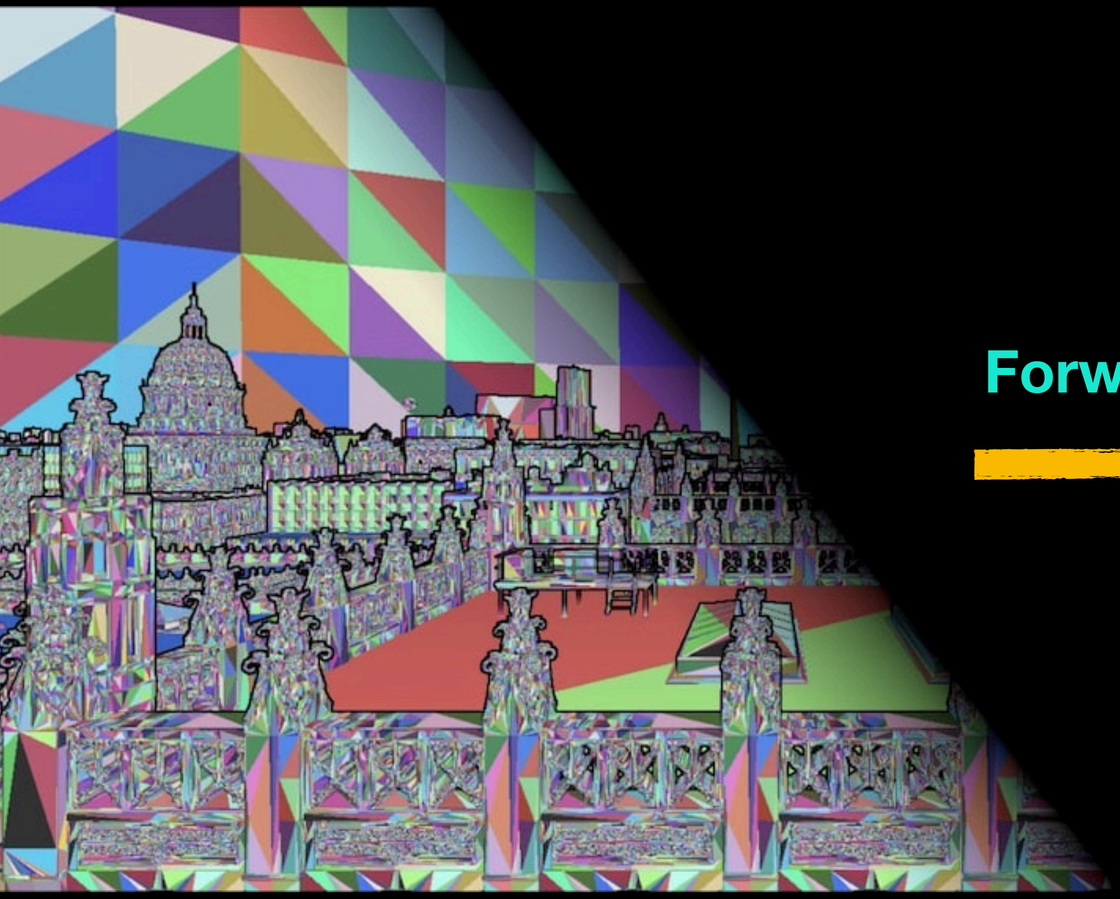
Дифференцируемая графика

Part 1: Forward and Inverse Rendering

Дмитрий Сенюшкин, Николай Патакин — Samsung AI Center Moscow

Введение

Forward Rendering



Forward Rendering

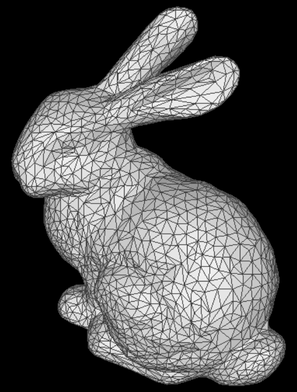
$$I = \mathscr{R}(\theta)$$

$\theta =$ Геометрия, Материалы, …

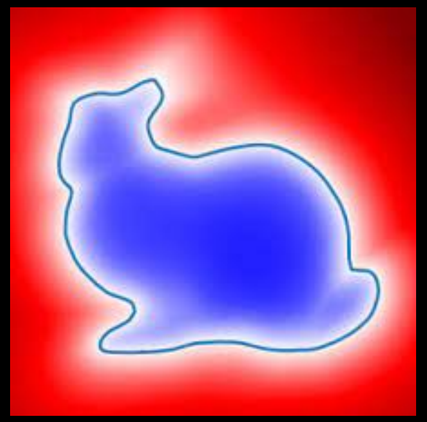
$I =$ Цвет пикселя

Forward rendering

Представления сцены



Triangular mesh



Implicit
representation
(e.g. SDF)

Volumetric
representation
(e.g. NeRF)

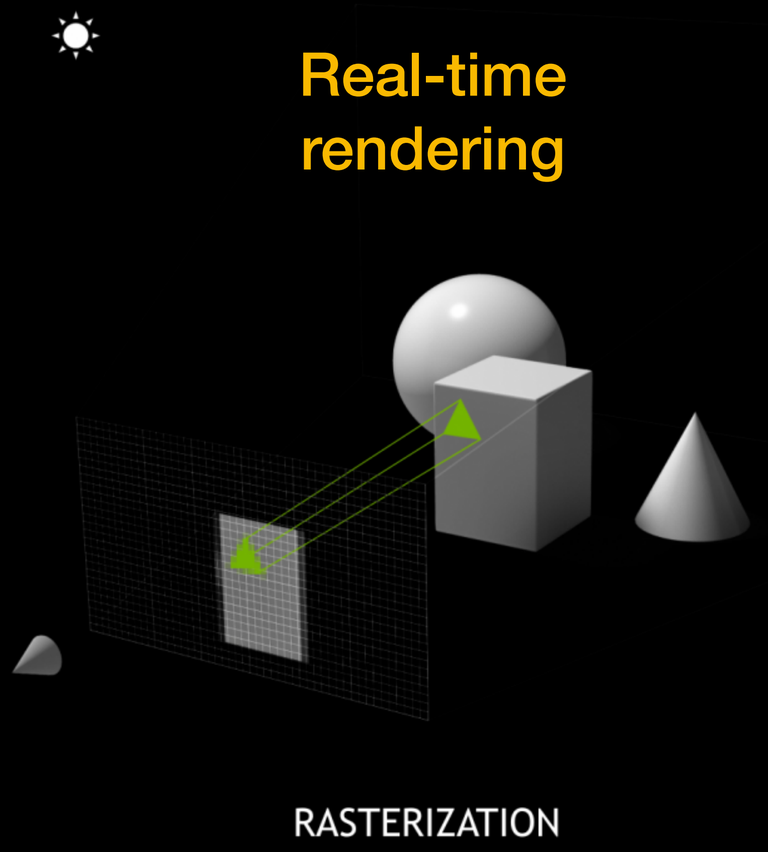
Явные — треугольные меши с текстурами;
Неявные — неявное функциональные представления (SDF, NeRF)

В зависимости от представлений удобно применять разные методы рендеринга

THE HOLY GRAIL OF GRAPHICS

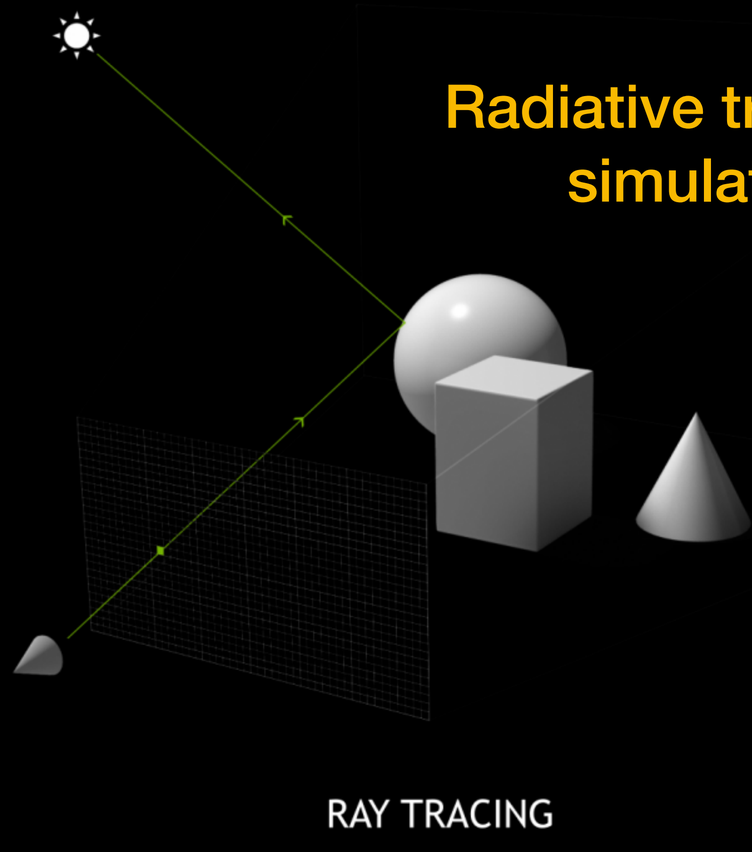
Два основных подхода:

Real-time rendering



Предмет нашей лекции

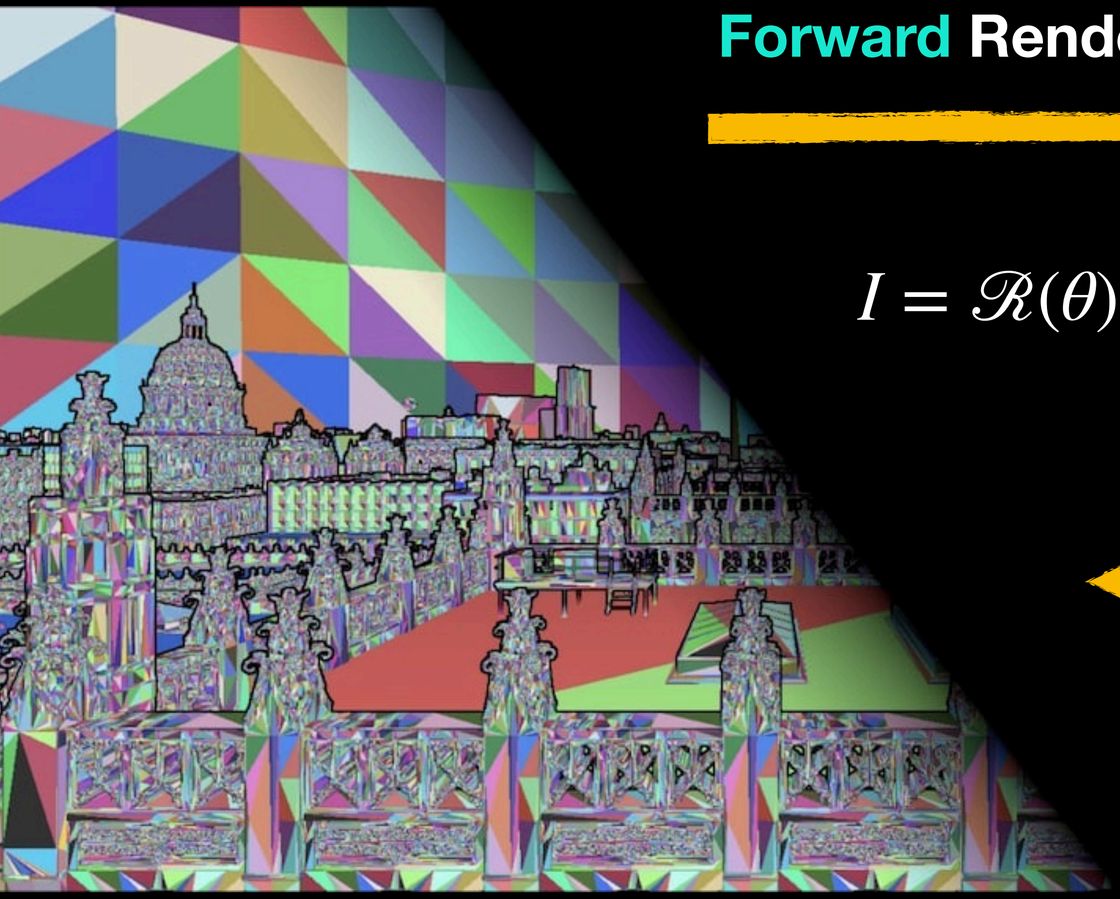
Radiative transport simulation




UNREAL
ENGINE


UNREAL
ENGINE

Forward and Inverse Rendering



Forward Rendering

$$I = \mathscr{R}(\theta)$$

Inverse Rendering

$$\theta = \mathscr{R}^{-}1(I)$$

θ = Геометрия, Материалы, …

I = Цвет пикселя

Inverse rendering

Формальная постановка задачи

- Inverse rendering — это процесс реконструкции сцены из входных изображений этой сцены.
- Формально мы рассматриваем его как процесс оптимизации параметров сцены θ относительно функции ошибки $\mathscr{L}$ расчитанной на обучающей выборке D

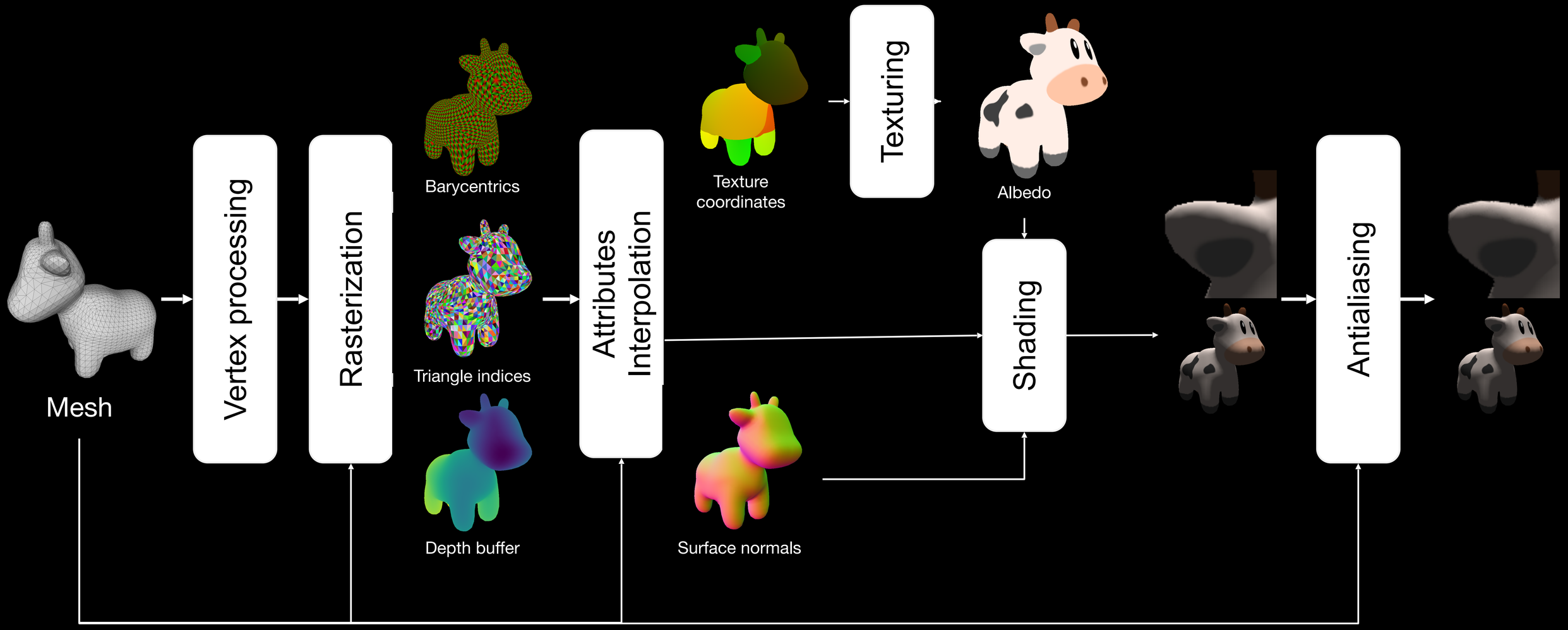
$$\min_{\theta} \mathscr{L}(\hat{I}, I \mid D)$$

- Для осуществления такой минимизации градиентными методами необходимо уметь рассчитывать производные по параметра, т.е. необходимо уметь дифференцировать пайплайны прямого рендеринга

Дифференцируемая Растеризация

Rasterization

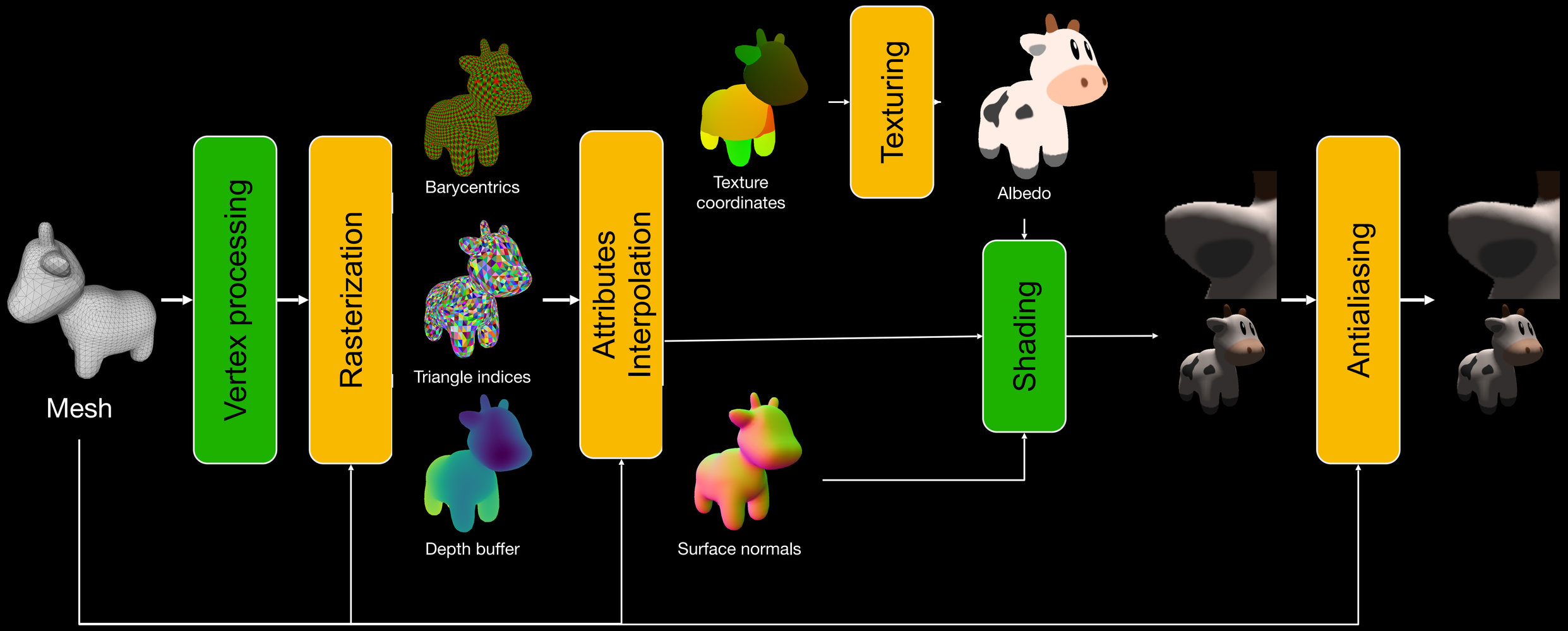
Общий пайплайн

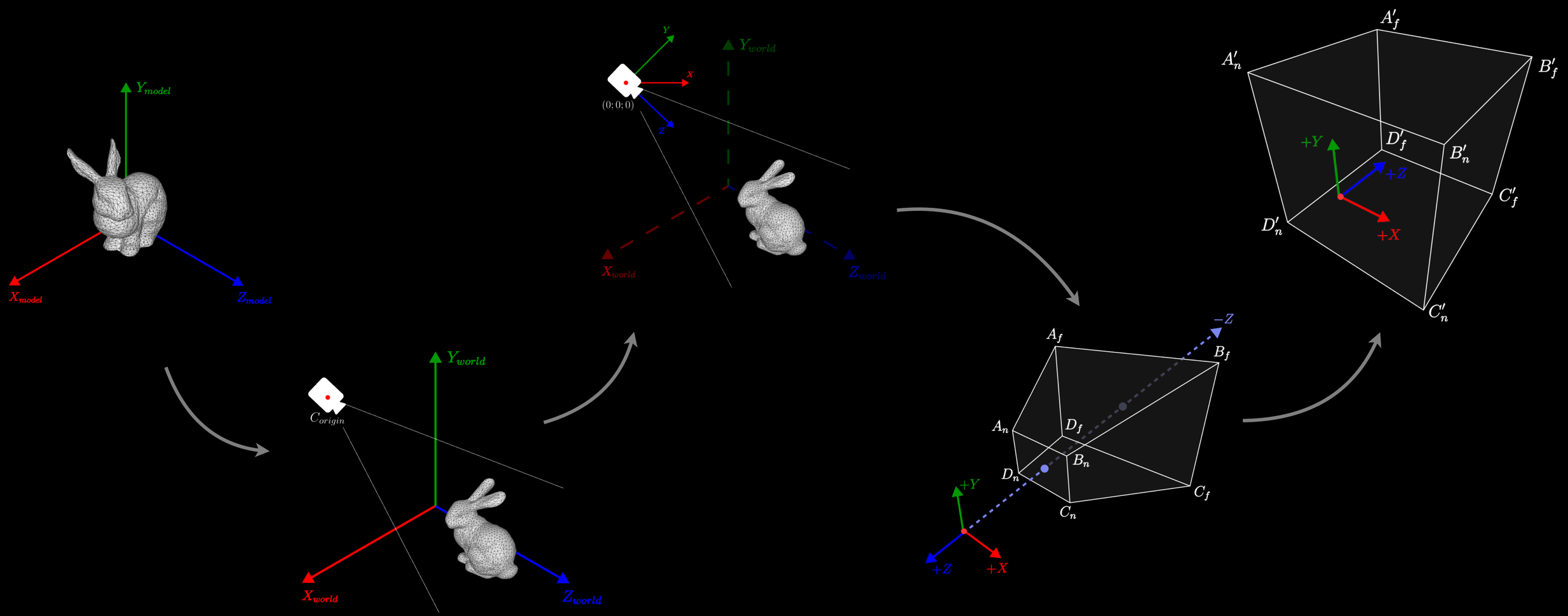


NVDIFFRAST

– Определены NVDIFFRAST

– Определяются пользователем

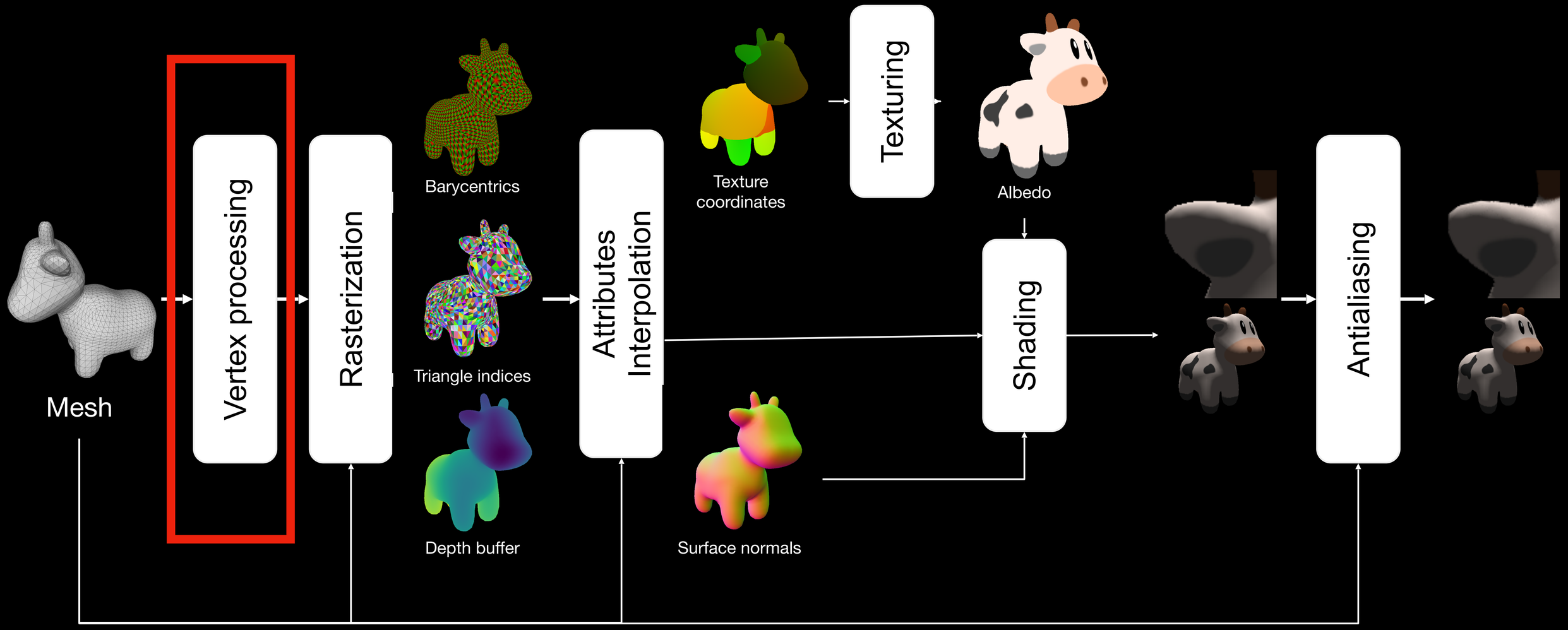




Вершинные преобразования

Rasterization

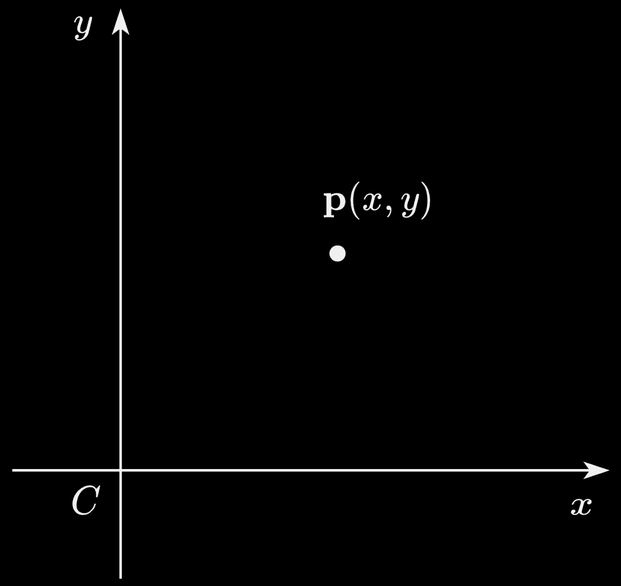
Общий пайплайн



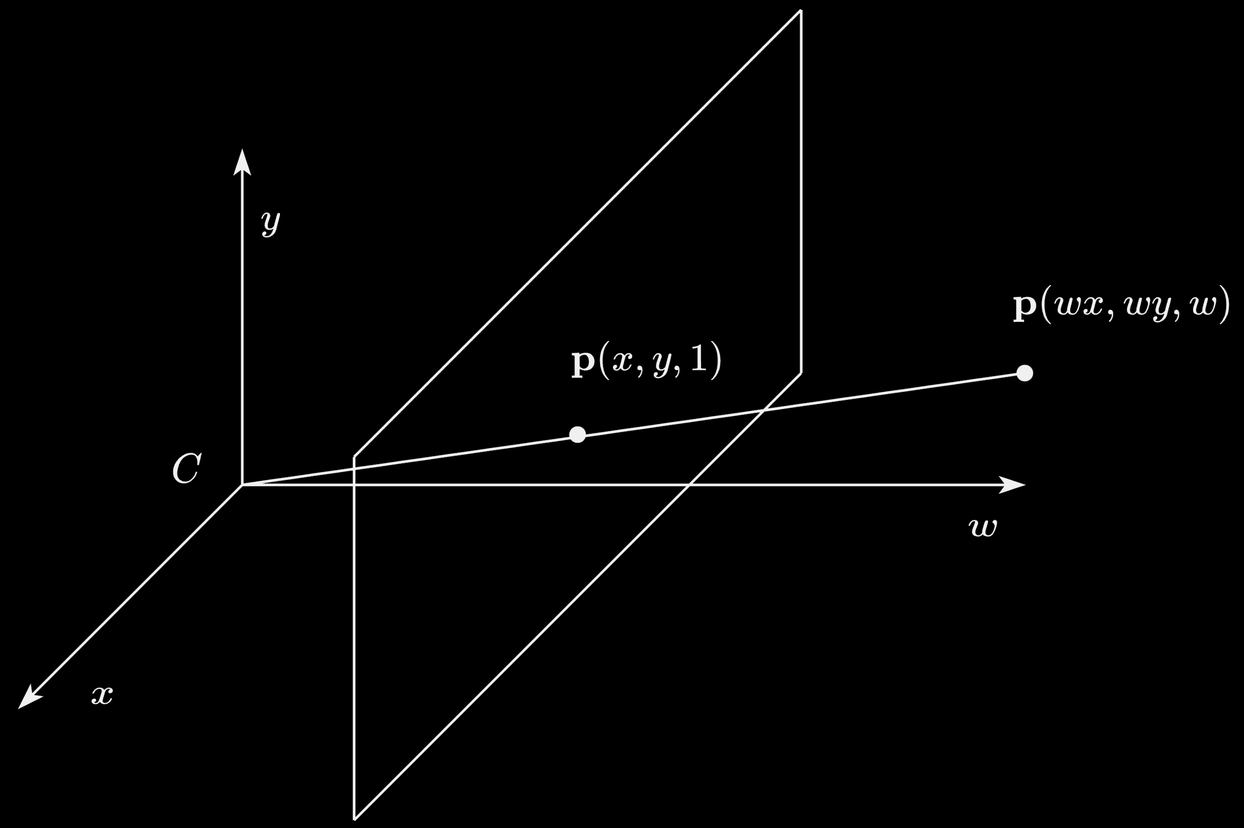
Однородные координаты

Связь евклидова и проективного пространств

Евклидово, $\mathbb{R}^2$



Проективное, $\mathbb{H}^2$

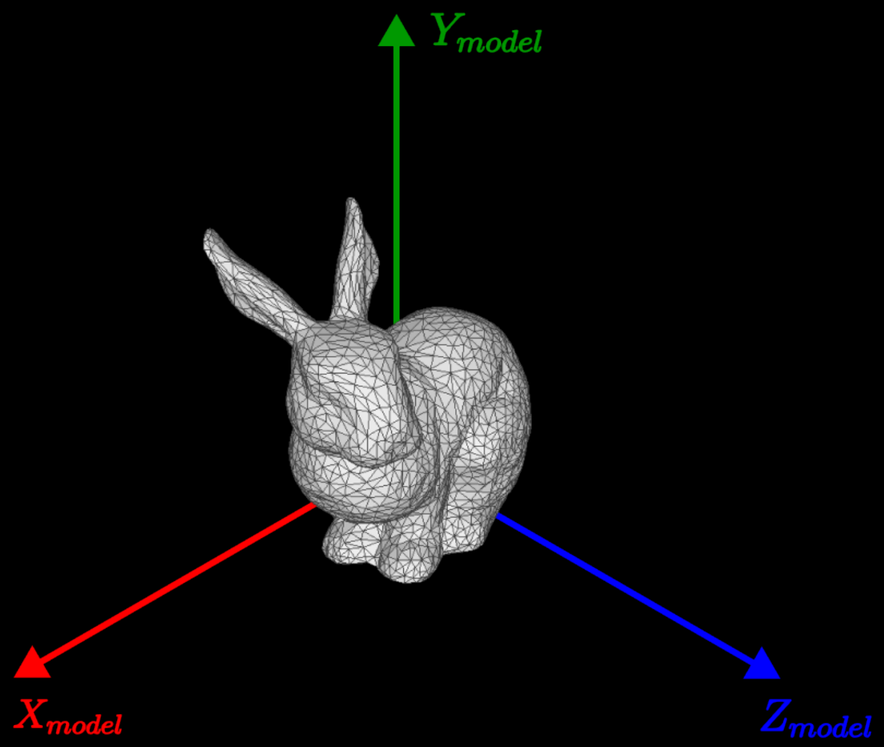


$$\mathbf{p}(x, y) \in \mathbb{R}^2 \leftrightarrow \mathbf{p}(wx, wy, w) \in \mathbb{H}^2$$

