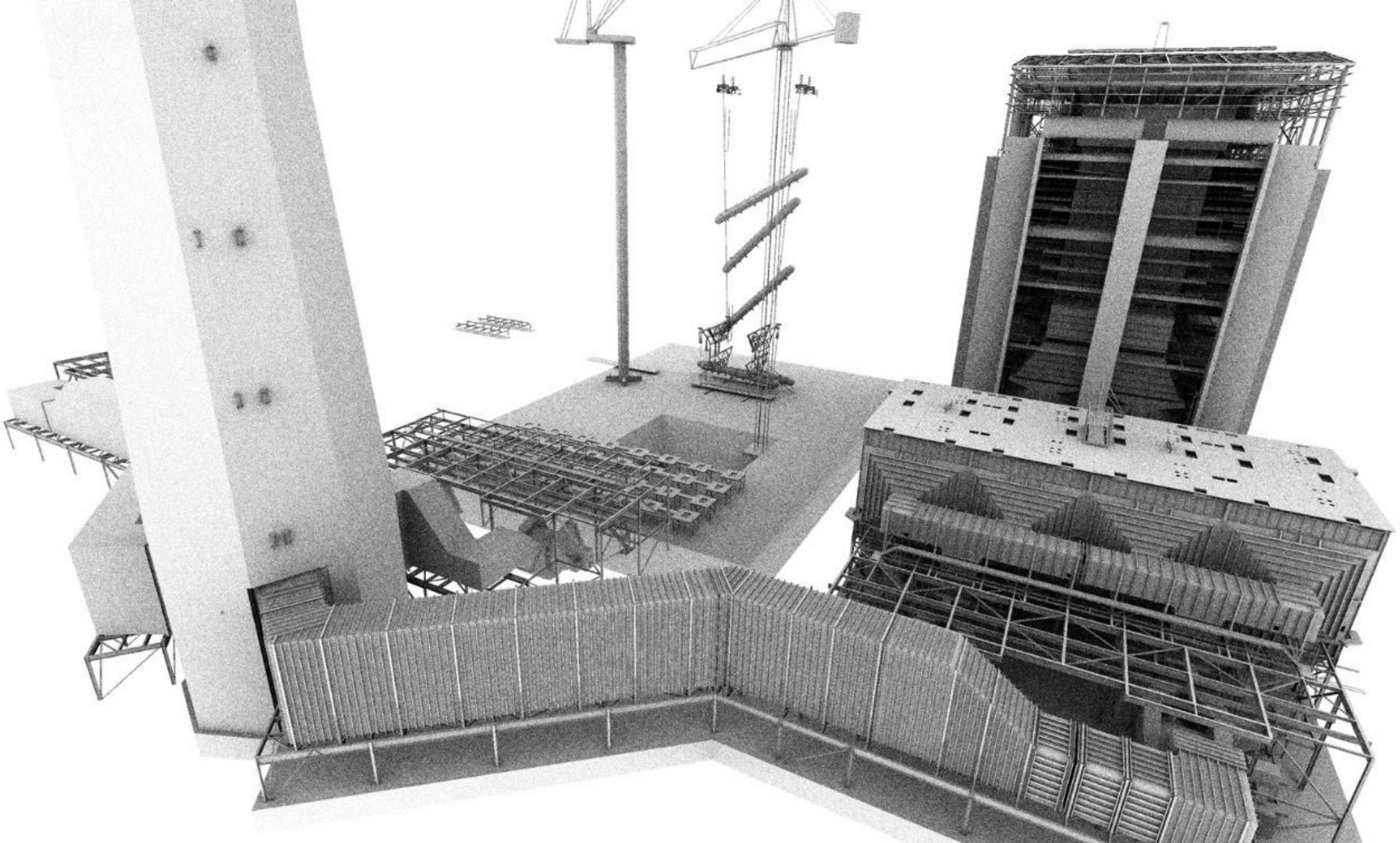
CS Space
Клуб технологий и науки

Вычисления на видеокартах

Лекция 12 - **Nanite** (Unreal Engine 5)

- Виртуальная текстура и геометрия
- QSlim упрощение геометрии
- Hierarchical Z Buffer - HZB
- Deferred rendering
- Rigging, деревья

@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный



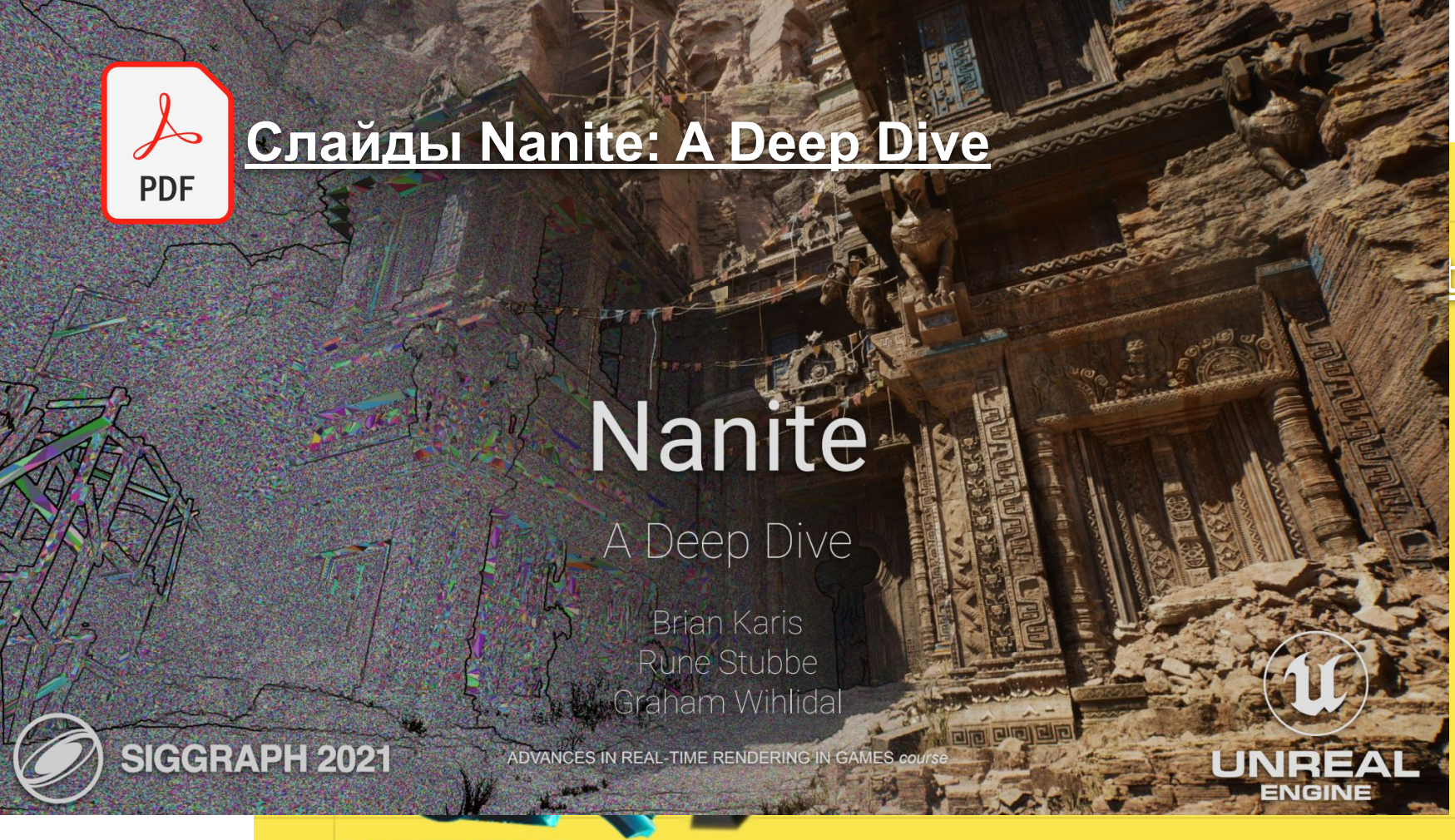

HSE
Team
ITMO
Team
HSE
Team

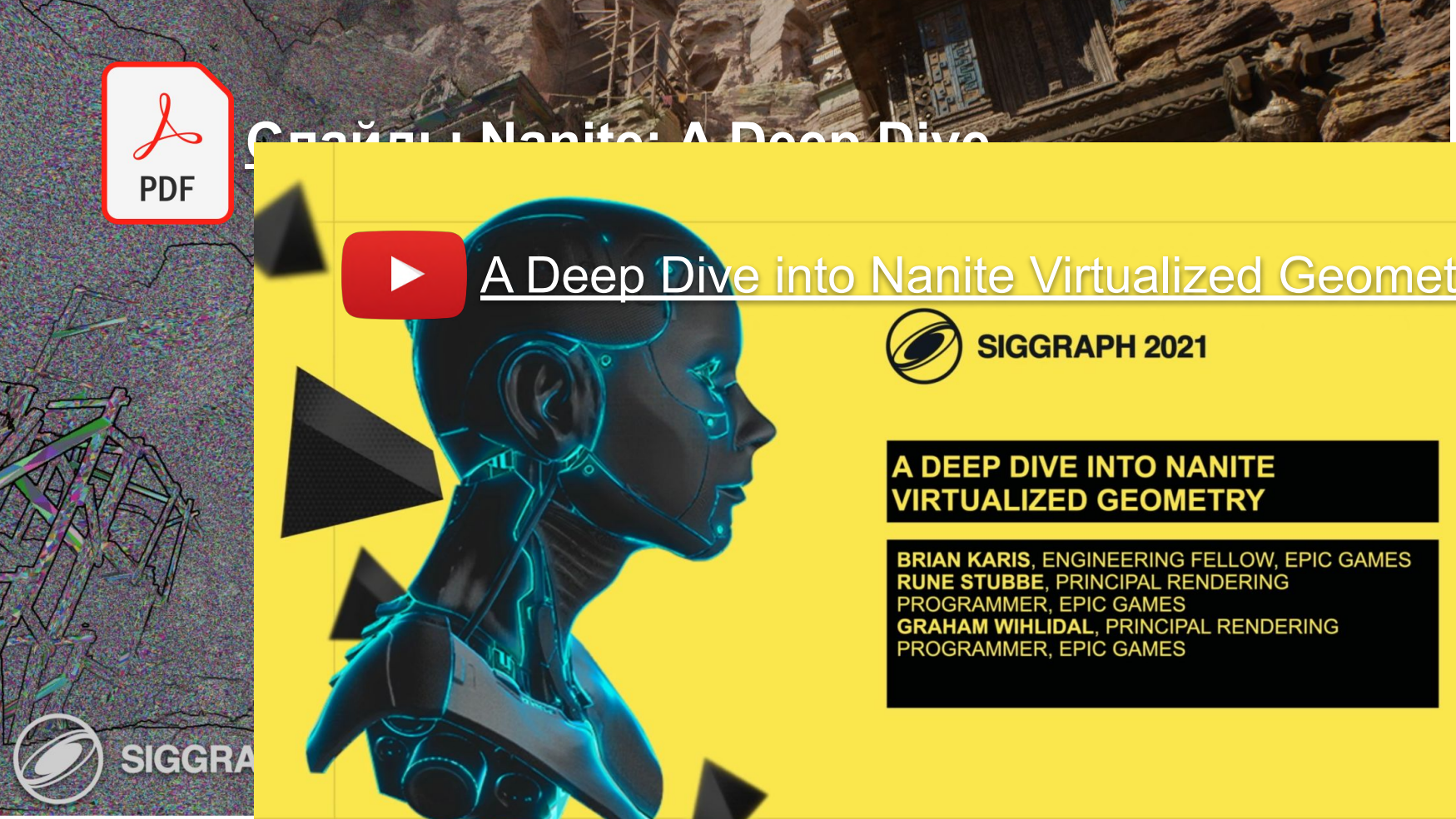
LBVH + Ray Tracing на **Tesla T4**

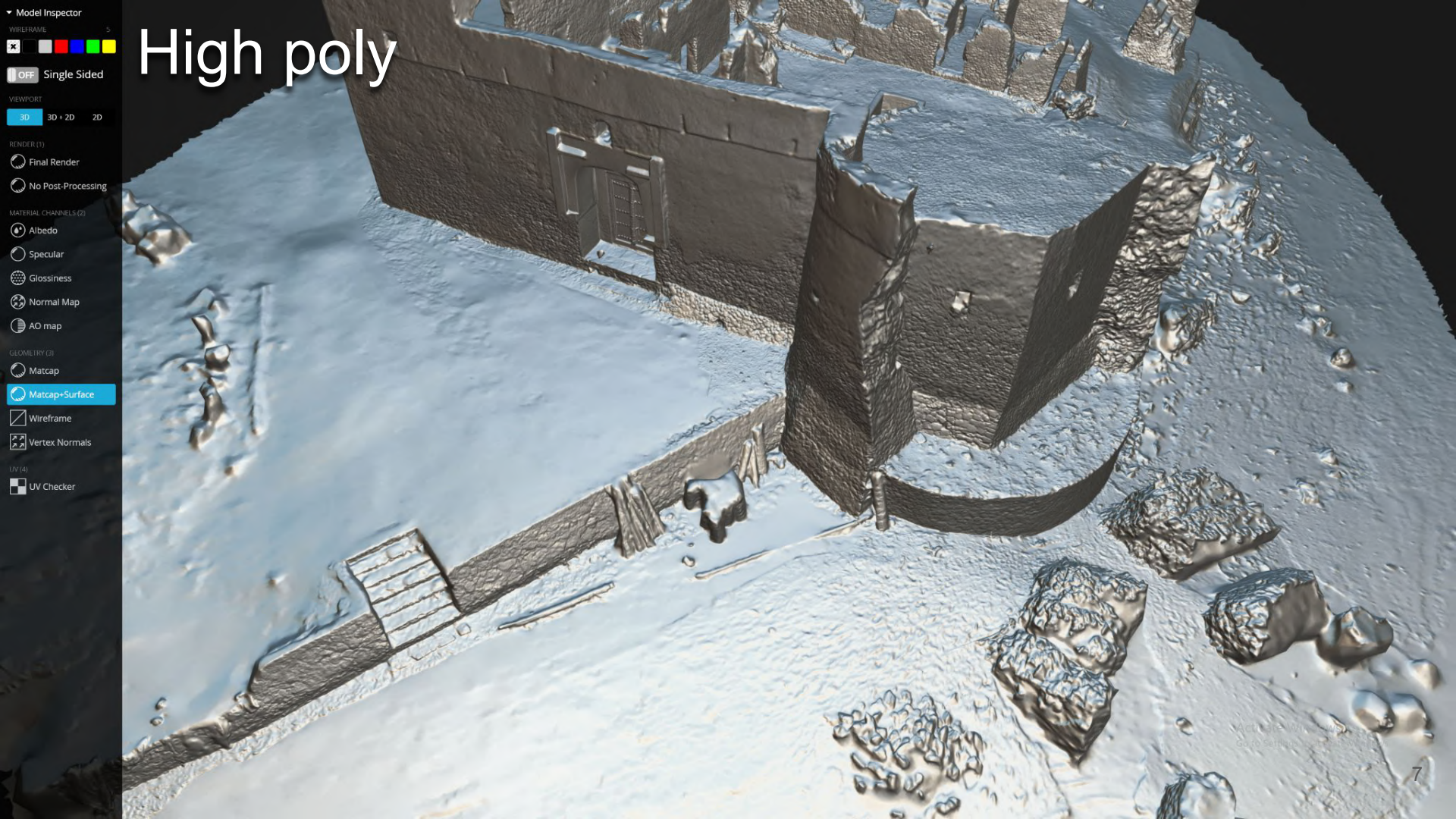
- Тяньшэн Цю - ITMO Team - **40183 coolness** = **139 MTris/s x 290 MRays/s**, OpenCL
- Ostapenko Vladislav - HSE Team - **34832 coolness** = **165 MTris/s x 212 MRays/s**, OpenCL
- Sanan Kornyakov - HSE Team - **19683 coolness** = **83 MTris/s x 238 MRays/s**, OpenCL

P.S. CPU LBVH построение: **1.9 MTris/s**

Слайды Nanite: A Deep Dive




PDF
Слайды Nanite: A Deep Dive
A Deep Dive into Nanite Virtualized Geometry
SIGGRAPH 2021
A DEEP DIVE INTO NANITE VIRTUALIZED GEOMETRY
BRIAN KARIS, ENGINEERING FELLOW, EPIC GAMES
RUNE STUBBE, PRINCIPAL RENDERING PROGRAMMER, EPIC GAMES
GRAHAM WIHLIDAL, PRINCIPAL RENDERING PROGRAMMER, EPIC GAMES
SIGGRA
© 2021 SIGGRAPH. ALL RIGHTS RESERVED.
SIGGRAPH 2021 ADVANCES IN REAL-TIME RENDERING IN GAMES course
6

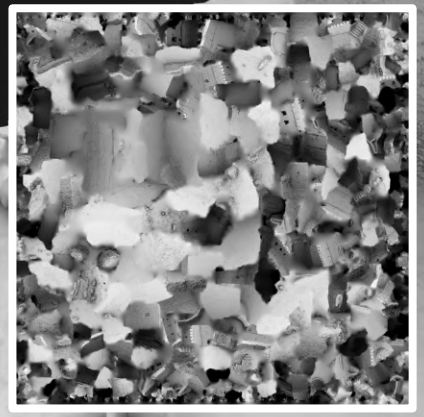

Model Inspector
WIREFRAME
5
OFF
Single Sided
VIEWPORT
3D
3D + 2D
2D
RENDER (1)
Final Render
No Post-Processing
MATERIAL CHANNELS (2)
Albedo
Specular
Glossiness
Normal Map
AO map
GEOMETRY (3)
Matcap
Matcap+Surface
Wireframe
Vertex Normals
UV (4)
UV Checker
High poly

Albedo/Diffuse map

Albedo/Diffuse map

Как добавить реалистичности?

Ambient Occlusion map (AO map)



Model Inspector
WIREFRAME
Single Sided
VIEWPORT
3D 3D + 2D 2D
RENDER (1)
Final Render
No Post-Processing
MATERIAL CHANNELS (2)
Albedo
Specular
Glossiness
Normal Map
AO map
GEOMETRY (3)
Matcap
Matcap+Surface
Wireframe
Vertex Normals
UV (4)
UV Checker

High poly

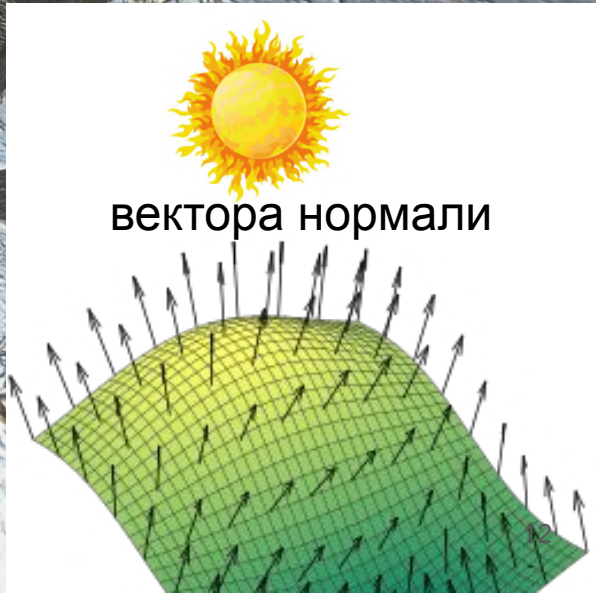
Как оставить детальность геометрии?

Как человек замечает геометрию?

High poly

Как оставить детальность геометрии?

Как человек замечает геометрию?



Low poly

Model Inspector

WIREFRAME 5

OFF Single Sided

VIEWPORT

3D | 3D + 2D | 2D

RENDER (1)

Final Render

No Post-Processing

MATERIAL CHANNELS (2)

Albedo

Specular

Glossiness

Normal Map

AO map

GEOMETRY (3)

Matcap

Matcap+Surface

Wireframe

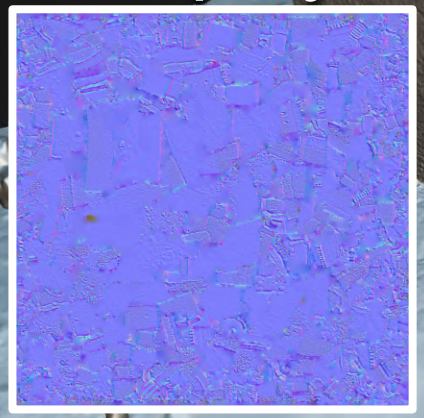
Vertex Normals

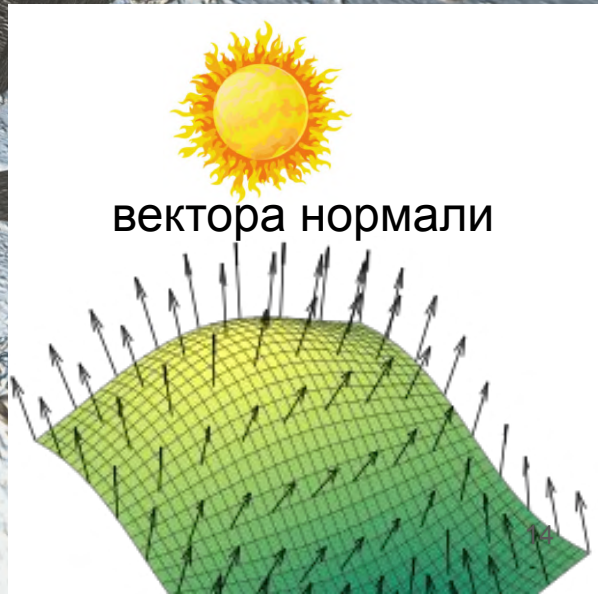
UV (4)

UV Checker

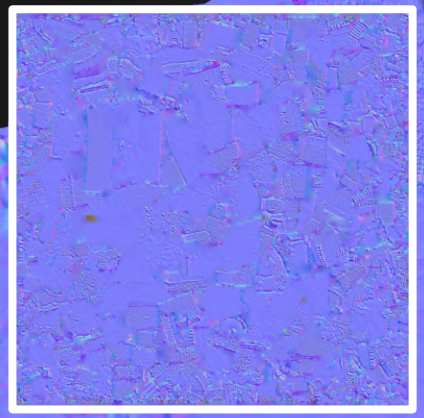
Select an annotation

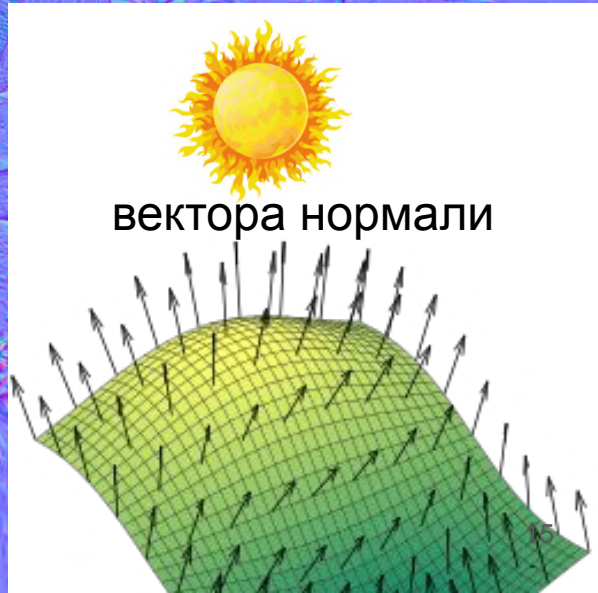
Low poly + Normal map





Normal map






Model Inspector
WIREFRAME
Single Sided
VIEWPORT
3D
3D + 2D
2D
RENDER (1)
Final Render
No Post-Processing
MATERIAL CHANNELS (2)
Albedo
Specular
Glossiness
Normal Map
AO map
GEOMETRY (3)
Matcap
Matcap+Surface
Wireframe
Vertex Normals
UV (4)
UV Checker
Low poly
В чем задача 3D моделлера?
Select an annotation


Model Inspector
WIREFRAME
Single Sided
VIEWPORT
3D
3D + 2D
2D
RENDER (1)
Final Render
No Post-Processing
MATERIAL CHANNELS (2)
Albedo
Specular
Glossiness
Normal Map
AO map
GEOMETRY (3)
Matcap
Matcap+Surface
Wireframe
Vertex Normals
UV (4)
UV Checker
Low poly
В чем задача 3D моделлера?
Как минимум от 3D моделлера требуется:
- распределить доступный бюджет треугольников
- не потерять в Low poly важные детали - шпиль/копье
- тонко чувствовать что бросается игроку в глаза
Select an annotation

Глава 1: Виртуальные текстуры

MegaTexture, virtual texture, texture streaming

История GPU и **виртуальных текстур**

До **1996** года 3D графика на CPU (DOOM, Quake, Nukem 3D)
1995-1997 - становятся популярны 3D-ускорители с **Fixed pipeline**
2000-2002 - в DirectX 8.0 и OpenGL появляются программируемые ***шейдеры***
2002-2004 - Cg (NVIDIA) и Brook (Stanford) - зарождение **GPGPU**

2006-2008 - Close-to-Metal (ATI/AMD), **CUDA** (NVIDIA), **OpenCL** (Khronos)

2012 - Compute Shaders в OpenGL
2016 - Vulkan

История GPU и **виртуальных текстур**

До **1996** года 3D графика на CPU (DOOM, Quake, Nukem 3D)

1995-1997 - становятся популярны 3D-ускорители с **Fixed pipeline**

2000-2002 - в DirectX 8.0 и OpenGL появляются программируемые ***шейдеры***

2002-2004 - Cg (NVIDIA) и Brook (Stanford) - зарождение **GPGPU**

2006-2008 - Close-to-Metal (ATI/AMD), **CUDA** (NVIDIA), **OpenCL** (Khronos)

→ **2007 - *MegaTexture*,** ET: Quake Wars, термин ввел John Carmack, Splash Damage

2012 - Compute Shaders в OpenGL

2016 - Vulkan

История GPU и **виртуальных текстур**

До **1996** года 3D графика на CPU (DOOM, Quake, Nukem 3D)

1995-1997 - становятся популярны 3D-ускорители с **Fixed pipeline**

2000-2002 - в DirectX 8.0 и OpenGL появляются программируемые ***шейдеры***

2002-2004 - Cg (NVIDIA) и Brook (Stanford) - зарождение **GPGPU**

2006-2008 - Close-to-Metal (ATI/AMD), **CUDA** (NVIDIA), **OpenCL** (Khronos)

→ **2007 - *MegaTexture*,** ET: Quake Wars, термин ввел John Carmack, Splash Damage

→ **2011 -** Brink, Rage, John Carmack, id Tech

2012 - Compute Shaders в OpenGL

2016 - Vulkan

История GPU и **виртуальных текстур**

До **1996** года 3D графика на CPU (DOOM, Quake, Nukem 3D)

1995-1997 - становятся популярны 3D-ускорители с **Fixed pipeline**

2000-2002 - в DirectX 8.0 и OpenGL появляются программируемые ***шейдеры***

2002-2004 - Cg (NVIDIA) и Brook (Stanford) - зарождение **GPGPU**

2006-2008 - Close-to-Metal (ATI/AMD), **CUDA** (NVIDIA), **OpenCL** (Khronos)

→ **2007 - *MegaTexture*,** ET: Quake Wars, термин ввел John Carmack, Splash Damage

→ **2011 -** Brink, Rage, John Carmack, id Tech

2012 - Compute Shaders в OpenGL

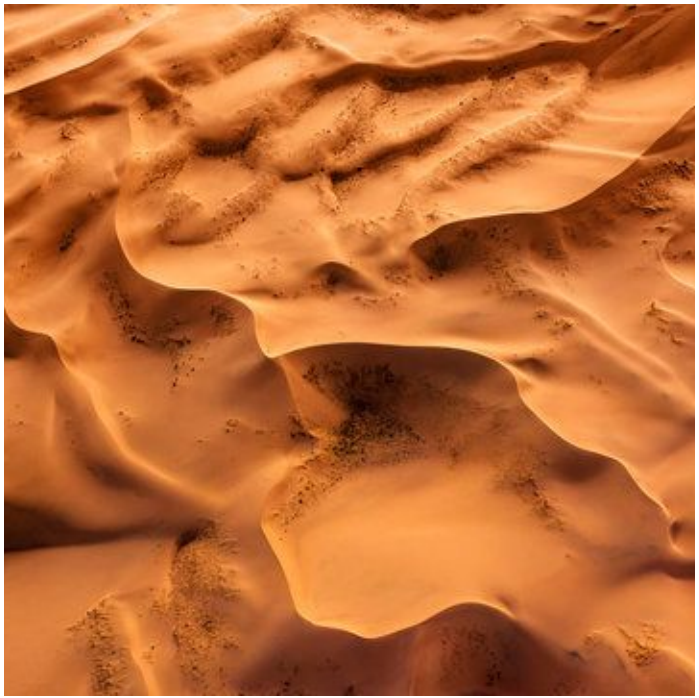
2016 - Vulkan

→ **2020 - Nanite, Brian Karis, Unreal Engine 5**

UNREAL
ENGINE

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура
В те времена - 16К х 16К

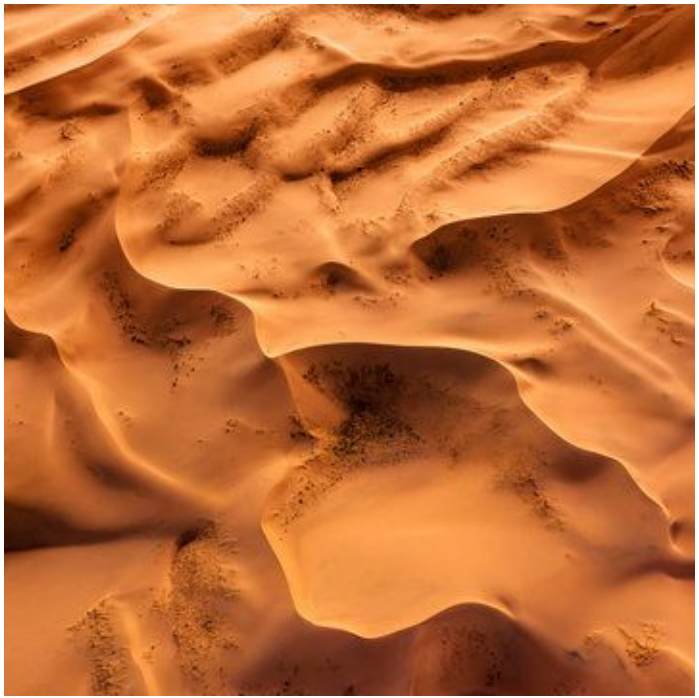


Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16К х 16К



Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16К x 16К

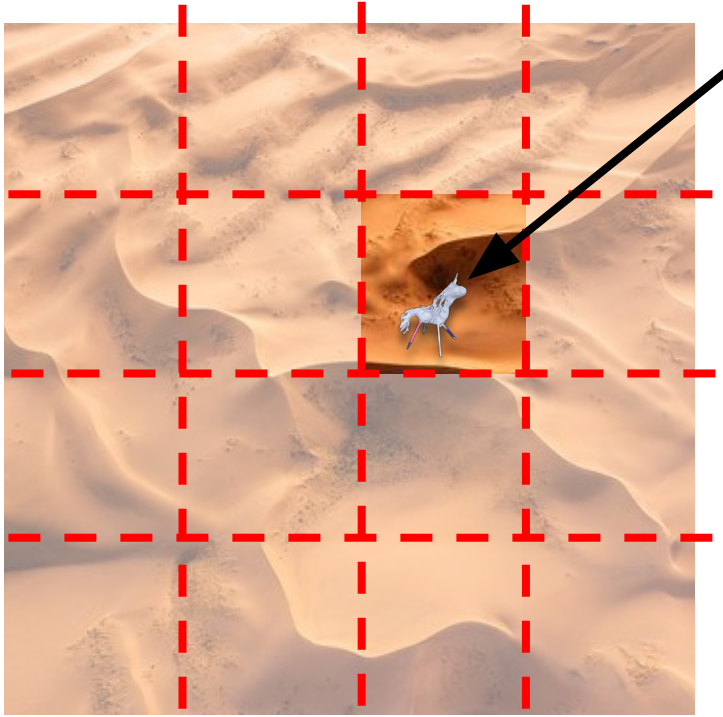
Игрок видит мир вокруг:

Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16К x 16К



Игрок видит мир вокруг:

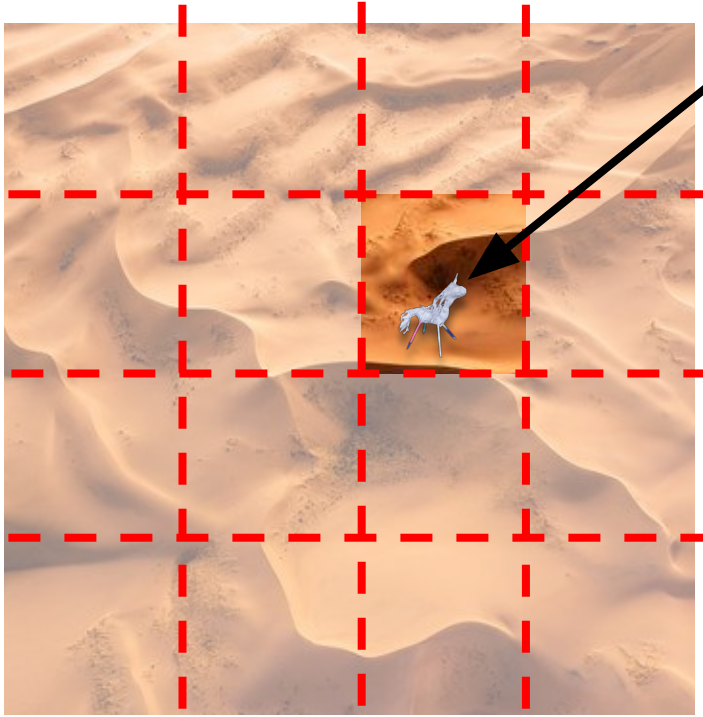
Может нарежем на куски, один кусок влезет VRAM?

Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16К x 16К



Игрок видит мир вокруг:

Может нарежем на куски, один кусок влезет VRAM?

- подвисание при переходе из куска в кусок

- непонятно как показать другие куски вдалеке

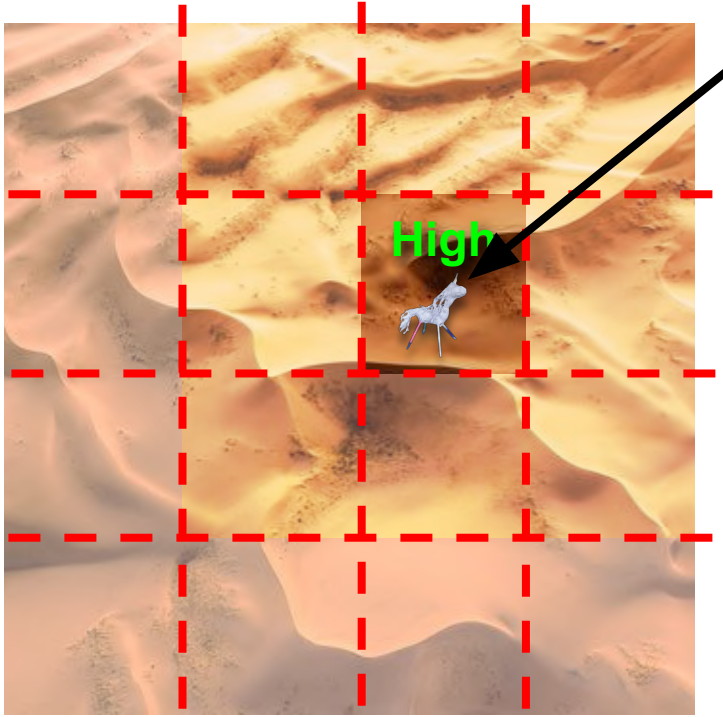
Идеи?

Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16К x 16К



Игрок видит мир вокруг:

Может нарежем на куски, один кусок влезет VRAM?

- подвисание при переходе из куска в кусок

- непонятно как показать другие куски вдалеке

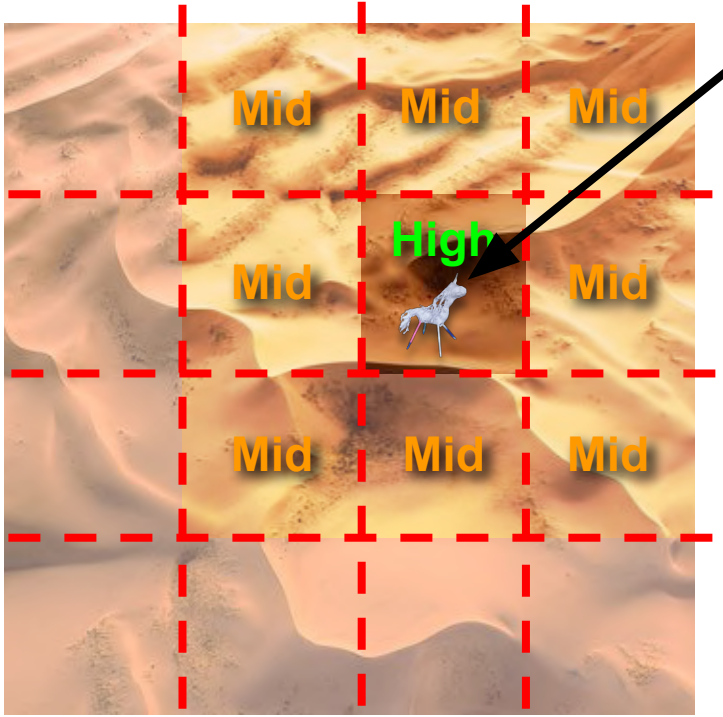
Может нарежем на куски + LODs (Levels of Details)?

Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16К x 16К



Игрок видит мир вокруг:

Может нарежем на куски, один кусок влезет VRAM?

- подвисание при переходе из куска в кусок

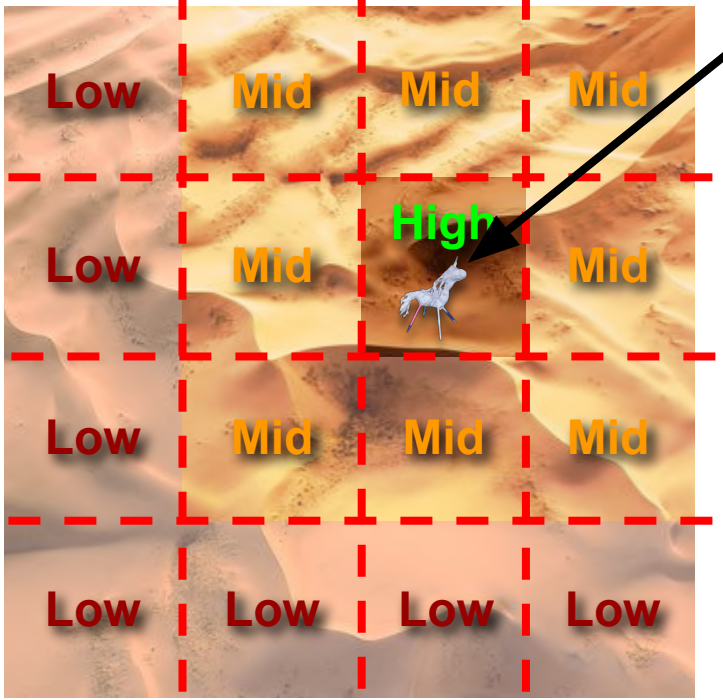
- непонятно как показать другие куски вдалеке

Может нарежем на куски + LODs (Levels of Details)?

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура
В те времена - 16К x 16К

Нельзя загрузить целиком в VRAM! Что делать?



Игрок видит мир вокруг:

Может нарежем на куски, один кусок влезет VRAM?

- **подвисание при переходе из куска в кусок**
- **непонятно как показать другие куски вдалеке**

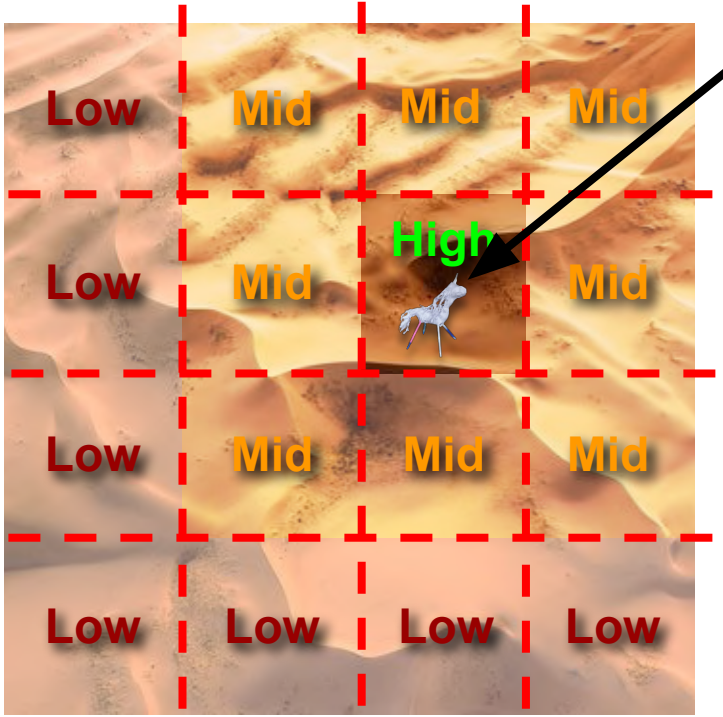
Может нарежем на куски + LODs (Levels of Details)?

Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16К x 16К



Игрок видит мир вокруг:

Может нарежем на куски, один кусок влезет VRAM?

- **подвисание при переходе из куска в кусок**
- **непонятно как показать другие куски вдалеке**

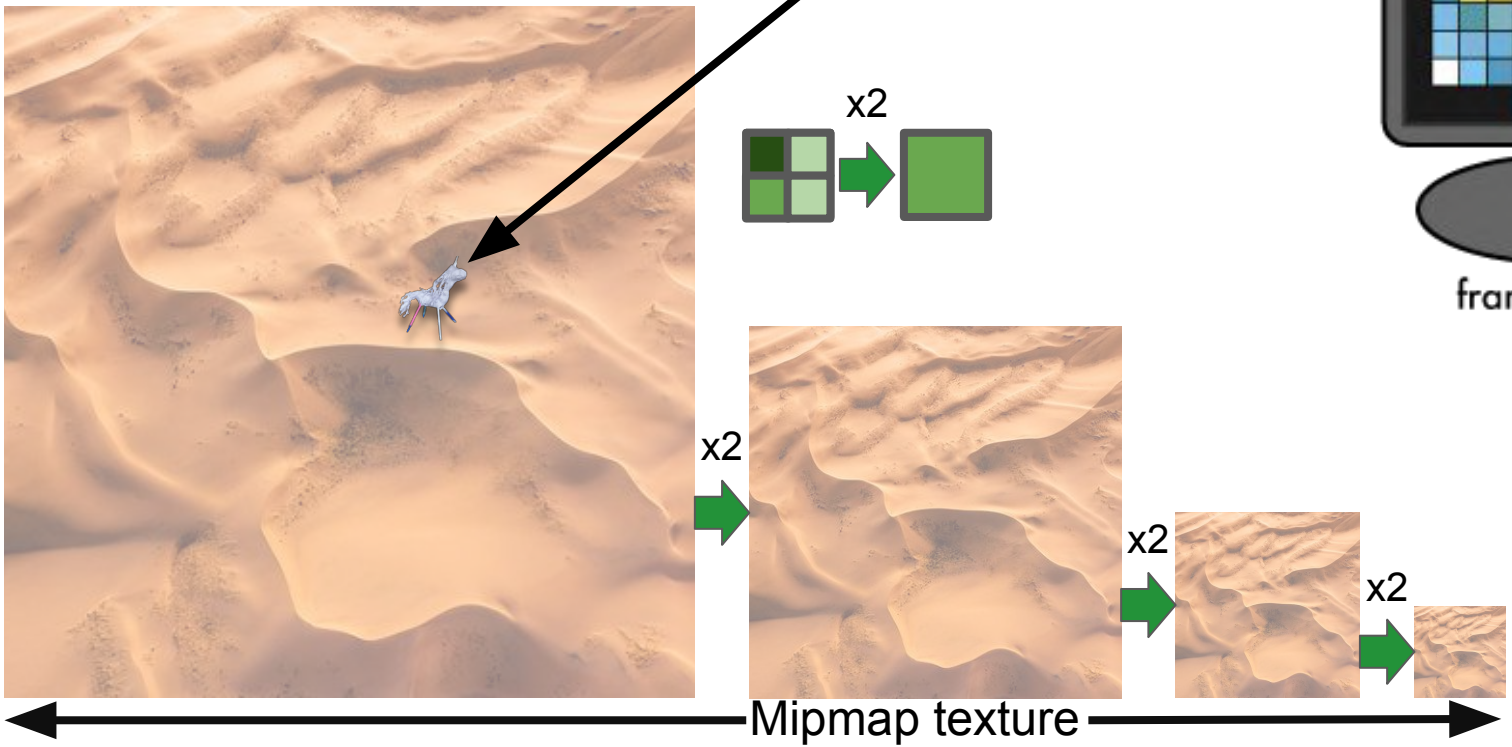
Может нарежем на куски + LODs (Levels of Details)?

- **заметные негладкости на стыках High-Mid, Mid-Low**
- **подвисание при переходе из куска в кусок**
- **нужно подобрать размеры кусков, построить LOD**

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

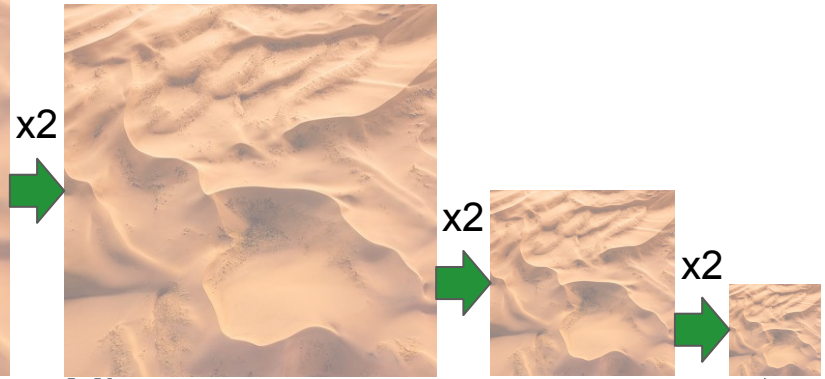
Игрок видит мир вокруг:



Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

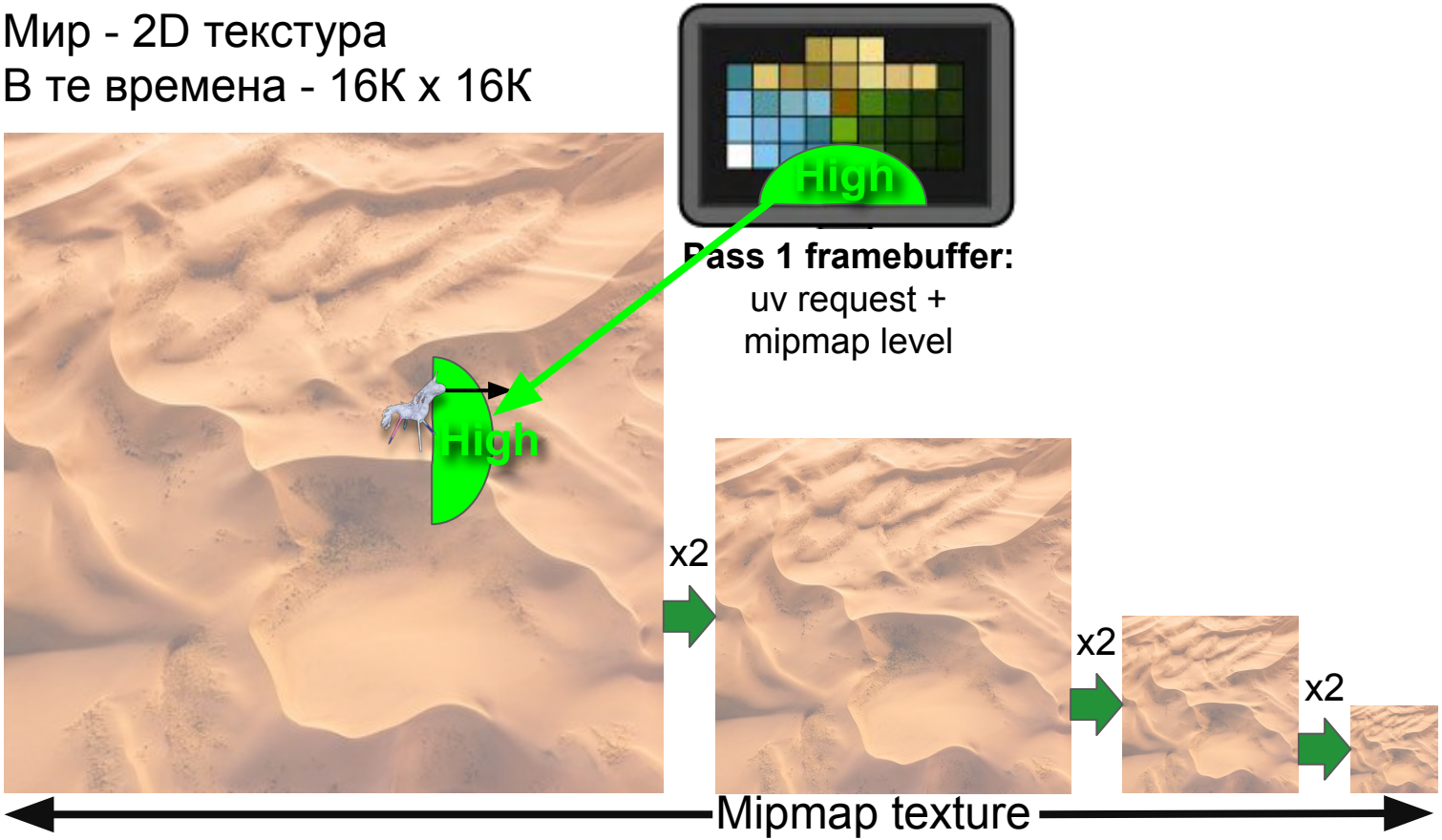
Pass 1 framebuffer:
uv request +
mipmap level



Mipmap texture

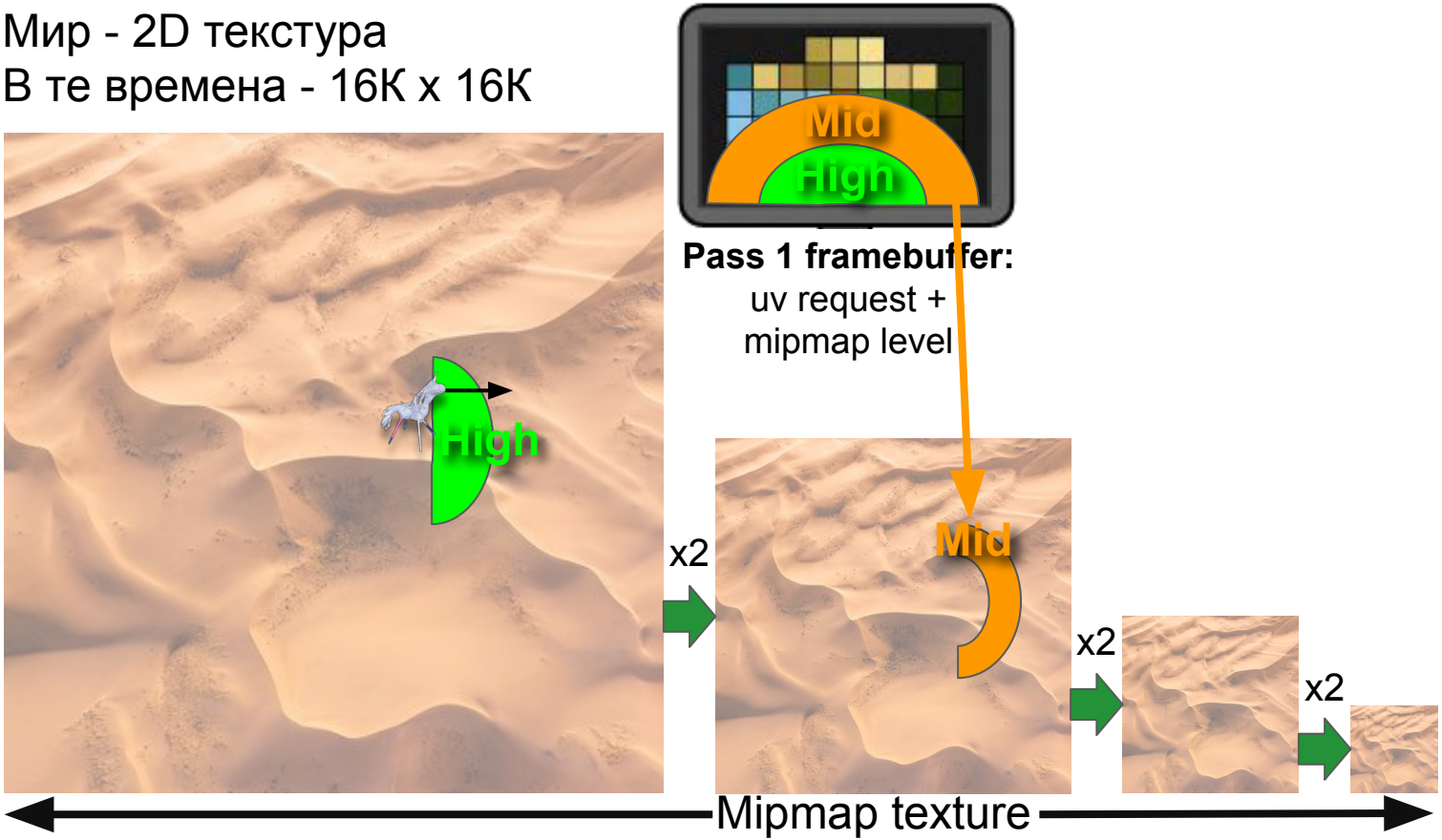
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К



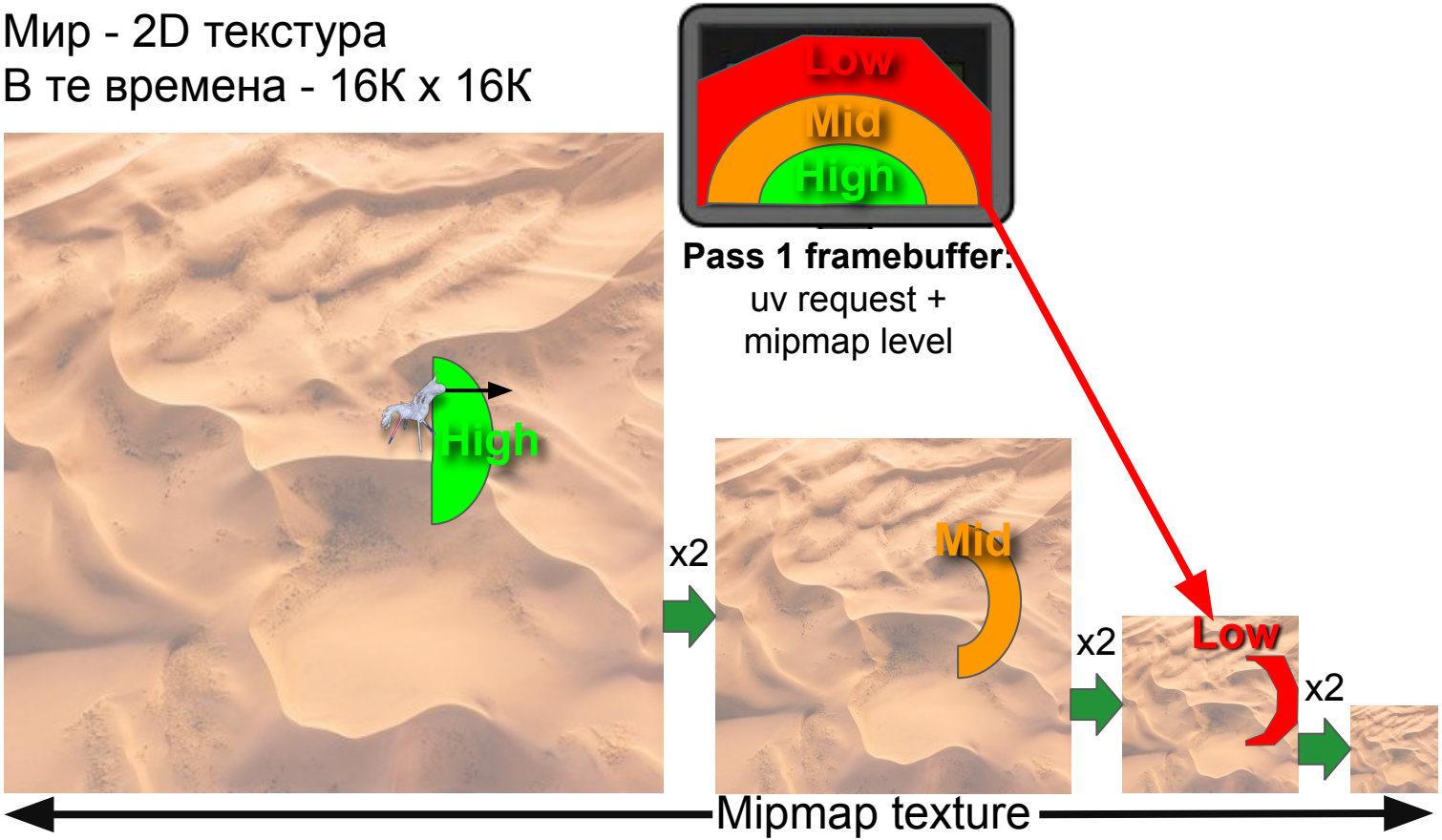
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К



