King Abdullah University of Science and Technology

KAUST

# CS 380 - GPU and GPGPU Programming
# Lecture 24: GPU Virtual Texturing

Markus Hadwiger, KAUST

# Reading Assignment #13 (until Dec 1)

Read (required):

- Look at Unreal Engine 5 virtual geometry system (Nanite)
  - **https://dev.epicgames.com/documentation/en-us/unreal-engine/nanite-virtualized-geometry-in-unreal-engine/**
- Look at Unreal Engine 5 Nanite tech talk by Brian Karis *(Journey to Nanite)* slides
  - **https://www.highperformancegraphics.org/slides22/Journey_to_Nanite.pdf**
- Look at Unreal Engine 5 Lumen tech talk by Daniel Wright et al. slides
  - **https://advances.realtimerendering.com/s2022/SIGGRAPH2022-Advances-Lumen-Wright%20et%20al.pdf**

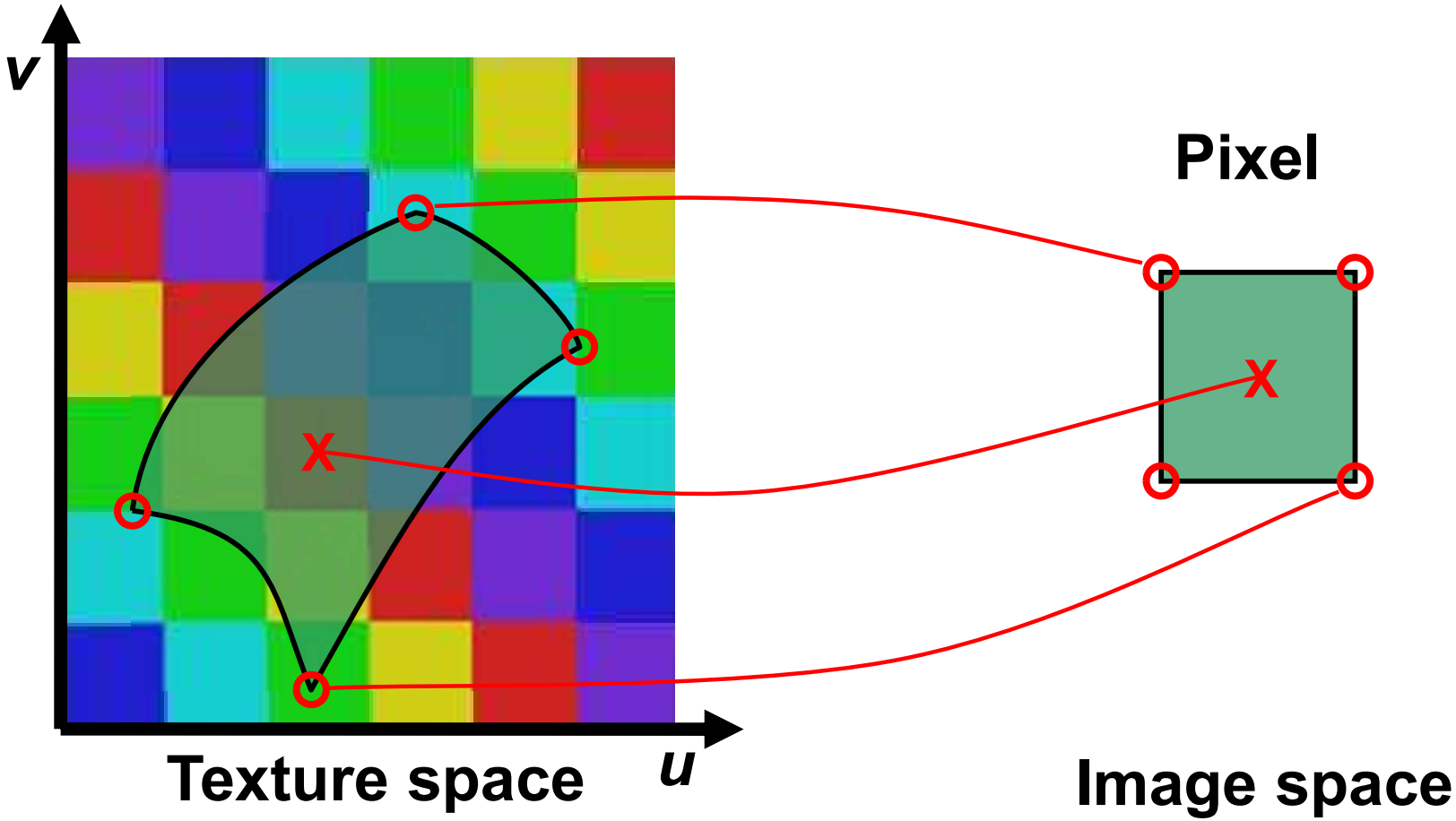
Read (optional):

- Look at Unreal Engine 5 Nanite tech talk by Brian Karis *(Journey to Nanite)* talk
  - **https://www.youtube.com/watch?v=NRnj_lnpORU**