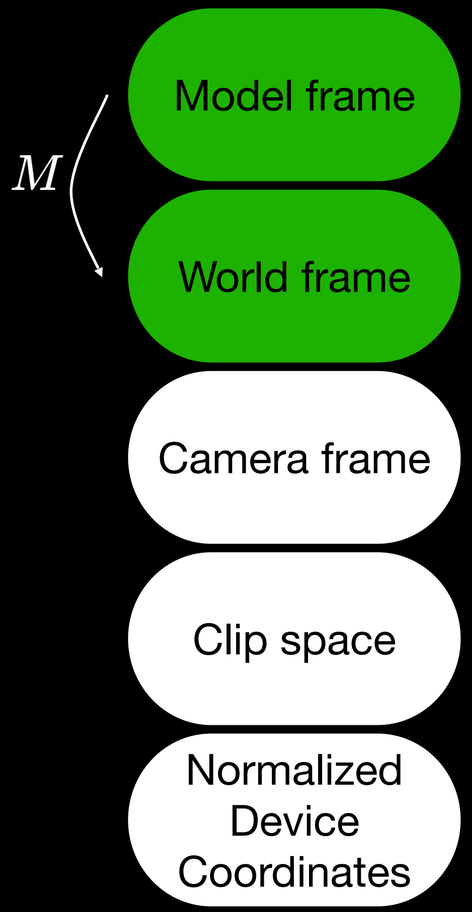
Иерархия преобразований

- Model frame
- World frame
- Camera frame
- Clip space
- Normalized Device Coordinates

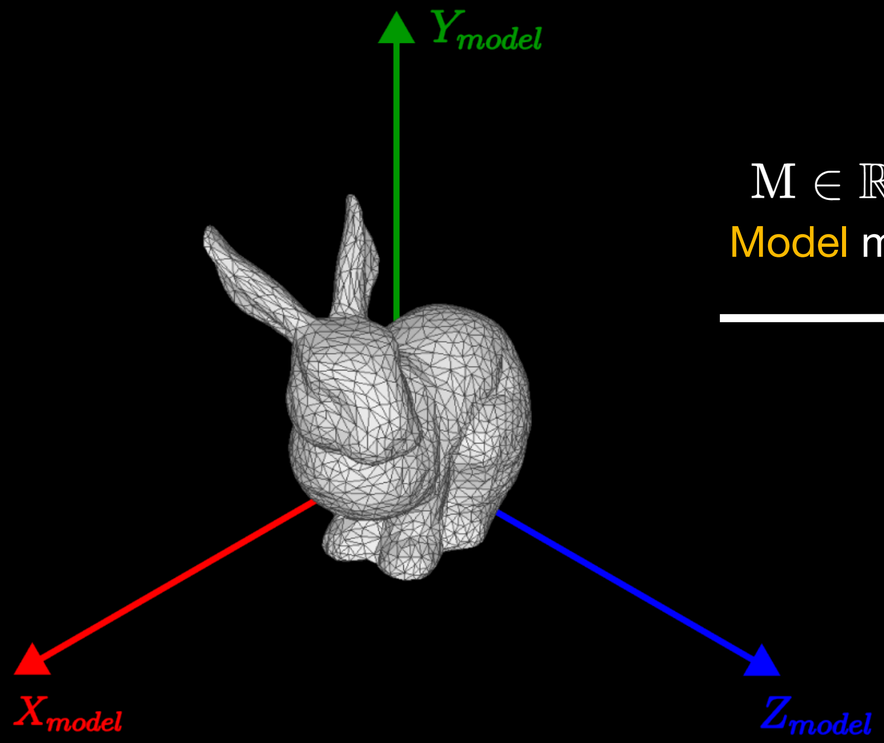
Model frame -
координатная система объекта



Иерархия преобразований

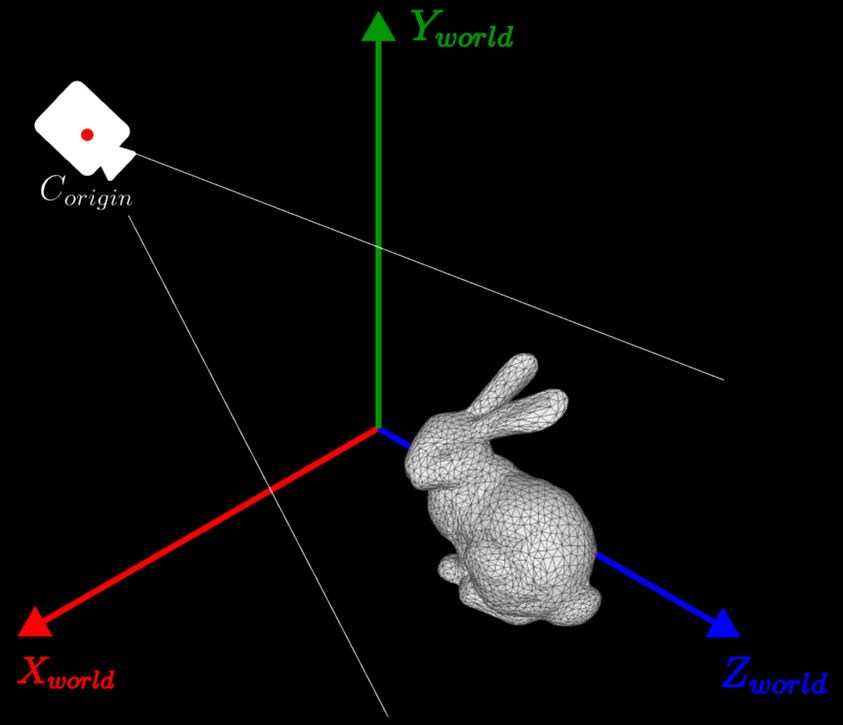


Model frame -
координатная система объекта



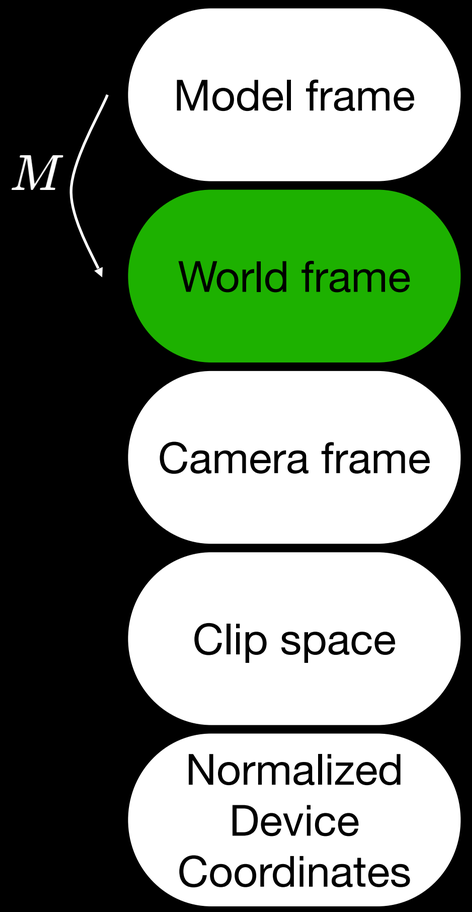
$\mathbf{M} \in \mathbb{R}^{4\times4}$
Model matrix

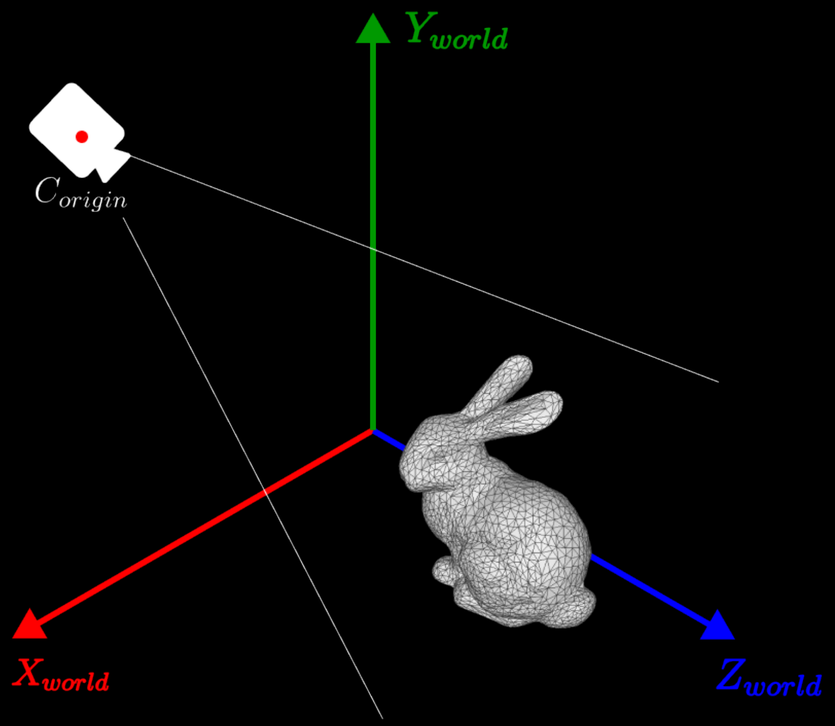
World frame -
координатная система сцены



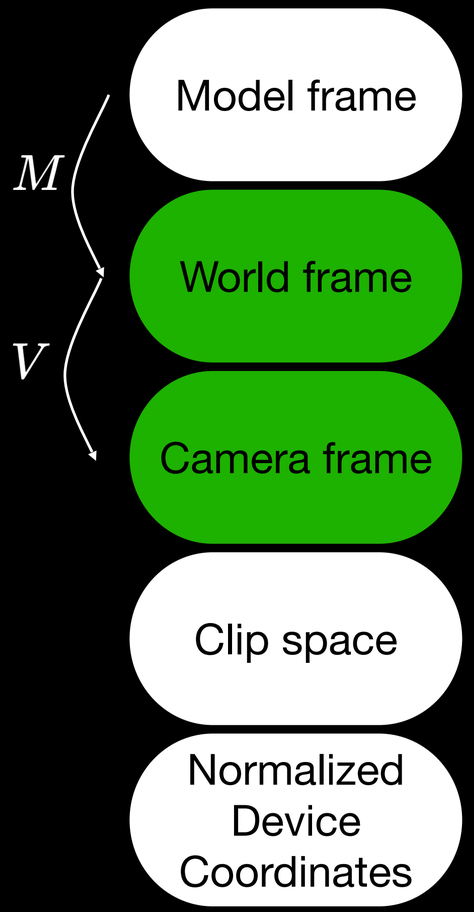
Иерархия преобразований

World frame -
координатная система сцены



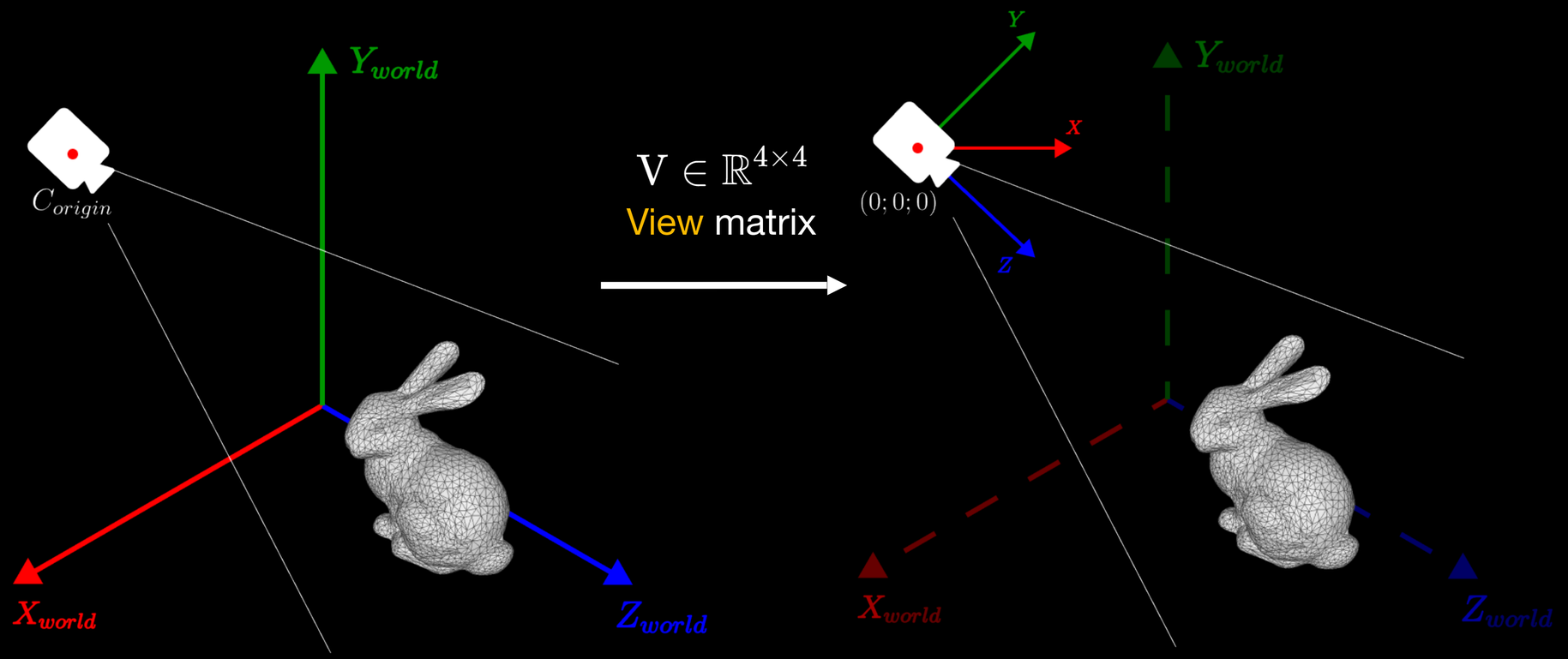


Иерархия преобразований



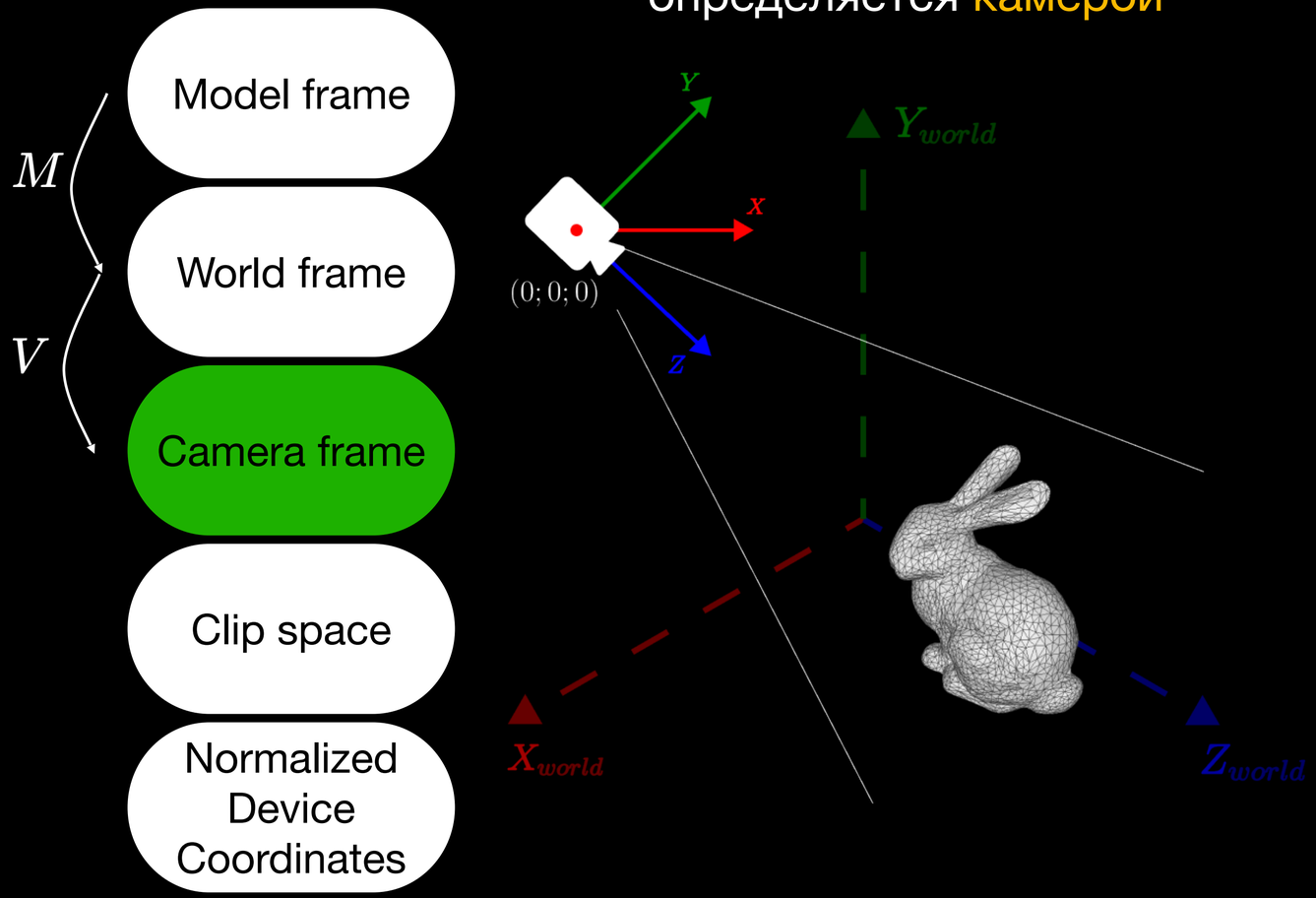
World frame -
координатная система сцены

Camera frame -
определяется камерой



Иерархия преобразований

Camera frame -
определяется камерой



Иерархия преобразований

Model frame

M

World frame

V

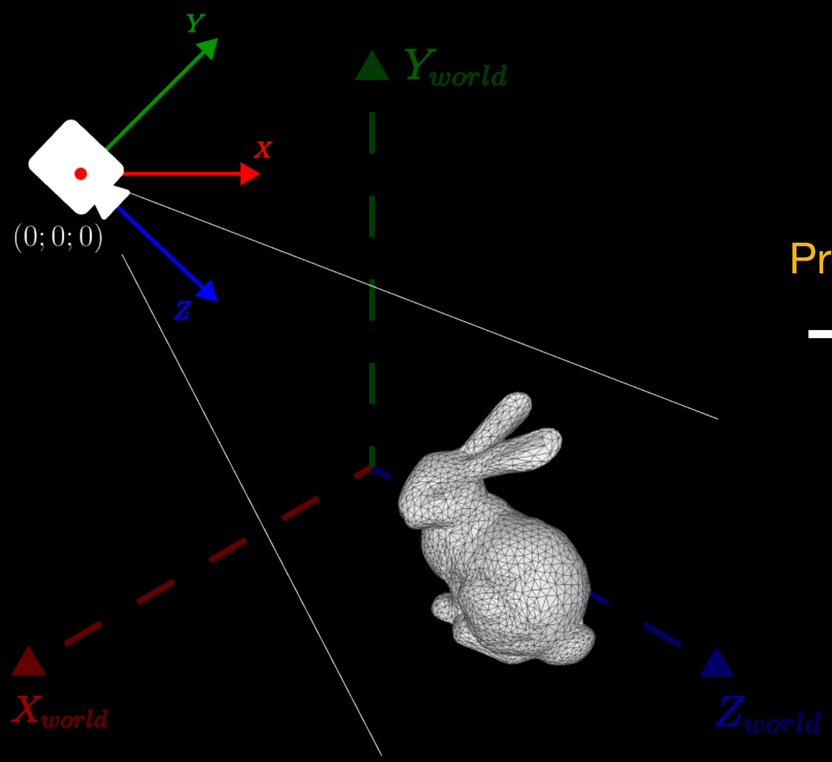
Camera frame

P

Clip space

Normalized Device Coordinates

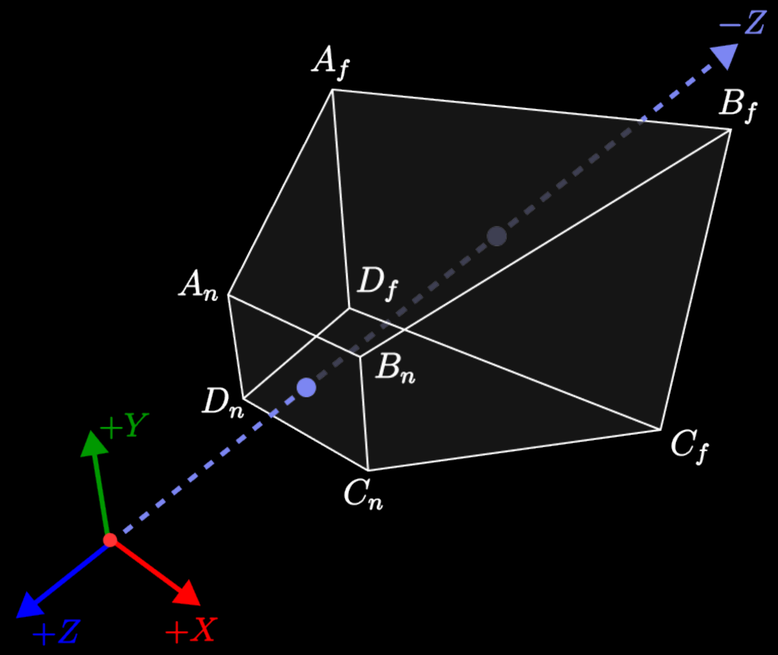
Camera frame - определяется камерой



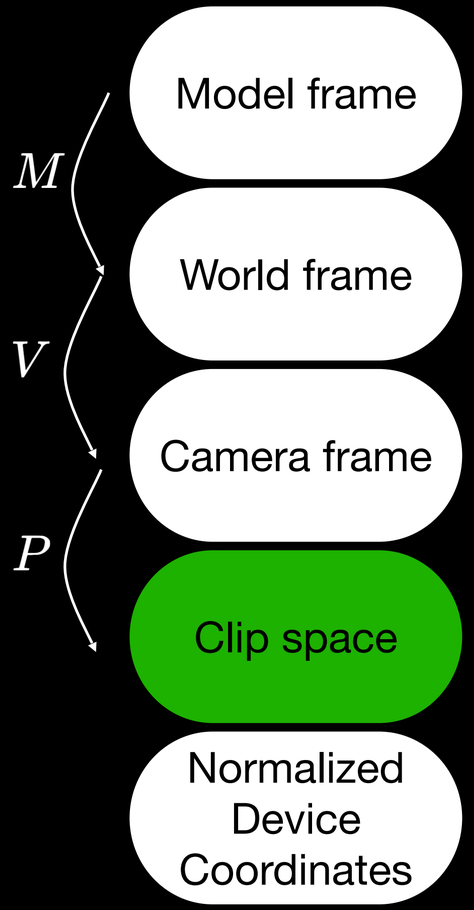
$\mathrm{P} \in \mathbb{R}^{4\times4}$

Projection matrix

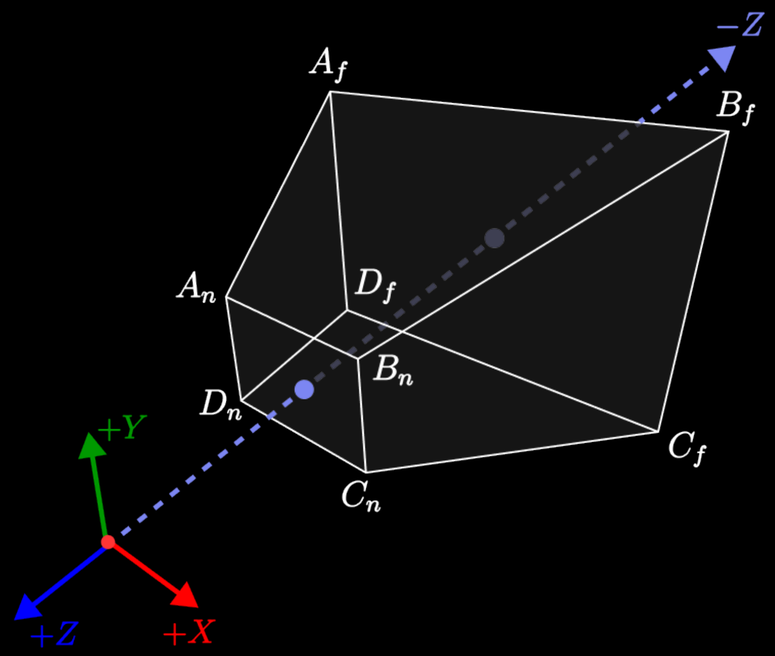
Clip space - определяется камерой



Иерархия преобразований

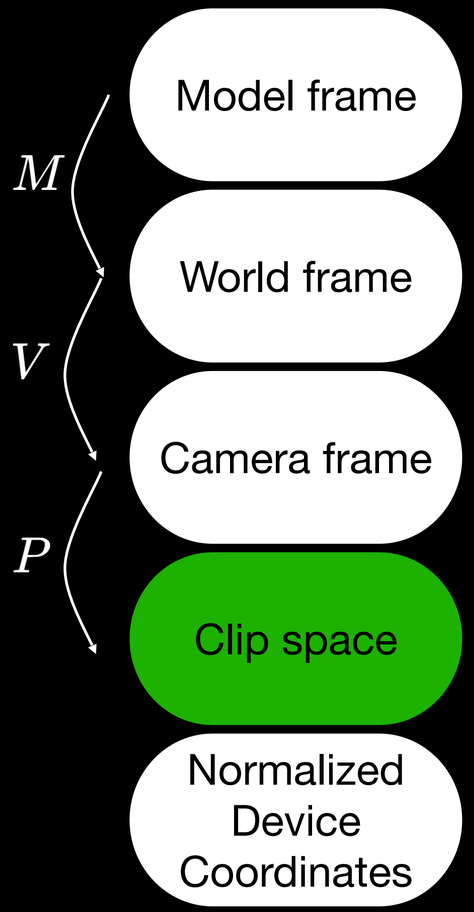


Clip space - определяется камерой

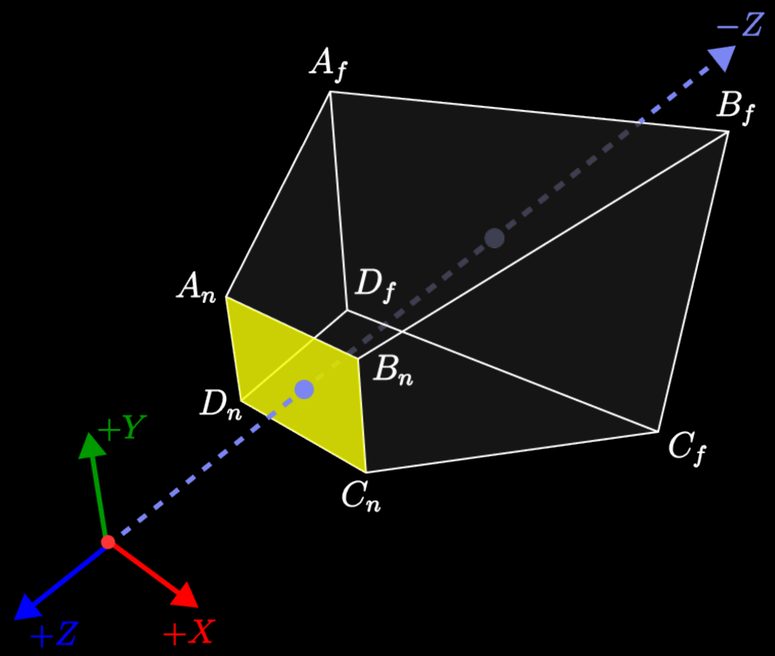


Ограничивается объем между ближней и дальней плоскостями:

Иерархия преобразований



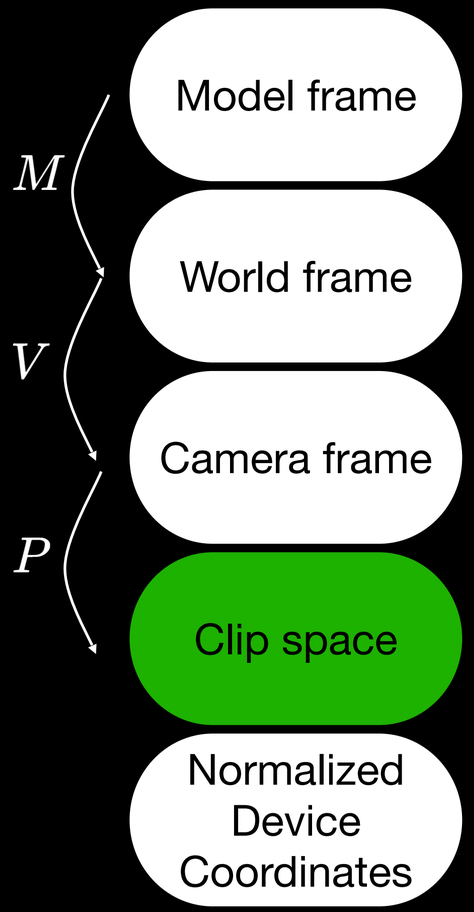
Clip space - определяется камерой



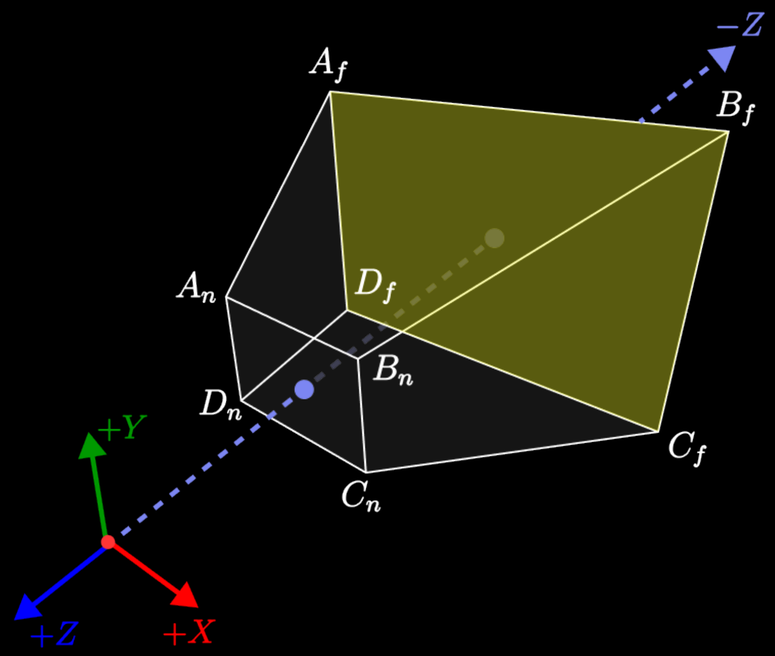
Ограничивается объем между ближней и дальней плоскостями:

$A_nB_nC_nD_n$ - ближняя плоскость, проекционный экран

Иерархия преобразований



Clip space - определяется камерой

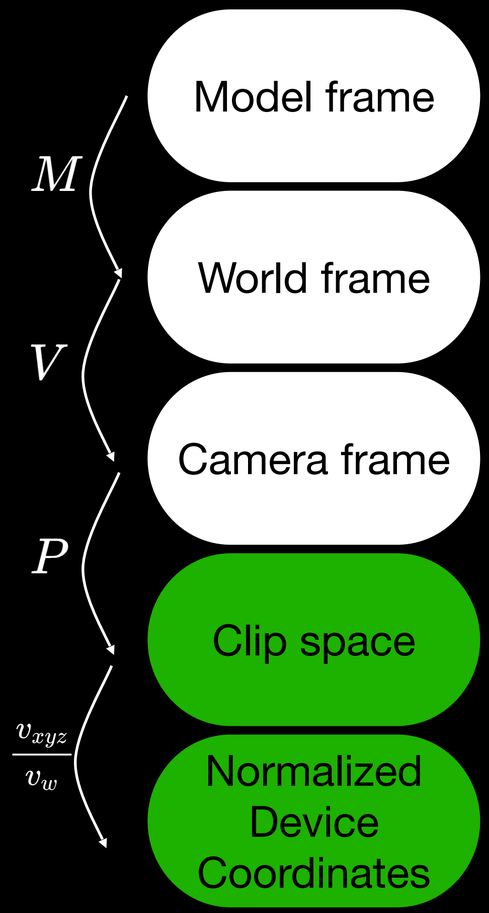


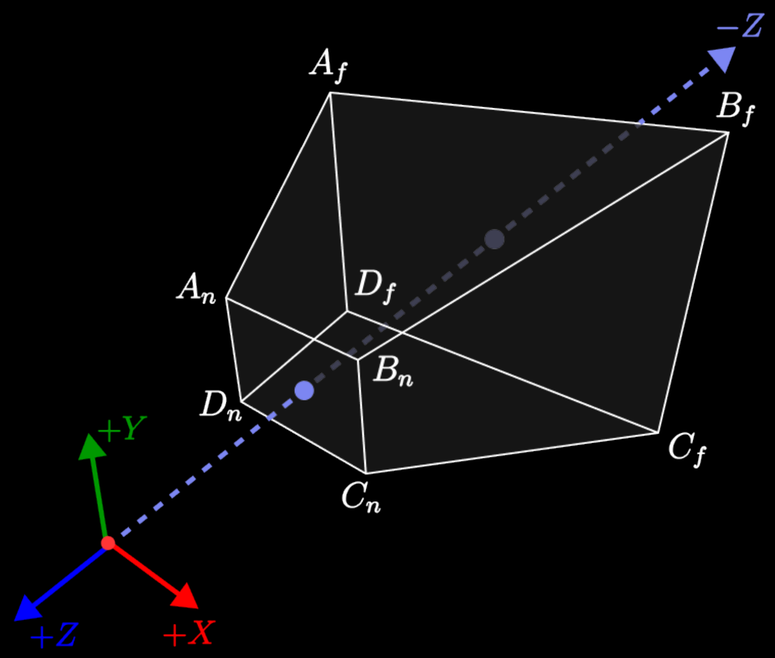
Ограничивается объем между ближней и дальней плоскостями:

$A_nB_nC_nD_n$ - ближняя плоскость, проекционный экран

$A_fB_fC_fD_f$ - дальняя плоскость

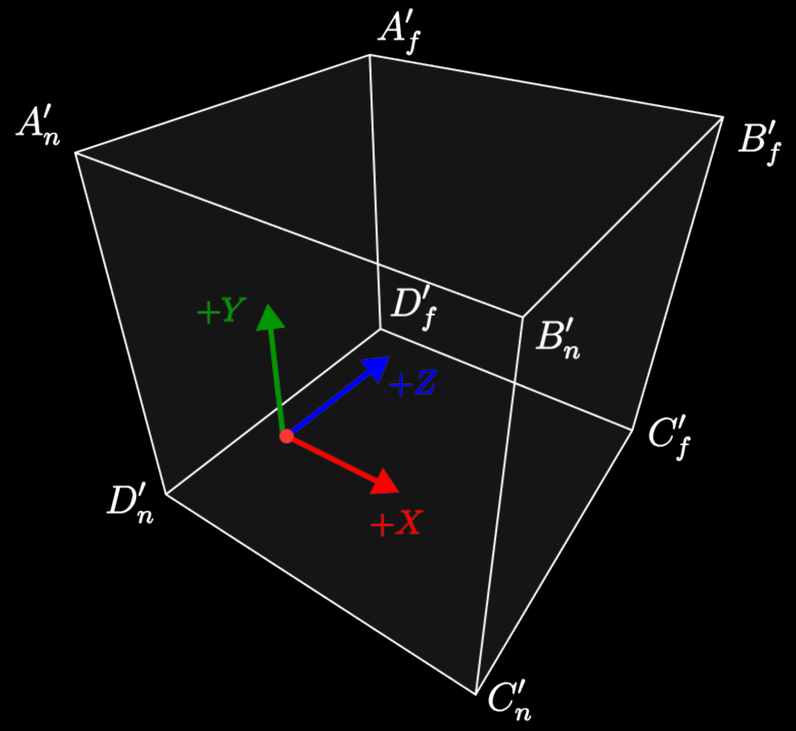
Иерархия преобразований





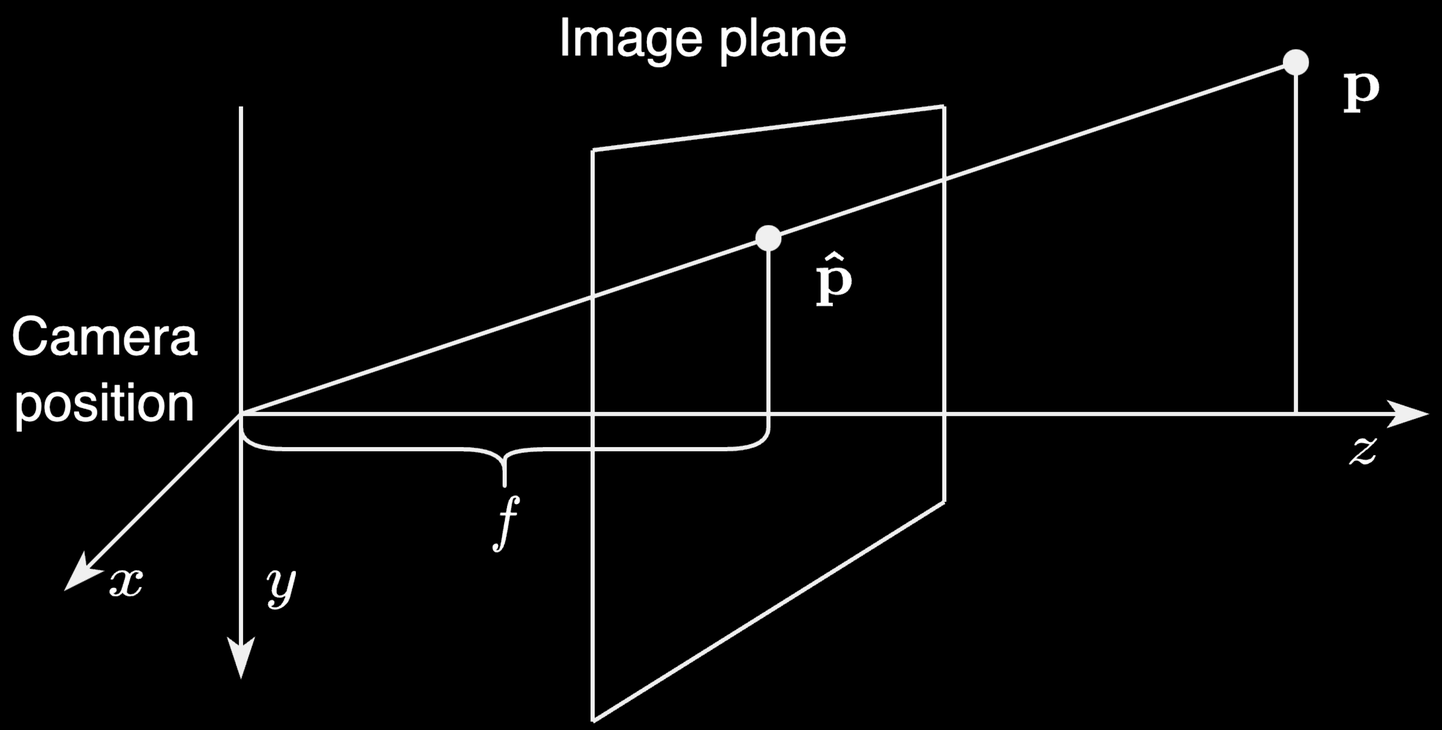
Perspective division

$$v' = \frac{v_{xyz}}{v_w}$$



Перспективная проекция

Pinhole camera



$\mathbf{p}$ — точка в координатной системе камеры

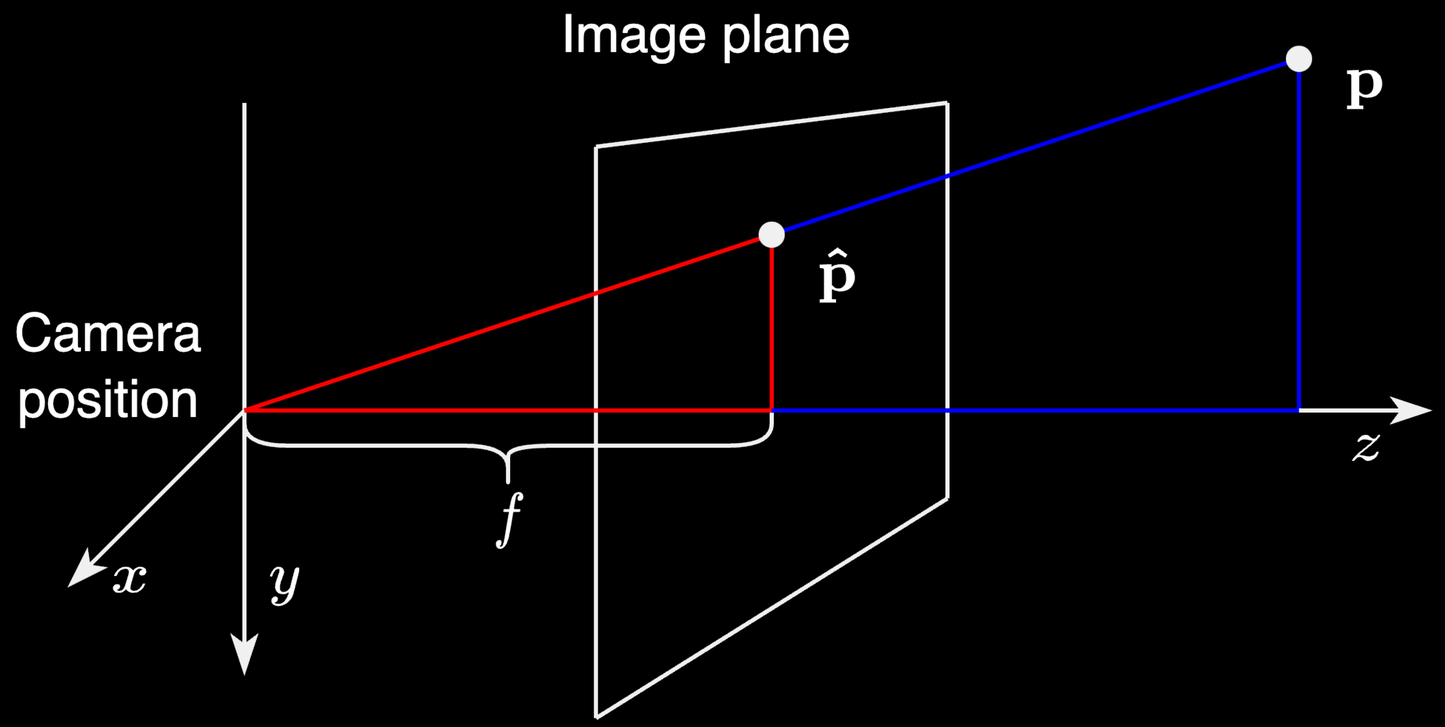
$\hat{\mathbf{p}}$ — проекция точки на плоскость изображения

f — фокусное расстояние камеры

Pinhole camera - простейшая модель камера без линз с бесконечно малой апертурой (размер отверстия, через которое попадает свет)

Перспективная проекция

Pinhole camera



Матрица проекции $\mathbb{P} : \mathbb{H}^3 \mapsto \mathbb{H}^3$

Из подобия треугольников:

$$\mathbf{p}\left(p_x, p_y, p_z\right) \overset{\mathbb{P}}{\longmapsto} \hat{\mathbf{p}}\left(f\frac{p_x}{p_z}, f\frac{p_y}{p_z}, f\right)$$

В однородных координатах:

$$\underbrace{\hat{\mathbf{p}}\left(f\frac{p_x}{p_z}, f\frac{p_y}{p_z}, f\right)}_{\in\mathbb{R}^3} \longmapsto \underbrace{\hat{\mathbf{p}}\left(fp_x, fp_y, fp_z, p_z\right)}_{\in\mathbb{H}^3}$$

$$\hat{\mathbf{p}} = \begin{pmatrix} fp_x \\ fp_y \\ fp_z \\ p_z \end{pmatrix} = \begin{pmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

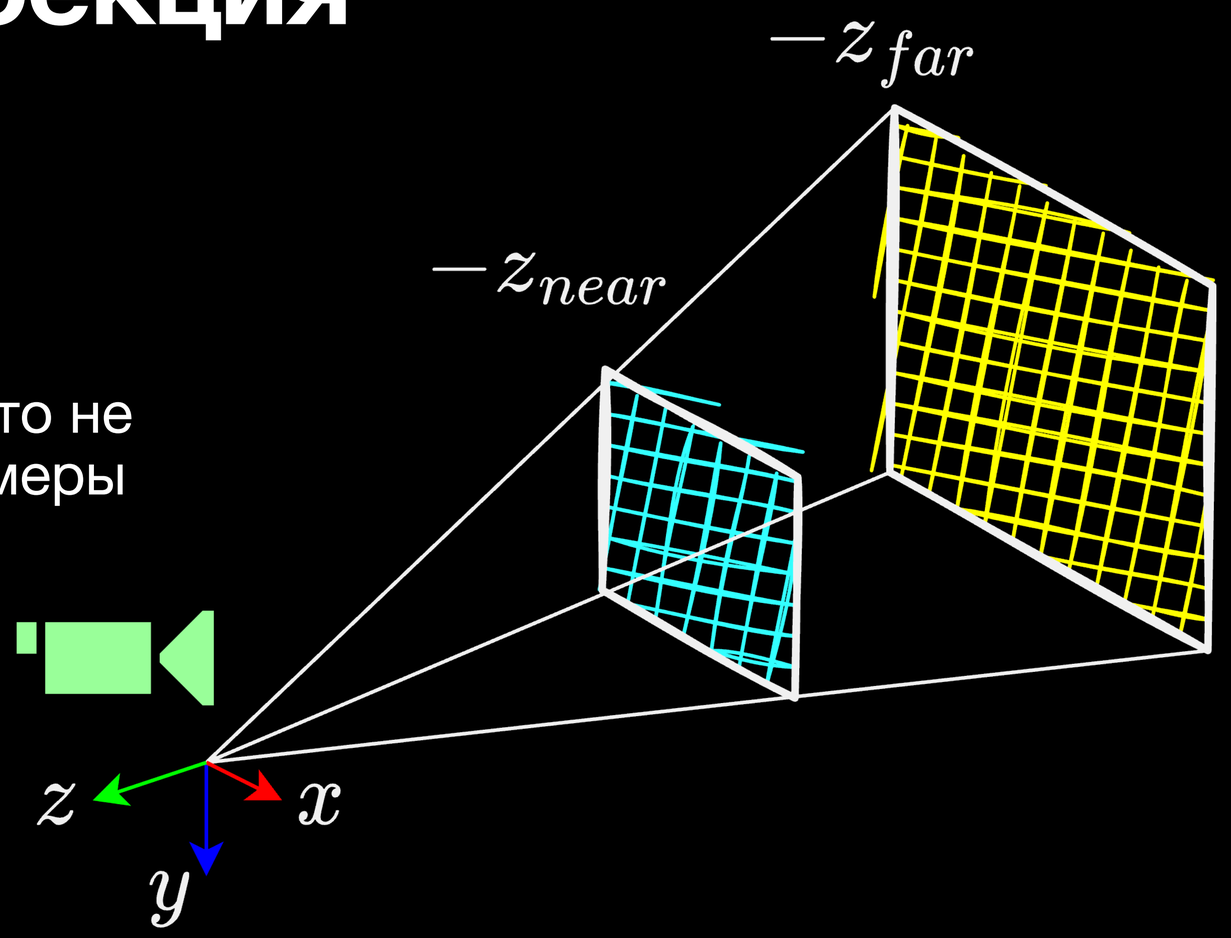
Перспективная проекция
OpenGL. Camera Frustum

- Ось Z инвертирована.
- Присутствуют clip-плоскости. Все что не попадает в усеченную пирамиду камеры игнорируется
- Нормировка глубины:

$$p_z = -z_{near} \longmapsto p_z = -1$$

$$p_z = -z_{far} \longmapsto p_z = 1$$

- Нормировка положения: $p_x, p_y \in [-1,1]$ (деление на W, H)



Перспективная проекция

OpenGL. Camera Frustum

В матрицу проекции добавились параметры α и β:

$$\hat{\mathbf{p}} = \begin{pmatrix} p_x \frac{f}{W} \\ p_y \frac{f}{H} \\ \alpha p_z + \beta \\ -p_z \end{pmatrix} = \begin{pmatrix} \frac{f}{W} & 0 & 0 & 0 \\ 0 & \frac{f}{H} & 0 & 0 \\ 0 & 0 & \alpha & \beta \\ 0 & 0 & -1 & 0 \end{pmatrix} \times \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

СЛАУ граничных условий

$$-\alpha z_{near} + \beta = -z_{near}$$
$$-\alpha z_{far} + \beta = z_{far}$$

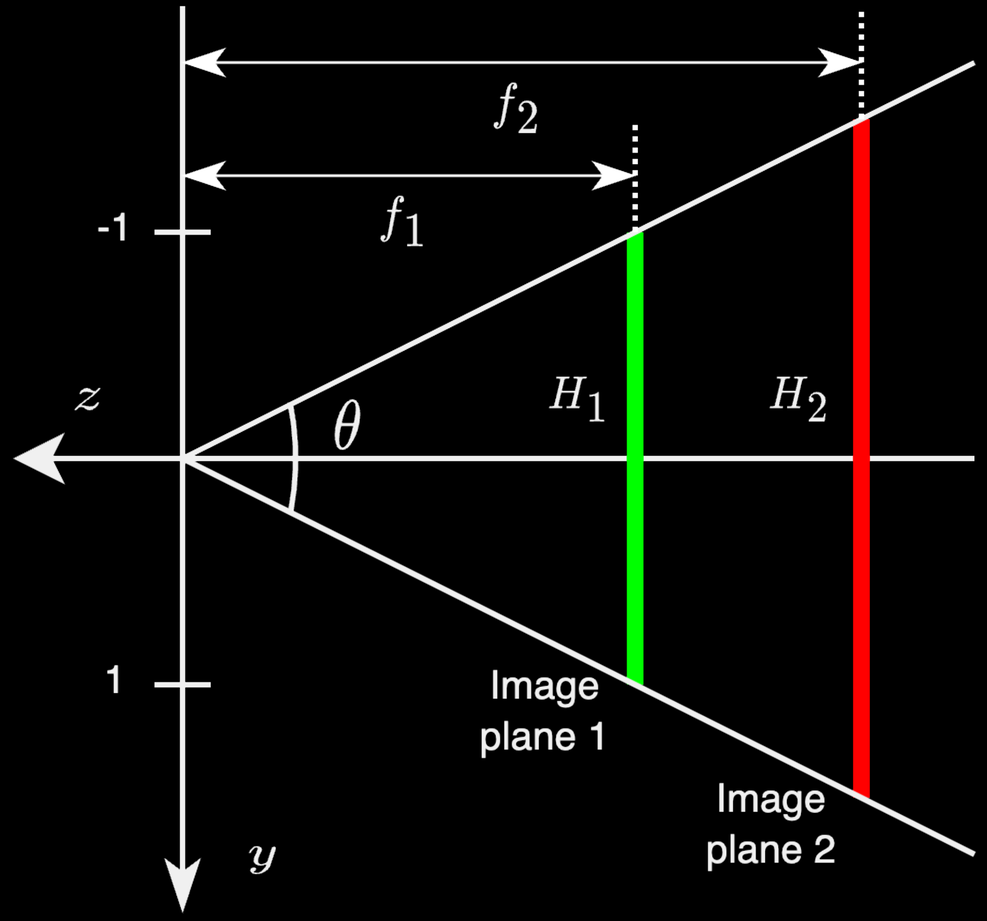
$$\alpha = \frac{z_{far} + z_{near}}{z_{near} - z_{far}}$$

$$\beta = \frac{2 z_{far} z_{near}}{z_{near} - z_{far}}$$

Перспективная проекция

OpenGL. Camera Frustum

- Изображение однозначно задается соотношениями фокального расстояния и размера изображения
- $\frac{f}{W}, \frac{f}{H}$ определяются fov-углом (θ)
- Изображения с одинаковым fov идентичны с точностью до масштаба. Конвенциально выбрали единичный масштаб



Перспективная проекция

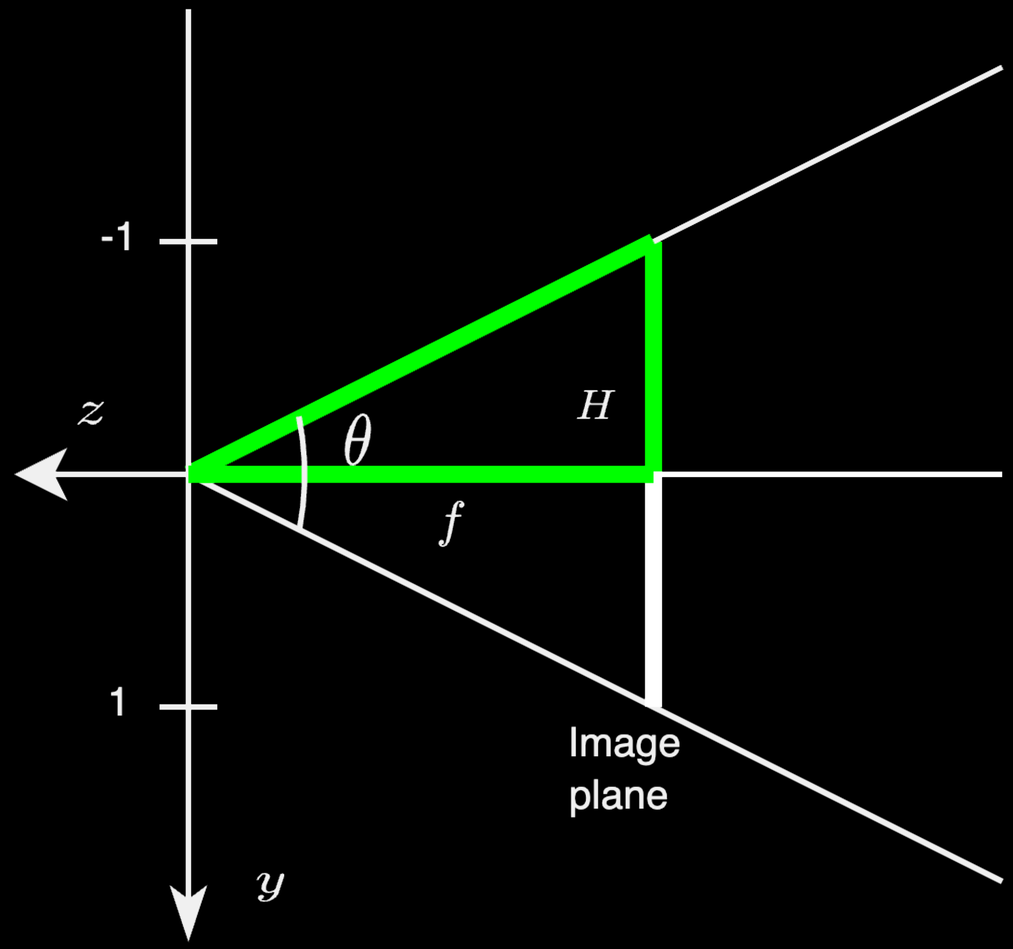
OpenGL. Camera Frustum

Из треугольника находим:

$$\tan\left(\frac{\theta_y}{2}\right) = \frac{H}{f}, \quad \tan\left(\frac{\theta_x}{2}\right) = \frac{W}{f}, \quad a = \frac{H}{W}$$

Матрица проекции $\mathbb{P} : \mathbb{H}^3 \mapsto \mathbb{H}^3$

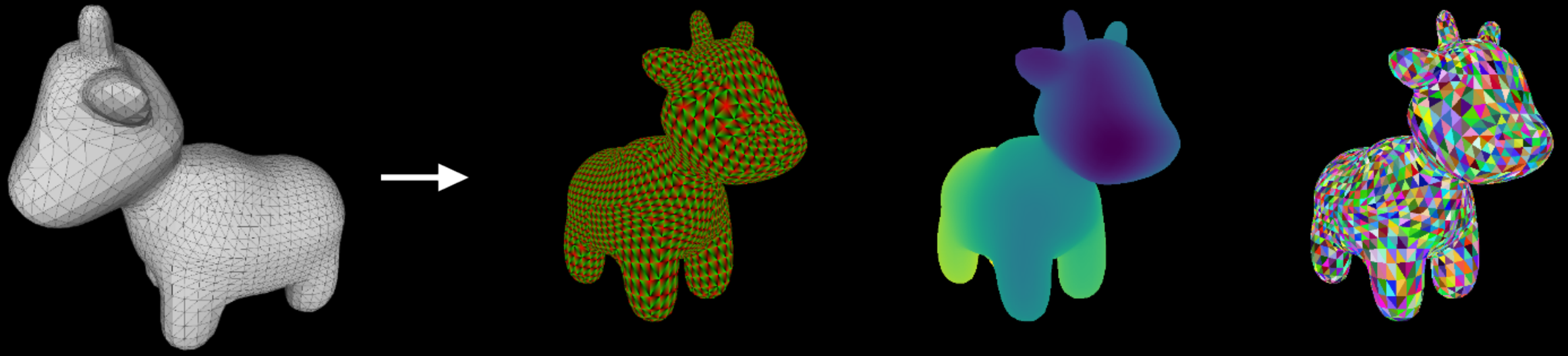
$$\mathbb{P} = \begin{pmatrix} \frac{1}{a\tan\left(\frac{\theta_y}{2}\right)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\left(\frac{\theta_y}{2}\right)} & 0 & 0 \\ 0 & 0 & \frac{z_{far}+z_{near}}{z_{near}-z_{far}} & \frac{2z_{far}z_{near}}{z_{near}-z_{far}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$



Вершинные преобразования

Основное

- Все операции происходят в однородных координатах
- Вершины преобразуются из model-space в clip-space для последующей растеризации
- Для перехода в clip-space используется MVP матрица.
- Перспективная матрица определяется clip плоскостями, одна из которых это проекционный экран
- Операции вершинных преобразований непрерывны и дифференцируемы.



Mesh (Vertices + Indices)

Barycentric coordinates

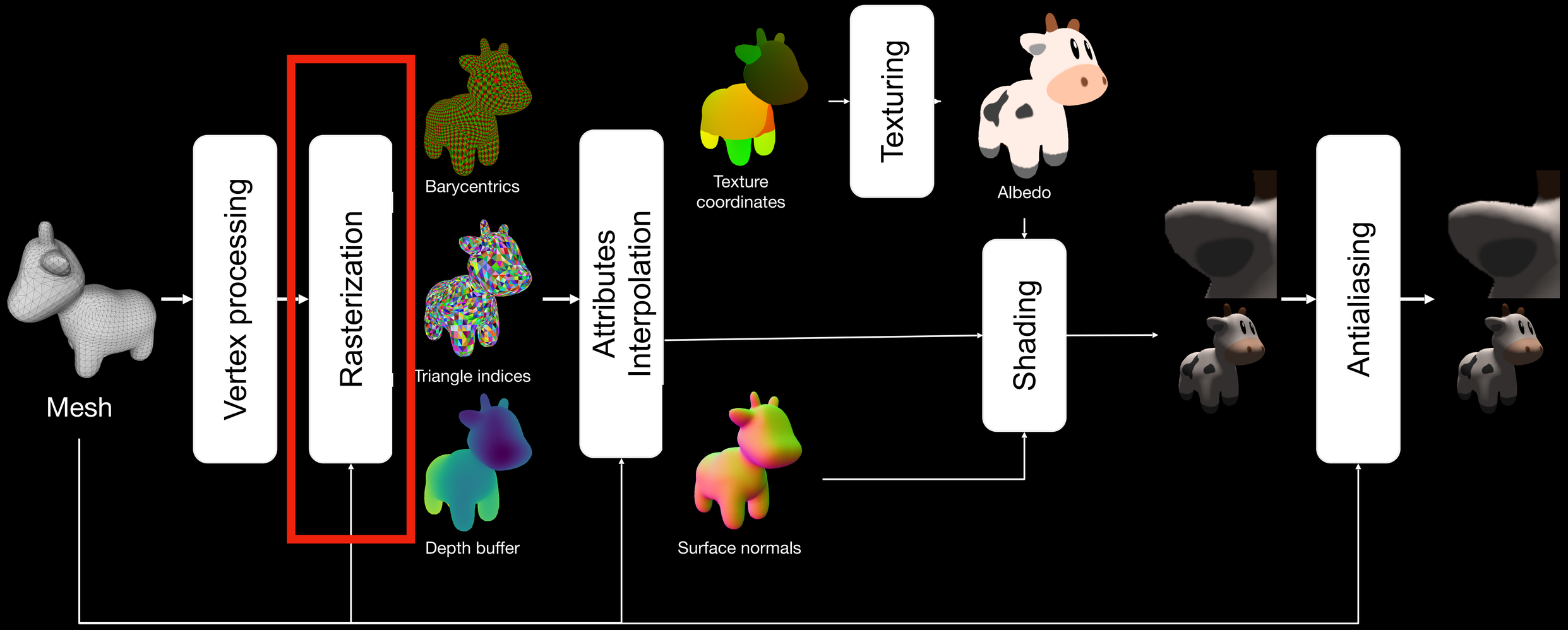
Depth buffer

Triangle indices

Растеризация

Rasterization

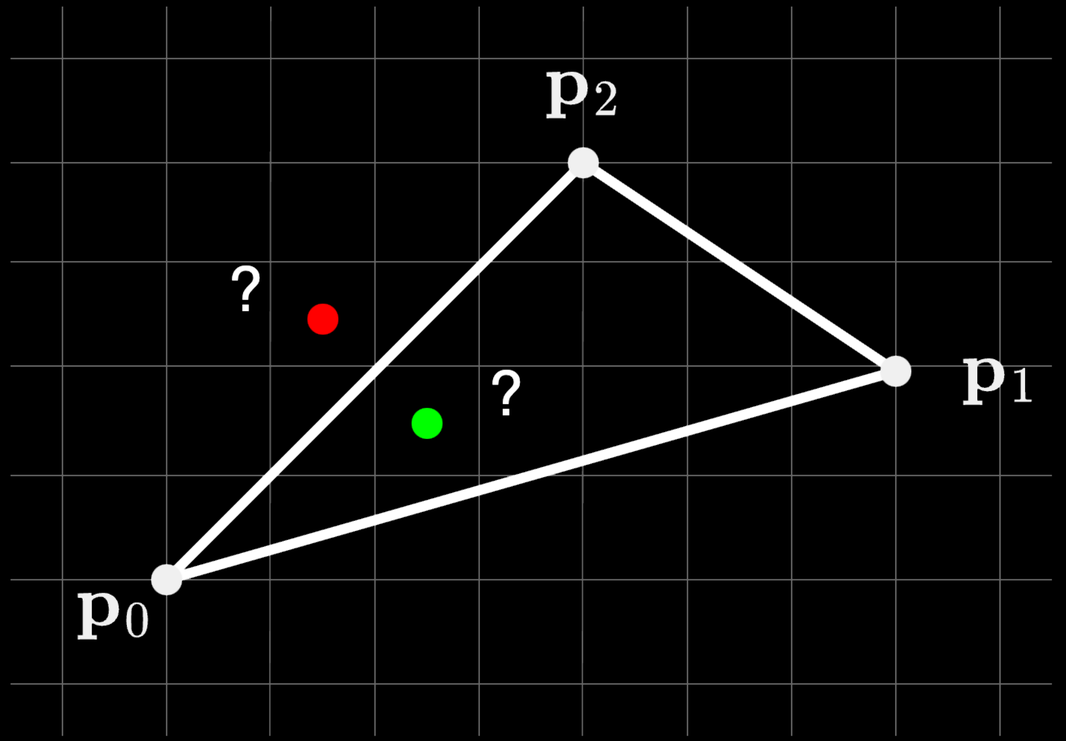
Общий пайплайн



Растеризация

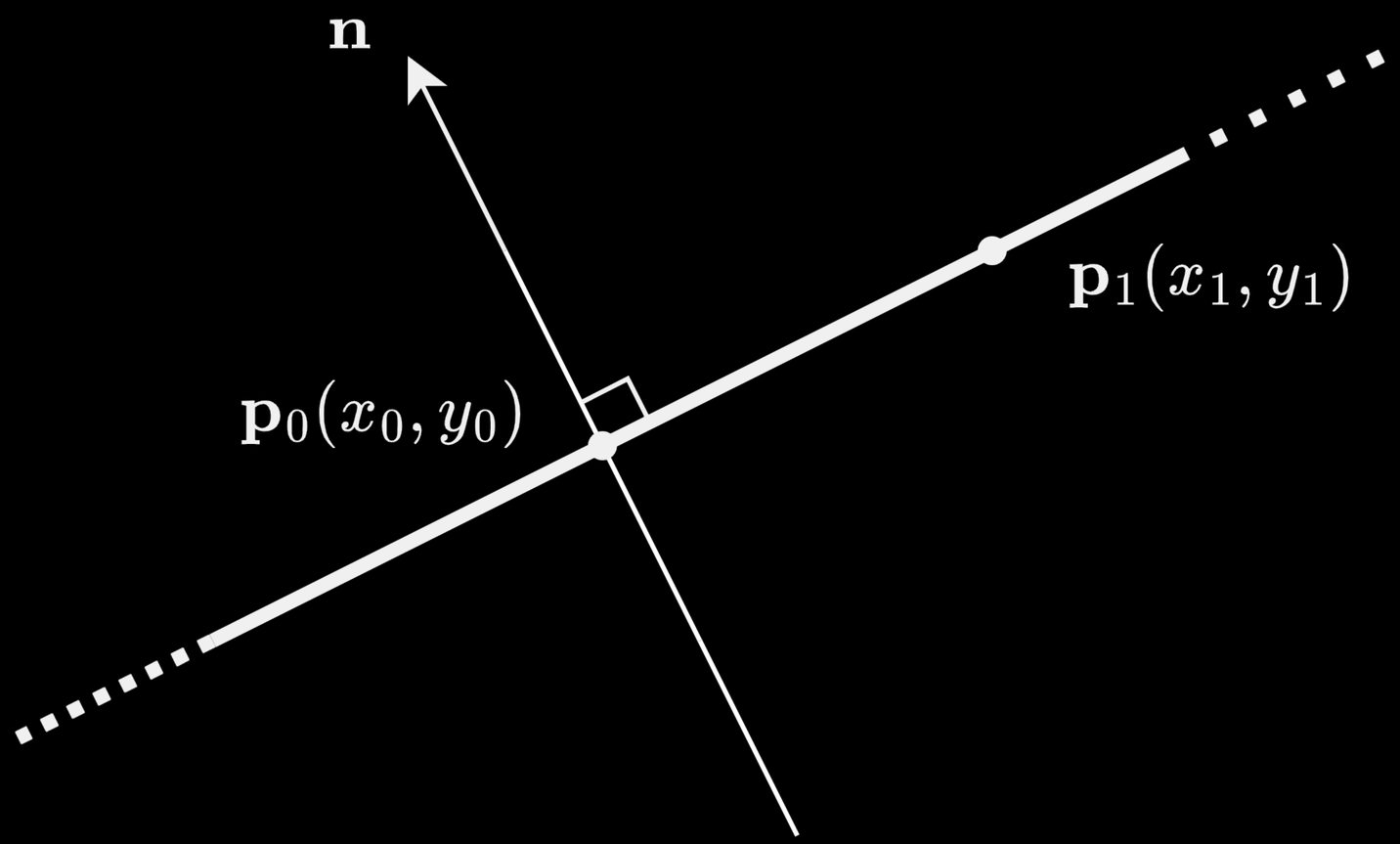
Основная идея

- Растеризация принимает на вход треугольный меш и соотносит каждому пикселю на экране не более одного треугольника.
- Тест покрытия пикселя треугольником строится на основе трех уравнений ребер
- Тест глубины сортирует треугольники по буферу глубины
- Для каждого пикселя рассчитываются барицентрический координаты на треугольнике (+ производные по ним по screen-space координатам, опционально)



Растеризация

Тест покрытия. Edge function.