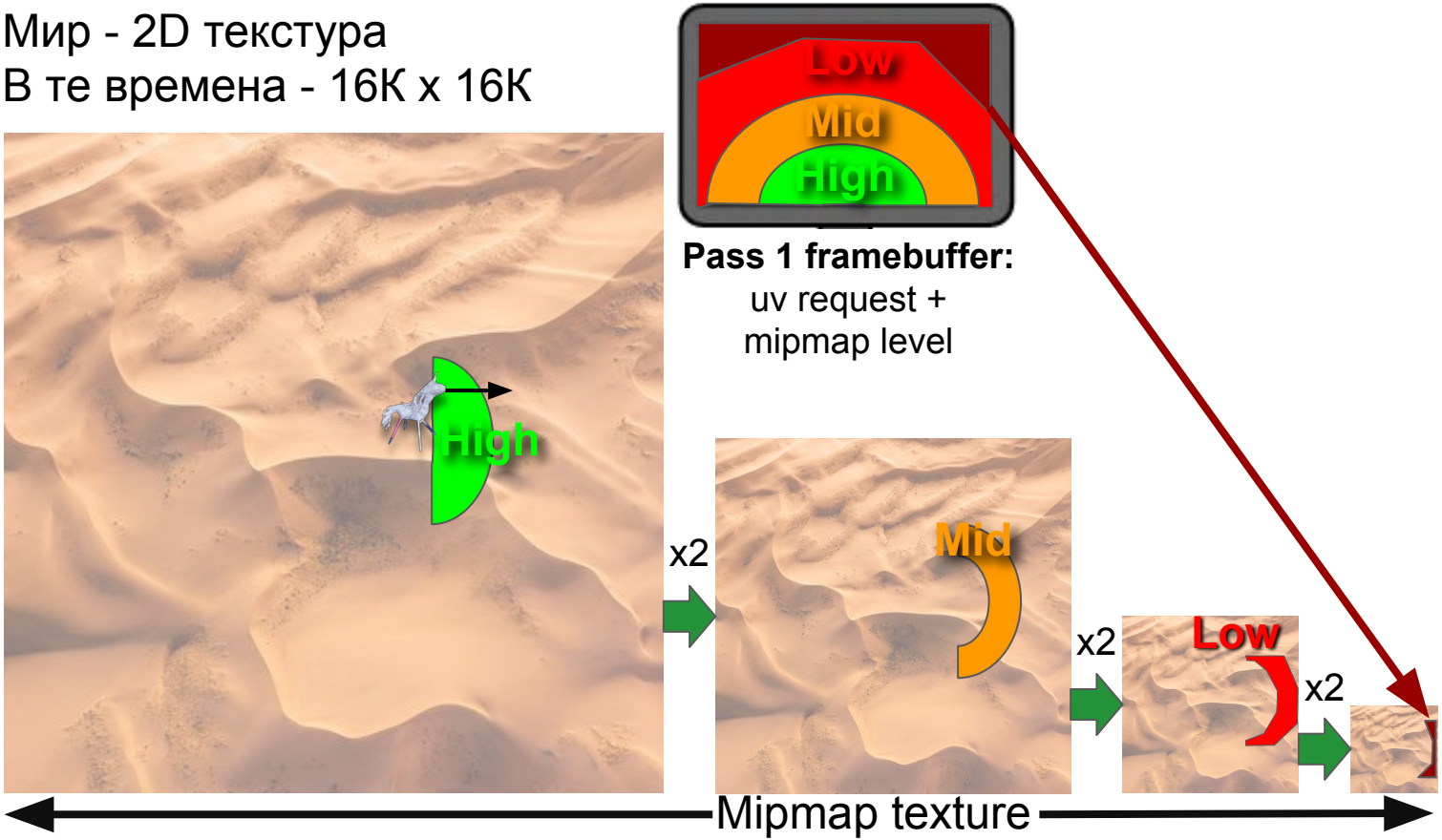
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К



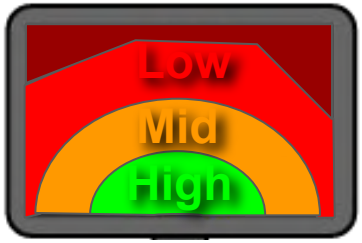
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К



Виртуальные текстуры (**virtual texture, texture streaming**)

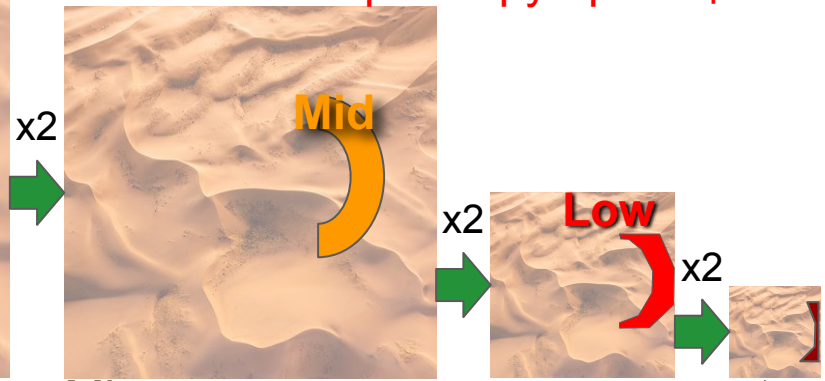
Мир - 2D текстура
В те времена - 16К x 16К



Как определить детальность?
(mipmap level)

Pass 1 framebuffer:
uv request +
mipmap level

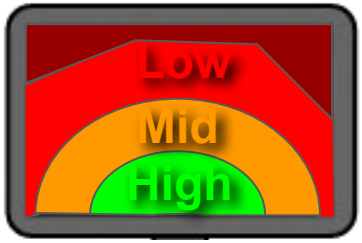
downscale ~ размеру проекции в экране



Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

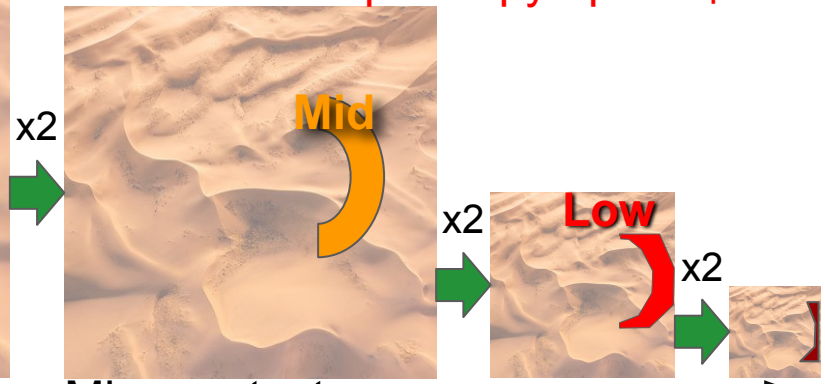


Pass 1 framebuffer:
uv request +
mipmap level

Как определить детальность?
(mipmap level)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y

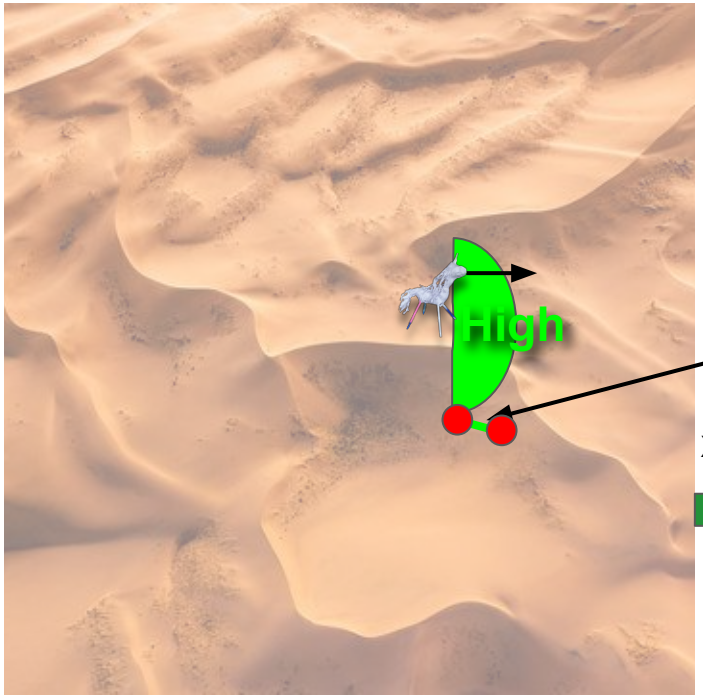
downscale ~ размеру проекции в экране

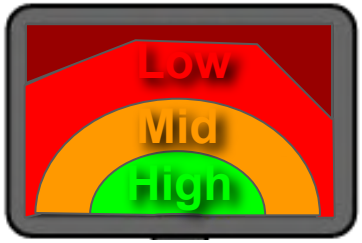


Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

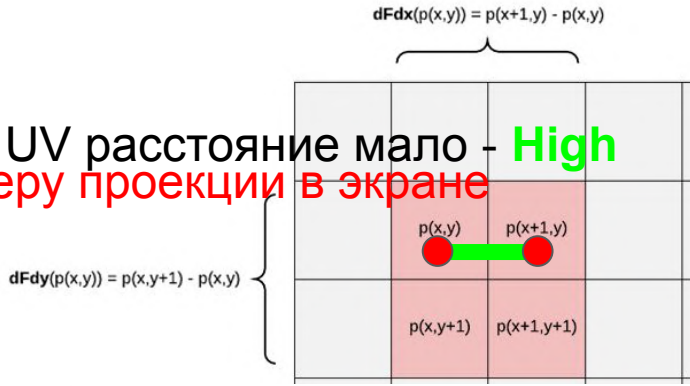




Как определить детальность?
(mipmap level)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y

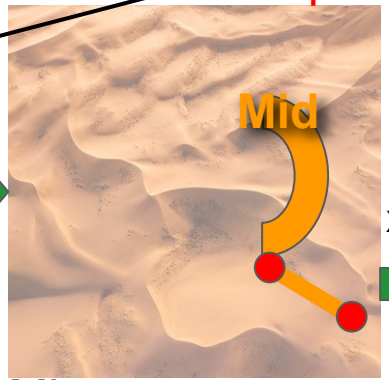
Pass 1 framebuffer:
uv request +
mipmap level



если UV расстояние мало - High

downscale ~ размеру проекции в экране

x2



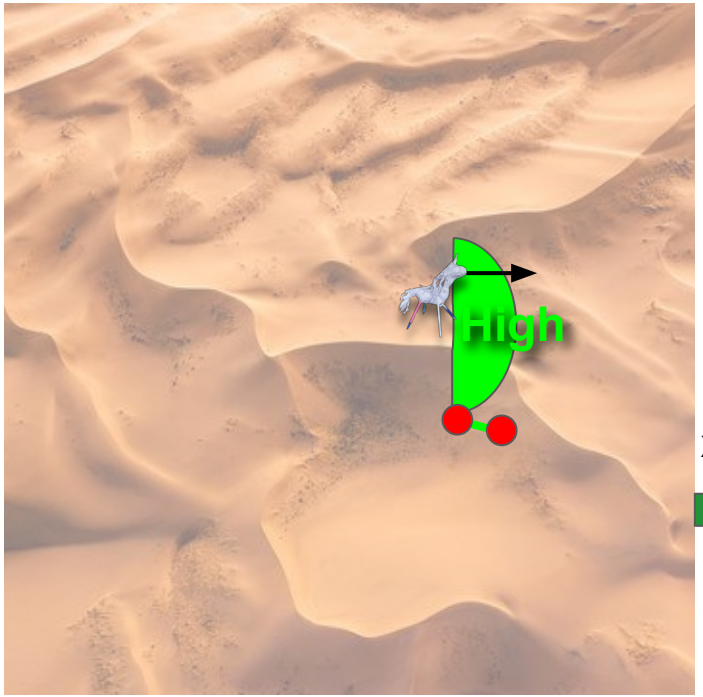
x2

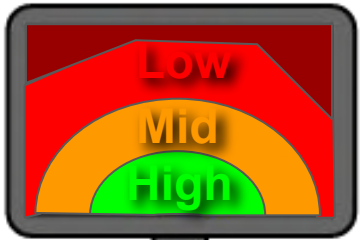
x2

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

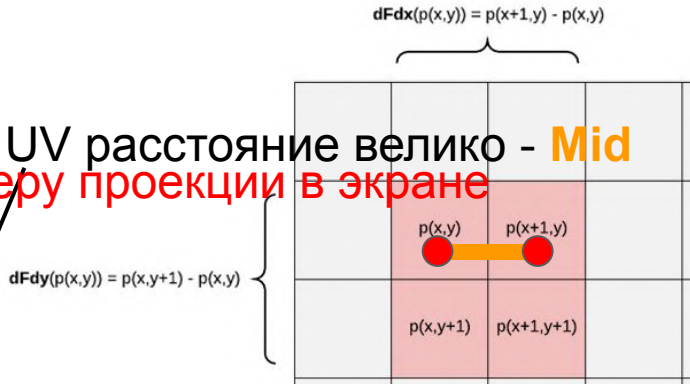




Как определить детальность?
(mipmap level)

dFdx(...), **dFdy**(...) — return the partial derivative of an argument with respect to x or y

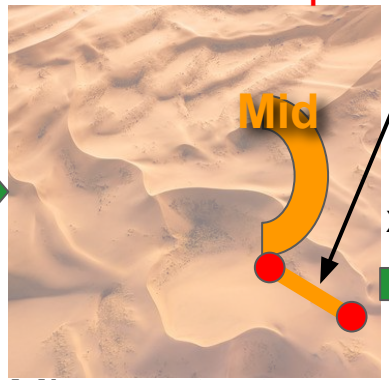
Pass 1 framebuffer:
uv request +
mipmap level



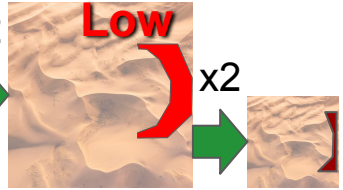
если UV расстояние велико - **Mid**

downscale ~ размеру проекции в экране

x2



x2

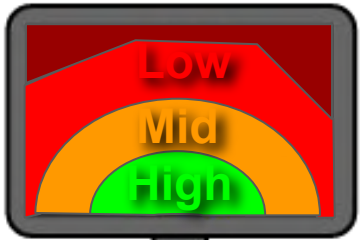


x2

Mipmap texture

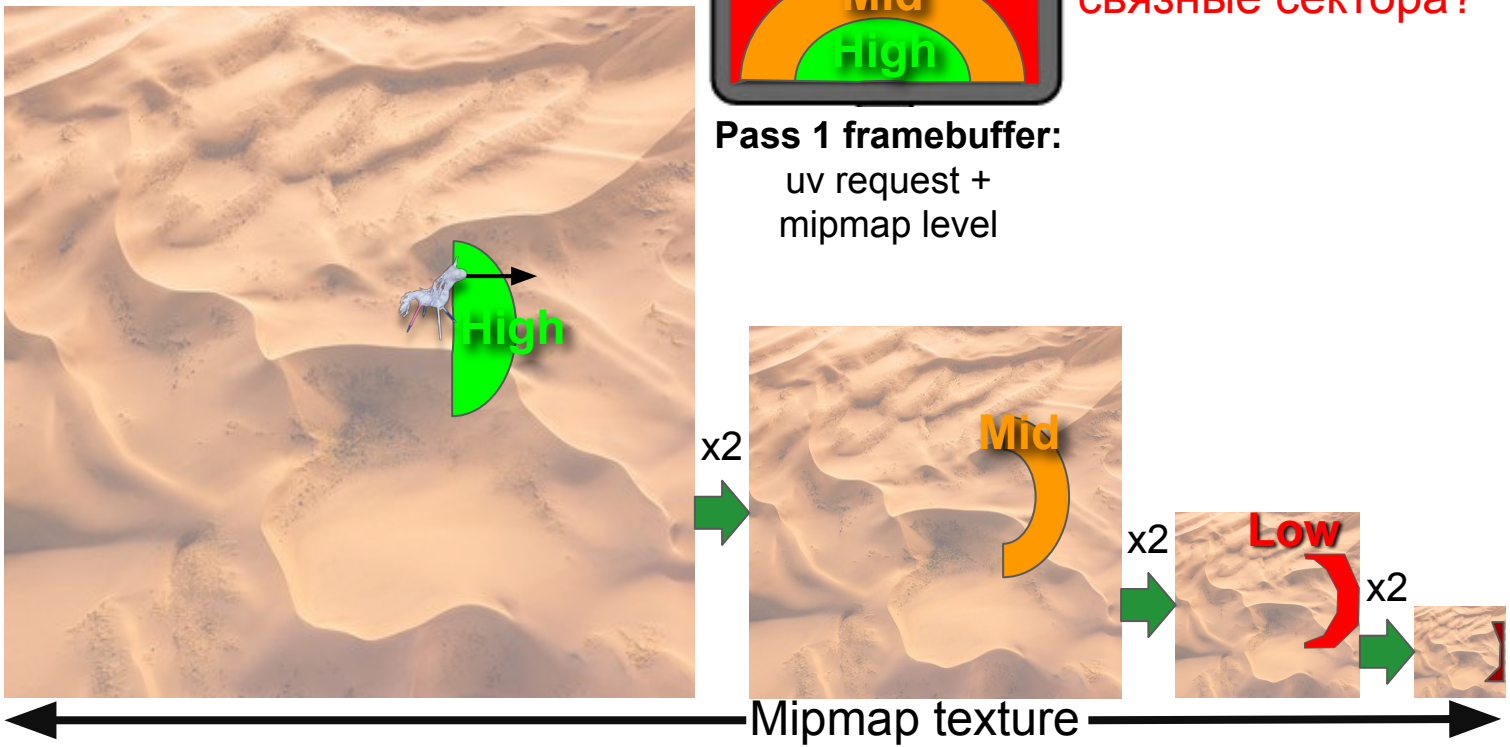
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К



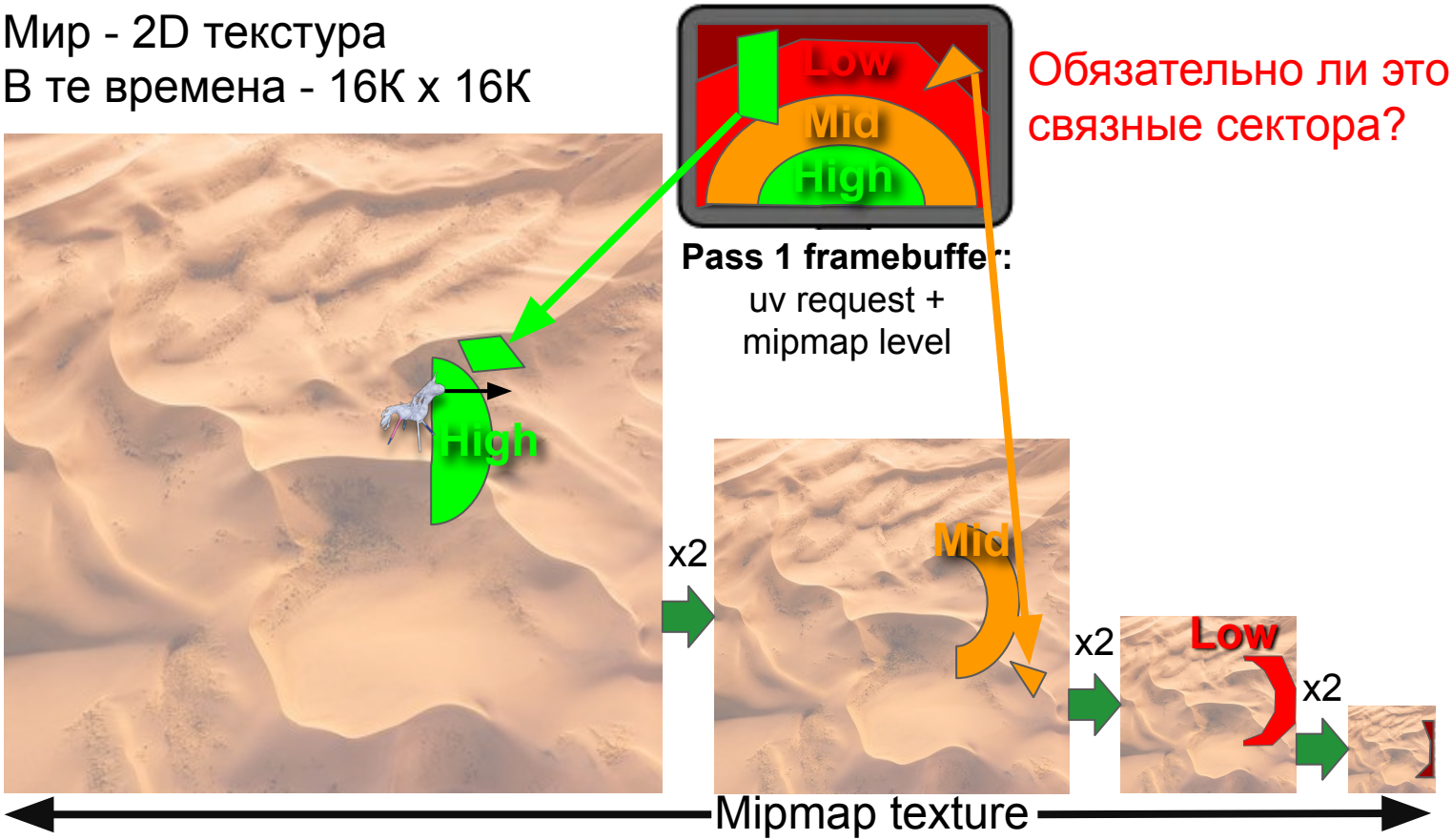
Pass 1 framebuffer:
uv request +
mipmap level

Обязательно ли это связные сектора?



Виртуальные текстуры (**virtual texture, texture streaming**)

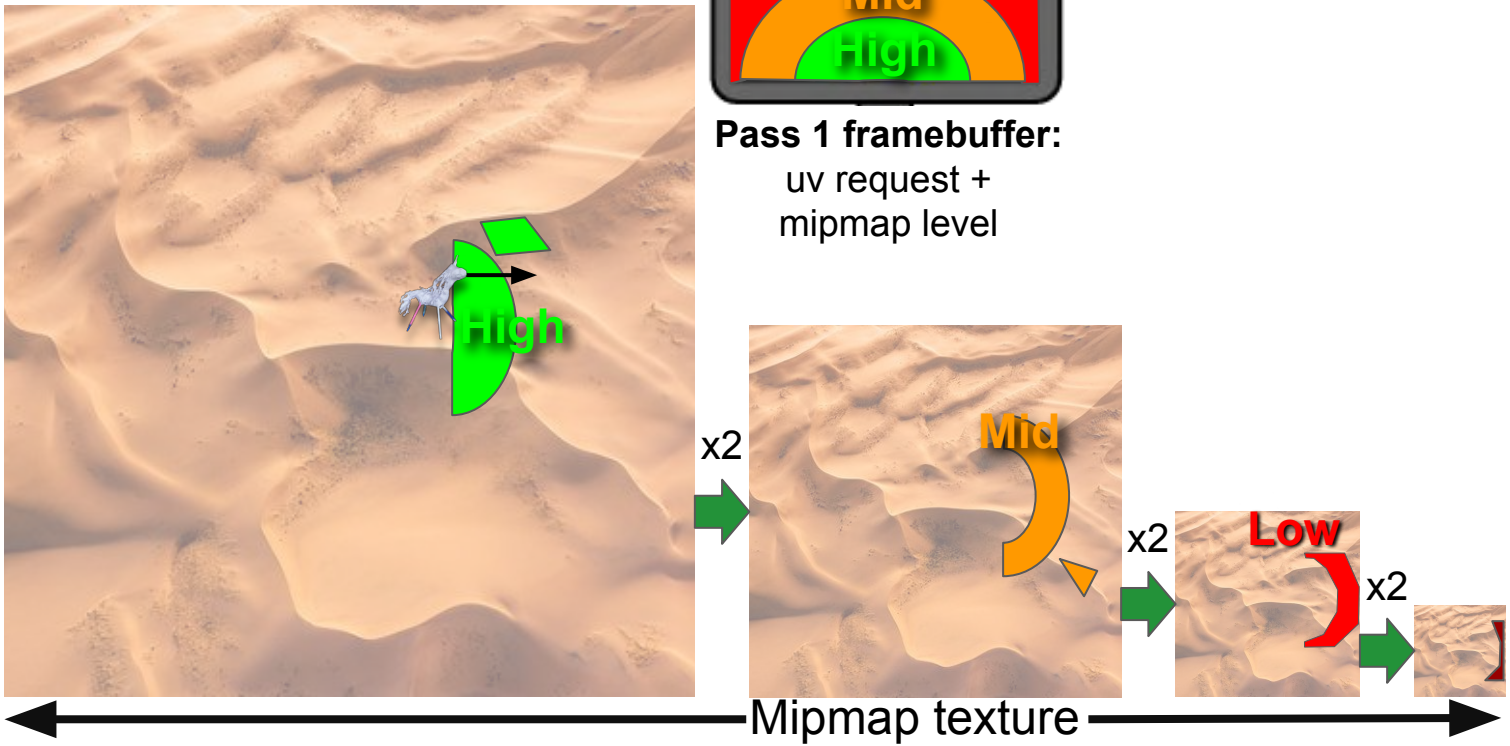
Мир - 2D текстура
В те времена - 16К x 16К



Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

Pass 1 framebuffer:
uv request +
mipmap level



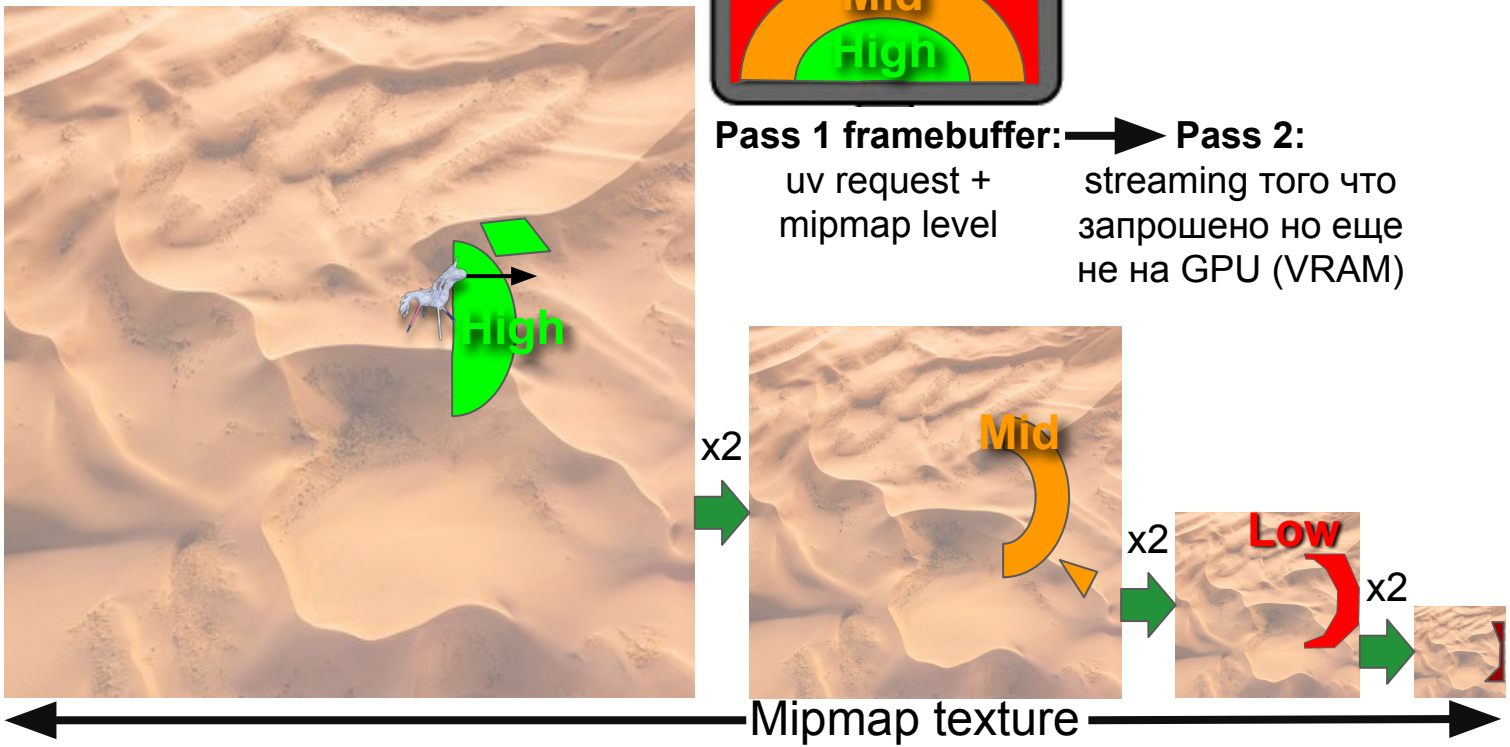
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

Pass 1 framebuffer: → **Pass 2:**

uv request + mipmap level

streaming того что запрошено но еще не на GPU (VRAM)



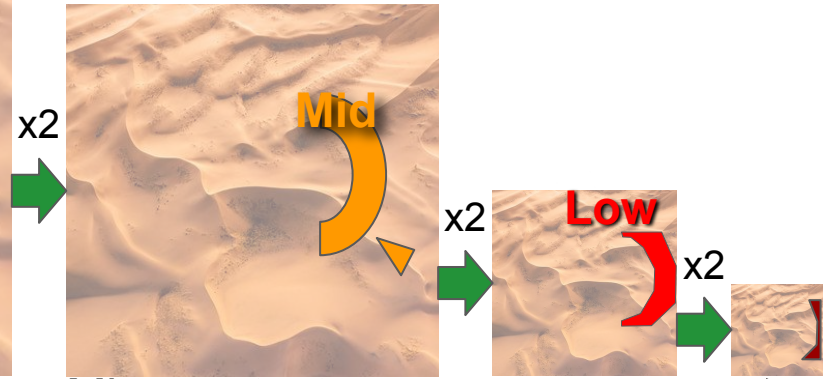
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

Pass 1 framebuffer:
uv request + mipmap level

Pass 2:
streaming того что запрошено но еще не на GPU (VRAM)



Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

Pass 1 framebuffer:
uv request + mipmap level

Pass 2:
streaming того что запрошено но еще не на GPU (VRAM)

Влезет ли?
Есть ли какая-то оценка сверху?

x2

x2

x2

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

Pass 1 framebuffer:
uv request + mipmap level

Pass 2:
streaming того что запрошено но еще не на GPU (VRAM)

Pass 3 framebuffer:
финальный **Render Pass** (все есть на GPU)

x2

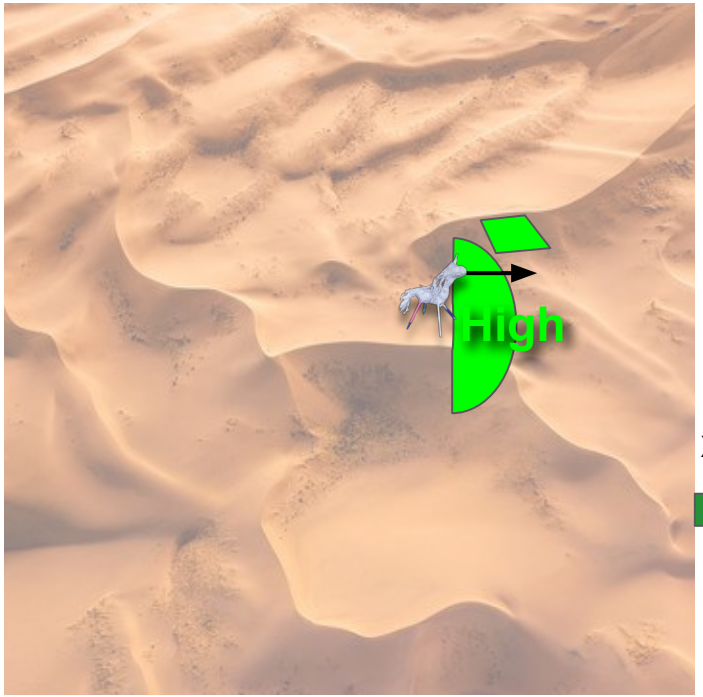
x2

x2

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К



Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

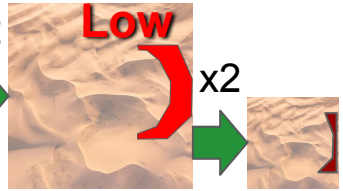
Pass 1 framebuffer:
uv request + mipmap level

Pass 2:
streaming того что запрошено но еще не на GPU (VRAM)

Pass 3 framebuffer:
финальный **Render Pass** (все есть на GPU)

x2

x2



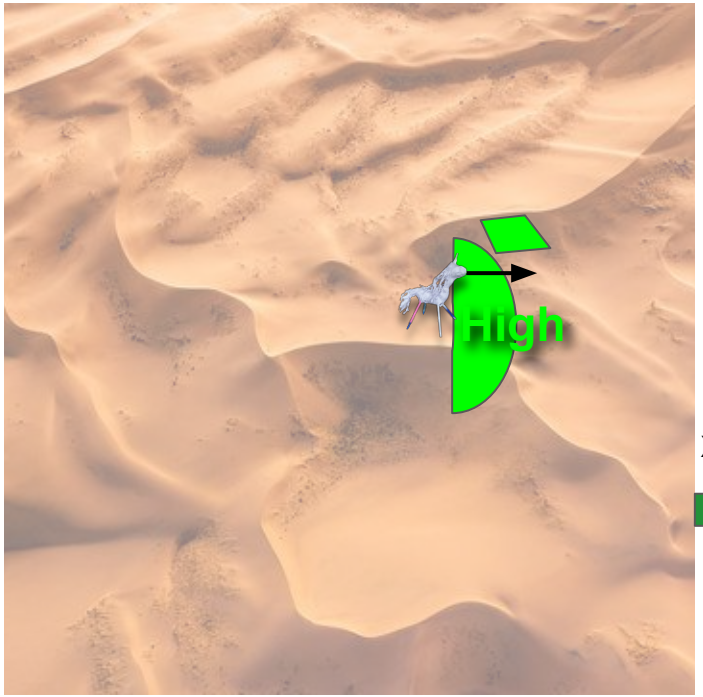
x2

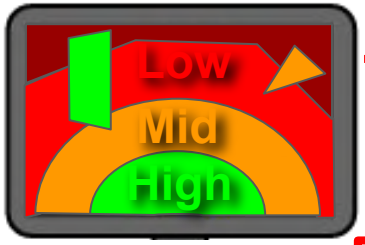
Есть ли гарантии на время отрисовки кадра/на FPS?
Не будет ли **stuttering/freeze**?

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К





Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

Pass 1 framebuffer:
uv request +
mipmap level

Pass 2:
streaming того что запрошено но еще не на GPU (VRAM)

Pass 3 framebuffer:
финальный
Render Pass
(все есть на GPU)

x2

x2

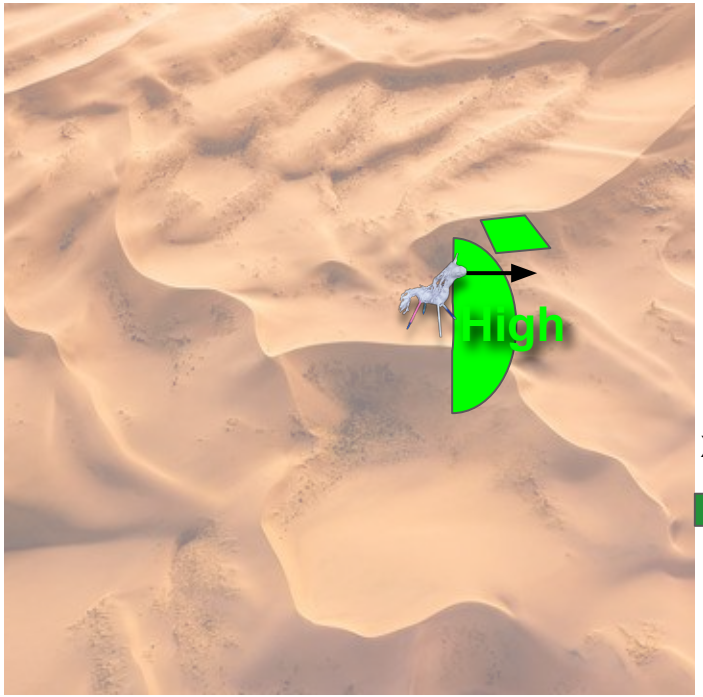
x2

Есть ли гарантии на время отрисовки кадра/на FPS?
Не будет ли **stuttering/freeze**?

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К



x2

x2

x2

Mipmap texture

Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

Pass 1 framebuffer:
uv request +
mipmap level

Pass 2:
async streaming
того что запрошено
но еще не на GPU
(VRAM)

Pass 3 framebuffer:
финальный
Render Pass
(все есть на GPU)

Есть ли гарантии на время отрисовки кадра/на FPS?
Не будет ли **stuttering/freeze**?

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

Pass 1 framebuffer:
uv request +
mipmap level

Pass 2:
async streaming
того что запрошено
но еще не на GPU
(VRAM)

Pass 3 framebuffer:
финальный
Render Pass
(все есть на GPU)

x2

x2

x2

А **не зря** ли мы начали **IO-грузить** то что не впишется в бюджет времени на кадр?

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

Pass 1 framebuffer:
uv request + mipmap level

Pass 2:
async streaming
того что запрошено но еще не на GPU (VRAM)

Pass 3 framebuffer:
финальный **Render Pass**
(все есть на GPU)

x2

Медленное движение => хорошая когерентность кадров, **пригодится позже!**

x2

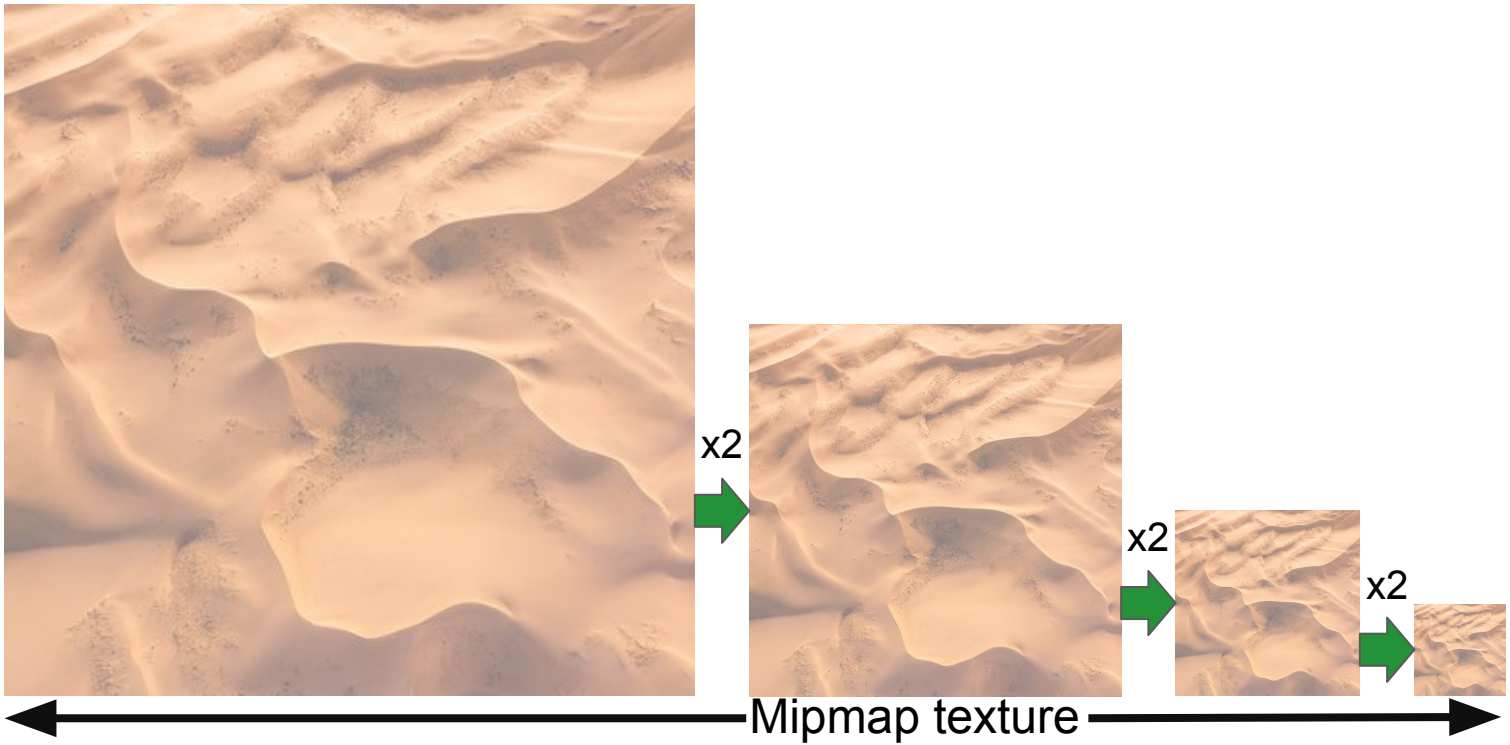
x2

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?



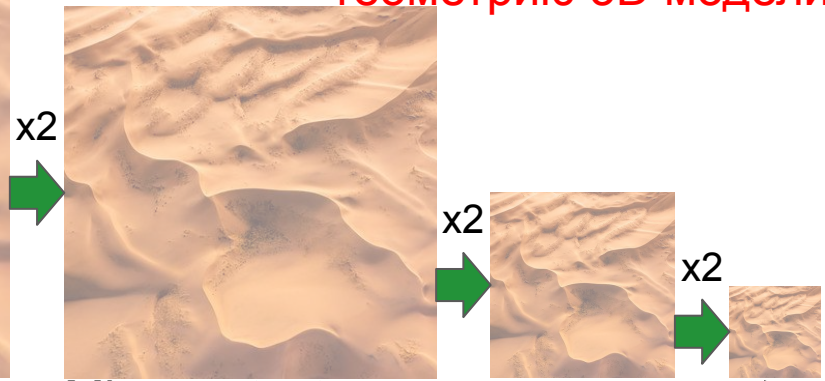
Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?

3D model UV texture

Можно ли стримить текстуру и/или геометрию 3D модели произвольной топологии?

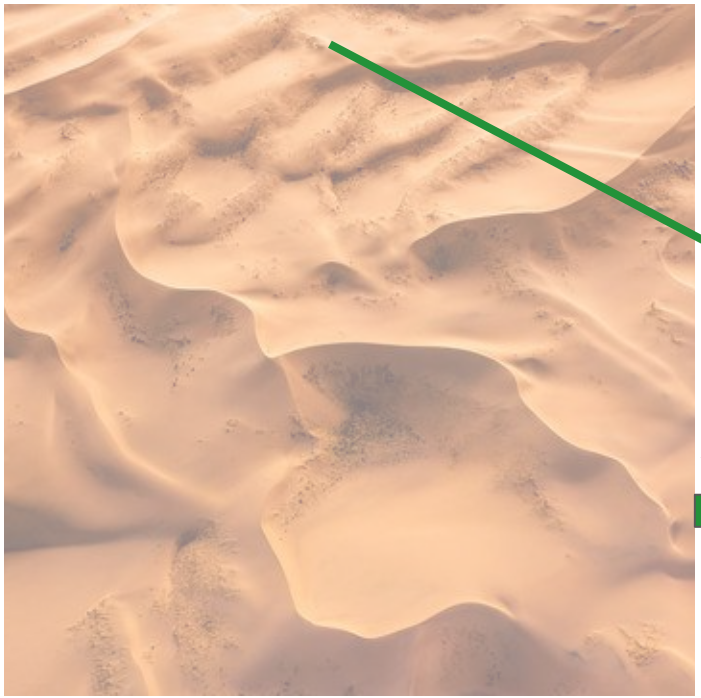


Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура

В те времена - 16К x 16К

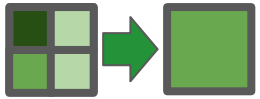


За счет чего так удачно? Универсален ли подход?

3D model UV texture

Можно ли стримить текстуру и/или геометрию 3D модели произвольной топологии?

полагались на **filterable** свойство



на **однородность** семантики

x2

x2

x2

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?

3D model UV texture

Можно ли стримить текстуру
3D модели произвольной топологии?
Чем отличается от пустыни?
полагались на **filterable** свойство

на **однородность** семантики

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?

3D model UV texture

Можно ли стримить текстуру
3D модели произвольной топологии?
Чем отличается от пустыни?
полагались на **filterable** свойство

Mipmap texture

на **однородность** семантики

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?

3D model UV texture

Можно ли стримить текстуру
3D модели произвольной топологии?
Чем отличается от пустыни?
полагались на **filterable** свойство

Mipmap texture

на **однородность** семантики

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура

В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?

3D model UV texture

Можно ли стримить текстуру

3D модели произвольной топологии?

Чем отличается от пустыни?

полагались на **filterable** свойство

Mipmap texture

на **однородность** семантики

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура

В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?

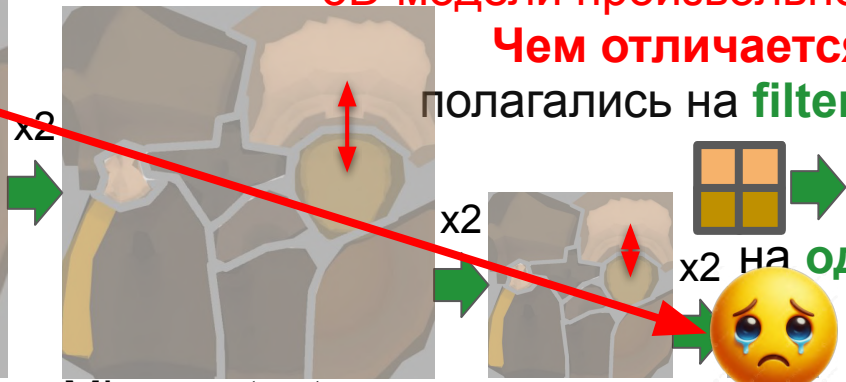
3D model UV texture

Можно ли стримить текстуру

3D модели произвольной топологии?

Чем отличается от пустыни?

полагались на **filterable** свойство



на **однородность** семантики

Mipmap texture

Виртуальные текстуры (**virtual texture, texture streaming**)

Мир - 2D текстура
В те времена - 16К x 16К

За счет чего так удачно? Универсален ли подход?

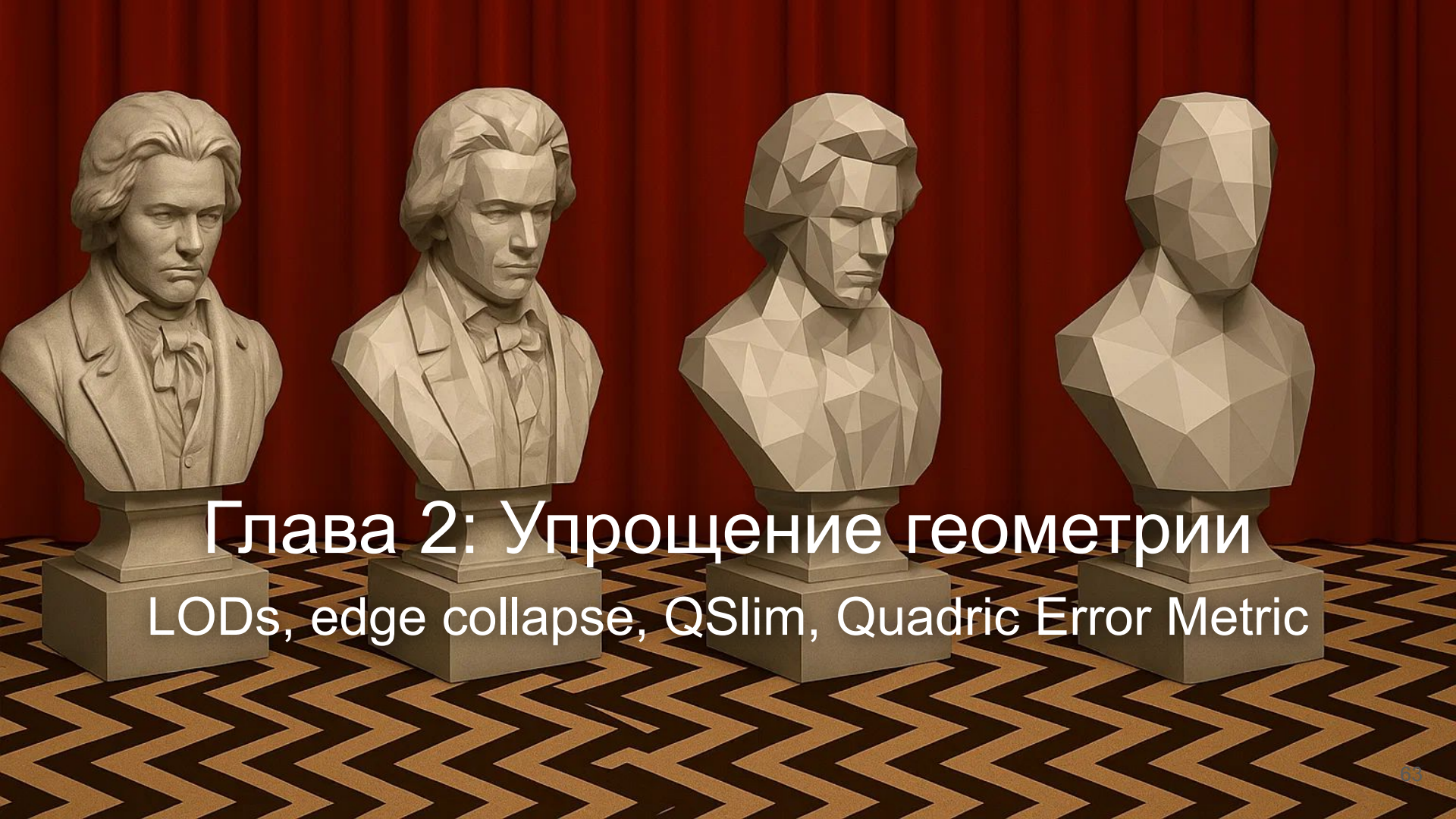
3D model UV texture

Можно ли стримить геометрию
3D модели произвольной топологии?

полагались на **filterable** свойство

на **однородность** семантики

Mipmap texture

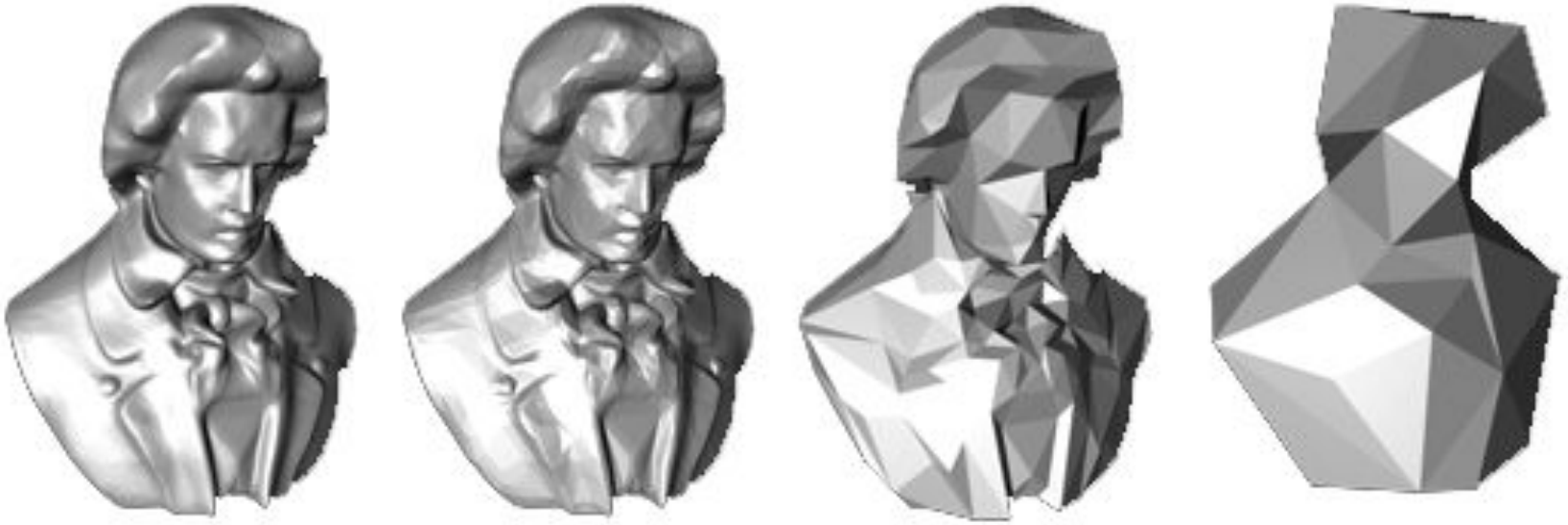


Глава 2: Упрощение геометрии

LODs, edge collapse, QSlim, Quadric Error Metric

Упрощение геометрии

Levels **o**f **D**etails (**LOD**s):

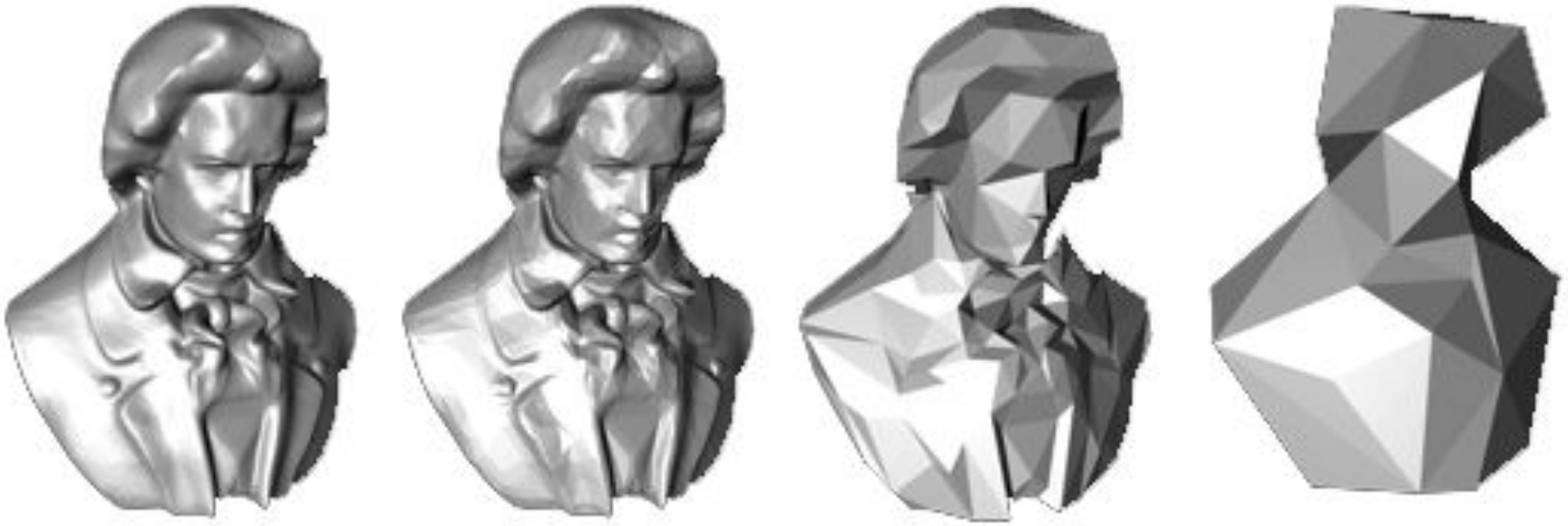


Polygons: 60.000 6.000 600 60

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

Levels **o**f **D**etails (**LOD**s):



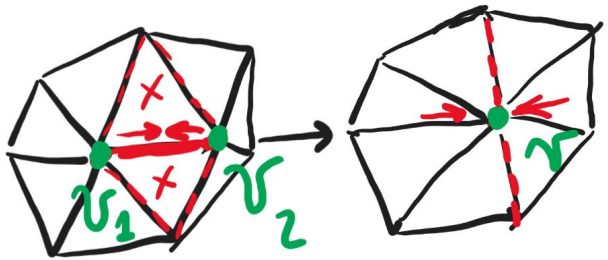
Polygons: 60.000 6.000 600 60

close **distance to camera** **far**

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

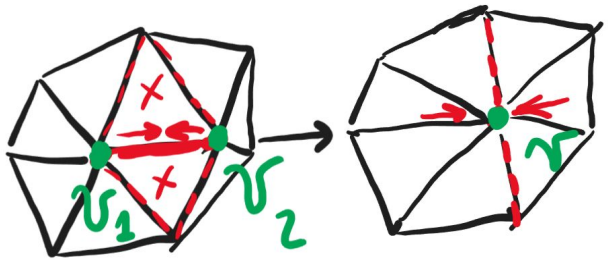
QSlim decimation: edge collapse (схлопывание ребер)



Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



Quadric Error Metric (**QEM**)

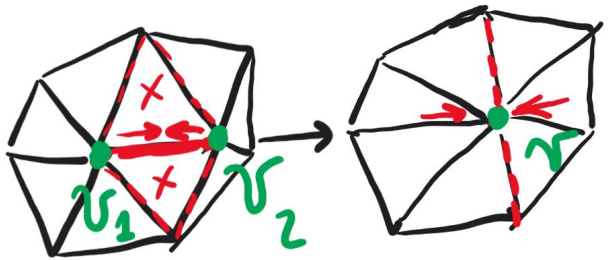
На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)

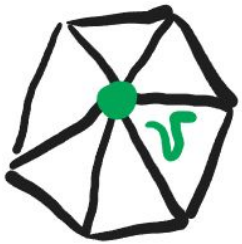


Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$



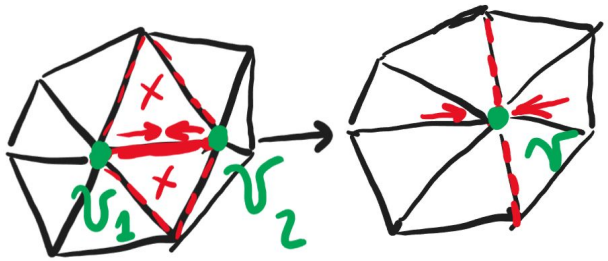
v = **пересечение плоскостей**

$error = \sum plane\ dist^2$

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



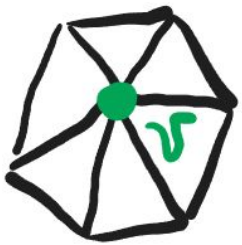
$(v_1, v_2) \xrightarrow{\text{edge collapse}} v = ???$

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$



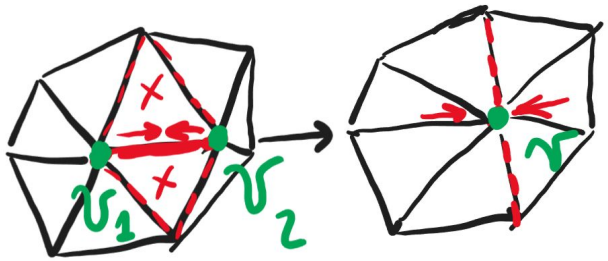
v = **пересечение плоскостей**

$error = \sum plane\ dist^2$

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



$(v_1, v_2) \xrightarrow{\text{edge collapse}} v = \underset{v}{\operatorname{argmin}}\, v^T \bar{Q} v$

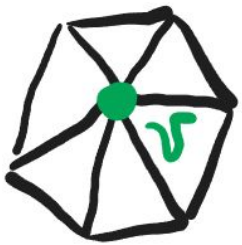
$\bar{Q} = ???$

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$



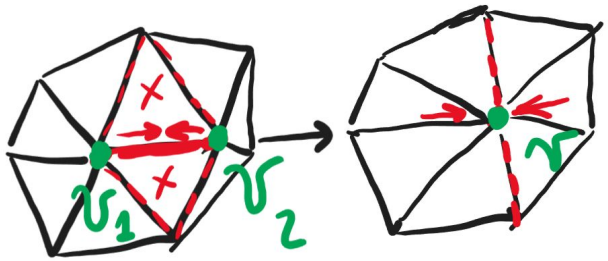
v = **пересечение плоскостей**

$error = \sum plane\ dist^2$

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



$(v_1, v_2) \xrightarrow{\text{edge collapse}} v = \underset{v}{\operatorname{argmin}}\, v^T \bar{Q} v$

$Q_1 \quad Q_2$

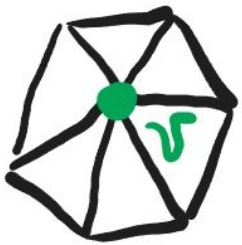
$\bar{Q} = Q_1 + Q_2$

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$



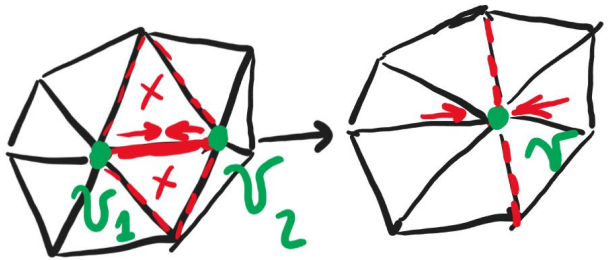
v = **пересечение плоскостей**

$error = \sum plane\ dist^2$

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



$(v_1, v_2) \xrightarrow{\text{edge collapse}} v = \underset{v}{\operatorname{argmin}}\, v^T \bar{Q} v$

$Q_1 \quad Q_2$

$\bar{Q} = Q_1 + Q_2$

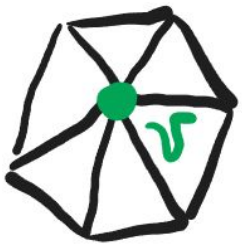
Можем ли мы запретить топологические изменения?

