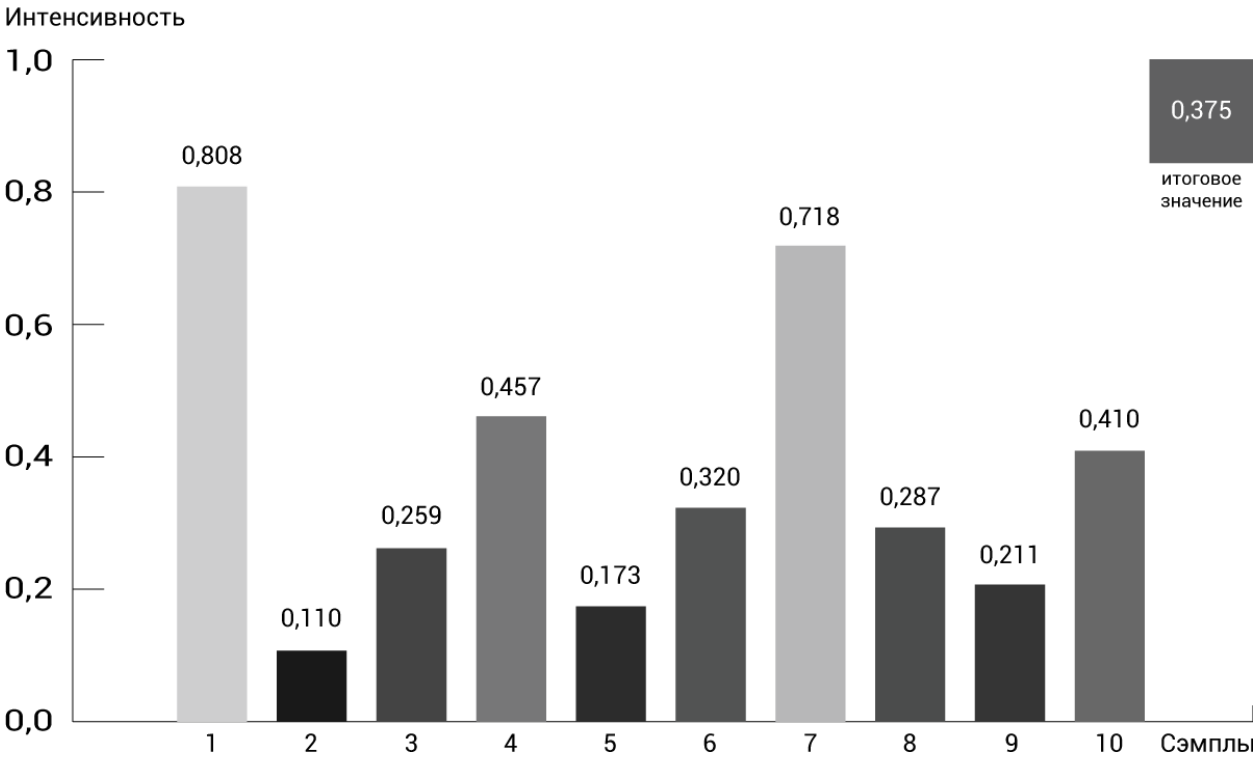


Рис. 1.6 –Хранящимиеся значениями интенсивности

## 1.6 Поведение луча при попадании на поверхность

На рисунках выше может показаться, что путь луча хорошо определен, но они иллюстрируют лишь один из возможных его путей. Помимо случайного выбора шейдера поверхности при попадании на нее, Cycles также случайным образом выбирает направление луча. Практически все шейдеры имеют компонент, который указывает Cycles возможные направления пути луча при попадании на поверхность. Исходя из этой информации, Cycles случайным образом выбирает направление луча в соответствии с заданным диапазоном. В этом диапазоне различные углы отскоков имеют различные вероятности. Так, например, при попадании на диффузную поверхность, вероятности всех углов одинаковы, в то время как для глянцевого шейдера вероятность отразиться под углом падения значительно выше, чем под любым другим.

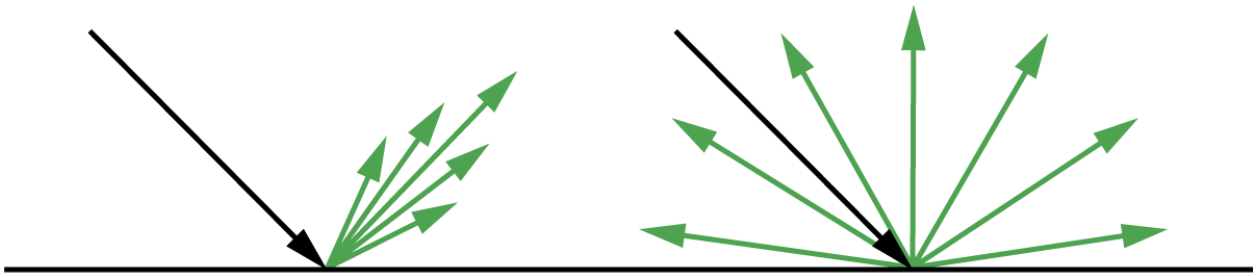

Рис. 1.7 – Отображение вариантов отражения луча

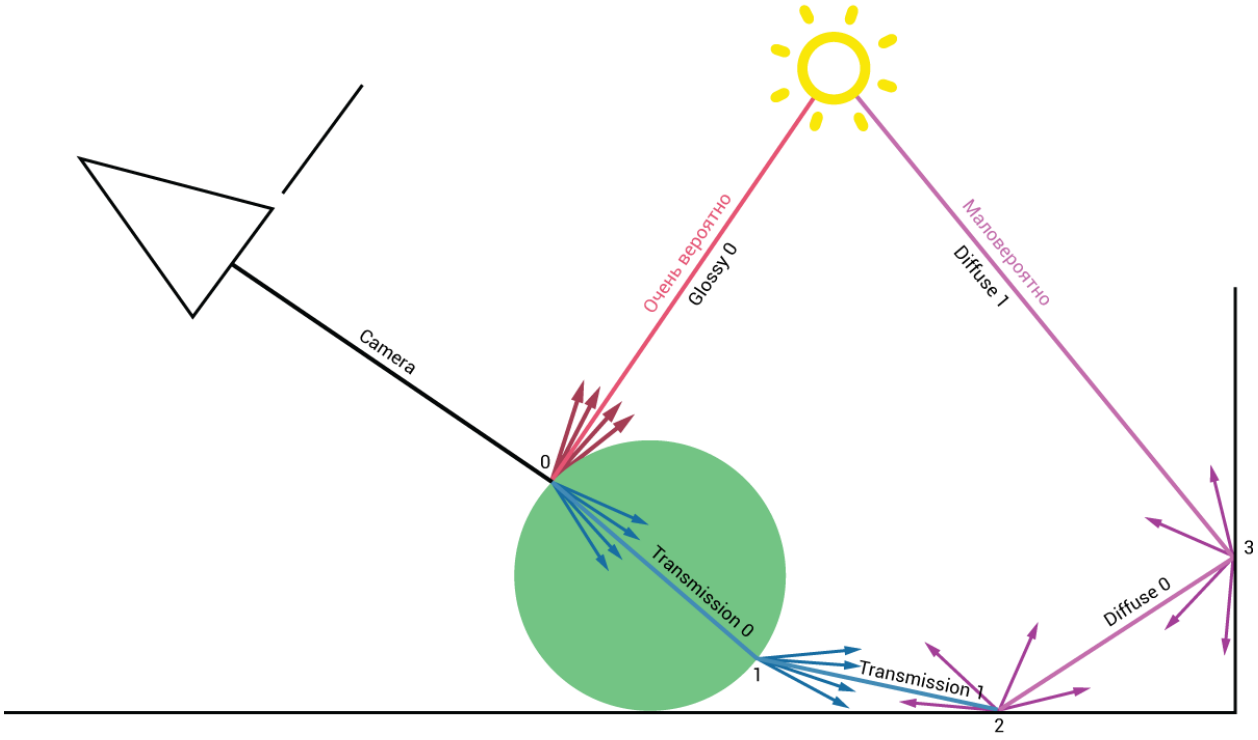


Рис. 1.8 – Возможные варианты направления луча после каждого отскока

Вероятность добраться до задней стенки у луча не большая, но исходя из рисунка, мы видим, что задняя часть сцены должна быть освещена. Так как же этого достичь? Использовать пятизначные значения количества сэмплов? К счастью, в Cycles есть умный способ сбора информации об освещении для каждого отскока, который называется Light Sampling.

Процесс, показанный в предыдущей главе, это упрощенная версия того, что Cycles делает в реальности. На самом деле во время каждого отскока создается еще один новый луч. Хитрость состоит в том, что Cycles заранее знает, где находятся источники света. Поэтому при попадании на поверхность, Cycles в первую очередь создает луч в направлении к случайному источнику света. Этот процесс называется Light Sampling, а лучи — Shadow rays. Теневые лучи (shadow rays) либо находят путь к источнику света, либо блокируются другими объектами. В первом случае количество света от этого источника сохраняется. Затем Cycles случайным образом выбирает шейдер поверхности, на которую попал луч. Этот процесс называется "оценка шейде-

ров". Следующим шагом Cycles определяет в каком направлении луч пойдет дальше. Он случайным образом выберет одно из возможных направлений для данного шейдера. Этот процесс называется "выборка шейдеров". Луч продолжит свой путь в новом направлении.

Так в чем разница между "оценкой шейдеров" и "выборкой шейдеров"? Выборка используется для выбора нового направления луча, в зависимости от типа шейдера. Оценка же означает, что вы уже смотрите в определенном направлении и спрашиваете шейдер, насколько вероятно, что луч отразится в этом же направлении.

Давайте рассмотрим пример с диффузной поверхностью. При попадании на нее Cycles сразу же определит теневой луч:

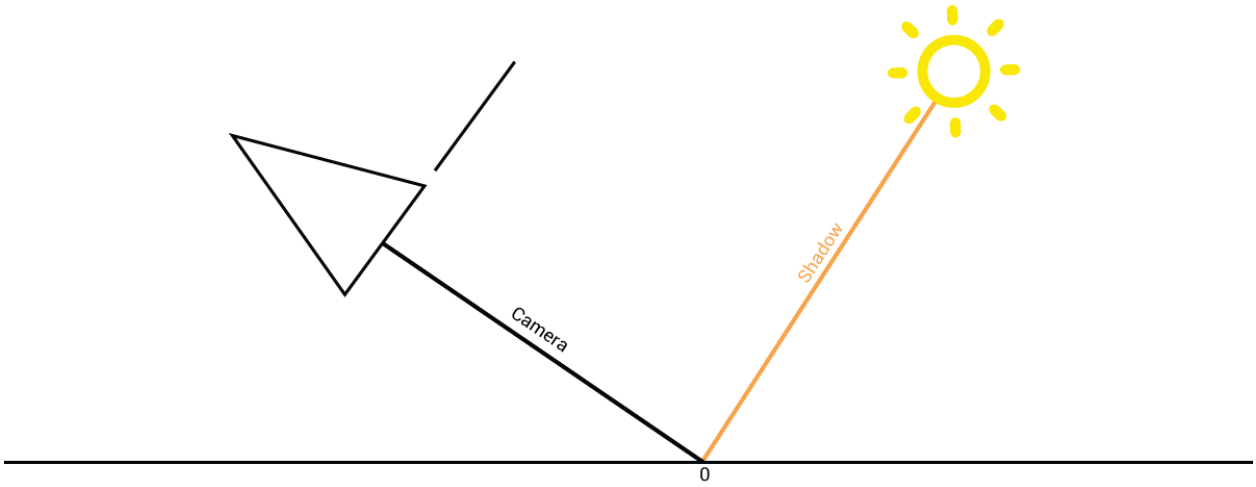


Рис. 1.9 – Теневой луч в направлении источника света

Затем он рассмотрит вероятности отскока луча. В случае с диффузным шейдером все они будут одинаковыми, и ему придется выбрать один из вариантов случайным образом.

