Компьютерная графика

Лекция 4: Камера, ортографическая и перспективная проекции, буфер глубины

2021

Проекции

- Мир – двухмерный/трёхмерный, произвольного размера, смотрим с произвольного ракурса

Проекции

- Мир – двухмерный/трёхмерный, произвольного размера, смотрим с произвольного ракурса
- Экран – двухмерный, $[-1, 1] \times [-1, 1]$

Проекции

- Мир – двухмерный/трёхмерный, произвольного размера, смотрим с произвольного ракурса
- Экран – двухмерный, $[-1, 1] \times [-1, 1]$
  - Координата Z тоже $[-1, 1]$
  - Про неё поговорим подробнее чуть позже

Проекции

- Мир – двухмерный/трёхмерный, произвольного размера, смотрим с произвольного ракурса
- Экран – двухмерный, $[-1, 1] \times [-1, 1]$
  - Координата Z тоже $[-1, 1]$
  - Про неё поговорим подробнее чуть позже
- За преобразование отвечает т.н. проекция

Ортографическая проекция

- Самый простой способ:

Ортографическая проекция

- Самый простой способ: игнорировать третью координату

Концептуально: $(x, y, z) \mapsto (x, y)$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ортографическая проекция

- Самый простой способ: игнорировать третью координату

$$\text{Концептуально: } (x, y, z) \mapsto (x, y)$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Именно это делает OpenGL (если `gl_Position.w = 1`)

Ортографическая проекция

- Самый простой способ: игнорировать третью координату

$$\text{Концептуально: } (x, y, z) \mapsto (x, y)$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Именно это делает OpenGL (если `gl_Position.w = 1`)
- X и Y всё ещё $[-1, 1]$

Ортографическая проекция

- Самый простой способ: игнорировать третью координату

$$\text{Концептуально: } (x, y, z) \mapsto (x, y)$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Именно это делает OpenGL (если `gl_Position.w = 1`)
- X и Y всё ещё $[-1, 1]$
- Что, если $X \in [-W, W]$ и $Y \in [-H, H]$, и нам нужно их перевести в стандартные для OpenGL $[-1, 1]^2$?

Ортографическая проекция

- Самый простой способ: игнорировать третью координату

$$\text{Концептуально: } (x, y, z) \mapsto (x, y)$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Именно это делает OpenGL (если `gl_Position.w = 1`)
- X и Y всё ещё $[-1, 1]$
- Что, если $X \in [-W, W]$ и $Y \in [-H, H]$, и нам нужно их перевести в стандартные для OpenGL $[-1, 1]^2$?

$$\begin{pmatrix} \frac{1}{W} & 0 & 0 & 0 \\ 0 & \frac{1}{H} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ортографическая проекция

- Что, если $X \in [X_0 - W, X_0 + W]$ и $Y \in [Y_0 - H, Y_0 + H]$?

Ортографическая проекция

- Что, если $X \in [X_0 - W, X_0 + W]$ и $Y \in [Y_0 - H, Y_0 + H]$?
  - Сдвинуть на $(-X_0, -Y_0)$, а затем применить масштабирование:
    $$\begin{pmatrix} \frac{1}{W} & 0 & 0 & 0 \\ 0 & \frac{1}{H} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ортографическая проекция

- Что, если $X \in [X_0 - W, X_0 + W]$ и $Y \in [Y_0 - H, Y_0 + H]$?
  - Сдвинуть на $(-X_0, -Y_0)$, а затем применить масштабирование:
  $$\begin{pmatrix} \frac{1}{W} & 0 & 0 & 0 \\ 0 & \frac{1}{H} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$
- Если размер области W, нужно разделить на W
- Если центр в X_0, нужно сдвинуть на $-X_0$

Ортографическая проекция

- Что, если $X \in [X_0 - W, X_0 + W]$ и $Y \in [Y_0 - H, Y_0 + H]$?
  - Сдвинуть на $(-X_0, -Y_0)$, а затем применить масштабирование:

$$\begin{pmatrix} \frac{1}{W} & 0 & 0 & 0 \\ 0 & \frac{1}{H} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Если размер области W, нужно разделить на W
- Если центр в X_0, нужно сдвинуть на $-X_0$
- Общая идея: если камера получена каким-то преобразованием, нужно применить обратное преобразование

Ортографическая проекция: обратное преобразование

- Можно считать, что по умолчанию (если `gl_Position.w = 1`) OpenGL делает ортографическую проекцию на квадрат $[-1, 1]^2$ в плоскости XY параллельно оси Z
- Камера находится в начале координат и никак явным образом не настраивается

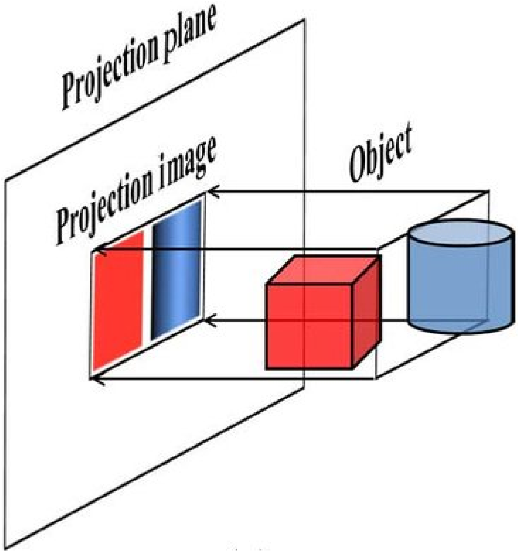
Ортографическая проекция: обратное преобразование

- Можно считать, что по умолчанию (если `gl_Position.w = 1`) OpenGL делает ортографическую проекцию на квадрат $[-1, 1]^2$ в плоскости XY параллельно оси Z
- Камера находится в начале координат и никак явным образом не настраивается
- Если ко всему виртуальному миру (включая камеру!) применить аффинное преобразование, изображение не изменится

Ортографическая проекция: обратное преобразование

- Можно считать, что по умолчанию (если `gl_Position.w = 1`) OpenGL делает ортографическую проекцию на квадрат $[-1, 1]^2$ в плоскости XY параллельно оси Z
- Камера находится в начале координат и никак явным образом не настраивается
- Если ко всему виртуальному миру (включая камеру!) применить аффинное преобразование, изображение не изменится
- $\Rightarrow$ Если наша камера получена аффинным преобразованием из стандартной камеры OpenGL, к объектам мира нужно применить обратное преобразование

Ортографическая проекция



Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать

Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать
  - Положением камеры $C = (C_x, C_y, C_z)$

Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать
  - Положением камеры $C = (C_x, C_y, C_z)$
  - Осями координат камеры $X = (X_x, X_y, X_z), Y = (Y_x, Y_y, Y_z), Z = (Z_x, Z_y, Z_z)$

Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать
  - Положением камеры $C = (C_x, C_y, C_z)$
  - Осями координат камеры $X = (X_x, X_y, X_z), Y = (Y_x, Y_y, Y_z), Z = (Z_x, Z_y, Z_z)$
- Делает проекцию параллельно вектору Z на параллелограмм $C \pm X \pm Y$
- Параллелограмм отождествляется с экраном

Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать
  - Положением камеры $C = (C_x, C_y, C_z)$
  - Осями координат камеры $X = (X_x, X_y, X_z), Y = (Y_x, Y_y, Y_z), Z = (Z_x, Z_y, Z_z)$
- Делает проекцию параллельно вектору Z на параллелограмм $C \pm X \pm Y$
- Параллелограмм отождествляется с экраном
- Обычно X, Y, Z взаимно ортогональны

Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать
  - Положением камеры $C = (C_x, C_y, C_z)$
  - Осями координат камеры $X = (X_x, X_y, X_z), Y = (Y_x, Y_y, Y_z), Z = (Z_x, Z_y, Z_z)$
- Делает проекцию параллельно вектору Z на параллелограмм $C \pm X \pm Y$
- Параллелограмм отождествляется с экраном
- Обычно X, Y, Z взаимно ортогональны
- Как выразить эту проекцию матрицей?

Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать
  - Положением камеры $C = (C_x, C_y, C_z)$
  - Осями координат камеры $X = (X_x, X_y, X_z), Y = (Y_x, Y_y, Y_z), Z = (Z_x, Z_y, Z_z)$
- Делает проекцию параллельно вектору Z на параллелограмм $C \pm X \pm Y$
- Параллелограмм отождествляется с экраном
- Обычно X, Y, Z взаимно ортогональны
- Как выразить эту проекцию матрицей?
- Преобразование из стандартной системы координат OpenGL $[-1, 1]^3$ в координаты области, видимой через эту камеру: смена системы координат + параллельный перенос

$$\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ортографическая проекция: общий случай

- В общем случае ортографическую камеру можно задать
  - Положением камеры $C = (C_x, C_y, C_z)$
  - Осями координат камеры
    $X = (X_x, X_y, X_z), Y = (Y_x, Y_y, Y_z), Z = (Z_x, Z_y, Z_z)$
- Делает проекцию параллельно вектору Z на параллелограмм $C \pm X \pm Y$
- Параллелограмм отождествляется с экраном
- Обычно X, Y, Z взаимно ортогональны
- Как выразить эту проекцию матрицей?
- Преобразование из стандартной системы координат OpenGL $[-1, 1]^3$ в координаты области, видимой через эту камеру: смена системы координат + параллельный перенос

$$\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Обратное преобразование (матрица проекции) – обратная матрица

Ортографическая проекция: общий случай

- Можно представить X, Y, Z как произведение длины и нормированного вектора:

$$X = W \cdot \hat{X} \quad Y = H \cdot \hat{Y} \quad Z = D \cdot \hat{Z}$$

Ортографическая проекция: общий случай

- Можно представить X, Y, Z как произведение длины и нормированного вектора:
$$X = W \cdot \hat{X} \quad Y = H \cdot \hat{Y} \quad Z = D \cdot \hat{Z}$$
- Матрицу можно разбить на масштабирование, поворот и перенос:
$$\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} =$$
$$\begin{pmatrix} 1 & 0 & 0 & C_x \\ 0 & 1 & 0 & C_y \\ 0 & 0 & 1 & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} W & 0 & 0 & 0 \\ 0 & H & 0 & 0 \\ 0 & 0 & D & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ортографическая проекция: общий случай

- Тогда обратная матрица (матрица проекции):

$$\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} =$$

$$\begin{pmatrix} \frac{1}{W} & 0 & 0 & 0 \\ 0 & \frac{1}{H} & 0 & 0 \\ 0 & 0 & \frac{1}{D} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ортографическая проекция: общий случай

- Тогда обратная матрица (матрица проекции):

$$\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} =$$

$$\begin{pmatrix} \frac{1}{W} & 0 & 0 & 0 \\ 0 & \frac{1}{H} & 0 & 0 \\ 0 & 0 & \frac{1}{D} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 1 & 0 & 0 & -C_x \\ 0 & 1 & 0 & -C_y \\ 0 & 0 & 1 & -C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Эту матрицу мы передаём в шейдер и применяем к входным вершинам

Ортографическая проекция: ортогональный случай

- Если X, Y, Z ортогональны, то

$$\begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{T} = \begin{pmatrix} \hat{X}_x & \hat{X}_y & \hat{X}_z & 0 \\ \hat{Y}_x & \hat{Y}_y & \hat{Y}_z & 0 \\ \hat{Z}_x & \hat{Z}_y & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Ортографическая проекция: применение

Ортографическая проекция: применение

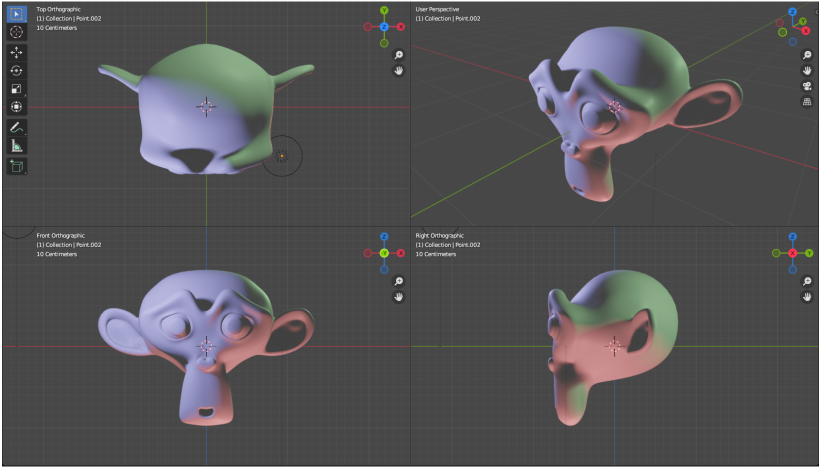
- 2D рендеринг: 2D игры, UI, карты


SUPER
MARIO BROS.


Поиск мест и адресов
Норвежское море
Печорское море
Мурманск
Воркута
Норильск
ИСЛАНДИЯ
НОРВЕГИЯ
Архангельск
ФИНЛЯНДИЯ
ШВЕЦИЯ
Петрозаводск
Сыктывкар
Хельсинки
Стокгольм
Санкт-Петербург
ЭСТОНИЯ
Пермь
ЛАТВИЯ
Северное море
ДАНИЯ
Нижний Новгород
Екатеринбург
Тюмень
Москва
Казань
ЛИТВА
Калининград
Минск
Уфа
Челябинск
Омск
Новосибирск
Атлантический океан
ИРЛАНДИЯ
ВЕЛИКОБРИТАНИЯ
НИДЕРЛАНДЫ
Берлин
БЕЛАРУСЬ
Самара
ПОЛЬША
Воронеж
Нур-Султан (Астана)
Лондон
ГЕРМАНИЯ
Киев
Харьков
БЕЛЬГИЯ
Львов
Париж
ЧЕХИЯ
УКРАИНА
Волгоград
СЛОВАКИЯ
Днепр
КАЗАХСТАН
АВСТРИЯ
МОЛДОВА
Ростов-на-Дону
ФРАНЦИЯ
ШВЕЙЦАРИЯ
ВЕНГРИЯ
РУМЫНИЯ
Одесса
ХОРВАТИЯ
Краснодар
ИТАЛИЯ
Бухарест
СЕРБИЯ
Чёрное море
Махачкала
Алматы
Рим
БОЛГАРИЯ
Мадрид
СЕВЕРНАЯ МАКЕДОНИЯ
ГРУЗИЯ
УЗБЕКИСТАН
Ташкент
КИРГИЗИЯ
Стамбул
АЛБАНИЯ
АЗЕРБАЙДЖАН
ПОРТУГАЛИЯ
ИСПАНИЯ
ГРЕЦИЯ
ТУРЦИЯ
ТУРКМЕНИСТАН
ТАДЖИКИСТАН
Ашхабад
Алжир
Тунис
Афины
Тегеран
ТУНИС
СИРИЯ
ИРАК
Кабул
Касабланка
ИРАН
АФГАНИСТАН
Триполи
Амман
Яндекс
© Яндекс Разместить рекламу Редактировать карту API Карт Справка Условия использования
300 км