Уравнение ребра — это прямая

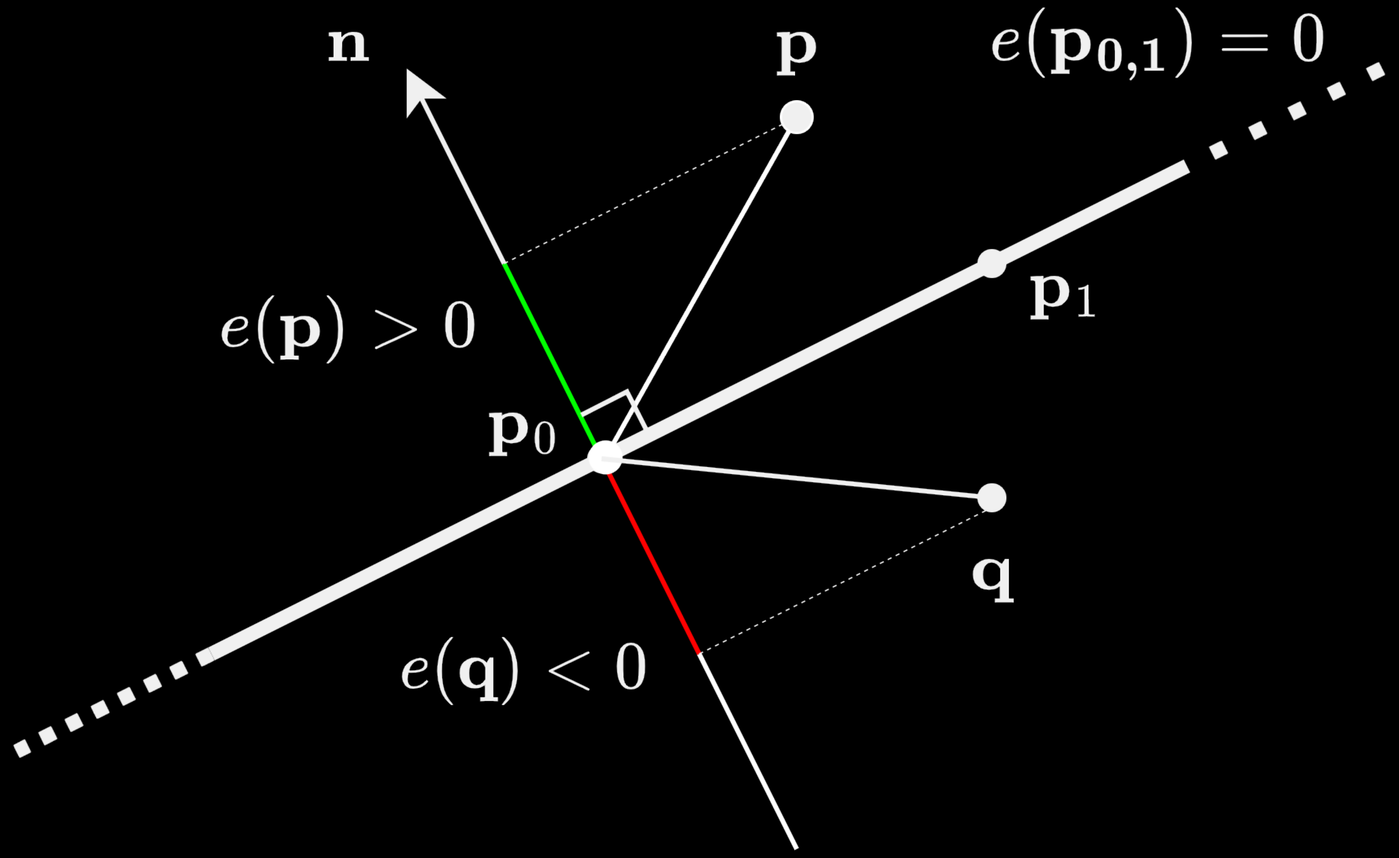
$$e(x, y) = ax + by + c$$

Она проходит через две вершины $\mathbf{p}_0, \mathbf{p}_1$

$$e(x, y) = \det \begin{vmatrix} x & y & 1 \\ x_0 & y_0 & 1 \\ x_1 & y_1 & 1 \end{vmatrix}$$

Растеризация

Тест покрытия. Edge function.



В векторной форме:

$$e(\mathbf{p}) = \mathbf{n}^{\mathsf{T}}(\mathbf{p} - \mathbf{p}_0)$$

$$\mathbf{n} = \left(y_0 - y_1, x_1 - x_0\right)^{\mathsf{T}}$$

$$\|\mathbf{n}\| = \|\mathbf{p}_0 - \mathbf{p}_1\|$$

Ориентация нормали зависит от порядка вершин

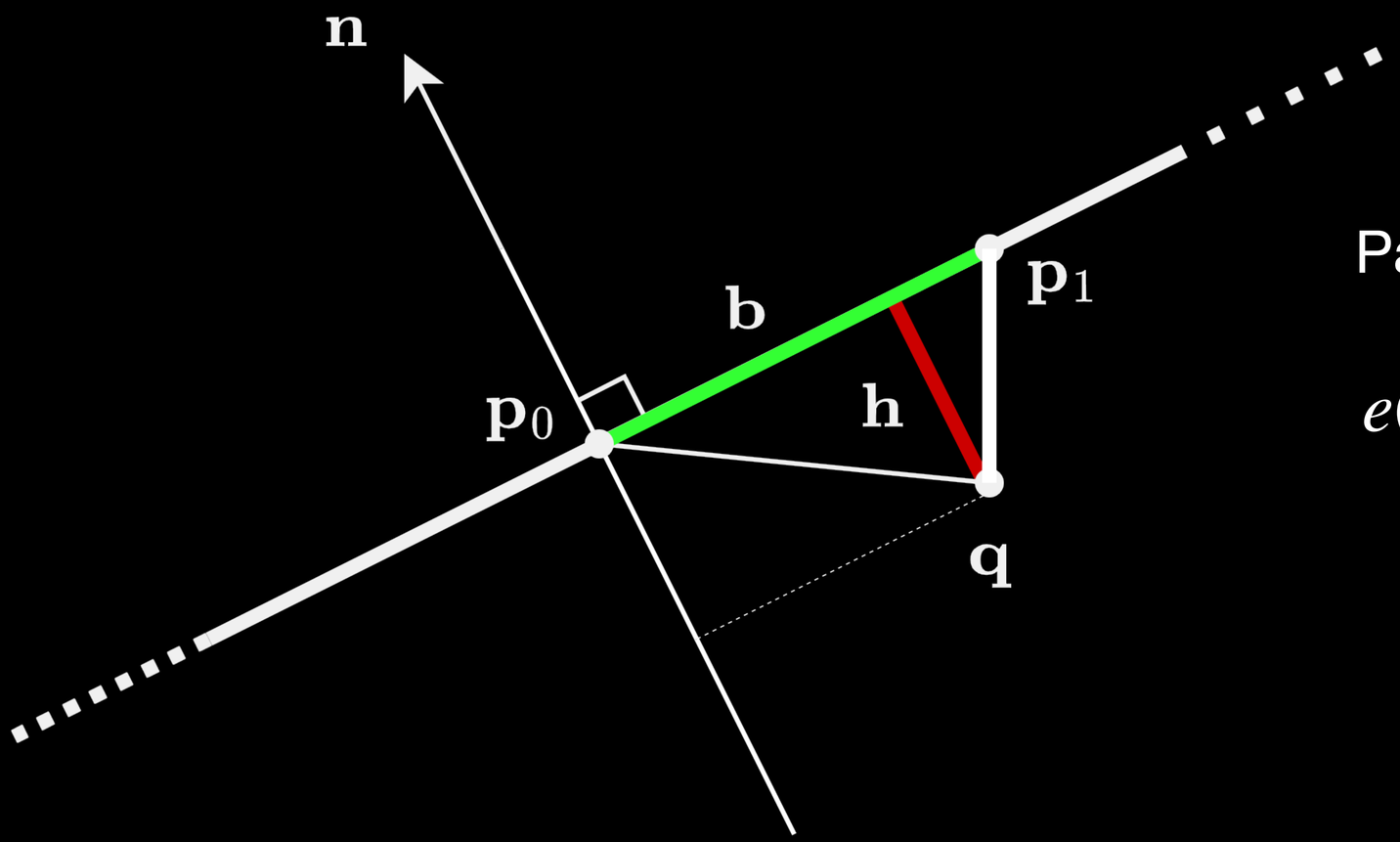
Функция $e(\mathbf{p})$ знаковая!

$$e(\mathbf{p}) = \|\mathbf{p}_0 - \mathbf{p}_1\| \|\mathbf{p} - \mathbf{p}_0\| \cos\phi$$

Она меняет знак при переходе через границу

Растеризация

Тест покрытия. Edge function.



Рассмотрим треугольник $\Delta\mathbf{p}_0\mathbf{p}_1\mathbf{q}$

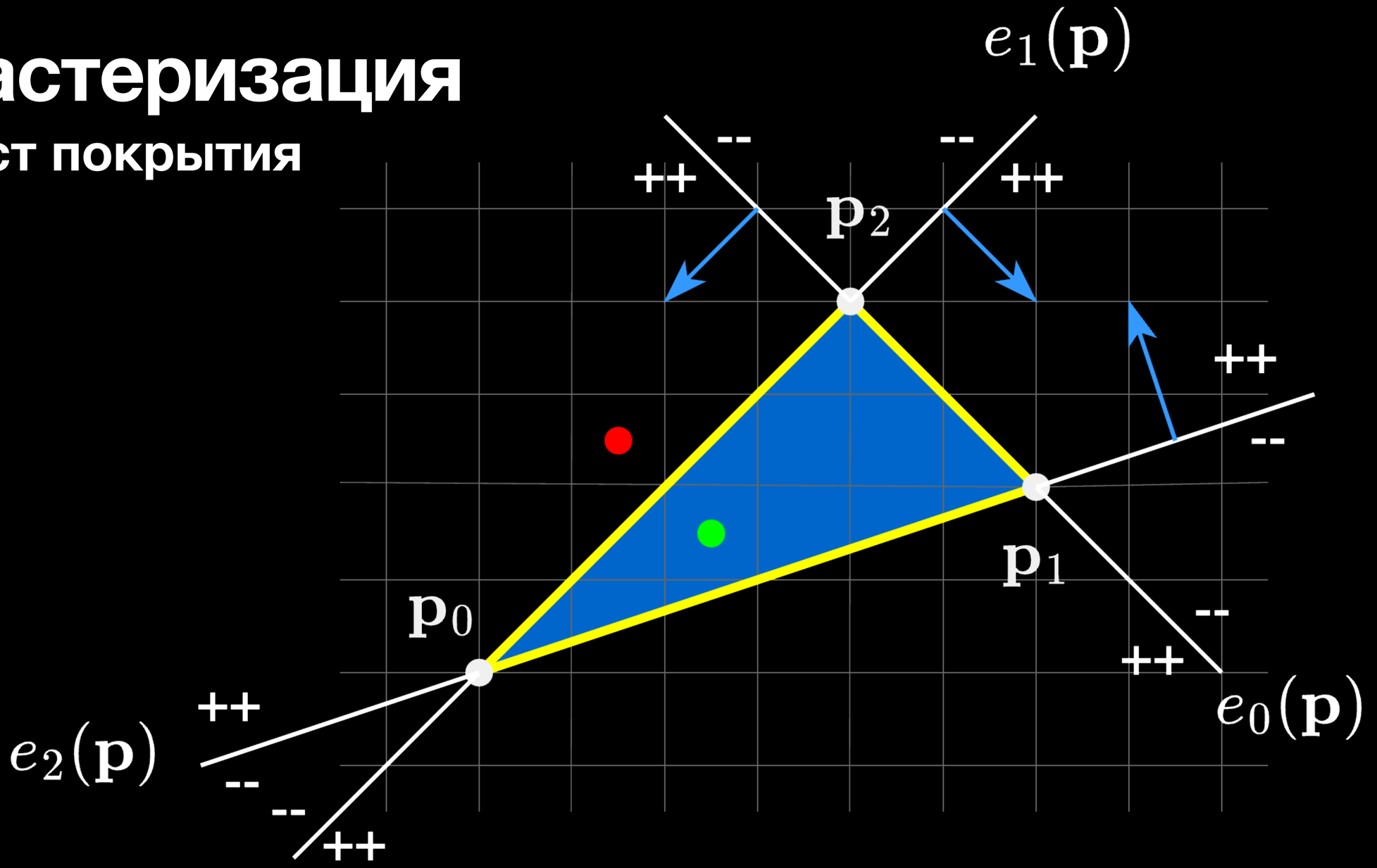
$$e(\mathbf{p}) = \underbrace{\|\mathbf{p}_0 - \mathbf{p}_1\|}_{b} \underbrace{\|\mathbf{p} - \mathbf{p}_0\| \cos\phi}_{h}$$

Модуль $e(\mathbf{p})$ равен удвоенной площади этого треугольника:

$$|e(\mathbf{p})| = 2 \times \frac{1}{2} bh = 2A_{\Delta\mathbf{p}_0\mathbf{p}_1\mathbf{q}}$$

Растеризация

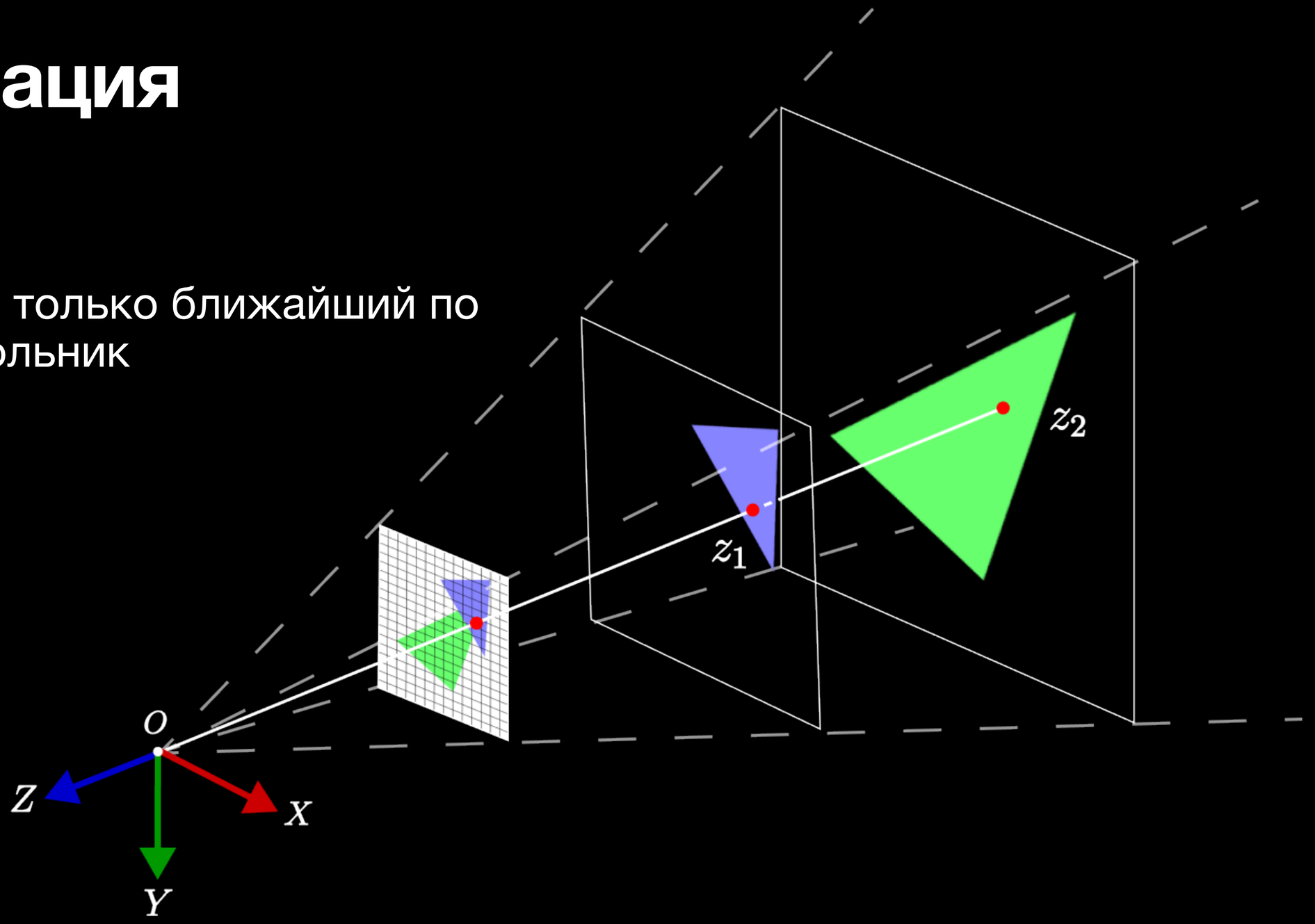
Тест покрытия



Растеризация

Тест глубины

- Растеризуется только ближайший по глубине треугольник



Растеризация

Барицентрические координаты

Барицентрики u, v, w:

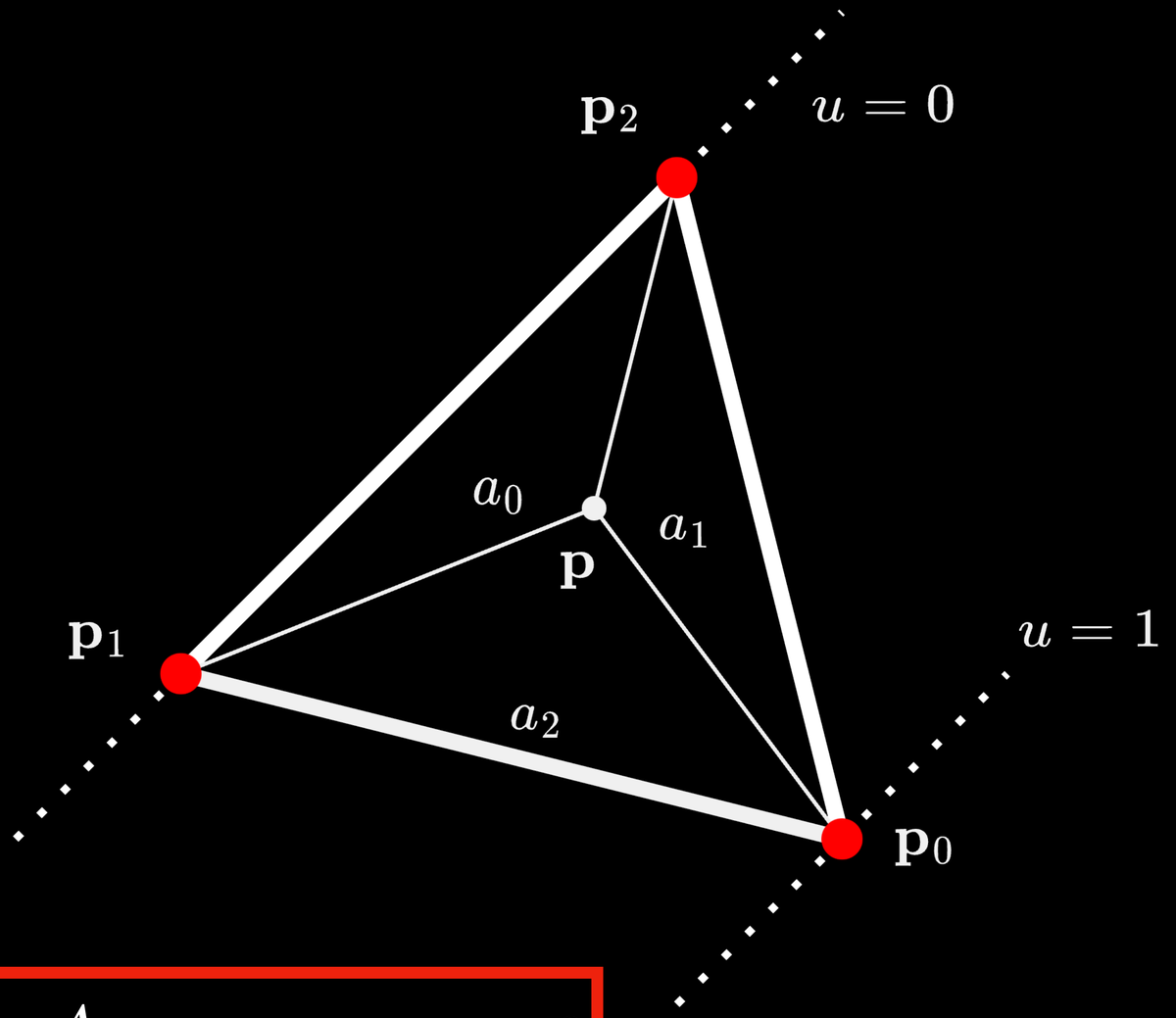
1. $u, v, w > 0$
2. $u + v + w = 1$

$$u(x,y) = \frac{a_0}{A_\Delta} = \frac{e_0(x,y)}{2A_\Delta}$$

$$v(x,y) = \frac{a_1}{A_\Delta} = \frac{e_1(x,y)}{2A_\Delta}$$

$$w = 1 - u - v$$

Площадь треугольника A_Δ не зависит от $\mathbf{p}$

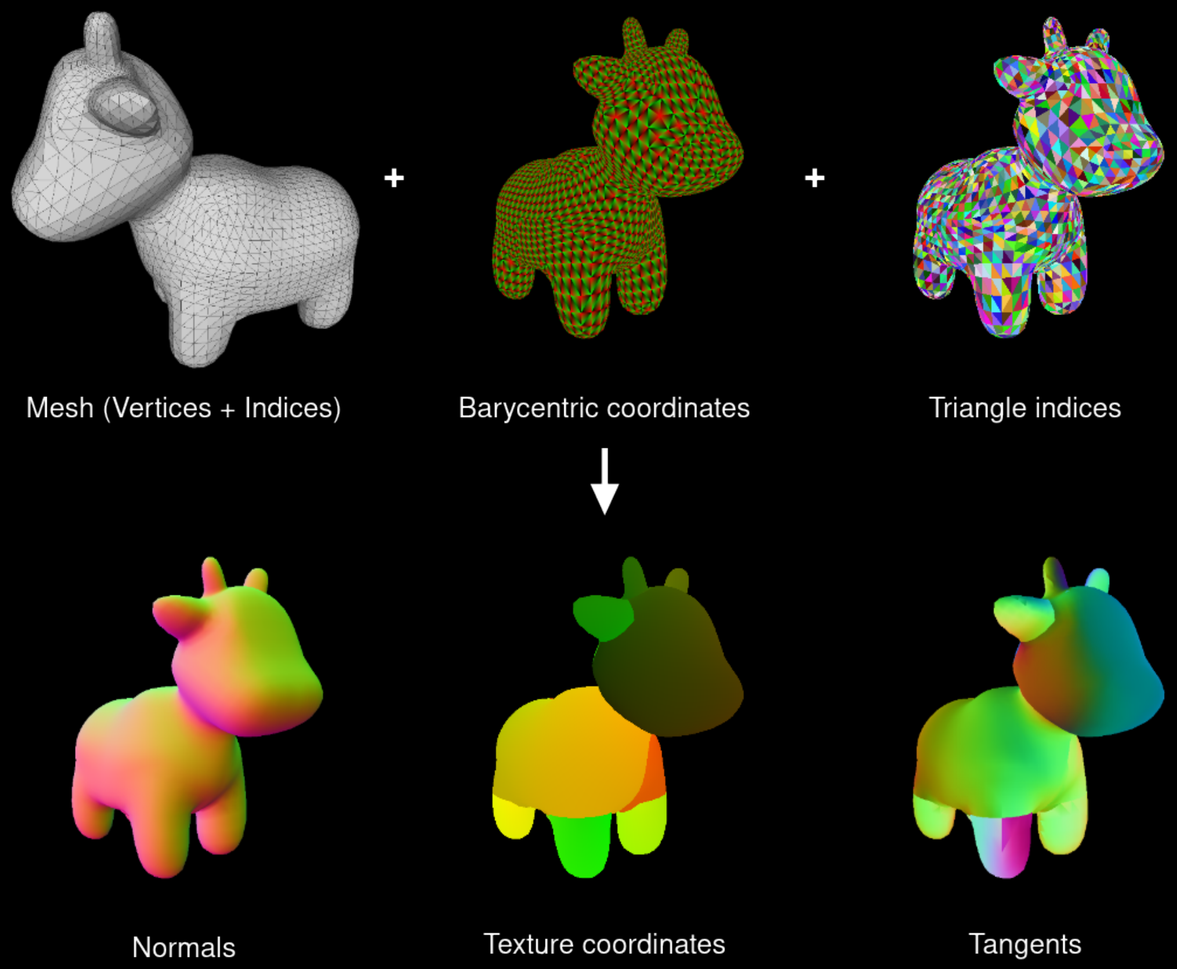


Растеризация

Основное

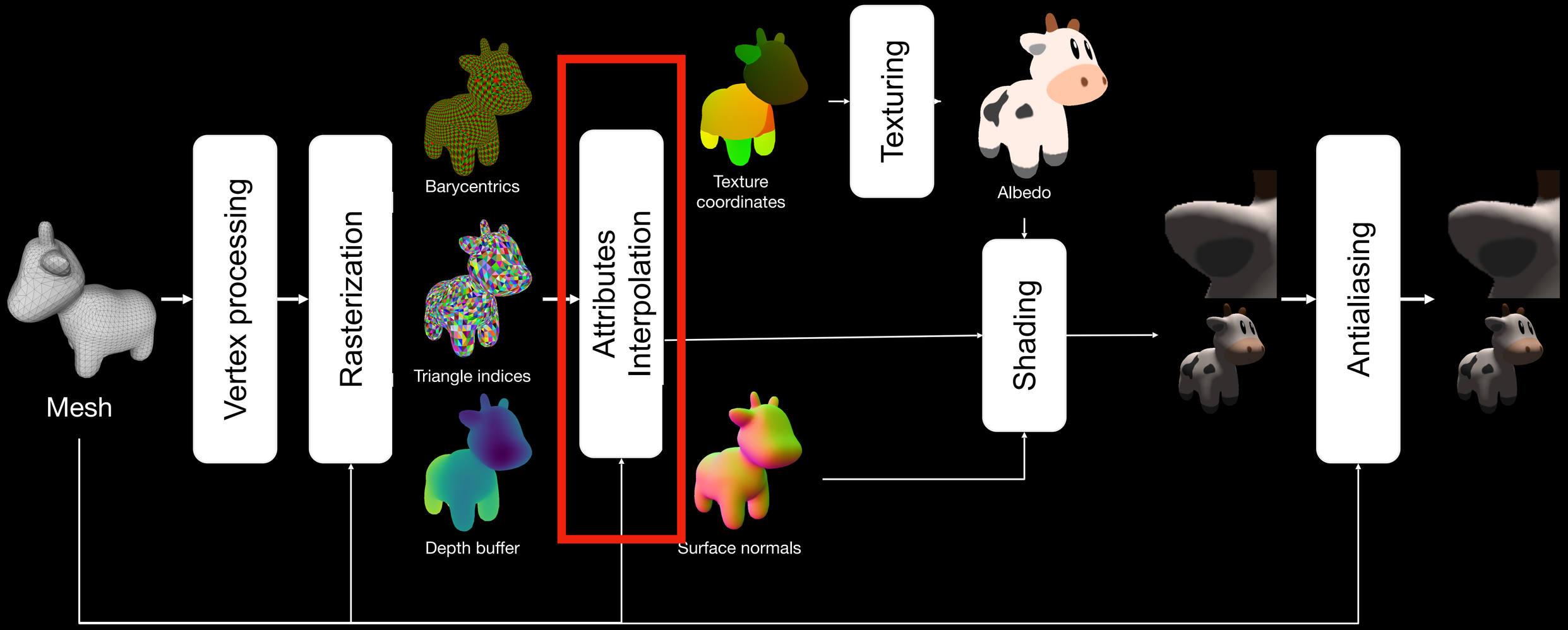
- Мы ввели понятие edge function
- Растеризация для каждого пикселя рассчитывает индекс ближайшего треугольника, покрывающего его, и барицентрический координаты этого пикселя в треугольнике
- Барицентрики непрерывно и дифференцируемое зависят от вершин меша

