Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

Студент Сучков Александр Дмитриевич
фамилия, имя, отчество

Группа ИУ7-42Б

Тип практики стационарная

Название предприятия МГТУ им. Н. Э. Баумана, каф. ИУ7

Студент _____ Сучков А.Д.
подпись, дата *фамилия, и.о.*

Руководитель практики _____ Куров А.В.
подпись, дата *фамилия, и.о.*

Оценка _____

2020 г.

Индивидуальное задание:

Разработать компьютерную программу-игру с подвижной по горизонтали камерой и визуализацией тумана и освещения. Выбрать методы для построения изображения, визуализации эффектов тумана и освещения.

Оглавление

Введение

В настоящее время компьютерная графика имеет достаточно широкий охват применимости во многих отраслях нашей жизни: для визуализации данных на производстве, для создания реалистичных специальных эффектов в кино и для создания видеоигр.

Из-за этого перед специалистами, создающими трёхмерные изображения, встаёт множество трудностей связанных с учётом таких явлений, как преломление, отражение и рассеивание света, а для достижения максимально реалистичного изображения также учитываются дифракция, интерференция, вторичные отражения света, цвет и текстура объектов.

В компьютерной графике существует множество разнообразных алгоритмов, помогающих решить эти задачи, однако большинство являются достаточно ресурсоёмкими, так как чем более реалистичное изображение мы хотим получить, тем больше нам необходимо времени и памяти на синтез. Это, пожалуй, самая главная проблема при создании реалистичных изображений (особенно динамических сцен), которую пытаются решить и по сей день.

Создание видеоигр крепко вплелось в индустрию развлечений. На данный момент существует множество больших компаний, состоящих из огромного количества специалистов, которые соревнуются друг с другом в достижении наивысшего реализма и качества графики. Именно в этой отрасли можно наглядно видеть эволюцию в компьютерной графике, как с каждым годом создаётся всё больше игр, использующих сложные технологии и алгоритмы визуализации.

Целью моей практики будет выбор и, при необходимости, модифицирование наиболее подходящего алгоритма создания трёхмерной динамической сцены.

1. Аналитическая часть

В данном части проводится анализ объектов сцены и существующих алгоритмов построения изображений и выбор более подходящих алгоритмов для дальнейшего использования.

1.1 Описание и формализация объектов сцены

Плоскость земли – ограничивающая плоскость, под которой, по предположению, не могут находится объекты.

Стена – имеет основание, параллельное плоскости земли и рёбра, перпендикулярные плоскости земли.

Источник света – материальная точка в пространстве, из которой исходят лучи света во все стороны. Положение на сцене регулируется координатами (x, y, z), также может иметь цвет.

Для описания трёхмерных геометрических объектов существует три модели: каркасная, поверхностная и объёмная. Для реализации поставленной задачи, более подходящей моделью будет поверхностная, так как, по сравнению с каркасной, она может дать более реалистичное и понятное зрителю изображение. В то же время, объёмной модели требуется больше памяти и поэтому она будет скорее излишняя, в условиях моей задачи.

В свою очередь поверхностная модель может задаваться параметрическим представлением или полигональной сеткой.

При параметрическом представлении поверхность можно получить при вычислении параметрической функции, что очень удобно при просчёте поверхностей вращения, однако, ввиду их отсутствия, данное представление будет не выгодно.

В случае полигональной сетки форма объекта задаётся некоторой совокупностью вершин, рёбер и граней, что позволяет выделить несколько способов представления:

1. Вершинное представление – при таком представлении вершины хранят указатели на соседние вершины. В таком случае для

рендеринга нужно будет обойти все данные по списку, что может занимать достаточно много времени при переборе.

2. Список граней – при таком представлении объект хранится, как множество граней и вершин. В таком случае достаточно удобно производить различные манипуляции над данными.
3. Таблица углов – при таком представлении вершины хранятся в предопределённой таблице, такой что обход таблицы неявно задаёт полигоны. Такое представление более компактное и более производительное для нахождения полигонов, однако операции по замене достаточно медлительны.

