CS 380 - GPU and GPGPU Programming
Lecture 17: GPU Texturing, Pt. 4; Virtual Texturing, Virtual Geometry

Markus Hadwiger, KAUST

Reading Assignment #10 (until Nov 8)

Read (required):

- **Brook for GPUs: Stream Computing on Graphics Hardware**
  Ian Buck et al., SIGGRAPH 2004

`http://graphics.stanford.edu/papers/brookgpu/`

Read (optional):

- **The Imagine Stream Processor**
  Ujval Kapasi et al.; IEEE ICCD 2002

`http://cva.stanford.edu/publications/2002/imagine-overview-iccd/`

- **Merrimac: Supercomputing with Streams**
  Bill Dally et al.; SC 2003

`https://dl.acm.org/citation.cfm?doid=1048935.1050187`

Quiz #2: Nov 10

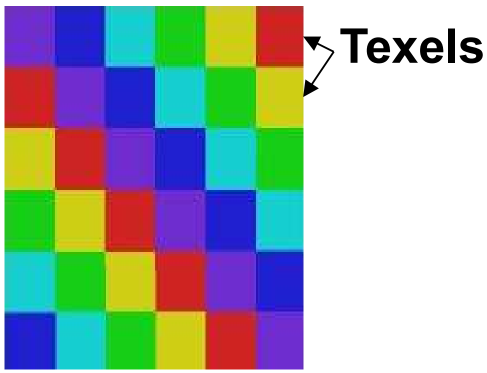
Organization

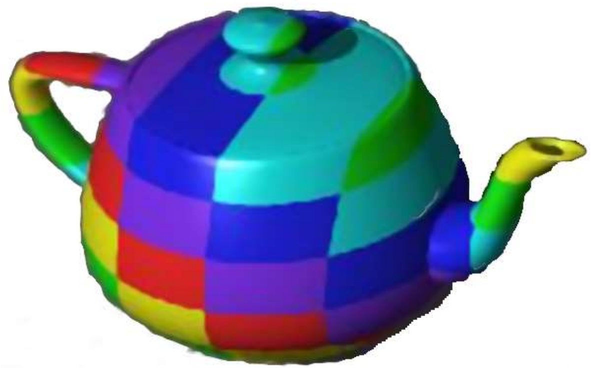
- First 30 min of lecture
- No material (book, notes, ...) allowed

Content of questions

- Lectures (both actual lectures and slides)
- Reading assigments
- Programming assignments (algorithms, methods)
- Solve short practical examples

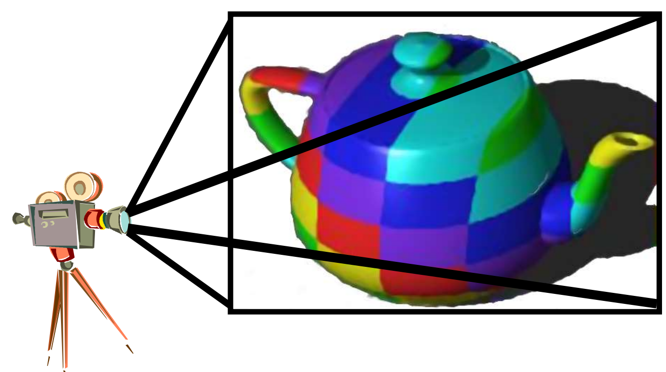
Texturing: General Approach





Texture space (u,v) **Object space** (x_O,y_O,z_O) **Image Space** (x_I,y_I)

Parametrization → **Rendering (Projection etc.)** →



Interpolation Type + Purpose #2:

Interpolation of Samples in Texture Space

(Multi-Linear Interpolation)

Magnification (Bi-linear Filtering Example)

Original image

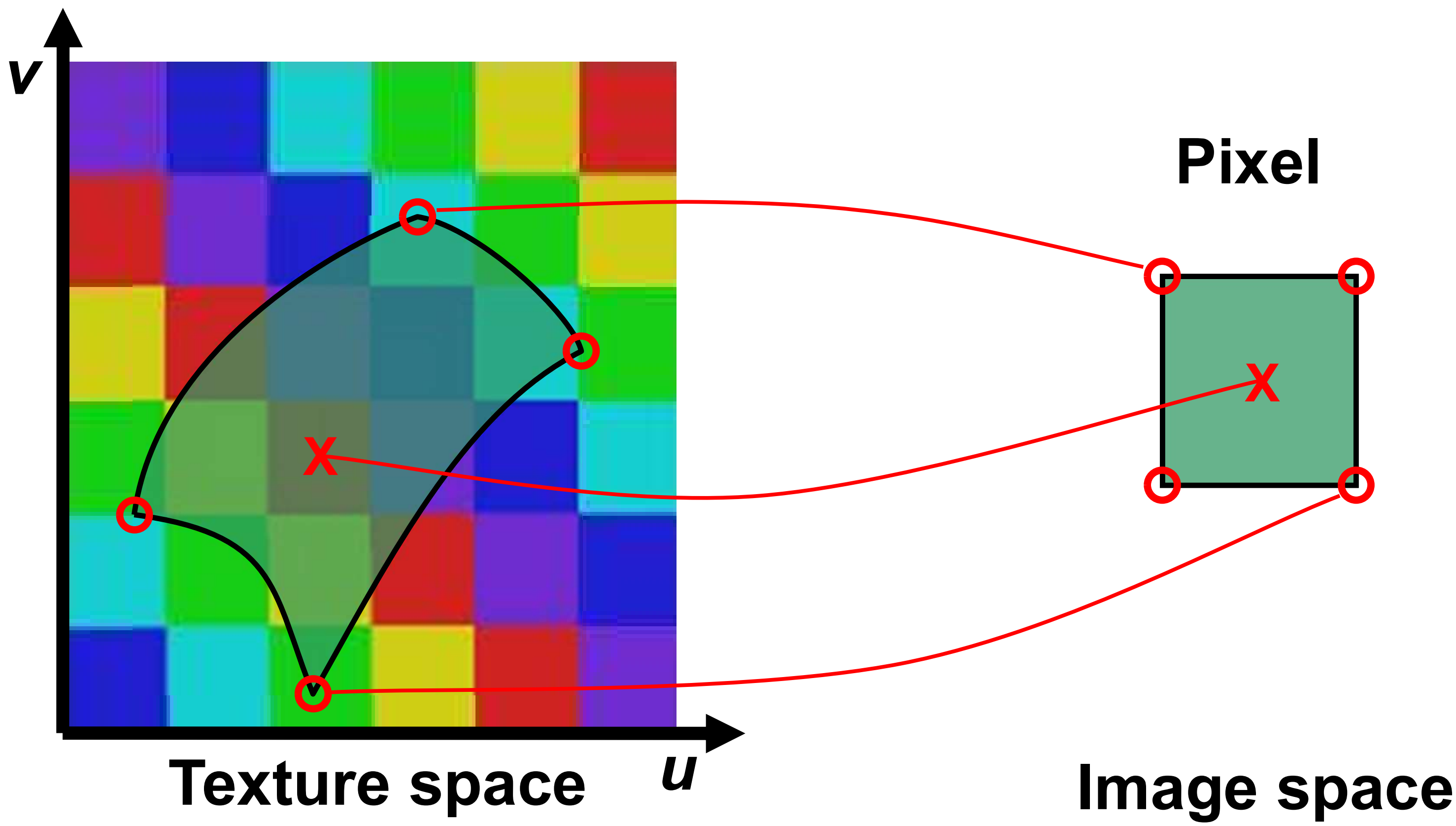
Nearest neighbor

Bi-linear filtering

Texture Minification

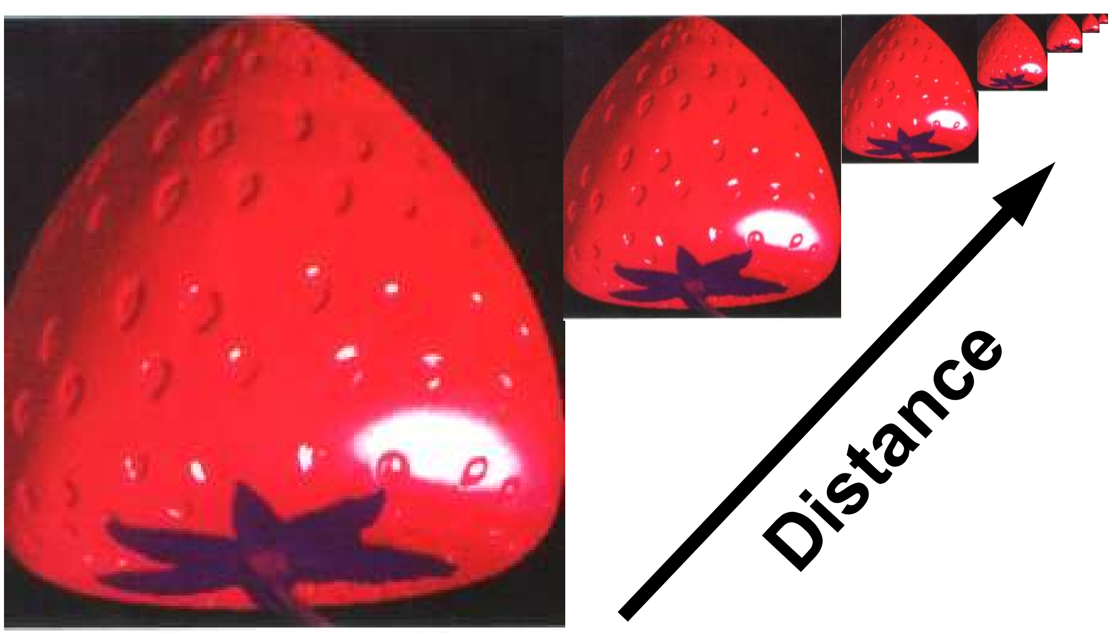
Texture Anti-Aliasing: Minification

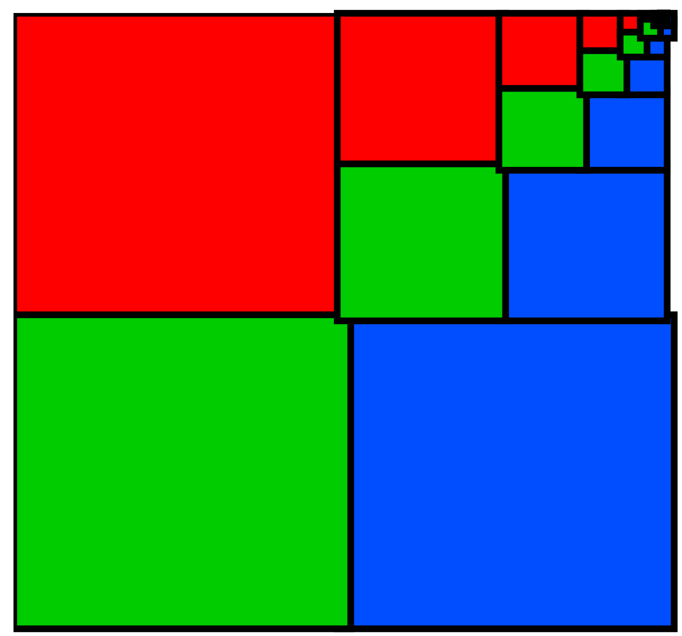
- A good pixel value is the weighted mean of the pixel area projected into texture space



Texture Anti-Aliasing: MIP Mapping

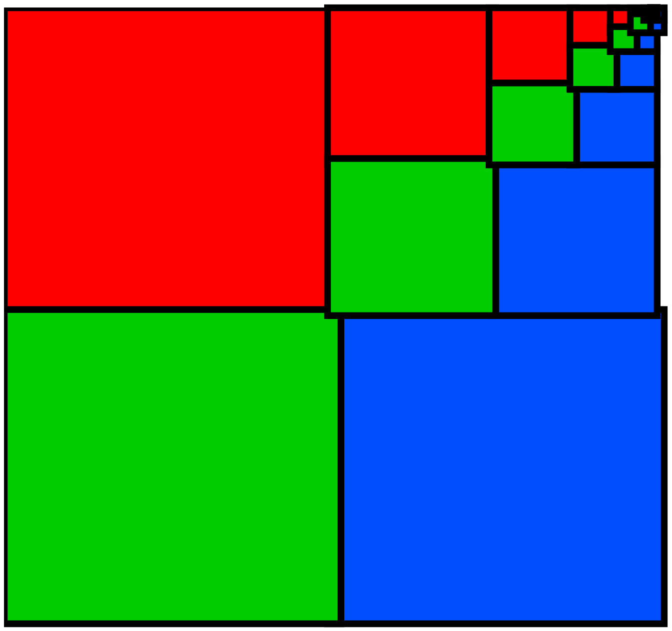
- MIP Mapping (“Multum In Parvo”)
  - Texture size is reduced by factors of 2 (*downsampling* = "many things in a small place")
  - Simple (4 pixel average) and memory efficient
  - Last image is only ONE texel





Texture Anti-Aliasing: MIP Mapping

- MIP Mapping ("Multum In Parvo")
  - Texture size is reduced by factors of 2 (*downsampling* = "many things in a small place")
  - Simple (4 pixel average) and memory efficient
  - Last image is only ONE texel



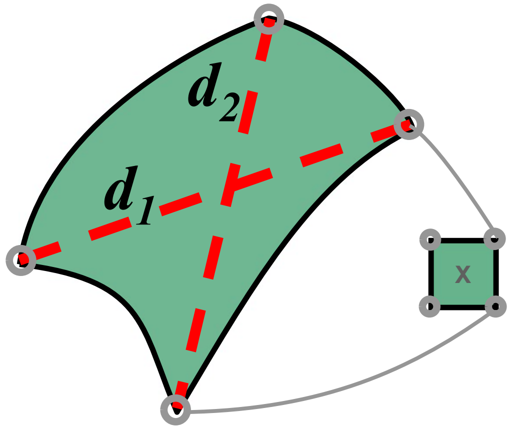
geometric series:

