

Вычисления на видеокартах

Лекция 12 - Nanite (Unreal Engine 5)

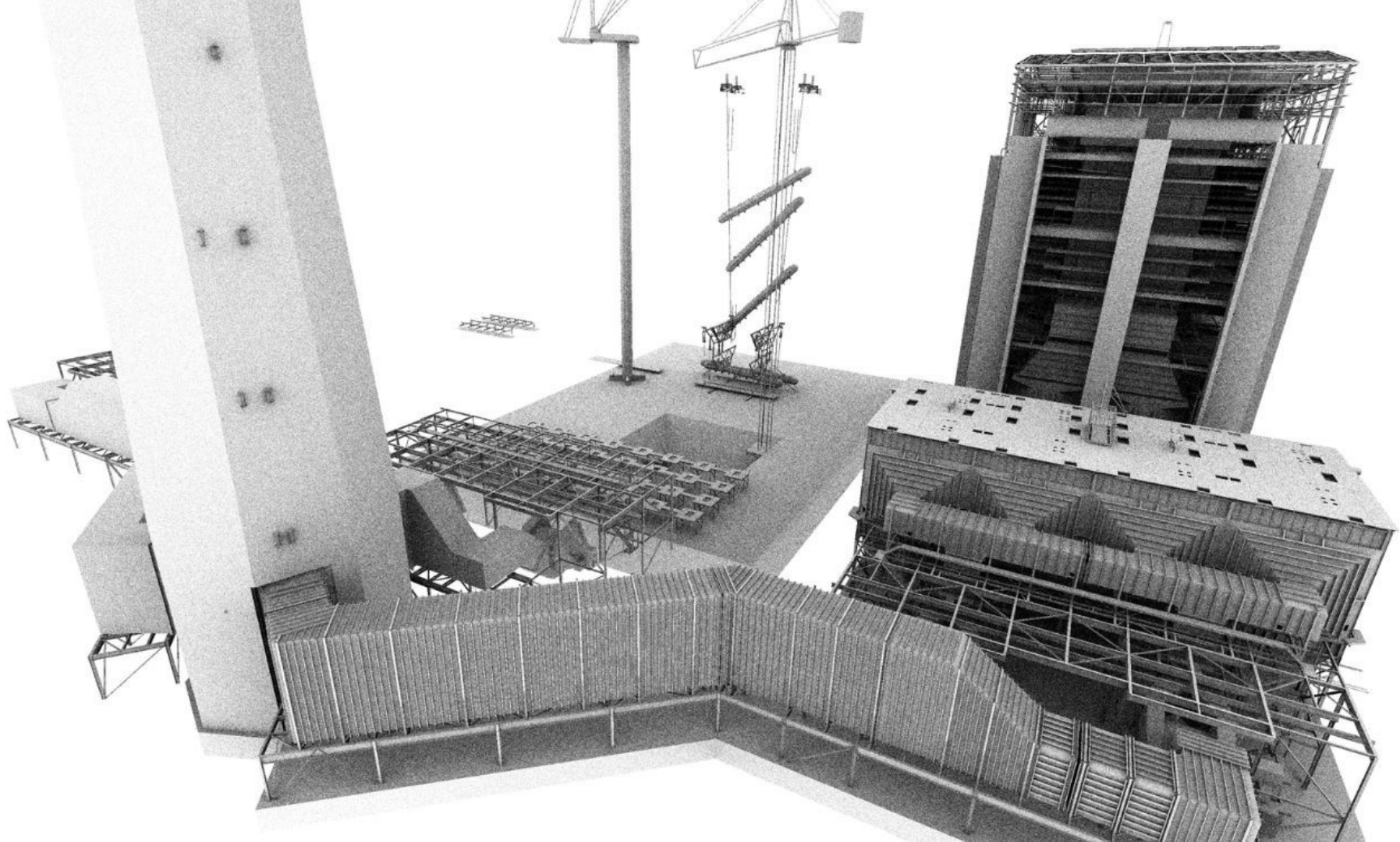
- Виртуальная текстура и геометрия
- QSlim упрощение геометрии
- Hierarchical Z Buffer - HZB
- Deferred rendering
- Rigging, деревья



 @UnicornGlade
 @PolarNick239
 polarnick239@gmail.com

 Николай Полярный

Activate Ubuntu
Go to Settings to activate Ubuntu.





H2E
Team

1
MO
am
ITMO
Team

3
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 Korniyakov

- 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

Nanite

A Deep Dive

Brian Karis
Rune Stubbe
Graham Wihlidal



SIGGRAPH 2021

ADVANCES IN REAL-TIME RENDERING IN GAMES course



UNREAL
ENGINE



Snayru 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



Model Inspector

WIREFRAME

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



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

Albedo/Diffuse map



Activate Windows
Go to Settings to activate Windows.

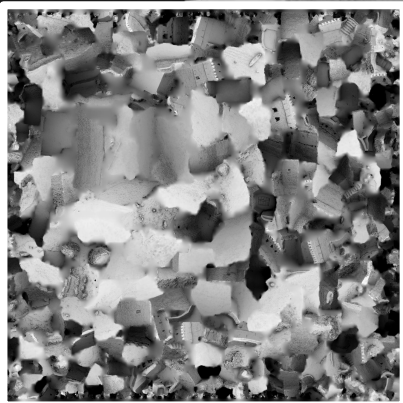
Albedo/Diffuse map



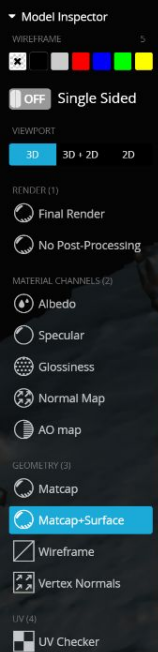
Как добавить реалистичности?



Ambient Occlusion map (AO map)

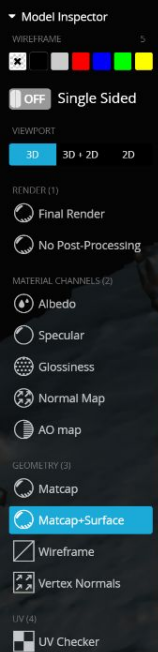


Activate Wind
Go to Settings to activate Wind



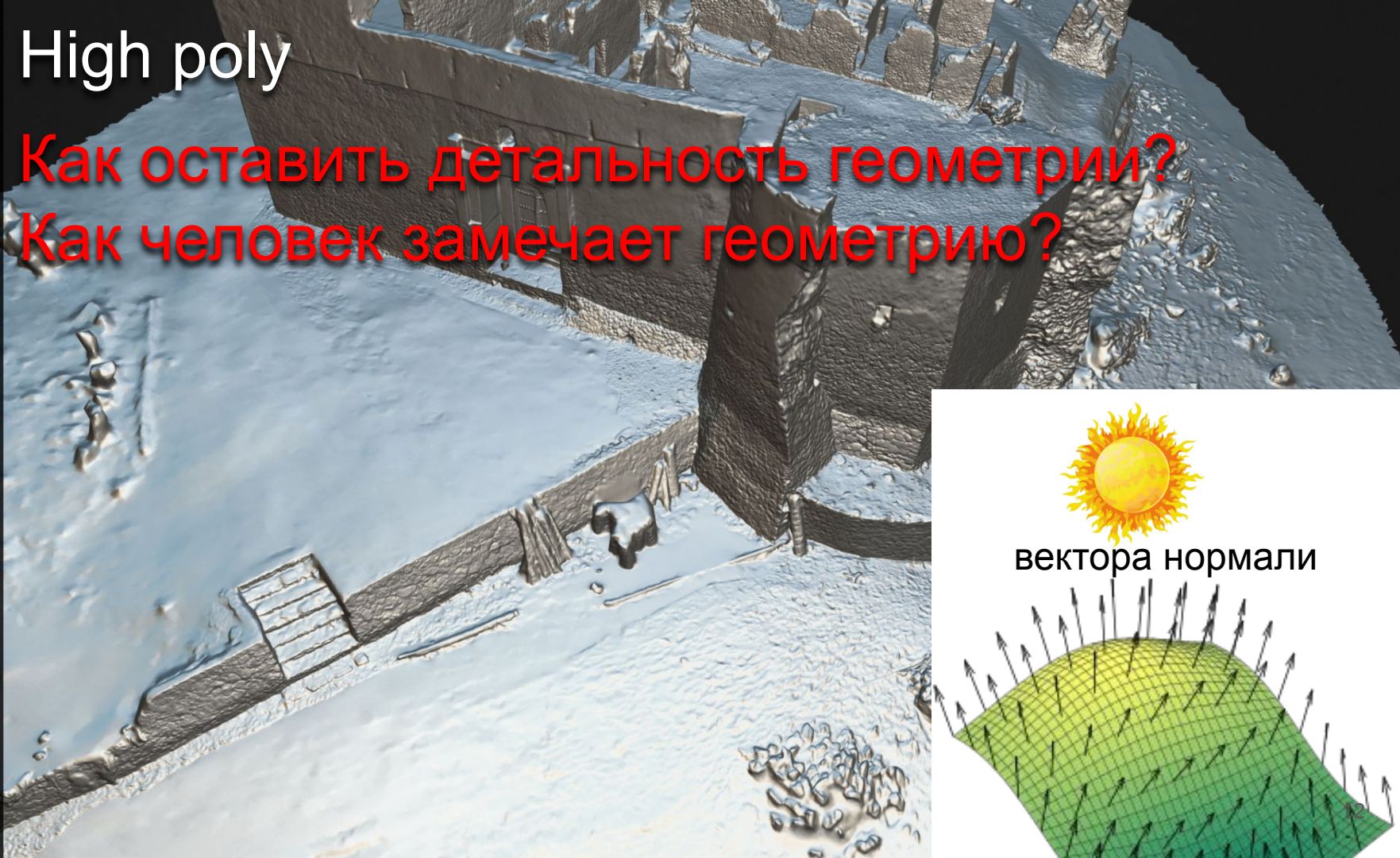
High poly

Как оставить детальность геометрии?
Как человек замечает геометрию?

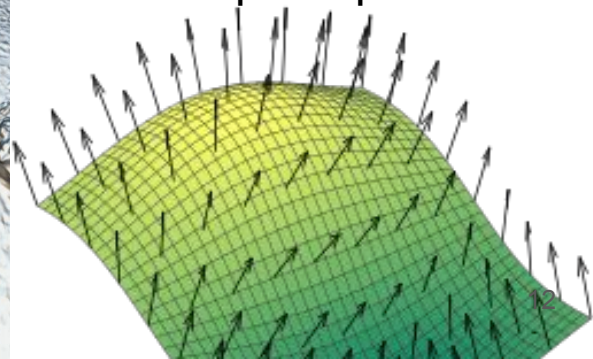


High poly

Как оставить детальность геометрии?
Как человек замечает геометрию?



вектора нормали





OFF Single Sided

3D 3D + 2D 2D

Final Render

No Post-Processing

Albedo

Specular

Glossiness

Normal Map

AO map

Matcap

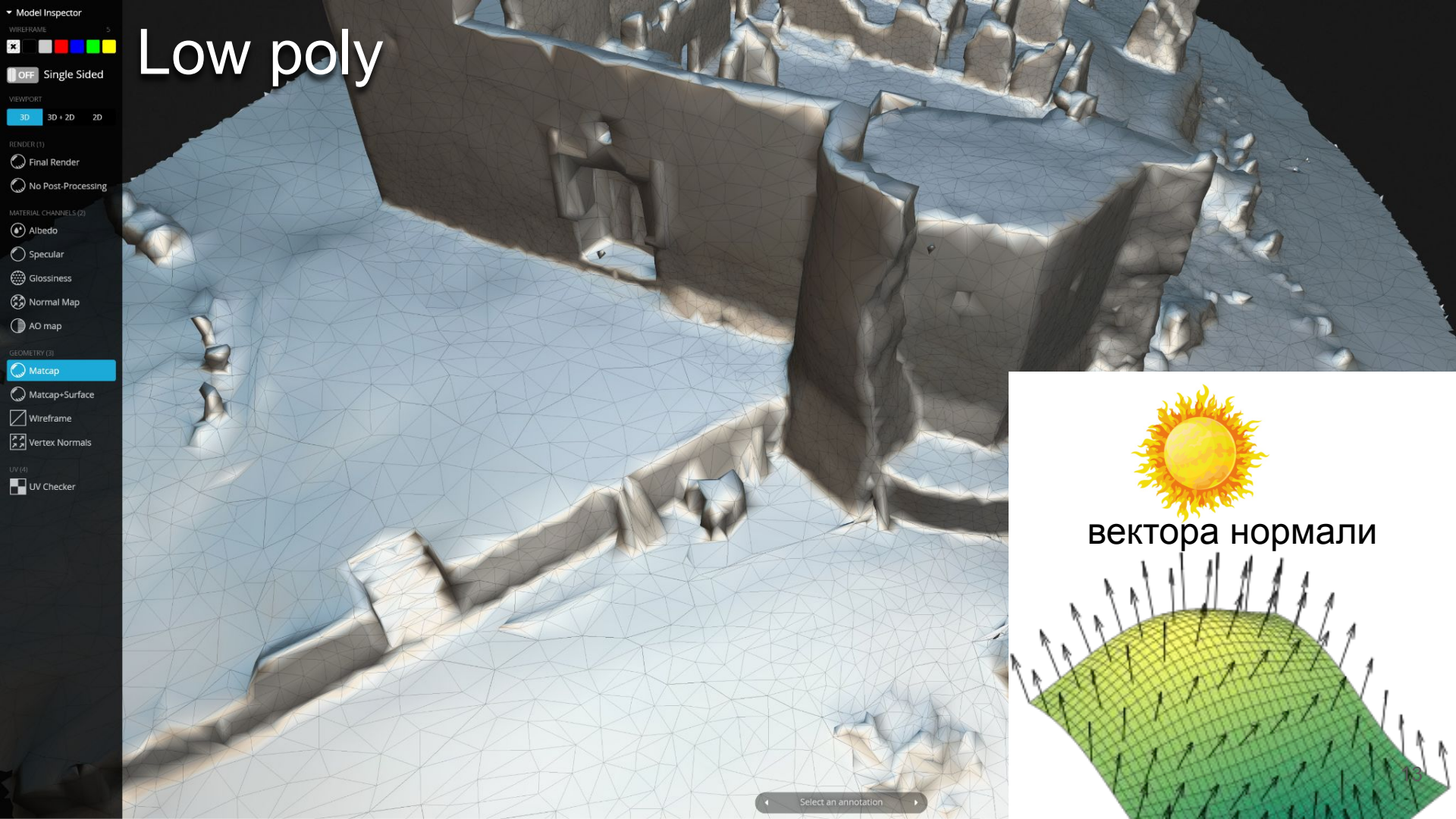
Matcap+Surface

Wireframe

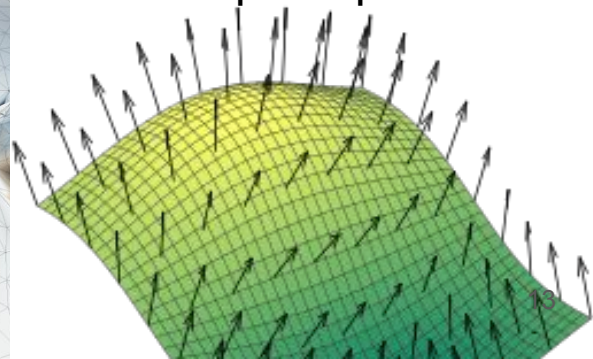
Vertex Normals

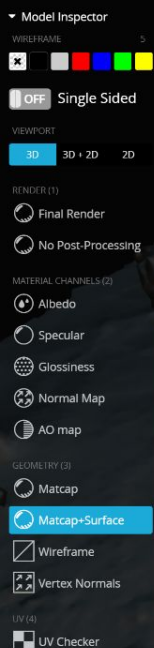
UV Checker

Low poly

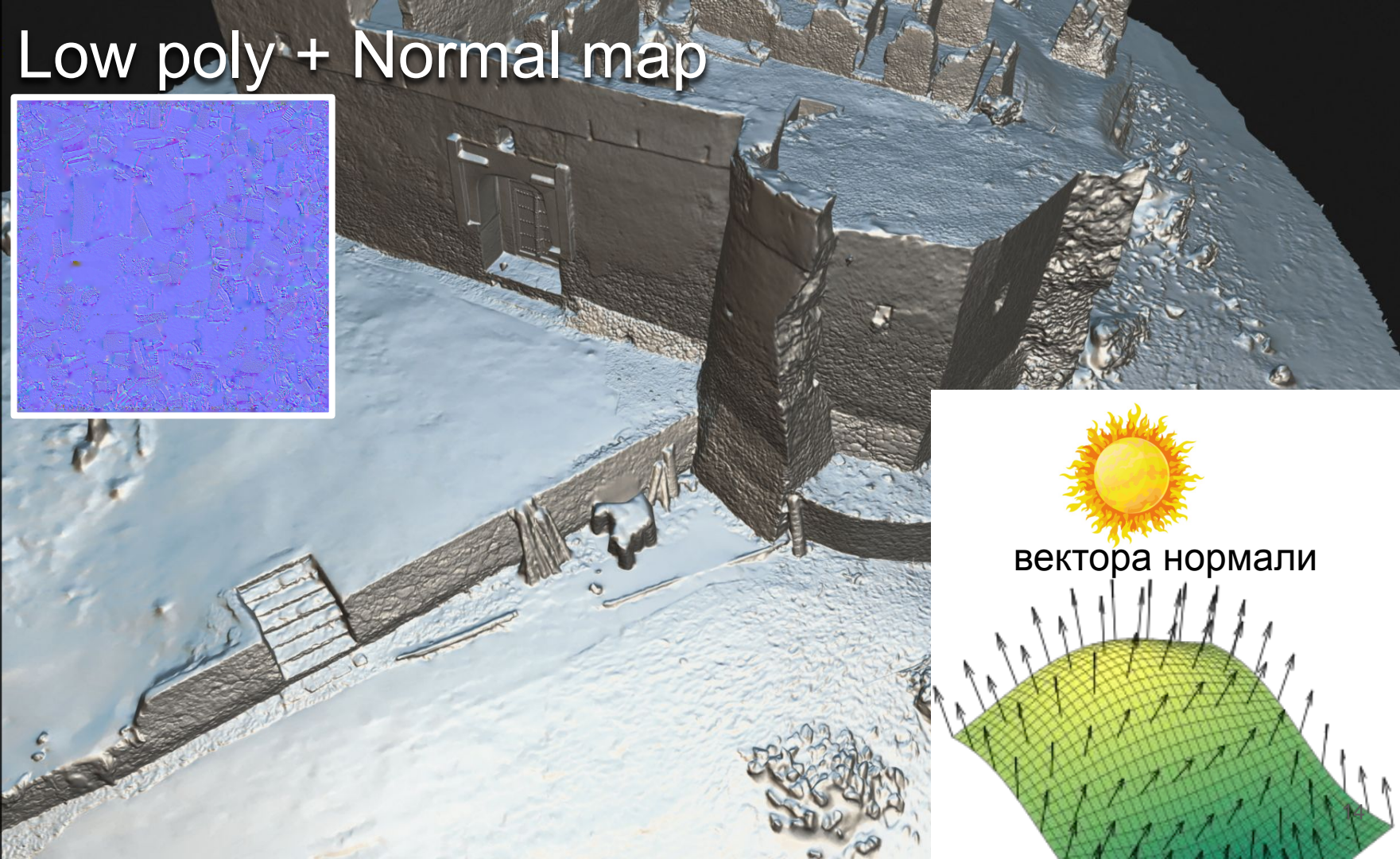
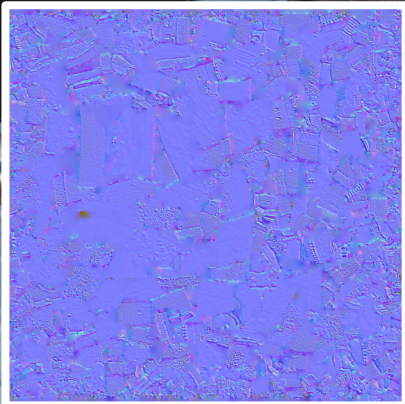


вектора нормали

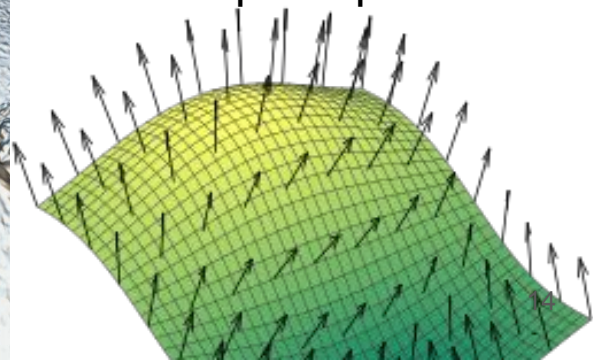




Low poly + Normal map



вектора нормали



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

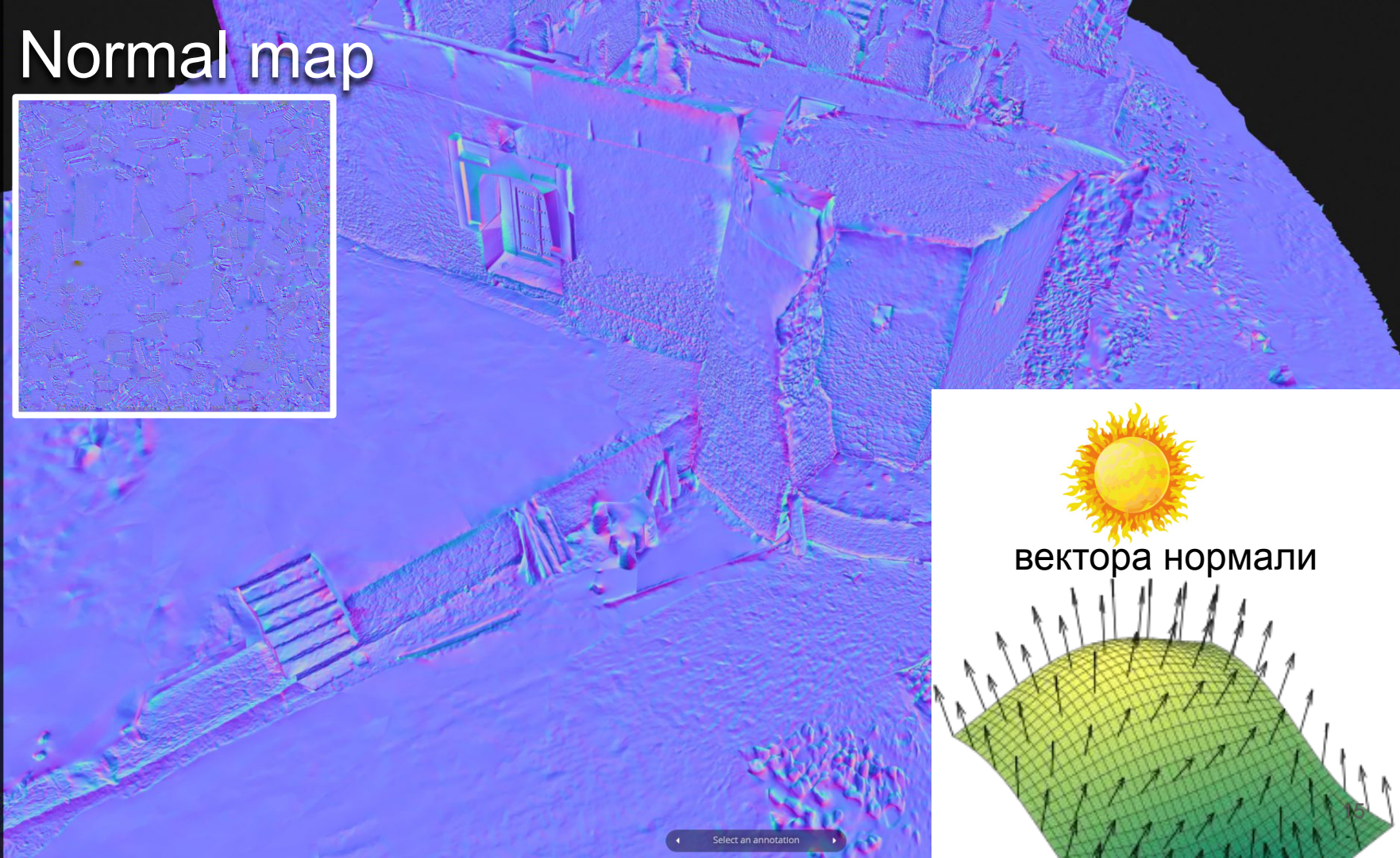
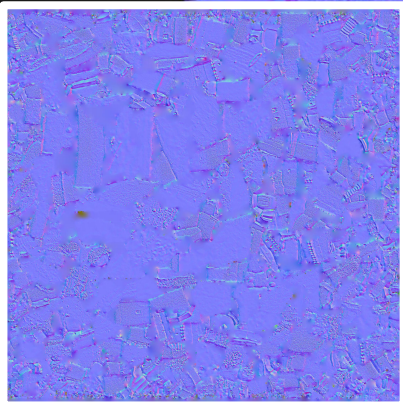
Wireframe

Vertex Normals

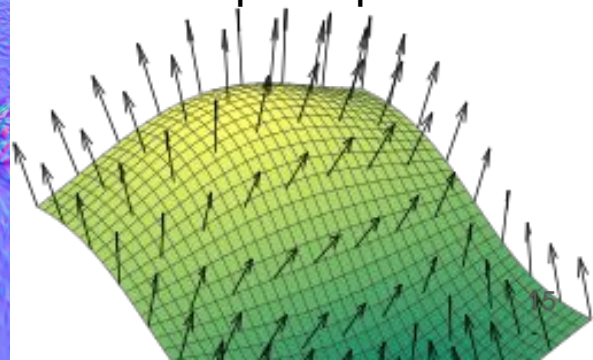
UV (4)

UV Checker

Normal map



вектора нормали



Select an annotation



Low poly

В чем задача 3D моделиера?



Low poly

В чем задача 3D моделилера?

Как минимум от 3D моделилера требуется:

- распределить доступный бюджет треугольников
- не потерять в Low poly важные детали - шпиль/копье
- тонко чувствовать что бросается игроку в глаза

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 - 16K x 16K

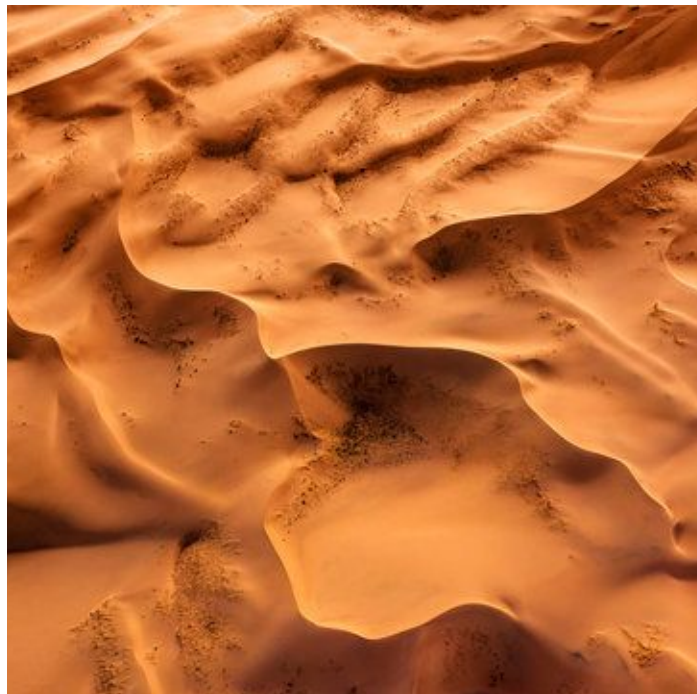


Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

Нельзя загрузить целиком в VRAM! Что делать?

В те времена - 16K x 16K



Виртуальные текстуры (virtual texture, texture streaming)

Нельзя загрузить целиком в VRAM! Что делать?

Мир - 2D текстура

В те времена - 16K x 16K

Игрок видит мир вокруг:



framebuffer

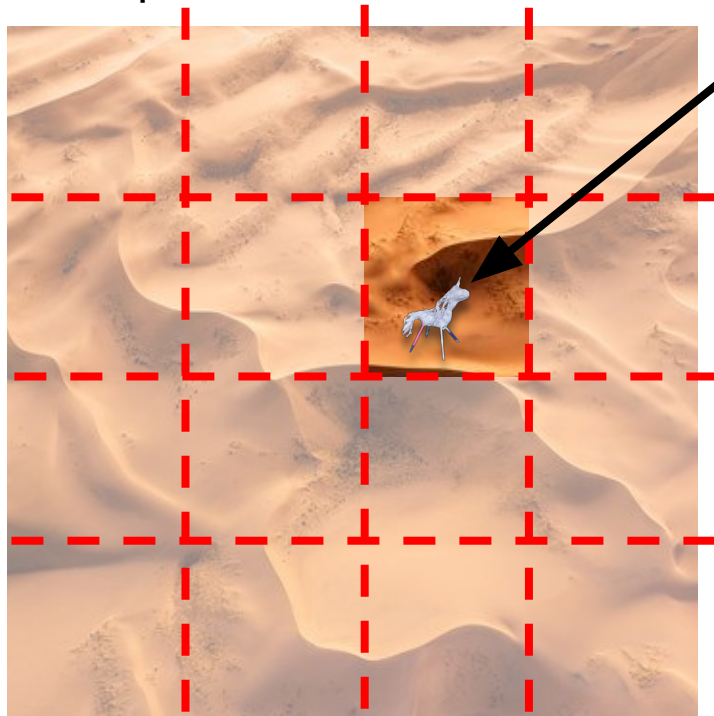
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K

Нельзя загрузить целиком в VRAM! Что делать?

Игрок видит мир вокруг:



framebuffer

Может нарежем на куски, один кусок влезет VRAM?

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

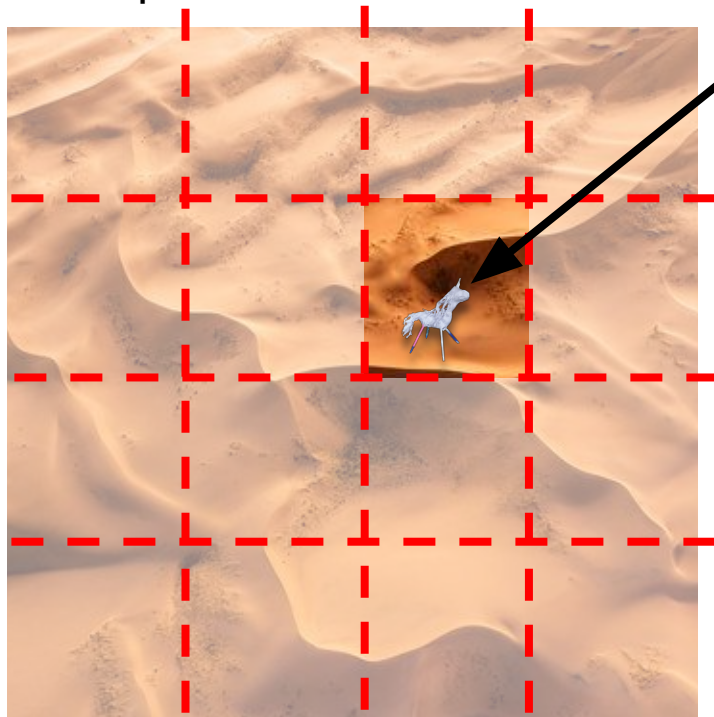
В те времена - 16K x 16K

Нельзя загрузить целиком в VRAM! Что делать?

Игрок видит мир вокруг:



framebuffer



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)

Мир - 2D текстура

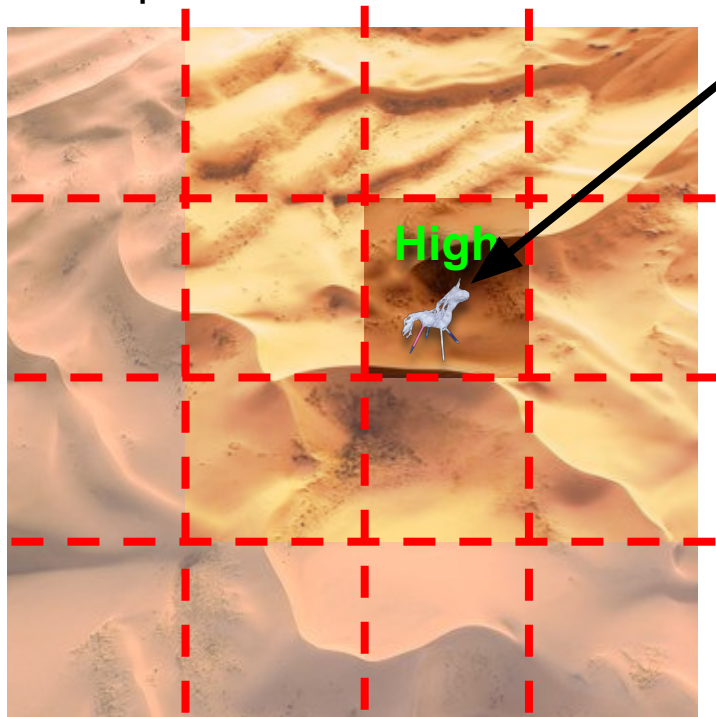
В те времена - 16K x 16K

Нельзя загрузить целиком в VRAM! Что делать?

Игрок видит мир вокруг:



framebuffer



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

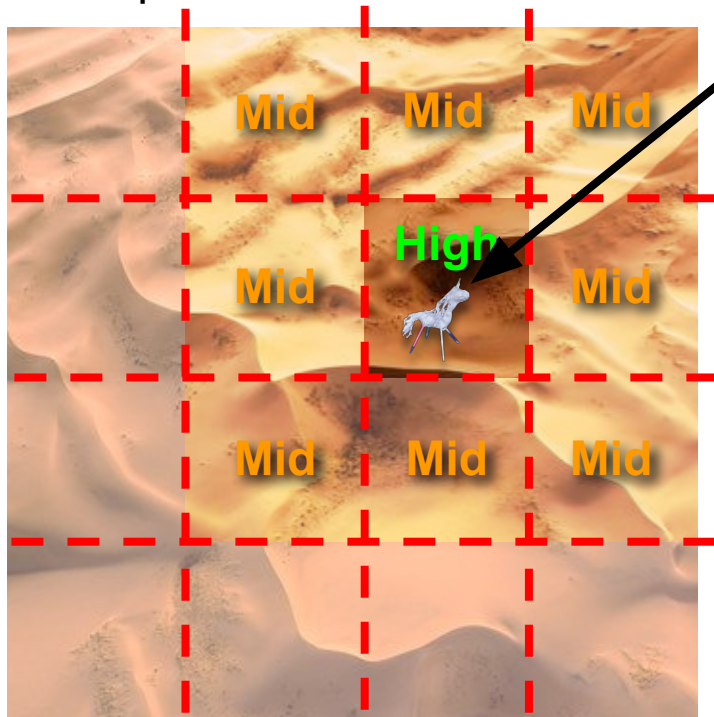
В те времена - 16K x 16K

Нельзя загрузить целиком в VRAM! Что делать?

Игрок видит мир вокруг:



framebuffer



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

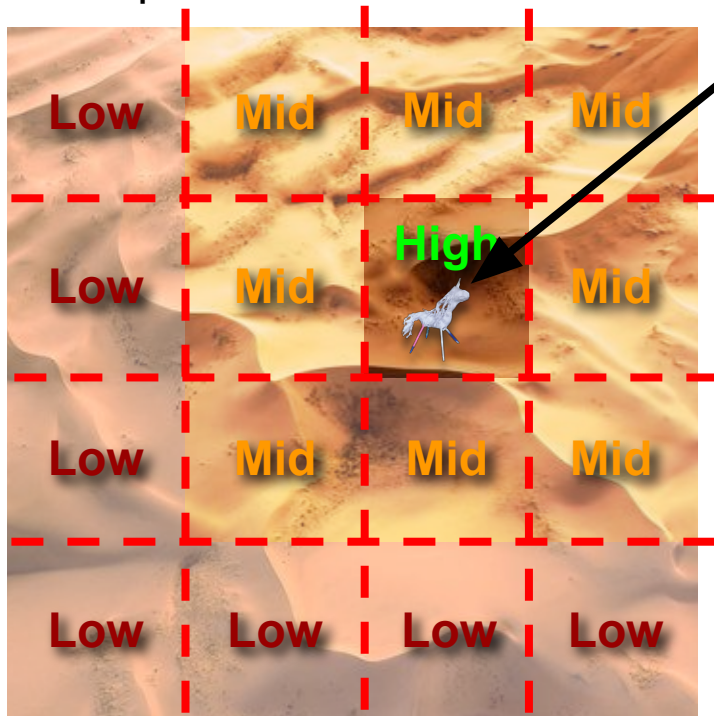
В те времена - 16K x 16K

Нельзя загрузить целиком в VRAM! Что делать?

Игрок видит мир вокруг:



framebuffer



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

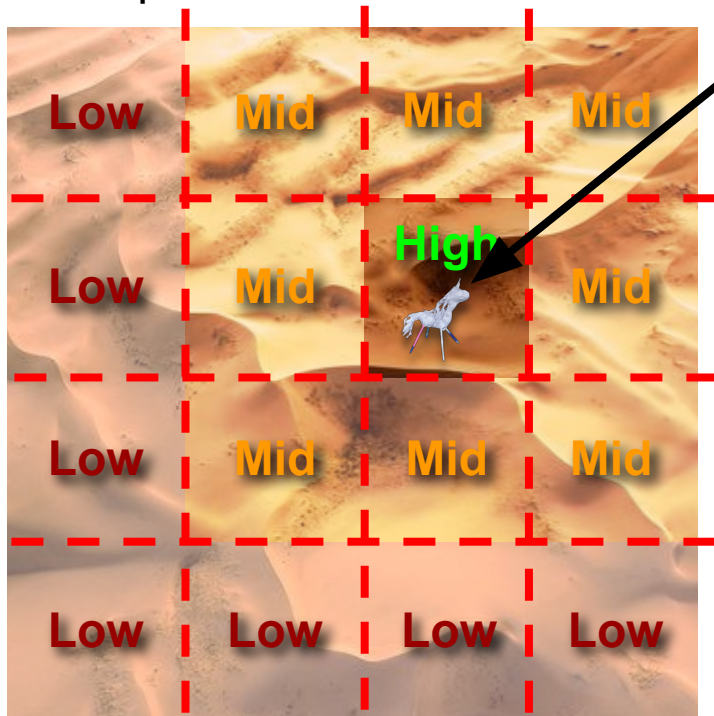
В те времена - 16K x 16K

Нельзя загрузить целиком в VRAM! Что делать?

Игрок видит мир вокруг:



framebuffer



Может нарежем на куски, один кусок влезет VRAM?

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

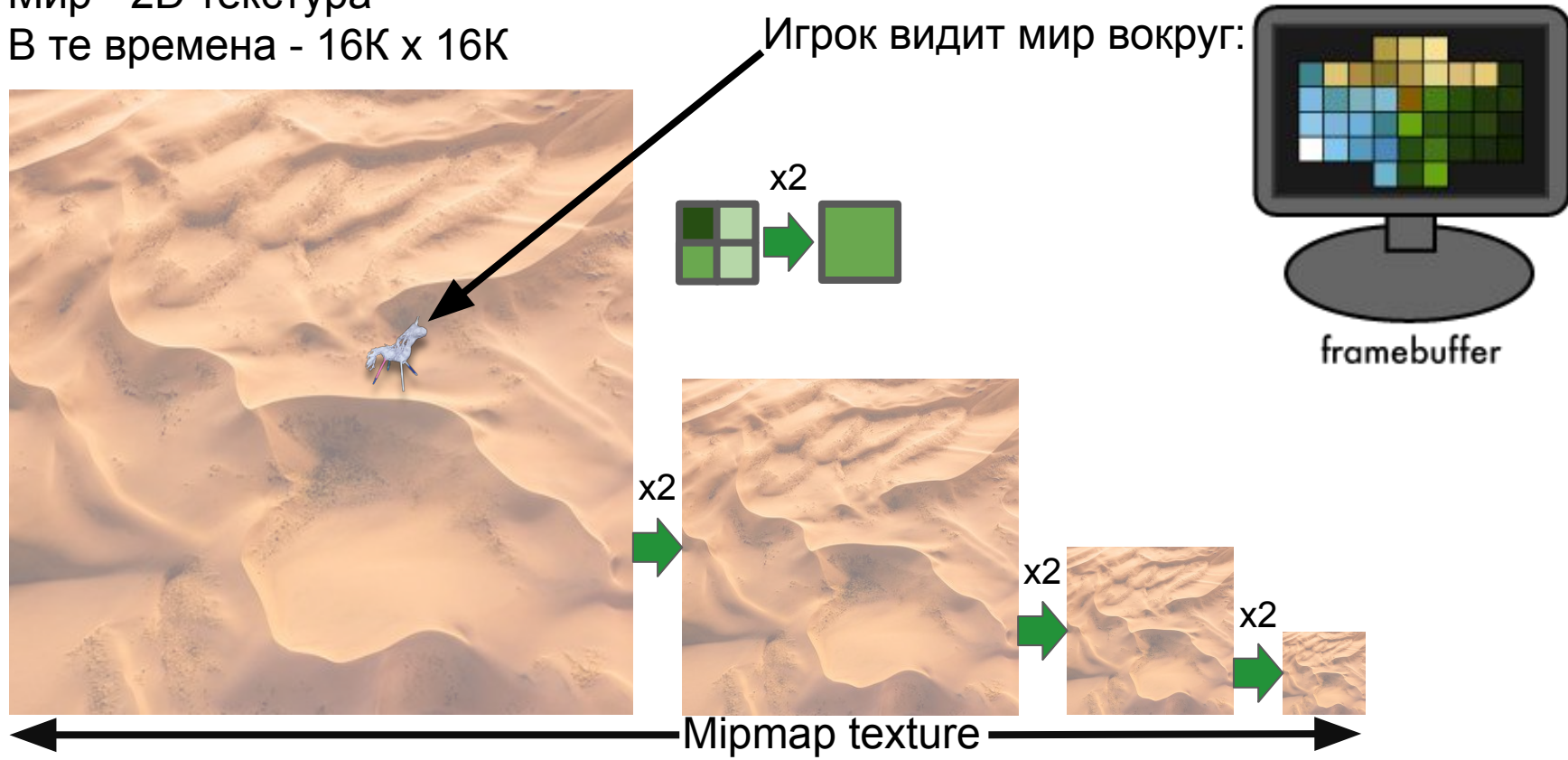
Может нарежем на куски + LODs (Levels of Details)?

- 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

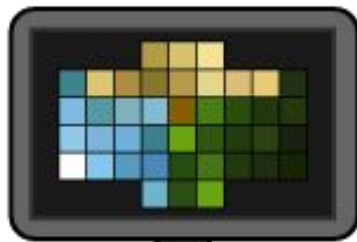
В те времена - 16K x 16K



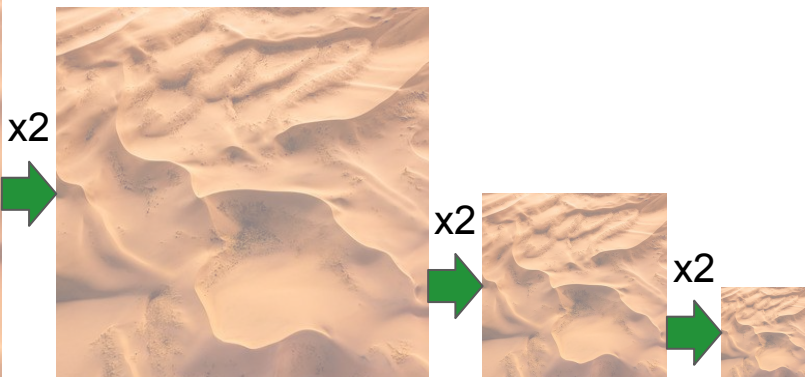
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



Pass 1 framebuffer:
uv request +
mipmap level

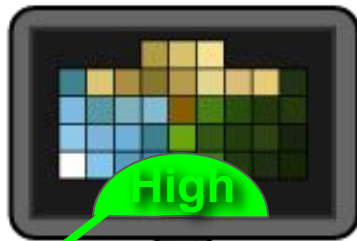
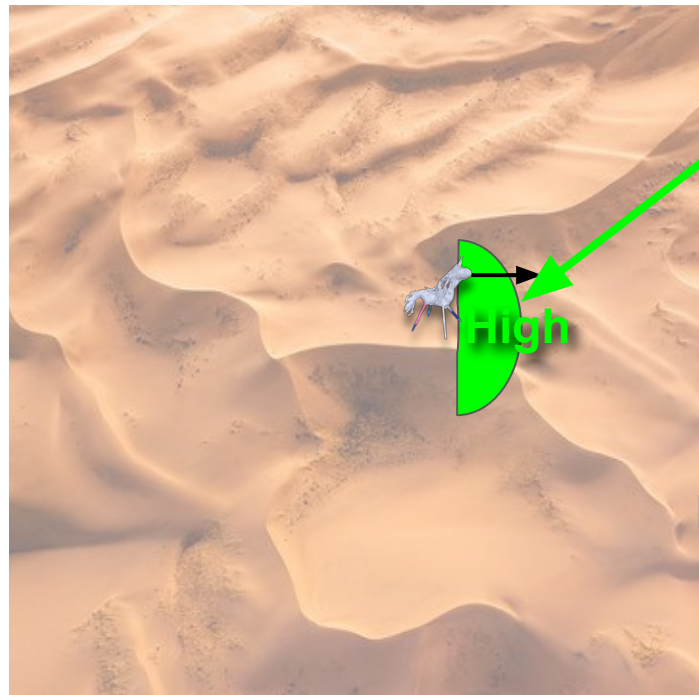


Mipmap texture

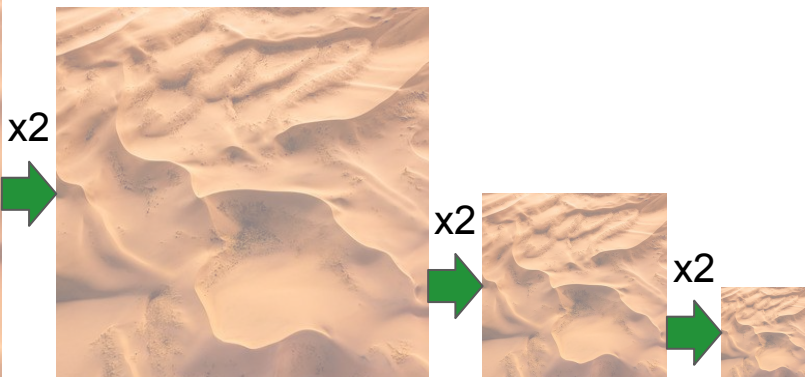
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



Pass 1 framebuffer:
uv request +
mipmap level

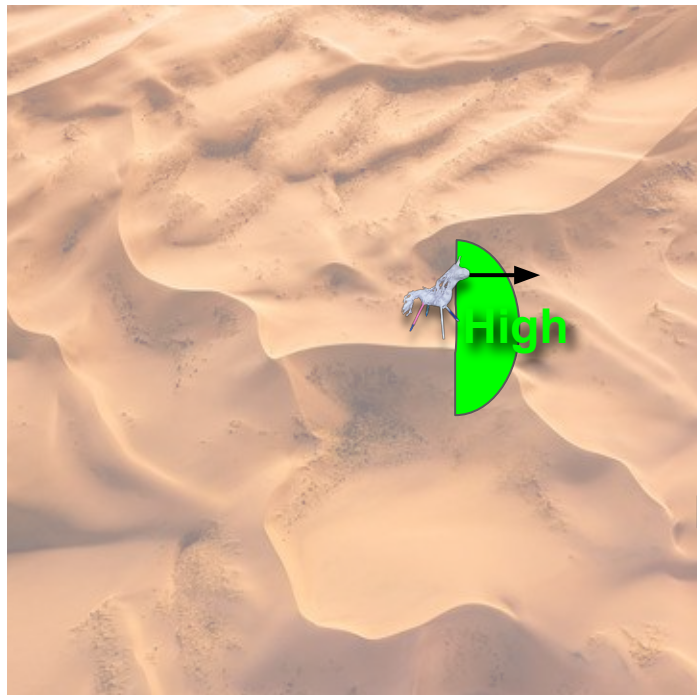


Mipmap texture

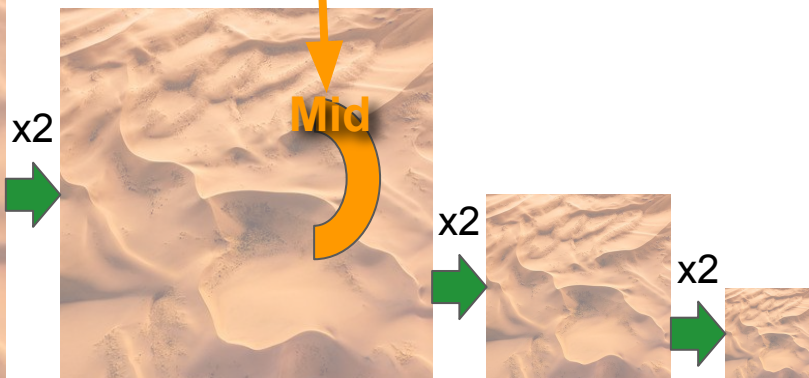
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



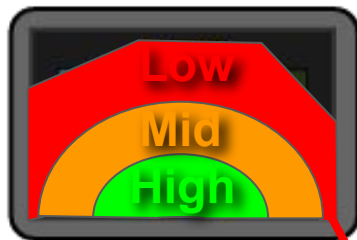
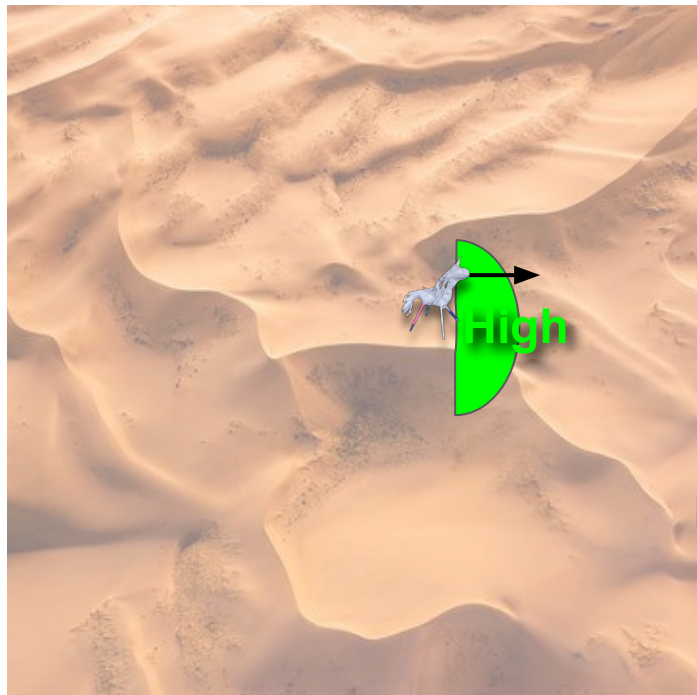
Pass 1 framebuffer:
uv request +
mipmap level



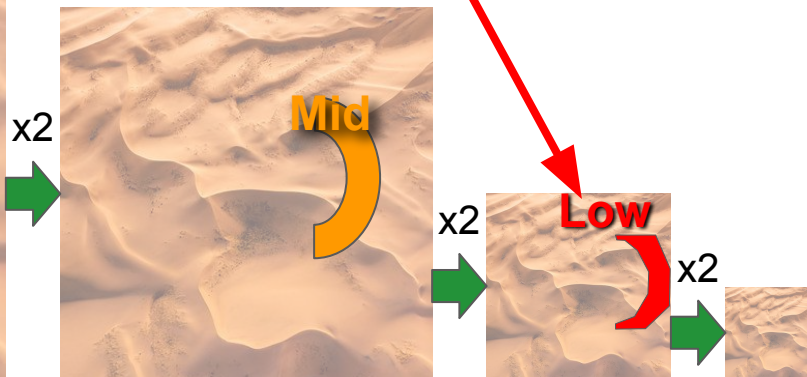
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



Pass 1 framebuffer:
uv request +
mipmap level

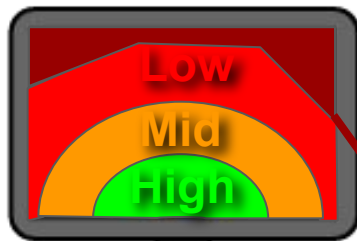
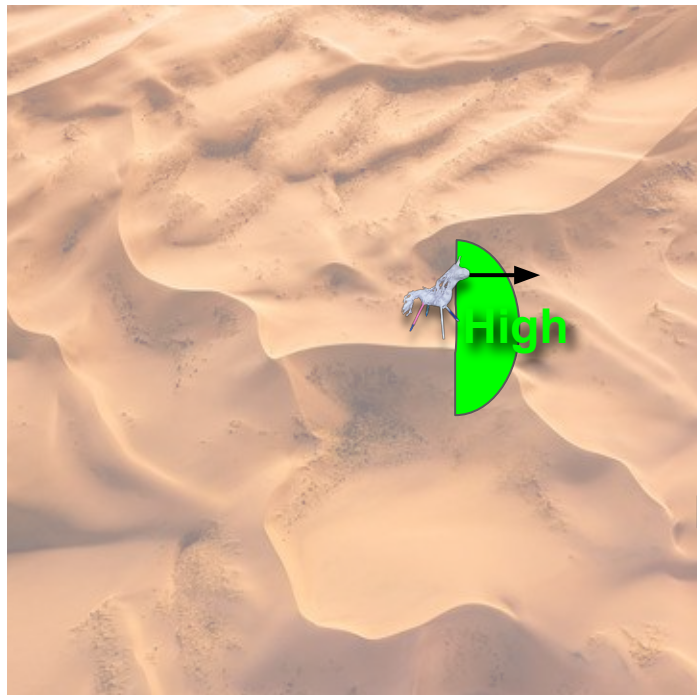


Mipmap texture

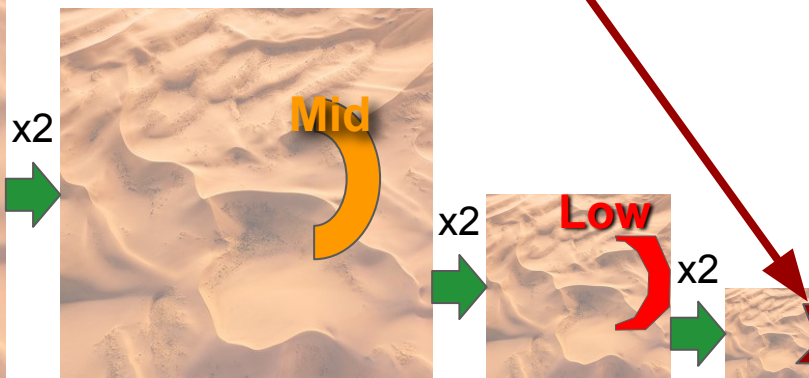
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



Pass 1 framebuffer:
uv request +
mipmap level

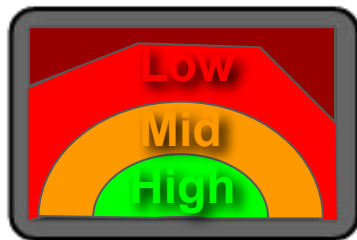
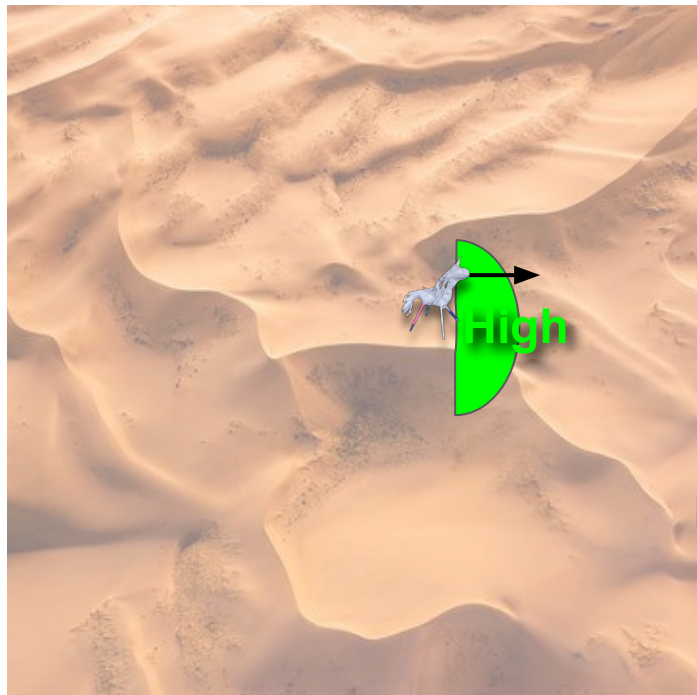


Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K

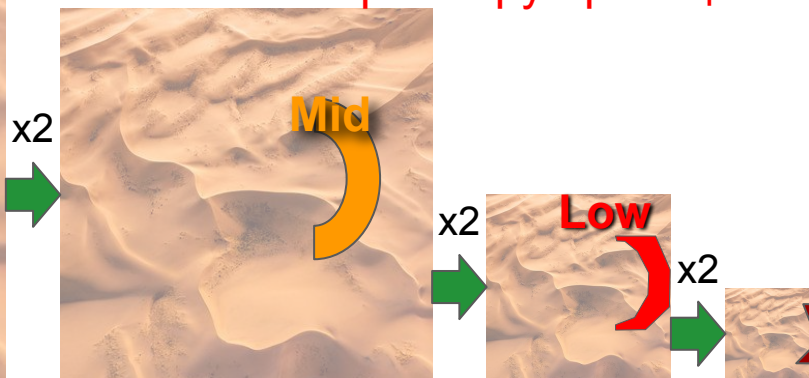


Как определить детальность?
(mipmap level)

Pass 1 framebuffer:

uv request +
mipmap level

downscale ~ размеру проекции в экране

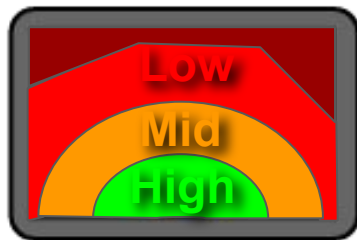
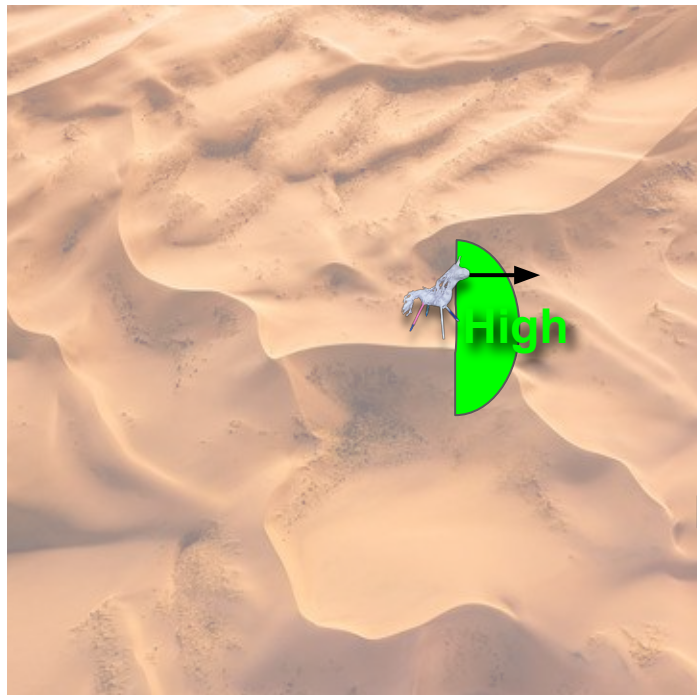


Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



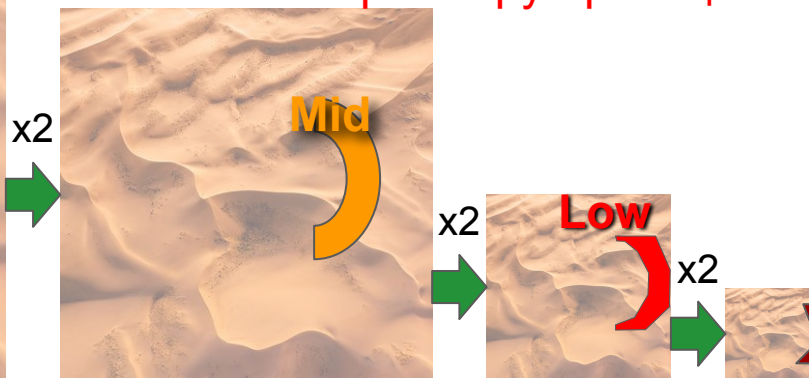
Как определить детальность?
(mipmap level)

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

Pass 1 framebuffer:

uv request +
mipmap level

downscale ~ размеру проекции в экране

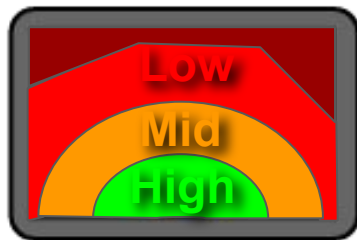
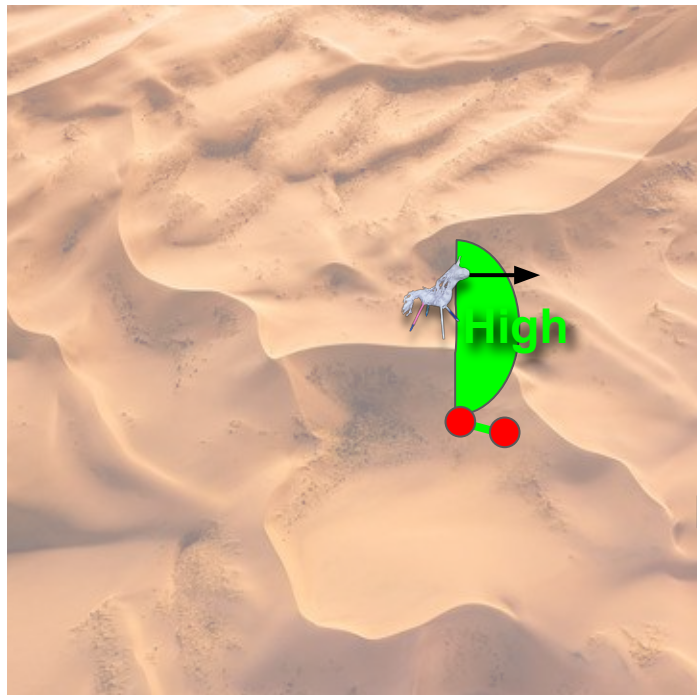


Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



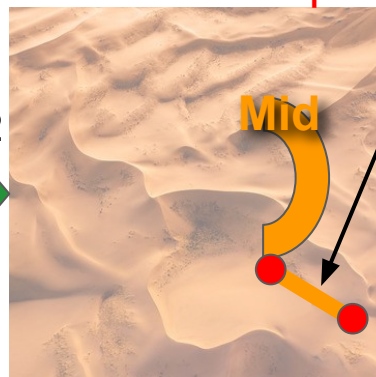
Как определить детальность?
(mipmap level)

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

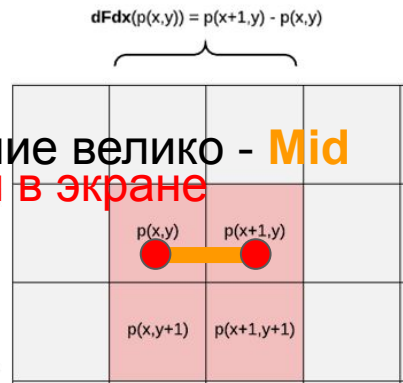
Pass 1 framebuffer:

uv request +
mipmap level

если UV расстояние велико - Mid
downscale ~ размеру проекции в экране



$$dFdy(p(x,y)) = p(x,y+1) - p(x,y)$$

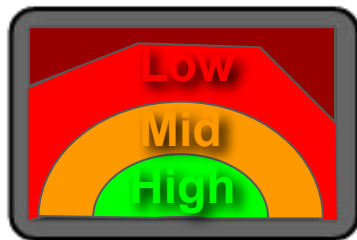
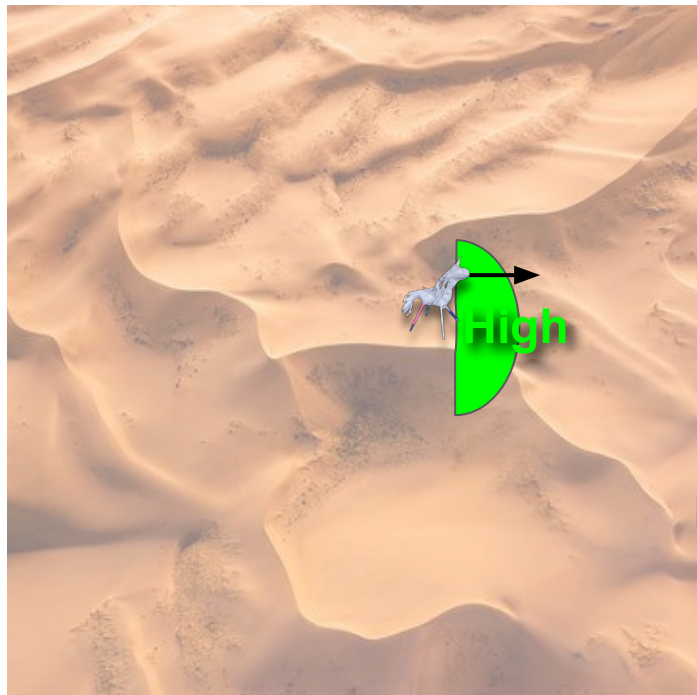


Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

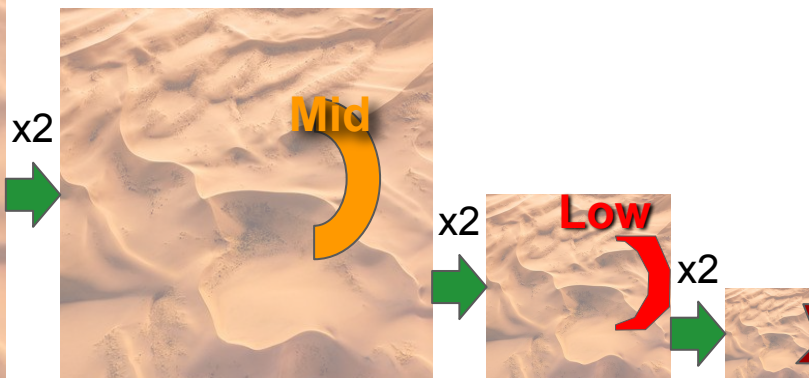
Мир - 2D текстура

В те времена - 16K x 16K



Обязательно ли это
связные сектора?

