

Компьютерная графика

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

2025

Проекции

- В большинстве ситуаций виртуальный мир – двухмерный/трёхмерный, произвольного размера, наблюдаемый с произвольного ракурса

Проекции

- В большинстве ситуаций виртуальный мир – двухмерный/трёхмерный, произвольного размера, наблюдаемый с произвольного ракурса
- Экран – двухмерный, в системе координат OpenGL $[-1, 1] \times [-1, 1]$

Проекции

- В большинстве ситуаций виртуальный мир – двухмерный/трёхмерный, произвольного размера, наблюдаемый с произвольного ракурса
- Экран – двухмерный, в системе координат OpenGL $[-1, 1] \times [-1, 1]$
 - Координата Z тоже $[-1, 1]$
 - Про неё поговорим подробнее чуть позже

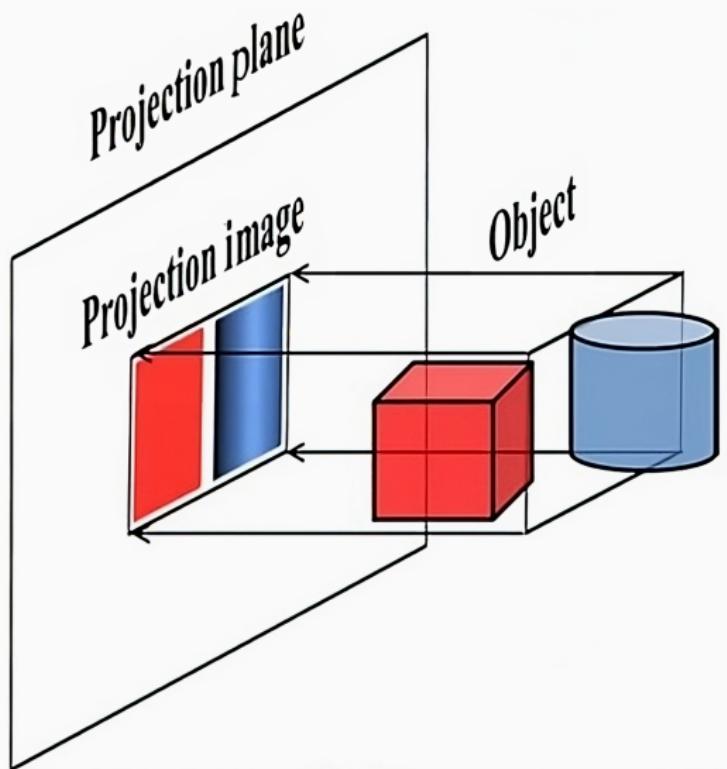
Проекции

- В большинстве ситуаций виртуальный мир – двухмерный/трёхмерный, произвольного размера, наблюдаемый с произвольного ракурса
- Экран – двухмерный, в системе координат OpenGL $[-1, 1] \times [-1, 1]$
 - Координата Z тоже $[-1, 1]$
 - Про неё поговорим подробнее чуть позже
- \Rightarrow Нам нужно преобразование из первого во второе, т.н. проекция

Проекции

- В большинстве ситуаций виртуальный мир – двухмерный/трёхмерный, произвольного размера, наблюдаемый с произвольного ракурса
- Экран – двухмерный, в системе координат OpenGL $[-1, 1] \times [-1, 1]$
 - Координата Z тоже $[-1, 1]$
 - Про неё поговорим подробнее чуть позже
- \Rightarrow Нам нужно преобразование из первого во второе, т.н. проекция
- Два основных вида проекции:
 - Ортографическая проекция
 - Перспективная проекция

Ортографическая проекция



Ортографическая проекция

- Самый простой способ проекции:

Ортографическая проекция

- Самый простой способ проекции: 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 проекции: 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 проекции: 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 проекции: 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 $[-1, 1]$
- 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 проекции: 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 $[-1, 1]$
- 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

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

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: 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 $C \pm X \pm Y$, который 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 $C \pm X \pm Y$, который отождествляется с экраном
- Обычно X, Y, Z взаимно ортогональны

Ортографическая проекция: общий случай

- Как выразить эту проекцию матрицей?

Ортографическая проекция: общий случай

- 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

- 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, применяем к входным вершинам, и записываем результат в `gl_Position`

