Компьютерная графика

ЛЕКЦИЯ 5.

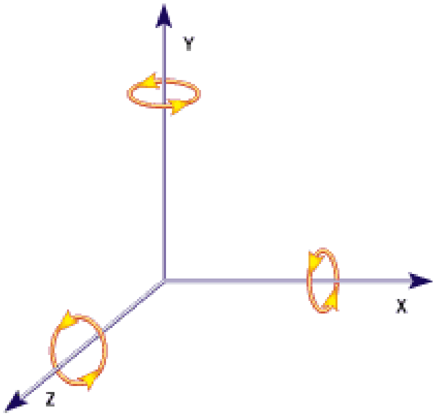
РАЗНЫЕ УЛУЧШЕНИЯ

Кватернионы

Кватернионы для поворота, принцип работы

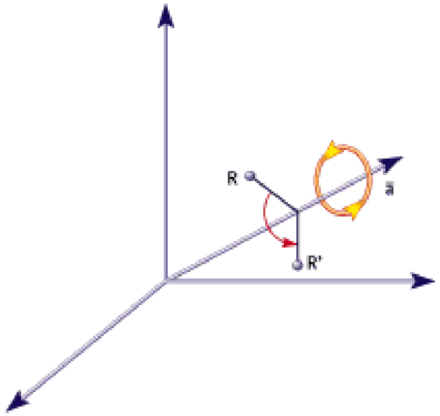
УГЛЫ ЭЙЛЕРА

GD,9801,Fig1



КВАТЕРНИОНЫ

GD,9801,Fig2



Кватернионы

Кватернион – это система чисел, расширяющая систему комплексных чисел.

Кватернион традиционно представляется в форме

$a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$

где

a, b, c, d – вещественные числа, а $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$ – базовые кватернионы.

Их можно интерпретировать как мнимые единицы, связанные

друг с другом следующим образом:

$\mathbf{i^2 + j^2 + k^2 = ijk = -1}$

Сложение и умножение

Операция сложения:

$$(a_1 + b_1\mathbf{i} + c_1\mathbf{j} + d_1\mathbf{k}) + (a_2 + b_2\mathbf{i} + c_2\mathbf{j} + d_2\mathbf{k}) = \big((a_1 + a_2)\ + (b_1 + b_2)\mathbf{i} + (c_1 + c_2)\mathbf{j} + (d_1 + d_2)\mathbf{k}\big)$$

Операция умножения:

$$(a_1 + b_1\mathbf{i} + c_1\mathbf{j} + d_1\mathbf{k})(a_2 + b_2\mathbf{i} + c_2\mathbf{j} + d_2\mathbf{k}) =$$

$$(a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2) +$$

$$(a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2)\mathbf{i} +$$

$$(a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2)\mathbf{j} +$$

$$(a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2)\mathbf{k}$$

Сопряжение и норма

Операция сопряжения:

$$(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k})^* = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$$

Норма кватерниона:

$$\|a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}\| = \sqrt{(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k})(a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k})^*} = \sqrt{a^2 + b^2 + c^2 + d^2}$$

Поворот с помощью кватерниона

Единичным кватернионом называется кватернион, норма которого равна единице.

Рассмотрим поворот вокруг оси $\mathrm{u} = [u_x, u_y, u_z]$ на угол θ.

Пусть вектор $\mathbf{u}$ единичный, тогда этому повороту можно поставить в соответствие единичный кватернион

$$q = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$$

Можно показать, что для поворота вектора $[x, y, z]$ с использованием кватерниона q нужно выполнить умножение

$$p' = qpq^*$$

где $p = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$.

Последовательность поворотов

Для получения кватерниона двух последовательных вращений достаточно перемножить кватернионы соответствующих вращений.

$$p'' = q_2 p' q_2^*$$

$$p'' = q_2 q_1 p q_1^* q_2^* = q_2 q_1 p (q_2 q_1)^*$$

Матрица поворота

Можно использовать кватернионы поворота не только в их исходной форме.

Существует однозначное преобразование кватерниона поворота в матрицу поворота (обратное в общем случае может быть неверным).