Pass 1 framebuffer:
uv request +
mipmap level

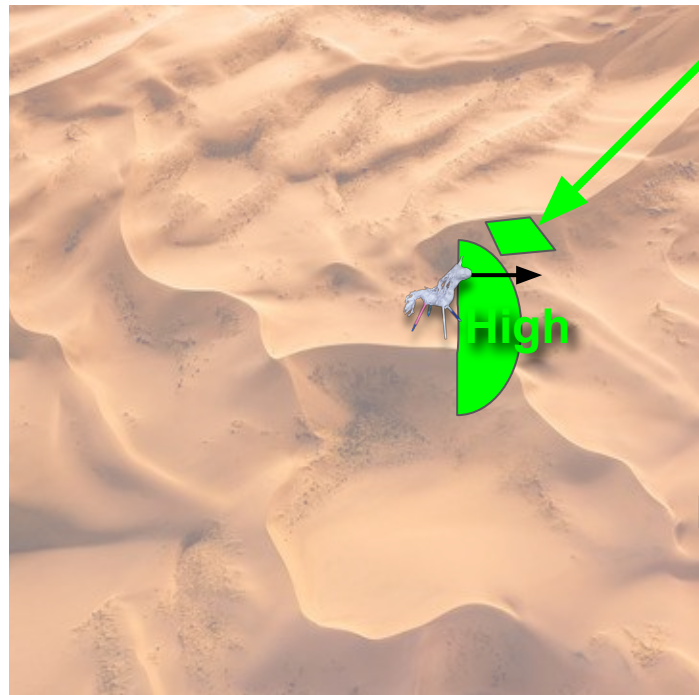


Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

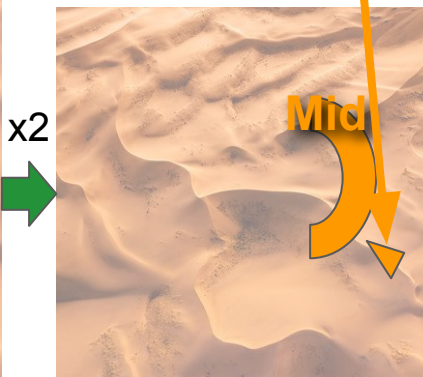
Мир - 2D текстура

В те времена - 16K x 16K



Обязательно ли это
связные сектора?

Pass 1 framebuffer:
uv request +
mipmap level

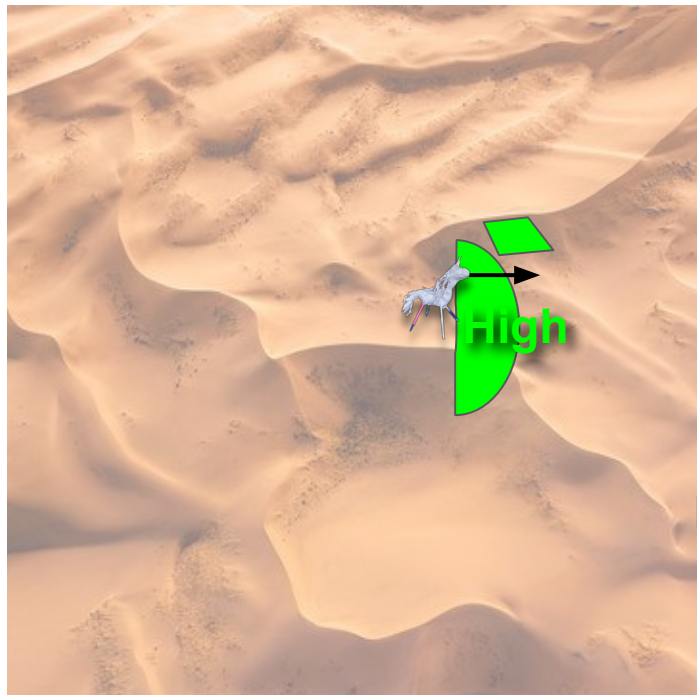


Mipmap texture

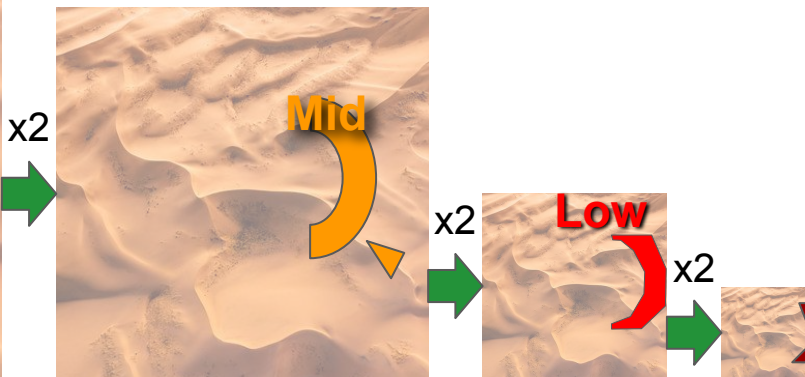
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



Pass 1 framebuffer:
uv request +
mipmap level

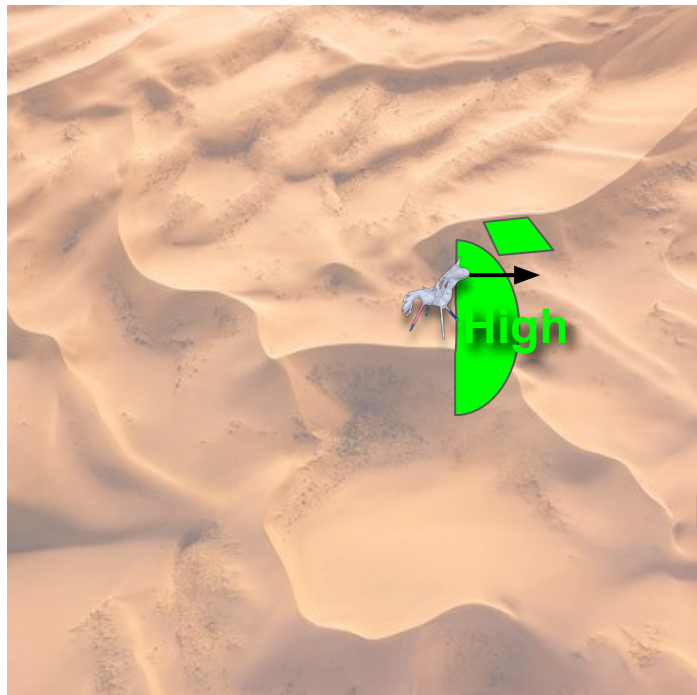


Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K

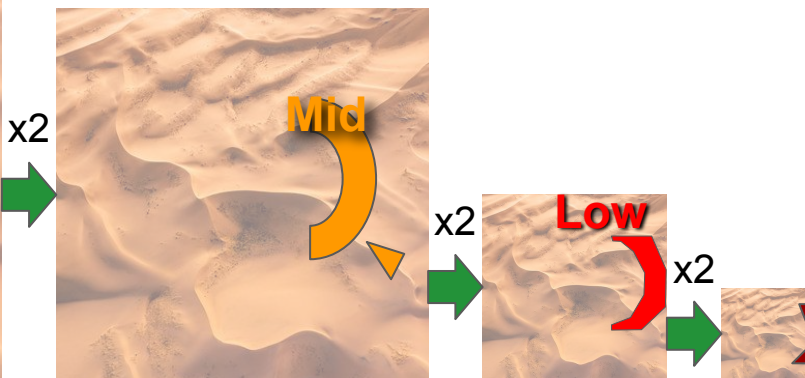


Pass 1 framebuffer:

uv request +
mipmap level

Pass 2:

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

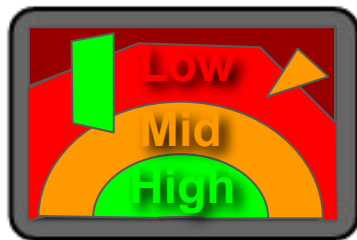
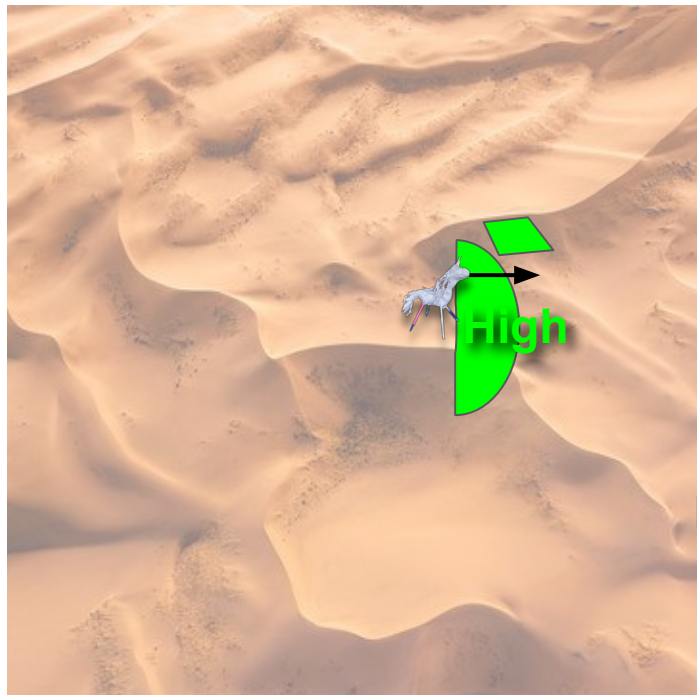


Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



Disk IO (jpg)

RAM (jpg)

RAM (raw)

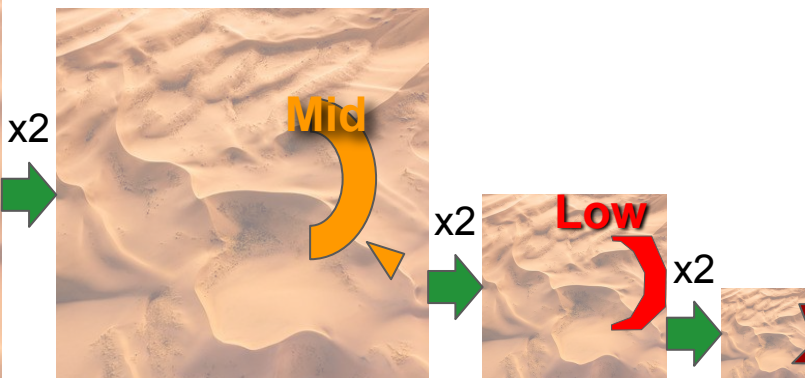
VRAM

Pass 1 framebuffer:

uv request +
mipmap level

Pass 2:

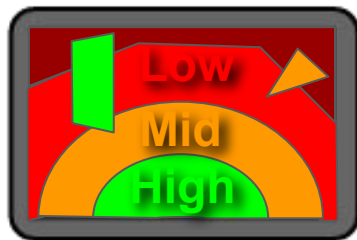
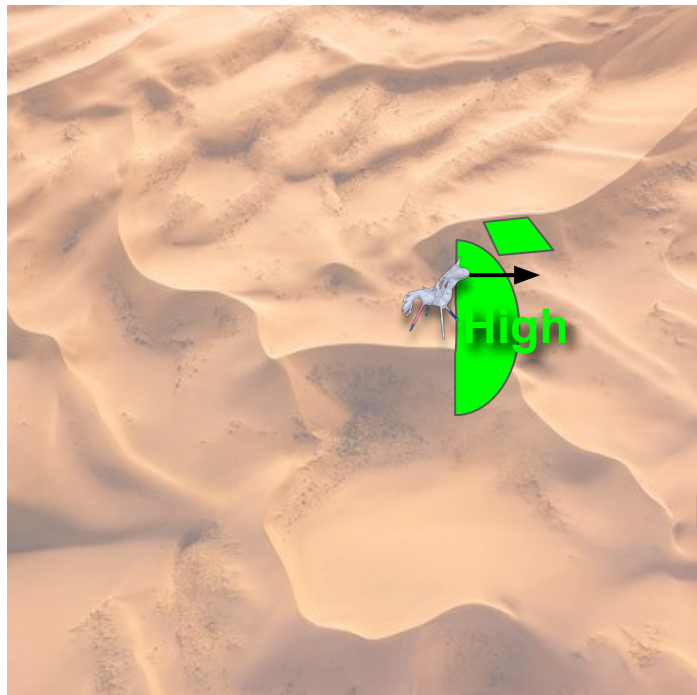
streaming того что
запрошено но еще
не на GPU (VRAM)



Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



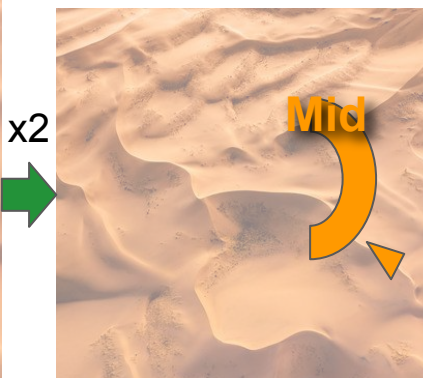
Disk IO (jpg)
RAM (jpg)
RAM (raw)
VRAM

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?**

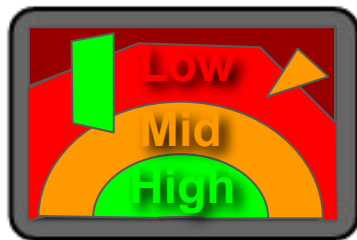
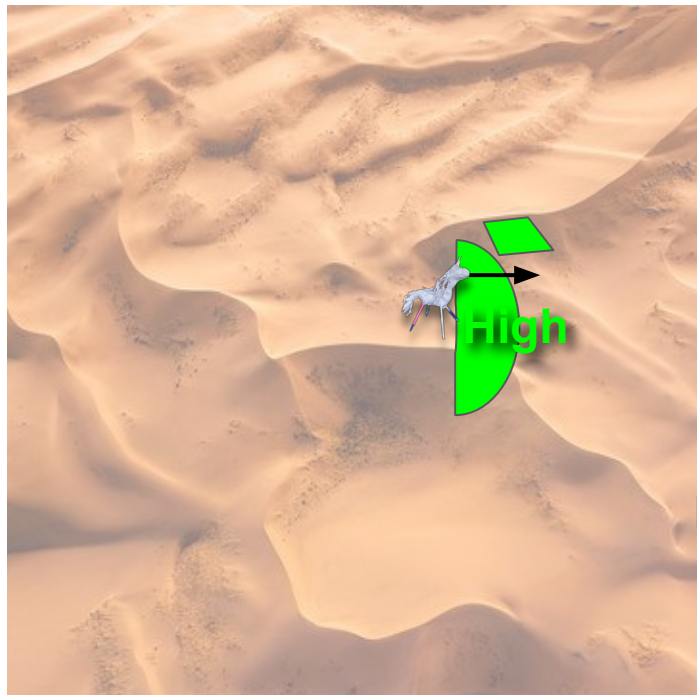


Mipmap texture

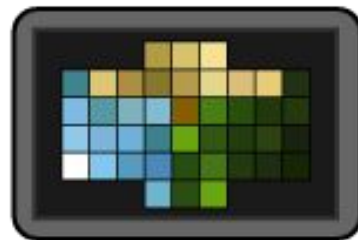
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



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



Pass 1 framebuffer:

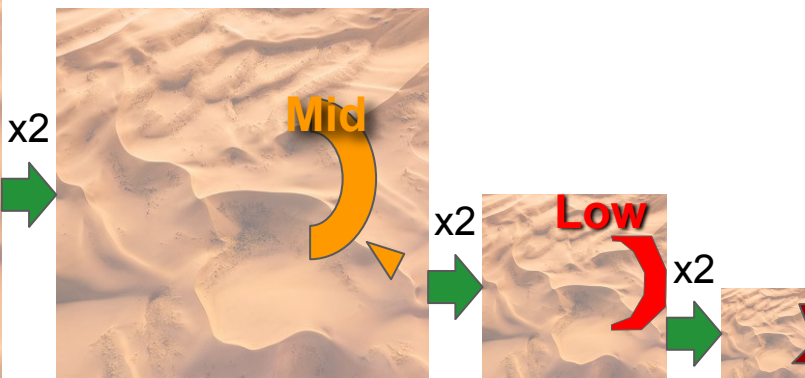
uv request +
mipmap level

Pass 2:

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

Pass 3 framebuffer:

финальный
Render Pass
(все есть на GPU)

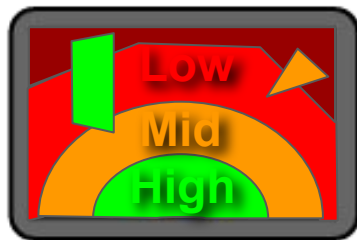
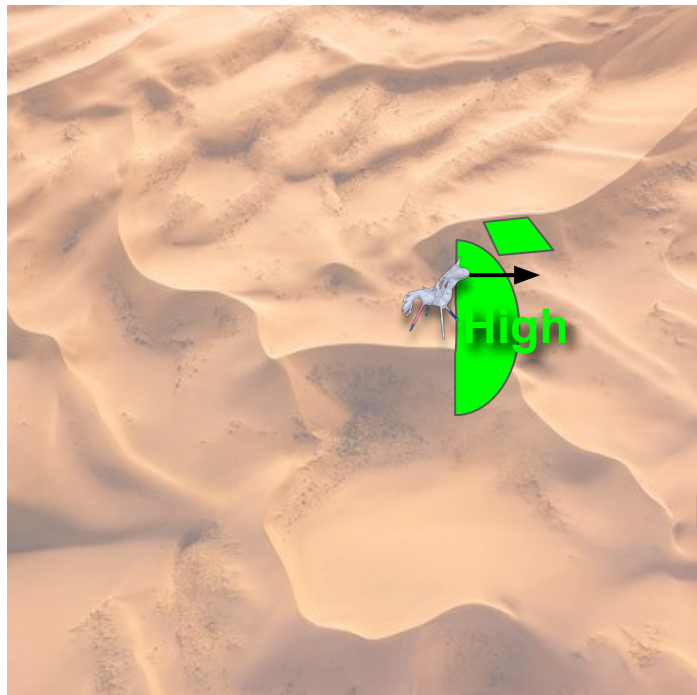


Mipmap texture

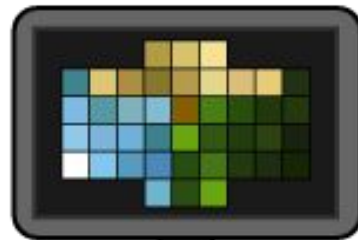
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



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



Pass 1 framebuffer:

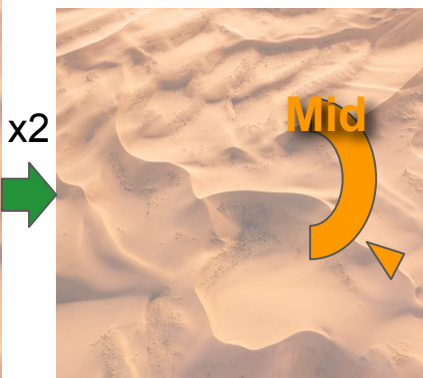
uv request +
mipmap level

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



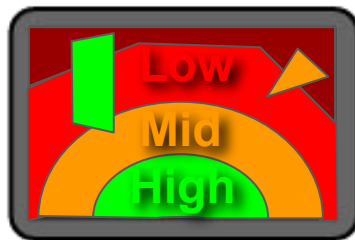
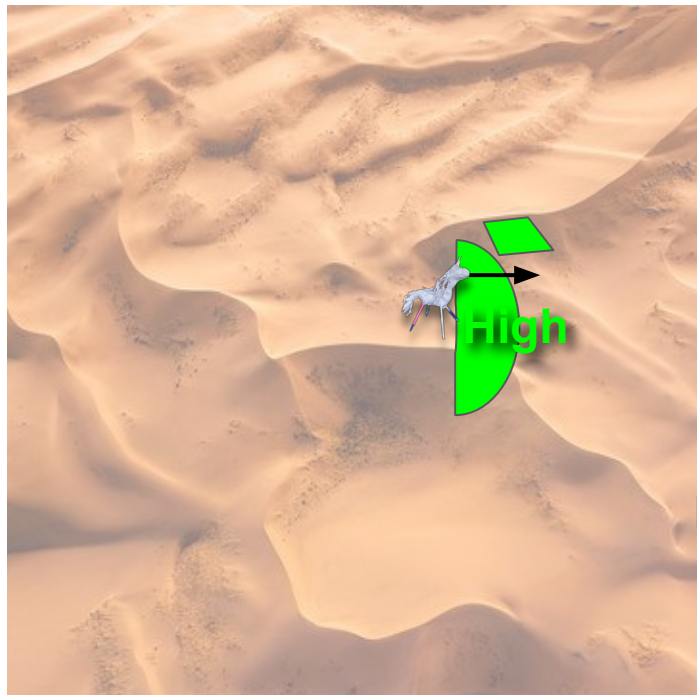
**Есть ли гарантии на время
отрисовки кадра/на FPS?
Не будет ли **stuttering/freeze**?**

Mipmap texture

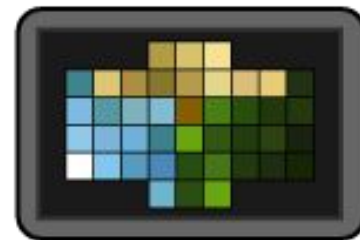
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



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)



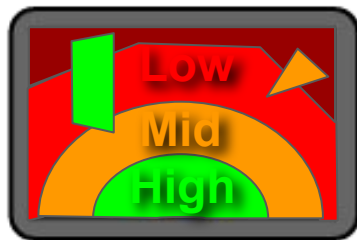
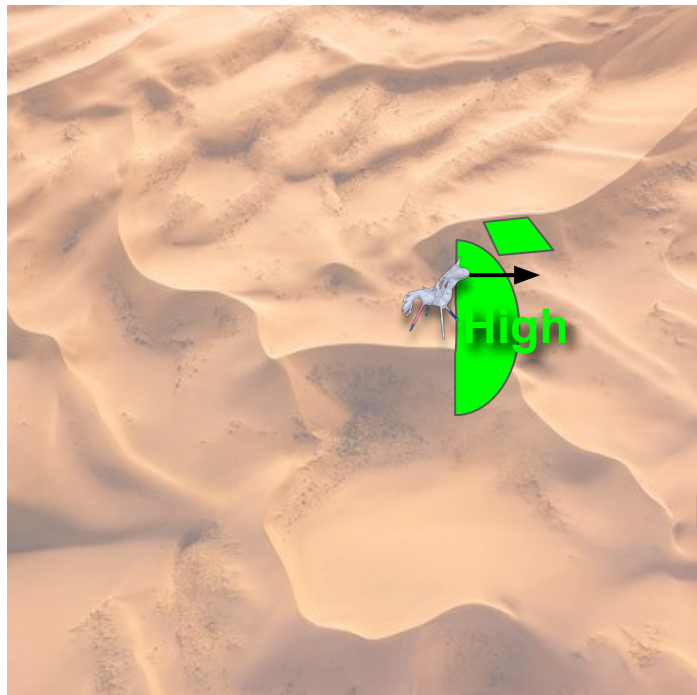
**Есть ли гарантии на время
отрисовки кадра/на FPS?
Не будет ли **stuttering/freeze**?**

Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



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



Pass 1 framebuffer:

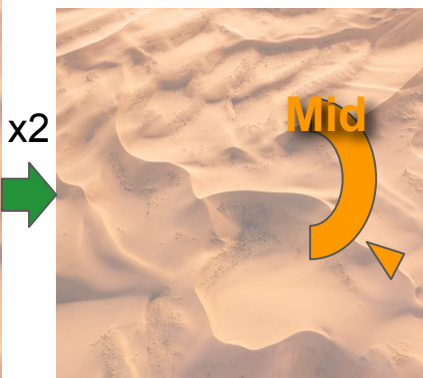
uv request +
mipmap level

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

x2

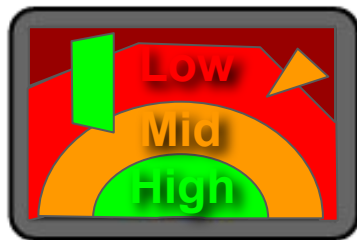
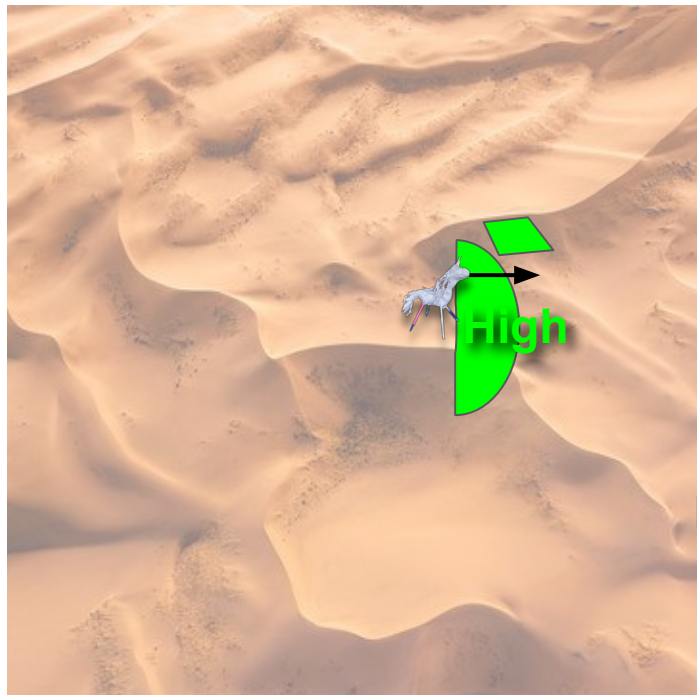
Mipmap texture

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

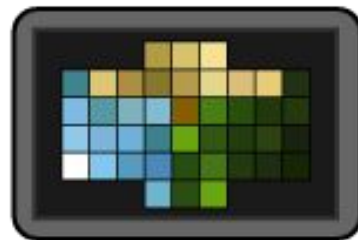
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



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



Pass 1 framebuffer:

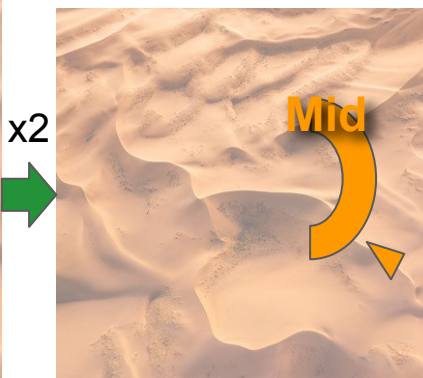
uv request +
mipmap level

Pass 2:

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

Pass 3 framebuffer:

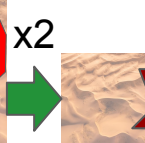
финальный
Render Pass
(все есть на GPU)



x2



x2



x2

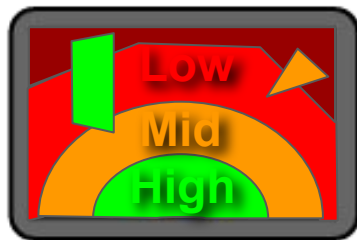
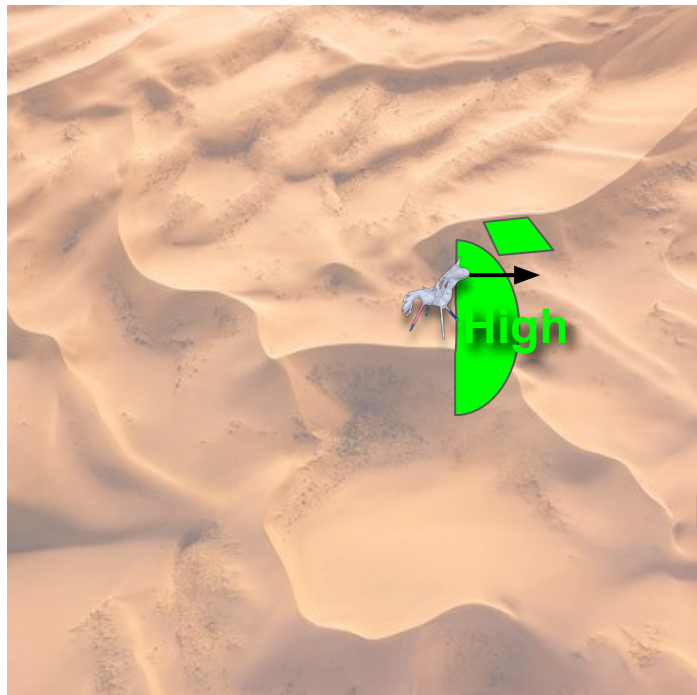
Mipmap texture

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

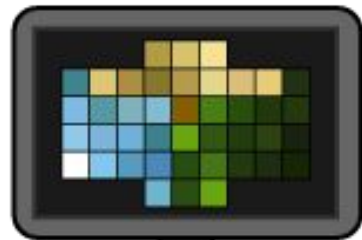
Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



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



Pass 1 framebuffer:

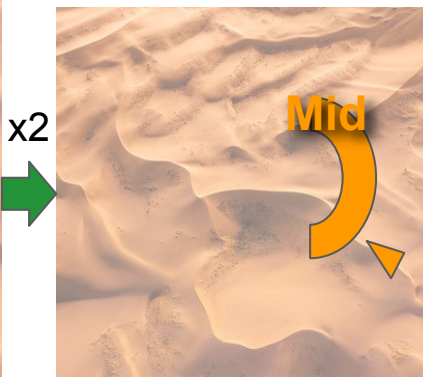
uv request +
mipmap level

Pass 2:

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

Pass 3 framebuffer:

финальный
Render Pass
(все есть на GPU)



Медленное движение =>
хорошая когерентность кадров,
пригодится позже!



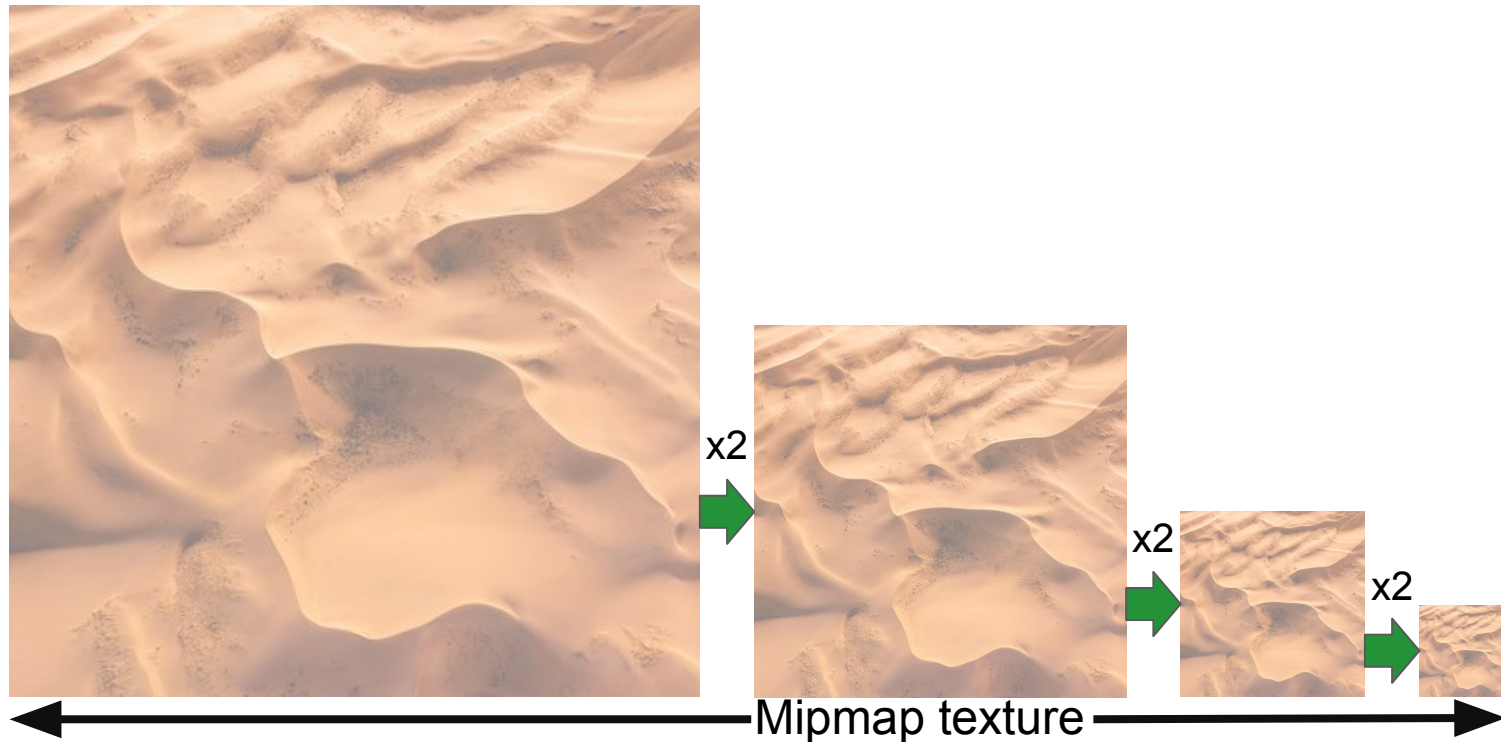
Mipmap texture

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

Мир - 2D текстура

В те времена - 16K x 16K

За счет чего так удачно? Универсален ли подход?



Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K

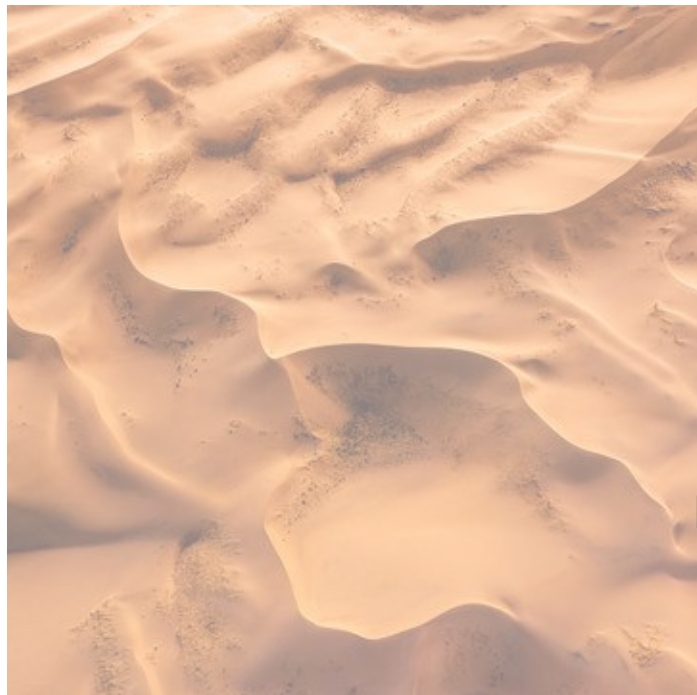
За счет чего так удачно? Универсален ли подход?



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 - 16K x 16K

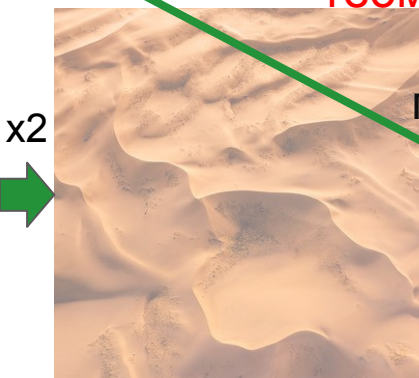
За счет чего так удачно? Универсален ли подход?



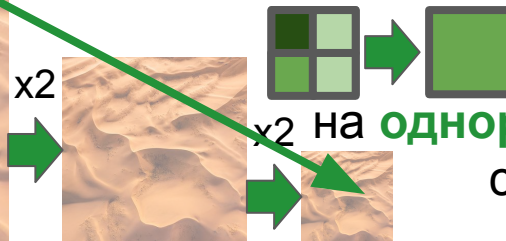
3D model

UV texture

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



полагались на **filterable** свойство



На **однородность семантики**



Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K

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 - 16K x 16K

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



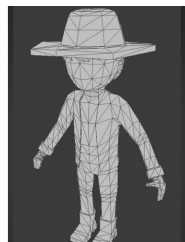
Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K

За счет чего так удачно? Универсален ли подход?



3D model



UV texture

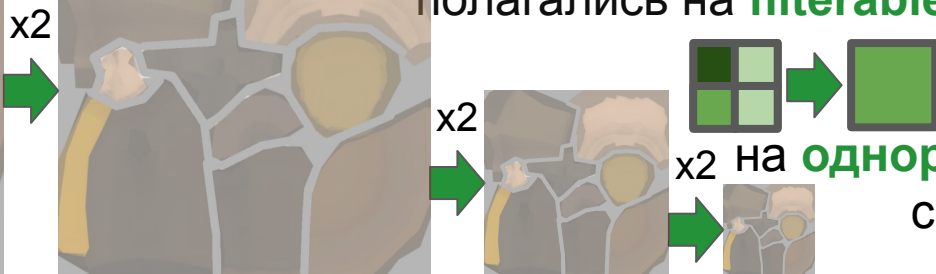


Можно ли стримить текстуру

3D модели произвольной топологии?

Чем отличается от пустыни?

полагались на **filterable** свойство



Mipmap texture

Виртуальные текстуры (virtual texture, texture streaming)

Мир - 2D текстура

В те времена - 16K x 16K



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 - 16K x 16K

За счет чего так удачно? Универсален ли подход?



3D model

UV texture

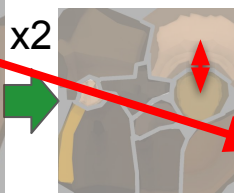


Можно ли стримить текстуру

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 - 16K x 16K

За счет чего так удачно? Универсален ли подход?



3D model

UV texture

Можно ли стримить геометрию 3D модели произвольной топологии?

~~x2~~

полагались на **filterable** свойство

x2

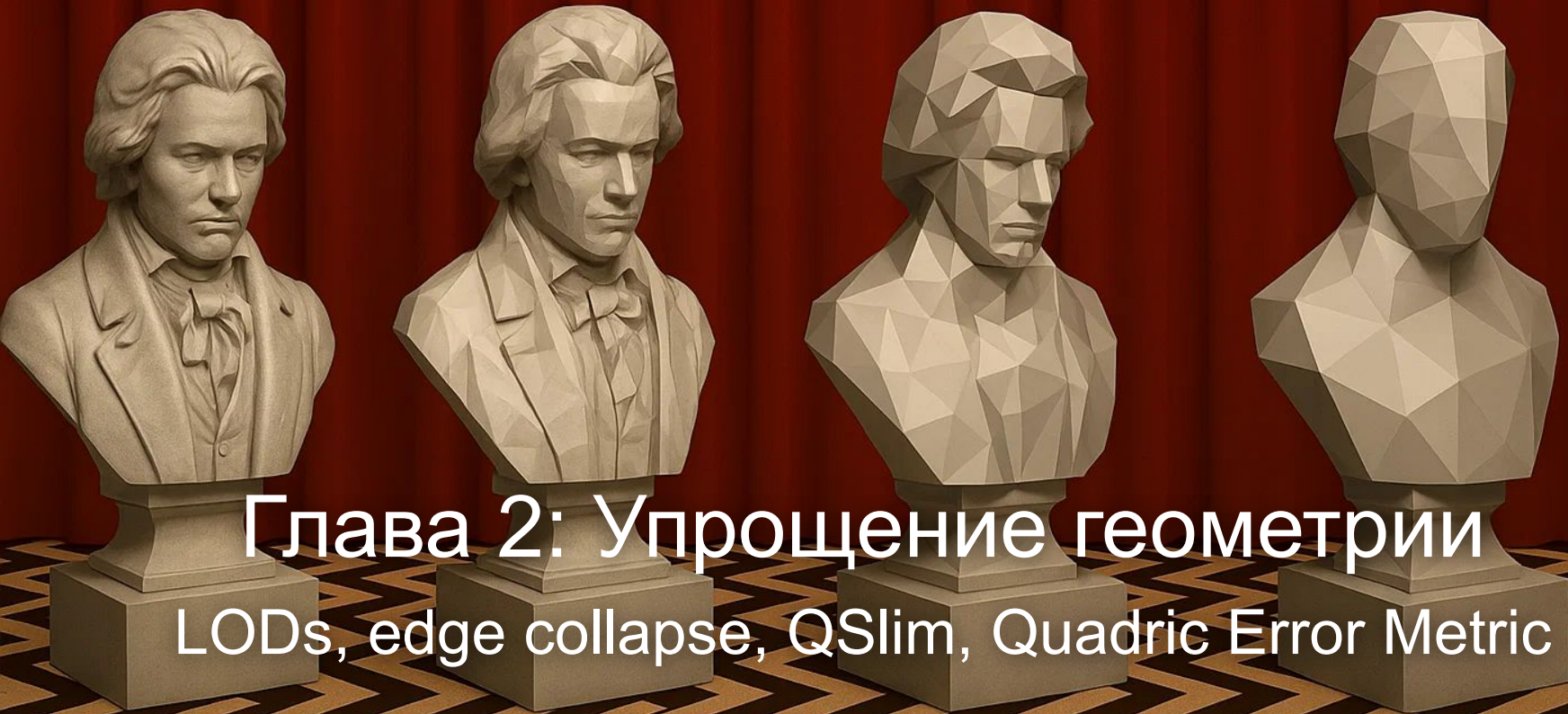
 x^2 

H2

o



однородность
семантики



Глава 2: Упрощение геометрии

LODs, edge collapse, QSlim, Quadric Error Metric

Упрощение геометрии

Levels of Details (LODs):



Polygons: 60.000

6.000

600

60

Упрощение геометрии

Levels of Details (LODs):



Polygons: 60.000

6.000

600

60

← close

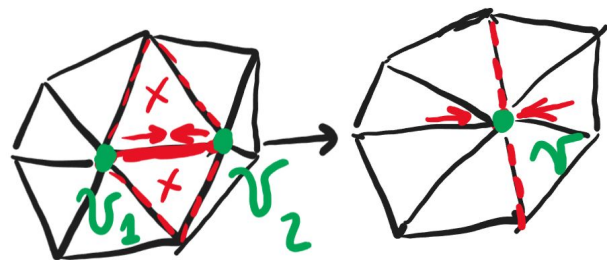
distance to camera

far →

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

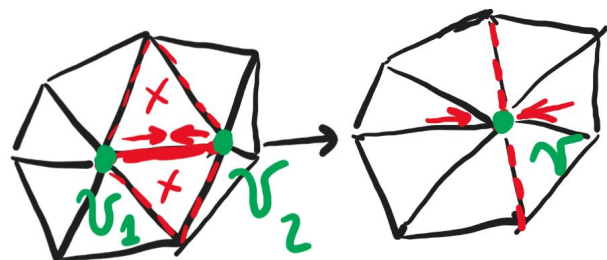
Упрощение геометрии

QSlm decimation: edge collapse (схлопывание ребер)



Упрощение геометрии

QSlm decimation: edge collapse (схлопывание ребер)



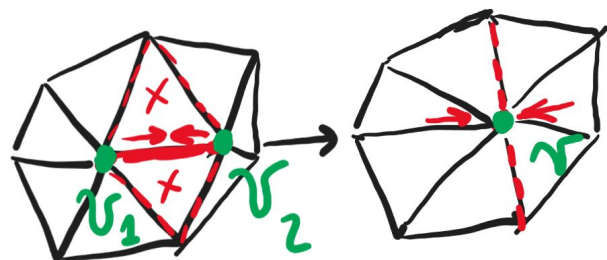
Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



Quadratic Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$

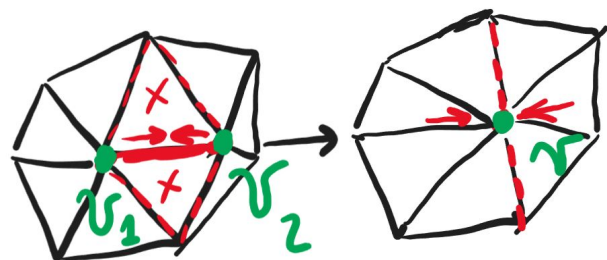


v = пересечение плоскостей

$error = \sum plane dist^2$

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: $v^T Q v$

$$Q = ?$$

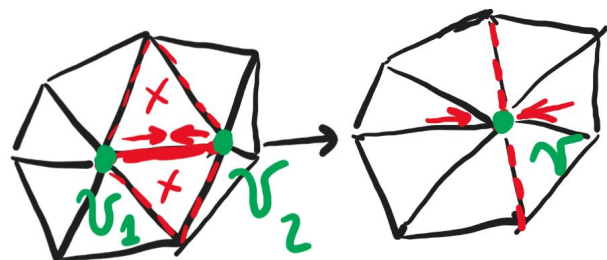


v = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

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: $v^T Q v$

$$Q = ?$$

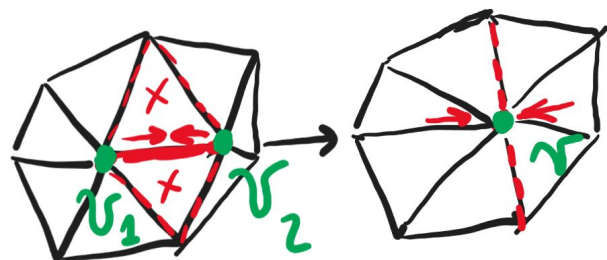


v = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



$$\begin{matrix} (v_1, v_2) \\ Q_1 \quad Q_2 \end{matrix} \xrightarrow{\text{edge collapse}} v = \underset{v}{\operatorname{argmin}} v^T \bar{Q} v$$
$$\bar{Q} = Q_1 + Q_2$$

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица Q

Ошибка каждой вершины: $v^T Q v$

$$Q = ?$$

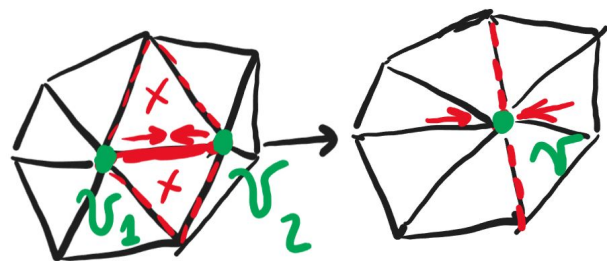


v = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



$$\begin{matrix} (v_1, v_2) \\ Q_1 \quad Q_2 \end{matrix} \xrightarrow{\text{edge collapse}} v = \underset{v}{\operatorname{argmin}} v^T \bar{Q} v$$
$$\bar{Q} = Q_1 + Q_2$$

Можем ли мы запретить
топологические изменения?

Quadric Error Metric (**QEM**)

На каждую вершину: 4x4 матрица Q

Ошибка каждой вершины: $v^T Q v$

$Q = ?$

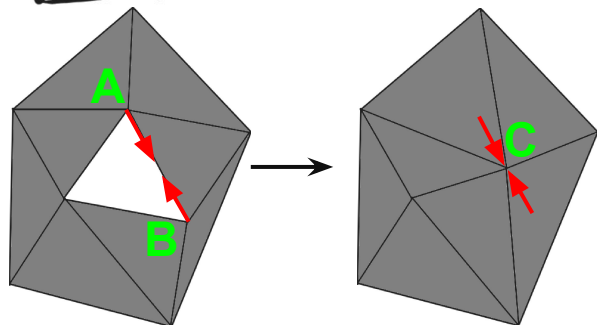
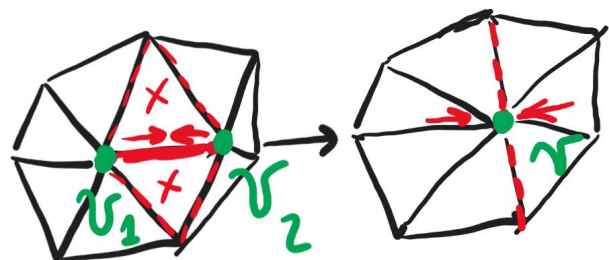


v = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

Упрощение геометрии

QSlim decimation: edge collapse (схлопывание ребер)



Можем ли мы запретить
топологические изменения?