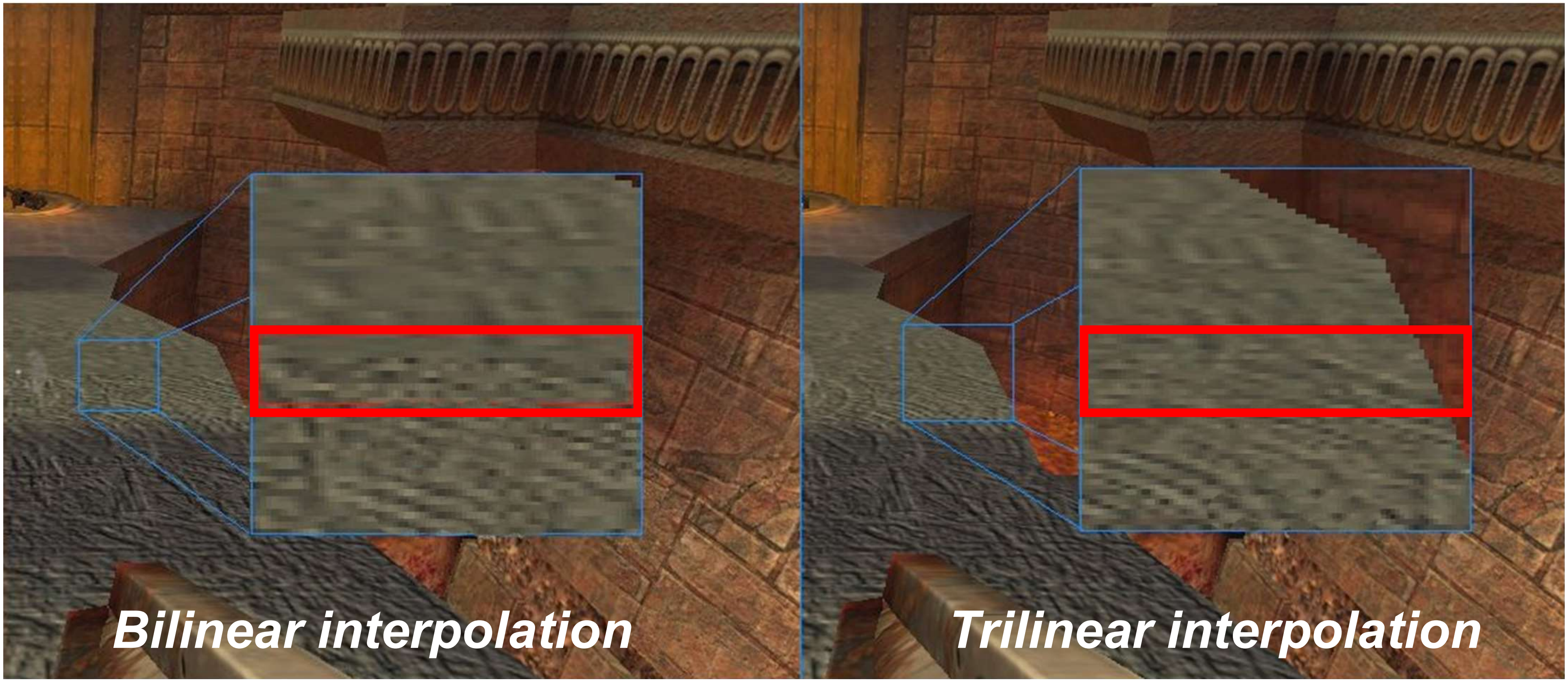
$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} =$$

$$= \sum_{k=0}^{n-1} ar^k = a\left(\frac{1-r^n}{1-r}\right)$$

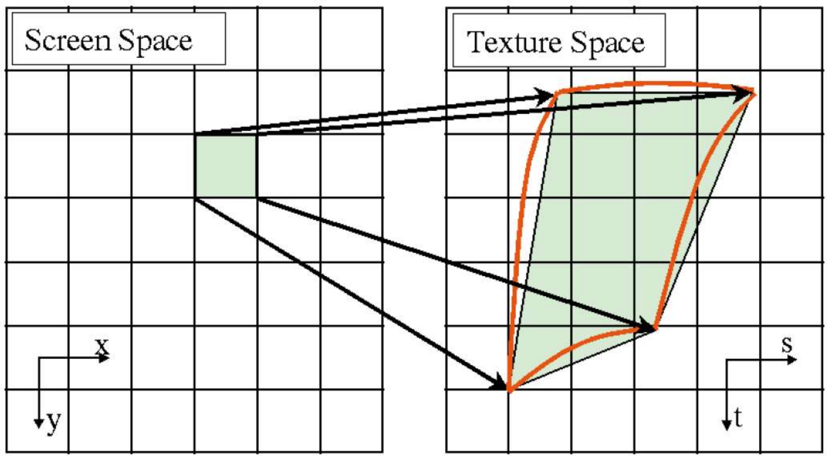
Texture Anti-Aliasing: MIP Mapping

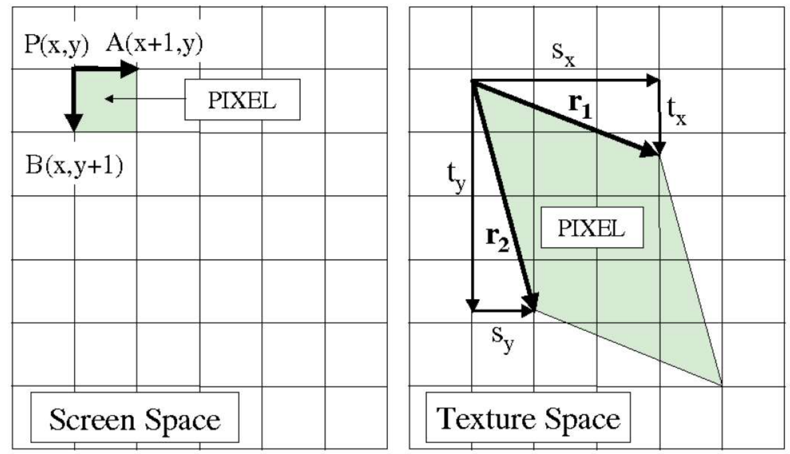
- MIP Mapping Algorithm
- $D := ld(max(d_1, d_2))$
- T_0 := value from texture $D_0 = trunc\ (D)$ "Mip Map level"
  - Use *bilinear interpolation*





MIP-Map Level Computation





- Use the partial derivatives of texture coordinates with respect to screen space coordinates
- This is the Jacobian matrix

$$\begin{pmatrix} \partial u/\partial x & \partial u/\partial y \\ \partial v/\partial x & \partial v/\partial y \end{pmatrix} = \begin{pmatrix} s_x & s_y \\ t_x & t_y \end{pmatrix}$$

- Area of parallelogram is the absolute value of the Jacobian determinant (the *Jacobian*)

MIP-Map Level Computation (OpenGL)

- OpenGL 4.6 core specification, pp. 251-264

(3D tex coords!)

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

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

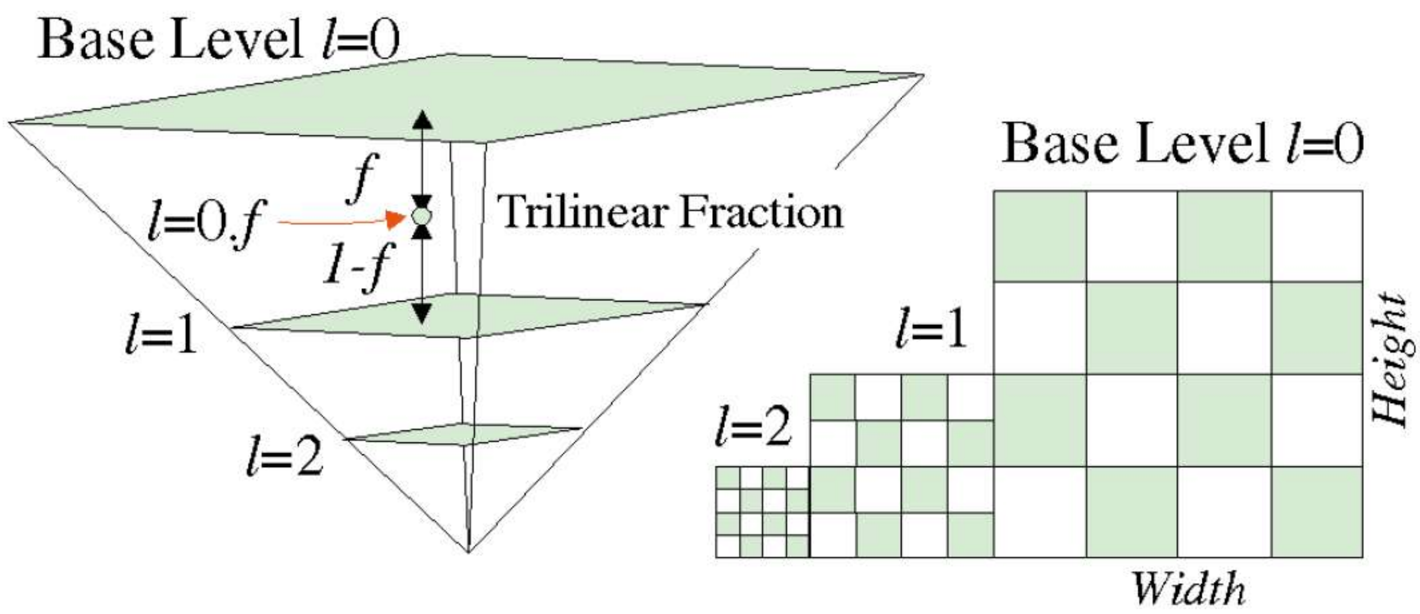
Does not use area of parallelogram but greater hypotenuse [Heckbert, 1983]

- Approximation without square-roots

$$m_u = \max\left\{\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|\right\} \quad m_v = \max\left\{\left|\frac{\partial v}{\partial x}\right|, \left|\frac{\partial v}{\partial y}\right|\right\} \quad m_w = \max\left\{\left|\frac{\partial w}{\partial x}\right|, \left|\frac{\partial w}{\partial y}\right|\right\}$$

$$\max\{m_u, m_v, m_w\} \le f(x,y) \le m_u + m_v + m_w$$

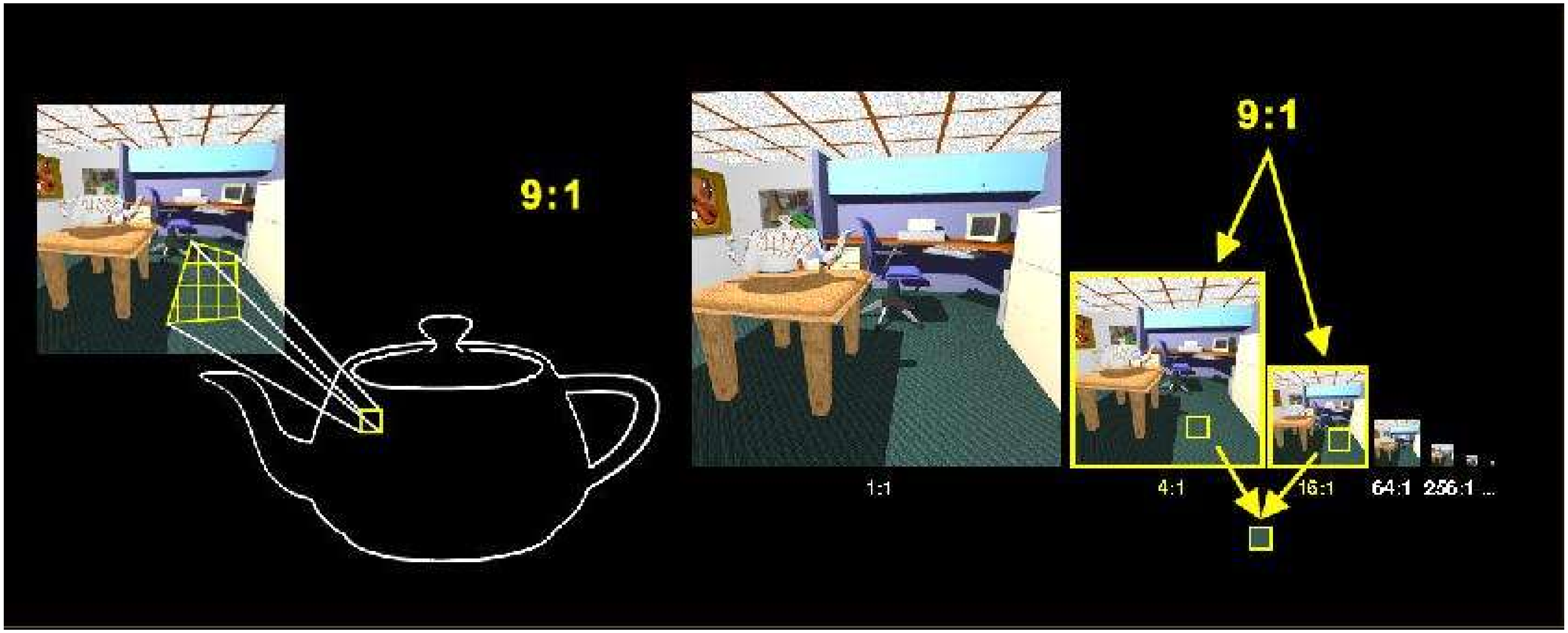
MIP-Map Level Interpolation



- Level of detail value is fractional!
- Use fractional part to blend (lin.) between two adjacent mipmap levels