Наиболее подходящим представлением сцены в условиях поставленной задачи будет представление в виде списка граней, т.к. оно позволяет эффективно манипулировать данными, а также позволяет проводить явный поиск вершин грани и самих граней, которые окружают вершину.

1.2 Анализ алгоритмов удаления невидимых линий и поверхностей

Выбирая подходящий алгоритм удаления невидимых линий, необходимо учитывать поставленную задачу и её особенности, а именно то, что сцена должна быть динамической, а следовательно, алгоритм должен быть достаточно быстрым. Также, стоит заметить, что в реализации оригинальной игры DOOM 1993 года использовались BSP деревья для построения корректного изображения. В настоящее время существует множество различных алгоритмов построения изображений: алгоритм трассировки лучей, алгоритм Варнока, алгоритм Робертса, алгоритм Z-буфера и т.д. Рассмотрим преимущества и недостатки каждого из алгоритмов.

BSP-деревья

Двоичное разбиение пространства – метод рекурсивного разбиения евклидового пространства в выпуклые множества и гиперплоскости. В результате объекты сцены хранятся в виде структуры данных, называющаяся

BSP-деревом. Каждый узел дерева связан с разбивающей плоскостью, при этом все объекты, которые лежат с фронтальной стороны плоскости, относятся к фронтальному поддереву, а все объекты с другой стороны, относятся к оборотному поддереву. Для определения положения объекта анализируются положения каждой его точки.

Данный метод достаточно эффективен в:

- сортировке визуальных объектов в порядке удаления от наблюдателя;
- обнаружении столкновений.

К недостатку можно отнести то, что для достижения высокой производительности в программе, алгоритм необходимо оптимизировать, что достаточно сильно может повлиять на время и сложность разработки.

Алгоритм трассировки лучей

Данный алгоритм обладает следующими преимуществами:

- вычислительная сложность слабо зависит от сложности сцены;
- отсечение невидимых поверхностей, перспектива и корректное изменения поля зрения являются логическим следствием алгоритма;
- возможность рендеринга гладких объектов без аппроксимации их полигональными поверхностями;
- возможность параллельно и независимо трассировать два и более лучей, разделять участки (или зоны экрана) для трассирования на разных узлах кластера и т.д.

Однако, недостатком алгоритма является его производительность. Метод трассирования лучей каждый раз начинает процесс определения цвета пикселя заново, рассматривая каждый луч наблюдения в отдельности.

В поставленной мною задаче не используются явления отражения и преломления света и поэтому некоторые вычисления окажутся излишними,

кроме того, скорость синтеза сцены должна быть высокой, из-за чего алгоритм трассировки лучей не подходит.

Алгоритм Варнока

Преимуществом данного алгоритма является то, что он работает в пространстве изображений. Алгоритм предлагает разбиение области рисунка на более мелкие окна, и для каждого такого окна определяются связанные с ней многоугольники и те, видимость которых «легко» определить, изображаются на сцене.

К недостаткам можно отнести то, что в противном случае разбиение повторяется, и для каждой из вновь полученных подобластей рекурсивно применяется процедура принятия решения. По предположению, с уменьшением размеров области, её будет перекрывать всё меньшее количество многоугольников. Считается, что в пределе будут получены области, которые содержат не более одного многоугольника, и решение будет принято достаточно просто.

Из-за того, что поиск может продолжаться до тех пор, пока либо остаются области, содержащие не один многоугольник, либо пока размер области не станет совпадать с одним пикселом, алгоритм не подходит под условия моей задачи.

Алгоритм Робертса

Данный алгоритм обладает достаточно простыми и одновременно мощными математическими методами, что, несомненно, является преимуществом. Некоторые реализации, например использующие предварительную сортировку по оси z, могут похвастаться линейной зависимостью от числа объектов на сцене.

Однако алгоритм обладает недостатком, который заключается в вычислительной трудоёмкости, растущей теоретически, как квадрат числа объектов, что может негативно сказаться на производительности. В то же время реализация оптимизированных алгоритмов достаточно сложна.