# Texture Minification

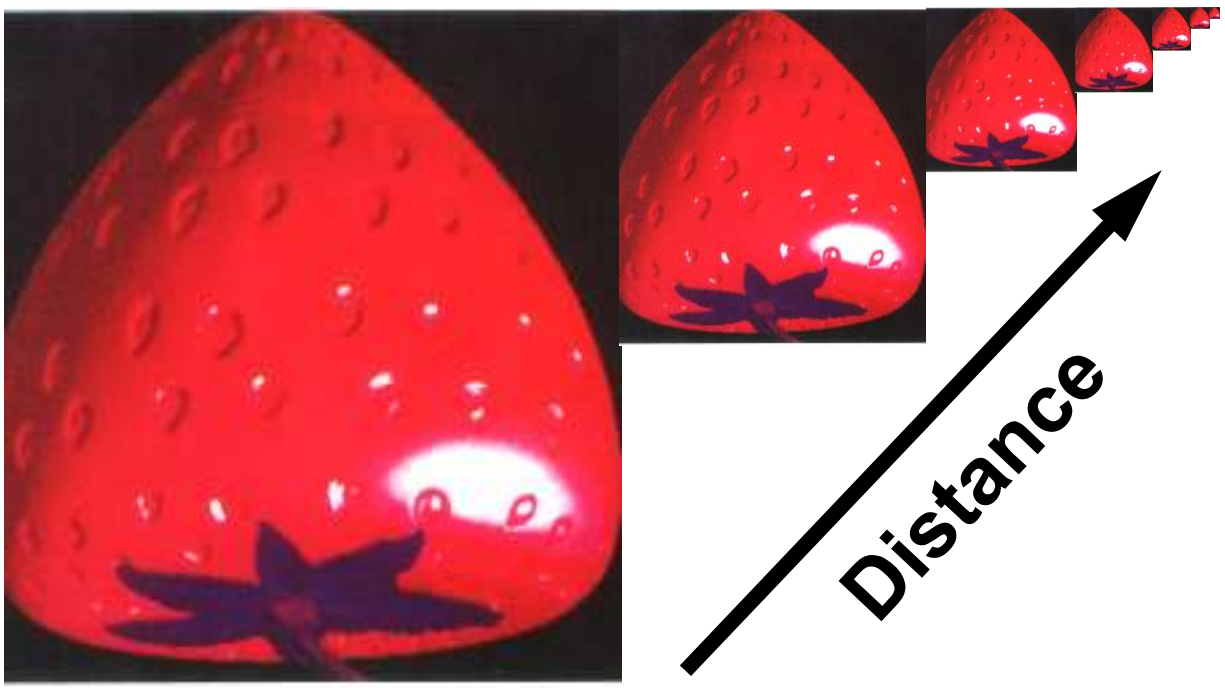
# Texture Anti-Aliasing: Minification

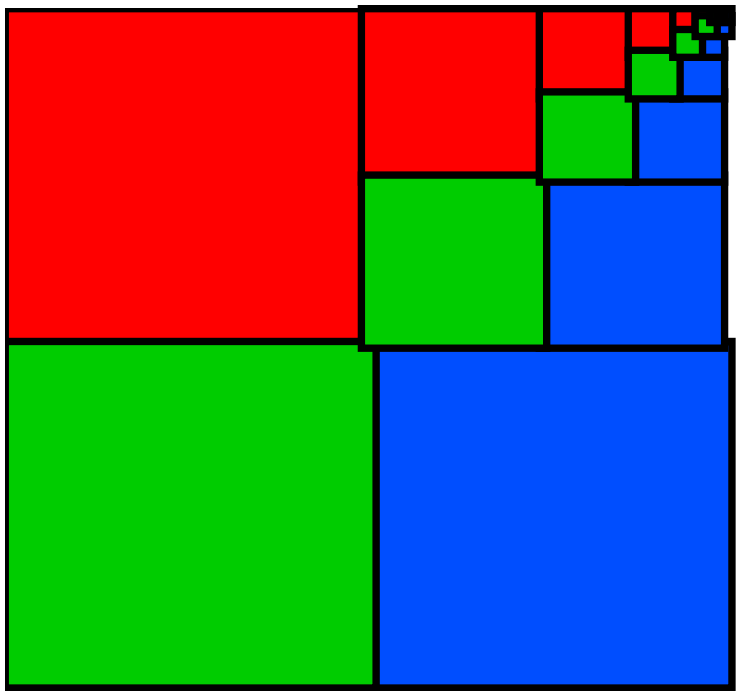
- A good pixel value is the weighted mean of the pixel area projected into texture space



# Texture Anti-Aliasing: MIP Mapping

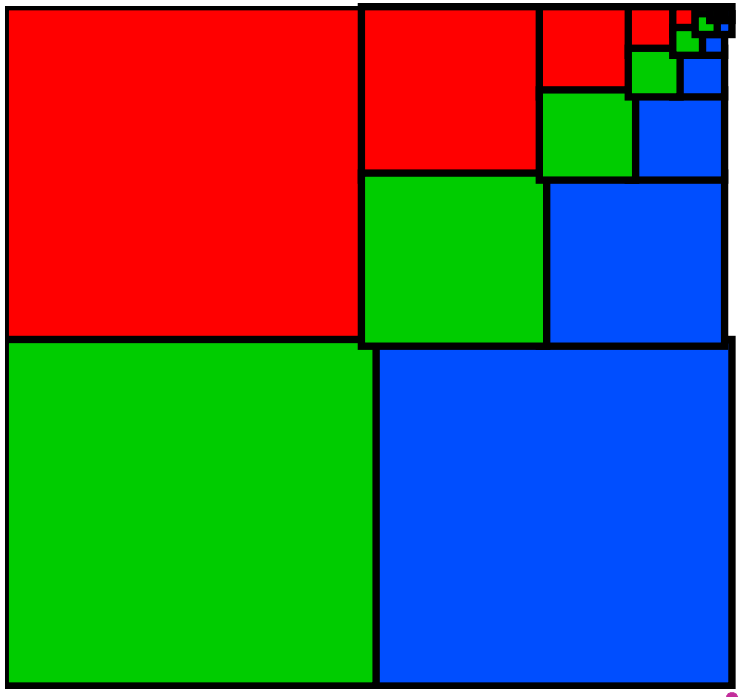
- MIP Mapping (“Multum In Parvo”)
  - Texture size is reduced by factors of 2 (*downsampling* = "many things in a small place")
  - Simple (4 pixel average) and memory efficient
  - Last image is only ONE texel



# 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



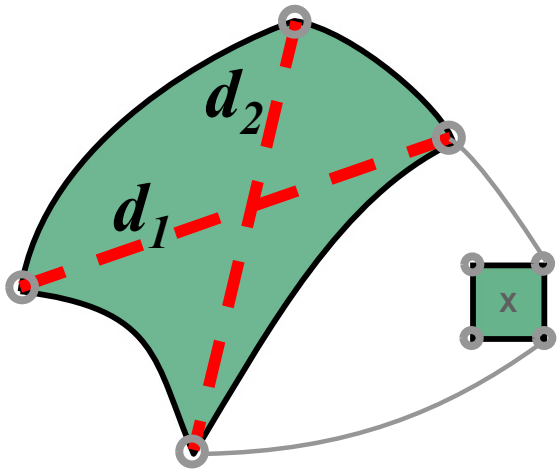
geometric series:

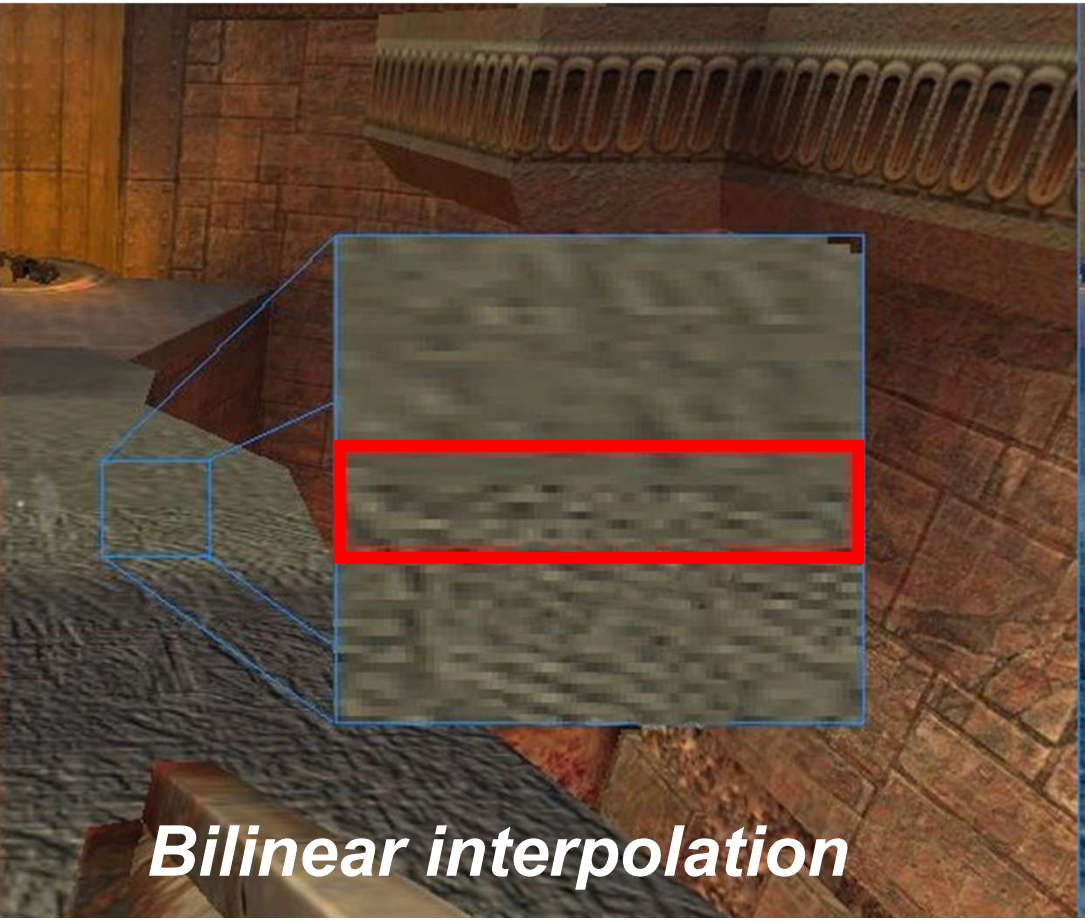
$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} =$$