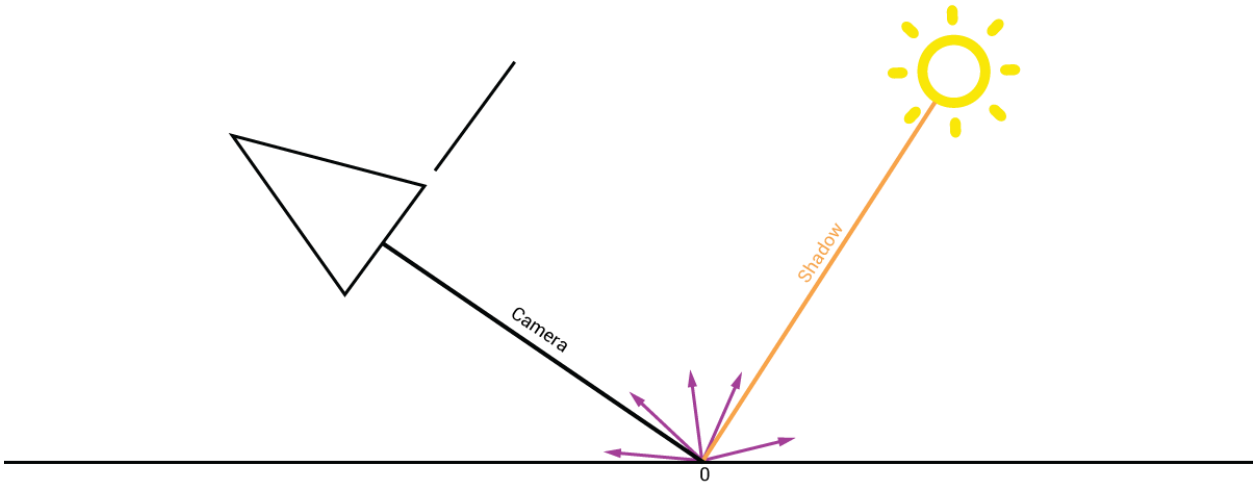


Рис. 1.10 – Вероятность отскока луча от диффузной поверхности во всех направлениях одинаковая

Выбранное направление может быть далеко от источника света, но благодаря теневому лучу, для данного отскока все еще будет храниться информация об источнике освещения.

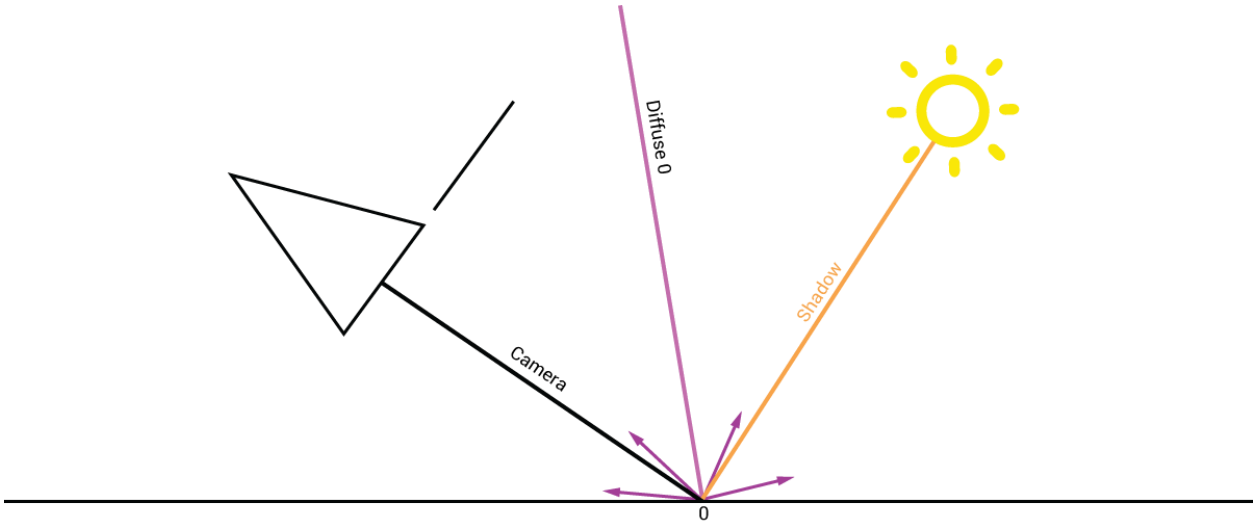


Рис. 1.11 – информация об освещенности данного участка получена с помощью теневого луча

Выборка света (light sampling) — это хороший способ избавления от шума в сцене, потому что практически для каждого прохода луча хранится информация об освещении источником света. Но теневые лучи не являются безошибочными. Бывают случаи, когда они никак не влияют на результат. Один из таких случаев, когда данные лучи блокируются другим объектом, не излучающим свет. Причем тип поверхности не играет никакой роли, за исключением шейдера Transparent, который позволяет лучу проходить беспрепятственно. Именно с помощью данных лучей Cycles определяет места, в которых должна быть тень.

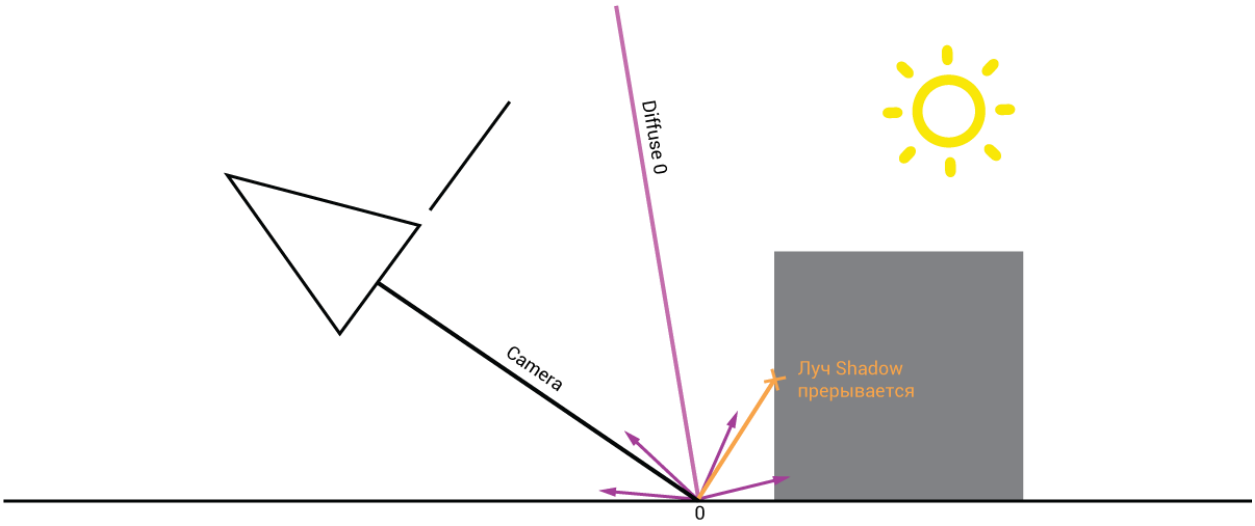


Рис. 1.12 – Теневой луч прерывается, и в данном месте образуется тень

Существует еще одна ситуация, где теневые лучи не смогут предоставить информацию об освещении. Это может произойти в случае, если теневой луч не попадает в диапазон углов отскока от поверхности. В таких случаях они также никак не влияют на результат. Одним из примеров может быть шейдер Glossy с небольшим значением шероховатости.

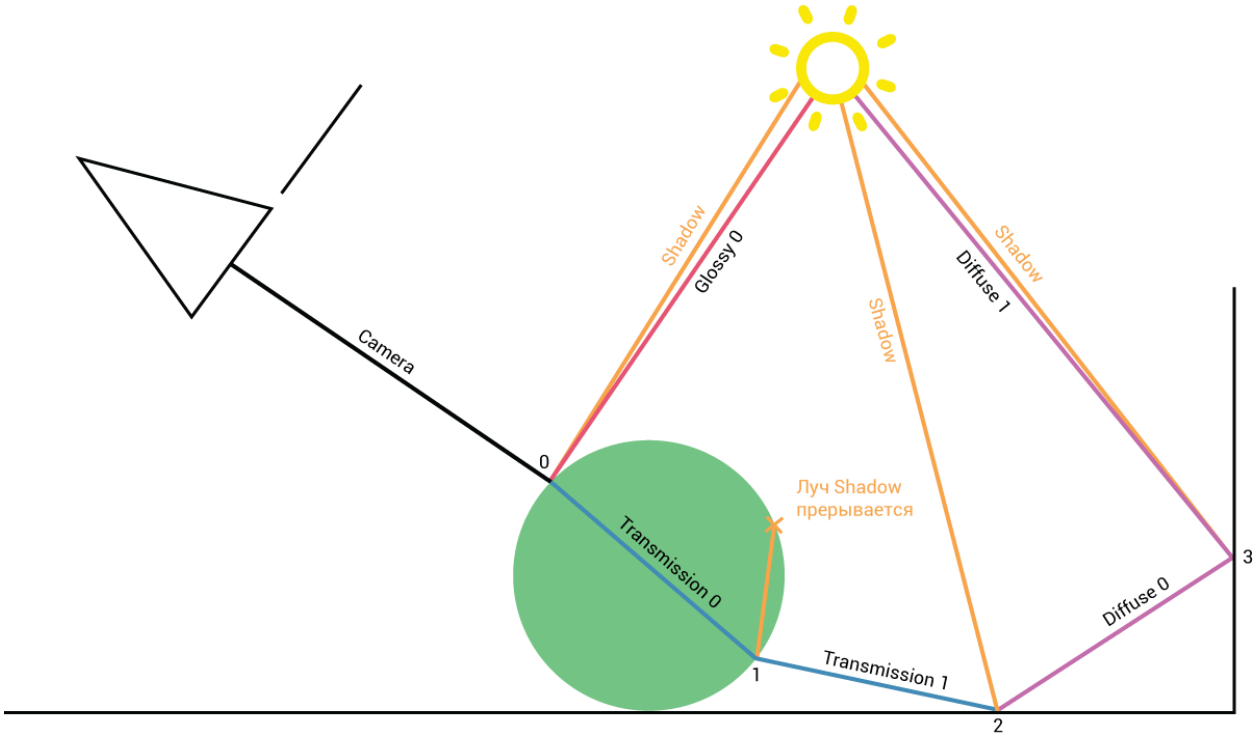


Рис. 1.13 – Во время каждого отскока луча был создан теневой луч

Как видно, теневой луч блокируется на отскоке № 1 и не может хранить информацию об источнике света. Также маловероятно, что при отскоке № 3 луч попадет на источник света, как мы говорили об этом ранее.

