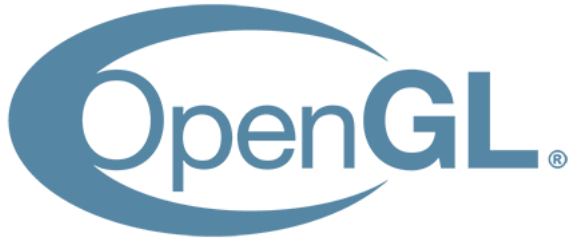
# Depth Component Textures

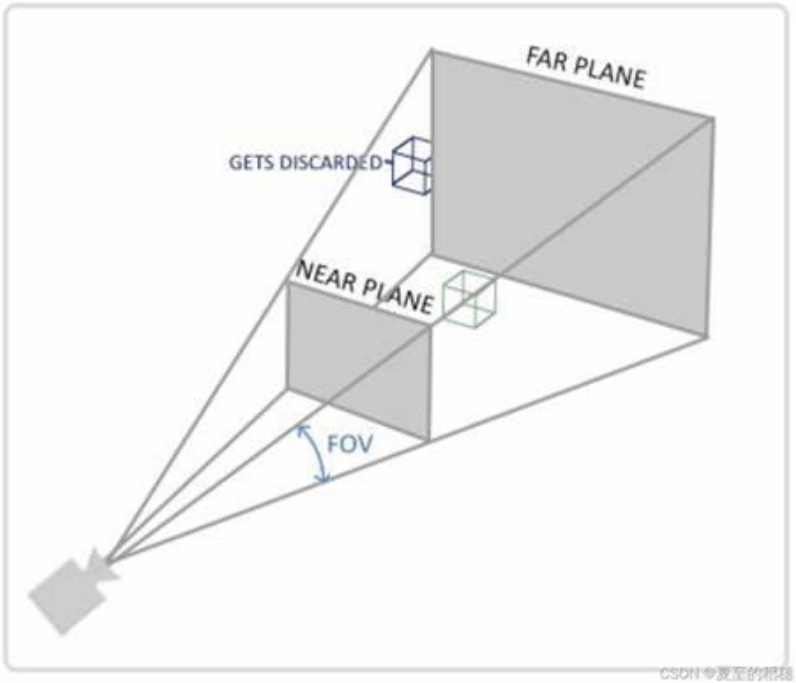
Обзор на основе документации OpenGL 4.6

Выполнил: студент группы 3823Б1ФИ1
Балдин Андрей



# Зачем нужно определять глубину?

- В 3D-графике нам важно не только то, какого цвета объект, но и как далеко он находится от камеры.
- Эта информация о расстоянии называется глубиной и хранится в Depth Textures.
- Без информации о глубине невозможно правильно отобразить, какие объекты находятся перед другими.



# Что такое Depth Component Textures?

Depth Component Textures - это текстуры, которые хранят информацию о глубине.  Глубина – это расстояние от камеры до каждой точки на поверхности объектов в 3D-сцене.

Данные о глубине необходимы для правильного отображения трёхмерных сцен и используются в таких графических техниках:

- Создание теней: Определение, какие участки сцены затенены.
- Тестирование глубины: Определение, какие пиксели видны, а какие закрыты другими объектами.
- Эффекты постобработки: Например, создание эффекта глубины резкости.

# Depth Textures рассматриваются как RED Textures

- OpenGL рассматривает текстуры глубины (Depth Textures) как RED текстуры при некоторых операциях, например, при фильтрации и применении текстур.

- Это значит, что при использовании Depth Texture, значение глубины воспринимается как значение красного (RED) канала. Значения других цветовых каналов (зелёного, синего, альфа) при этом игнорируются или берутся значения по умолчанию.

- Такой подход позволяет единообразно обрабатывать текстуры глубины с помощью стандартных механизмов работы с цветными текстурами, где каждый тексел содержит каналы R, G, B и A (или их часть).

# Cube Map Texture Selection
# (Выбор текстуры кубической карты)

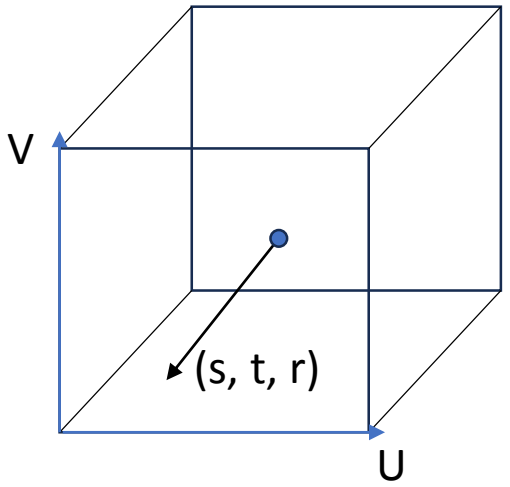
# От 3d адресации к 2d граням на примере куба

- Кубическая текстура состоит из шести квадратных 2D-текстур, каждая из которых соответствует одной грани куба.

- Мы обращаемся к кубической текстуре с помощью 3D-вектора (s, t, r), но каждая грань является плоской (2D). Чтобы получить цвет (или значение глубины) в точке на грани, нужны 2D-координаты (u, v), определяющие положение внутри этой грани.

- Главный вопрос: как 3D-вектор направления (s, t, r) определяет, какую из шести граней использовать и какие 2D-координаты (u, v) соответствуют этому направлению?

- OpenGL использует механизм, основанный на анализе компонент вектора (s, t, r), чтобы однозначно определить "ближайшую" к заданному направлению грань куба.