Quadratic Error Metric (**QEM**)

На каждую вершину: 4x4 матрица **Q**

Ошибка каждой вершины: $v^T Q v$

$Q = ?$



v = пересечение плоскостей

$$\text{error} = \sum \text{plane dist}^2$$

Упрощение геометрии

Levels of Details (LODs):

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



Polygons: 60.000

6.000

600

60

← close

distance to camera

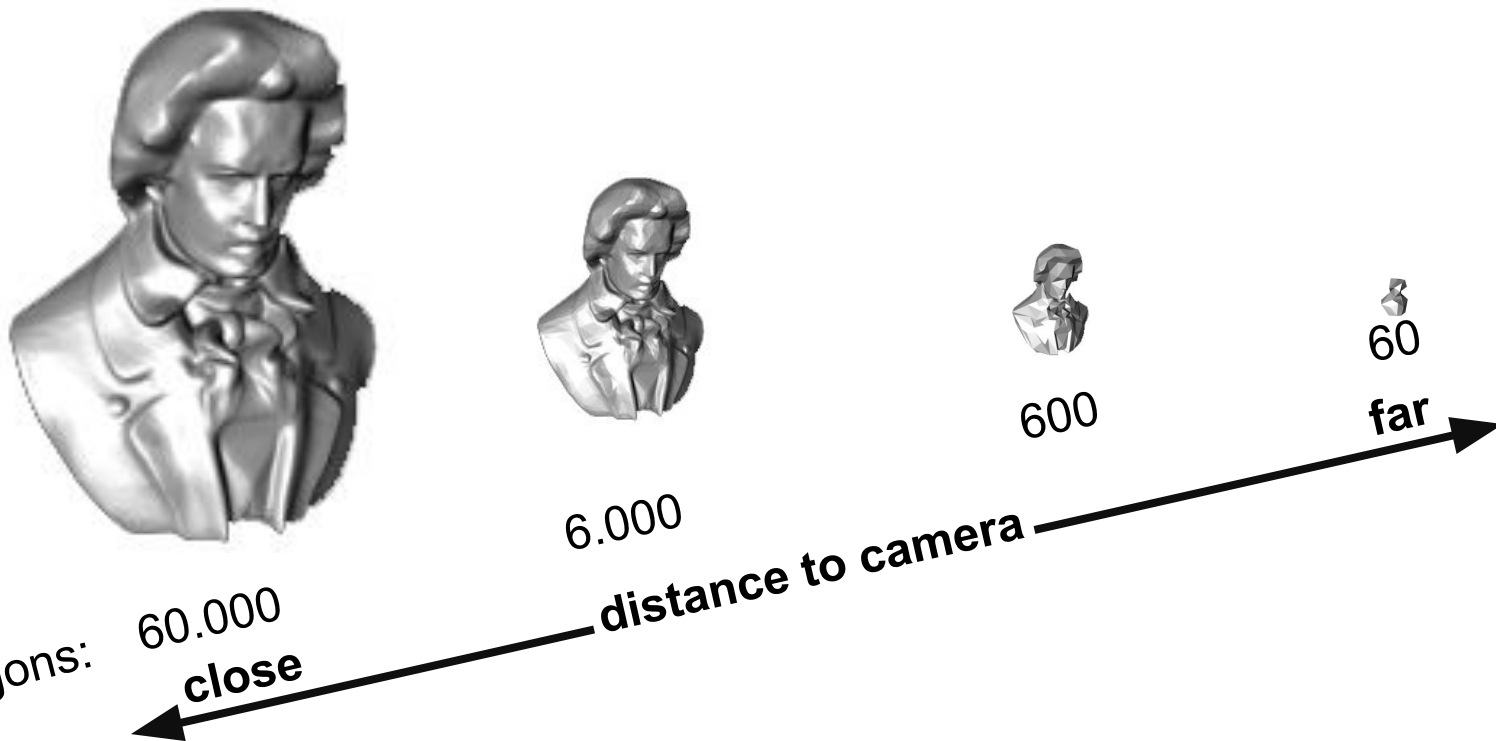
far →

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

Упрощение геометрии

Levels of Details (LODs):

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



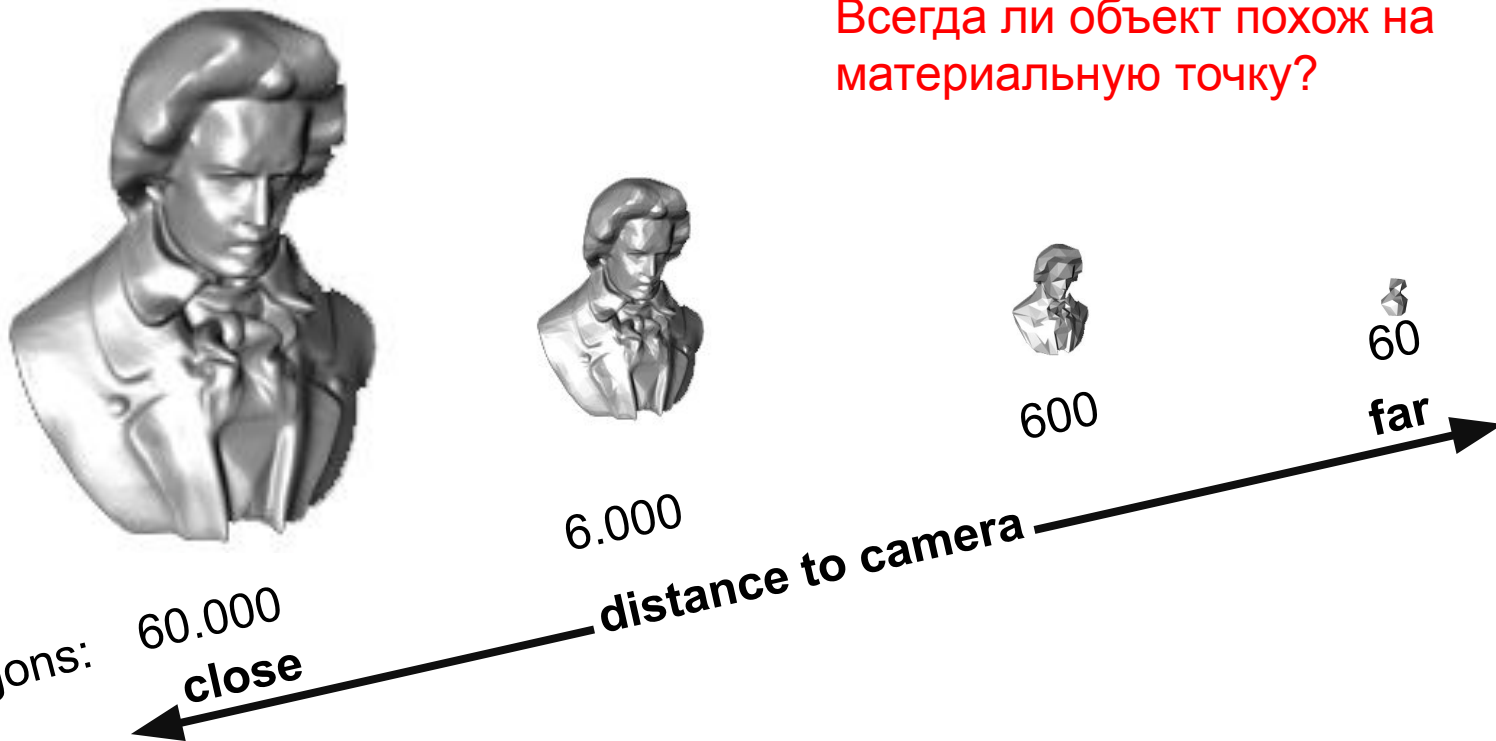
Simplifying surfaces with color and texture using quadric error metrics, Garland et. al., 1998

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?



High poly

Low poly



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**:

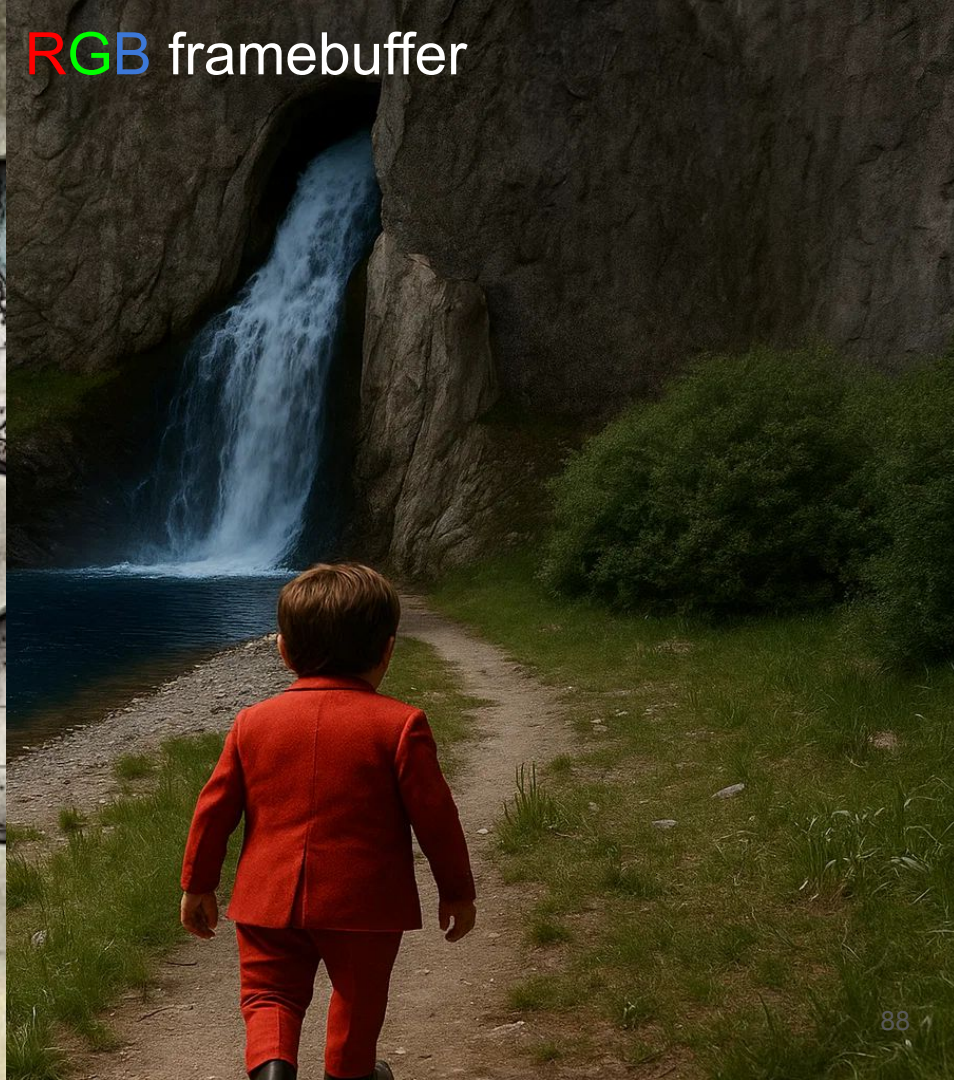
- визуальная неразличимость с оригиналом
- ограниченный бюджет вычислений на кадр
- автоматически и без компромиссов

Глава NANITE

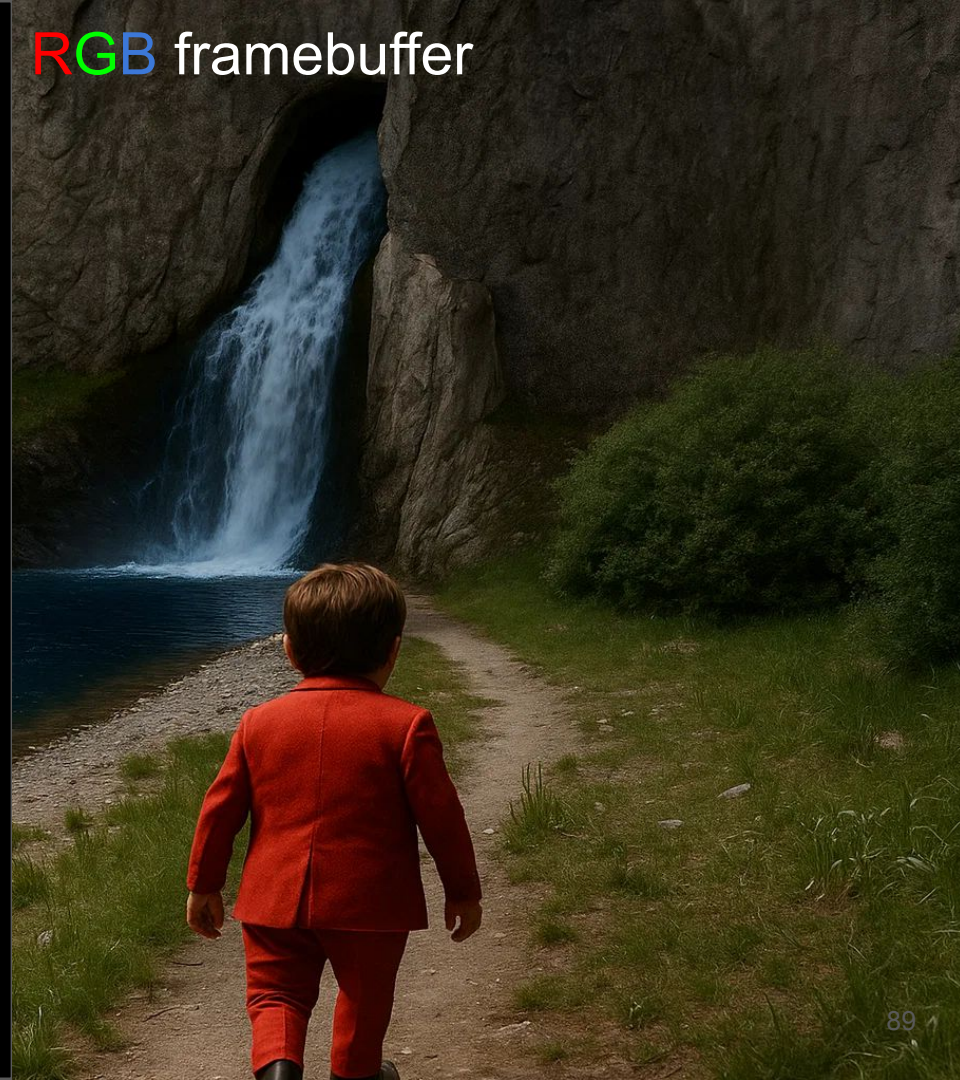




RGB framebuffer



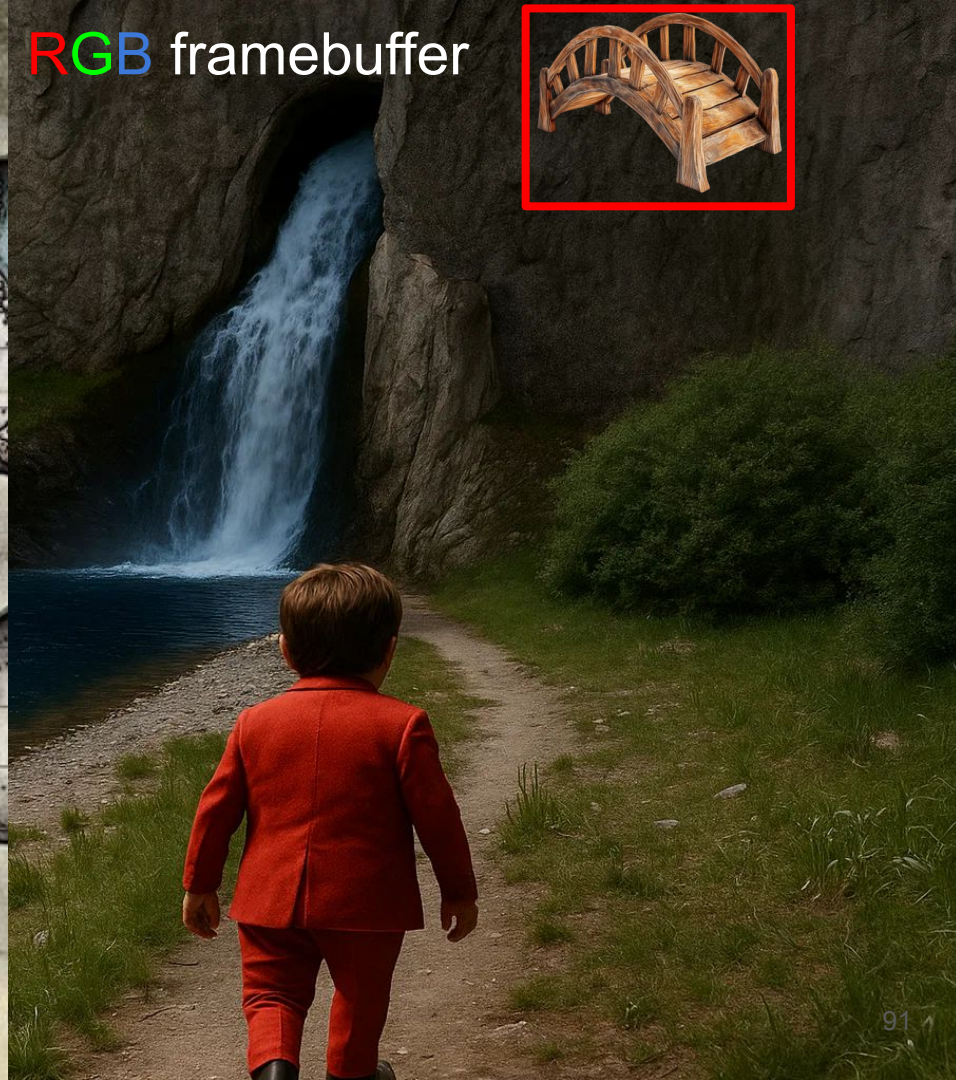
RGB framebuffer







RGB framebuffer





RGB framebuffer



RGB

VS

RGB



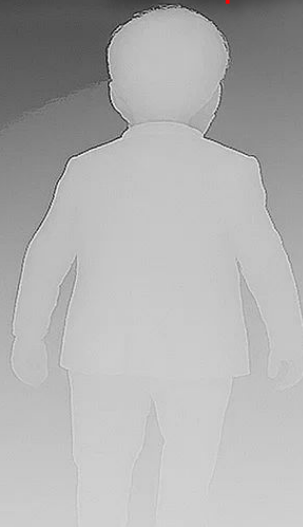


Depth framebuffer
Z framebuffer



f32 depth=A

VS





Depth framebuffer
Z framebuffer

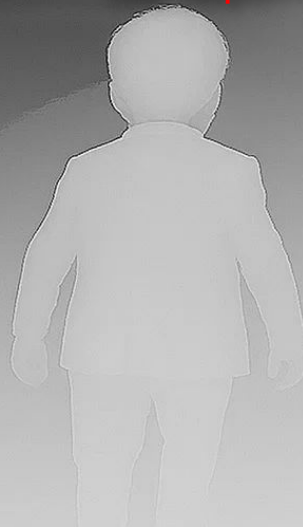


f32 depth=A

VS



f32 depth=B





Depth framebuffer
Z framebuffer

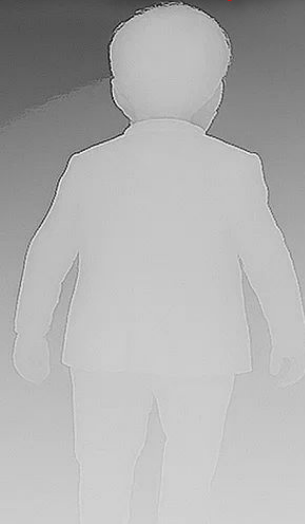


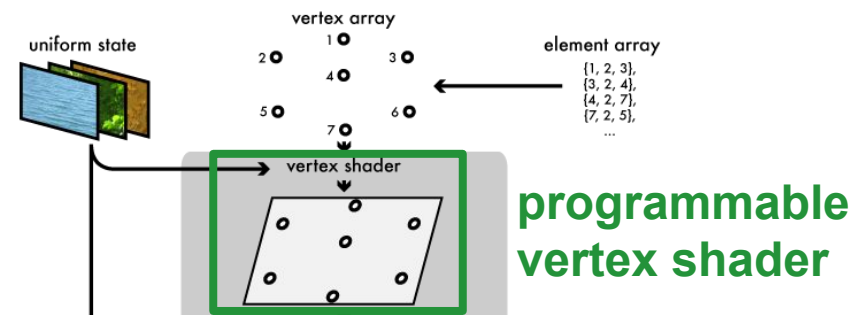
f32 depth=A

VS



f32 depth=B





Depth framebuffer
Z framebuffer

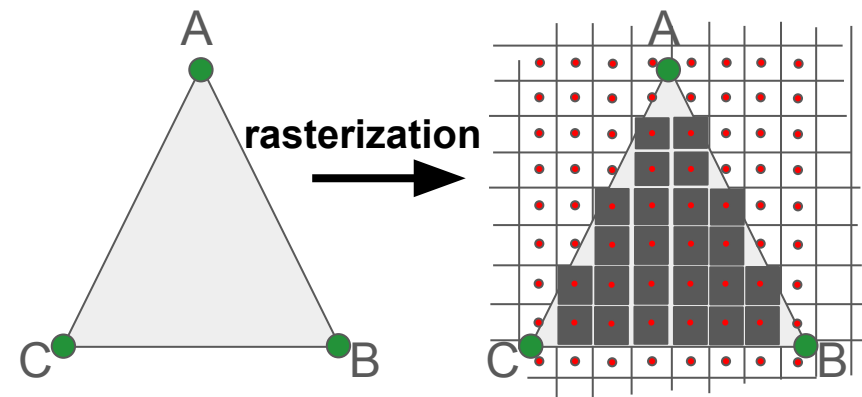
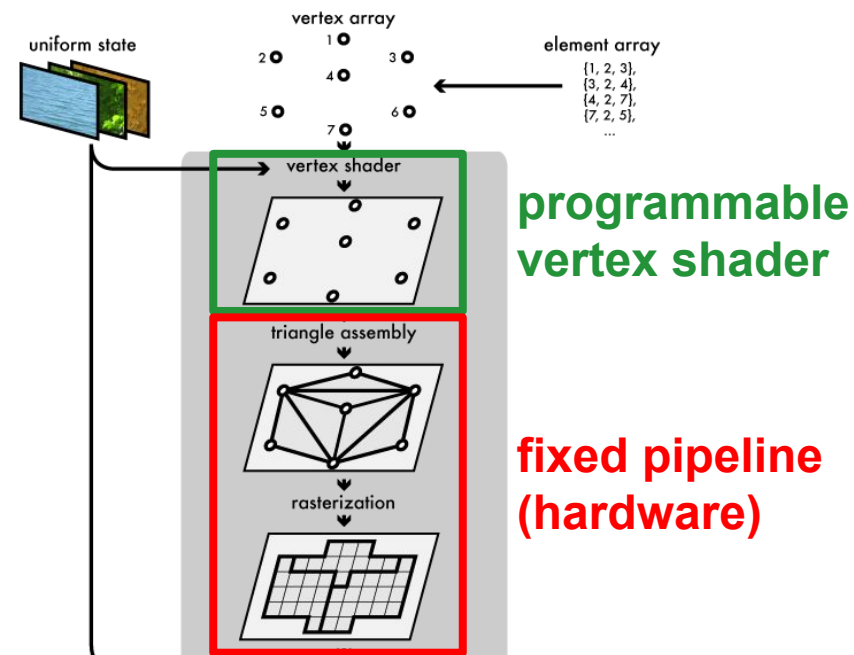


f32 depth=A

VS



f32 depth=B



Depth framebuffer
Z framebuffer

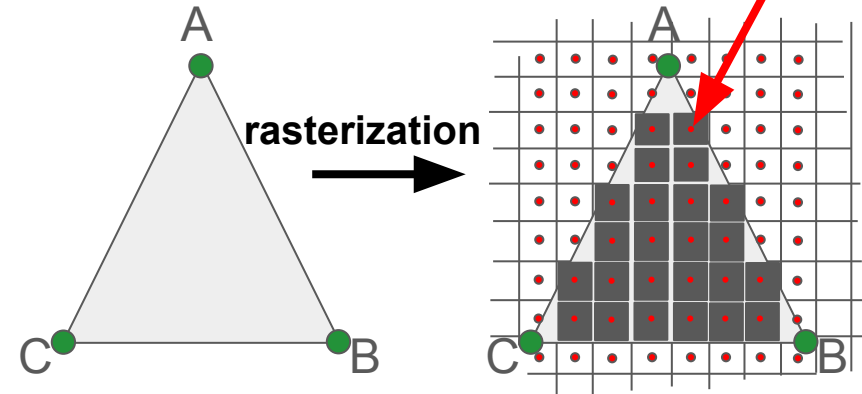
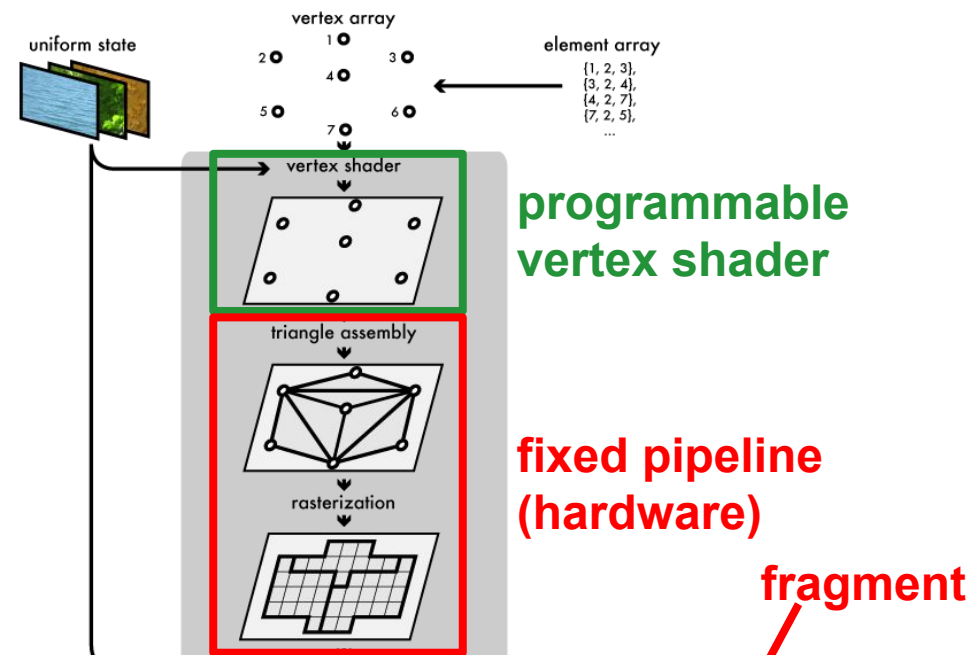


f32 depth=A

VS



f32 depth=B



Depth framebuffer
Z framebuffer

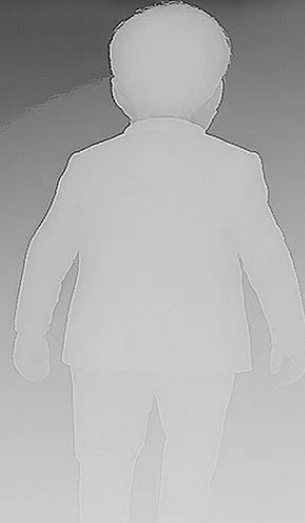


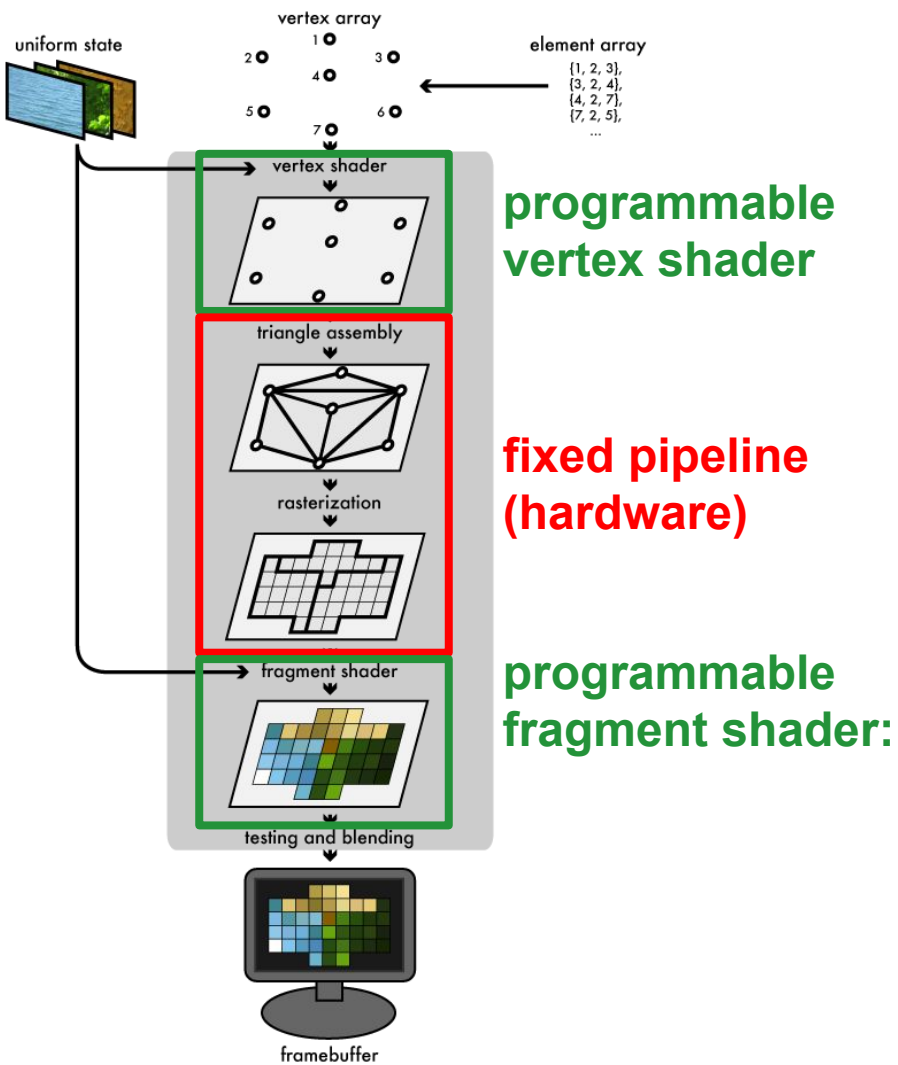
f32 depth=A

VS



f32 depth=B





Depth framebuffer
Z framebuffer

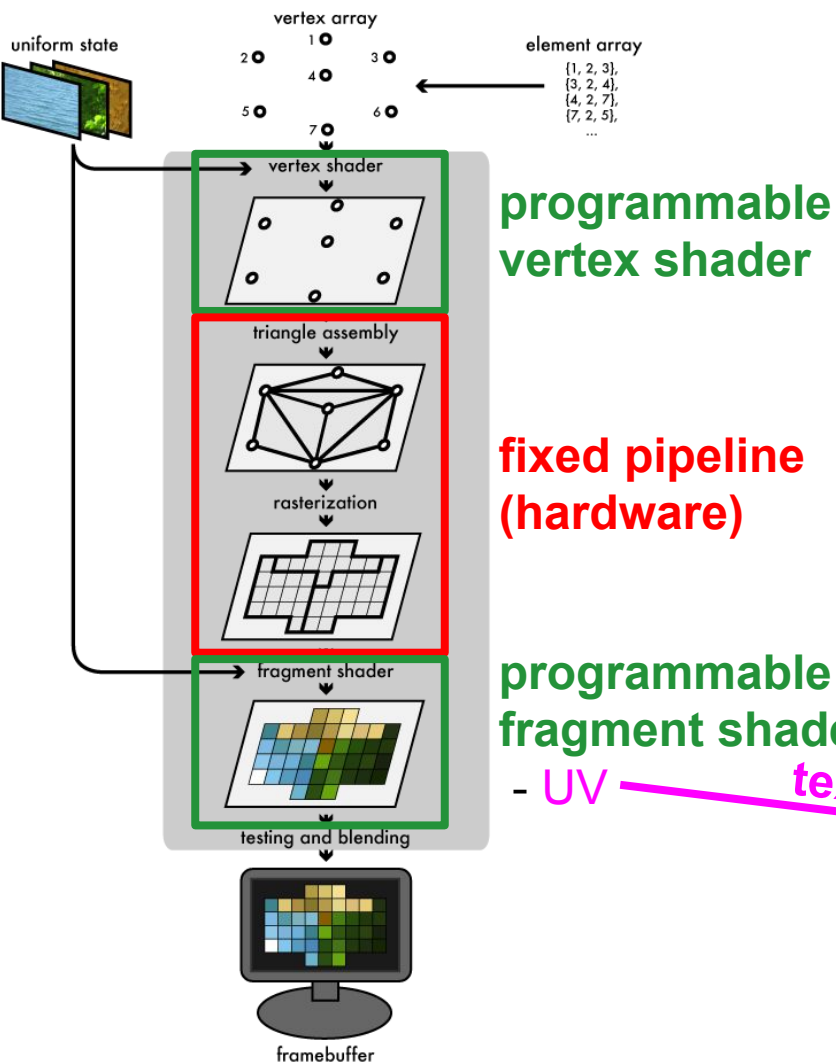


f32 depth=A

VS



f32 depth=B



Depth framebuffer
Z framebuffer

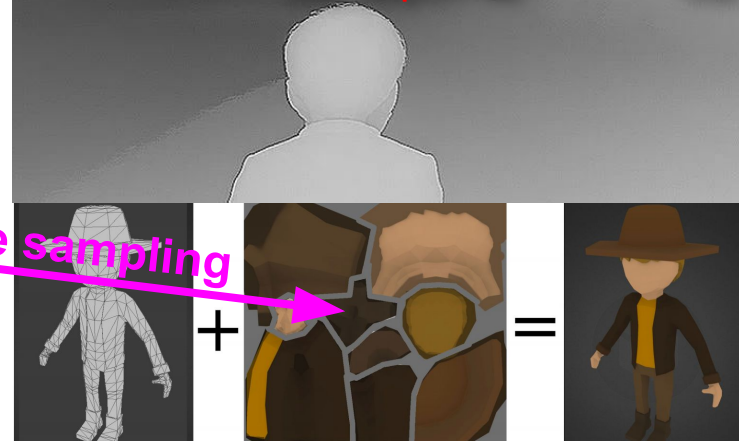


f32 depth=A

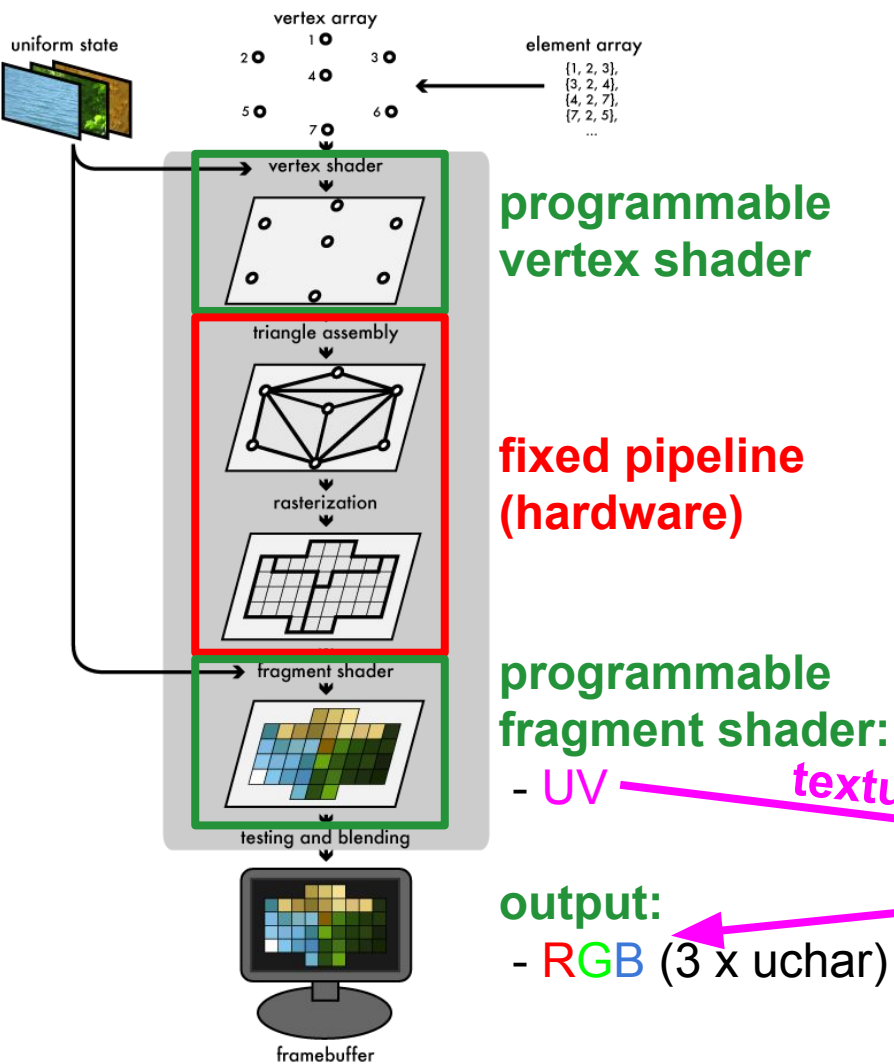


f32 depth=B

VS



UV texture



Depth framebuffer
Z framebuffer

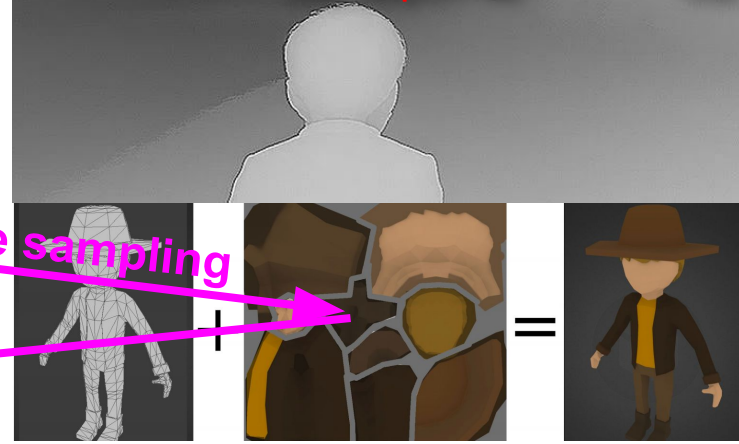


f32 depth=A

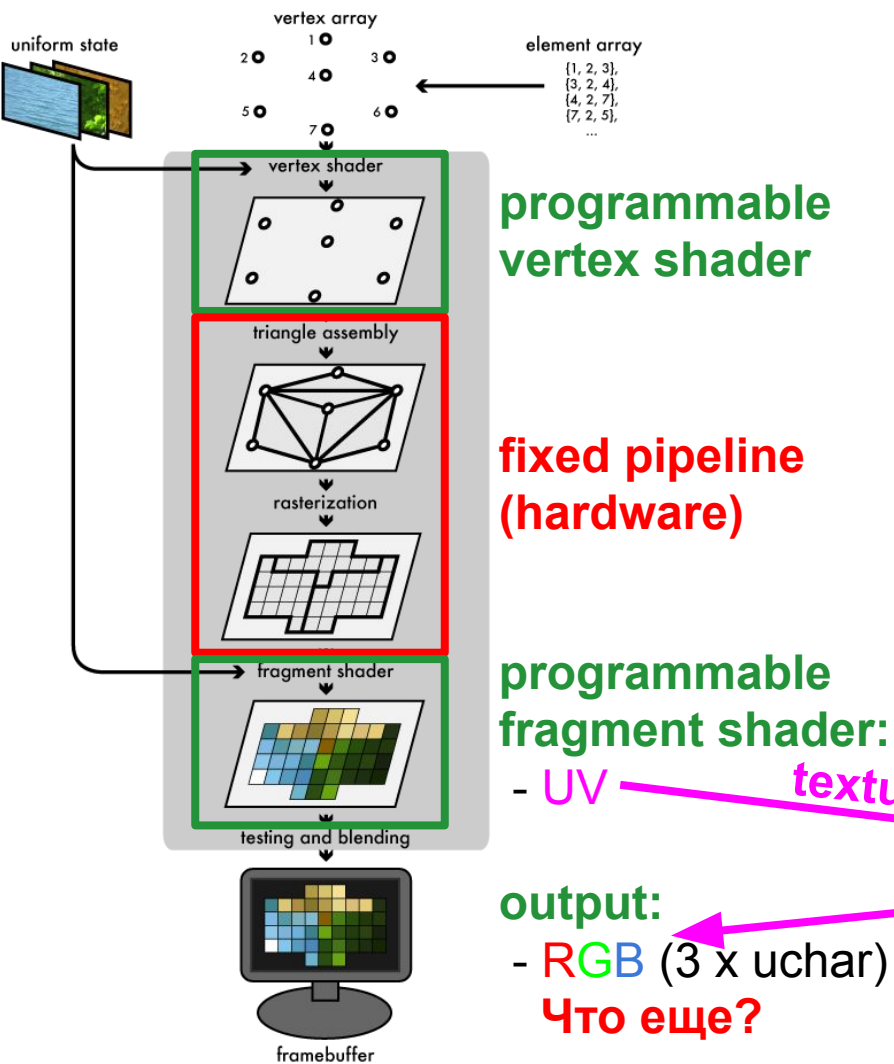


f32 depth=B

VS



UV texture



Depth framebuffer
Z framebuffer

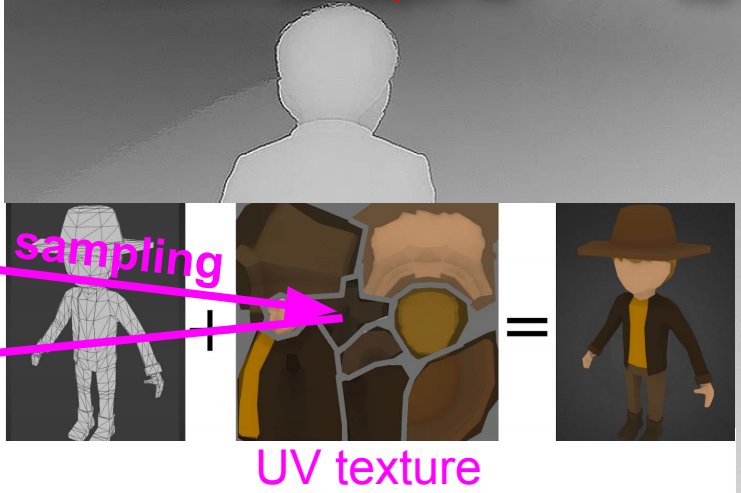


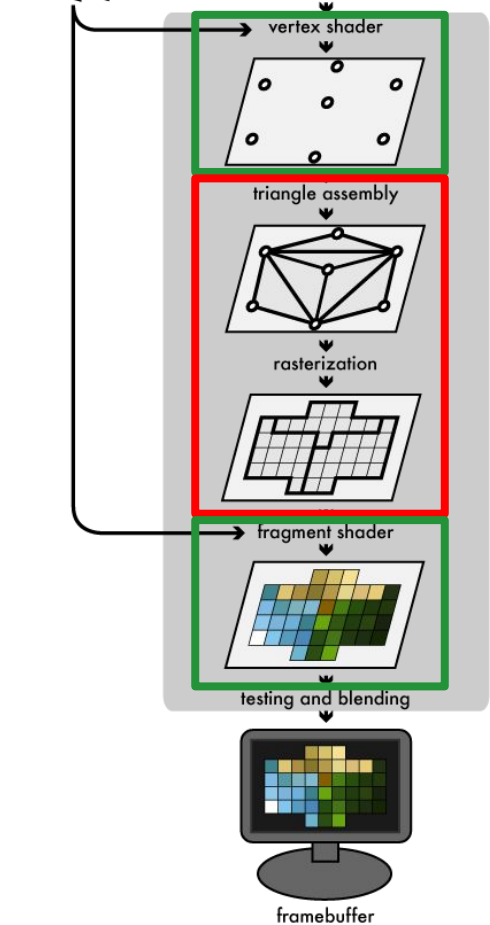
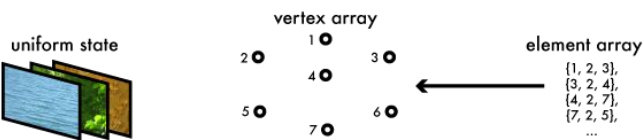
f32 depth=A



f32 depth=B

VS





**programmable
vertex shader**

**fixed pipeline
(hardware)**

**programmable
fragment shader:**

- UV *texture sampling*
 - shadows/lighting
- output:**
- RGB (3 x uchar)
 - Depth (32-bit float)

Depth framebuffer
Z framebuffer

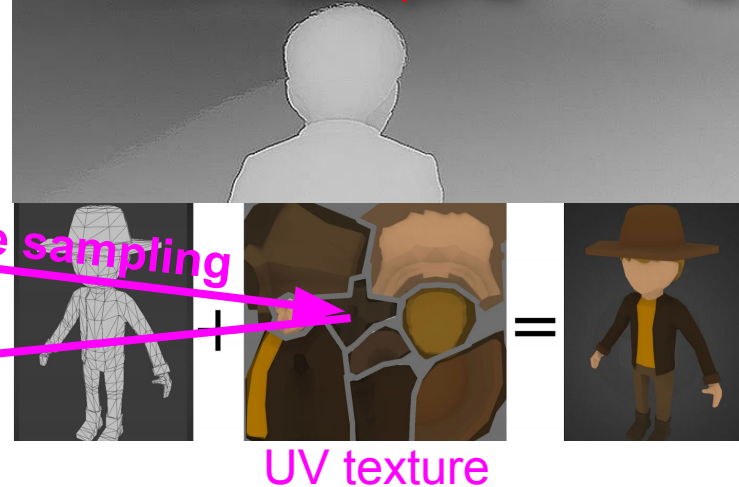


f32 depth=A



f32 depth=B

VS

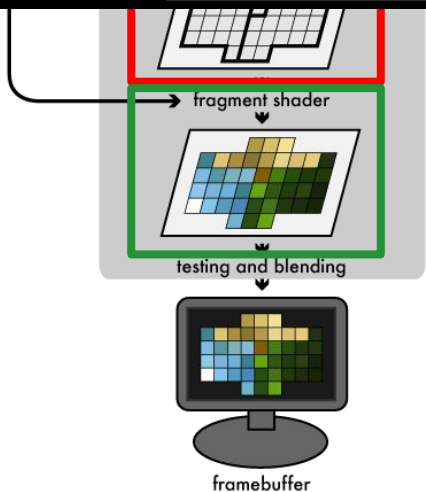


UV texture

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

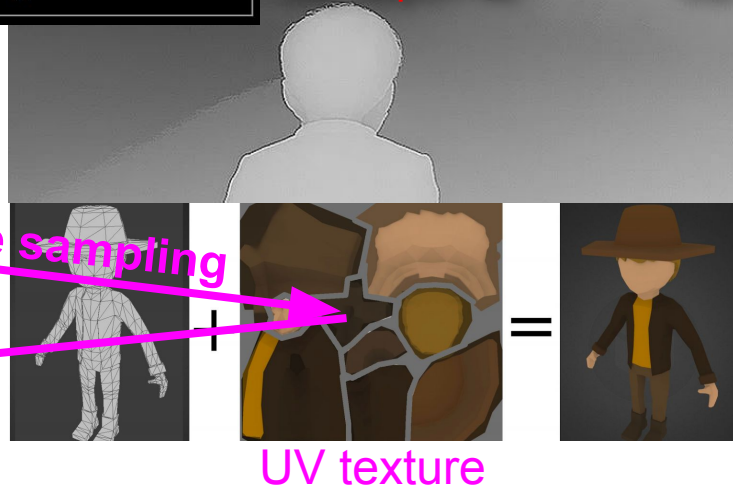


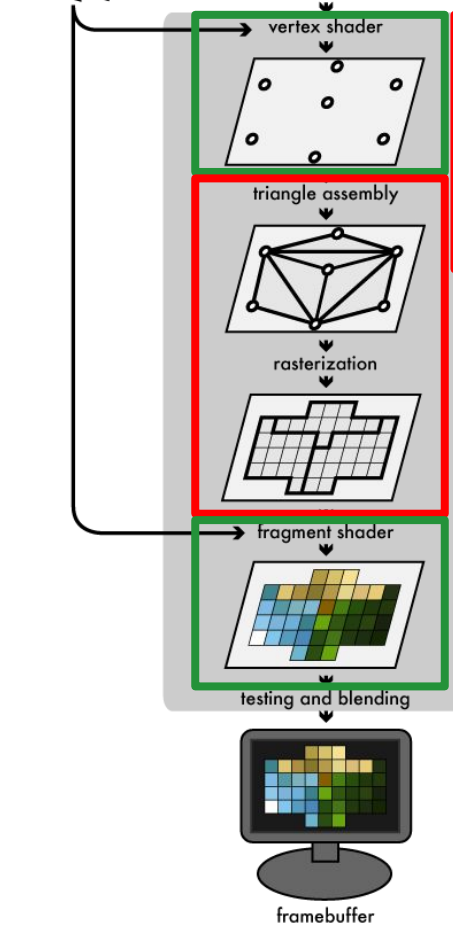
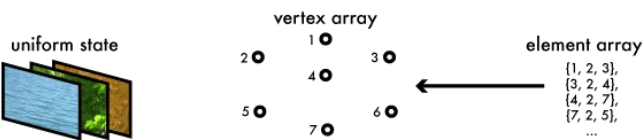
programmable
fragment shader:

- UV
- shadows/lighting

output:

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





**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

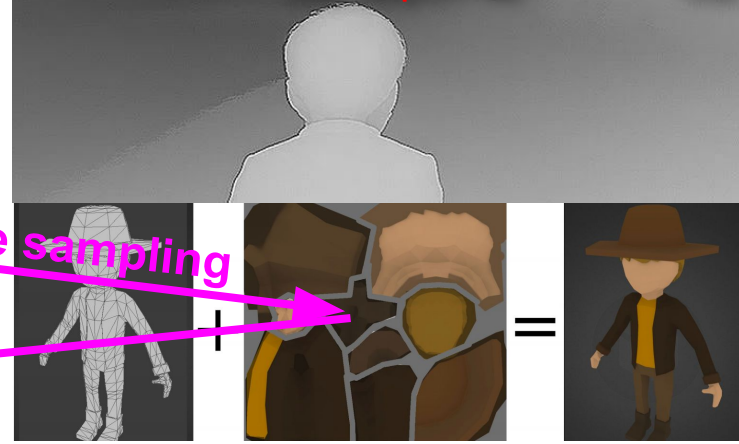


f32 depth=A



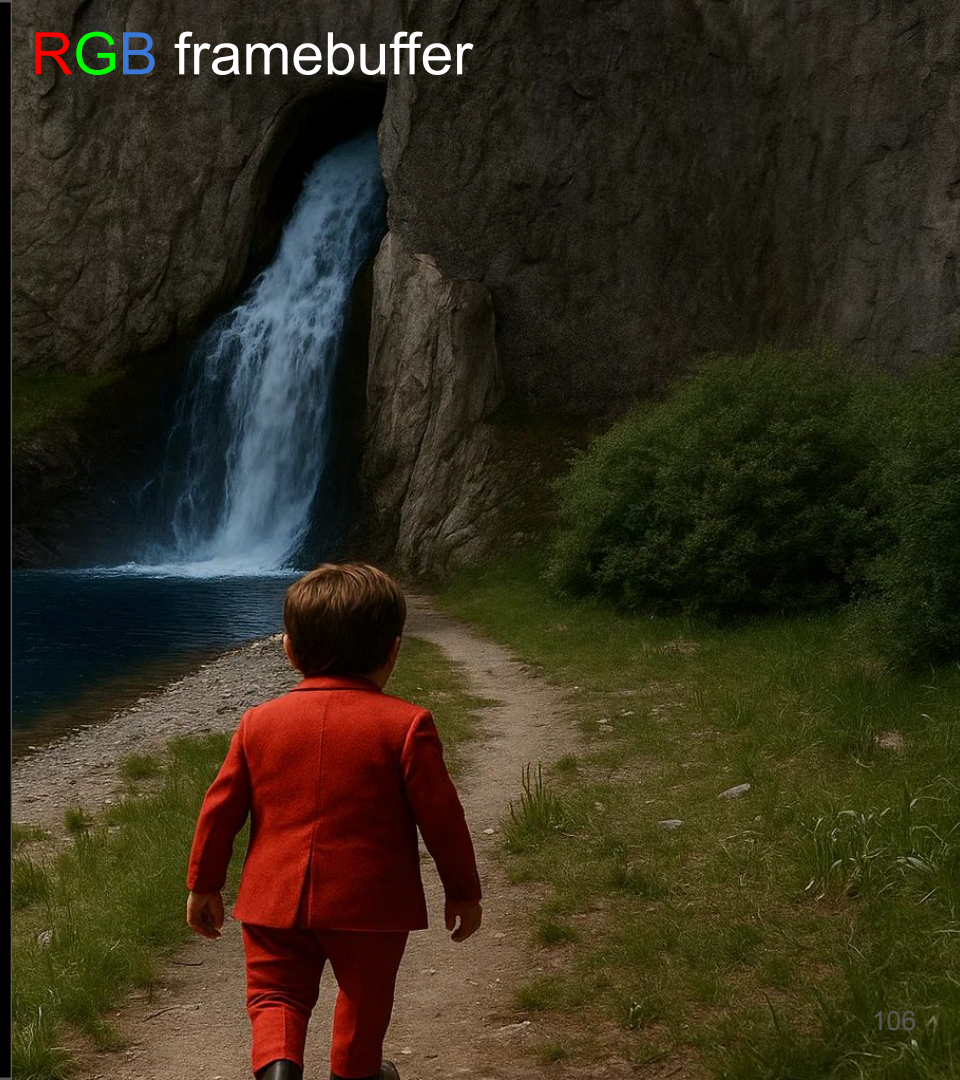
f32 depth=B

VS



UV texture

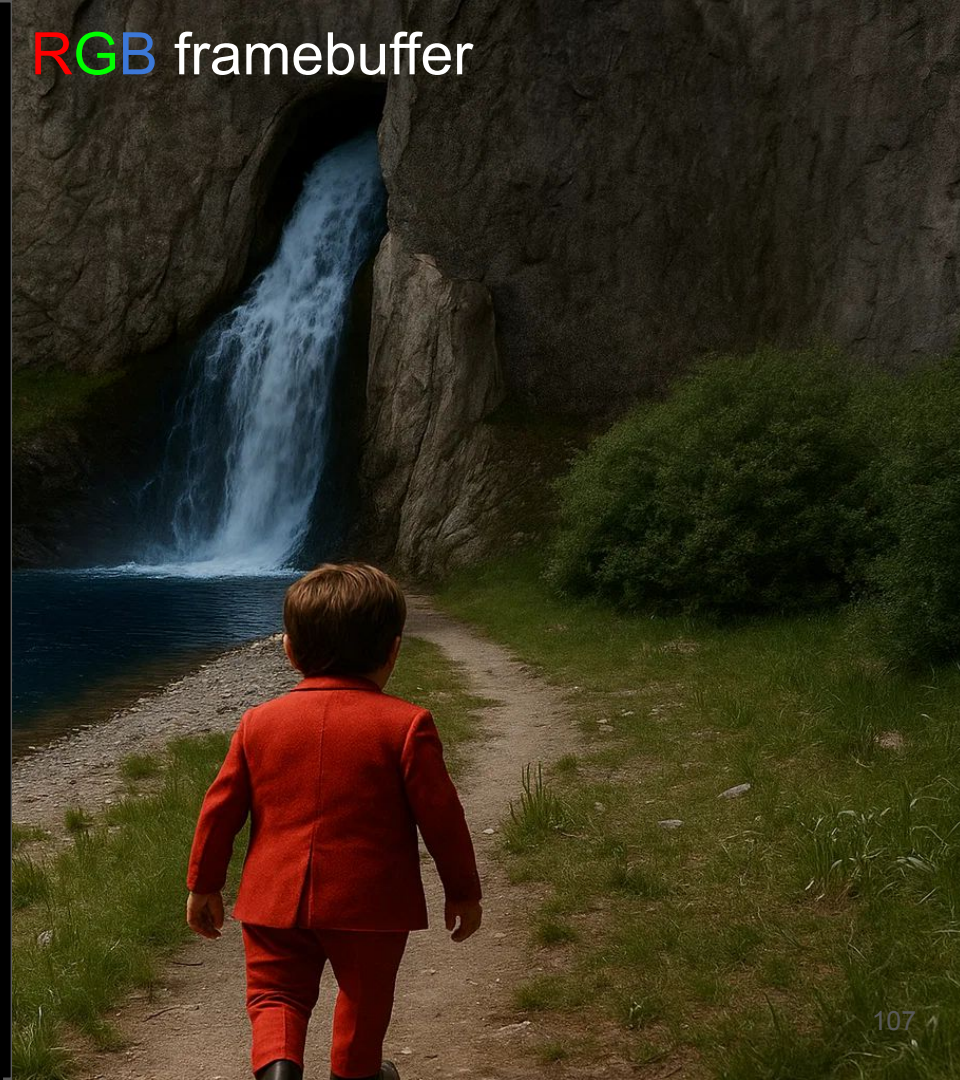
RGB framebuffer



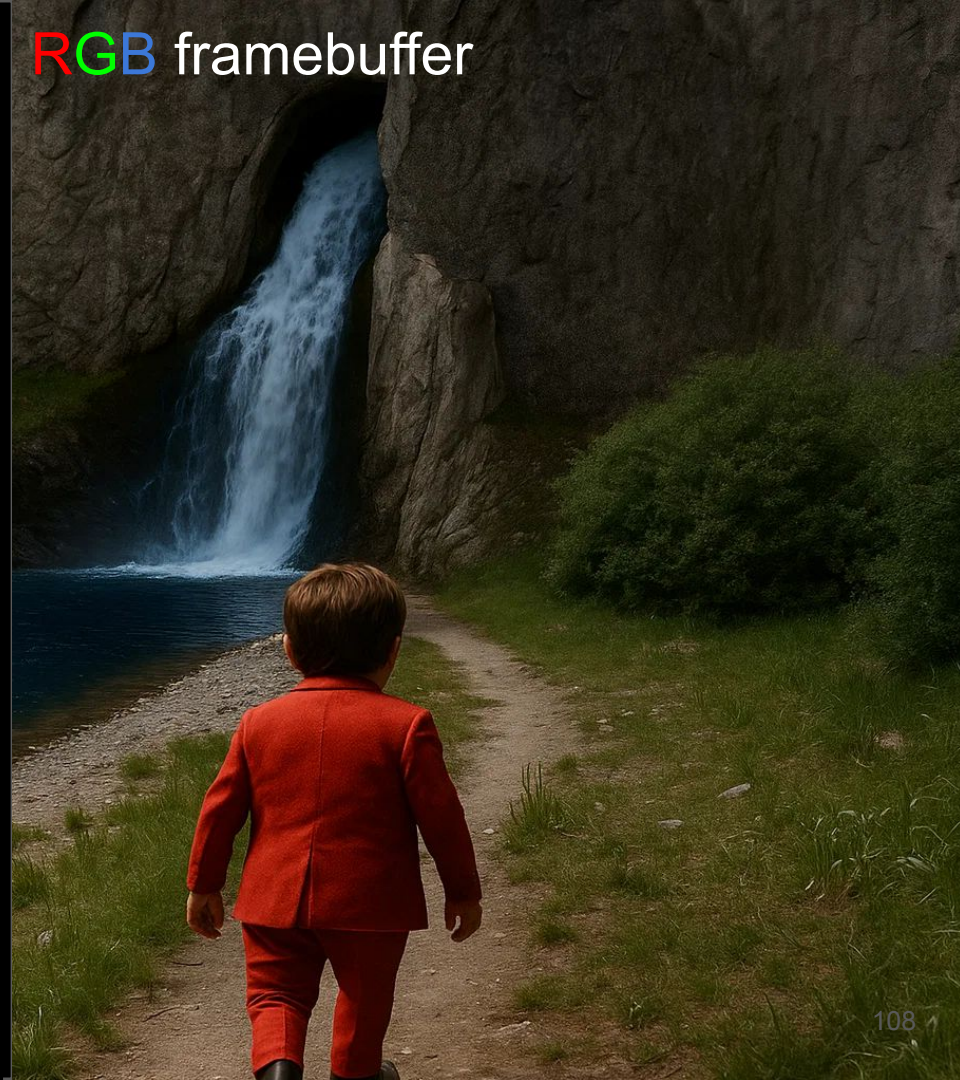
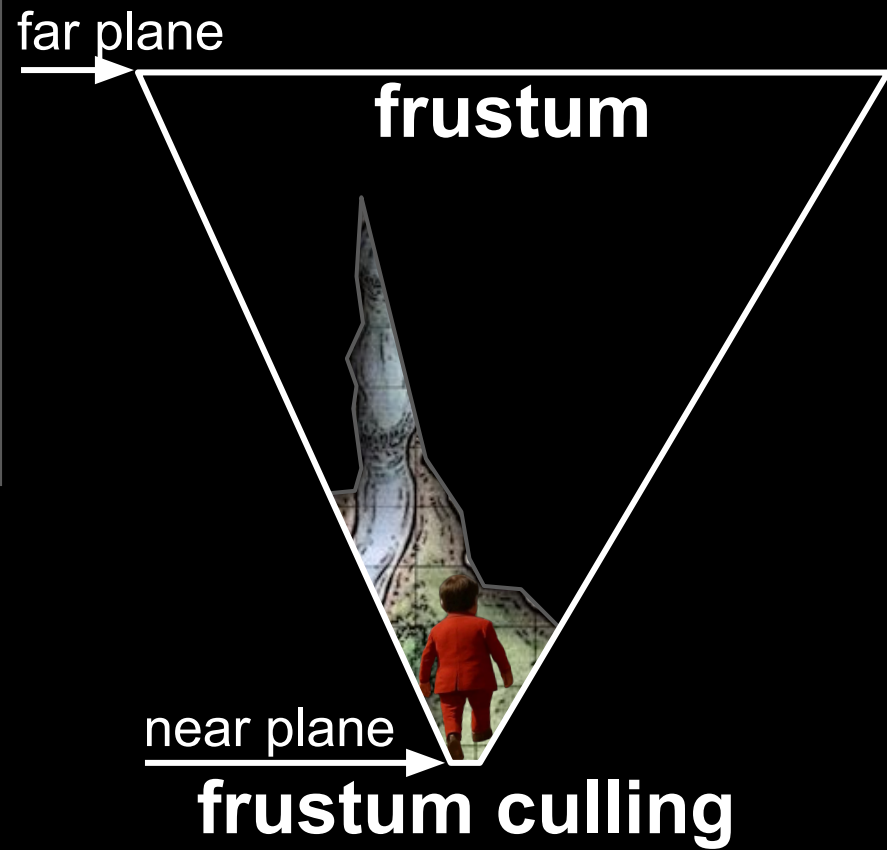
RGB framebuffer