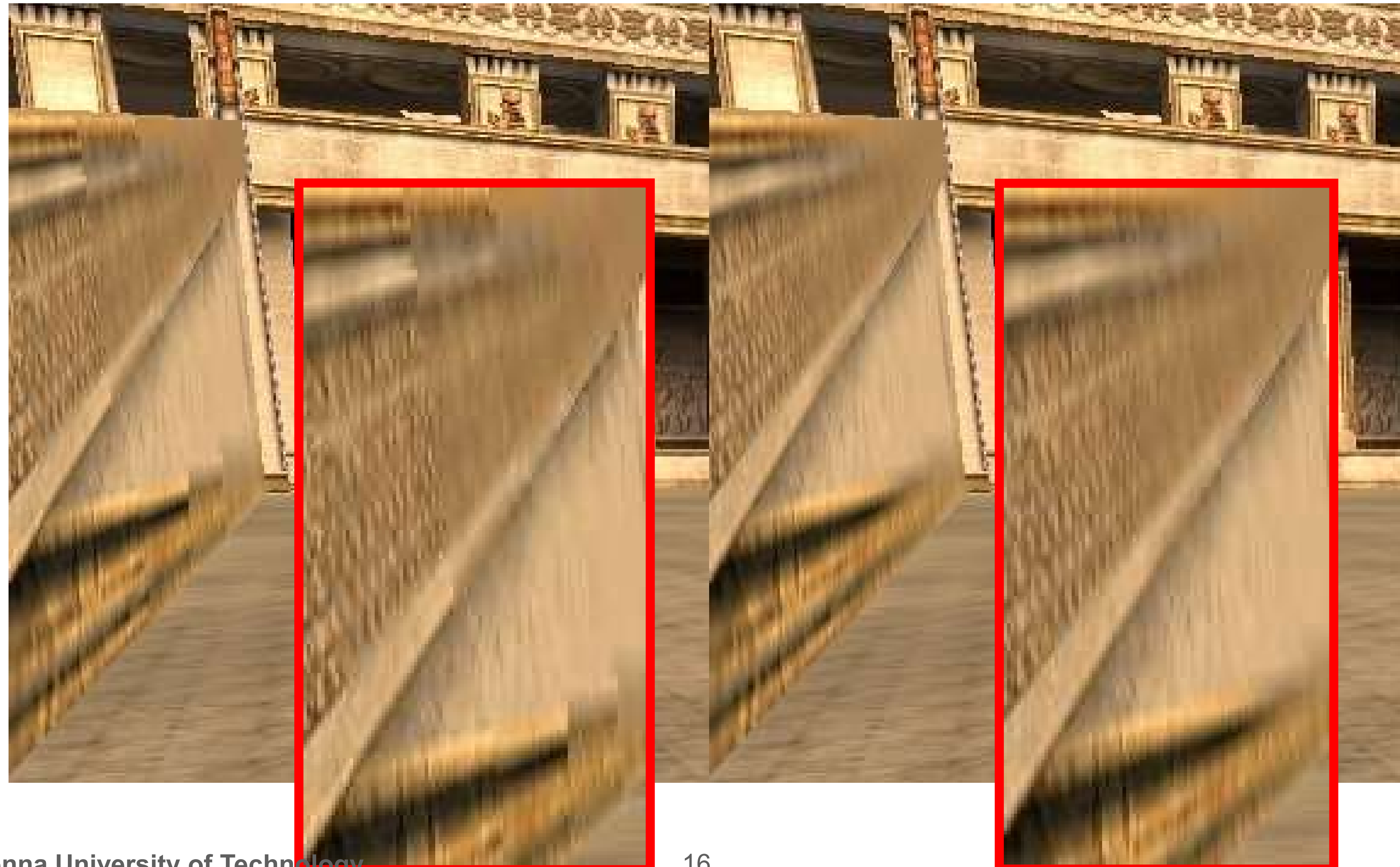
Texture Anti-Aliasing: MIP Mapping

- Trilinear interpolation:
  - T_1 := value from texture $D_1 = D_0+1$ (bilin.interpolation)
  - Pixel value := $(D_1 - D) \cdot T_0 + (D - D_0) \cdot T_1$
    - Linear interpolation between successive MIP Maps
  - Avoids "Mip banding" (but doubles texture lookups)

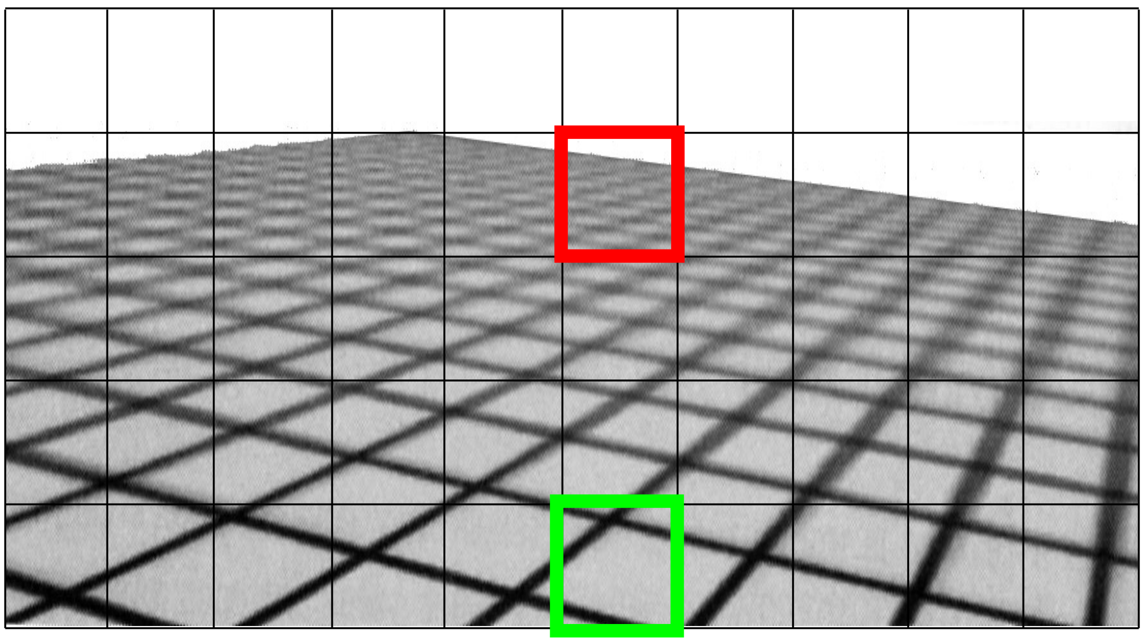


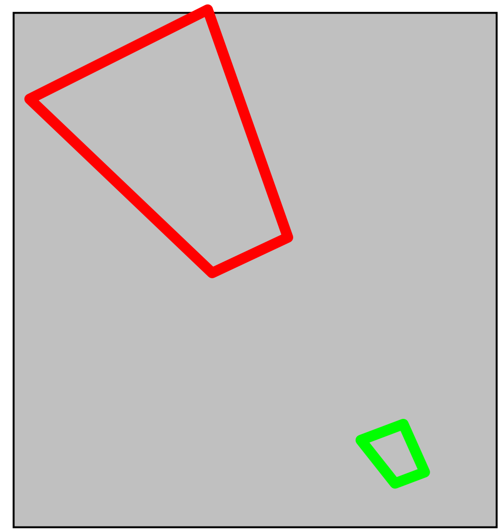
Texture Anti-Aliasing: MIP Mapping

- Other example for bilinear vs. trilinear filtering



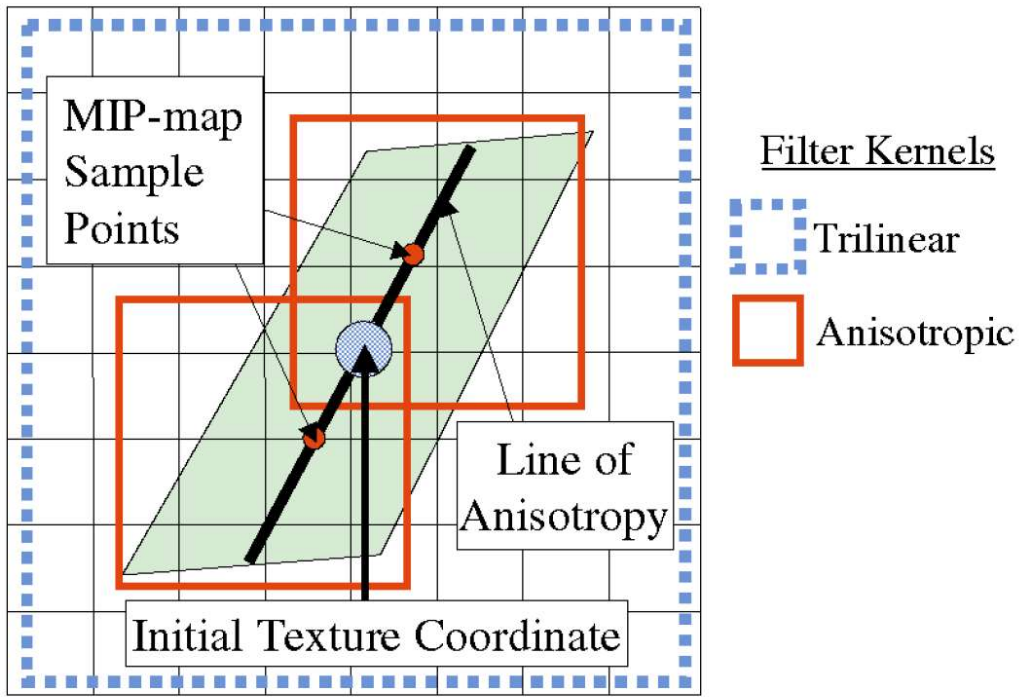
- Anisotropic filtering
  - View-dependent filter kernel
  - Implementation: *summed area table*, "*RIP Mapping*", *footprint assembly*, *elliptical weighted average* (EWA)





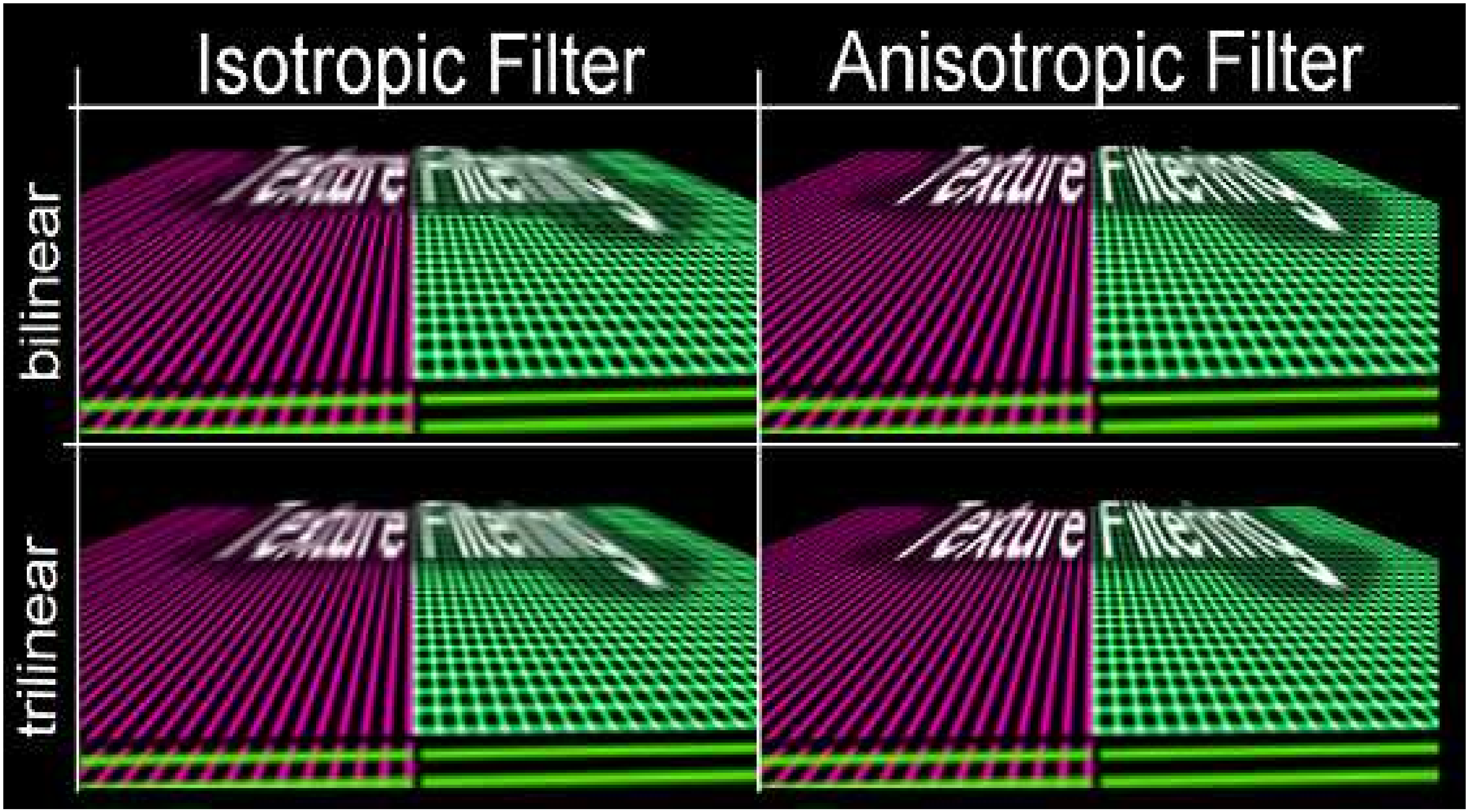
Texture space

Anisotropic Filtering: Footprint Assembly



Anti-Aliasing: Anisotropic Filtering

- Example



Texture Anti-aliasing

- Basically, everything done in hardware
- `gluBuild2DMipmaps()` generates MIPmaps
- Set parameters in `glTexParameter()`
  - `GL_TEXTURE_MAG_FILTER: GL_NEAREST, GL_LINEAR, …`
  - `GL_TEXTURE_MIN_FILTER: GL_LINEAR_MIPMAP_NEAREST`
- Anisotropic filtering is an extension:
  - `GL_EXT_texture_filter_anisotropic`
  - Number of samples can be varied (4x,8x,16x)
    - Vendor specific support and extensions

Virtual Textures

Virtual Texturing

Example #1:

ARB Sparse Textures (originally: AMD Partially Resident Textures)

ARB_sparse_texture / ARB_sparse_texture2

https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_sparse_texture.txt

https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_sparse_texture2.txt

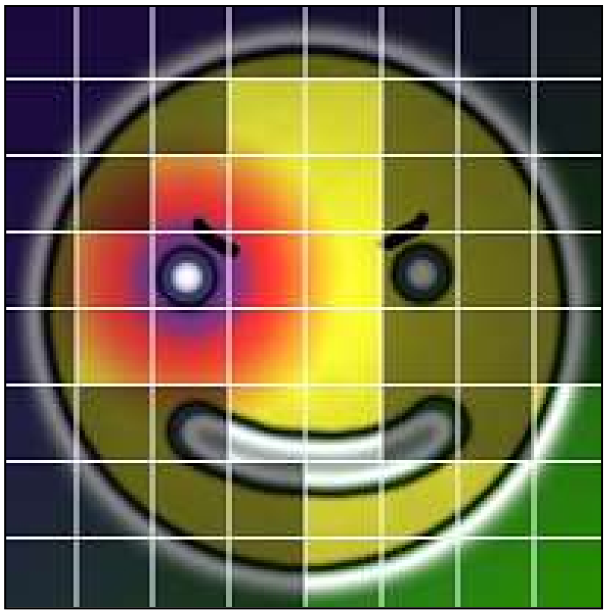
Hardware Virtual Texturing, Graham Sellers,
from SIGGRAPH 2013 course “Rendering Massive Virtual Worlds”

https://cesiumjs.org/hosted-apps/massiveworlds/downloads/Graham/Hardware_Virtual_Textures.pptx

