

# Компьютерная графика

## Лекция 4: Камера, 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 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

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

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

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  $[-1, 1]$
- ▶ Как сделать  $X \in [-W, W]$  и  $Y \in [-H, H]$ ?

# Ортографическая проекция

- ▶ Самый простой способ: игнорировать третью координату

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

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

- ▶ Именно это делает OpenGL (если `gl_Position.w = 1`)
- ▶  $X$  и  $Y$  всё ещё  $[-1, 1]$
- ▶ Как сделать  $X \in [-W, W]$  и  $Y \in [-H, H]$ ?

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

# Ортографическая проекция

- ▶ Как сделать  $X \in [X_0 - W, X_0 + W]$  и  $Y \in [Y_0 - H, Y_0 + H]$ ?

# Ортографическая проекция

- ▶ Как сделать  $X \in [X_0 - W, X_0 + W]$  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

- ▶ Как сделать  $X \in [X_0 - W, X_0 + W]$  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

- ▶ Как сделать  $X \in [X_0 - W, X_0 + W]$  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



# 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
- ▶ 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
- ▶ 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

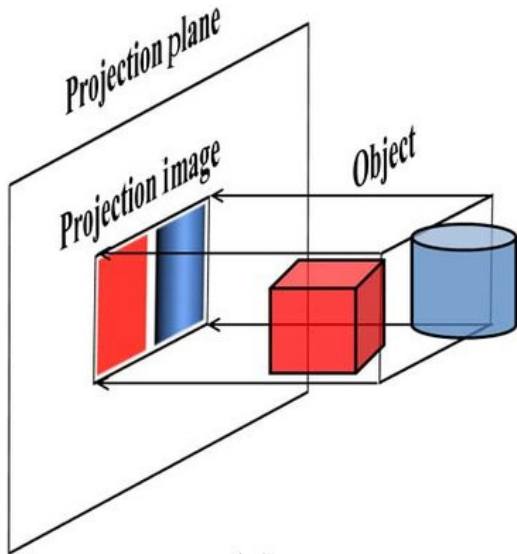


Fig. 1.23

# 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 (проекция) - 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 перенос, масштабирование и поворот

$$\begin{pmatrix} X_x & Y_x & Z_x & C_x \\ X_y & Y_y & Z_y & C_y \\ X_z & Y_z & Z_z & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & C_x \\ 0 & 1 & 0 & C_y \\ 0 & 0 & 1 & C_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} W & 0 & 0 & 0 \\ 0 & H & 0 & 0 \\ 0 & 0 & D & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

## 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: 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

# Ортографическая проекция: применение

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

# Ортографическая проекция: применение

- ▶ 2D рендеринг: 2D игры, UI, карты
- ▶ Проектирование моделей, зданий, деталей (вид сверху, вид сбоку, вид спереди)
- ▶ Стилизация: 3D мир с ортографической проекцией