Ортографическая проекция: применение

- 2D рендеринг: 2D игры, UI, карты
- Проектировние моделей, зданий, деталей (вид сверху, вид сбоку, вид спереди)


Top Orthographic
(1) Collection | Point.002
10 Centimeters
User Perspective
(1) Collection | Point.002
Front Orthographic
(1) Collection | Point.002
10 Centimeters
Right Orthographic
(1) Collection | Point.002
10 Centimeters

Ортографическая проекция: применение

- 2D рендеринг: 2D игры, UI, карты
- Проектировние моделей, зданий, деталей (вид сверху, вид сбоку, вид спереди)
- Стилизация: 3D мир с ортографической проекцией (напр. изометрическая проекция)


Ixtapaluca
2 Abydos : 10
Temple : 13
3 Elephantine : 9
Settler : 25
2 Huexotla
2 Pi-Ramesses : 6
Settler : 20
War Chariot
Regular
2.1.2/2
Grassland
Egypt - Despotism (6.1.3)
290 BC 443 Gold (+14 per turn)
The Republic (9 turns)


V0.23698
MEGAERA, THE FURY
Lv.3
+1
Lv.2
23 / 100
1 / 1
1
23

Ортографическая проекция: проблемы

Ортографическая проекция: проблемы

- Объекты на разном расстоянии от камеры выглядят одинаково

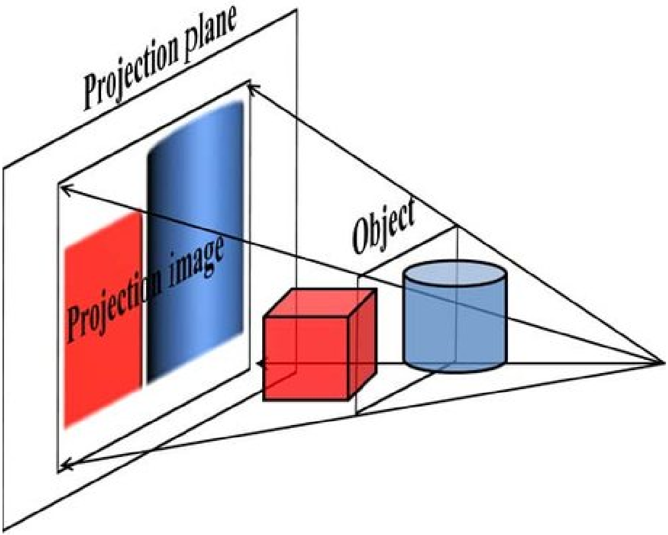
Ортографическая проекция: проблемы

- Объекты на разном расстоянии от камеры выглядят одинаково
- Нельзя оценить расстояние до объекта по его изображению

Ортографическая проекция: проблемы

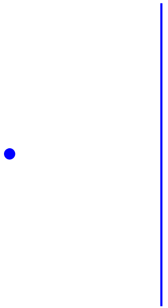
- Объекты на разном расстоянии от камеры выглядят одинаково
- Нельзя оценить расстояние до объекта по его изображению
- Реальные камеры и глаза работают не так

Перспективная проекция



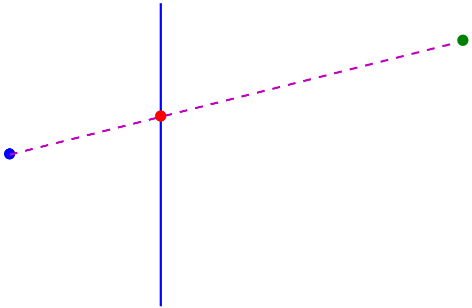
Перспективная проекция

- Есть центр проекции и плоскость проекции
-

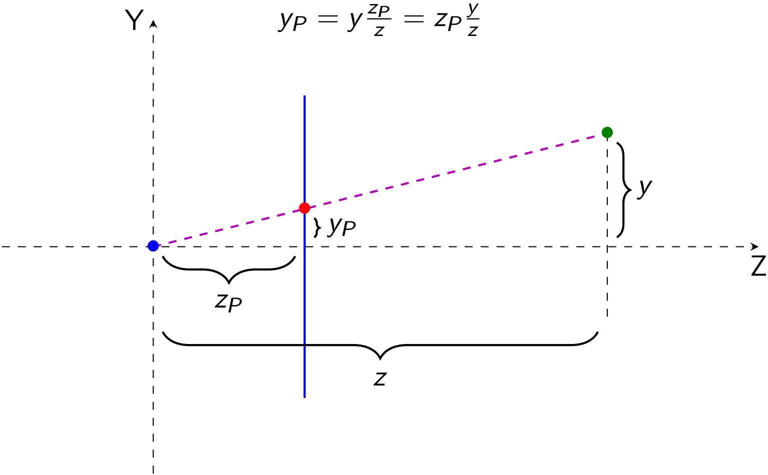


Перспективная проекция

- Есть центр проекции и плоскость проекции
- Проекция точки – пересечение прямой, проходящей через эту точку и центр проекции, с плоскостью проекции



Перспективная проекция



Перспективная проекция

- Чтобы вычислить перспективную проекцию с центром в начале координат и плоскостью проеции $Z = z_P$, нужно разделить на Z-координату:
$$(x, y, z) \mapsto \left(z_P \frac{x}{z}, z_P \frac{y}{z}\right)$$

Перспективная проекция

- Чтобы вычислить перспективную проекцию с центром в начале координат и плоскостью проеции $Z = z_P$, нужно разделить на Z-координату:
$$(x, y, z) \mapsto \left(z_P \frac{x}{z}, z_P \frac{y}{z}\right)$$
- Как выразить эту проекцию матрицей?

Перспективная проекция

- Чтобы вычислить перспективную проекцию с центром в начале координат и плоскостью проеции $Z = z_P$, нужно разделить на Z-координату:
$$(x, y, z) \mapsto \left(z_P \frac{x}{z}, z_P \frac{y}{z}\right)$$
- Как выразить эту проекцию матрицей? Никак: матрицы не умеют делить одни координаты на другие

Perspective divide

- Чтобы поддержать перспективную проекцию, в графическом конвейере (после вершинного шейдера, перед переводом в экранные координаты) есть специальный обязательный шаг: perspective divide

Perspective divide

- Чтобы поддержать перспективную проекцию, в графическом конвейере (после вершинного шейдера, перед переводом в экранные координаты) есть специальный обязательный шаг: perspective divide
- `gl_Position` делится на последнюю координату `gl_Position.w`

$$(x, y, z, w) \mapsto \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$$

Perspective divide

- Чтобы поддержать перспективную проекцию, в графическом конвейере (после вершинного шейдера, перед переводом в экранные координаты) есть специальный обязательный шаг: perspective divide
- `gl_Position` делится на последнюю координату `gl_Position.w`

$$(x, y, z, w) \mapsto \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$$

- Если $w = 1$, ничего не меняется (ортографическая проекция)

Perspective divide

- Чтобы поддержать перспективную проекцию, в графическом конвейере (после вершинного шейдера, перед переводом в экранные координаты) есть специальный обязательный шаг: perspective divide
- `gl_Position` делится на последнюю координату `gl_Position.w`

$$(x, y, z, w) \mapsto \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w}\right)$$

- Если $w = 1$, ничего не меняется (ортографическая проекция)
- Если $w =$ расстояние до камеры, получается перспективная проекция

Перспективная проекция

- Как выразить перспективную проекцию матрицей с последующим perspective divide?