Интерполяция



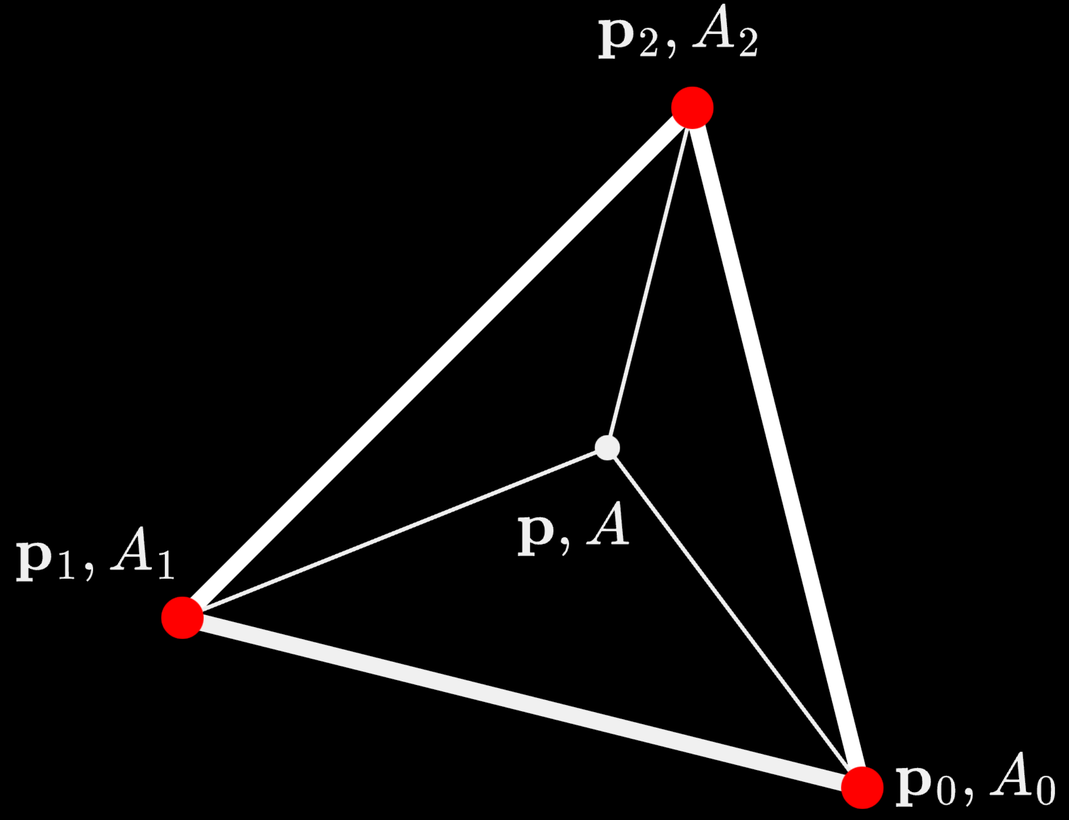
Rasterization

Общий пайплайн



Интерполяция

Наивный подход. 2D screen-space



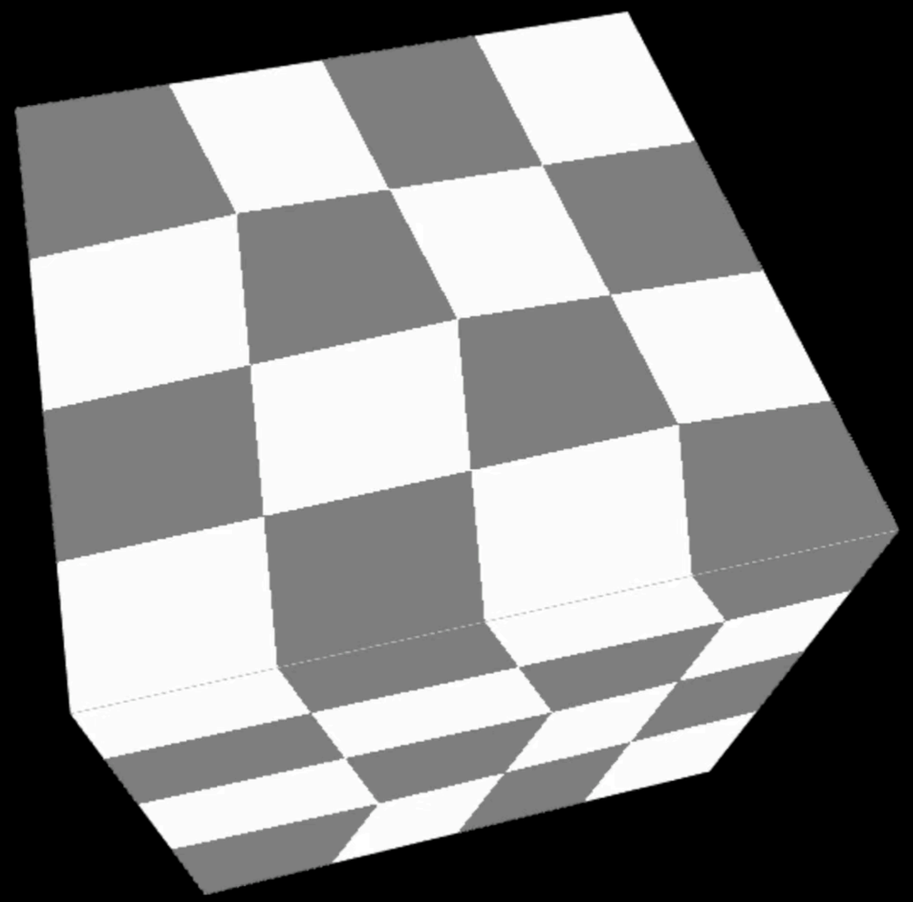
$\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2$ — точки на экране

A_0, A_1, A_2 — некоторые атрибуты точек

Интерполяция:

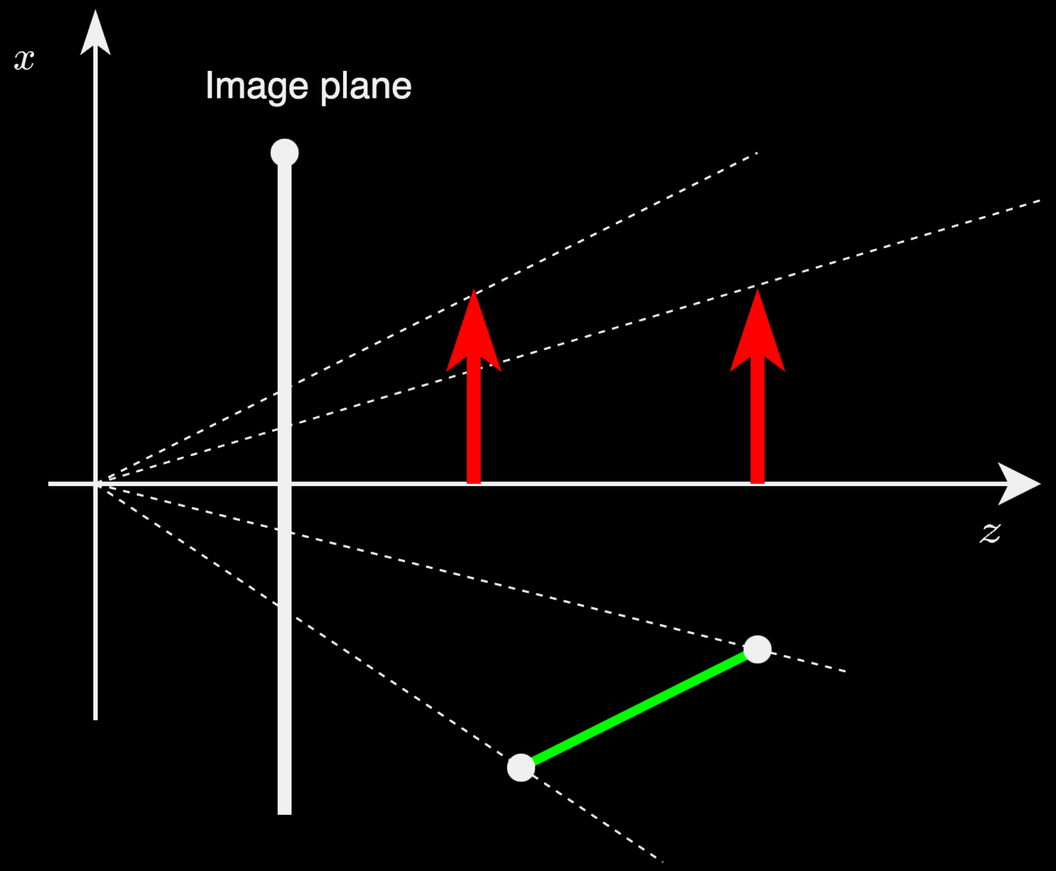
$$A = uA_0 + vA_1 + (1 - u - v)A_2$$

Попробуем использовать….



Интерполяция

Перспективные искажения



- Объекты одного размера на разном удалении проецируются неодинаково (красная стрелка)
- Перспективные искажения проявляются так же при проецирование одного объекта расположенного под углом к плоскости экрана (зеленый отрезок)

Интерполяция

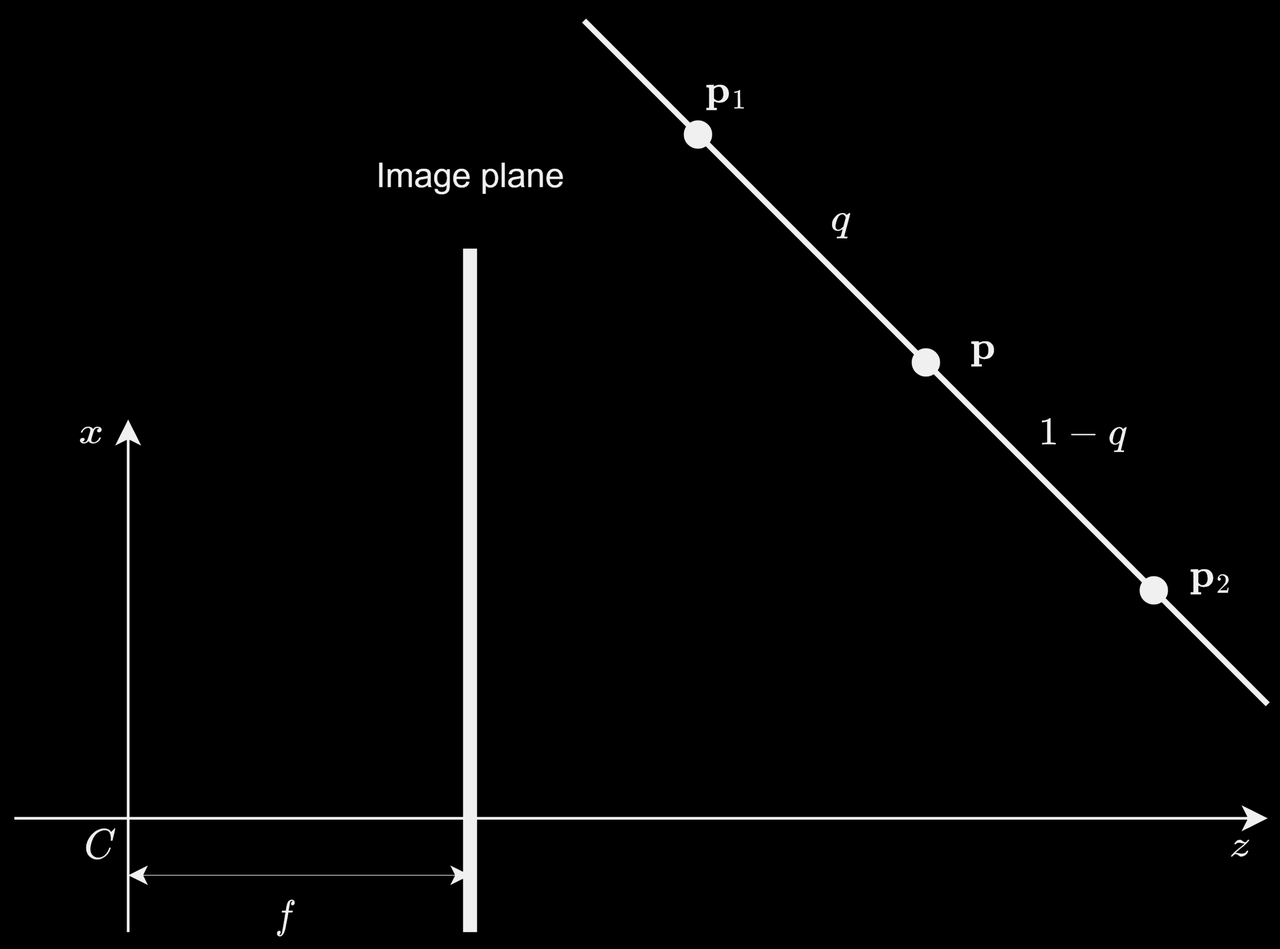
Перспективная коррекция

Рассмотрим случай плоской (XZ) перспективной линейной интерполяции вдоль прямой проходящей через

$\mathbf{p}_1, \mathbf{p}_2$

Коэф-т интерполяции q:

$$\begin{cases} \mathbf{p}_x = \mathbf{p}_x^1 + q(\mathbf{p}_x^2 - \mathbf{p}_x^1) \\ \mathbf{p}_z = \mathbf{p}_z^1 + q(\mathbf{p}_z^2 - \mathbf{p}_z^1) \end{cases}$$



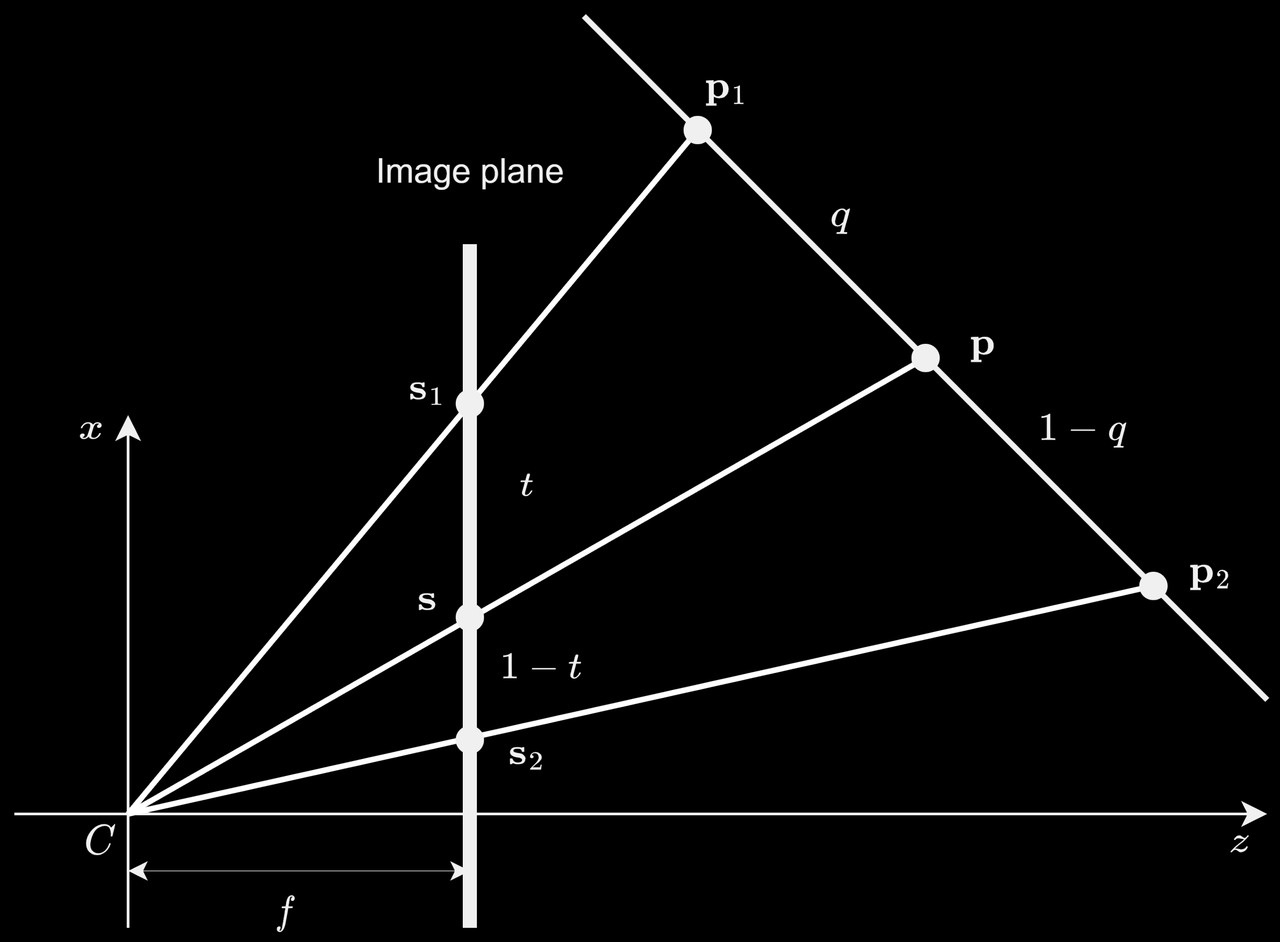
Интерполяция

Перспективная коррекция

Точки $\mathbf{p}_1, \mathbf{p}_2, \mathbf{p}$ проецируются на плоскость в точки $\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}$

Нам известен коэф-т линейной интерполяции t в плоскость экрана (это барицентрики)

$$\begin{cases} \mathbf{s}_x = \mathbf{s}_x^1 + t(\mathbf{s}_x^2 - \mathbf{s}_x^1) \\ \mathbf{p}_x = \mathbf{p}_x^1 + q(\mathbf{p}_x^2 - \mathbf{p}_x^1) \\ \mathbf{p}_z = \mathbf{p}_z^1 + q(\mathbf{p}_z^2 - \mathbf{p}_z^1) \end{cases}$$



Интерполяция

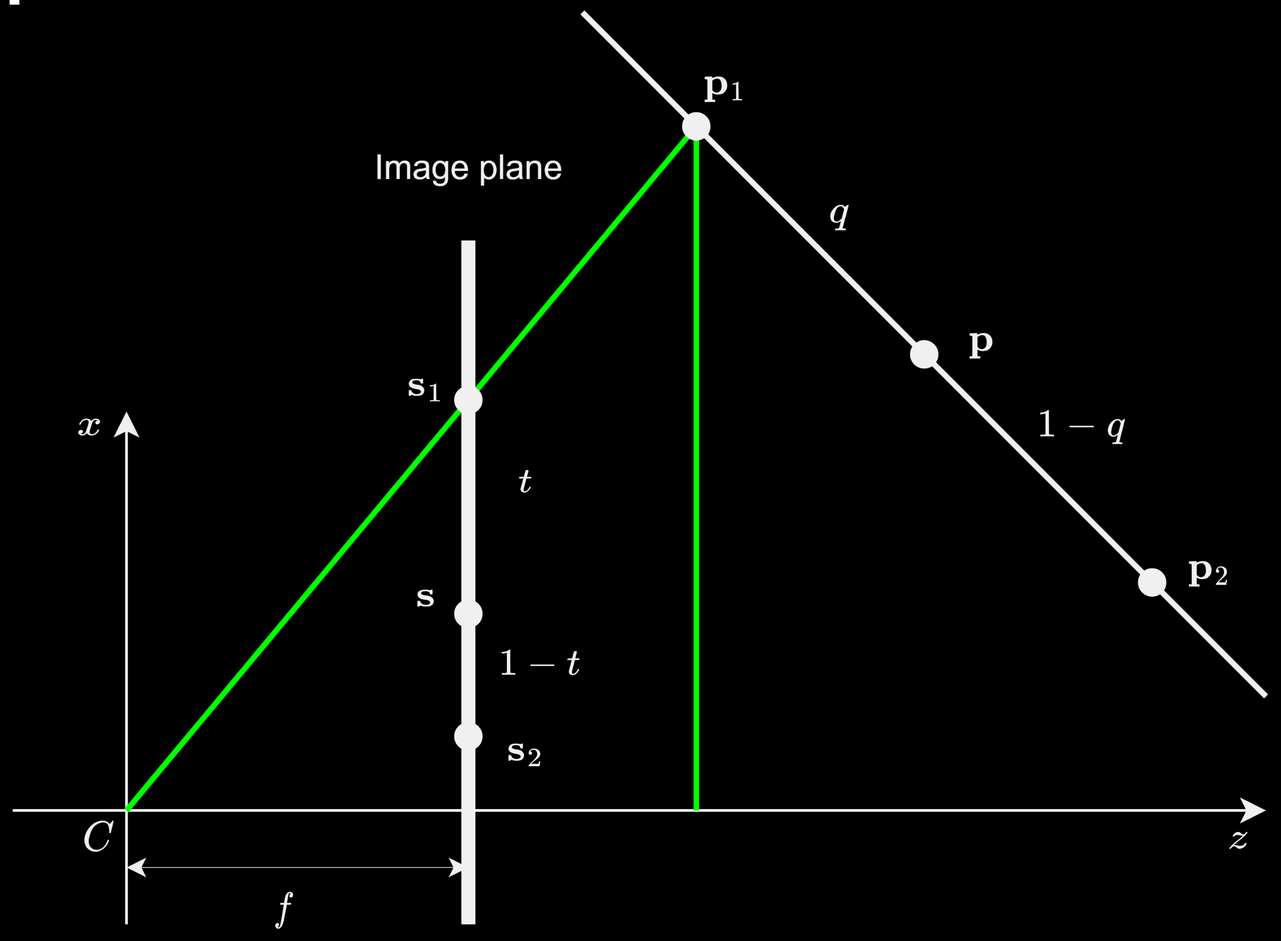
Перспективная коррекция

Подобие треугольников:

$$\frac{\mathbf{p}_x^1}{p_z^1} = \frac{\mathbf{s}_x^1}{f}$$

Интерполяция:

$$\begin{cases} \mathbf{s}_x = \mathbf{s}_x^1 + t(\mathbf{s}_x^2 - \mathbf{s}_x^1) \\ \mathbf{p}_x = \mathbf{p}_x^1 + q(\mathbf{p}_x^2 - \mathbf{p}_x^1) \\ \mathbf{p}_z = \mathbf{p}_z^1 + q(\mathbf{p}_z^2 - \mathbf{p}_z^1) \end{cases}$$



Интерполяция

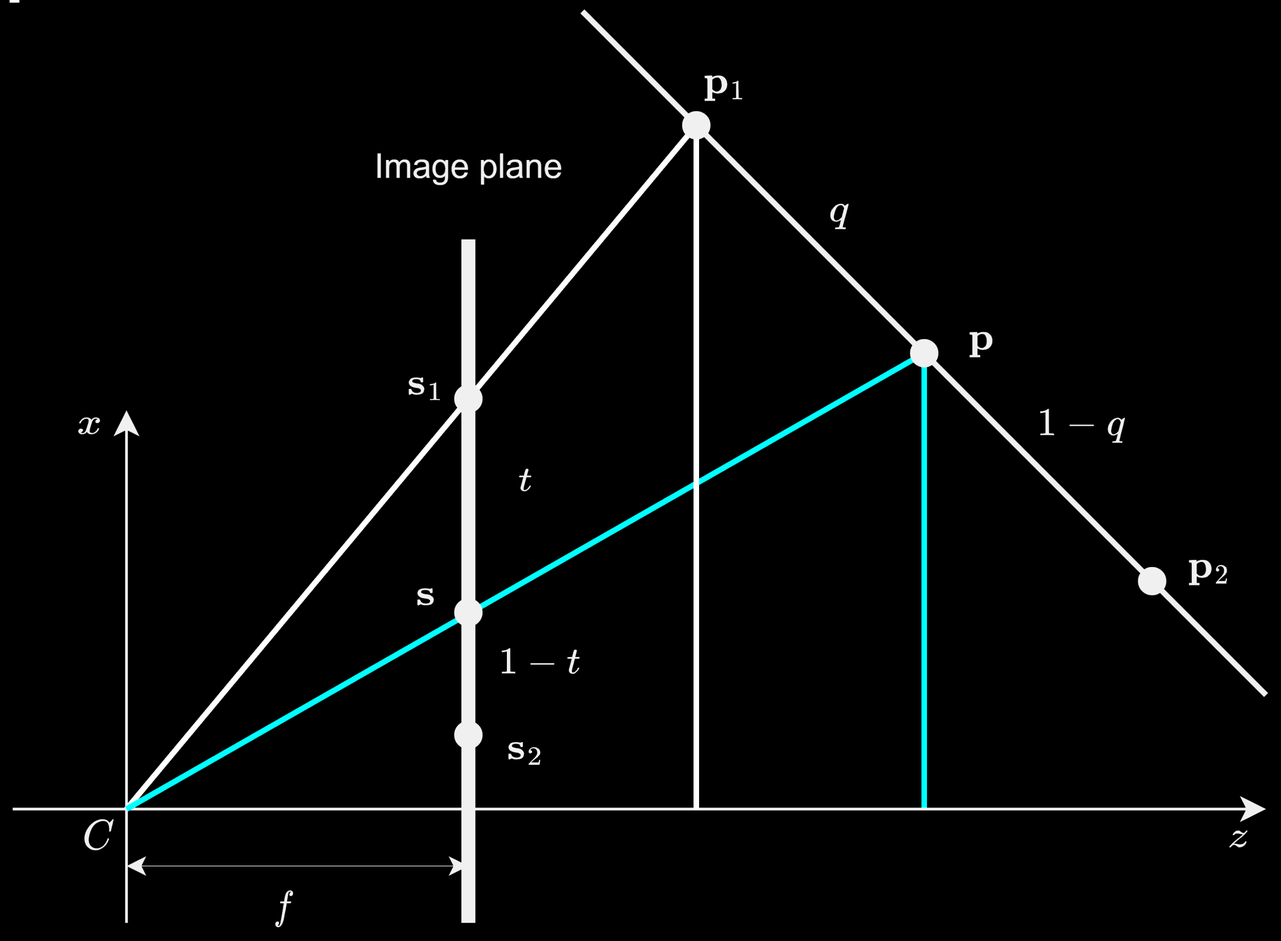
Перспективная коррекция

Подобие треугольников:

$$\frac{\mathbf{p}_x^1}{p_z^1} = \frac{\mathbf{s}_x^1}{f}, \quad \frac{\mathbf{p}_x}{p_z} = \frac{\mathbf{s}_x}{f}$$

Интерполяция:

$$\begin{cases} \mathbf{s}_x = \mathbf{s}_x^1 + t(\mathbf{s}_x^2 - \mathbf{s}_x^1) \\ \mathbf{p}_x = \mathbf{p}_x^1 + q(\mathbf{p}_x^2 - \mathbf{p}_x^1) \\ \mathbf{p}_z = \mathbf{p}_z^1 + q(\mathbf{p}_z^2 - \mathbf{p}_z^1) \end{cases}$$



Интерполяция

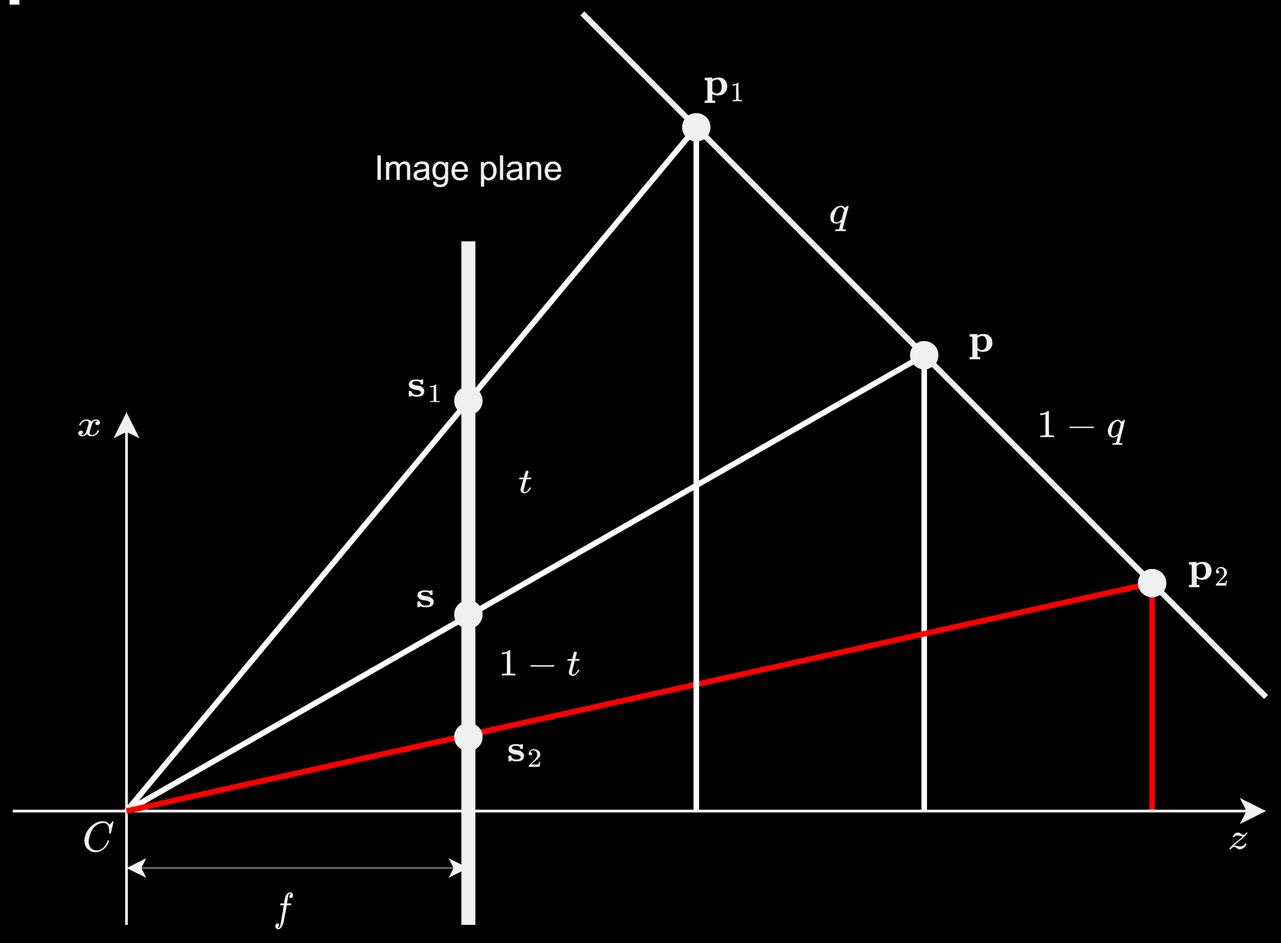
Перспективная коррекция

Подобие треугольников:

$$\frac{\mathbf{p}_x^1}{p_z^1} = \frac{\mathbf{s}_x^1}{f}, \quad \frac{\mathbf{p}_x}{p_z} = \frac{\mathbf{s}_x}{f}, \quad \frac{\mathbf{p}_x^2}{p_z^2} = \frac{\mathbf{s}_x^2}{f}$$

Интерполяция:

$$\begin{cases} \mathbf{s}_x = \mathbf{s}_x^1 + t(\mathbf{s}_x^2 - \mathbf{s}_x^1) \\ \mathbf{p}_x = \mathbf{p}_x^1 + q(\mathbf{p}_x^2 - \mathbf{p}_x^1) \\ \mathbf{p}_z = \mathbf{p}_z^1 + q(\mathbf{p}_z^2 - \mathbf{p}_z^1) \end{cases}$$



Интерполяция

Перспективная коррекция

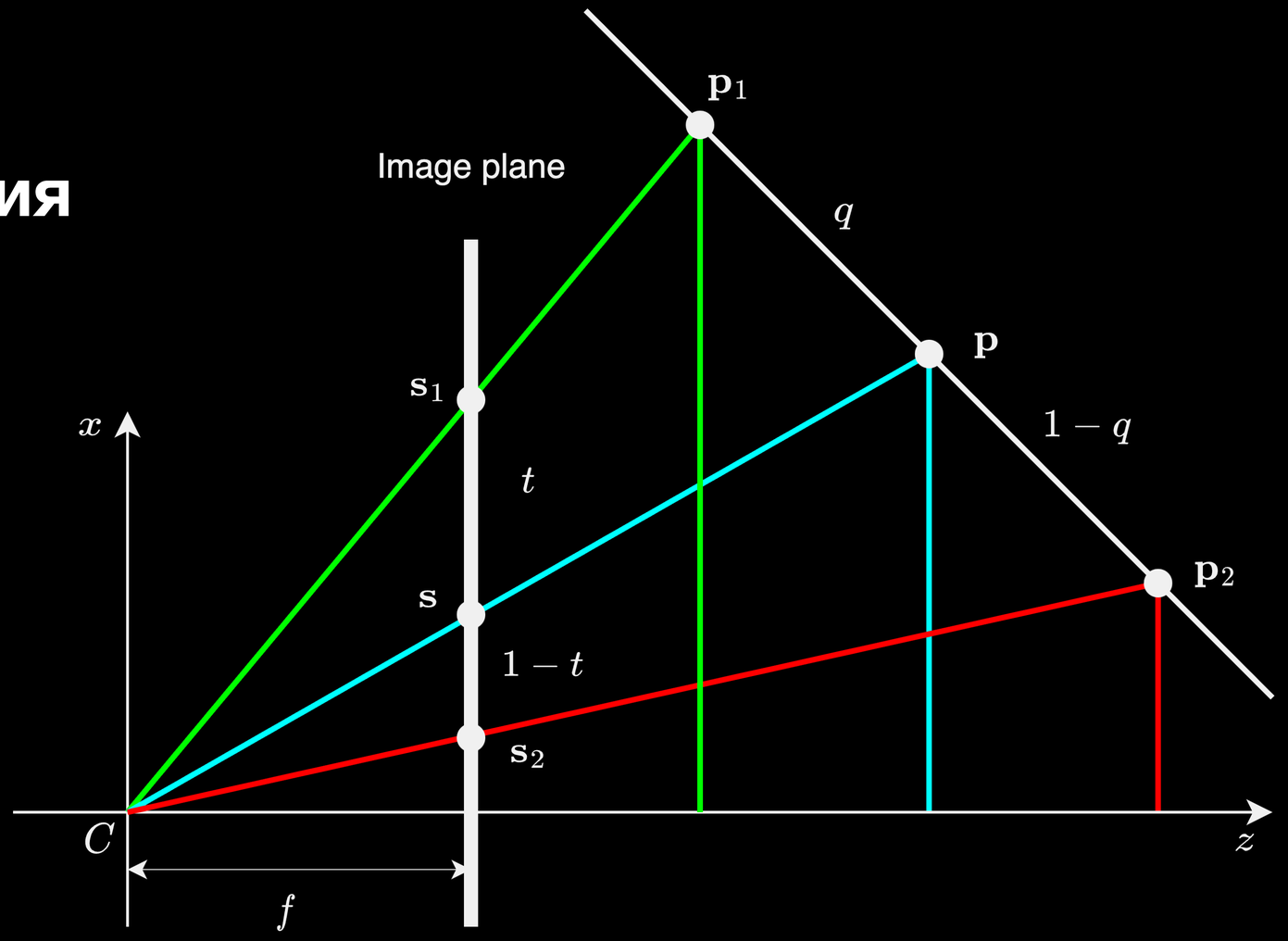
Подобие треугольников:

$$\frac{\mathbf{p}_x^1}{p_z^1} = \frac{\mathbf{s}_x^1}{f}, \quad \frac{\mathbf{p}_x^2}{p_z^2} = \frac{\mathbf{s}_x^2}{f}, \quad \frac{\mathbf{p}_x}{p_z} = \frac{\mathbf{s}_x}{f}$$

