King Abdullah University of Science and Technology

KAUST

# CS 380 - GPU and GPGPU Programming
# Lecture 21: GPU Virtual Geometry (and GPU Virtual Texturing)

Markus Hadwiger, KAUST

# Reading Assignment #12 (until Nov 20)

## Read (required):

- Programming Massively Parallel Processors book, 4th edition

  **Chapter 5** (Memory architecture and data locality)
  **Chapter 6** (Performance considerations)

## Read (optional):

- Stream processing

  **https://en.wikipedia.org/wiki/Stream_processing**

- Linear algebra operators for GPU implementation of numerical algorithms, Krueger and Westermann, SIGGRAPH 2003

  **https://dl.acm.org/doi/10.1145/882262.882363**

- A Survey of General-Purpose Computation on Graphics Hardware (2007)

  **https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2007.01012.x**

# Virtual Geometry (and Texturing)

# Unreal Engine 5 Virtual Geometry: Nanite

A Deep Dive into Nanite Virtualized Geometry (Siggraph 2021 course talk)
`https://www.youtube.com/watch?v=eviSykqSUUw`

Brian Karis, Epic Games


See also

- Keynote at HPG 2022:
  Journey to Nanite, Brian Karis
  `https://www.youtube.com/watch?v=NRnj_lnpORU`
- Lumen: Real-time Global Illumination in Unreal Engine 5 (Siggraph 2022 course talk),
  Daniel Wright et al., Epic Games
  `https://advances.realtimerendering.com/s2022/SIGGRAPH2022-Advances-Lumen-Wright%20et%20al.pdf`

# Virtual Geometry

## The Dream

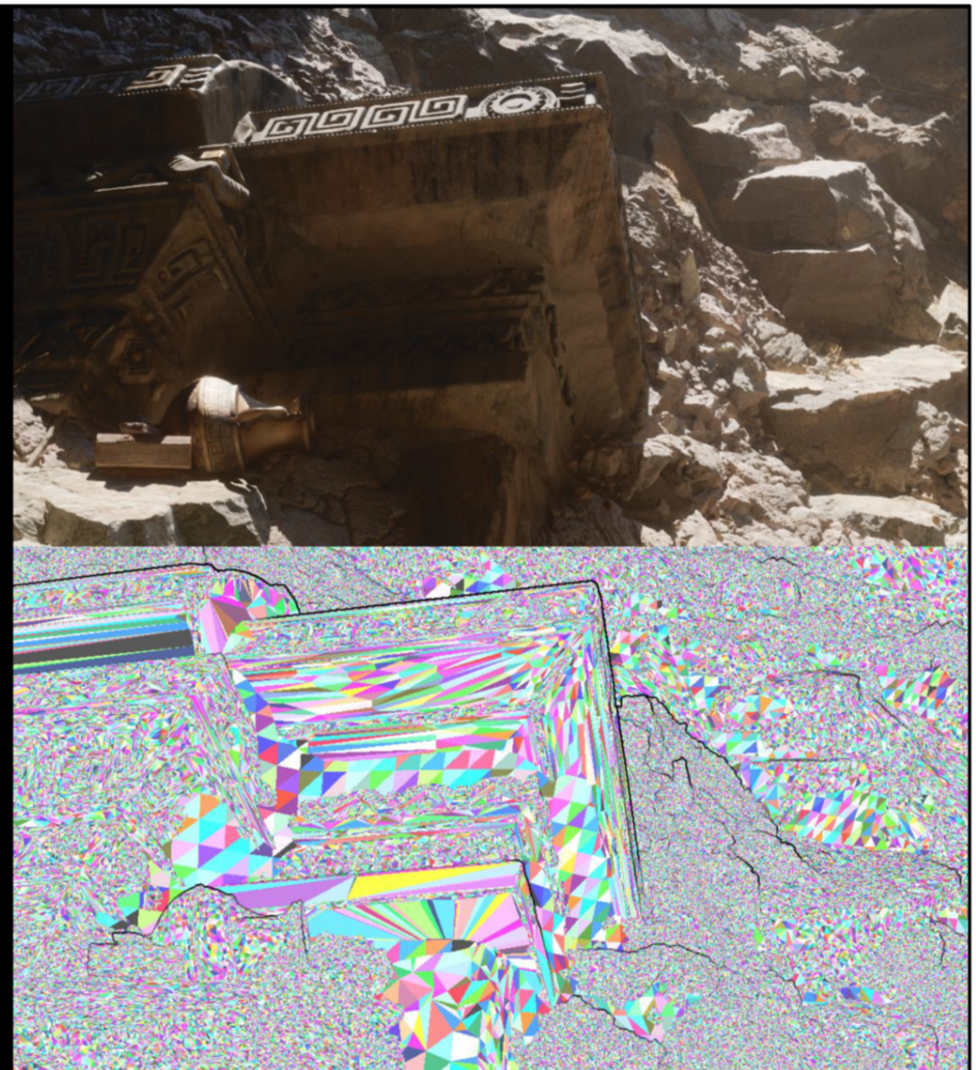
- Virtualize geometry like we did textures
- No more budgets
  - Polycount
  - Draw calls
  - Memory
- Directly use film quality source art
  - No manual optimization required
- No loss in quality

