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:

$$\begin{gathered} ZAxis = normalize(eye - target), \\ XAxis = normalize(up \times ZAxis), \\ YAxis = up. \end{gathered} \tag{1.7}$$

Итоговая матрица будет выглядеть следующим образом:

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

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:

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

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:

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

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 $2 \times error > (x_2 - x_1)$.