Матрица поворота может быть получена с помощью следующей формулы):

$$\mathbf{R} = \begin{pmatrix} a^2 + b^2 - c^2 - d^2 & 2bc - 2ad & 2bd + 2ac \\ 2bc + 2ad & a^2 - b^2 + c^2 - d^2 & 2cd - 2ab \\ 2bd - 2ac & 2cd + 2ab & a^2 - b^2 - c^2 + d^2 \end{pmatrix}$$

Связь с углами Эйлера

Можно заметить, что трём матрицам поворота вокруг осей x, y и z соответствуют три кватерниона:

$$q_x = \cos\left(\frac{\alpha}{2}\right) + \sin(\frac{\alpha}{2})\mathbf{i}$$

$$q_y = \cos\left(\frac{\beta}{2}\right) + \sin\left(\frac{\beta}{2}\right)\mathbf{j}$$

$$q_z = \cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\gamma}{2}\right)\mathbf{k}$$

Тогда кватернион, соответствующий повороту относительно углов

Эйлера выглядит как:

$$q_x q_y q_z = \left(\cos\left(\frac{\alpha}{2}\right) + \sin\left(\frac{\alpha}{2}\right)\mathbf{i}\right)\left(\cos\left(\frac{\beta}{2}\right) + \sin\left(\frac{\beta}{2}\right)\mathbf{j}\right)\left(\cos\left(\frac{\gamma}{2}\right) + \sin\left(\frac{\gamma}{2}\right)\mathbf{k}\right)$$

Преимущества кватернионов

Меньше избыточности.

Гладкое вращение. Возможность реализации сферической линейной интерполяции.

Накопление ошибок округления приводят к изменению нормы вектора, что устраняется намного легче, чем не ортогональность матрицы поворота.

Последняя лабораторная

Чек-лист на пятую лабораторную

Парсер:

- Если есть координаты текстур (vt) и задан файл текстуры, они должны быть загружены из obj-файла.
- Парсер должен корректно срабатывать для полигонов, состоящих из четырёх и более вершин. В том числе, в случае, когда в одном файле содержатся полигоны с разным количеством вершин.
- Парсер должен корректно отрабатываться для корректно составленного obj-файла. В obj-файле могут находиться пустые строки.

Трёхмерные преобразования координат:

- Должна быть возможность задания поворота, сдвига и масштаба **модели**.
- Должна быть возможность задания поворота модели как в виде углов Эйлера (в градусах или радианах), так в виде кватернионов.

Общее:

- Программа должна допускать загрузку второй модели (или повторную загрузку той же) с другими трёхмерными параметрами и её отрисовку на том же изображении (с учётом ранее полученного z-буфера).

Рендер:

- Должна существовать возможность задания размеров изображения и коэффициента для проецирования. Эти свойства должны быть заданы в виде параметров, то есть их изменение должно производиться в одном месте и не требовать изменения остального кода. Изображение может быть задано прямоугольным.
- Программа должна корректно отрабатывать случай, когда модель частично находится за пределами изображения.
- Тонировка Гуро должна применяться к модели, независимо от того, задана ли текстура или нет. Если в obj-файле есть нормали, они должны загружаться из файла, в противном случае, они вычисляются через усреднение нормалей к полигонам.
- На изображении не должно быть систематических артефактов, например, вертикальных и горизонтальных полос. Допускается наличие мелких погрешностей отрисовки.
- При визуализации модели без поворота и сдвига, изображение не должно быть повёрнутым/перевёрнутым/зеркально отражённым.
- Должна существовать возможность сохранить изображение (не только вывести на экран).

Необязательное

Поворот, смещение и масштабирование

Предположим, что есть задача: применить к точкам модели поворот $\mathbf{R_1}$, сдвиг $\mathbf{t_1}$, затем снова поворот $\mathbf{R_2}$ и сдвиг $\mathbf{t_2}$.

Как это будет выглядеть в привычном виде?