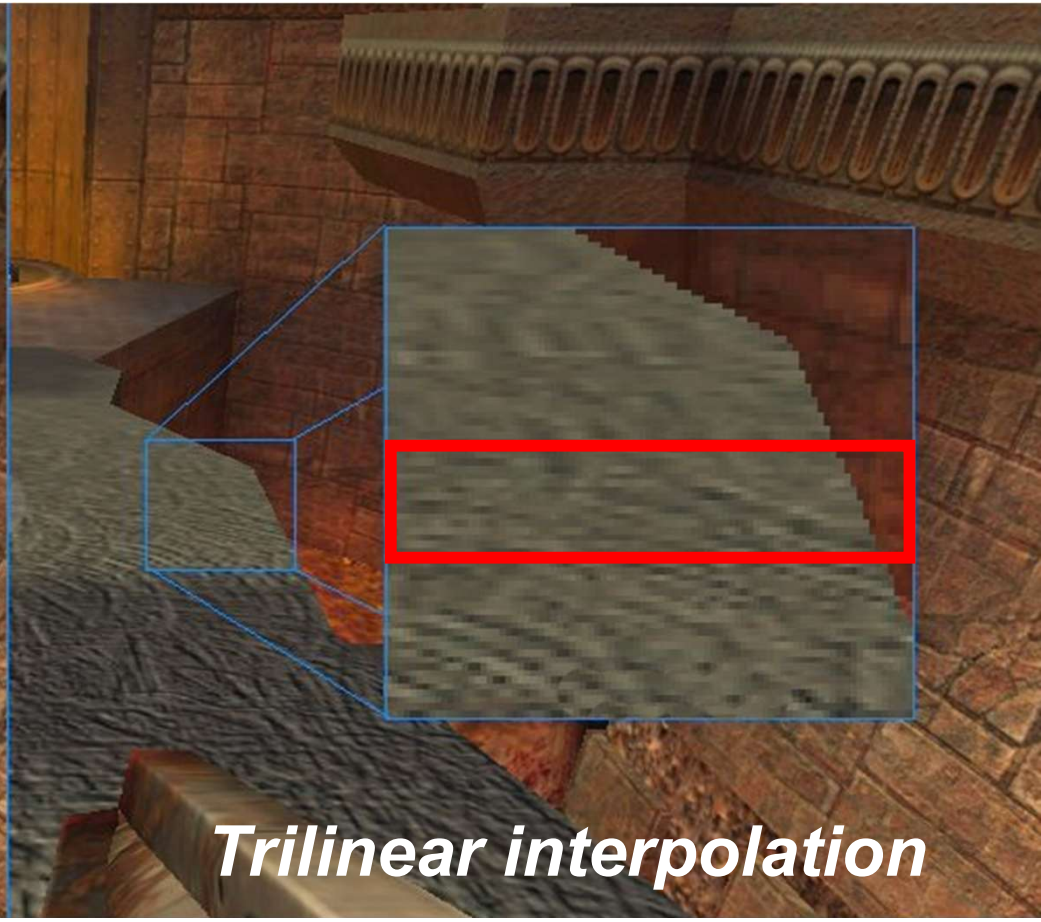
$$= \sum_{k=0}^{n-1} ar^k = a\left(\frac{1-r^n}{1-r}\right)$$

# Texture Anti-Aliasing: MIP Mapping

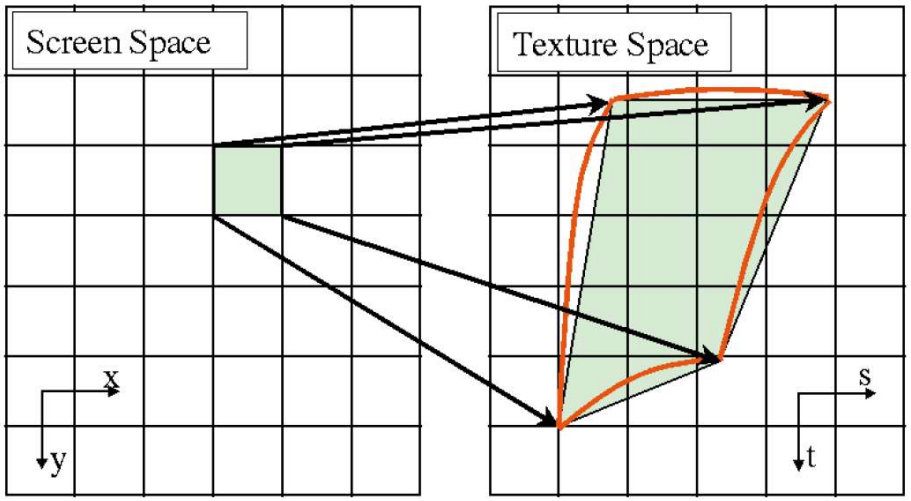
- MIP Mapping Algorithm
- $D := ld(max(d_1,d_2))$ "Mip Map level"
- $T_0 :=$ value from texture $D_0 = trunc\ (D)$
  - Use *bilinear interpolation*

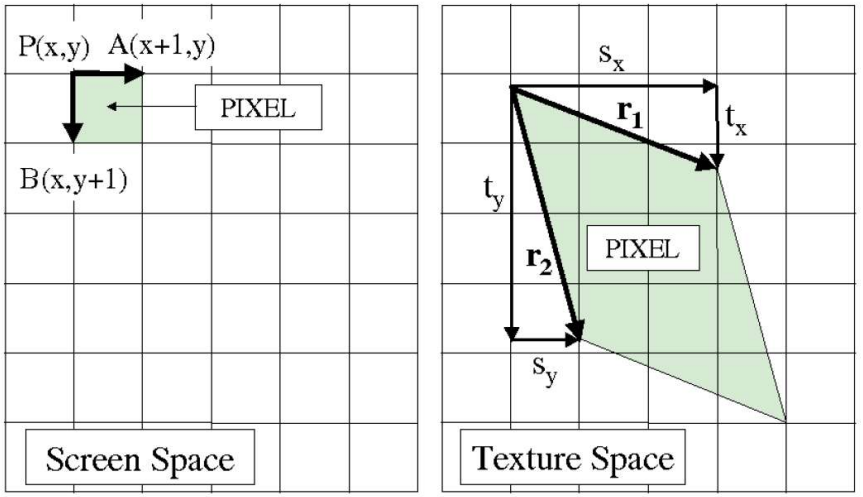


# MIP-Map Level Computation




- 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!)