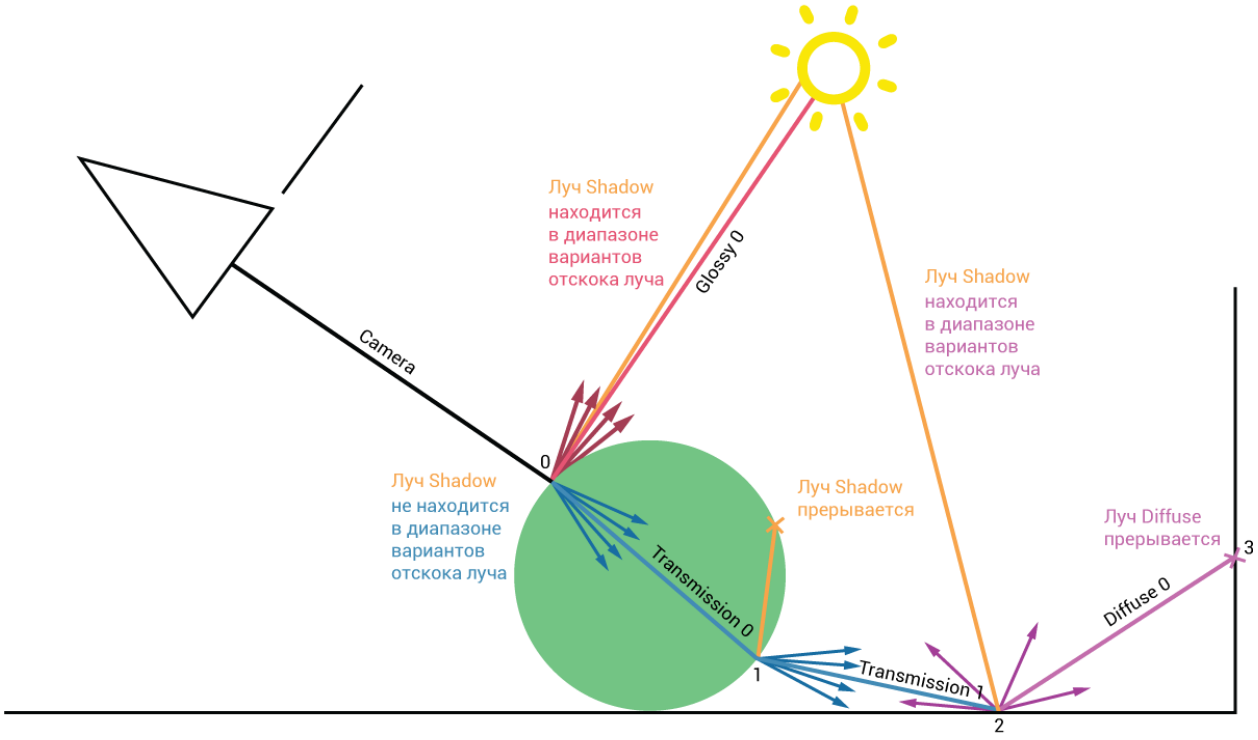


Рис. 1.14 – Варианты отскока луча в зависимости от типа объекта

Глядя на изображение 1.14, можно сказать, что зеркальные отражения и преломления уменьшают вероятность попадания луча на источник света, даже если они двигаются в их направлении. Именно поэтому каустика является наибольшим источником шума в сцене.

# 2. ОСНОВНЫЕ ТЕХНОЛОГИИ ПРОЕКТА

Для создания приложения необходимо изучить и использовать 3 технологии: процедурное текстурирование, параллактическое отображение и PBR текстурирование. Рассмотрим их детальнее.

## 2.1 Процедурное текстурирование

Генерируются процедурные текстуры с помощью математических функций для создания бесшовных текстур. Одним из основных преимуществ данных текстур является возможность масштабировать их сколь угодно и рассматривать с очень близкого расстояния без какой-либо потери качества, которую часто можно наблюдать при использовании изображений. Также данные текстуры являются трехмерными (за исключением текстуры Brick), и для их наложения нет необходимости выполнять UV-развертку.

Текстура волны

Данный нод генерирует два типа текстур: параллельные линии или кольца. Настройки для обоих типов идентичны, и текстуры ведут себя похожим образом.

Позволяет выбирать метод генерации линий/колец.

- Sine — стандартный метод, генерирующий синусоидальные линии.
- Saw — пилообразный метод (sawtooth) генерации линий.

Vector

Определяет метод проецирования текстуры на поверхность. По умолчанию используется метод Generated.

Контролирует толщину полос, не длину, так как она бесконечна. Увеличивая данный параметр, вы создадите большее количество линий.

Искажает полосы, делая их похожими на волны. При больших значениях, полосы будут значительно искажаться и становиться более похожими на текстуру Noise с четкими переходами. При значении 0, следующие две опции не будут иметь никакого значения.

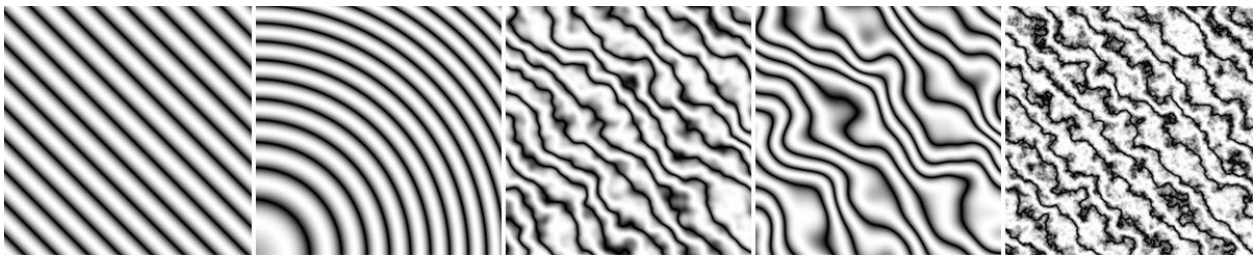

Рис. 2.1 – Процедурная текстура волны

Диаграмма Вороного

Диаграмма Вороного конечного множества точек S на плоскости представляет такое разбиение плоскости, при котором каждая область этого разбиения образует множество точек, более близких к одному из элементов множества S, чем к любому другому элементу множества[7].

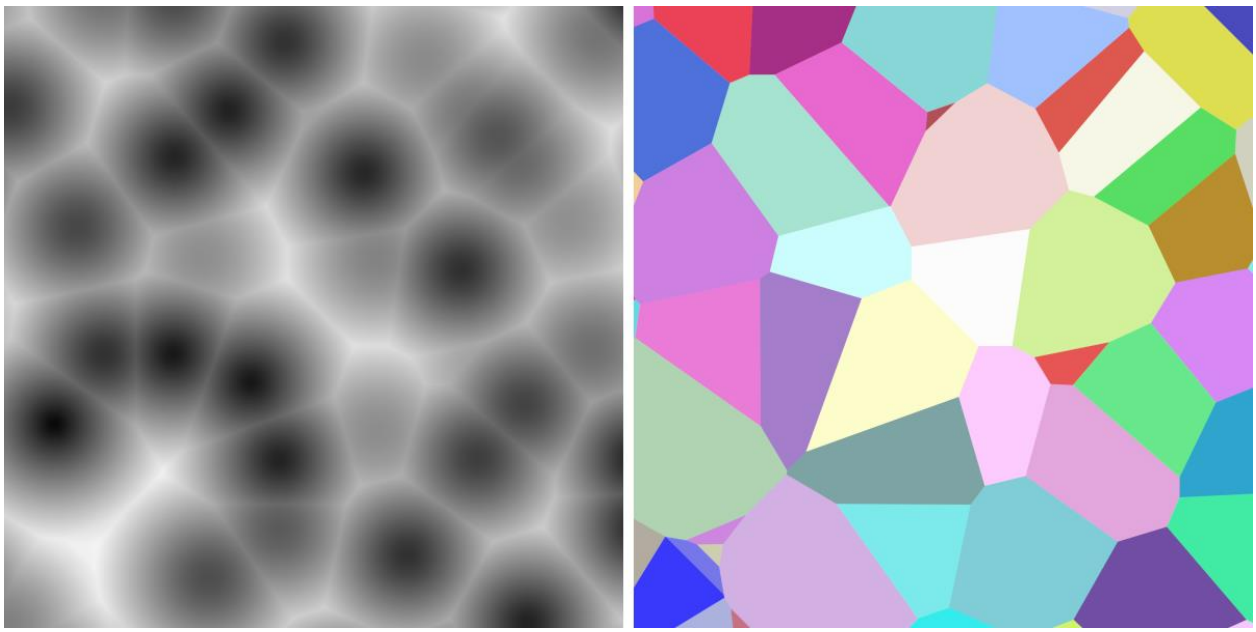

Рис. 2.2 – Процедурная текстура Воронова

Названа в честь российского учёного Георгия Феодосьевича Вороного, который изучил общий n-мерный случай в 1908 году. Также известна как: мозаика Вороного, разбиение Вороного, разбиение Дирихле.

Рассмотрим серединный перпендикуляр отрезка, соединяющего некоторую пару точек p и q.

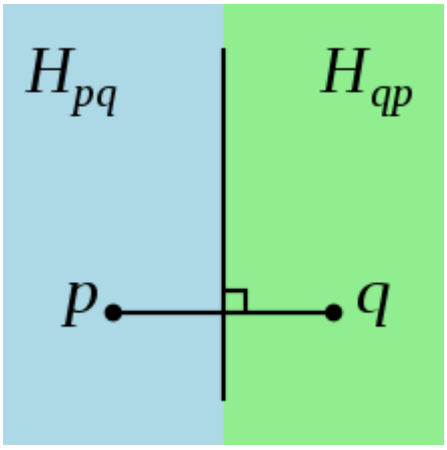


Рис. 2.3 – Диаграмма Вороного для двух точек

Этот перпендикуляр разбивает плоскость на две полуплоскости $H_{pq}$ и $H_{qp}$ причём область Вороного точки p целиком содержится в одной из них, а область точки q — в другой. Область Вороного точкисовпадает с пересечением всех таких полуплоскостей $H_{pq}$:

$$V_p = \bigcap_{q \in S/\{p\}} H_{pq}.$$

Текстура «Масгрейв»

Данный нод был назван в честь создателя данного алгоритма F. Kenton Musgrave. Он генерирует белые участки (своеобразные острова) на черном фоне. В полной мере данный нод раскрывается при смене алгоритмов его работы.

Изначально данная текстура разрабатывалась с целью генерирования ландшафта, где черный цвет задает уровень моря, а градации серого — непосредственно сам ландшафт. При работе с данной текстурой вы часто будете сталкиваться с тем, что некоторые ее параметры работают лишь в определенном диапазоне.

Алгоритм

На данный момент доступно 5 различных алгоритмов генерации островов: fBM, Hetero Terrain, Hybrid Multifractal, Ridged Multifractal, Multifractal.

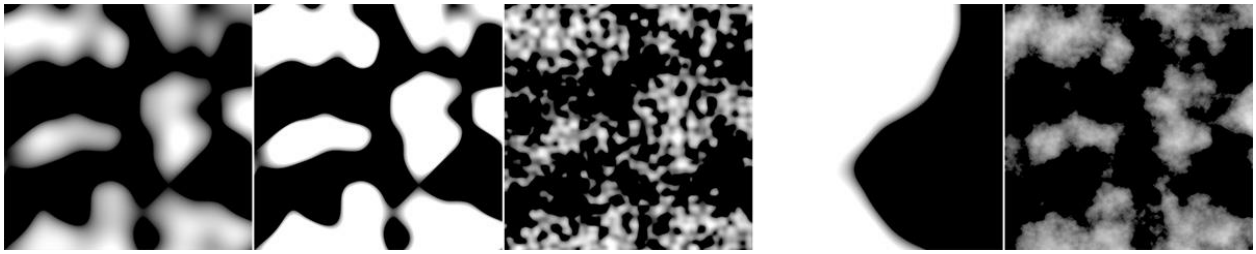

Рис. 2.4 – Процедурная текстура «Масгрейва»

2.2 PBR текстурирование

PBR и PBS это технология шейдинга и визуализации, которая более полноценно и точно описывает-диктует освещение и материалы. Физически корректную визуализацию сокращенно называют PBR (Physically-Based Rendering), а физически корректный шейдинг PBS (Physically-Based Shading).

Главная отличие объекта с использованием PBR и без, это наличие полного набора текстурных карт. В стандартный текстурный лист входят карты: нормалей, отражения, шероховатостей, метала, цвета.

Самой сложной и важной картой для достижения поставленной цели является карта нормалей.

Так что же такое нормали? Давайте немного углубимся в математику. Вы не можете определить угол луча, падающего на поверхность. Если вы приставите карандаш к столу, то как узнать под каким из углов вы это сделали? Если луч падает на поверхность под углом 45°, вы можете измерить как его острый угол (будет равен 45°), так и тупой (будет равен 135°), но вам нужен всего один. Ответом будет «нормаль» вашего стола. Если вы поставите карандаш перпендикулярно вашему столу, то получится угол 90 градусов между ними. Исходя из этого, можно дать определение. Нормаль — это линия перпендикулярная вашей грани. И теперь при наличии двух линий, мы можем измерить угол между ними и определить, под каким углом падает свет на поверхность.

Помимо определения угла падения луча, у нормали есть второе предназначение. С ее помощью можно различать переднюю и заднюю часть грани, потому что нормаль всегда исходит из передней части грани.