# Virtual Geometry

# Virtual Geometry

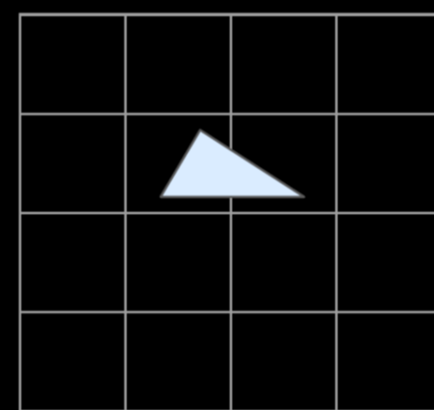
## Pixel scale detail

- Can we hit pixel scale detail with triangles > 1 pixel?
  - Depends how smooth
  - In general no
- We need to draw pixel sized triangles



# Tiny triangles

- Terrible for typical rasterizer
- Typical rasterizer:
  - Macro tile binning
  - Micro tile 4x4
  - Output 2x2 pixel quads
  - Highly parallel in pixels not triangles
- Modern GPUs setup 4 tris/clock max
  - Outputting SV_PrimitiveID makes it even worse
- Can we beat the HW rasterizer in SW?

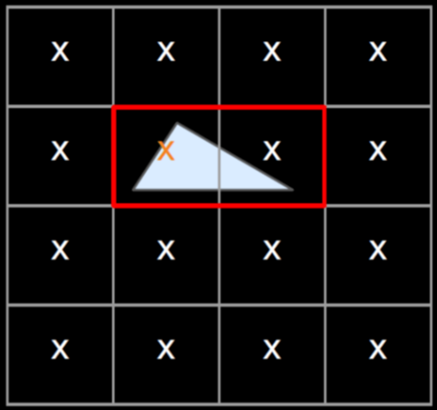


# Software Rasterization

3x faster!

# **Micropoly** software rasterizer

- 128 triangle clusters => threadgroup size 128
- 1 thread per vertex
  - Transform position
  - Store in groupshared
  - If more than 128 verts loop (max 2)
- 1 thread per triangle
  - Fetch indexes
  - Fetch transformed positions
  - Calculate edge equations and depth gradient
  - Calculate screen bounding rect
  - For all pixels in rect
    - If inside all edges then write pixel

