ЛАБОРАТОРНАЯ РАБОТА №1

1) Разработать парсер файлов формата .obj

Далее будут рассмотрены основные компоненты, которые нужно прочитать из файла, чтобы можно было визуализировать объекты.

`v x y z [w]`

Описывает координаты геометрической вершины в трехмерном пространстве. Данные координаты задают положение вершины в пространстве модели. Для кривых и поверхностей также указывается необязательная координата **`w`**. По умолчанию она равна 1.

`vt u [v] [w]`

Описывает текстурные координаты вершины. Для одномерной текстуры требуется только координата **`u`**, для двухмерной — **`u`** и **`v`**, а для трехмерной — все три координаты. Координаты **`v`** и **`w`** необязательны и по умолчанию равны 0. Обычно текстурные координаты находятся в диапазоне от 0 до 1. Значения координат, которые выходят за этот диапазон, используются для создания повторяющейся текстуры.

`vn i j k`

Описывает трехмерный вектор нормали вершины. Он необходим для построения плавного освещения. При наличии нормали группы сглаживания игнорируются. Вектор может быть ненормированным.

`f v1 v2 v3 v4 ...`

Описывает полигон, который должен содержать не менее трех вершин, перечисленных через пробел. В качестве параметров данная конструкция использует индексы координат вершин, прочитанных ранее. Нумерация индексов начинается с 1. Также индексы могут быть отрицательными. Если индекс имеет значение -1, он ссылается на последний прочитанный элемент.

Вместе с индексами координат вершин могут быть записаны индексы текстурных координат:

`f v1/vt1 v2/vt2 v3/vt3 v4/vt4 ...`

Также допустимо хранение индексов вершинных нормалей:

`f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3 v4/vt4/vn4 ...`

Если индексы текстурных координат указывать не нужно, их можно пропустить:

`f v1//vn1 v2//vn2 v3//vn3 v4//vn4 ...`

2) Реализовать матричные преобразования координат из пространства модели в мировое пространство

Так как прочитанные из файла координаты вершин находятся в пространстве модели, перед отрисовкой их необходимо преобразовать в мировые координаты. Для этого используются матричные преобразования.

Если необходимо представить преобразование из одного трехмерного пространства в другое, то понадобится матрица размером 4 × 4. В данном случае будет предполагаться нотация вектора-столбца, как в OpenGL. Чтобы применить преобразование, нужно умножить все векторы, которые необходимо преобразовать, на матрицы преобразования.

Ниже представлено общее преобразование в матричной форме:

$$\begin{bmatrix} XAxis.x & YAxis.x & ZAxis.x & Translation.x \\ XAxis.y & YAxis.y & ZAxis.y & Translation.y \\ XAxis.z & YAxis.z & ZAxis.z & Translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.1)$$

где $XAxis$ является ориентацией оси X в новом пространстве, $YAxis$ является ориентацией оси Y в новом пространстве, $ZAxis$ является ориентацией оси Z в новом пространстве, а $Translation$ описывает положение, в котором новое пространство будет находиться относительно активного пространства.

Иногда необходимо сделать простые преобразования, такие как перемещение или вращение. В этих случаях можно использовать следующие матрицы, которые являются частными случаями только что представленной общей формы.

Матрица перемещения:

$$\begin{bmatrix} 1 & 0 & 0 & Translation.x \\ 0 & 1 & 0 & Translation.y \\ 0 & 0 & 1 & Translation.z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.2)$$

где $Translation$ — это трехмерный вектор, который представляет позицию, в которую необходимо переместить пространство. Матрица перемещения оставляет все оси повернутыми в том же направлении, что и направление осей активного пространства.

Матрица масштаба:

$$\begin{bmatrix} Scale.x & 0 & 0 & 0 \\ 0 & Scale.y & 0 & 0 \\ 0 & 0 & Scale.z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.3)$$

где $Scale$ — трехмерный вектор, который представляет масштаб вдоль каждой оси. Если прочитать первый столбец, можно увидеть, как новая ось X все еще находится в том же направлении, но масштабируется скалярным $Scale.x$. То же самое происходит со всеми другими осями. Также стоит обратить внимание на то, что столбец перемещения — это все нули, что означает, что перемещение не требуется.

Матрица поворота вокруг оси X:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.4)$$

где θ — угол, на который нужно повернуть вектор. Следует отметить, что первый столбец никогда не изменится, так как вращение происходит вокруг оси X. Также необходимо обратить внимание на то, как изменение θ на 90° преобразует ось Y в ось Z и ось Z в ось Y.

Матрица поворота вокруг оси Y:

$$\begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.5)$$

Матрица поворота вокруг оси Z:

$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.6)$$

Матрицы вращения для оси Z и оси Y ведут себя так же, как и матрица вращения для оси X.

Представленные матрицы часто используются, и все они необходимы для описания преобразований. Можно связать несколько преобразований вместе, умножая матрицы одну за другой. Результатом будет одиночная матрица, которая кодирует полное преобразование. Порядок, который применятся для

преобразований, очень важен. Это отражается в математике тем, что матричное умножение не является коммутативным. Поэтому в общем случае $[Translate] \times [Rotate]$ отличается от $[Rotate] \times [Translate]$.