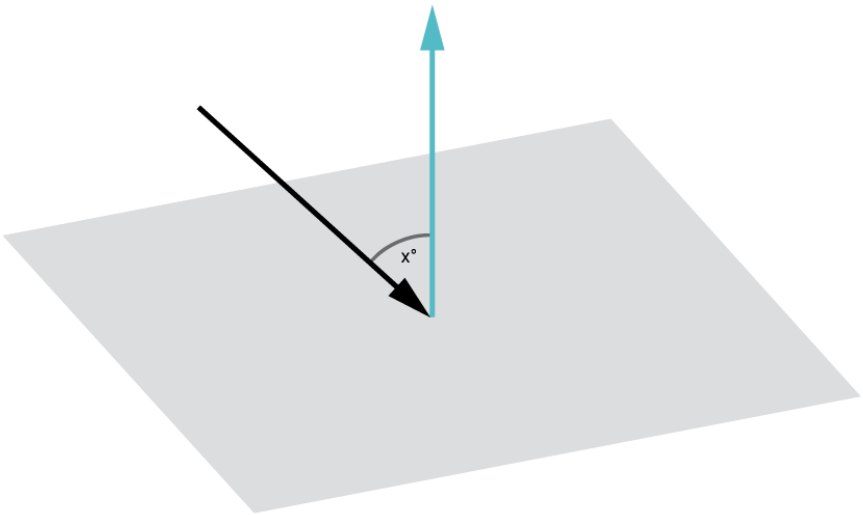


Рис. 2.2 – Определение угла падения луча

Можно манипулировать нормалями, тем самым определяя внешний вид ваших объектов. Именно таким образом создается сглаженное затенение (Smooth Shading). Даже при наличии достаточно жестких переходов между гранями, они могут выглядеть сглажено, благодаря интерполяции нормалей между вершинами.

Таким образом, грани, пересекающиеся под углом 90° могут выглядеть сглажено. В Cycles данный метод используется не только для изменения внешнего вида объектов, но и для изменения пути луча, то есть его отражений и преломлений.

Не только грани имеют нормаль, но и вершины. Для вершин направление нормали определяется с помощью смежных граней. Вы можете визуализировать направление нормалей, перейдя в режим редактирования и отметив соответствующие пункты на панели свойств, в меню Mesh Display:

Как уже было сказано, направление нормалей можно изменять. Смена типов затенения — это один из примеров манипуляции с нормалями. Вы также можете повлиять на направление нормалей, используя текстуру, так

называемую карту нормалей. Три канала цвета данной текстуры будут влиять на три компонента вектора: Красный — X, Зеленый — Y, Синий — Z.

Давайте снова взглянем на математику нормалей. Место падения луча на поверхность называется точкой затенения (Shading Point). Каждая точка затенения имеет лишь одну нормаль. Угол, под которым луч падает на поверхность, рассчитывается на основании угла между направлением луча и нормалью. Помимо этого, настройки материала влияют на поведение луча. Для примера лучшим вариантом будет идеально гладкая поверхность, так как угол отражения от нее всегда равен углу падения. Если вы подключите выход Normal нода Geometry к шейдеру Emission, вы увидите в цветовом представлении нормали вашего объекта. При использовании карты нормалей в вашем материале, угол индивидуальных точек затенения может быть представлен цветовой информацией, соответствующей осям XYZ (Красный — X, Зеленый — Y, Синий — Z). Таким образом, поверхность как бы притворяется и указывает другое направление луча, отличное от угла падения. Но у данного метода есть один недостаток. Так как на самом деле поверхность плоская и не имеет никаких неровностей, то выглядеть данная поверхность будет хорошо лишь под прямым углом.

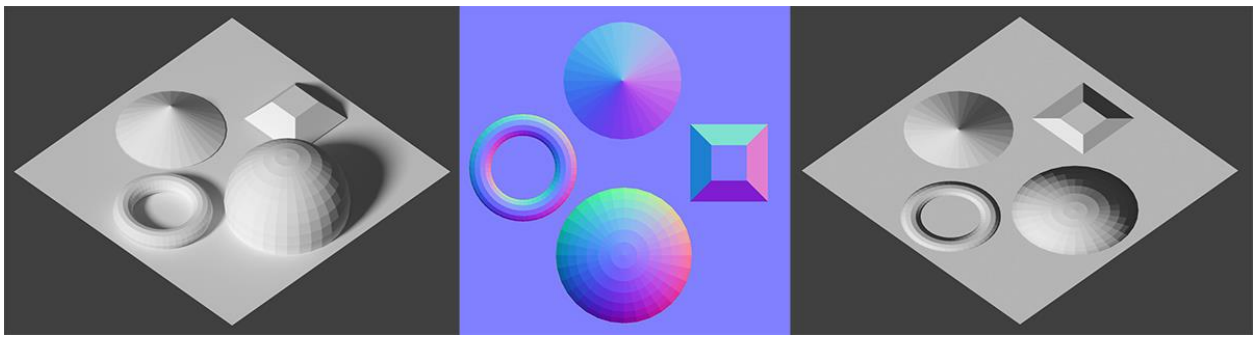

Рис. 2.2 – Слева — исходный объект. По центру — карта нормалей, созданная для объекта слева. Справа — плоскость с примененной к ней картой нормалей

## 2.3 Параллактическое отображение

Parallax mapping («параллактическое отображение», также известен как offset mapping, per-pixel displacement mapping или virtual displacement mapping) — программная техника (методика) в трёхмерной компьютерной графике, усовершенствованный вариант техник bump mapping или normal mapping. Parallax mapping используется для процедурного создания трёхмерного описания текстурированной поверхности с использованием карт смещения (не путать с Displacement mapping (англ.)русск.) вместо непосредственного генерирования новой геометрии. Методику «Parallax mapping» условно можно назвать «2.5D», так как она позволяет добавлять трёхмерную сложность в текстуры, не создавая реальные трёхмерные графические структуры. Например, текстура каменной стены будет иметь визуальную объемность, хотя на самом деле геометрически она будет плоской. Parallax mapping был представлен Томомити Канэко (англ. Tomomichi Kaneko) в 2001 году[1]. Parallax mapping полностью исполняется на графических процессорах видеокарты как пиксельный шейдер.

Parallax mapping осуществляется смещением текстурных координат так, чтобы поверхность казалась объёмной[2]. Главное отличие parallax mapping от displacement mapping в том, что в нём все расчеты попиксельные, а не повершинные. Идея метода состоит в том, чтобы возвращать текстурные координаты той точки, где видовой вектор пересекает поверхность. Это требует просчета лучей (рейтрейсинг) для карты высот, но если она не имеет слишком сильно изменяющихся значений («гладкая» или «плавная»), то можно обойтись аппроксимацией без использования рейтрейсинга. Если же в parallax mapping используется рейтрейсинг, то такой вариант называется «Parallax occlusion mapping».

Таким образом, parallax mapping хорош для поверхностей с плавно изменяющимися высотами, без просчета пересечений и больших значений смещения. Подобный простой алгоритм отличается от normal mapping всего

тремя инструкциями пиксельного шейдера: две математические инструкции и одна дополнительная выборка из текстуры. После того, как вычислена новая текстурная координата, она используется дальше для чтения других текстурных слоев: базовой текстуры, карты нормалей и т. п. Такой метод parallax mapping на современных графических процессорах почти так же эффективен, как обычное наложение текстур, а его результатом является более реалистичное отображение поверхности по сравнению с простым normal mapping.

# 3. ПРАКТИЧЕСКАЯ ЧАСТЬ

## 3.1 Нодовая система разработки

Большинство нодов имеют входные и выходные гнезда (сокеты), которые окрашены в соответствующие цвета, чтобы было гораздо проще определить какую информацию он передает, либо что следует к нему подключать. Данные правила не являются непоколебимыми и во многих случаях вы можете соединять входы и выходы, имеющие различные цвета, но чаще всего подключение происходит в соответствии с цветовыми индикаторами.

Синий

Синим цветом обозначаются вектора. Другими словами, данные сокеты содержат три значения, представляющих оси X, Y и Z. Если их использовать в качестве цветовой информации, то оси X, Y и Z будут соответствовать R (red) G (green) и B (blue).

Серый

Серым цветом обозначаются входы и выходы с числовыми значениями. Часто это текстуры в градациях серого, либо просто числовые значения.

Желтый

Желтым цветом обозначаются сокеты цветовой информации. Это могут быть цвета вершин или текстуры.

Зеленый

Зеленым обозначаются шейдерные сокеты. Все шейдерные ноды имеют зеленый выход и лишь ноды Mix, Add и Output имеют входные зеленые

сокеты. Если вы подключите что-либо отличное от шейдеров к зеленым сокетам, то на выходе получите черный материал.

Для работы с нодами вам необходимо открыть окно редактора нодов (Node Editor) или же переключиться на рабочее пространство Compositing. Сделать это можно с помощью сочетания клавиш Ctrl + стрелка влево, либо выбрать это пространство из списка (рис 1.21 № 1). По умолчанию это окно настроено на работу с нодами пост-обработки. С их помощью производится обработка уже готовых изображений, что не является темой данной книги. Внизу окна (рис 1.21 № 2) вы можете выбрать какие ноды использовать: шейдерные (shader nodes), пост-обработки (compositing nodes) или текстурные (texture nodes). Часть нодов идентичны для всех трех режимов, но в данной книге мы будем рассматривать работу лишь в режиме Shader nodes.

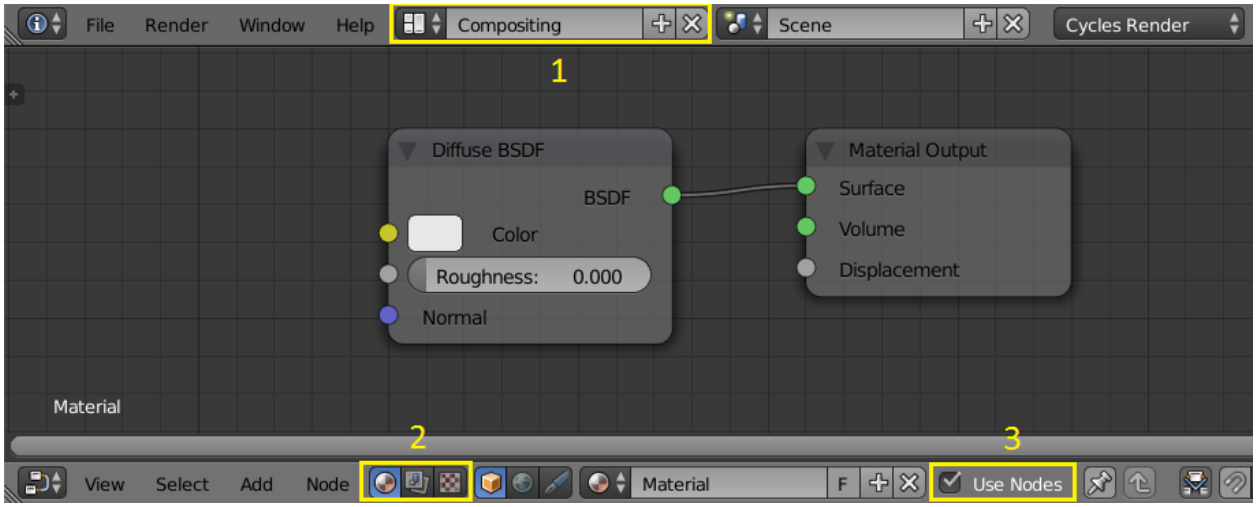


Рис. 3.1 – Редактор нодов Blender

Связку из нескольких нодов часто называют картой нодов или деревом нодов. Так или иначе, это своеобразный конвейер, на который подаются исходные данные, которые обрабатываются, и на выходе получаем результат. Каждый нод имеет различные входные параметры, обрабатывает их по заданному алгоритму и на выходе дает результат.

Непосредственно для начала использования нодов вам понадобится объект и назначенный для него материал. Если оба этих пункта у вас выпол-

нены, но вы по-прежнему не видите ни одного нода, в таком случае вам необходимо отметить пункт "Use Nodes" (рис 1.21 № 3). Материалом по умолчанию в Cycles является диффузный шейдер, подключенный к ноду Output.