# Переход от 3D к 2D координатам в кубической карте

- При выборке из кубической текстуры, 3D-координаты (rx, ry, rz) рассматриваются как вектор направления из центра куба. OpenGL определяет, какую грань куба использовать, анализируя этот вектор.
- Ключевой шаг – определение главной оси (major axis direction), то есть оси, вдоль которой компонента вектора (rx, ry, или rz) имеет наибольшее абсолютное значение. Знак этой компоненты также важен.
- Таблица показывает, как главная ось определяет выбор грани кубической текстуры

| Major Axis Direction | Target | $s_c$ | $t_c$ | $m_a$ |
|---|---|---|---|---|
| $+r_x$ | TEXTURE_CUBE_MAP_POSITIVE_X | $-r_z$ | $-r_y$ | $r_x$ |
| $-r_x$ | TEXTURE_CUBE_MAP_NEGATIVE_X | $r_z$ | $-r_y$ | $r_x$ |
| $+r_y$ | TEXTURE_CUBE_MAP_POSITIVE_Y | $r_x$ | $r_z$ | $r_y$ |
| $-r_y$ | TEXTURE_CUBE_MAP_NEGATIVE_Y | $r_x$ | $-r_z$ | $r_y$ |
| $+r_z$ | TEXTURE_CUBE_MAP_POSITIVE_Z | $r_x$ | $-r_y$ | $r_z$ |
| $-r_z$ | TEXTURE_CUBE_MAP_NEGATIVE_Z | $-r_x$ | $-r_y$ | $r_z$ |

Table 8.19: Selection of cube map images based on major axis direction of texture coordinates.

"Major Axis Direction" – указывает ось с наибольшей абсолютной величиной
"Target" - определяет константу OpenGL для выбранной грани куба
$S_c$ – представляет первую скорректированную координату для выборки внутри выбранной 2D-грани
$T_c$ - Представляет вторую скорректированную координату
$m_a$ - Указывает, какая из исходных координат ($r_x$, $r_y$, $r_z$ ) используется для нормализации $s_c$ и $t_c$ в дальнейших вычислениях.

# Вычисление финальных 2d координат

После определения $s_c$, $t_c$ и $m_a$ (на основе таблицы с предыдущего слайда), OpenGL вычисляет окончательные 2D-текстурные координаты (s, t) для выборки текселя на выбранной грани по формулам:

$$s = \frac{1}{2}\left(\frac{s_c}{|m_a|} + 1\right)$$

$$t = \frac{1}{2}\left(\frac{t_c}{|m_a|} + 1\right)$$

# Seamless Cube Map Filtering
# (Бесшовное фильтрование кубических карт)

# Бесшовное фильтрование кубических карт

- При использовании кубических текстур могут возникать видимые швы на гранях куба, особенно при фильтрации. Это происходит потому, что каждая грань – отдельная 2D-текстура, и фильтрация на границе грани может не учитывать соседние грани.
- OpenGL предоставляет возможность включить бесшовное фильтрование кубических карт для решения этой проблемы.

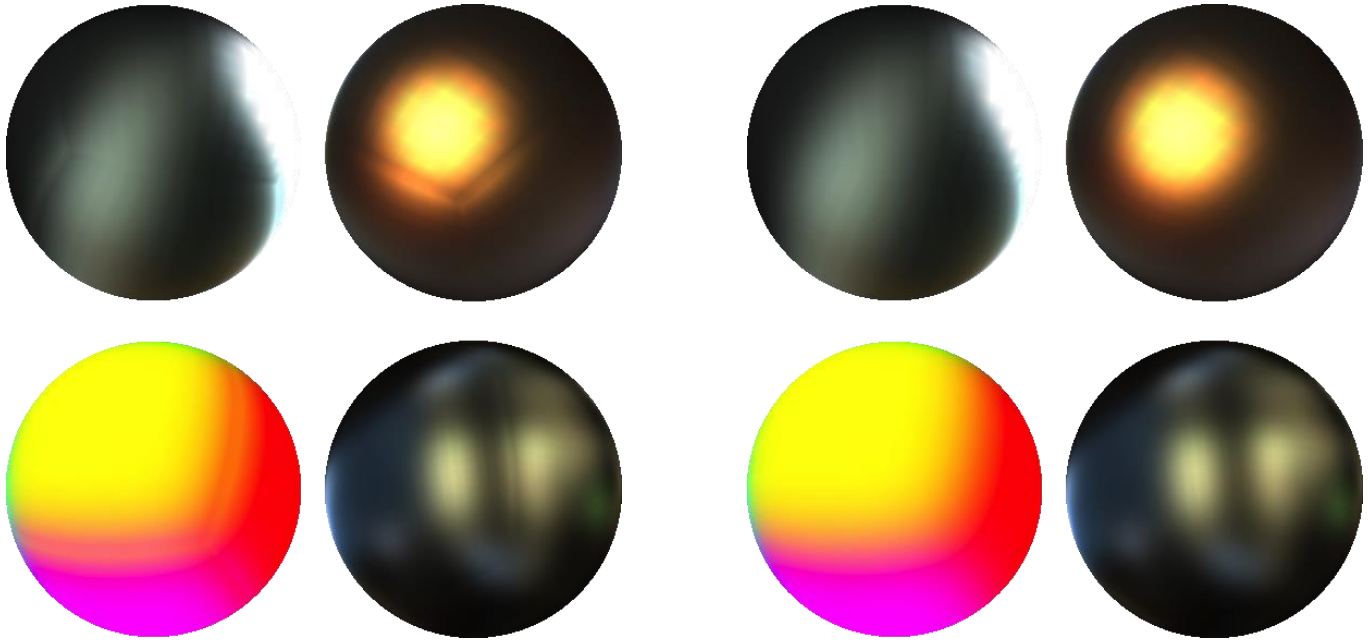
Использование (C#):

```
using OpenTK.Graphics.OpenGL4;

// ... ваш код ...

// Включение бесшовной фильтрации кубических карт
GL.Enable(EnableCap.TextureCubeMapSeamless);

// ... код, использующий кубические карты ...

// Отключение бесшовной фильтрации кубических карт (если необходимо)
GL.Disable(EnableCap.TextureCubeMapSeamless);

// ... остальной код ...
```

# Как работает Seamless Filtering?

Когда включено бесшовное фильтрование, для определения цвета текселя на границе грани куба используются специальные правила. Эти правила учитывают тексели с соседних граней, сглаживая переход и устраняя видимые швы.