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$



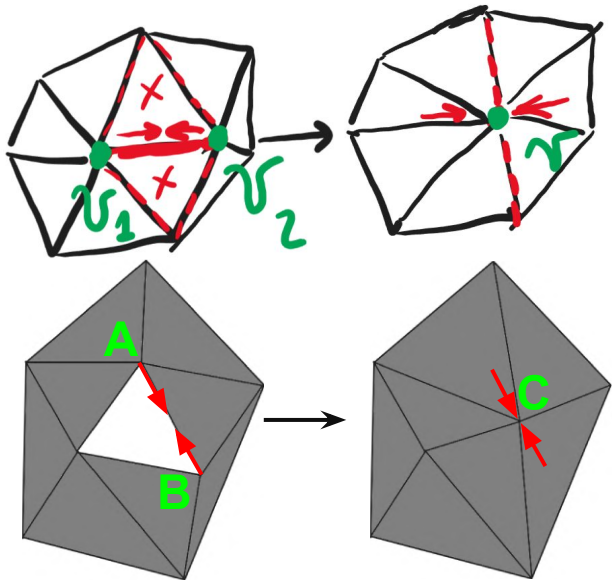
v = **пересечение плоскостей**

$error = \sum plane\ dist^2$

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



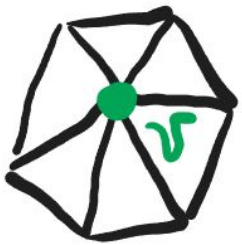
Можем ли мы запретить топологические изменения?

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$



v = **пересечение плоскостей**

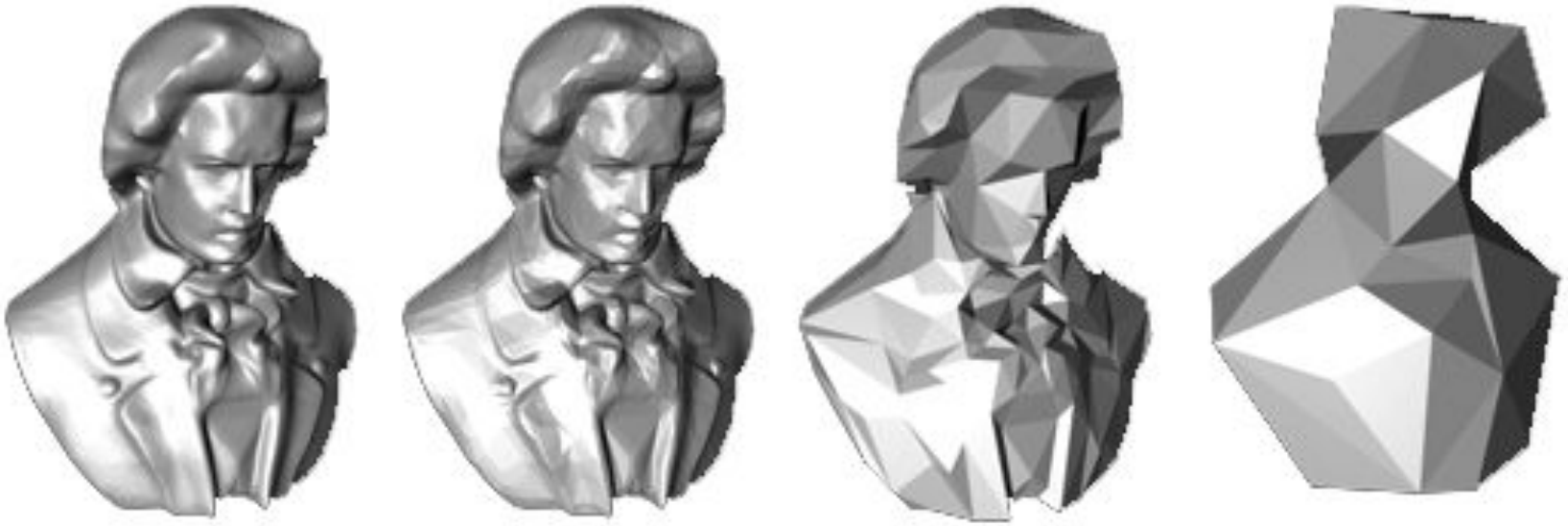
$error = \sum plane\ dist^2$

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

Levels **o**f **D**etails (**LOD**s):

Решают ли **LOD**-ы полностью задачу “большой FPS без потери качества”?



Polygons: 60.000 6.000 600 60

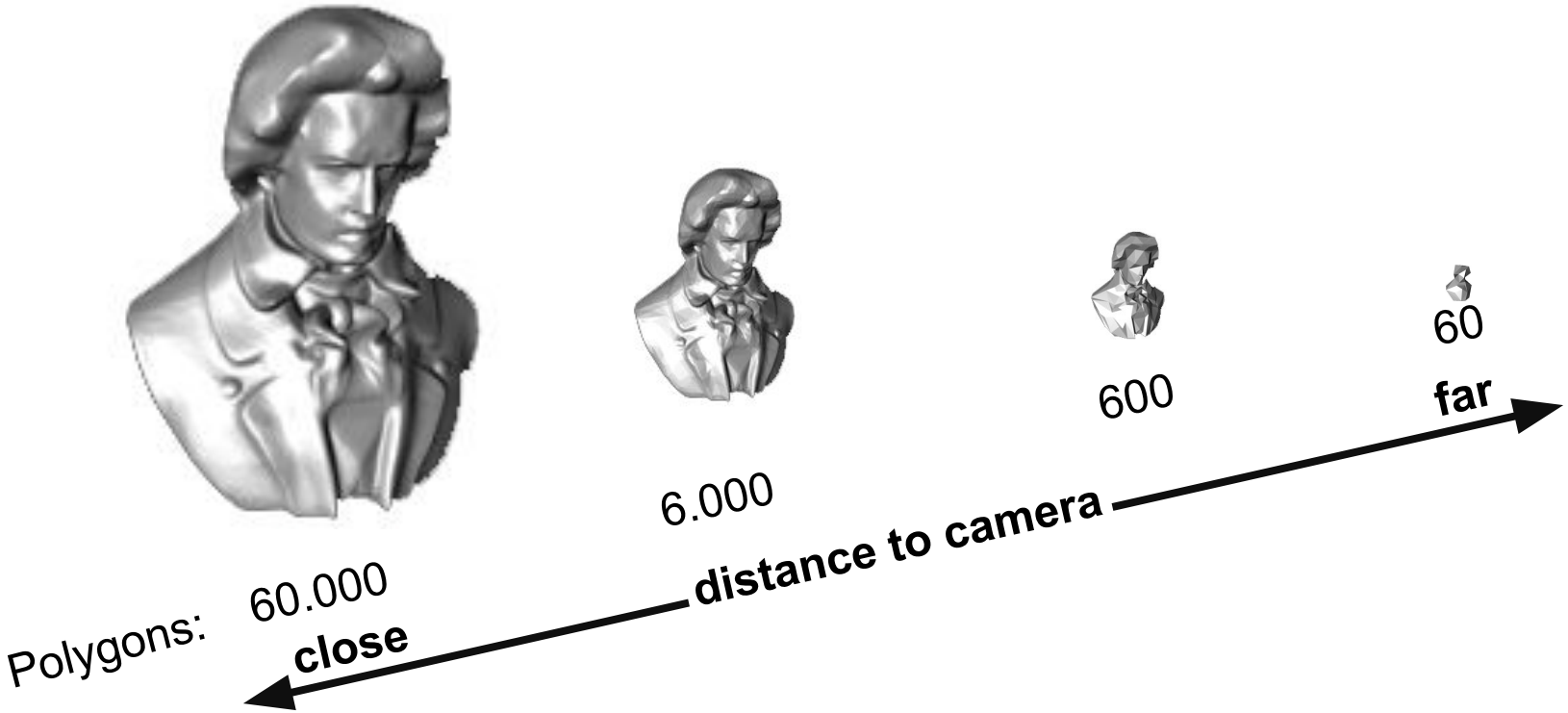
close **distance to camera** **far**

Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

Levels **o**f **D**etails (**LOD**s):

Решают ли **LOD**-ы полностью задачу “большой FPS без потери качества”?



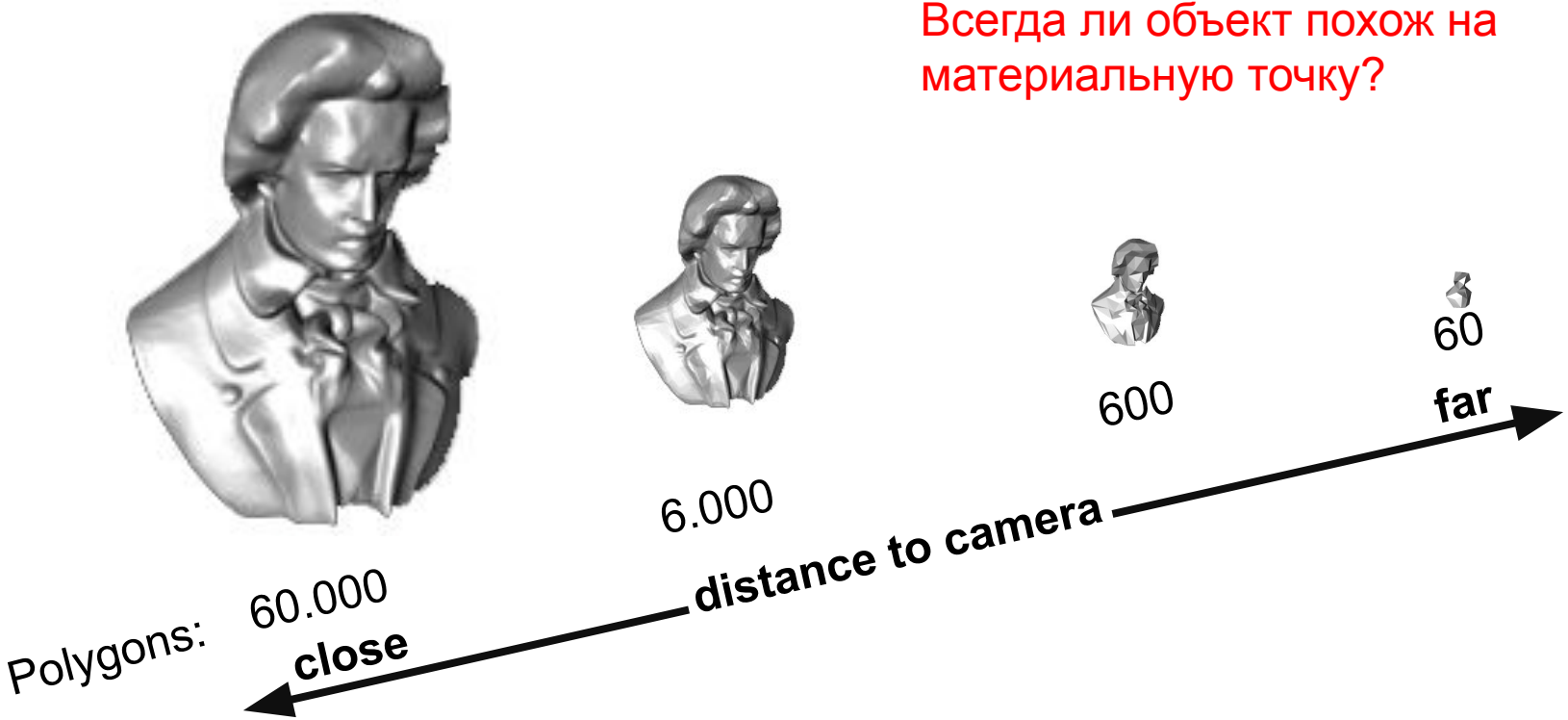
Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

Упрощение геометрии

Levels of Details (LODs):

Решают ли **LOD**-ы полностью задачу “большой FPS без потери качества”?

Всегда ли объект похож на материальную точку?



Simplifying surfaces with color and texture using quadric error metrics, **Garland et. al., 1998**

High poly

Можем ли мы выбрать **LOD** (Level of Details) для всего замка на основании расстояния?

Low poly

Можем ли мы выбрать **LOD** (Level of Details) для всего замка на основании расстояния?

Можем ли мы выбрать **LOD** (Level of Details) для частей замка **независимо**?


High poly
Low poly
Можем ли мы выбрать **LOD** (Level of Details) для частей замка **независимо**?
трещины


High poly
Low poly
Можем ли мы выбрать **LOD** (Level of Details) для частей замка **независимо**?
трещины
наложения
+Z fighting


High poly
Low poly
Можем ли мы выбрать **LOD** (Level of Details) для частей замка **независимо**?
трещины
наложения
+Z fighting

High poly

Low poly

Задача алгоритма (Nanite) - **МЕЧТА**:
- визуальная неразличимость с оригиналом

High poly

Low poly

Задача алгоритма (Nanite) - **МЕЧТА**:
- визуальная неразличимость с оригиналом
- ограниченный бюджет вычислений на кадр

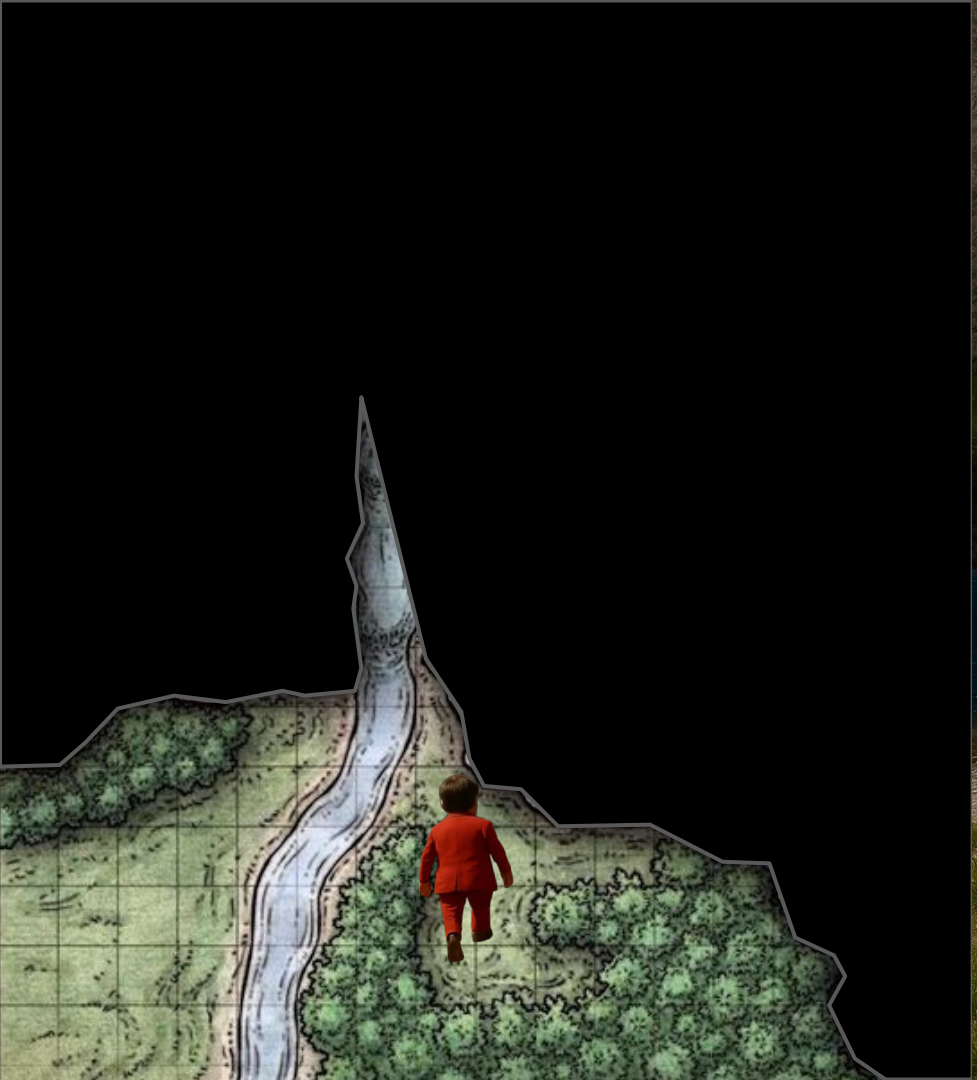
High poly

Low poly

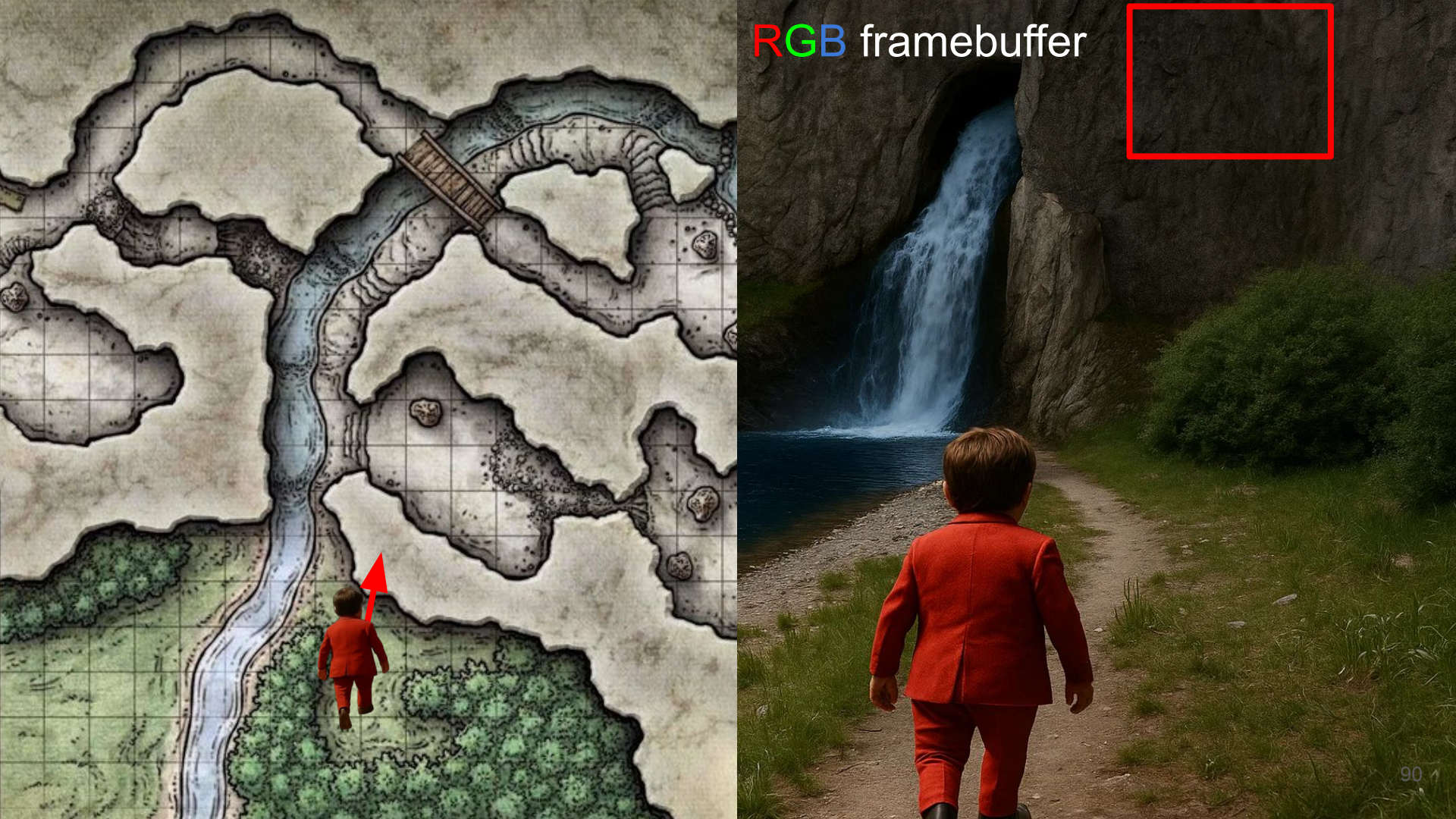
Задача алгоритма (Nanite) - **МЕЧТА**:
- визуальная неразличимость с оригиналом
- ограниченный бюджет вычислений на кадр
- автоматически и без компромиссов


Глава
NANITE

RGB framebuffer



RGB framebuffer


RGB framebuffer

RGB framebuffer

RGB framebuffer

RGB

VS

Depth framebuffer
Z framebuffer

RGB

VS

RGB

f32 depth=A

Depth framebuffer
Z framebuffer

VS

f32 depth=A

f32 depth=B

Depth framebuffer
Z framebuffer

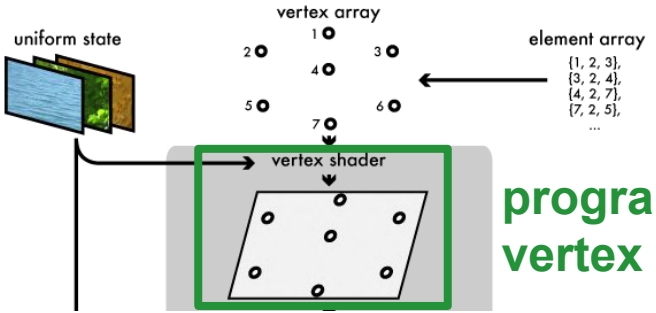
RGB

VS

RGB

f32 depth=A

f32 depth=B



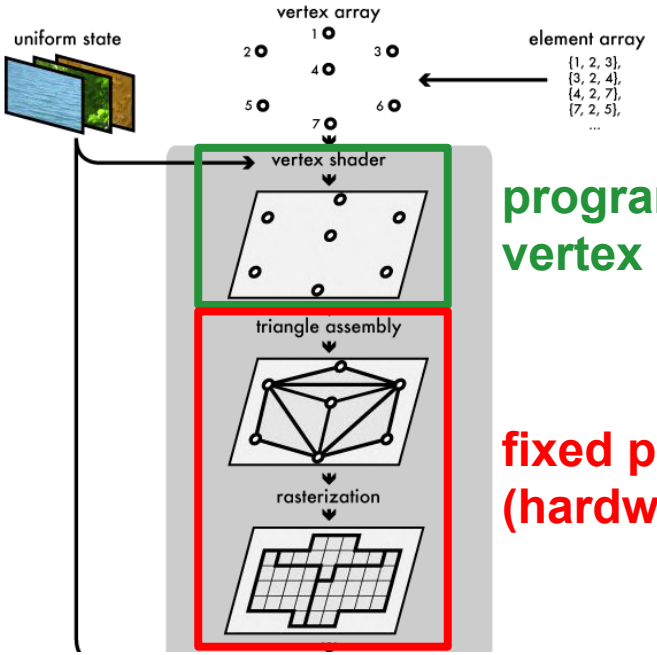
programmable vertex shader

Depth framebuffer
Z framebuffer

f32 depth=A

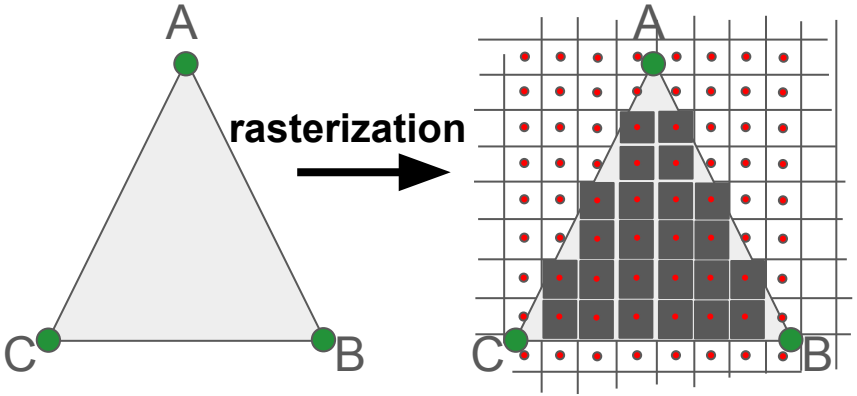
VS

f32 depth=B



programmable vertex shader

fixed pipeline (hardware)

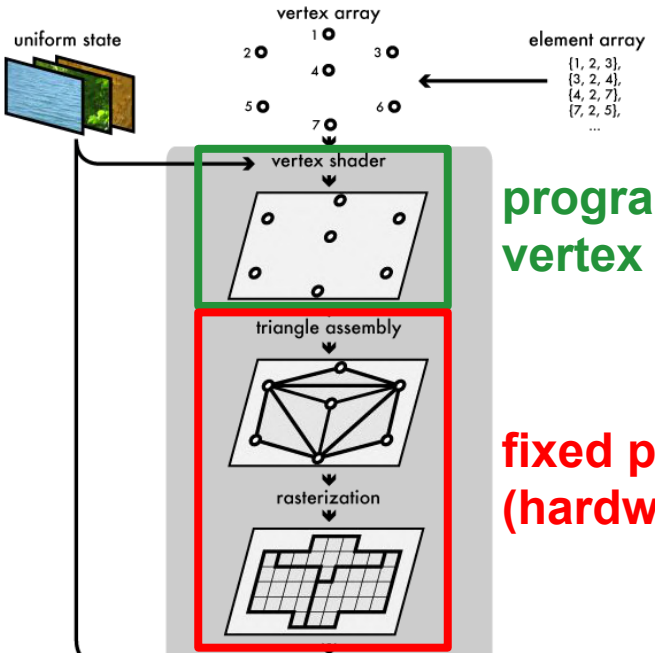


Depth framebuffer
Z framebuffer

VS

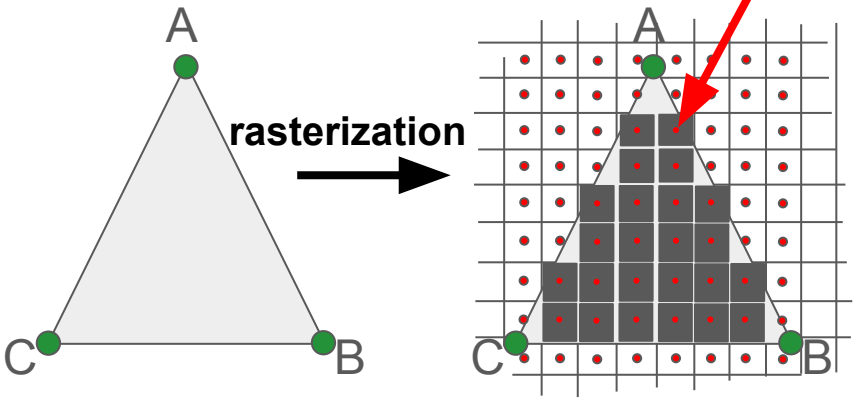
f32 depth=A

f32 depth=B



programmable vertex shader

fixed pipeline (hardware)

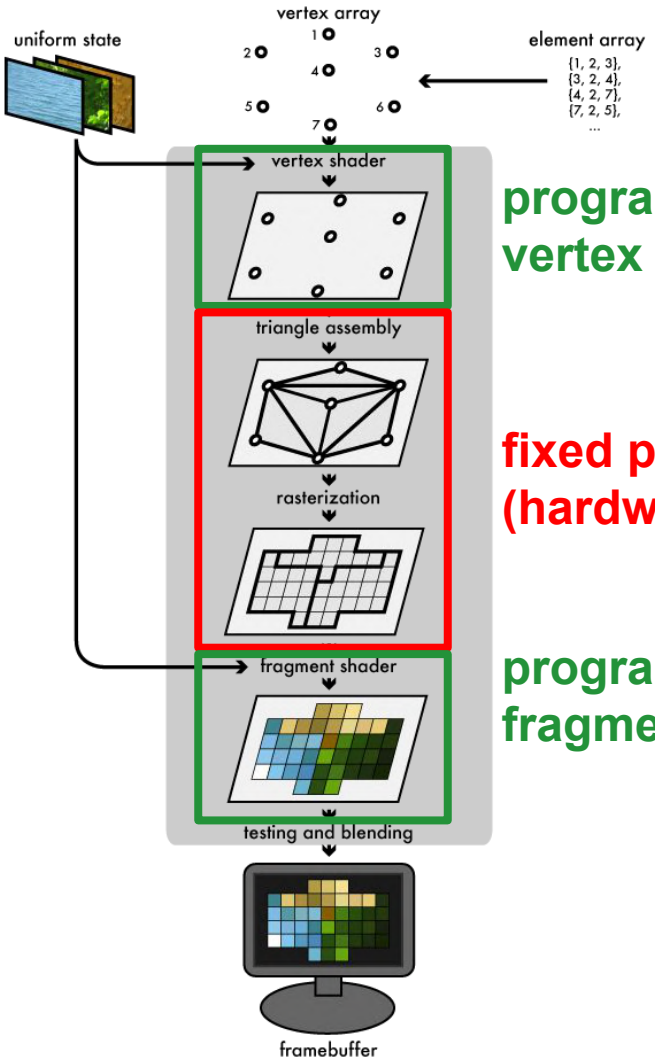


Depth framebuffer
Z framebuffer

VS

f32 depth=A

f32 depth=B



programmable vertex shader

fixed pipeline (hardware)

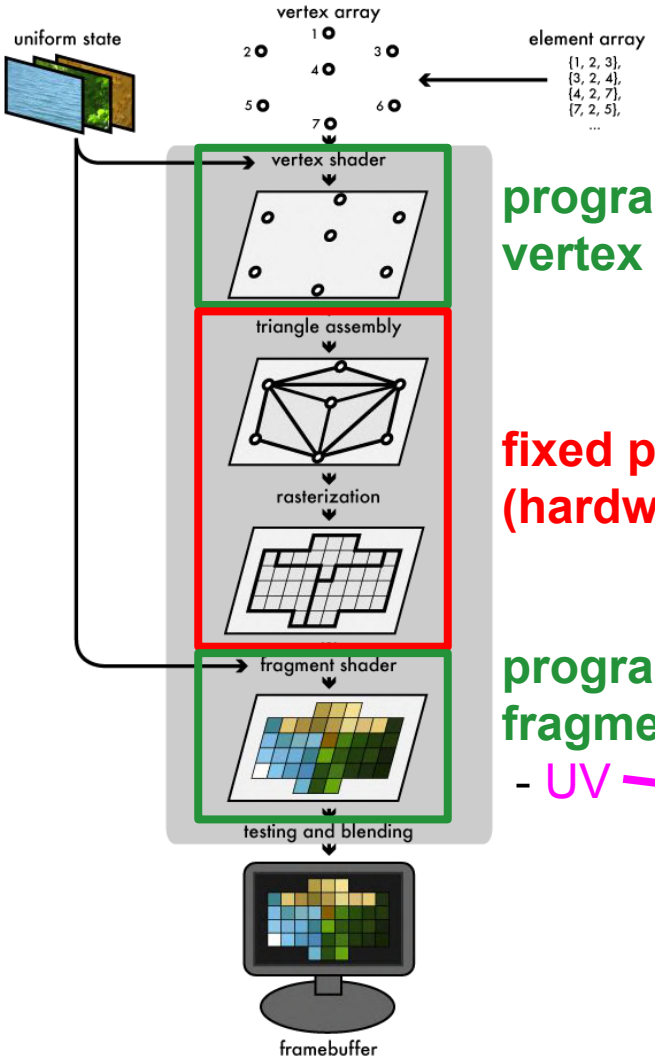
programmable fragment shader:

Depth framebuffer
Z framebuffer

VS

f32 depth=A

f32 depth=B



programmable vertex shader

fixed pipeline (hardware)

programmable fragment shader:

- UV

Depth framebuffer
Z framebuffer

RGB

VS

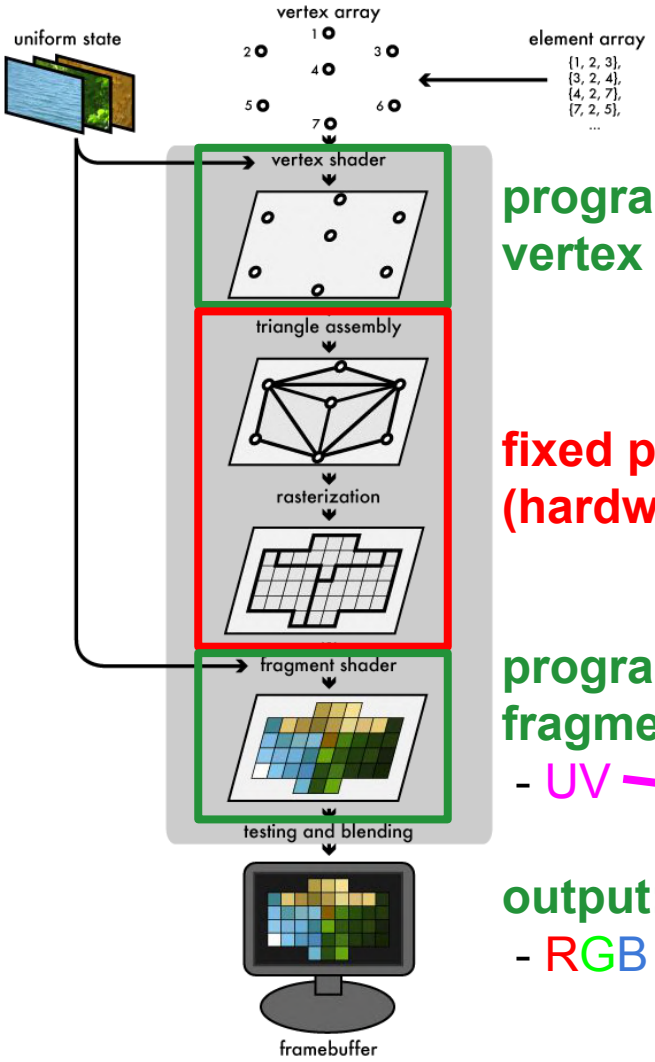
RGB

f32 depth=A

f32 depth=B

texture sampling

UV texture



programmable vertex shader

fixed pipeline (hardware)

programmable fragment shader:

- UV

output:

- RGB (3 x uchar)

Depth framebuffer
Z framebuffer

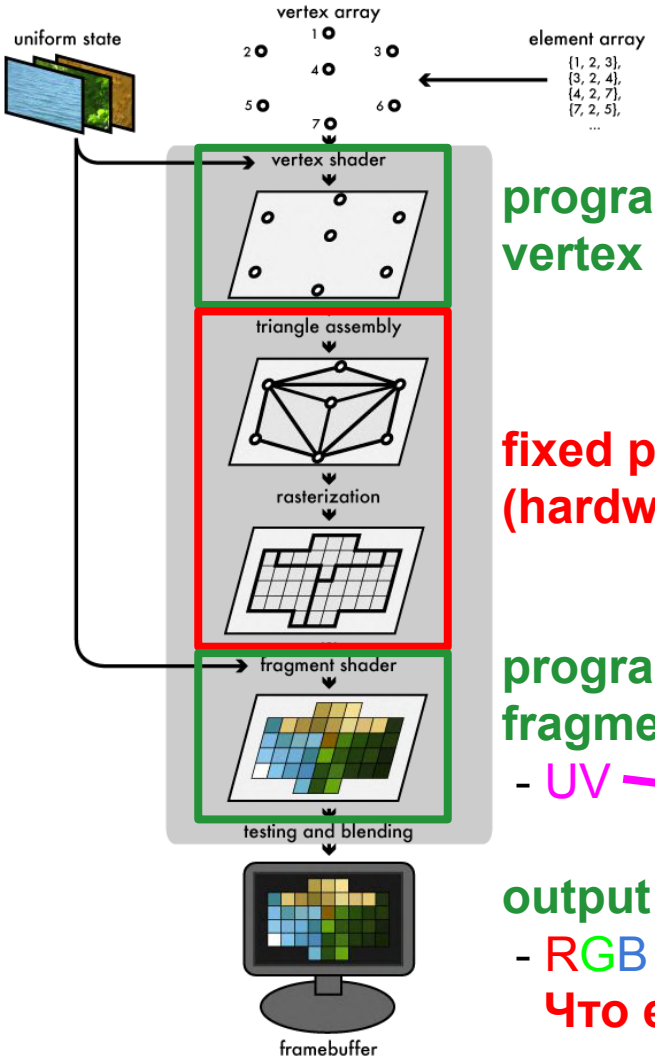
VS

f32 depth=A

f32 depth=B

texture sampling

UV texture



programmable vertex shader

fixed pipeline (hardware)

programmable fragment shader:

- UV

output:

- RGB (3 x uchar)

Что еще?

Depth framebuffer
Z framebuffer

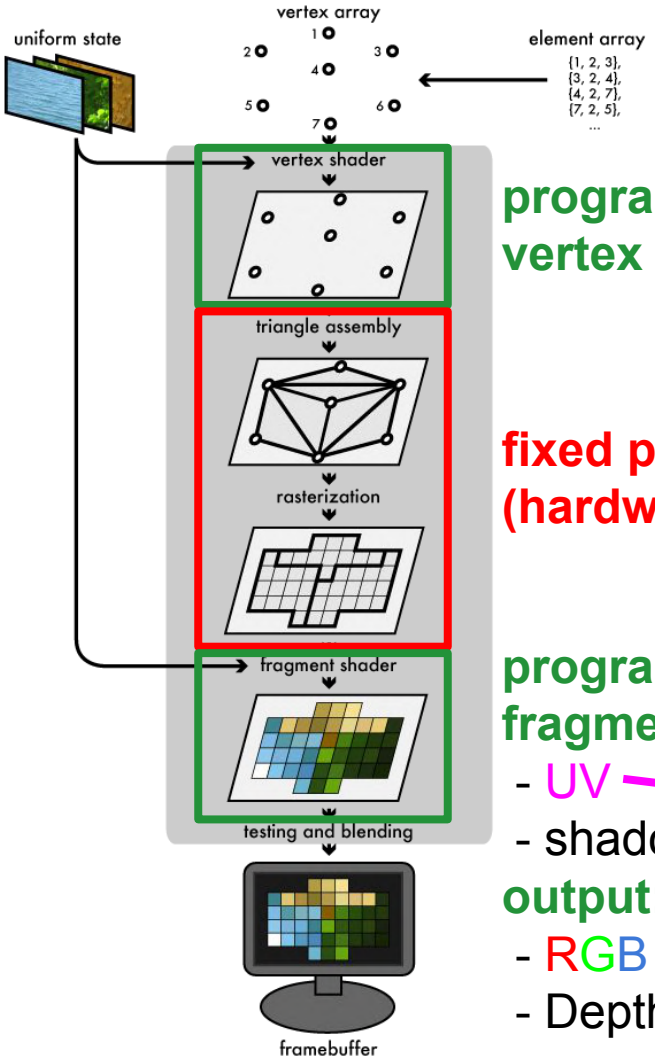
f32 depth=A

VS

f32 depth=B

texture sampling

UV texture



programmable vertex shader

fixed pipeline (hardware)

programmable fragment shader:

- UV
- shadows/lighting

output:

- RGB (3 x uchar)
- Depth (32-bit float)

Depth framebuffer
Z framebuffer

f32 depth=A

VS

f32 depth=B

texture sampling

UV texture

How to depth test?

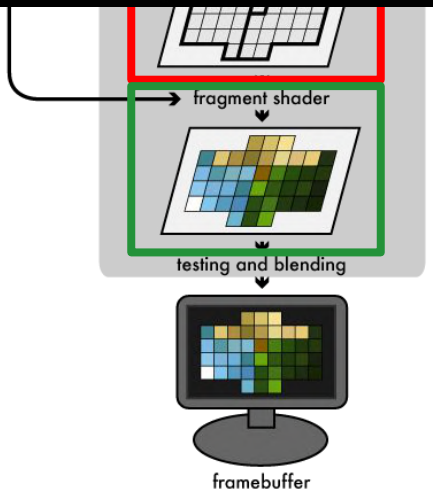
- Don't have ROP or depth test hardware
- Need Z-buffering
  - Can't serialize at tiles
  - Many tris may be in parallel for single tile or even single pixel
- Use 64 bit atomics!
- InterlockedMax

30	27	7
Depth	Visible cluster index	Triangle index

mebuffer
uffer

RGB VS RGB

f32 depth=A f32 depth=B

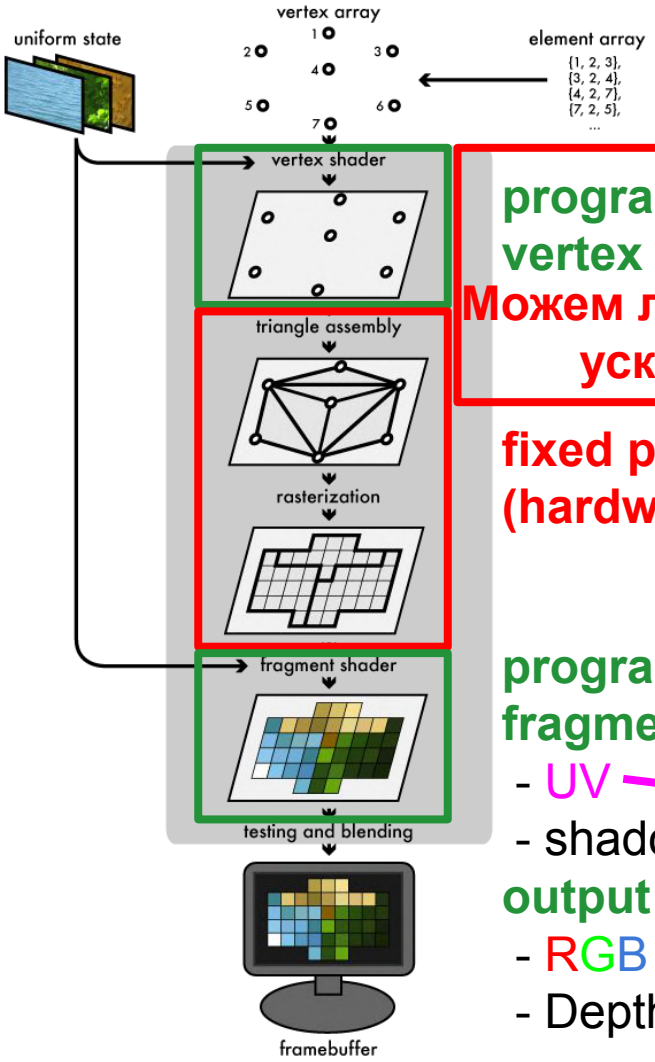


programmable fragment shader:
- UV
- shadows/lighting

output:
- RGB (3 x uchar)
- Depth (32-bit float)

texture sampling

UV texture



programmable vertex shader

Можем ли мы как-то ускорить?

fixed pipeline (hardware)

programmable fragment shader:

- UV
- shadows/lighting

output:

- RGB (3 x uchar)
- Depth (32-bit float)

Depth framebuffer
Z framebuffer

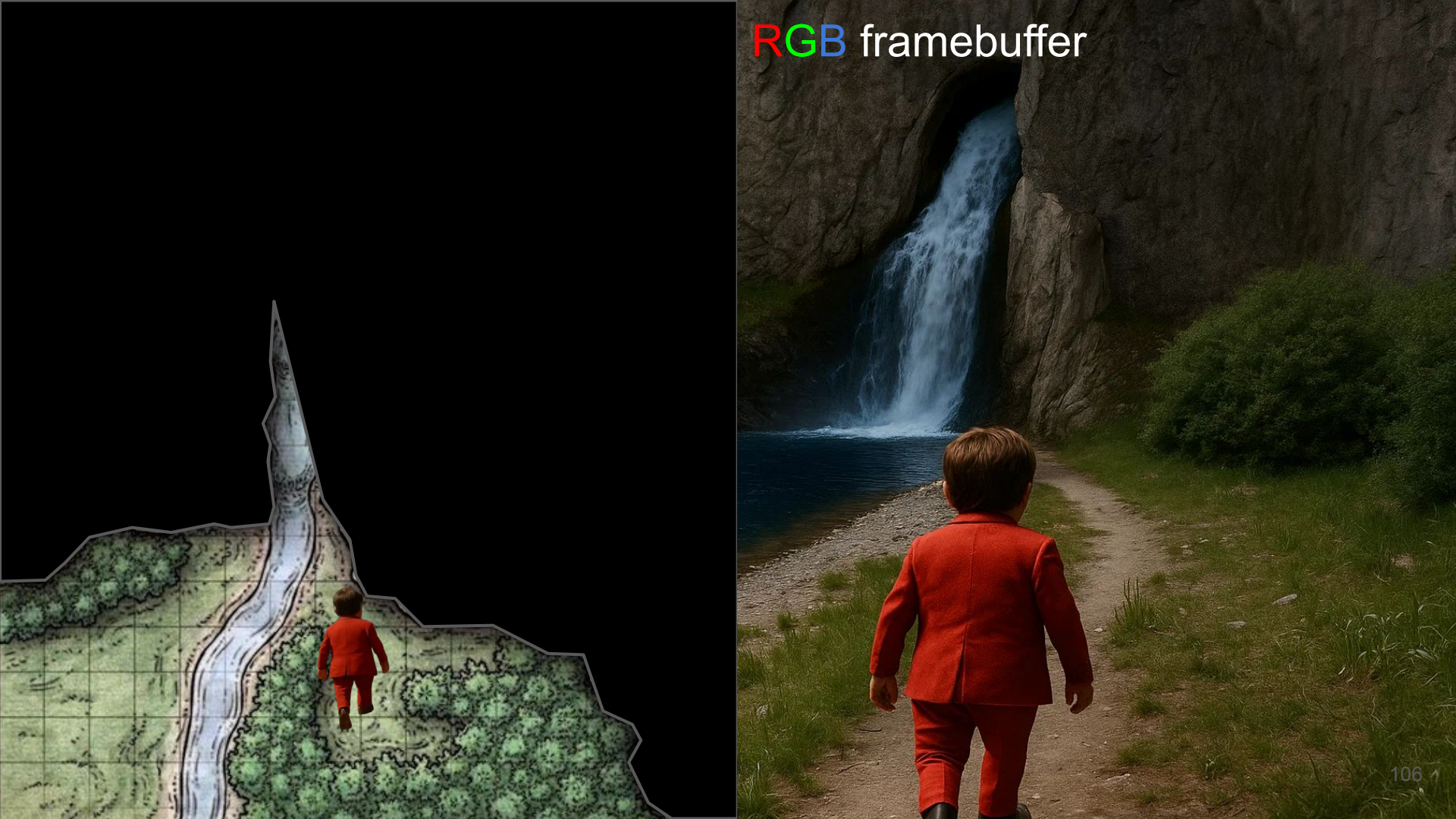
VS

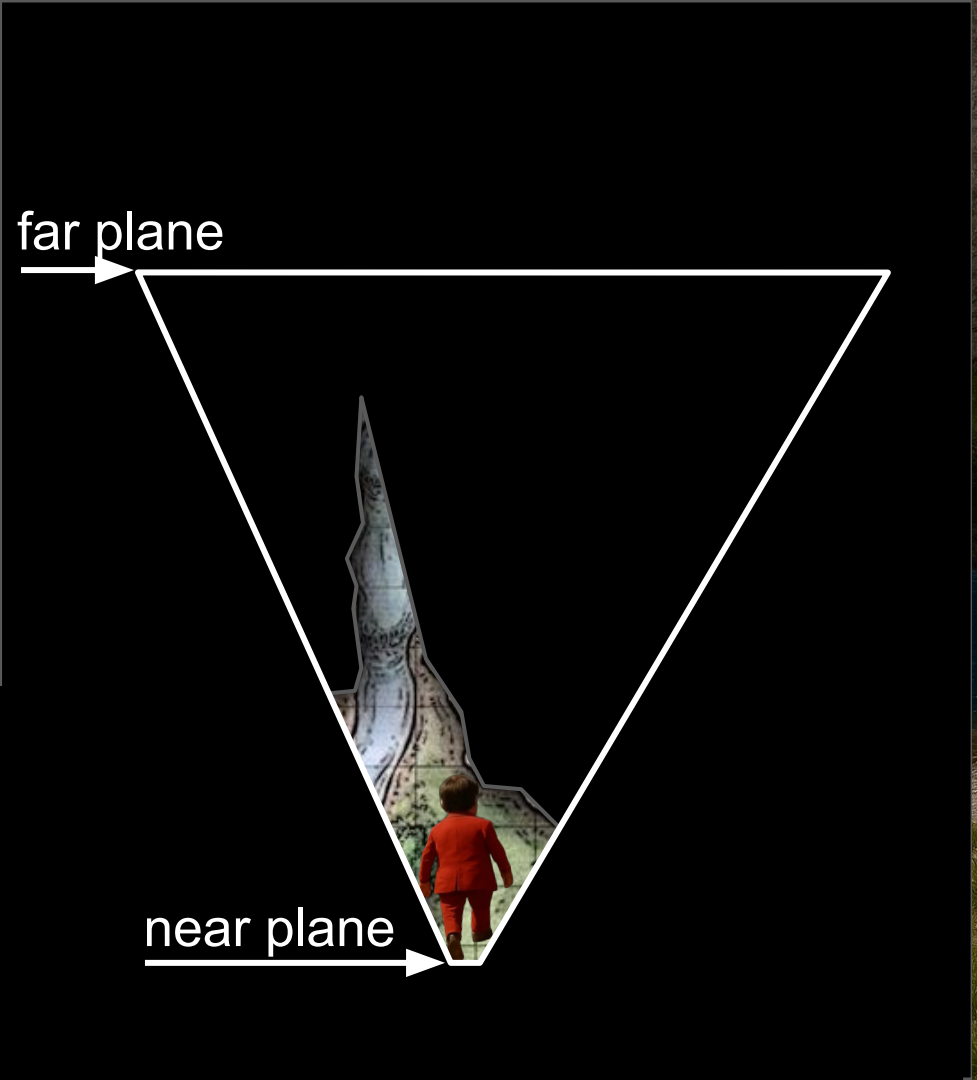
f32 depth=A

f32 depth=B

texture sampling

UV texture


RGB framebuffer



RGB framebuffer

far plane

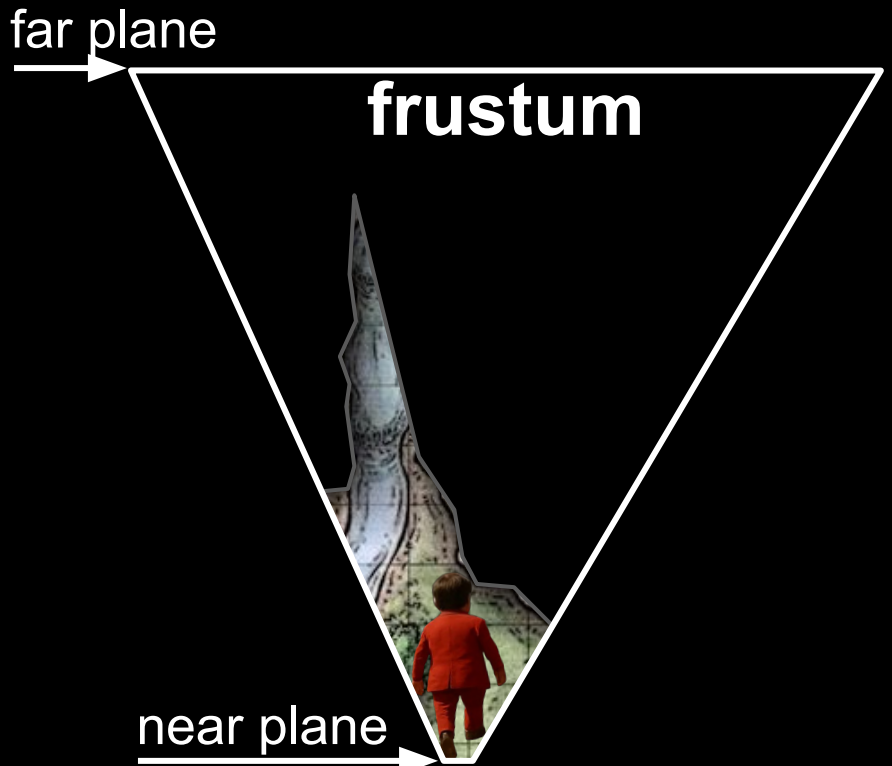
frustum

near plane

frustum culling

RGB framebuffer

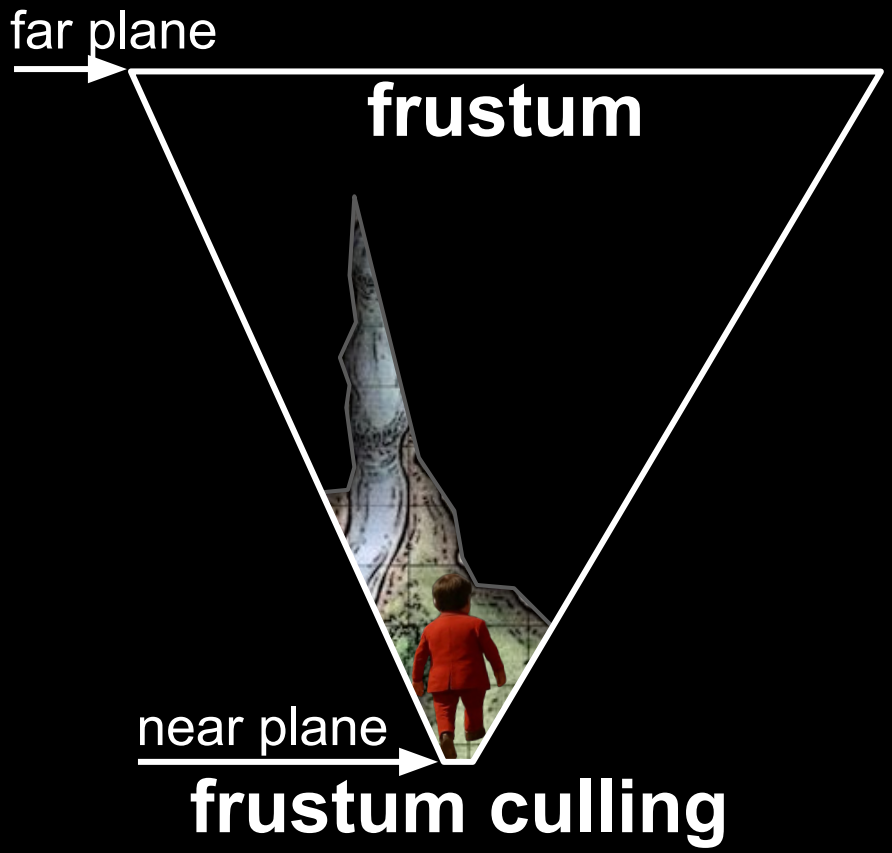
Придется вычислять **в каждой вершине!**
Как ускорить **vertex shader?**

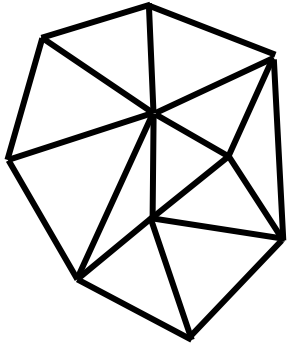


frustum culling

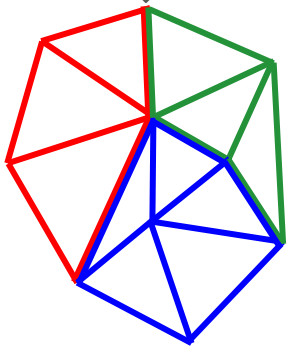
RGB framebuffer

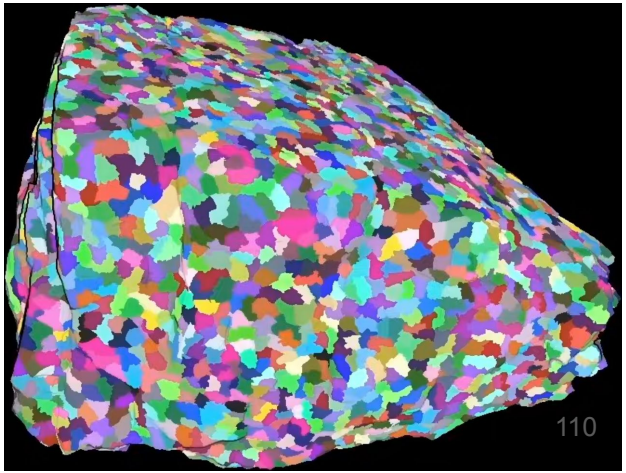
Придется вычислять **в каждой вершине!**
Как ускорить **vertex shader?**



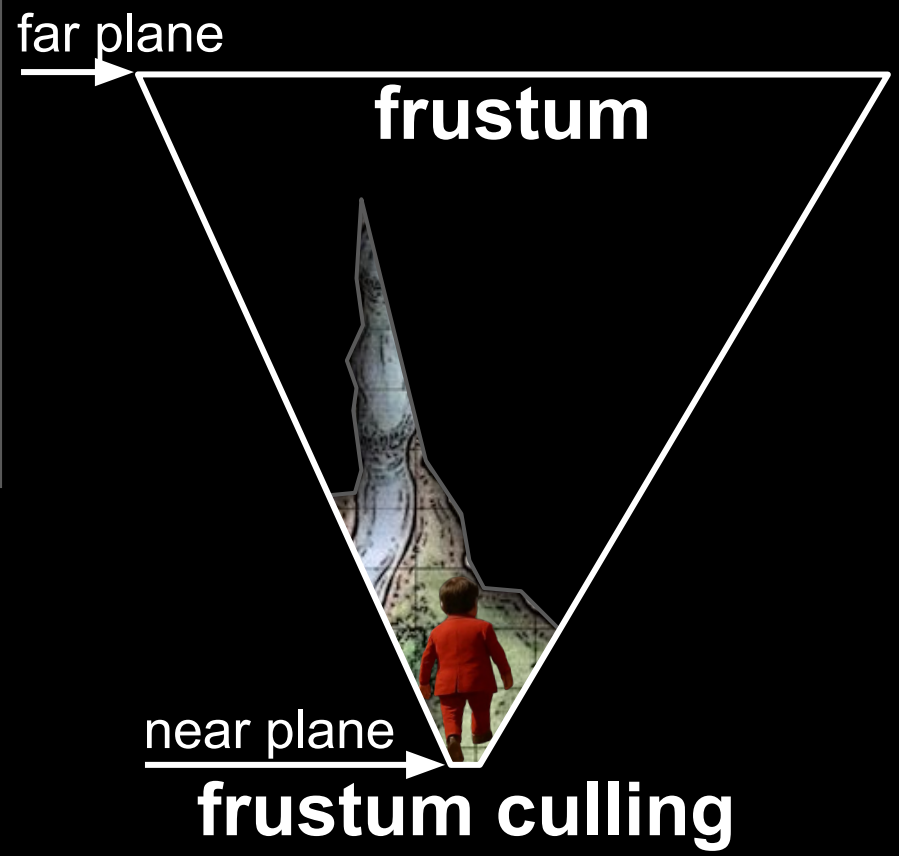


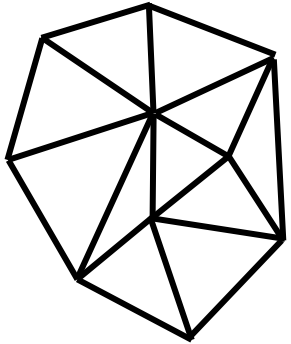
объединим в кластеры
по 128 треугольников



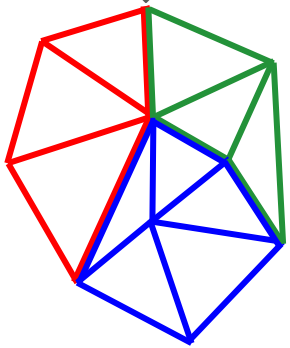


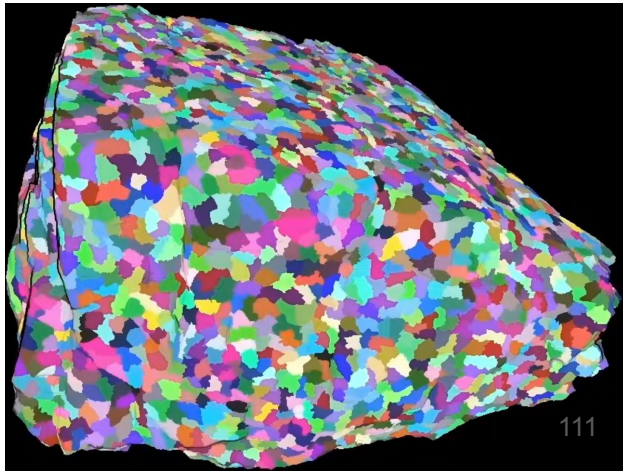
Придется вычислять **в каждой вершине!**
Как ускорить **vertex shader?**
Как для кластера выполнить frustum culling?