Перспективная проекция

- Как выразить перспективную проекцию матрицей с последующим perspective divide?

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z \end{pmatrix} \mapsto \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}$$

Перспективная проекция

- Как выразить перспективную проекцию матрицей с последующим perspective divide?

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ z \end{pmatrix} \mapsto \begin{pmatrix} x/z \\ y/z \\ 1 \end{pmatrix}$$

- Обычно матрица перспективной проекции выглядит **не так**

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке
  - 2D игра-платформер: фон, потом ландшафт, потом персонажи, потом эффекты

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке
  - 2D игра-платформер: фон, потом ландшафт, потом персонажи, потом эффекты
  - Карта: ландшафт, потом дороги и здания, потом иконки заведений

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке
  - 2D игра-платформер: фон, потом ландшафт, потом персонажи, потом эффекты
  - Карта: ландшафт, потом дороги и здания, потом иконки заведений
  - UI: фон виджета, потом рисунок кнопки, потом текст кнопки

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке
  - 2D игра-платформер: фон, потом ландшафт, потом персонажи, потом эффекты
  - Карта: ландшафт, потом дороги и здания, потом иконки заведений
  - UI: фон виджета, потом рисунок кнопки, потом текст кнопки
- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это очень медленно:

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке
  - 2D игра-платформер: фон, потом ландшафт, потом персонажи, потом эффекты
  - Карта: ландшафт, потом дороги и здания, потом иконки заведений
  - UI: фон виджета, потом рисунок кнопки, потом текст кнопки
- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это очень медленно:
  - Нужно отсортировать миллионы треугольников каждый кадр

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке
  - 2D игра-платформер: фон, потом ландшафт, потом персонажи, потом эффекты
  - Карта: ландшафт, потом дороги и здания, потом иконки заведений
  - UI: фон виджета, потом рисунок кнопки, потом текст кнопки
- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это очень медленно:
  - Нужно отсортировать миллионы треугольников каждый кадр
  - Нужно загрузить их на GPU

Наложение объектов

- По умолчанию примитивы, рисующиеся позже, накладываются поверх уже нарисованных
- Обычно не проблема в 2D рендеринге: рисуем слои в нужном порядке
  - 2D игра-платформер: фон, потом ландшафт, потом персонажи, потом эффекты
  - Карта: ландшафт, потом дороги и здания, потом иконки заведений
  - UI: фон виджета, потом рисунок кнопки, потом текст кнопки
- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это очень медленно:
  - Нужно отсортировать миллионы треугольников каждый кадр
  - Нужно загрузить их на GPU
  - Треугольники одного объекта не идут непрерывно $\Rightarrow$ больше изменений состояния (переключений шейдеров, etc) при рендеринге

Z-буфер

- Решение: буфер глубины (z-buffer)

Z-буфер

- Решение: буфер глубины (z-buffer)
- Хранит для каждого пикселя экрана расстояние до камеры

Z-буфер

- Решение: буфер глубины (z-buffer)
- Хранит для каждого пикселя экрана расстояние до камеры
- Тест глубины: при рисовании очередного пикселя (после фрагментного шейдера) проверим: если уже нарисованный пиксель ближе к камере, то новый рисовать не будем

Z-буфер

- Решение: буфер глубины (z-buffer)
- Хранит для каждого пикселя экрана расстояние до камеры
- Тест глубины: при рисовании очередного пикселя (после фрагментного шейдера) проверим: если уже нарисованный пиксель ближе к камере, то новый рисовать не будем
- Включить: `glEnable(GL_DEPTH_TEST)`

Z-буфер

- Решение: буфер глубины (z-buffer)
- Хранит для каждого пикселя экрана расстояние до камеры
- Тест глубины: при рисовании очередного пикселя (после фрагментного шейдера) проверим: если уже нарисованный пиксель ближе к камере, то новый рисовать не будем
- Включить: `glEnable(GL_DEPTH_TEST)`
- Настроить поведение теста глубины: `glDepthFunc(func)`
  - Значения `func`: `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`

Z-буфер

- Решение: буфер глубины (z-buffer)
- Хранит для каждого пикселя экрана расстояние до камеры
- Тест глубины: при рисовании очередного пикселя (после фрагментного шейдера) проверим: если уже нарисованный пиксель ближе к камере, то новый рисовать не будем
- Включить: `glEnable(GL_DEPTH_TEST)`
- Настроить поведение теста глубины: `glDepthFunc(func)`
  - Значения `func`: `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`
  - Например, `GL_LEQUAL` = less or equal: пиксель рисуется, если его расстояние до камеры меньше или равно расстояния, записанного в z-буфер

Z-буфер

- Решение: буфер глубины (z-buffer)
- Хранит для каждого пикселя экрана расстояние до камеры
- Тест глубины: при рисовании очередного пикселя (после фрагментного шейдера) проверим: если уже нарисованный пиксель ближе к камере, то новый рисовать не будем
- Включить: `glEnable(GL_DEPTH_TEST)`
- Настроить поведение теста глубины: `glDepthFunc(func)`
  - Значения `func`: `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`
  - Например, `GL_LEQUAL` = less or equal: пиксель рисуется, если его расстояние до камеры меньше или равно расстояния, записанного в z-буфер
  - По умолчанию `GL_LESS`

Z-буфер

- К координате Z после perspective divide применяется преобразование $[-1, 1] \mapsto [0, 1]$ (т.е. $z \mapsto \frac{z+1}{2}$), и это значение записывается в z-буфер

Z-буфер

- К координате Z после perspective divide применяется преобразование $[-1, 1] \mapsto [0, 1]$ (т.е. $z \mapsto \frac{z+1}{2}$), и это значение записывается в z-буфер
  - Доступно во фрагментном шейдере: `gl_FragDepth`

Z-буфер

- К координате Z после perspective divide применяется преобразование $[-1, 1] \mapsto [0, 1]$ (т.е. $z \mapsto \frac{z+1}{2}$), и это значение записывается в z-буфер
  - Доступно во фрагментном шейдере: `gl_FragDepth`
  - Интервал $[0, 1]$ можно заменить с помощью `glDepthRange`

Z-буфер

- К координате Z после perspective divide применяется преобразование $[-1, 1] \mapsto [0, 1]$ (т.е. $z \mapsto \frac{z+1}{2}$), и это значение записывается в z-буфер
  - Доступно во фрагментном шейдере: `gl_FragDepth`
  - Интервал $[0, 1]$ можно заменить с помощью `glDepthRange`
- Z-буфер хранит или в формате unsigned normalized (16/24/32-bit), или floating-point (32-bit)
  - Обычно unsigned normalized 24-bit