Поскольку для преобразований используются векторы-столбцы, читать цепочку преобразований следует справа налево, поэтому, если необходимо повернуть объект на 90° влево вокруг оси Y, а затем переместить его на 10 единиц вдоль оси Z, цепочка будет следующей: $[TranslateX\ 10] \times [RotateY\ 90°]$.

3) Реализовать матричное преобразование координат из мирового пространства в пространство наблюдателя

Первый шаг отображения 3D сцены — это поместить все модели в одно и то же пространство — мировое пространство. Поскольку каждый объект в мире будет иметь свое собственное положение и ориентацию, у каждого из них есть своя матрица трансформации.

Затем каждый объект нужно спроецировать на экран. Обычно это делается в два этапа. Первый шаг перемещает весь объект в другое пространство, называемое пространством наблюдателя. Второй шаг выполняет фактическую проекцию с использованием матрицы проекции.

Пространство наблюдателя — это вспомогательное пространство, которое используется для упрощения математики. Идея состоит в том, чтобы расположить все объекты относительно камеры, предполагая проецирование всех их вершин на экран камеры, который может быть произвольно ориентирован в пространстве. Математика сильно бы упростилась, если бы была возможность сосредоточить камеру в начале координат и направить ее в сторону одной из трех осей, например, оси Z. Таким образом, происходит переход из мирового пространства в пространство наблюдателя (иногда называемое пространством камеры).

Теперь, если необходимо поместить камеру в мировое пространство, нужно использовать матрицу преобразования, которая находится там, где находится камера, и ориентирована так, что ось Z смотрит на цель камеры. Обратное преобразование, применяемое ко всем объектам в мировом пространстве, переместит весь мир в пространство наблюдателя. Следует обратить внимание на то, что можно объединить две трансформации из пространства модели в мировое пространство и из мирового пространства в пространство наблюдателя в единое преобразование из пространства модели в пространство наблюдателя.

Для построения матрицы должны быть заданы следующие величины:

– позиция камеры в мировом пространстве (eye);

– позиция цели, на которую направлена камера ($target$);

– вектор, направленный вертикально вверх с точки зрения камеры (up).

Далее необходимо вычислить базисные векторы камеры:

$$ZAxis = normalize(eye - target),$$
$$XAxis = normalize(up \times ZAxis),$$
$$YAxis = up. \quad (1.7)$$

Итоговая матрица будет выглядеть следующим образом:

viewport

$$\begin{bmatrix} XAxis_x & XAxis_y & XAxis_z & -(XAxis \cdot eye) \\ YAxis_x & YAxis_y & YAxis_z & -(YAxis \cdot eye) \\ ZAxis_x & ZAxis_y & ZAxis_z & -(ZAxis \cdot eye) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (1.8)$$

С помощью данной матрицы все объекты сцены получат новые координаты в пространстве, центром которого является позиция камеры.

4) Реализовать матричное преобразование координат из пространства наблюдателя в пространство проекции

Сцена теперь находится в пространстве, которое удобно проецировать. Все, что теперь нужно сделать, это спроецировать его на воображаемый экран камеры. Прежде чем сделать изображение плоским, необходимо перейти в другое, последнее пространство, пространство проекции. Это пространство является кубоидом, размеры которого находятся между значениями -1 и 1 для осей X и Y и между значениями 0 и 1 для оси Z. Это пространство очень удобно для отсечения (все, что находится за пределами диапазона от -1 до 1 и от 0 до 1, находится за пределами области просмотра камеры) и упрощает операцию проецирования (нужно просто отбросить значение z, чтобы получить плоское изображение).

Чтобы перейти из пространства наблюдателя в пространство проецирования, нужна другая матрица — матрица преобразования из пространства наблюдателя в пространство проекции. Значения этой матрицы зависят от того, какой тип проекции необходимо выполнить. Двумя наиболее часто используемыми проекциями являются ортографическая проекция и перспективная проекция.

Чтобы сделать ортографическую проекцию, нужно определить размер области, которую может видеть камера. Обычно он задается значениями ширины и высоты для оси X и Y и значениями ближнего и дальнего z для оси Z.

Учитывая эти значения, можно создать матрицу преобразования, которая преобразует область прямоугольника в кубоид. Следующая матрица преобразует векторы из пространства наблюдателя в пространство ортографической проекции и предполагает правостороннюю систему координат:

$$\begin{bmatrix} \frac{2}{width} & 0 & 0 & 0 \\ 0 & \frac{2}{height} & 0 & 0 \\ 0 & 0 & \frac{1}{z_{near} - z_{far}} & \frac{z_{near}}{z_{near} - z_{far}} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{1.9}$$

где $width$ — ширина обзора камеры;

$height$ — высота обзора камеры;

z_{near} — расстояние до ближней плоскости обзора камеры;

z_{far} — расстояние до дальней плоскости обзора камеры.