Алгоритм, использующий Z буфер

Главными преимуществами данного алгоритма являются его достаточная простота реализации и функциональность, которая более чем подходит для визуализации динамической сцены игры. Также алгоритм не тратит время на сортировку элементов сцены, что даёт значительный прирост к производительности.

К недостатку алгоритма можно отнести большой объём памяти необходимый для хранения информации о каждом пикселе. Однако данный недостаток является незначительным ввиду того, что большинство современных компьютеров обладает достаточно большим объёмом памяти для корректной работы алгоритма.

Вывод

Алгоритм, использующий Z буфер, является наиболее подходящим для построения динамической игровой сцены. На его основе будет просто визуализировать эффект тумана, при этом будет удобно хранить глубину пиксела, с помощью которой можно будет вычислить «затуманенность».

1.3 Анализ моделей освещения

На сегодняшний день существует две модели освещения, которые используются для построения света на трёхмерных сценах: локальная и глобальная. Исходя из поставленной задачи, необходимо выбрать более производительный вариант, который позволит достаточно быстро просчитывать освещения на сцене.

Локальная модель освещения

Является самой простой и не рассматривает процессы светового взаимодействия объектов сцены между собой и рассчитывает освещённость только самих объектов.

Глобальная модель освещения

Данная модель рассматривает трёхмерную сцену, как единую систему и описывает освещение с учётом взаимного влияния объектов, что позволяет рассматривать такие явления, как многократное отражение и преломление света, а также рассеянное освещение.

Вывод

Так как программа не должна выводить реалистичного изображения, рациональнее будет использовать локальную модель освещения, что значительно упростит вычисления и повысит производительность игры.

1.4 Анализ алгоритмов закрашивания

Алгоритм простой закраски

По закону Ламберта, вся грань закрашивается одним уровнем интенсивности. Метод является достаточно простым в реализации и не требовательным к ресурсам. Однако алгоритм плохо учитывает отражения и при отрисовки тел вращения, возникают проблемы.

Алгоритм закраски по Гуро

В основу данного алгоритма положена билинейная интерполяция интенсивностей, позволяющая устранять дискретность изменения интенсивности. Благодаря этому криволинейные поверхности будут более гладкими.

Алгоритм закраски по Фонгу

В данном алгоритме за основу берётся билинейная интерполяция векторов нормалей, благодаря чему достигается лучшая локальная аппроксимация кривизны поверхности и, следовательно, изображение выглядит более реалистичным.

Однако алгоритм требует больших вычислений, по сравнению с алгоритмом по Гуро, так как происходит интерполяция значений векторов нормалей.

Вывод

В условиях поставленной задачи, рациональнее будет использовать алгоритм по Гуро, так как программа не должна выводить реалистичное изображение и при этом должна быть достаточно быстрой. Кроме того, алгоритм достаточно хорошо сочетается с выбранным ранее Z буфером.

1.5 Анализ алгоритмов построения теней

Существует множество методов построения теней, например: с помощью математических расчётов проекций на объект, с помощью трассировки лучей или с помощью теневых карт.

Алгоритм трассировки лучей выгоден тем, что тени можно получить без дополнительных вычислений. Пиксел затеняется тогда, когда луч, попадая в объект, позже не попадает ни в другие объекты, ни в источники света. Однако, так как при анализе алгоритмов трассировка лучей не была выбрана, тени необходимо просчитывать отдельно.

Алгоритм, использующий карты теней, подразумевает использование ещё одного Z буфера, который строится от источника света и накладывается на Z буфер камеры.

Вывод

Так как ранее был выбран алгоритм Z буфера для удаления невидимых линий и поверхностей, то более выгодным и подходящим для поставленной задачи будет использование карты теней.

2. Конструкторская часть

2.1 Алгоритм Z-буфера

Данный алгоритм является одним из простейших алгоритмов удаления невидимых поверхностей. Впервые он был предложен Кэтмулом. Алгоритм работает в пространстве изображения, а сама идея z-буфера является простым обобщением идеи о буфере кадра, который используется для запоминания атрибутов или интенсивности каждого пиксела в пространстве изображения.