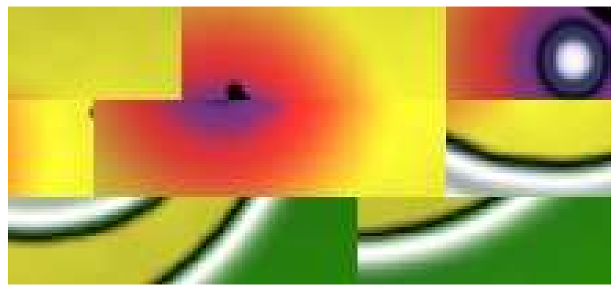
Virtual Texturing

Divide texture up into tiles

- Commit only *used* tiles to memory
- Store data in separate physical texture



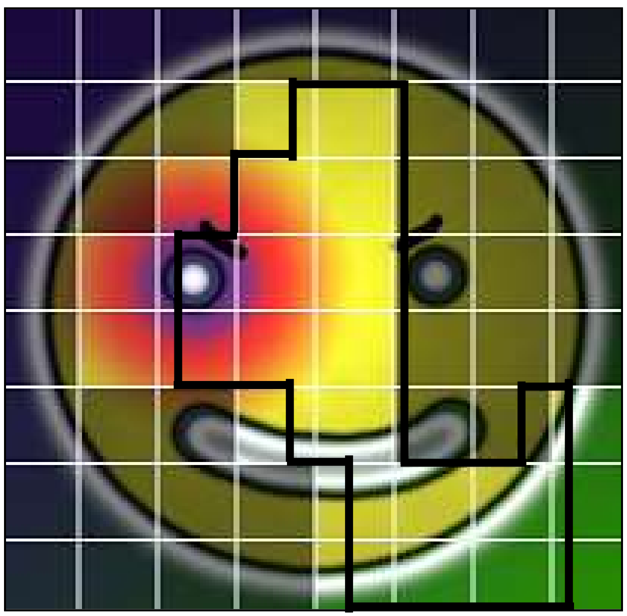
Virtual Texture



Physical Texture

Virtual Texturing

Memory requirements set by number of resident tiles, not texture dimensions



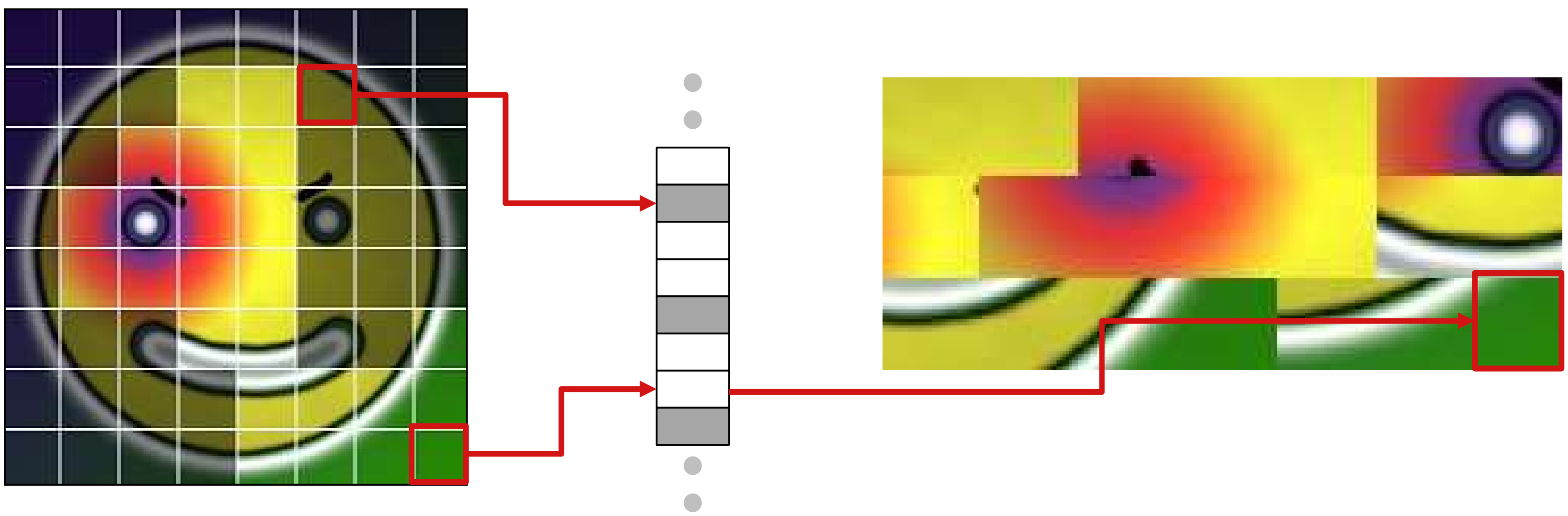
RGBA8, 1024x1024, 64 tiles

	Virtual	Physical
Memory	4096 kB	1536 kB

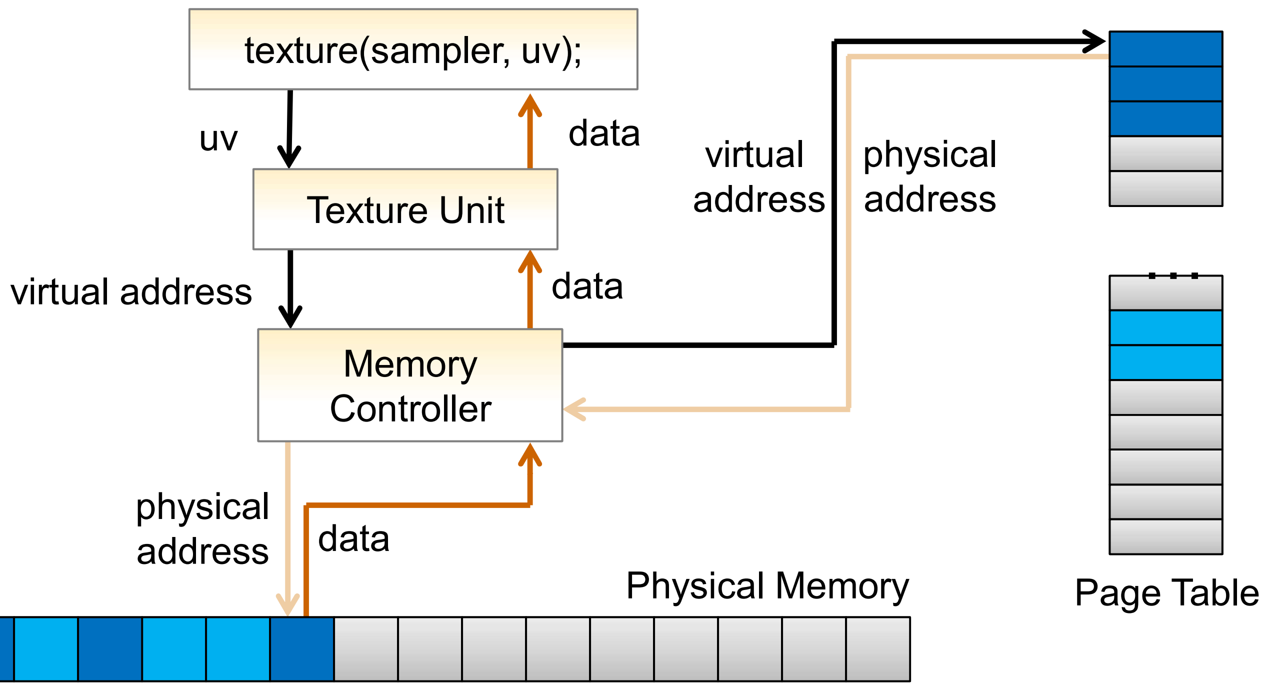
Virtual Texturing

Use indirection table to map virtual to physical

- This is also known as a *page table*



GPU Virtual Memory



Summary (Shader vs. Full Hardware Support)

	SVTs	HVTs
Address translation	Shader code	HW page table
Filtering	HW + shader code	HW only
# of texture fetches	2, dependent	1
Supported formats	The ones implemented	All supported by HW
Supported texture types	The ones implemented	All supported by HW

Virtual Texturing

Example #2:

Adaptive Shadow Maps (ASM)

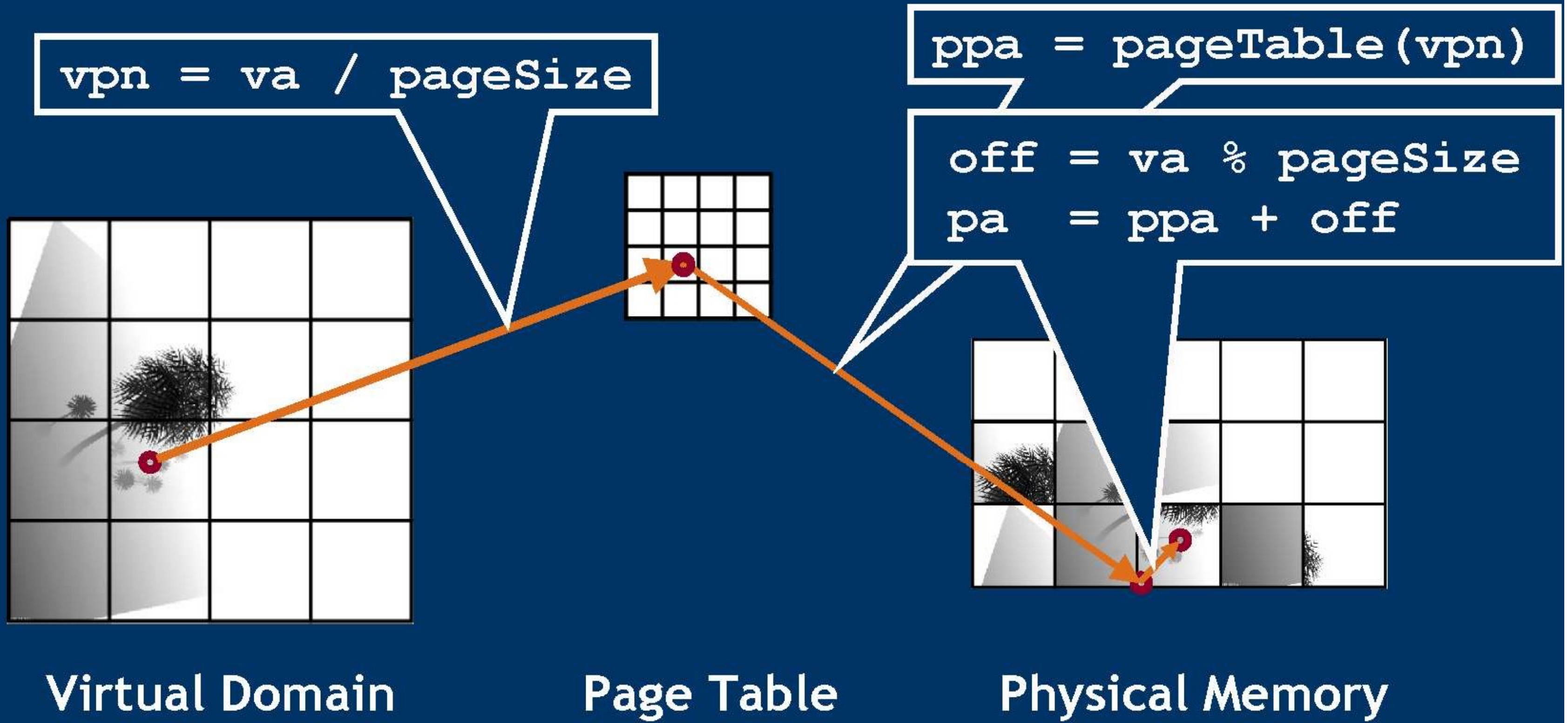
- On CPUs: Fernando et al., ACM SIGGRAPH 2001

Resolution-Matched Shadow Maps

- On GPUs: Aaron Lefohn et al., ACM Transactions on Graphics 2007

ASM Data Structure (Adaptive Shadow Maps)