Другой проекцией является перспективная проекция. Идея похожа на орфографическую проекцию, но на этот раз область просмотра является усеченной пирамидой, и поэтому преобразования немного сложнее. К сожалению, матричного умножения в этом случае недостаточно, потому что после умножения на матрицу результат не находится в одном и том же пространстве проекции (что означает, что w-компонент не равен 1 для каждой вершины). Чтобы завершить преобразование, нужно разделить каждую компоненту вектора на компонент w. Современные графические API выполняют деление автоматически, поэтому можно просто умножить все вершины на матрицу перспективной проекции и отправить результат на графический процессор.

$$\begin{bmatrix} \frac{2 \cdot z_{near}}{width} & 0 & 0 & 0 \\ 0 & \frac{2 \cdot z_{near}}{height} & 0 & 0 \\ 0 & 0 & \frac{z_{far}}{z_{near} - z_{far}} & \frac{z_{near} \cdot z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}. \tag{1.10}$$

Также существует другая форма матрицы перспективной проекции, использующая угол обзора камеры:

projection

$$\begin{bmatrix} \frac{1}{aspect \cdot \tan\left(\frac{FOV}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{FOV}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{z_{far}}{z_{near} - z_{far}} & \frac{z_{near} \cdot z_{far}}{z_{near} - z_{far}} \\ 0 & 0 & -1 & 0 \end{bmatrix}, \quad (1.11)$$

где FOV — поле зрение камеры по оси Y;
$aspect$ — соотношение сторон обзора камеры.

5) Реализовать матричное преобразование координат из пространства проекции в пространство окна просмотра

Окно просмотра — это область в окне приложения, в которой отображается визуализируемая сцена. Оно может занимать всю область окна приложения или его часть.

Т.к. экранная ось Y направлена вниз (начало координат находится в верхнем левом углу экрана), ее необходимо перевернуть. Чтобы адаптировать размеры проекции под размеры окна просмотра, учесть направление оси Y экрана и переместить начало координат в центр окна просмотра, необходимо воспользоваться следующей матрицей:

view

$$\begin{bmatrix} \frac{width}{2} & 0 & 0 & x_{min} + \frac{width}{2} \\ 0 & -\frac{height}{2} & 0 & y_{min} + \frac{height}{2} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad (1.12)$$

Стоит учитывать, что соотношение сторон проекции и соотношение сторон окна просмотра должны совпадать, иначе итоговое изображение будет растянуто или сжато внутри окна просмотра.

Наконец, модель может быть преобразована для отображения с помощью цепочки $[Viewport] \times [Projection] \times [View] \times [Model]$.

6) Реализовать отрисовку проволочной 3D модели

После того как были преобразованы все координаты вершин, необходимо нарисовать линии, соединяющие соответствующие вершины полигонов.

Для этого можно использовать следующие алгоритмы:

– алгоритм DDA-линии;

– алгоритм Брезенхема.

Алгоритм DDA-линии

Пусть отрезок задан координатами концов $(x_1, y_1); (x_2, y_2)$. Большее по абсолютной величине число, $(x_2 - x_1)$ или $(y_2 - y_1)$, принимается за количество шагов L цикла растеризации. На каждом шаге цикла значение функции и ее аргумента получают приращения $(x_2 - x_1)/L\,; (y_2 - y_1)/L$. Растровые координаты отрезка получаются в результате округления соответствующих x и y, полученных на каждом шаге цикла растеризации.

Алгоритм Брезенхема

В процессе растеризации линии можно заметить, что на каждом шаге цикла происходит одно из двух: значение y остается неизменным либо увеличивается на 1. Выбор производится на основе *значения ошибки*, которое означает вертикальное расстояние между текущим значением y и точным значением функции для текущего x. Всякий раз, когда мы увеличиваем x, мы увеличиваем значение ошибки на величину наклона $s = \frac{y_2 - y_1}{x_2 - x_1}$. Если ошибка превысила 0.5, линия стала ближе к следующему y, поэтому нужно увеличить y на единицу, а значение ошибки уменьшить на 1.

В данном алгоритме значение ошибки, величина наклона s, а также константа 0.5 являются числами с плавающей запятой. Чтобы ускорить вычисления, необходимо перейти к целым числам. Это можно сделать, если умножить все используемые вещественные величины на $(x_2 - x_1)$. Тогда на каждом шаге цикла значение ошибки будет увеличиваться на $(y_2 - y_1)$, а при увеличении y на 1 значение ошибки будет уменьшаться на $(x_2 - x_1)$. Чтобы избавиться от константы 0.5, необходимо умножить обе части неравенства на 2, тогда вместо $error > 0.5 \times (x_2 - x_1)$ получится $2 \times error > (x_2 - x_1)$.

ЛАБОРАТОРНАЯ РАБОТА №2

1) Реализовать алгоритм растеризации треугольников

Для отображения заполненного треугольника на экране необходимо найти координаты всех точек, которые попадут в область треугольника, ограниченную тремя его ребрами. Процесс нахождения координат этих точек называется растеризацией.

Одним из алгоритмов растеризации треугольников является метод сканирующей линии, где горизонтальная линия перемещается вниз от самой верхней вершины треугольника до самой нижней. На каждом этапе алгоритма находятся координаты точек пересечения ребер треугольника со сканирующей линией, а затем производится рисование горизонтальной линии между найденными координатами. Для нахождения точек пересечения и рисования горизонтальной линии можно воспользоваться алгоритмами, реализованными в предыдущей лабораторной работе.