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

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

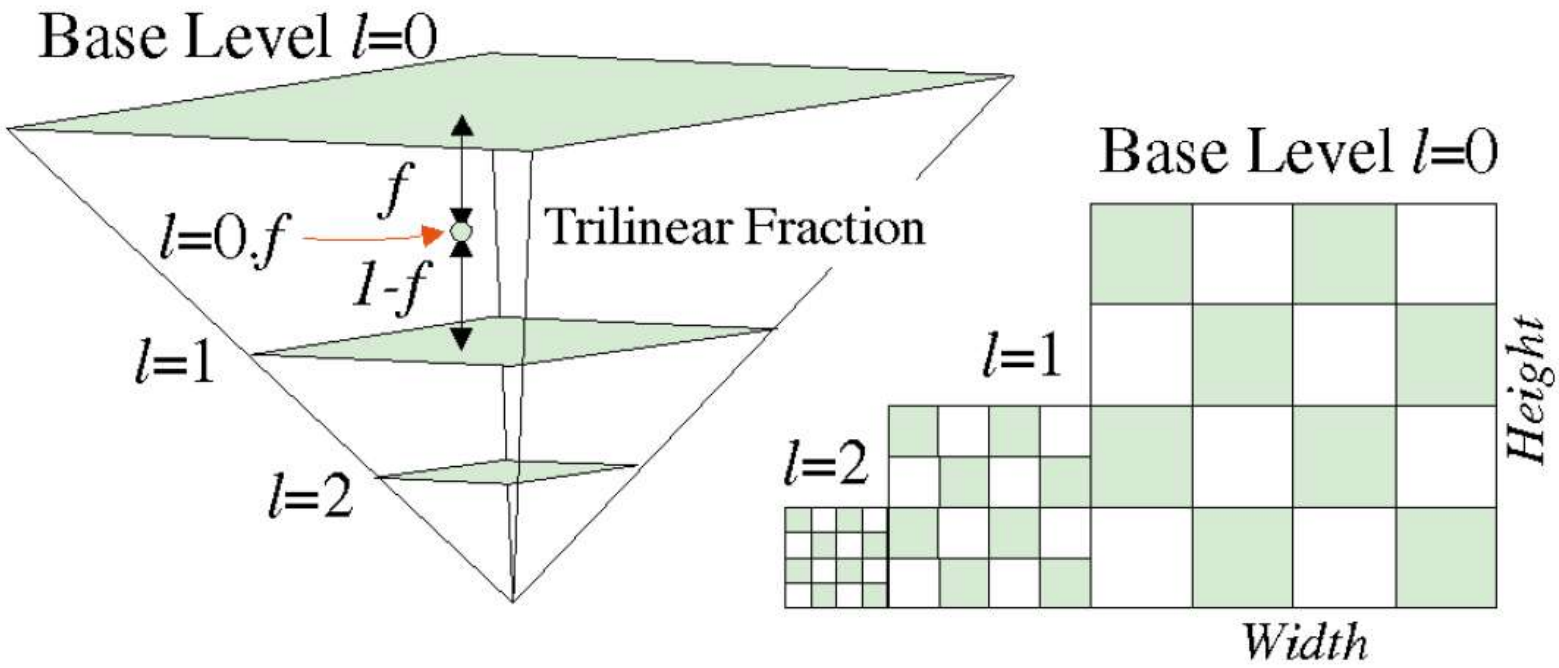
Does not use area of parallelogram but greater hypotenuse [Heckbert, 1983]