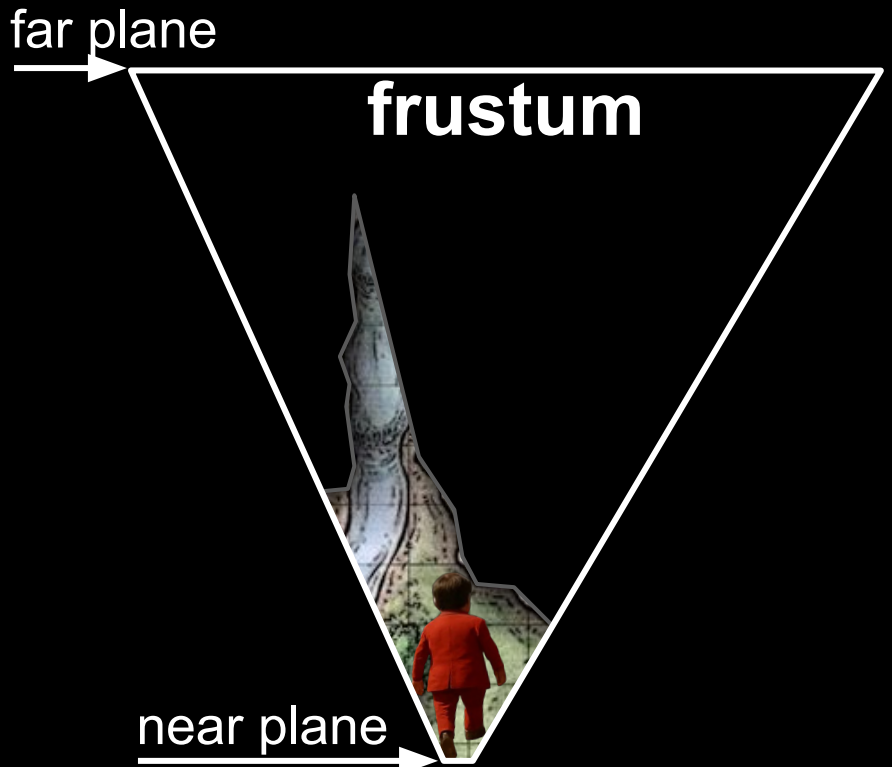


объединим в кластеры по 128 треугольников

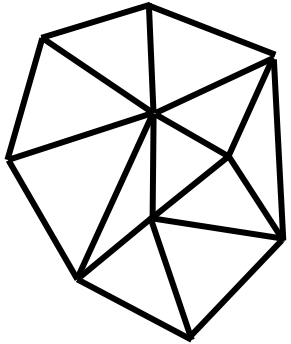


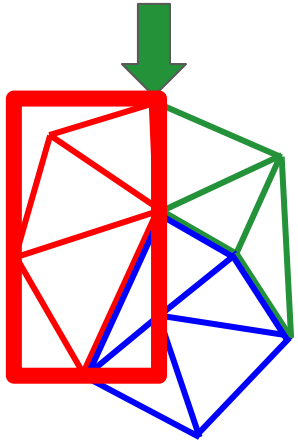


Придется вычислять **в каждой вершине!**
Как ускорить **vertex shader?**
Как для кластера выполнить frustum culling?



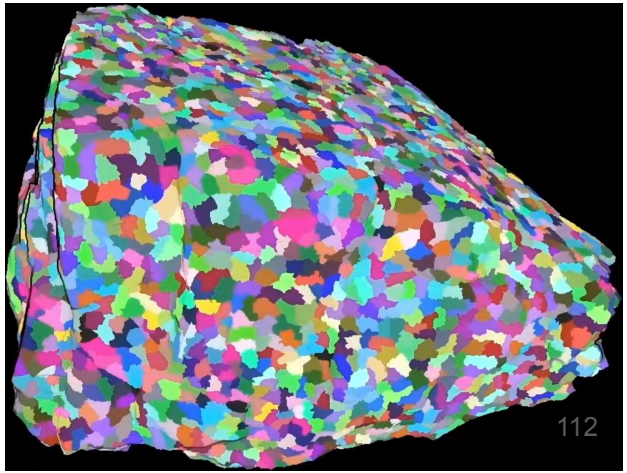
frustum culling



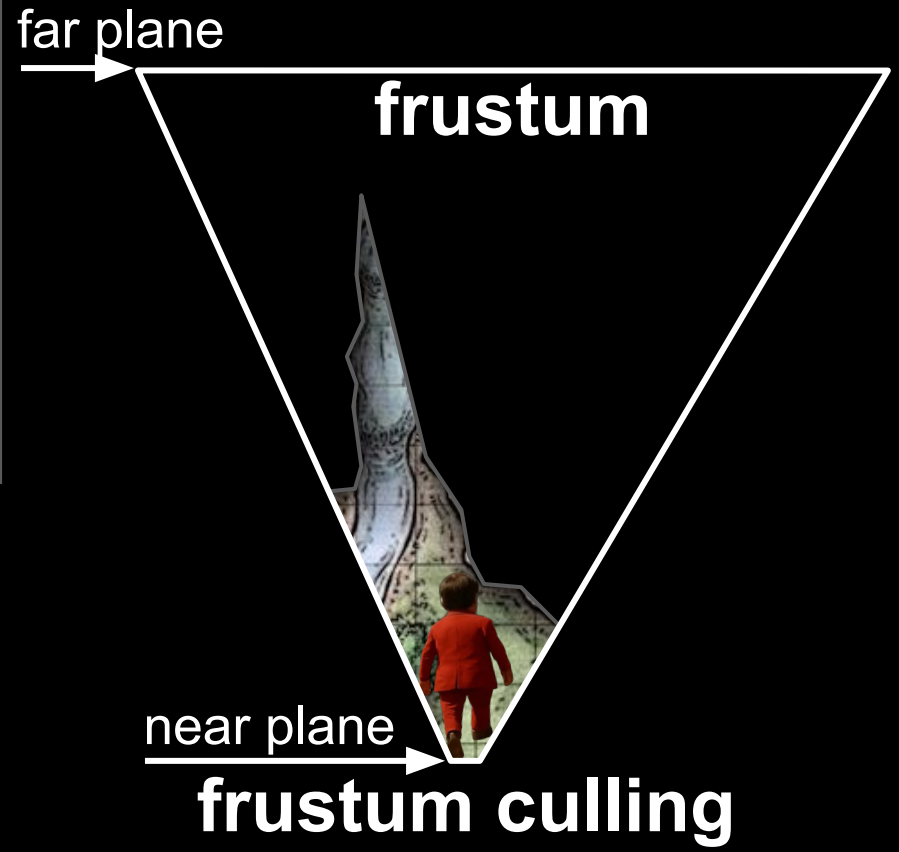


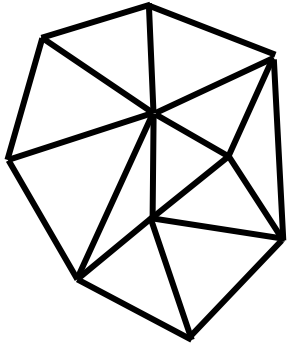
AABB

объединим в кластеры по 128 треугольников

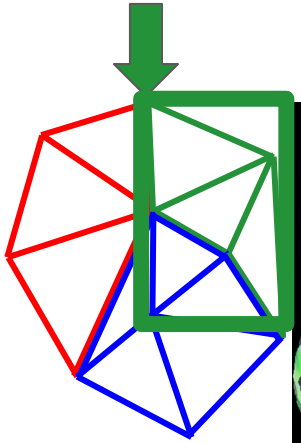


Придется вычислять **в каждой вершине!**
Как ускорить **vertex shader?**
Как для кластера выполнить frustum culling?

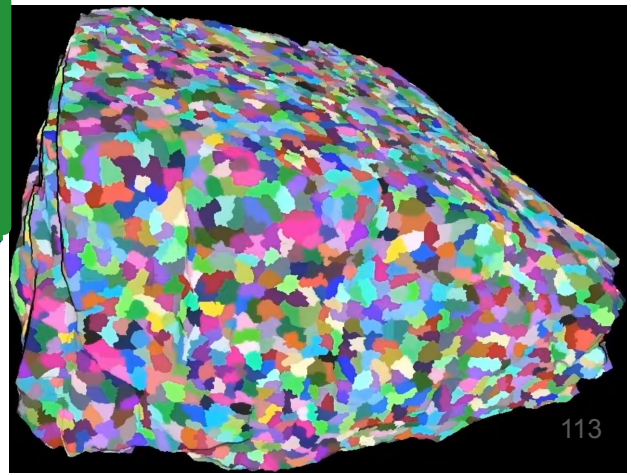




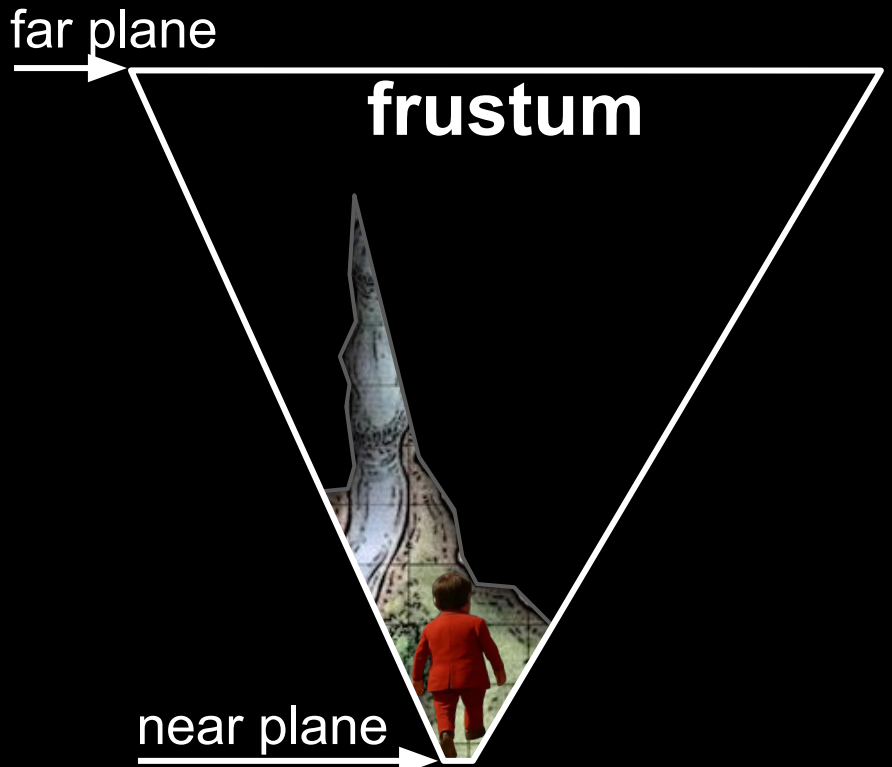
объединим в кластеры
по 128 треугольников



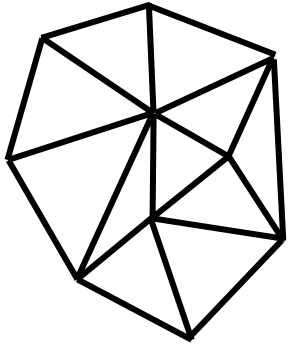
AABB



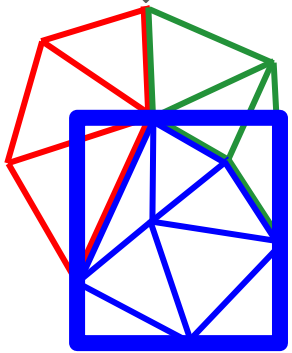
Придется вычислять **в каждой вершине!**
Как ускорить **vertex shader?**
Как для кластера выполнить frustum culling?



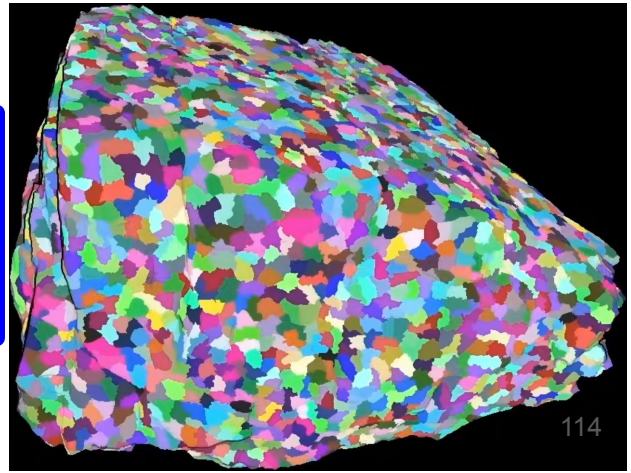
frustum culling



объединим в кластеры по 128 треугольников

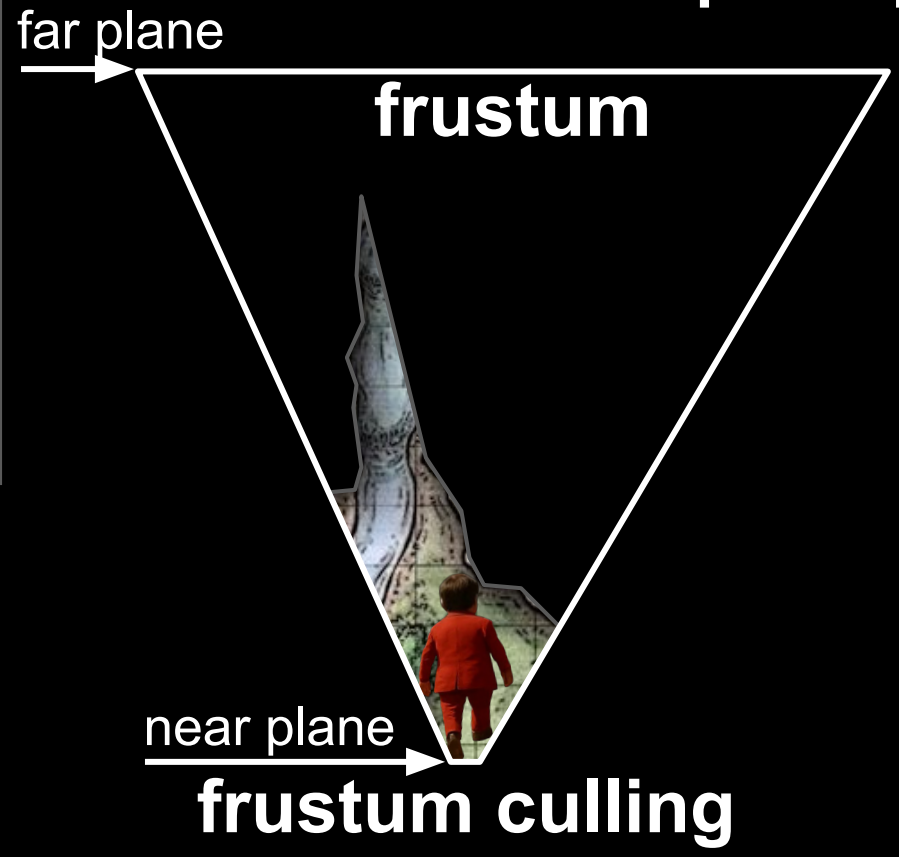


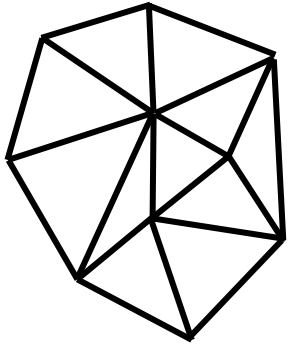
AABB



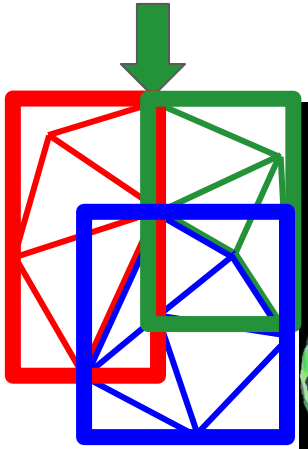
Теперь проецируем в экран **AABB** каждого кластера! И делаем frustum culling!

x128 speedup!

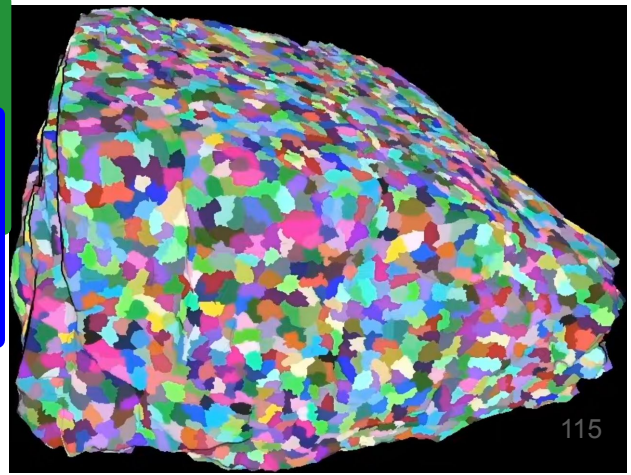




объединим в кластеры по 128 треугольников

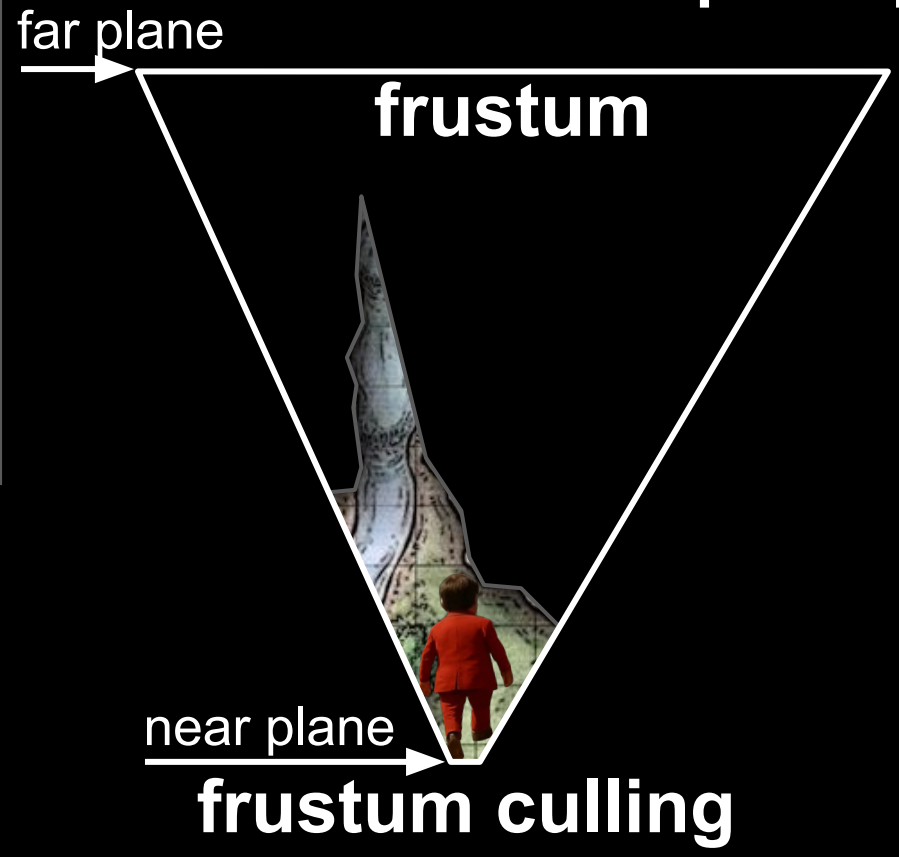


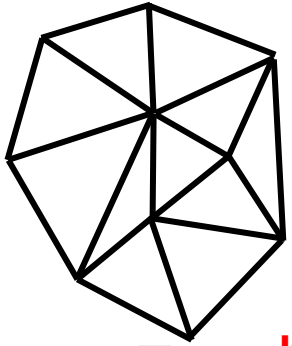
AABB



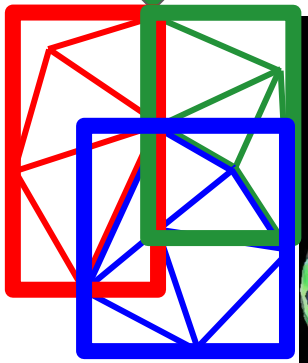
Теперь проецируем в экран **AABB** каждого кластера! И делаем frustum culling!

x128 speedup!

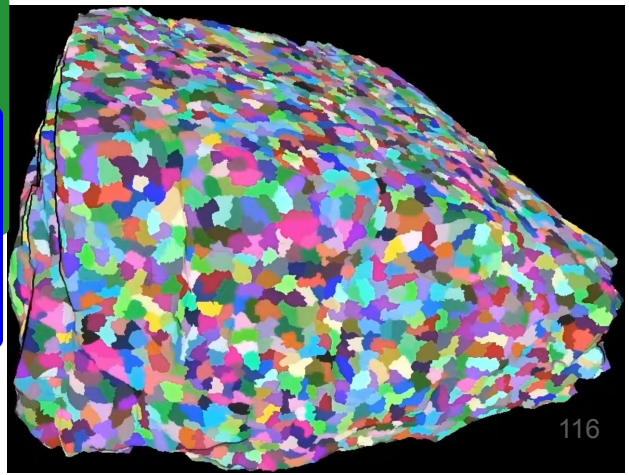




Как реализовать?
(graph partitioning)

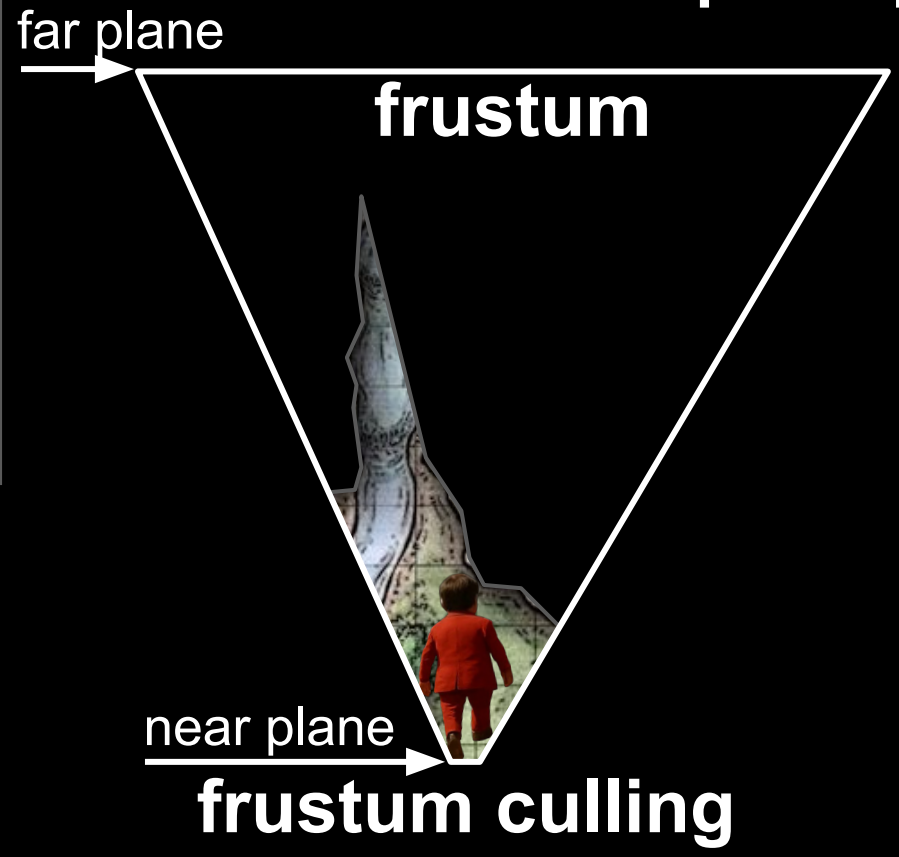


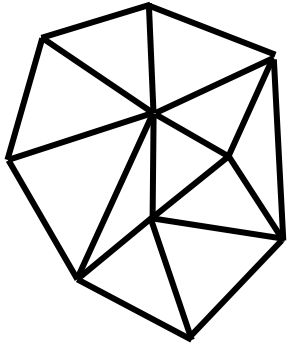
AABB



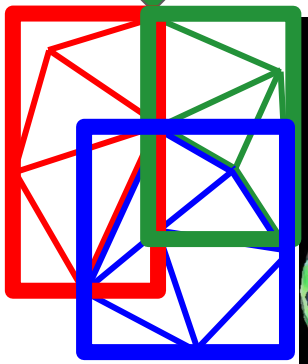
Теперь проецируем в экран **AABB** каждого кластера! И делаем frustum culling!

x128 speedup!

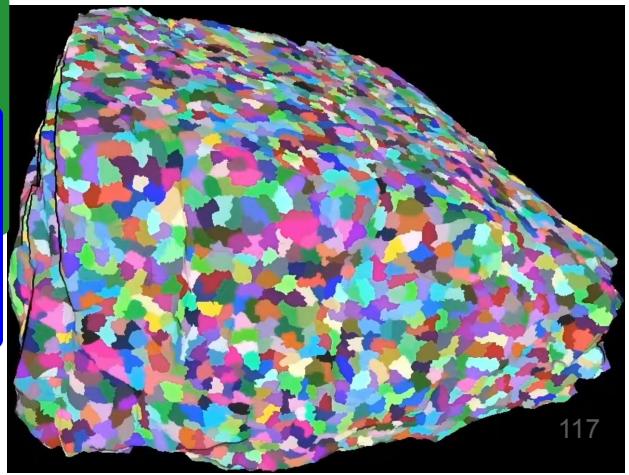


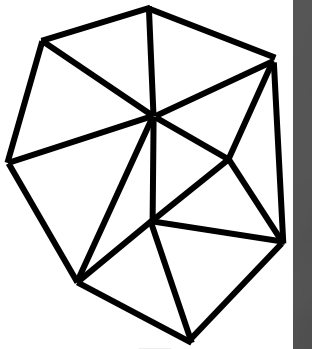


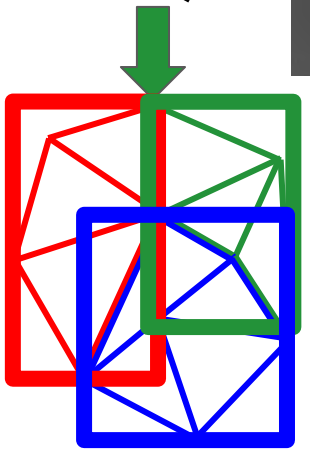
Например graph partitioning через **METIS**



AABB



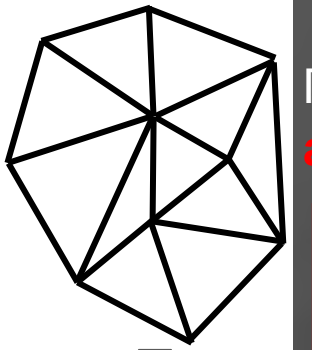


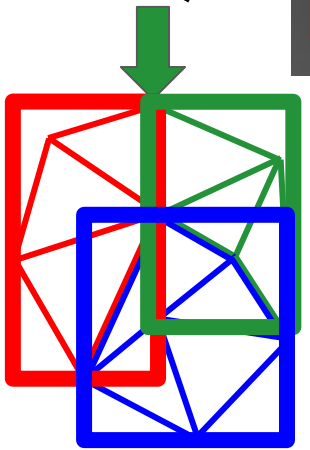


AABB

Depth framebuffer
Z framebuffer

fragment





AABB

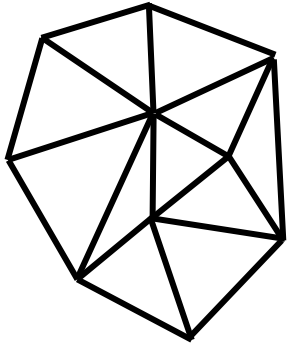
Depth framebuffer
Z framebuffer

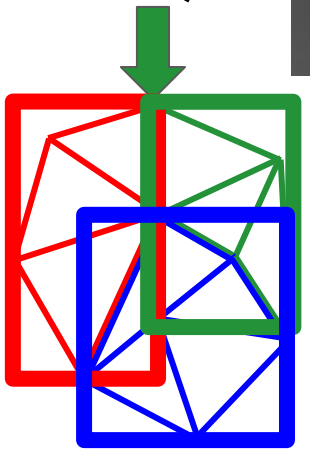
Пусть **Z-test** через
atomicMin64(RGB+Z)

RGB
+Z=A

VS

RGB
+Z=B





AABB

Depth framebuffer
Z framebuffer

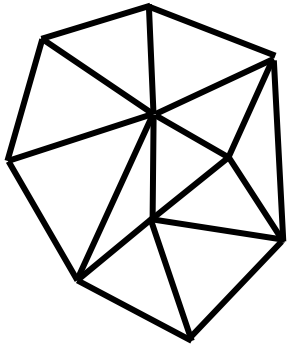
Пусть **Z-test** через
atomicMin64(RGB+Z)

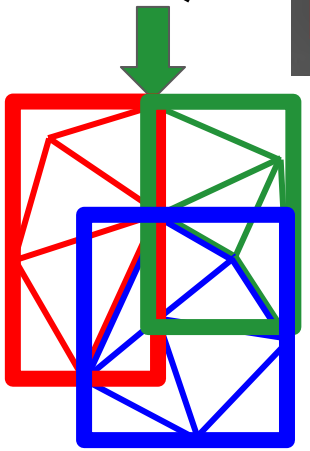
RGB
+Z=A

VS

RGB
+Z=B

Можем ли мы как-то ускорить **Z test**?





AABB

Depth framebuffer
Z framebuffer

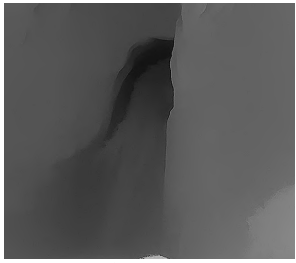
Пусть **Z-test** через
atomicMin64(RGB+Z)

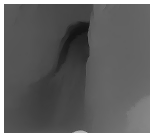
RGB
+Z=A

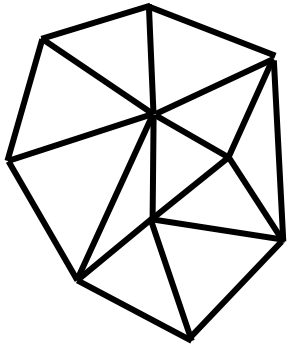
VS

Hierarcal Z-Buffer

x2







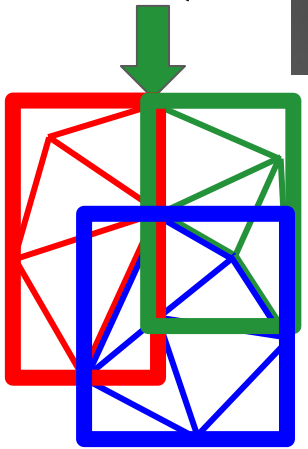
Depth framebuffer
Z framebuffer

Пусть **Z-test** через
atomicMin64(RGB+Z)

RGB
+Z=A

VS

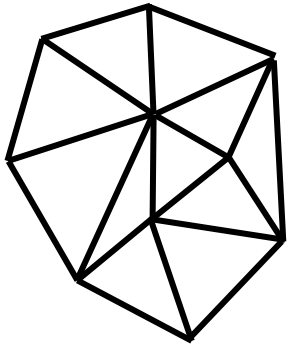
RGB
+Z=B

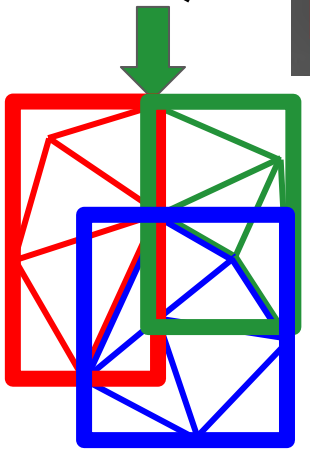


AABB Можем ли мы как-то ускорить **Z test для моста**?

Hierarcal Z-Buffer

x2





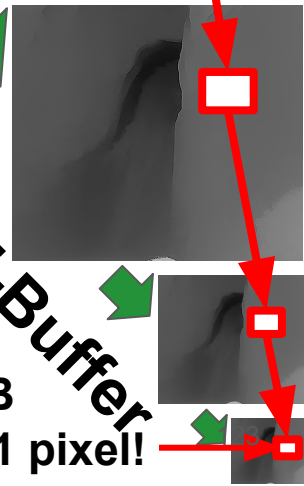
Depth framebuffer
Z framebuffer

Пусть **Z-test** через
atomicMin64(RGB+Z

RGB
+Z=A

VS

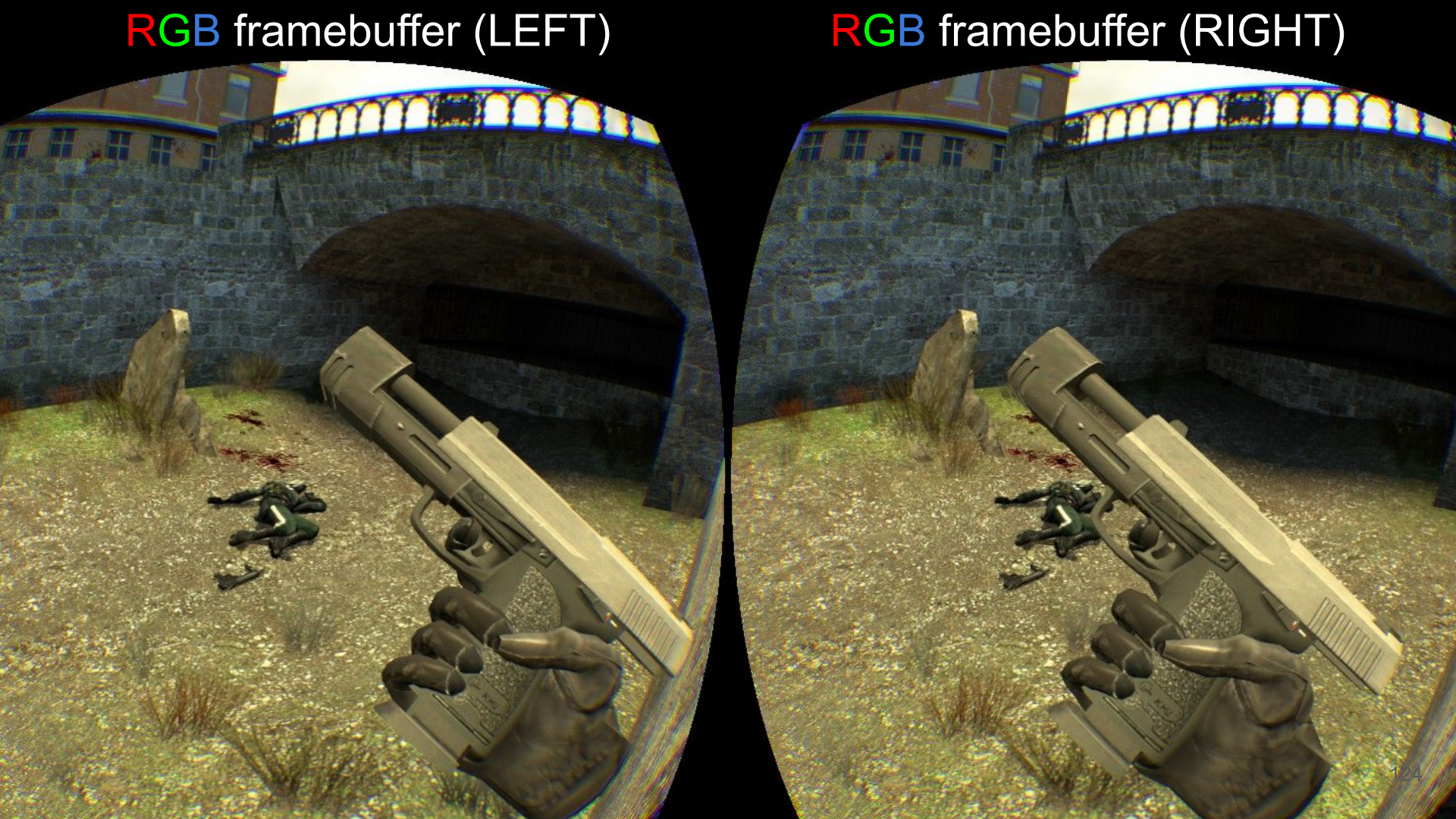
RGB
+Z=B



x2

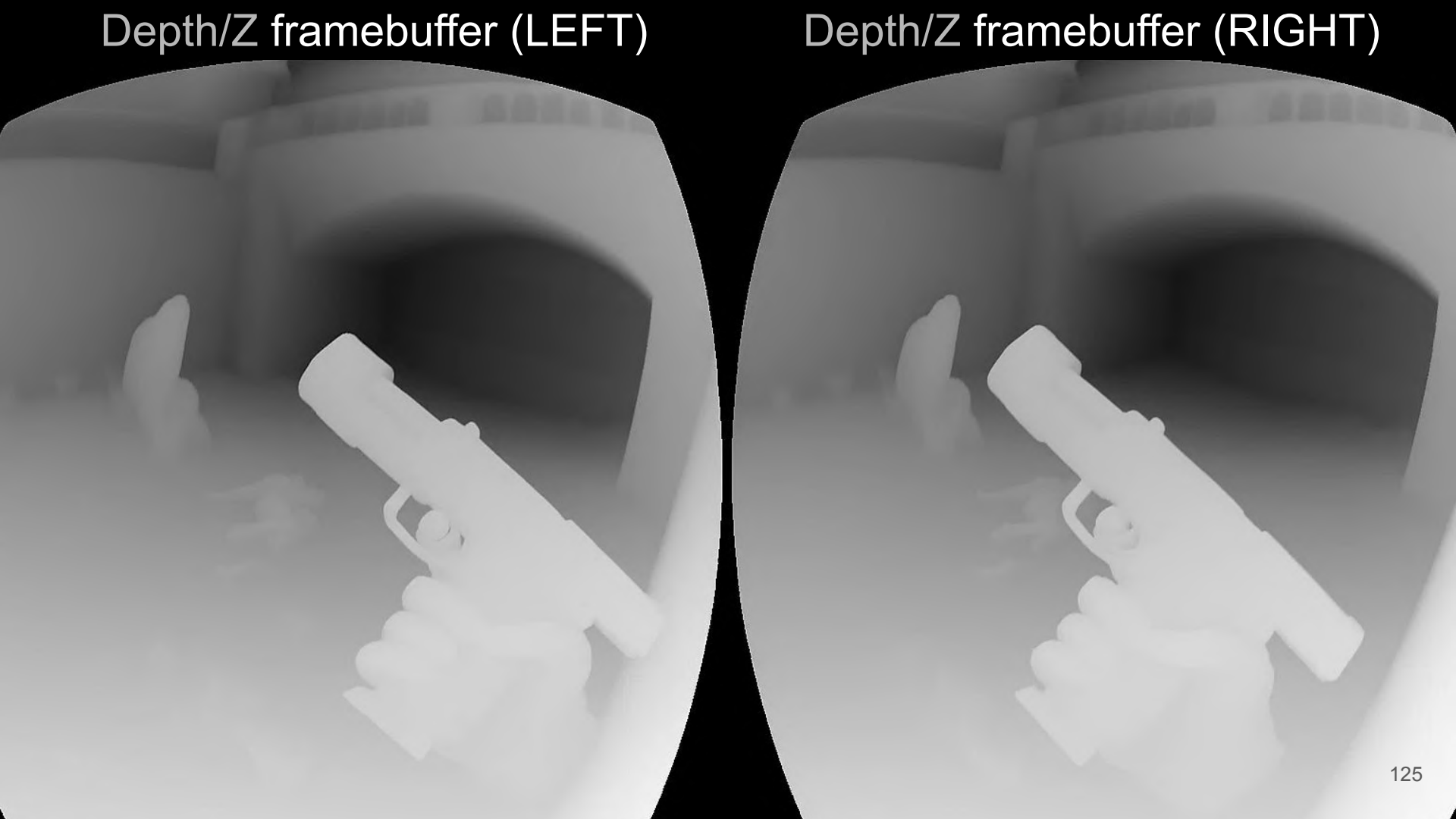
Hierarcal Z-Buffer

AABB делает **Z test** в **HZB**
на уровне где **proj(AABB) ~ 1 pixel!**


RGB framebuffer (LEFT)
RGB framebuffer (RIGHT)

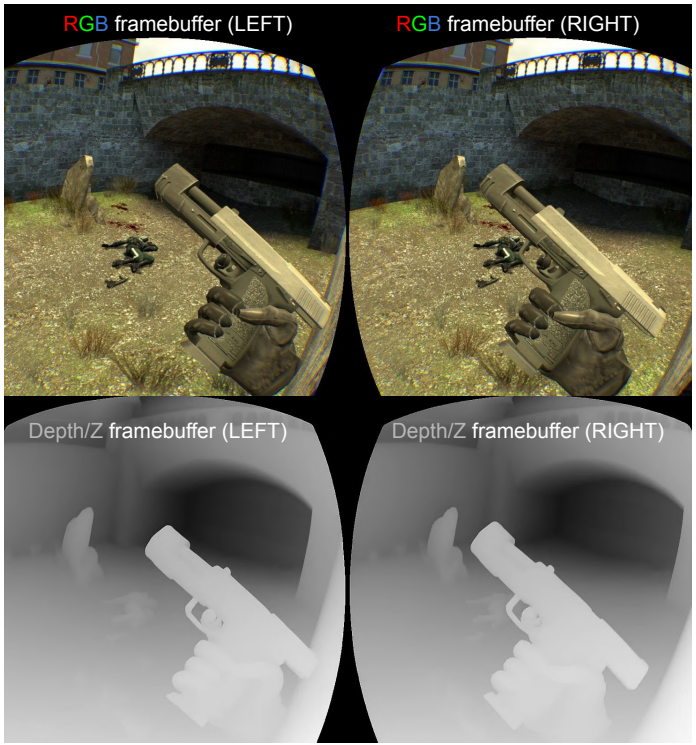
Depth/Z framebuffer (LEFT)

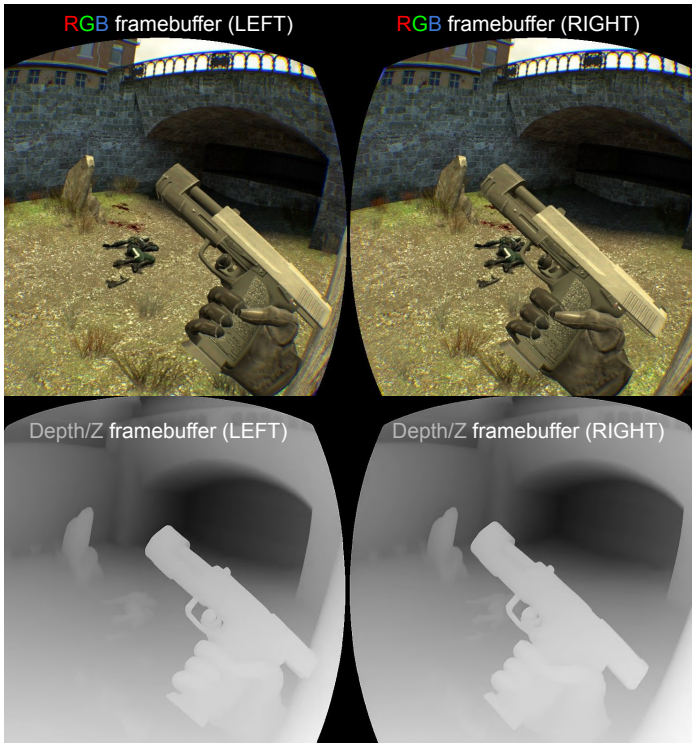
Depth/Z framebuffer (RIGHT)



Frame #i → Игрок чуть повернул голову → Frame #i+1

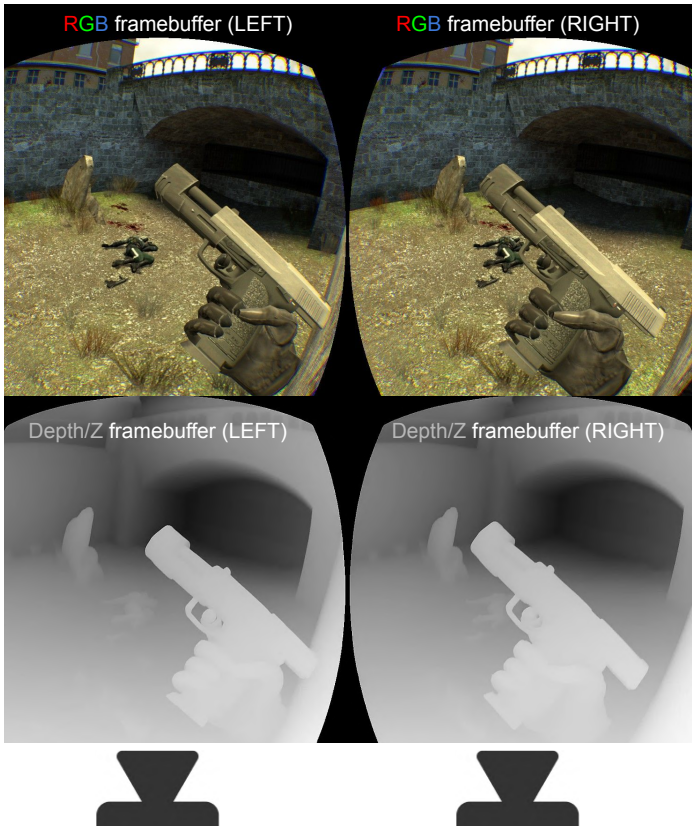
90+ FPS





Frame #i → Игрок чуть повернул голову → Frame #i+1

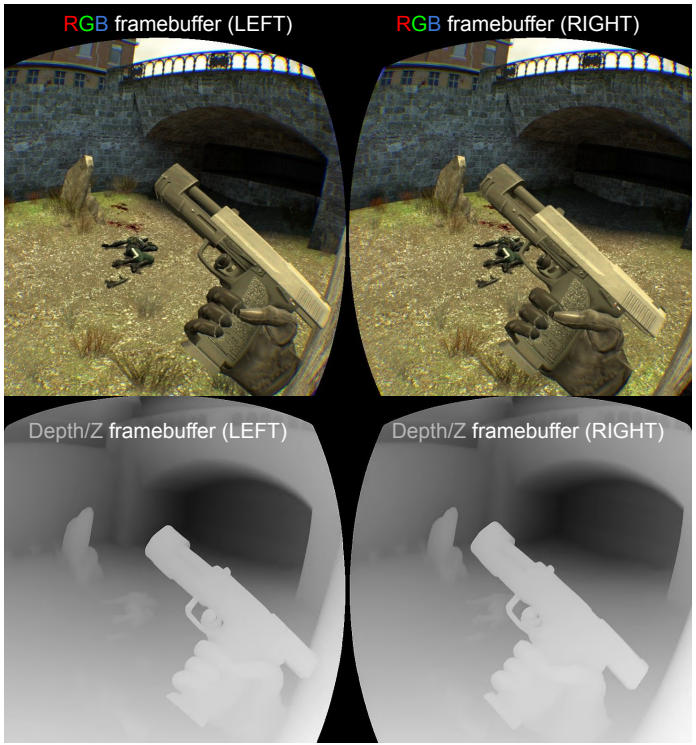
кадр дольше 1/90 секунды
(например IO)



???

Frame #i → Игрок чуть повернул голову → Frame #i+1

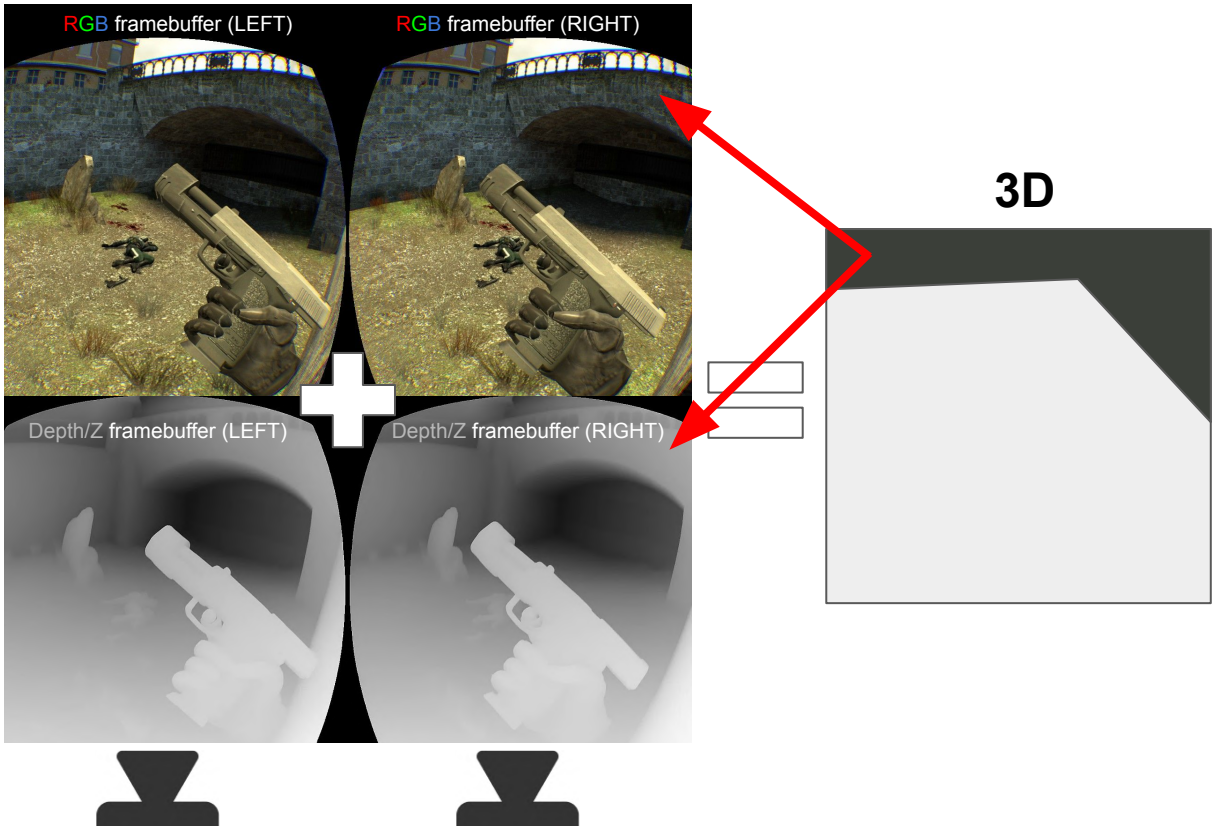
кадр дольше 1/90 секунды
(например IO)



Как избежать?