# Ортографическая проекция: проблемы

# Ортографическая проекция: проблемы

- ▶ Объекты на разном расстоянии от камеры выглядят одинаково

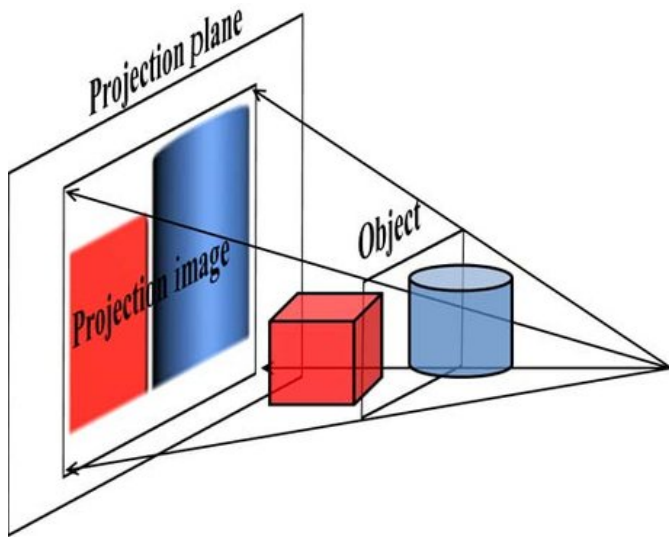
# Ортографическая проекция: проблемы

- ▶ 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



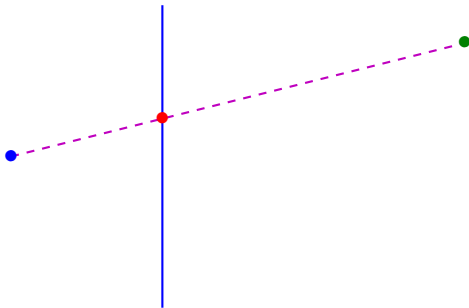


# Перспективная проекция

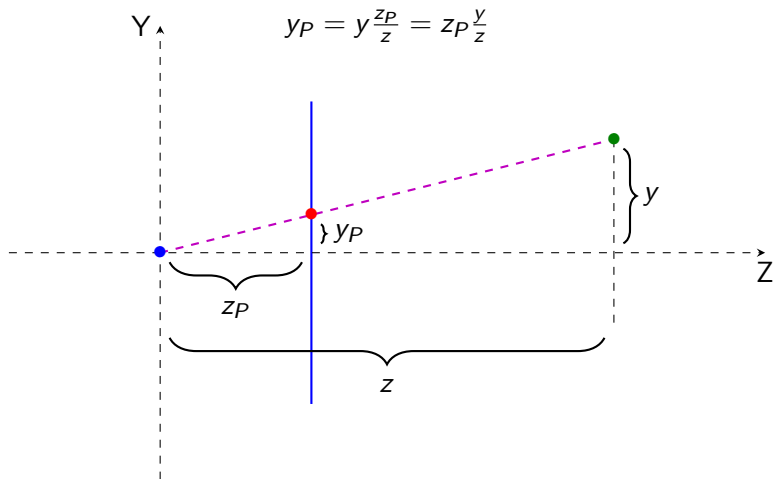
- ▶ Есть центр проекции и плоскость проекции
- ▶ Проекция точки - пересечение прямой, проходящей через эту точку и центр проекции, с плоскостью проекции

# 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 проекции  $Z = z_P$ , нужно разделить на  $Z$ -координату:

$$(x, y, z) \mapsto (z_P \frac{x}{z}, z_P \frac{y}{z})$$

# Перспективная проекция

- ▶ Чтобы вычислить перспективную проекцию с центром в начале координат и плоскостью проекции  $Z = z_P$ , нужно разделить на  $Z$ -координату:

$$(x, y, z) \mapsto (z_P \frac{x}{z}, z_P \frac{y}{z})$$

- ▶ Как выразить эту проекцию матрицей?

# Перспективная проекция

- ▶ Чтобы вычислить перспективную проекцию с центром в начале координат и плоскостью проекции  $Z = z_P$ , нужно разделить на  $Z$ -координату:

$$(x, y, z) \mapsto (z_P \frac{x}{z}, z_P \frac{y}{z})$$

- ▶ 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 (ортогографическая 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, рисуемые позже, накладываются поверх уже нарисованных

# Наложение объектов

- ▶ По умолчанию примитивы, рисуемые 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(GL_LEQUAL)`
  - ▶ `LEQUAL` = less or equal: пиксель рисуется, если его расстояние до камеры меньше или равно расстояния, записанного 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

# 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`
  - ▶ Преобразование можно настроить 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`
  - ▶ Преобразование можно настроить 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`
  - ▶ Преобразование можно настроить 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`
  - ▶ Преобразование можно настроить 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 framebuffer (тот, что используется по умолчанию, для рисования в окно) может иметь свой z-буфер - это нужно настраивать перед созданием 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 сжимается до диапазона  $[-1, 1]$

# 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 сжимается до диапазона  $[-1, 1]$
  - ▶ 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

$$-1 \leq X \leq 1$$

$$-1 \leq Y \leq 1$$

$$-1 \leq Z \leq 1$$

# Clipping

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

$$-1 \leq X \leq 1$$

$$-1 \leq Y \leq 1$$

$$-1 \leq Z \leq 1$$

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

$$-1 \leq X/W \leq 1 \quad -W \leq X \leq W$$

$$-1 \leq Y/W \leq 1 \Leftrightarrow -W \leq Y \leq W$$

$$-1 \leq Z/W \leq 1 \quad -W \leq Z \leq W$$

# Clipping

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

$$-1 \leq X \leq 1$$

$$-1 \leq Y \leq 1$$

$$-1 \leq Z \leq 1$$

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

$$-1 \leq X/W \leq 1 \quad -W \leq X \leq W$$

$$-1 \leq Y/W \leq 1 \Leftrightarrow -W \leq Y \leq W$$

$$-1 \leq Z/W \leq 1 \quad -W \leq Z \leq W$$

- ▶ 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 на экране
  - ▶  $z_{ndc}$  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:

$$A \cdot near + B = -near \quad A = \frac{far + near}{far - near}$$

$\Rightarrow$

$$A \cdot far + B = far \quad B = \frac{-2far \cdot near}{far - near}$$

## 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 взгляда 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 взгляда 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 взгляда 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 взгляда 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 взгляда 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  $-Z$