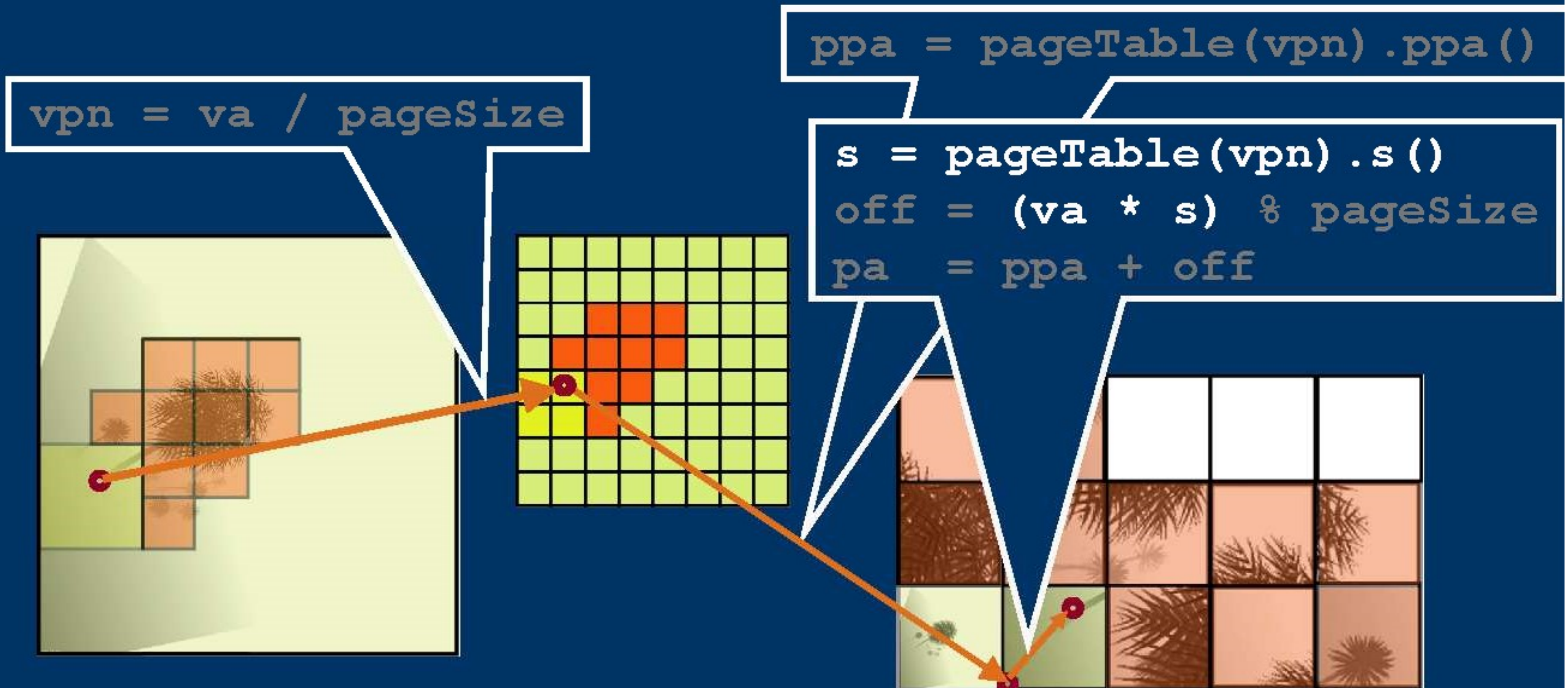
- Page table example



Aaron Lefohn
University of California, Davis

ASM Data Structure (Adaptive Shadow Maps)

- Adaptive Page Table
  - Map multiple virtual pages to single physical page



Virtual Domain Page Table Physical Memory

Aaron Lefohn
University of California, Davis

Virtual Texturing

Example #3:

id Tech 5 Megatextures, id Software

Rage

- Virtual Texturing in Software and Hardware, van Waveren et al., SIGGRAPH 2012 course notes + slides

`http://www.jurajobert.com/data/Virtual_Texturing_in_Software_and_Hardware_course_notes.pdf`

`http://www.mrelusive.com/publications/papers/Software-Virtual-Textures.pdf`

`http://www.mrelusive.com/publications/presentations/2013_siggraph/hq_sw_hw_vts_12.pdf`

Virtual Texturing

Rage / id Tech 5 (id Software)

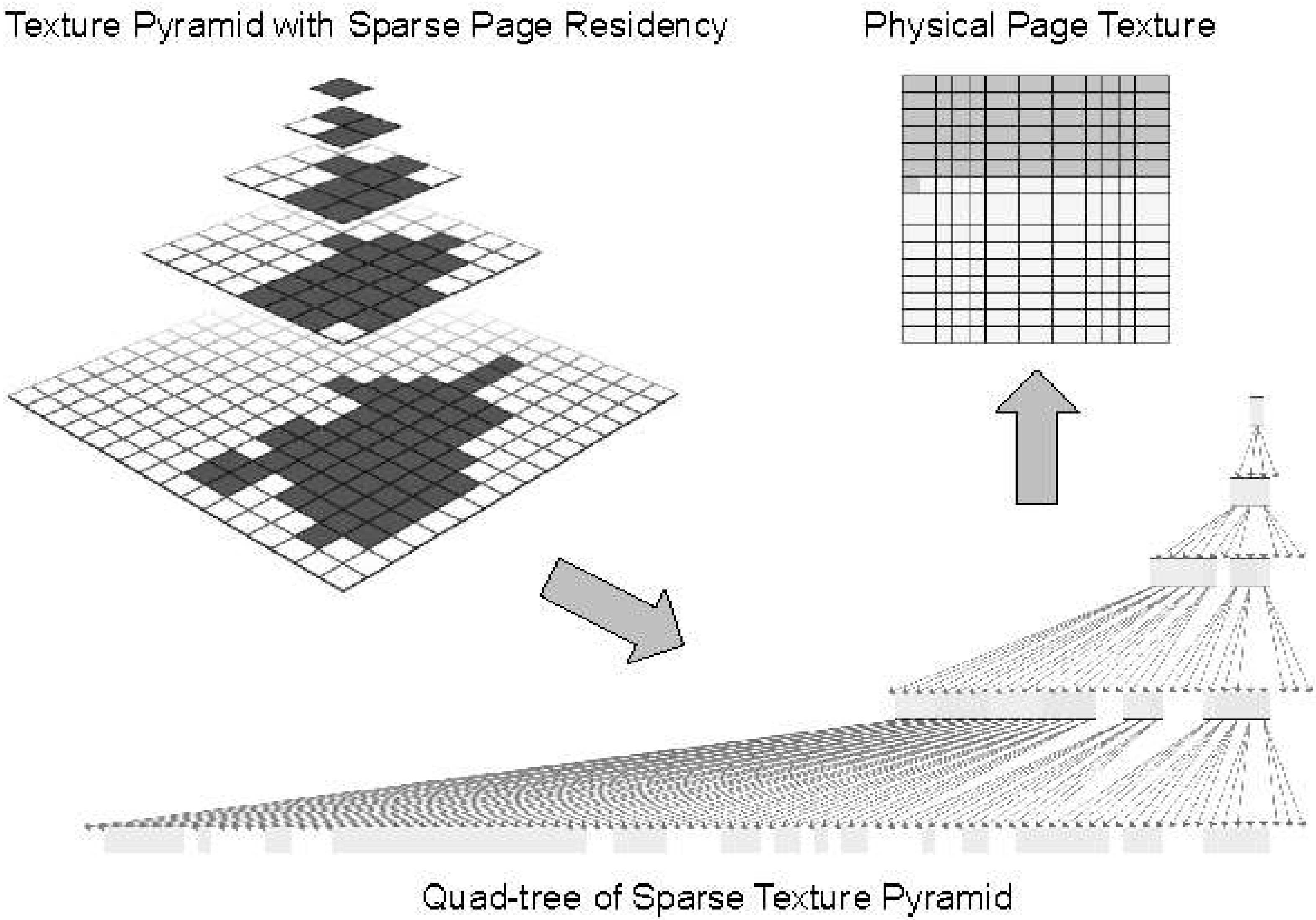
Virtual Texturing

- Unique, very large virtual textures key to id tech 5 rendering
- Full description beyond the scope of this talk

Virtual Texturing

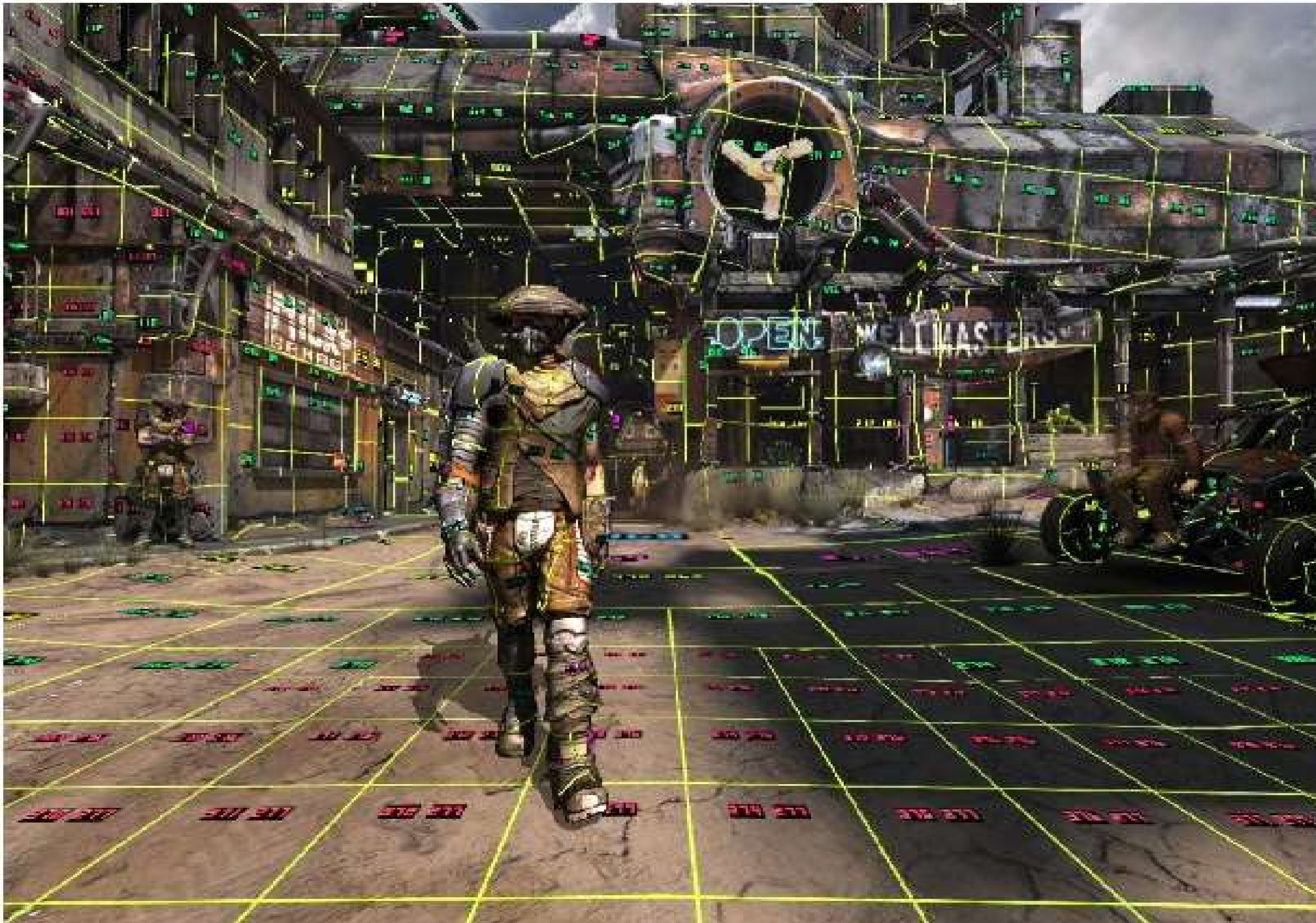
Virtual Texturing

Virtual Texturing



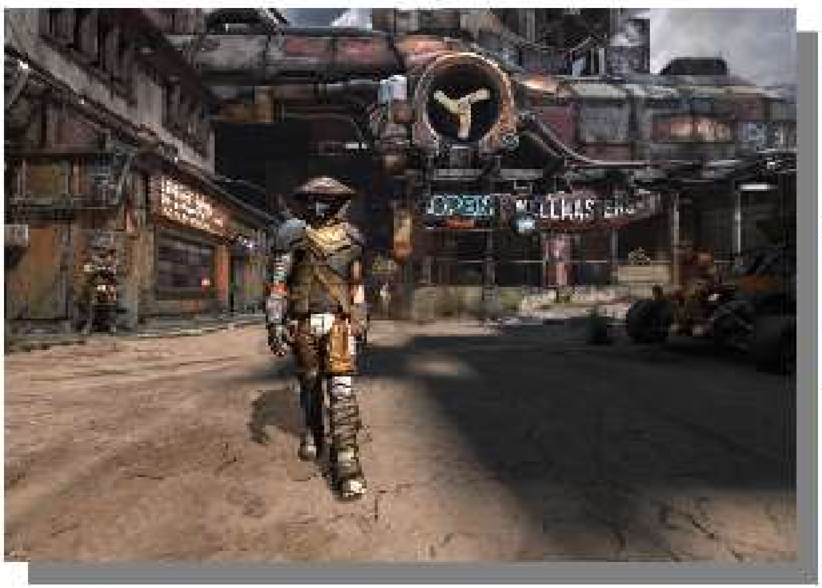
Virtual Texturing

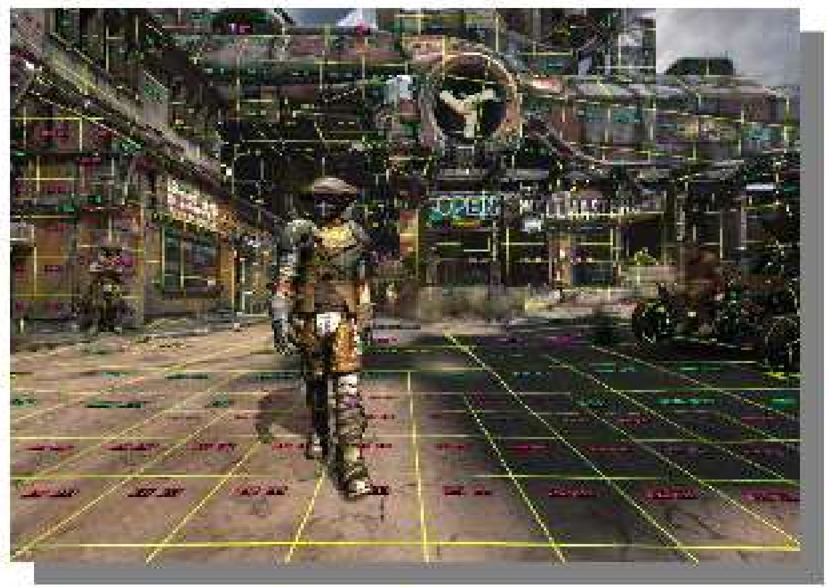
Virtual Texturing



Virtual Texturing

A few interesting issues...
- Texture filtering
- Thrashing due to physical memory oversubscription
- LOD transitions under high latency