- Approximation without square-roots

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

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

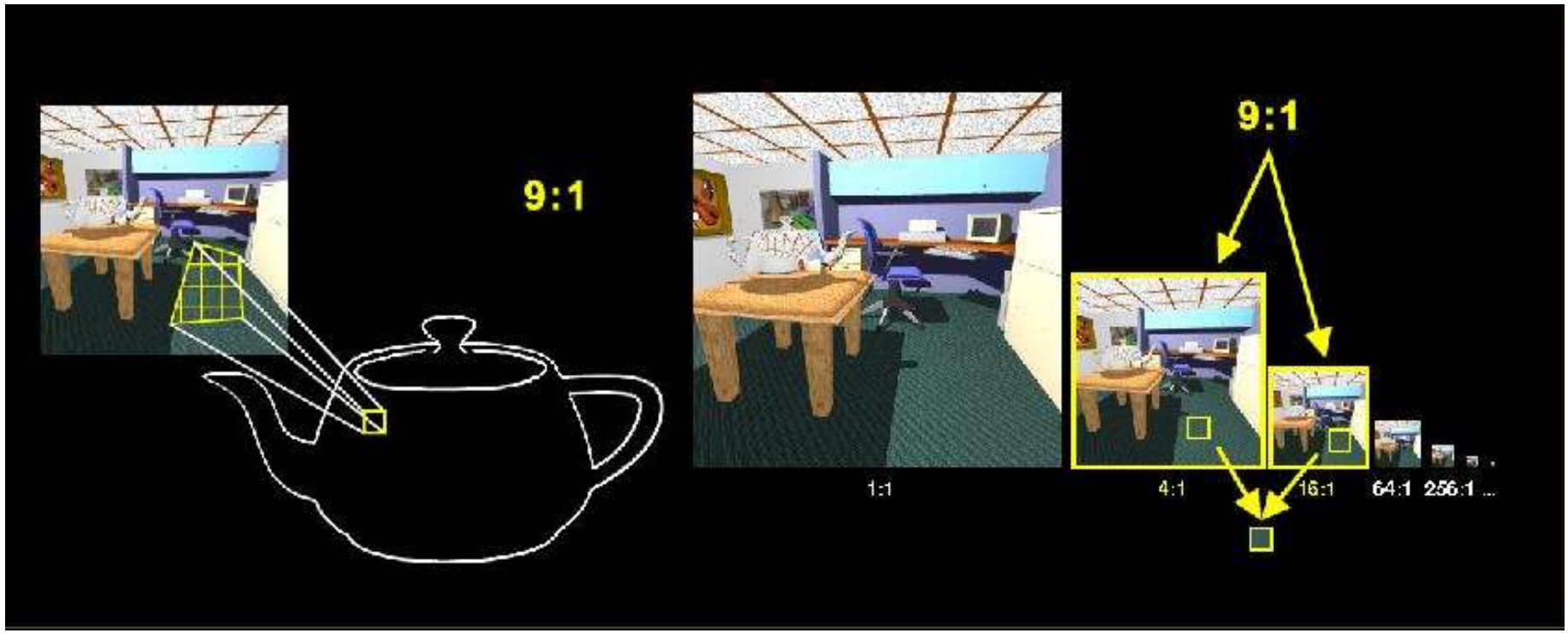
# MIP-Map Level Interpolation




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

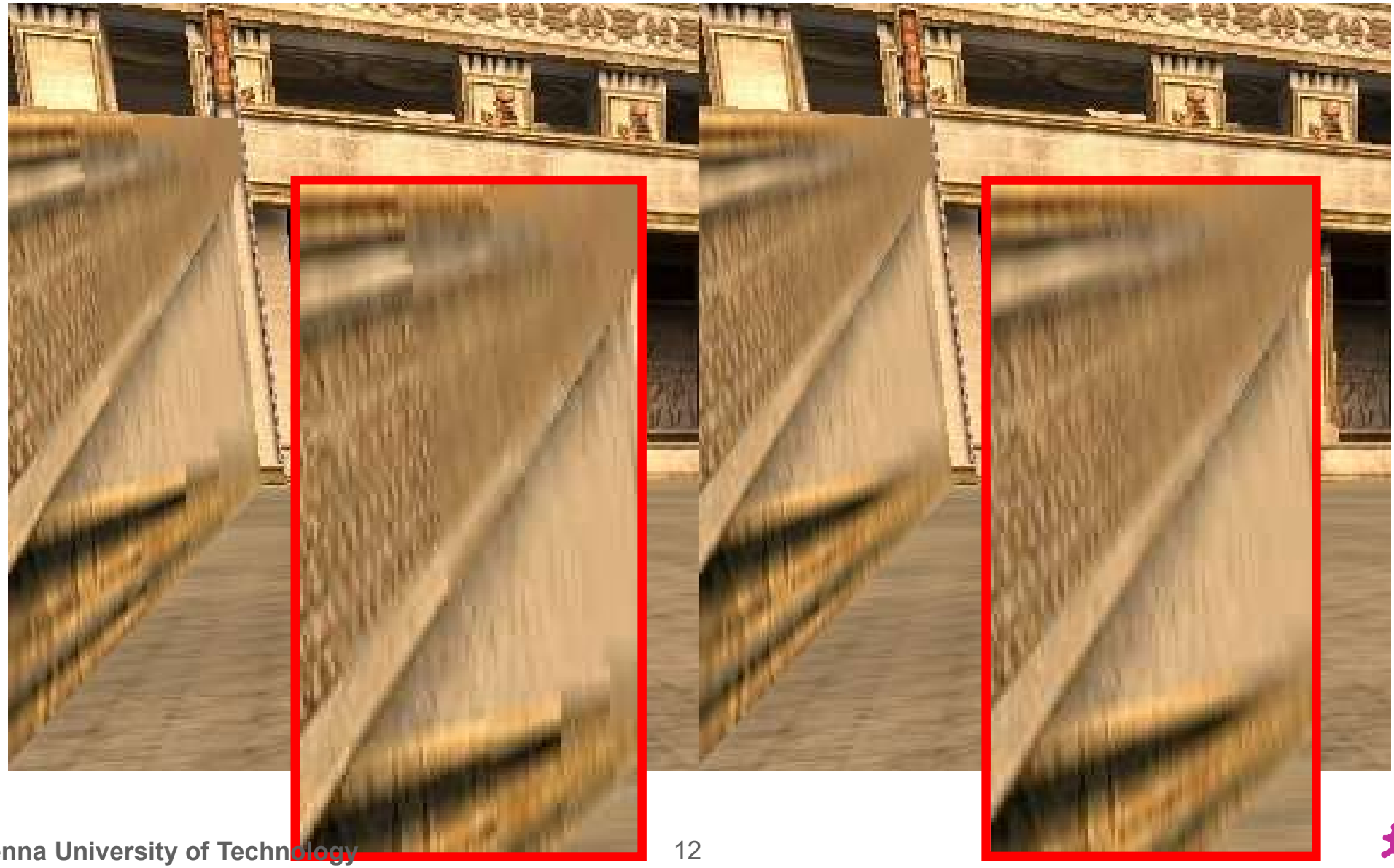
# Texture Anti-Aliasing: MIP Mapping

- 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)



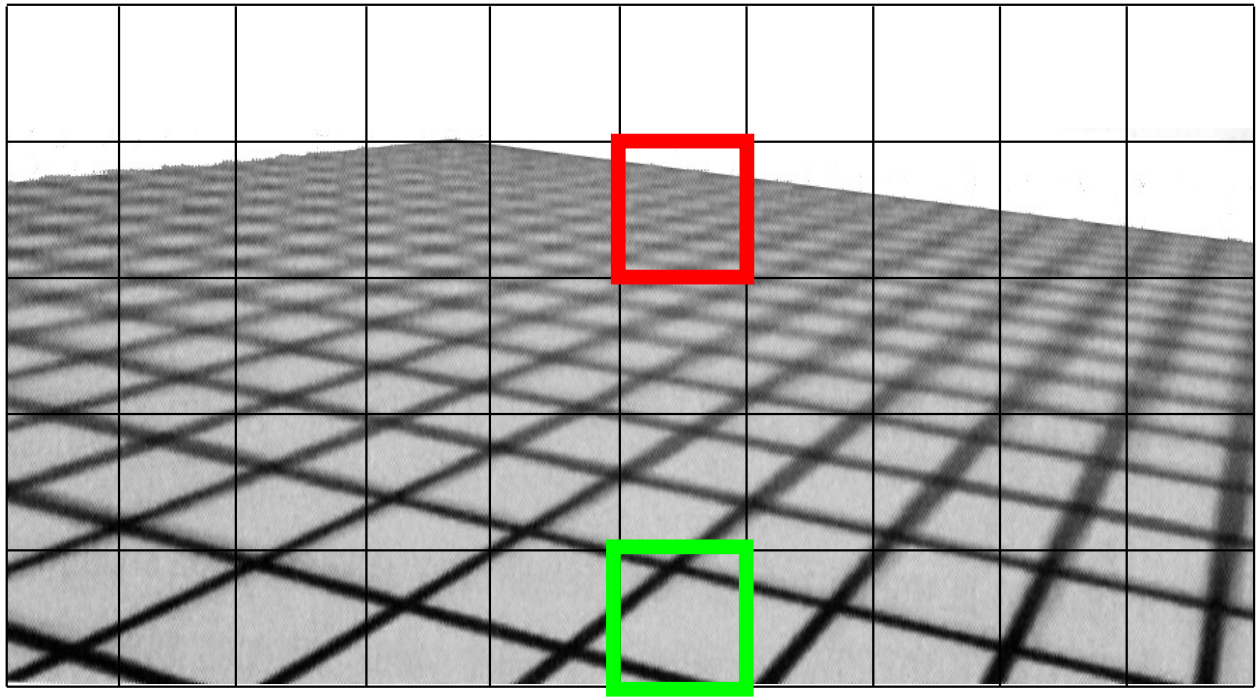
# Texture Anti-Aliasing: MIP Mapping

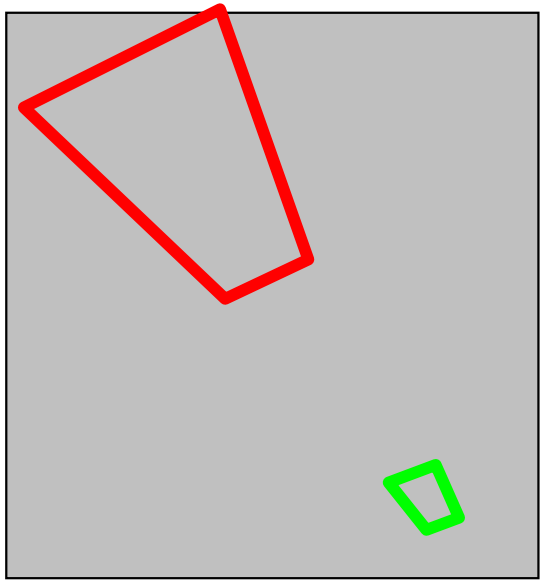
- Other example for bilinear vs. trilinear filtering