Z-буфер

- К координате Z после perspective divide применяется преобразование $[-1, 1] \mapsto [0, 1]$ (т.е. $z \mapsto \frac{z+1}{2}$), и это значение записывается в z-буфер
  - Доступно во фрагментном шейдере: `gl_FragDepth`
  - Интервал $[0, 1]$ можно заменить с помощью `glDepthRange`
- Z-буфер хранит или в формате unsigned normalized (16/24/32-bit), или floating-point (32-bit)
  - Обычно unsigned normalized 24-bit
- Нужно очищать перед каждым кадром: `glClear(GL_DEPTH_BUFFER_BIT)`

Z-буфер

- К координате Z после perspective divide применяется преобразование $[-1,1] \mapsto [0,1]$ (т.е. $z \mapsto \frac{z+1}{2}$), и это значение записывается в z-буфер
  - Доступно во фрагментном шейдере: `gl_FragDepth`
  - Интервал $[0,1]$ можно заменить с помощью `glDepthRange`
- Z-буфер хранит или в формате unsigned normalized (16/24/32-bit), или floating-point (32-bit)
  - Обычно unsigned normalized 24-bit
- Нужно очищать перед каждым кадром: `glClear(GL_DEPTH_BUFFER_BIT)`
- Default framebufer (тот, что используется по умолчанию, для рисования в окно) может иметь свой z-буфер – это нужно настраивать перед созданием OpenGL-контекста

Depth clipping/clamping

- Z-координата (после perspective divide) должна быть в диапазоне $[-1, 1]$

Depth clipping/clamping

- Z-координата (после perspective divide) должна быть в диапазоне $[-1, 1]$
- Что делать с вершинами, выпадающими за диапазон? Два варианта:

Depth clipping/clamping

- Z-координата (после perspective divide) должна быть в диапазоне $[-1, 1]$
- Что делать с вершинами, выпадающими за диапазон? Два варианта:
  - Depth clipping (отсечение по глубине): примитив (точка/линия/треугольник) обрезается по плоскостям $Z = \pm 1$ (как будто вне области $|Z| \leq 1$ ничего нет)

Depth clipping/clamping

- Z-координата (после perspective divide) должна быть в диапазоне $[-1, 1]$
- Что делать с вершинами, выпадающими за диапазон? Два варианта:
  - Depth clipping (отсечение по глубине): примитив (точка/линия/треугольник) обрезается по плоскостям $Z = \pm 1$ (как будто вне области $|Z| \leq 1$ ничего нет)
  - Depth clamping: отсечения не происходит, но после растеризации глубина всех пикселей, не попавших в диапазон $[-1, 1]$, сжимается до $[-1, 1]$

Depth clipping/clamping

- Z-координата (после perspective divide) должна быть в диапазоне $[-1, 1]$
- Что делать с вершинами, выпадающими за диапазон? Два варианта:
  - Depth clipping (отсечение по глубине): примитив (точка/линия/треугольник) обрезается по плоскостям $Z = \pm 1$ (как будто вне области $|Z| \leq 1$ ничего нет)
  - Depth clamping: отсечения не происходит, но после растеризации глубина всех пикселей, не попавших в диапазон $[-1, 1]$, сжимается до $[-1, 1]$
  - По умолчанию – depth clipping
  - Включить depth clamping: `glEnable(GL_DEPTH_CLAMP)`

Clipping

- На самом деле, с координатами X и Y тоже происходит отсечение

Clipping

- На самом деле, с координатами X и Y тоже происходит отсечение
- Как будто пиксели, не попадающие на экран, просто игнорируются

Clipping

- На самом деле, с координатами X и Y тоже происходит отсечение
- Как будто пиксели, не попадающие на экран, просто игнорируются
- Не настраивается

Clipping

- На самом деле, с координатами X и Y тоже происходит отсечение
- Как будто пиксели, не попадающие на экран, просто игнорируются
- Не настраивается

$$-1 \le X \le 1$$
$$-1 \le Y \le 1$$
$$-1 \le Z \le 1$$

Clipping

- На самом деле, с координатами X и Y тоже происходит отсечение
- Как будто пиксели, не попадающие на экран, просто игнорируются
- Не настраивается

$$
\begin{aligned}
-1 \le X \le 1 \\
-1 \le Y \le 1 \\
-1 \le Z \le 1
\end{aligned}
$$

- С учётом perspective divide (если $W > 0$):

$$
\begin{aligned}
-1 \le X/W \le 1 & \qquad -W \le X \le W \\
-1 \le Y/W \le 1 & \Leftrightarrow -W \le Y \le W \\
-1 \le Z/W \le 1 & \qquad -W \le Z \le W
\end{aligned}
$$

Clipping

- На самом деле, с координатами X и Y тоже происходит отсечение
- Как будто пиксели, не попадающие на экран, просто игнорируются
- Не настраивается

$$
\begin{aligned}
-1 \le X \le 1 \\
-1 \le Y \le 1 \\
-1 \le Z \le 1
\end{aligned}
$$

- С учётом perspective divide (если $W > 0$):

$$
\begin{aligned}
-1 \le X/W \le 1 & \qquad -W \le X \le W \\
-1 \le Y/W \le 1 & \Leftrightarrow -W \le Y \le W \\
-1 \le Z/W \le 1 & \qquad -W \le Z \le W
\end{aligned}
$$

- Если $W < 0$, точка не попадает на экран

Полный путь координат

- Входные данные: точка q_{object} в системе координат объекта (object space)

Полный путь координат

- Входные данные: точка q_{object} в системе координат объекта (object space)
- Позиционируем объект: $q_{world} = \text{Transform} \cdot q_{object}$ в мировых координатах (world space)

Полный путь координат

- Входные данные: точка q_{object} в системе координат объекта (object space)
- Позиционируем объект: $q_{world} = \text{Transform} \cdot q_{object}$ в мировых координатах (world space)
- Учитываем расположение камеры: $q_{camera} = \text{View} \cdot q_{world}$ в координатах камеры (view/camera space)

Полный путь координат

- Входные данные: точка q_{object} в системе координат объекта (object space)
- Позиционируем объект: $q_{world} = \text{Transform} \cdot q_{object}$ в мировых координатах (world space)
- Учитываем расположение камеры: $q_{camera} = \text{View} \cdot q_{world}$ в координатах камеры (view/camera space)
- Проекция камеры:
  `gl_Position =` $q_{clip} = \text{Projection} \cdot q_{camera}$ в clip space

Полный путь координат

- Входные данные: точка q_{object} в системе координат объекта (object space)
- Позиционируем объект: $q_{world} = \text{Transform} \cdot q_{object}$ в мировых координатах (world space)
- Учитываем расположение камеры: $q_{camera} = \text{View} \cdot q_{world}$ в координатах камеры (view/camera space)
- Проекция камеры:
  `gl_Position =` $q_{clip} = \text{Projection} \cdot q_{camera}$ в clip space
- Perspective divide: $q_{ndc} = xyz_{clip} / w_{clip}$ в normalized device coordinates

Полный путь координат

- Входные данные: точка q_{object} в системе координат объекта (object space)
- Позиционируем объект: $q_{world} = \text{Transform} \cdot q_{object}$ в мировых координатах (world space)
- Учитываем расположение камеры: $q_{camera} = \text{View} \cdot q_{world}$ в координатах камеры (view/camera space)
- Проекция камеры: `gl_Position` = $q_{clip} = \text{Projection} \cdot q_{camera}$ в clip space
- Perspective divide: $q_{ndc} = xyz_{clip}/w_{clip}$ в normalized device coordinates
  - x_{ndc} и y_{ndc} определяют координату пикселя на экране (через `glViewport`)
  - z_{ndc} определяет глубину

Перспективная проекция: near & far clip planes

- Если записывать в w координату z, то после perspective divide $z = 1$, т.е. у всех точек одинаковая глубина $\Rightarrow$ нам нужна другая формула проекции!