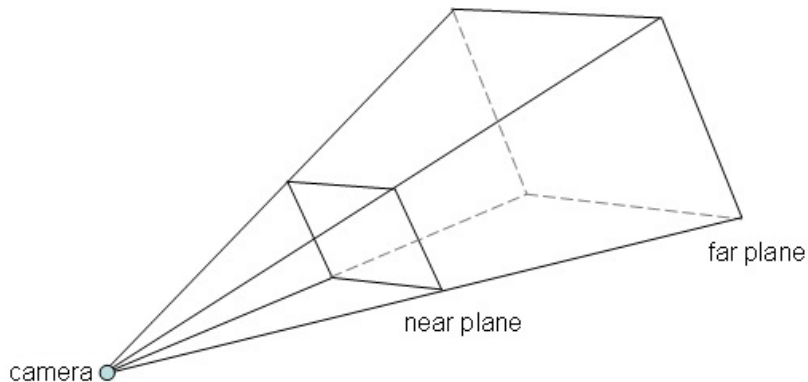
$$\frac{A(-near)+B}{near} = -1$$

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

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

# 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?

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

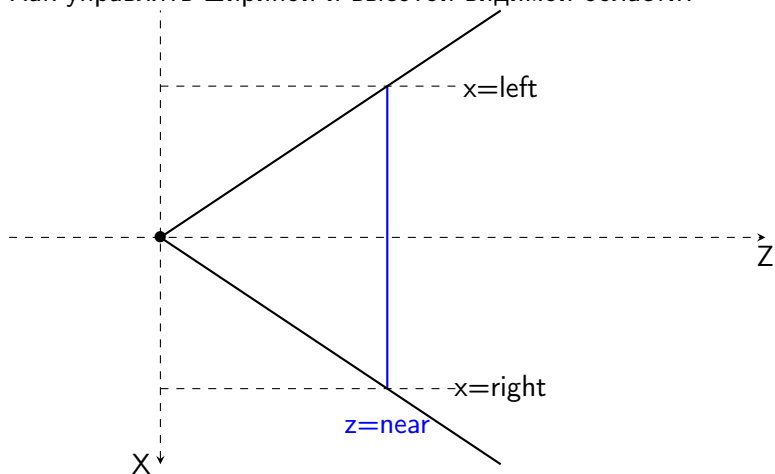
## Перспективная проекция: view frustum

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  $\text{left} = -\text{right}$

# Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

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

# Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

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

## Перспективная проекция: view frustum

Как управлять шириной и высотой видимой области?

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



## Перспективная проекция: view frustum

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

## Перспективная проекция: view frustum

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

Аналогично для Y

$$\begin{aligned}\frac{E \cdot \text{bottom} + F(-\text{near})}{\text{near}} &= -1 & E &= \frac{2 \cdot \text{near}}{\text{top} - \text{bottom}} \\ \Rightarrow \\ \frac{E \cdot \text{top} + F(-\text{near})}{\text{near}} &= 1 & F &= \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}}\end{aligned}$$

## Перспективная проекция: матрица проекции

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

## Перспективная проекция: матрица проекции

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

Часто  $\text{left} = -\text{right}$  и  $\text{bottom} = -\text{top}$ , тогда

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

Это самый часто встречающийся вид матрицы перспективной проекции

# Перспективная проекция

- ▶ Как выбирать параметры камеры?

# Перспективная проекция

- ▶ Как выбирать параметры камеры?
- ▶ Пусть хотим камеру с шириной обзора по  $X$  равной  $\theta$  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  $\theta$  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  $\text{near} \approx 0$  и  $\text{far} = \infty$ ?

# Перспективная проекция

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

# 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 искусственно держится на некотором расстоянии от любых объектов

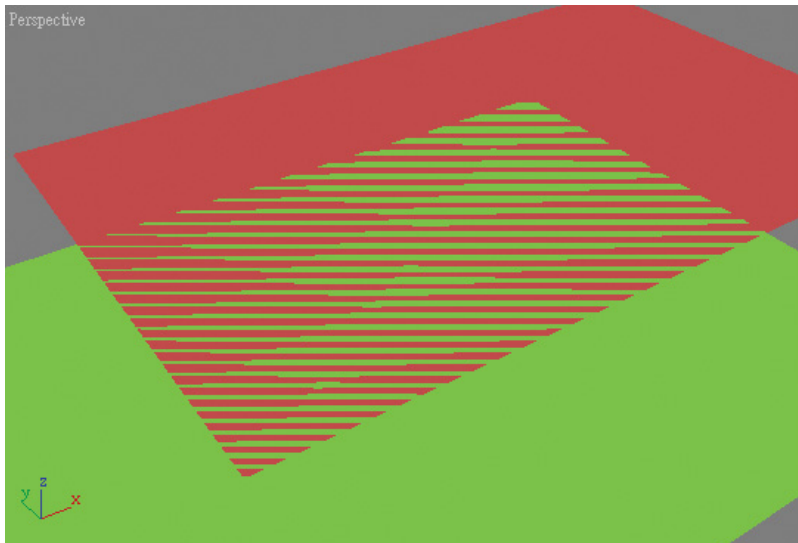
# Перспективная проекция

- ▶ 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 искусственно держится на некотором расстоянии от любых объектов
- ▶ far: максимально возможный, при котором не возникает Z-fighting'a



## 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

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

## 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

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

- ▶ Обычно именно эту матрицу передают в шейдер

## Проекции: ссылки

- ▶ [songho.ca/opengl/gl\\_transform.html](http://songho.ca/opengl/gl_transform.html)
- ▶ [songho.ca/opengl/gl\\_projectionmatrix.html](http://songho.ca/opengl/gl_projectionmatrix.html)