Ортографическая проекция: ортогональный случай

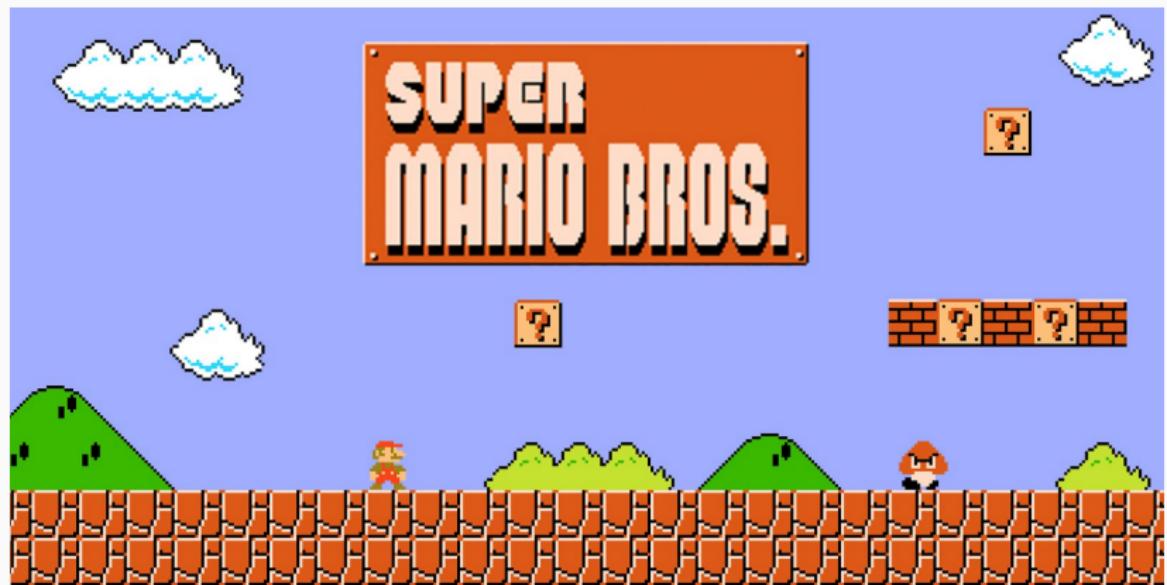
- Если X, Y, Z ортогональны, то

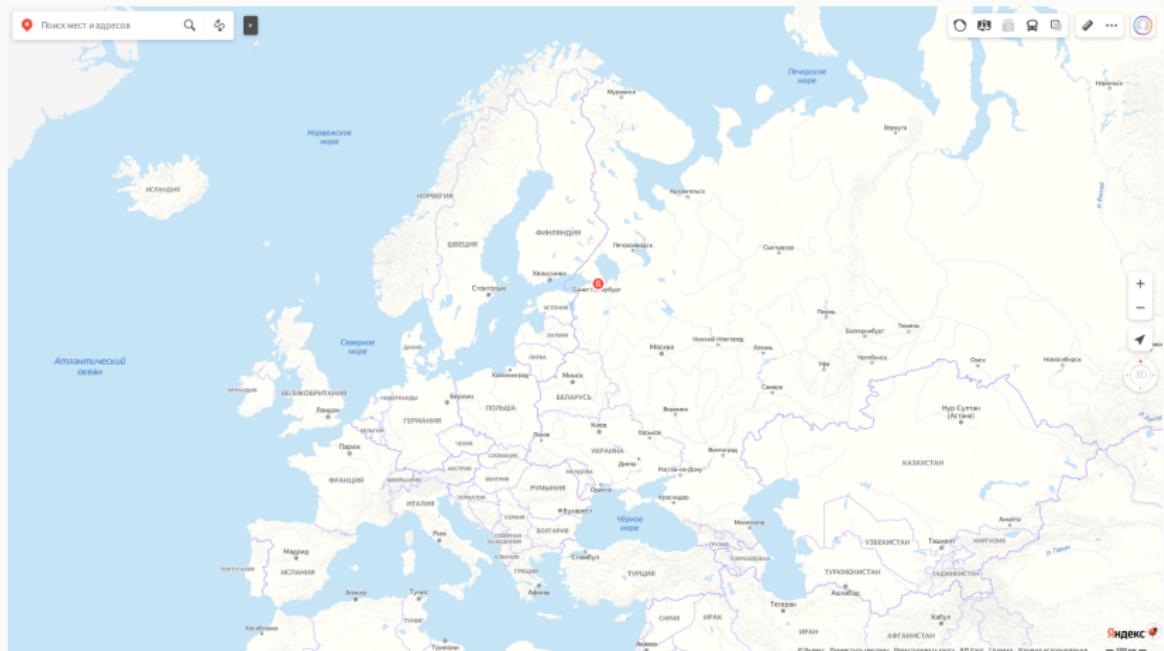
$$\begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \hat{X}_x & \hat{Y}_x & \hat{Z}_x & 0 \\ \hat{X}_y & \hat{Y}_y & \hat{Z}_y & 0 \\ \hat{X}_z & \hat{Y}_z & \hat{Z}_z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