Интерполяция:

$$\begin{cases} \mathbf{s}_x = \mathbf{s}_x^1 + t(\mathbf{s}_x^2 - \mathbf{s}_x^1) \\ \mathbf{p}_x = \mathbf{p}_x^1 + q(\mathbf{p}_x^2 - \mathbf{p}_x^1) \\ \mathbf{p}_z = \mathbf{p}_z^1 + q(\mathbf{p}_z^2 - \mathbf{p}_z^1) \end{cases}$$

$$q = \frac{t\mathbf{p}_z^1}{t\mathbf{p}_z^1 + (1-t)\mathbf{p}_z^2}$$



- Коэффициент интерполяции в пространстве **нелинейно** зависит от коэффициента в плоскости!
- Точки $\mathbf{p}$ находятся в clip-space, т.е. $\mathbf{p}_z$ — это w координата

Интерполяция
Расчет атрибутов

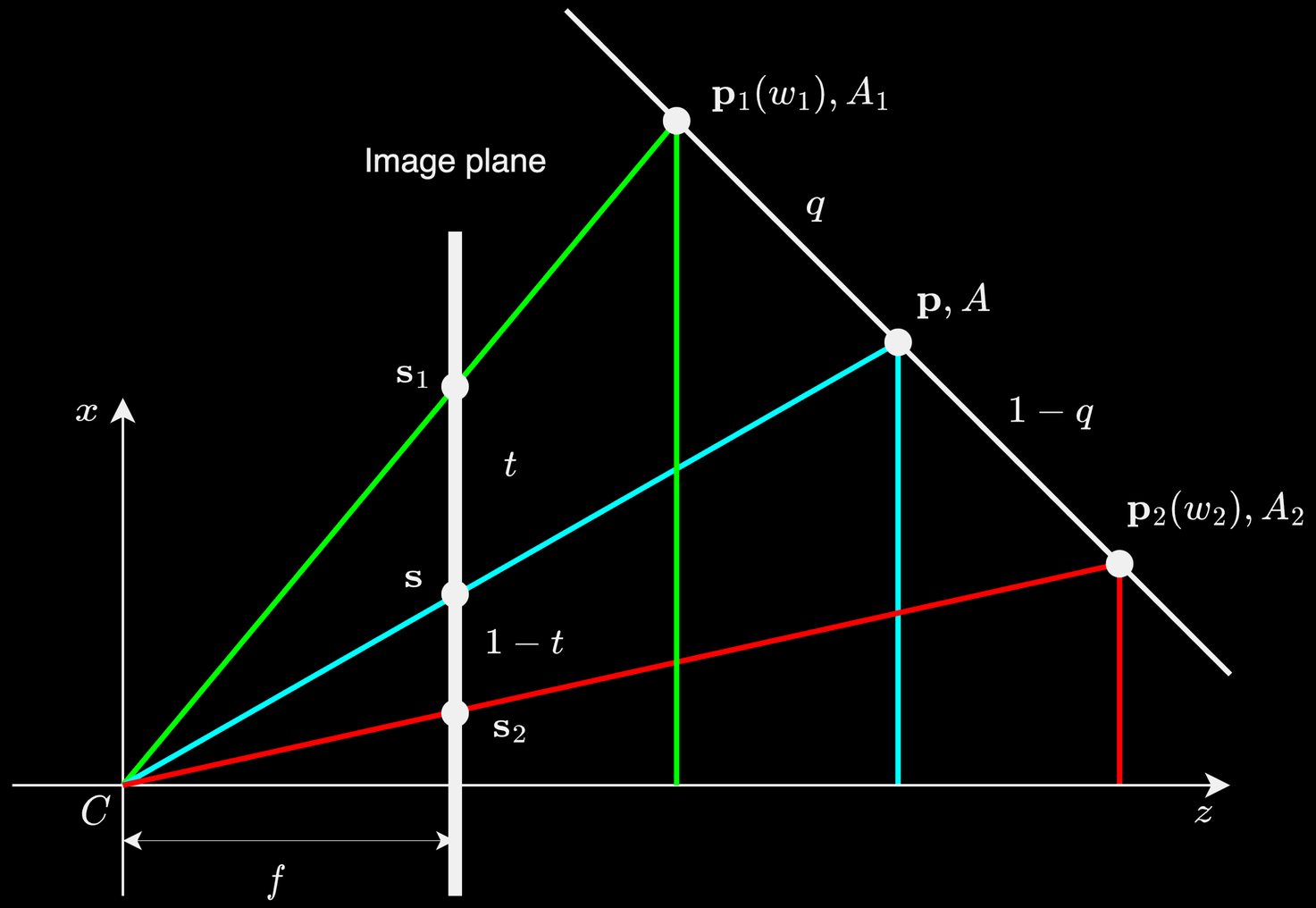
Интерполяция в clip-space:

$$A(p) = A_1 + q(A_2 - A_1)$$

$$q = \frac{tw_1}{tw_1 + (1-t)w_2}$$

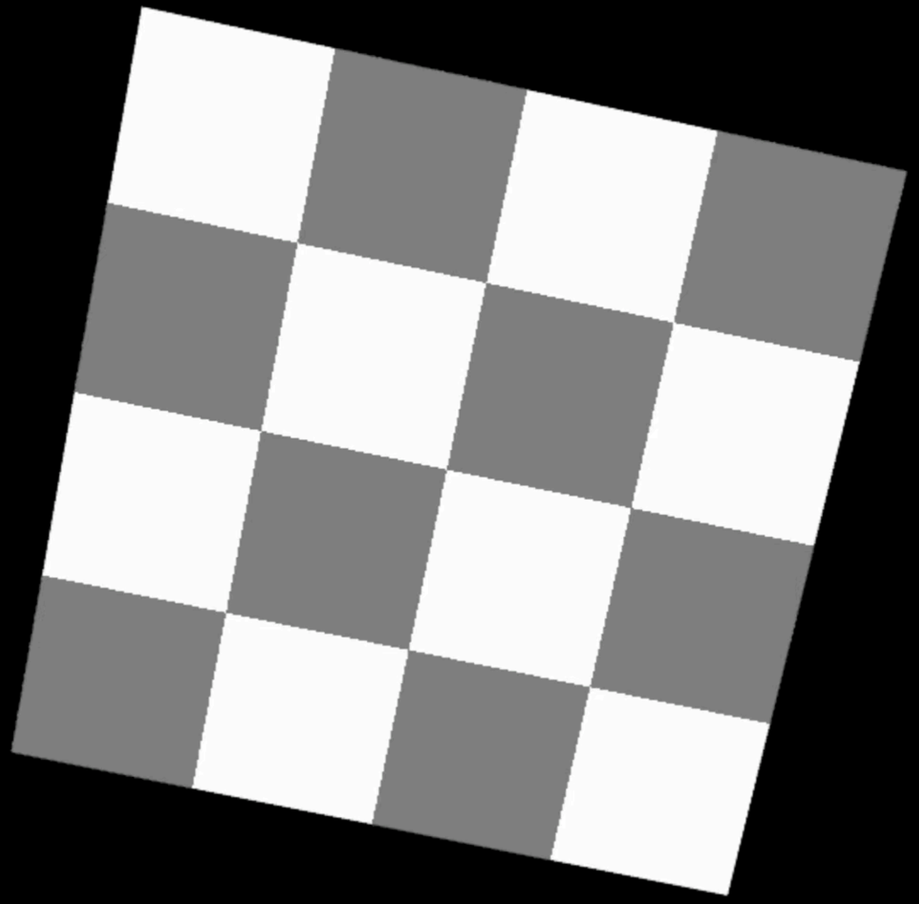
$$A(p) = \frac{\frac{A_1}{w_1} + t\left(\frac{A_2}{w_2} - \frac{A_1}{w_1}\right)}{\frac{1}{w_1} + t\left(\frac{1}{w_2} - \frac{1}{w_1}\right)}$$

$$A(p) = \frac{lerp(A/w, t)}{lerp(1/w, t)}$$



- Для корректной интерполяции атрибут необходимо использовать пространственный коэффициент сглаживания

Попробуем использовать…. Успех!



Интерполяция

Коррекция барицентриков. Интуиция

До

$$A(p) = A_1 \frac{(1-t)}{1-t+t} + A_2 \frac{t}{1-t+t}$$

После

$$A(p) = A_1 \frac{\frac{(1-t)}{w_1}}{\frac{(1-t)}{w_1} + \frac{t}{w_2}} + A_2 \frac{\frac{t}{w_2}}{\frac{(1-t)}{w_1} + \frac{t}{w_2}}$$

По аналогии

$$\hat{u}(x,y) = \frac{\frac{u}{w_0}}{\frac{u}{w_0} + \frac{v}{w_1} + \frac{(1-u-v)}{w_2}}$$

$$\hat{v}(x,y) = \frac{\frac{v}{w_1}}{\frac{u}{w_0} + \frac{v}{w_1} + \frac{(1-u-v)}{w_2}}$$

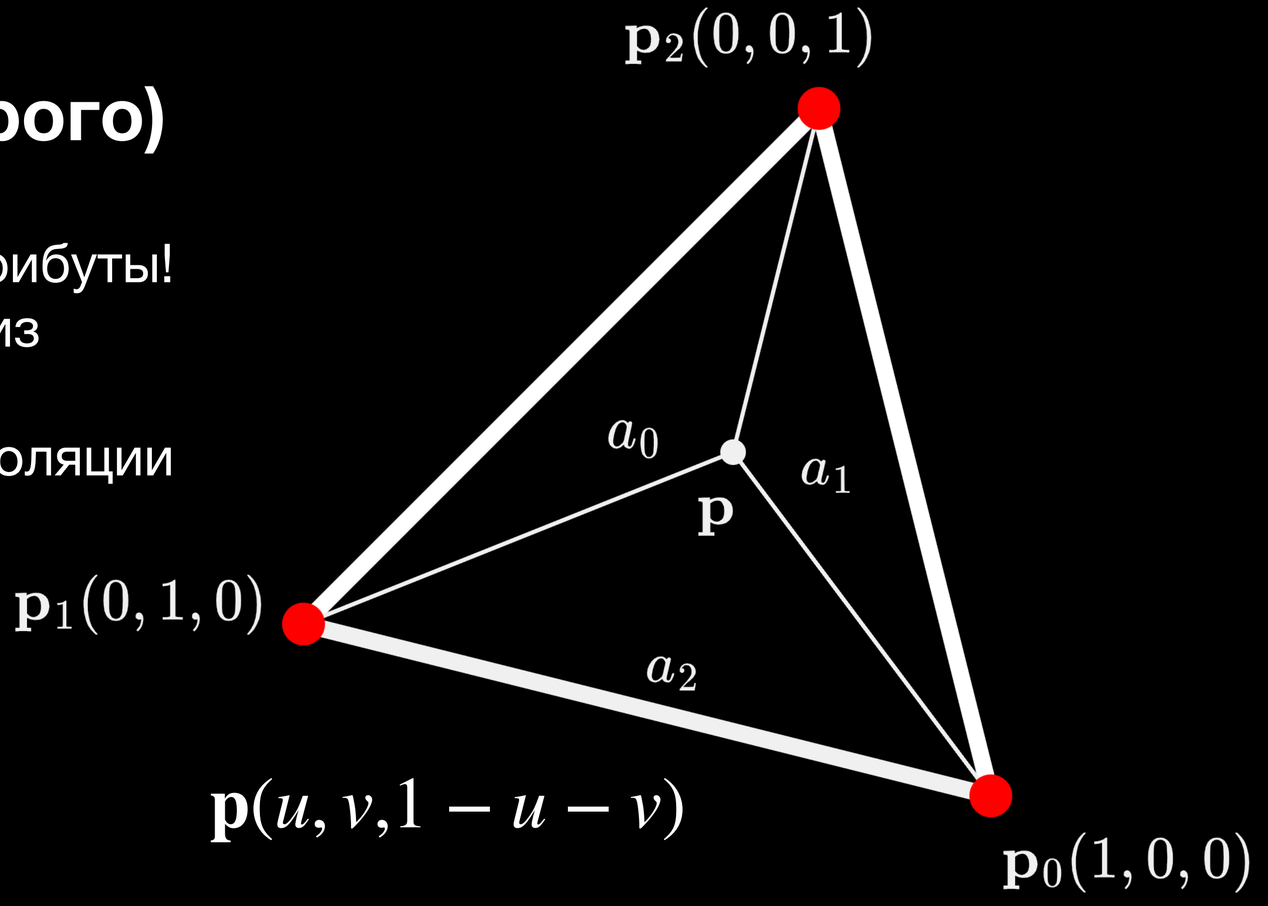
Для перспективной интерполяции можно скорректировать плоские барицентрики и использовать для линейной интерполяции.

Интерполяция

Коррекция барицентриков (строго)

- Барицентрические координаты — это атрибуты!
- Для каждой вершины, только один из барицентриков не нулевой.
- Используем формулу корректной интерполяции атрибутов

$$u(x,y) = \frac{u\frac{1}{w_0} + v\frac{0}{w_1} + (1-u-v)\frac{0}{w_2}}{u\frac{1}{w_0} + v\frac{1}{w_1} + (1-u-v)\frac{1}{w_2}}$$



$$u(x,y) = \frac{\frac{e_0(x,y)}{w_0}}{\frac{e_0(x,y)}{w_0} + \frac{e_1(x,y)}{w_1} + \frac{e_2(x,y)}{w_2}}$$

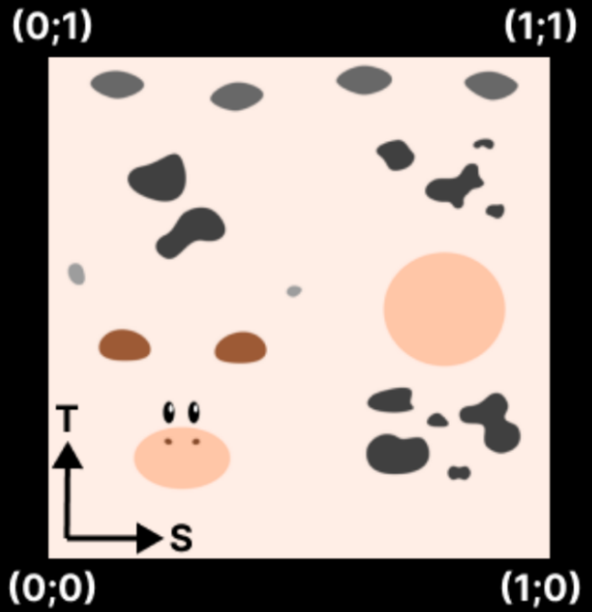
- И числитель и знаменатель зависит от положения $\mathbf{p}$
- Барицентрики после коррекции линейно интерполируют атрибуты

Интерполяция

Основное

- Мы получили универсальную формулу для перспективно корректной интерполяции атрибутов
- С ее помощью научились корректировать плоские барицентрики для корректной линейной интерполяции без необходимости применения универсальной формулы

Texture coordinates
$s, t \in \mathbb{R}^{H \times W}$



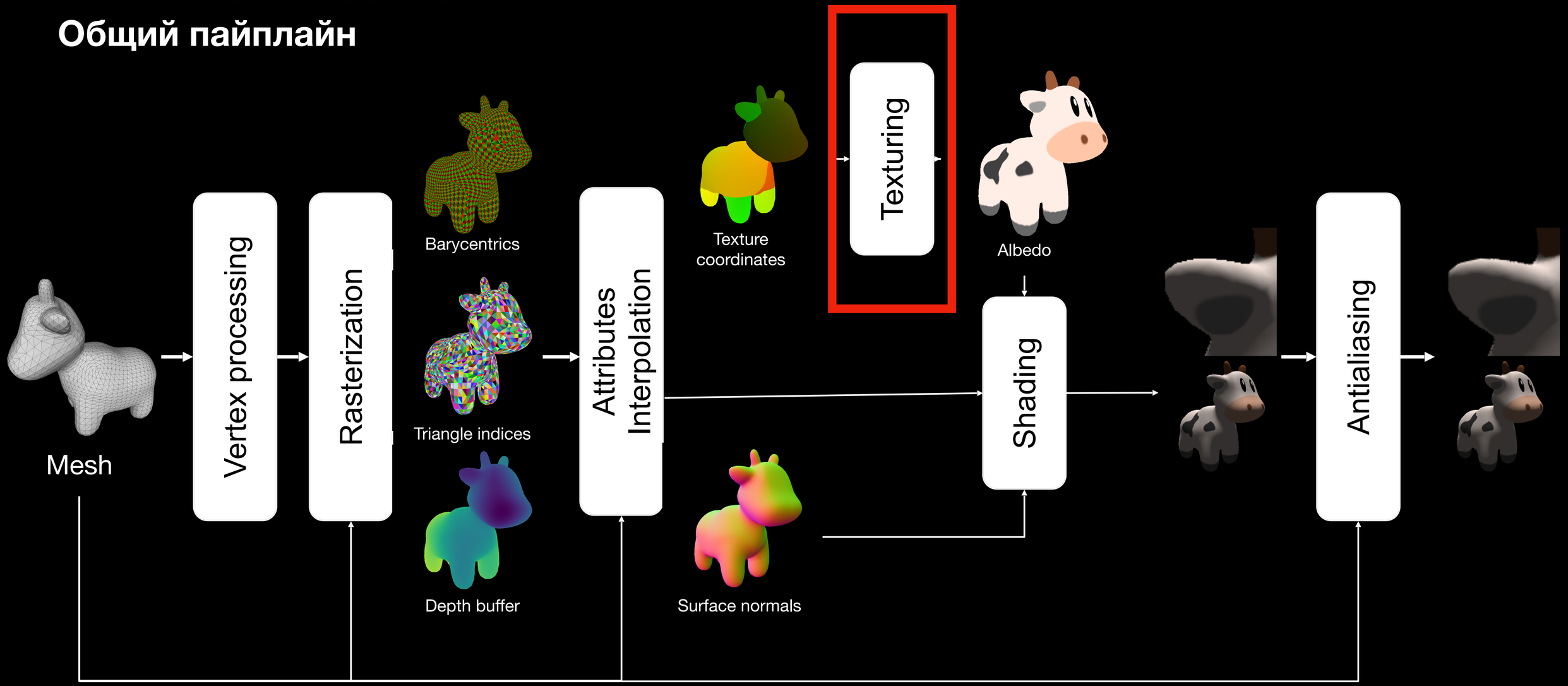
2D Texture
$T \in \mathbb{R}^{h \times w \times K}$

Sampled texture
$M \in \mathbb{R}^{H \times W \times K}$

Текстурирование

Rasterization

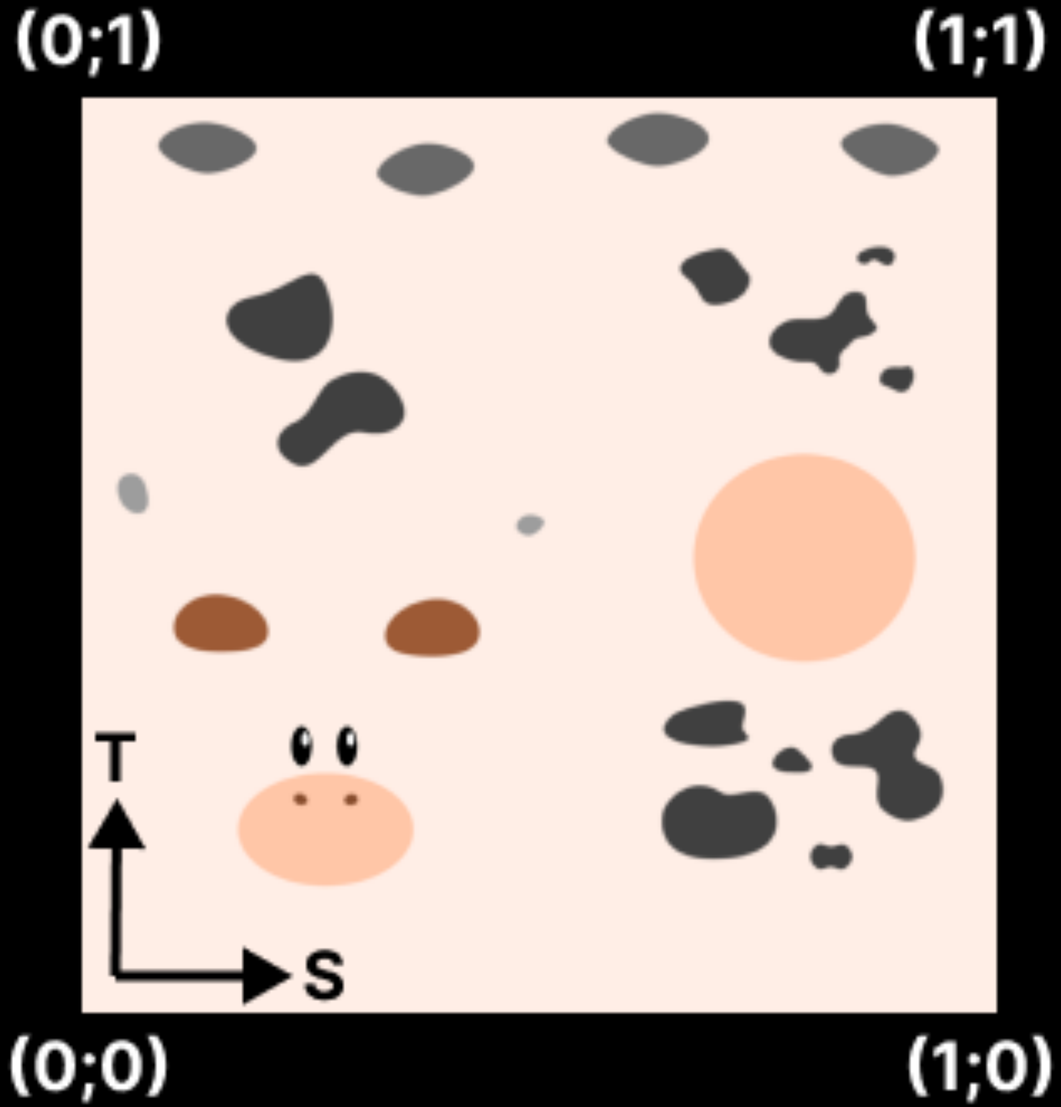
Общий пайплайн



Текстурирование
2D текстуры

- Текстура — это плоское изображение, каждый пиксель которого задает значение атрибута (цвета) какой-то точке на меше

- На этапе текстурирования из этого буфера семплируется значение по двум числам — текстурным координатам

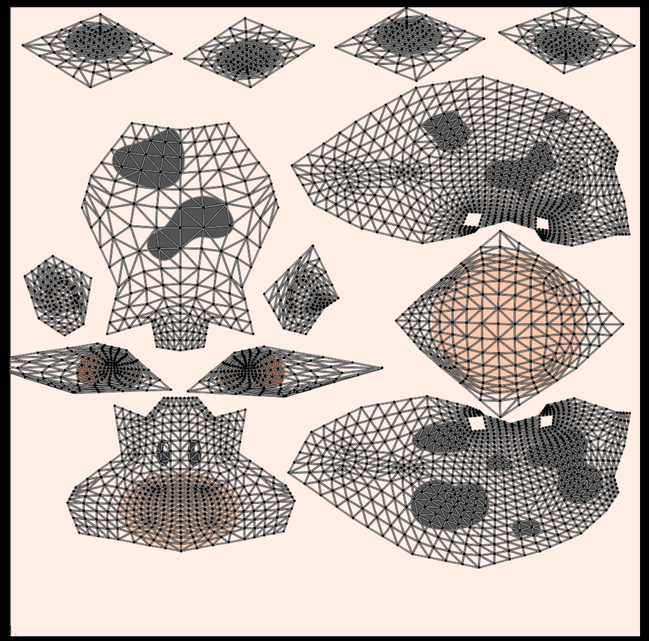


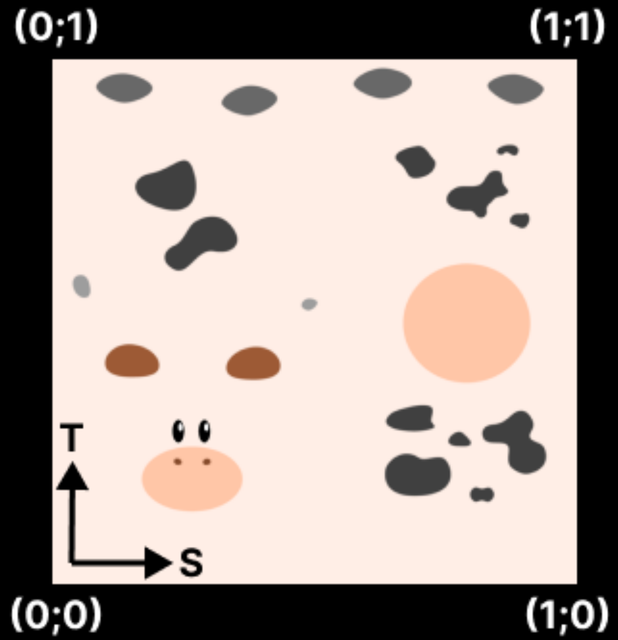
Текстурирование

2D текстуры. Текстурные координаты

- Текстурные координаты определены для каждой вершины меша
- Текстурные координаты описываются двумя нормализованными координатами (UV- или ST-):

$$(s_i,\ t_i)\ \in [0;\ 1]^2$$





- Построение текстурных координат — UV / ST развертка (за рамками лекции)
- Для семплирования для каждого пикселя сначала интерполируются текстурные координаты, затем по ним интерполируется текстура

Текстурирование

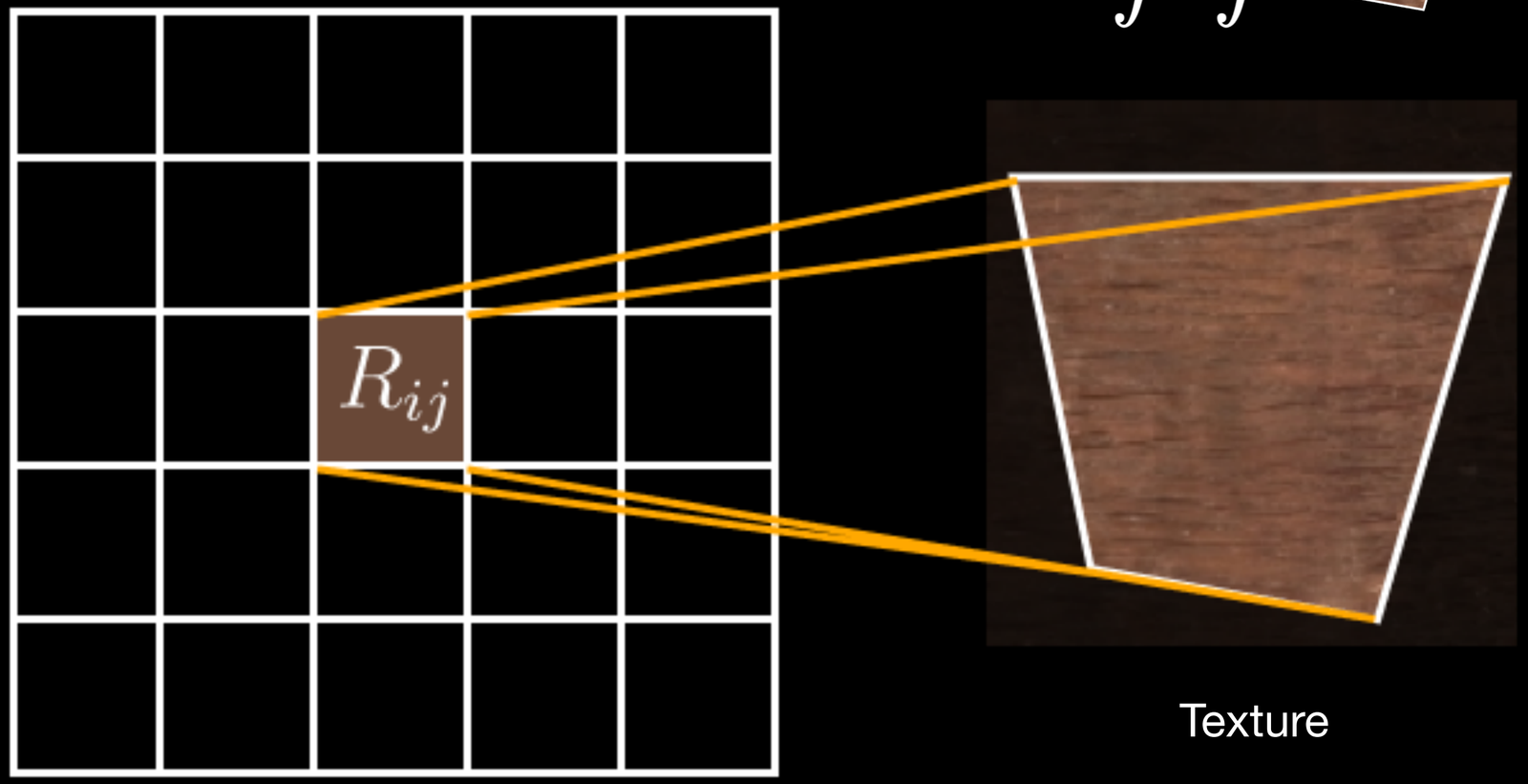
2D текстуры. Сэмплирование

Откуда появляются
артефакты?