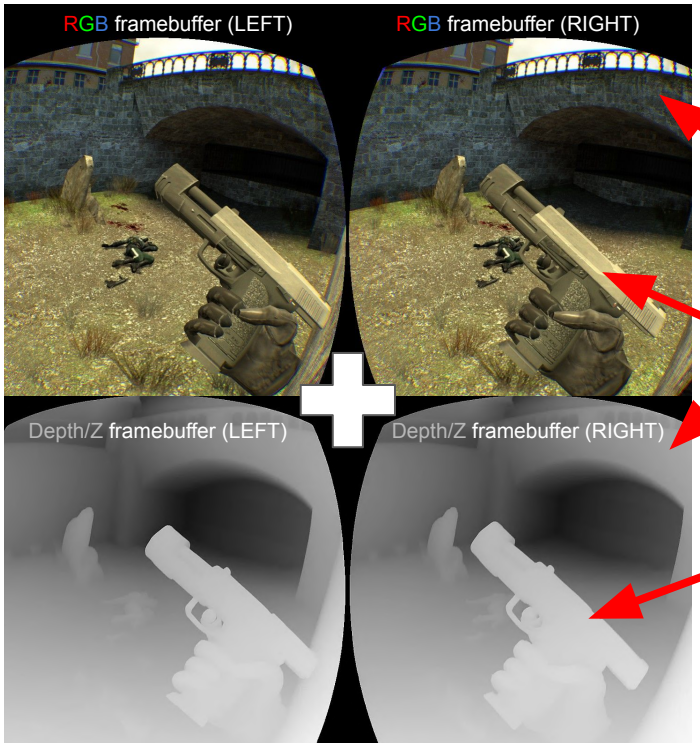
Frame #i → Игрок чуть повернул голову → Frame #i+1

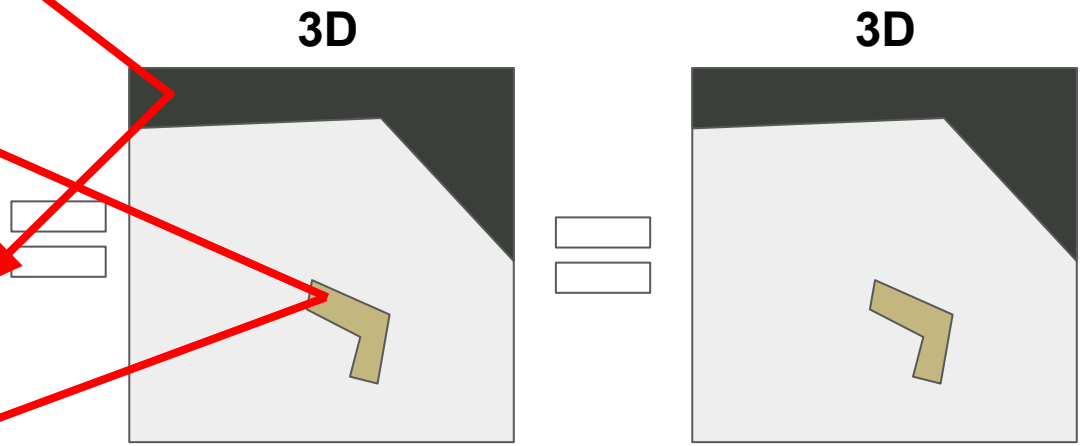
кадр дольше 1/90 секунды
(например IO)



Frame #i → Игрок чуть повернул голову → Frame #i+1

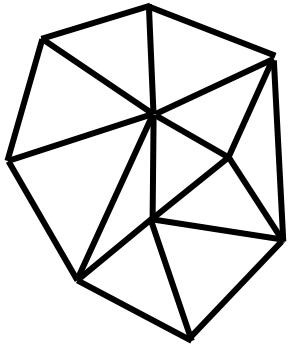
кадр дольше 1/90 секунды
(например IO)

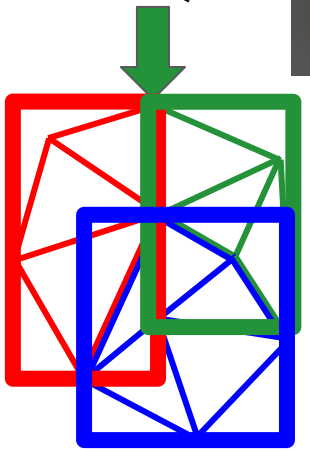




frame reprojection

Игрок чуть повернул голову





AABB

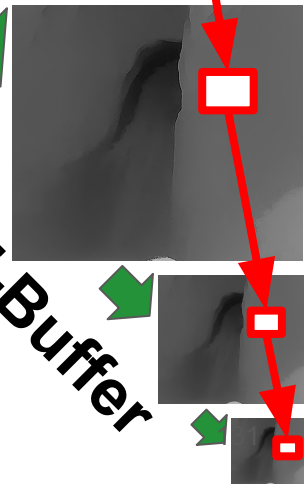
Depth framebuffer
Z framebuffer

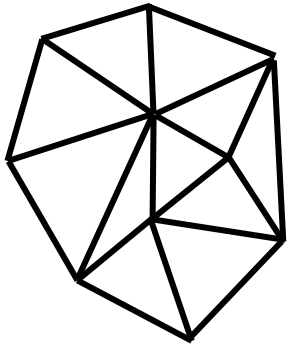
Пусть **Z-test** через
atomicMin64(RGB+Z

RGB
+Z=A

VS

RGB
+Z=B





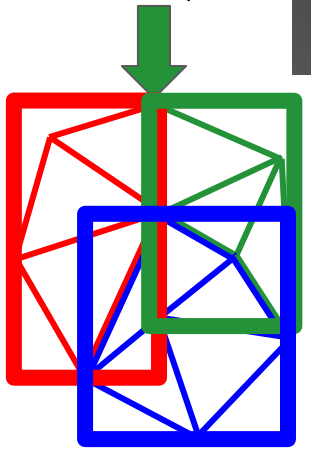
Depth framebuffer
Z framebuffer

Пусть **Z-test** через
atomicMin64(RGB+Z

RGB
+Z=A

VS

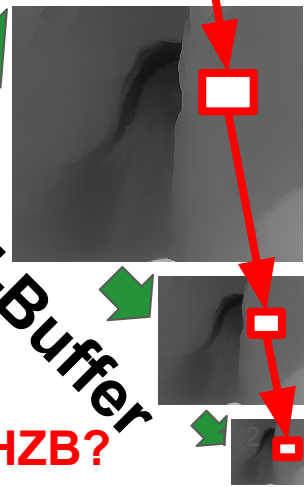
RGB
+Z=B



AABB

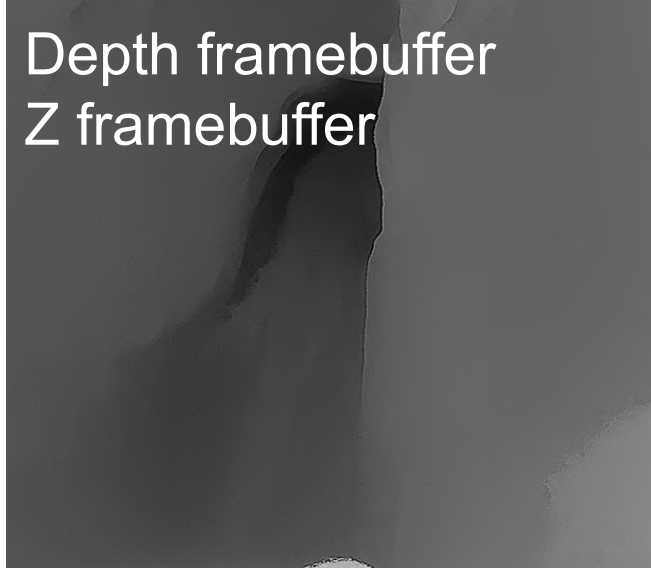
Hierarcal Z-Buffer

x2



Рисуем кадр, **откуда взять HZB?**

Depth framebuffer
Z framebuffer



x2

Hierarcal Z-Buffer

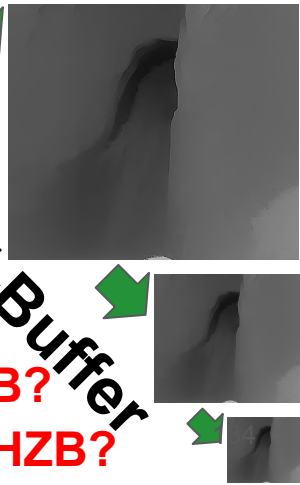
Что мы **можем** сказать о **HZB?**

Что мы **не можем** сказать о **HZB?**

Depth framebuffer
Z framebuffer

x2

Hierarcal Z-Buffer

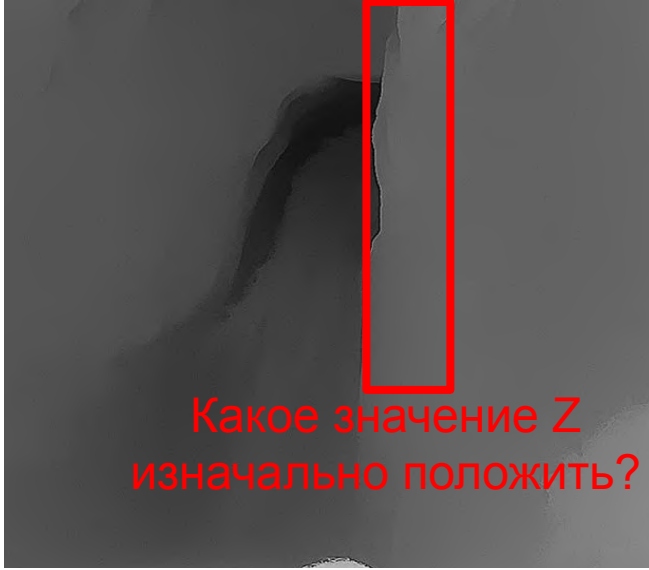


Что мы **можем** сказать о **HZB?**

Что мы **не можем** сказать о **HZB?**

раньше не видели

Камера двинулась влево



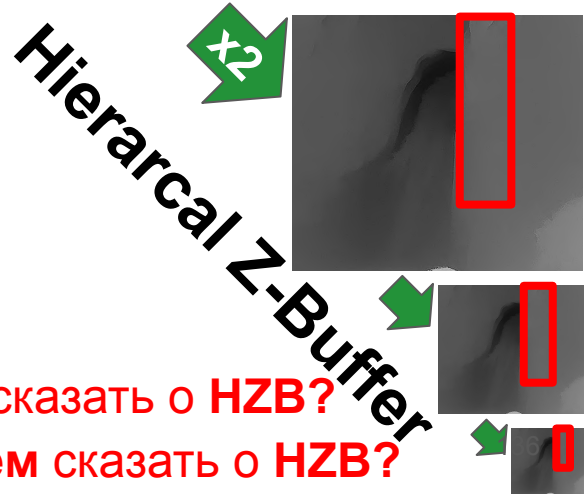
Hierarcal Z-Buffer

Что мы **можем** сказать о **HZB?**

Что мы **не можем** сказать о **HZB?**

раньше не видели

Камера двинулась влево



Что мы **можем** сказать о **HZB?**

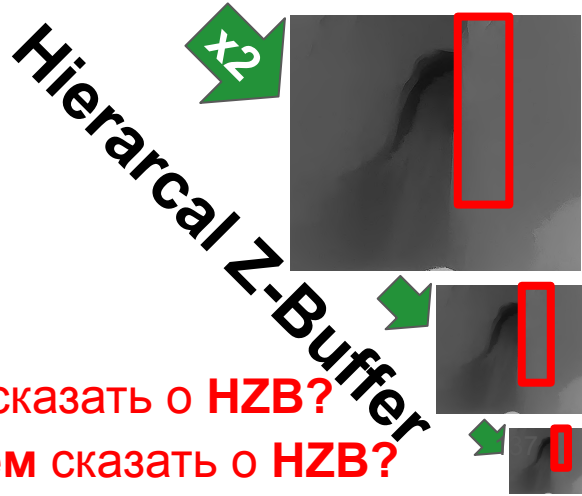
Что мы **не можем** сказать о **HZB?**

раньше не видели

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

Камера двинулась влево



Что мы **можем** сказать о **HZB**?

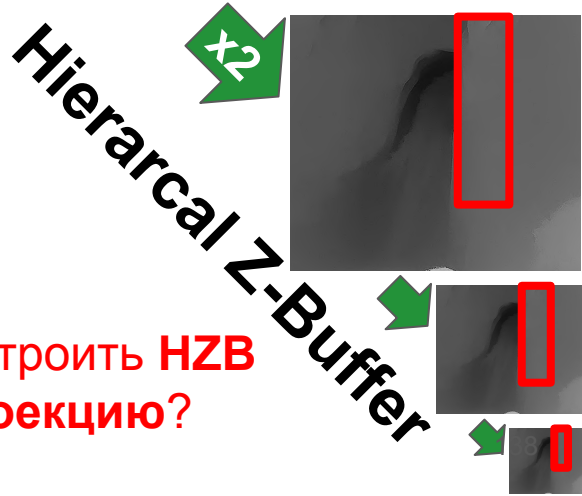
Что мы **не можем** сказать о **HZB**?

раньше не видели

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

Камера двинулась влево



Как быстро построить **HZB не через репроекцию**?

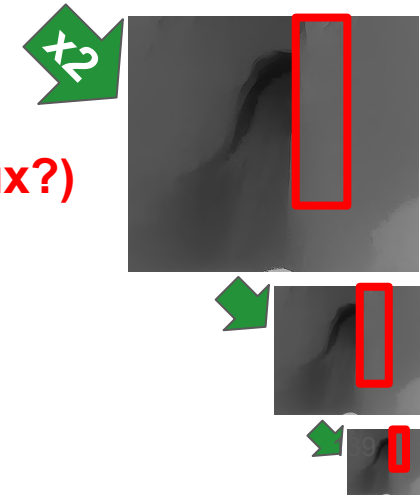
раньше не видели

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

Камера двинулась влево

1) Берем кластеры попавшие в прошлый framebuffer **(сколько их?)**



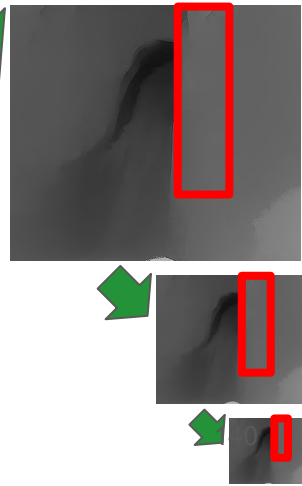
раньше не видели

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

Камера двинулась влево

1) Берем кластеры попавшие в прошлый framebuffer - **O(screen size)**



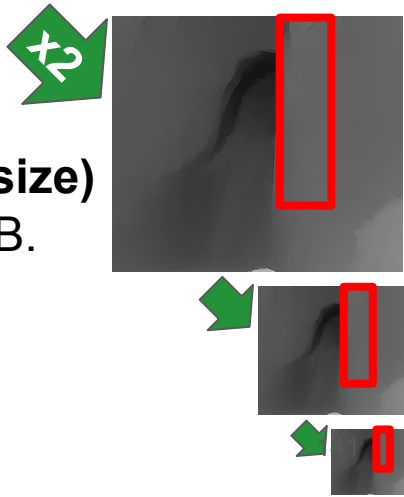
раньше не видели

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

Камера двинулась влево

1) Берем кластеры попавшие в прошлый framebuffer - **O(screen size)**
2) Растеризуем их = HZB.
(полный? точный?)



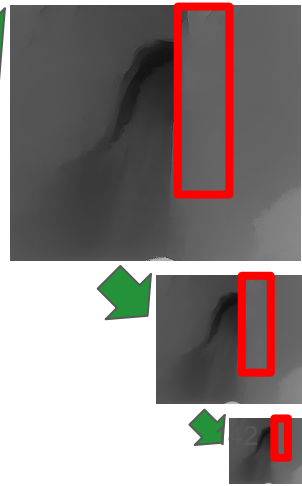
раньше не видели

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

Камера двинулась влево

1) Берем кластеры попавшие в прошлый framebuffer - **O(screen size)**
2) Растеризуем их = **~HZB**
3) Растеризуем остальных. Очень быстро т.к. **???**



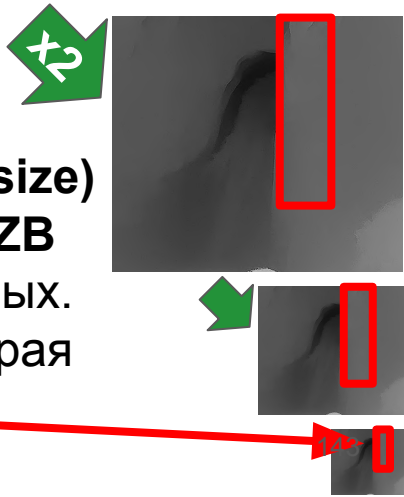
раньше не видели

все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

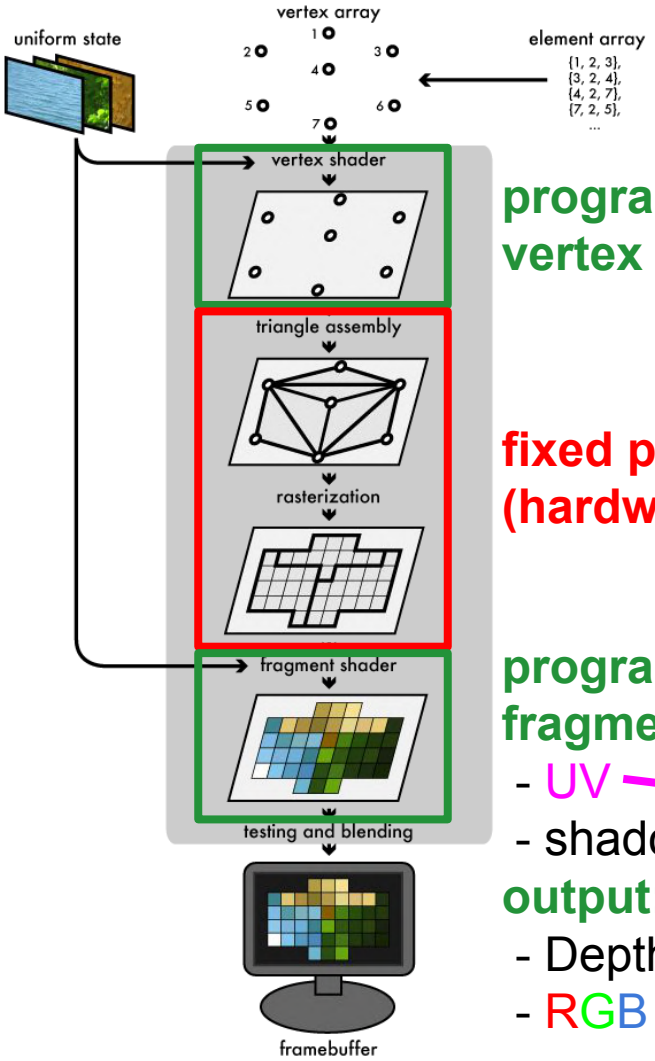
все эти **кластеры** отсечены быстрым O(1) тестом по репроекции **HZB**

Камера двинулась влево

1) Берем кластеры попавшие в прошлый framebuffer - **O(screen size)**
2) Растеризуем их = **~HZB**
3) Растеризуем остальных. Очень быстро т.к. быстрая отсечка по **~HZB**!




Перерыв!
Streaming
Pubiprocestor
VIDIA



programmable vertex shader

fixed pipeline (hardware)

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)

Depth framebuffer
Z framebuffer

RGB

VS

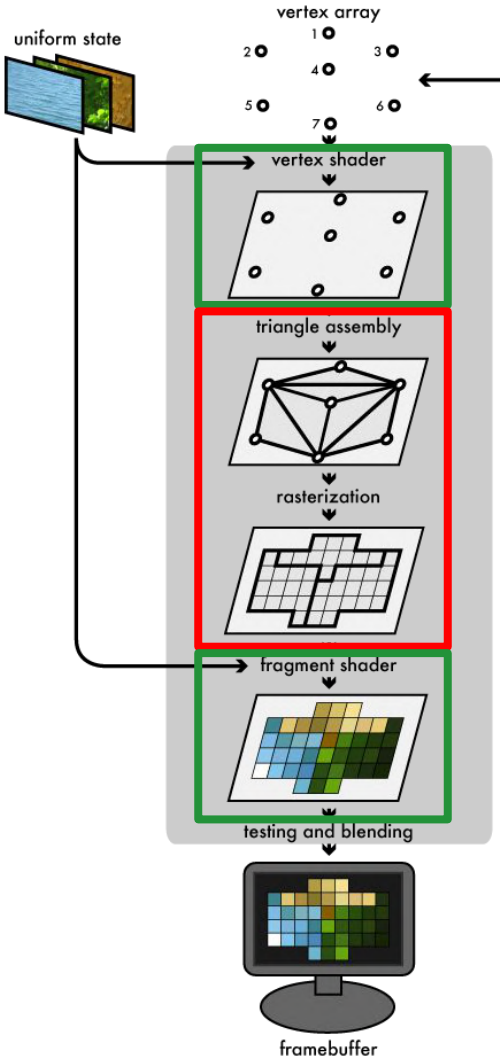
RGB

f32 depth=A

f32 depth=B

texture sampling

UV texture



programmable vertex shader

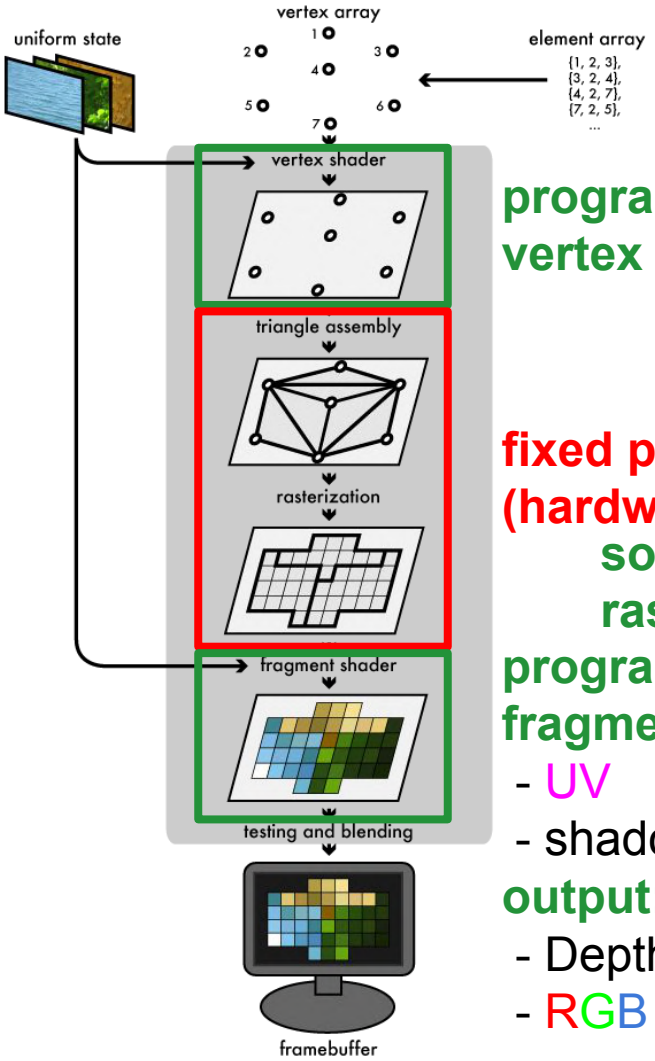
fixed pipeline (hardware) ← Очень эффективно для больших треугольников

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)



programmable vertex shader

fixed pipeline (hardware) ← Очень эффективно для больших треугольников

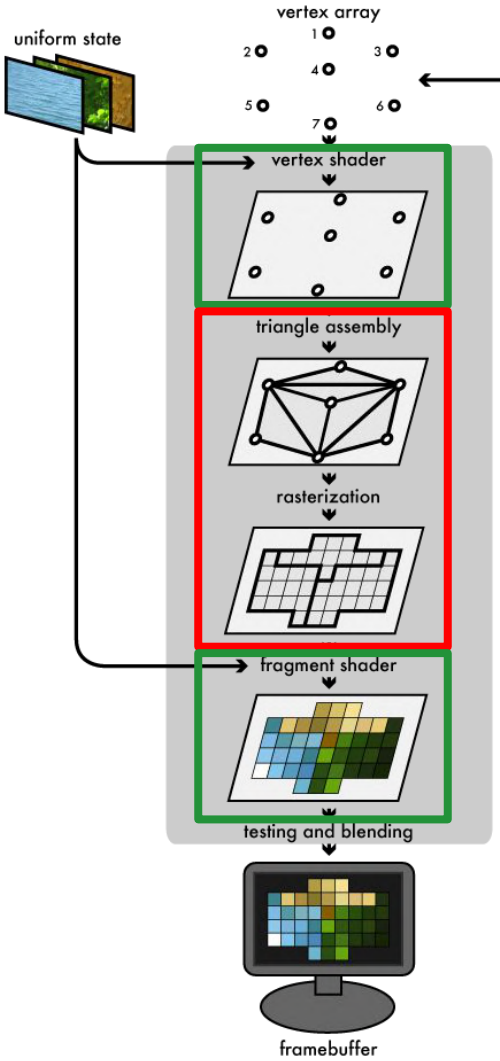
software rasterizer ← Очень эффективно для треугольников ~ 1 пиксель (у нас очень много таких!!!)

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)



programmable vertex shader

fixed pipeline (hardware)

software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

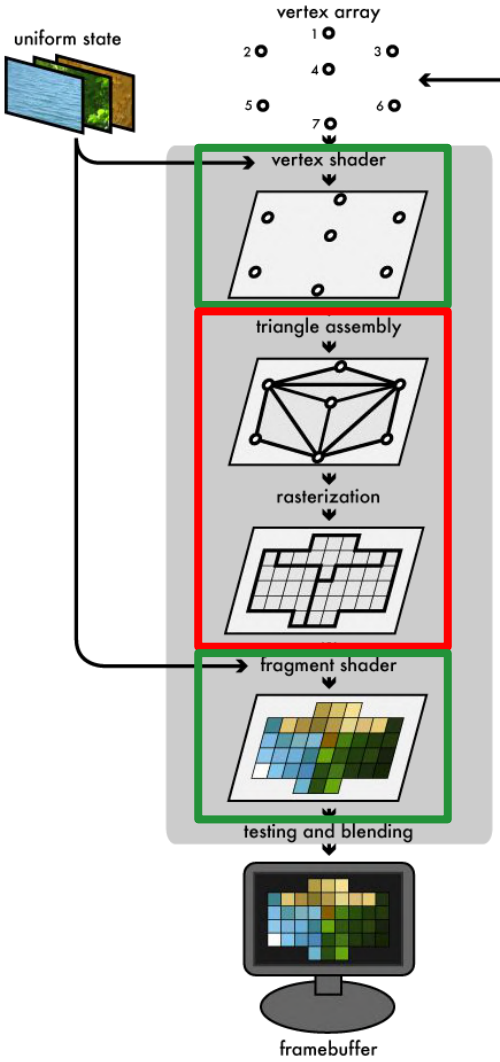
- Depth (32-bit float)
- RGB (3 x uchar)

???

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(???)



programmable vertex shader

fixed pipeline (hardware)

software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

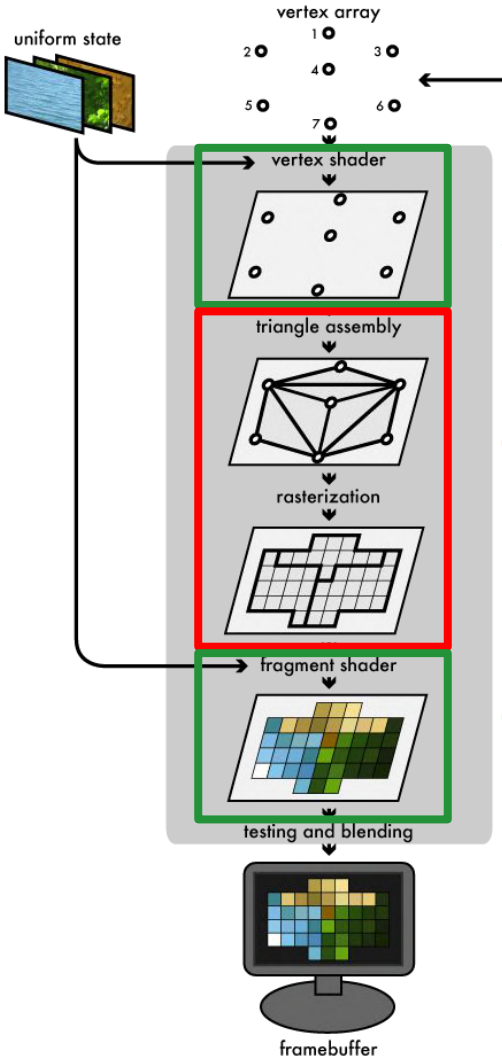
- Depth (32-bit float)
- RGB (3 x uchar)

???

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)



programmable vertex shader

fixed pipeline (hardware)

software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

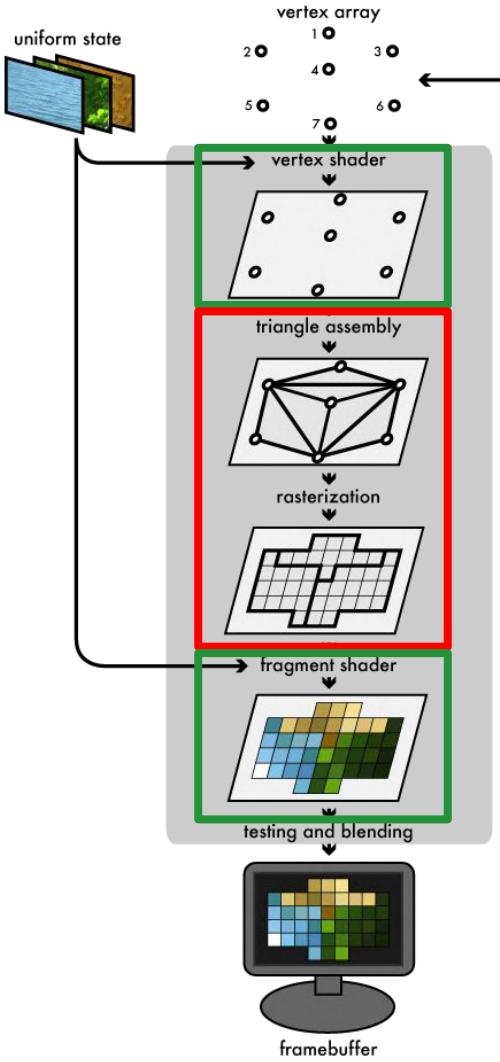
- Depth (32-bit float)
- RGB (3 x uchar)

Какой минимум в каждый пиксель записать?

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)



programmable vertex shader

fixed pipeline (hardware)

software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)

Deffered rendering/shading

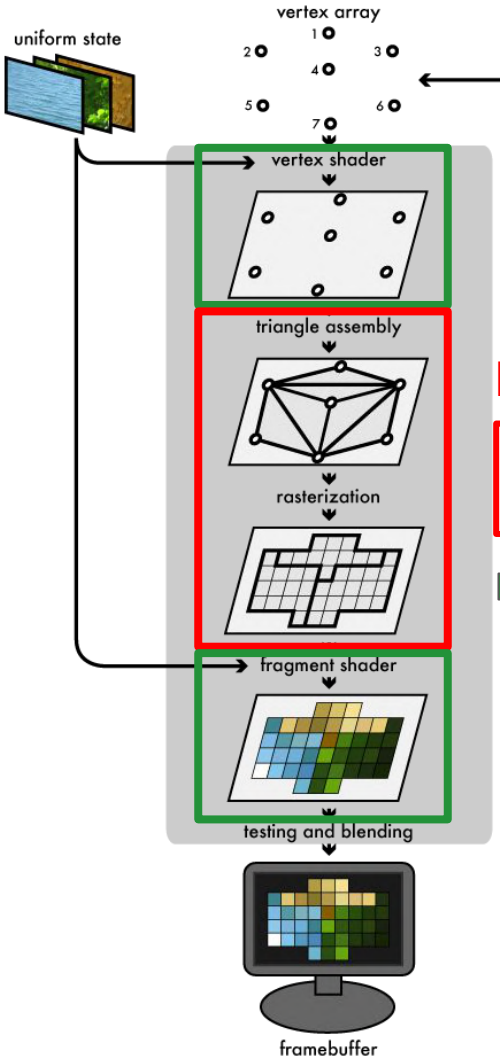
Geometry pass:

- depth
- face index

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)



programmable vertex shader

Как совместить?

fixed pipeline (hardware)

+ software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)

Deffered rendering/shading

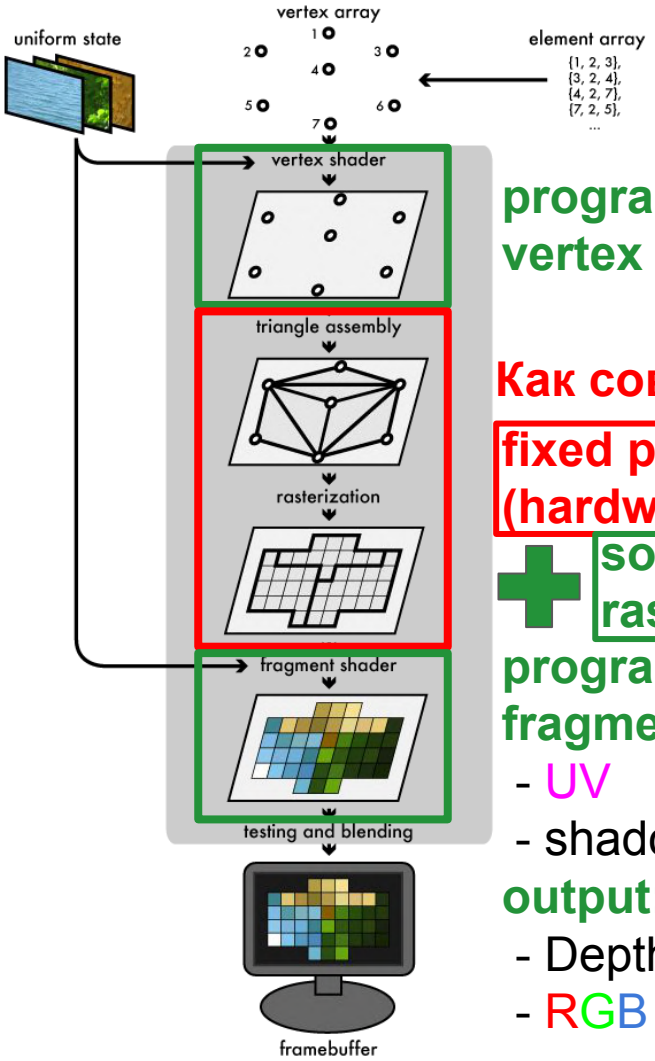
Geometry pass:

- depth
- face index

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)



Deffered rendering/shading

programmable vertex shader

Как совместить?

fixed pipeline (hardware)

\+ **software rasterizer**

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)

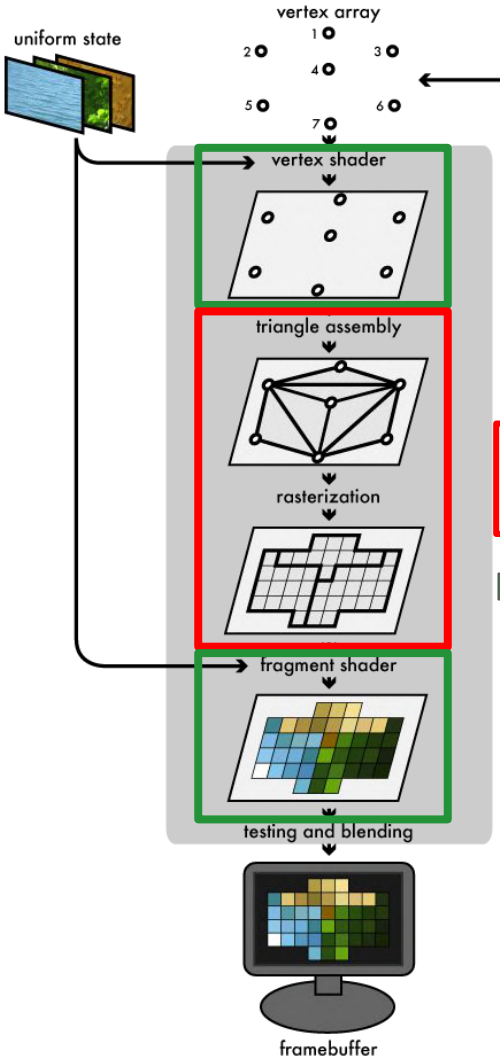
Geometry pass:

- depth
- face index

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)



programmable vertex shader

fixed pipeline (hardware)

+ software rasterizer

programmable fragment shader:
- UV
- shadows/lighting

output:
- Depth (32-bit float)
- RGB (3 x uchar)

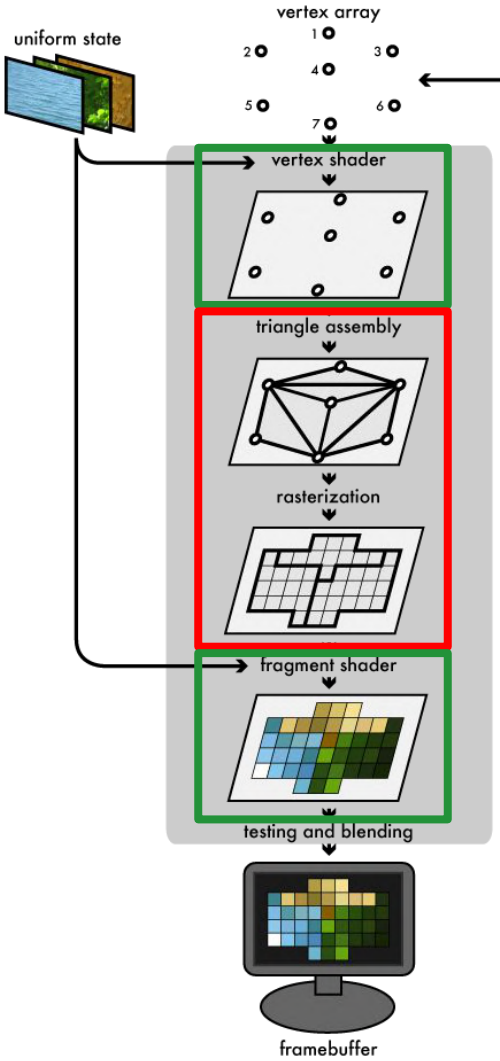
Deffered rendering/shading

Geometry pass:
- depth
- face index

O(???)

In-framebuffer material/light pass:
- RGB (3 x uchar)

O(screen size)



programmable vertex shader

fixed pipeline (hardware)

software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)

Deffered rendering/shading

Geometry pass:

- depth
- face index

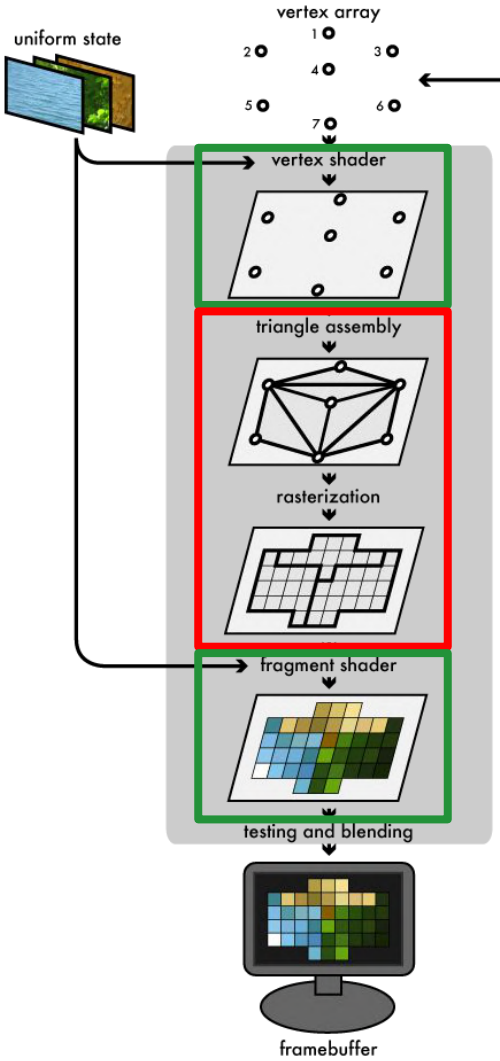
O(???) - init HZB

O(???) - rasterization

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)



programmable vertex shader

fixed pipeline (hardware)

software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)

Deffered rendering/shading

Geometry pass:

- depth
- face index

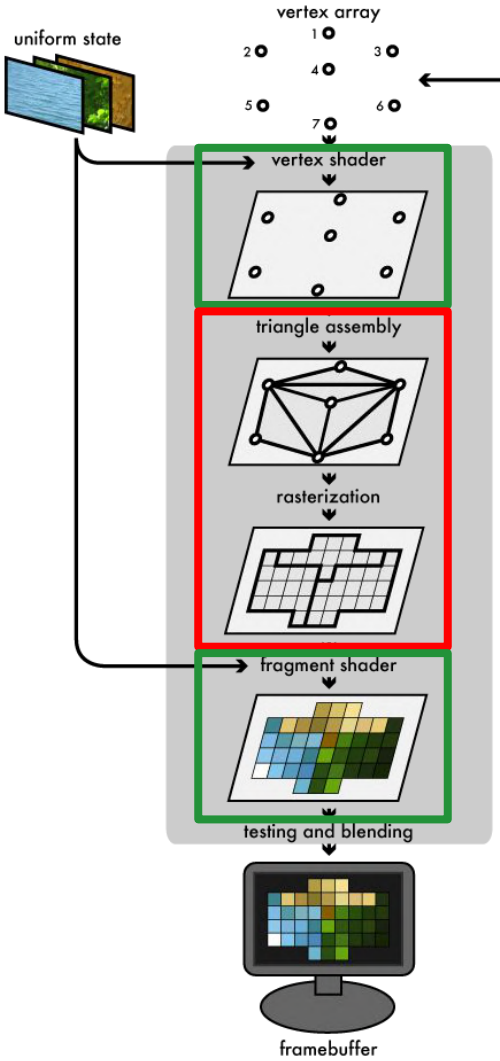
O(screen size) - init HZB

O(???) - rasterization

In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)



programmable vertex shader

fixed pipeline (hardware)

software rasterizer

programmable fragment shader:

- UV
- shadows/lighting

output:

- Depth (32-bit float)
- RGB (3 x uchar)

Deffered rendering/shading

Geometry pass:

- depth
- face index

O(screen size) - init HZB

O(cluster count) - rasterization

В чем был секрет виртуальных текстур?

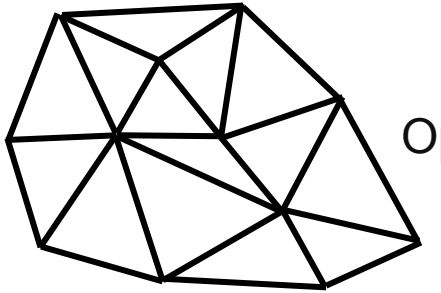
In-framebuffer material/light pass:

- RGB (3 x uchar)

O(screen size)

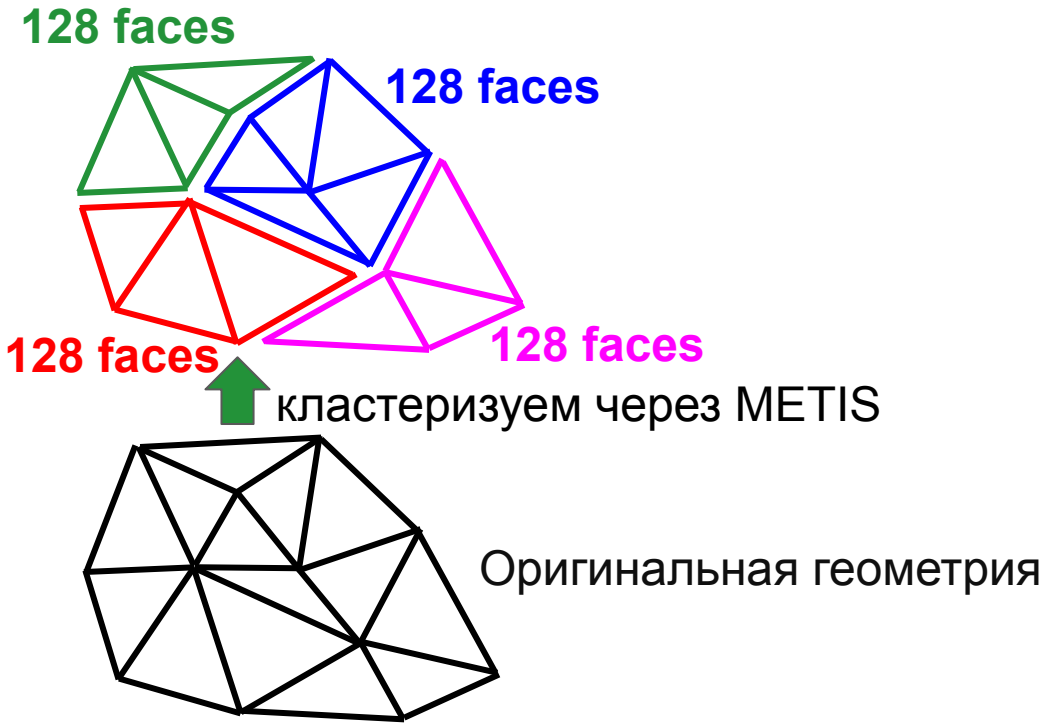
Geometry clusters hierarchy

кластеризуем через METIS

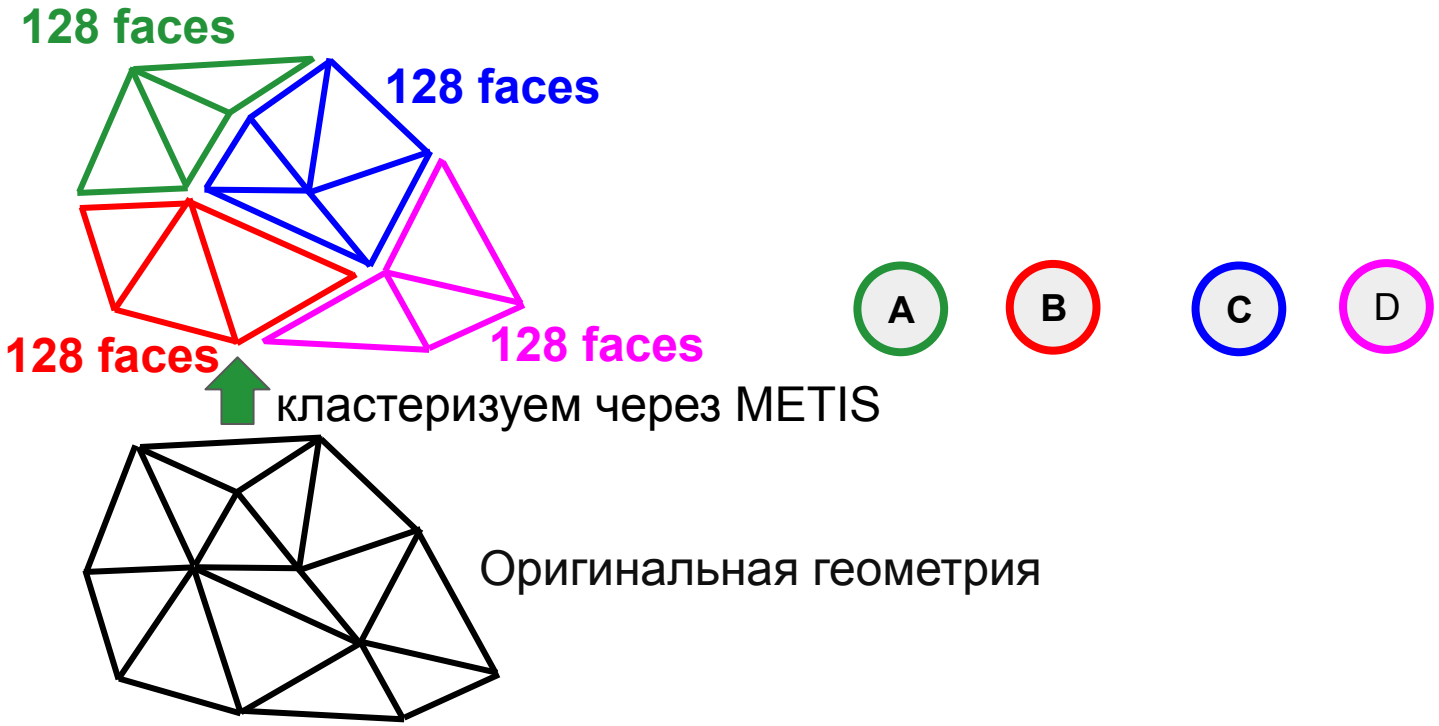


Оригинальная геометрия

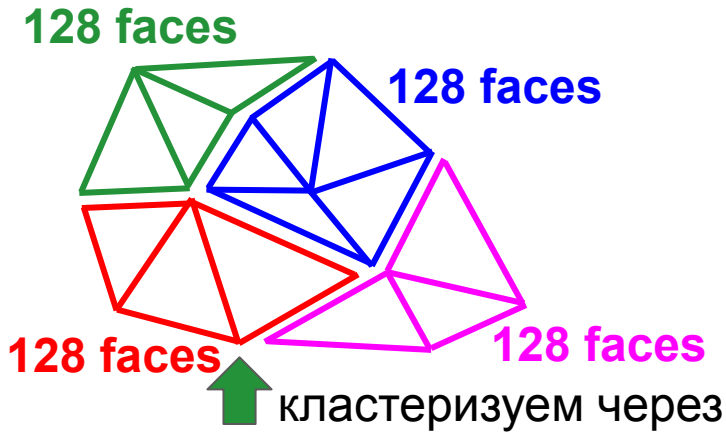
Geometry clusters hierarchy



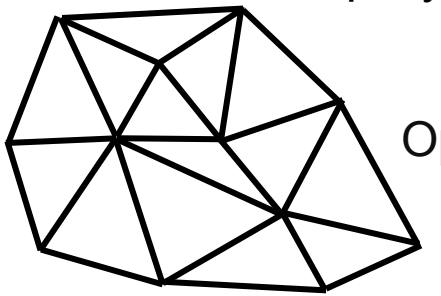
Geometry clusters hierarchy



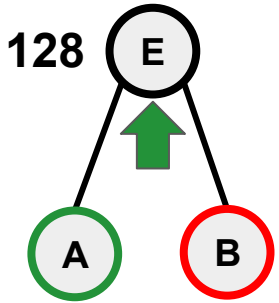
Geometry clusters hierarchy

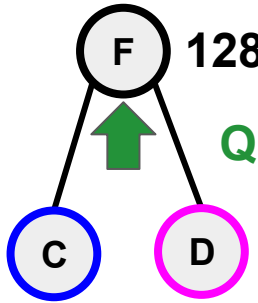


кластеризуем через METIS



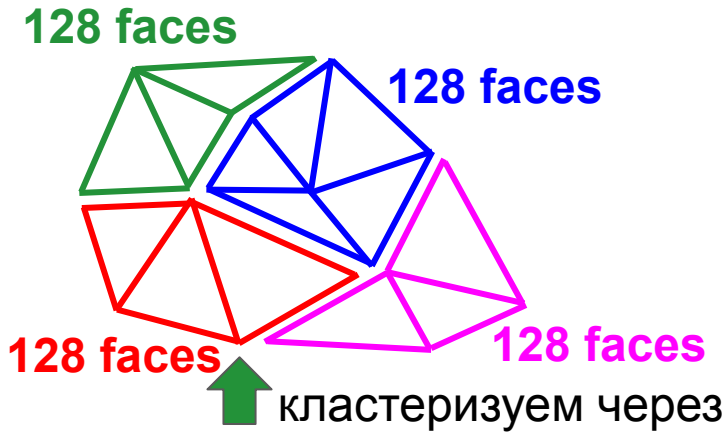
Оригинальная геометрия



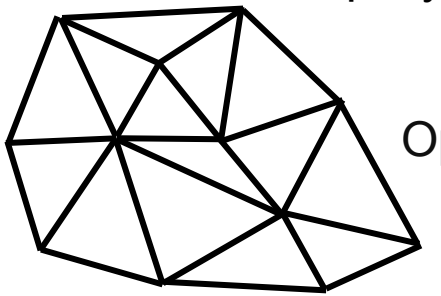


QSlim decimation

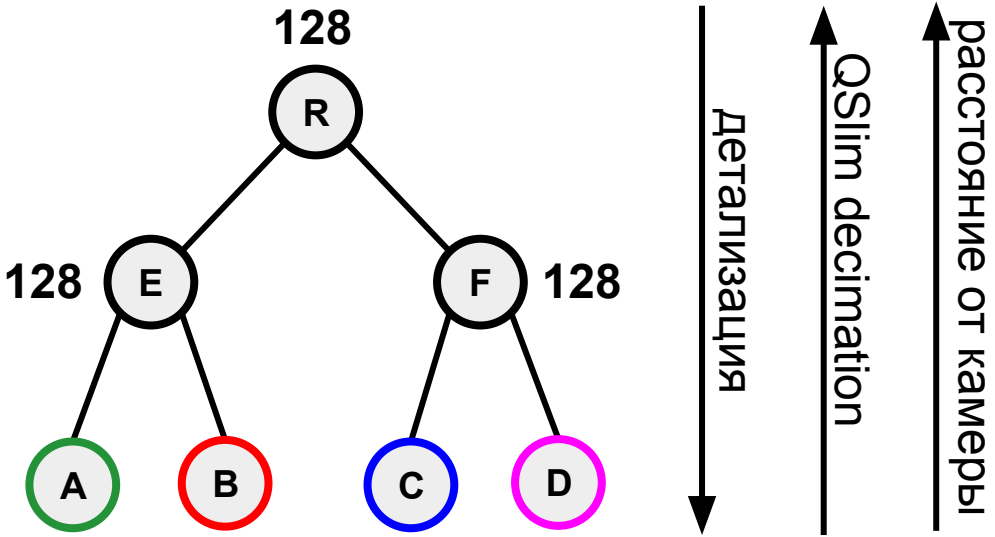
Geometry clusters hierarchy



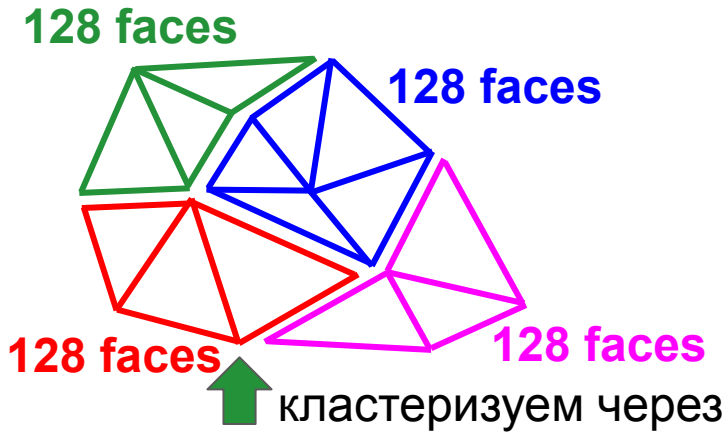
кластеризуем через METIS



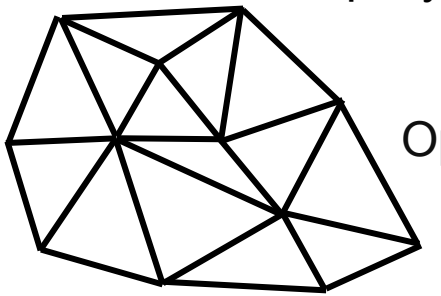
Оригинальная геометрия



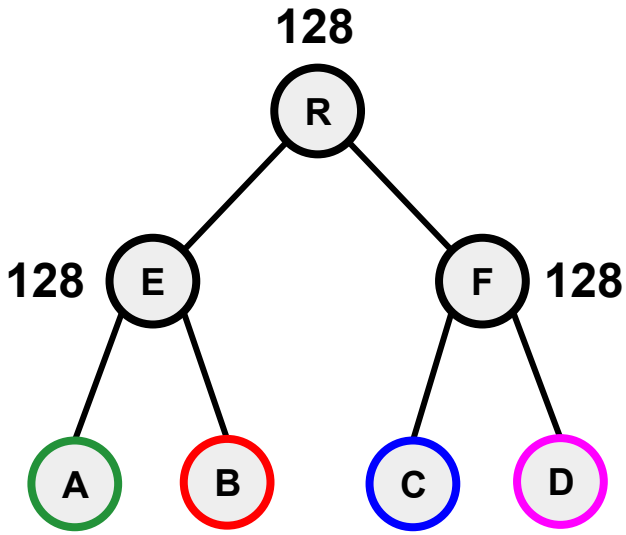
Geometry clusters hierarchy



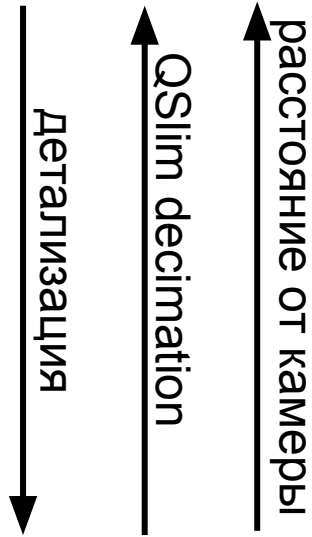
кластеризуем через METIS



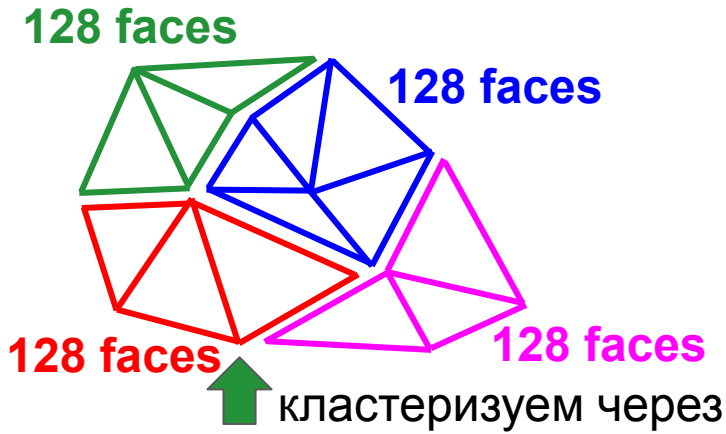
Оригинальная геометрия



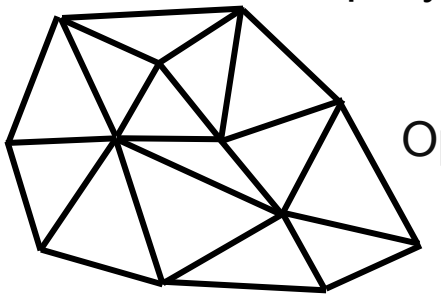
Какие узлы рисуем?