Ортографическая проекция: применение

Ортографическая проекция: применение

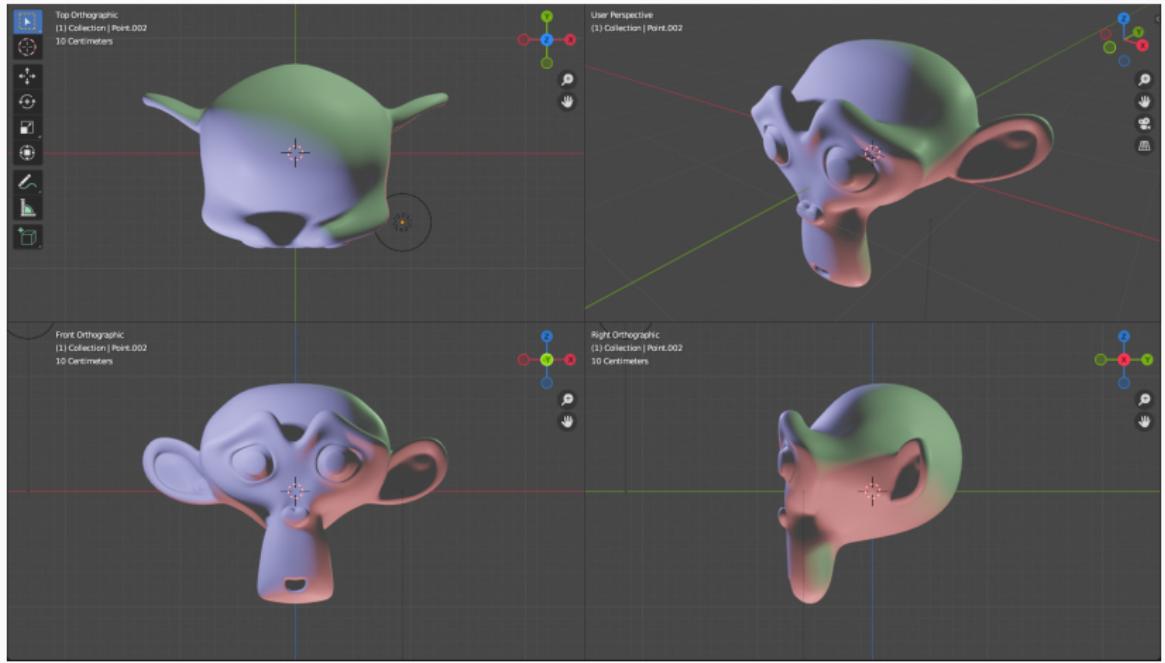
- 2D рендеринг: 2D игры, UI, карты





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 (напр. изометрическая проекция)