При этом правила отсечения текстурных координат (wrap modes), применяемые к отдельным 2D-текстурам, игнорируются. Вместо них используются специальные алгоритмы для определения соседних текселей на смежных гранях.



Слева: Кубическая карта, отфильтрованная без бесшовной фильтрации. Видны швы и резкие переходы на поверхности шара.

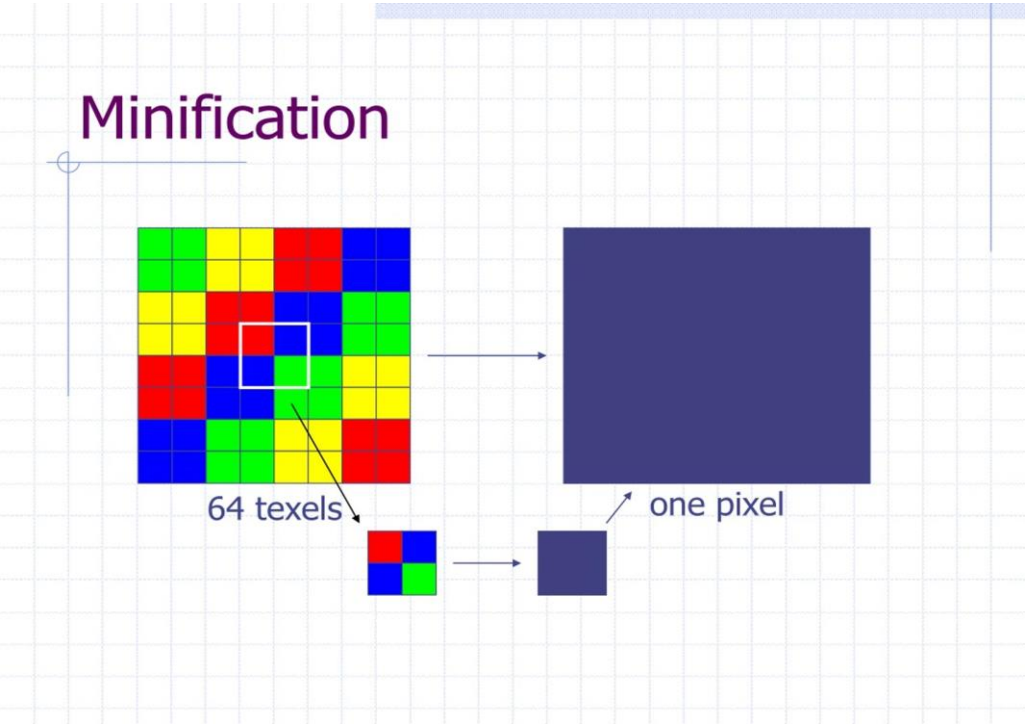
Справа: Кубическая карта, отфильтрованная с бесшовной фильтрацией. Плавные и непрерывные отражения на поверхности шара, без видимых швов.

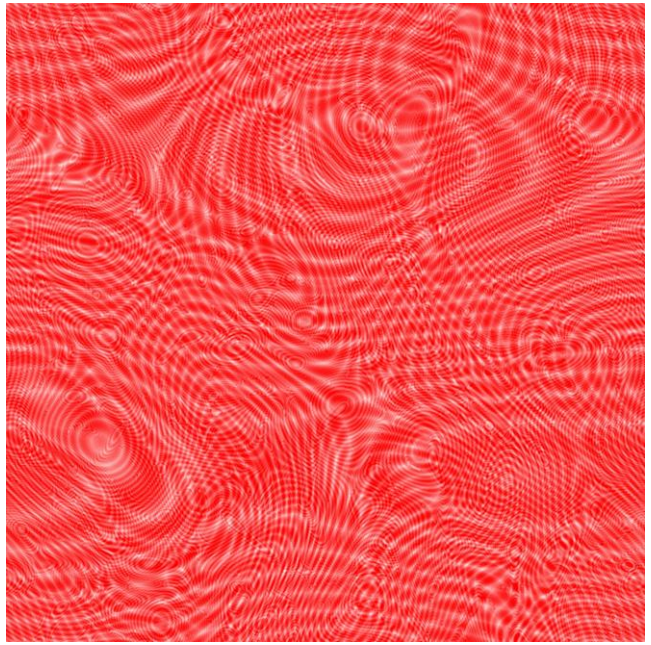
Источник: Игнасио Кастаньо, "Seamless Cube Map Filtering"

# Texture Minification
# (Минификация текстуры)

# Texture Minification (Минификация текстуры)

- Когда текстура отображается на поверхности, занимающей на экране область меньше, чем сама текстура (то есть, когда много текселей текстуры проецируются на один пиксель экрана), возникает необходимость в минификации текстуры.
- Минификация текстуры – это процесс выбора или комбинирования нескольких текселей текстуры для определения цвета пикселя на экране. Это нужно для предотвращения артефактов (муар, зазубрины) и оптимизации производительности.



Пример муара

# Анизотропная фильтрация

Анизотропная фильтрация – это метод улучшения качества текстур, которые просматриваются под углом. В таких случаях проекция текселей на экран становится вытянутой, и обычная фильтрация может привести к размытию. Анизотропная фильтрация учитывает эту вытянутость и применяет более сложное усреднение текселей, беря больше образцов вдоль длинной оси проекции и меньше – вдоль короткой.