2) Реализовать отбраковку невидимых и задних поверхностей трехмерных объектов

В процессе растеризации некоторые задние грани объекта могут быть нарисованы поверх передних граней. Для решения этой проблемы используется Z-буфер, который хранит глубину нарисованных фрагментов. При рисовании нового фрагмента нужно сравнить значение глубины этого фрагмента со значением глубины нарисованного в той же позиции фрагмента в Z-буфере. Если значение глубины нового фрагмента меньше значения глубины в Z-буфере (фрагмент находится ближе к наблюдателю), то значение глубины в Z-буфере заменяется значением глубины нового фрагмента. В противном случае новый фрагмент не рисуется.

Для ускорения отрисовки объектов можно использовать отбраковку поверхностей, которые не видны из-за их ориентации. Это все треугольники с нормалью, которая смотрит почти в том же направлении, что и зритель. Если скалярное произведение вектора нормали и вектора взгляда отрицательное, то такой треугольник можно отбросить. Чтобы найти нормаль к поверхности треугольника, необходимо вычислить векторное произведение двух его смежных ребер. Если на экране вершины треугольника перечислены по часовой стрелке, то вектор нормали будет направлен по направлению взгляда, следовательно такой треугольник можно отбраковать.

3) Реализовать плоское затенение и модель освещения Ламберта

Простейшей моделью затенения является «постоянное затенение», также известное как «граненое затенение» или «плоское затенение». Этот подход применяет модель освещения один раз, чтобы определить одно значение интенсивности, которое затем используется для затенения целого полигона.

Идея алгоритма плоского затенения довольна простая. Сначала вычисляется цвет в каждой вершине треугольника, затем полученные значения усредняются, и весь треугольник закрашивается в полученный цвет. При использовании этой модели не возникает трудностей, когда нормали заданы для граней, а не для вершин. В этом случае цвет треугольника можно рассчитывать в его геометрическом центре.

Чтобы вычислить интенсивность света для всего полигона, можно воспользоваться моделью освещения Ламберта, которая является простейшей моделью освещения (рисунок 2.1). Для этого нужно найти скалярное произведение вектора нормали и обратного вектора направления света. Полученный результат будет являться косинусом угла между этими векторами. Чем меньше угол между векторами, тем больше косинус угла, тем сильнее свет освещает поверхность.

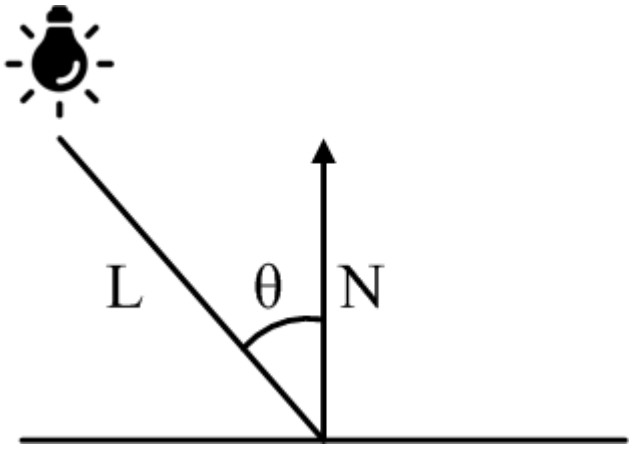

Рисунок 2.1 — Схема модели освещения Ламберта

ЛАБОРАТОРНАЯ РАБОТА №3

1) Реализовать модель затенения по Фонгу

В отличие от плоского затенения, в этой модели освещение не усредняется для грани, а линейно интерполируется между вершинами, поэтому для данной модели важно то, что нормали задаются в вершинах. Этот алгоритм позволяет получить гораздо более приятное изображение, чем при использовании алгоритма плоского затенения, но требует гораздо больше вычислительных ресурсов.

Модели обычно задаются набором плоских выпуклых граней, хотя большинство реальных трехмерных предметов имеют гладкие криволинейные поверхности. Таким образом, криволинейная поверхность рисуется как ребристая полигональная сетка. Для того, чтобы эта сетка выглядела гладкой, используется тот или иной метод интерполяции освещенности вершин полигональной сетки.

Если используется затенение по Гуро, то расчет цвета производится в каждой вершине каждой грани, а затем рассчитанный цвет интерполируется по всей грани. В результате блики, которые должны проявиться в середине полигона, нарисованы не будут — при интерполяции цветов вершин более яркое освещение центра многоугольника невозможно.

При затенении по Фонгу между вершинами интерполируется не цвет, а нормаль. Цвет, в свою очередь, рассчитывается для каждого пикселя в отдельности.

Если нормали вершин не заданы в сетке и не могут быть определены непосредственно с реальной поверхности, то их можно аппроксимировать путем усреднения нормалей поверхности всех полигонов, разделяющих одну вершину. Если ребро должно быть видимым (как на стыке между крылом и корпусом самолета), то следует найти две вершинные нормали, по одной для каждой стороны ребра, путем усреднения нормалей полигонов отдельно на каждой стороне ребра.

2) Реализовать модель освещения по Фонгу

Освещение в реальном мире чрезвычайно сложно и зависит от слишком многих факторов, что невозможно вычислить на ограниченной производительности, которая имеется на сегодняшний день. Модель освещения по Фонгу основана на физике света в реальной жизни и используется для создания реалистичного изображения. Освещение по Фонгу состоит из трех

компонентов: фонового, рассеянного и бликового освещения. На рисунке 3.1 показано, как выглядят эти компоненты.