Проще всего добавить новый нод с помощью сочетания клавиш Shift + A. После этого, с помощью мышки, выбрать нужную категорию в появившемся меню и нужный нод. Появившееся окно не пропадет, пока вы не выберите нод или пока не уведете мышку за его пределы. В данном меню вы можете заметить подчеркнутые буквы в названиях категорий и шейдеров. Это горячие клавиши, с помощью которых их можно выбрать. Например, чтобы быстро выбрать шейдер Diffuse, необходимо нажать Shift + A — H — D. После выбора нода, он будет перемещаться за курсором мышки до тех пор, пока вы не подтвердите его местоположение, с помощью левой кнопки мышки. Чтобы соединять ноды между собой, вы можете зажать левую кнопку мышки и потянуть выход одного нода к входу другого, или выделить оба нода с зажатой клавишей Shift, и затем нажать клавишу F. Данное действие соединит первые два сокета с одинаковым именем. Например, если первый выход нода назван Color, то Blender будет пытаться найти такой же вход у другого нода.

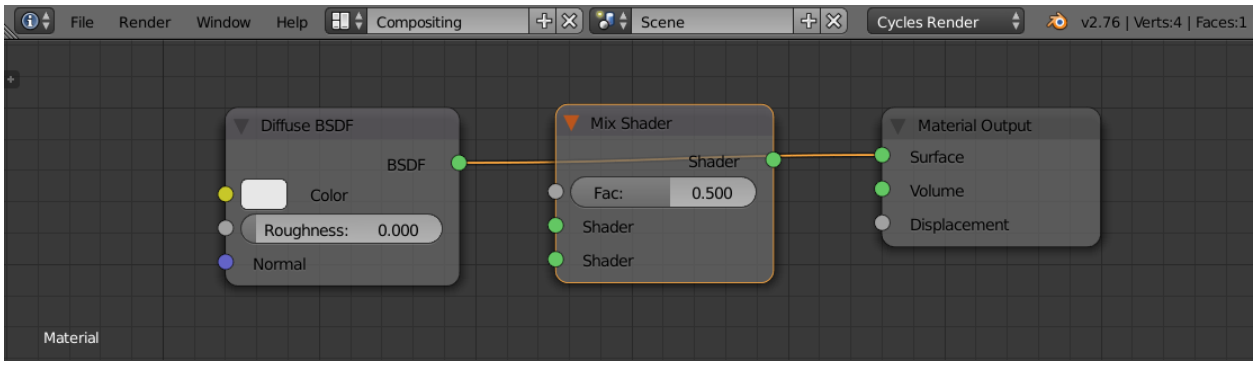


Рис. 3.2 – Соединение нодов Blender

Если два нода уже соединены между собой, вы можете расположить третий посреди них, и он автоматически соединится с двумя соседними нодами. Для этого необходимо навести третий нод на связь, соединяющую два

нода, и когда она подсветится оранжевым цветом, подтвердить перемещение. С выходом версии Blender 2.76 появилась новая опция Auto-offset, позволяющая отодвинуть ноды вправо или влево для вставки нового между ними.

## 3.2 Ноды из категории script

Данная категория содержит всего один нод, но здесь больше и не нужно, так как с помощью скриптов он становится настолько универсальным, что может заменить собой все остальные ноды.

Язык, на котором данный нод понимает скрипты называется Open Shading Language (OSL). Чтобы использовать данный нод, вам необходимо переключиться на вычислительное устройство CPU (видеокарта не способна на данный момент рендерить OSL) и отметить пункт Open Shading Language в меню Render.

Данный нод поддерживает работу скриптов, как написанных непосредственно в самом Blender, так и возможность подключения внешних файлов с расширением .osl. С помощью скриптов вы можете указать произвольное количество входных и выходных значений, для связи этого нода с другими. Если произведенные изменения скрипта не отображаются на рендере, необходимо перезагрузить скрипт, нажав на кнопку Refresh.

Данный нод имеет всего две кнопки:

- Internal — OSL-скрипт присутствует в текстовом редакторе Blender.
- External — OSL-скрипт находится в файле на жестком диске компьютера.

Определившись с методом, скопируйте приведенный ниже простой скрипт в ваш файл или текстовый редактор Blender, и подключите получившийся нод к шейдеру Diffuse.

```
shader darken(
  color In = 1,
  output color Out = 1
){
  Out = In * 0.5;
}
```

В первой строке объявляется шейдер, который называется darken. Шейдер выглядит как функция с параметрами, и во второй строке определяется входной параметр In. Тип этого параметра — цвет (color, один из основных типов в OSL), и его значение по умолчанию равно 1. Это означает, что все три компонента цвета (красный, зеленый и синий) будут установлены в значение 1, что делает цвет белым.

Следующая строчка определяет параметр Out, который также имеет тип Color, и его значение по умолчанию равно 1. Создали мы этот выходной параметр при помощи ключевого слова output. Когда шейдер компилируется, Blender создает соответствующие сокеты с обеих сторон нода (входные — слева, выходные — справа). Также он окрашивает сокеты в соответствующие цвета. В случае с параметром color, сокеты будут желтыми.

Все параметры должны быть определены по умолчанию. Если к входному параметру нода не будет подключен другой нод, то будет использоваться значение, определенное по умолчанию. Также если ничего не подклю-

чено к входу, то данное значение может быть изменено пользователем. В зависимости от типа сокета, соответствующий редактор появится при клике по параметру (например, колесо выбора цвета, если это цвет). Значение не подключенного сокета может быть даже анимировано (при помощи горячей клавиши I), как и практически все в Blender.

Значение по умолчанию для выходного сокета используется в том случае, если нигде в теле шейдера оно не было изменено.

Тело этого шейдера состоит из одной строки (строка 5), в которой входной параметр (In) умножается на 0.5, и результат присваивается параметру Out.

Данный шейдер слишком прост и не показывает многих возможностей, но он дает четкое представление того, как много работы берет на себя сам Blender: создает нужные сокеты в нужных местах, окрашивает их в соответствующий цвет и помогает легко изменять параметры шейдера.

## 3.3 Группировка нодов

Редактор нодов в Blender позволяет вам создавать группы нодов. Для этого нужно выделить ноды, которые вы хотите объединить в группу, и нажать сочетание клавиш Ctrl + G. Выделять ноды можно с помощью прямоугольного выделения (B), кругового (C) или с зажатой клавишей Shift.

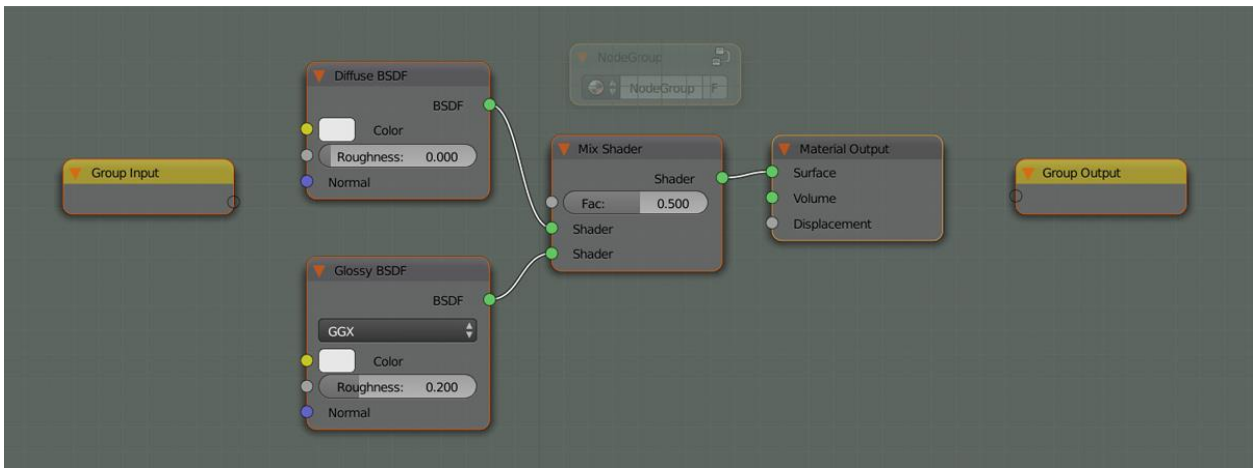


Рис. 3.3 – Группа нодов

Переключение между группами нодов производится с помощью горячей клавиши Tab. Вы можете создавать группы нодов в группах нодов сколько угодно раз. Но практической пользы в этом чаще всего нет и вы лишь запутаете сами себя и, возможно, других. Поэтому старайтесь избегать вложений глубже, чем на два уровня.

Это очень удобно, если вам нужно использовать одну группу нодов с различными материалами. Также изменив настройки всего в одном месте, вы поменяете их во всех материалах, использующих данную группу нодов. Вы можете импортировать группы нодов из других файлов, без необходимости переносить весь материал целиком.

После того, как вы объедините ноды в группу, фон немного затемнится и появятся два дополнительных нода (Group Input и Group Output), отвечающих за входные и выходные значения созданной вами группы. Если вы потяните один из сокетов к данным нодам, он к ним подключится, и образовавшийся сокет получит название того параметра, который вы к нему только что подключили. Если данное название вас не устраивает, вы всегда можете изменить его на панели свойств (N), а также реорганизовать порядок, созданных вами сокетов. Помимо названия и порядка, вы можете задать значения по умолчанию для входных значений, а также установить лимиты, чтобы

пользователь не смог ввести те значения, которые не предусмотрены для данной группы нодов.

Очень часто подобные группы создаются не только для личного использования, а еще и для того, чтобы поделиться с кем-то собственным материалом, и при этом упростить работу с ним, или для продажи на стоках. Таким образом, пользователю не придется вникать в ваше дерево нодов, он сможет тут же начать с ним работать, а также не сможет получить непредвиденных результатов.

Помимо опций Make Group и Ungroup (Shift + A > Group), можно заметить линию, под которой будут появляться все созданные вами группы нодов. Сортироваться они будут в алфавитном порядке. При использовании двух групп с одинаковым названием, горячим сочетанием клавиши будет вызываться вторая созданная группа.

## 3.4 Программирование на языке Pyton

Python в русском языке распространено название — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное программирование. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединяться в пакеты.

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ[11]. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные[12]. Есть реализация интерпретатора для JVM с возможностью компиляции, CLR, LLVM, другие независимые реализации. Проект PyPy использует JIT-компиляцию, которая значительно увеличивает скорость выполнения Python-программ.

Python — активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года. Язык не подвергался официальной стандартизации, роль стандарта де-факто выполняет CPython, разрабатываемый под контролем автора языка.

Python поддерживает динамическую типизацию, то есть тип переменной определяется только во время исполнения. Поэтому вместо «присваивания значения переменной» лучше говорить о «связывании значения с некоторым именем». В Python имеются встроенные типы: булевый, строка, Unicode-строка, целое число произвольной точности, число с плавающей запятой, комплексное число и некоторые другие. Из коллекций в Python встроены: список, кортеж (неизменяемый список), словарь, множество и другие[28]. Все значения являются объектами, в том числе функции, методы, модули, классы.