far plane

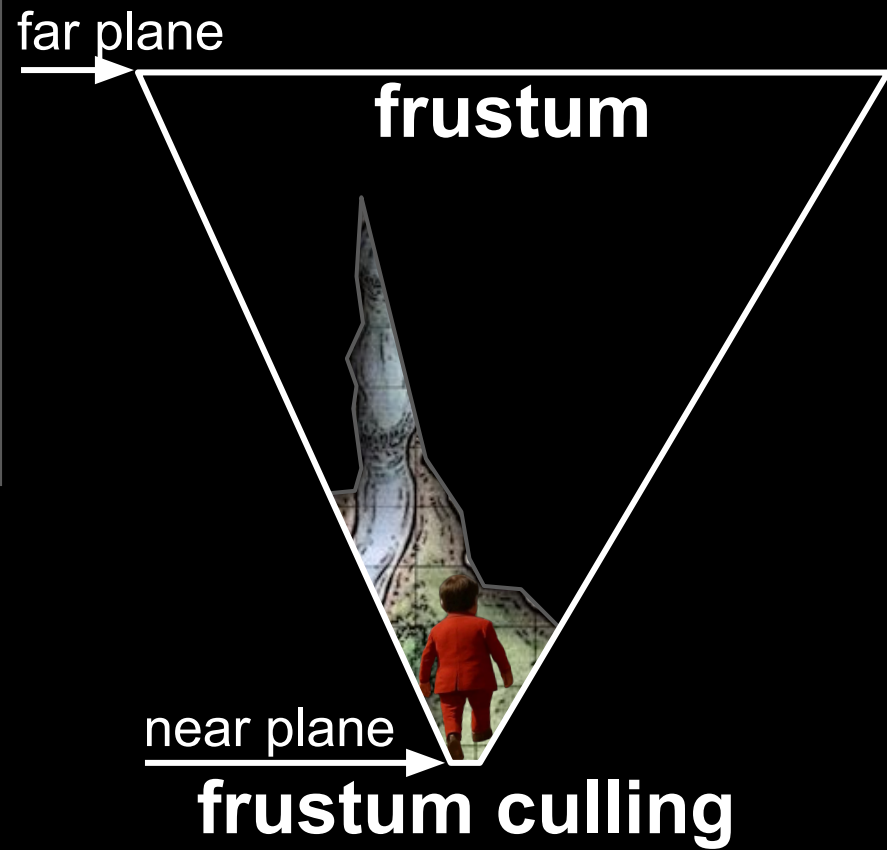
near plane



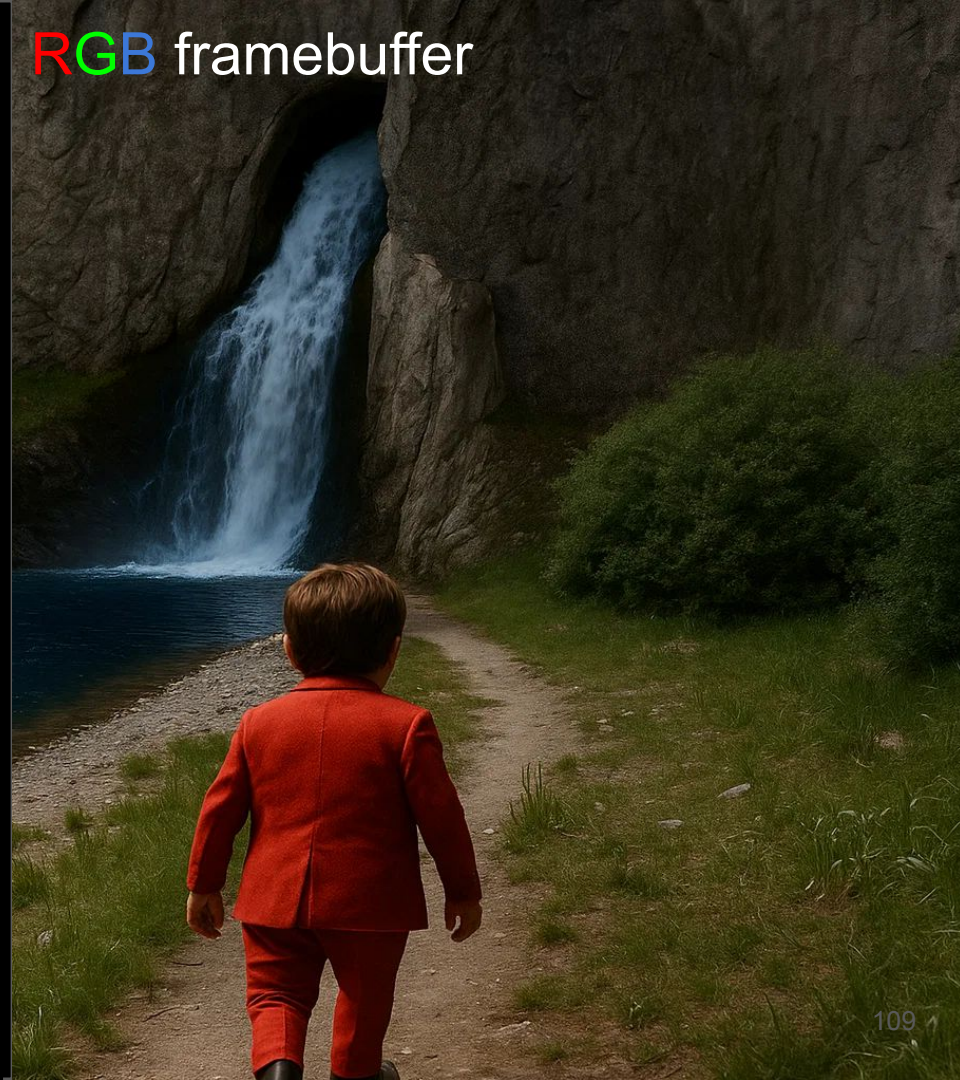
RGB framebuffer



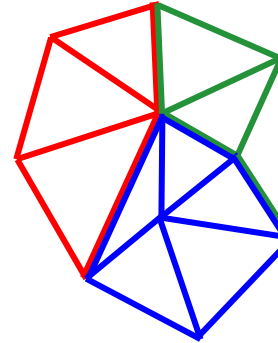
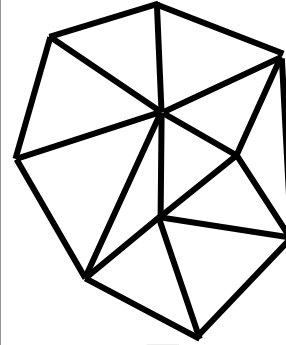
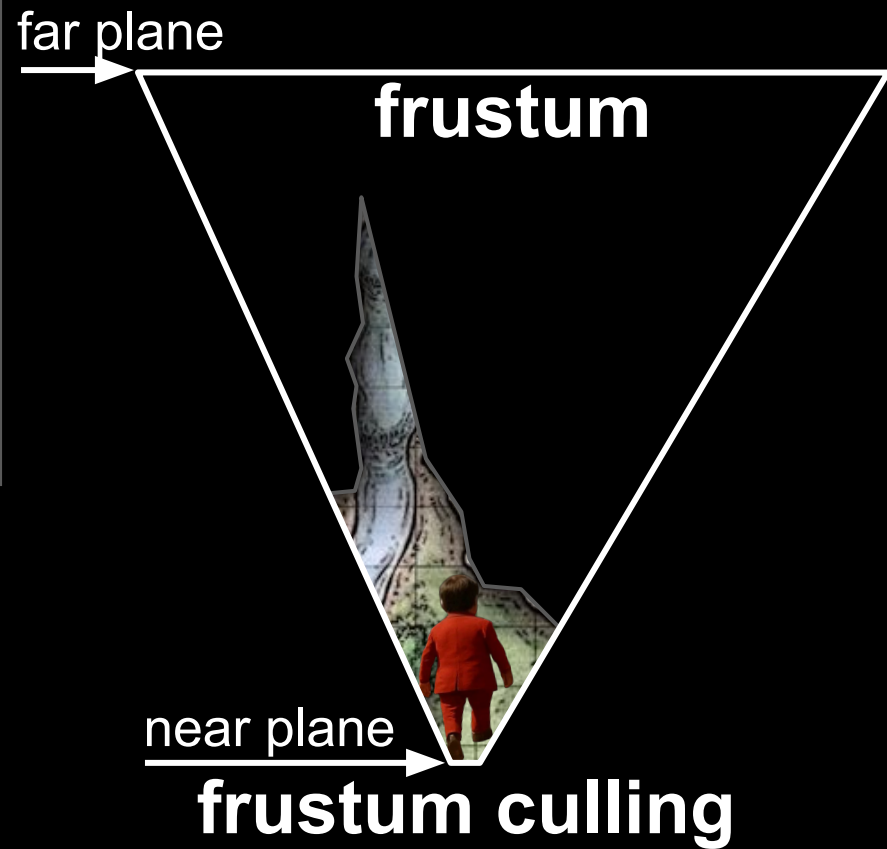
Придется вычислять в каждой вершине!
Как ускорить **vertex shader**?



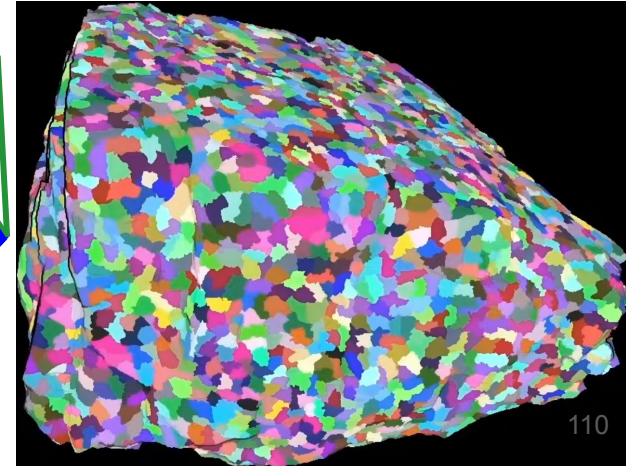
RGB framebuffer



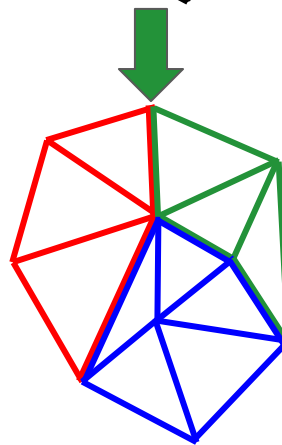
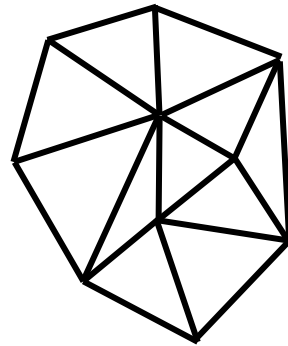
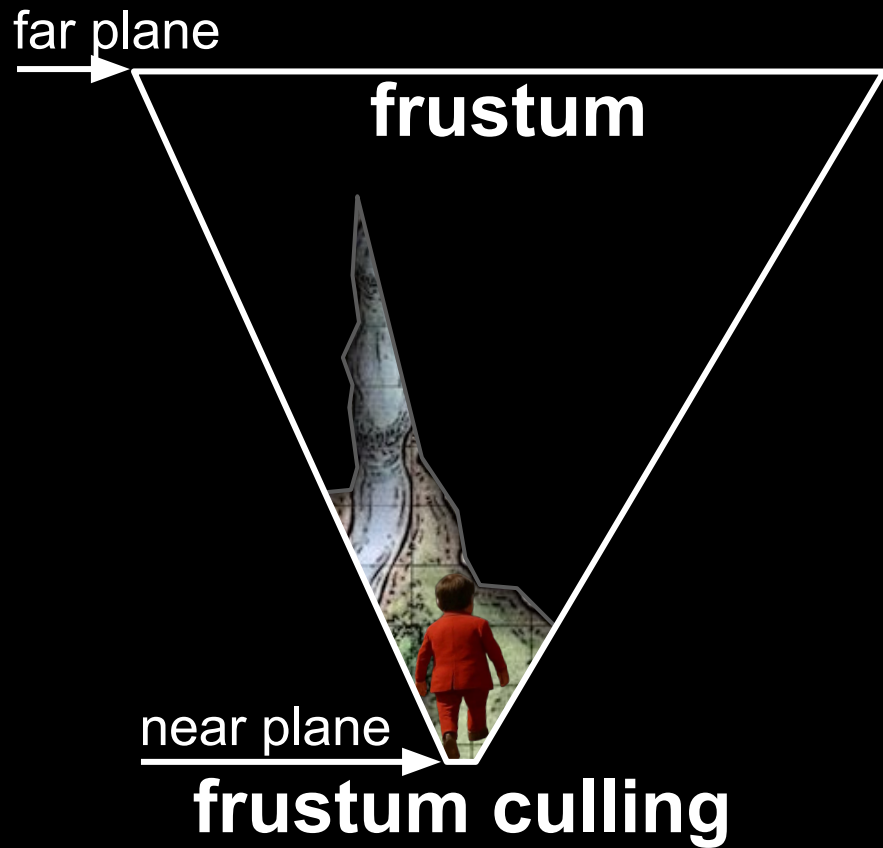
Придется вычислять в каждой вершине!
Как ускорить **vertex shader**?



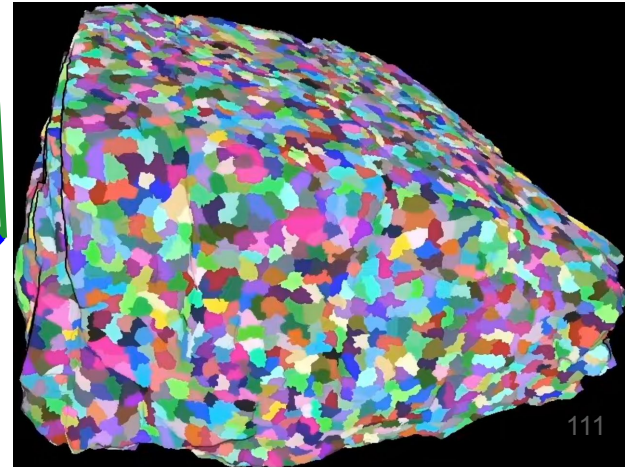
объединим в кластеры
по 128 треугольников



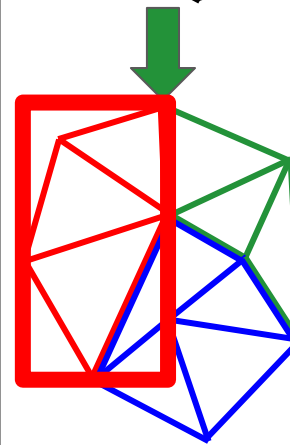
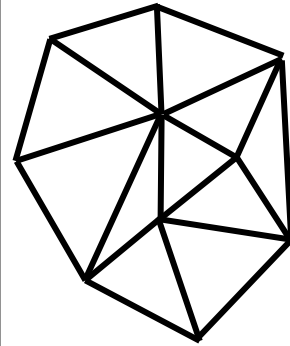
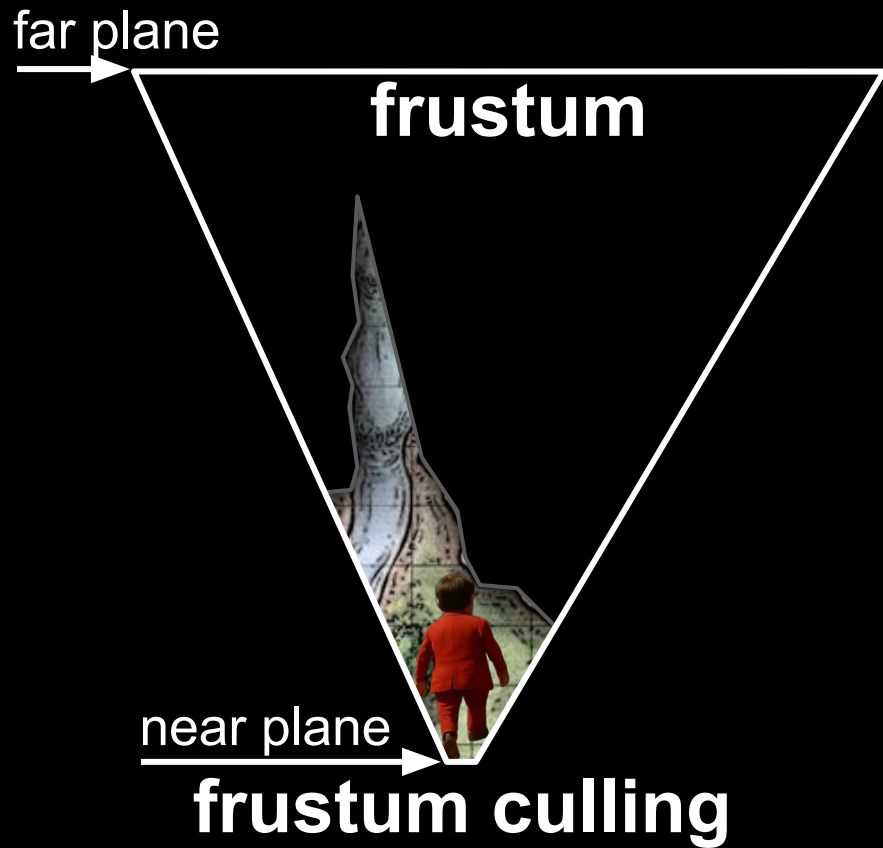
Придется вычислять в каждой вершине!
Как ускорить **vertex shader**?
Как для кластера выполнить **frustum culling**?



↓ объединим в кластеры
по 128 треугольников



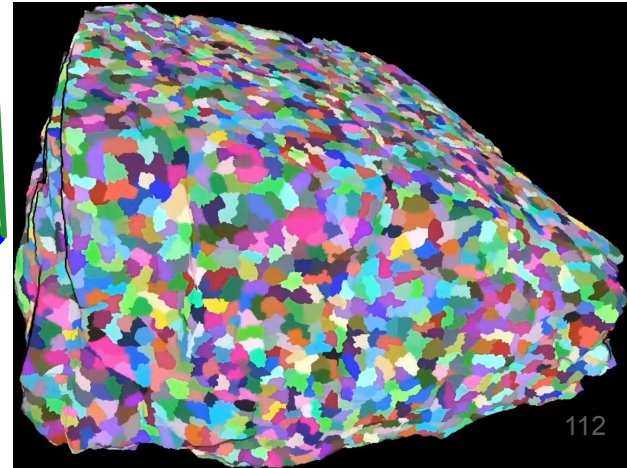
Придется вычислять в каждой вершине!
Как ускорить **vertex shader**?
Как для кластера выполнить **frustum culling**?



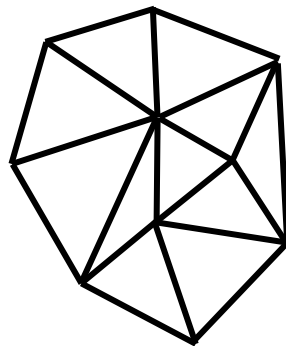
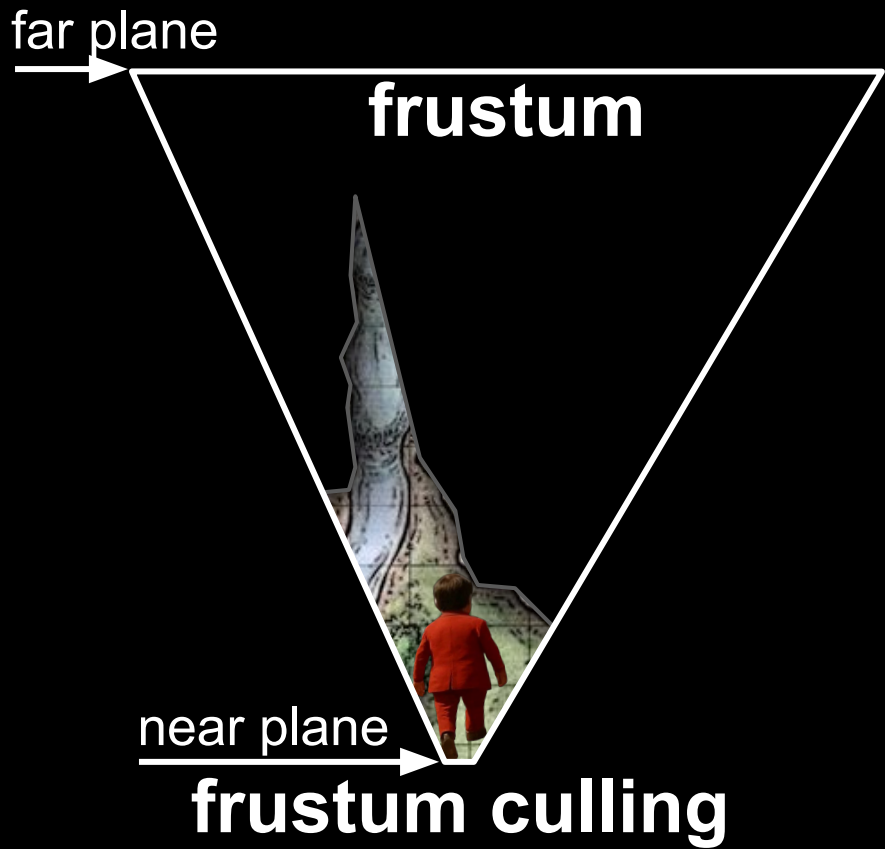
AABB



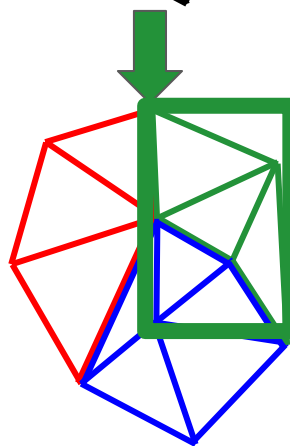
объединим в кластеры
по 128 треугольников



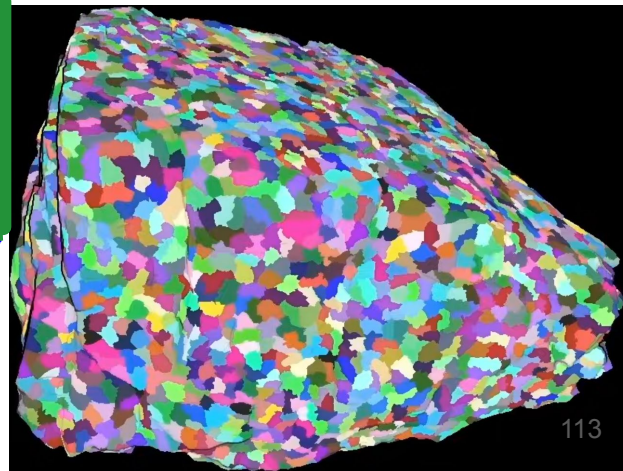
Придется вычислять в каждой вершине!
Как ускорить **vertex shader**?
Как для кластера выполнить **frustum culling**?



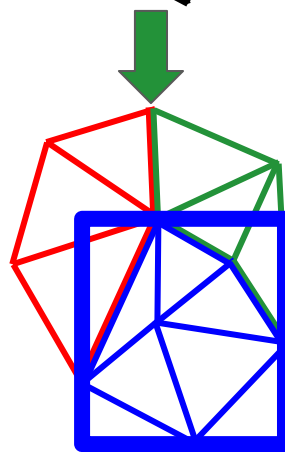
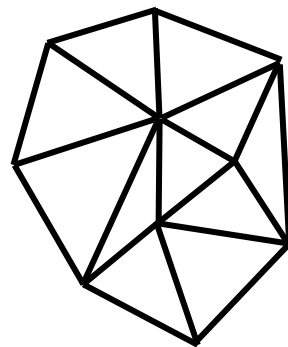
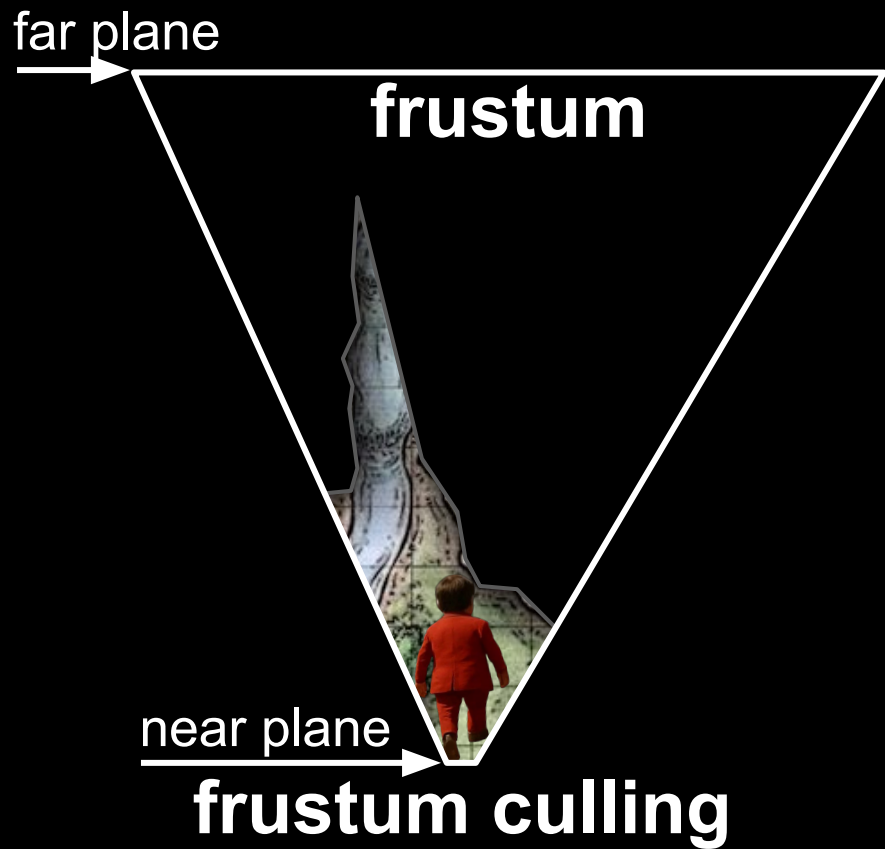
объединим в кластеры
по 128 треугольников



AABB



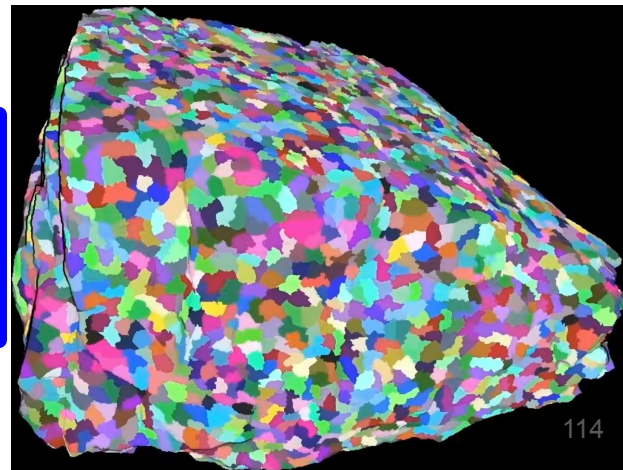
Придется вычислять в каждой вершине!
Как ускорить **vertex shader**?
Как для кластера выполнить **frustum culling**?



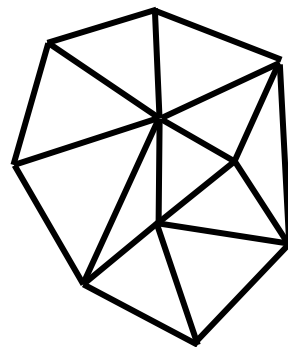
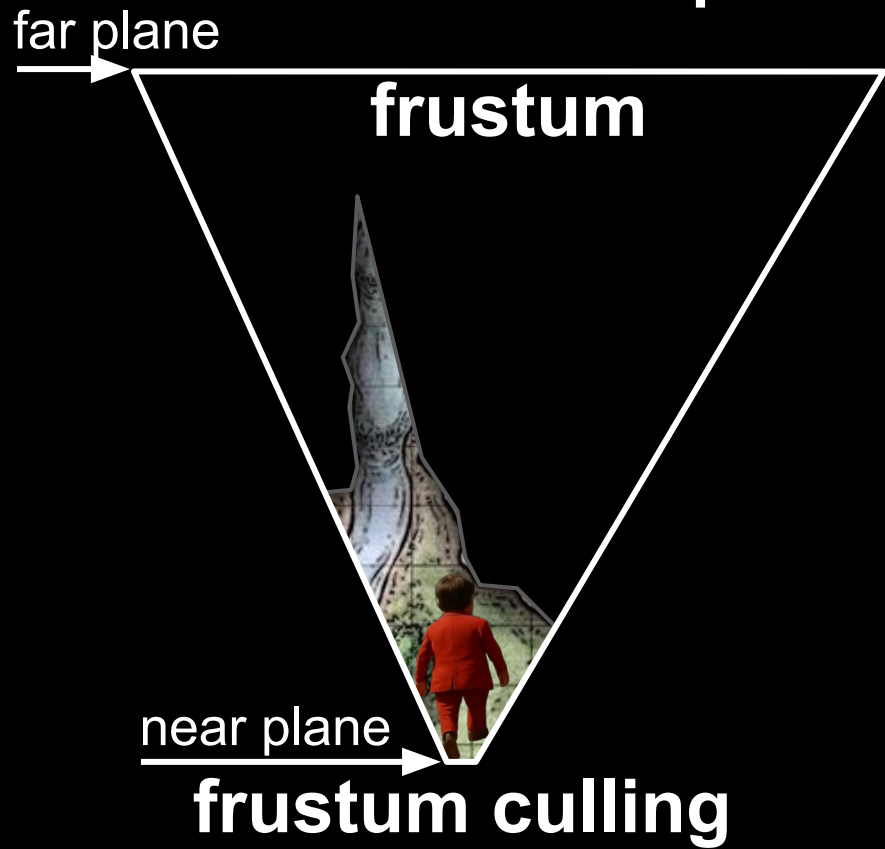
AABB



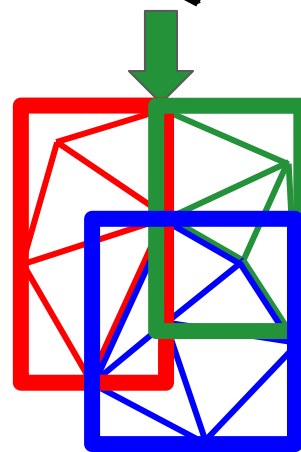
объединим в кластеры
по 128 треугольников



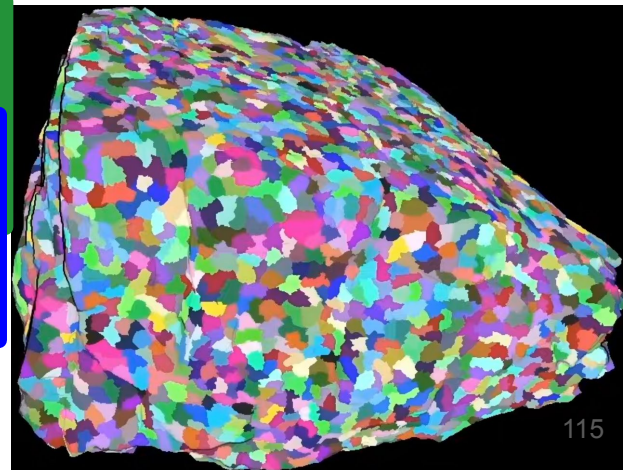
Теперь проецируем в экран **AABB**
каждого кластера! И делаем frustum culling!
x128 speedup!



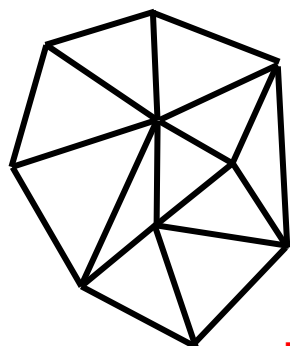
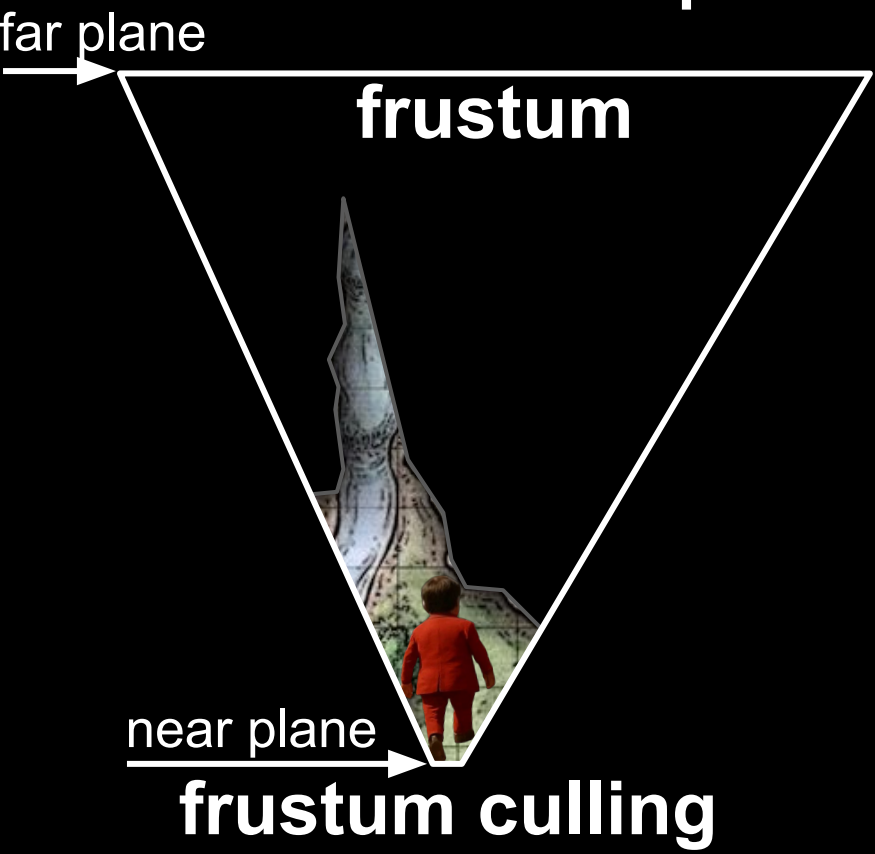
объединим в кластеры
по 128 треугольников



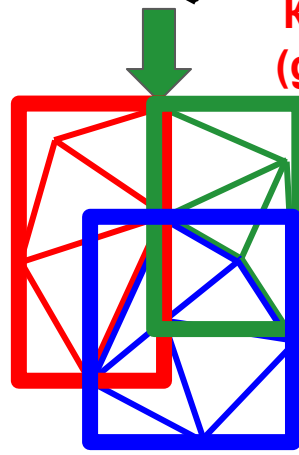
AABB



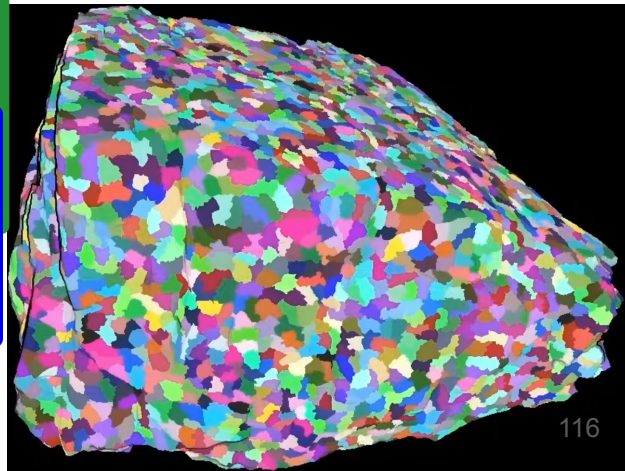
Теперь проецируем в экран **AABB**
каждого кластера! И делаем frustum culling!
x128 speedup!



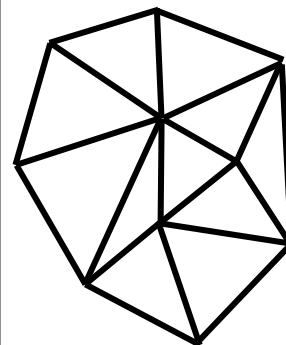
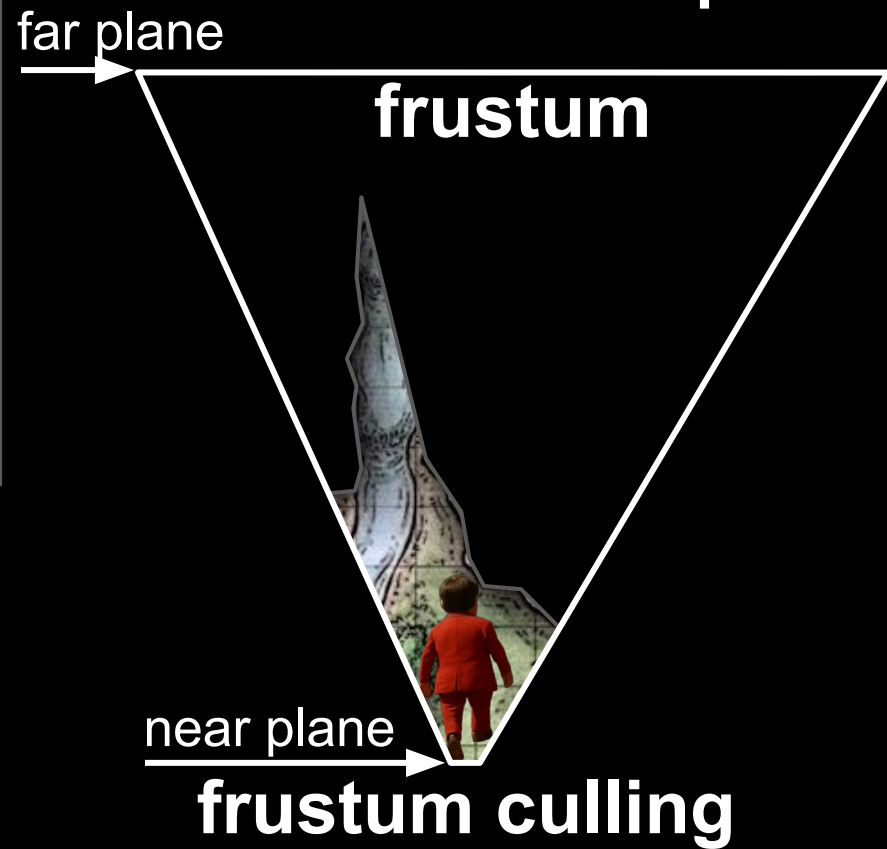
Как реализовать?
(graph partitioning)



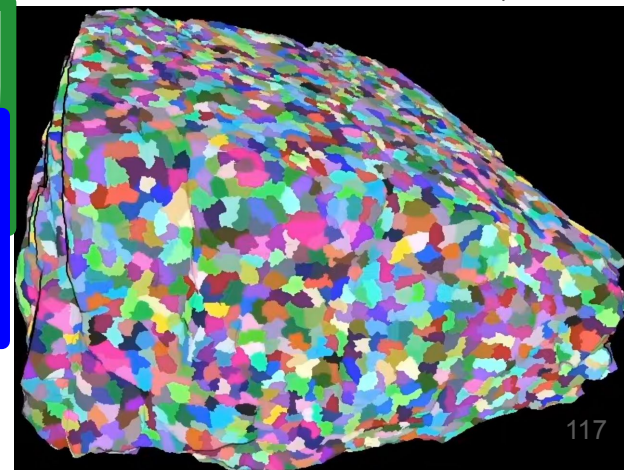
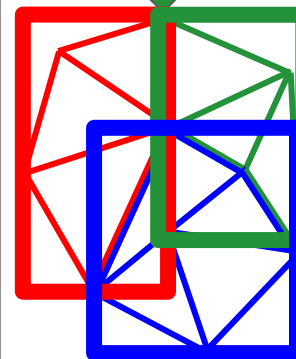
AABB

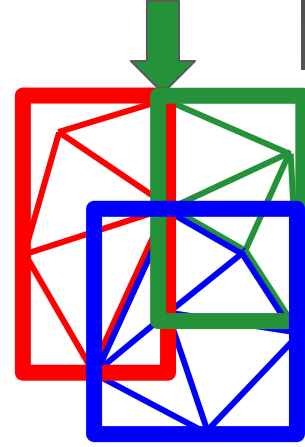
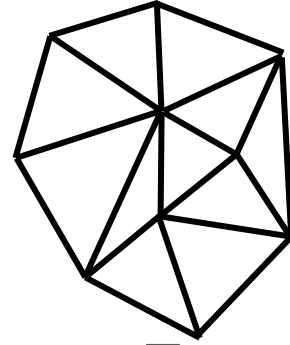


Теперь проецируем в экран **AABB**
каждого кластера! И делаем frustum culling!
x128 speedup!



Например graph partitioning
через **METIS**



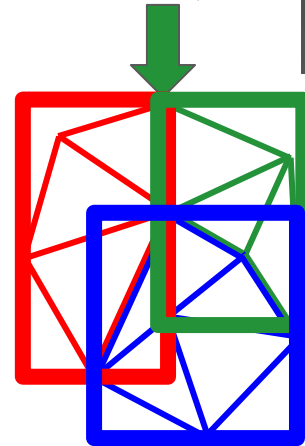
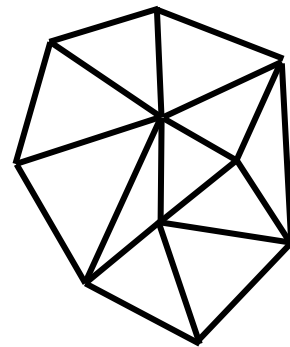


AABB

Depth framebuffer
Z framebuffer



fragment



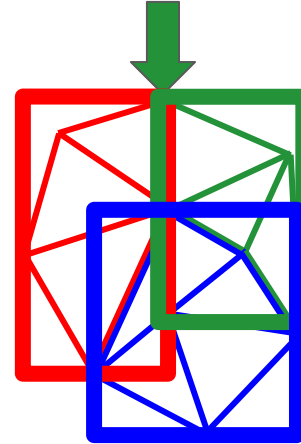
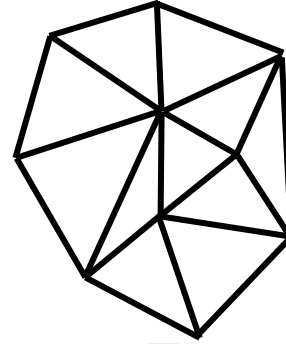
Depth framebuffer
Z framebuffer

Пусть Z-test через $\text{atomicMin64}(\text{RGB} + \text{Z})$



VS





Depth framebuffer
Z framebuffer

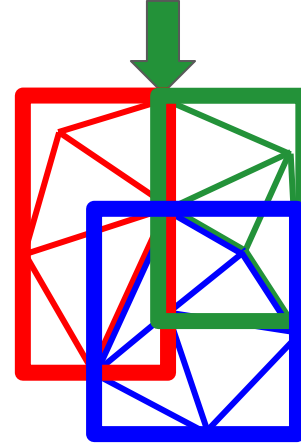
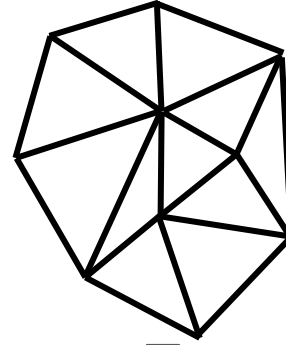
Пусть Z-test через $\text{atomicMin64}(\text{RGB} + \text{Z})$



VS



Можем ли мы как-то
ускорить Z test?



Depth framebuffer
Z framebuffer

Пусть Z-test через
 $\text{atomicMin64}(\text{RGB} + \text{Z})$

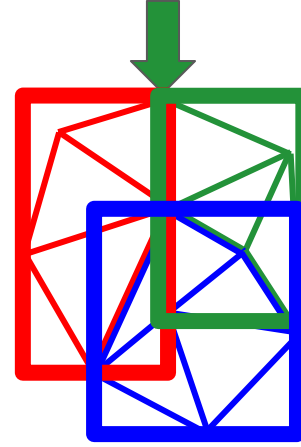
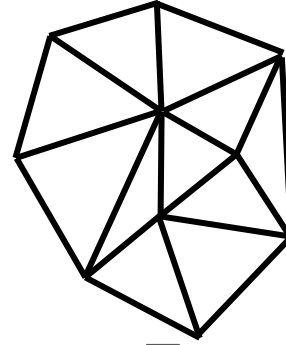


VS



Hierarcal Z-Buffer





AABB

Depth framebuffer
Z framebuffer

Пусть Z-test через
 $\text{atomicMin64}(\text{RGB} + \text{Z})$



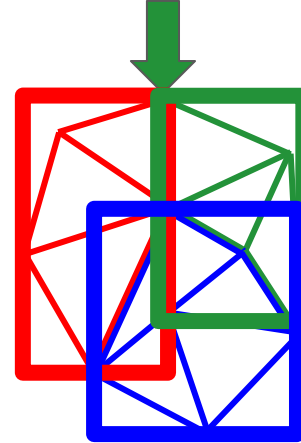
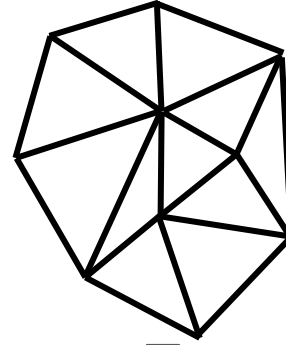
VS



Hierarcal Z-Buffer

x2

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



AABB делает **Z test** в **HZB**
на уровне где $\text{proj}(\text{AABB}) \sim 1 \text{ pixel}$!

Depth framebuffer
Z framebuffer

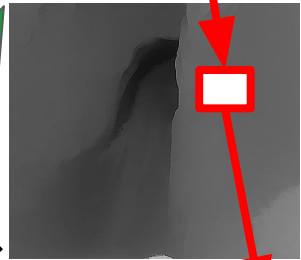
Пусть **Z-test** через
 $\text{atomicMin64}(\text{RGB} + \text{Z})$



VS



Hierarcal Z-Buffer



RGB framebuffer (LEFT)



RGB framebuffer (RIGHT)



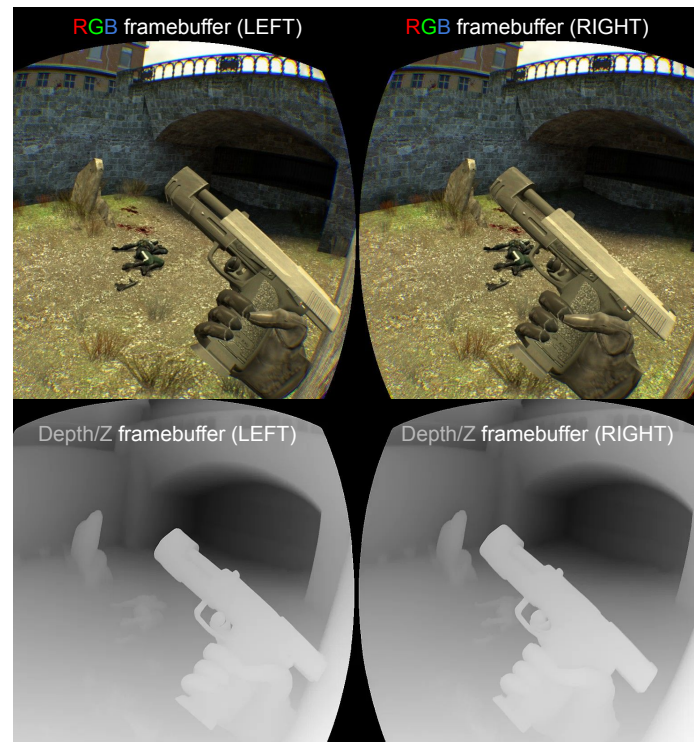
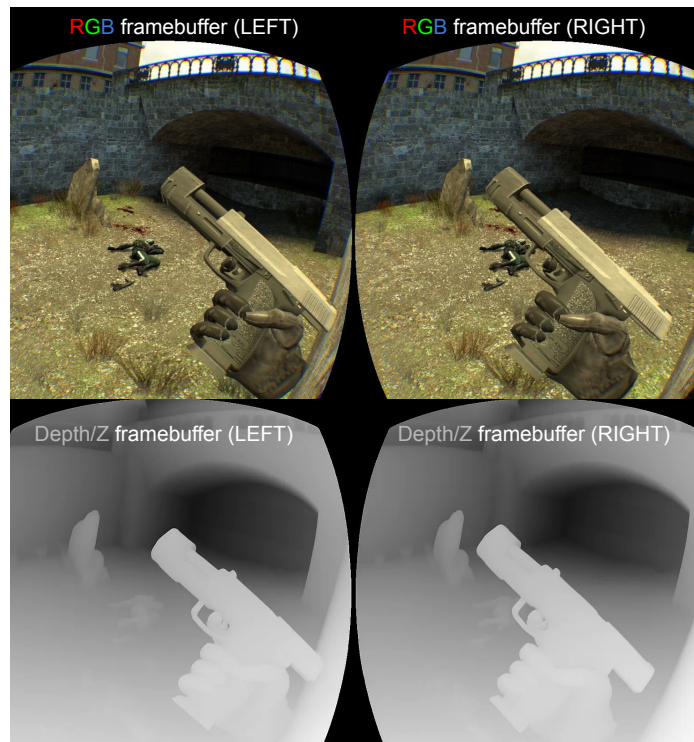
Depth/Z framebuffer (LEFT)



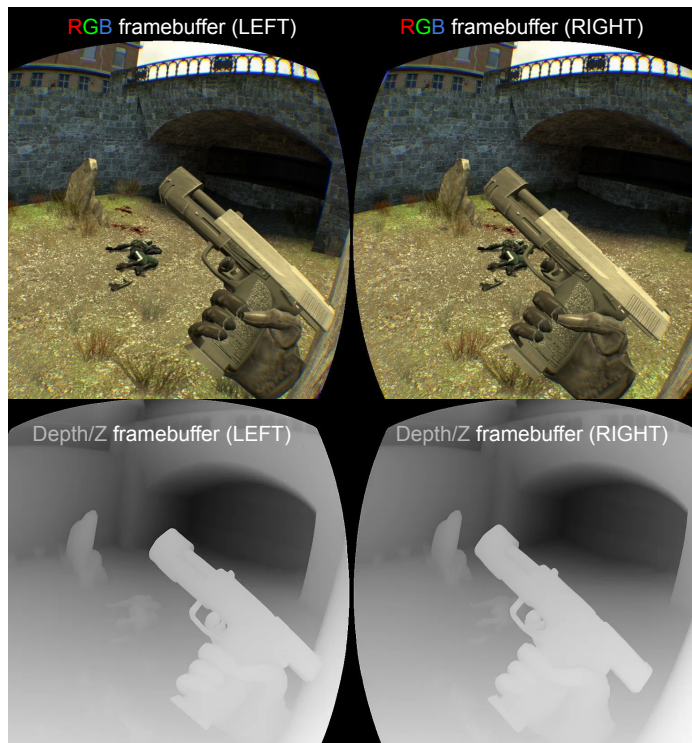
Depth/Z framebuffer (RIGHT)



Frame #i $\xrightarrow{\text{Игрок чуть повернул голову}}$ Frame #i+1
90+ FPS

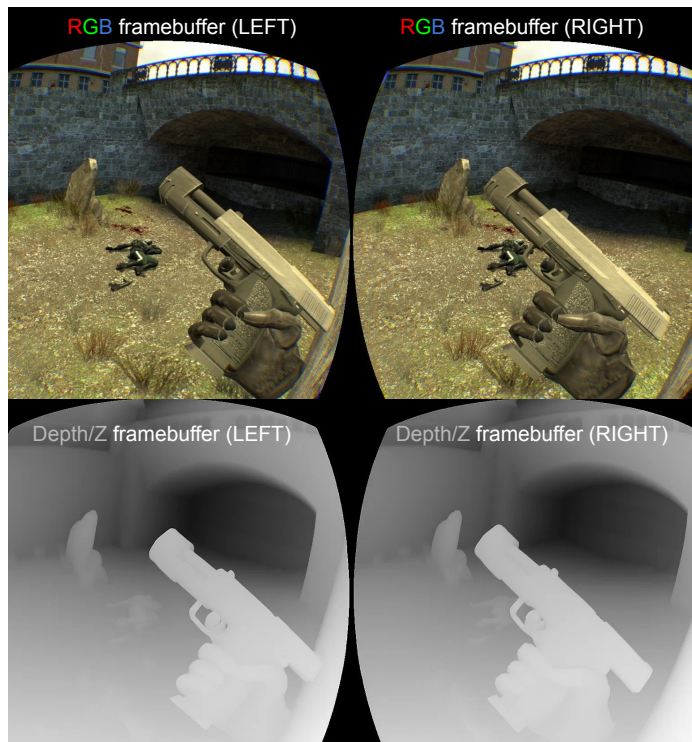


Frame #i $\xrightarrow{\text{Игрок чуть повернул голову}}$ Frame #i+1
кадр дольше 1/90 секунды
(например IO)



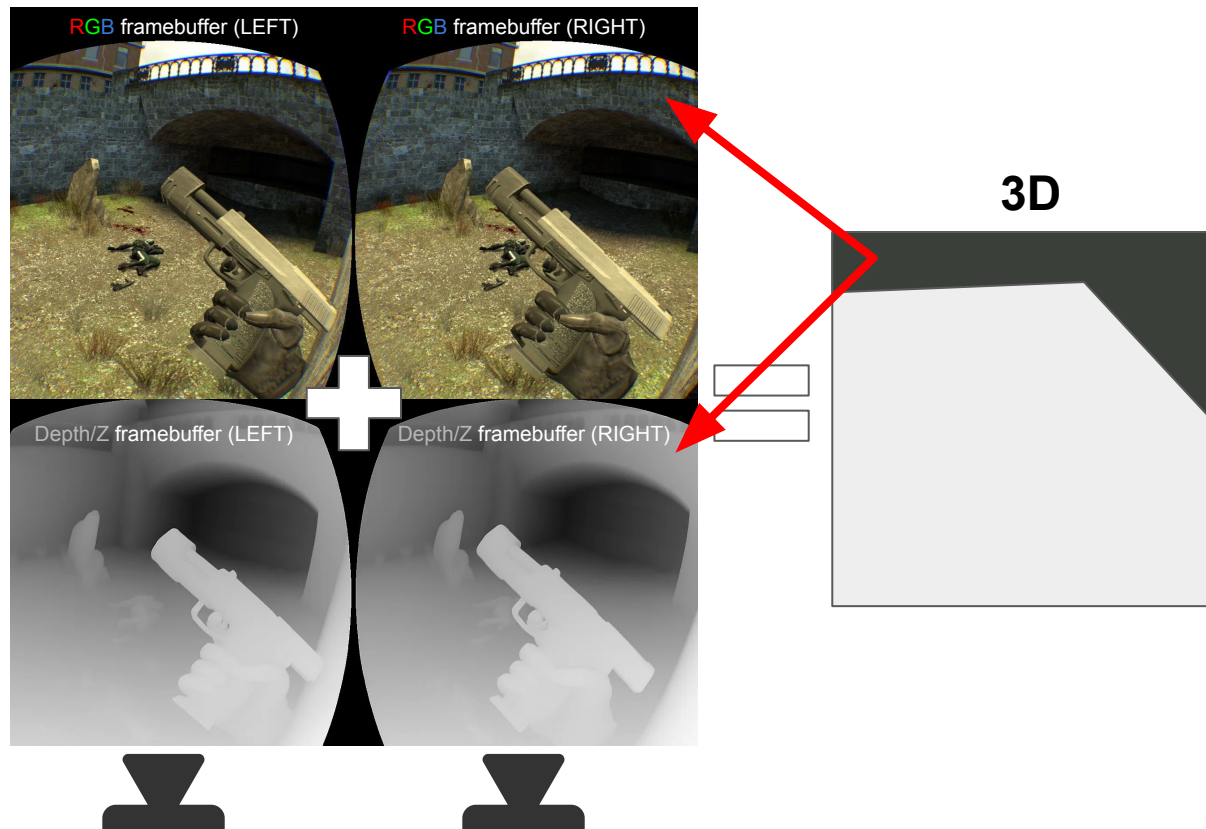
???