Текстурирование

Аналитическое значение

$$R_{ij} = \int\int \, ds\, dt$$

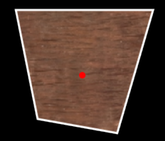


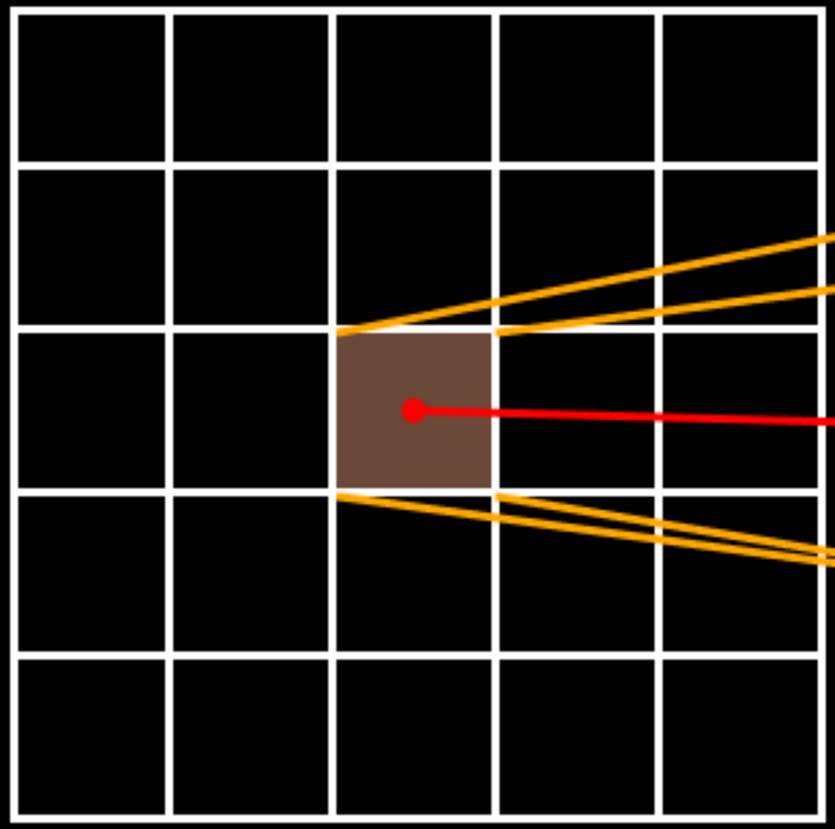
Rendered image

Texture

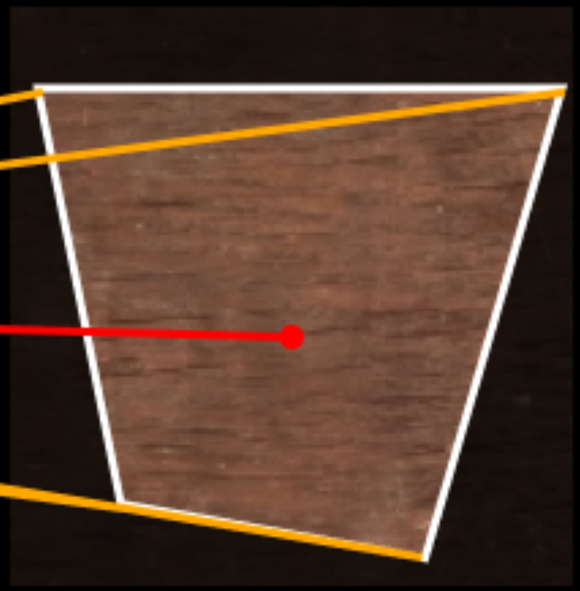
Текстурирование

Nearest оценка

$R_{ij} \approx$  => ALIASING



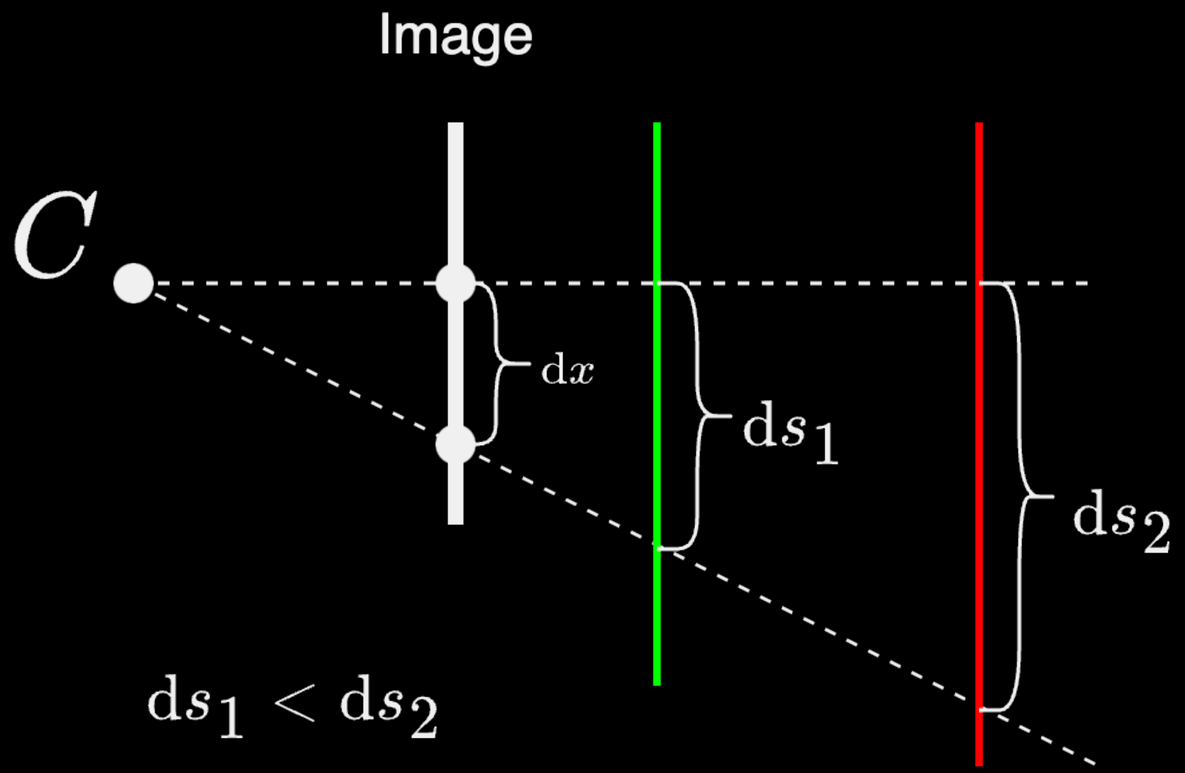
Rendered image



Texture

Текстурирование

MIP-mapping. Интуиция



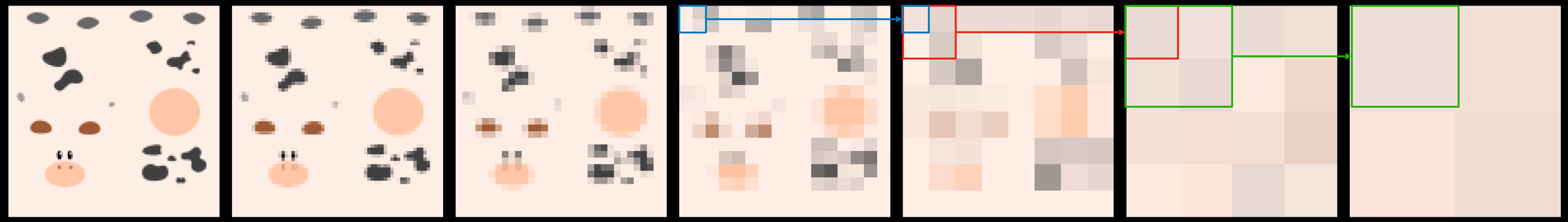
Интуиция: Чем дальше пиксель, тем большую площадь он занимает на текстуре

ИДЕЯ: Предынтегрировать текстуры по областям в зависимости от размера пикселя на текстуре и учесть это при семплирования

Текстурирование
MIP-mapping. Построение MIP уровней

- MIP уровни строятся усреднением пикселей по областям кратным двойке.
- Логарифм размера области (степень двойки) усреднения определяет MIP уровень.

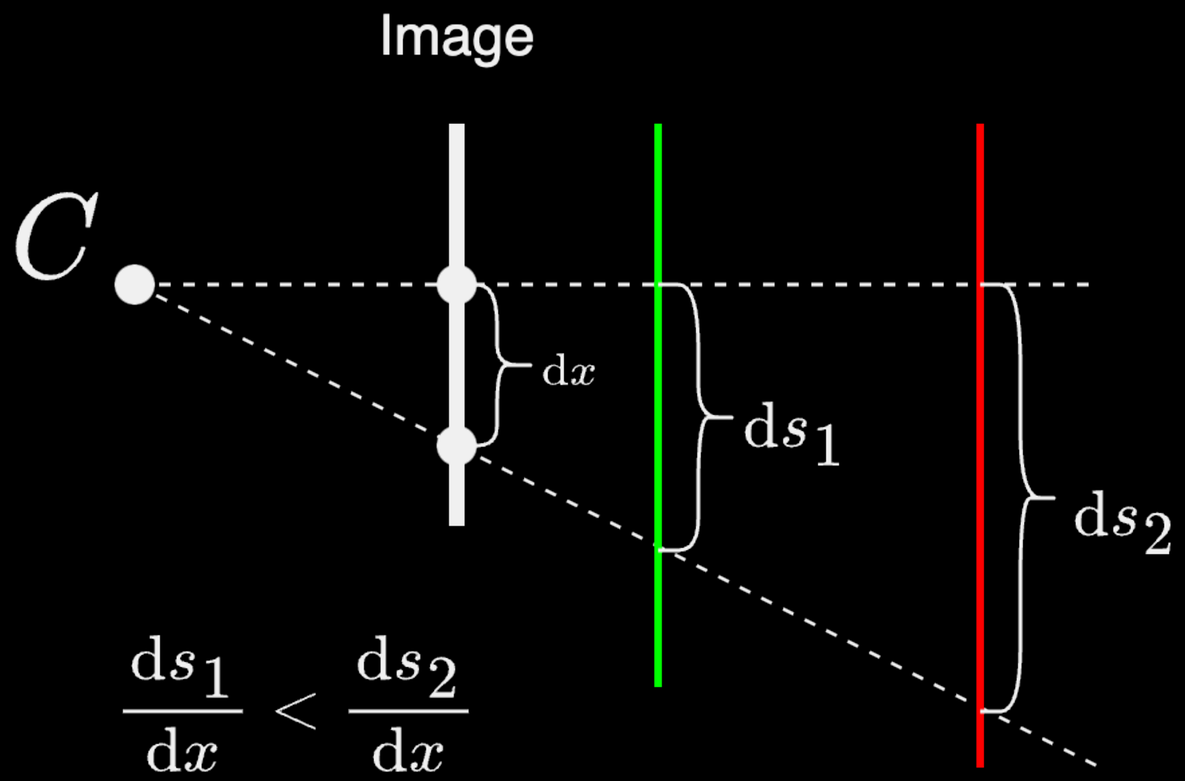
Average Pooling 2x2:



Mip 0 64x64 → Mip 1 32x32 → Mip 2 16x16 → Mip 3 8x8 → Mip 4 8x8 → Mip 5 4x4 → Mip 6 2x2

Текстурирование

Выбор MIP уровня. Интуиция



Так как лучи из камеры расходящиеся, то при смещении пикселя значение текстурных координат меняется меньше, если треугольник близко расположен, и сильно — если далеко.

ИДЕЯ: Использовать производные текстурных координат по координатам экрана для расчета MIP уровня

Текстурирование

Выбор MIP уровня

Необходимы screen-space производные текстурных координат:

$$J = \begin{pmatrix} \frac{ds}{dx} & \frac{ds}{dy} \\ \frac{dt}{dx} & \frac{dt}{dy} \end{pmatrix}$$

Вспомним, что s, t — это интерполированные текстурные координаты:

$$s = u(x, y)s_1 + v(x, y)s_2 + (1 - u(x, y) - v(x, y))s_3$$

Для расчета их производных необходимы производные барицентриков по screen-space координатам (x, y). Вычисляются дифференцированием перспективно корректных барицентриков аналитически.

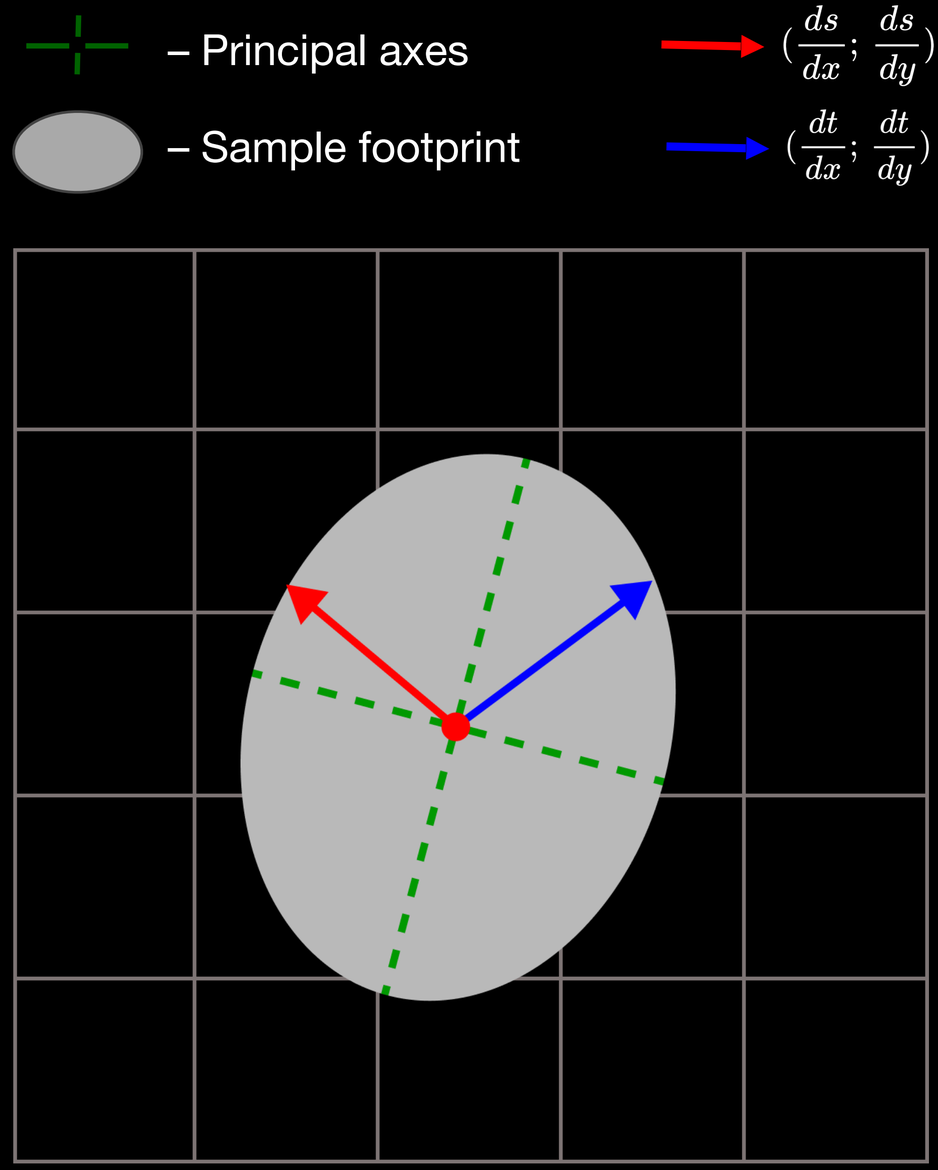
Текстурирование
Выбор MIP уровня

Необходимы screen-space производные текстурных координат:

$$J = \begin{pmatrix} \frac{ds}{dx} & \frac{ds}{dy} \\ \frac{dt}{dx} & \frac{dt}{dy} \end{pmatrix}$$

Определение нужного MIP уровня - задача расчета наибольшего сингулярного значения (длины наибольшей оси - собственного вектора):

$$w = \log_2 \sigma_1(J)$$



Текстурирование
Выбор MIP уровня

Наибольшее сингулярное значение ищется из уравнения:

$$\det\left(J^T J - \lambda I\right) = 0$$

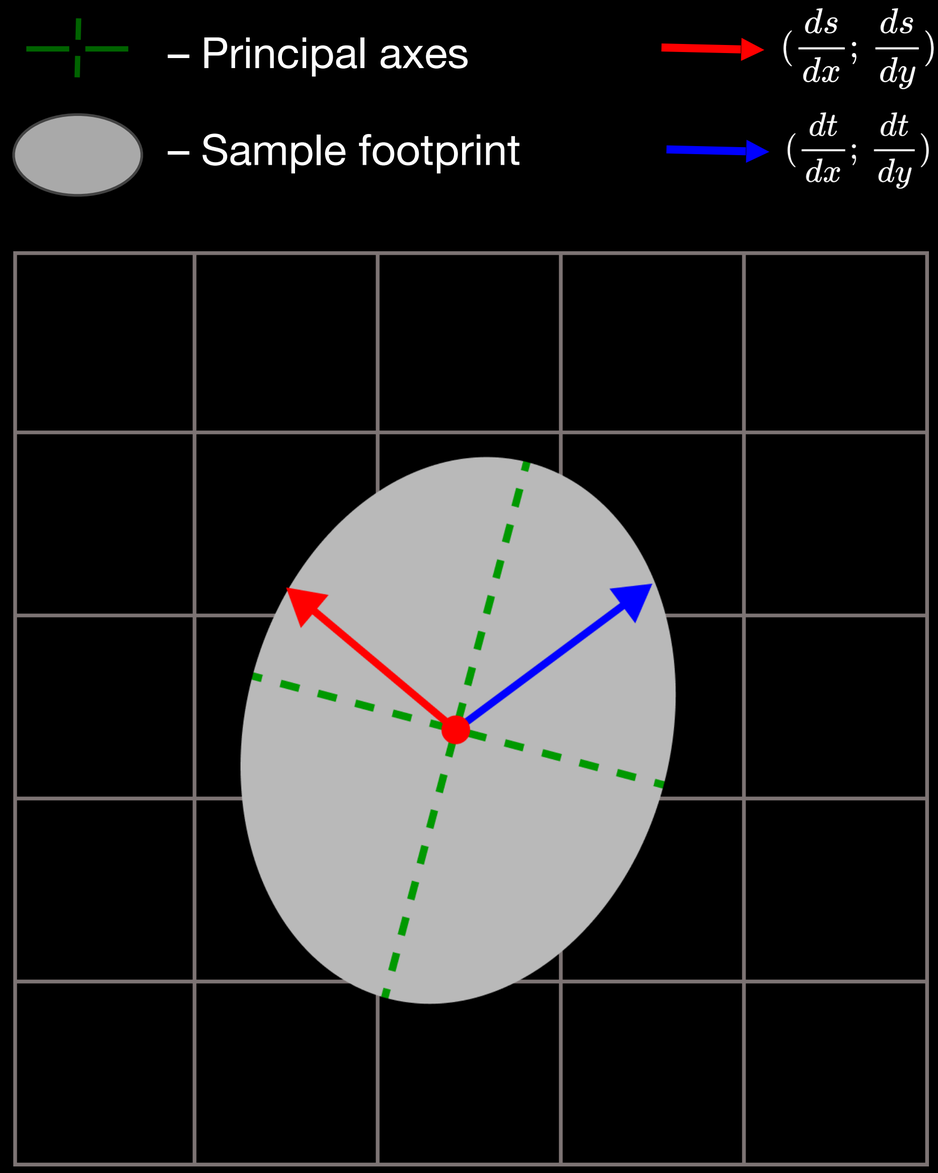
Решение:

$$A = \left(\frac{ds}{dx}\right)^2 + \left(\frac{dt}{dx}\right)^2$$

$$B = \left(\frac{ds}{dy}\right)^2 + \left(\frac{dt}{dy}\right)^2$$

$$C = \frac{ds}{dx}\frac{ds}{dy} + \frac{dt}{dx}\frac{dt}{dy}$$

$$w = \frac{1}{2}\log_2\left[\frac{A+B}{2} + \sqrt{\frac{1}{4}(A-B)^2 + C^2}\right]$$



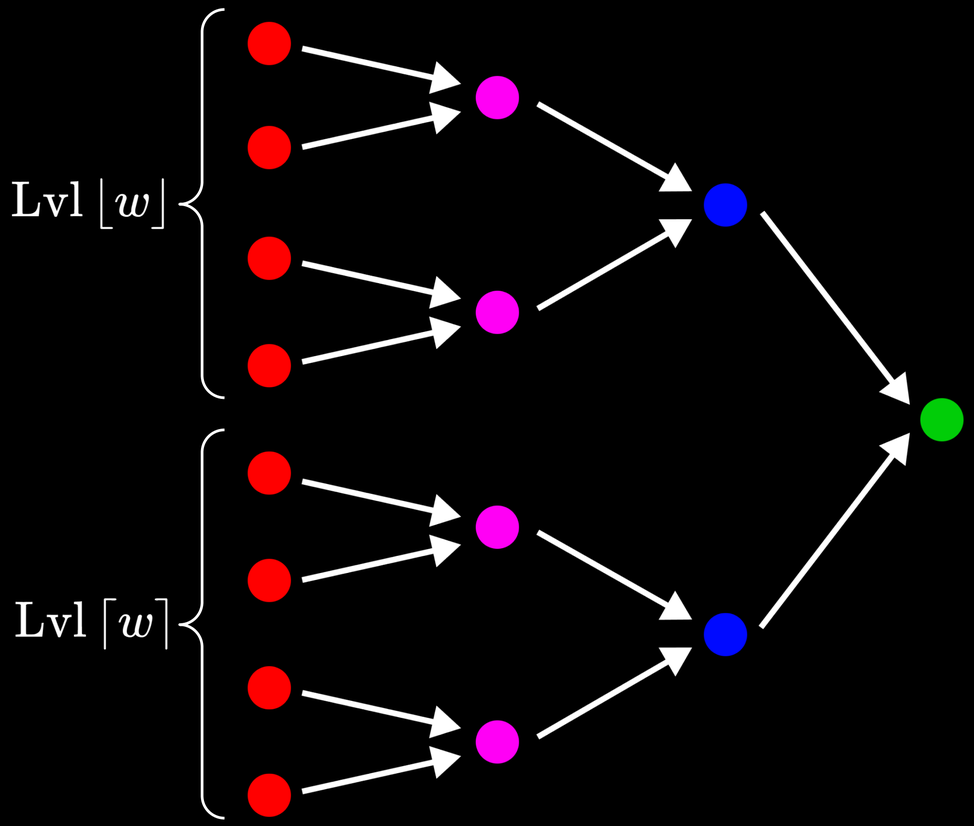
Текстурирование

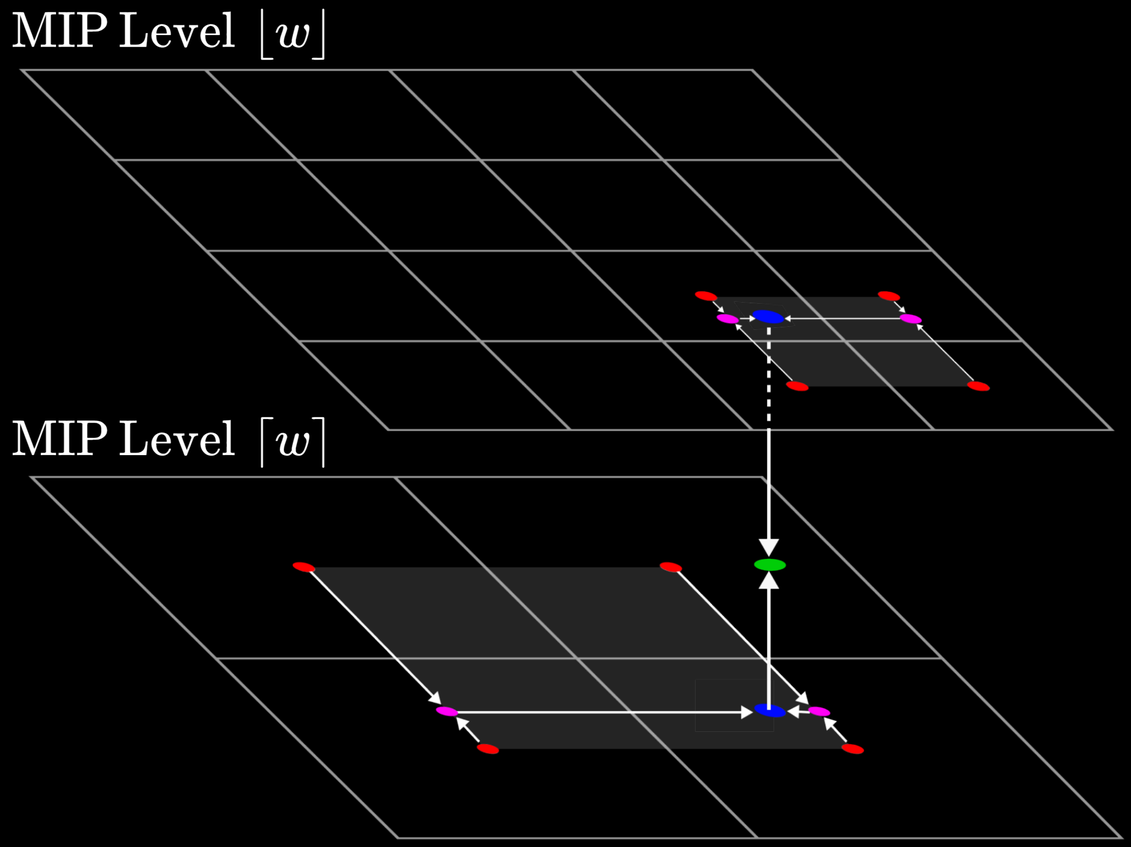
Три-линейная интерполяция

– Значения пикселей

– Результат

– Линейная интерполяция

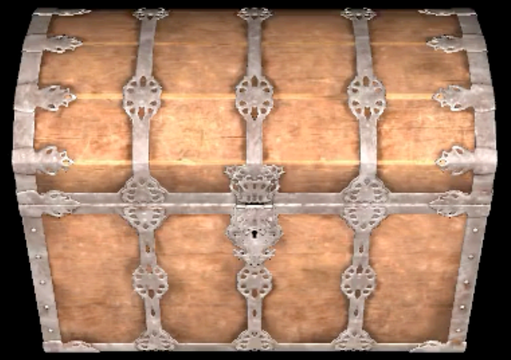




Текстурирование

Три-линейная интерполяция

До



После

Текстурирование

Основное

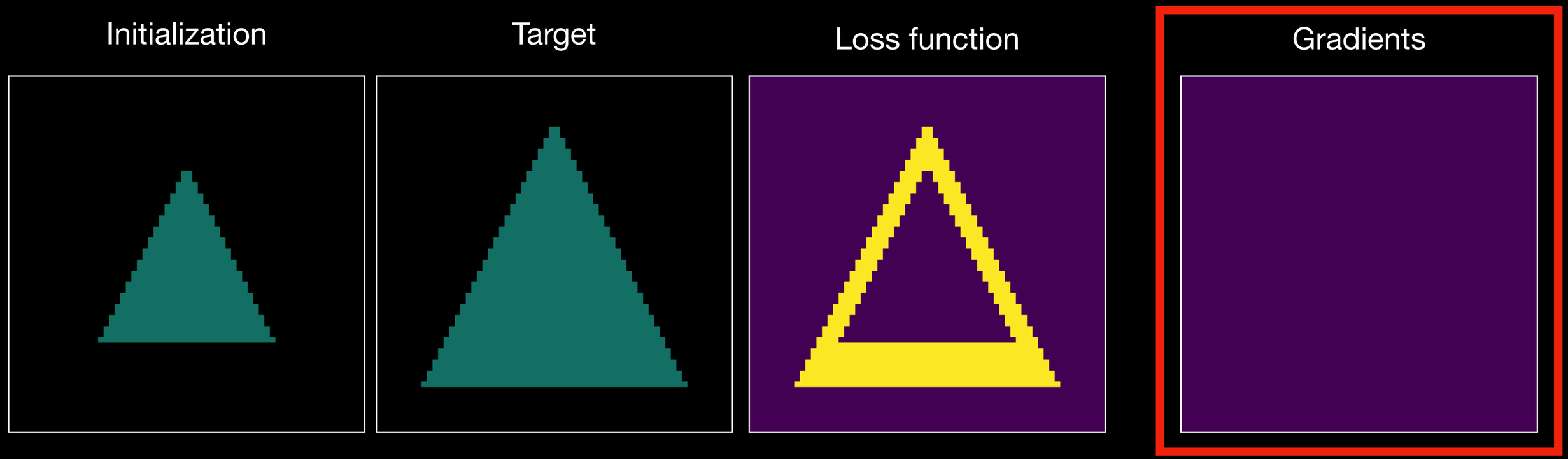
- При текстурировании необходимо учитывать размер области, занимаемой пикселем на текстуре. Чем дальше расположен растеризованный треугольник, тем больше эта область
- Симулировать этот эффект можно предынтегрированием текстуры по областям кратным двойке — построение MIP уровней. Выбор уровня — это логарифм характерного размера области покрытия пикселем.
- Для выбора MIP уровня можно использовать screen-space производные текстурных координат
- Для достижения непрерывности и дифференцируемости используются screen-space производные текстурных координат. Решается задача на поиск наибольшего сингулярного число матрицы якобиана screen-space производных текстурных координат.

Мы можем уже неплохо рендерить,
но что там с оптимизацией?

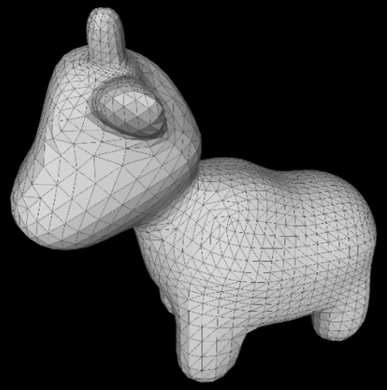
Оптимизация

Проблема visibility gradients

Рассмотрим простую задачу оптимизации треугольника под шаблон:

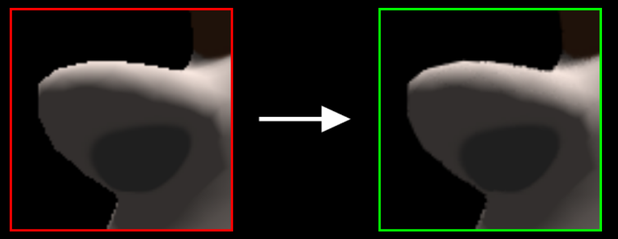


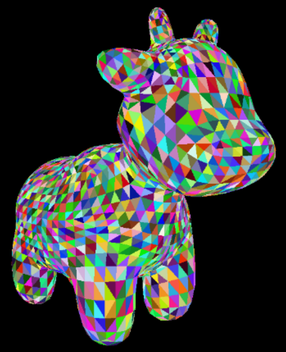
Функция потерь ненулевая, однако оптимизация не происходит, т.к. градиенты равны нулю



Mesh (Vertices + Indices)

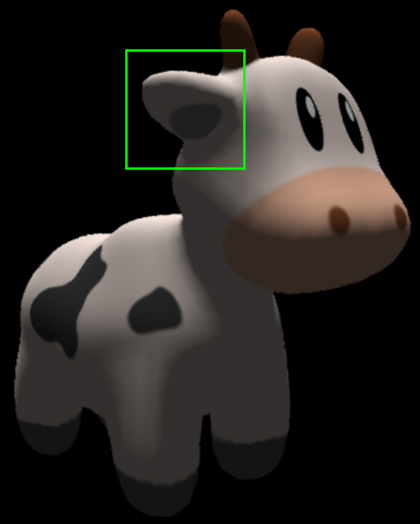
Shaded image





Triangle indices

Depth buffer

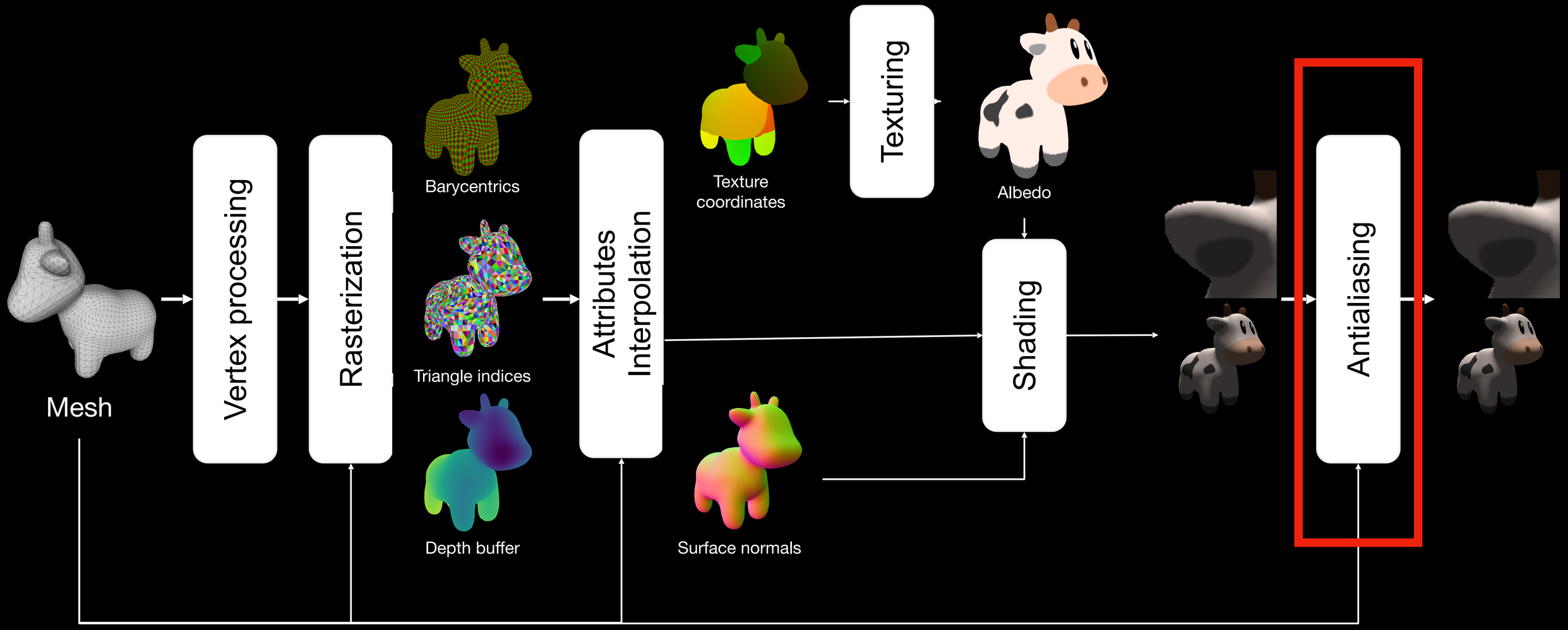


Anti-aliased image

Anti-aliasing

Rasterization

Общий пайплайн



Anti-aliasing

Зачем он вообще нужен?