Z-буфер – это отдельный буфер глубины, который используется для запоминания координаты z или глубины каждого видимого пиксела в пространстве изображения. В процессе работы глубина или значение z каждого нового пиксела, который нужно занести в буфер кадра, сравнивается с глубиной того пиксела, который уже занесён в z-буфере.

Если это сравнение показывает, что новый пиксел расположен впереди пиксела, который находится в буфере кадра, то новое значение заносится в буфер и, кроме этого, производится корректировка z-буфера новым значением z. Если при сравнении получается противоположный результат, то никаких действий не производится.

По сути, алгоритм представляет из себя поиск по x и y наибольшего значения функции z(x, y). Формальное описание алгоритма z-буфера:

1. Заполнить буфер кадра фоновым значением интенсивности или цвета
2. Провести инициализацию Z-буфера минимальным значением глубины
3. Преобразовать каждый многоугольник в растровую форму в произвольном порядке

    3.1 Для всех пикселей, которые связаны с многоугольником, вычислить его глубину z(x, y)

3.2 Глубину пиксела сравнить со значением, которое хранится в буфере: если $z(x, y) > z_{буф}(x, y)$, то $z_{буф}(x, y) = z(x, y)$, $цвет(x, y) = цвет_{пикселя}$

4. Отобразить результат

2.2 Модель освещения

Для достижения наиболее высокой производительности, будем считать в локальной модели освещения две составляющие интенсивности, а именно фоновую освещённость (I_{amb}) и диффузное отражение (I_{diff}).

Фоновое освещение это постоянная в каждой точке величина надбавки к освещению. Вычисляется следующим образом:

$$\mathbf{I_{amb} = k_a \cdot I_a},$$

где $\mathbf{I_{amb}}$ – интенсивность отражённого Ambient-освещения, $\mathbf{k_a}$ – коэффициент в пределах от 0 до 1, характеризующий отражающие свойства поверхности для Ambient-освещения, $\mathbf{I_a}$ – исходная интенсивность Ambient-освещения, которое падает на поверхность.

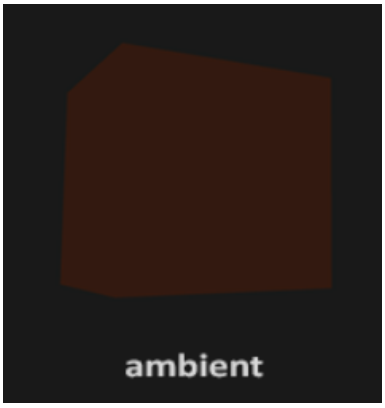

Рисунок 1

С учётом только фоновой освещённости, получим следующее изображение (рис. 1).

$$\mathbf{I_{diff} = I_d \cdot k_{diff} \cdot \cos(\theta)},$$

где $\mathbf{I_d}$ – интенсивность падающего на поверхность света, $\mathbf{k_{diff}}$ – коэффициент в пределах от 0 до 1, характеризующий рассеивающие свойства поверхности, **cos(θ)** – угол между направлением на источник света и нормалью поверхности.

Рисунок 2

С учётом и фоновой освещённости, и диффузного отражения, получим более реалистичное изображение (рис. 2). Тогда, конечная формула принимает вид:

$$\mathbf{I = I_{amb} + I_{diff} = k_a \cdot I_a + I_d \cdot k_{diff} \cdot \cos(\theta)},$$

где для более красивого и приятного изображения $k_a = 1$, $k_{diff} = 0.8$.

2.3 Алгоритм Гуро

Данный метод закраски, основанный на интерполяции интенсивности, позволяет устранять дискретность изменения интенсивности.