Frame #i $\xrightarrow{\text{Игрок чуть повернул голову}}$ Frame #i+1
кадр дольше 1/90 секунды
(например IO)

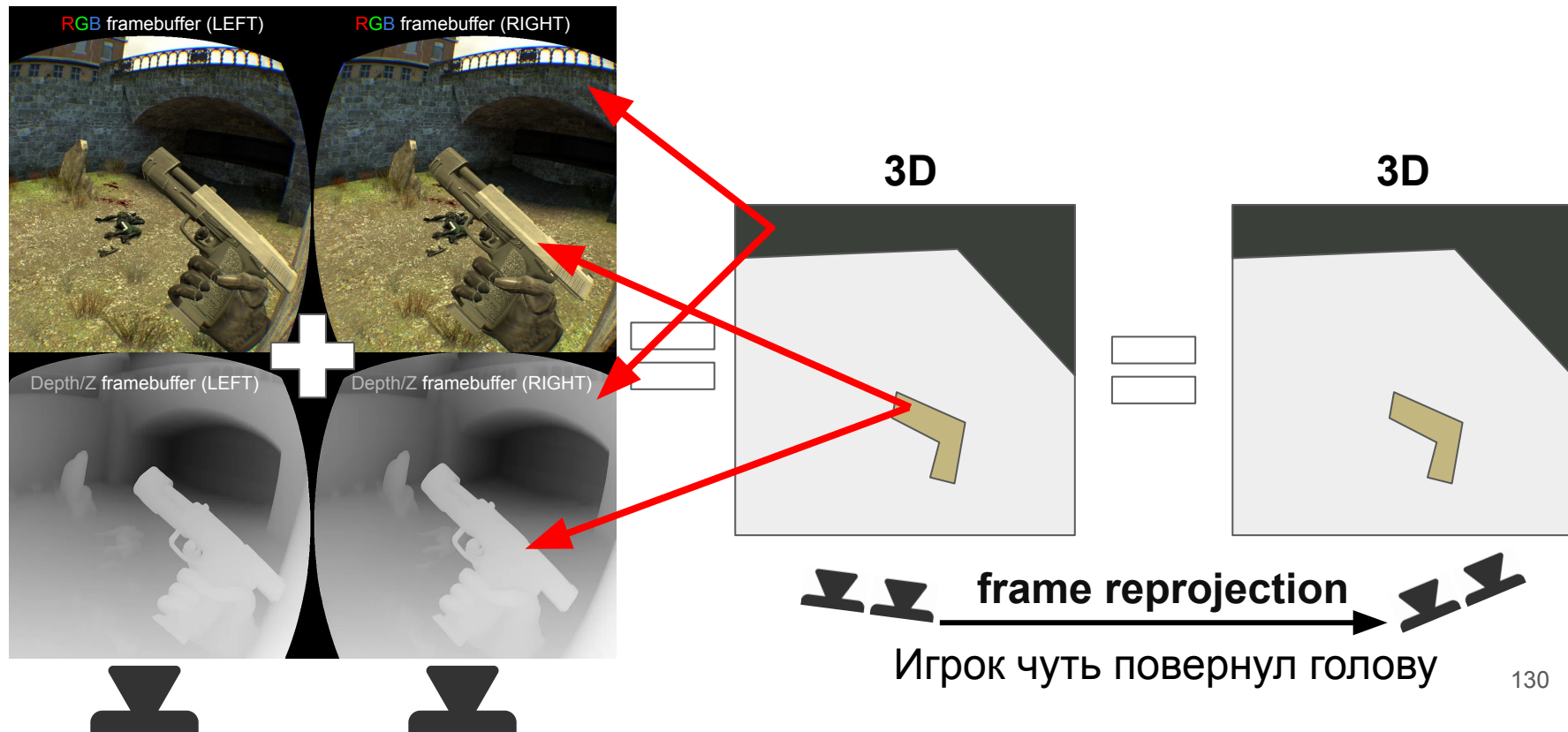


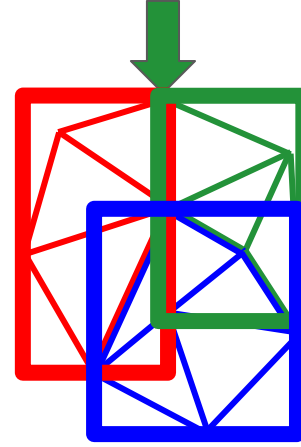
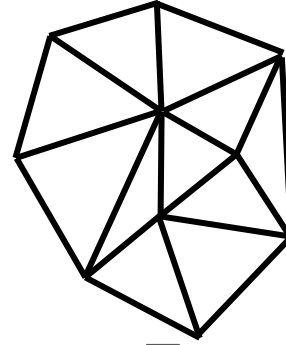
Как избежать?

Frame #i $\xrightarrow{\text{Игрок чуть повернул голову}}$ Frame #i+1
кадр дольше 1/90 секунды
(например IO)



Frame #i $\xrightarrow{\text{Игрок чуть повернул голову}}$ Frame #i+1
кадр дольше 1/90 секунды
(например IO)





Depth framebuffer
Z framebuffer

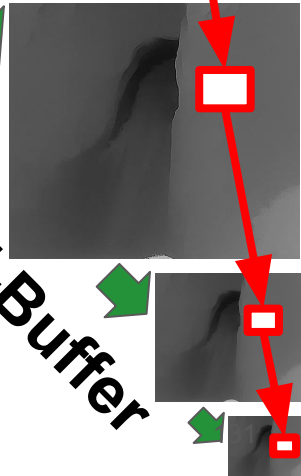
Пусть Z-test через
 $\text{atomicMin64}(\text{RGB} + \text{Z})$

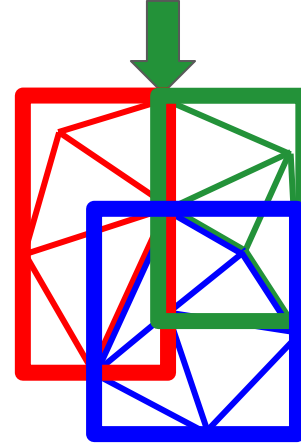
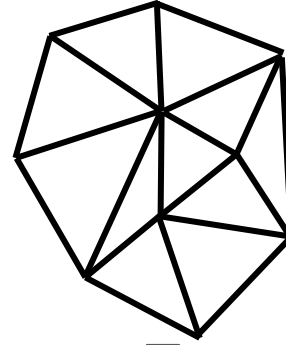


VS



Hierarcal Z-Buffer





AABB

Рисуем кадр, откуда взять HZB?

Depth framebuffer
Z framebuffer

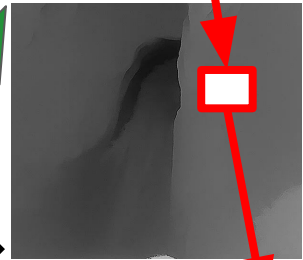
Пусть Z-test через
 $\text{atomicMin64}(\text{RGB} + \text{Z})$



VS



Hierarcal Z-Buffer





Камера двинулась влево

Depth framebuffer
Z framebuffer

x2
Hierarcal Z-Buffer

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

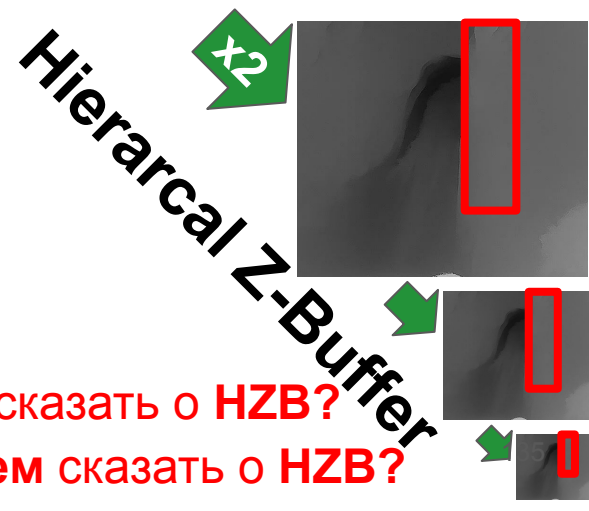


Depth framebuffer
Z framebuffer

x2
Hierarcal Z-Buffer

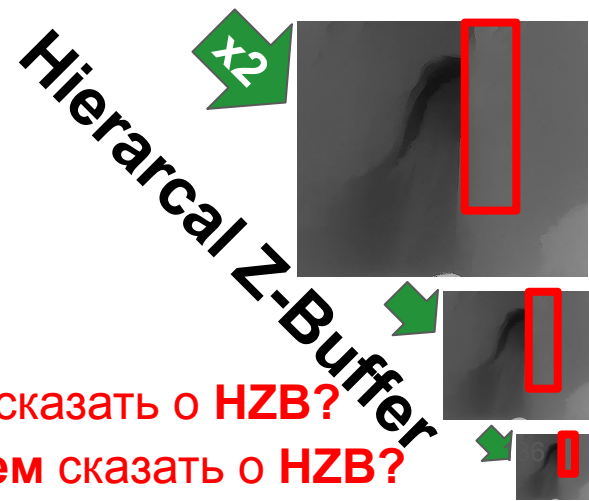
Что мы **можем** сказать о HZB?

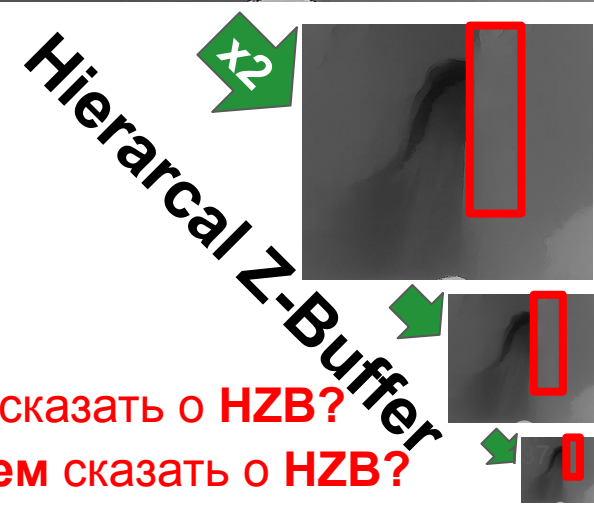
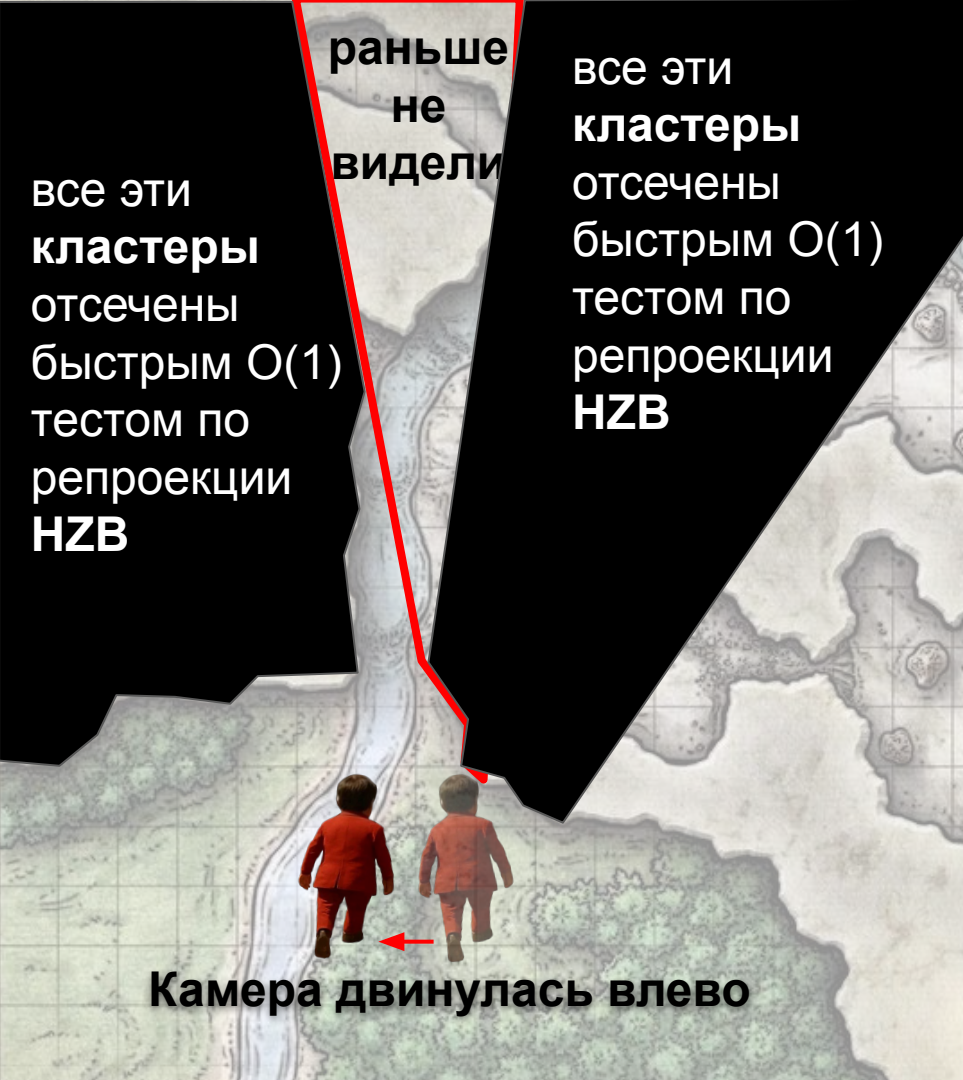
Что мы **не можем** сказать о HZB?



Что мы **можем** сказать о HZB?

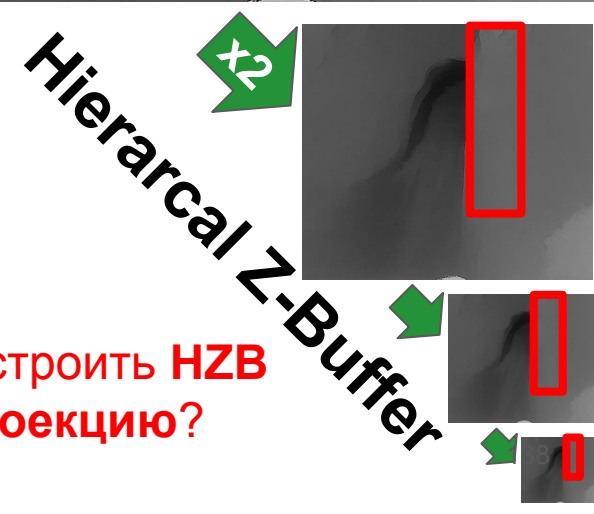
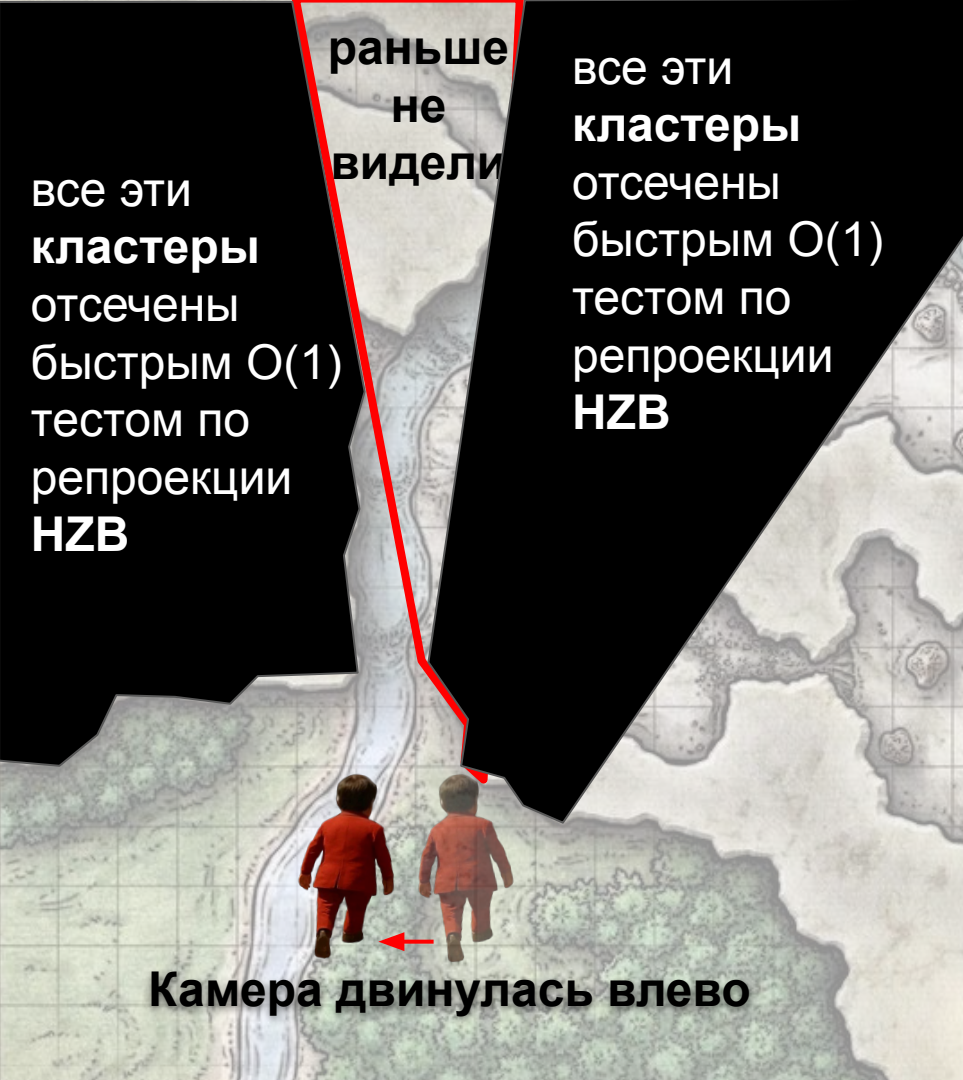
Что мы **не можем** сказать о HZB?



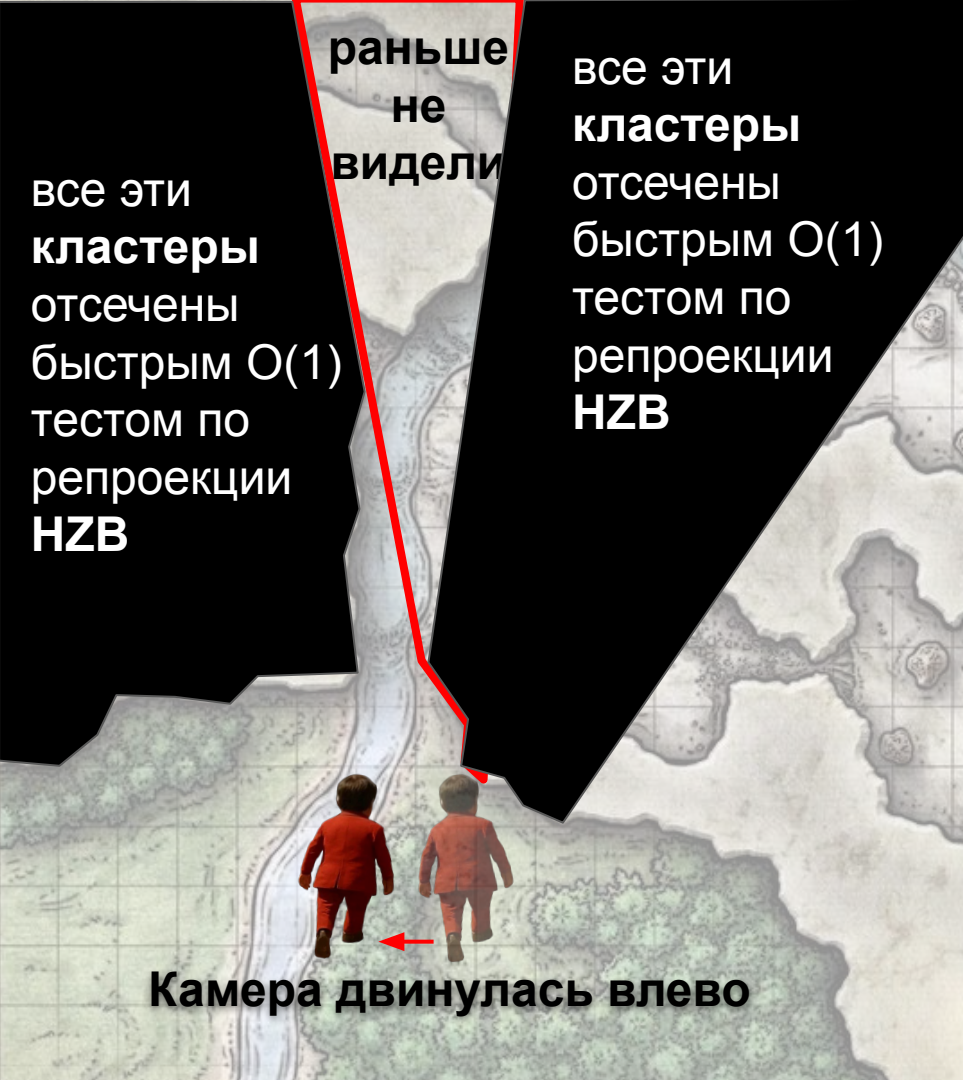


Что мы **можем** сказать о HZB?

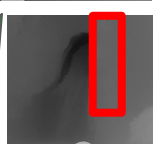
Что мы **не можем** сказать о HZB?

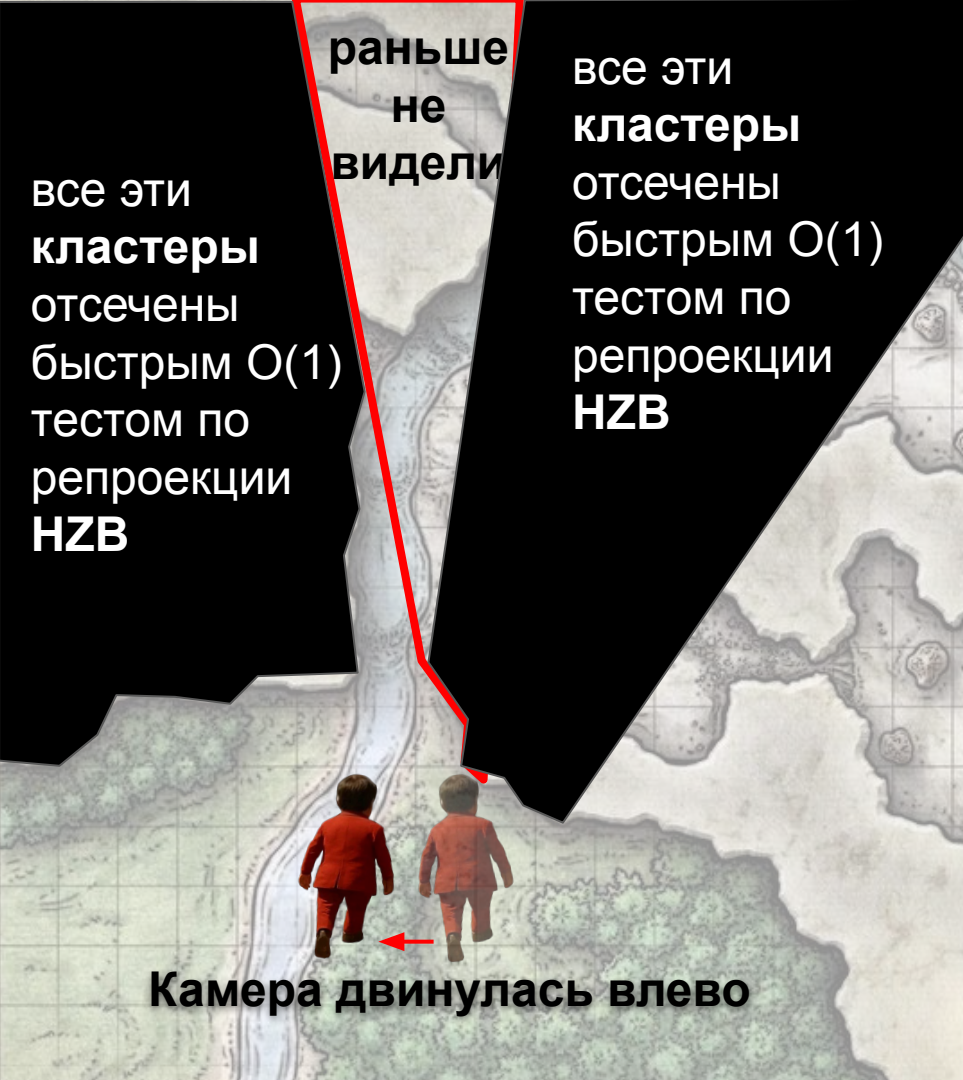


Как быстро построить **HZB**
не через репроекцию?

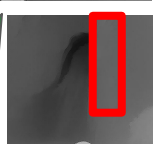


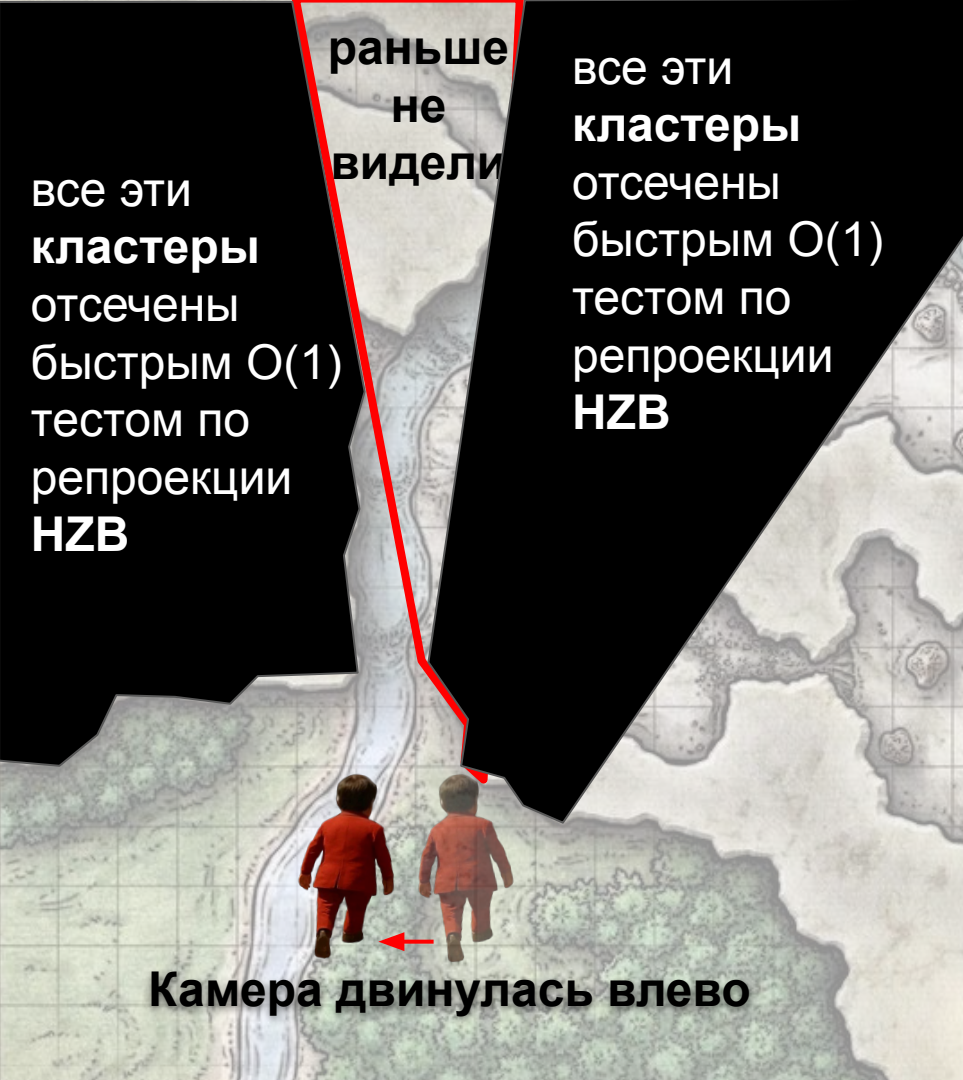
- 1) Берем кластеры попавшие в прошлый framebuffer (сколько их?)





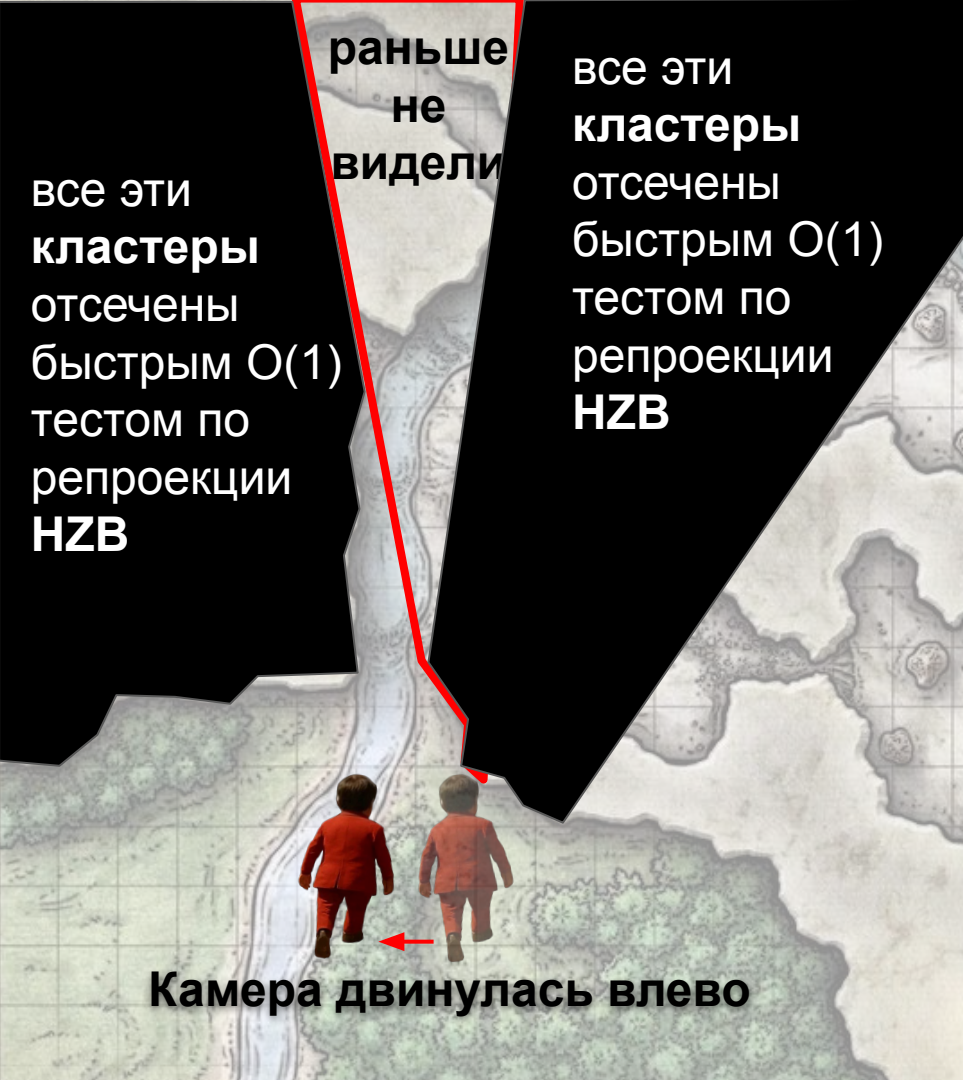
1) Берем кластеры попавшие в прошлый framebuffer - $O(\text{screen size})$



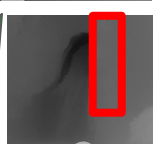


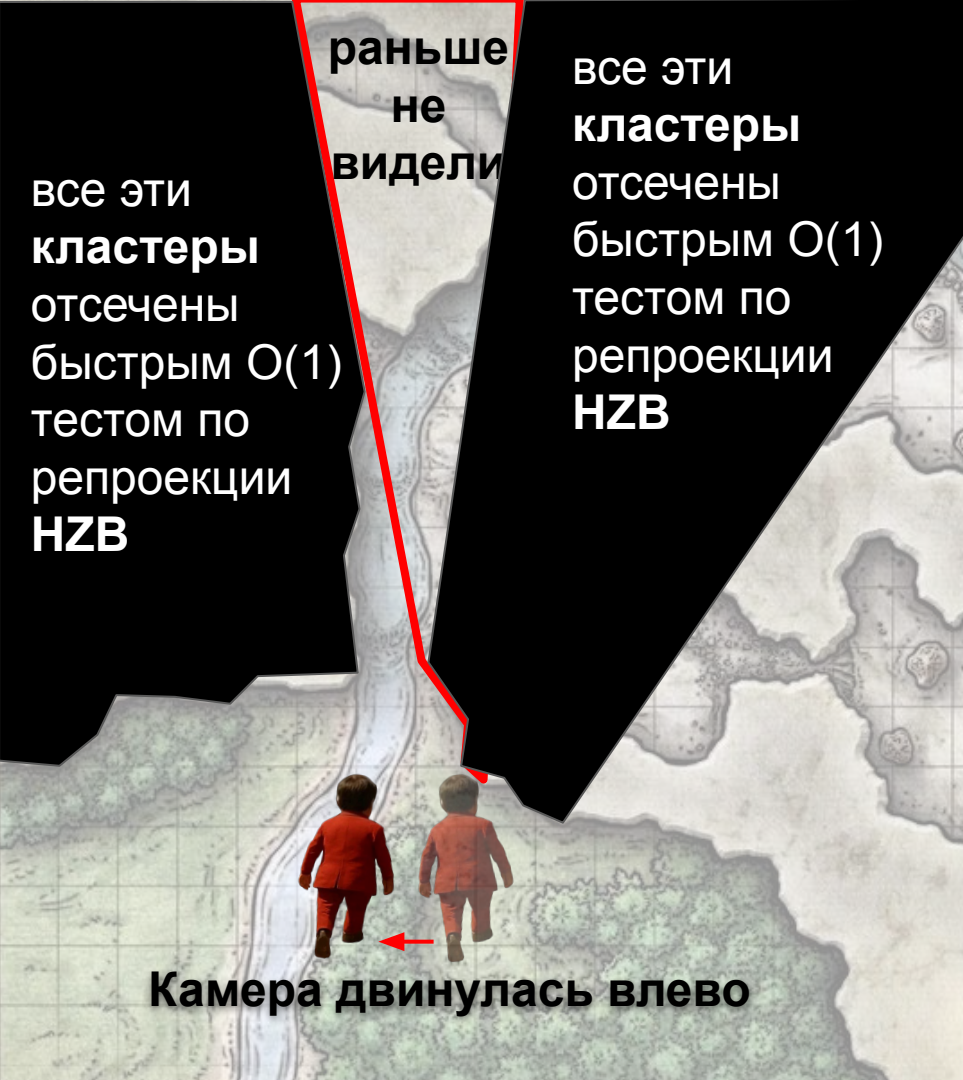
- 1) Берем кластеры попавшие в прошлый framebuffer - $O(\text{screen size})$
- 2) Растеризуем их = HZB.
(полный? точный?)





- 1) Берем кластеры попавшие в прошлый framebuffer - $O(\text{screen size})$
- 2) Растеризуем их = $\sim HZB$
- 3) Растеризуем остальных. Очень быстро т.к. ???



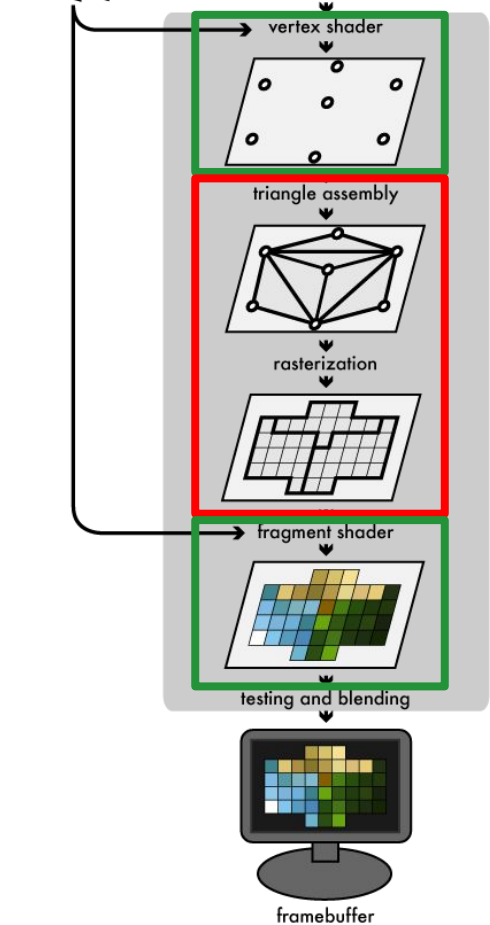
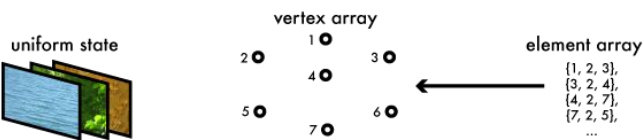


- 1) Берем кластеры попавшие в прошлый framebuffer - $O(\text{screen size})$
- 2) Растеризуем их = $\sim HZB$
- 3) Растеризуем остальных. Очень быстро т.к. быстрая отсечка по $\sim HZB$!



Перерыв!





**programmable
vertex shader**

**fixed pipeline
(hardware)**

**programmable
fragment shader:**

- UV *texture sampling*
- shadows/lighting

output:

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

Depth framebuffer
Z framebuffer

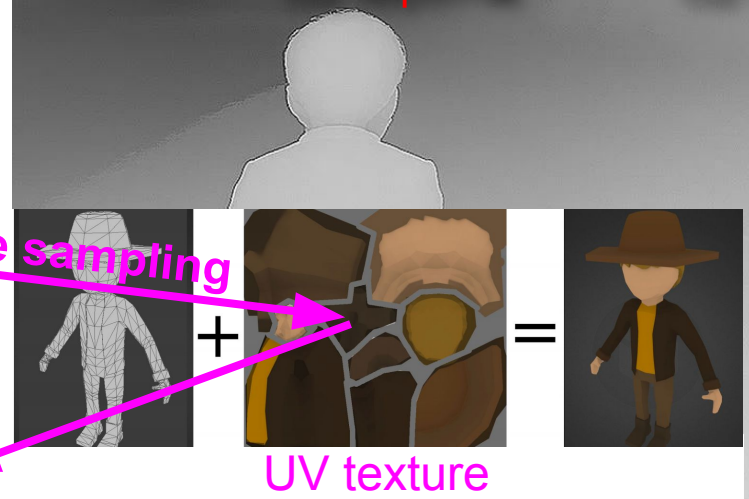


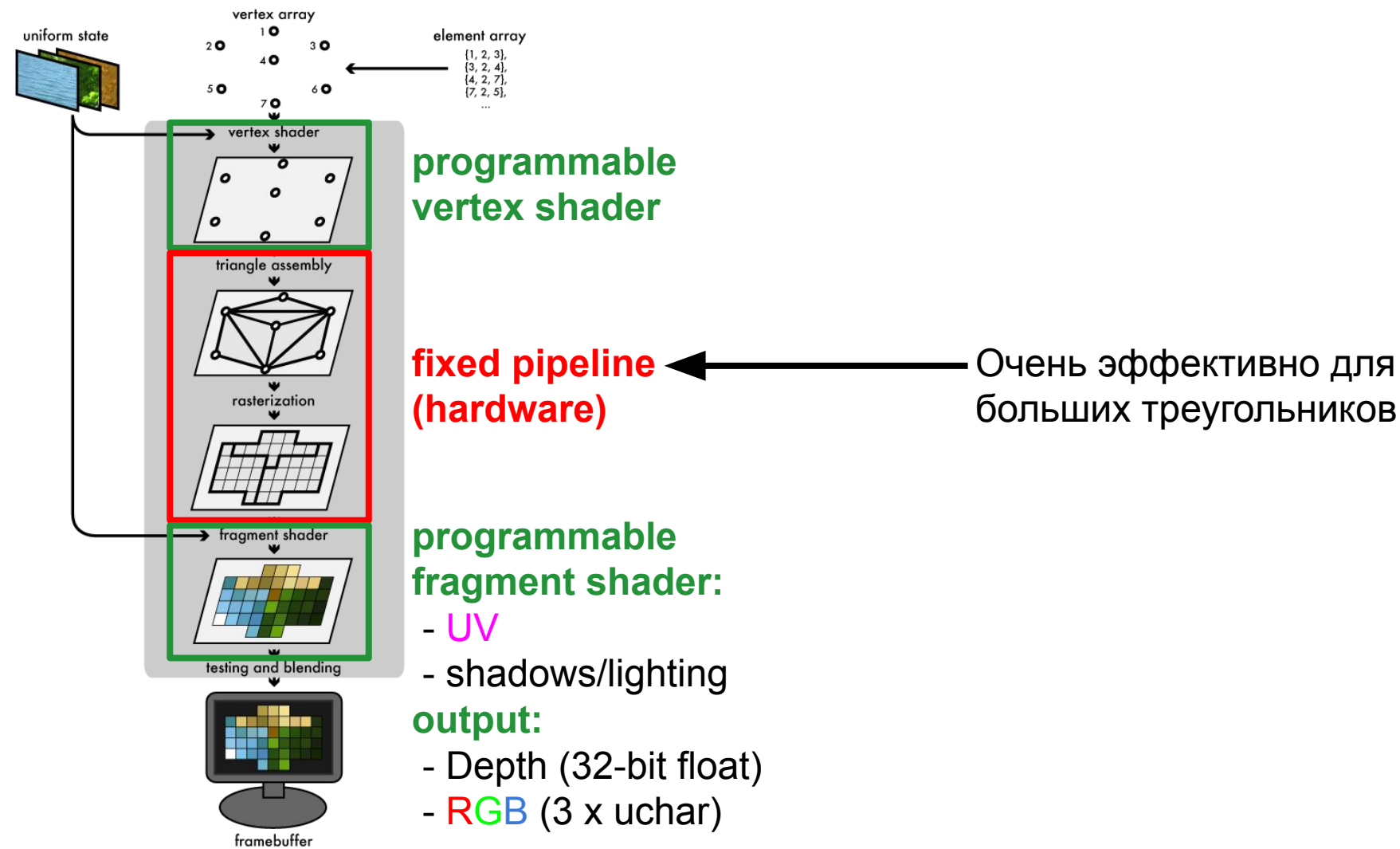
f32 depth=A

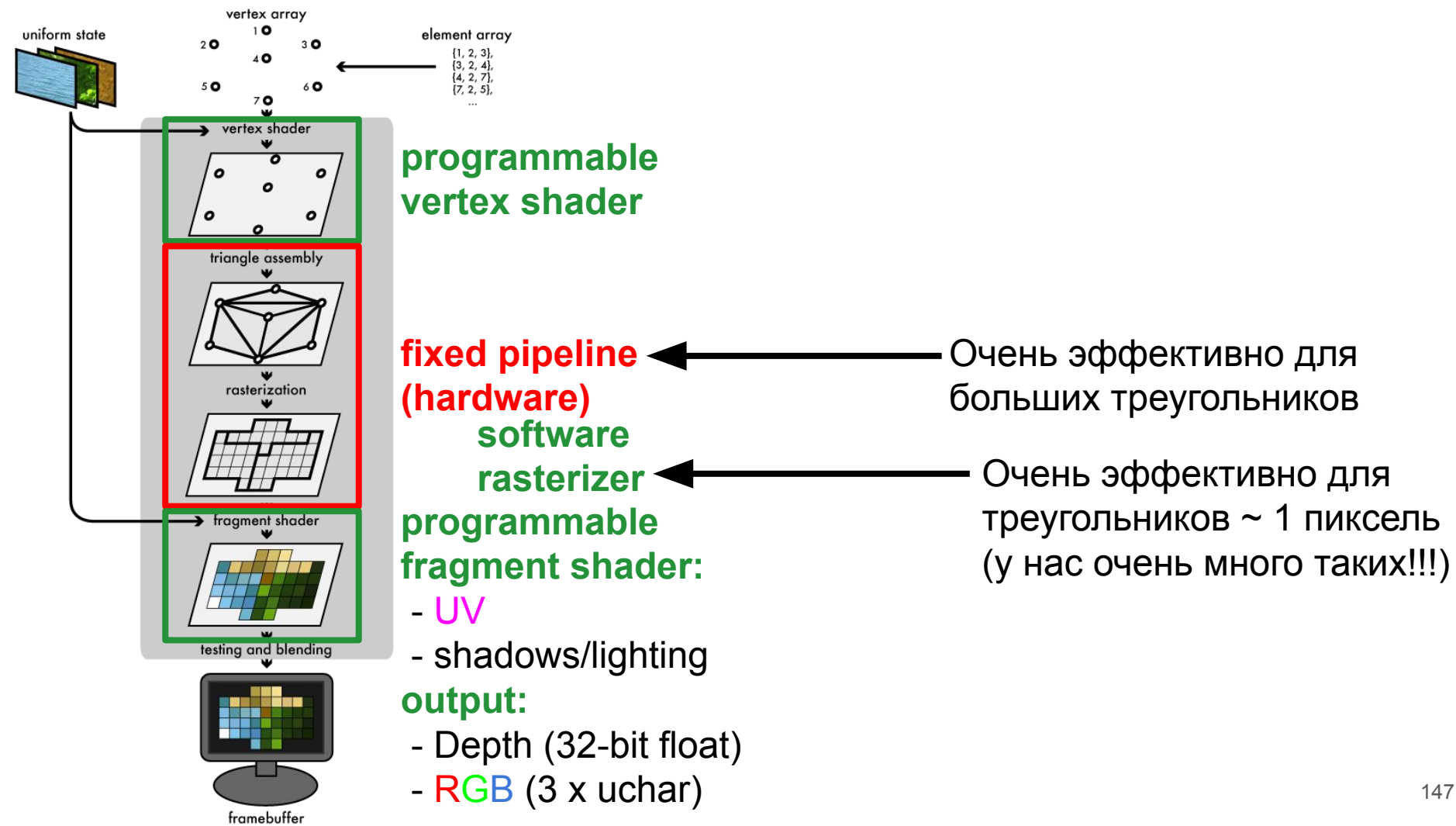


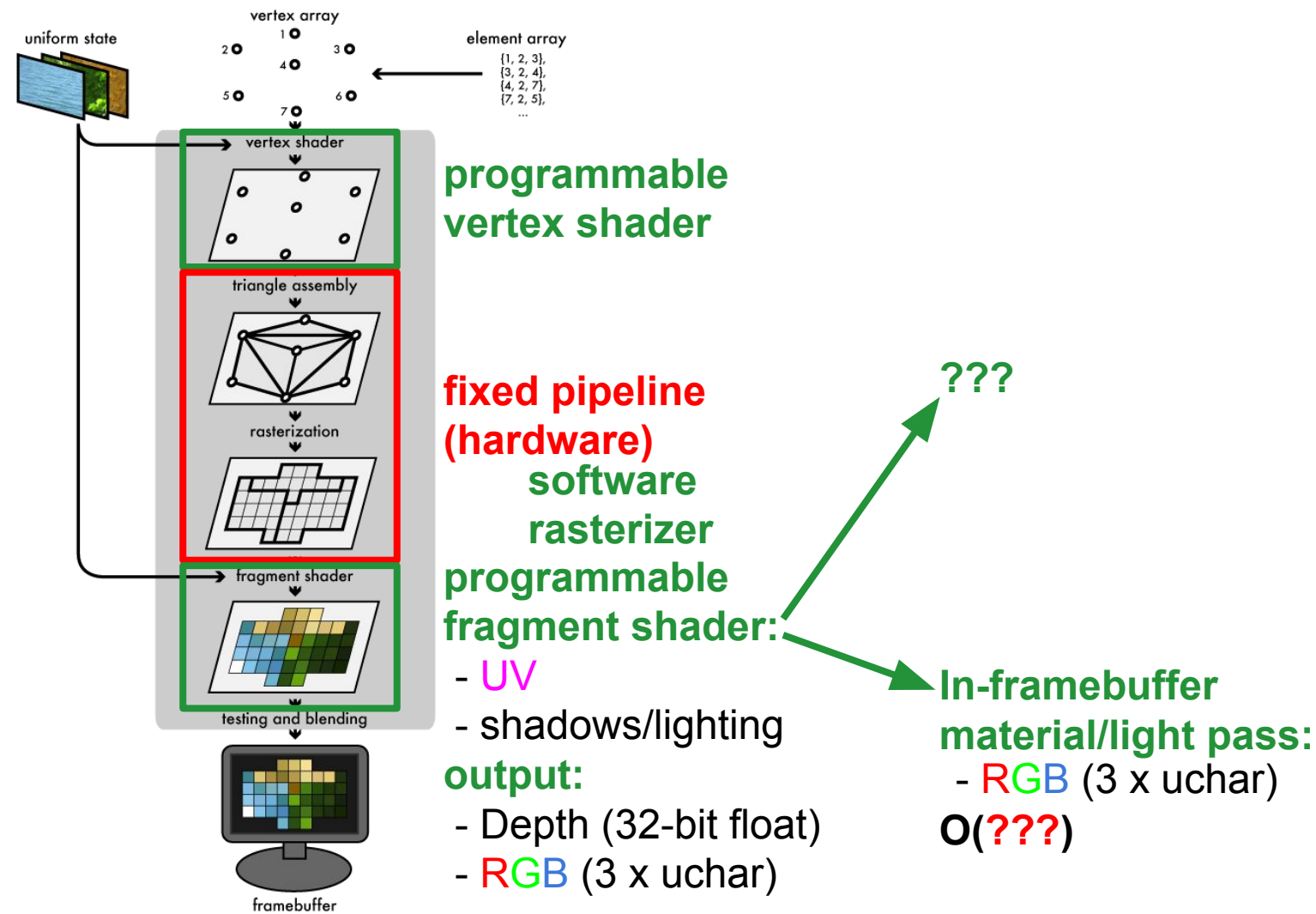
f32 depth=B

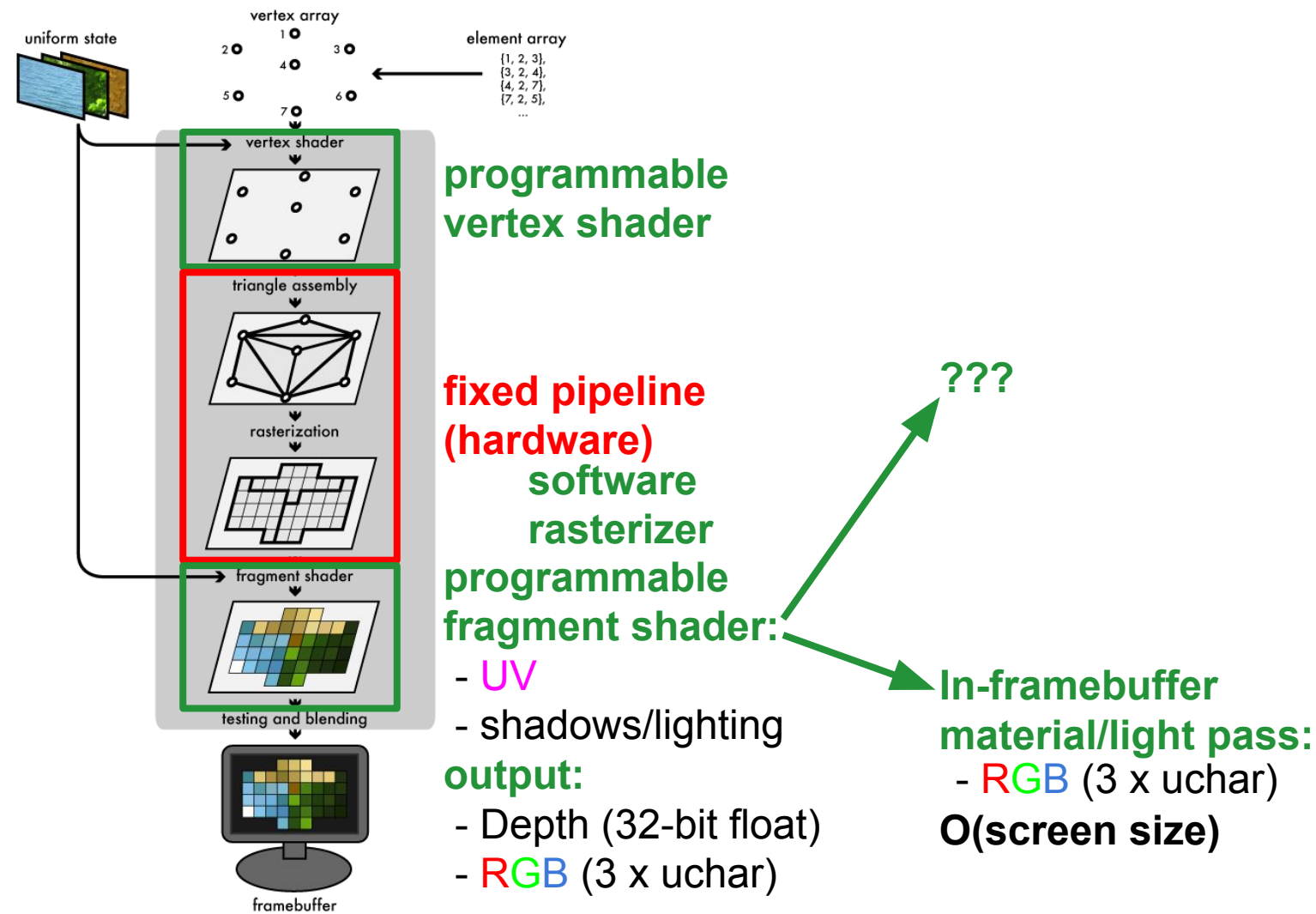
VS

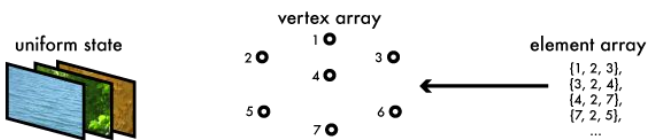












programmable
vertex shader

fixed pipeline
(hardware)

software
rasterizer

programmable
fragment shader:

- UV
- shadows/lighting

output:

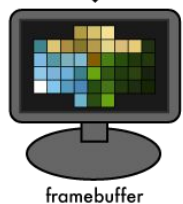
- Depth (32-bit float)
- RGB (3 x uchar)

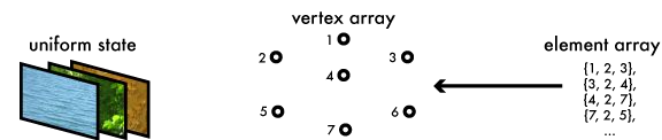
Какой минимум в каждый
пиксель записать?