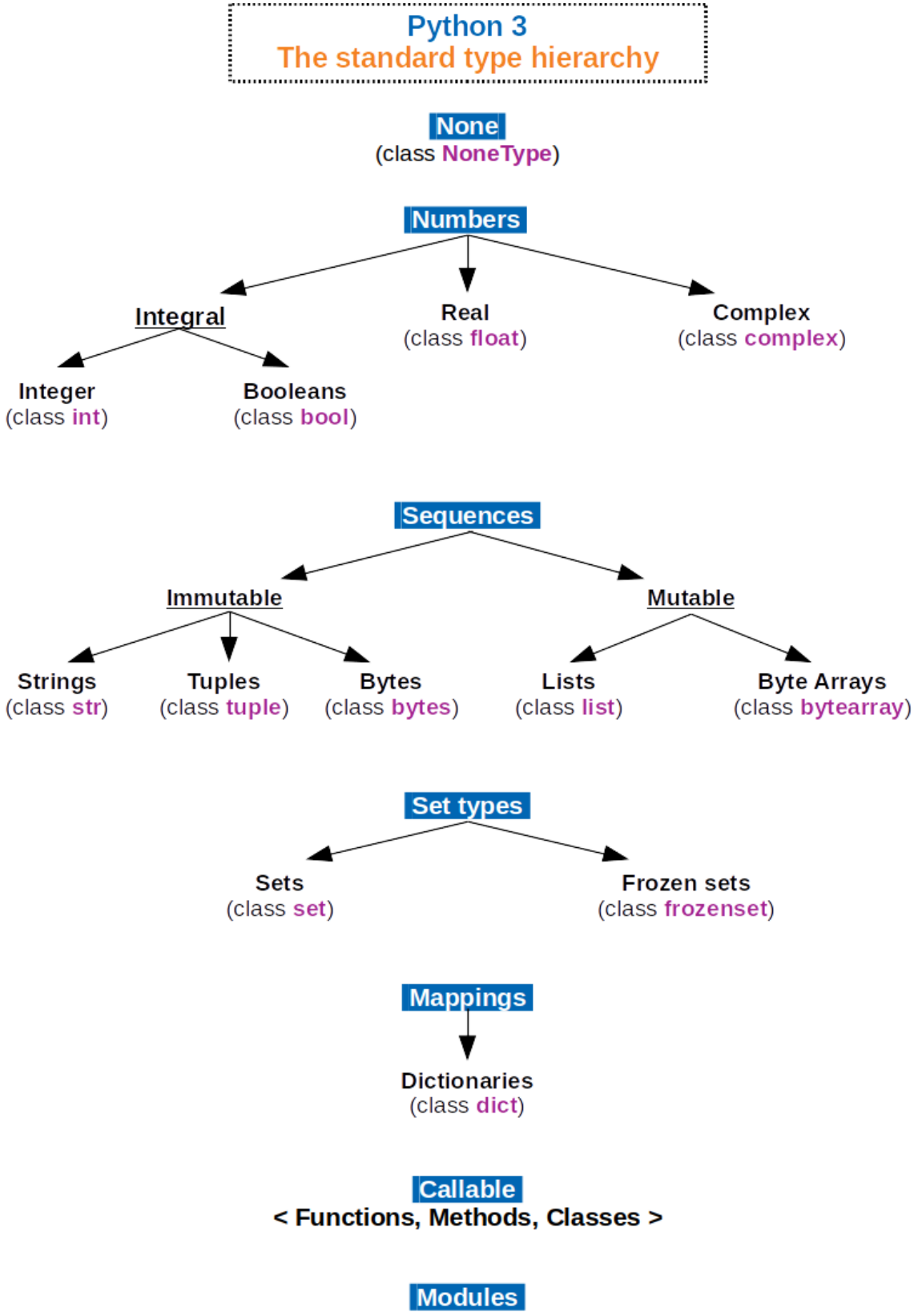


Рис. 3.4 – Типизация данных

Добавить новый тип можно либо написав класс (class), либо определив новый тип в модуле расширения (например, написанном на языке C). Система классов поддерживает наследование (одиночное и множественное) и метапрограммирование. Возможно наследование от большинства встроенных типов и типов расширений.

Все объекты делятся на ссылочные и атомарные. К атомарным относятся int, long (в версии 3 любое число int, так как в версии 3 нет ограничения на размер), complex и некоторые другие. При присваивании атомарных объектов копируется их значение, в то время как для ссылочных копируется только указатель на объект, таким образом, обе переменные после присваивания используют одно и то же значение. Ссылочные объекты бывают изменяемые и неизменяемые. Например, строки и кортежи являются неизменяемыми, а списки, словари и многие другие объекты — изменяемыми. Кортеж в Python является, по сути, неизменяемым списком. Во многих случаях кортежи работают быстрее списков[19], поэтому если вы не планируете изменять последовательность, то лучше использовать именно их.

## 3.5 Создание аддона для Blender

Система скриптов в Blender предоставляет широкие возможности по упрощению и ускорению рабочего процесса, позволяя перекладывать выполнение рутинных операций на систему и расширяя возможности работы благодаря доступу к скриптовому языку. Однако, написав удачный скрипт, который будет часто использоваться в разных проектах, неудобно каждый раз заново подключать его в каждый новый проект. К тому же такой скрипт вполне может потребовать улучшений в виде окон и полей с редактируемыми параметрами. Превратив скрипт в полноценный аддон, можно оснастить

его дополнительным функционалом и подключить в систему дополнений Blender.

Возьмем простой скрипт создания трубы:

```
import math
import bpy

n = 8
r1 = 1
r2 = 0.5
h = 1

verts = []
ring1 = []
ring2 = []
ring3 = []
ring4 = []
faces = []

for i in range(n):
    grad = (360 * i / n) * math.pi / 180
    ring1.append([r1 * math.cos(grad), r1 * math.sin(grad), 0])
    ring2.append([r2 * math.cos(grad), r2 * math.sin(grad), 0])
    ring3.append([r1 * math.cos(grad), r1 * math.sin(grad), h])
    ring4.append([r2 * math.cos(grad), r2 * math.sin(grad), h])
    if i == 0:
        faces.append([i - 1 + n, i, i + 2 * n, i - 1 + 3 * n])
        faces.append([i - 1 + 2 * n, i - 1 + 4 * n, i + 3 * n, i + n])
        faces.append([i - 1 + n, i - 1 + 2 * n, i + n, i])
        faces.append([i - 1 + 3 * n, i + 2 * n, i + 3 * n, i - 1 + 4 *
n])
    else:
        faces.append([i - 1, i, i + 2 * n, i - 1 + 2 * n])
        faces.append([i - 1 + n, i - 1 + 3 * n, i + 3 * n, i + n])
        faces.append([i - 1, i - 1 + n, i + n, i])
        faces.append([i - 1 + 2 * n, i + 2 * n, i + 3 * n, i - 1 + 3 *
n])

verts.extend(ring1)
verts.extend(ring2)
verts.extend(ring3)
verts.extend(ring4)

tubeMesh = bpy.data.meshes.new("Tube")
tubeMesh.from_pydata(verts, [], faces)
tubeMesh.update()
tube = bpy.data.objects.new("Tube", tubeMesh)
bpy.context.scene.objects.link(tube)
bpy.ops.object.select_all(action="DESELECT")
tube.select = True
bpy.context.scene.objects.active = tube
```

Рис. 3.5 – Скрипт создания трубы

Дополним его функционал до полноценного аддона.

Для превращения скрипта в аддон Blender нужно выполнить 4 обязательных требования Blender API:

В первую очередь весь код, который на данный момент представляет собой простую последовательность команд, нужно обернуть в класс. API требует, чтобы пользовательские классы обязательно наследовались от нескольких предопределенных классов:

```
Python
1 bpy.types.Panel
2 bpy.types.Menu
3 bpy.types.Operator
4 bpy.types.PropertyGroup
5 bpy.types.KeyingSet
6 bpy.types.RenderEngine
```

Рис. 3.6 – Пользовательские классы

Так же API требует, чтобы все пользовательские классы были статическими, поэтому в них не требуется определять конструктор __init__ и деструктор __del__.

В нашем примере от аддона требуется выполнить действие по созданию меша. В классе bpy.types.Operator определена функцию execute, которая выполняется в момент обращения к классу через API Blender — как раз то, что нужно для выполнения какого-то определенного действия. Поэтому оборачиваем код скрипта в класс и наследуем его от bpy.types.Operator. Переопределяем функцию execute и заносим в нее весь исполняемый код нашего скрипта. Функция execute должна возвращать указание о успешном выполнении {‘FINISHED’}:

```python
import math
import bpy

class createTube(bpy.types.Operator):

    def execute(self, context):
        n = 8
        r1 = 1
        r2 = 0.5
        h = 1

        verts = []
        ring1 = []
        ring2 = []
        ring3 = []
        ring4 = []
        faces = []

        for i in range(n):
            grad = (360 * i / n) * math.pi / 180
            ring1.append([r1 * math.cos(grad), r1 * math.sin(grad), 0])
            ring2.append([r2 * math.cos(grad), r2 * math.sin(grad), 0])
            ring3.append([r1 * math.cos(grad), r1 * math.sin(grad), h])
            ring4.append([r2 * math.cos(grad), r2 * math.sin(grad), h])
            if i == 0:
                faces.append([i - 1 + n, i, i + 2 * n, i - 1 + 3 * n])
                faces.append([i - 1 + 2 * n, i - 1 + 4 * n, i + 3 * n, i + n])
                faces.append([i - 1 + n, i - 1 + 2 * n, i + n, i])
                faces.append([i - 1 + 3 * n, i + 2 * n, i + 3 * n, i - 1 + 4 * n])
            else:
                faces.append([i - 1, i, i + 2 * n, i - 1 + 2 * n])
                faces.append([i - 1 + n, i - 1 + 3 * n, i + 3 * n, i + n])
                faces.append([i - 1, i - 1 + n, i + n, i])
                faces.append([i - 1 + 2 * n, i + 2 * n, i + 3 * n, i - 1 + 3 * n])

        verts.extend(ring1)
        verts.extend(ring2)
        verts.extend(ring3)
        verts.extend(ring4)

        tubeMesh = bpy.data.meshes.new("Tube")
        tubeMesh.from_pydata(verts, [], faces)
        tubeMesh.update()
        tube = bpy.data.objects.new("Tube", tubeMesh)
        bpy.context.scene.objects.link(tube)
        bpy.ops.object.select_all(action="DESELECT")
        tube.select = True
        bpy.context.scene.objects.active = tube
        return {'FINISHED'}
```

Рис. 3.7 – Фрагмент кода

Класс, наследующий bpy.types.Operator становится полноправным оператором Blender API.

Кроме функции execute операторы имеют следующие предопределенные переопределяемые функции:

poll — выполняется перед выполнением самого оператора, и если в ходе ее выполнения произошла ошибка — сам оператор далее не выполняется

invoke — используется для интерактивных операций, таких как перетаскивание элементов

draw — вызывается для создания графических элементов, панелей

modal — используется в операторах, которые требуют периодического вызова, например при разрезании нескольких ребер. Не прерывают работу после возвращения {‘FINISH’}

cancel — вызывается при отмене выполнения оператора

Для того, чтобы созданный класс подключился к Blender API, нужно его зарегистрировать, вызвав функцию bpy.utils.register_class(), указав в ее параметрах имя регистрируемого класса:

При завершении работы зарегистрированный класс требуется разрегистрировать:

Так как мы оформляем наш класс в виде аддона, операцию регистрации класса нужно выполнить в момент инициализации аддона, а операцию разрегистрации класса — в момент его отключения. Для этого в API предусмотрено следующее условие: если в коде аддона определены функции register и unregister, они будут вызваны: первая при подключении аддона, вторая — при его отключении.

Добавим в конец кода нашего аддона эти две функции и проведем в них соответственно регистрацию и разрегистрацию нашего класса:

```
import math
import bpy

class createTube(bpy.types.Operator):

    def execute(self, context):
        n = 8
        r1 = 1
        r2 = 0.5
        h = 1

        verts = []
        ring1 = []
        ring2 = []
        ring3 = []
        ring4 = []
        faces = []

        for i in range(n):
            grad = (360 * i / n) * math.pi / 180
            ring1.append([r1 * math.cos(grad), r1 * math.sin(grad), 0])
            ring2.append([r2 * math.cos(grad), r2 * math.sin(grad), 0])
            ring3.append([r1 * math.cos(grad), r1 * math.sin(grad), h])
            ring4.append([r2 * math.cos(grad), r2 * math.sin(grad), h])
            if i == 0:
                faces.append([i - 1 + n, i, i + 2 * n, i - 1 + 3 * n])
                faces.append([i - 1 + 2 * n, i - 1 + 4 * n, i + 3 * n,
i + n])
                faces.append([i - 1 + n, i - 1 + 2 * n, i + n, i])
                faces.append([i - 1 + 3 * n, i + 2 * n, i + 3 * n, i -
1 + 4 * n])
            else:
                faces.append([i - 1, i, i + 2 * n, i - 1 + 2 * n])
                faces.append([i - 1 + n, i - 1 + 3 * n, i + 3 * n, i +
n])
                faces.append([i - 1, i - 1 + n, i + n, i])
                faces.append([i - 1 + 2 * n, i + 2 * n, i + 3 * n, i -
1 + 3 * n])

        verts.extend(ring1)
        verts.extend(ring2)
        verts.extend(ring3)
        verts.extend(ring4)

        tubeMesh = bpy.data.meshes.new("Tube")
        tubeMesh.from_pydata(verts, [], faces)
        tubeMesh.update()
        tube = bpy.data.objects.new("Tube", tubeMesh)
        bpy.context.scene.objects.link(tube)
        bpy.ops.object.select_all(action="DESELECT")
        tube.select = True
        bpy.context.scene.objects.active = tube
        return {'FINISHED'}
```

Рис. 3.8 – Фрагмент кода

Любой класс, зарегистрированный в Blender API, должен иметь уникальный идентификатор, чтобы можно было к нему обращаться по этому идентификатору.