# Anti-Aliasing: Anisotropic Filtering

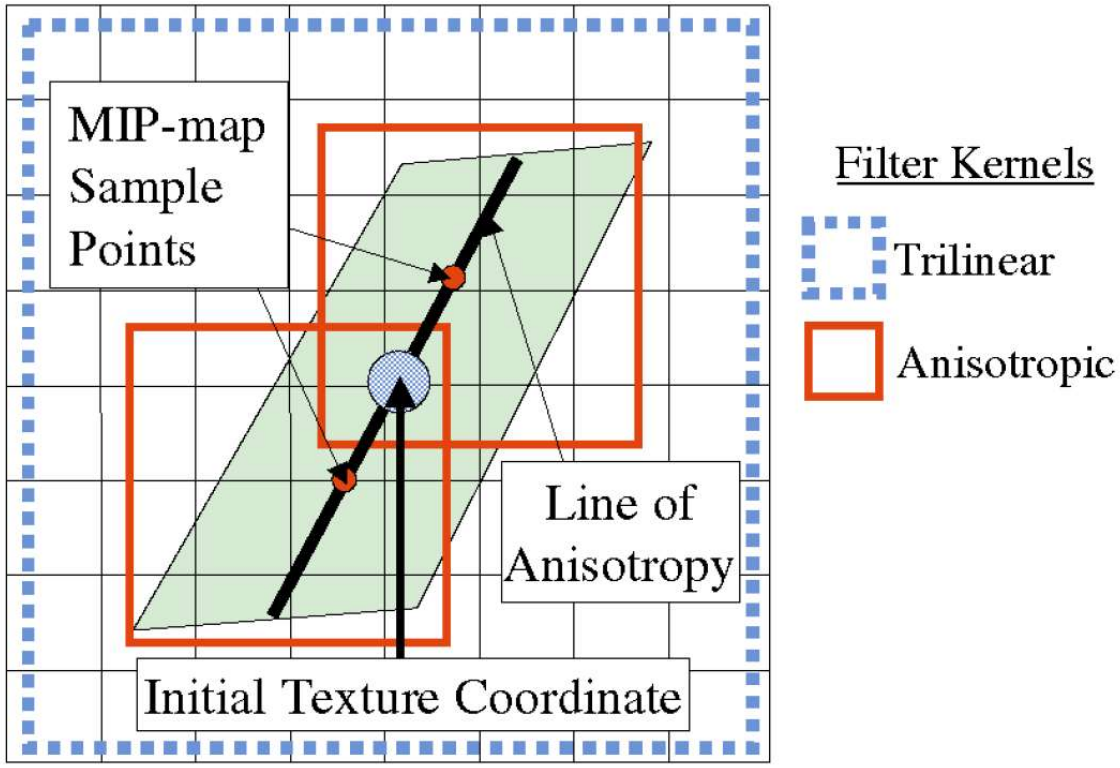
- Anisotropic filtering
  - View-dependent filter kernel
  - Implementation: *summed area table*, "*RIP Mapping*", *footprint assembly*, *elliptical weighted average* (EWA)





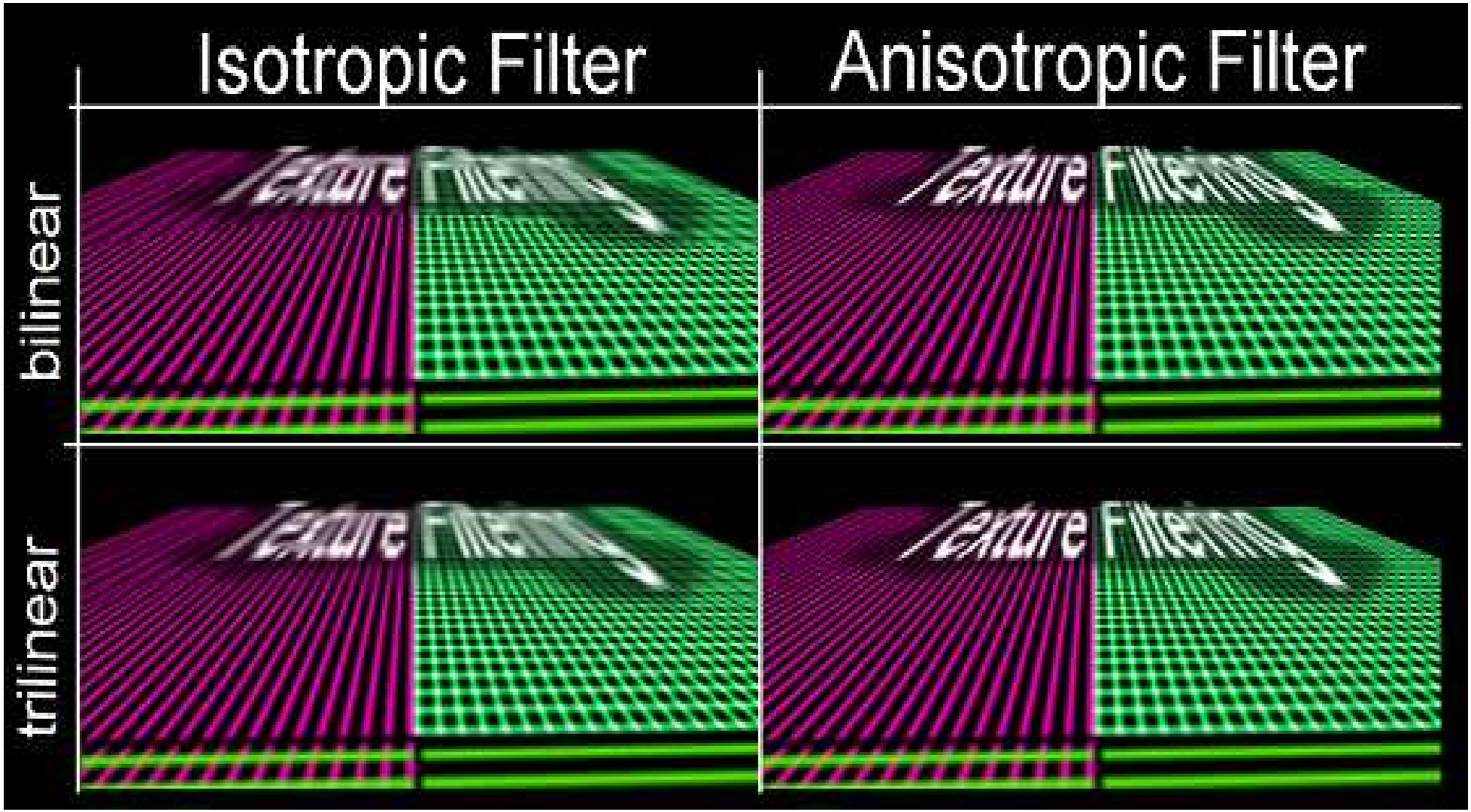
**Texture space**

# 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

for Vulkan, see `vkSampler`, `VkSamplerCreateInfo::magFilter`, `VkSamplerCreateInfo::minFilter`, `VK_FILTER_NEAREST`, `VK_FILTER_LINEAR`, `VkSamplerCreateInfo::mipmapMode`, `VK_SAMPLER_MIPMAP_MODE_NEAREST`, `VK_SAMPLER_MIPMAP_MODE_LINEAR`, …