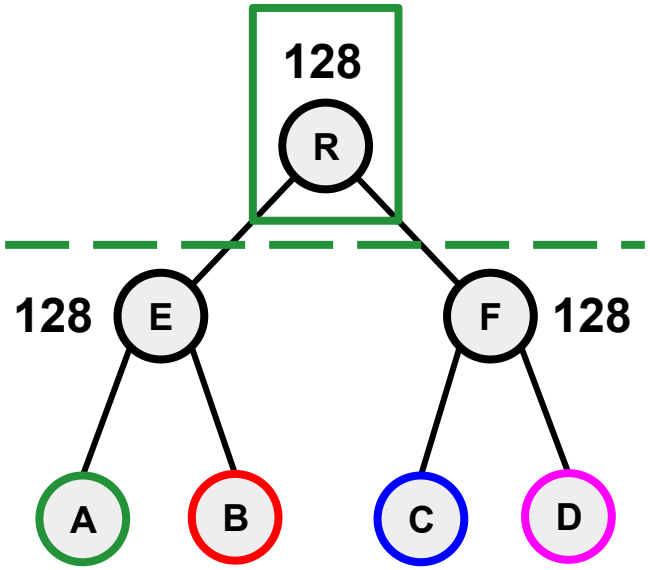
Geometry clusters hierarchy



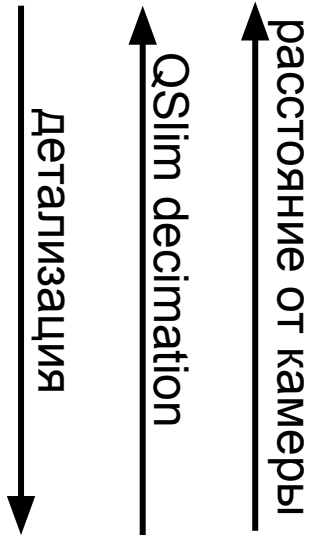
кластеризуем через METIS



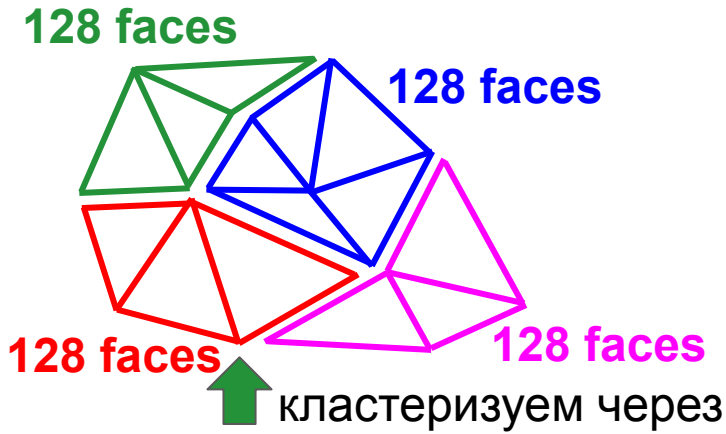
Оригинальная геометрия



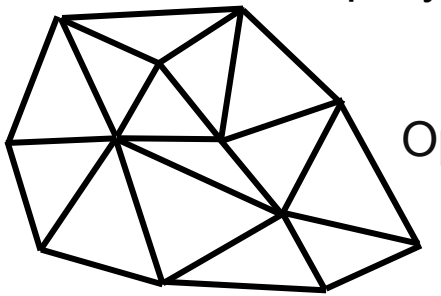
Какие узлы рисуем?



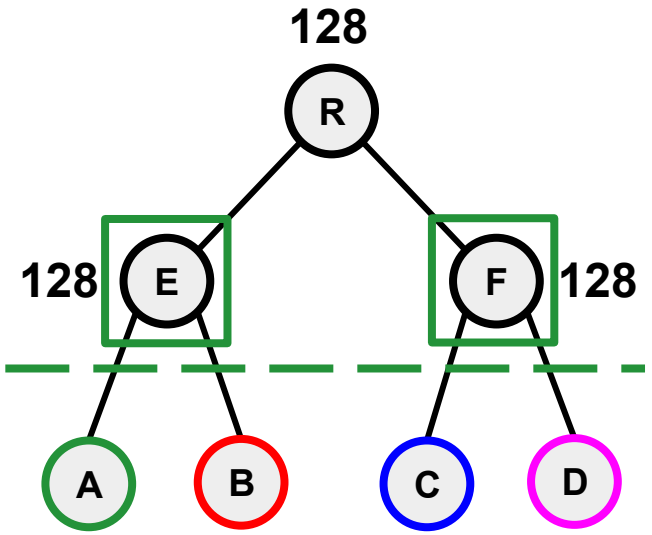
Geometry clusters hierarchy



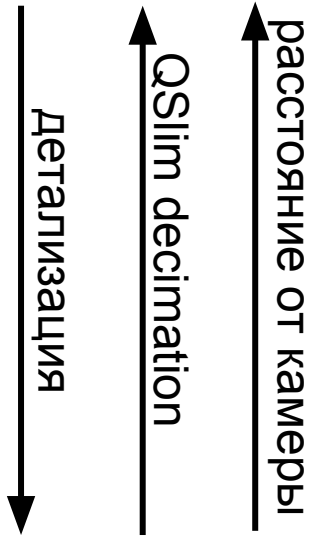
кластеризуем через METIS



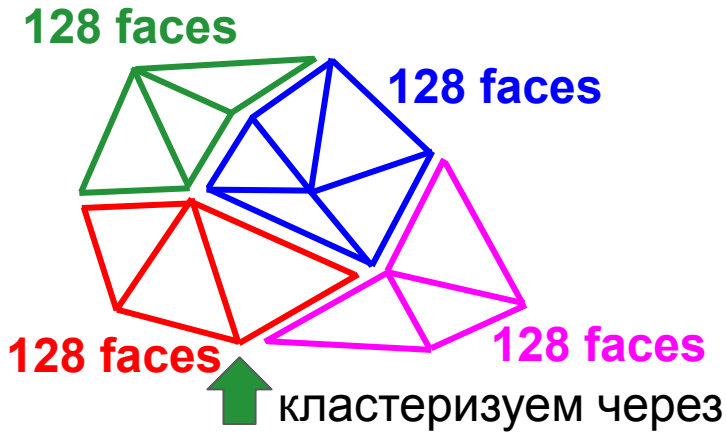
Оригинальная геометрия



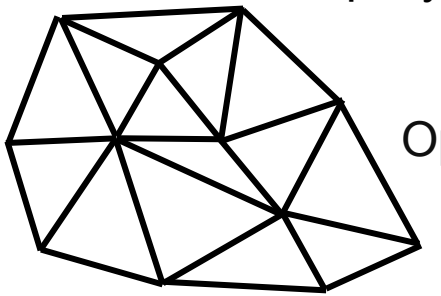
Какие узлы рисуем?



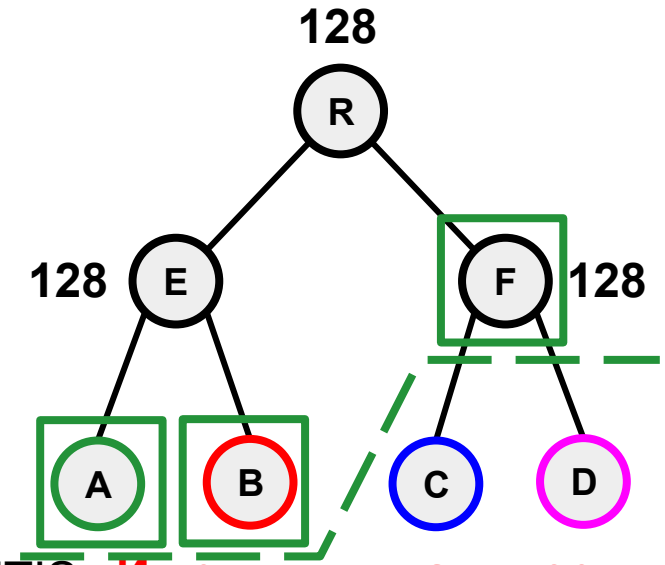
Geometry clusters hierarchy

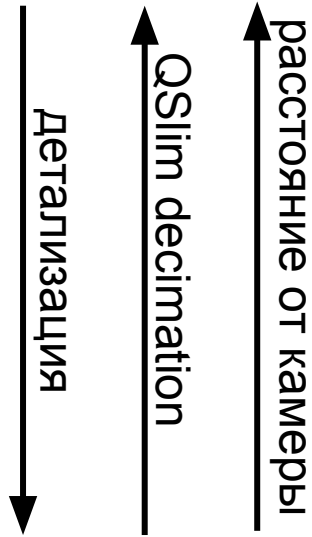


кластеризуем через METIS



Оригинальная геометрия



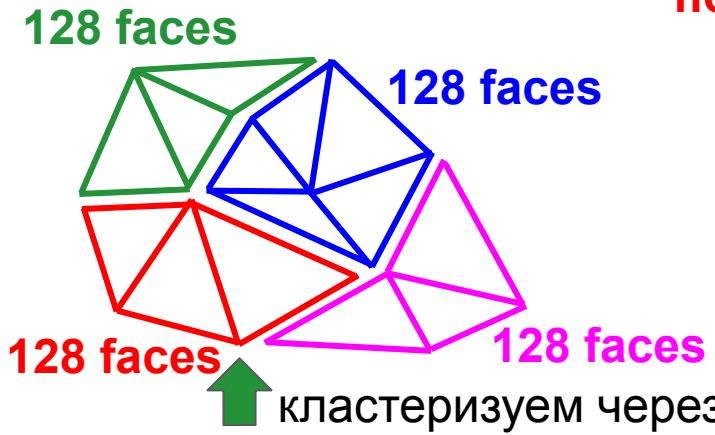


Ищем минимальное покрытие с учетом пиксельной проекции

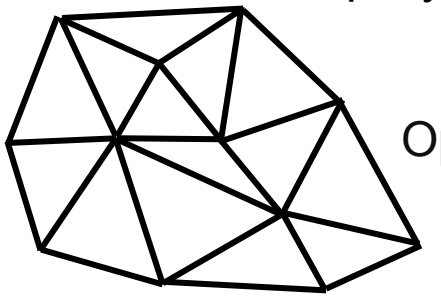
AABB

1 face ~ 1 pixel!

Geometry clusters hierarchy

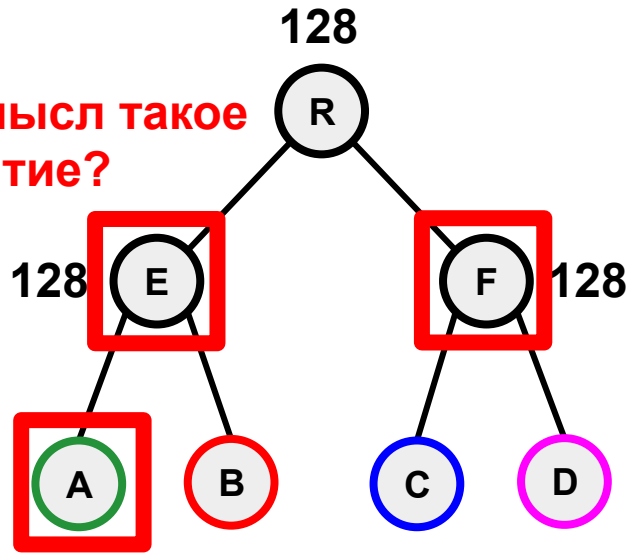


кластеризуем через METIS



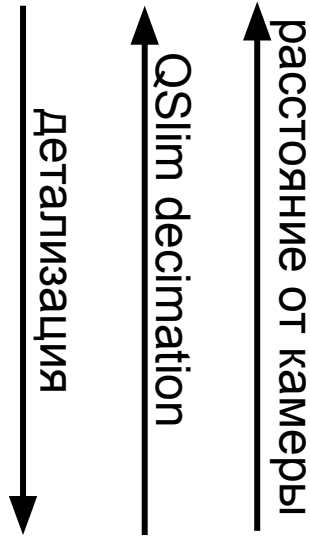
Оригинальная геометрия

Имеет ли смысл такое покрытие?

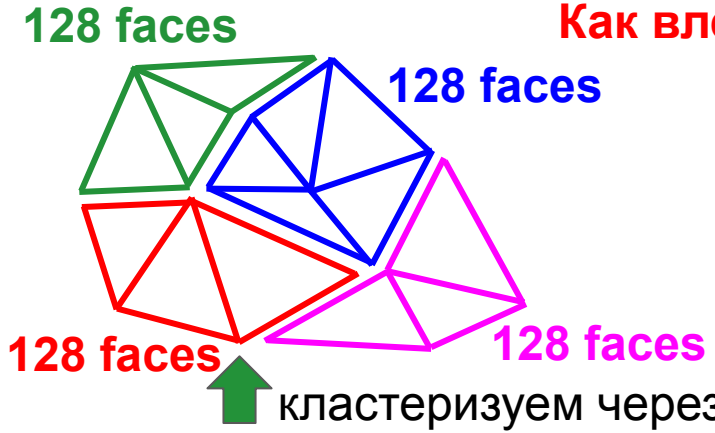


Ищем минимальное покрытие с учетом пиксельной проекции

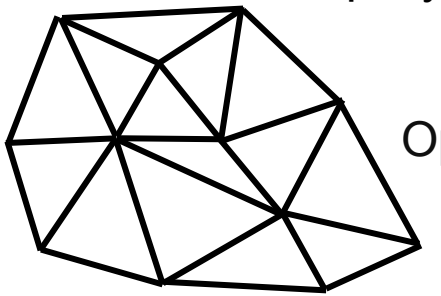
AABB **1 face ~ 1 pixel!**



Geometry clusters hierarchy

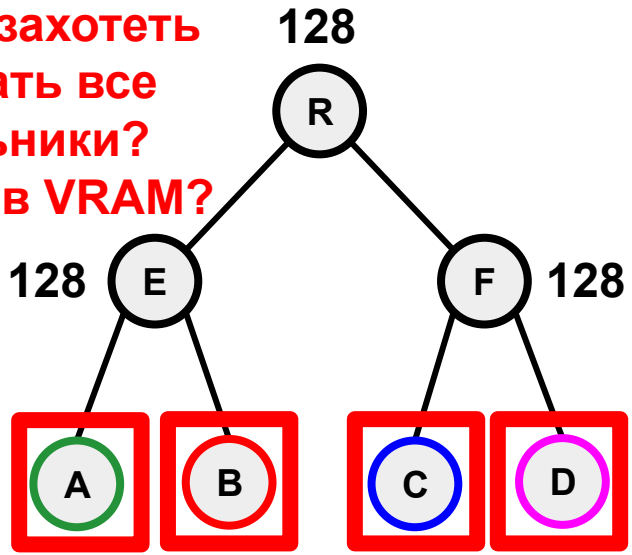


кластеризуем через METIS



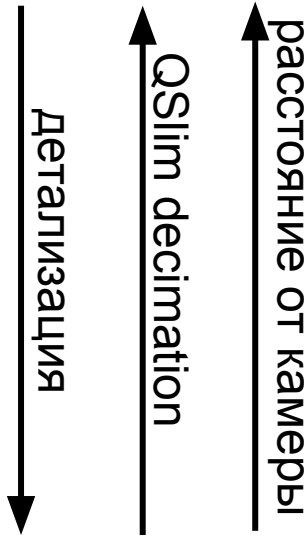
Оригинальная геометрия

Можем ли захотеть отрисовать все треугольники? Как влезть в VRAM?



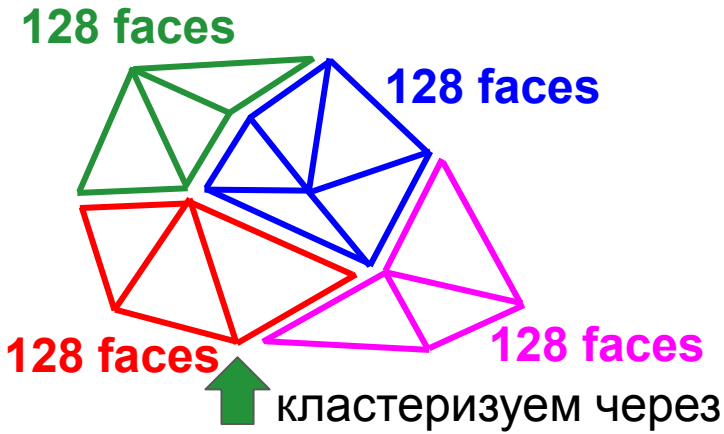
Ищем минимальное покрытие с учетом пиксельной проекции

AABB **1 face ~ 1 pixel!**

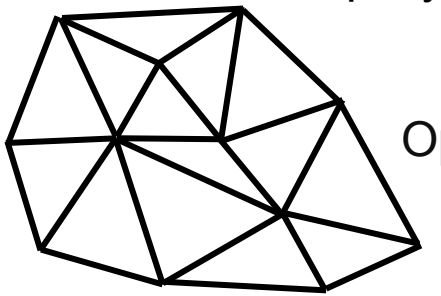


Geometry clusters hierarchy

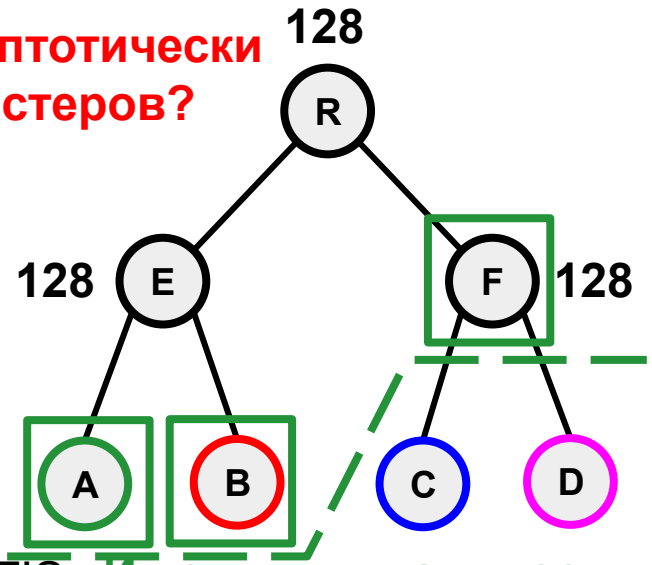
Сколько асимптотически рисуем кластеров?



кластеризуем через METIS

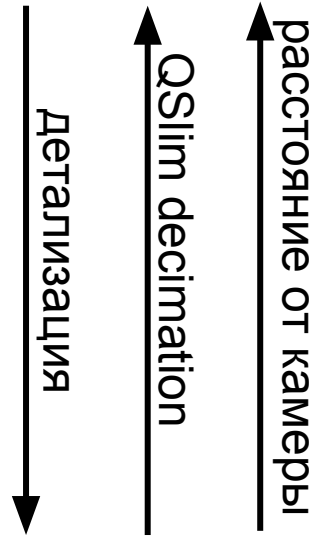


Оригинальная геометрия



Ищем минимальное покрытие с учетом пиксельной проекции

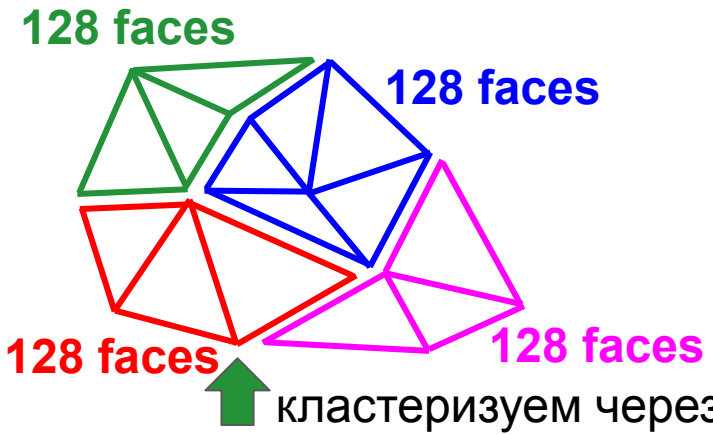
AABB **1 face ~ 1 pixel!**



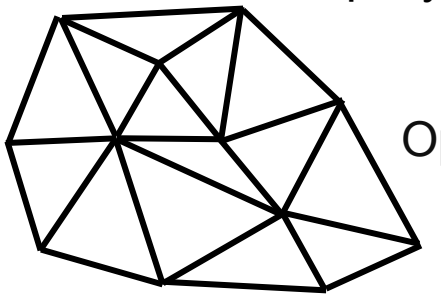
Geometry clusters hierarchy

Сколько асимптотически рисуем кластеров?

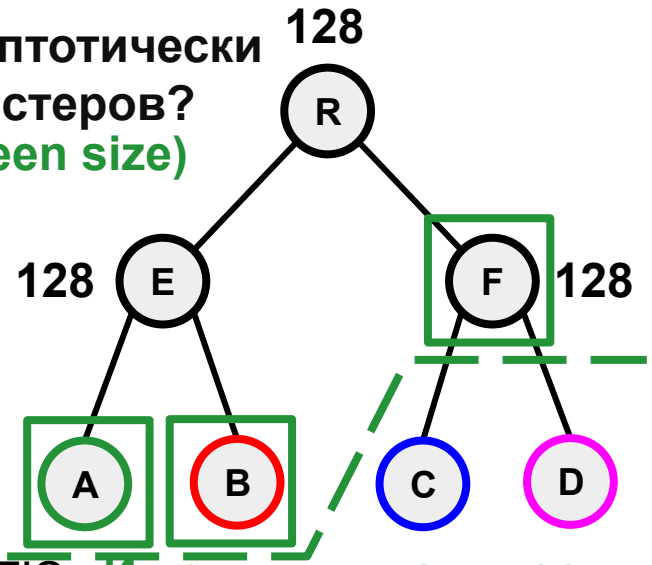
O(screen size)

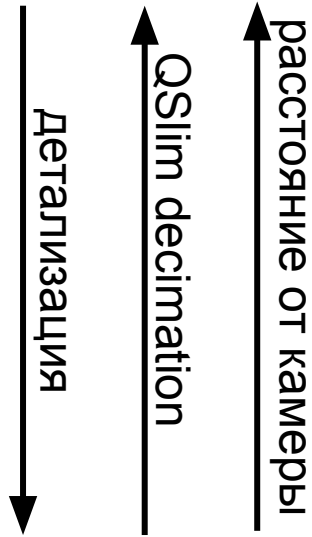


кластеризуем через METIS



Оригинальная геометрия

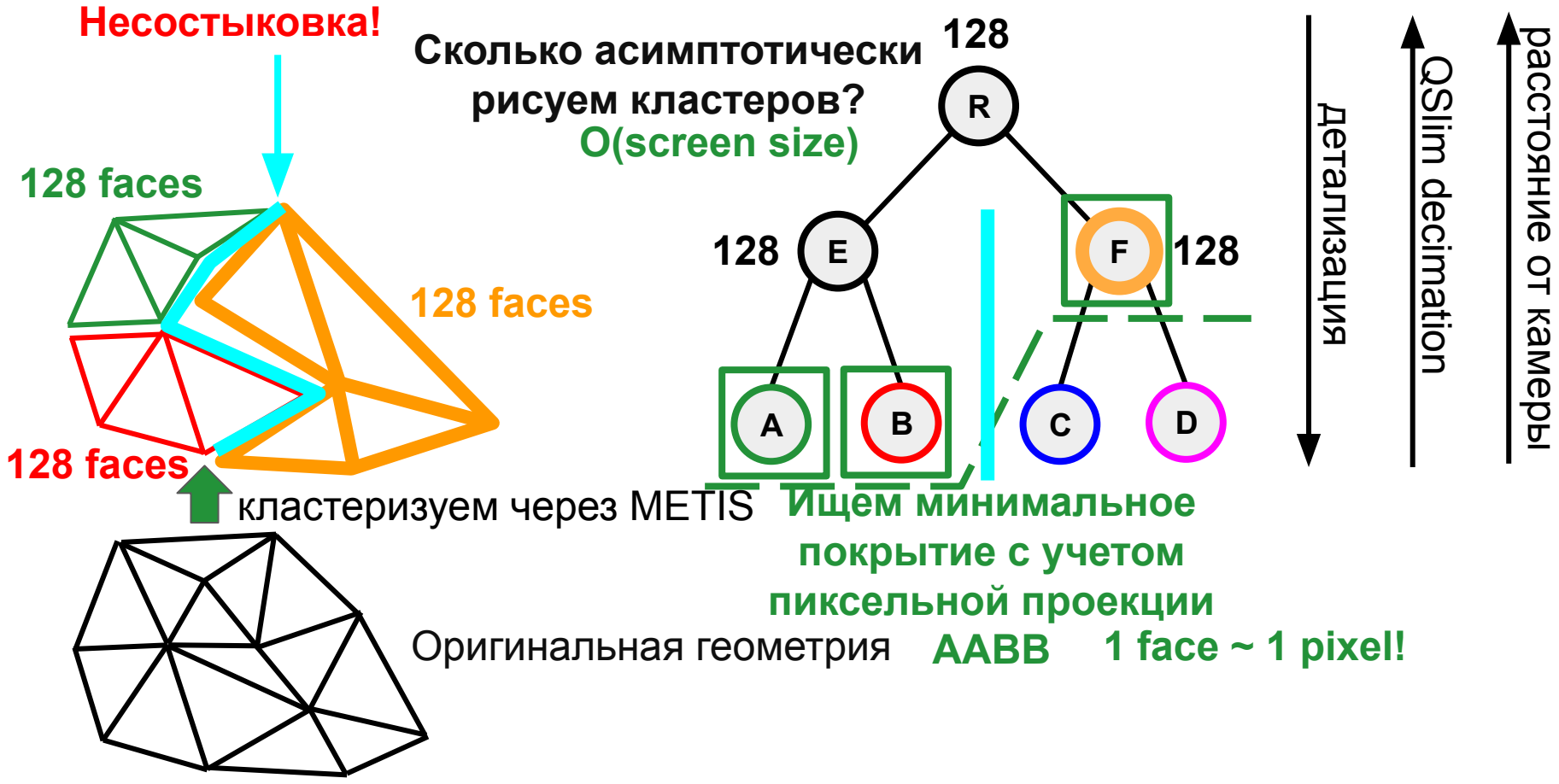




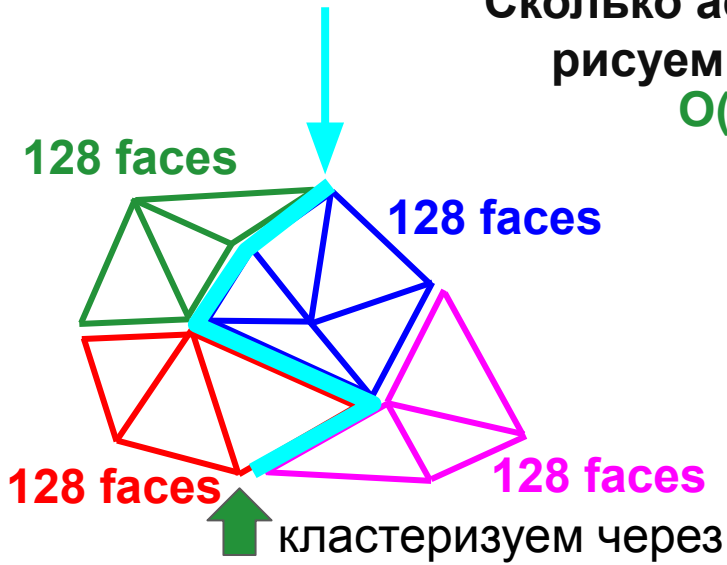
Ищем минимальное покрытие с учетом пиксельной проекции

AABB **1 face ~ 1 pixel!**

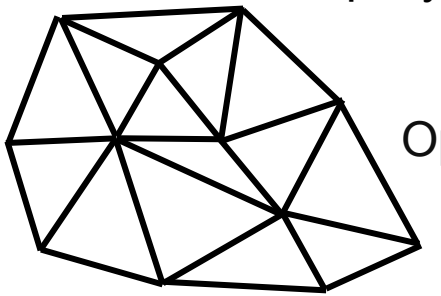
Geometry clusters hierarchy



Geometry clusters hierarchy



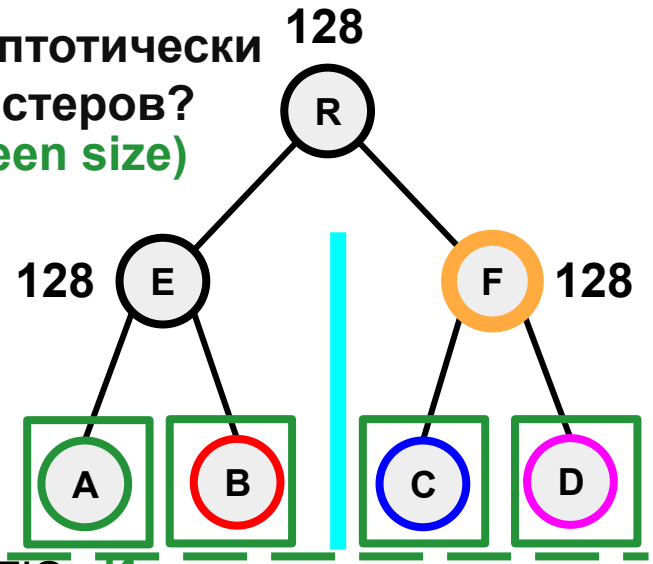
кластеризуем через METIS

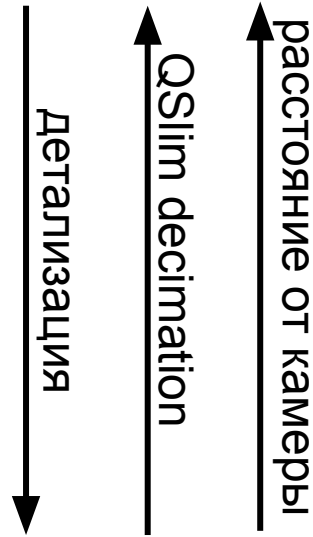


Оригинальная геометрия

Сколько асимптотически рисуем кластеров?

O(screen size)





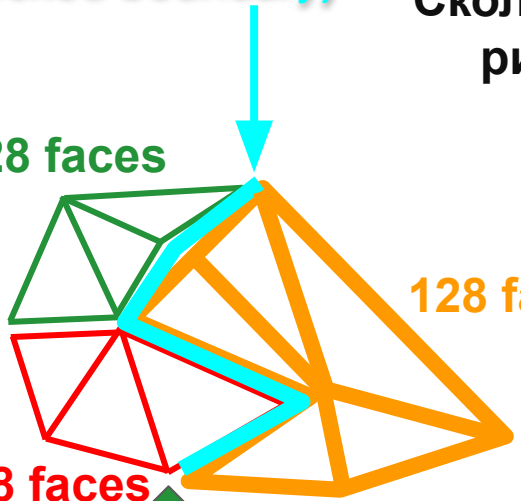
Ищем минимальное покрытие с учетом пиксельной проекции

AABB

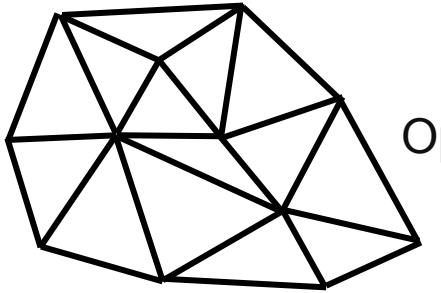
1 face ~ 1 pixel!

Geometry clusters hierarchy

Заморозка границ!
(Locked boundary)

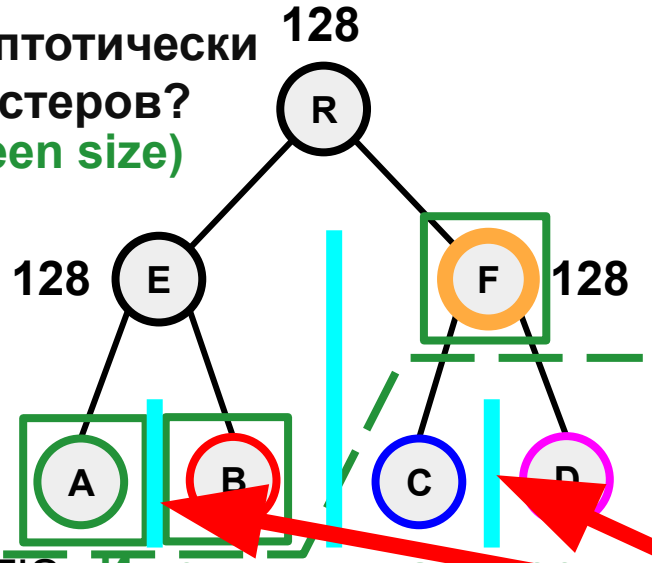


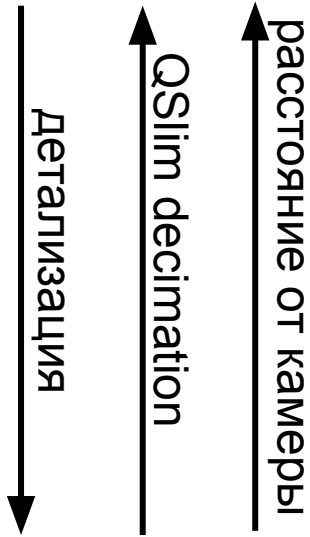
кластеризуем через METIS



Оригинальная геометрия

Сколько асимптотически рисуем кластеров?
O(screen size)





Ищем минимальное покрытие с учетом пиксельной проекции

AABB

1 face ~ 1 pixel!

Почему тут корка не так плоха?

Заморозка границ - много треугольников не упростить!
Dense Crust!

Geometry clusters hierarchy

Заморозка границ!
(Locked boundary)

Сколько
рисуе

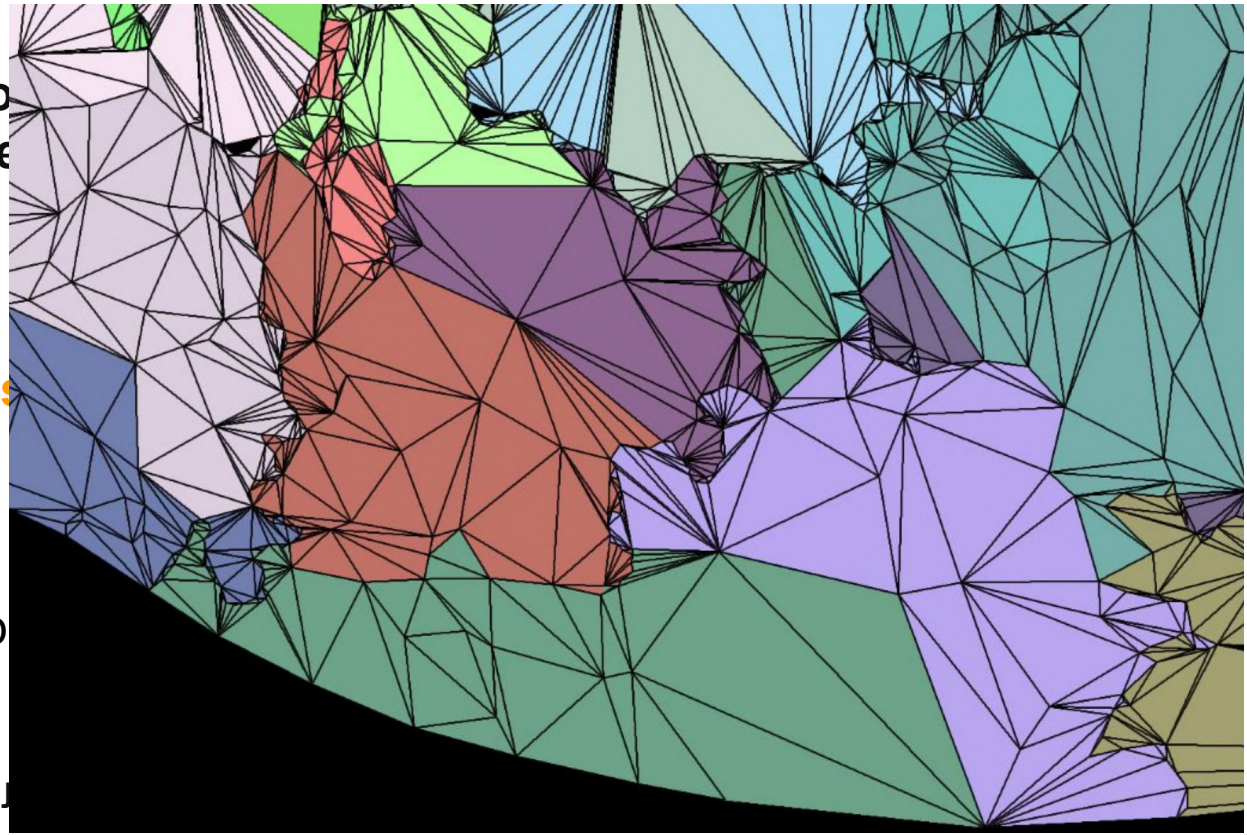
128 faces

128 faces

128 faces

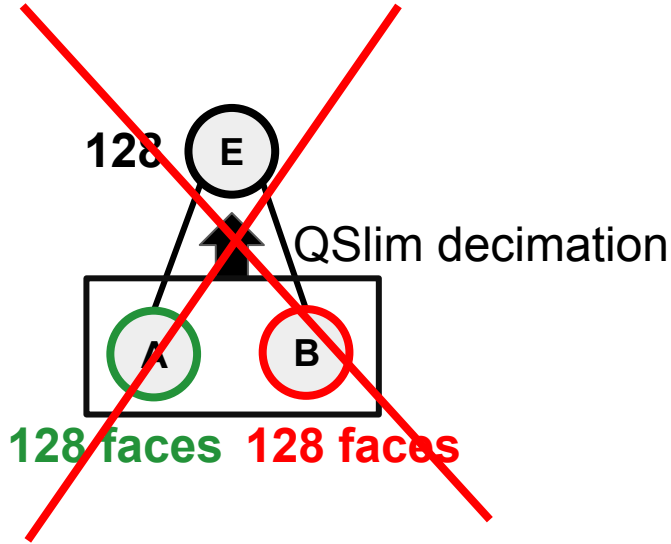
кластеризуем чер

Оригина

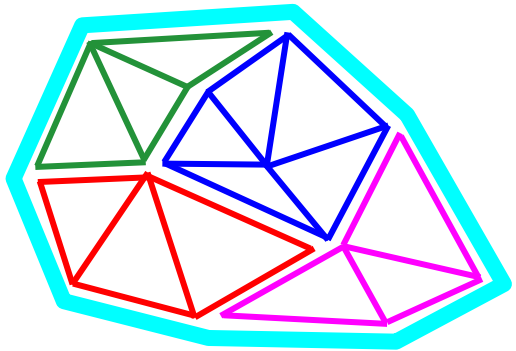


Заморозка границ - много треугольников не упростить!
Dense Crust!

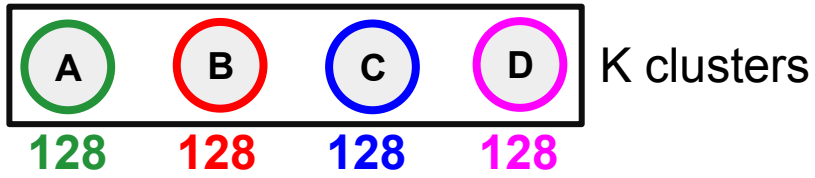
Geometry clusters hierarchy: DAG



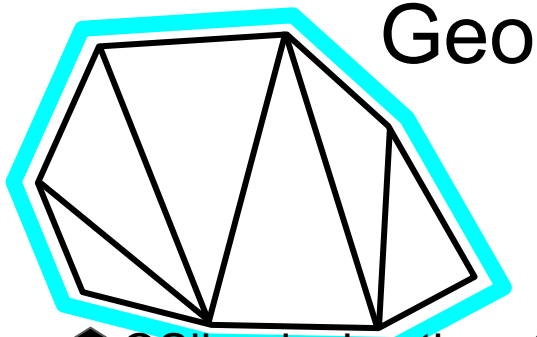
Geometry clusters hierarchy: DAG



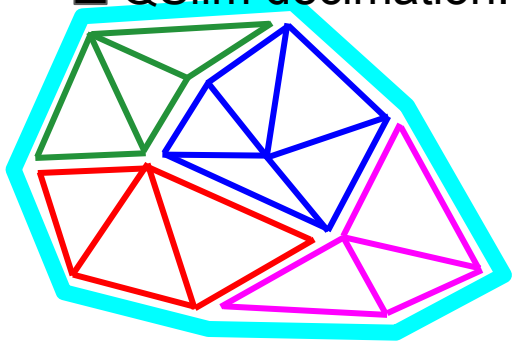
K=4 clusters



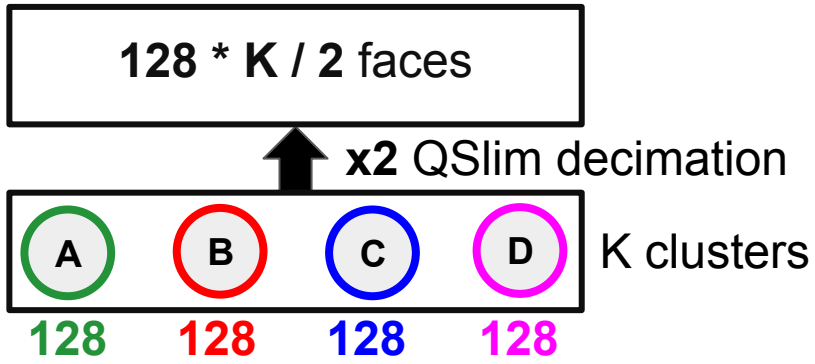
Geometry clusters hierarchy: DAG



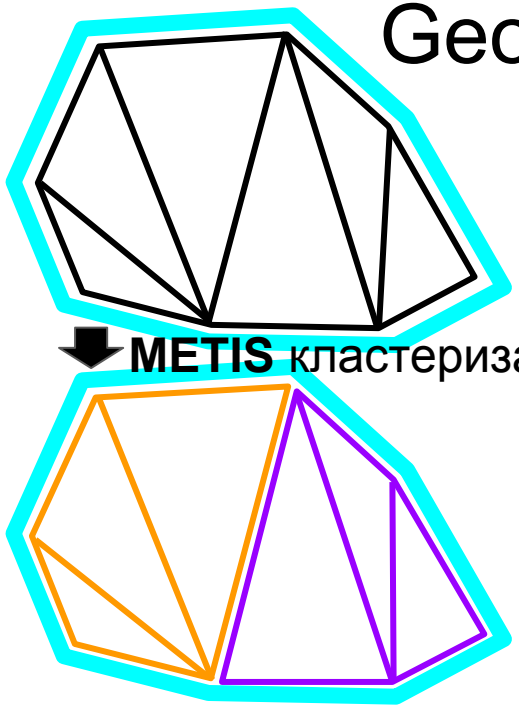
QSlim decimation: 14 -> 7 faces



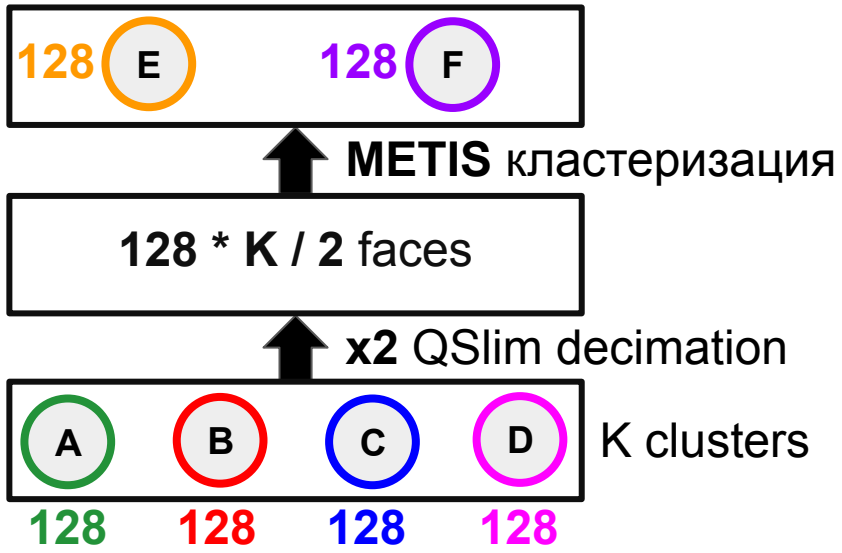
K=4 clusters



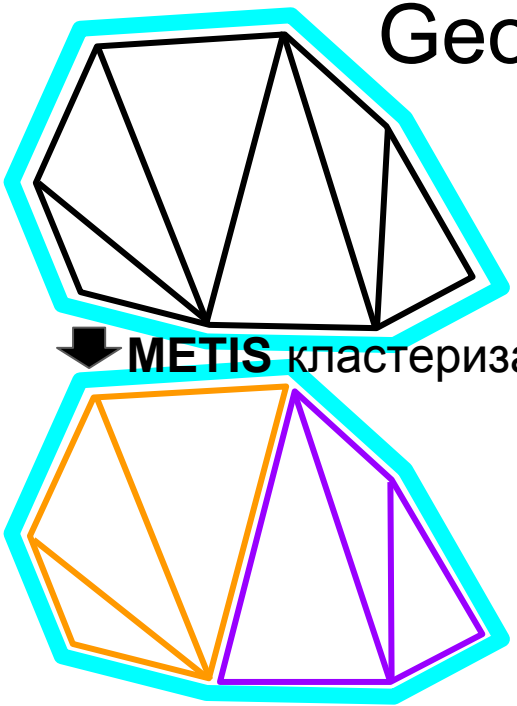
Geometry clusters hierarchy: DAG

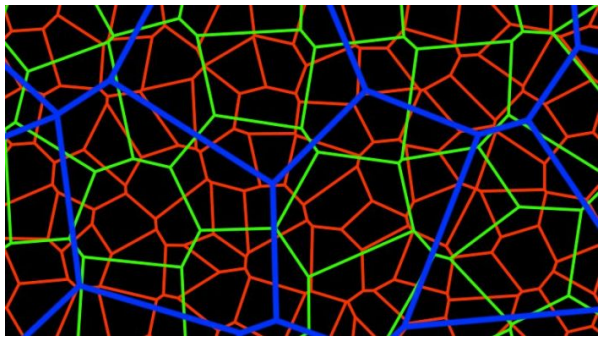


METIS кластеризация

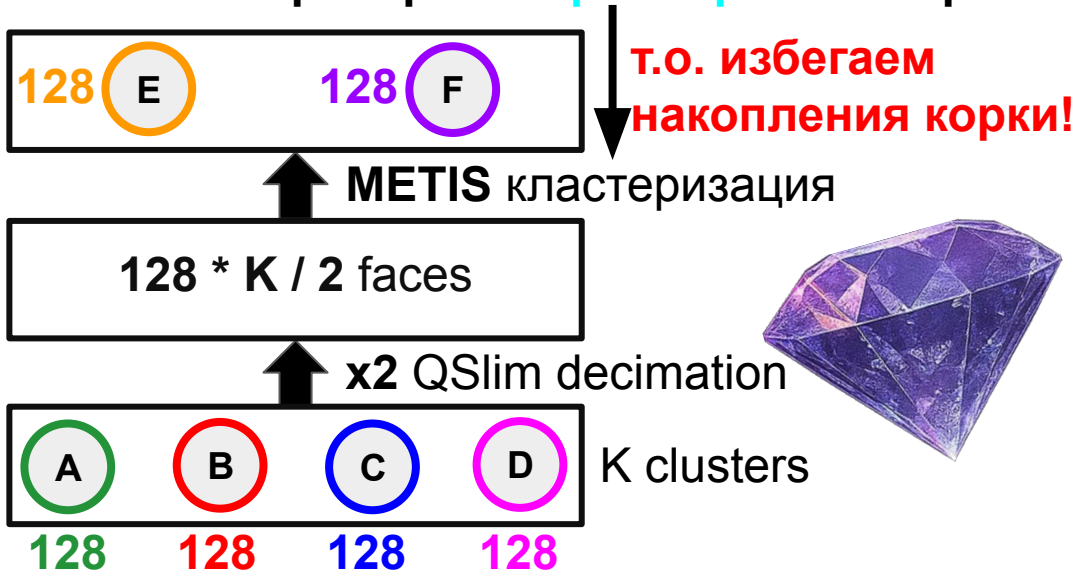


Geometry clusters hierarchy: DAG



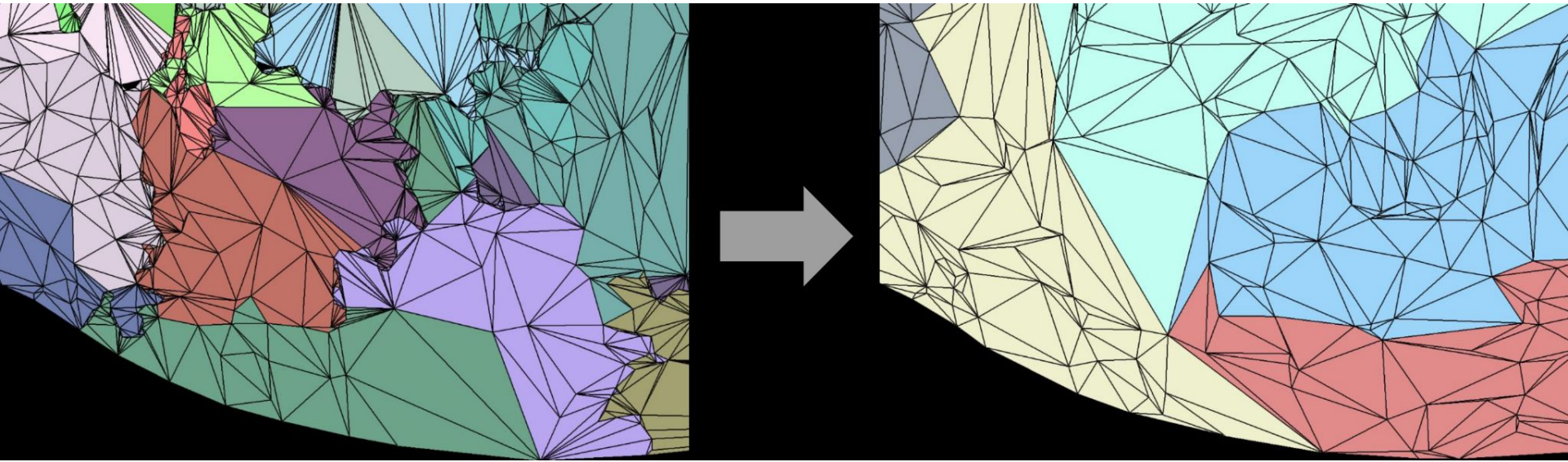


Кластеризируем минимизируя число ребер на периметре кластера

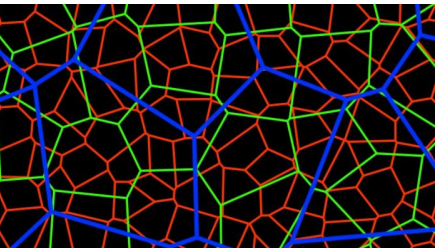


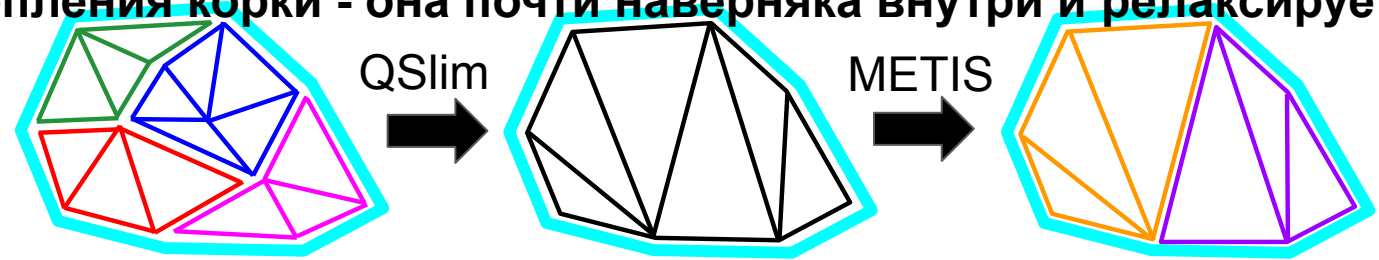
Geometry clusters hierarchy: DAG

Кластеризируем минимизируя число ребер на периметре кластера



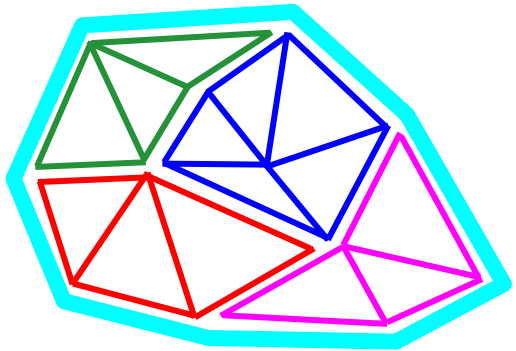
т.о. избегаем накопления корки - она почти наверняка внутри и релаксируется

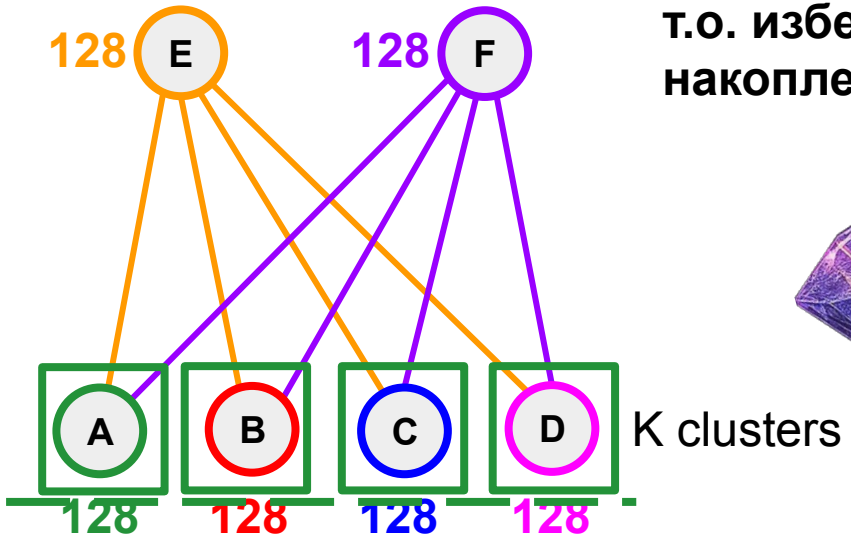




Geometry clusters hierarchy: DAG

Кластеризируем минимизируя число ребер на периметре кластера

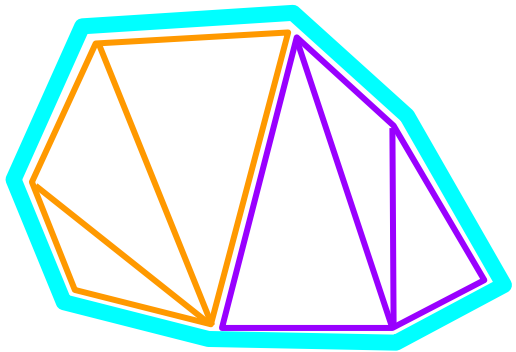




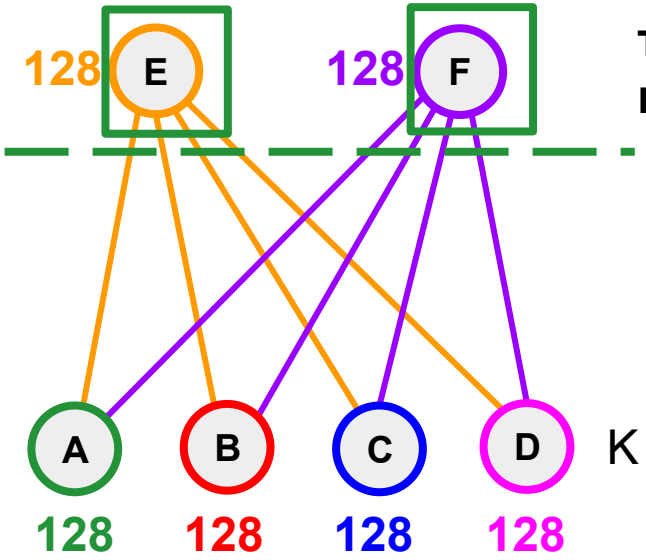
т.о. избегаем накопления корки!

Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG



Кластеризируем минимизируя число ребер на периметре кластера

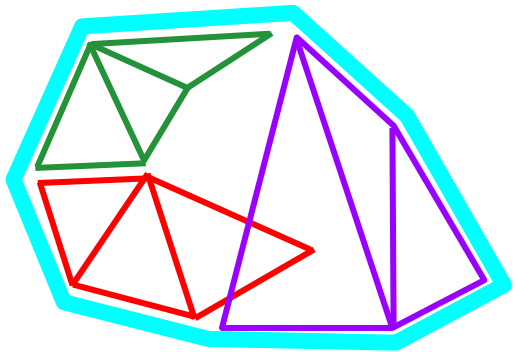


т.о. избегаем накопления корки!

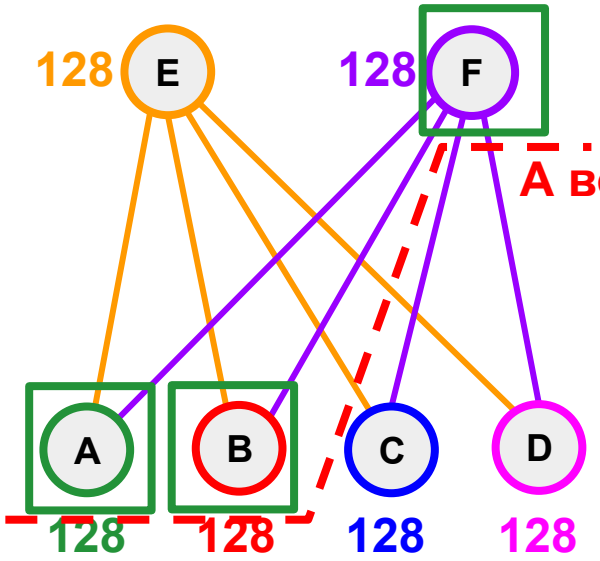
K clusters

Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG



Кластеризируем минимизируя число ребер на периметре кластера



т.о. избегаем накопления корки!