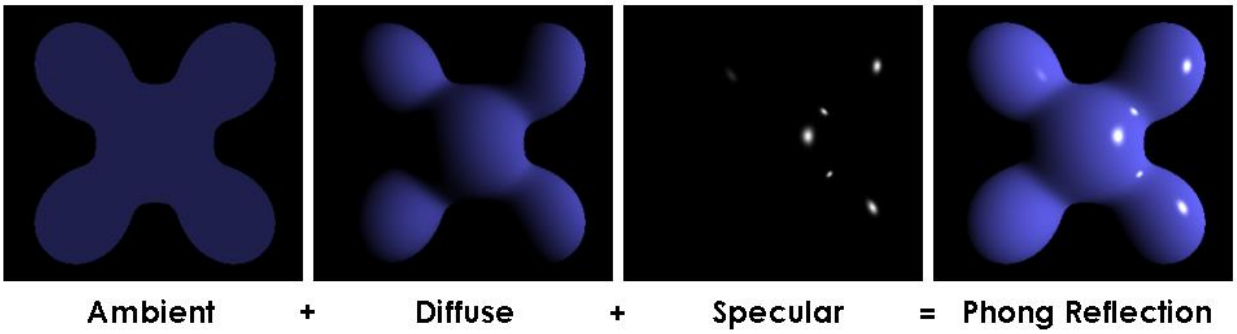

Рисунок 3.1 — Компоненты освещения по Фонгу

Фоновое освещение: даже когда темно, обычно где-то в мире присутствует какой-то свет (луна, дальний свет), поэтому объекты почти никогда не становятся абсолютно черными. Чтобы имитировать это освещение, используется постоянная освещенность, которая всегда придает объекту некоторый цвет (слева на рисунке 3.1).

Рассеянное освещение: имитирует направленное воздействие, которое источник света оказывает на объект. Это самый визуально значимый компонент модели освещения. Чем больше объект обращен к источнику света, тем ярче он становится (второе слева на рисунке 3.1).

Зеркальное освещение: имитирует яркое пятно света, которое появляется на блестящих объектах. Зеркальные блики часто имеют цвет света, а не цвет объекта (второе справа на рисунке 3.1).

Чтобы создать визуально интересные сцены, необходимо имитировать, по крайней мере, эти три компонента освещения.

Фоновое освещение

Свет обычно исходит не от одного источника света, а от многих источников света, расположенных вокруг, даже когда они не видны. Одним из свойств света является то, что он может рассеиваться и отражаться во многих направлениях, достигая объектов, которые не находятся в непосредственной близости. Таким образом, свет может отражаться от других поверхностей и оказывать косвенное влияние на освещение объекта. Алгоритмы, которые принимают это во внимание, называются глобальными алгоритмами освещения, но они трудоемкие и / или сложные.

Поскольку расчет фонового освещения требует больших вычислительных ресурсов, что неприемлемо, необходимо использовать очень упрощенную модель глобального освещения, а именно фонового освещения. Для этого

используется небольшой постоянный (светлый) цвет, который добавляется к окончательному цвету фрагментов объекта, что имитирует постоянное присутствие отраженного света, даже если нет прямого источника света.

Добавление фонового освещения на сцену очень просто. Цвет фонового света умножается на небольшой постоянный коэффициент фонового освещения:

$$I_a = k_a \cdot i_a, \tag{3.1}$$

где I_a — значение фонового освещения;

k_a — коэффициент фонового освещения;

i_a — цвет фонового света.

Рассеянное освещение

Фоновое освещение само по себе не дает интересных результатов, но рассеянное освещение оказывает значительное визуальное воздействие на объект. Рассеянное освещение дает объекту большую яркость, чем ближе его фрагменты повернуты к лучу света, испускаемого источником света. На рисунке 2.1 представлено схематическое описание рассеянного света.

Слева находится источник света с лучом света, нацеленным на один из фрагментов объекта. Затем нужно измерить, под каким углом световой луч падает на фрагмент. Если луч света перпендикулярен к поверхности объекта, свет оказывает наибольшее влияние. Чтобы измерить угол между лучом света и фрагментом, используется вектор нормали, который перпендикулярен поверхности фрагмента. Тогда угол между двумя векторами можно легко вычислить с помощью скалярного произведения.

Важным свойством скалярного произведения является то, что чем ниже угол между двумя единичными векторами, тем больше скалярное произведение стремится к значению 1. Когда угол между обоими векторами равен 90 градусам, скалярное произведение становится 0. То же самое относится и к θ: чем больше становится угол, тем меньше влияние света на цвет фрагмента.

Следует отметить, что для получения (только) косинуса угла между обоими векторами необходимо работать с единичными векторами (векторами с длиной, равной 1), поэтому нужно убедиться, что все векторы нормированы, в противном случае скалярное произведение будет возвращать больше, чем просто косинус.

Таким образом, полученное скалярное произведение возвращает скаляр, который можно использовать для вычисления влияния света на цвет фрагмента, что приводит к фрагментам разной освещенности в зависимости от их ориентации относительно света.