# GPU Virtual Texturing

# 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**

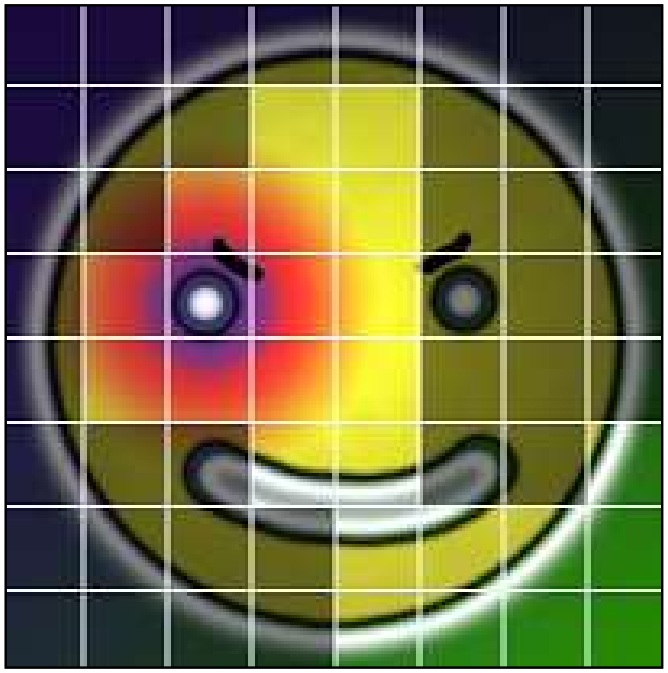
Hardware Virtual Texturing, Graham Sellers,
from SIGGRAPH 2013 course “Rendering Massive Virtual Worlds”

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

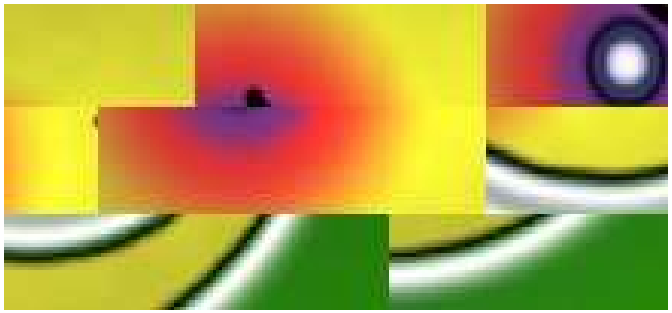
# Virtual Texturing

Divide texture up into tiles

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



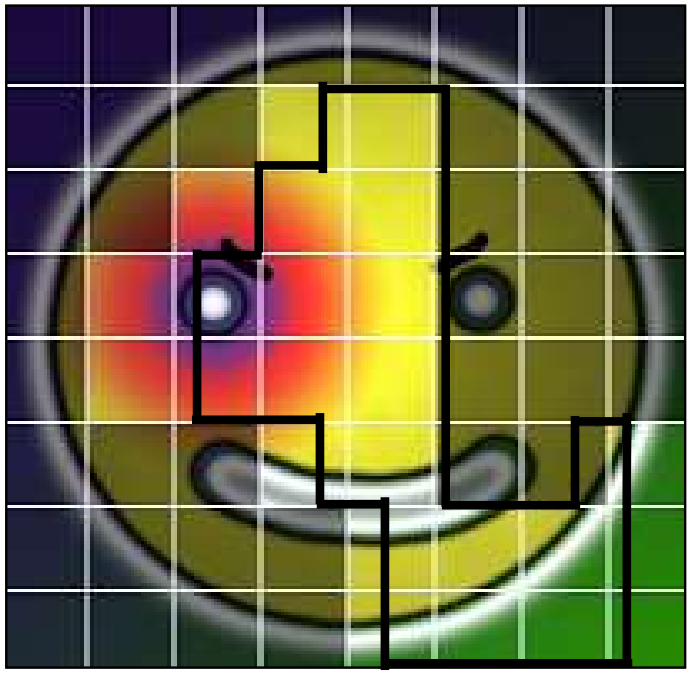
Virtual Texture



Physical Texture

# Virtual Texturing

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



RGBA8, 1024x1024, 64 tiles

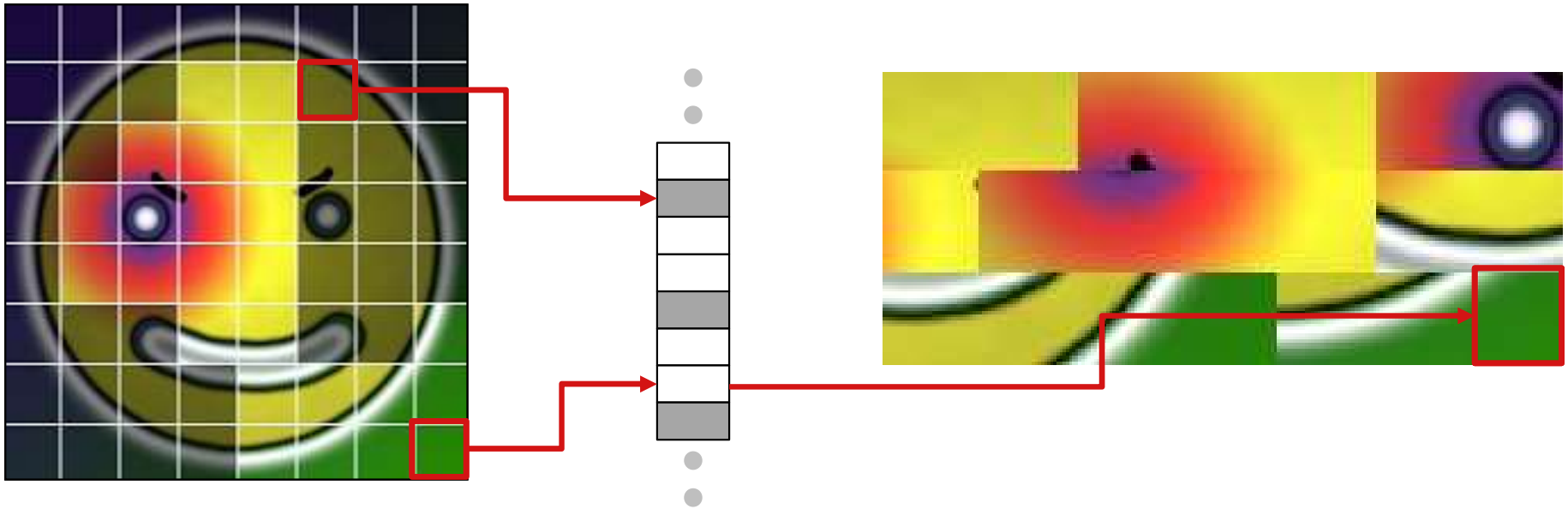
| | Virtual | Physical |
|---|---|---|
| Memory | 4096 kB | 1536 kB |