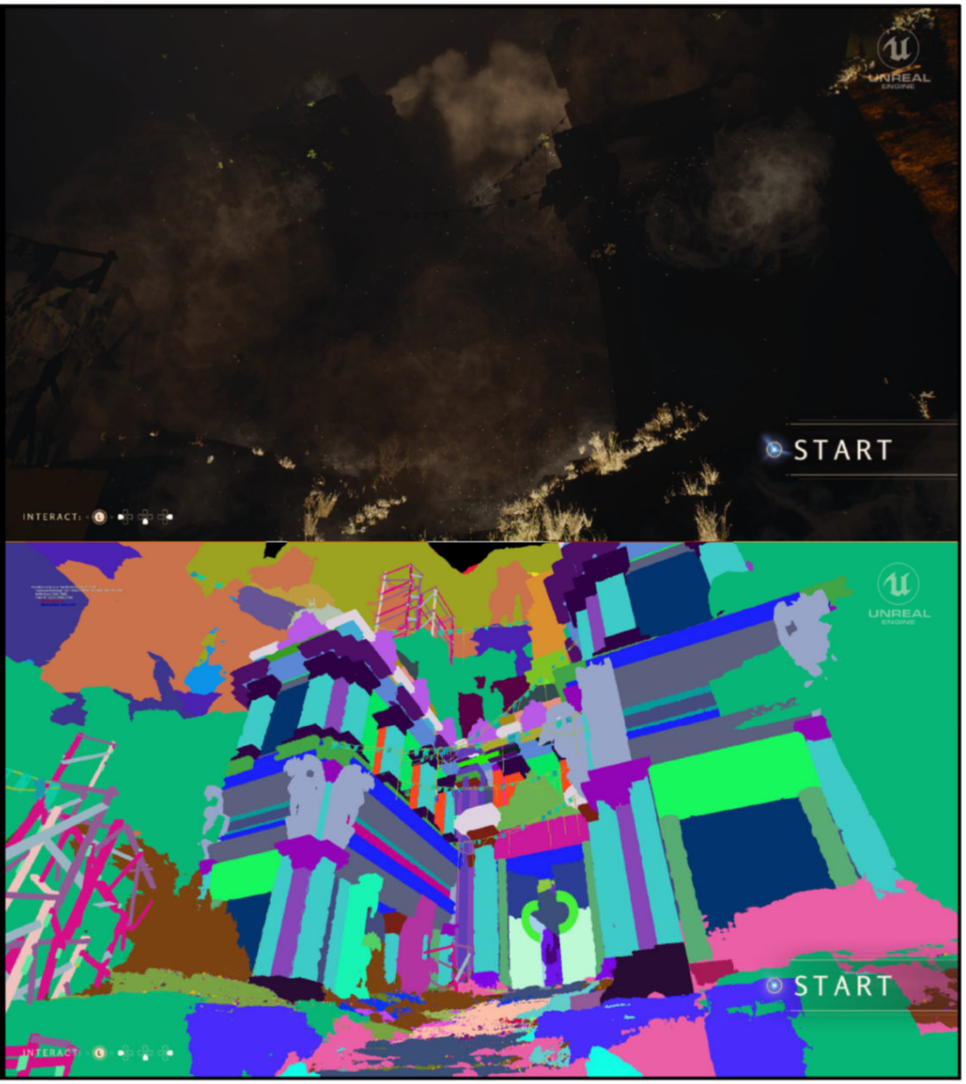


# Hardware Rasterization

- What about big triangles?
  - Use HW rasterizer
- Choose SW or HW per cluster
- Also uses 64b atomic writes to UAV