В OpenGL есть параметр, управляющий уровнем анизотропной фильтрации: TEXTURE_MAX_ANISOTROPY

Когда TEXTURE_MAX_ANISOTROPY равен 1.0, OpenGL использует обычную (изотропную) фильтрацию. Изотропная фильтрация применяет фильтрацию одинаково во всех направлениях.

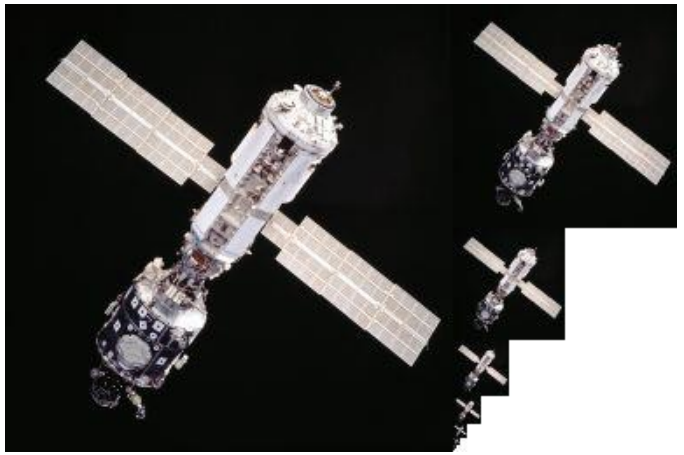
Когда TEXTURE_MAX_ANISOTROPY больше 1.0, OpenGL использует анизотропную фильтрацию, учитывая степень вытянутости проекции, но не больше, чем:

- значение TEXTURE_MAX_ANISOTROPY самой текстуры;
- максимальное значение анизотропии, поддерживаемое видеокартой (MAX_TEXTURE_MAX_ANISOTROPY).


- Без анизотропной фильтрации (или с низким значением): Текстуры, расположенные под углом к камере, выглядят размытыми и менее детализированными.
- С высокой анизотропной фильтрацией (высокое значение TEXTURE_MAX_ANISOTROPY): Текстуры остаются чёткими и детализированными даже при взгляде под острым углом. Это особенно заметно на таких поверхностях, как пол, дороги, стены, уходящие вдаль.

# Управление уровнями детализации (LOD)

- OpenGL позволяет управлять тем, как анизотропная фильтрация использует мипмапы.
- Использование мипмапов зависит от метода минификации текстуры.
- Параметры TEXTURE_BASE_LEVEL (по умолчанию 0) и TEXTURE_MAX_LEVEL (по умолчанию 1000) задают диапазон используемых уровней мипмапов. TEXTURE_BASE_LEVEL – самый детальный уровень, TEXTURE_MAX_LEVEL – самый размытый.
- Параметры TEXTURE_MAX_LOD (максимальный уровень детализации) и TEXTURE_MIN_LOD (минимальный уровень детализации) дополнительно управляют выбором уровня детализации (Level of Detail - LOD). LOD определяет, какой мипмап будет выбран для рендеринга объекта в зависимости от его размера и расстояния до камеры. Ограничивая значения LOD, можно влиять на четкость текстур на разных расстояниях.
- (* ) Мипмапы – это предварительно сгенерированные уменьшенные копии текстуры. Каждый следующий уровень мипмапа вдвое меньше предыдущего. Чем выше уровень мипмапа (больший индекс), тем меньше разрешение и детализация текстуры.