В качестве идентификатора необходимо использовать строковую константу с предопределенным именем bl_idname. Такая константа обязана присутствовать в любом подключаемом к API Blender классе. Кроме обязательного предопределенного имени константы к ней есть еще одно требование — в ее значении должна присутствовать точка. Скорее всего это сделано для более удобной группировки идентификаторов классов внутри API.

Для определения идентификатора нашего класса создадим такую переменную и присвоим ей значение ‘mesh.create_tube’.

Определим необходимые константы в классе нашего аддона:

```python
import math
import bpy

class createTube(bpy.types.Operator):
    bl_idname = 'mesh.create_tube'
    bl_label = 'Create Tube'

    def execute(self, context):
        n = 8
        r1 = 1
        r2 = 0.5
        h = 1

        verts = []
        ring1 = []
        ring2 = []
        ring3 = []
        ring4 = []
        faces = []

        for i in range(n):
            grad = (360 * i / n) * math.pi / 180
            ring1.append([r1 * math.cos(grad), r1 * math.sin(grad), 0])
            ring2.append([r2 * math.cos(grad), r2 * math.sin(grad), 0])
            ring3.append([r1 * math.cos(grad), r1 * math.sin(grad), h])
            ring4.append([r2 * math.cos(grad), r2 * math.sin(grad), h])
            if i == 0:
                faces.append([i - 1 + n, i, i + 2 * n, i - 1 + 3 * n])
                faces.append([i - 1 + 2 * n, i - 1 + 4 * n, i + 3 * n, i + n])
                faces.append([i - 1 + n, i - 1 + 2 * n, i + n, i])
                faces.append([i - 1 + 3 * n, i + 2 * n, i + 3 * n, i - 1 + 4 * n])
            else:
                faces.append([i - 1, i, i + 2 * n, i - 1 + 2 * n])
                faces.append([i - 1 + n, i - 1 + 3 * n, i + 3 * n, i + n])
                faces.append([i - 1, i - 1 + n, i + n, i])
                faces.append([i - 1 + 2 * n, i + 2 * n, i + 3 * n, i - 1 + 3 * n])

        verts.extend(ring1)
        verts.extend(ring2)
        verts.extend(ring3)
        verts.extend(ring4)

        tubeMesh = bpy.data.meshes.new("Tube")
        tubeMesh.from_pydata(verts, [], faces)
        tubeMesh.update()
        tube = bpy.data.objects.new("Tube", tubeMesh)
        bpy.context.scene.objects.link(tube)
        bpy.ops.object.select_all(action="DESELECT")
```

Рис. 3.9 – Фрагмент кода

Для окончательного оформления аддона нужно сделать его описание. Это требование не является в полной мере обязательным, но его все равно стоит выполнять. Написав несколько аддонов без описаний, даже их автору очень сложно разобраться в их назначении. Для составления описания аддона служит словарь с предопределенным именем bl_info, который имеет следующие предопределенные пункты:

name — название аддона

author — ФИО автора

version — версия аддона

blender — версия Blender под которую разрабатывался аддон

category — категория, в которую аддон будет помещен

location — указание на то, где искать панель аддона

url — указание на исходный код аддона (откуда он распространяется)

description — строка с более подробным описанием аддона

По минимуму стоит заполнять пункты name, category и blender.

## 3.6 Процесс работы реализованного аддона Blender

С использованием вышеперечисленных данных был разработан аддон для генерации растений: деревьев, плющей, травы. Растения могут превратить плоское изображение в полноценную работу, но созданный вручную плющ может стать ночным кошмаром, даже для 3d художника эксперта. Этот плагин представляет собой процедурную систему, которая генерирует реалистичную растительность на вашем меше. Это не симуляция биологического

роста растительности, однако он генерирует комплекс правдоподобных растений.

Для примера работы рассмотрим создание плюща на поверхности 3D объекта. Для этого после установки аддона в Blender нужно создать любой 3Д объект, который мы будем использовать как основу для роста растений и выбрать любую вершину. Вершина будет являться началом плюща. Плющ будет расти из этой точки.

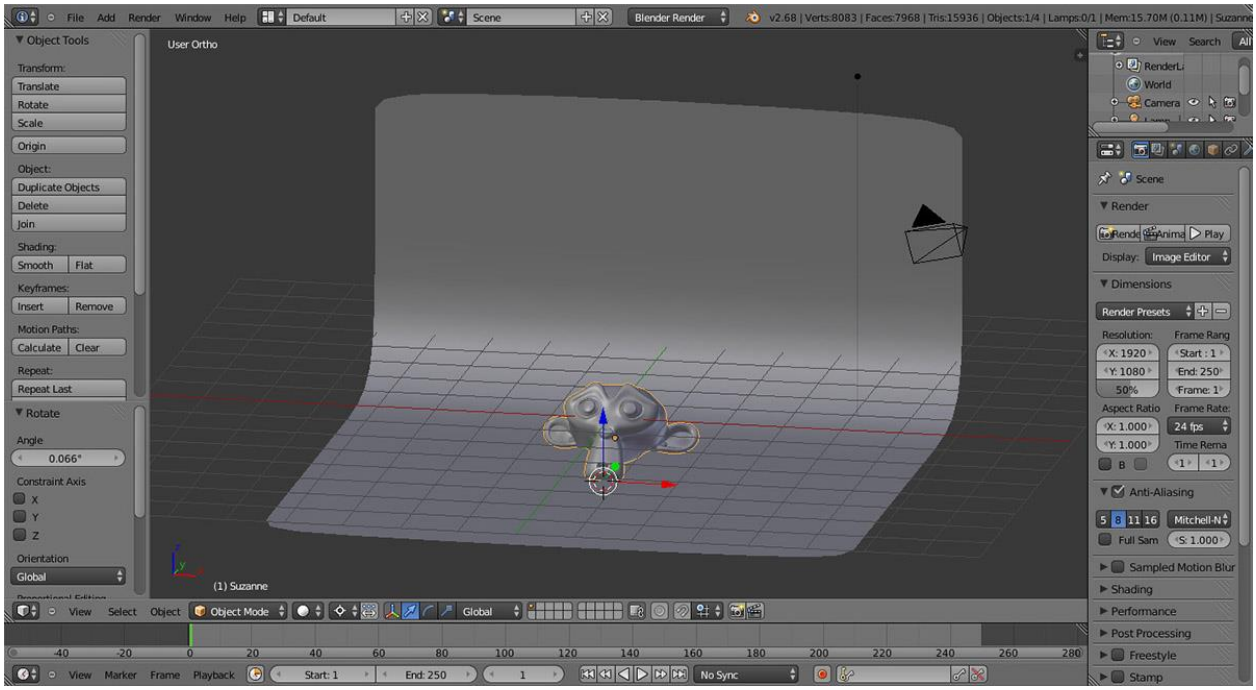


Рис. 3.10 – Настройка сцены

В режиме объектов добавляем новую кривую и выбераем Add Ivy to Mesh. И Вуаля! Теперь давайте посмотрим основные опции:

Maximum Time: Время за которое сгенерируется плющ, очень полезно от зависания компьютера (длинный плющ может использовать свыше 3 миллионов вершин).

Size Setting: Вы этой вкладке мы имеем настройки максимальной длины в единицах Blender (1 единица Blender = 1 метру) – длина сегментов

плюща, максимальная длина ветки без прилипания к объекту и степень адгезии веток.

Weight Setting: Здесь мы можем уточнить как плющ будет расти (насколько рост будет зависеть от силы тяжести, адгезии к объекту, первичного направления вдоль оси Z или вращения в случайном направлении). Не забудьте, что плющ будет расти в направлении выделенного объекта.

Branch Setting: Здесь у нас два поля: первое является вероятностью создания новой ветви, а второе размер.

Последняя вкладка для настроек листьев: размер и вероятность формирования листьев.

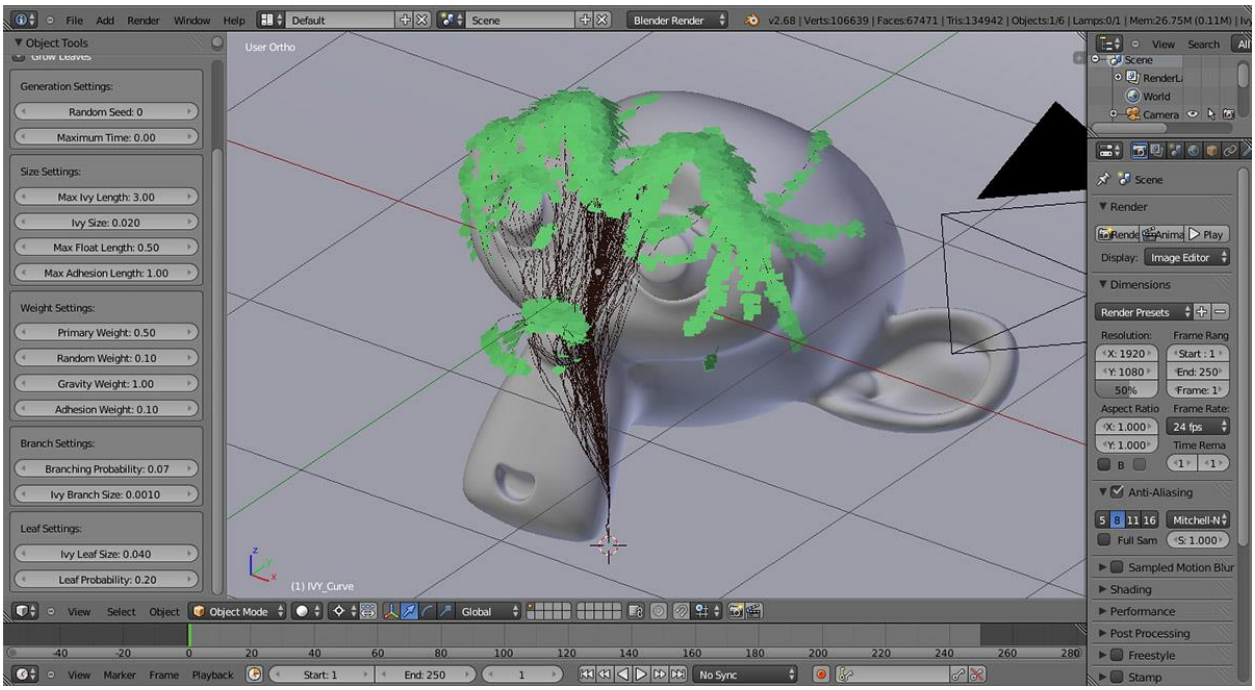


Рис. 3.10 – Настройка плюща

Выбираем Cycles render engine и разделяем вьюпорт на 3 части, одна – для настройки нодов, другая – для UV раскроя и последняя – для 3d вида. Выделяем листья и добавляем новый материал. Соединиям цветовой вход дифузного материала с текстурной нодой (используйте текстуру efeu0.jpg).

Используем текстурные UV координаты (по умолчанию Blender наложит поверх UV все плоскости используемые как листья).

Смешиваем материал diffuse с glossy shader (95% diffuse и 5% glossy) и добавляем следующую текстурную ноду, выделяем изображение карты нормалей (efeu0_norm.jpg) устанавливаем как non color data и соединим с нодой normal map. Теперь соединим выход карты нормалей с входом карты нормалей дифузного и глянцевого шейдеров. Смешаем дифузный и глянцевый материалы с прозрачным (transparent) шейдером и используем чёрно-белое изображение (efeu0_alpha.jpg) в качестве маски.