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)

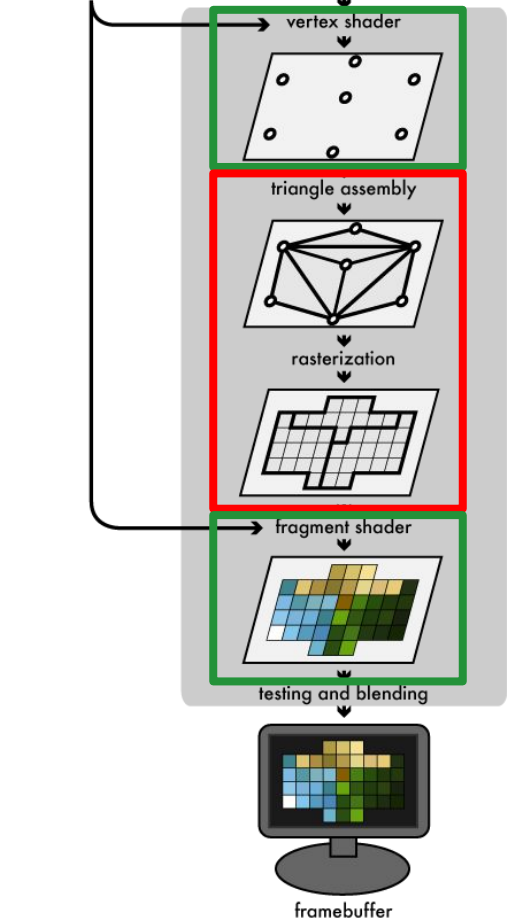
Deferred rendering/shading

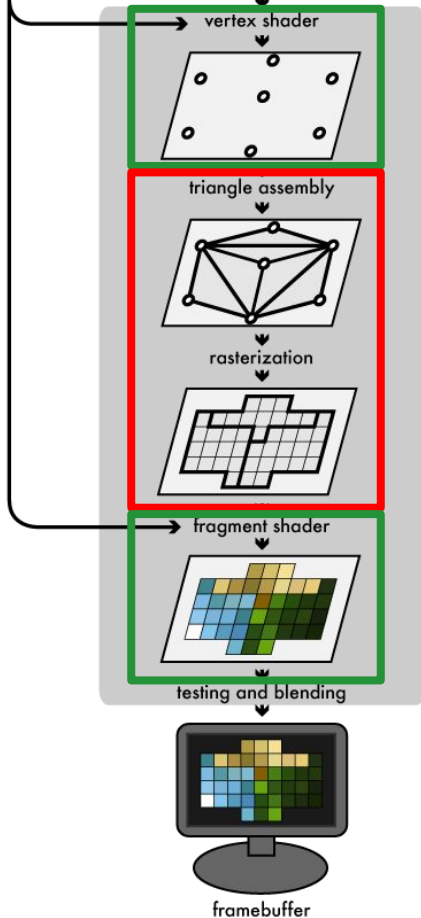
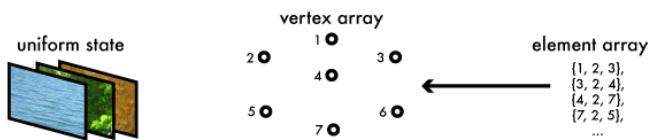
Geometry pass:

- 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)

Deferred rendering/shading

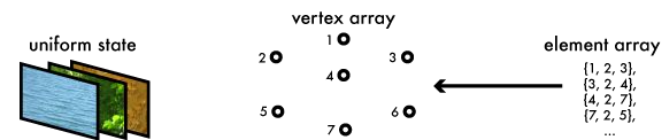
Geometry pass:

- depth
- face index

In-framebuffer
material/light pass:

- RGB (3 x uchar)

O(screen size)



programmable
vertex shader

Deferred rendering/shading

Как совместить?

fixed pipeline
(hardware)

+ software
rasterizer

programmable
fragment shader:

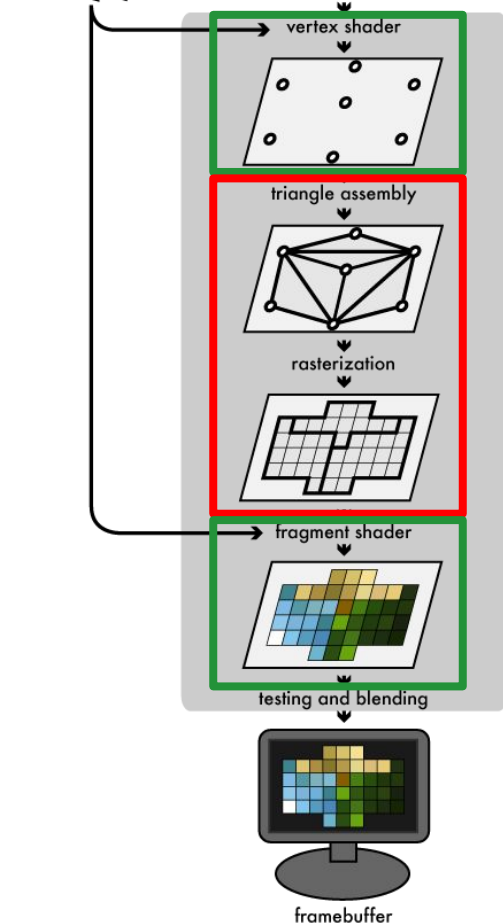
- UV
 - shadows/lighting
- output:
- Depth (32-bit float)
 - RGB (3 x uchar)

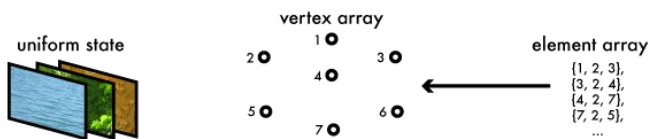
Geometry pass:

- depth
- face index

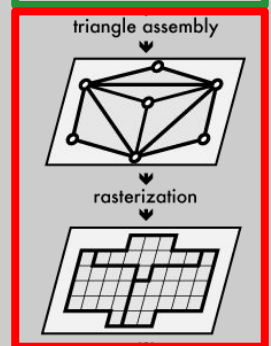
In-framebuffer
material/light pass:

- 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)

Deferred rendering/shading

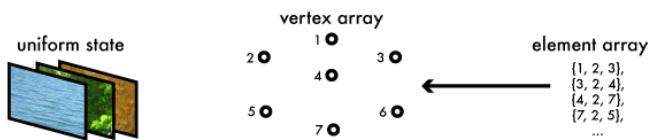
Geometry pass:

- depth
 - face index
- $O(???)$**

**In-framebuffer
material/light pass:**

- RGB (3 x uchar)
- $O(\text{screen size})$**





**programmable
vertex shader**

**fixed pipeline
(hardware)**

**software
rasterizer**

**programmable
fragment shader:**

- UV
- shadows/lighting

output:

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

Deferred rendering/shading

Geometry pass:

- depth
- face index

$O(???)$ - init HZB

$O(???)$ - rasterization

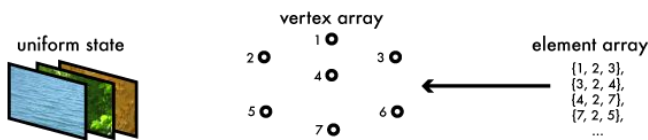
In-framebuffer

material/light pass:

- RGB (3 x uchar)

$O(\text{screen size})$





**programmable
vertex shader**

**fixed pipeline
(hardware)**

**software
rasterizer**

**programmable
fragment shader:**

- UV
 - shadows/lighting
- output:**
- Depth (32-bit float)
 - RGB (3 x uchar)

Deferred rendering/shading

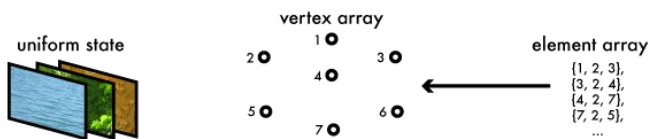
Geometry pass:

- depth
 - face index
- $O(\text{screen size})$ - init HZB**
 $O(???)$ - rasterization

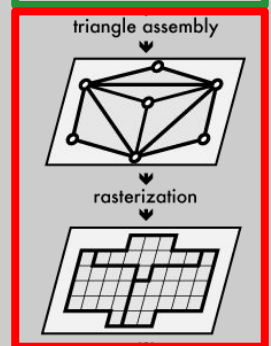
**In-framebuffer
material/light pass:**

- RGB (3 x uchar)
- $O(\text{screen size})$**





**programmable
vertex shader**



**fixed pipeline
(hardware)**
**software
rasterizer**

**programmable
fragment shader:**

- UV
- shadows/lighting

output:

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



Deferred rendering/shading

Geometry pass:

- depth
- face index

$O(\text{screen size})$ - init HZB

$O(\text{cluster count})$ - rasterization

В чем был секрет виртуальных текстур?

In-framebuffer

material/light pass:

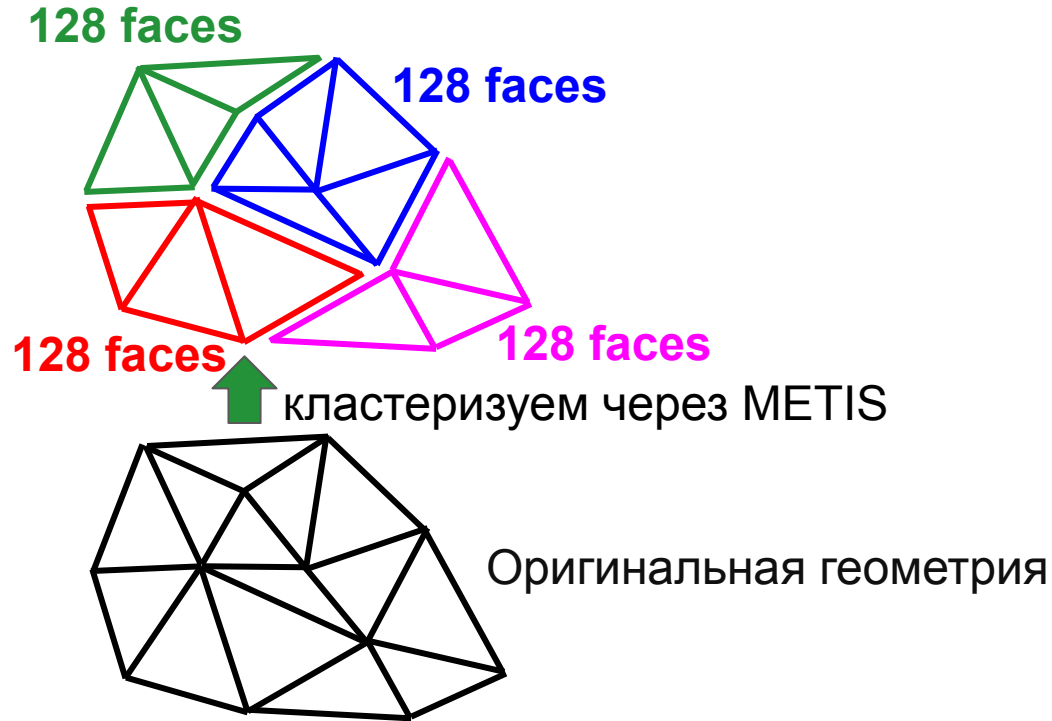
- RGB (3 x uchar)

$O(\text{screen size})$

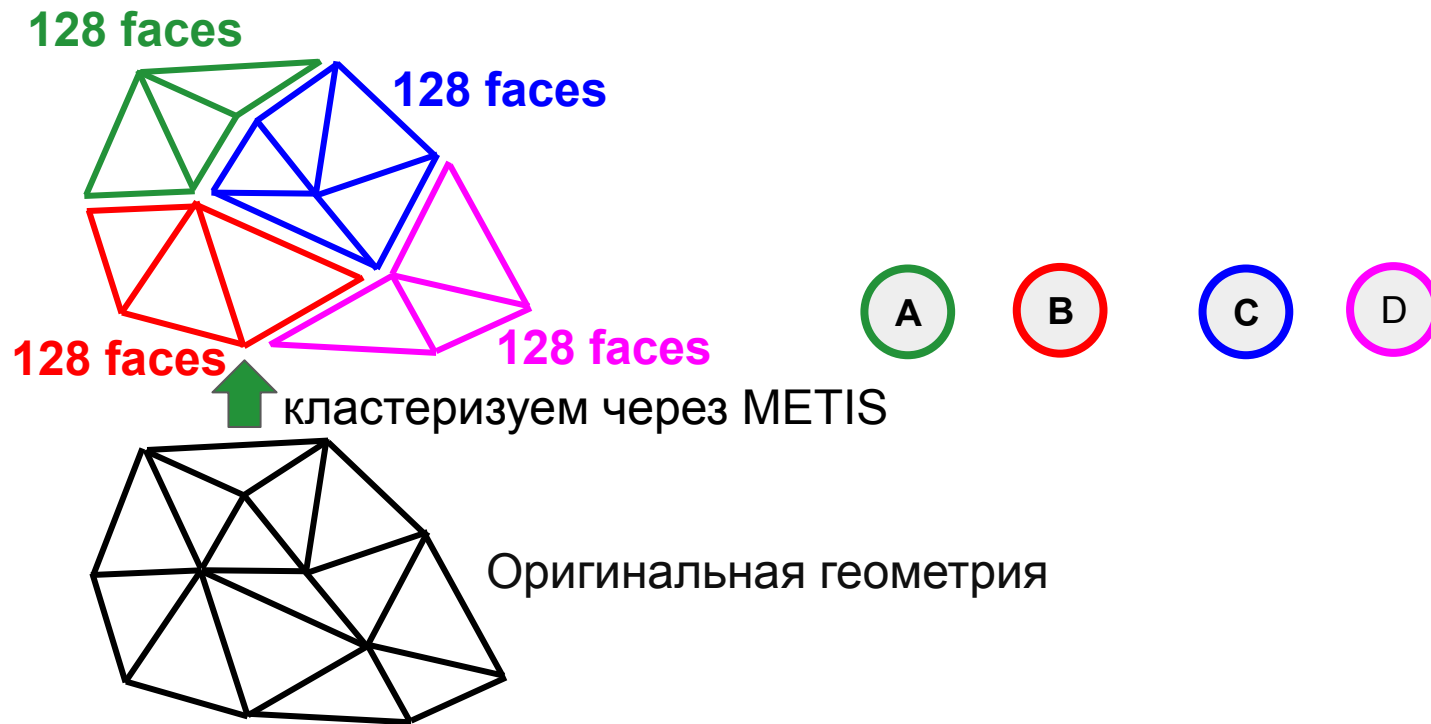
Geometry clusters hierarchy



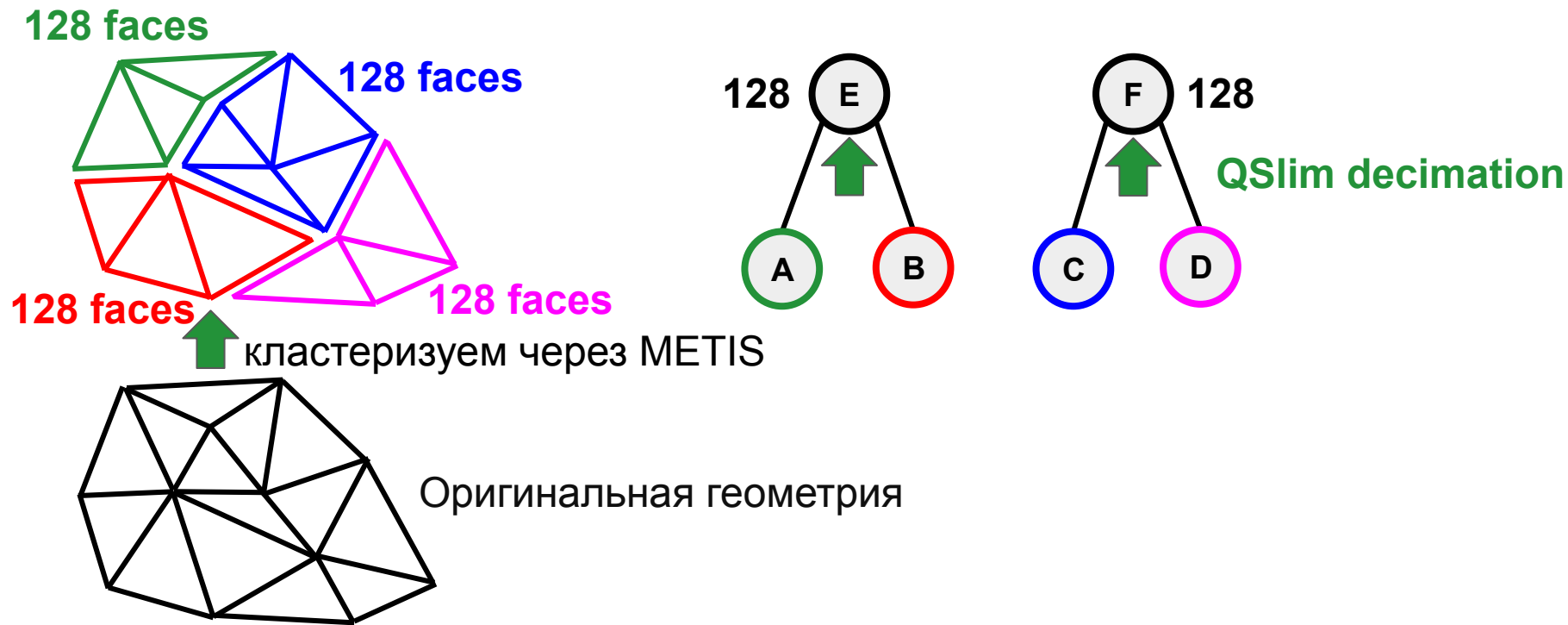
Geometry clusters hierarchy



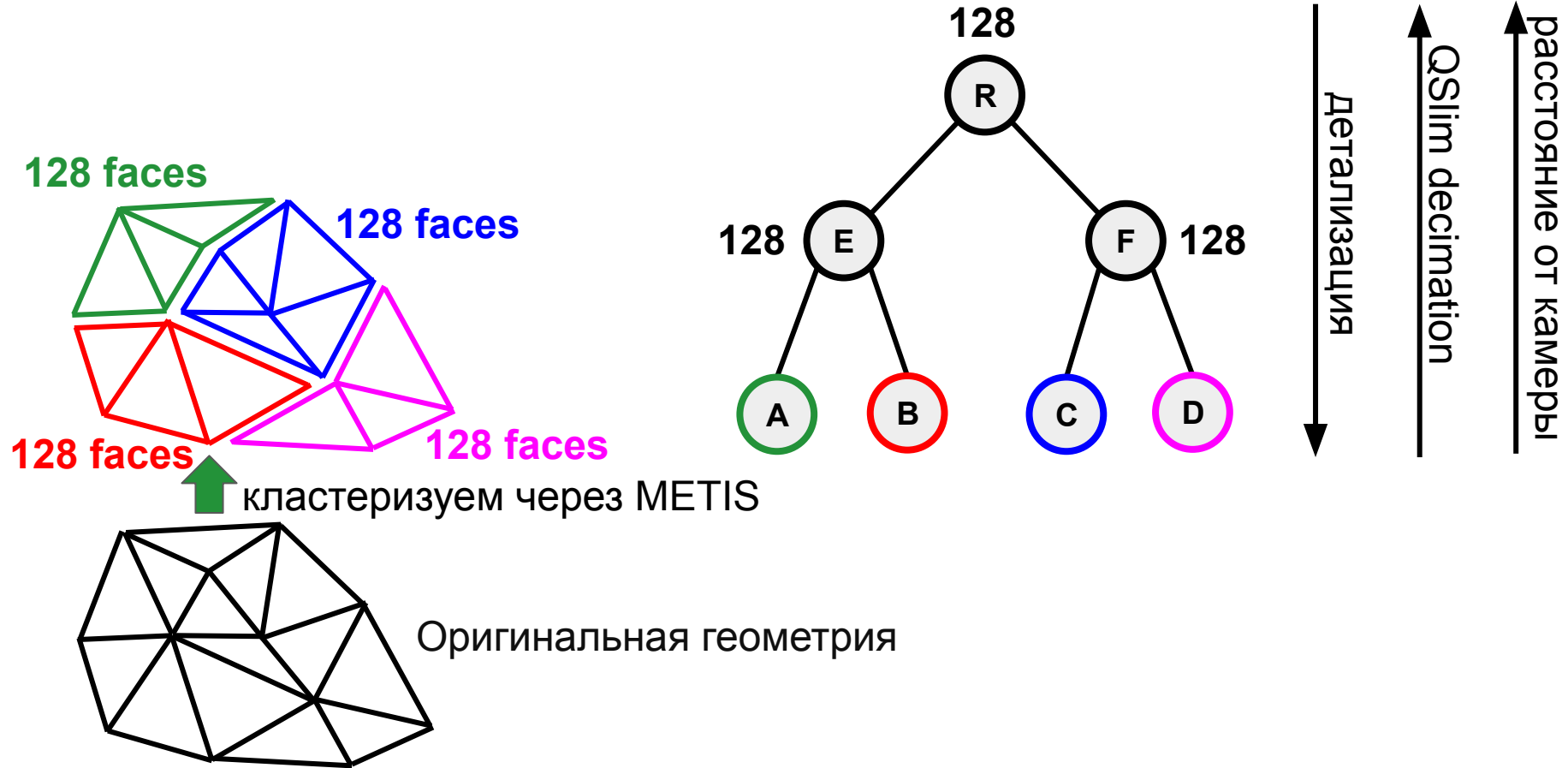
Geometry clusters hierarchy



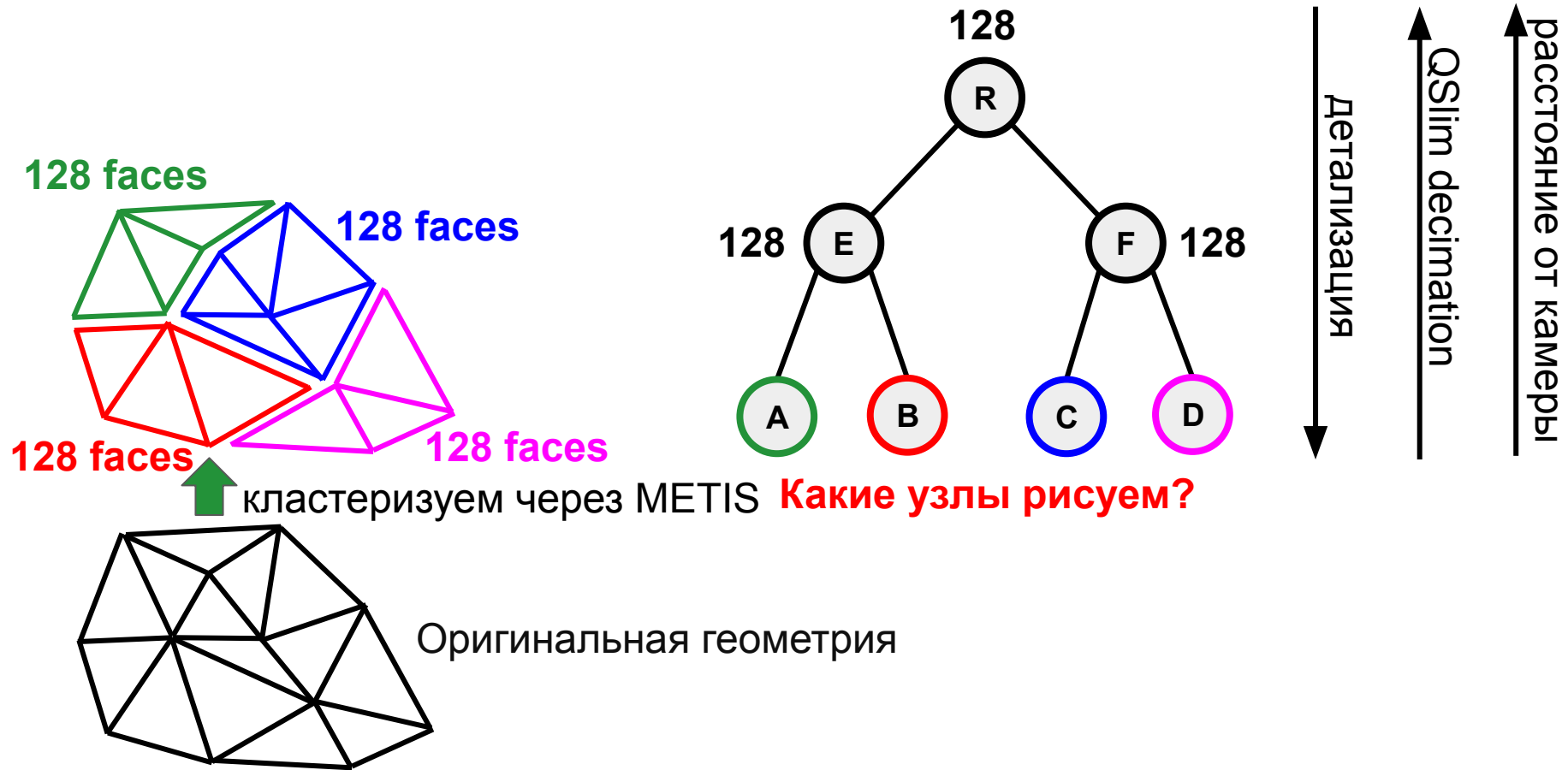
Geometry clusters hierarchy



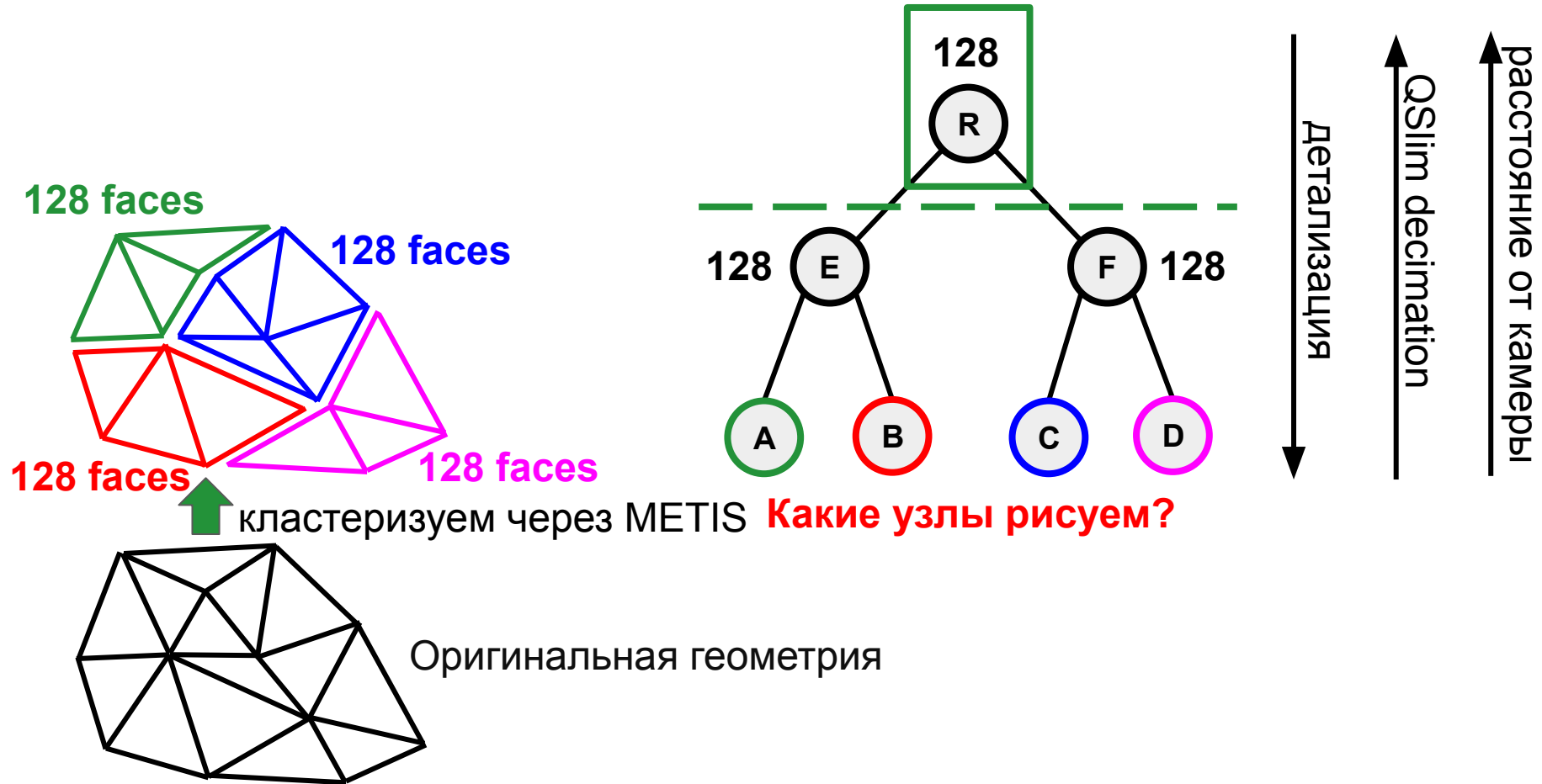
Geometry clusters hierarchy



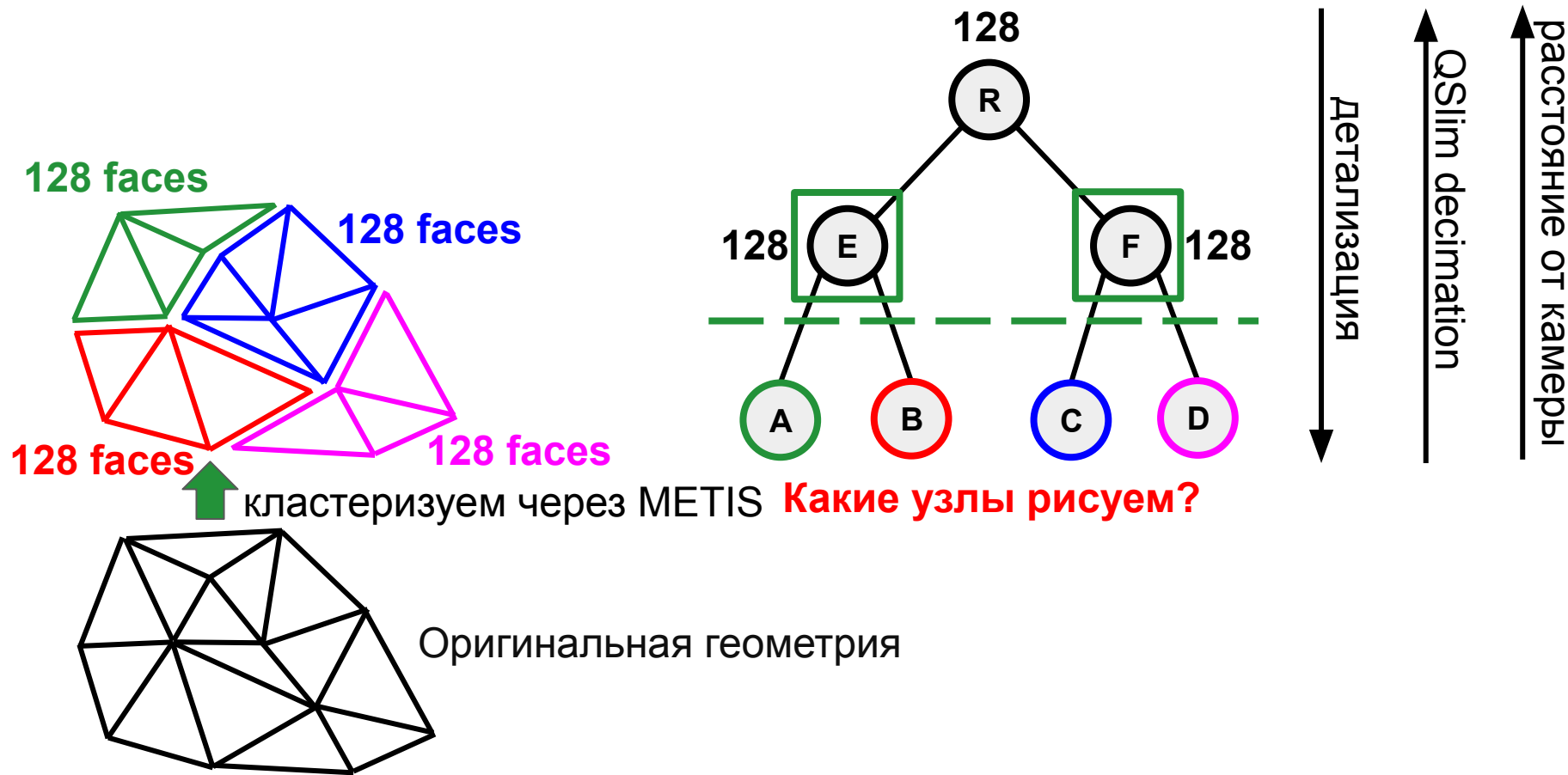
Geometry clusters hierarchy



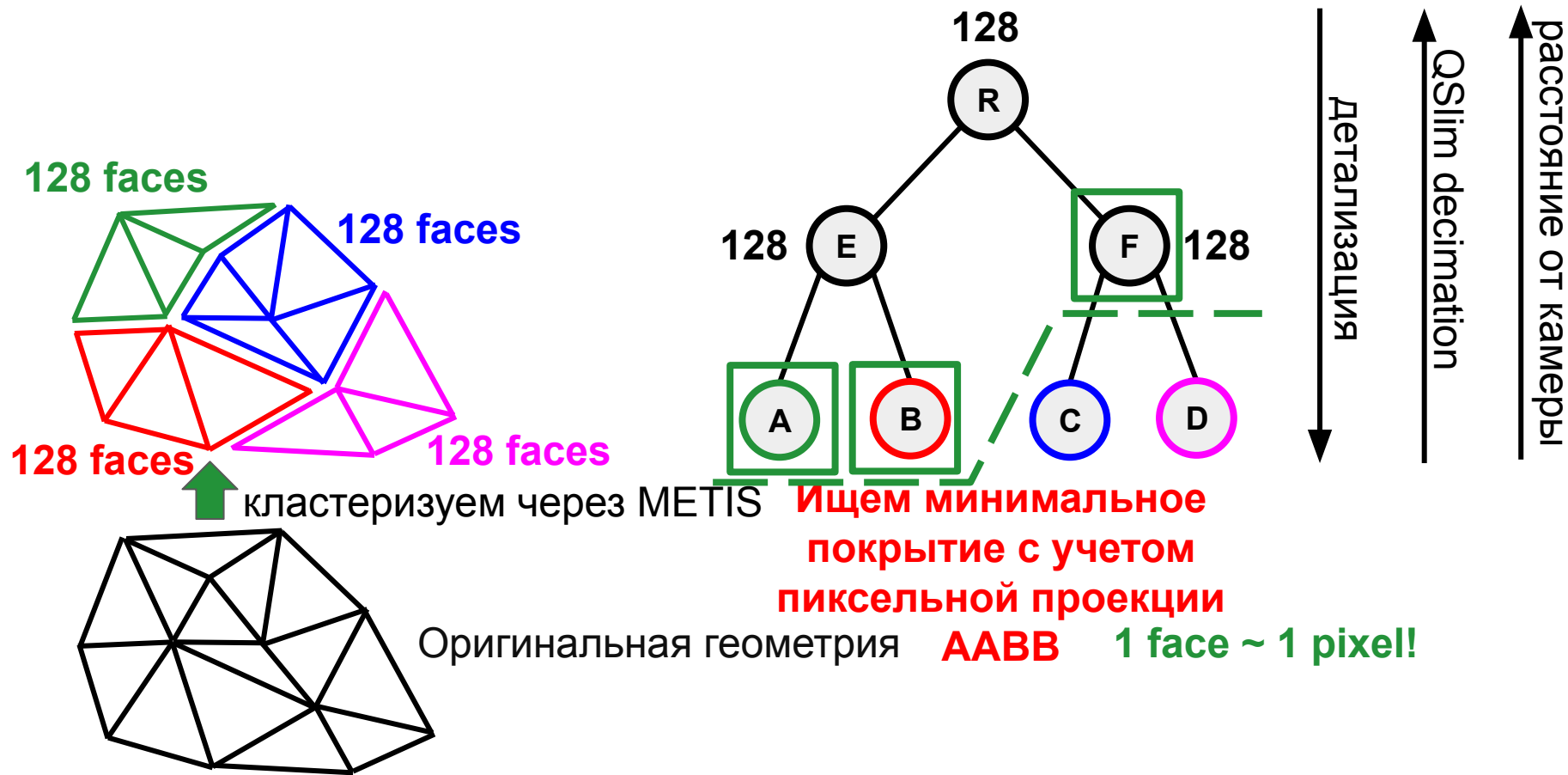
Geometry clusters hierarchy



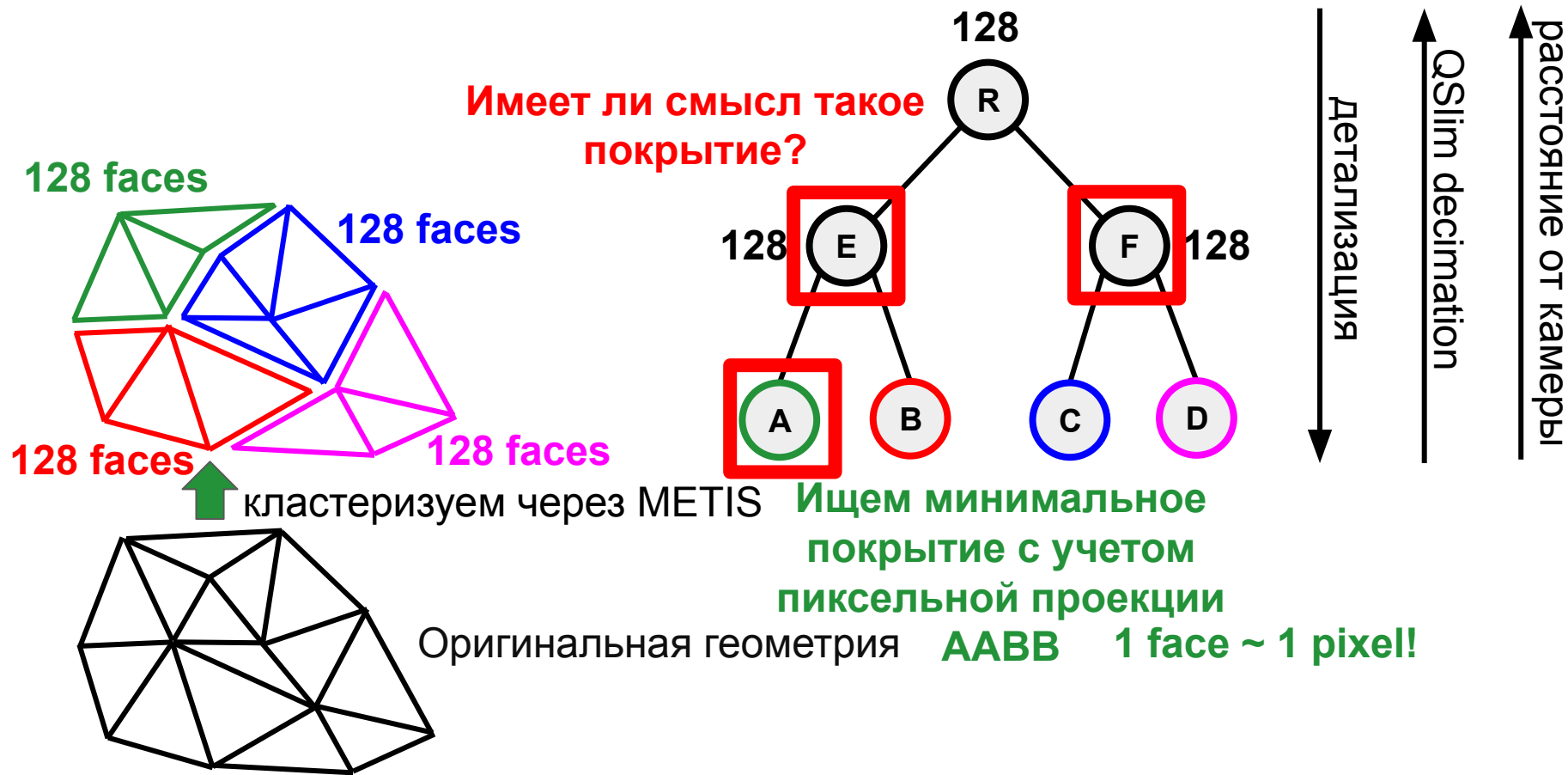
Geometry clusters hierarchy



Geometry clusters hierarchy

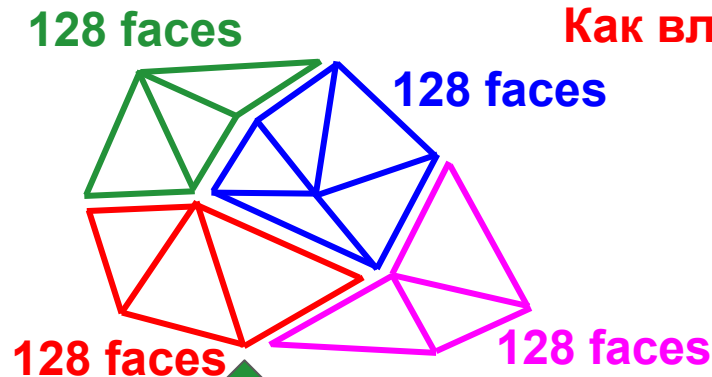


Geometry clusters hierarchy

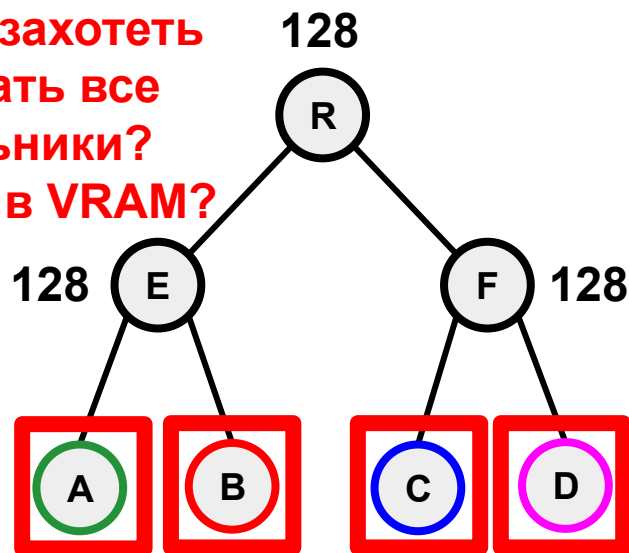


Geometry clusters hierarchy

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



↑ кластеризуем через METIS



↓ детализация

↑ QSlim decimation

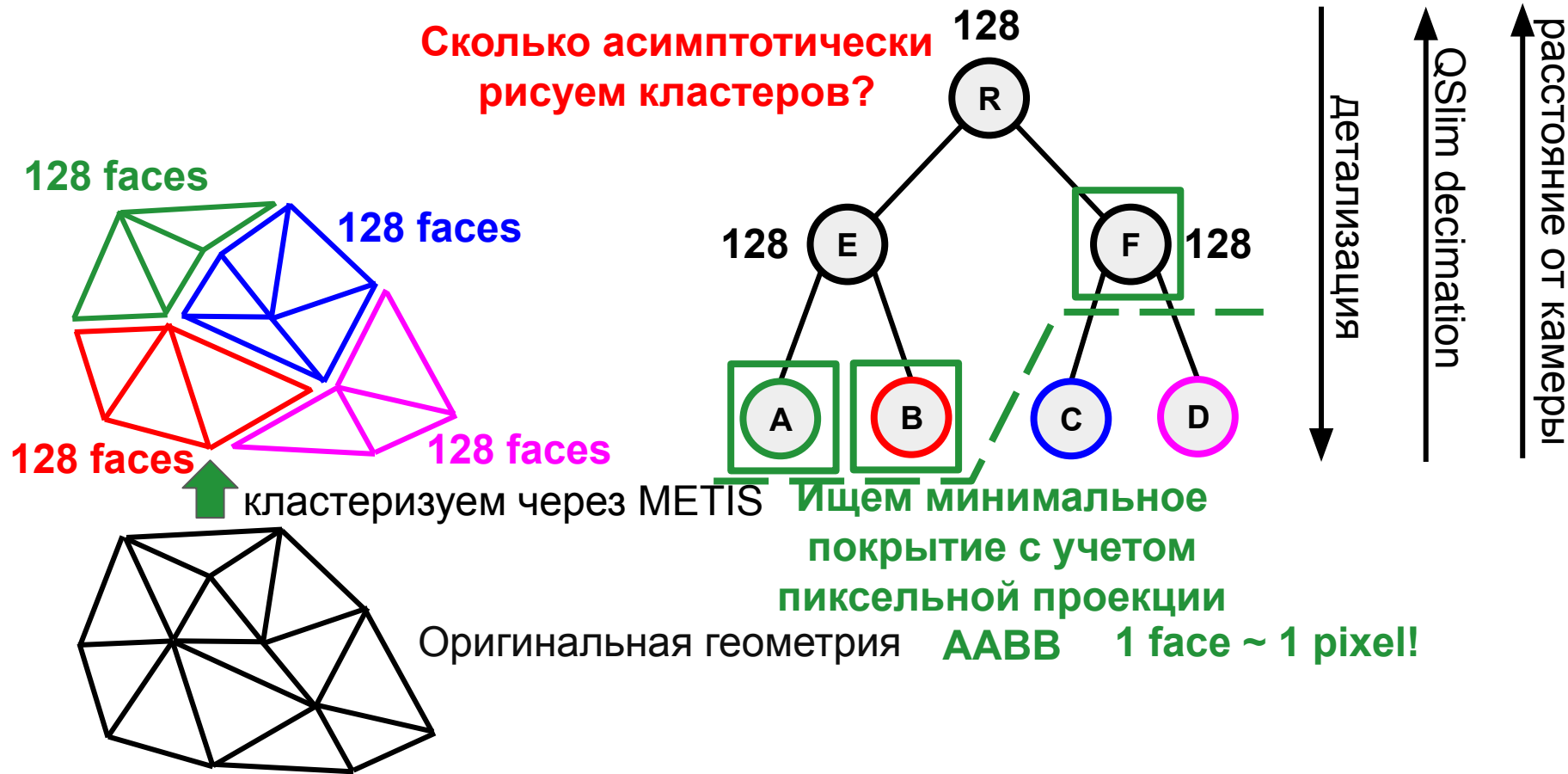
↑ расстояние от камеры

Ищем минимальное
покрытие с учетом
пиксельной проекции

Оригинальная геометрия **AABB** 1 face ~ 1 pixel!

Geometry clusters hierarchy

Сколько асимптотически
рисуем кластеров?



Geometry clusters hierarchy

Сколько асимптотически
рисуем кластеров?
 $O(\text{screen size})$

128



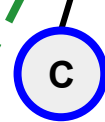
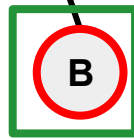
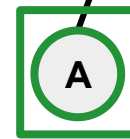
128



128



128

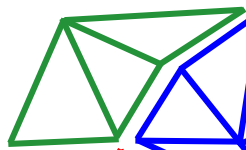


детализация

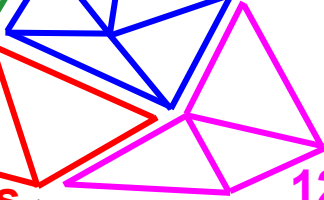
QSLim decimation

расстояние от камеры

128 faces



128 faces



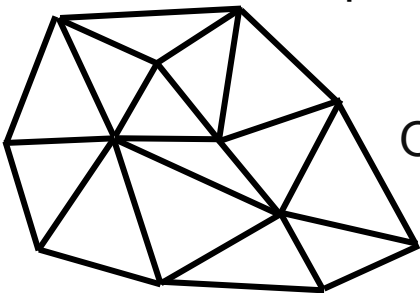
128 faces



128 faces



кластеризуем через METIS



Оригинальная геометрия

Ищем минимальное
покрытие с учетом
пиксельной проекции

AABB

1 face ~ 1 pixel!

Geometry clusters hierarchy

Несостыковка!

Сколько асимптотически
рисуем кластеров?
 $O(\text{screen size})$

128

R

128

E

F

128

A

B

C

D

детализация

QSLim decimation

расстояние от камеры

128 faces

128 faces

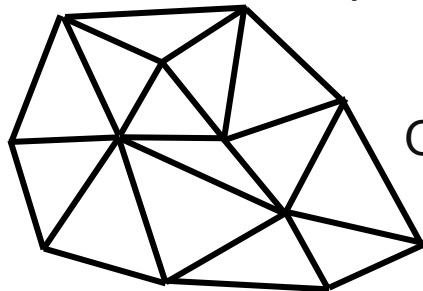
128 faces

кластеризуем через METIS

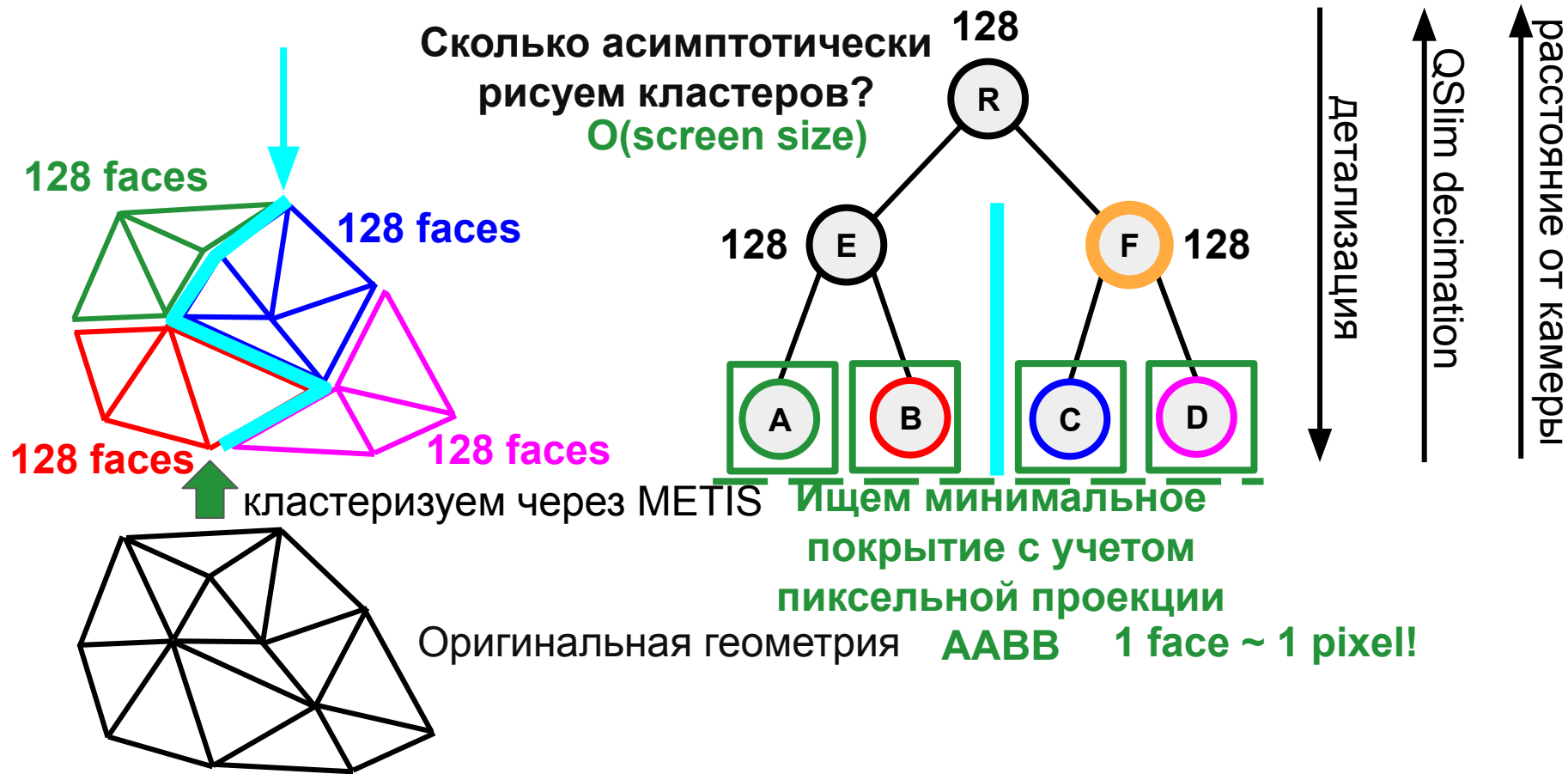
Ищем минимальное
покрытие с учетом
пиксельной проекции

Оригинальная геометрия

AABB 1 face ~ 1 pixel!

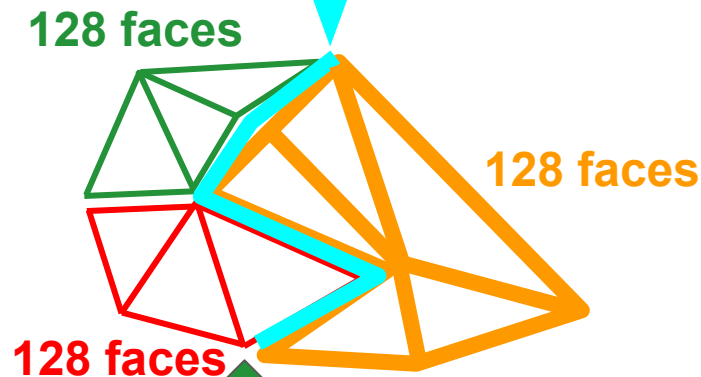


Geometry clusters hierarchy

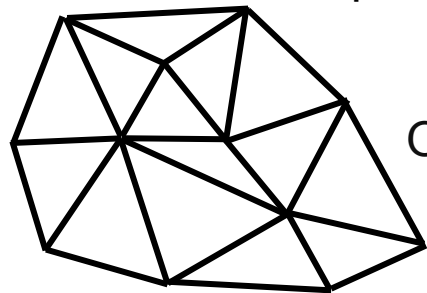


Geometry clusters hierarchy

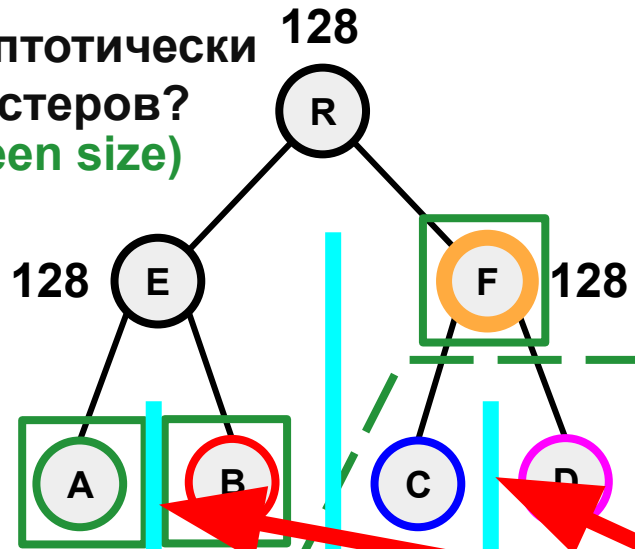
Заморозка границ!
(Locked boundary)



кластеризуем через METIS



Сколько асимптотически
рисуем кластеров?
 $O(\text{screen size})$



Ищем минимальное
покрытие с учетом
пиксельной проекции

Оригинальная геометрия **AABB** 1 face ~ 1 pixel!

Заморозка границ - много треугольников не упростить!
Dense Crust!

детализация

QSLim decimation

расстояние от камеры

Почему тут
корка не
так плоха?

Geometry clusters hierarchy

Заморозка границ!
(Locked boundary)

Сколько
рисует

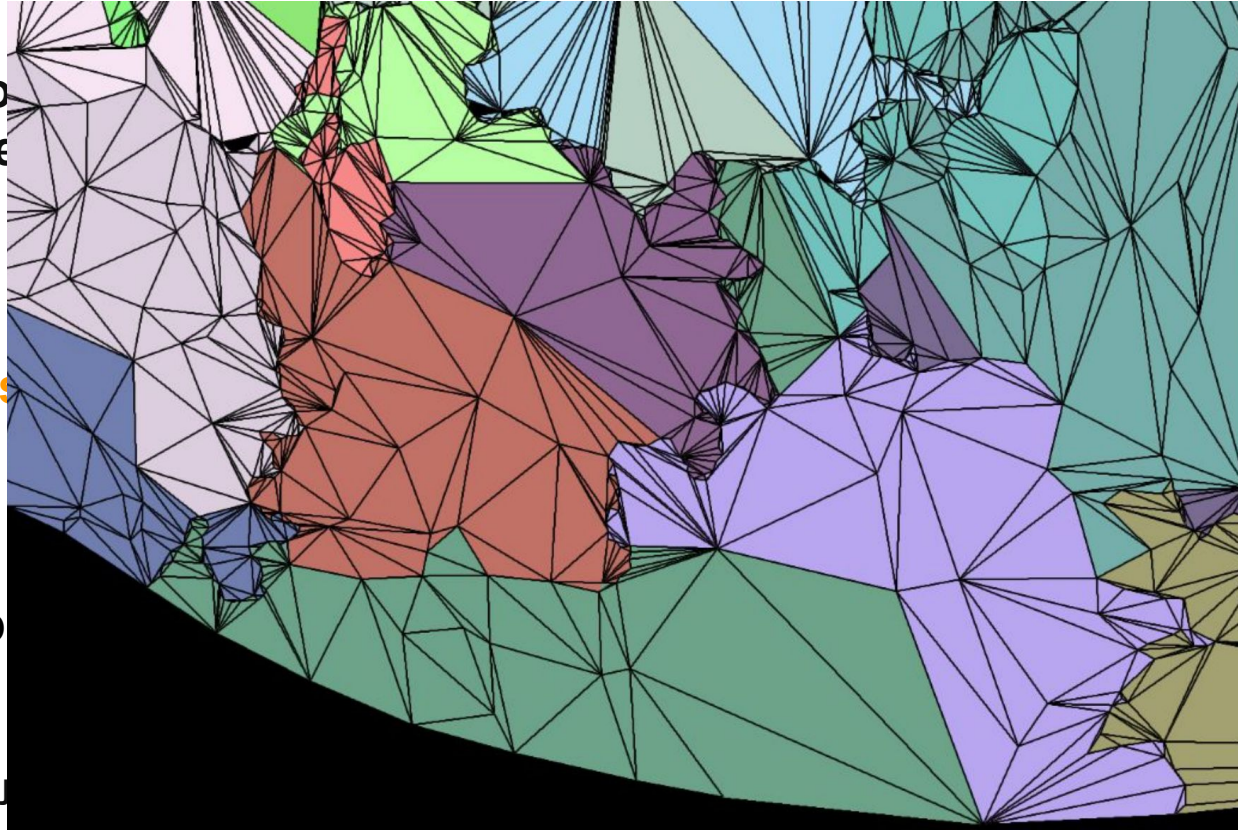
128 faces

128 faces

128 faces

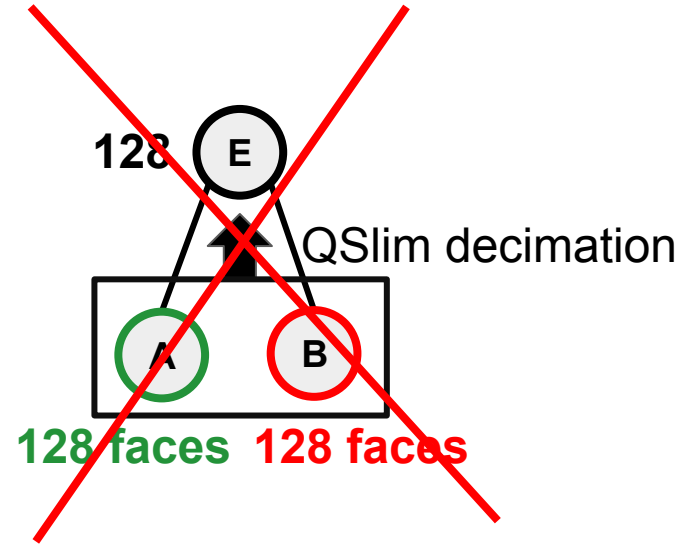
↑ кластеризуем чер

Оригина

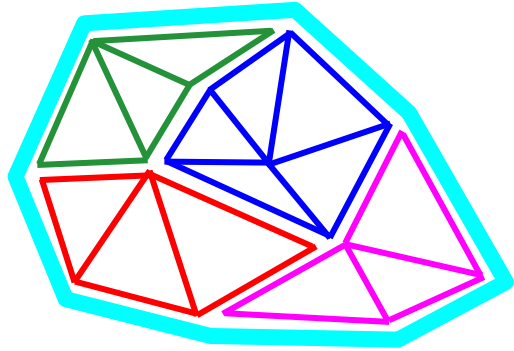


Заморозка границ - много треугольников не упростить!
Dense Crust!