Чтобы найти воздействие рассеянного света на текущий фрагмент, нужно посчитать скалярное произведение нормали и вектора света. Полученное значение затем умножается на цвет рассеянного света и на коэффициент рассеянного освещения, в результате чего получается более темный фрагмент, если между обоими векторами большой угол.

$$I_d = k_d \cdot (N \cdot L) \cdot i_d, \tag{3.2}$$

где I_d — значение рассеянного освещения;

k_d — коэффициент рассеянного освещения;

i_d — цвет рассеянного света.

Если угол между обоими векторами больше 90 градусов, то результат скалярного произведения будет отрицательным, и рассеянная составляющая также будет отрицательной. По этой причине отрицательные значения всегда приравниваются к нулю (т.е. поверхность не освещается). Освещение для отрицательных значений не определено, поэтому этого лучше избегать.

Зеркальное освещение

Подобно рассеянному освещению, зеркальное освещение зависит от вектора направления света и от векторов нормалей объекта, но на этот раз оно также зависит и от направления взгляда. Зеркальное освещение основано на отражающих свойствах света. Если объект имеет зеркальную поверхность, то зеркальное освещение достигает максимального эффекта, независимо от положения света, отраженного от поверхности. Этот эффект можно увидеть на рисунке 3.2.

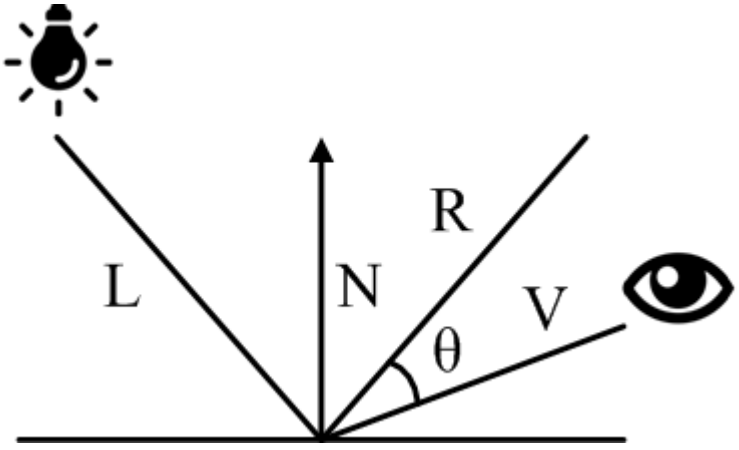

Рисунок 3.2 — Схема зеркального освещения

Чтобы вычислить вектор отражения, нужно отразить направление света относительно вектора нормали. Затем вычисляется угловое расстояние между этим вектором отражения и направлением взгляда, и чем меньше угол между

ними, тем больше влияние зеркального света. В результате получается эффект, при котором на поверхности объекта образуется светлый блик, который является отраженным от поверхности светом, попавшим в объектив камеры или глаз человека.

Вектор взгляда — это одна дополнительная переменная, которая нужна для нахождения зеркального освещения и которую можно вычислить, используя положение наблюдателя и положение фрагмента в мировом пространстве.

Иногда для вычисления освещения используют мировое пространство, но большинство людей предпочитает рассчитывать освещение в пространстве наблюдателя. Дополнительным преимуществом вычислений в пространстве наблюдателя является то, что позиция наблюдателя всегда находится в (0, 0, 0), поэтому не нужно находить эту позицию. Тем не менее, подсчет освещения в мировом пространстве более понятен. Если необходимо рассчитать освещение в пространстве наблюдателя, то нужно преобразовать все соответствующие векторы с помощью матрицы наблюдателя (а также векторы нормалей).

Теперь, когда есть все необходимые переменные, можно вычислить интенсивность зеркального освещения, используя следующую формулу:

$$I_s = k_s \cdot (R \cdot V)^{\propto} \cdot i_s, \tag{3.3}$$

где I_s — значение зеркального освещения;

k_s — коэффициент зеркального освещения;

$\propto$ — коэффициент блеска поверхности;

i_s — цвет зеркального света.

Сначала вычисляется скалярное произведение вектора взгляда и вектора отражения (результат должен быть неотрицательным), а затем результат возводится в степень $\propto$. Чем выше коэффициент блеска поверхности, тем лучше она отражает свет и хуже рассеивает его, и тем самым отраженный блик становится меньше.

Чтобы найти вектор отраженного луча света, можно воспользоваться следующей формулой:

$$R = L - 2 \cdot (L \cdot N) \cdot N. \tag{3.4}$$

Совмещение компонентов освещения

После того как были вычислены все основные компоненты освещения, можно воспользоваться итоговой формулой для расчета конечного цвета фрагмента:

$$I = I_a + I_d + I_s. \tag{3.5}$$

ЛАБОРАТОРНАЯ РАБОТА №4

В данной лабораторной работе необходимо реализовать алгоритмы наложения текстур различных типов на трехмерные объекты.

Текстуры представляют собой двухмерные массивы данных, например, данные цвета, данные яркости или данные прозрачности. Отдельные пиксели текстуры часто называют текселями. Важной особенностью текстурирования является то, что прямоугольная текстура может быть применена к непрямоугольным областям.