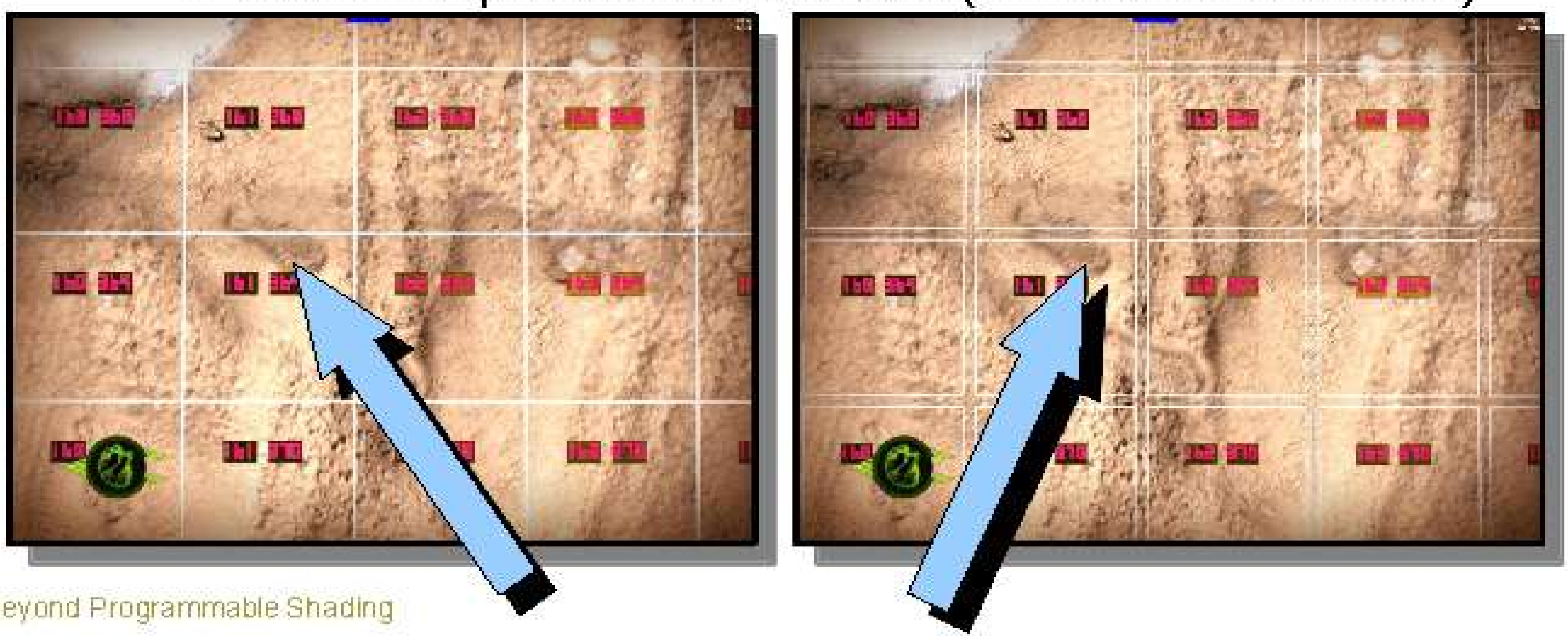




RAGE with PRTs (Image courtesy of id Software)

Virtual Texturing - Filtering

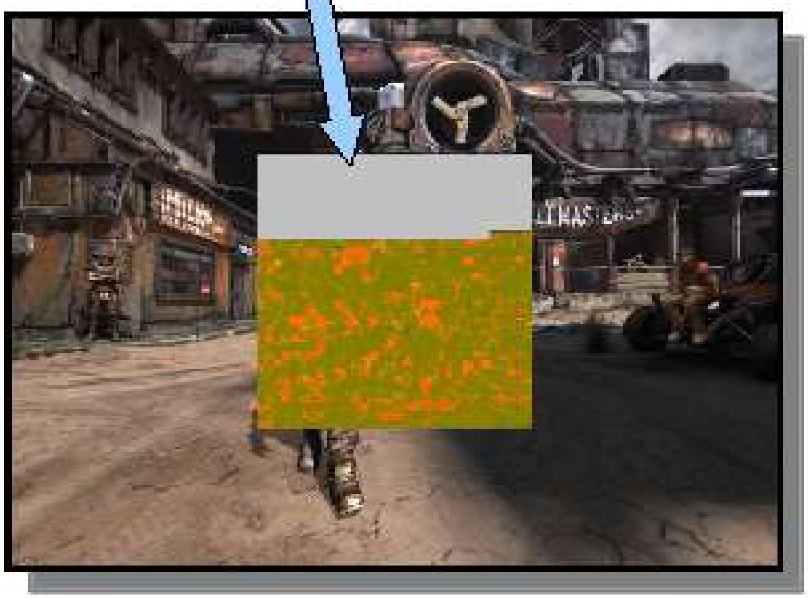
- We tried no filtering at all
- We tried bilinear filtering without borders
- Bilinear filtering with border works well
- Trilinear filtering reasonably but still expensive
- Anisotropic filtering possible via TXD (texgrad)
  - 4-texel border necessary (max aniso = 4)
  - TEX with implicit derivs ok too (on some hardware)



Virtual Texturing - Thrashing

- Sometimes you need more physical pages than you have
- With conventional virtual memory, you must thrash
- With virtual texturing, you can globally adjust feedback LOD bias until working set fits

32 x 32 pages



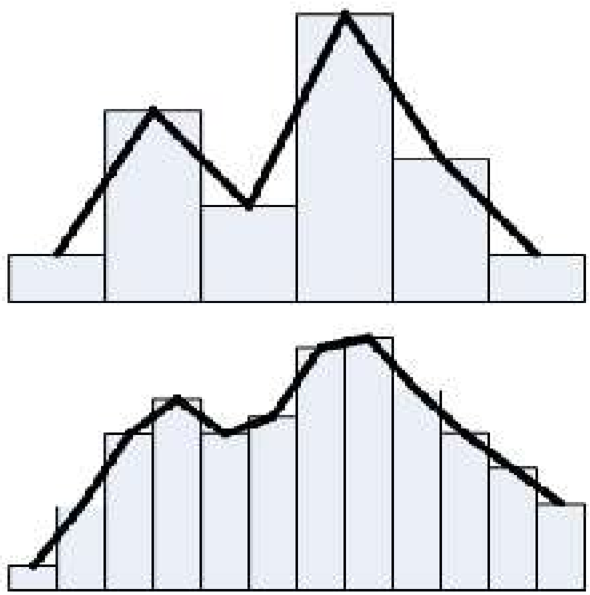
1024 Physical Pages

8x8 pages

64 Physical Pages

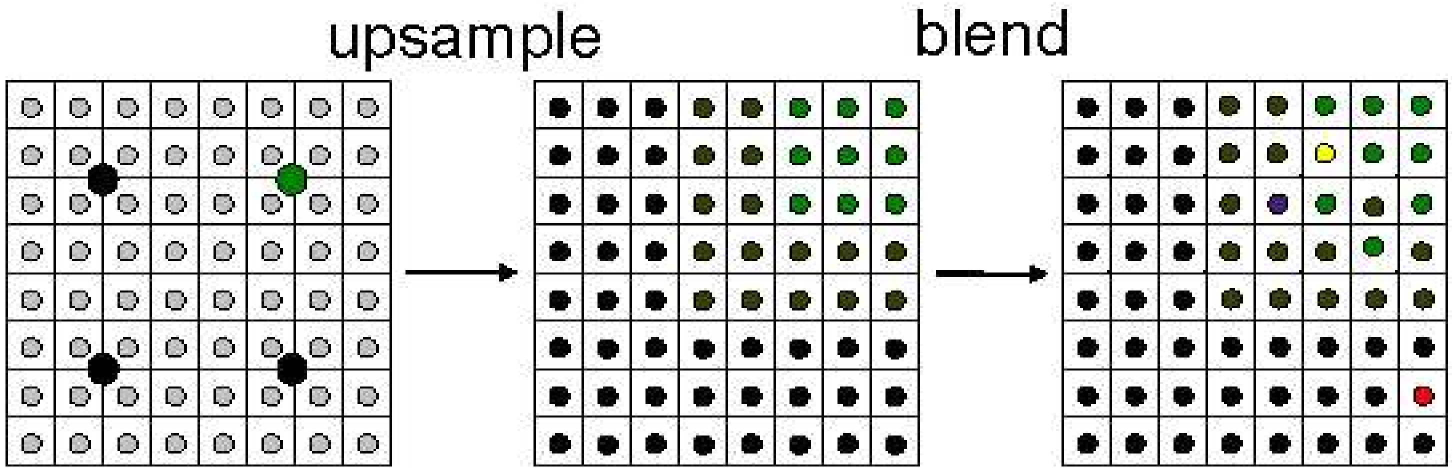
Virtual Texturing – LOD Snap

- Latency between first need and availability can be high
  - Especially if optical disk read required (>100 msec seek!)
- Visible snap happens when magnified texture changes LOD
- If we used trilinear filtering, blending in detail would be easy
- Instead continuously update physical pages with blended data



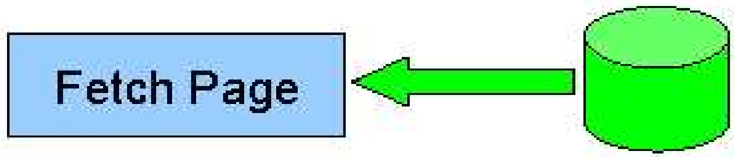
Virtual Texturing – LOD Snap

- Upsample coarse page immediately
- Then blend in finer data when available



Virtual Texturing - Management

- Analysis tells us what pages we need
- We fetch what we can



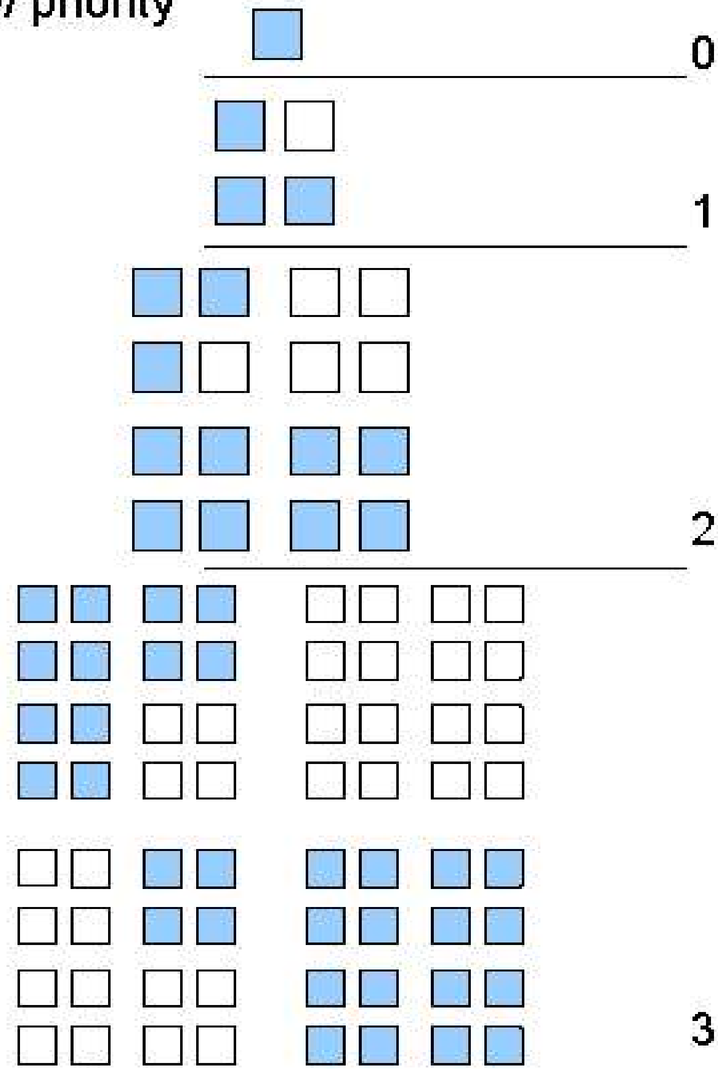
- But this is a real-time app... so no blocking allowed
- Cache handles hits, schedules misses to load in background
- Resident pages managed independent of disk cache
- Physical pages organized as quad-tree per virtual texture
- Linked lists for free, LRU, and locked pages

Virtual Texturing - Feedback

- Feedback Analysis
  - Gen ~breadth-first quad-tree order w/ priority

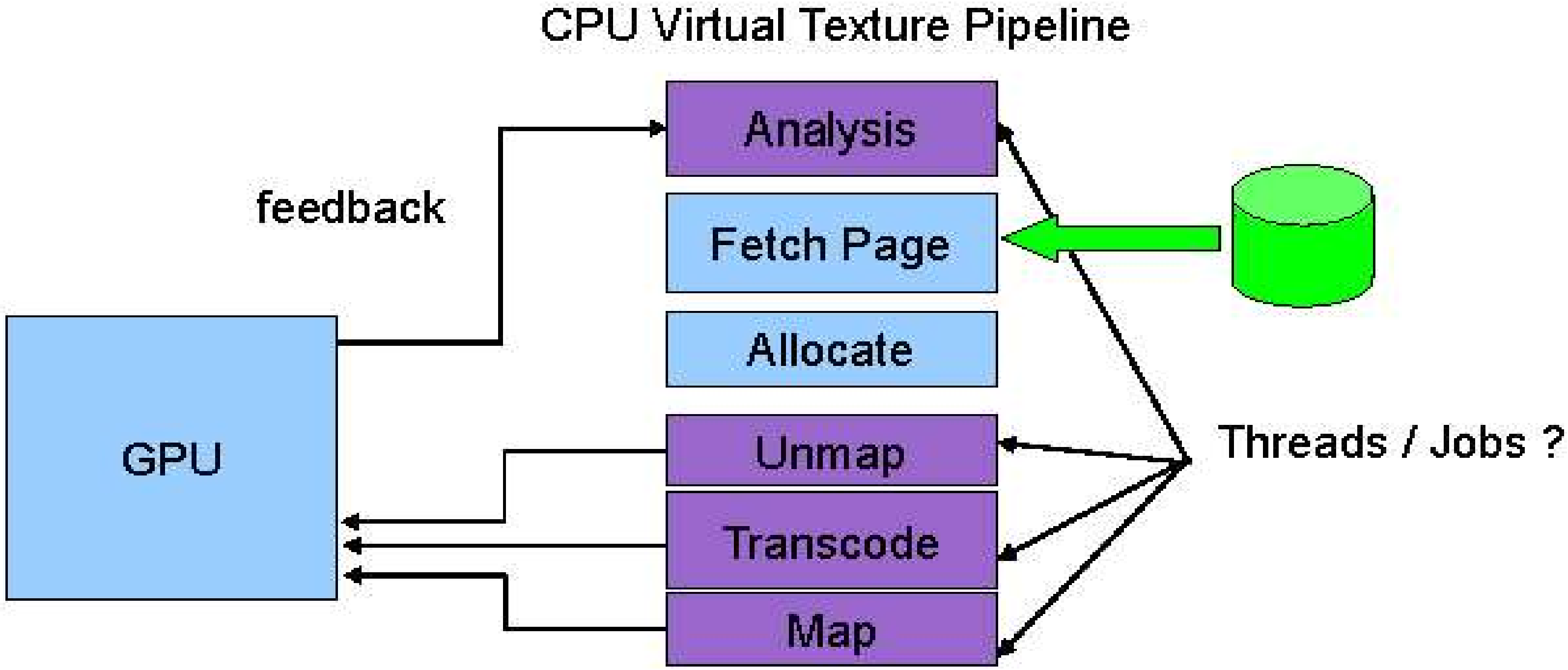
Color Buffer

Feedback Buffer



Virtual Texturing - Pipeline

- Compute intensive complex system with dependencies that we want to run in parallel on all the different platforms