Ортографическая проекция: проблемы

Ортографическая проекция: проблемы

- Объекты на разном расстоянии от камеры выглядят одинаково

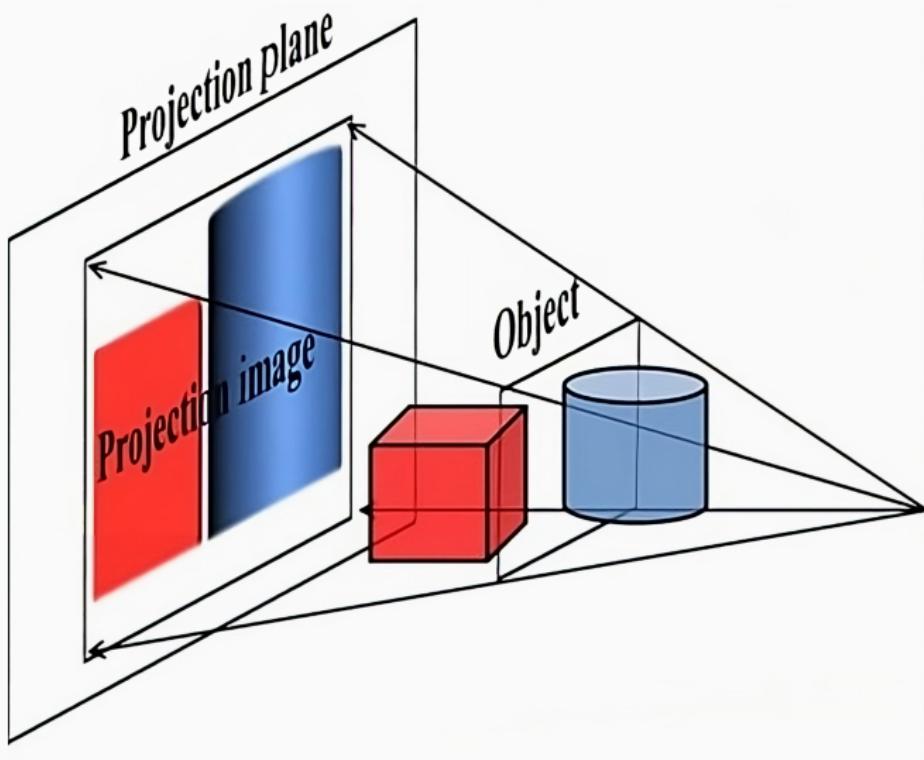
Ортографическая проекция: проблемы

- 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



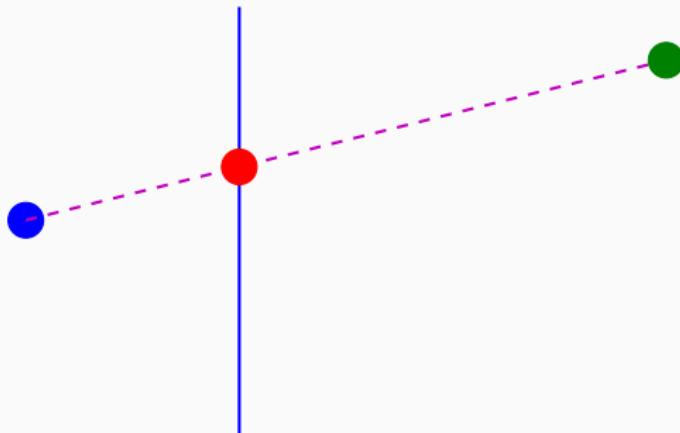
Перспективная проекция

- Есть **центр проекции** и **плоскость проекции**
-

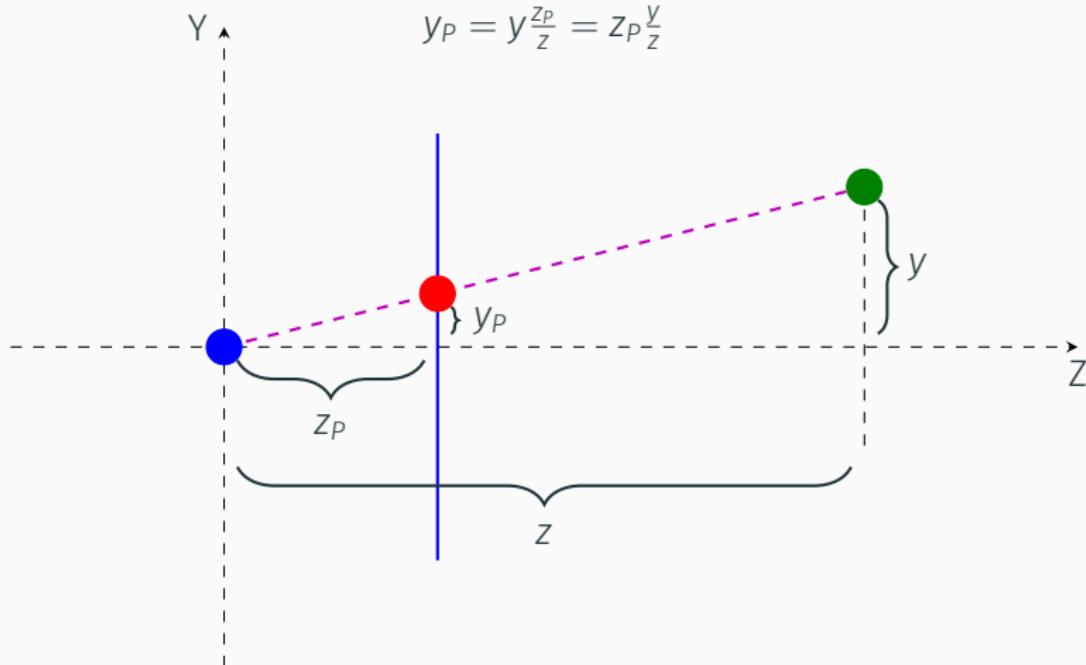


Перспективная проекция

- 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 позволяют делить одни координаты на другие (это нелинейная операция)

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** из-за буфера глубины

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

- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это плохое решение:

Наложение объектов

- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это плохое решение:
 - Нужно отсортировать миллионы треугольников каждый кадр и загрузить их на GPU \implies хуже производительность

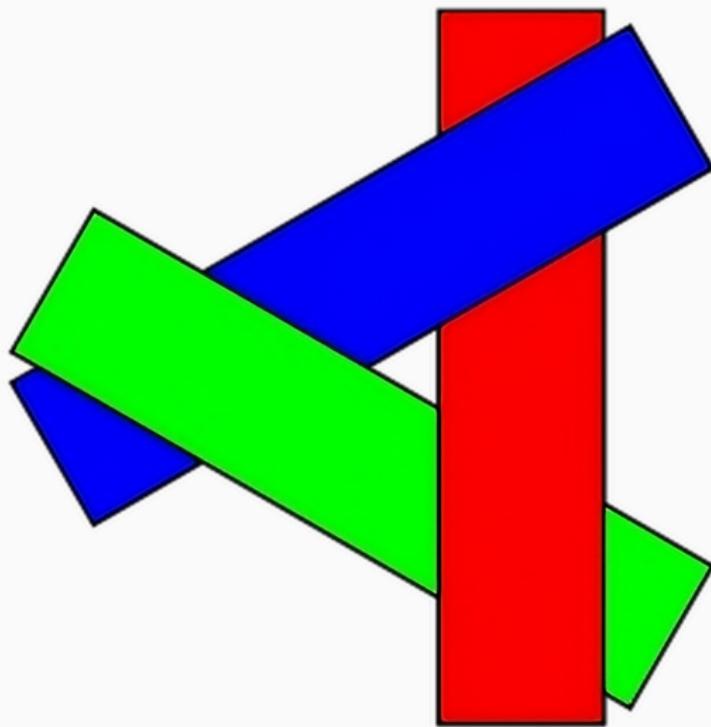
Наложение объектов

- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это плохое решение:
 - Нужно отсортировать миллионы треугольников каждый кадр и загрузить их на GPU \Rightarrow хуже производительность
 - Треугольники одного объекта не идут непрерывно \Rightarrow больше изменений состояния (переключений шейдеров, etc) при рендеринге \Rightarrow хуже производительность

Наложение объектов

- В 3D можно сортировать треугольники в порядке убывания расстояния до камеры (painter's algorithm), но это плохое решение:
 - Нужно отсортировать миллионы треугольников каждый кадр и загрузить их на GPU \Rightarrow хуже производительность
 - Треугольники одного объекта не идут непрерывно \Rightarrow больше изменений состояния (переключений шейдеров, etc) при рендеринге \Rightarrow хуже производительность
 - Всё равно решает проблему не во всех случаях (напр. когда в графе наложений объектов есть циклы, или треугольники пересекаются)

Случай, когда painter's algorithm не работает



Z-буфер

- Решение: буфер глубины (z-buffer)

Z-буфер

- Решение: буфер глубины (z-buffer)
- Одноцветное изображение такого же размера, как экран/окно

Z-буфер

- Решение: буфер глубины (z-buffer)
- Одноцветное изображение такого же размера, как экран/окно
- Каждый пиксель хранит расстояние от нарисованного на экране пикселя до камеры

Z-6үфөр



Z-бүфөр



Тест глубины

- Тест глубины (depth test): при рисовании очередного пикселя (после фрагментного шейдера) проверим: если уже нарисованный пиксель ближе к камере, то новый рисовать не будем

Тест глубины

- Тест глубины (depth test): 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)` (по умолчанию выключен)

Тест глубины

- Тест глубины (depth test): 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)` (по умолчанию выключен)
- Настроить поведение теста глубины: `glDepthFunc(func)`
 - Значения func: `GL_NEVER`, `GL_LESS`, `GL_EQUAL`, `GL_LEQUAL`, `GL_GREATER`, `GL_NOTEQUAL`, `GL_GEQUAL`, `GL_ALWAYS`