На рисунке 4.1 показан процесс наложения текстуры. Левая часть рисунка представляет всю текстуру, а салатовый контур на ней представляет собой пятиугольную фигуру, на которую нужно наложить эту текстуру. Когда пятиугольник отображается на экране, он может быть искажен после применения различных преобразований — вращения, перемещения, масштабирования и проецирования. На правой части рисунка показано, как пятиугольник с текстурой отобразится на экране после этих преобразований.

Рисунок 4.1 — Процесс наложения текстуры

Можно заметить, как текстура искажается в соответствии с искажением пятиугольника. В данном случае он сжат в направлении X, а также немного повернут.

Для наложения текстуры требуется, чтобы для каждой вершины полигона были заданы текстурные координаты. Эти координаты не зависят от размеров текстуры (имеют значения в диапазоне от 0 до 1) и задают каждой вершине полигона соответствующие тексели текстуры. Для заполнения полигона текселями используются такие же алгоритмы, как и при растеризации

треугольников. Но вместо нахождения координат фрагментов, заполняющих треугольник, нужно найти текстурные координаты для каждого фрагмента, интерполируя текстурные координаты, заданные в вершинах треугольника.

1) Реализовать алгоритм наложения диффузной карты

Диффузная карта задает основной цвет поверхности объекта (способность объекта рассеивать определенный спектр падающего на него света). Каждому фрагменту поверхности назначаются собственные значения коэффициентов фонового (k_a) и рассеянного (k_d) освещения, полученных из соответствующего текселя диффузной карты. Данный тип текстур очень важен и используется повсеместно для создания фотореалистичной графики. На рисунке 4.2 представлен пример диффузной карты.

Рисунок 4.2 — Пример диффузной текстуры

2) Реализовать алгоритм наложения карты нормалей

Затенение по Фонгу позволяет создавать плавное освещение поверхностей, используя линейное интерполирование нормалей вершин полигона. Тем не менее, для создания более реалистичного изображения можно использовать текстуры, называемые картами нормалей. В таких картах в каждом тексселе текстуры заданы нормали к соответствующим фрагментам поверхности. Такой подход позволяет визуализировать объекты с малым числом полигонов,

но с высокой детализацией, что увеличивает скорость рендеринга по сравнению с объектами, имеющими большое число полигонов.

В то время как векторы нормалей являются геометрическими объектами, текстуры обычно используются только для хранения информации о цвете, поэтому хранение нормалей в текстуре может быть неочевидным. Цвет в текстуре представлен вектором с компонентами R, G и B. Аналогичным образом можно сохранить компоненты X, Y и Z вектора нормали в соответствующих цветовых компонентах.

Т.к. компоненты вектора нормали находятся в диапазоне от -1 до 1, а компоненты цвета текстуры — от 0 до 1, требуется преобразовать значение цвета в значение вектора:

$$N = \| C_t \cdot 2 - 1 \|, \tag{4.1}$$

где C_t — цвет текстуры.

Существует две разновидности карт нормалей. На рисунке 4.3 слева представлена карта нормалей, векторы которой находятся в пространстве модели, поэтому эта текстура имеет разноцветный вид, т.к. все векторы могут смотреть в абсолютно любом направлении. Справа на рисунке показана карта нормалей, у которой все векторы находятся в пространстве плоскости полигона (в касательном пространстве). Эта карта имеет синий оттенок. Это связано с тем, что все нормали смотрят в положительном направлении оси Z, которая равна (0, 0, 1) (синий цвет). Карта нормалей в касательном пространстве очень подходит для создания анимации, т.к. для каждого состояния объекта потребовалась бы отдельная карта нормалей в пространстве модели.

Однако существует одна проблема, которая сильно ограничивает использование карт нормалей в касательном пространстве. Требуется преобразовать нормали таким образом, чтобы полученные векторы смотрели в том же направлении, что и плоскость полигона (т.е. нужно перейти из касательного пространства в мировое пространство). Другое решение заключается в том, чтобы перенести освещение и камеру в касательное пространство, а векторы нормалей оставить неизменными. Таким образом, можно использовать одну и ту же карту нормалей, независимо от ориентации объекта.

В данной лабораторной работе необходимо реализовать алгоритм наложения карты нормалей, векторы которой находятся в пространстве модели (слева на рисунке 4.3).