Virtual Texturing

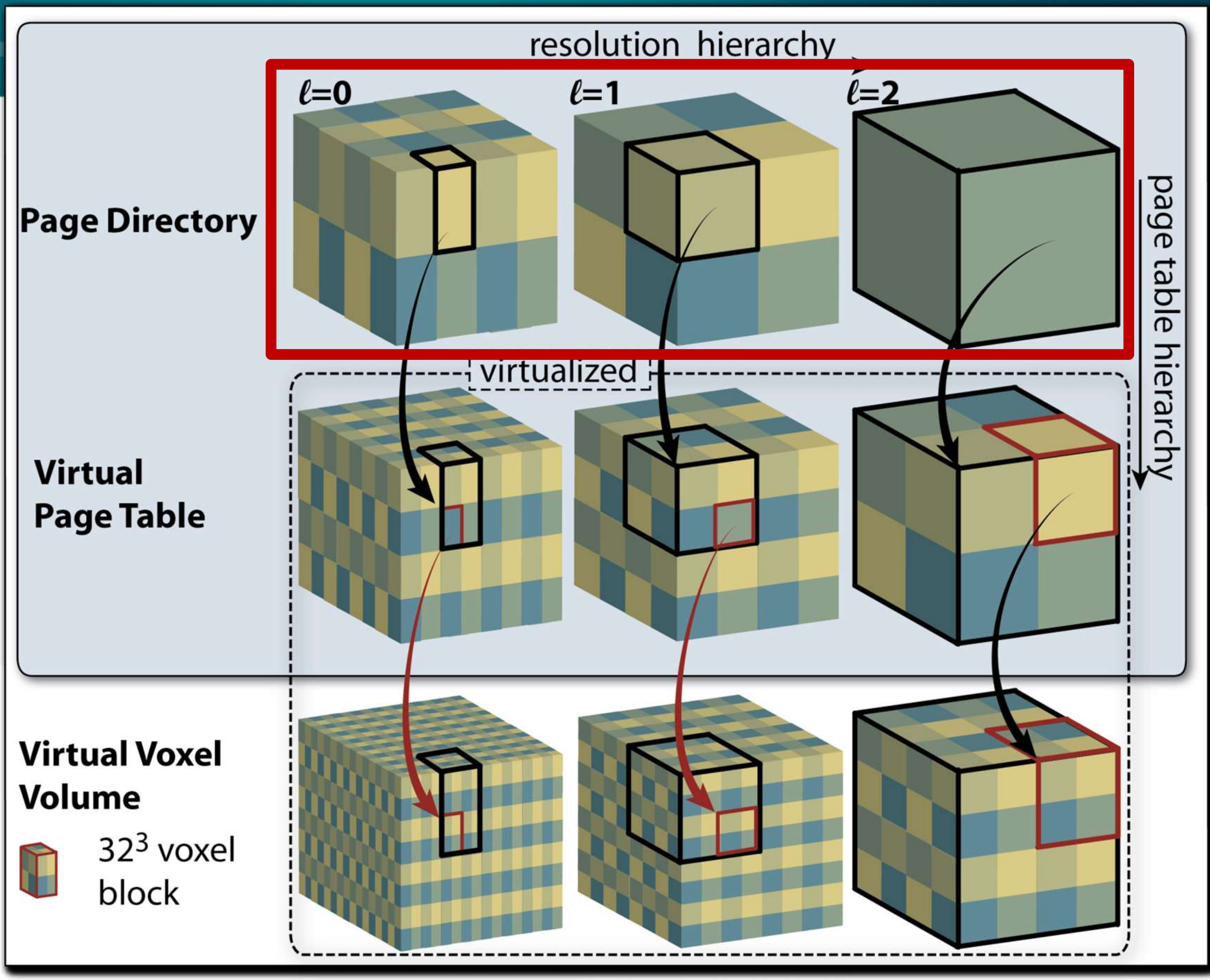
Example #4:

Petascale Volume Rendering

- Interactive Volume Exploration of Petascale Microscopy Data Streams Using a Visualization-Driven Virtual Memory Approach,
  Hadwiger et al., IEEE SciVis 2012

http://dx.doi.org/10.1109/TVCG.2012.240

Petascale Volume Rendering



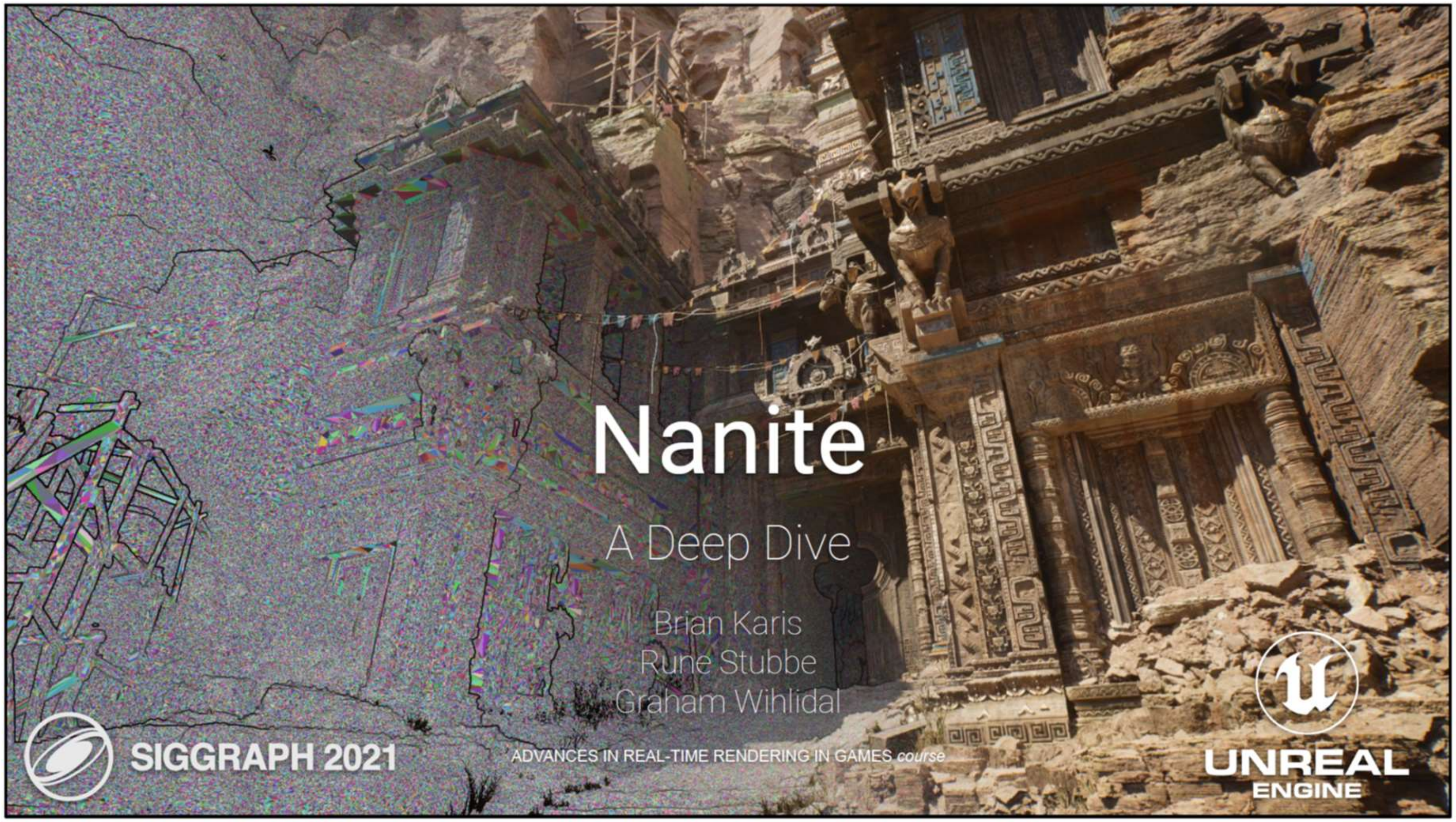
multi-resolution page directory

Virtual Geometry

Virtual Geometry

Unreal Engine 5 virtual geometry pipeline: Nanite

Tech talk by Brian Karis, Epic Games



The Dream

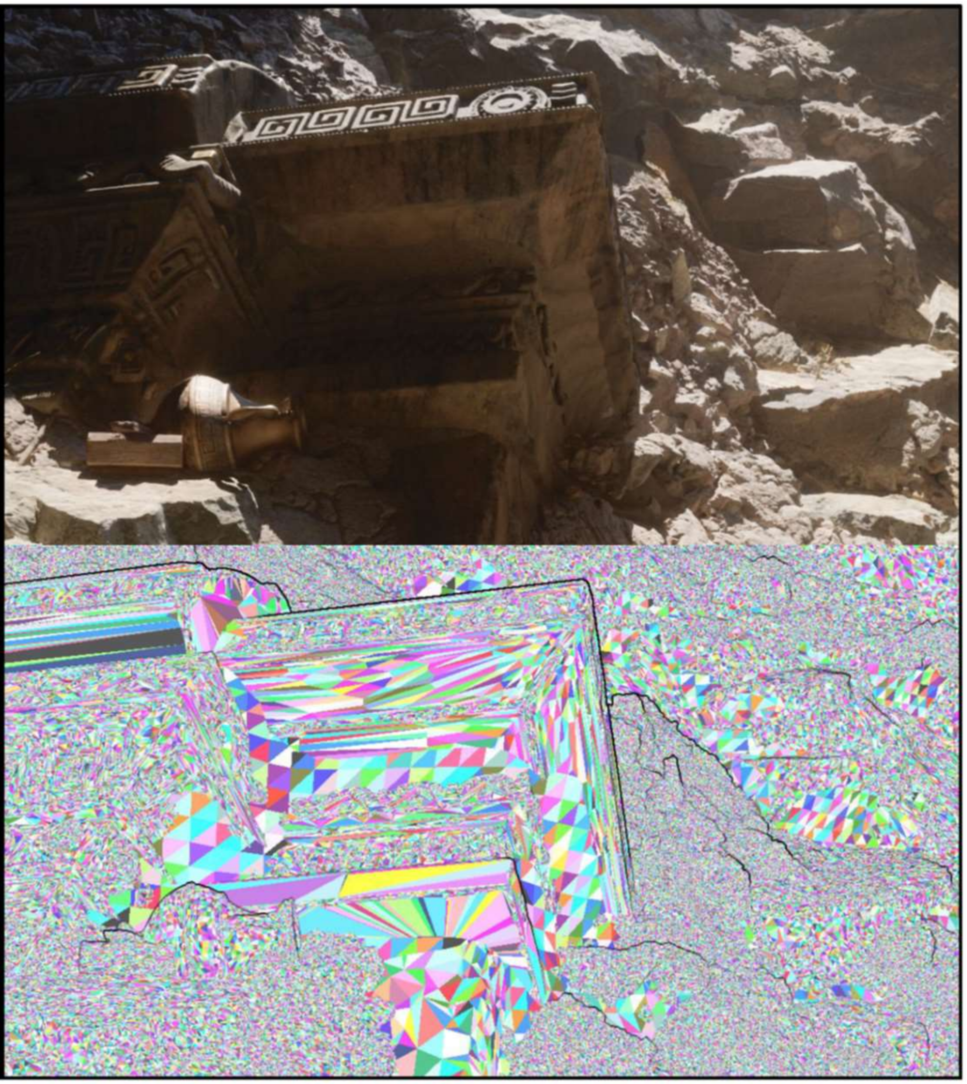
- Virtualize geometry like we did textures
- No more budgets
  - Polycount
  - Draw calls
  - Memory
- Directly use film quality source art
  - No manual optimization required
- No loss in quality

Virtual Geometry

Virtual Geometry

Pixel scale detail

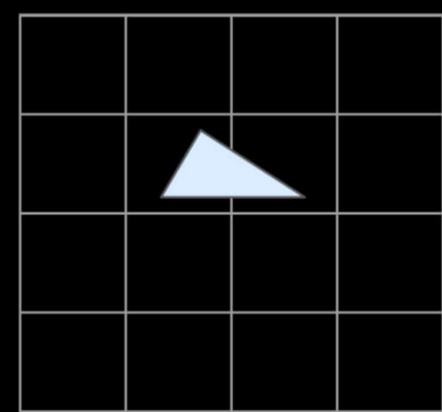
- Can we hit pixel scale detail with triangles > 1 pixel?
  - Depends how smooth
  - In general no
- We need to draw pixel sized triangles



SIGGRAPH 2021

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?

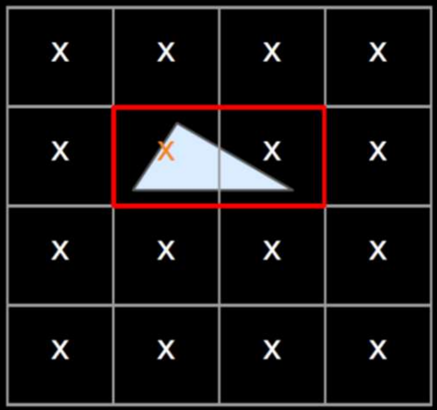


Software Rasterization

3x faster!