Тест глубины

- Тест глубины (depth test): 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)` (по умолчанию выключен)
- 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 расстоянию, записанному в z-буфер

Тест глубины

- Тест глубины (depth test): 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)` (по умолчанию выключен)
- 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 расстоянию, записанному в z-буфер
 - По умолчанию `GL_LESS`

Z-буфер

- Z-буфер хранит значения в формате unsigned normalized (16/24/32-bit) или floating-point (32-bit)
 - Обычно unsigned normalized 24-bit

Z-буфер

- Z-буфер хранит значения в формате unsigned normalized (16/24/32-bit) или floating-point (32-bit)
 - Обычно unsigned normalized 24-bit
- 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

- Z-буфер хранит значения в формате unsigned normalized (16/24/32-bit) или floating-point (32-bit)
 - Обычно unsigned normalized 24-bit
- 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_FragCoord.z`

Z-буфер

- Z-буфер хранит значения в формате unsigned normalized (16/24/32-bit) или floating-point (32-bit)
 - Обычно unsigned normalized 24-bit
- 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_FragCoord.z`
 - Интервал $[0, 1]$ можно заменить с помощью `glDepthRange`

Z-буфер

- Z-буфер хранит значения в формате unsigned normalized (16/24/32-bit) или floating-point (32-bit)
 - Обычно unsigned normalized 24-bit
- 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_FragCoord.z`
 - Интервал $[0, 1]$ можно заменить с помощью `glDepthRange`
 - Во многих графических API исходный диапазон глубин – $[0, 1]$, и Z-координата после perspective divide записывается в буфер глубины без преобразований

Z-буфер: early depth test

- Фрагментный шейдер может изменить глубину фрагмента, записав новую глубину в `gl_FragDepth`

Z-буфер: early depth test

- Фрагментный шейдер может изменить глубину фрагмента, записав новую глубину в `gl_FragDepth`
- Это почти никогда не нужно и выключает *early depth test* – оптимизацию, при которой depth test происходит до фрагментного шейдера, а не после, и пропускает шейдер, если фрагмент всё равно не попадёт на экран

Z-буфер

- Нужно очищать перед каждым кадром:
`glClear(GL_DEPTH_BUFFER_BIT)`

Z-буфер

- 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 OpenGL-контекста

Z-буфер

- 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 OpenGL-контекста
- Можно использовать с любой проекцией (ортографической, перспективной)
- Можно использовать и для рисования в 2D

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

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

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

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

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: $q_{view} = \text{View} \cdot q_{world}$ 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: $q_{view} = \text{View} \cdot q_{world}$ в координатах камеры (view/camera space)
- Проекция камеры: $\text{gl_Position} = q_{clip} = \text{Projection} \cdot q_{view}$ 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: $q_{view} = \text{View} \cdot q_{world}$ в координатах камеры (view/camera space)
- Проекция камеры: $\text{gl_Position} = q_{clip} = \text{Projection} \cdot q_{view}$ 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: $q_{view} = \text{View} \cdot q_{world}$ в координатах камеры (view/camera space)
- Проекция камеры: $\text{gl_Position} = q_{clip} = \text{Projection} \cdot q_{view}$ 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, и не работает тест глубины \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, и не работает тест глубины \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, и не работает тест глубины \Rightarrow нам нужна другая формула проекции!
- Итоговый диапазон глубин ограничен \Rightarrow нужно как-то ограничить диапазон возможных расстояний до камеры
- *Near* и *far* clip planes: $0 < \text{near} < \text{far} < \infty$

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, и не работает тест глубины \Rightarrow нам нужна другая формула проекции!
- Итоговый диапазон глубин ограничен \Rightarrow нужно как-то ограничить диапазон возможных расстояний до камеры
- *Near* и *far* clip planes: $0 < \text{near} < \text{far} < \infty$
- Хотим, чтобы диапазон $z \in [\text{near}, \text{far}]$ 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 $[-1, 1]$
- Матрицей можно применить к z линейную функцию $Az + B$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ Az + B \\ z \end{pmatrix}$$

Перспективная проекция: near & far clip planes

- Хотим, чтобы диапазон $z \in [near, far]$ после применения матрицы проекции и perspective divide превратился в $[-1, 1]$
- Матрицей можно применить к z линейную функцию $Az + B$:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & A & B \\ 0 & 0 & 1 & 0 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ y \\ Az + B \\ z \end{pmatrix}$$

- С учётом perspective divide:

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

Перспективная проекция: near & far clip planes

- С учётом perspective divide:

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

Перспективная проекция: near & far clip planes

- С учётом perspective divide:

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

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

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

Перспективная проекция: near & far clip planes

- С учётом perspective divide:

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

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

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

- Линейная система на A и B:

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

\Rightarrow

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

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

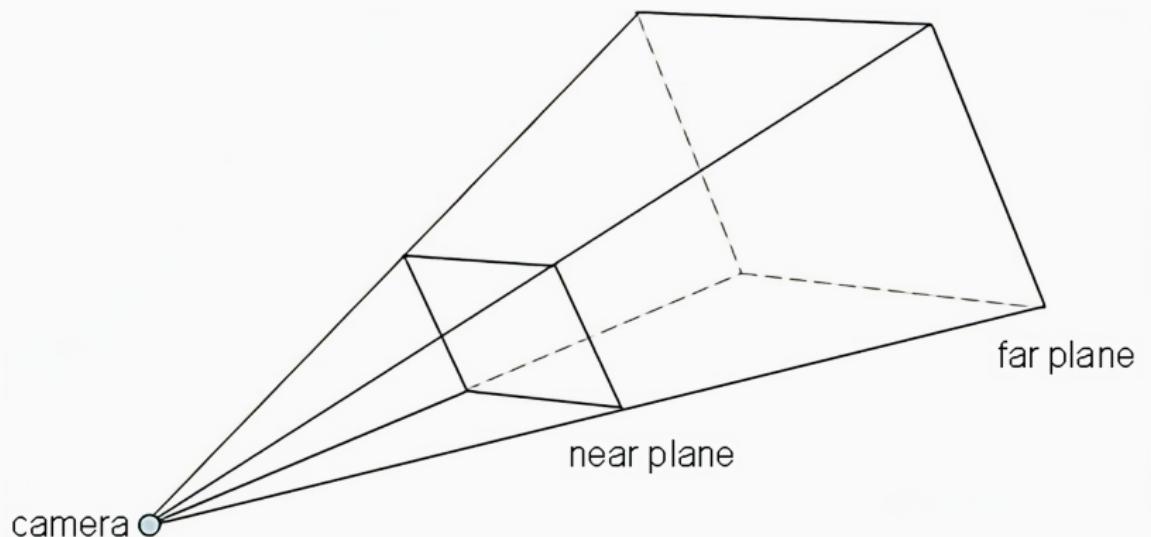
$$\frac{A(-near)+B}{near} = -1$$

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

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

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?

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

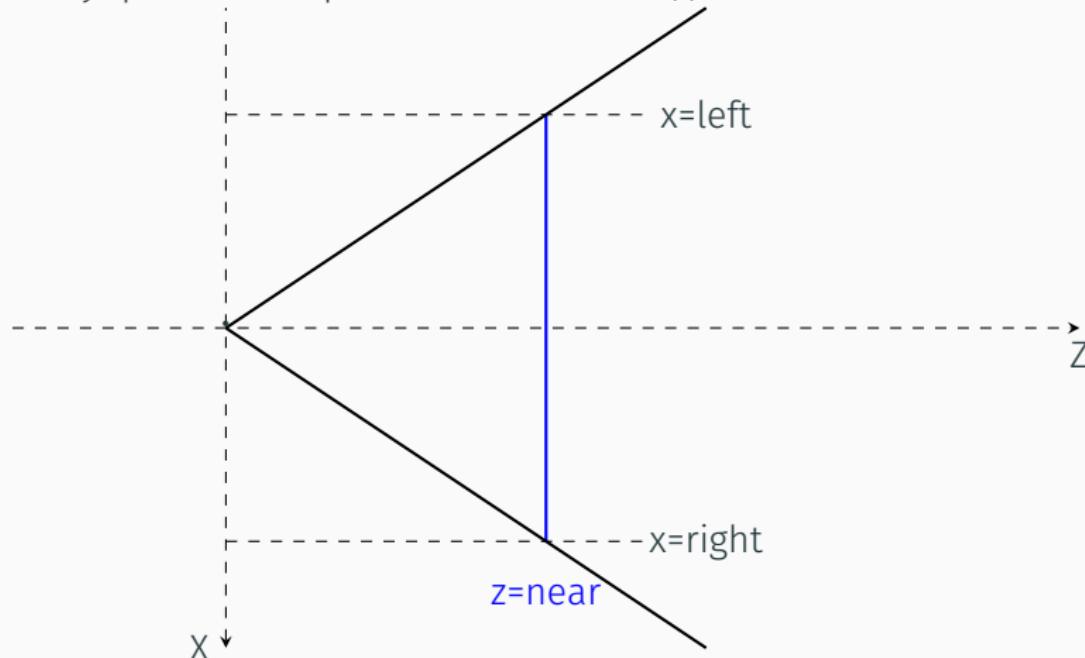
Перспективная проекция: view frustum

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 `left = -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 top и 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 матрицы 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 матрицы проекции и perspective divide) в точку с $X = 1$ и $Z = -1$

Перспективная проекция: view frustum

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

Перспективная проекция: view frustum

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

Аналогично для Y

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

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

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

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: 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*
- Эту матрицы мы передаём в шейдер и применяем к входным вершинам (после других преобразований)

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`?
- Все объекты ближе `near` или дальше `far` обрезаются – почему не взять `near=0` и `far= ∞` ?

Перспективная проекция

- Как выбирать параметры near и far?
- Все объекты ближе near или дальше far обрезаются – почему не взять $\text{near} = 0$ и $\text{far} = \infty$?
- Если $\text{near} = 0$, 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?
- Все объекты ближе near или дальше far обрезаются – почему не взять $\text{near} = 0$ и $\text{far} = \infty$?
- Если $\text{near} = 0$, 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 $\text{far} \rightarrow \infty$, третья строка стремится к $\begin{pmatrix} 0 & 0 & -1 & -2\text{near} \end{pmatrix}$, что само по себе ничего не ломает