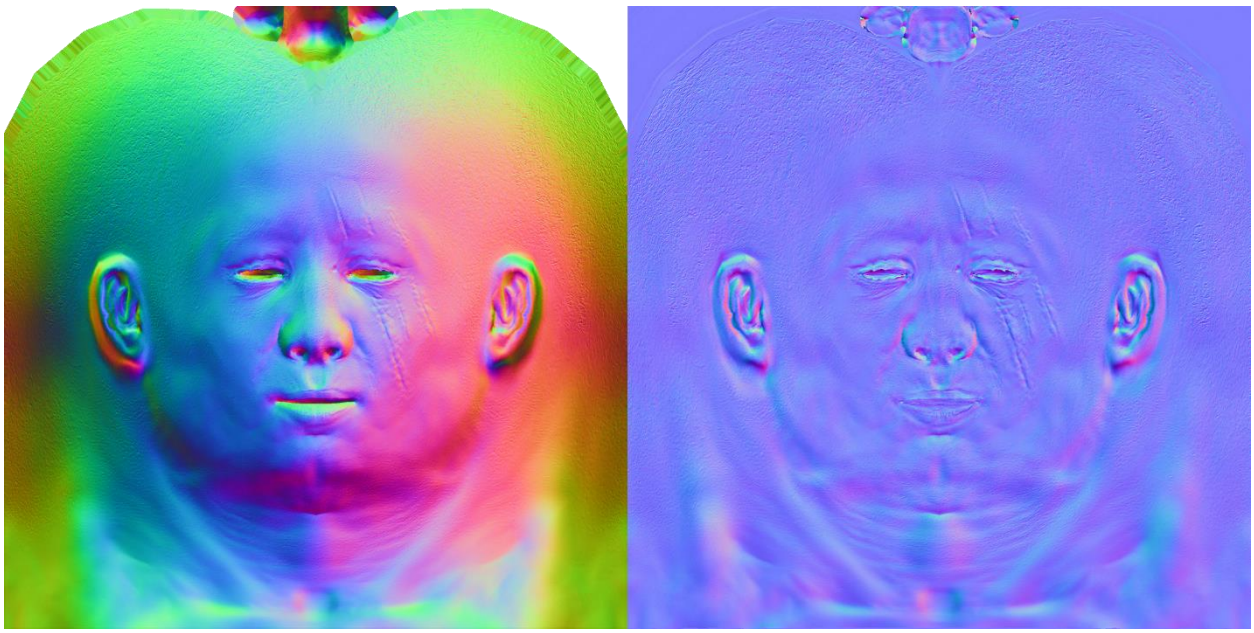

Рисунок 4.3 — Разновидности карт нормалей

3) Реализовать алгоритм наложения зеркальной карты

Зеркальная карта содержит данные о способности объекта отражать свет. В большинстве случаев данная текстура является черно-белой модификацией диффузной карты. Каждый тексель зеркальной карты содержит коэффициент зеркального освещения (k_s) для соответствующего фрагмента поверхности. На рисунке 4.4 представлен пример зеркальной карты.

Рисунок 4.4 — Пример зеркальной карты

4) Реализовать алгоритм интерполяции атрибутов вершин с перспективной коррекцией

Аффинное наложение текстур линейно интерполирует текстурные координаты по поверхности и является самым быстрым методом текстурирования. Поскольку данный способ не учитывает информацию о глубине фрагмента, в ситуации, когда поверхность расположена под углом к наблюдателю, можно заметить искажение текстуры (рисунок 4.5).

Наложение текстур с перспективной коррекцией учитывает положение фрагмента в трехмерном пространстве, а не просто интерполирует координаты в двухмерном пространстве экрана. Это позволяет достичь правильного отображение текстуры, но требует больше вычислительного времени.

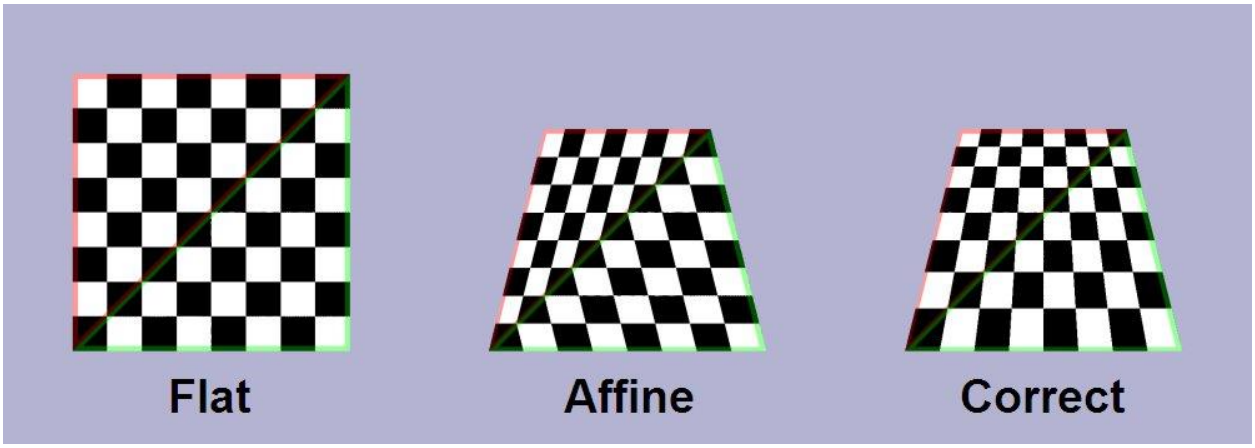

Рисунок 4.5 — Различие в способах наложения текстур

При аффинном наложении текстур текстурные координаты вершин линейно интерполируются по формуле:

$$uv_t = (1 - t) \cdot uv_0 + t \cdot uv_1, \tag{4.2}$$

где t — коэффициент интерполяции.

При наложении текстур с перспективной коррекцией текстурные координаты вершин линейно интерполируются после деления на глубину z, а затем полученное значение делится на интерполированное обратное значение глубины:

$$uv_t = \frac{(1 - t) \cdot \frac{uv_0}{z_0} + t \cdot \frac{uv_1}{z_1}}{(1 - t) \cdot \frac{1}{z_0} + t \cdot \frac{1}{z_1}}. \tag{4.3}$$