Последняя вещ, которую осталось сделать, это добавить карту рельефа, добавляем ещё одну текстурную ноду (efeu0_bump.jpg) и соединяем его с входом дисплейсмента (обычно я использую math node, чтобы усилить эффект рельефа).

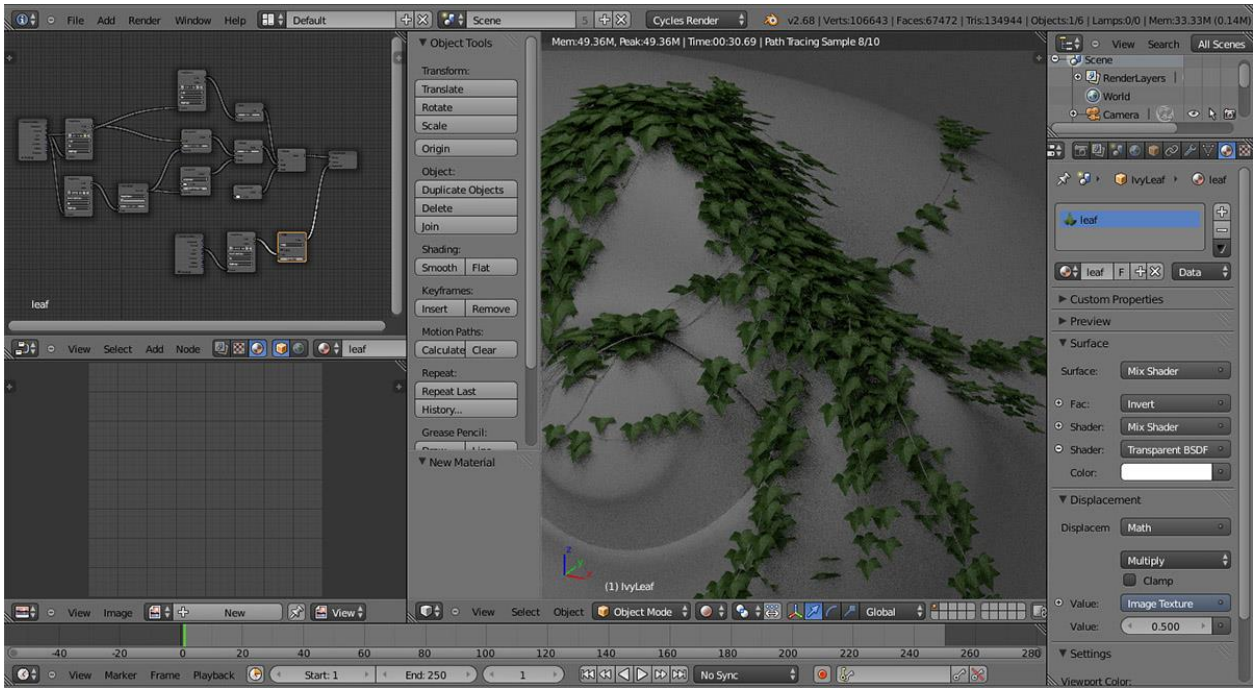


Рис. 3.11 – Настройка нодов плюща

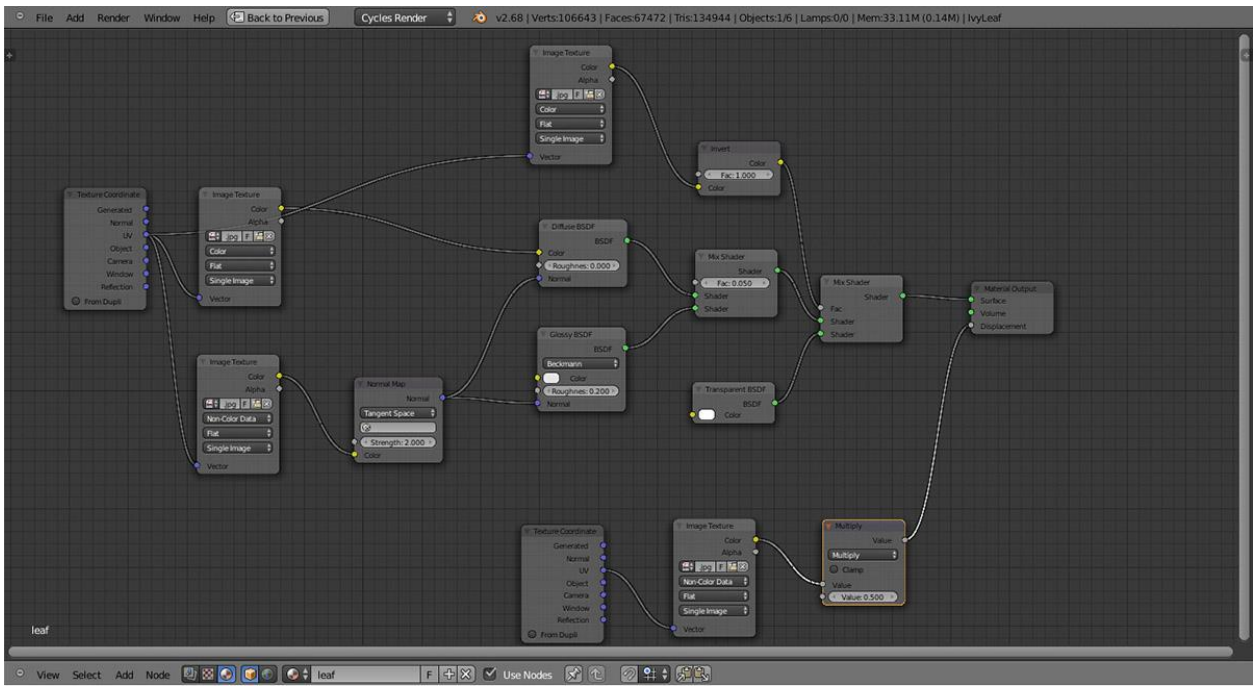


Рис. 3.12 – Настройка нодов дерева

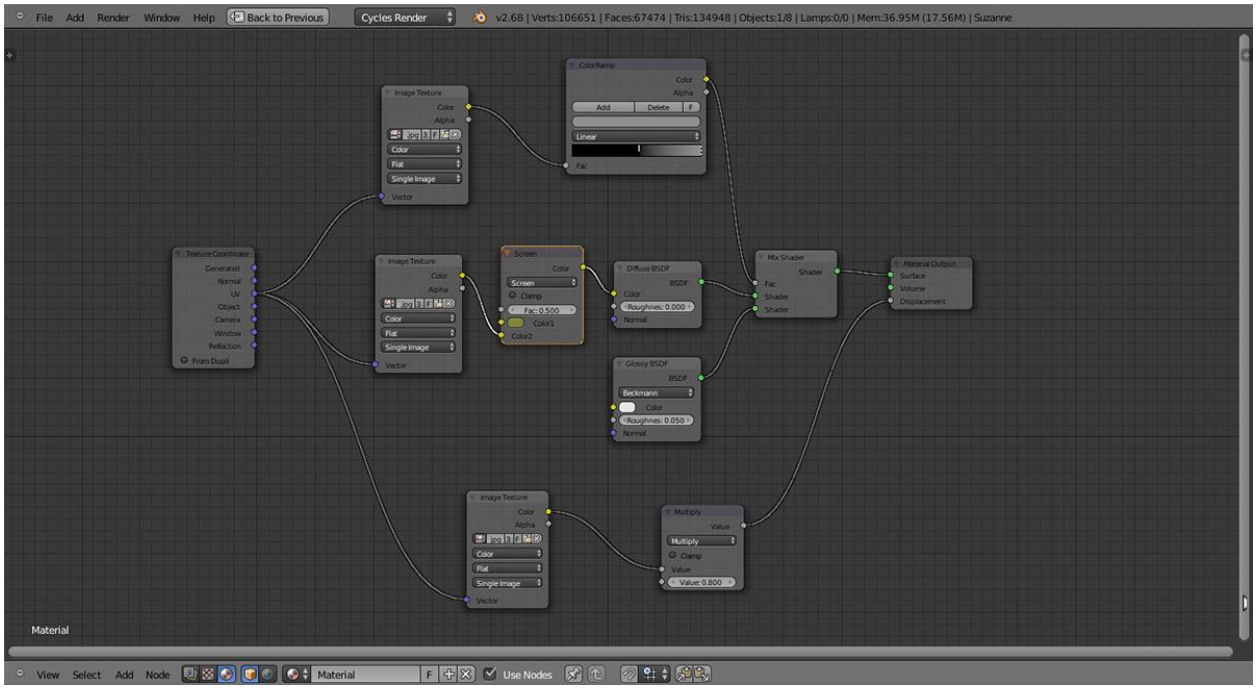


Рис. 3.13 – Настройка нодов листьев

После запуска видим в окне просмотра следующий результат:

Рис. 3.14 – Результат работы аддона


Рис. 3.15 – Пример сцены с использованием аддона

# ЗАКЛЮЧЕНИЕ

В результате изучения способов построения трёхмерных объектов из процедурных текстур были получены следующие результаты:

1. Выполнен анализ существующих на рынке программных комплексов для выполнения поставленных целей;

2. Выбрано и изучено программное обеспечение для создания трёхмерной компьютерной графики;

3. Разработана и внедрена встраиваемая подпрограмма;

4. Протестирована на разносложных 3д объектах.

Полученные результаты могут быть использованы для дальнейших разработок, касающихся генерации трёхмерных объектов с помощью процедурных текстур.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ


1. ГОСТ 7.9 – 77. Реферат и аннотация. – Москва: Изд-во стандартов, 1981. – 6 с.

2. ГОСТ 7.53 – 2001. Издания. Международная стандартная нумерация книг [Текст]. – Взамен ГОСТ 7.53 – 86; введ. 2002 – 07 – 01. – Минск: Межгос. Совет по стандартизации, метрологии и сертификации; Москва: Изд-во стандартов, 2002. – 3 с.

3. Артёмов, Н.С. Анимация 3D персонажей. Самоучитель / Артёмов, Н.С. - М.: ИТ Пресс, 2006 – 264 с.

4. Алямовский, А.А. SolidWorks 2007/2008. Компьютерное моделирование в инженерной практике / А.А. Алямовский. - М.: СПб: БХВ-Петербург, 2008. - 192 с.

5. Архипов, Г. И. Теория кратных тригонометрических сумм / Карацуба А. А., Чубариков В. Н. - М.: Наука. Гл. ред. физ.-мат. лит., 1987. – 368 с.

6. Айерлэнд К., Роузен М. Классическое введение в современную теорию чисел. – М.: Мир, 1987. – 416 с.

7. Виноградов И.М. Метод тригонометрических сумм в теории чисел. М: Наука, 1971. 160 с.

8. Лоу, А. М. Имитационное моделирование. Классика CS / Лоу, А. М., Кельтон, В. Дэвид. - М.: СПб: Питер, 2004. - 848 с

9. Лазарев, А.И. Информация и безопасность. Композиционная технология информационного моделирования сложных объектов принятия решений / Лазарев, А. И. - М.: Московский городской центр научно-технической информации, 1997. - 336 с.

10. Прахов, А. Blender. 3D-моделирование и анимация. Руководство для начинающих / А. Прахов. - М.: БХВ-Петербург, 2009. - 272 с.

11. Россум, Г.М. Язык программирования Python / Россум, Г.М. , Дрейк Ф.Л.Дж., Откидач Д.С. 2011. - 463 с.

12. Beazley D. Python essential reference; - , 2015. - 734 с.

13. Budnev V. M., Ginzburg I. F., Meledin G. V. The two-photon particle production mechanism; - , 2010. - 919 с.

14. Jones C.A., Drake F.L. Python & XML, 2014. - 807 с.

15. Iwaniec H., Kovalski E. Analitic number theory. – Providence, Rhode Island: American Math. Soc., 2004. – 615 p.

16. Hardy G.H. Ramanujan’s trigonometrical function // Proc. Cambr. Soc. 1920. P. 263–271.