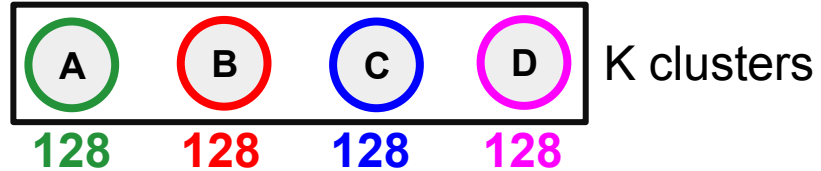
Geometry clusters hierarchy: DAG



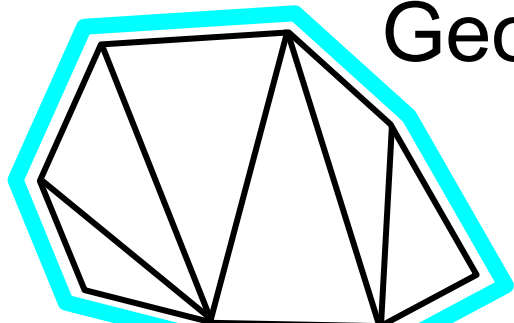
Geometry clusters hierarchy: DAG



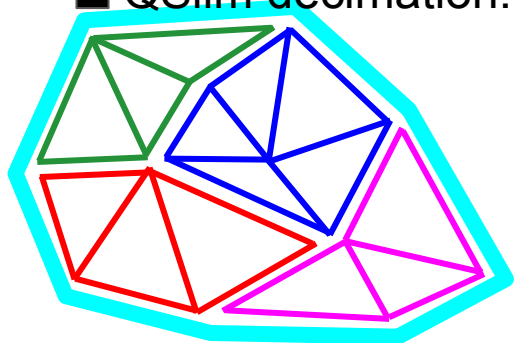
K=4 clusters



Geometry clusters hierarchy: DAG



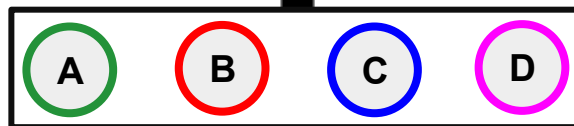
↑ QSlim decimation: 14 $\rightarrow$ 7 faces



K=4 clusters

$128 * K / 2$ faces

↑ x2 QSlim decimation



K clusters

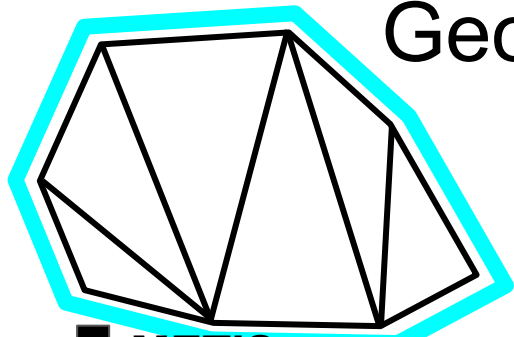
128

128

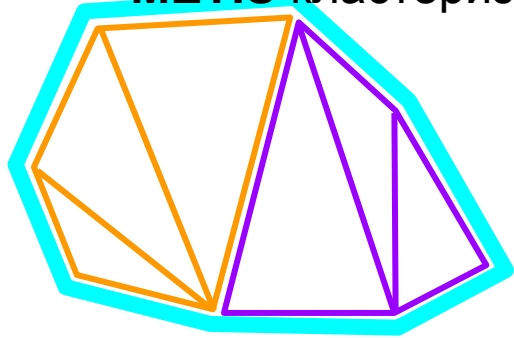
128

128

Geometry clusters hierarchy: DAG



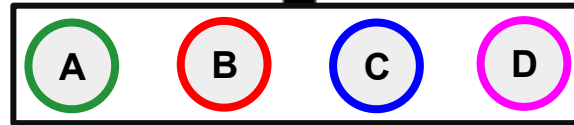
↓ METIS кластеризация



↑ METIS кластеризация



↑ x2 QSlim decimation

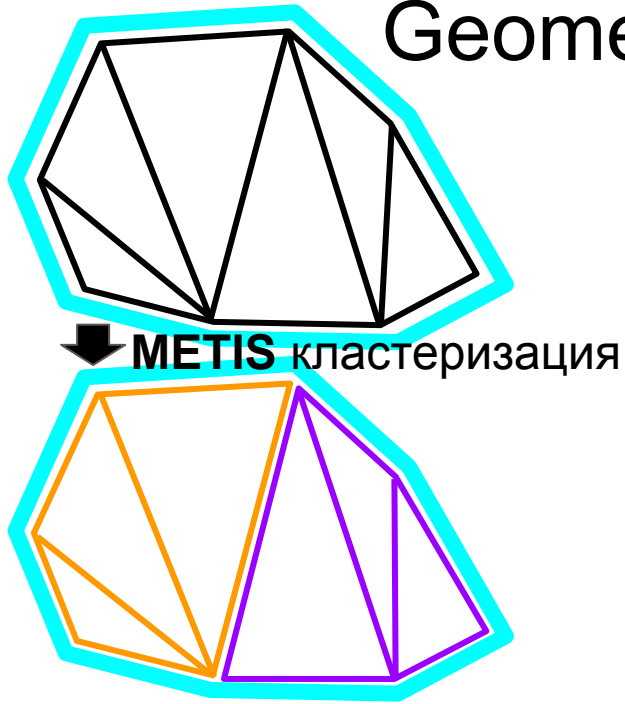


K clusters

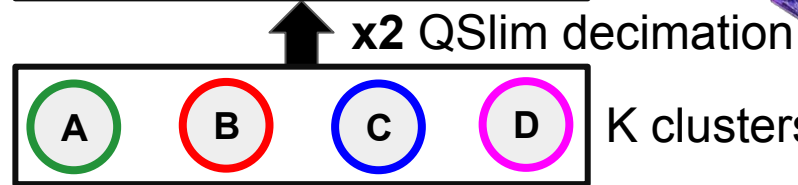
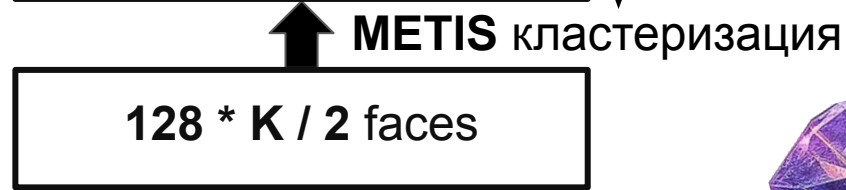
128 128 128 128

Geometry clusters hierarchy: DAG

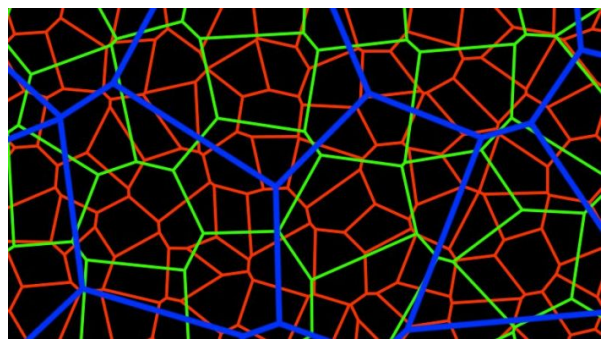
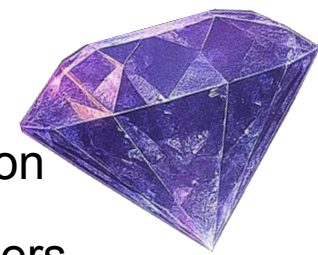
Кластеризируем минимизируя
число ребер на **периметре** кластера



т.о. избегаем
накопления корки!

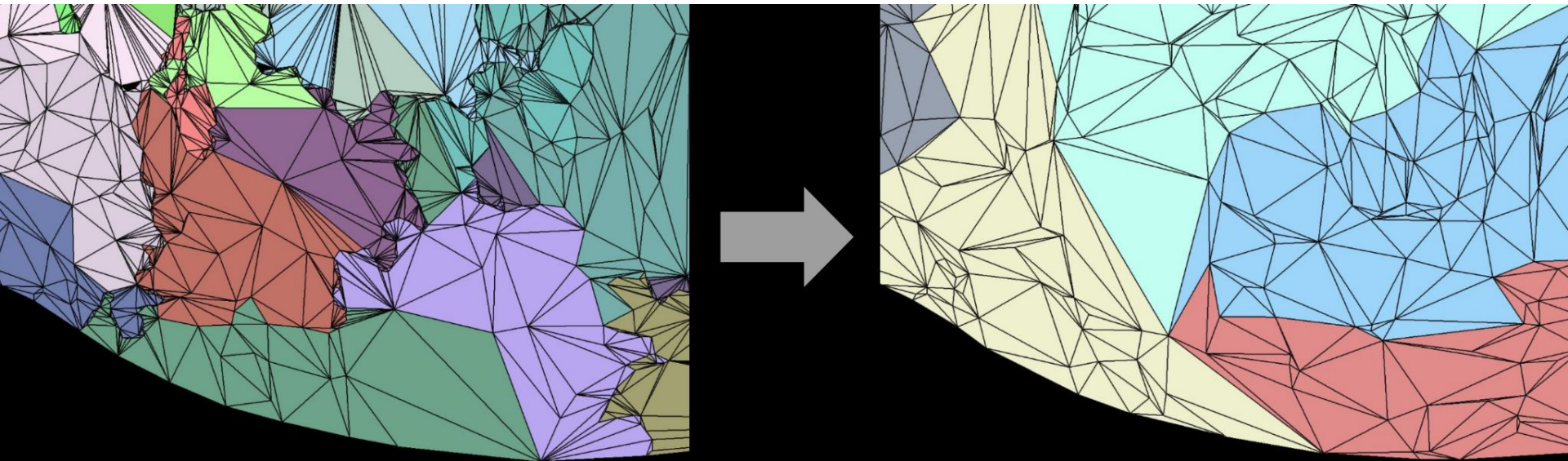


K clusters

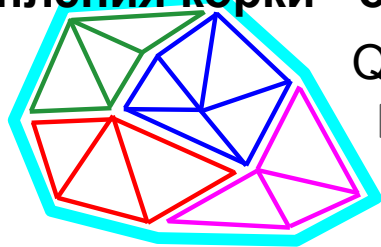
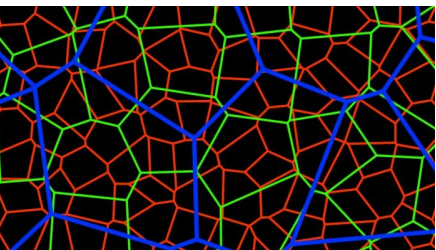


Geometry clusters hierarchy: DAG

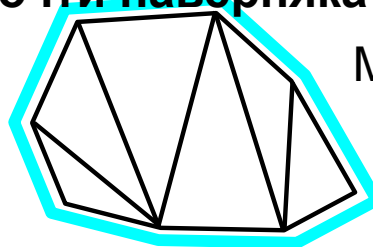
Кластеризируем минимизируя
число ребер на **периметре** кластера



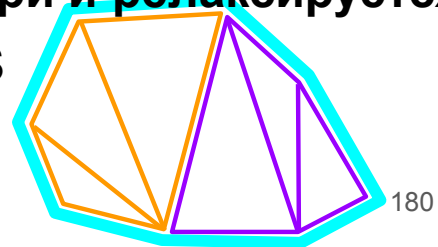
т.о. избегаем накопления корки - она почти наверняка внутри и релаксируется



QSlm



METIS

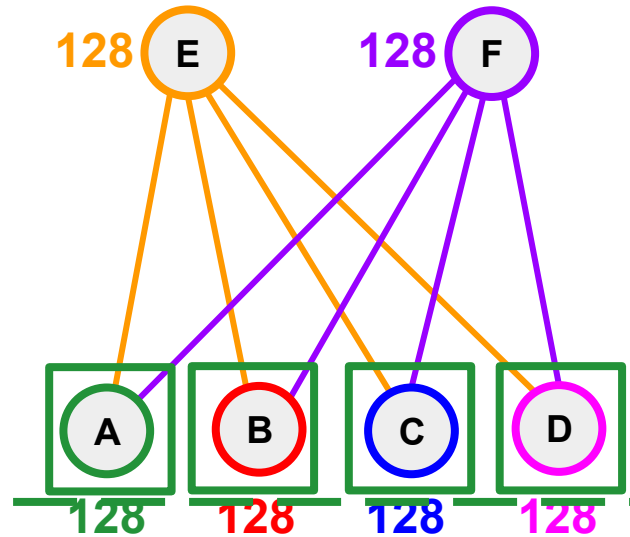
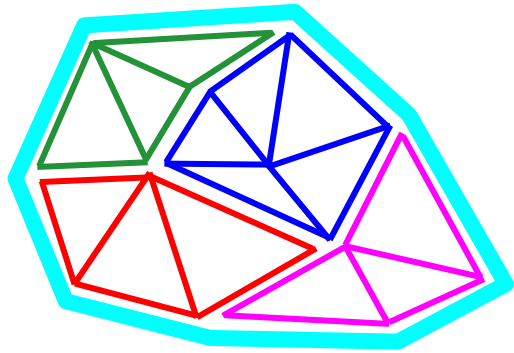


180

Geometry clusters hierarchy: DAG

Кластеризируем минимизируя
число ребер на **периметре** кластера

т.о. избегаем
накопления корки!

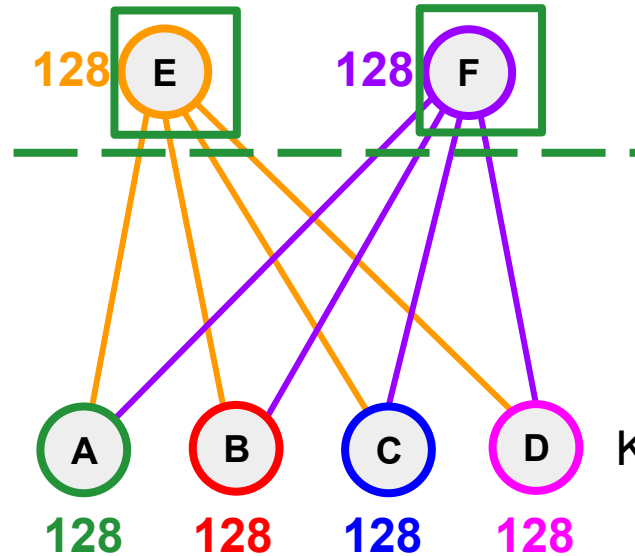
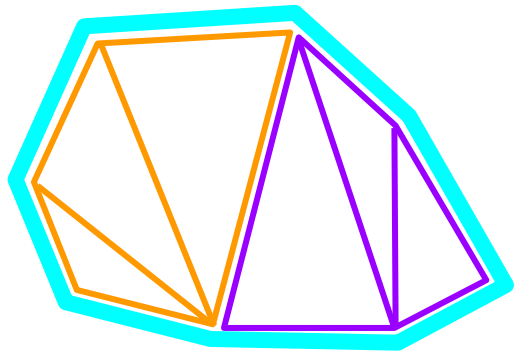


Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

Кластеризируем минимизируя
число ребер на **периметре** кластера

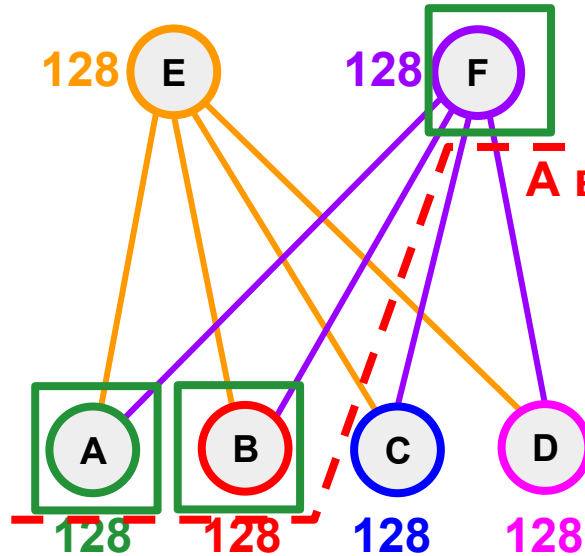
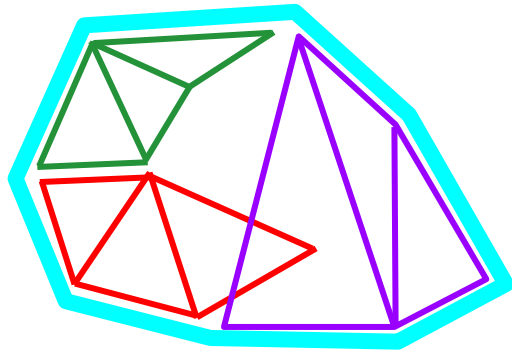
т.о. избегаем
накопления корки!



Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

Кластеризируем минимизируя
число ребер на **периметре** кластера



т.о. избегаем
накопления корки!

А вот так можно?

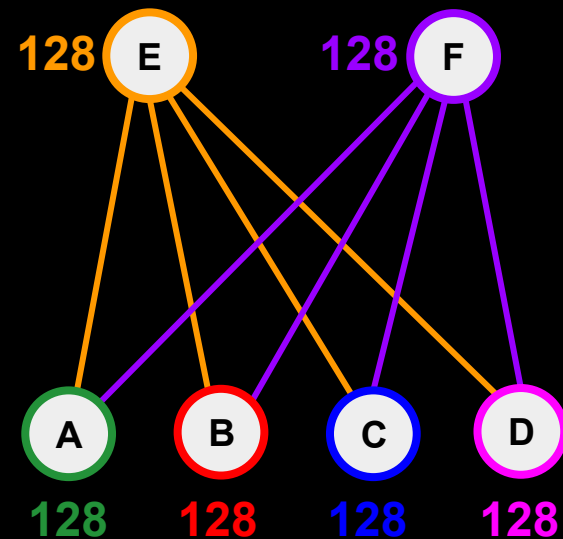


K clusters

Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

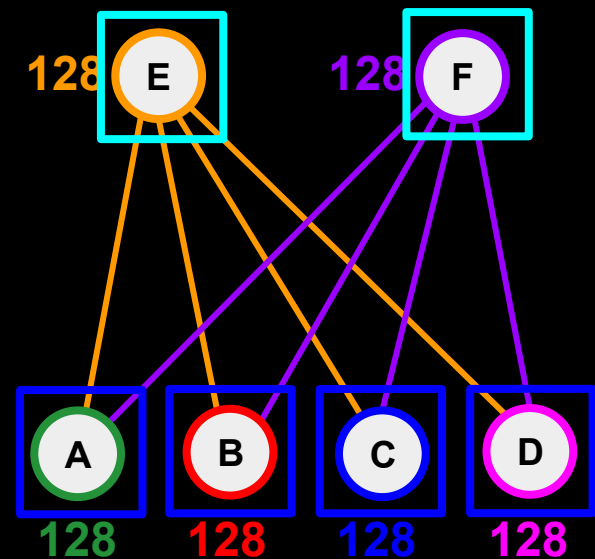
- Two submeshes with same boundary, but different LOD
- Choose between them based on screen-space error
 - Error calculated by simplifier projected to screen
 - Corrected for distance and angle distortion at worst-case point in sphere bounds
- All clusters in group must make same LOD decision
 - How? 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

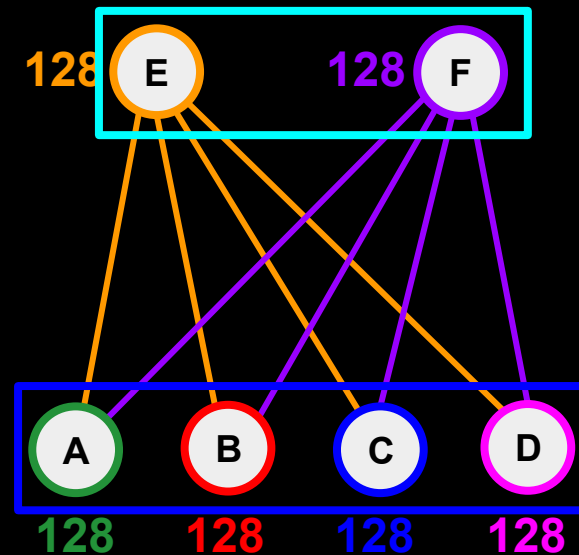


Оцениваем проекцию AABV каждого кластера?

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

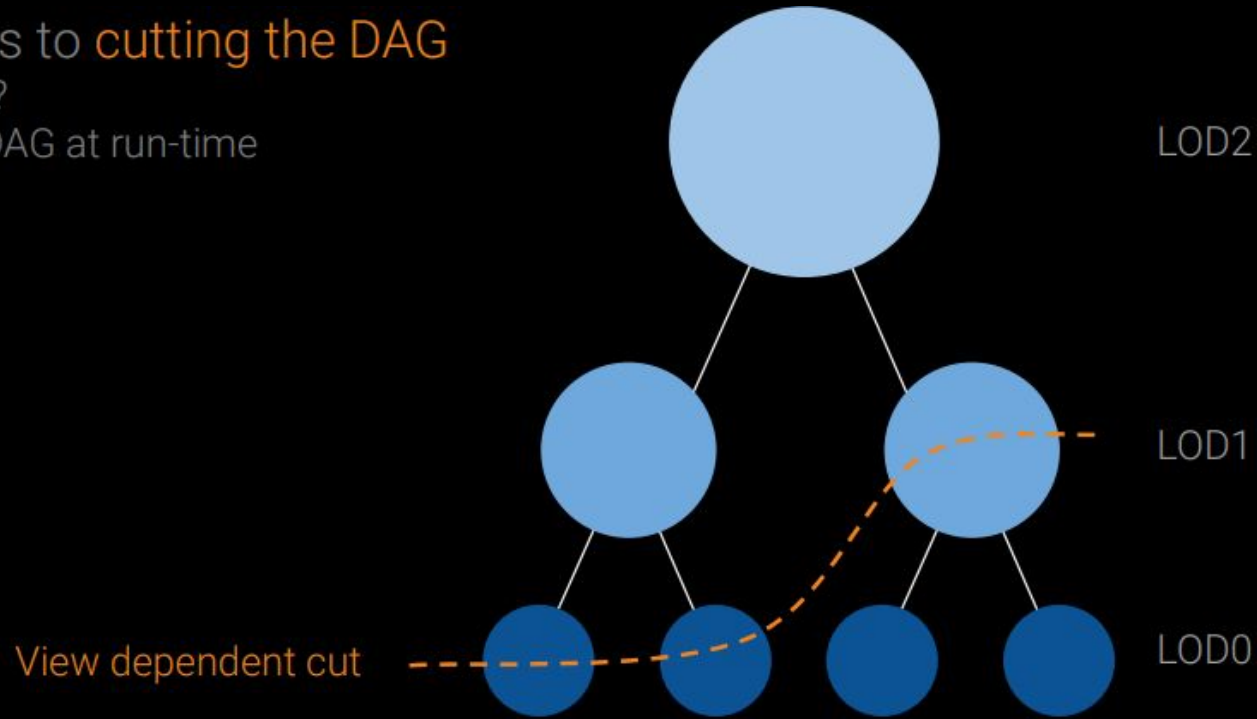


Оцениваем проекцию AABV не по кластеру,
а по нашей **группе К-кластеров!**

Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

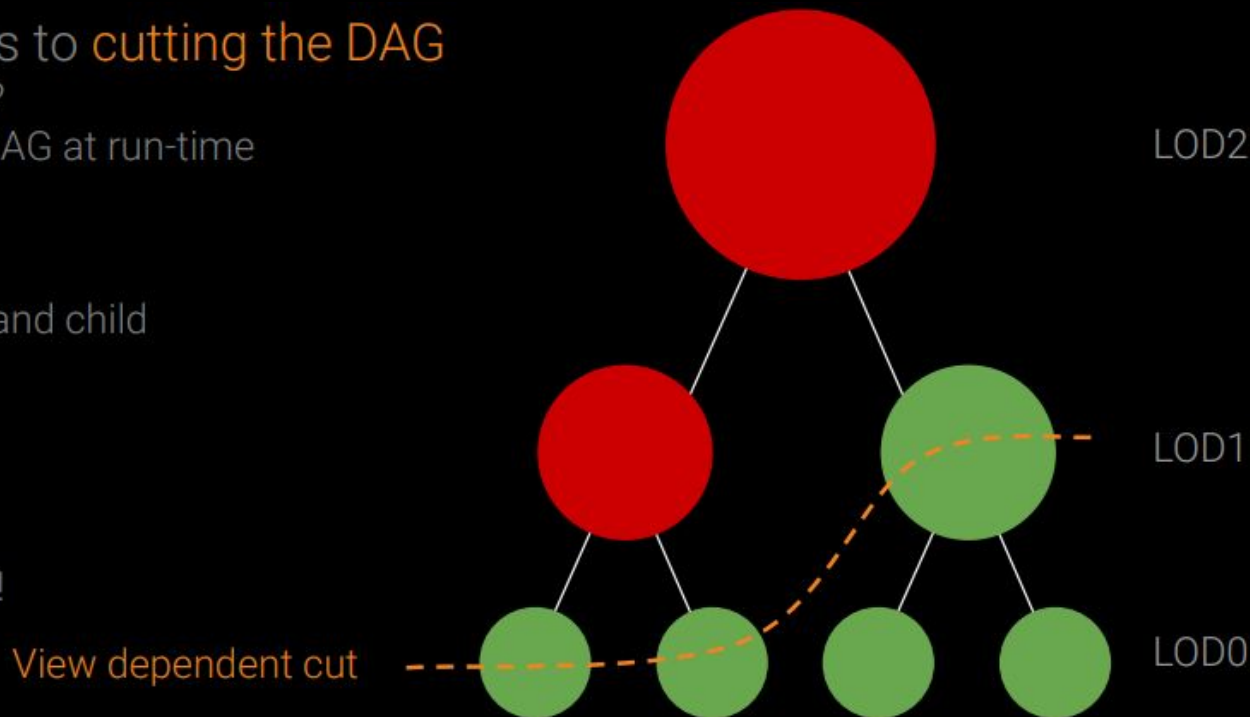
- LOD selection corresponds to **cutting the DAG**
 - How to compute in parallel?
 - Don't want to traverse the DAG at run-time
- What defines the cut?



Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

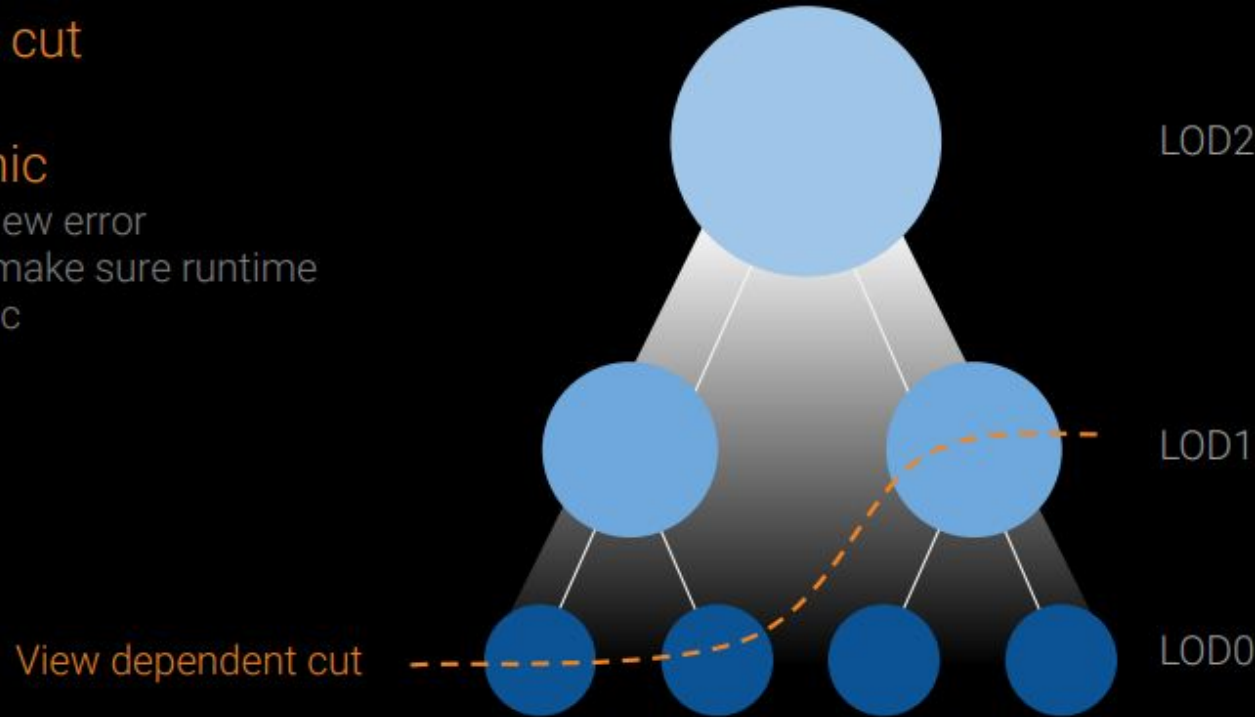
- LOD selection corresponds to **cutting the DAG**
 - How to compute in parallel?
 - Don't want to traverse the DAG at run-time
- What defines the cut?
 - Difference between parent and child
- Draw a cluster when:
 - Parent error is too high &&
 - Our error is small enough
 - Can be evaluated in parallel!



Какое подмножество кластеров рисовать?

Geometry clusters hierarchy: DAG

- Only if there is **one unique cut**
- Force error to be **monotonic**
 - Parent view error $\geq$ child view error
 - Careful implementation to make sure runtime correction is also monotonic



Какое подмножество кластеров рисовать?

What is Nanite?

Virtualized geometry: streaming

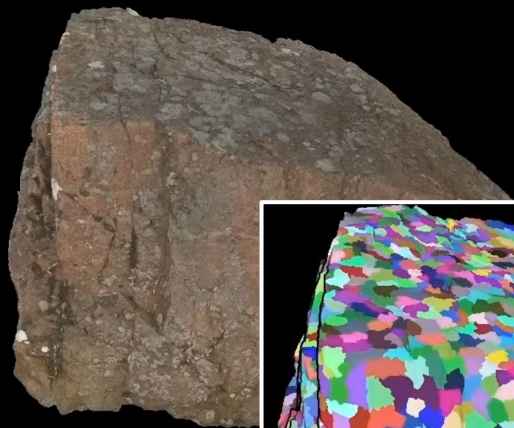
- s.t. face edge's projection $\sim$ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost



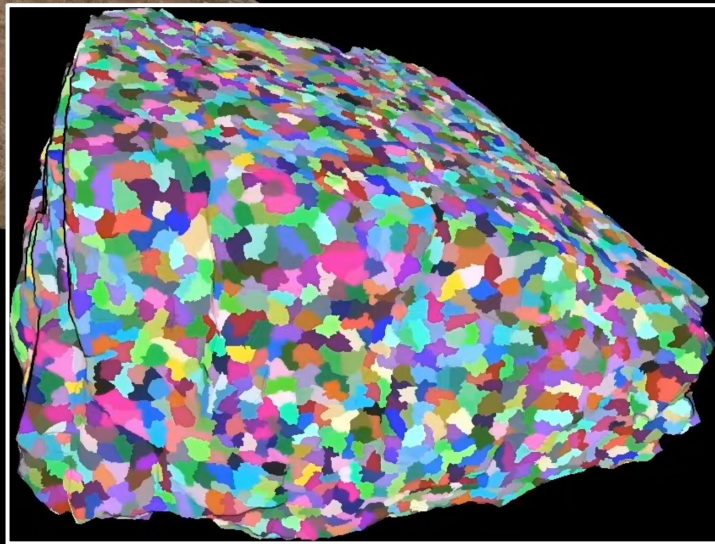
What is Nanite?

Virtualized geometry: streaming

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



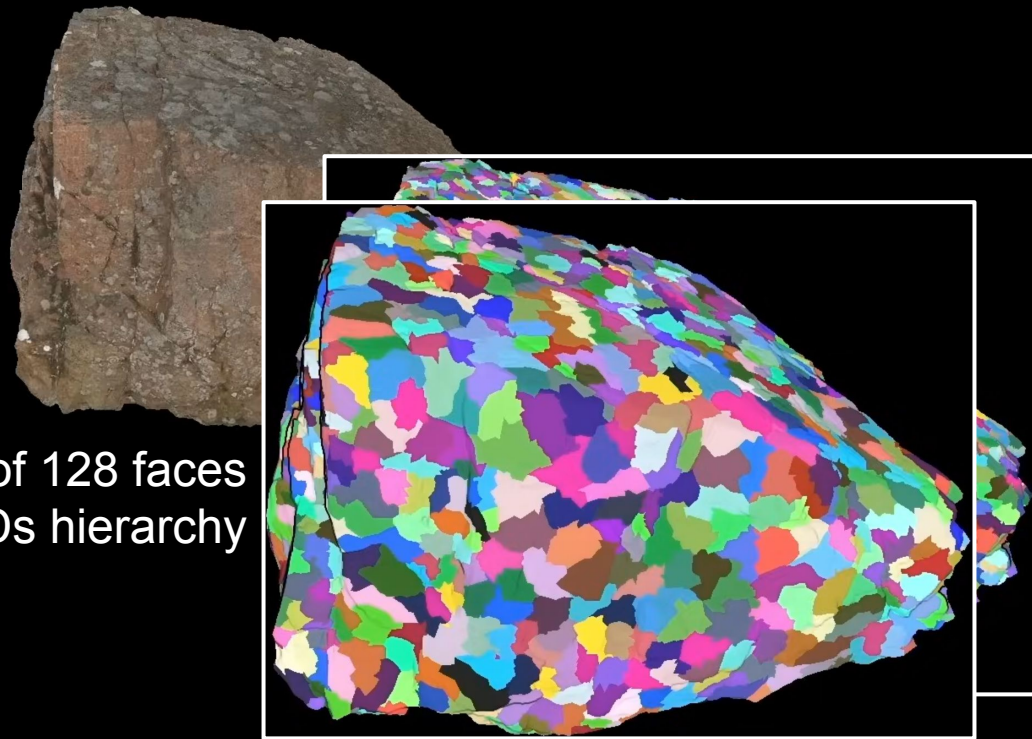
Clusters of 128 faces



What is Nanite?

Virtualized geometry: streaming

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

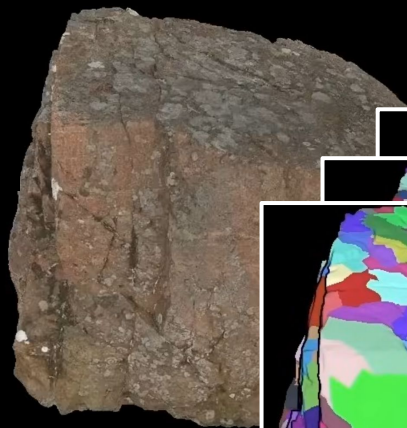


Clusters of 128 faces
+ LODs hierarchy

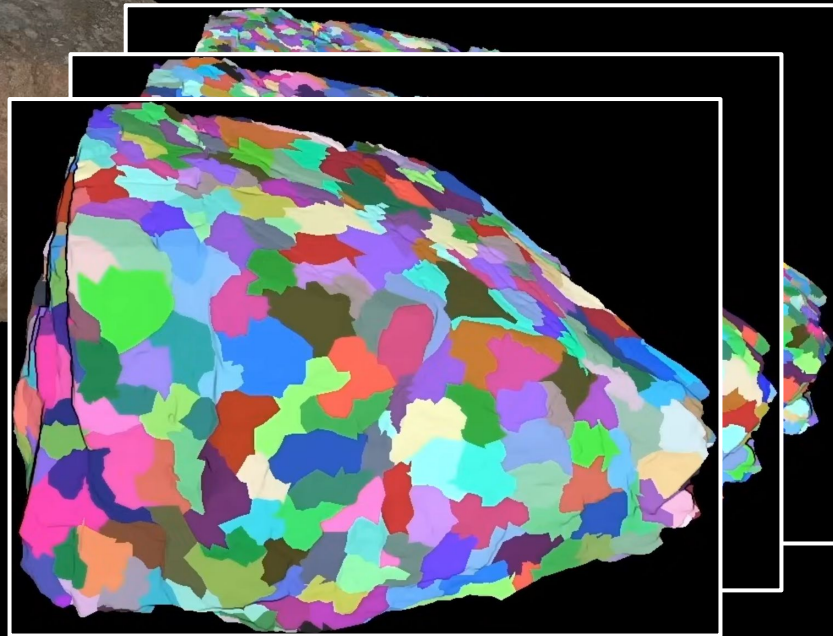
What is Nanite?

Virtualized geometry: streaming

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



Clusters of 128 faces
+ LODs hierarchy



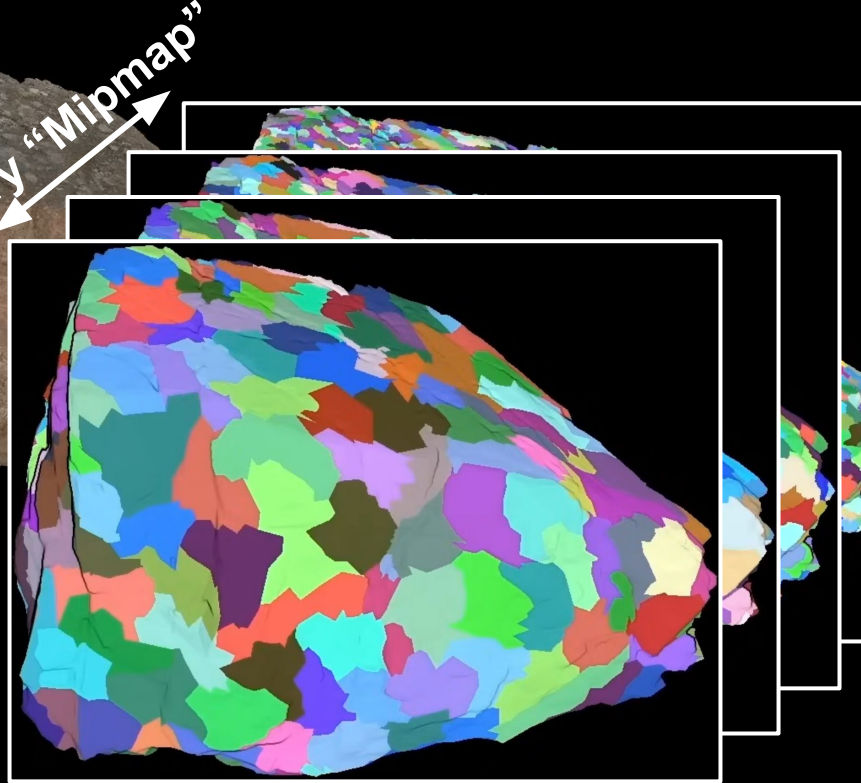
What is Nanite?

Virtualized geometry: streaming

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

Geometry "Mipmap"

Clusters of 128 faces
+ LODs hierarchy

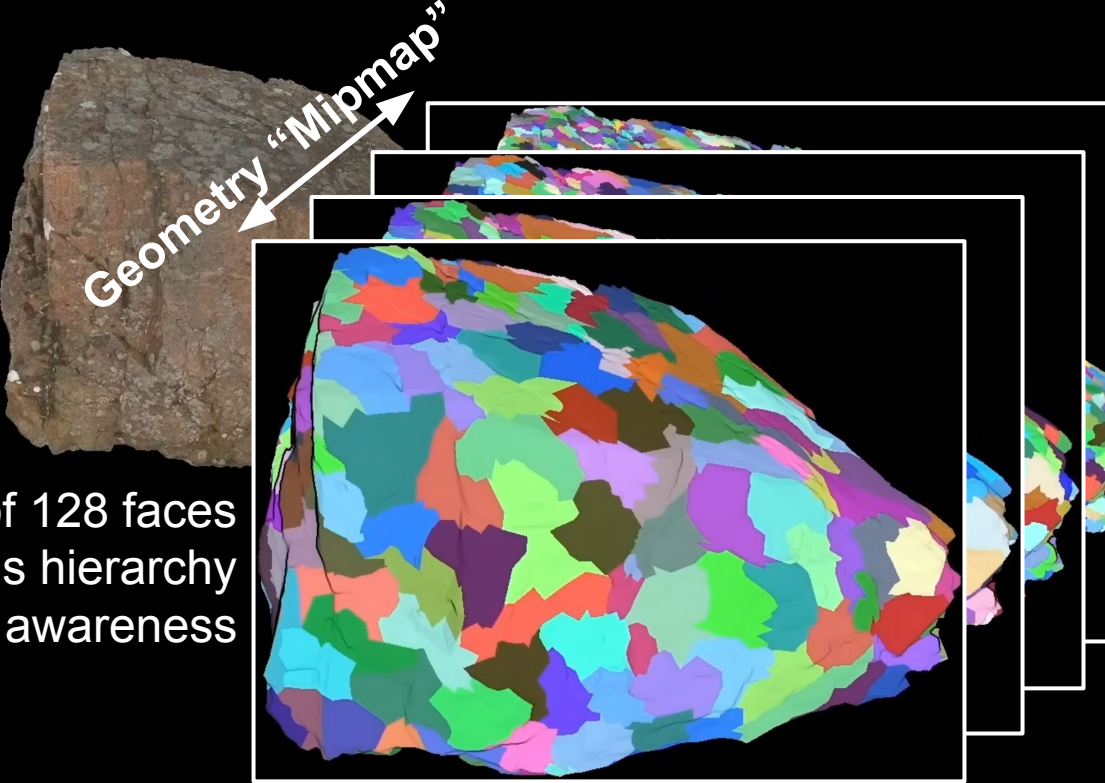


What is Nanite?

Virtualized geometry: streaming

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

Geometry "Mipmap"



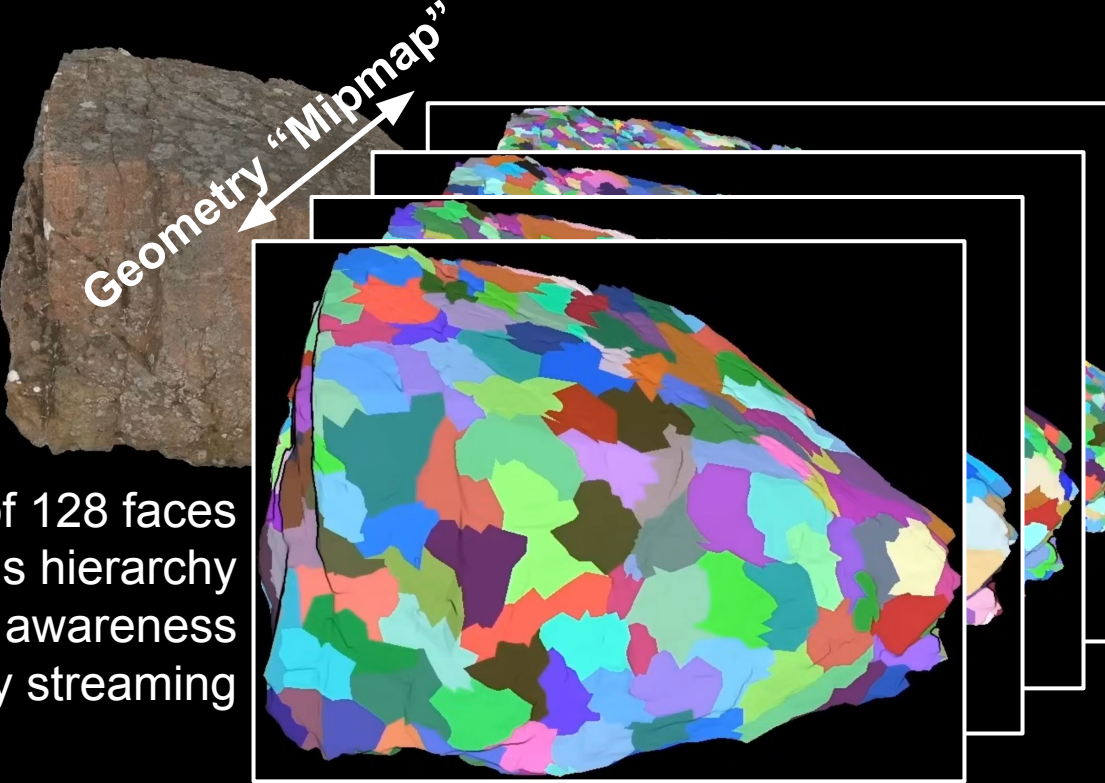
Clusters of 128 faces
+ LODs hierarchy
+ Cracks awareness

What is Nanite?

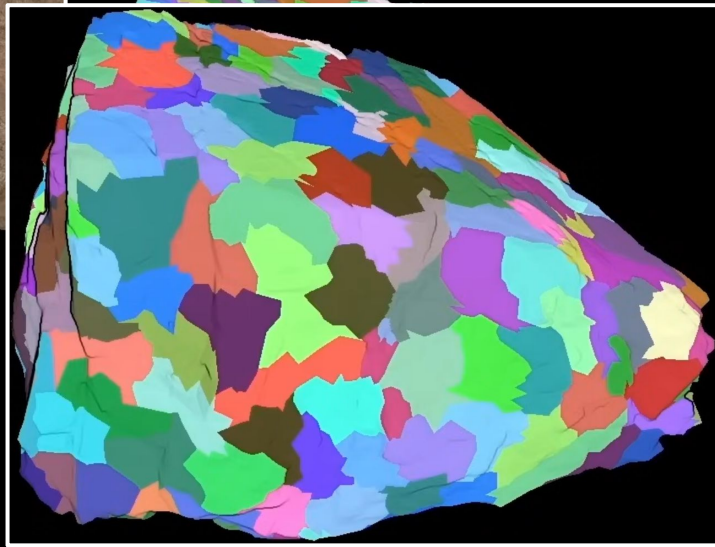
Virtualized geometry: streaming

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

Geometry "Mipmap"



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



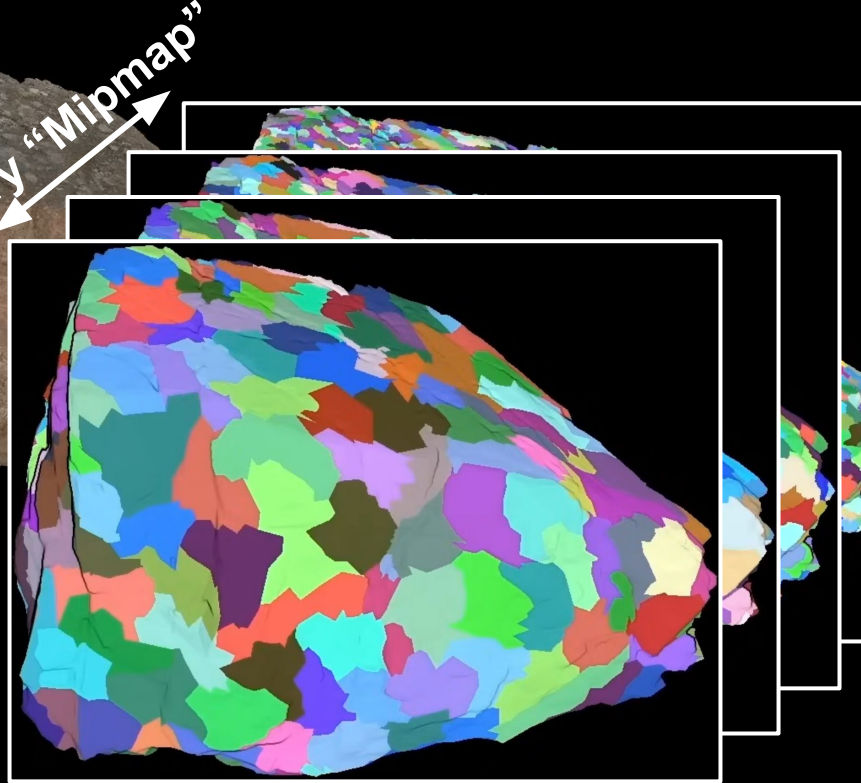
What is Nanite?

Virtualized geometry: streaming

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

Geometry "Mipmap"

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

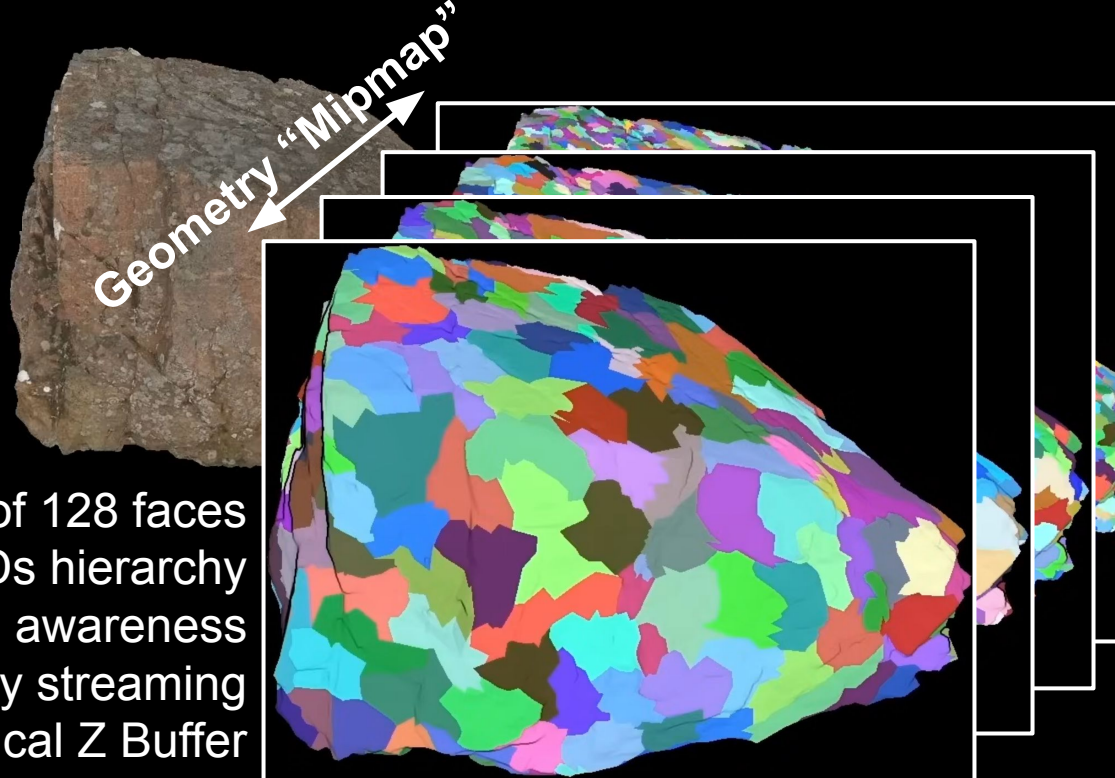


What is Nanite?

Virtualized geometry: streaming

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

Geometry "Mipmap"



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

Hardware (big faces)

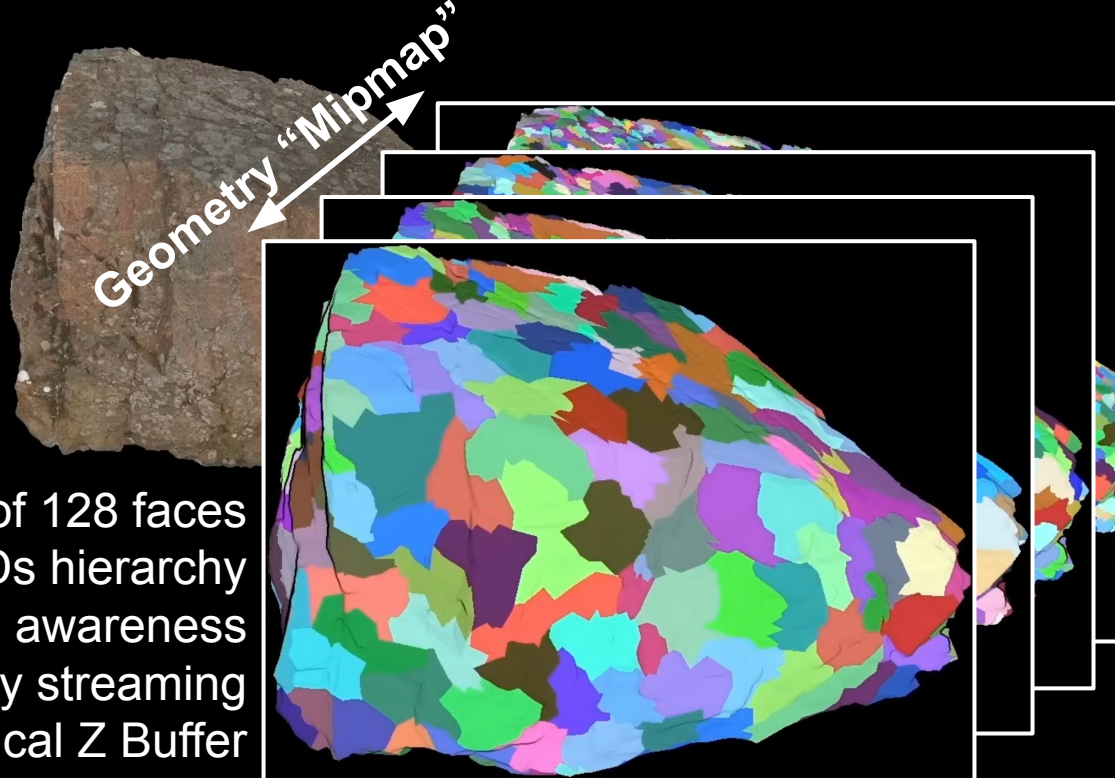
Rasterization

What is Nanite?

Virtualized geometry: streaming

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

Geometry "Mipmap"

A diagram illustrating the Nanite virtualized geometry system. It shows a sequence of rock models from high to low fidelity. The top-most model is a high-resolution texture of a rock. Below it, several other models are shown, each with a different level of detail. The bottom-most model is a low-resolution version where the rock's surface is composed of many small, colorful, irregular polygons. An arrow labeled "Geometry 'Mipmap'" points from the high-resolution model down to the low-resolution model, indicating the streaming of different levels of detail.

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

Rasterization $\left\{ \begin{array}{l} \text{Hardware (big faces)} \\ \text{+} \\ \text{Software (\sim\text{pixel-sized faces})} \end{array} \right.$

What is Nanite?

Virtualized geometry: streaming

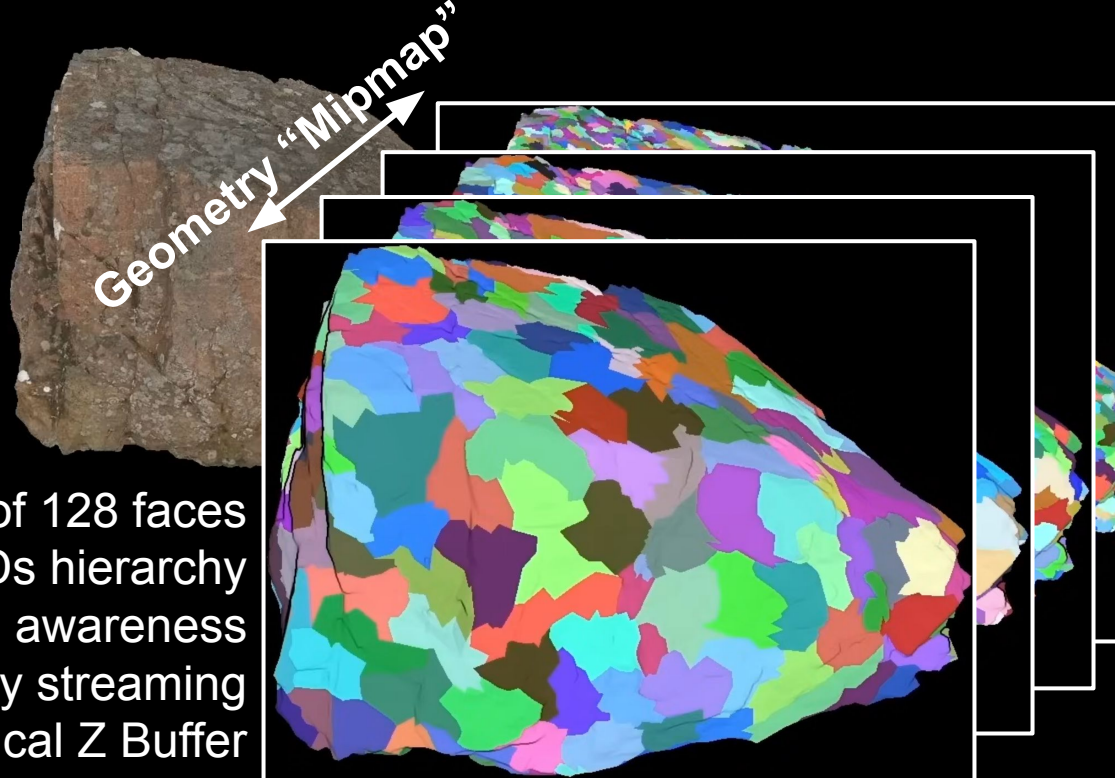
- s.t. face edge's projection $\sim$ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

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

Geometry "Mipmap"



Rasterization

Hardware (big faces)
+
Software ($\sim$ pixel-sized faces)

What is Nanite?

Virtualized geometry: streaming

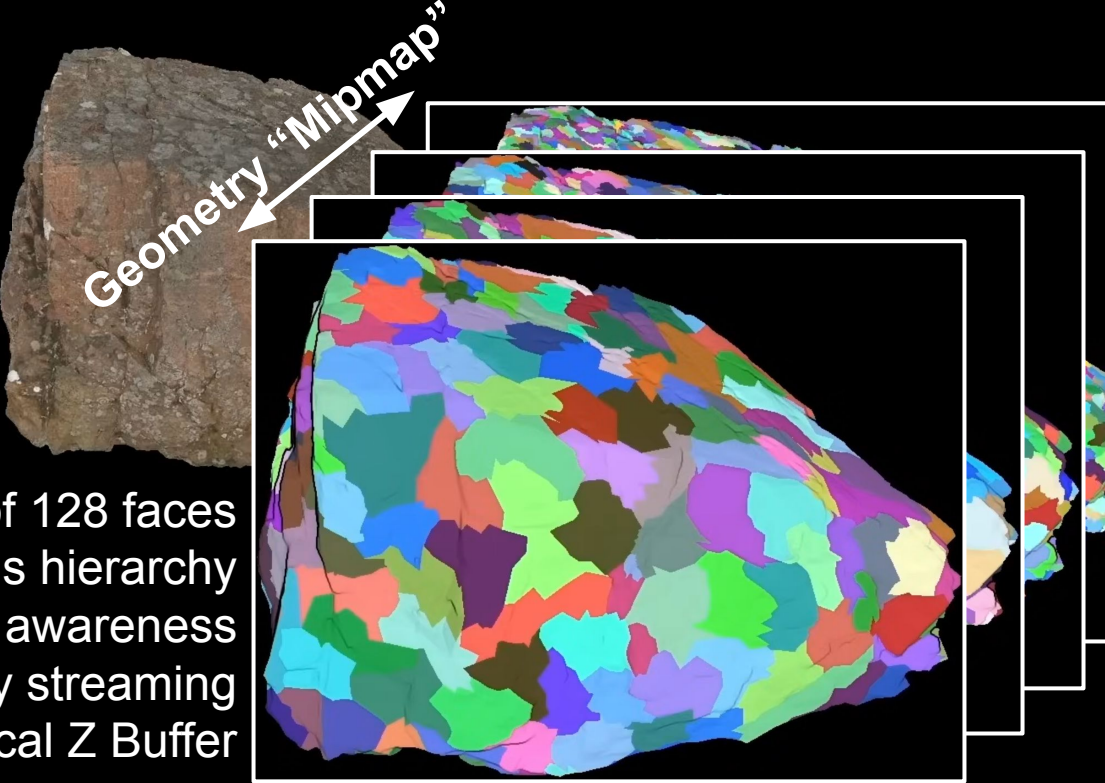
- s.t. face edge's projection $\sim$ pixel
- Nearly pixel-perfect LODs
- Higher fidelity, lower cost

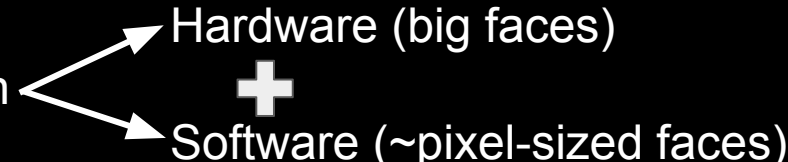
What can't be Nanite

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

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

Geometry "Mipmap"



Rasterization 

Hardware (big faces)
+
Software ($\sim$ pixel-sized faces)

What is Nanite?