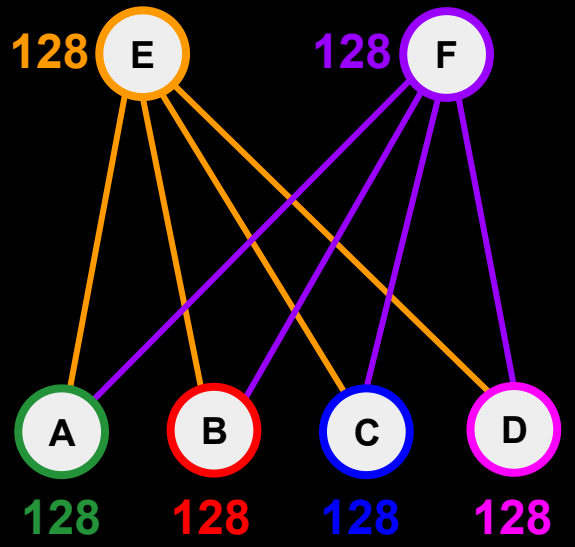
А вот так можно?

K clusters

Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

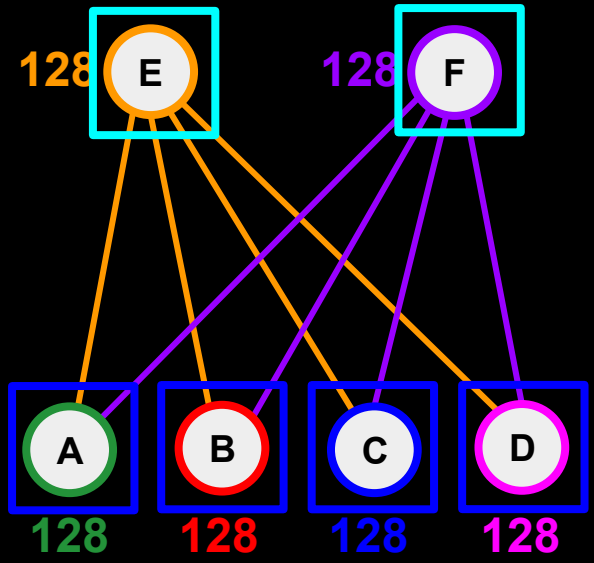
- Two submeshes with same boundary, but different LOD
- Choose between them based on screen-space error
  - Error calculated by simplifier projected to screen
  - Corrected for distance and angle distortion at worst-case point in sphere bounds
- All clusters in group must make same LOD decision
  - How? Communicate? No!
  - Same input => same output



Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

- Two submeshes with same boundary, but different LOD
- Choose between them based on screen-space error
  - Error calculated by simplifier projected to screen
  - Corrected for distance and angle distortion at worst-case point in sphere bounds
- All clusters in group must make same LOD decision
  - How? Communicate? No!
  - Same input => same output

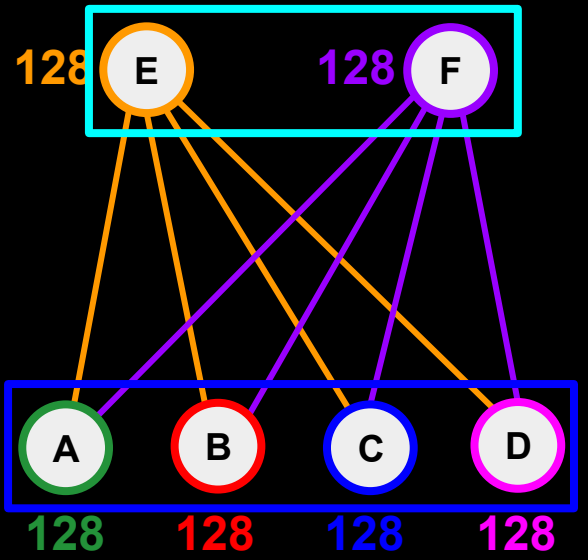


Оцениваем проекцию AABB каждого кластера?

Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

- Two submeshes with same boundary, but different LOD
- Choose between them based on screen-space error
  - Error calculated by simplifier projected to screen
  - Corrected for distance and angle distortion at worst-case point in sphere bounds
- All clusters in group must make same LOD decision
  - How? Communicate? No!
  - Same input => same output

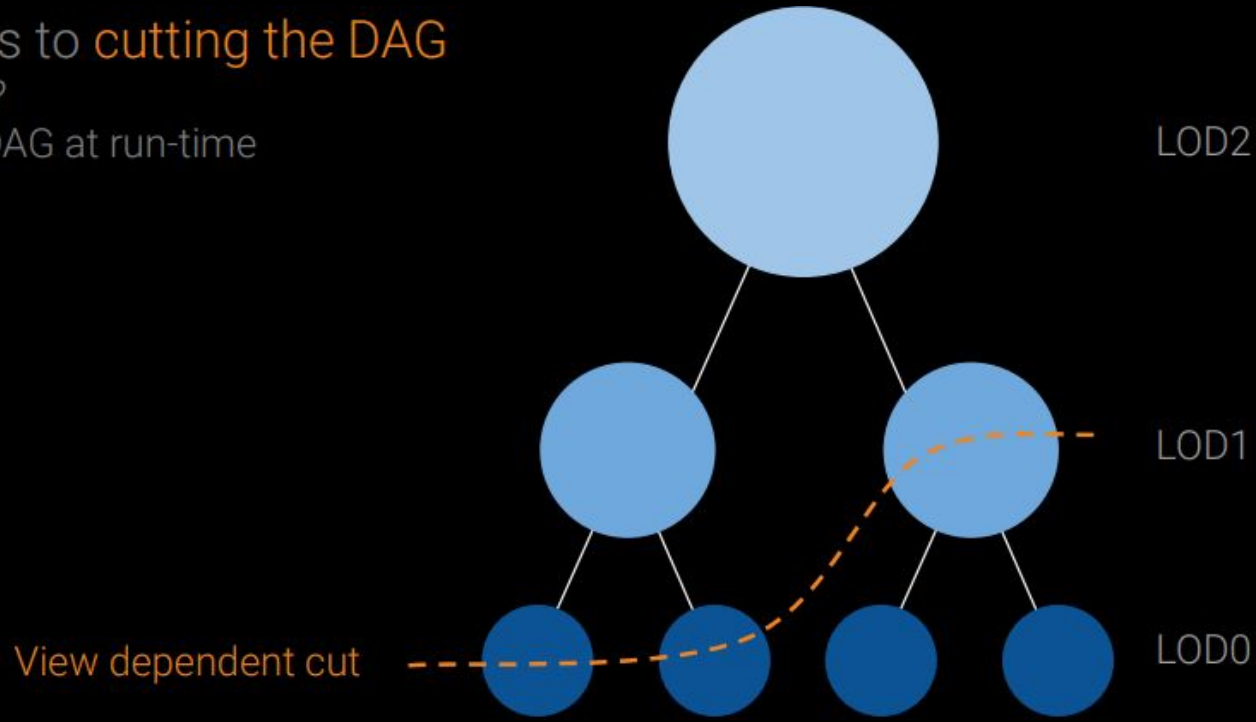


Оцениваем проекцию AABB не по кластеру,
а по нашей **группе K-кластеров**!

Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

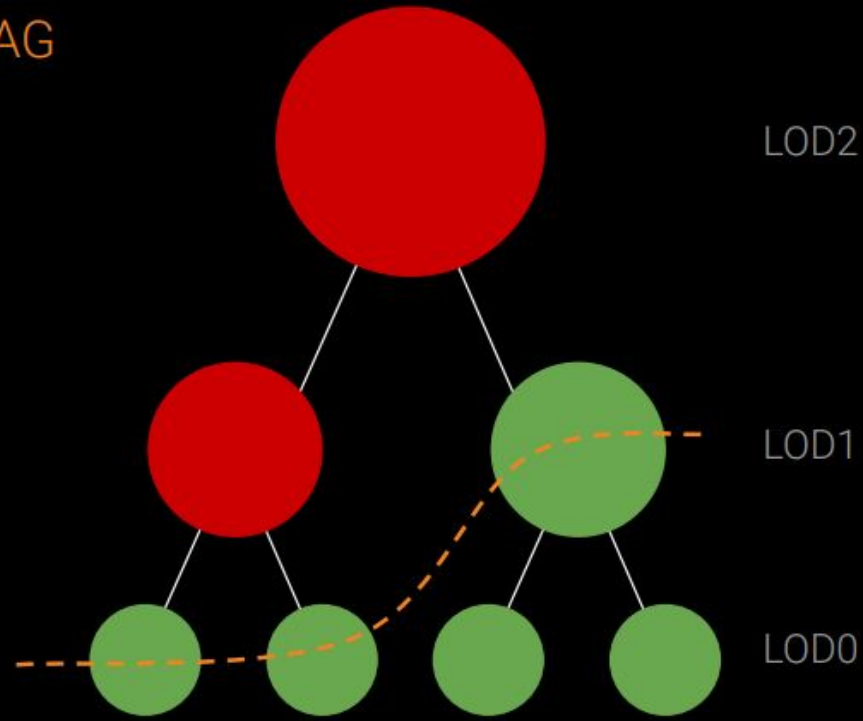
- LOD selection corresponds to cutting the DAG
  - How to compute in parallel?
  - Don't want to traverse the DAG at run-time
- What defines the cut?



Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

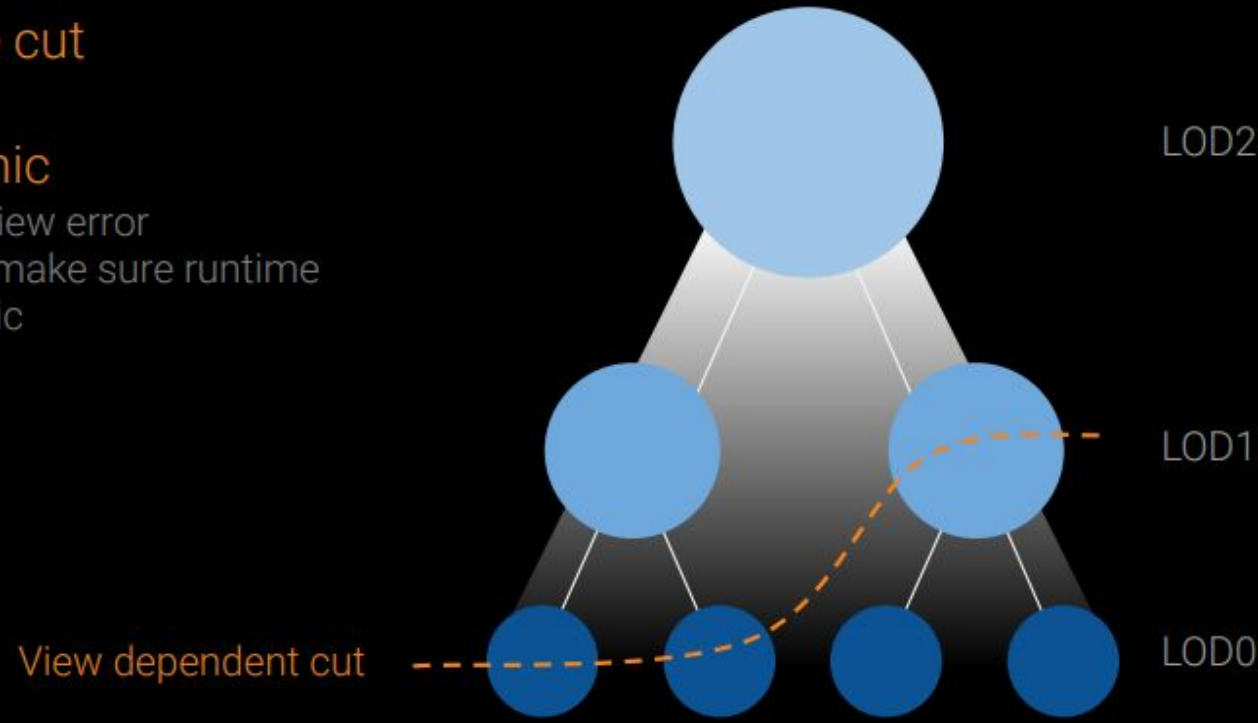
- LOD selection corresponds to cutting the DAG
  - How to compute in parallel?
  - Don't want to traverse the DAG at run-time
- What defines the cut?
  - Difference between parent and child
- Draw a cluster when:
  - Parent error is too high &&
  - Our error is small enough
  - Can be evaluated in parallel!



Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

- Only if there is one unique cut
- Force error to be monotonic
  - Parent view error >= child view error
  - Careful implementation to make sure runtime correction is also monotonic



Какое подмножество кластеров рисовать?

What *is* Nanite?

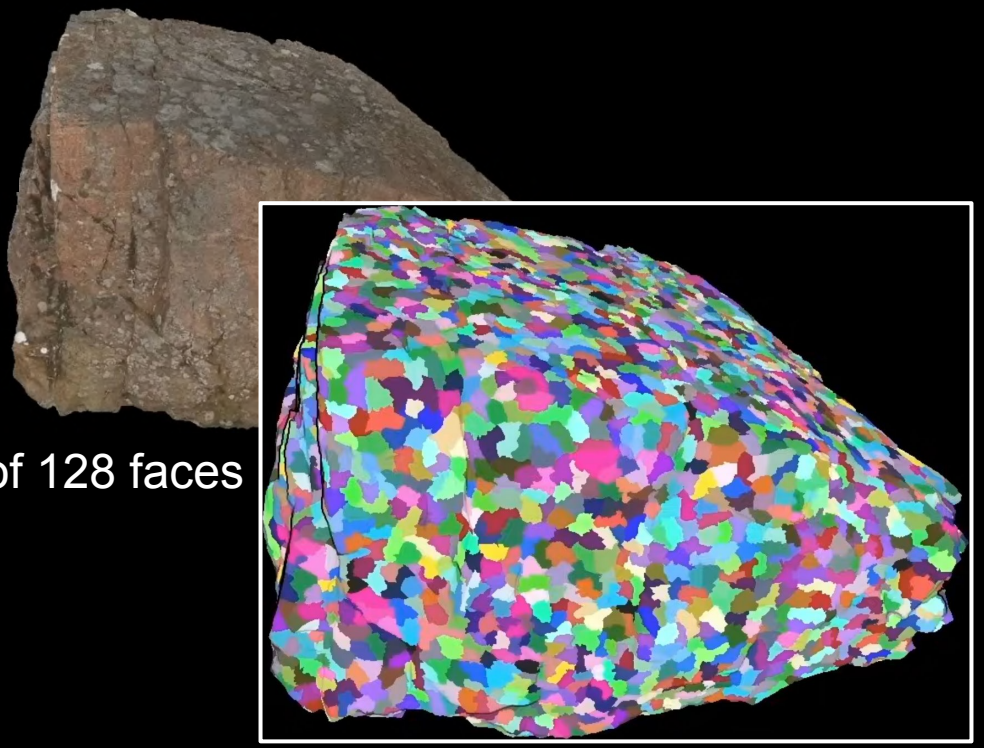
Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

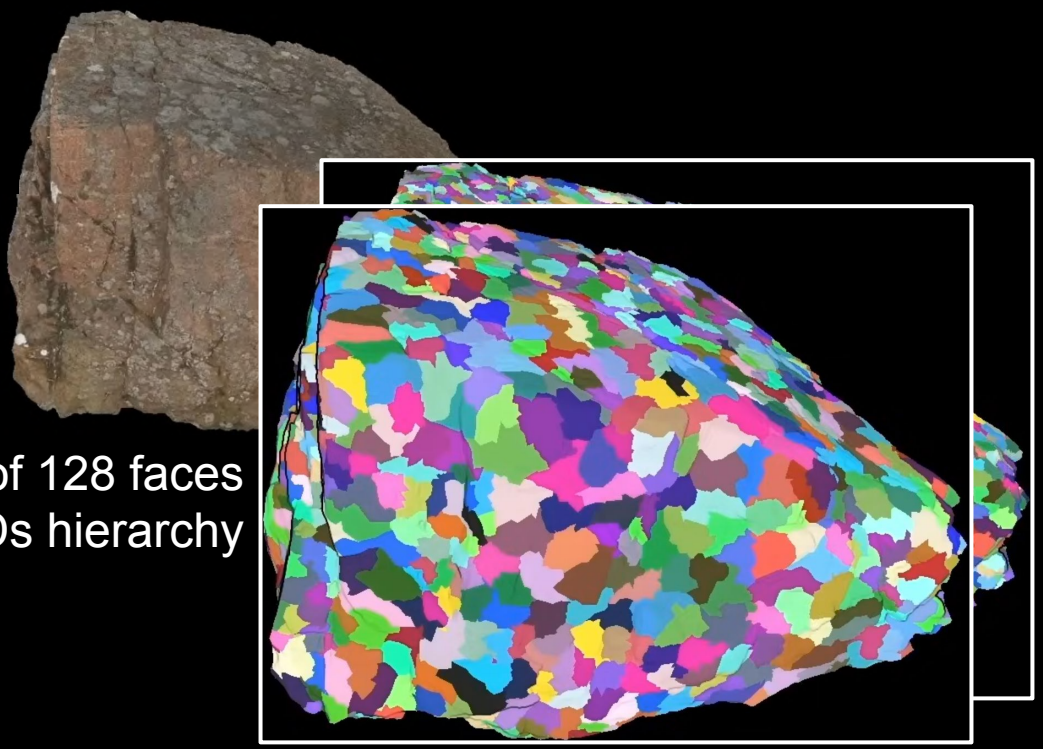


Clusters of 128 faces

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

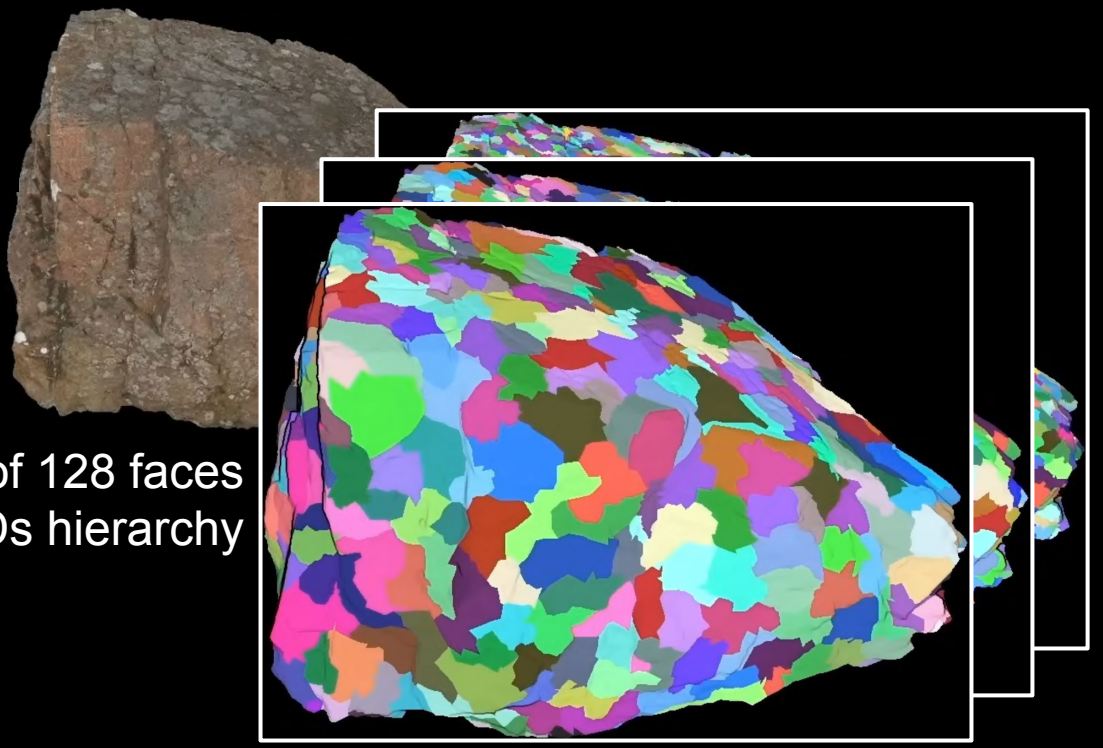


Clusters of 128 faces
+ LODs hierarchy

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

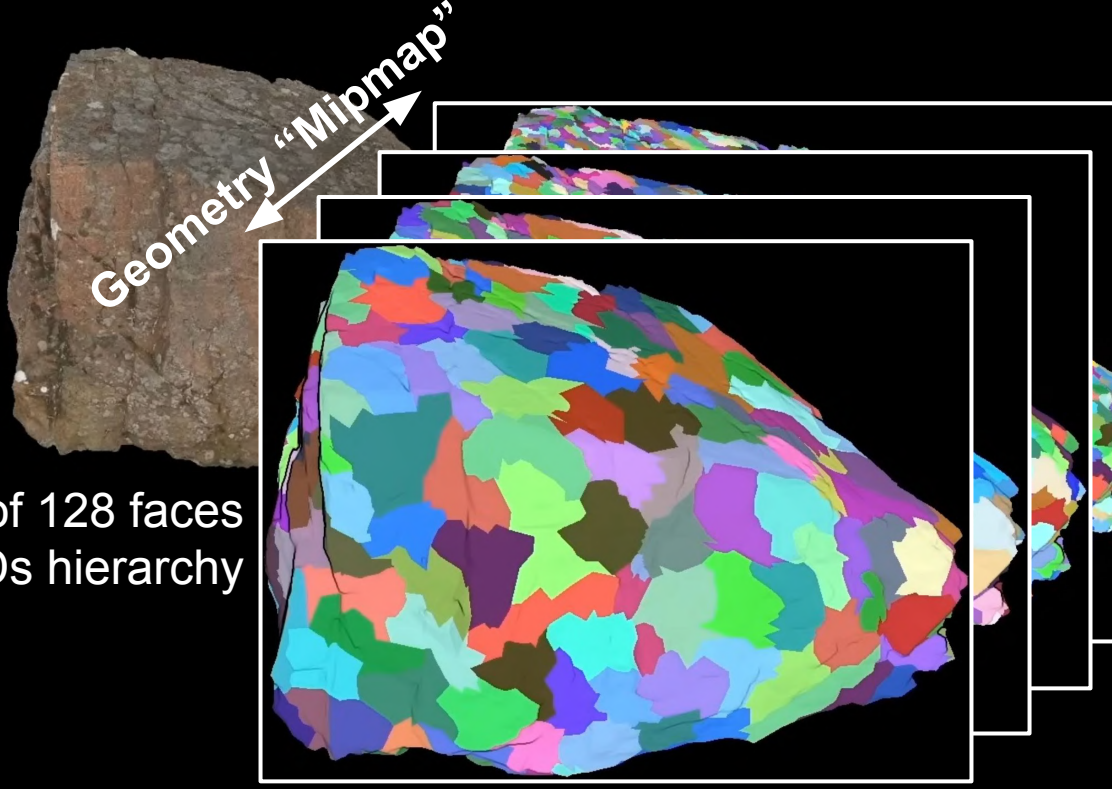


Clusters of 128 faces
+ LODs hierarchy

Myth-Busting "Best Practices" in Unreal Engine | Unreal Fest 2024

What *is* Nanite?

Virtualized geometry: streaming
- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

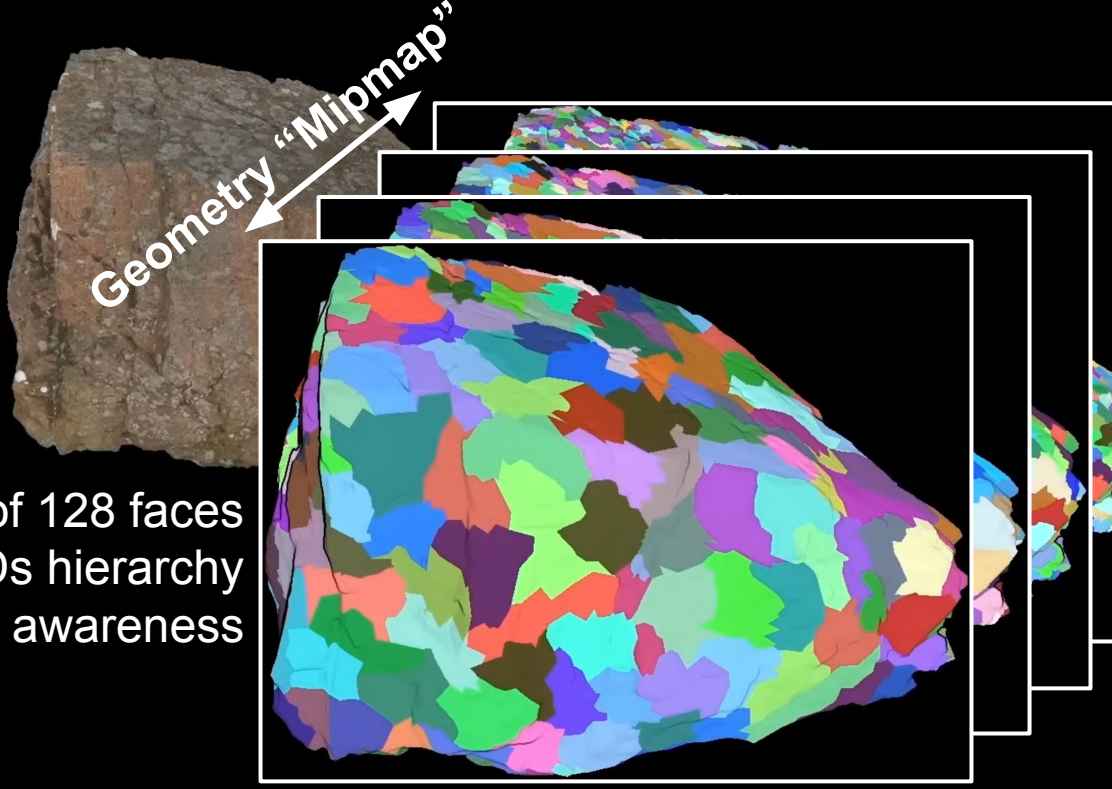


Clusters of 128 faces
+ LODs hierarchy

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

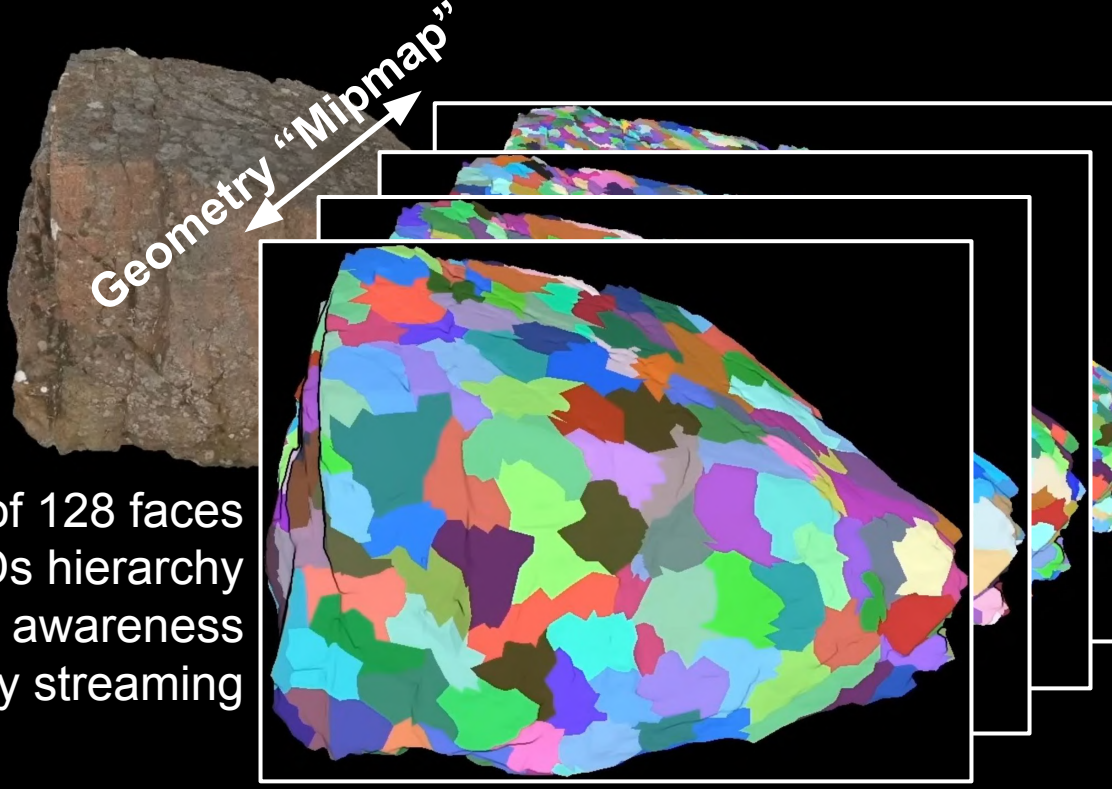


Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

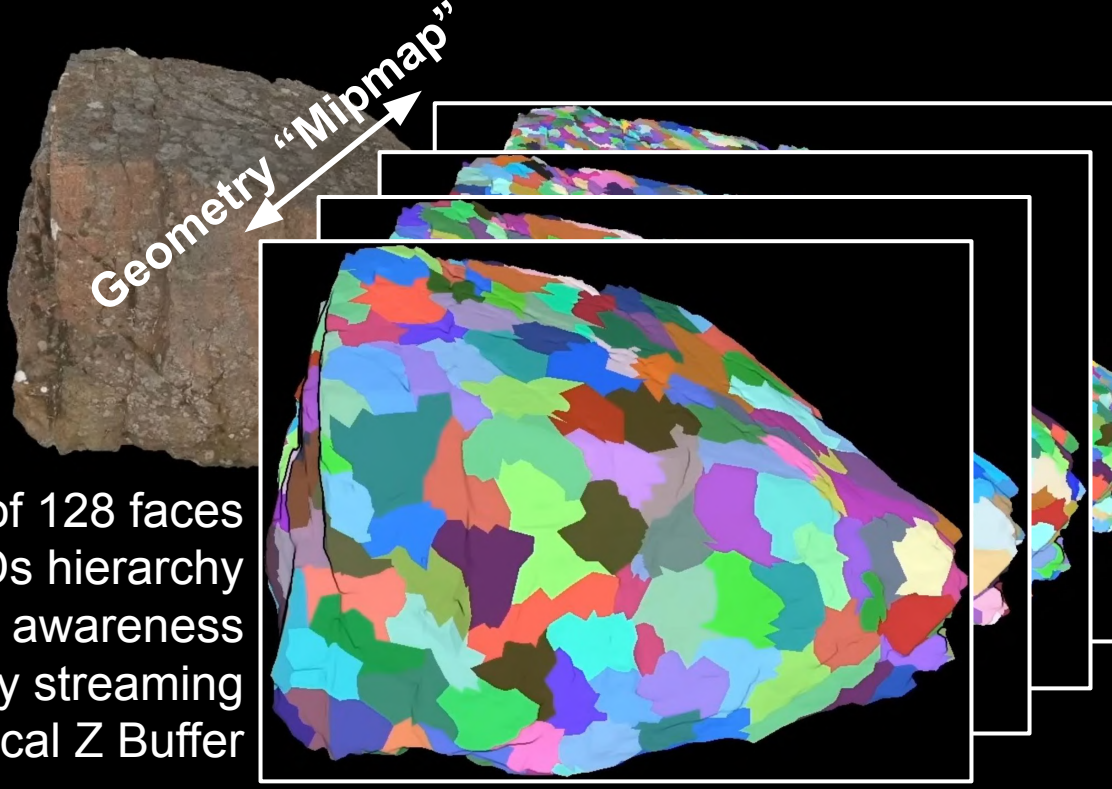


Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness
+ Geometry streaming

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

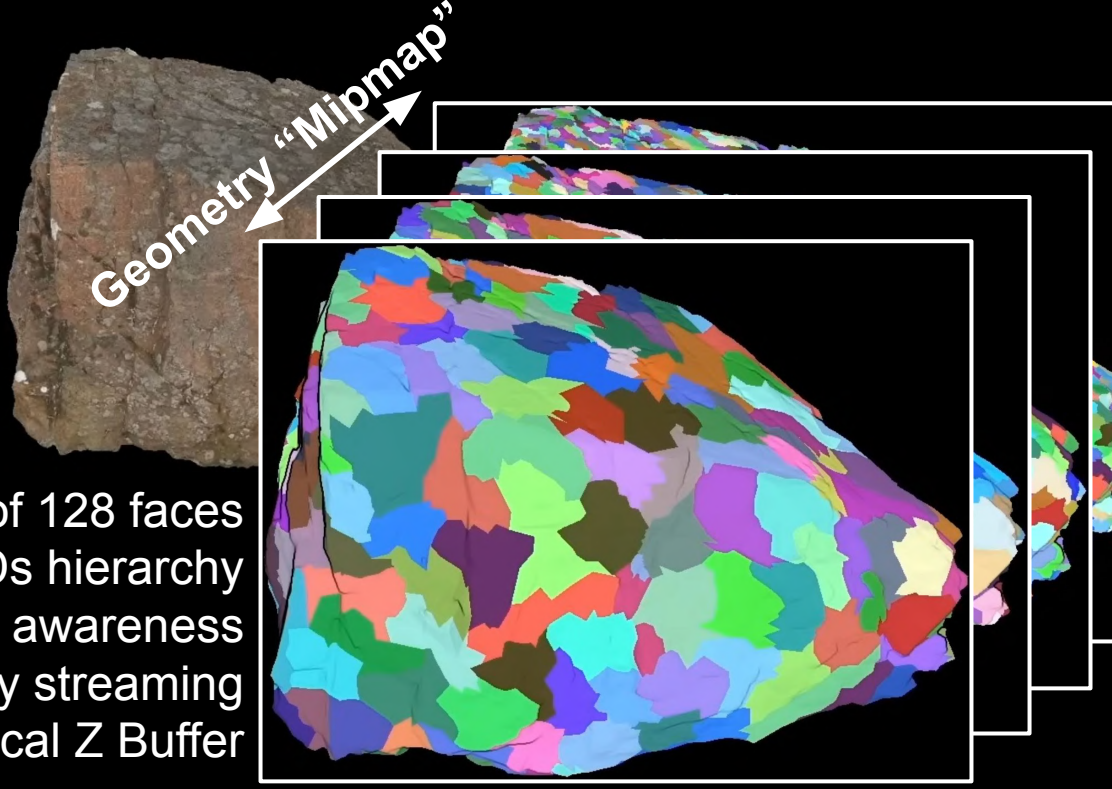


Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness
+ Geometry streaming
+ Hierarchical Z Buffer

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost



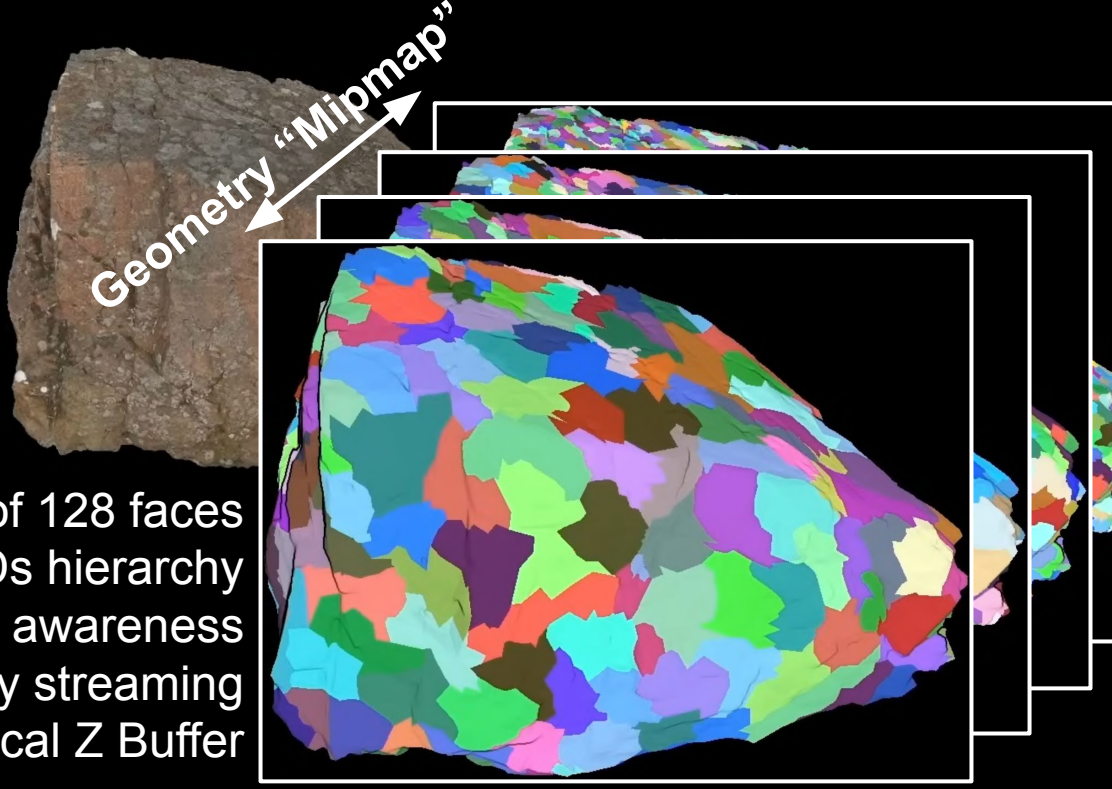
Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness
+ Geometry streaming
+ Hierarchical Z Buffer

Rasterization → Hardware (big faces)

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost



Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness
+ Geometry streaming
+ Hierarchical Z Buffer

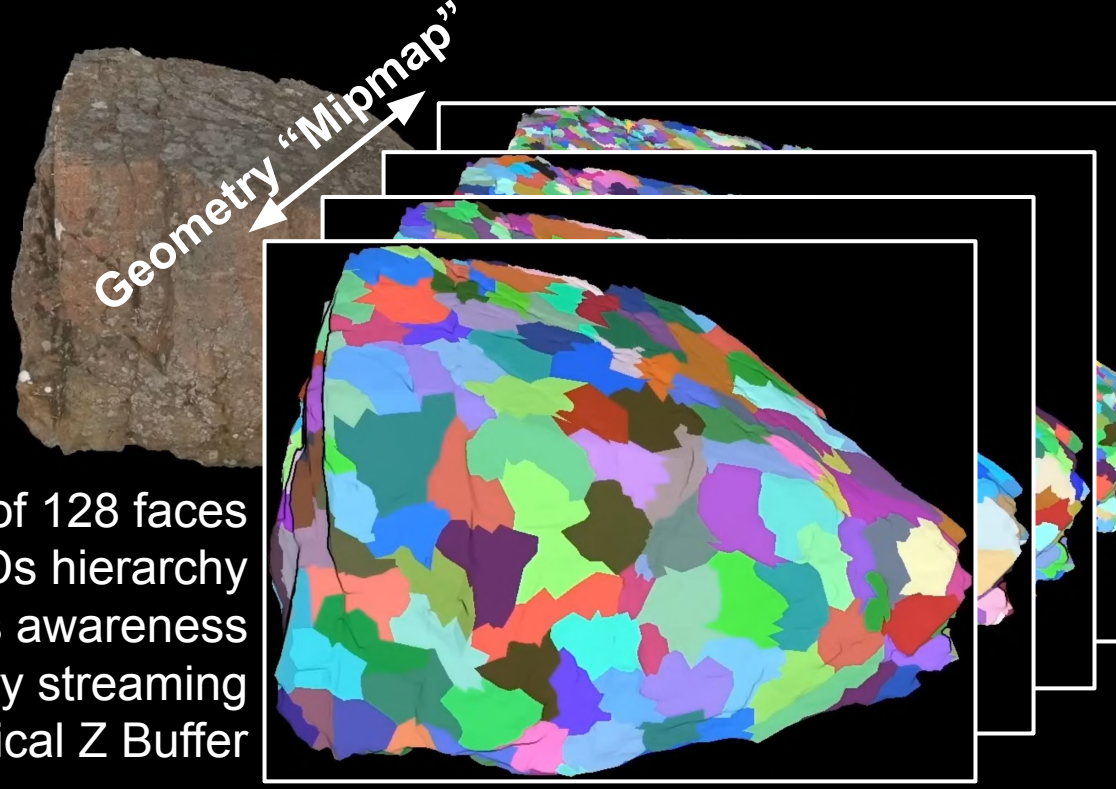
Rasterization → Hardware (big faces) + Software (~pixel-sized faces)

Myth-Busting "Best Practices" in Unreal Engine | Unreal Fest 2024

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost



What can't be Nanite

- Translucency
- Niagara (particles)
- Skeletal Meshes experimental
- Some platforms

Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness
+ Geometry streaming
+ Hierarchical Z Buffer

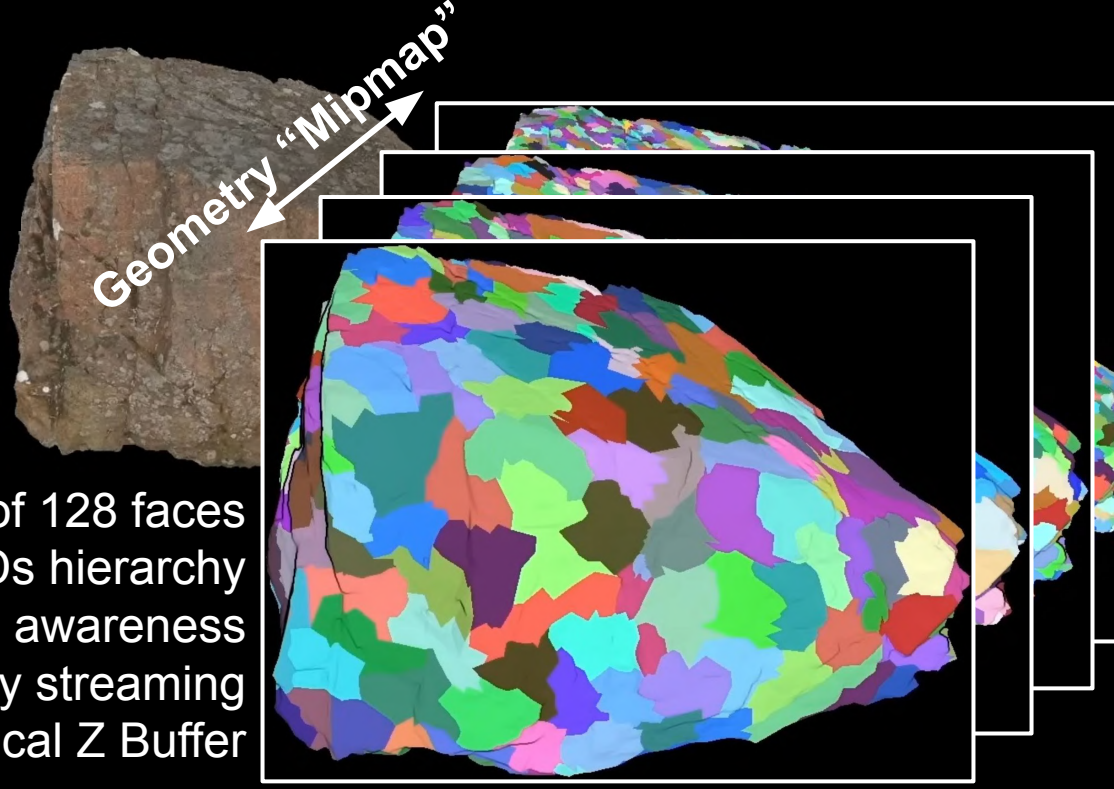
Rasterization → Hardware (big faces) + Software (~pixel-sized faces)

Myth-Busting "Best Practices" in Unreal Engine | Unreal Fest 2024

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost



What can't be Nanite

- Translucency 
- Niagara (particles)
- Skeletal Meshes experimental
- Some platforms

Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness
+ Geometry streaming
+ Hierarchical Z Buffer

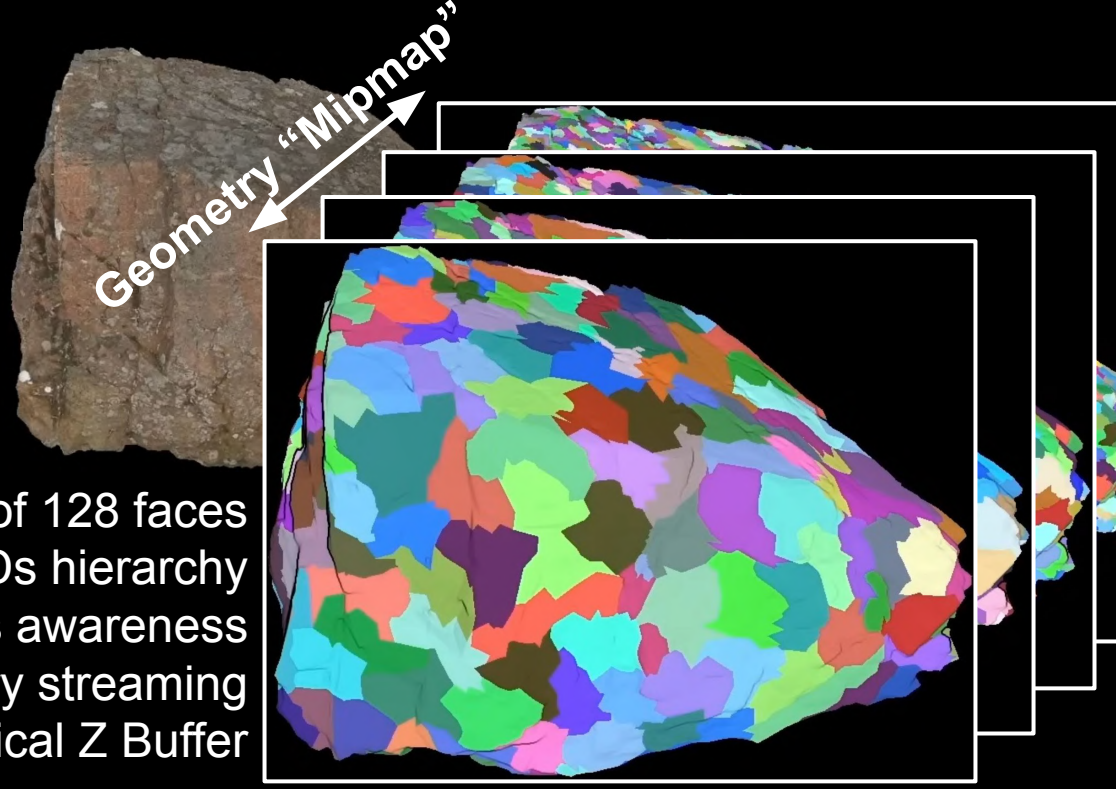
Rasterization → Hardware (big faces) + Software (~pixel-sized faces)

[Myth-Busting "Best Practices" in Unreal Engine | Unreal Fest 2024]

What *is* Nanite?

Virtualized geometry: streaming

- s.t. face edge's projection ~ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost



What can't be Nanite

- Translucency
- Niagara (particles)
- Skeletal Meshes experimental
- Some platforms

Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness
+ Geometry streaming
+ Hierarchical Z Buffer

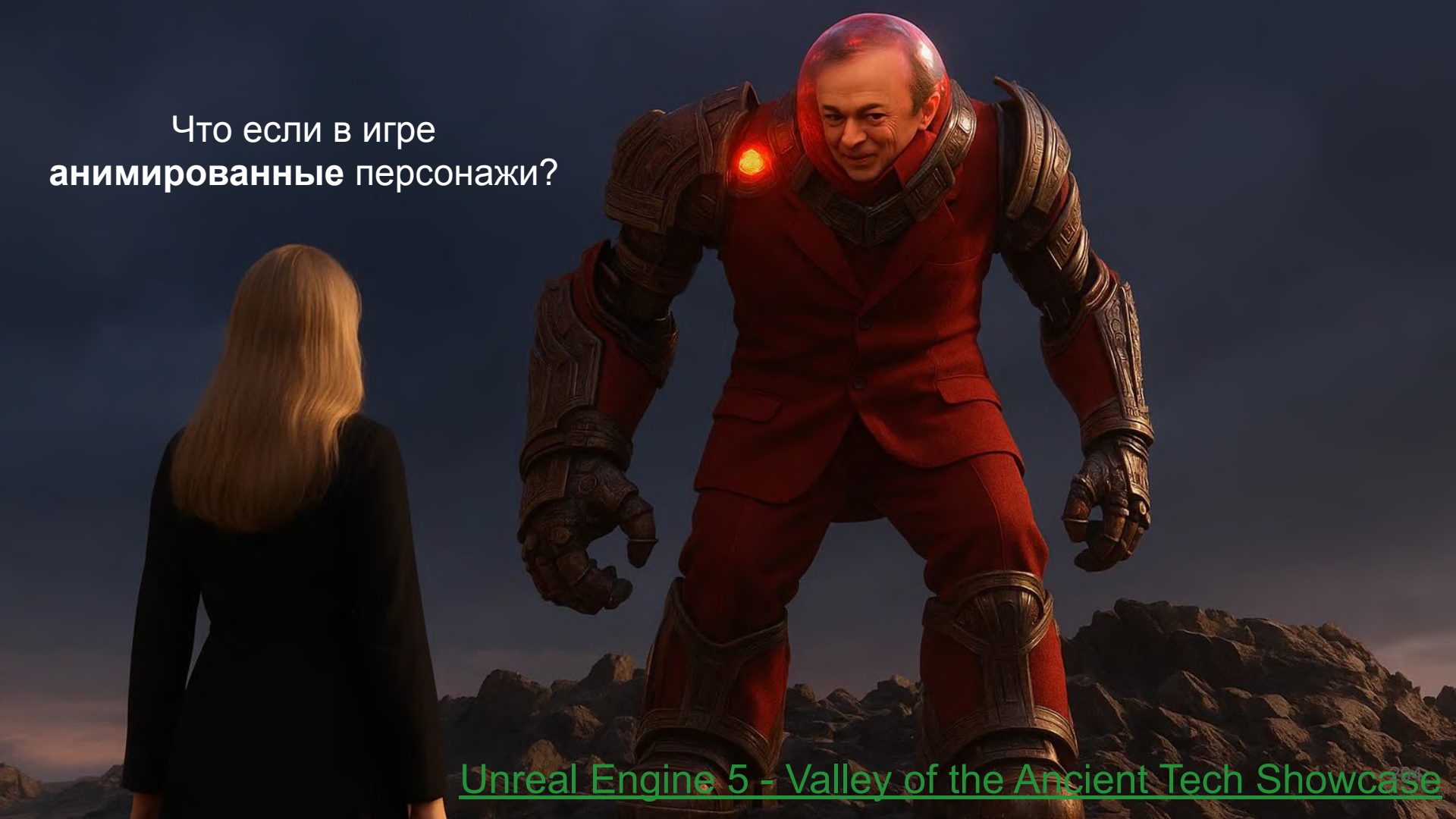
Rasterization <

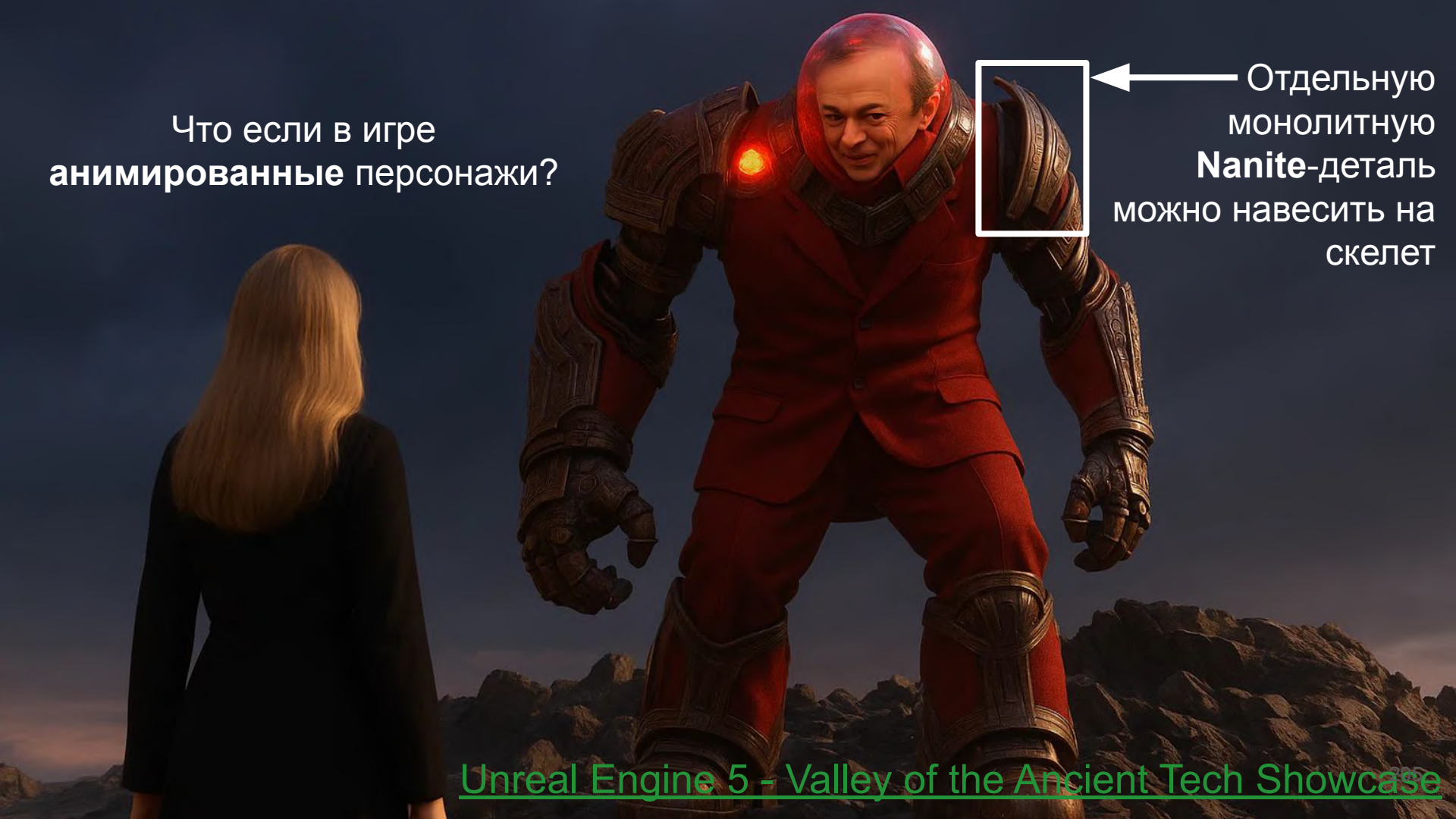
lware (big faces)

+

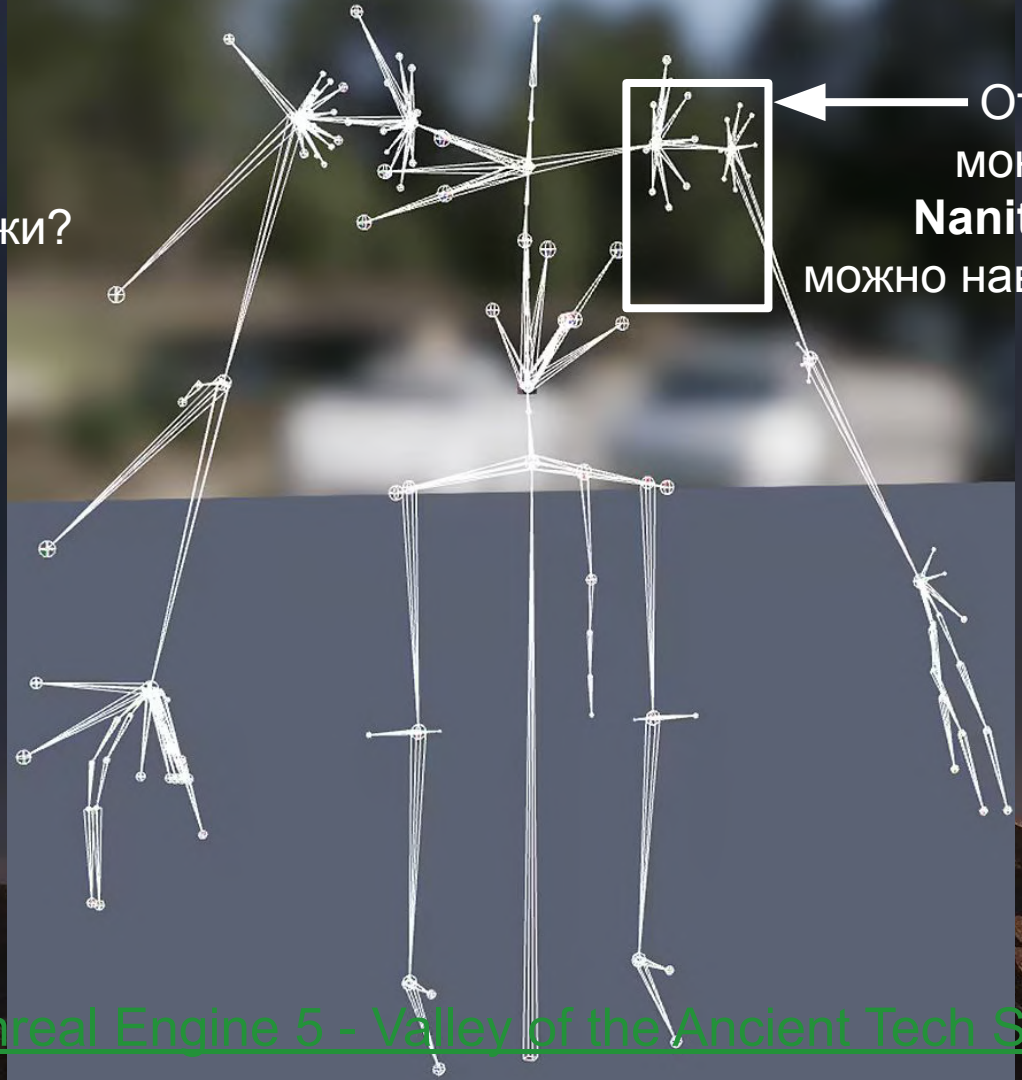
ware (~pixel-sized faces)

Глава 30: Анимация и rigging


Что если в игре
анимированные персонажи?
Unreal Engine 5 - Valley of the Ancient Tech Showcase


Что если в игре
анимированные персонажи?
Отдельную монолитную **Nanite**-деталь можно навесить на скелет
Unreal Engine 5 - Valley of the Ancient Tech Showcase

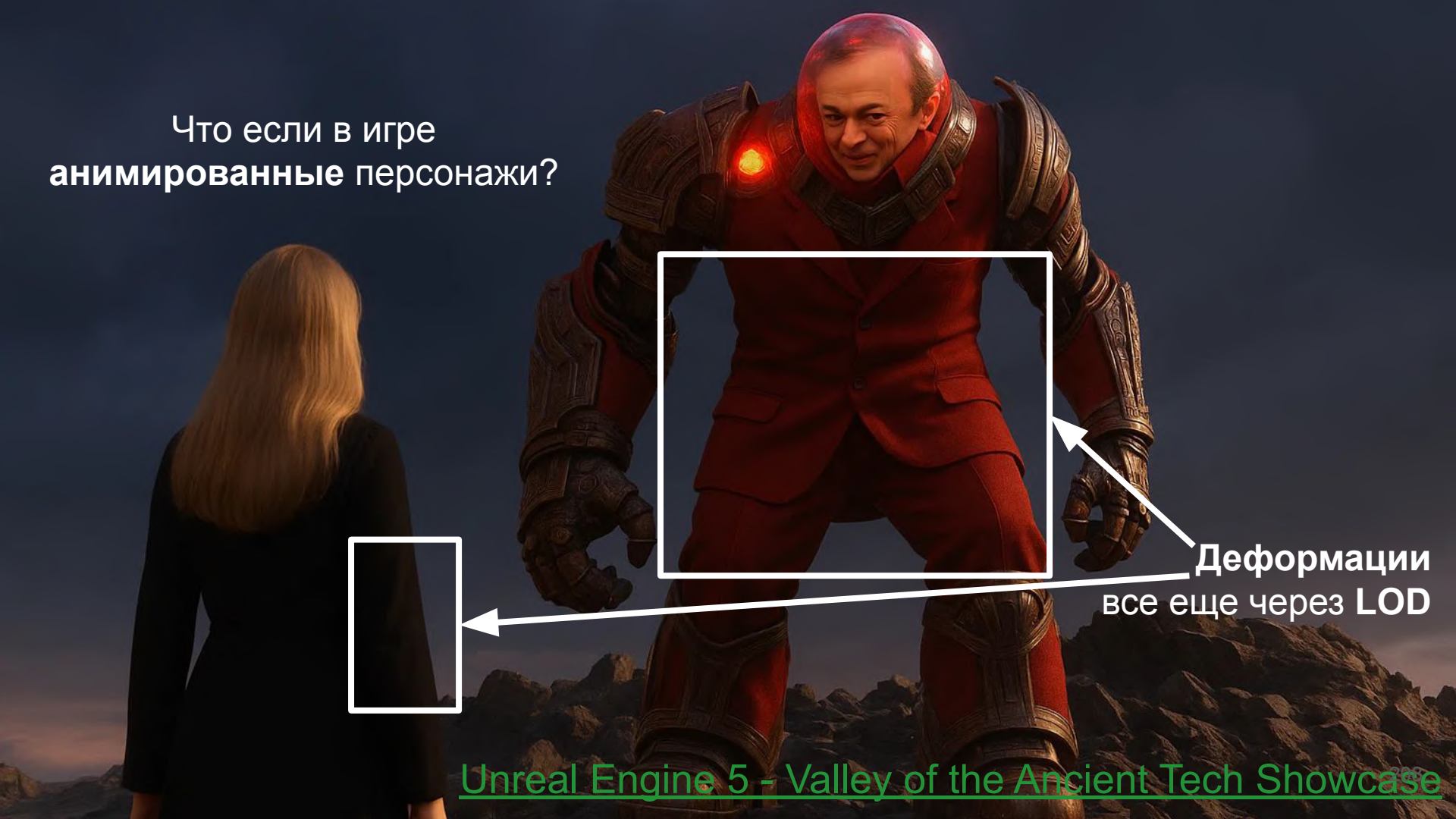
Что если в игре
анимированные персонажи?