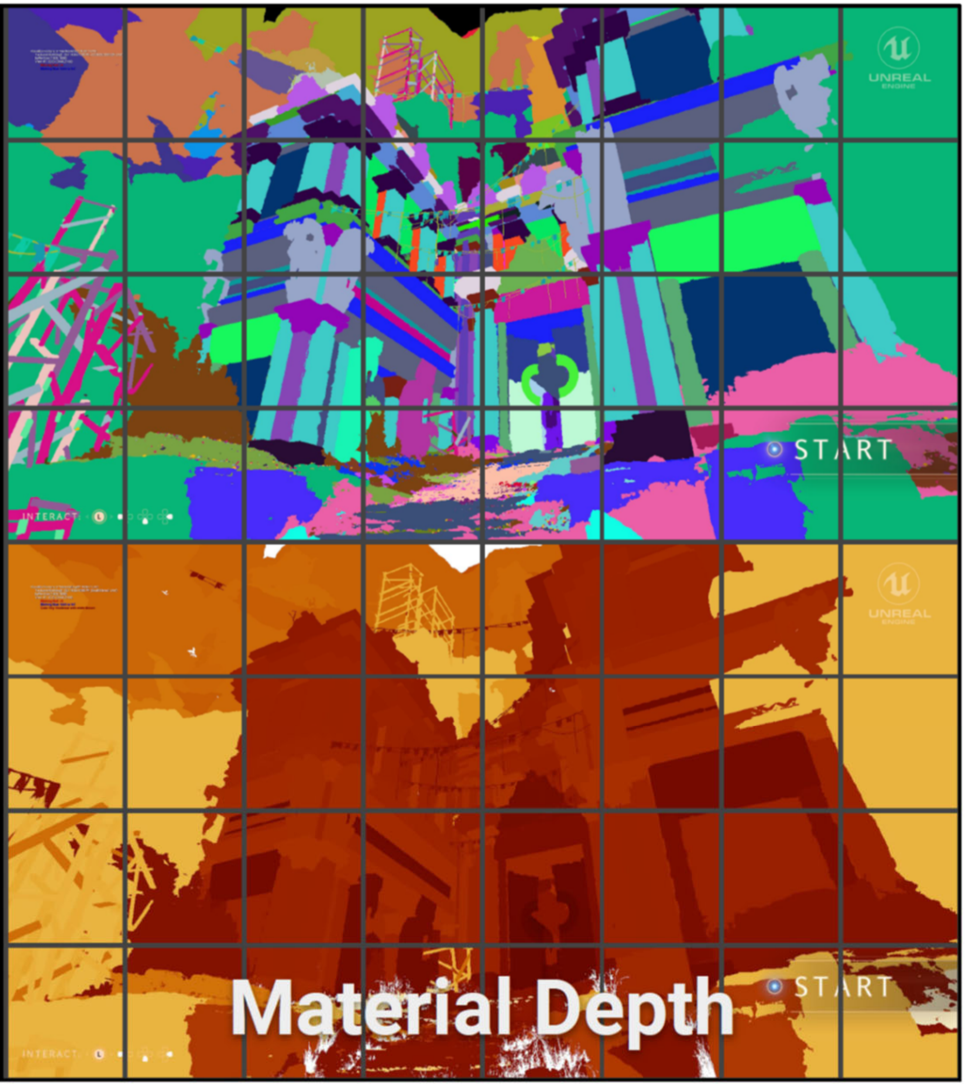
# Material shading

- Full screen quad per unique material
- Skip pixels not matching this material ID
- CPU unaware if some materials have no visible pixels
  - Material draw calls issued regardless
  - Unfortunate side effect of GPU driven
- How to do efficiently?
  - Don't test every pixel for matching material ID for every material pass



# Material culling

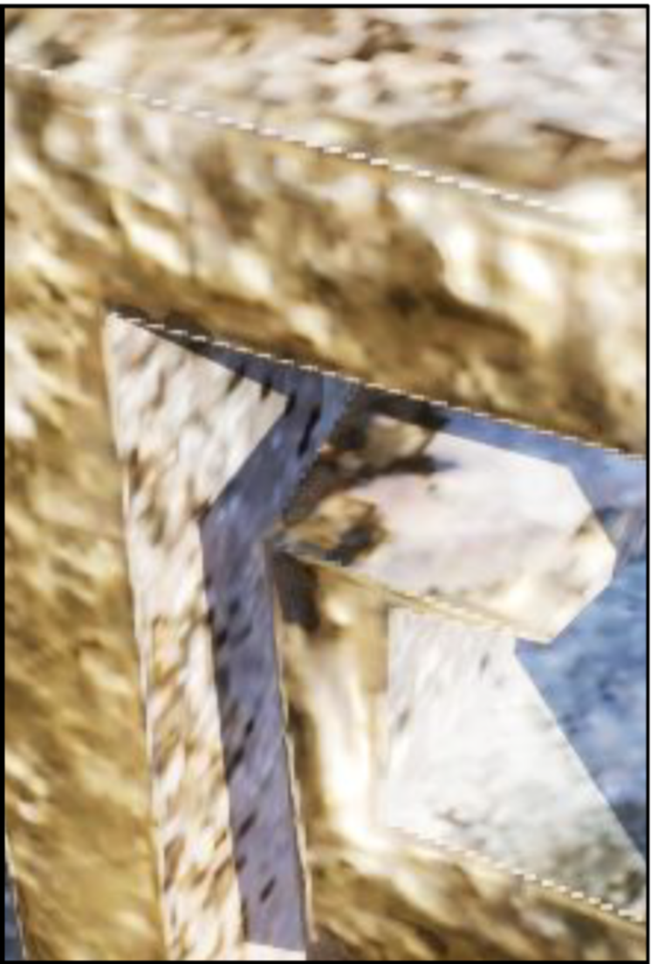
- Material covers small portion of the screen
  - HiZ handles this OK
  - We can do better
- Coarse tile classification / culling
  - Render 8x4 grid of tiles per material
  - Same shading approach as full screen quads
- Tile killed in vertex shader from 32b mask
  - X=NaN