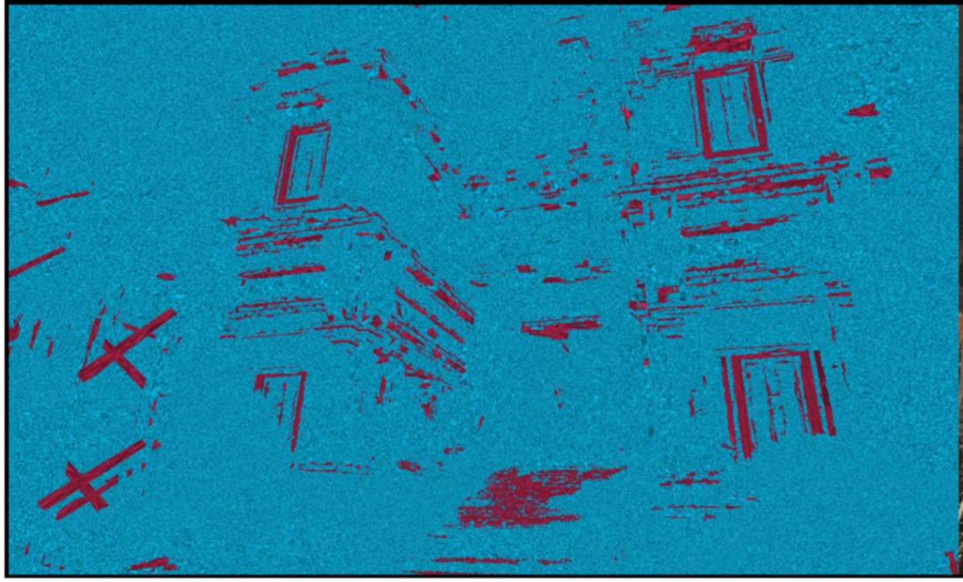
Micropoly software rasterizer

- 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



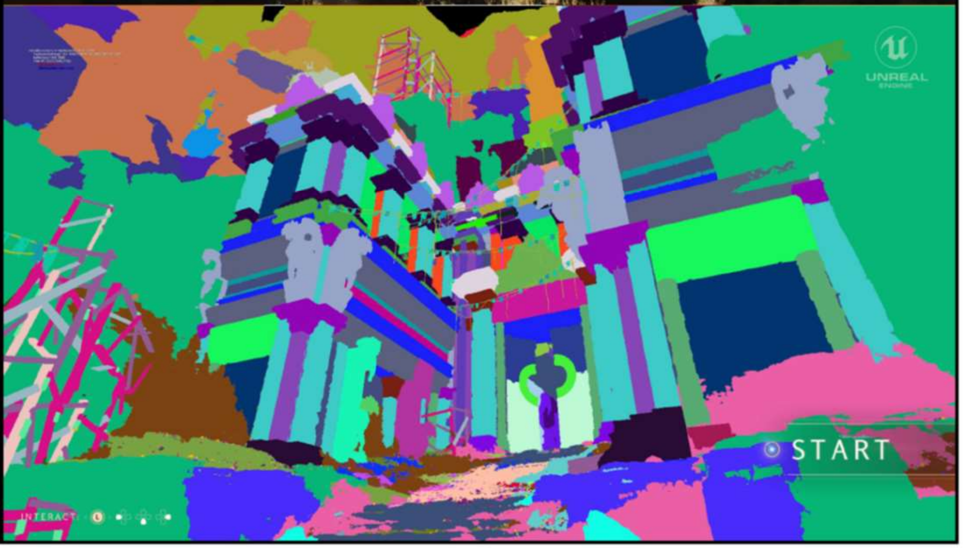
Hardware Rasterization

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



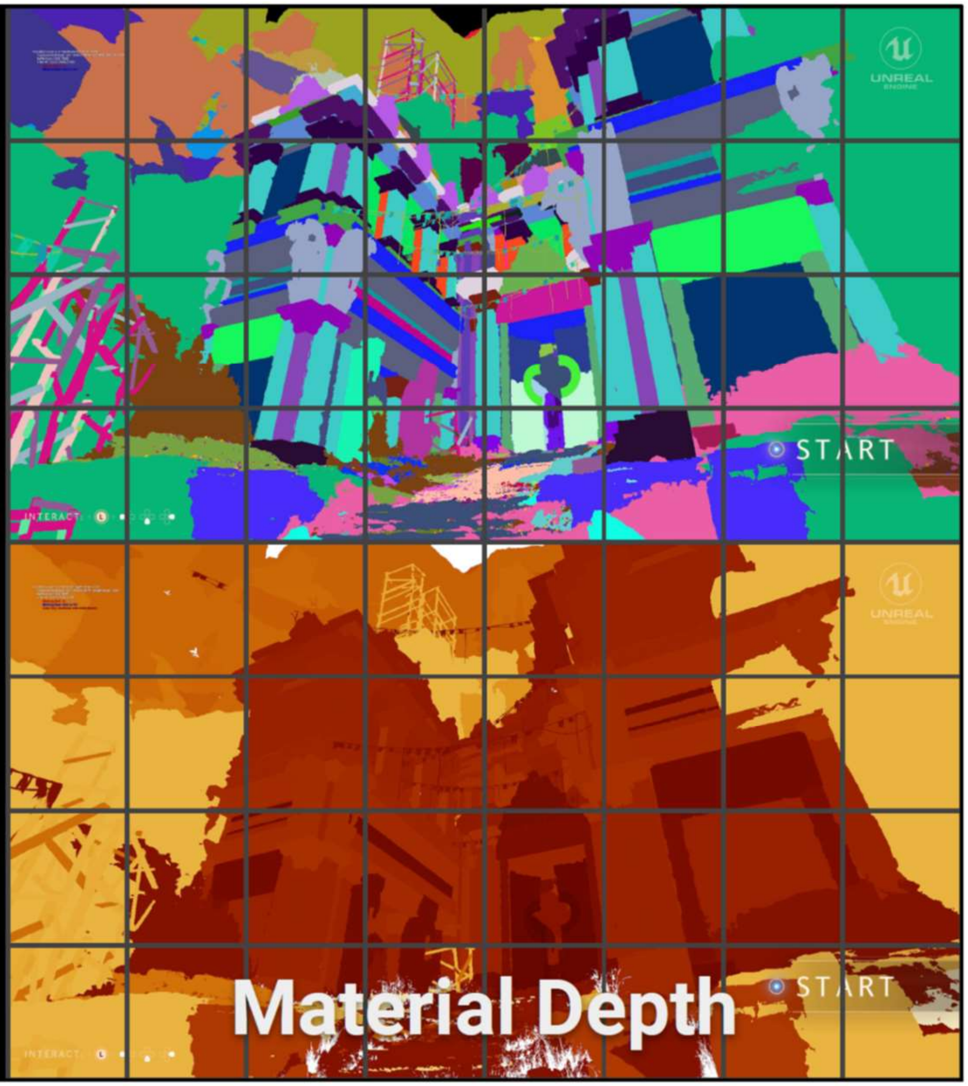
Material shading

- 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

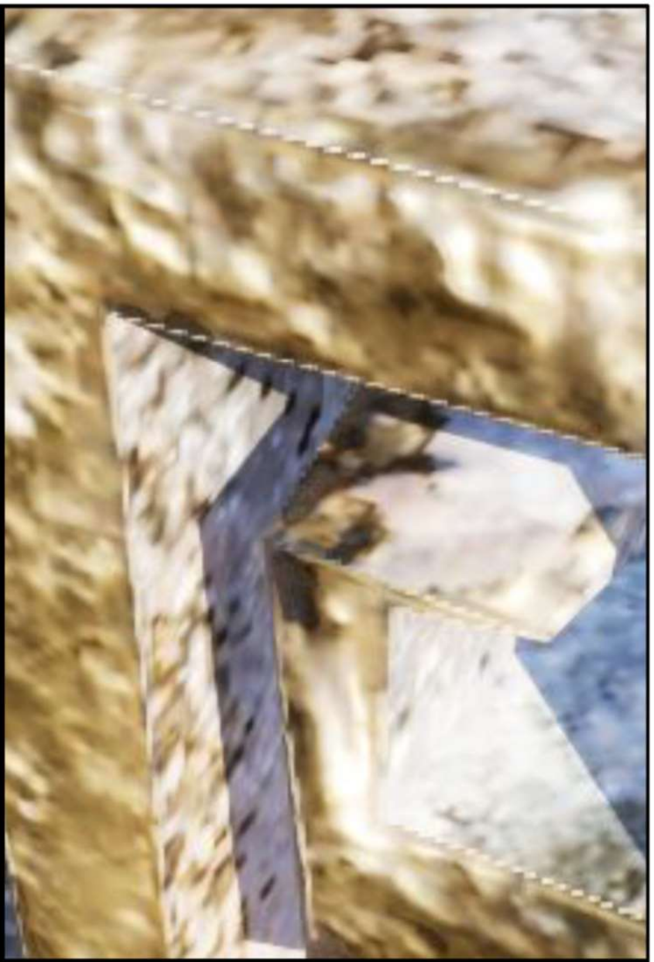
- Material covers small portion of the screen
  - HiZ handles this OK
  - We can do better
- Coarse tile classification / culling
  - Render 8x4 grid of tiles per material
  - Same shading approach as full screen quads
- Tile killed in vertex shader from 32b mask
  - X=NaN



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

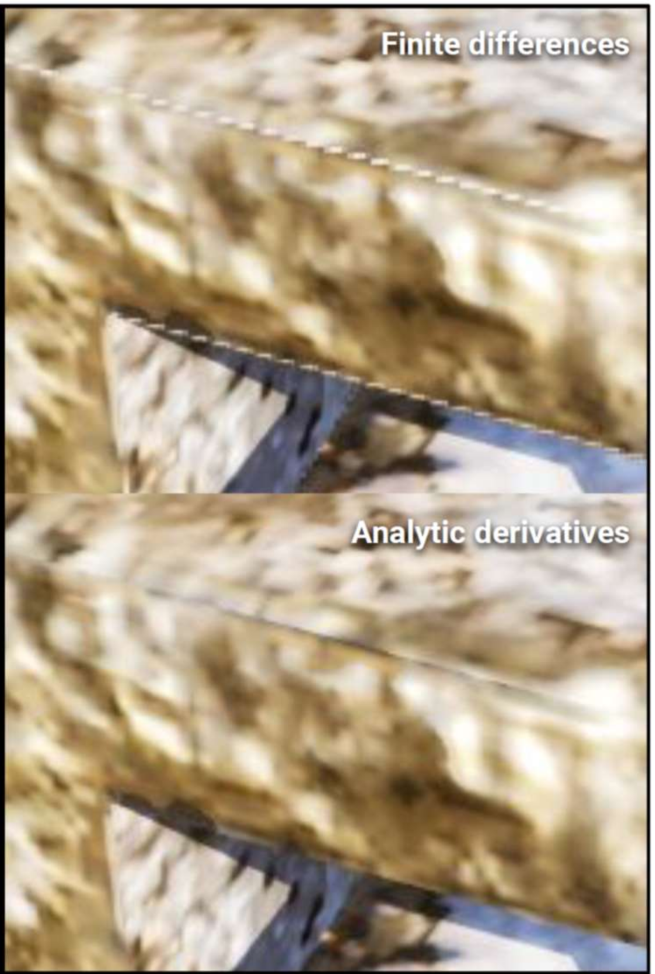
  Not good!



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

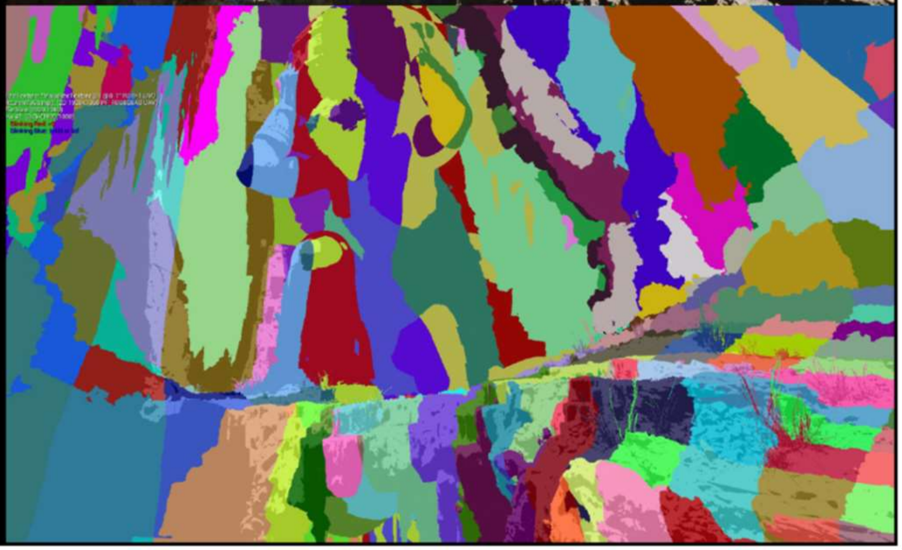
Total rasterized

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



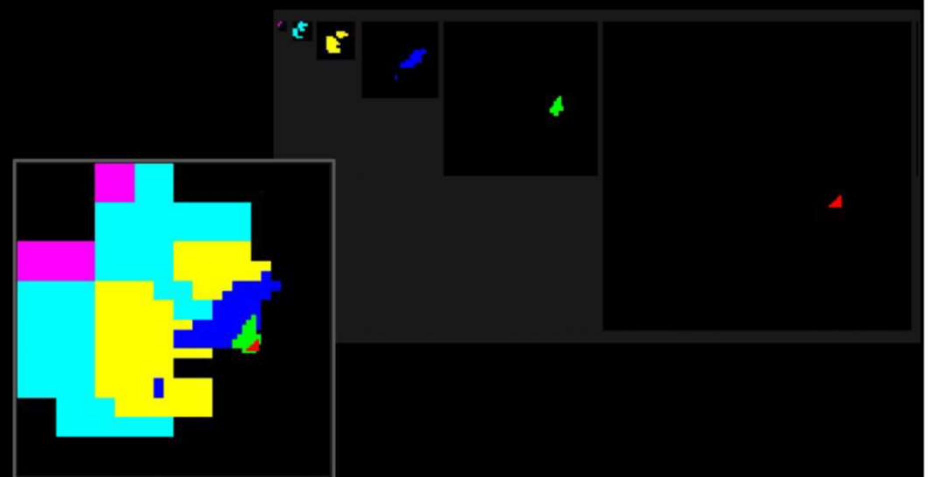
Virtual shadow maps

- 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.