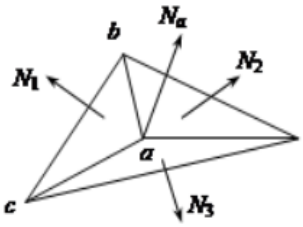

Рисунок 3

Нормали к вершинам: $\overline{N}\,a = (\overline{N}_1 + \overline{N}_2 + \overline{N}_3) / 3$

Алгоритм делится на четыре этапа:

1. Вычисляются нормали к каждой грани.
2. Определяются нормали в вершинах путём усреднения нормалей по всем полигональным граням, которым принадлежит вершина.
3. Используя нормали в вершинах и применяя произвольный метод закраски, вычисляются значения интенсивности в вершинах.
4. Каждый многоугольник закрашивается путём линейной интерполяции значений интенсивностей в вершинах сначала вдоль каждого ребра, а затем и между рёбрами вдоль каждой сканирующей строки (рис. 3)

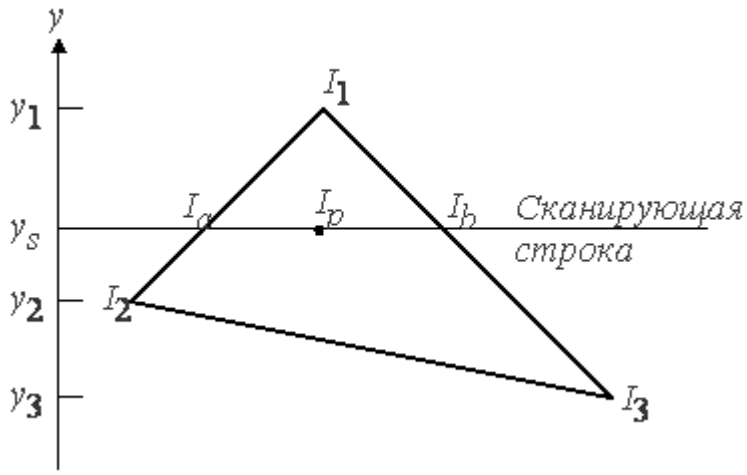

Рисунок 4

Интерполяцию вдоль рёбер легко объединить с алгоритмом удаления скрытых поверхностей, которое построено на принципе построчного сканирования. Для всех рёбер запоминается начальная интенсивность, а также изменение интенсивности при каждом единичном шаге по координате y. Заполнение видимого интервала на сканирующей строке производится путём интерполяции между значениями интенсивности на двух рёбрах, которые ограничивают интервал (рис. 4).

$$I_a = I_1 \frac{y_s - y_2}{y_1 - y_2} + I_2 \frac{y_1 - y_s}{y_1 - y_2} \ ;$$

$$I_b = I_1 \frac{y_s - y_3}{y_1 - y_3} + I_3 \frac{y_1 - y_s}{y_1 - y_3};$$

$$I_p = I_a \frac{x_b - x_p}{x_b - x_a} + I_b \frac{x_p - x_a}{x_b - x_a}.$$

Для цветных объектов отдельно интерполируется каждая из компонент цвета.

2.4 Теневые карты

Потенциально, метод теневых буферов является самым универсальным методом построения теней в реальном времени, однако до недавнего времени его реализация была затруднена ввиду отсутствия необходимого оборудования достаточной мощности. Впервые его представил Ланс Вильямс в 1978 году.

Базируется данный алгоритм на факте, что область, которая находится в тени относительно источника света, не будет видна из точки его положения. В начале сцена рисуется из источника света во вне экранный буфер. Далее нас будут интересовать только z-значения точек буфера. Сохранённая в z-буфере (теневом буфере) картина глубин, будет использоваться для проверки на принадлежность точек тени. В буфере записаны глубины всех ближайших к источнику света точек. Любая точка, которая находится дальше соответствующей точки в теневом буфере, будет в тени.

Во время основного рендеринга сцены, каждый пиксел, видимый из камеры, сравнивается с пикселами теневого буфера, спроецированными в плоскость камеры. Если расстояние от точки до источника света равно соответствующему z-значению буфера, то пиксел освещён, однако если точка дальше, то пиксел в тени.

2.5 Геометрические преобразования

Для осуществления поворота и перемещения камеры используются аффинные преобразования.