# UV derivatives

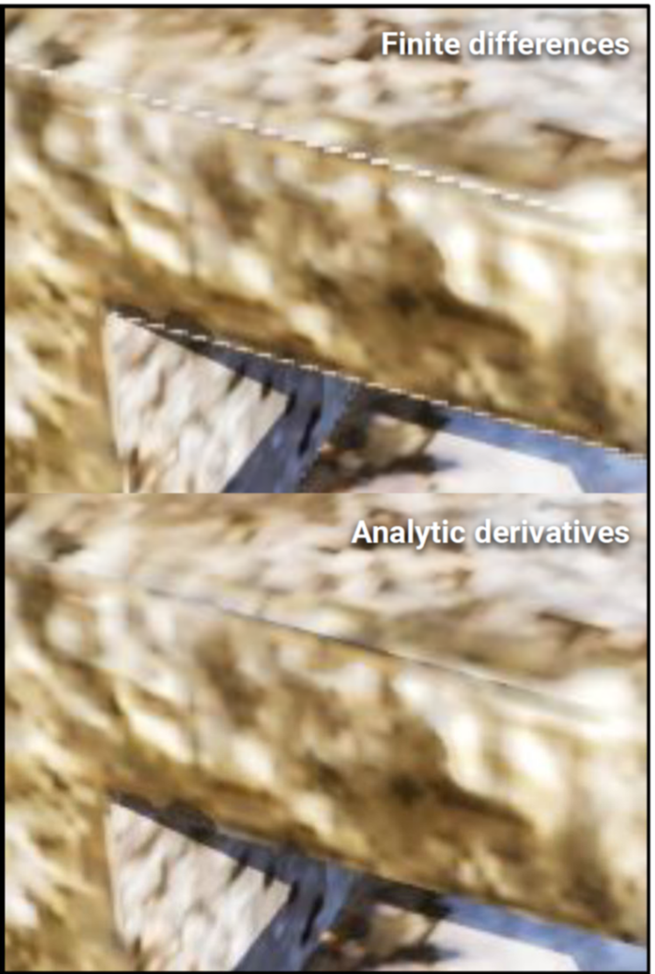
- Still a coherent pixel shader so we have finite difference derivatives
- Pixel quads span
  - Triangles ⇦ Good!
- Also span
  - Depth discontinuities
  - UV seams
  - Different objects

} Not good!



# Analytic derivatives

- Compute analytic derivatives
  - Attribute gradient across triangle
- Propagate through material node graph using chain rule
- If derivative can't be evaluated analytically
  - Fall back to finite differences
- Used to sample textures with SampleGrad
- Additional cost tiny
  - <2% overhead for material pass
  - Only affects calculations that affect texture sampling
  - Virtual texturing code already does SampleGrad



# Pipeline numbers

## Main pass

| | |
|---|---|
| Instances pre-cull | 896322 |
| Instances post-cull | 3668 |
| Cluster node visits | 39274 |
| Cluster candidates | 1536794 |
| Visible clusters SW | 184828 |
| Visible clusters HW | 6686 |

## Post pass

| | |
|---|---|
| Instances pre-cull | 102804 |
| Instances post-cull | 365 |
| Cluster node visits | 19139 |
| Cluster candidates | 458805 |
| Visible clusters SW | 7370 |
| Visible clusters HW | 536 |