Перспективная проекция: near & far clip planes

- Если записывать в w координату z, то после perspective divide $z = 1$, т.е. у всех точек одинаковая глубина $\Rightarrow$ нам нужна другая формула проекции!
- Итоговый диапазон глубин ограничен $\Rightarrow$ нужно как-то ограничить диапазон возможных расстояний до камеры

Перспективная проекция: near & far clip planes

- Если записывать в w координату z, то после perspective divide $z = 1$, т.е. у всех точек одинаковая глубина $\Rightarrow$ нам нужна другая формула проекции!
- Итоговый диапазон глубин ограничен $\Rightarrow$ нужно как-то ограничить диапазон возможных расстояний до камеры
- $0 < near < far < \infty$

Перспективная проекция: near & far clip planes

- Если записывать в w координату z, то после perspective divide $z = 1$, т.е. у всех точек одинаковая глубина $\Rightarrow$ нам нужна другая формула проекции!
- Итоговый диапазон глубин ограничен $\Rightarrow$ нужно как-то ограничить диапазон возможных расстояний до камеры
- $0 < near < far < \infty$
- Хотим, чтобы диапазон $z \in [near, far]$ после применения матрицы проекции и perspective divide превратился в $[-1, 1]$

Перспективная проекция: near & far clip planes

- Хотим, чтобы диапазон $z \in [near, far]$ после применения матрицы проекции и perspective divide превратился в $[-1, 1]$

Перспективная проекция: near & far clip planes

- Хотим, чтобы диапазон $z \in [near, far]$ после применения матрицы проекции и perspective divide превратился в $[-1, 1]$

$$z \mapsto \frac{Az+B}{z}$$

$$near \mapsto \frac{A \cdot near+B}{near} = -1$$

$$far \mapsto \frac{A \cdot far+B}{far} = 1$$

Перспективная проекция: near & far clip planes

- Хотим, чтобы диапазон $z \in [near, far]$ после применения матрицы проекции и perspective divide превратился в $[-1, 1]$

$$z \mapsto \frac{Az+B}{z}$$

$$near \mapsto \frac{A \cdot near + B}{near} = -1$$

$$far \mapsto \frac{A \cdot far + B}{far} = 1$$

- Линейная система на A и B:

$$\begin{matrix} A \cdot near + B = -near \\ A \cdot far + B = far \end{matrix} \Rightarrow \begin{matrix} A = \frac{far+near}{far-near} \\ B = \frac{-2 far \cdot near}{far-near} \end{matrix}$$

Перспективная проекция: матрица проекции

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{far+near}{far-near} & -\frac{2far \cdot near}{far-near} \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Перспективная проекция: правая система координат

- Обычно направление вгзляда камеры выбирают не в сторону положительной оси Z, а в сторону отрицательной оси Z, чтобы получилась правая система координат

Перспективная проекция: правая система координат

- Обычно направление вгзляда камеры выбирают не в сторону положительной оси Z, а в сторону отрицательной оси Z, чтобы получилась правая система координат
- Отсечение по $z = -near$ и $z = -far$

Перспективная проекция: правая система координат

- Обычно направление вгзляда камеры выбирают не в сторону положительной оси Z, а в сторону отрицательной оси Z, чтобы получилась правая система координат
- Отсечение по $z = -near$ и $z = -far$
- Расстояние до камеры – не Z, а -Z

Перспективная проекция: правая система координат

- Обычно направление вгзляда камеры выбирают не в сторону положительной оси Z, а в сторону отрицательной оси Z, чтобы получилась правая система координат
- Отсечение по $z = -near$ и $z = -far$
- Расстояние до камеры – не Z, а -Z

$$\frac{A(-near)+B}{near} = -1$$

$$\frac{A(-far)+B}{far} = 1$$

Перспективная проекция: правая система координат

- Обычно направление вгзляда камеры выбирают не в сторону положительной оси Z, а в сторону отрицательной оси Z, чтобы получилась правая система координат
- Отсечение по $z = -near$ и $z = -far$
- Расстояние до камеры – не Z, а -Z

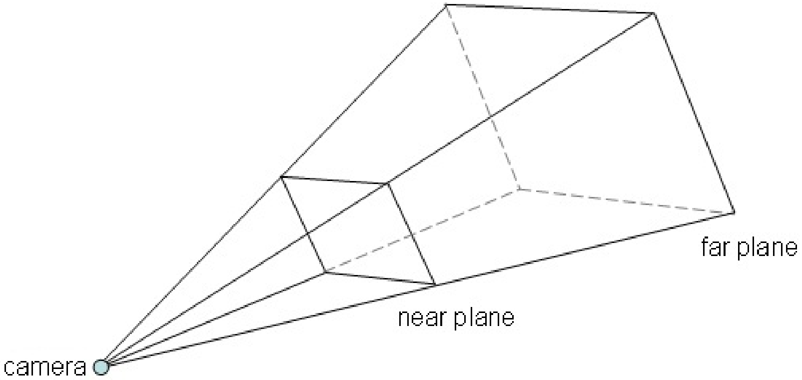
$$\frac{A(-near)+B}{near} = -1$$

$$\frac{A(-far)+B}{far} = 1$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far \cdot near}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Перспективная проекция: view frustum

Область, видимая перспективной камерой - усечённая пирамида (frustum)



Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

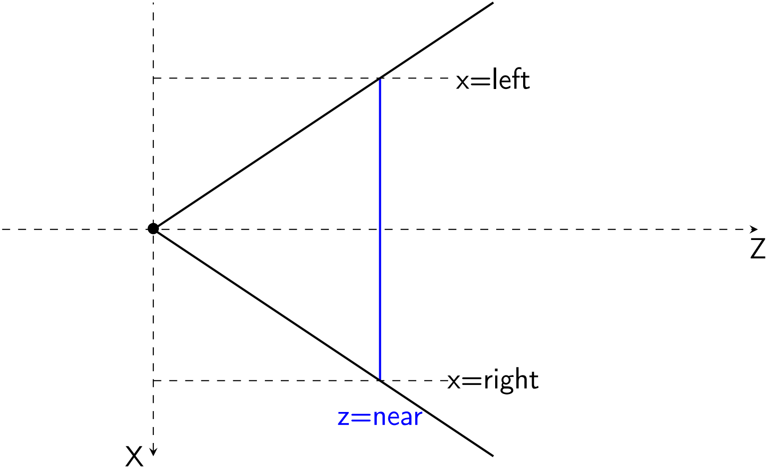
Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

$$\begin{pmatrix} C & 0 & D & 0 \\ 0 & E & F & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?



Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

- Обычно left = -right

Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

- Обычно left = -right
- $\frac{-left}{near}$ – тангенс угла между осью (Z) проекции и левой границей видимой области
- $\frac{right}{near}$ – тангенс угла между осью (Z) проекции и правой границей видимой области

Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

- Обычно left = -right
- $\frac{-left}{near}$ – тангенс угла между осью (Z) проекции и левой границей видимой области
- $\frac{right}{near}$ – тангенс угла между осью (Z) проекции и правой границей видимой области
- Аналогично top и bottom для Y

Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

- Обычно left = -right
- $\frac{-left}{near}$ – тангенс угла между осью (Z) проекции и левой границей видимой области
- $\frac{right}{near}$ – тангенс угла между осью (Z) проекции и правой границей видимой области
- Аналогично top и bottom для Y
- Хотим, чтобы точка с координатами X = left и Z = -near перешла (после применения матрица проекции и perspective divide) в точку с X = -1 и Z = -1
- Хотим, чтобы точка с координатами X = right и Z = -near перешла (после применения матрица проекции и perspective divide) в точку с X = 1 и Z = -1

Перспективная проекция: view frustum

$$\left.\begin{aligned} \frac{C \cdot \mathit{left} + D(-\mathit{near})}{\mathit{near}} &= -1 \\ \frac{C \cdot \mathit{right} + D(-\mathit{near})}{\mathit{near}} &= 1 \end{aligned}\right. \Rightarrow \begin{aligned} C &= \frac{2 \cdot \mathit{near}}{\mathit{right} - \mathit{left}} \\ D &= \frac{\mathit{right} + \mathit{left}}{\mathit{right} - \mathit{left}} \end{aligned}$$

Перспективная проекция: view frustum

$$\left.\begin{array}{l} \frac{C \cdot left + D(-near)}{near} = -1 \\ \\ \frac{C \cdot right + D(-near)}{near} = 1 \end{array}\right. \Rightarrow \begin{array}{l} C = \frac{2 \cdot near}{right - left} \\ \\ D = \frac{right + left}{right - left} \end{array}$$

Аналогично для Y

$$\left.\begin{array}{l} \frac{E \cdot bottom + F(-near)}{near} = -1 \\ \\ \frac{E \cdot top + F(-near)}{near} = 1 \end{array}\right. \Rightarrow \begin{array}{l} E = \frac{2 \cdot near}{top - bottom} \\ \\ F = \frac{top + bottom}{top - bottom} \end{array}$$

Перспективная проекция: матрица проекции

$$\begin{pmatrix} \frac{2 \cdot near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2 \cdot near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far \cdot near}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Перспективная проекция: матрица проекции

$$\begin{pmatrix} \frac{2\cdot near}{right-left} & 0 & \frac{right+left}{right-left} & 0 \\ 0 & \frac{2\cdot near}{top-bottom} & \frac{top+bottom}{top-bottom} & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far\cdot near}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Часто left=-right и bottom=-top, тогда

$$\begin{pmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far+near}{far-near} & -\frac{2far\cdot near}{far-near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

- Это самый часто встречающийся вид матрицы перспективной проекции

Перспективная проекция: матрица проекции

$$\begin{pmatrix} \frac{2 \cdot near}{right - left} & 0 & \frac{right + left}{right - left} & 0 \\ 0 & \frac{2 \cdot near}{top - bottom} & \frac{top + bottom}{top - bottom} & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2 far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

Часто left=-right и bottom=-top, тогда

$$\begin{pmatrix} \frac{near}{right} & 0 & 0 & 0 \\ 0 & \frac{near}{top} & 0 & 0 \\ 0 & 0 & -\frac{far + near}{far - near} & -\frac{2 far \cdot near}{far - near} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

- Это самый часто встречающийся вид матрицы перспективной проекции
- Эту матрицы мы передаём в шейдер и применяем к входным вершинам

Перспективная проекция

- Как выбирать параметры камеры?

Перспективная проекция

- Как выбирать параметры камеры?
- Пусть хотим камеру с шириной обзора по X равной θ радиан, и с aspect ratio (отношение ширины к высоте) равным R

Перспективная проекция

- Как выбирать параметры камеры?
- Пусть хотим камеру с шириной обзора по X равной θ радиан, и с aspect ratio (отношение ширины к высоте) равным R
- $-left = right = near \cdot \tan\left(\frac{\theta}{2}\right)$
- $-bottom = top = \frac{1}{R} \cdot right = \frac{near}{R} \cdot \tan\left(\frac{\theta}{2}\right)$

Перспективная проекция

- Как выбирать параметры near и far?

Перспективная проекция

- Как выбирать параметры near и far?
- Всё ближе near или дальше far обрезается - почему не взять near≈ 0 и far$=\infty$?

Перспективная проекция

- Как выбирать параметры near и far?
- Всё ближе near или дальше far обрезается - почему не взять near≈ 0 и far$=\infty$?
- Если near $= 0$, третья строка матрицы проекции вырождается в $\begin{pmatrix} 0 & 0 & -1 & 0 \end{pmatrix}$, и у всех точек будет одинаковая глубина, т.е. не получится использовать z-буфер

Перспективная проекция

- Как выбирать параметры near и far?
- Всё ближе near или дальше far обрезается - почему не взять near≈ 0 и far$=\infty$?
- Если near $= 0$, третья строка матрицы проекции вырождается в $\begin{pmatrix} 0 & 0 & -1 & 0 \end{pmatrix}$, и у всех точек будет одинаковая глубина, т.е. не получится использовать z-буфер
- Если far $\to \infty$, третья строка стремится к $\begin{pmatrix} 0 & 0 & -1 & -2near \end{pmatrix}$

Перспективная проекция

- Z-буфер имеет ограниченную точность

Перспективная проекция

- Z-буфер имеет ограниченную точность
- Чем больше отношение $\frac{far}{near}$, тем меньше точности приходится на единицу расстояния

Перспективная проекция

- Z-буфер имеет ограниченную точность
- Чем больше отношение $\frac{far}{near}$, тем меньше точности приходится на единицу расстояния
- Как выбирать параметры near и far?

Перспективная проекция

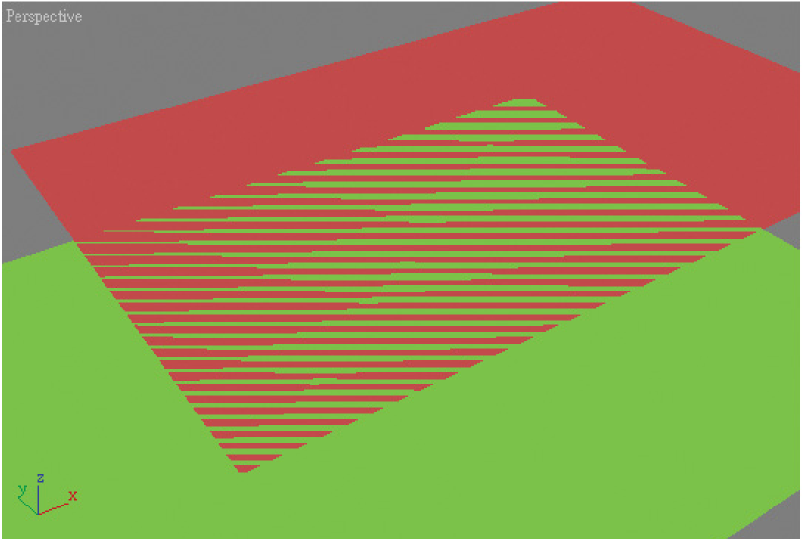
- Z-буфер имеет ограниченную точность
- Чем больше отношение $\frac{far}{near}$, тем меньше точности приходится на единицу расстояния
- Как выбирать параметры near и far?
- near: максимально возможный, при котором не обрезается что-то существенное
  - При этом камера искуственно держится на некотором расстоянии от любых объектов

Перспективная проекция

- Z-буфер имеет ограниченную точность
- Чем больше отношение $\frac{far}{near}$, тем меньше точности приходится на единицу расстояния
- Как выбирать параметры near и far?
- near: максимально возможный, при котором не обрезается что-то существенное
  - При этом камера искуственно держится на некотором расстоянии от любых объектов
- far: максимально возможный, при котором не возникает Z-fighting'а

Z-fighting

Ситуация, когда две очень близких почти (или полностью) параллельных плоскости рисуются с артефактом из-за недостаточной точности Z-буфера:



Перспективная проекция: матрица проекции

- Что, если мы хотим переместить и повернуть камеру?