Уравнения и матрицы преобразований:

- перемещение точки вдоль координатных осей на dx, dy, dz:

$$\begin{cases} X = x + dx \\ Y = y + dy \\ Z = z + dz \end{cases} \quad \begin{pmatrix} 1 & 0 & 0 & dx \\ 0 & 1 & 0 & dy \\ 0 & 0 & 1 & dz \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- поворот относительно осей x, y, z на угол φ :

ось x: $\begin{cases} X = x \\ Y = y\cos\varphi + z\sin\varphi \\ Z = -y\sin\varphi + z\cos\varphi \end{cases}$ $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi & 0 \\ 0 & -\sin\varphi & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

ось y: $\begin{cases} X = x\cos\varphi + z\sin\varphi \\ Y = y \\ Z = -x\sin\varphi + z\cos\varphi \end{cases}$ $\begin{pmatrix} \cos\varphi & 0 & \sin\varphi & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\varphi & 0 & \cos\varphi & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

ось z: $\begin{cases} X = x\cos\varphi + y\sin\varphi \\ Y = -x\sin\varphi + y\cos\varphi \\ Z = z \end{cases}$ $\begin{pmatrix} \cos\varphi & \sin\varphi & 0 & 0 \\ -\sin\varphi & \cos\varphi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

2.6 Оптимизация алгоритмов

Работу некоторых алгоритмов можно объединить, чтобы добиться более производительной работы. Алгоритм z-буфера можно объединить с алгоритмом Гуро, так как они оба используют интерполяцию, также в функцию z-буфера можно занести сравнение с буфером тени.

Чтобы добиться сокращения количества вычислений при проецировании точек из буфера тени на z-буфер, можно найти минимальный и максимальный пиксел, глубина которого отлична от максимальной глубины. Это позволит сократить итерации цикла.

3. Технологическая часть

3.1 Выбор языка программирования и среды разработки

Для реализации проекта в качестве языка программирования был выбран C++ — компилируемый статически типизированный язык программирования общего назначения. Поддерживает процедурное, объектно-ориентированное и обобщённое парадигмы программирования, также обеспечивает модульность, раздельную компиляцию, обработку исключений, абстракцию данных, объявление типов (классов) объектов, виртуальные функции. Стандартная библиотека включает в себя некоторые уже готовые контейнеры и алгоритмы.

C++ в себе сочетает свойства как высокоуровневых, так и низкоуровневых языков программирования. Также имеется возможность выполнять ассемблерные вставки – возможность встраивать низкоуровневый код, написанный на ассемблере, что может пригодится при оптимизации программы и увеличении её быстродействия. Язык широко используется для разработки программного обеспечения, область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также и для создания видеоигр.

Существует множество коммерческих и бесплатных реализаций языка C++ для различных платформ. К примеру GCC, Visual C++, Intel C++, Compiler.

Средой разработки была выбрана «Visual Studio 2019» из-за того, что она бесплатна в пользовании студентами, имеет множество удобств, связанных с написанием и отладкой кода, позволяет достаточно быстро подключать и настраивать сторонние библиотеки.

В качестве платформы выбрана OS Windows.

Заключение

Проделав данную работу, были рассмотрены и проанализированы преимущества и недостатки каждого из алгоритмов удаления невидимых линий и поверхностей, закрашивания, построения теней. Также, для выполнения поставленной задачи, были выбраны наиболее подходящие алгоритмы.

Список использованной литературы

1. Вишнякова Д. Ю., Надолинский Н. А. Программная реализация трехмерных сцен // Известия ЮФУ. Технические науки. 2001. №4.
2. Роджерс Д. Алгоритмические основы машинной графики. – М., «Мир», 1989
3. В. П. Иванов, А. С. Батраков. Трёхмерная компьютерная графика / Под ред. Г. М. Полищука. — М.: Радио и связь, 1995. — 224 с.
4. Дж. Ли, Б. Уэр. Трёхмерная графика и анимация. — 2-е изд. — М.: Вильямс, 2002. — 640 с.
5. Mark de Berg. Computational Geometry: Algorithms and Applications. — Springer Science & Business Media, 2008. — P. 259.