Оптимизация:

Для оптимизации геометрии необходимо решить проблему градиенты видимости (visibility gradients)

Графика:

Возникают эффекты алиасинга на границах. Необходим способ смешения цветов в этих областях.

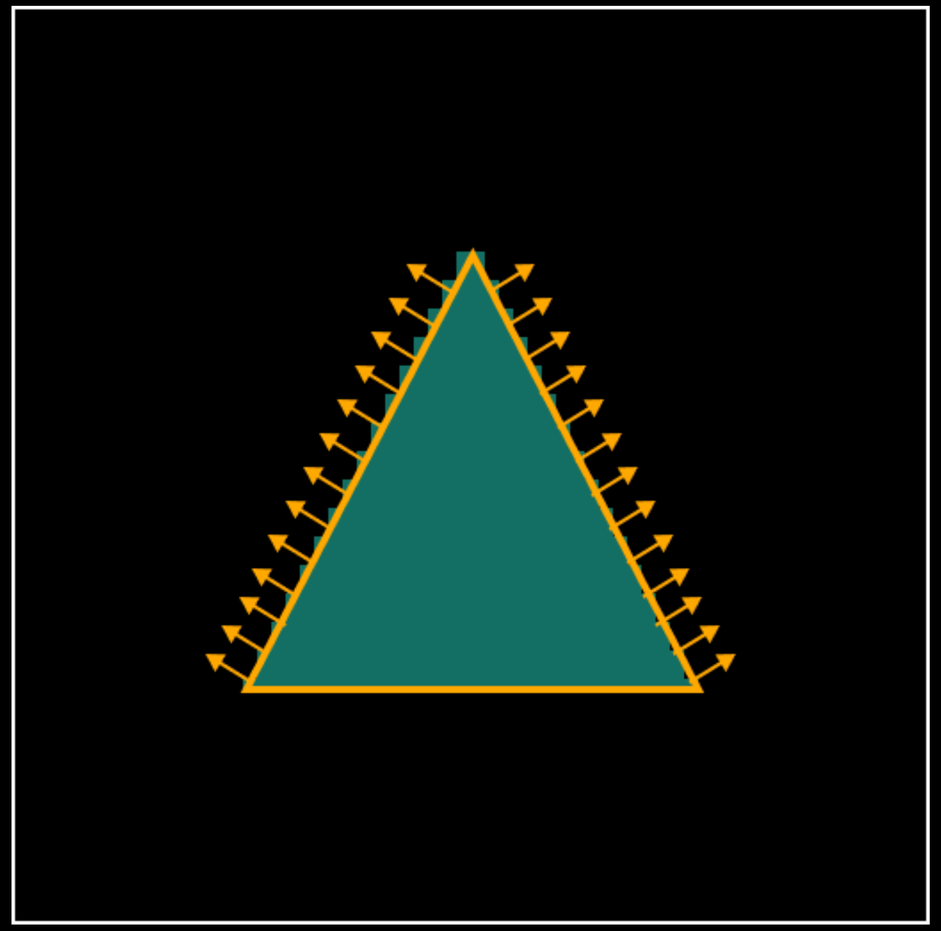
Anti-aliasing

Оптимизация. Visibility gradients

Мы не умеем дифференцировать
в точках разрыва функции.

Необходим корректный метод
дифференцирования, который
распространит градиенты
через точки разрыва.

Для *обратного рендеринга эта проблема первостепенна!*

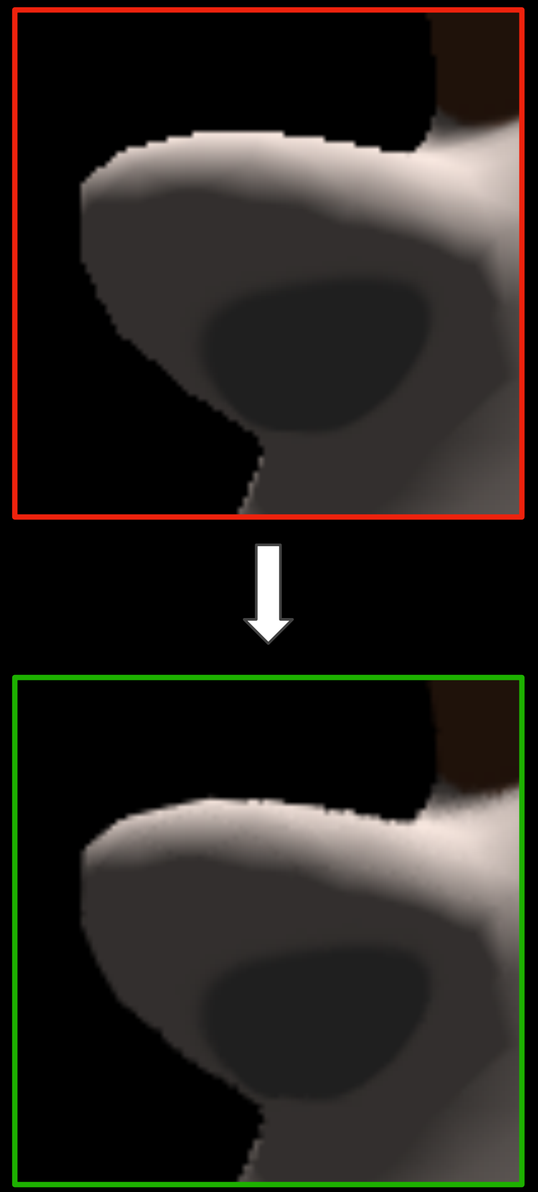


Anti-aliasing

Графика. Алиасинг на границах

Силуэтные ребра треугольника создают резкие зазубренные края на изображении.

Необходим метод сглаживающий границы



Anti-aliasing

Решение. Основная идея

Для решения проблемы видимых градиентов необходимо уметь дифференцировать в граничных точках.
Эти границы субпиксельные!

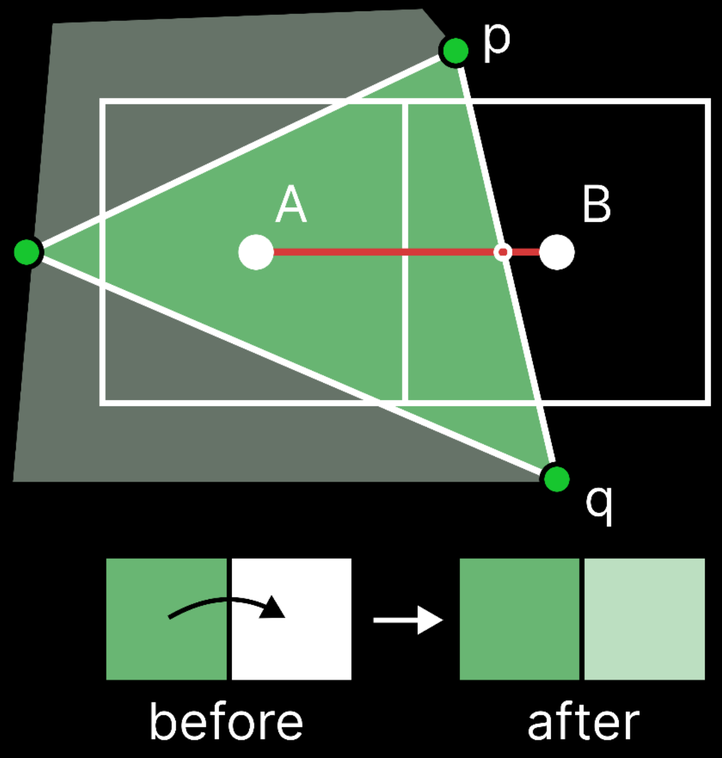
Необходимо ввести параметризованную модель границы в процедуру рендеринга, по параметрам которой уже можно считать градиенты

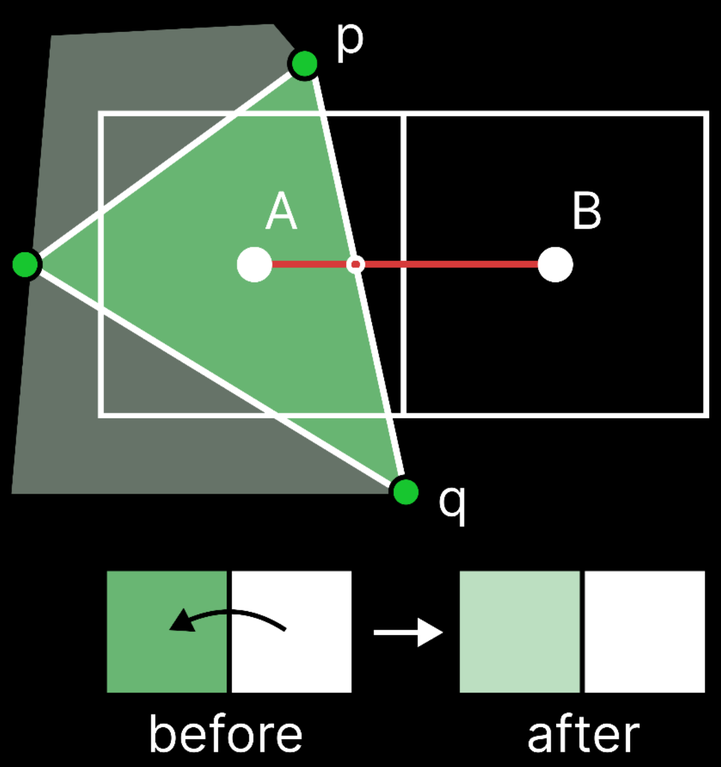
Anti-aliasing

Решение. Основная идея

Идея: ввести зависимость цвета пикселей при переходе через параметризованную границу непрерывную по параметрам этой границы —>

интерполяция вдоль AB параметризованная на грань PQ

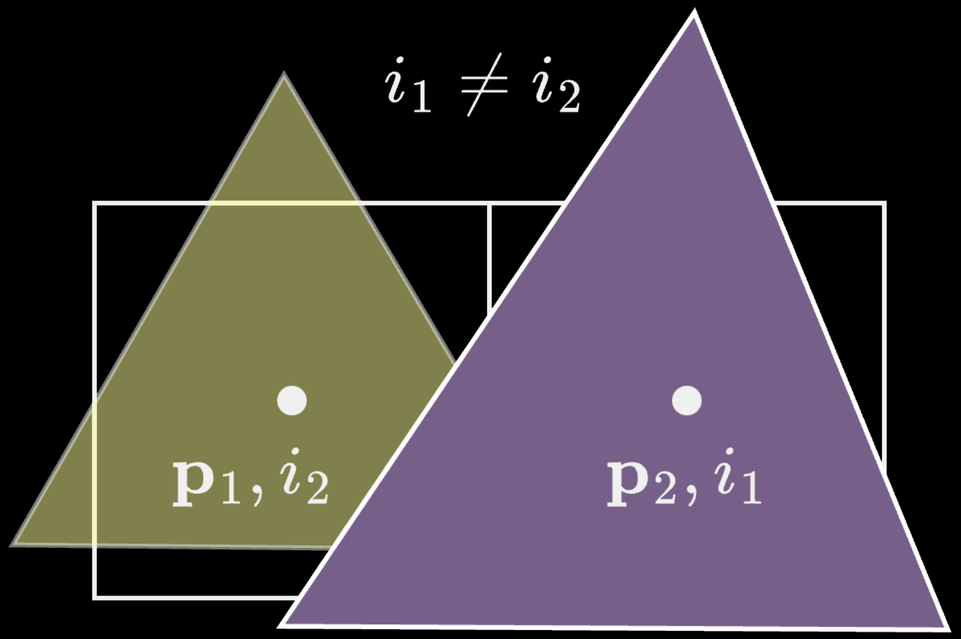




Anti-aliasing

Этап 1. Точки разрыва — как найти?

Точки разрыва возникают, если в соседние пиксели растеризуются разные треугольники не имеющие общих ребер



$\mathbf{p}_1, \mathbf{p}_2$ — центры соседних пикселей

i_1, i_2 — индексы растеризованных треугольников

Далее рассматривается только ближайший по глубине из этих двух

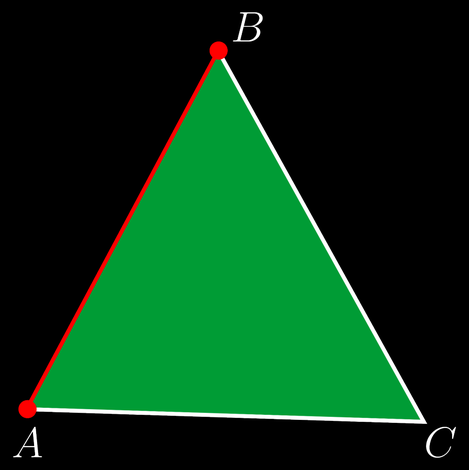
Растеризовать треугольник = треугольник покрывает центр пикселя

Anti-aliasing

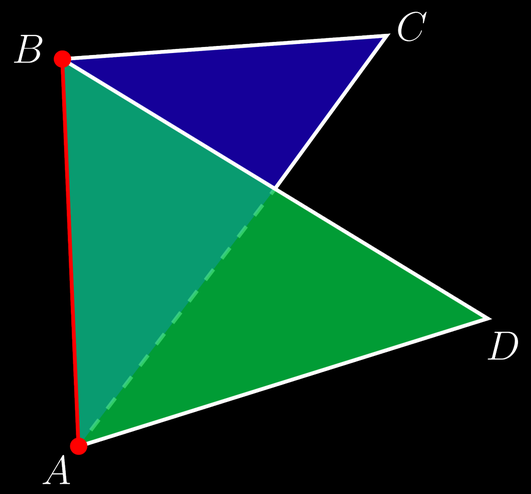
Этап 1. Поиск границы — силуэтное ребро

Для выбранного треугольника выбираются силуэтные ребра

$\nexists D$

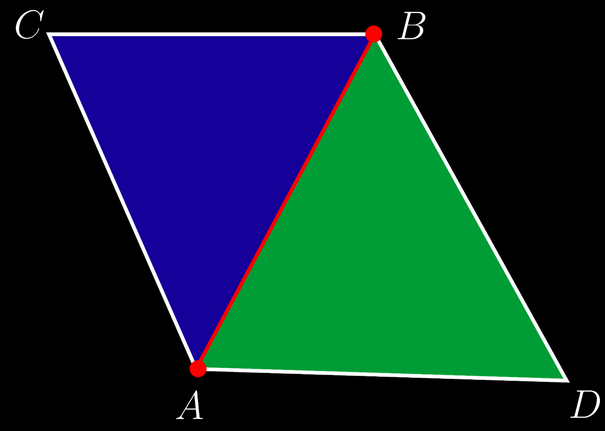


$sign(\, e_{A,B}(C)\,) = sign(\, e_{A,B}(D)\,)$



Силуэтное ребро. $e_{A,B}(p)$ - edge function

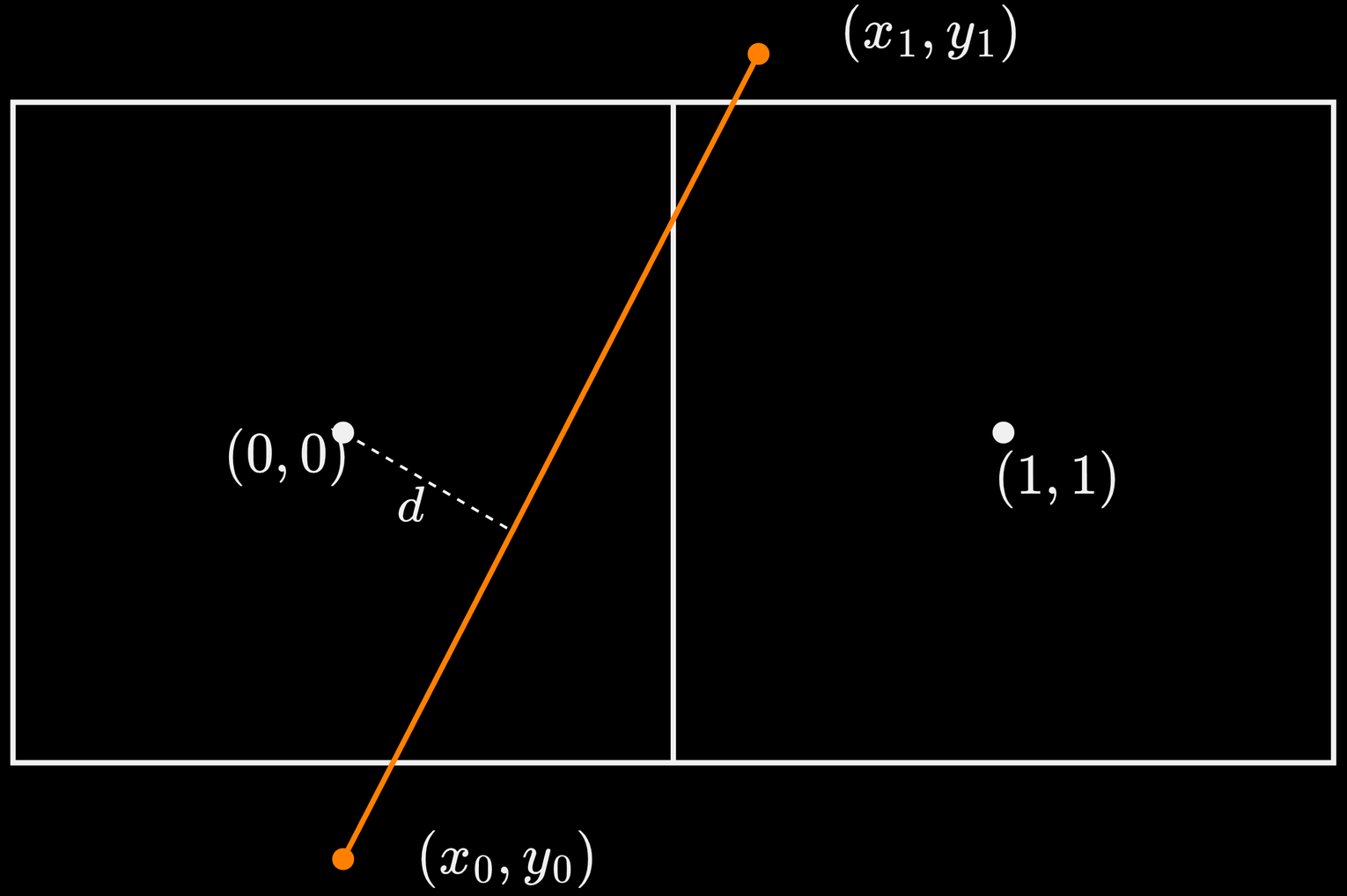
$sign(\, e_{A,B}(C)\,) \neq sign(\, e_{A,B}(D)\,)$



Обычное ребро

Anti-aliasing

Этап 2. Параметризация границы



Граница — это прямая:

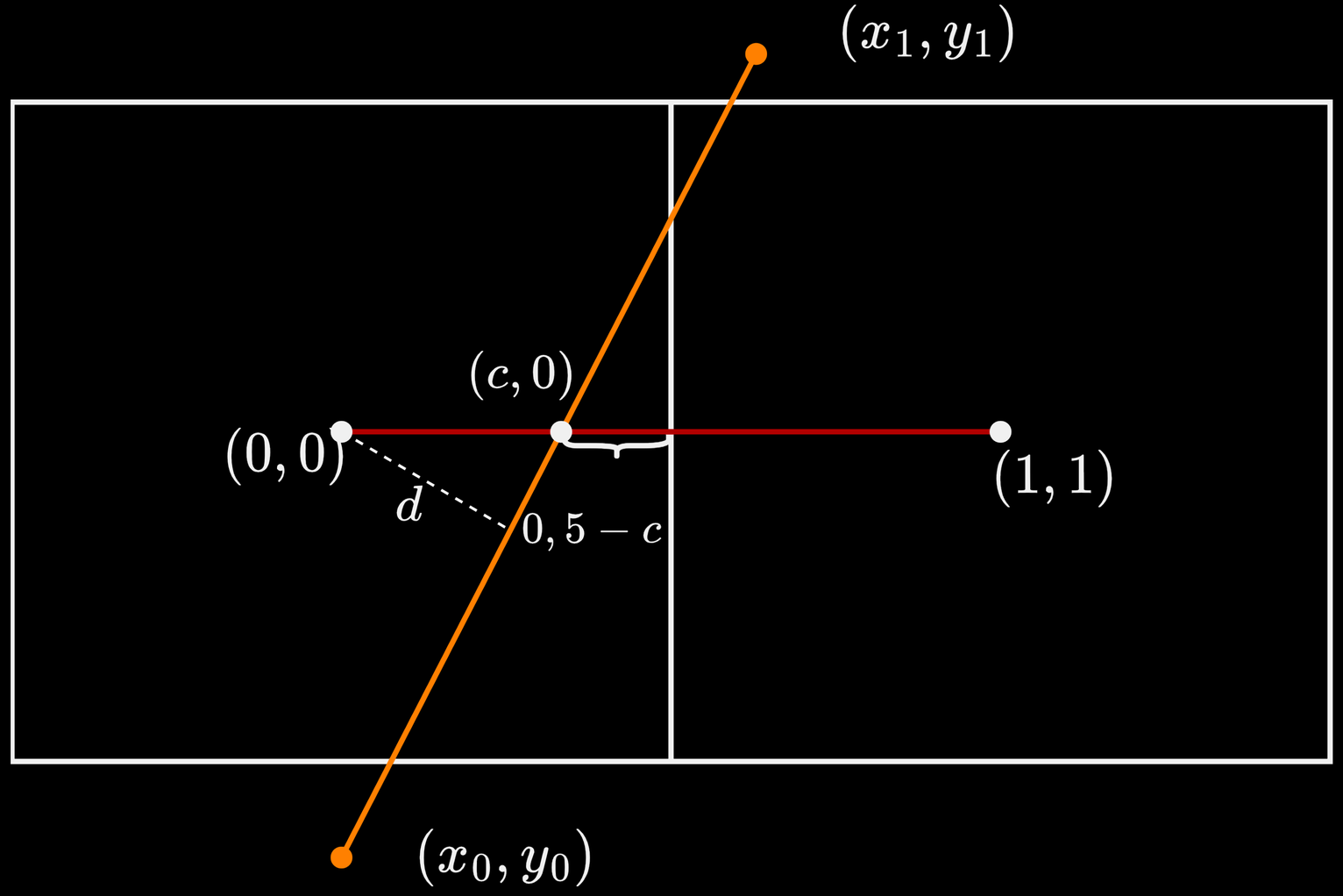
$$\left\langle (x_0, y_0), \mathbf{n} \right\rangle = d$$

Нормаль:

$$\mathbf{n} = (y_0 - y_1\,, x_1 - x_0)$$

Anti-aliasing

Этап 3. Коэффициента смешивания цветов



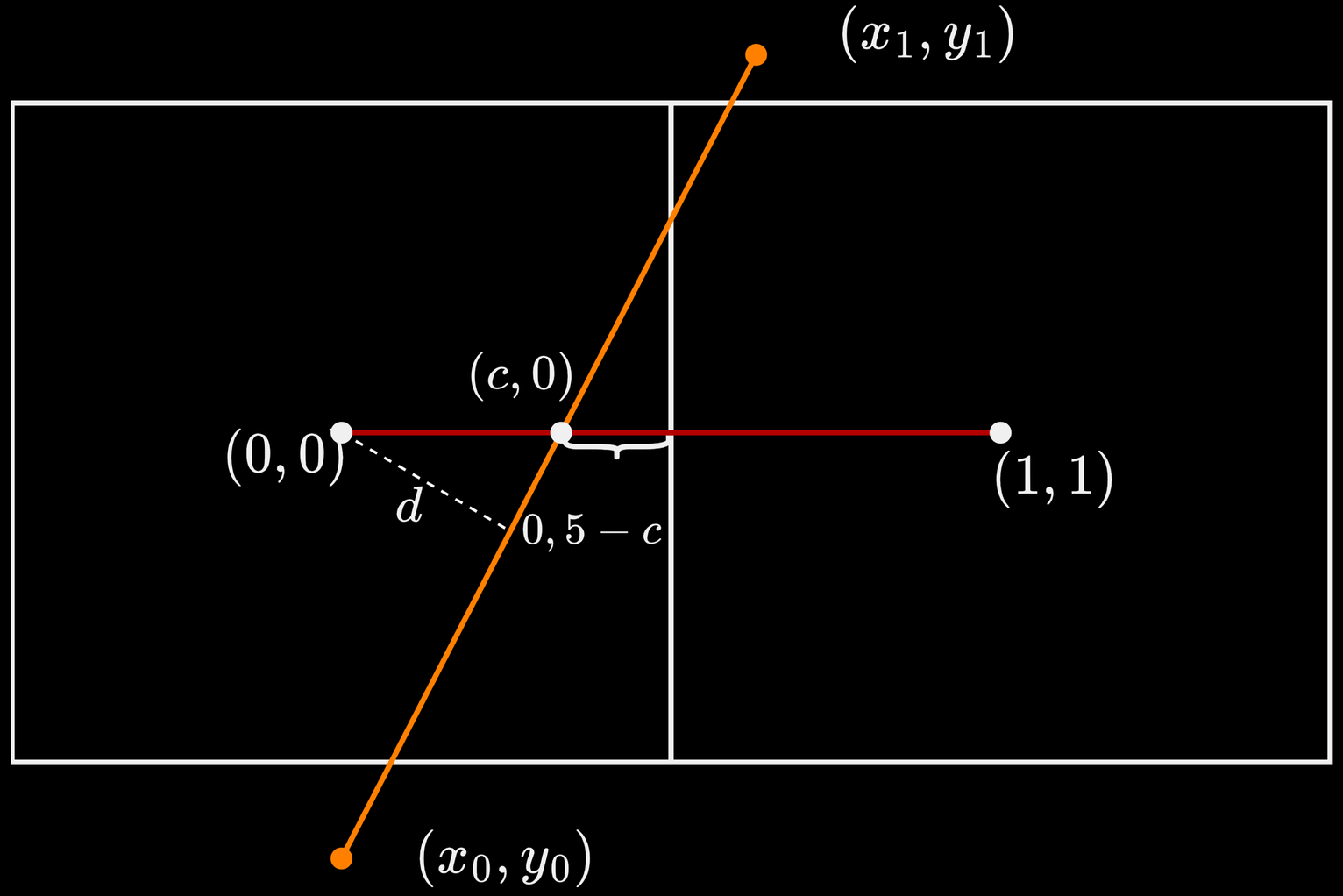
Цвета пикселей смешиваются линейно с коэф-том:

$$\alpha = 0.5 - c$$

Такая параметризация не меняет цвета, если точка пересечения — центр отрезка и позволяет добавлять или уменьшать цвета при переходе через эту точку

Anti-aliasing

Этап 3. Расчет коэффициента смешивания



Граница — это прямая:

$$\langle (x_0, y_0), \mathbf{n} \rangle = d$$

Точка $(c,0)$ лежит на границе:

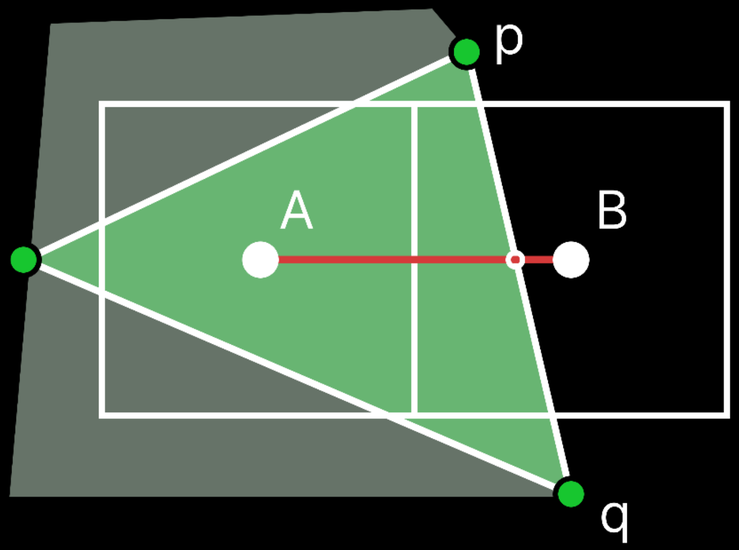
$$\langle (c,0), \mathbf{n} \rangle = d$$

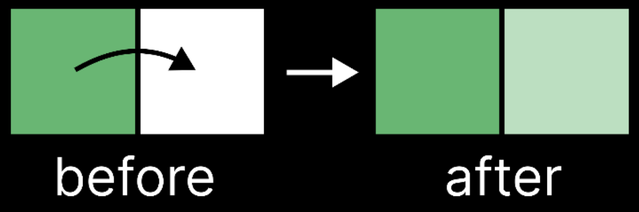
$$c = x_0 - y_0 \frac{x_1 - x_0}{y_1 - y_0}$$

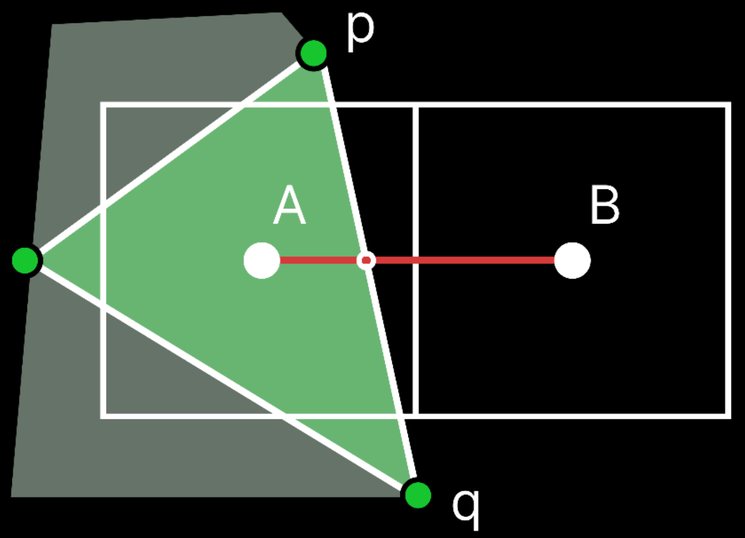
$$\alpha = 0.5 - c$$

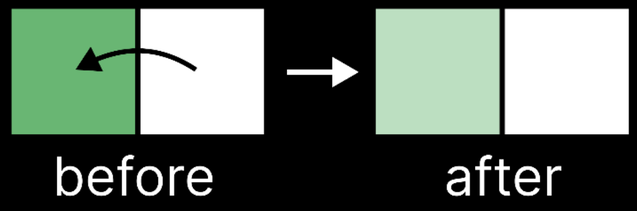
Anti-aliasing

Этап 4. Смешение цвета









Anti-aliasing

Optimization

Step 001/150

Ground-truth

Anti-aliasing

Основное

- Смешивает цвета пикселей в граничных точках
- Решает проблему разрывности функции на границах
- Параметризует границу силуэтными ребрами
- Коэффициент смешение цвета непрерывно и дифференцируемо зависит от параметров границы (вершин треугольников)

Где применяется?

Extracting Triangular Meshes, Materials, and Lighting From Images

Jacob Munkberg Jon Hasselgren Tianchang Shen Jun Gao

Wenzheng Chen Alex Evans Thomas Müller Sanja Fidler

NVIDIA University of Toronto Vector Institute

Magic3D : High-Resolution Text-to-3D Content Creation

Chen-Hsuan Lin* Jun Gao* Luming Tang* Towaki Takikawa* Xiaohui Zeng*
Xun Huang Karsten Kreis Sanja Fidler# Ming-Yu Liu# Tsung-Yi Lin

*# : equal contributions

NVIDIA Corporation