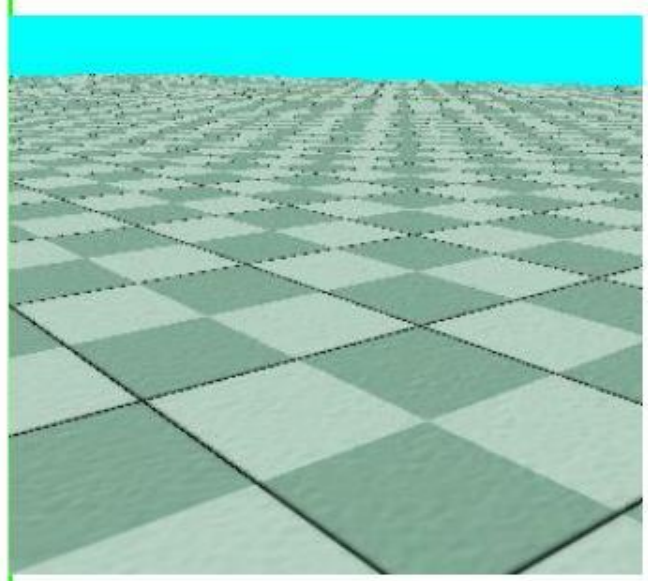


# Texture Minification

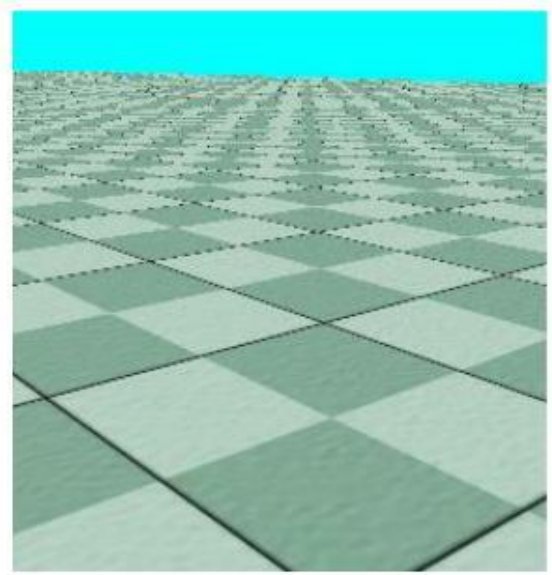


Multiple textures in a single pixel

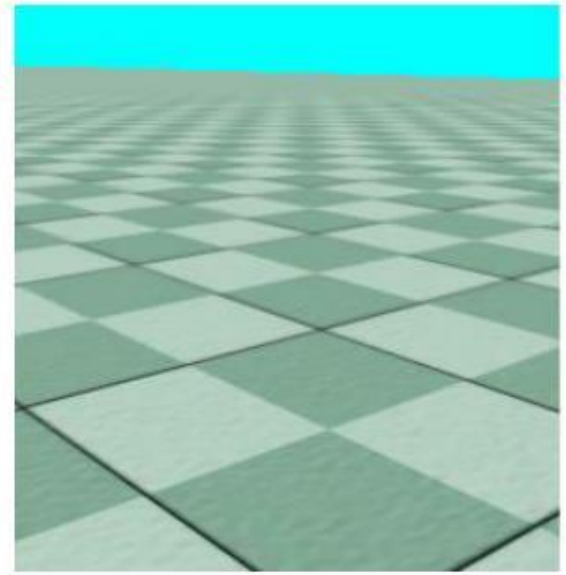
Solution:



Nearest neighbour



Bilinear blending



Mipmapping

# Один из подходов к анизотропной фильтрации

- В отличие от ранее используемого единственного масштабного коэффициента P(*), теперь вычисляются два коэффициента: ρx и ρy на основе частных производных текстурных координат (u, v) по экранным координатам (x, y):

$$\rho_x = \sqrt{\frac{\partial u}{\partial x}^2 + \frac{\partial v}{\partial x}^2}$$
$$\rho_y = \sqrt{\frac{\partial u}{\partial y}^2 + \frac{\partial v}{\partial y}^2}$$
$$\rho_{max} = \max(\rho_x, \rho_y)$$
$$\rho_{min} = \min(\rho_x, \rho_y)$$

Эти коэффициенты отражают степень растяжения или сжатия текстуры по осям x и y на экране.

(*) Раньше уровень детализации и фильтрация определялись единым коэффициентом P.
Этот усредненный подход не справлялся с анизотропной проекцией, вызывая размытие текстур, видимых под углом.

Далее вычисляется коэффициент анизотропии N (насколько сильно вытянута проекция):

$$N = \min(\left\lceil \frac{\rho_{max}}{\rho_{min}} \right\rceil, maxAniso)$$

где maxAniso - меньшее из значения TEXTURE_MAX_ANISOTROPY и MAX_TEXTURE_MAX_ANISOTROPY (определяется реализацией).

Затем вычисляется уровень детализации λ’: $\lambda' = log_2(\frac{\rho_{max}}{N})$

# Оптимизация вычислений

- Для повышения производительности реализации могут аппроксимировать ρx и ρy функциями fx и fy.
- Данные функции должны удовлетворять следующим ограничениям:

1. $f_x$ is continuous and monotonically increasing in $\left|\frac{\partial u}{\partial x}\right|$ and $\left|\frac{\partial v}{\partial x}\right|$.
2. $f_y$ is continuous and monotonically increasing in $\left|\frac{\partial u}{\partial y}\right|$ and $\left|\frac{\partial v}{\partial y}\right|$.
3. $\max(\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial v}{\partial x}\right|) \leq f_x \leq \sqrt{2}\left(\left|\frac{\partial u}{\partial x}\right| + \left|\frac{\partial v}{\partial x}\right|\right)$.
4. $\max(\left|\frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial y}\right|) \leq f_y \leq \sqrt{2}\left(\left|\frac{\partial u}{\partial y}\right| + \left|\frac{\partial v}{\partial y}\right|\right)$.

# Многократная выборка для повышения качества

- Вместо одной выборки текселя, анизотропная фильтрация производит N выборок внутри проекции пикселя на текстуру на выбранном уровне мипмапа (λ).
- Окончательное текстурное значение ($\tau_{aniso}$) вычисляется на основе усреднения этих N выборок.
- Формулы для вычисления $\tau_{aniso}$ зависят от ориентации проекции (Px vs Py) и представляют собой среднее значение N выборок текстуры (τ), взятых со смещением:

$$\tau_{aniso} = \begin{cases} \frac{1}{N}\sum_{i=1}^{N} \tau(u(x - \frac{1}{2} + \frac{i}{N+1}, y), v(x - \frac{1}{2} + \frac{i}{N+1}, y)), & P_x > P_y \\ \frac{1}{N}\sum_{i=1}^{N} \tau(u(x, y - \frac{1}{2} + \frac{i}{N+1}), v(x, y - \frac{1}{2} + \frac{i}{N+1})), & P_y \geq P_x \end{cases}$$

- Для оптимизации может использоваться аппроксимация с равномерно распределенными смещениями в текстурном пространстве:

$$\tau_{aniso} = \begin{cases} \frac{1}{N}\sum_{i=1}^{N} \tau(u(x,y) + \frac{\partial u}{\partial x}(\frac{i}{N+1} - \frac{1}{2}), v(x,y) + \frac{\partial v}{\partial x}(\frac{i}{N+1} - \frac{1}{2})), & P_x > P_y \\ \frac{1}{N}\sum_{i=1}^{N} \tau(u(x,y) + \frac{\partial u}{\partial y}(\frac{i}{N+1} - \frac{1}{2}), v(x,y) + \frac{\partial v}{\partial y}(\frac{i}{N+1} - \frac{1}{2})), & P_y \geq P_x \end{cases}$$

- Этот подход использует производные текстурных координат для определения смещений.

# Итог:

Проблема:

При уменьшении масштаба текстуры (минификации) множество текселей проецируются на один пиксель экрана. Это приводит к потере детализации и размытию изображения, особенно заметному при взгляде на текстуру под углом.