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

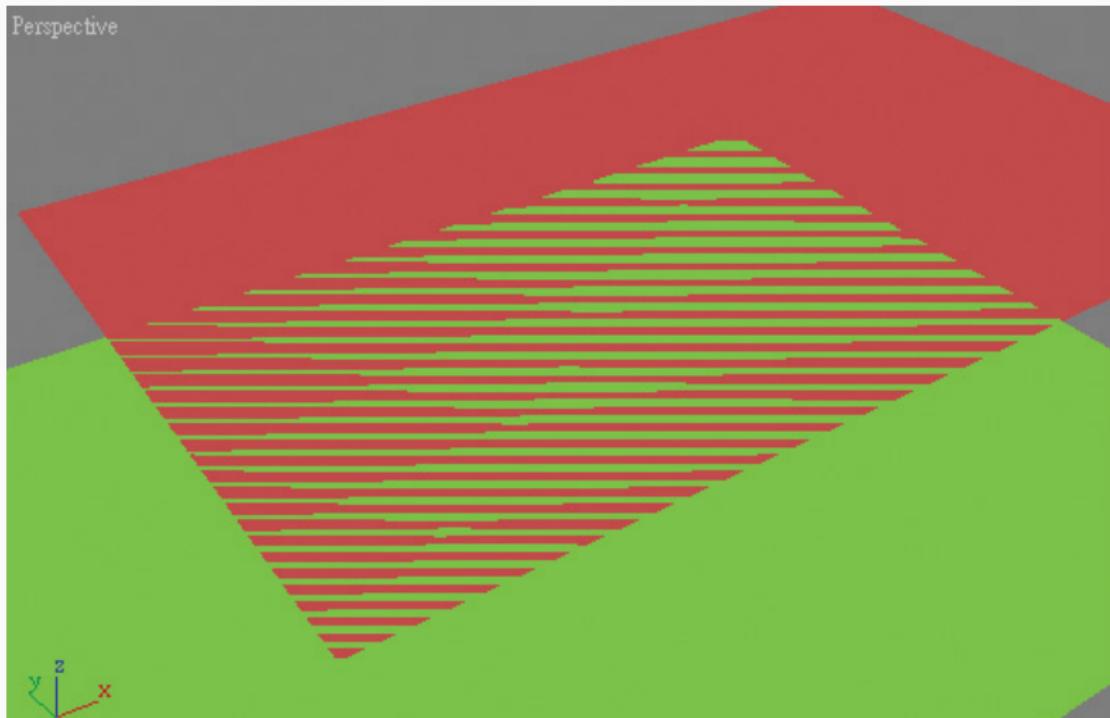
- Z-буфер имеет ограниченную точность
- Чем больше отношение $\frac{far}{near}$, тем меньше точности приходится на единицу расстояния
- Как выбирать параметры near и far?
 - near: максимально возможный, при котором не обрезается что-то существенное
 - При этом камера искусственно держится на некотором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

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: минимально возможный, при котором видно всю сцену
 - Нехватка точности z-буфера приводит к *z-fighting*'у

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

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

Projection · 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

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

Projection · View

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

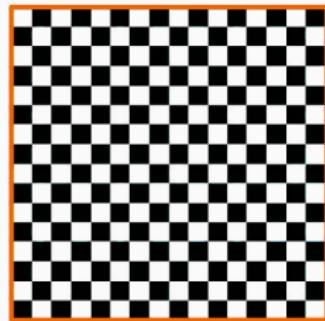
Проекции: ссылки

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

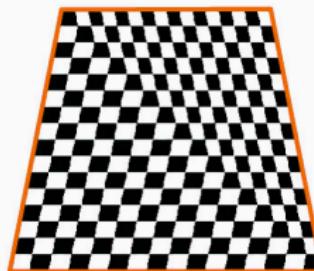
Perspective-correct интерполяция атрибутов

- Перспективная проекция портит интерполяцию атрибутов между вершинным и фрагментным шейдерами

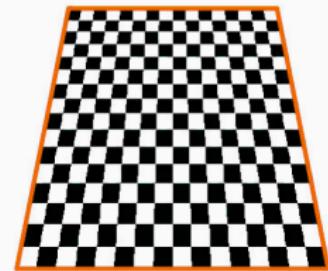
Perspective-correct интерполяция атрибутов



Texture



Affine
screen-space
interpolation



Perspective
world-space
interpolation

Perspective-correct интерполяция атрибутов

- Перспективная проекция портит интерполяцию атрибутов между вершинным и фрагментным шейдерами

Perspective-correct интерполяция атрибутов

- Перспективная проекция портит интерполяцию атрибутов между вершинным и фрагментным шейдерами
- На самом деле, барицентрические координаты вычисляются с учётом перспективной проекции:

Perspective-correct интерполяция атрибутов

- Перспективная проекция портит интерполяцию атрибутов между вершинным и фрагментным шейдерами
- На самом деле, барицентрические координаты вычисляются с учётом перспективной проекции:
 - Между вершинами линейно интерполируется $1 / \text{gl_Position.w}$

Perspective-correct интерполяция атрибутов

- Перспективная проекция портит интерполяцию атрибутов между вершинным и фрагментным шейдерами
- На самом деле, барицентрические координаты вычисляются с учётом перспективной проекции:
 - Между вершинами линейно интерполируются `1 / gl_Position.w`
 - Исходные барицентрические координаты λ_i (посчитанные в экранных координатах) заменяются на

$$\frac{\lambda_i/w_i}{\sum \lambda_k/w_k}$$

Perspective-correct интерполяция атрибутов

- Перспективная проекция портит интерполяцию атрибутов между вершинным и фрагментным шейдерами
- На самом деле, барицентрические координаты вычисляются с учётом перспективной проекции:
 - Между вершинами линейно интерполируется $1 / \text{gl_Position.w}$
 - Исходные барицентрические координаты λ_i (посчитанные в экранных координатах) заменяются на

$$\frac{\lambda_i/w_i}{\sum \lambda_k/w_k}$$

- Проинтерполированное значение $1/w$ доступно в шейдере: `gl_FragCoord.w`

Perspective-correct интерполяция атрибутов

- Перспективная проекция портит интерполяцию атрибутов между вершинным и фрагментным шейдерами
- На самом деле, барицентрические координаты вычисляются с учётом перспективной проекции:
 - Между вершинами линейно интерполируется `1 / gl_Position.w`
 - Исходные барицентрические координаты λ_i (посчитанные в экранных координатах) заменяются на

$$\frac{\lambda_i/w_i}{\sum \lambda_k/w_k}$$

- Проинтерполированное значение $1/w$ доступно в шейдере: `gl_FragCoord.w`
- Можно выключить это поведение для конкретного атрибута с помощью модификатора `noperspective` в шейдере