Отдельную
монолитную
Nanite-деталь
можно навесить на
скелет

Unreal Engine 5 - Valley of the Ancient Tech Showcase


Animating and rigging the robot in Valley of the Ancient


Что если в игре
анимированные персонажи?
Деформации
все еще через **LOD**
Unreal Engine 5 - Valley of the Ancient Tech Showcase


Что если в игре
анимированные персонажи?
Деформации
все еще через **LOD**
но уже появился
Nanite Skeletal Mesh в
Unreal Engine 5.5
Unreal Engine 5 - Valley of the Ancient Tech Showcase

Глава 239: Деревья и растительность

Large Scale Animated Foliage in The Witcher 4

UNREAL FEST
Stockholm 2025

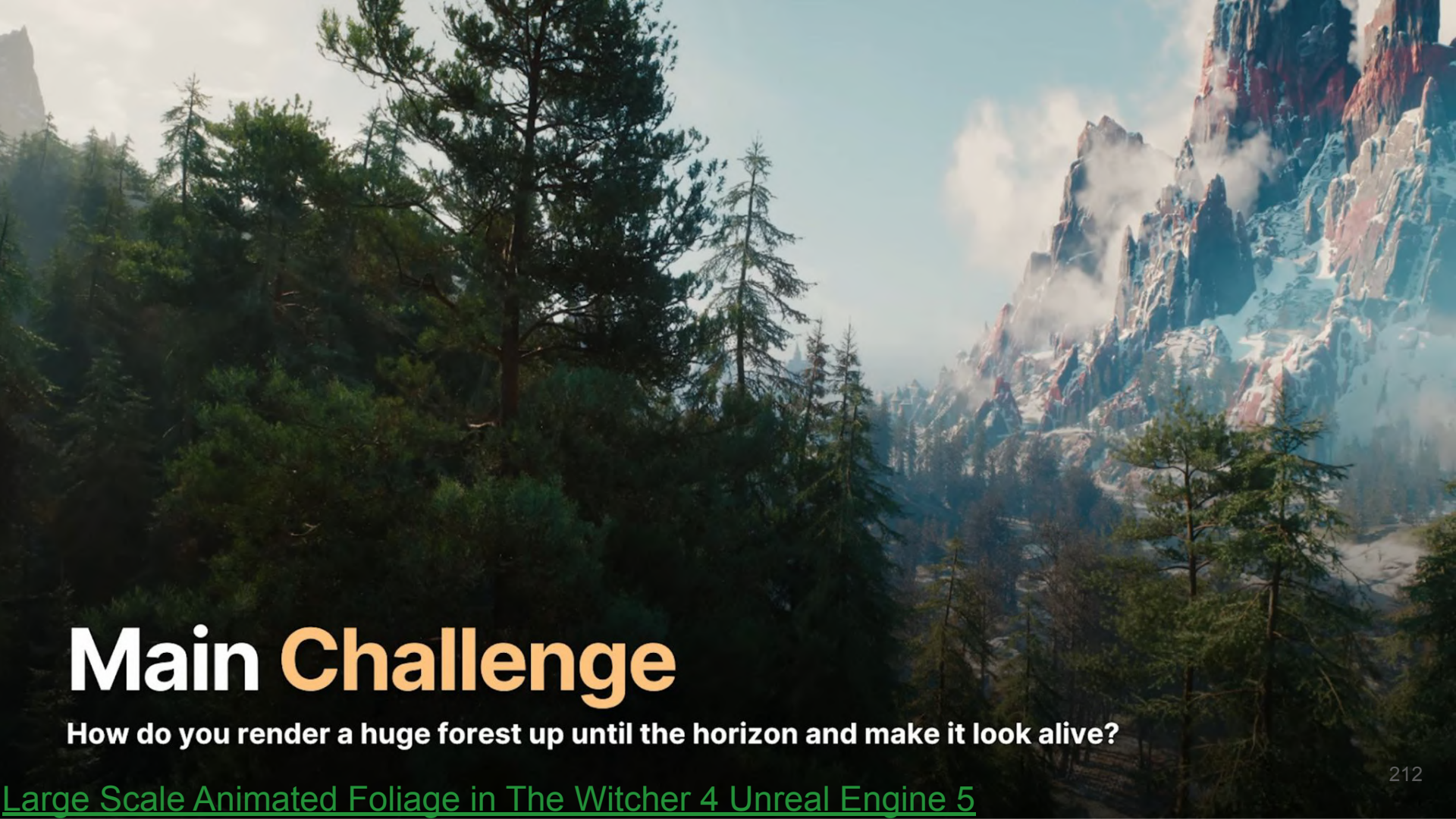
Large Scale Animated Foliage

in The Witcher 4 Unreal Engine 5 Tech Demo

Kevin Örtegren
Lead Rendering Programmer
at Epic Games

Thijs Wingerden
Engineering Manager, Rendering
at CD PROJEKT RED

Vignesh Gunasekaran
Senior Engineer, Rendering
at CD PROJEKT RED



Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

Why is foliage hard?

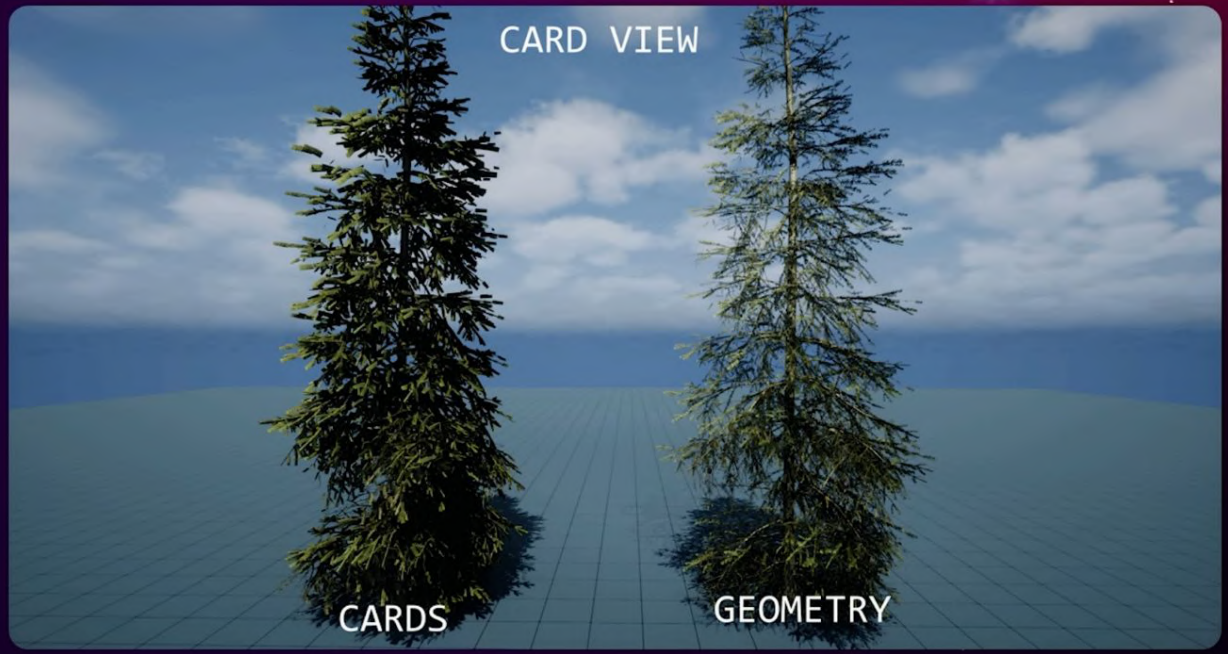
Nanite is great but it struggled with foliage

Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

Cards

Alpha-tested geometry

- Alpha testing still expensive.
- Avoid it as much as possible. 3-4x slower.



100k tris **6m tris**

Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

Cards

Alpha-tested geometry

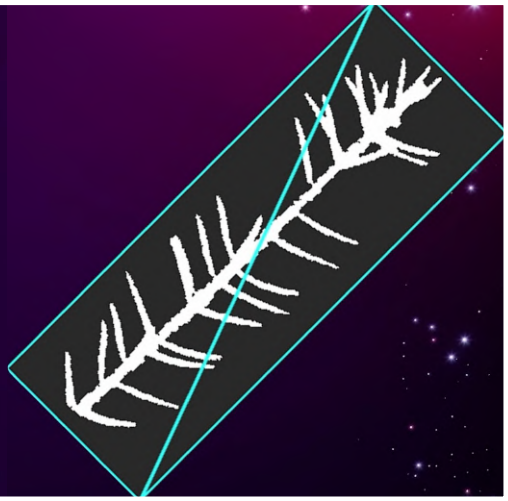
- Alpha testing still expensive.
- Avoid it as much as possible. 3-4x slower.

Barycentrics

~~Fixed function~~

Dispatch per material

- We need to calculate barycentrics per pixel to get UVs and then sample the alpha-mask texture.
- See NaniteRasterizationCommon.ush



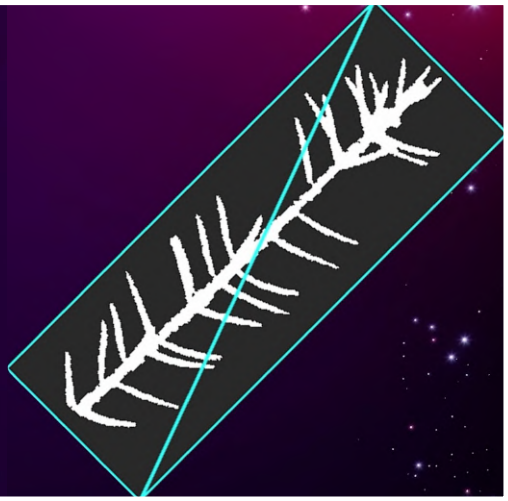
Software barycentrics (особенно в software rasterizer) - медленно! Особенно учитывая большой **overdraw.**

Barycentrics

~~Fixed function~~

Dispatch per material

- We need to calculate barycentrics per pixel to get UVs and then sample the alpha-mask texture.
- See NaniteRasterizationCommon.ush



Cards

Alpha-tested geometry

- Alpha testing still expensive.
- Avoid it as much as possible. 3-4x slower.

Cards

Alpha-tested geometry

- Alpha testing still expensive.
- Avoid it as much as possible. 3-4x slower.

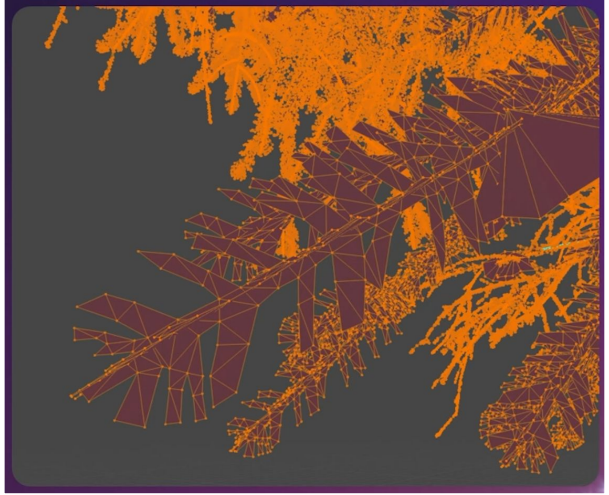
Barycentrics

~~Fixed function~~

Dispatch per material

- We need to calculate barycentrics per pixel to get UVs and then sample the alpha-mask texture.
- See NaniteRasterizationCommon.ush

Vertices at runtime
Cards 100k+
Geometry 1m+



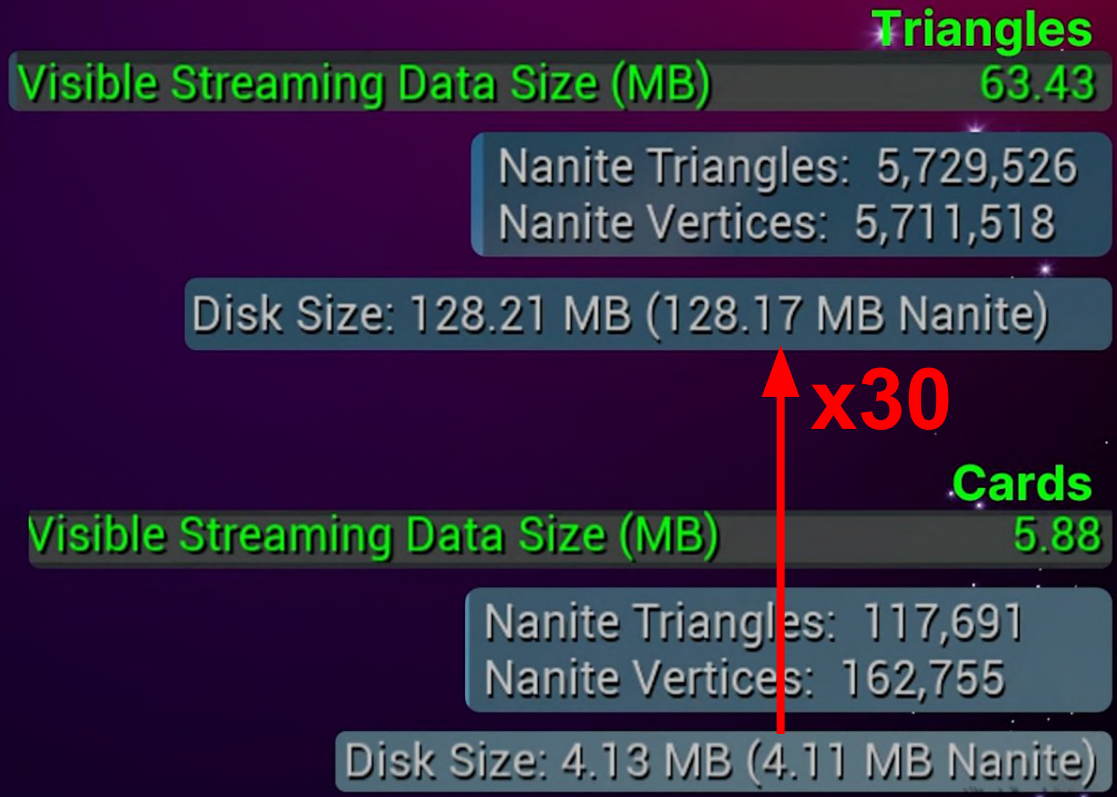
100k tris

6m tris

Geometry

Fully meshed out tree. No alpha.

- Faster. Looks better.
- No planes, no fake normals.
- That is a lot of data in RAM and on DISK

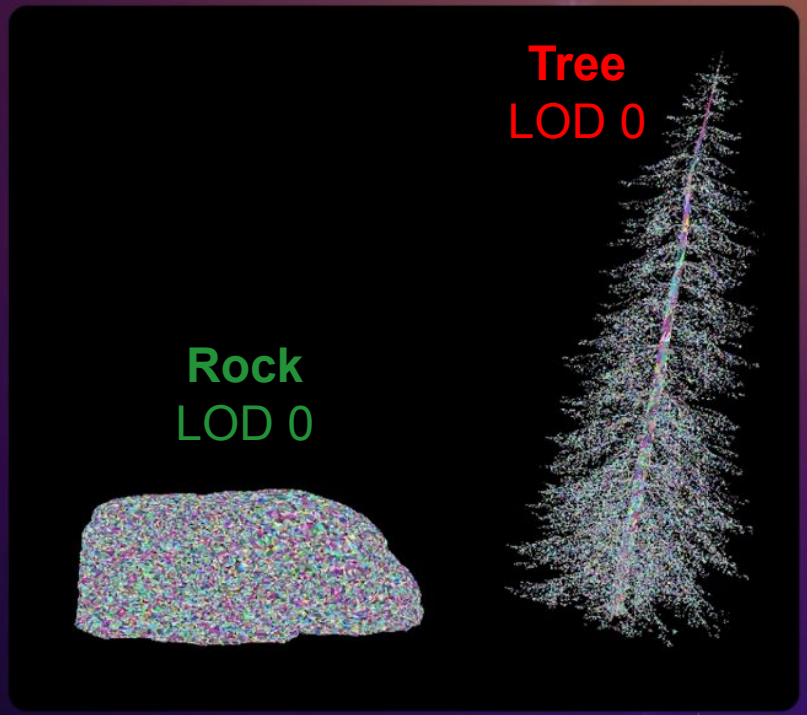


Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

When Nanite simplifies the geometry in the distance … it … starts removing some of these disjoint elements completely … the result **will look thinned** out because there was major surface area loss.
Preserve Area will redistribute that lost area to the remaining triangles by dilating out the open boundary edges.

Inefficient LODs

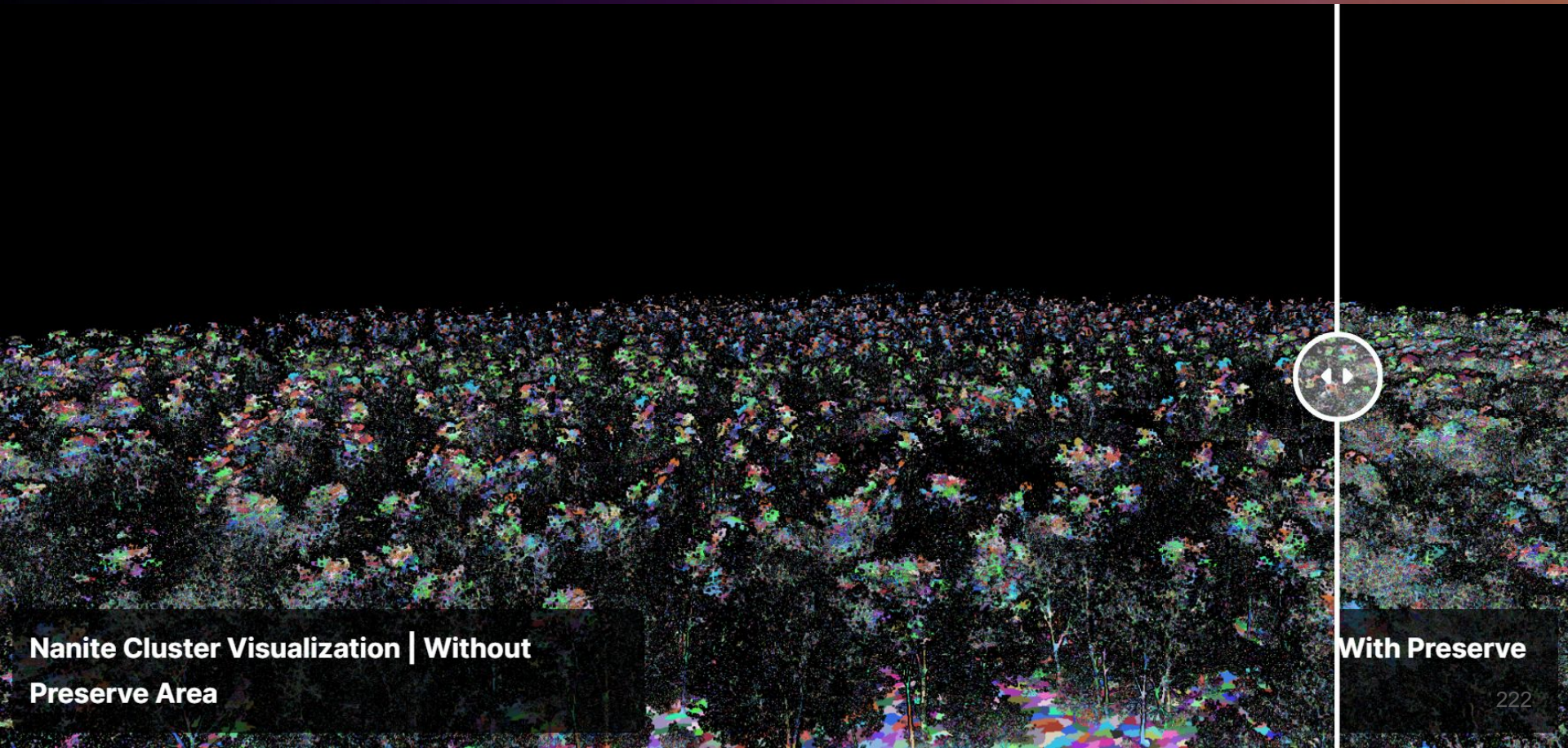
- Even with **Preserve Area**, foliage does not keep the shape at the distance.
- Nanite aggregate geometry reduction is sub-optimal.
- Foliage causes excessive overdraw.



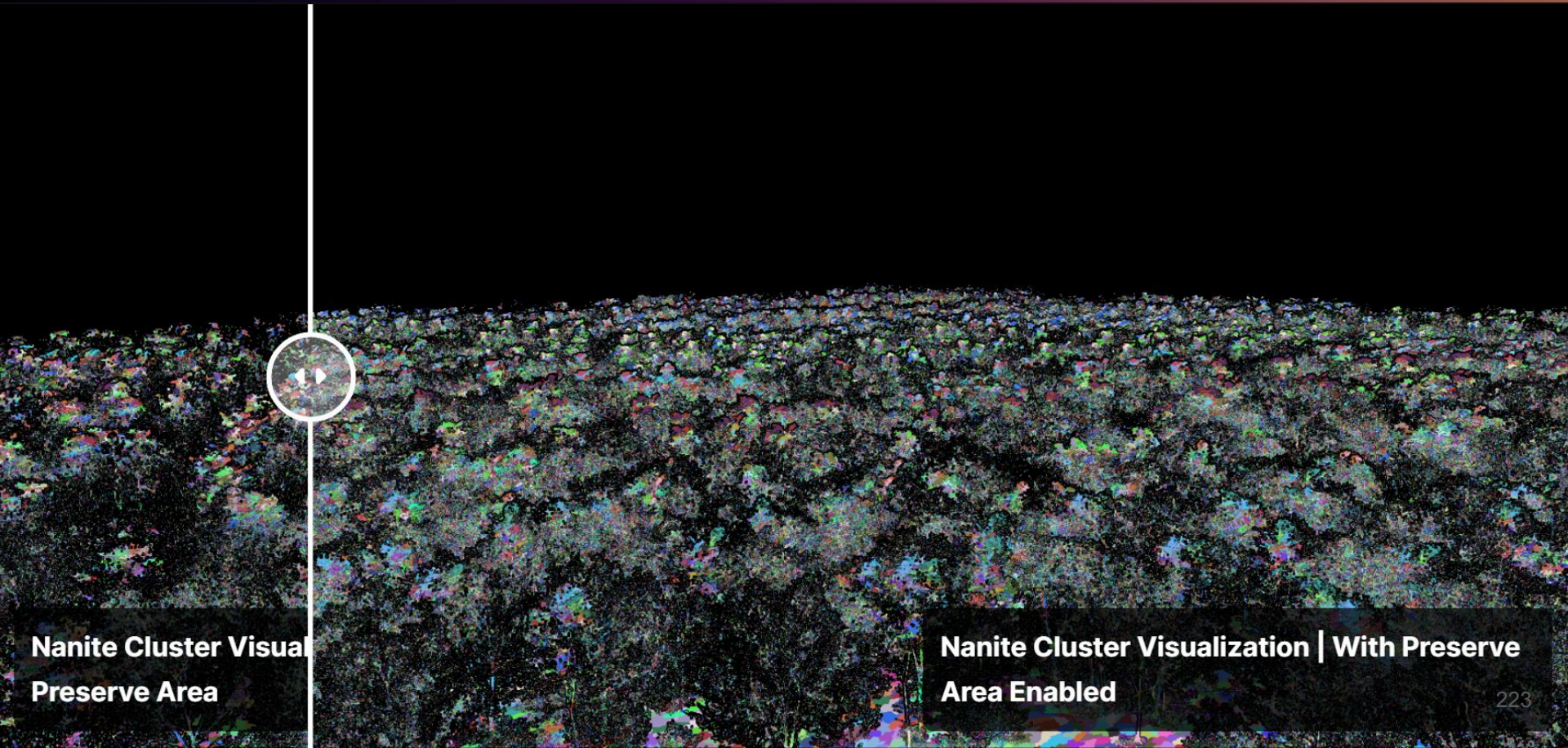
When Nanite simplifies the geometry in the distance … it … starts removing some of these disjoint elements completely … the result **will look thinned** out because there was major surface area loss.
Preserve Area will redistribute that lost area to the remaining triangles by dilating out the open boundary edges.

When Nanite simplifies the geometry in the distance … it … starts removing some of these disjoint elements completely … the result **will look thinned** out because there was major surface area loss.
Preserve Area will redistribute that lost area to the remaining triangles by dilating out the open boundary edges.

When Nanite simplifies the geometry in the distance … it … starts removing some of these disjoint elements completely … the result **will look thinned** out because there was major surface area loss.
Preserve Area will redistribute that lost area to the remaining triangles by dilating out the open boundary edges.

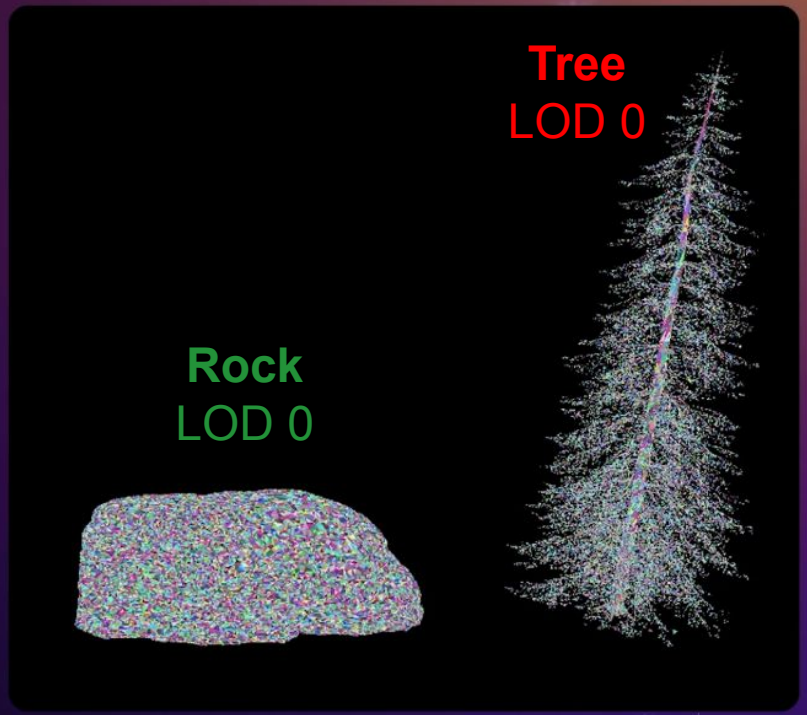


When Nanite simplifies the geometry in the distance … it … starts removing some of these disjoint elements completely … the result **will look thinned** out because there was major surface area loss.
Preserve Area will redistribute that lost area to the remaining triangles by dilating out the open boundary edges.



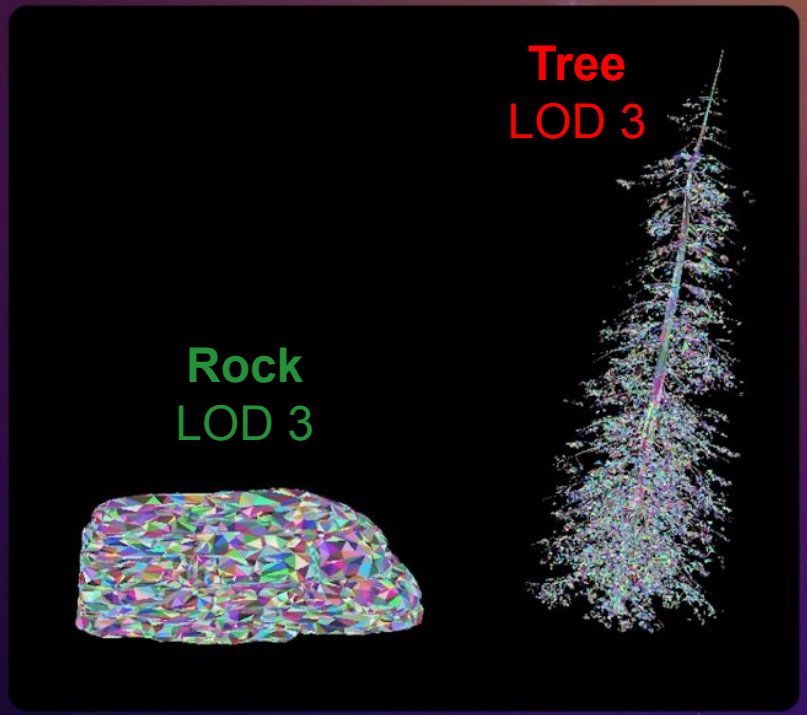
Inefficient LODs

- Even with **Preserve Area**, foliage does not keep the shape at the distance.
- Nanite aggregate geometry reduction is sub-optimal.
- Foliage causes excessive overdraw.



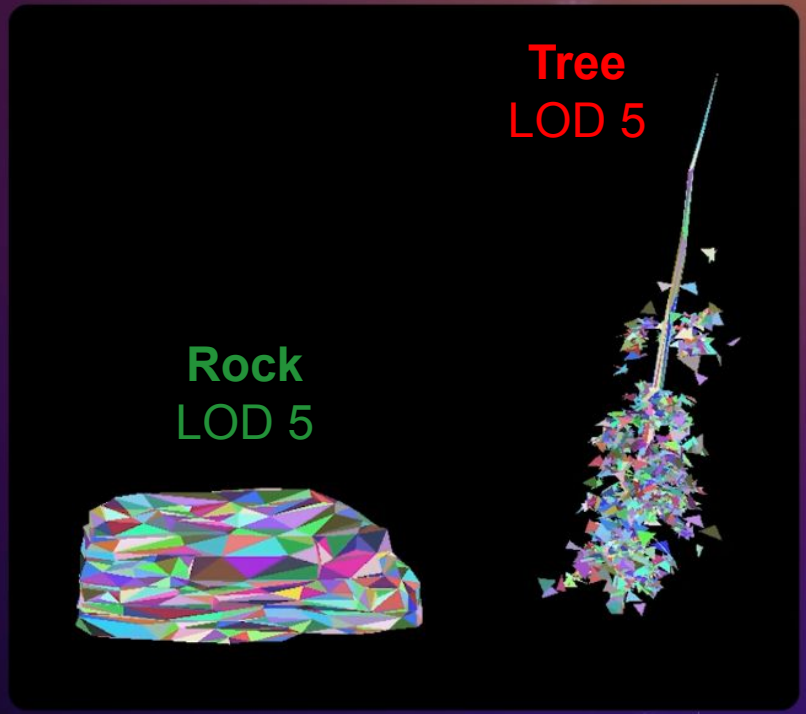
Inefficient LODs

- Even with **Preserve Area**, foliage does not keep the shape at the distance.
- Nanite aggregate geometry reduction is sub-optimal.
- Foliage causes excessive overdraw.



Inefficient LODs

- Even with **Preserve Area**, foliage does not keep the shape at the distance.
- Nanite aggregate geometry reduction is sub-optimal.
- Foliage causes excessive overdraw.

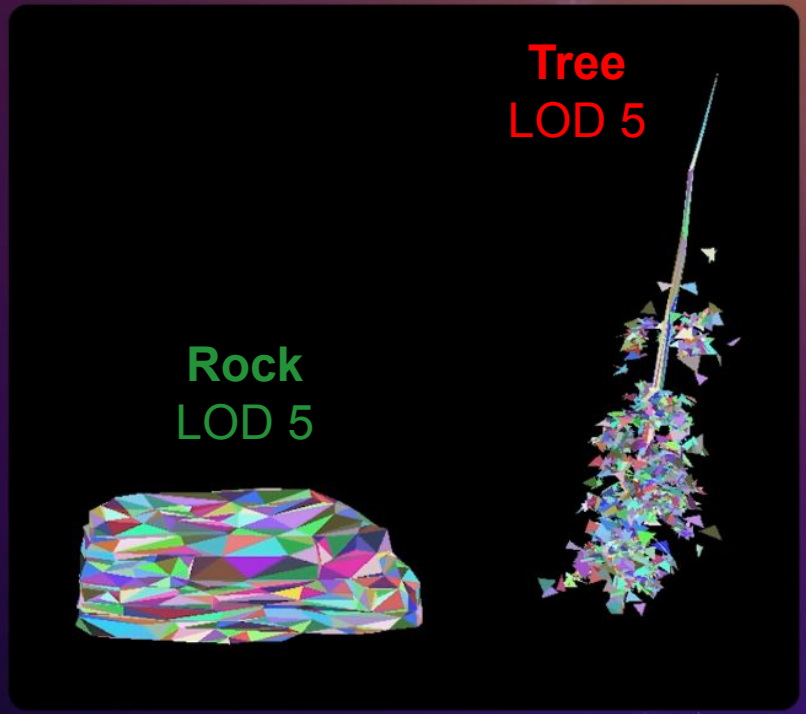


Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

When Nanite simplifies the geometry in the distance … it … starts removing some of these disjoint elements completely … the result **will look thinned** out because there was major surface area loss.
Preserve Area will redistribute that lost area to the remaining triangles by dilating out the open boundary edges.

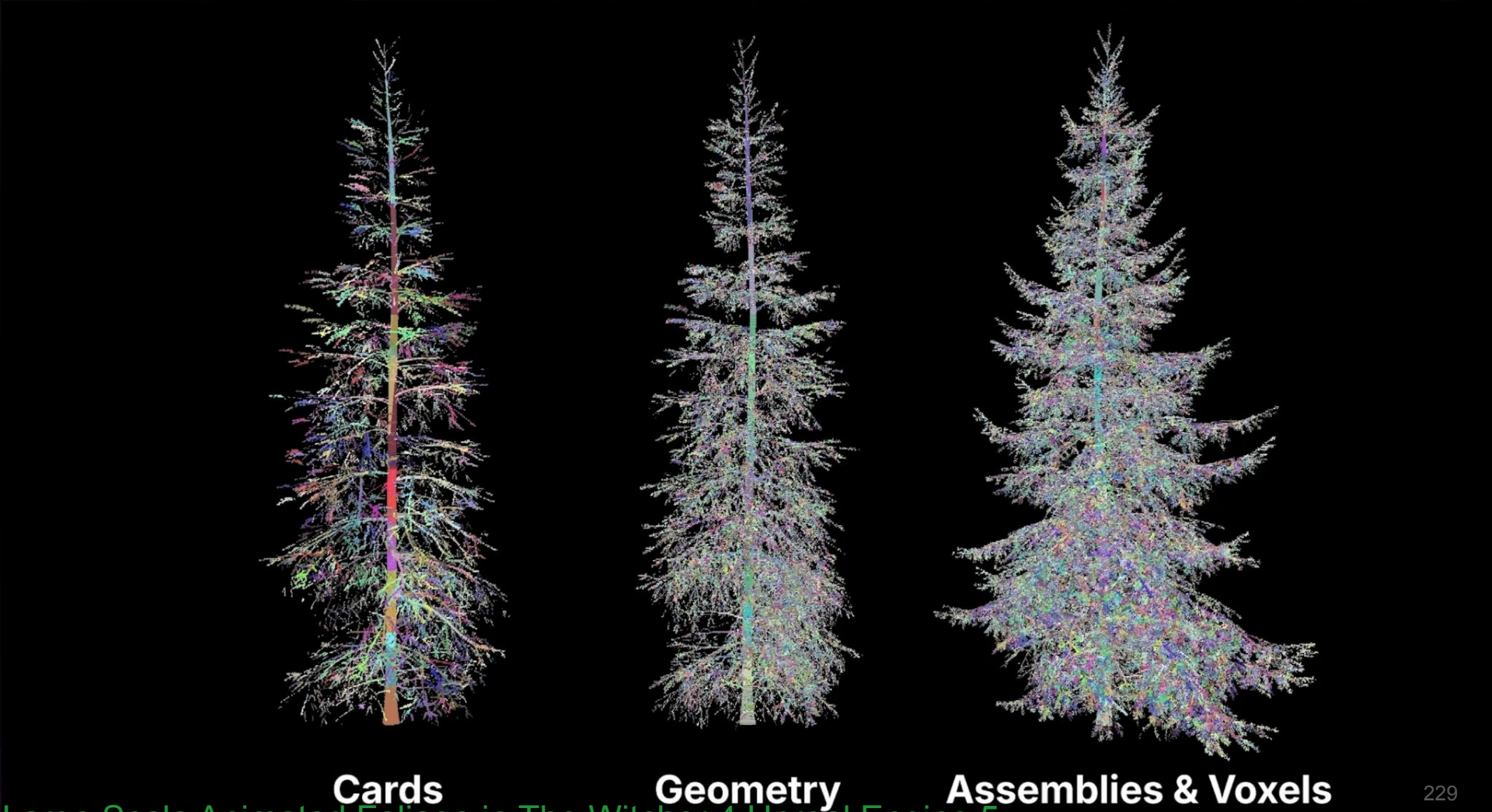
Inefficient LODs

- Even with **Preserve Area**, foliage does not keep the shape at the distance.
- Nanite aggregate geometry reduction is sub-optimal.
- Foliage causes excessive overdraw.

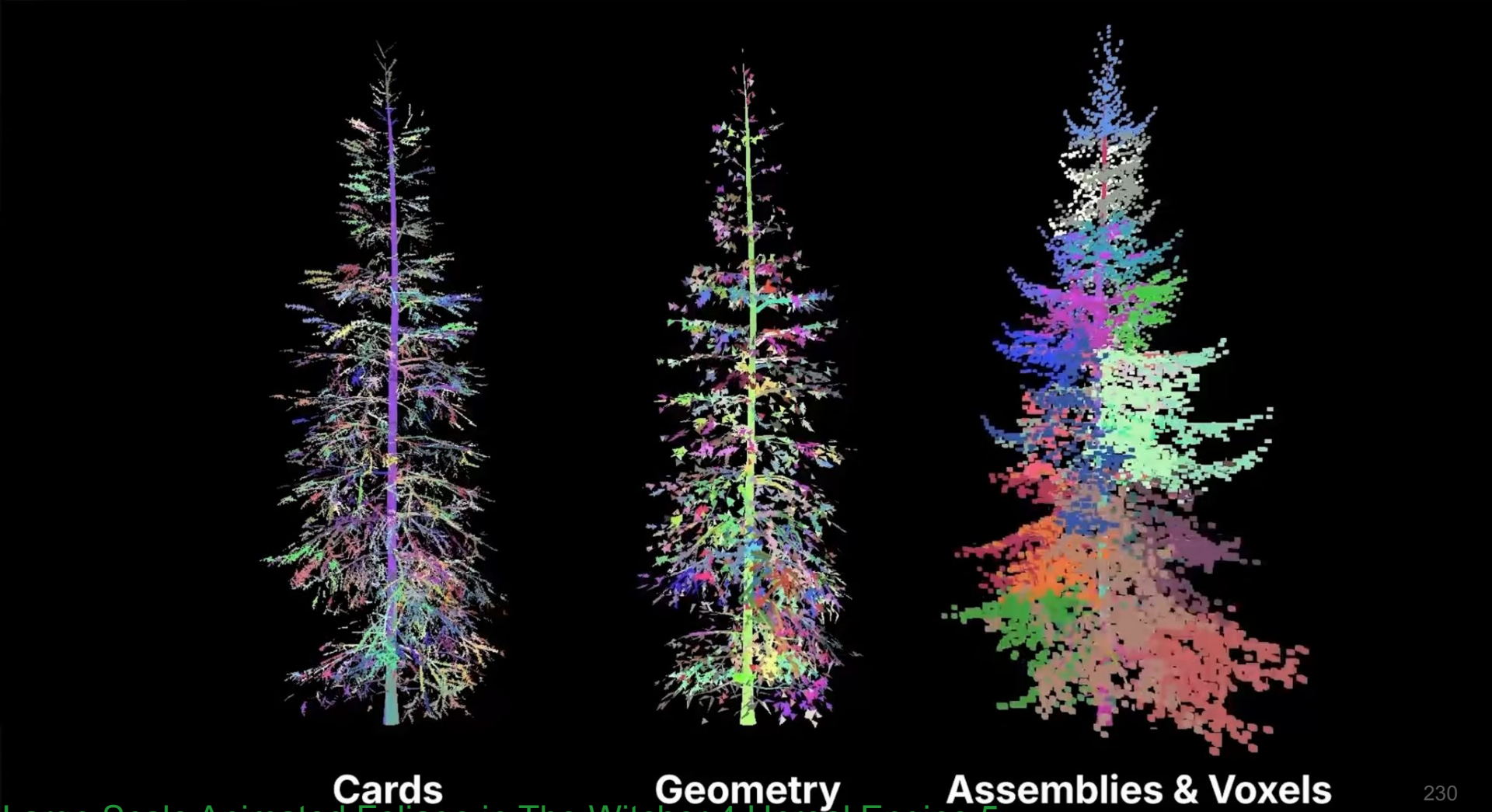


Voxels

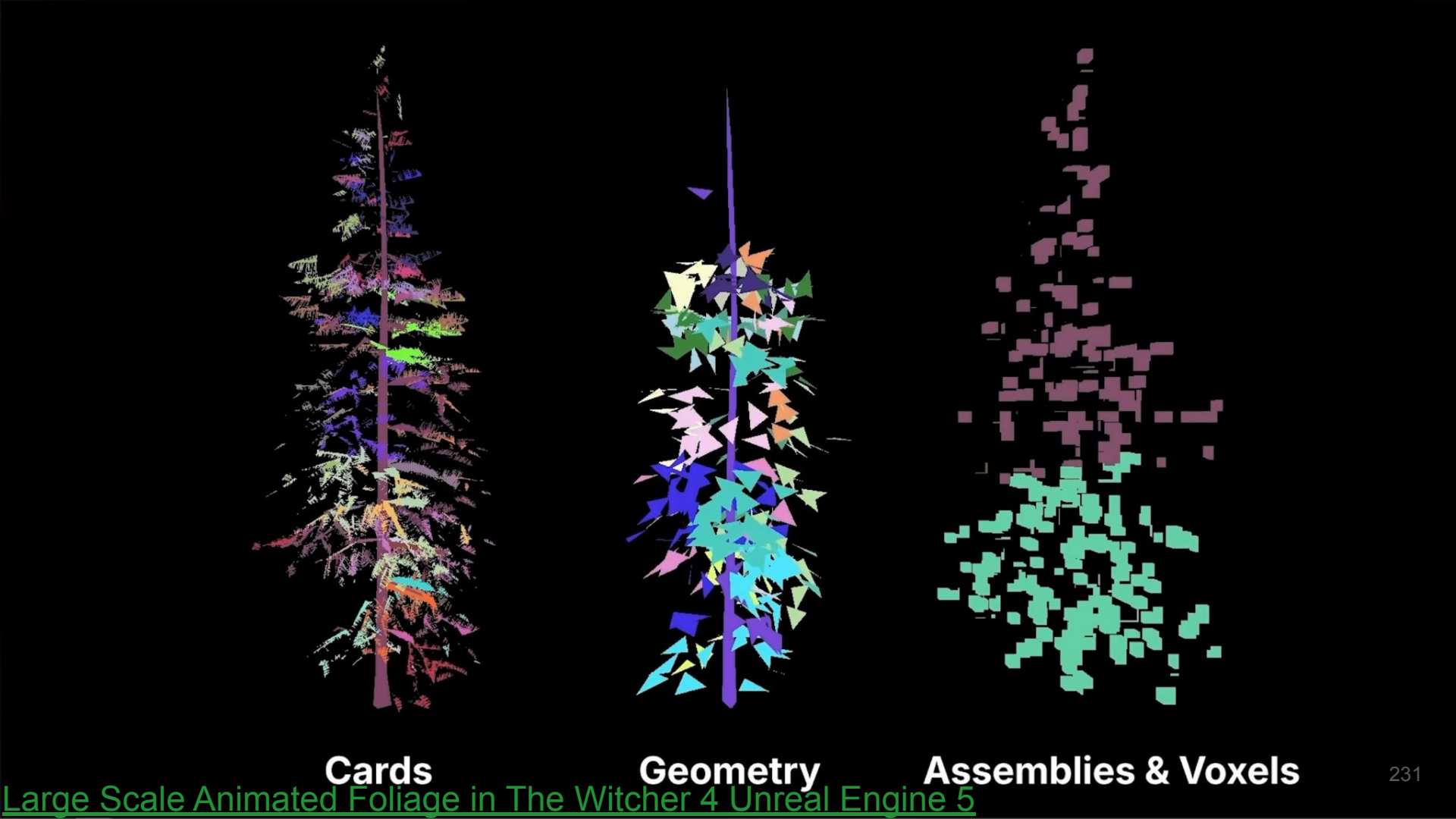
- Sub-pixel triangle clusters are now voxel clusters.
- Stored as 4x4x4 voxels bricks, represented by unit64.
- These bricks are traced against during nanite rasterization pass.
- Binned in depth buckets and rendered front to back for early depth testing.



Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5



Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

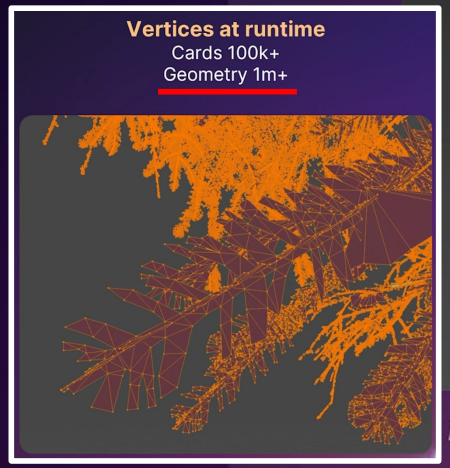


Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

Animation

We started with Pivot Painter 2 vertex shader wind

Pivots and their hierarchies are stored in textures. Then at runtime for each vertex the whole hierarchy is evaluated at the vertex rotated.



Example from Epic documentation

[Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5]

Skinned Foliage

- Brought down the animation cost by a lot.
- Two-pass compute based pipeline scaled much better with vertex count.

Large Scale Animated Foliage in The Witcher 4 Unreal Engine 5

Tech Demo Results in Numbers

- Vista shot renders 20K+ trees
  - Source triangle count in the billions
- Across the valley
  - 26 km^2
  - Instances
    - ~500K Trees
    - ~1.7M Shrubs
    - ~7M Grass
  - 28 species
  - Tree assets range from below 1 million to above 10 million triangles
  - Up to several hundred bones per tree

Large Scale Animated Foliage in The Witcher 4 | Unreal Engine 5

Ссылки

- A Deep Dive into Nanite Virtualized Geometry | SIGGRAPH 2021, Слайды, Конспект
- Keynote: The Journey to Nanite - Brian Karis, Epic Games | HPG 2022, Слайды
- Large Scale Animated Foliage in The Witcher 4 | Unreal Fest Stockholm 2025
- Разбор как в Nanite устроены **virtual shadows** ← не обсудили как и сжатие данных

@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный

@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный

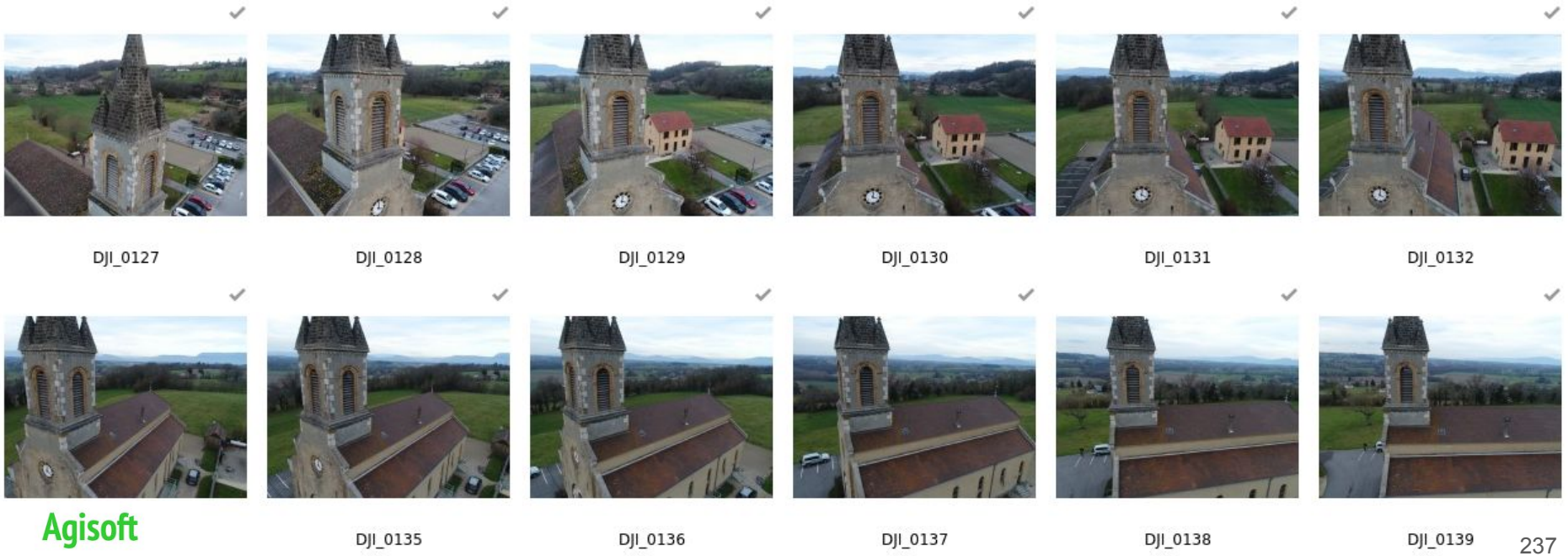
CS Space
Клуб технологий и науки

Записи прошлого года

Весной 2026 года в CS Space Клуб технологий и науки

Курс фотограмметрии: 3D реконструкция

По множеству фотографий одной и той же сцены получить цифровую модель



Agisoft Metashape

Данные предоставил Stéphane Prodent

Записи прошлого года

Весной 2026 года в CS Space Клуб технологий и науки

Курс фотограмметрии

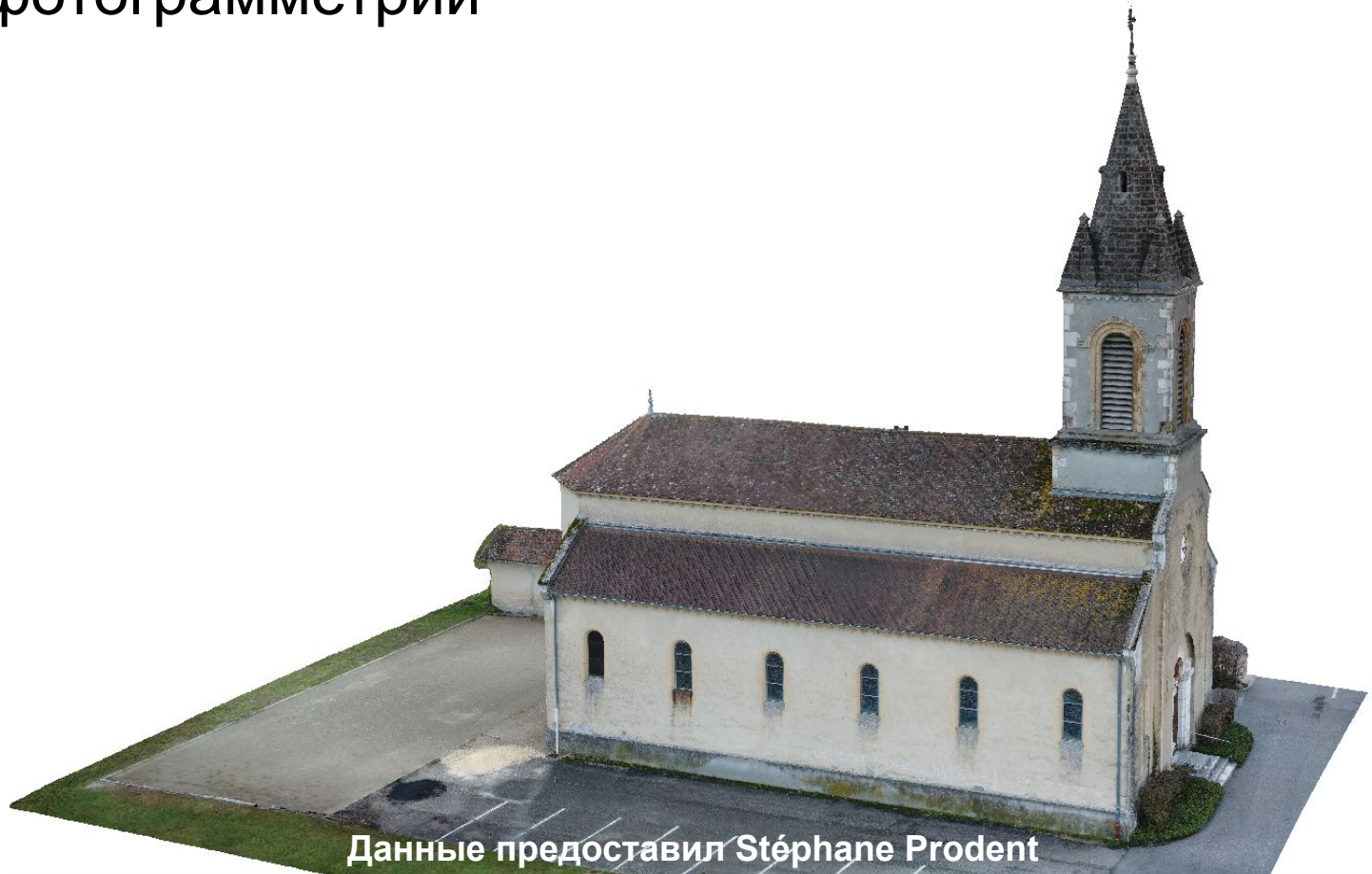
Данные предоставил Stéphane Prodent

Agisoft Metashape

Записи прошлого года

Весной 2026 года в CS Space Клуб технологий и науки

Курс фотограмметрии



Данные предоставил Stéphane Prodent

Agisoft Metashape