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \mathbf{R_1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t_1}, \begin{bmatrix} X'' \\ Y'' \\ Z'' \end{bmatrix} = \mathbf{R_2} \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} + \mathbf{t_2}$$

Если мы захотим объединить это в одну операцию, итоговые преобразования будут иметь вид:

$$\begin{bmatrix} X'' \\ Y'' \\ Z'' \end{bmatrix} = \mathbf{R_2} \left(\mathbf{R_1} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \mathbf{t_1} \right) + \mathbf{t_2} = (\mathbf{R_2 R_1}) \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + (\mathbf{R_2 t_1} + \mathbf{t_2})$$

Можно ли сделать это более единообразно?

Однородные 4D координаты

Будем работать с координатами $\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$. Тогда операции поворота и смещения могут быть записаны в виде:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & 0 \\ R_{21} & R_{22} & R_{23} & 0 \\ R_{31} & R_{32} & R_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \qquad \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_X \\ 0 & 1 & 0 & t_Y \\ 0 & 0 & 1 & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Такая запись позволяет объединить поворот и смещение в одну операцию:

$$\begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_X \\ R_{21} & R_{22} & R_{23} & t_Y \\ R_{31} & R_{32} & R_{33} & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}.$$

Комбинировать преобразования становится проще: достаточно перемножить матрицы 4x4 этих преобразований (в обратном порядке).

Проективные преобразования «как в OpenGL»

В графических движках проективные преобразования реализованы немного сложнее.

Для преобразования используется формула:

$$\begin{bmatrix} \boldsymbol{x_p} \\ \boldsymbol{y_p} \\ \boldsymbol{z_p} \\ \mathbf{1} \end{bmatrix} \propto \begin{bmatrix} \boldsymbol{e_x} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{e_y} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{\frac{f+n}{f-n}} & \boldsymbol{-\frac{2fn}{f-n}} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \\ \mathbf{1} \end{bmatrix}.$$

Здесь $\boldsymbol{e_x}$ и $\boldsymbol{e_y}$ определяют угол обзора $\boldsymbol{e} = \frac{\mathbf{1}}{\boldsymbol{tan}\left(\frac{\boldsymbol{FOV}}{\mathbf{2}}\right)}$, а $\boldsymbol{n}$ и $\boldsymbol{f}$ – ближняя и дальняя плоскости, соответственно.

Раскроем формулу

$$\begin{bmatrix} x_p \\ y_p \\ z_p \\ 1 \end{bmatrix} \propto \begin{bmatrix} e_x & 0 & 0 & 0 \\ 0 & e_y & 0 & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} e_x X \\ e_y Y \\ \frac{f+n}{f-n} Z - \frac{2fn}{f-n} \\ Z \end{bmatrix}.$$

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{Z} \\ \frac{e_y Y}{Z} \\ \frac{f+n}{f-n} - \frac{2fn}{f-n} \frac{1}{Z} \end{bmatrix}$$

Нормализованные координаты устройства

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{Z} \\ \frac{e_y Y}{Z} \\ \frac{f+n}{f-n} - \frac{2fn}{f-n}\frac{1}{Z} \end{bmatrix}$$

Эти координаты называются нормализованными координатами устройства.

На изображение попадут только точки, находящиеся в пределах от -1.0 до 1.0 по каждой координате.