Perspective-correct интерполяция атрибутов

- StackOverflow: How exactly does OpenGL do perspective-correct linear interpolation
- ScratchAPixel: Perspective-correct interpolation of vertex attributes

Бонус: как найти координаты камеры, зная матрицу камеры?

- Пусть $T = \text{Projection} \cdot \text{View}$ – матрица камеры

Бонус: как найти координаты камеры, зная матрицу камеры?

- Пусть $T = \text{Projection} \cdot \text{View}$ – матрица камеры
- При ортографической проекции центр видимой области переходит в $(0, 0, 0, 1)$

Бонус: как найти координаты камеры, зная матрицу камеры?

- Пусть $T = \text{Projection} \cdot \text{View}$ – матрица камеры
- При ортографической проекции центр видимой области переходит в $(0, 0, 0, 1)$
- \Rightarrow нужно вычислить $T^{-1} \cdot (0, 0, 0, 1)$

Бонус: как найти координаты камеры, зная матрицу камеры?

- Пусть $T = \text{Projection} \cdot \text{View}$ – матрица камеры
- При ортографической проекции центр видимой области переходит в $(0, 0, 0, 1)$
- \Rightarrow нужно вычислить $T^{-1} \cdot (0, 0, 0, 1)$
- При перспективной проекции центр проекции переходит в бесконечно удалённую точку $(0, 0, -1, 0)$

Бонус: как найти координаты камеры, зная матрицу камеры?

- Пусть $T = \text{Projection} \cdot \text{View}$ – матрица камеры
- При ортографической проекции центр видимой области переходит в $(0, 0, 0, 1)$
- \Rightarrow нужно вычислить $T^{-1} \cdot (0, 0, 0, 1)$
- При перспективной проекции центр проекции переходит в бесконечно удалённую точку $(0, 0, -1, 0)$
- \Rightarrow нужно вычислить $T^{-1} \cdot (0, 0, -1, 0)$ и применить к результату perspective divide (разделить XYZ на W)

Бонус: как найти координаты камеры, зная матрицу камеры?

- Пусть $T = \text{Projection} \cdot \text{View}$ – матрица камеры
- При ортографической проекции центр видимой области переходит в $(0, 0, 0, 1)$
- \Rightarrow нужно вычислить $T^{-1} \cdot (0, 0, 0, 1)$
- При перспективной проекции центр проекции переходит в бесконечно удалённую точку $(0, 0, -1, 0)$
- \Rightarrow нужно вычислить $T^{-1} \cdot (0, 0, -1, 0)$ и применить к результату perspective divide (разделить XYZ на W)
- Полезно, когда позиция камеры не задаётся явно, а вычисляется на основе других величин (точка фокуса камеры, углы поворота, расстояние до фокуса, etc)

Бонус: как перевести из 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)

Бонус: как перевести из 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)
- В пространстве точке на экране соответствует луч из камеры – как его найти?

Бонус: как перевести из 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)
- В пространстве точке на экране соответствует луч из камеры – как его найти?
- Для перспективной проекции: найдём центр проекции, и точку на near или far плоскости, где её пересекает этот луч

Бонус: как перевести из 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)
- В пространстве точке на экране соответствует луч из камеры – как его найти?
- Для перспективной проекции: найдём центр проекции, и точку на near или far плоскости, где её пересекает этот луч
- Для любой проекции: найдём две точки, пересекающие этот луч, на near и far плоскостях

Бонус: как перевести из экранных координат в мировые?

- Пусть Proj – оператор, выполняющий perspective divide:

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

Бонус: как перевести из экранных координат в мировые?

- Пусть Proj – оператор, выполняющий perspective divide:

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

- 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 из 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 из экранных координат в мировые?

- Мы хотим решить уравнение

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

- Как выглядит множество $\text{Proj}^{-1} \begin{pmatrix} x \\ y \\ z \end{pmatrix}$?

Бонус: как перевести из экранных координат в мировые?

- Мы хотим решить уравнение

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

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

Бонус: как перевести из экранных координат в мировые?

- Мы хотим решить уравнение

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

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

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

Бонус: как перевести из 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 из экранных координат в мировые?

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

- Приравняем последнюю координату левой и правой стороны уравнения:

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

Бонус: как перевести из экранных координат в мировые?

$$\lambda = \frac{1}{\begin{bmatrix} P_X \\ P_Y \\ \pm 1 \\ 1 \end{bmatrix}_{w^{-1}}}$$

Бонус: как перевести из экранных координат в мировые?

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

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

Бонус: как перевести из экранных координат в мировые?

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

Бонус: как перевести из экранных координат в мировые?

- 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 перевести из экранных координат в мировые?

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

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

- Работает для любой проекции (и перспективной, и ортографической)

Бонус: как перевести из экранных координат в мировые?

$$\text{Proj } T \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

$$\implies \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \text{Proj } T^{-1} \begin{pmatrix} a \\ b \\ c \\ 1 \end{pmatrix}$$

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

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