## Total rasterized

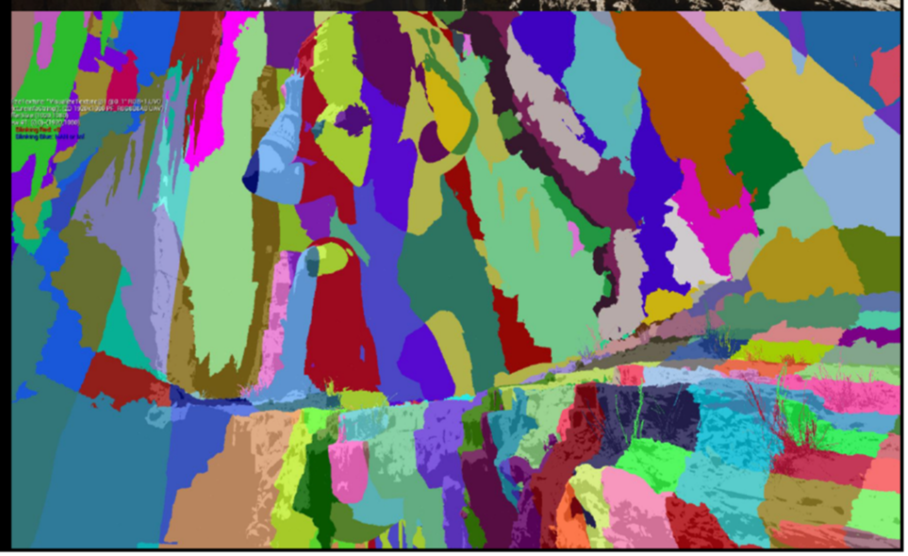
| | |
|---|---|
| Clusters | 199,420 |
| Triangles | 25,041,711 |
| Vertices | 19,851,262 |

# Nanite shadows

- Ray trace?
  - DXR isn't flexible enough
    - Complex LOD logic
    - Custom triangle encoding
    - No partial BVH updates
- Want a raster solution
  - Leverage all our other work
- Most lights don't move
  - Should cache as much as possible

# Virtual shadow maps

- Nanite enables new techniques
- 16k x 16k shadow maps everywhere
  - Spot: 1x projection
  - Point: 6x cube
  - Directional: Nx clipmaps
- Pick mip level where 1 texel = 1 pixel
- Only render the shadow map pixels that are visible
- Nanite culled and LODed to the detail required



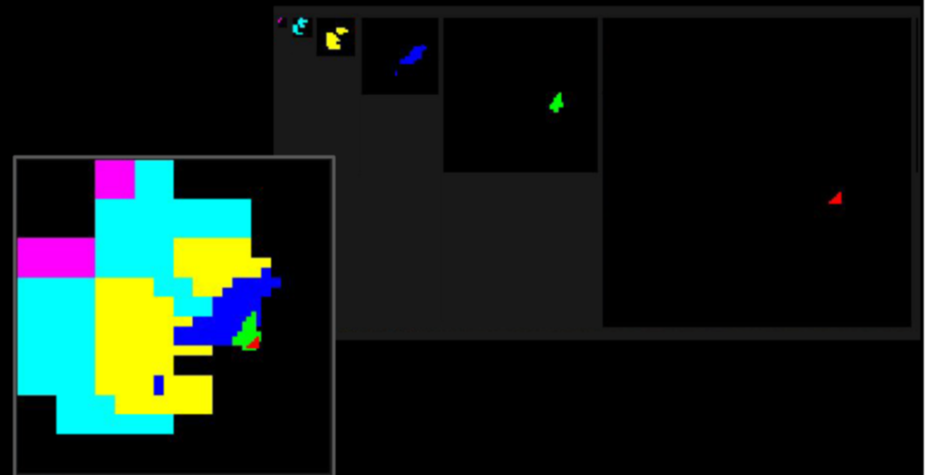
# Virtual shadow maps

- Page size = 128 x 128
- Page table = 128 x 128, with mips

- Mark needed pages
  - Screen pixels project to shadow space
  - Pick mip level where 1 texel = 1 pixel
  - Mark that page

- Allocate physical pages for all needed

- If cached page already exists use that
  - And wasn't invalidated
  - Remove from needed page mask



Thank you.