Ближняя и дальняя плоскости

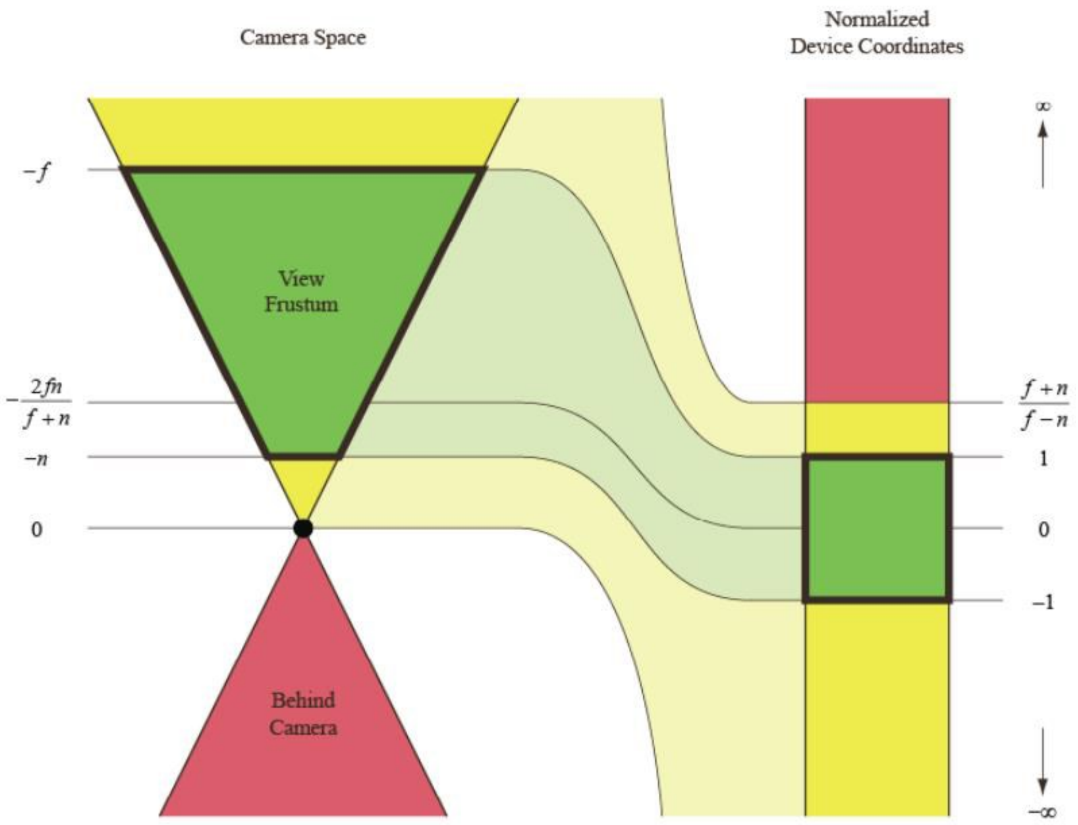
$\boldsymbol{Z = n}$:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{n} \\ \frac{e_y Y}{n} \\ \frac{f+n}{f-n} - \frac{2fn}{f-n}\frac{1}{n} \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{n} \\ \frac{e_y Y}{n} \\ \frac{f+n}{f-n} - \frac{2f}{f-n} \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{n} \\ \frac{e_y Y}{n} \\ -1 \end{bmatrix}$$

$\boldsymbol{Z = f}$:

$$\begin{bmatrix} x_p \\ y_p \\ z_p \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{f} \\ \frac{e_y Y}{f} \\ \frac{f+n}{f-n} - \frac{2fn}{f-n}\frac{1}{f} \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{n} \\ \frac{e_y Y}{n} \\ \frac{f+n}{f-n} - \frac{2n}{f-n} \end{bmatrix} = \begin{bmatrix} \frac{e_x X}{n} \\ \frac{e_y Y}{n} \\ 1 \end{bmatrix}$$

NDC (визуально)



Из NDC в пиксельные координаты

При использовании NDC финальное преобразование в пиксельные координаты становится предельно простым:

$$u = \left(1.0 + x_p\right)\frac{ImageSizeX}{2}, \qquad v = \left(1.0 + y_p\right)\frac{ImageSizeY}{2}$$

Поворот и смещение камеры

Иногда возникает необходимость повернуть не модель, а переместить камеру. Это полезно, например, в случае, когда на сцене расположено несколько объектов и задачей является рендер модели с различных точек.

$$\mathbf{R_1}\begin{bmatrix} \boldsymbol{X'} \\ \boldsymbol{Y'} \\ \boldsymbol{Z'} \end{bmatrix} + \mathbf{t_1} = \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \end{bmatrix} \Rightarrow$$

$$\begin{bmatrix} \boldsymbol{X'} \\ \boldsymbol{Y'} \\ \boldsymbol{Z'} \end{bmatrix} = \mathbf{R_1^{-1}}\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{Y} \\ \boldsymbol{Z} \end{bmatrix} - \mathbf{R_1^{-1}t_1}$$

Таким образом, поворот камеры выполняется с помощью тех же преобразований, но с другими параметрами.