Перспективная проекция: матрица проекции

- Что, если мы хотим переместить и повернуть камеру?
- Так же, как с ортографической проекцией: сначала применим обратное преобразование, а потом матрицу проекции

Перспективная проекция: матрица проекции

- Что, если мы хотим переместить и повернуть камеру?
- Так же, как с ортографической проекцией: сначала применим обратное преобразование, а потом матрицу проекции

$$\underbrace{\begin{pmatrix} C & 0 & D & 0 \\ 0 & E & F & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix}}_{\text{Projection}} \cdot \underbrace{\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}}_{\text{View}} \tag{1}$$

Projection $\cdot$ View

Перспективная проекция: матрица проекции

- Что, если мы хотим переместить и повернуть камеру?
- Так же, как с ортографической проекцией: сначала применим обратное преобразование, а потом матрицу проекции

$$\underbrace{\begin{pmatrix} C & 0 & D & 0 \\ 0 & E & F & 0 \\ 0 & 0 & A & B \\ 0 & 0 & -1 & 0 \end{pmatrix}}_{\text{Projection}} \cdot \underbrace{\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1}}_{\text{View}} \tag{1}$$

Projection · View

- Обычно именно эти матрицы (или их произведение) передают в шейдер

Проекции: ссылки

- songho.ca/opengl/gl_transform.html
- songho.ca/opengl/gl_projectionmatrix.html

Бонус: как перевести их экранных координат в мировые?

- Есть точка на экране с координатами $(P_X, P_Y) \in [-1, 1]^2$ (например, координаты указателя мыши, переведённые из пиксельных координат в OpenGL-ные) и матрица камеры $T = \text{Projection} \cdot \text{View}$

Бонус: как перевести их экранных координат в мировые?

- Есть точка на экране с координатами $(P_X, P_Y) \in [-1, 1]^2$ (например, координаты указателя мыши, переведённые из пиксельных координат в OpenGL-ные) и матрица камеры $T = \mathrm{Projection} \cdot \mathrm{View}$
- В пространстве ей соответствует луч из камеры – как его найти?

Бонус: как перевести их экранных координат в мировые?

- Есть точка на экране с координатами $(P_X, P_Y) \in [-1, 1]^2$ (например, координаты указателя мыши, переведённые из пиксельных координат в OpenGL-ные) и матрица камеры $T = \text{Projection} \cdot \text{View}$
- В пространстве ей соответствует луч из камеры – как его найти?
- Найдём две точки, пересекающие этот луч, на near и far плоскостях

Бонус: как перевести их экранных координат в мировые?

- Есть точка на экране с координатами $(P_X, P_Y) \in [-1, 1]^2$ (например, координаты указателя мыши, переведённые из пиксельных координат в OpenGL-ные) и матрица камеры $T = \mathrm{Projection} \cdot \mathrm{View}$
- В пространстве ей соответствует луч из камеры – как его найти?
- Найдём две точки, пересекающие этот луч, на near и far плоскостях
- Пусть Proj – оператор, выполняющий perspective divide:

$$\mathrm{Proj}\begin{pmatrix} x \\ y \\ z \\ w \end{pmatrix} = \begin{pmatrix} x/w \\ y/w \\ z/w \end{pmatrix}$$

Бонус: как перевести их экранных координат в мировые?

- Мы хотим решить уравнение

$$\text{Proj} \cdot T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \end{pmatrix}$$

Бонус: как перевести их экранных координат в мировые?

- Мы хотим решить уравнение
$$\mathrm{Proj} \cdot T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \end{pmatrix}$$
- Как выглядит множество $\mathrm{Proj}^{-1} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$?

Бонус: как перевести их экранных координат в мировые?

- Мы хотим решить уравнение
$$\mathrm{Proj} \cdot T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \end{pmatrix}$$
- Как выглядит множество $\mathrm{Proj}^{-1} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda \end{pmatrix} \forall \lambda \neq 0$

Бонус: как перевести их экранных координат в мировые?

- Мы хотим решить уравнение

$$\mathrm{Proj} \cdot T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \end{pmatrix}$$

- Как выглядит множество $\mathrm{Proj}^{-1} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \lambda x \\ \lambda y \\ \lambda z \\ \lambda \end{pmatrix} \forall \lambda \neq 0$

$$\Rightarrow T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \lambda P_X \\ \lambda P_Y \\ \pm \lambda \\ \lambda \end{pmatrix}$$, но мы не знаем λ

Бонус: как перевести их экранных координат в мировые?

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = T^{-1} \begin{pmatrix} \lambda P_X \\ \lambda P_Y \\ \pm\lambda \\ \lambda \end{pmatrix} = \lambda T^{-1} \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}$$

Бонус: как перевести их экранных координат в мировые?

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = T^{-1} \begin{pmatrix} \lambda P_X \\ \lambda P_Y \\ \pm\lambda \\ \lambda \end{pmatrix} = \lambda T^{-1} \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}$$

- С чего мы взяли, что последняя координата w будет равна 1?

Бонус: как перевести их экранных координат в мировые?

$$\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = T^{-1} \begin{pmatrix} \lambda P_X \\ \lambda P_Y \\ \pm\lambda \\ \lambda \end{pmatrix} = \lambda T^{-1} \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}$$

- С чего мы взяли, что последняя координата w будет равна 1? Нужно подобрать λ:

$$1 = \lambda \cdot \left[T^{-1} \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix} \right]_w$$

Бонус: как перевести их экранных координат в мировые?

$$\lambda = \frac{1}{\left[T^{-1}\begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}\right]_w}$$

Бонус: как перевести их экранных координат в мировые?

$$\lambda = \frac{1}{\left[T^{-1}\begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}\right]_w}$$

Вектор $T^{-1}\begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}$ нужно разделить на последнюю координату!

Бонус: как перевести их экранных координат в мировые?

$$\lambda = \frac{1}{\left[T^{-1} \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix} \right]_w}$$

Вектор $T^{-1} \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}$ нужно разделить на последнюю координату!

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \text{Proj } T^{-1} \begin{pmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{pmatrix}$$

Бонус: как найти координаты вершин видимой области (view frustum)?

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \text{Proj}\ T^{-1} \begin{pmatrix} \pm 1 \\ \pm 1 \\ \pm 1 \\ 1 \end{pmatrix}$$