Решение:

Минификация текстуры – это процесс выбора или комбинирования текселей для определения цвета пикселя. Для борьбы с размытием при угловом обзоре применяется анизотропная фильтрация. Этот метод анализирует вытянутость проекции текселя и выполняет "умное" усреднение, сохраняя детали.

Основные Этапы Анизотропной Фильтрации:

- Вычисление масштабных коэффициентов (ρx, ρy): Определение степени растяжения/сжатия текстуры вдоль осей экрана.
- Определение степени анизотропии (N): Расчет отношения максимального и минимального масштабов с учетом ограничения TEXTURE_MAX_ANISOTROPY (максимальная анизотропия).
- Определение уровня детализации (λ'): Выбор уровня мипмапа, наилучшего для данной степени анизотропии.
- Многократная выборка: Получение нескольких образцов текселей внутри проекции пикселя.
- Усреднение выборок ($\tau_{aniso}$): Комбинирование образцов для получения окончательного цвета пикселя.

# Scale Factor and Level-of-Detail (Масштабный фактор и уровень детализации)

# Масштабный фактор и уровень детализации

В предыдущей теме мы говорили об анизотропной фильтрации для улучшения качества уже выбранной текстуры. Но как OpenGL изначально решает, какую версию текстуры (какой уровень детализации) использовать?

Это определяется двумя основными факторами:

- Масштабный фактор (ρ(x, y)): Показывает, во сколько раз проекция текстуры на пиксель больше или меньше исходного размера текстуры.
- Параметр уровня детализации (λ(x, y)): Итоговое число, определяющее, какой уровень мипмапа будет использован.

Базовый уровень детализации:

Базовый уровень детализации ($\lambda_{base}(x, y)$) – это начальная оценка оптимального уровня мипмапа.  Вычисляется она по следующей формуле:

$$\lambda_{base}(x, y) = \log_2[\rho(x, y)]$$

# Коррекция и ограничение LOD

OpenGL предоставляет несколько параметров для управления выбором уровня детализации (LOD):

- Смещение LOD:
  - $bias_{texobj}$ (TEXTURE_LOD_BIAS): Смещение, применяемое ко всей текстуре.
    - Положительные значения: размытие (использование более высоких уровней мипмапов).
    - Отрицательные значения: резкость (использование более низких уровней).
  - $bias_{shader}$: Дополнительное смещение, передаваемое из шейдера (обычно 0).
  - Итоговое смещение ограничивается диапазоном [-biasmax, biasmax], где biasmax - MAX_TEXTURE_LOD_BIAS.
- Ограничение LOD:
  - $lod_{min}$ (TEXTURE_BASE_LEVEL): Самый детальный используемый уровень мипмапа.
  - $lod_{max}$ (TEXTURE_MAX_LEVEL): Самый размытый используемый уровень мипмапа.
- Если итоговый LOD (λ):
  - ≤ 0: Текстура увеличена (нужен более детальный уровень).
  - > 0: Текстура уменьшена (нужен менее детальный уровень - мипмап).
- По умолчанию, $lod_{min}$ и $lod_{max}$ выбираются так, чтобы использовать весь доступный диапазон мипмапов.
- Анизотропная фильтрация (если TEXTURE_MAX_ANISOTROPY > 1.0) применяется после выбора LOD для улучшения качества текстуры.

# Отображение координат и смещение текселей

- OpenGL использует функции (s(x, y), t(x, y), r(x, y)) для преобразования экранных координат (x, y) в текстурные координаты (s, t, r).
- Эти функции зависят от геометрии примитива и способа наложения текстуры.
- Смещение текселей:
  - В шейдерах можно использовать смещения (δu, δv, δw) для более точного определения координат выборки текстуры.
  - Формулы для u, v, w зависят от типа текстуры и включают:
    - Базовые текстурные координаты (s, t, r).
    - Масштабированные коэффициенты ($w_s$, $h_s$, $d_s$) для непрямоугольных текстур.
  - Если смещение не используется, δu = δv = δw = 0.
- **Важно**: Значения смещений (δu, δv, δw) должны быть в пределах, определяемых реализацией OpenGL:
  - MIN_PROGRAM_TEXEL_OFFSET, MAX_PROGRAM_TEXEL_OFFSET (для обычных поисков).
  - MIN_PROGRAM_TEXTURE_GATHER_OFFSET, MAX_PROGRAM_TEXTURE_GATHER_OFFSET (для textureGather).
- Выход за пределы приводит к неопределенному результату.

$$u(x,y) = \begin{cases} s(x,y) + \delta_u, & \text{rectangle texture} \\ w_s \times s(x,y) + \delta_u, & \text{otherwise} \end{cases}$$

$$v(x,y) = \begin{cases} t(x,y) + \delta_v, & \text{rectangle texture} \\ h_s \times t(x,y) + \delta_v, & \text{otherwise} \end{cases}$$

$$w(x,y) = d_s \times r(x,y) + \delta_w$$

# Масштабный фактор и проекция пикселя на текстуру

- Проекция области пикселя на текстуру обычно имеет форму эллипса.
- Масштабный фактор ρ отражает размер этой проекции.
- Идеальный ρ соответствует размеру большой оси эллипса, чтобы точно определить степень увеличения/уменьшения текстуры.