Virtualized geometry: streaming

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

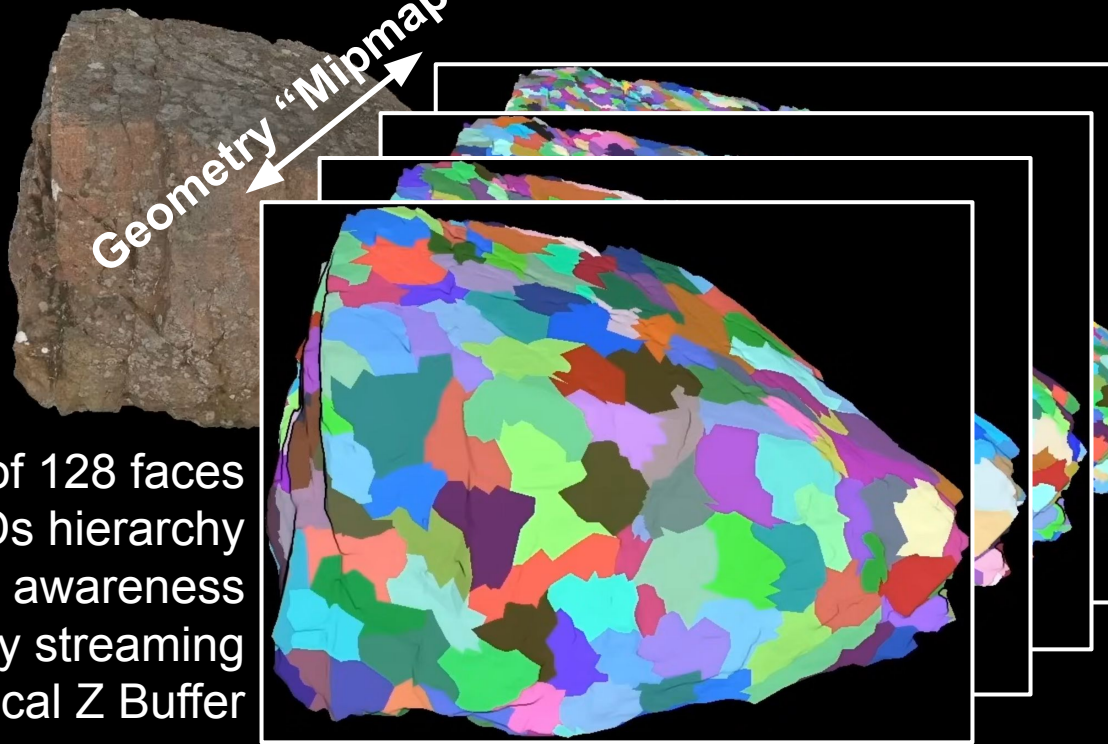
What can't be Nanite

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



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

Geometry "Mipmap"



Hardware (big faces)

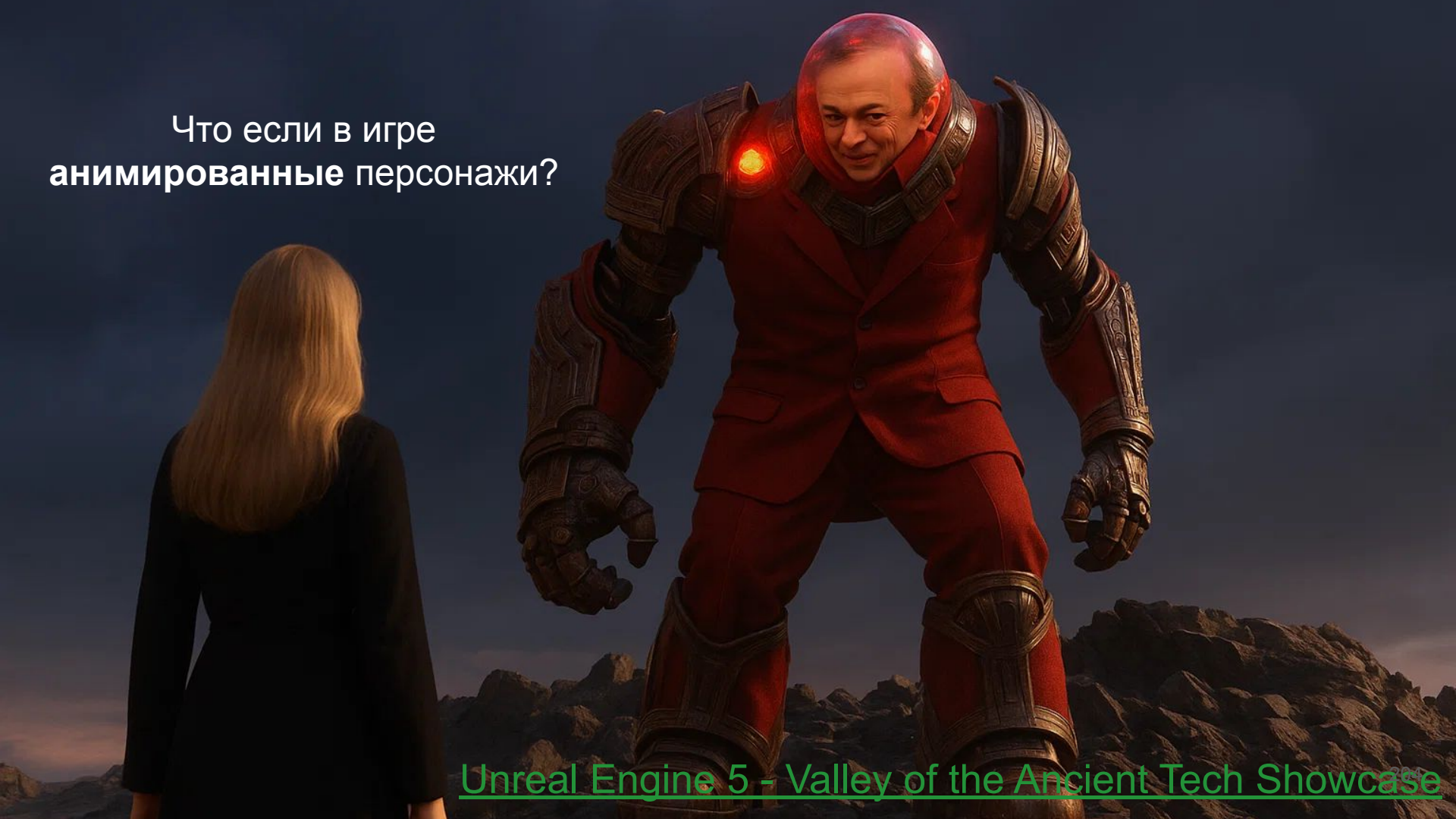
Software ($\sim$ pixel-sized faces)

Rasterization

Глава 30: Анимация и rigging



Что если в игре
анимированные персонажи?



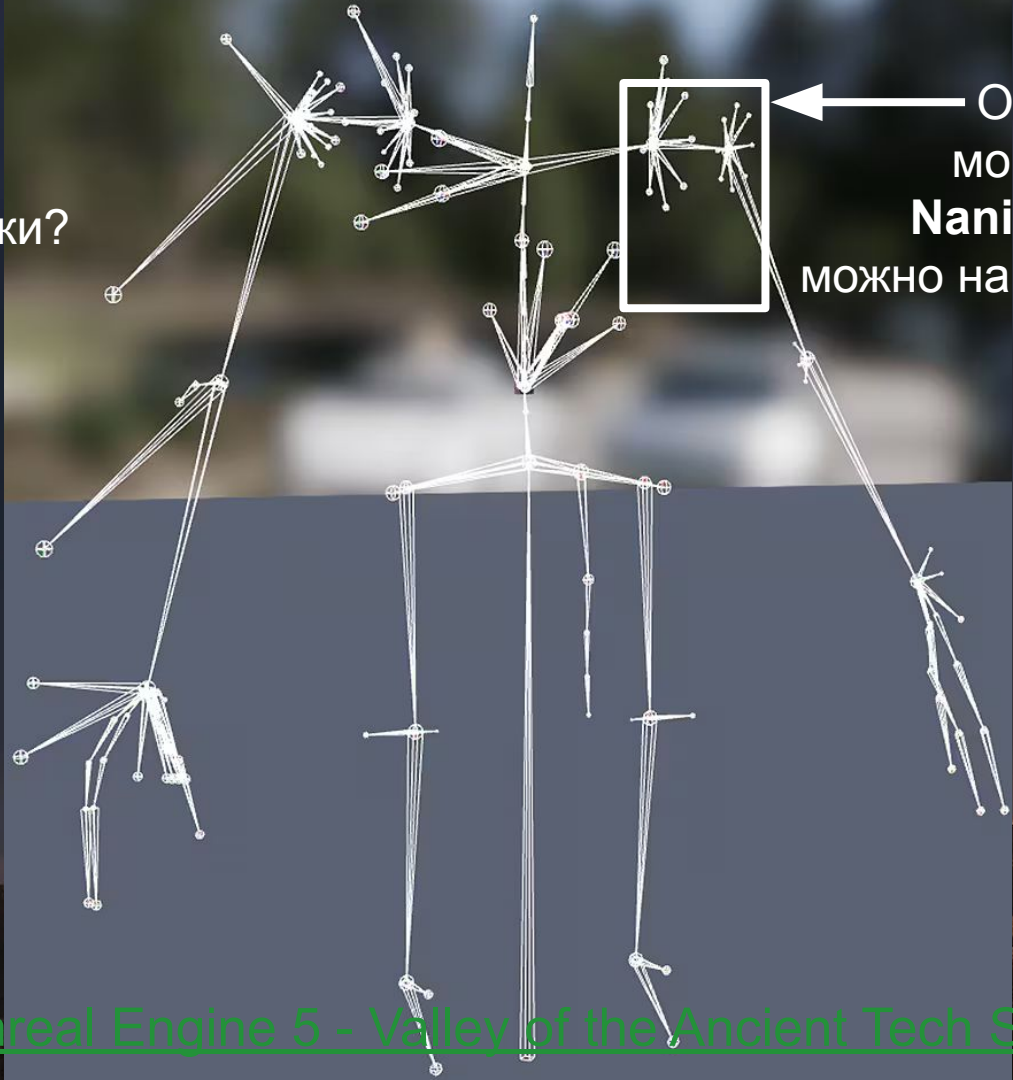
Unreal Engine 5 - Valley of the Ancient Tech Showcase

Что если в игре
анимированные персонажи?

Отдельную
монолитную
Nanite-деталь
можно навесить на
скелет

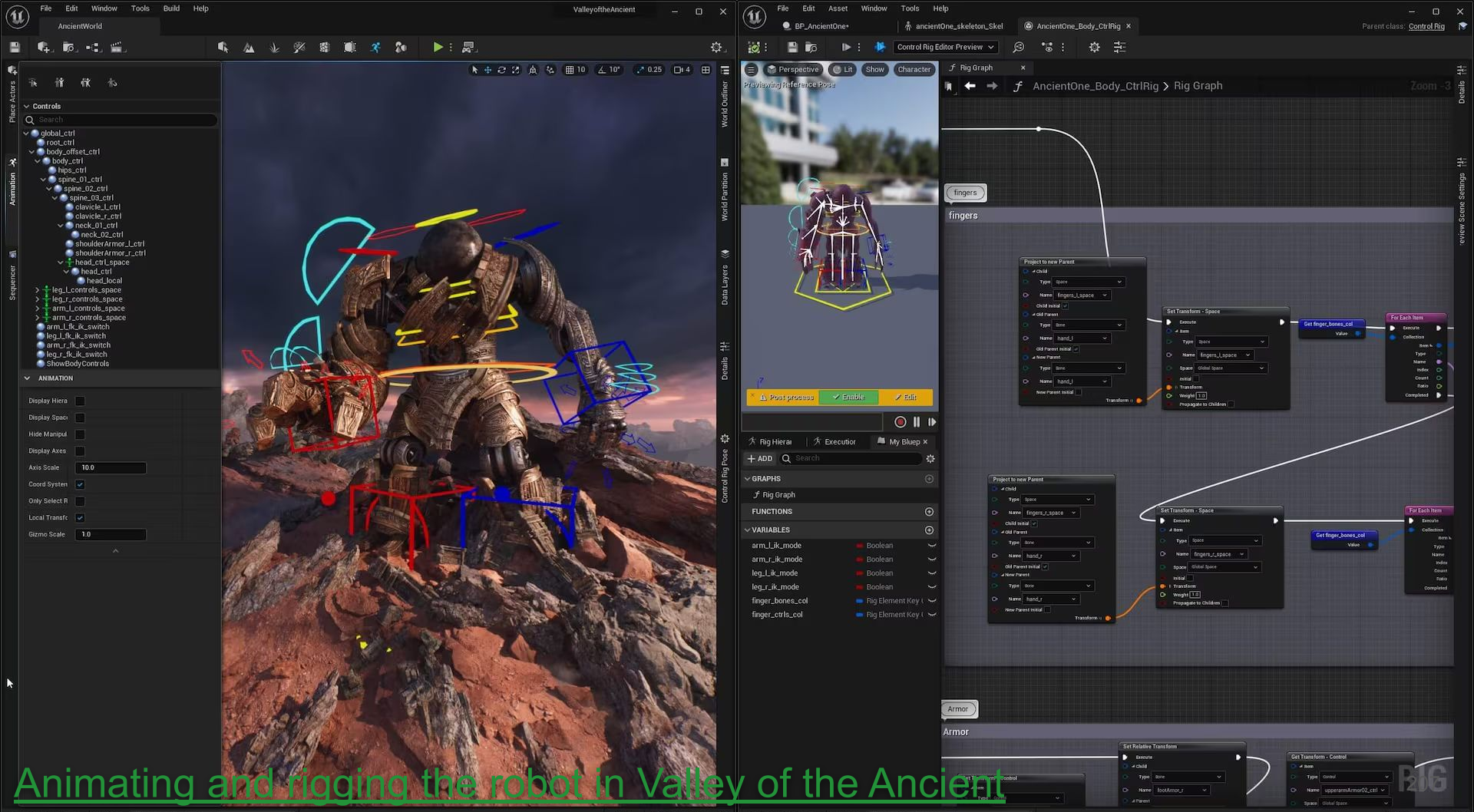
Unreal Engine 5 - Valley of the Ancient Tech Showcase

Что если в игре
анимированные персонажи?

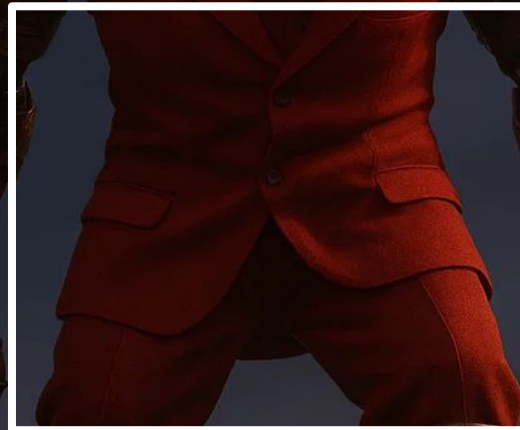
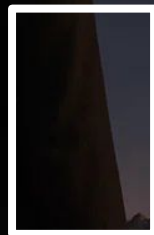


Отдельную
монолитную
Nanite-деталь
можно навесить на
скелет

Unreal Engine 5 - Valley of the Ancient Tech Showcase



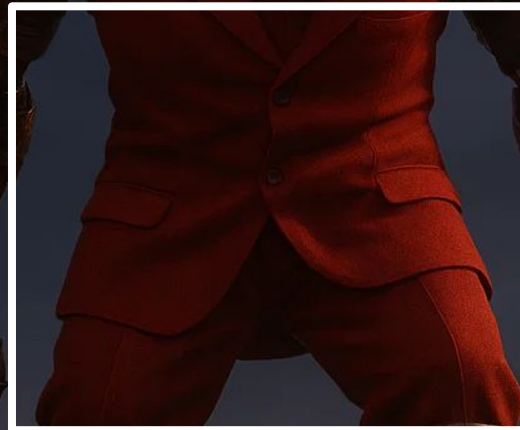
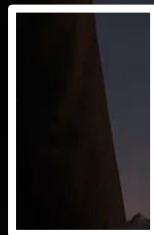
Что если в игре
анимированные персонажи?



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

A surreal illustration featuring a dark, leafless tree with a single, large, light-colored egg perched on its trunk. The background is a deep red, vertically pleated curtain. The floor is covered in a black and gold zigzag pattern. The text 'Глава 239: Деревья и растительность' is written in white at the bottom.

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

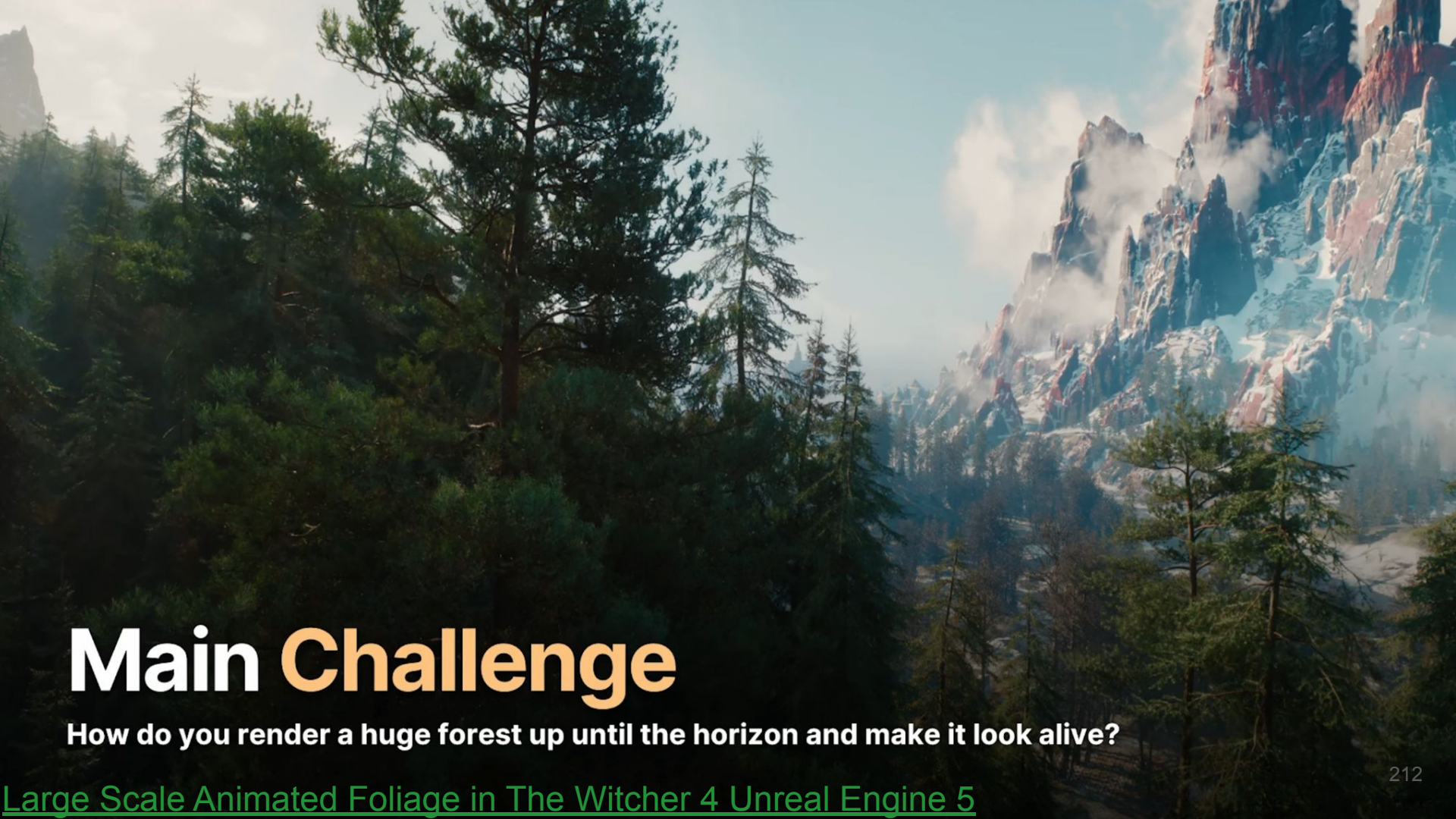
Lead Rendering Programmer
at Epic Games

Thijs Wingerden

Engineering Manager, Rendering
at CD PROJEKT RED

Vignesh Gunasekaran

Senior Engineer, Rendering
at CD PROJEKT RED



Main Challenge

How do you render a huge forest up until the horizon and make it look alive?

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

Why is foliage **hard**?

Nanite is great but it struggled with foliage



Cards

Alpha-tested geometry

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



100k tris

6m tris

Cards

Alpha-tested geometry

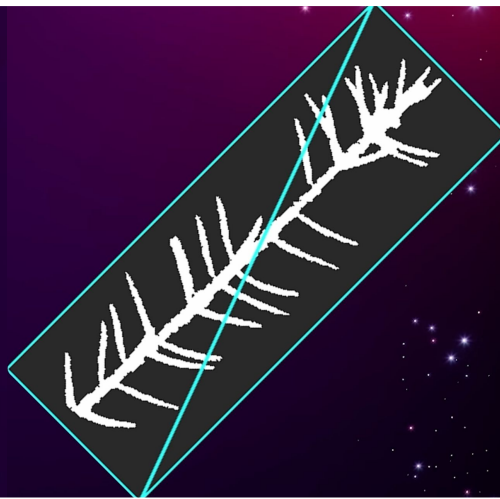
- Alpha testing still expensive.
- Avoid it as much as possible. 3-4x slower.

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](#)



CARDS

100k tris

GEOMETRY

6m tris

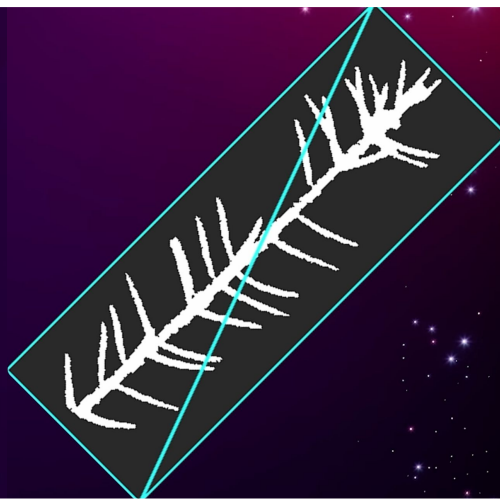
Software barycentrics
(особенно в software
rasterizer) - медленно!
Особенно учитывая
большой **overdraw**.

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

100k tris

GEOMETRY

6m tris

Cards

Alpha-tested geometry

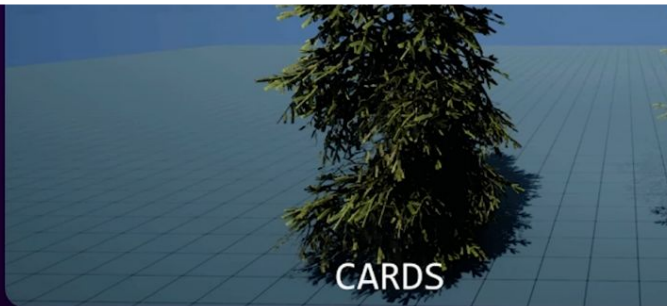
- Alpha testing still expensive.
- Avoid it as much as possible. 3-4x slower.

Barycentrics

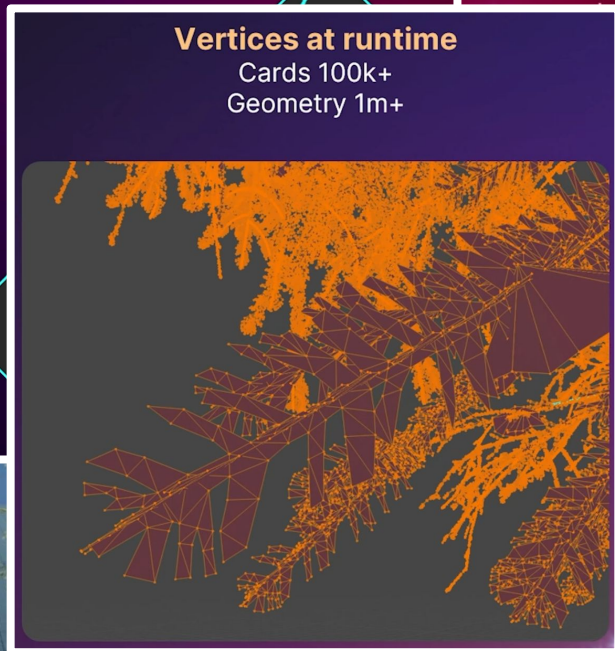
~~Fixed function~~

Dispatch per material

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



100k tris



GEOMETRY

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

Triangles
Visible Streaming Data Size (MB) 63.43

Nanite Triangles: 5,729,526
Nanite Vertices: 5,711,518

Disk Size: 128.21 MB (128.17 MB Nanite)

↑ **x30**

Cards
Visible Streaming Data Size (MB) 5.88

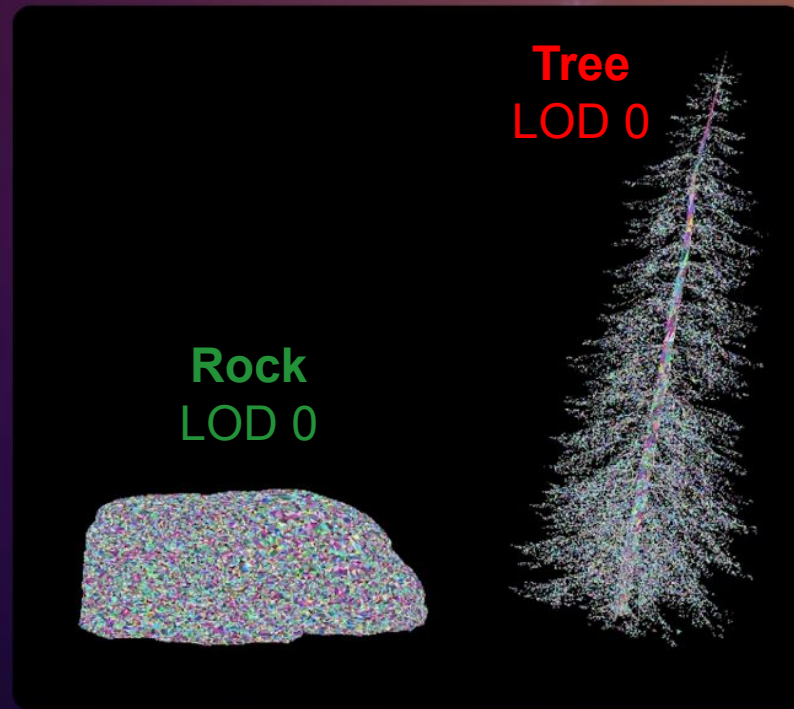
Nanite Triangles: 117,691
Nanite Vertices: 162,755

Disk Size: 4.13 MB (4.11 MB Nanite)

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.



Without Preserve Area

With Preserve Area Enabled

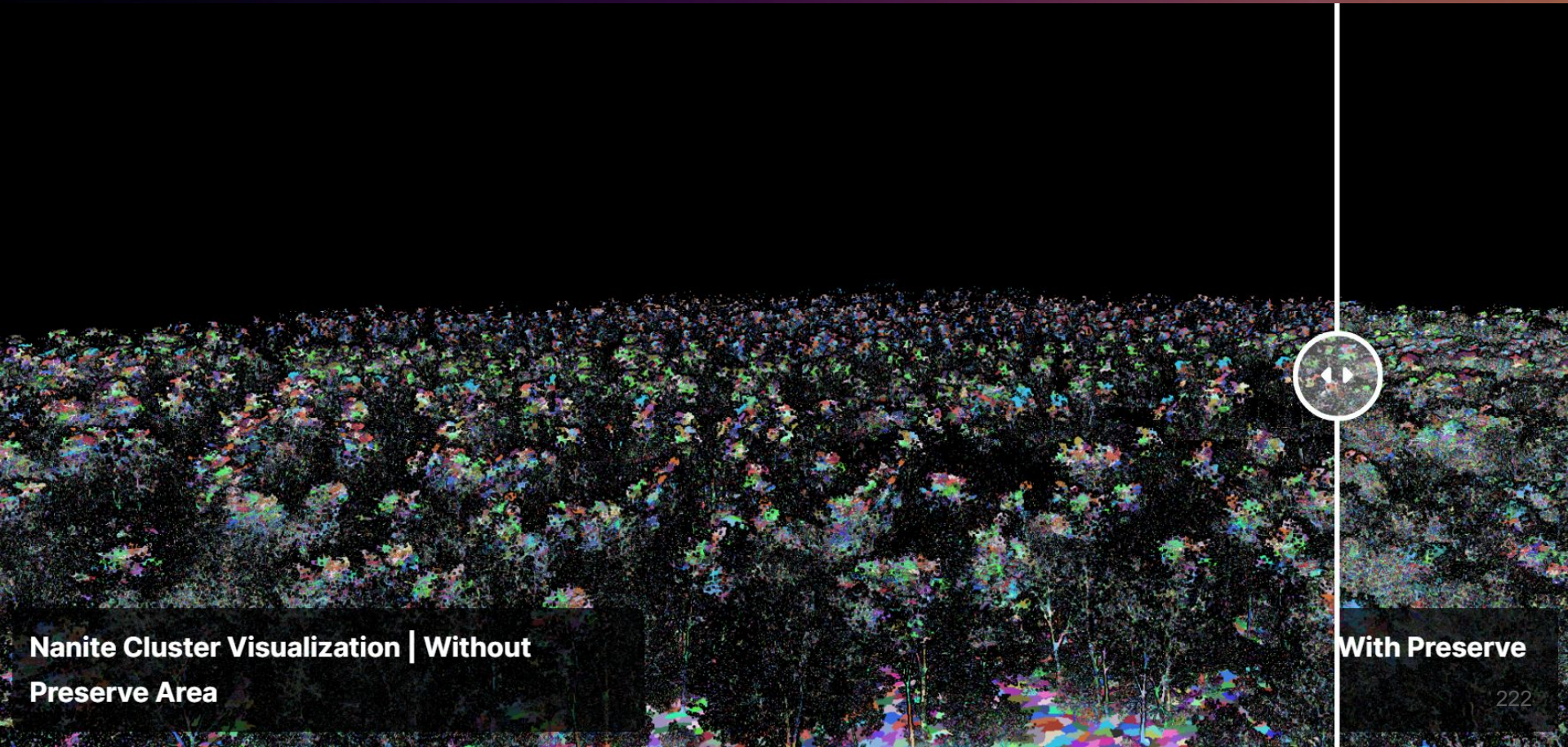
When Nanite simplifies the geometry in the distance ... it ... starts removing some of these disjoint elements completely ... the result **will look thinned** out because there was major surface area loss. **Preserve Area** will redistribute that lost area to the remaining triangles by dilating out the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



Without Preserve Area

With Preserve Area Enabled

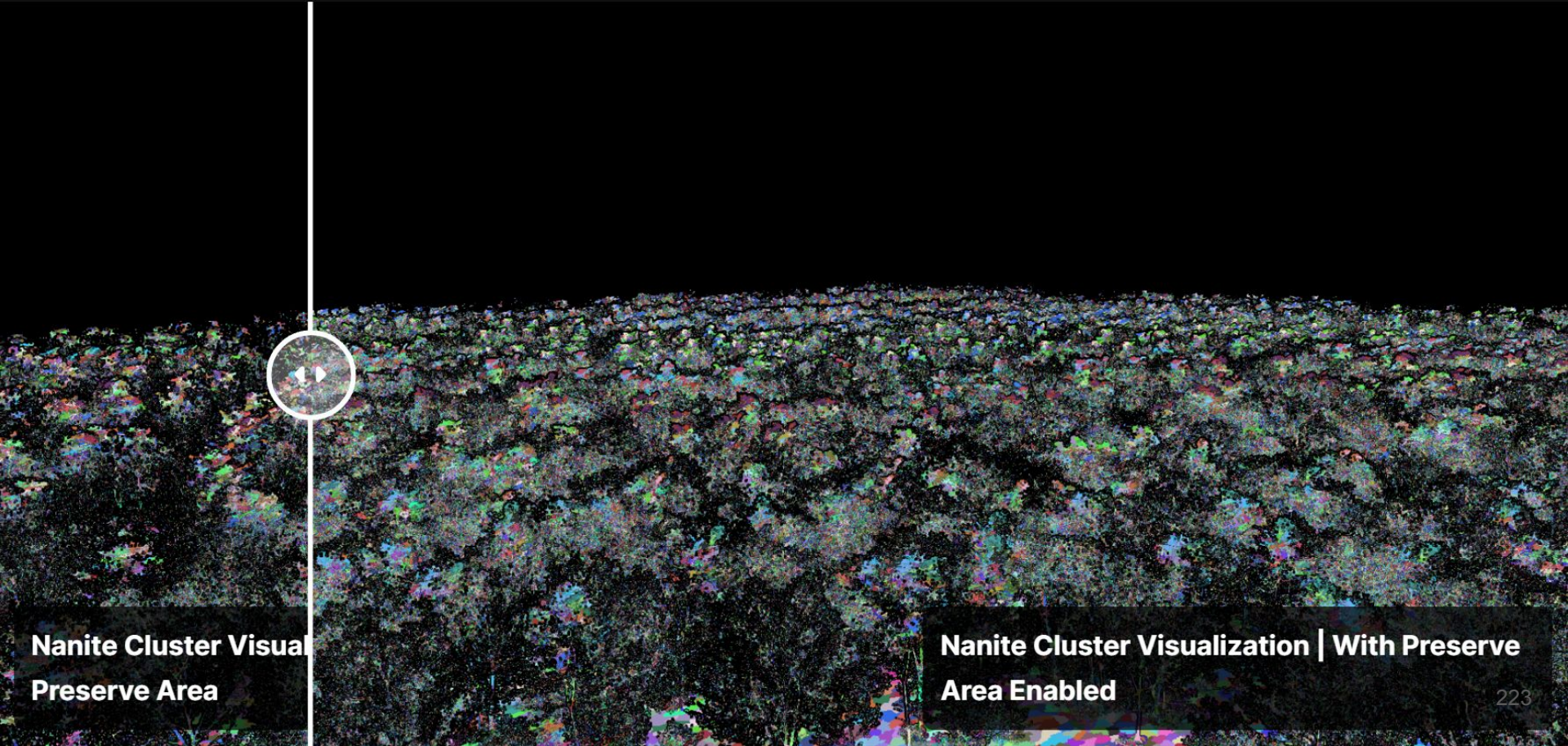
When Nanite simplifies the geometry in the distance ... it ... starts removing some of these disjoint elements completely ... the result **will look thinned** out because there was major surface area loss. **Preserve Area** will redistribute that lost area to the remaining triangles by dilating out the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



Nanite Cluster Visualization | Without
Preserve Area

With Preserve

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.

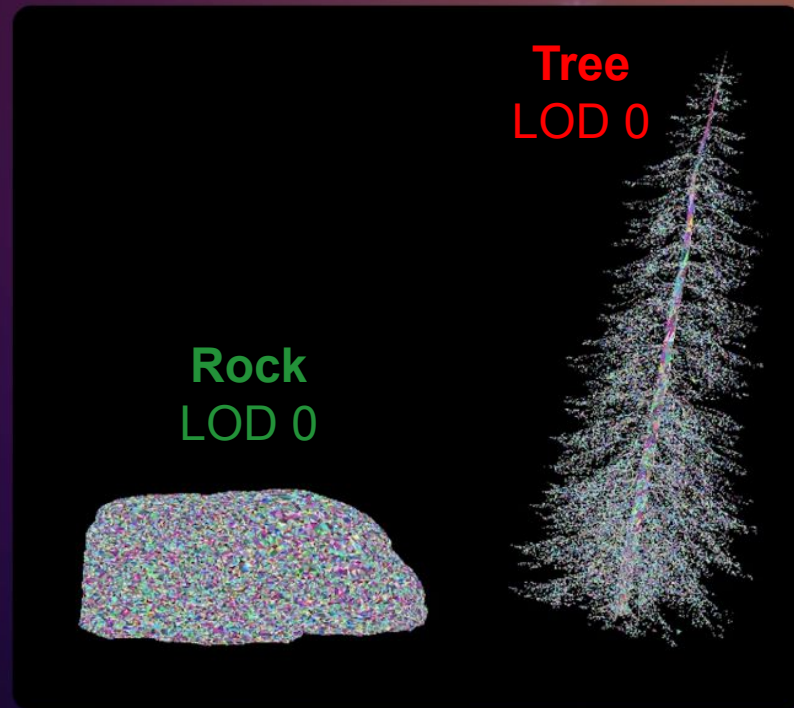


Nanite Cluster Visualization | Preserve Area

Nanite Cluster Visualization | With Preserve Area Enabled

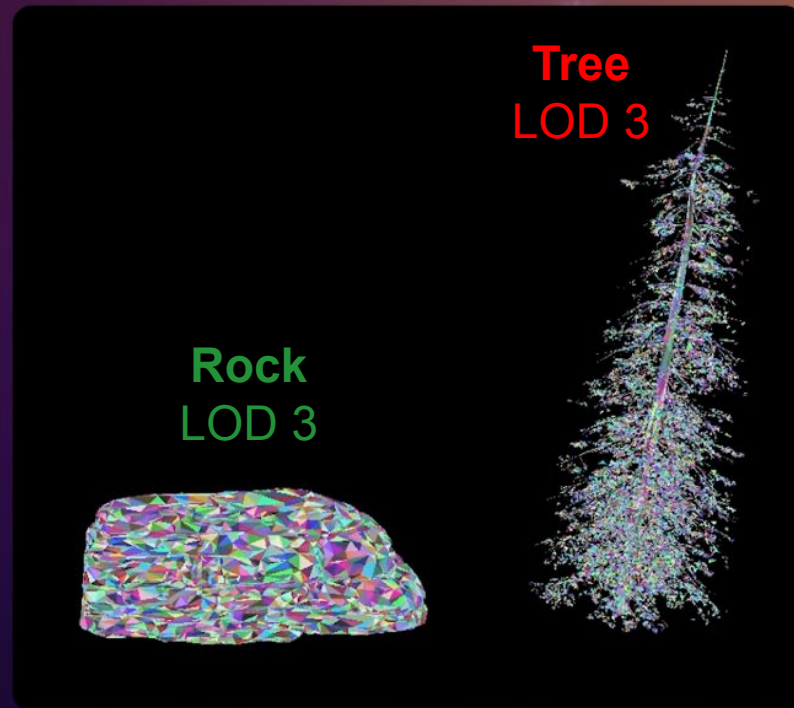
Inefficient LODs

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



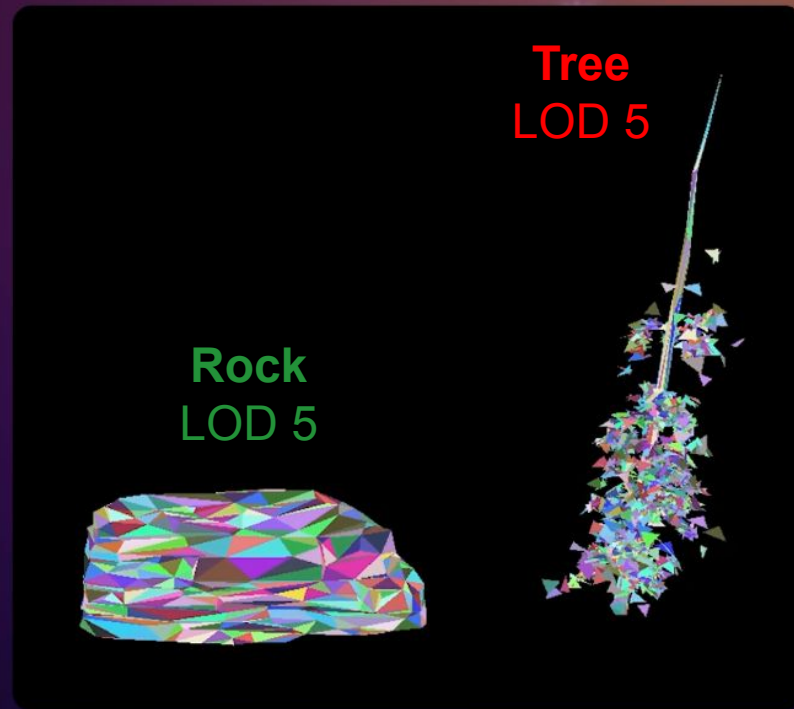
Inefficient LODs

- 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

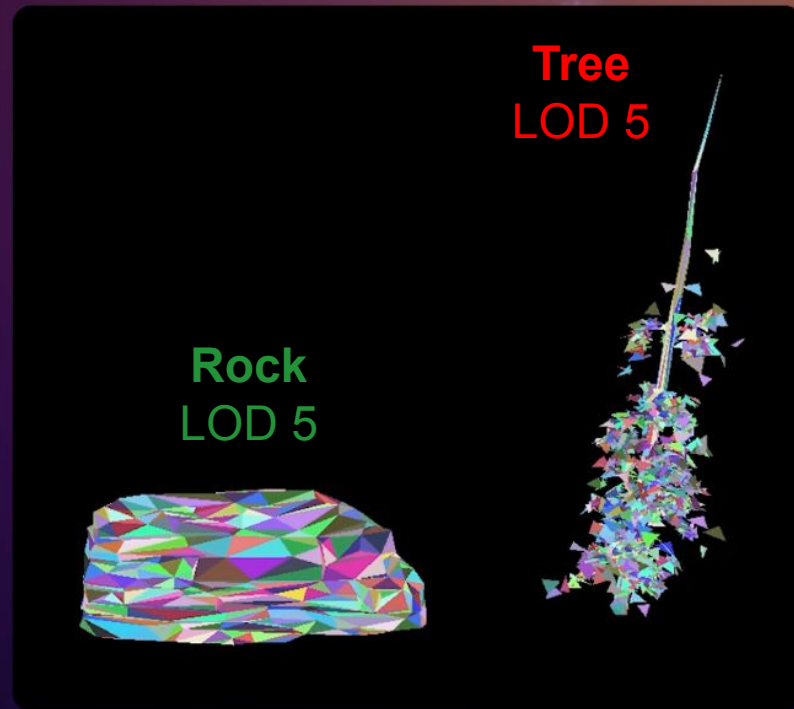
- Even with **Preserve Area**, foliage does not keep the shape at the distance.
- Nanite aggregate geometry reduction is sub-optimal.
- Foliage causes excessive overdraw.



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

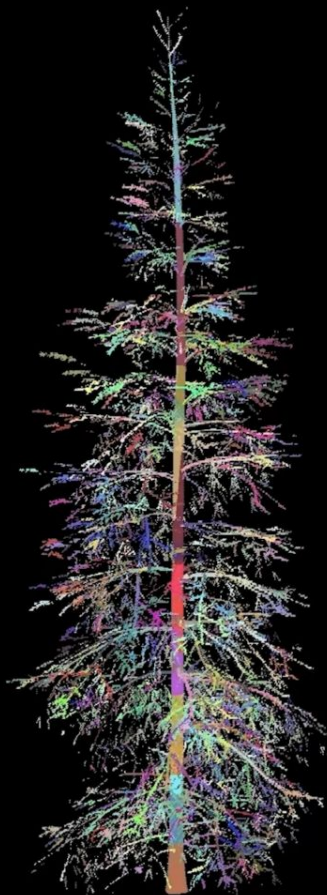
- Even with **Preserve Area**, foliage does not keep the shape at the distance.
- Nanite aggregate geometry reduction is sub-optimal.
- Foliage causes excessive overdraw.



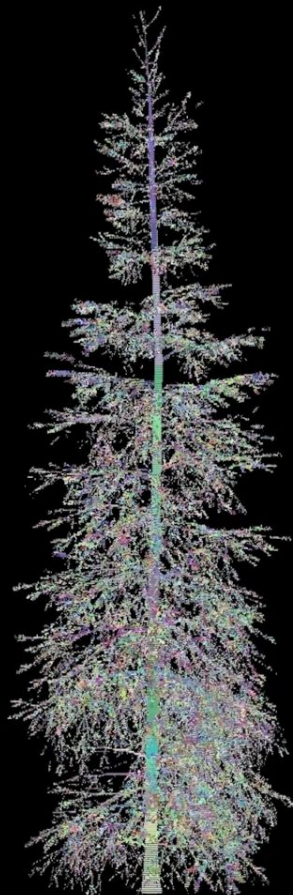
Voxels

- 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.

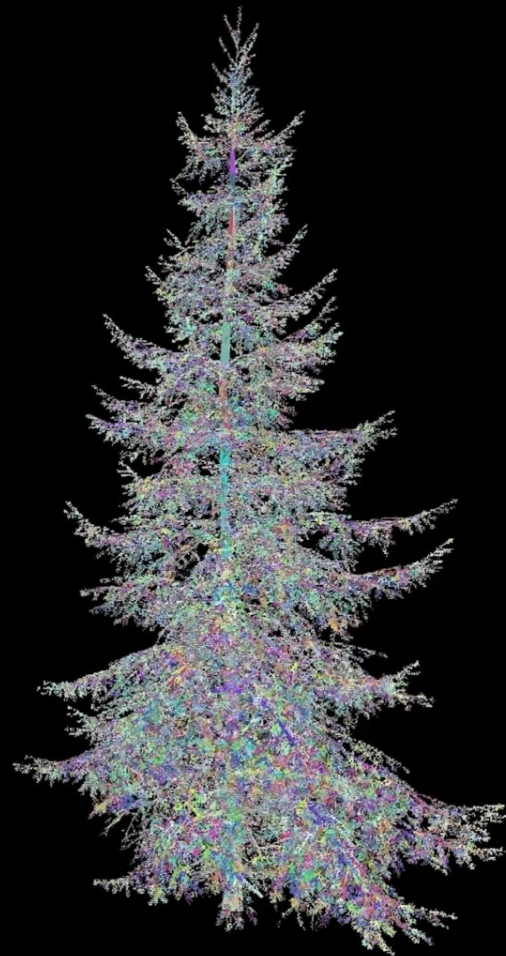




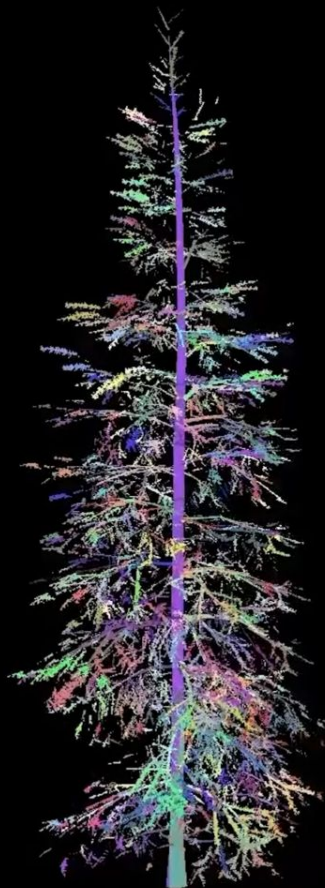
Cards



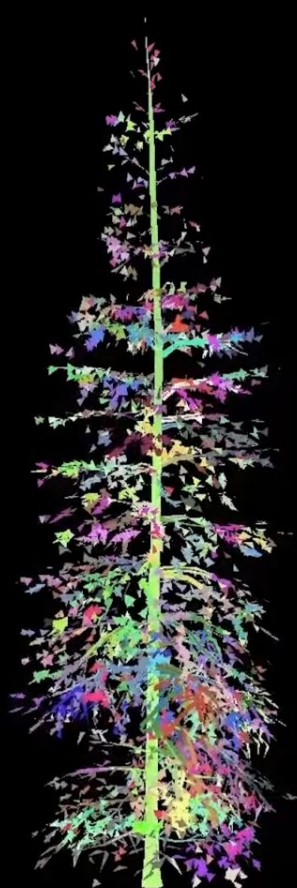
Geometry



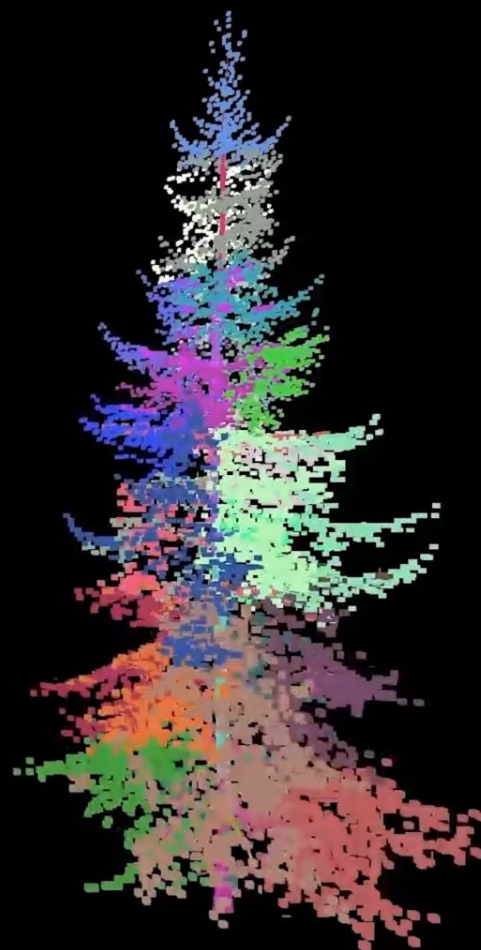
Assemblies & Voxels



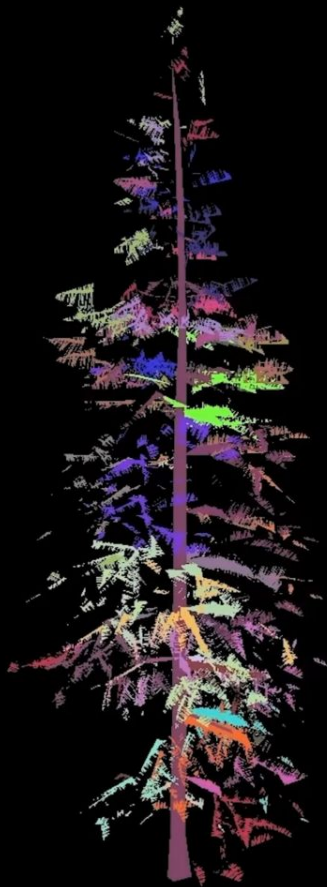
Cards



Geometry



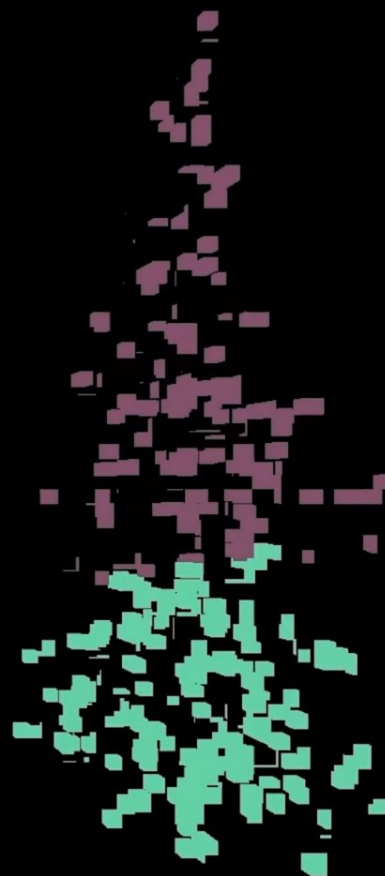
Assemblies & Voxels



Cards



Geometry

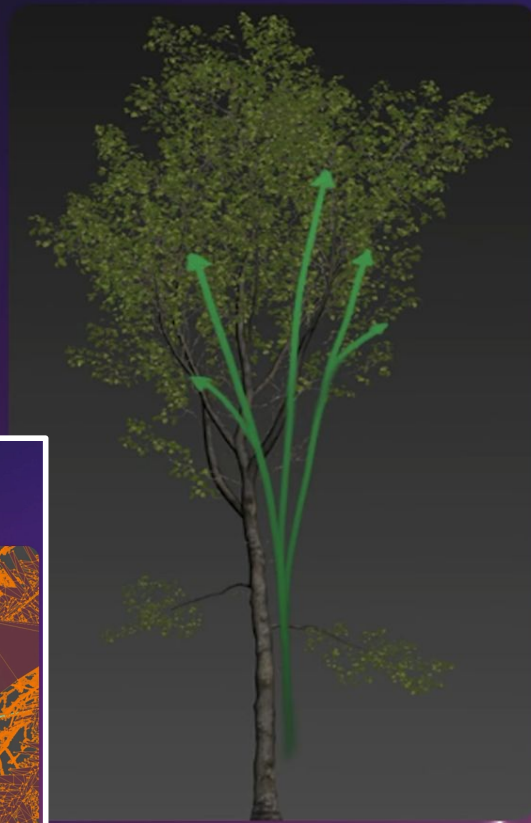


Assemblies & Voxels

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

Skinned Foliage

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



Tech Demo Results in Numbers

- Vista shot renders 20K+ trees
 - Source triangle count in the billions
- Across the valley
 - 26 km²
 - 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



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



CS Space³⁵
Клуб технологий и науки

NANITE

Вопросы?



[@UnicornGlade](https://t.me/UnicornGlade)



[@PolarNick239](https://t.me/PolarNick239)



polarnick239@gmail.com



Николай Полярный



CS Space³⁶

Клуб технологий и науки

Курс фотограмметрии: 3D реконструкция

По множеству фотографий одной и той же сцены получить цифровую модель



DJI_0127



DJI_0128



DJI_0129



DJI_0130



DJI_0131



DJI_0132



DJI_0135



DJI_0136



DJI_0137



DJI_0138



DJI_0139

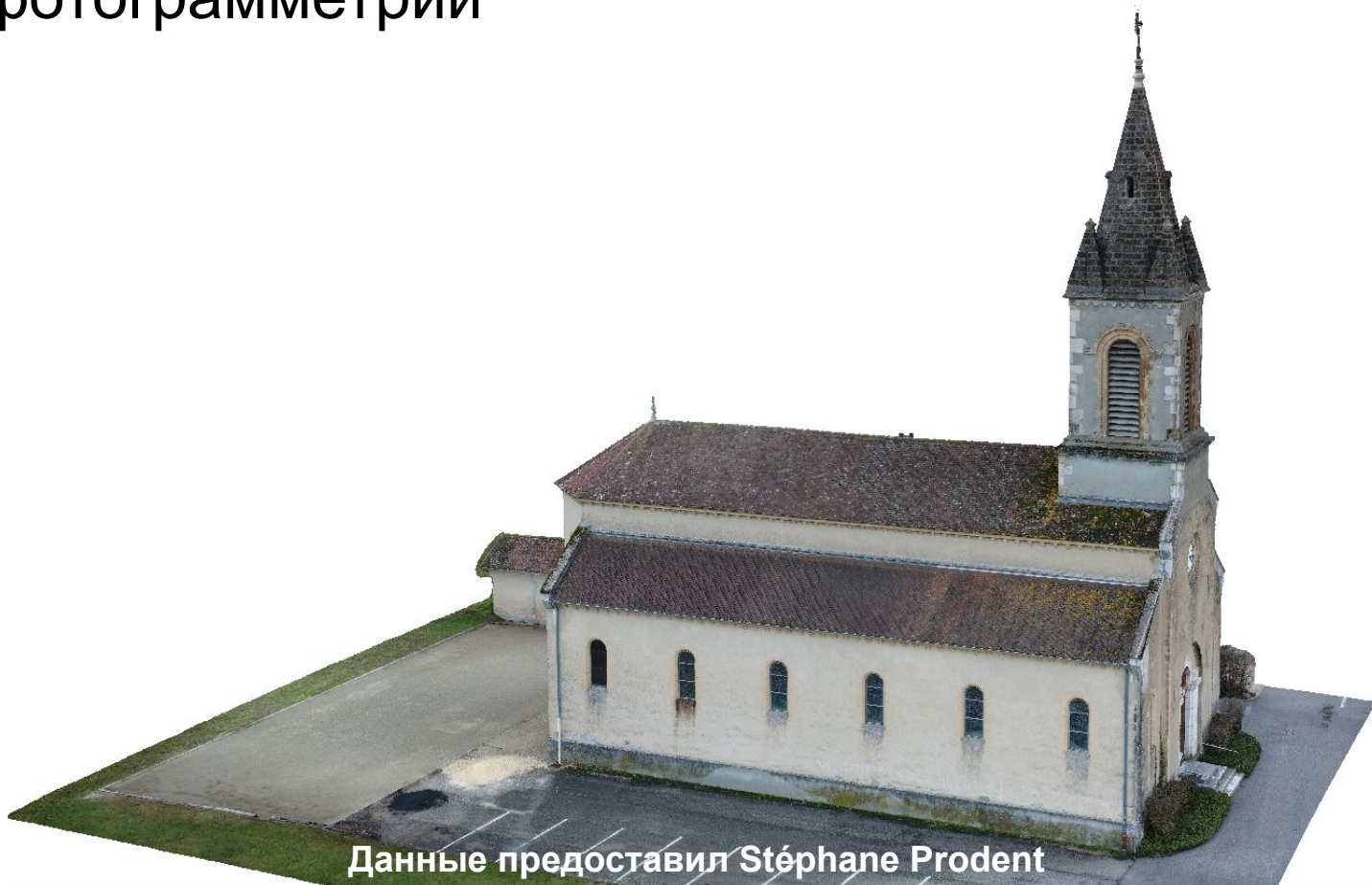


Курс фотограмметрии





Курс фотограмметрии



Agisoft



Metashape

Данные предоставил Stéphane Prodent