# Вычисление масштабного фактора (ρ): Общие принципы

- Уровень детализации (λ) зависит от масштабного фактора (ρ).
- Производные текстурных координат (u, v, w) по экранным координатам (x, y) (∂u/∂x, ∂v/∂x, ∂w/∂x, ∂u/∂y, ∂v/∂y, ∂w/∂y) показывают, как текстурные координаты меняются на экране.
- Эти изменения связаны с тем, насколько "большой" или "маленькой" выглядит текстура.
- **Проблема**: Точное вычисление большой оси эллипса с использованием производных вычислительно затратно.
- **Решение**: OpenGL допускает аппроксимацию ρ функцией f(x, y), удовлетворяющей условиям:

1. $f(x, y)$ is continuous and monotonically increasing in each of $\left|\frac{\partial u}{\partial x}\right|$, $\left|\frac{\partial u}{\partial y}\right|$, $\left|\frac{\partial v}{\partial x}\right|$, $\left|\frac{\partial v}{\partial y}\right|$, $\left|\frac{\partial w}{\partial x}\right|$, and $\left|\frac{\partial w}{\partial y}\right|$
2. $\max(\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|, \left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial v}{\partial y}\right|, \left|\frac{\partial w}{\partial x}\right|, \left|\frac{\partial w}{\partial y}\right|) \leq f(x, y) \leq \sqrt{2}\max(\left|\frac{\partial u}{\partial x}\right| + \left|\frac{\partial v}{\partial x}\right| + \left|\frac{\partial w}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right| + \left|\frac{\partial v}{\partial y}\right| + \left|\frac{\partial w}{\partial y}\right|)$

# Вычисление ρ для полигонов и точек (один из способов)

- Для полигонов и точек один из способов вычисления ρ заключается в определении максимальной скорости изменения текстурных координат вдоль осей x и y экрана:

$$\rho = \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial w}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial w}{\partial y}\right)^2}\right\}$$

Здесь вычисляется длина вектора производных по x и длина вектора производных по y, и выбирается большее из этих значений.

# Вычисление ρ для линий

- Для линий ρ вычисляется с учетом направления линии на экране, определяемого конечными точками (x1, y1) и (x2, y2):

$$\rho = \sqrt{\left(\frac{\partial u}{\partial x}\Delta x + \frac{\partial u}{\partial y}\Delta y\right)^2 + \left(\frac{\partial v}{\partial x}\Delta x + \frac{\partial v}{\partial y}\Delta y\right)^2 + \left(\frac{\partial w}{\partial x}\Delta x + \frac{\partial w}{\partial y}\Delta y\right)^2} \Big/ l$$

- Δx = x1 – x2 и Δy = y1 – y2 - разности координат конечных точек.
- $l = \sqrt{\Delta x^2 + \Delta y^2}.$ - длина сегмента линии.

Эта формула учитывает, как текстурные координаты меняются вдоль направления линии на экране.

# Альтернативная аппроксимация ρ

- В качестве альтернативной аппроксимации ρ может использоваться функция f(x, y), где:

1. $f(x, y)$ is continuous and monotonically increasing in each of $|\partial u/\partial x|$, $|\partial u/\partial y|$, $|\partial v/\partial x|$, $|\partial v/\partial y|$, $|\partial w/\partial x|$, and $|\partial w/\partial y|$
2. Let

$$m_u = \max\left\{\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|\right\}$$

$$m_v = \max\left\{\left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial v}{\partial y}\right|\right\}$$

$$m_w = \max\left\{\left|\frac{\partial w}{\partial x}\right|, \left|\frac{\partial w}{\partial y}\right|\right\}.$$

Then $\max\{m_u, m_v, m_w\} \leq f(x, y) \leq m_u + m_v + m_w$.

# Итог:

Проблема:

- При рендеринге 3D-объектов текстуры масштабируются: увеличиваются (становятся больше на экране) или уменьшаются (становятся меньше). При уменьшении множество текселей проецируются на один пиксель, что приводит к потере детализации и размытию.

Решение:

- OpenGL использует масштабный фактор (ρ) для оценки степени масштабирования текстуры и определяет уровень детализации (LOD, λ). LOD указывает, какой уровень мипмапа использовать при уменьшении или как интерполировать тексели при увеличении.

Основные этапы:

- Вычисление масштабного фактора (ρ(x, y)): Оценка изменения текстурных координат относительно экранных координат. Методы вычисления ρ различаются для точек/полигонов и линий.
- Определение базового уровня детализации ($\lambda_{base}$): Расчет на основе логарифма от масштабного фактора (log2(ρ)).
- Коррекция LOD (λ'): Применение смещений (TEXTURE_LOD_BIAS, $bias_{shader}$) для настройки уровня детализации.
- Ограничение LOD (λ): Приведение LOD к диапазону, определяемому $lod_{min}$ (TEXTURE_BASE_LEVEL) и $lod_{max}$ (TEXTURE_MAX_LEVEL).
- Отображение координат (s(x, y), t(x, y), r(x, y)): Определение соответствия между экранными и текстурными координатами. Зависит от геометрии и способа наложения.
- Выбор текселя: Выборка цвета из мипмапа или интерполяция текселей на основе итогового LOD и текстурных координат.

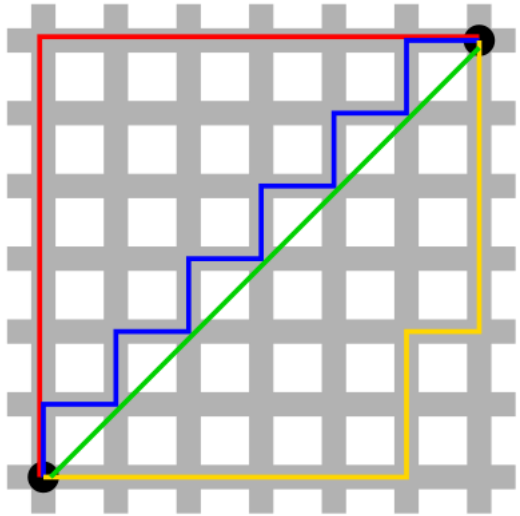
# Coordinate Wrapping and Texel Selection (Оборачивание координат и выбор текселей)

# Оборачивание координат и выбор текселей

- После вычисления текстурных координат u(x, y), v(x, y) и w(x, y) они могут быть ограничены и "обернуты".
- Это делается перед выборкой текселя и зависит от режимов обертывания текстуры.
- Обозначим: u'(x, y) = u(x, y), v'(x, y) = v(x, y), w'(x, y) = w(x, y).

# Выбор текселя при TEXTURE_MIN_FILTER = NEAREST

- TEXTURE_MIN_FILTER определяет, как выбирается тексель.
- Если TEXTURE_MIN_FILTER = NEAREST:
  - Выбирается тексель на уровне $level_{base}$, ближайший к (u', v', w') (по манхэттенскому расстоянию*).
  - Вычисляются целые числа i, j, k:
    - i = wrap(|u'(x, y)|)
    - j = wrap(|v'(x, y)|)
    - k = wrap(|w'(x, y)|)
  - wrap() - функция, определяющая режим обертывания.



(*)Манхеттенское расстояние не зависит от выбранного маршрута
D = ∑|x1,i - x2,i|, где x1,i и x2,i – i-я координата первого и второго объекта соответственно

# Выбор текселя (продолжение)

В зависимости от типа текстуры:

- 3D-текстура: тексель в позиции (i, j, k).
- 2D, 2D-массив, прямоугольная, кубическая: тексель в позиции (i, j) (k не имеет значения).
- 1D, 1D-массив: тексель в позиции i (j и k не имеют значения).

Для 1D и 2D-массивов тексель выбирается из слоя l:

$$l = \begin{cases} clamp(RNE(t), 0, h_s - 1), & \text{for one-dimensional array textures} \\ clamp(RNE(r), 0, d_s - 1), & \text{for two-dimensional array textures} \end{cases}$$

где:
RNE() - округление до ближайшего четного.
clamp(x, min, max) - ограничение значения x диапазоном [min, max].

# Режимы обертывания текстурных координат

- Функция wrap() определяет, как обрабатываются координаты, выходящие за границы текстуры.
- Режимы обертывания и их результаты:

| Wrap mode | Result of $wrap(coord)$ |
|---|---|
| `CLAMP_TO_EDGE` | $clamp(coord, 0, size - 1)$ |
| `CLAMP_TO_BORDER` | $clamp(coord, -1, size)$ |
| `REPEAT` | $coord \bmod size$ |
| `MIRRORED_REPEAT` | $(size - 1) - mirror(coord \bmod (2 \times size)) - size)$ |
| `MIRROR_CLAMP_TO_EDGE` | $clamp(mirror(coord), 0, size - 1)$ |

Функция mirror(a) возвращает a, если a >= 0, и (1 + a) в противном случае.

GL_REPEAT

GL_MIRRORED_REPEAT

GL_CLAMP_TO_EDGE

GL_CLAMP_TO_BORDER

# Граничные тексели

- Если выбранная позиция текселя (i, j, k), (i, j) или i является граничной, используются значения TEXTURE_BORDER_COLOR.
- Граничные условия:
  - $i < 0$ или $i \geq ws$
  - $j < 0$ или $j \geq hs$
  - $k < 0$ или $k \geq ds$
- Для цветных текстур TEXTURE_BORDER_COLOR интерпретируется как RGBA.
- Для текстур глубины используется первая компонента TEXTURE_BORDER_COLOR.
- Тип граничных значений должен быть совместим с типом текстуры.

# Выбор текселя при TEXTURE_MIN_FILTER = LINEAR

- Если TEXTURE_MIN_FILTER = LINEAR, выбирается куб из 2 × 2 × 2 текселей.
- Вычисляются координаты углов куба:
  - $i_0 = wrap(|u' - 1/2|)$
  - $j_0 = wrap(|v' - 1/2|)$
  - $k_0 = wrap(|w' - 1/2|)$
  - $i_1 = wrap(|u' - 1/2| + 1)$
  - $j_1 = wrap(|v' - 1/2| + 1)$
  - $k_1 = wrap(|w' - 1/2| + 1)$
- Вычисляются дробные части координат (frac(x) - дробная часть x):
  - $\alpha = frac(u' - 1/2)$
  - $\beta = frac(v' - 1/2)$
  - $\gamma = frac(w' - 1/2)$

Для 3D-текстуры значение текселя τ вычисляется как:

$$\begin{aligned}\tau = &(1-\alpha)(1-\beta)(1-\gamma)\tau_{i_0 j_0 k_0} + \alpha(1-\beta)(1-\gamma)\tau_{i_1 j_0 k_0} \\ &+ (1-\alpha)\beta(1-\gamma)\tau_{i_0 j_1 k_0} + \alpha\beta(1-\gamma)\tau_{i_1 j_1 k_0} \\ &+ (1-\alpha)(1-\beta)\gamma\tau_{i_0 j_0 k_1} + \alpha(1-\beta)\gamma\tau_{i_1 j_0 k_1} \\ &+ (1-\alpha)\beta\gamma\tau_{i_0 j_1 k_1} + \alpha\beta\gamma\tau_{i_1 j_1 k_1}\end{aligned}$$

Для 2D, 2D-массивов, прямоугольных и кубических текстур:

$$\begin{aligned}\tau = &(1-\alpha)(1-\beta)\tau_{i_0 j_0} + \alpha(1-\beta)\tau_{i_1 j_0} \\ &+ (1-\alpha)\beta\tau_{i_0 j_1} + \alpha\beta\tau_{i_1 j_1}\end{aligned}$$

Для 2D-массивов все тексели берутся из слоя l:

$$l = clamp(\left\lfloor r + \frac{1}{2} \right\rfloor, 0, d_s - 1).$$

GL_NEAREST

GL_LINEAR