# Virtual Texturing

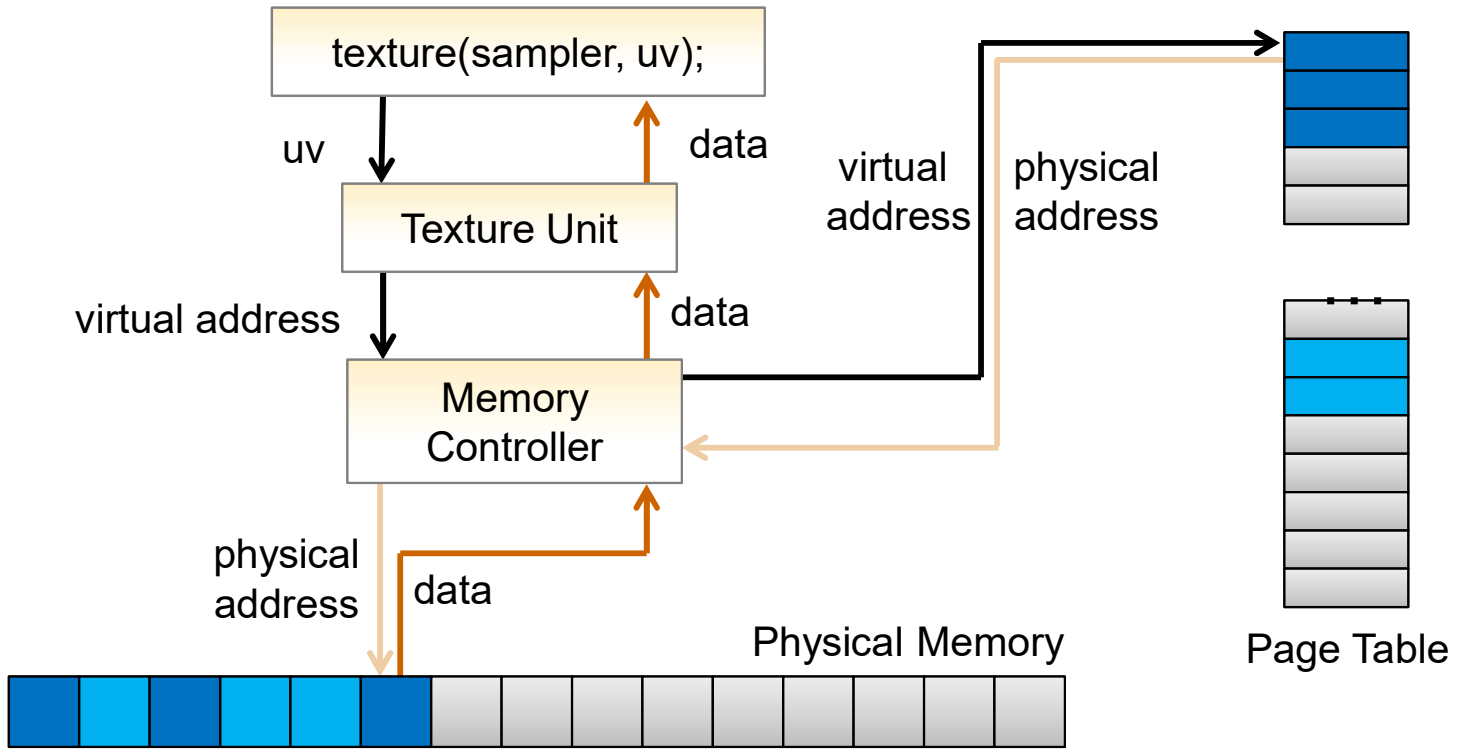


Use indirection table to map virtual to physical

- 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)**

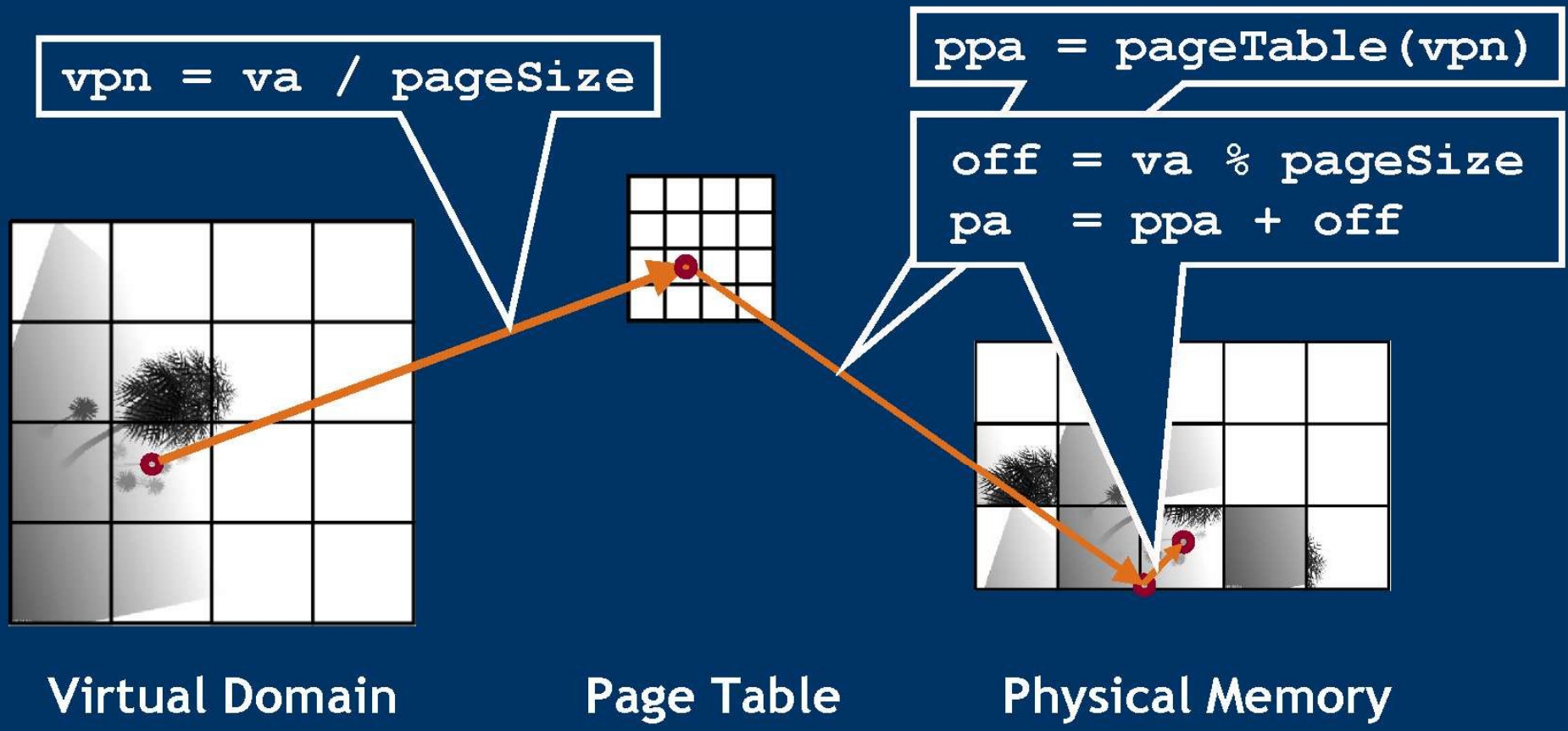
- On CPUs: Fernando et al., ACM SIGGRAPH 2001

**Resolution-Matched Shadow Maps**

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

# ASM Data Structure (Adaptive Shadow Maps)

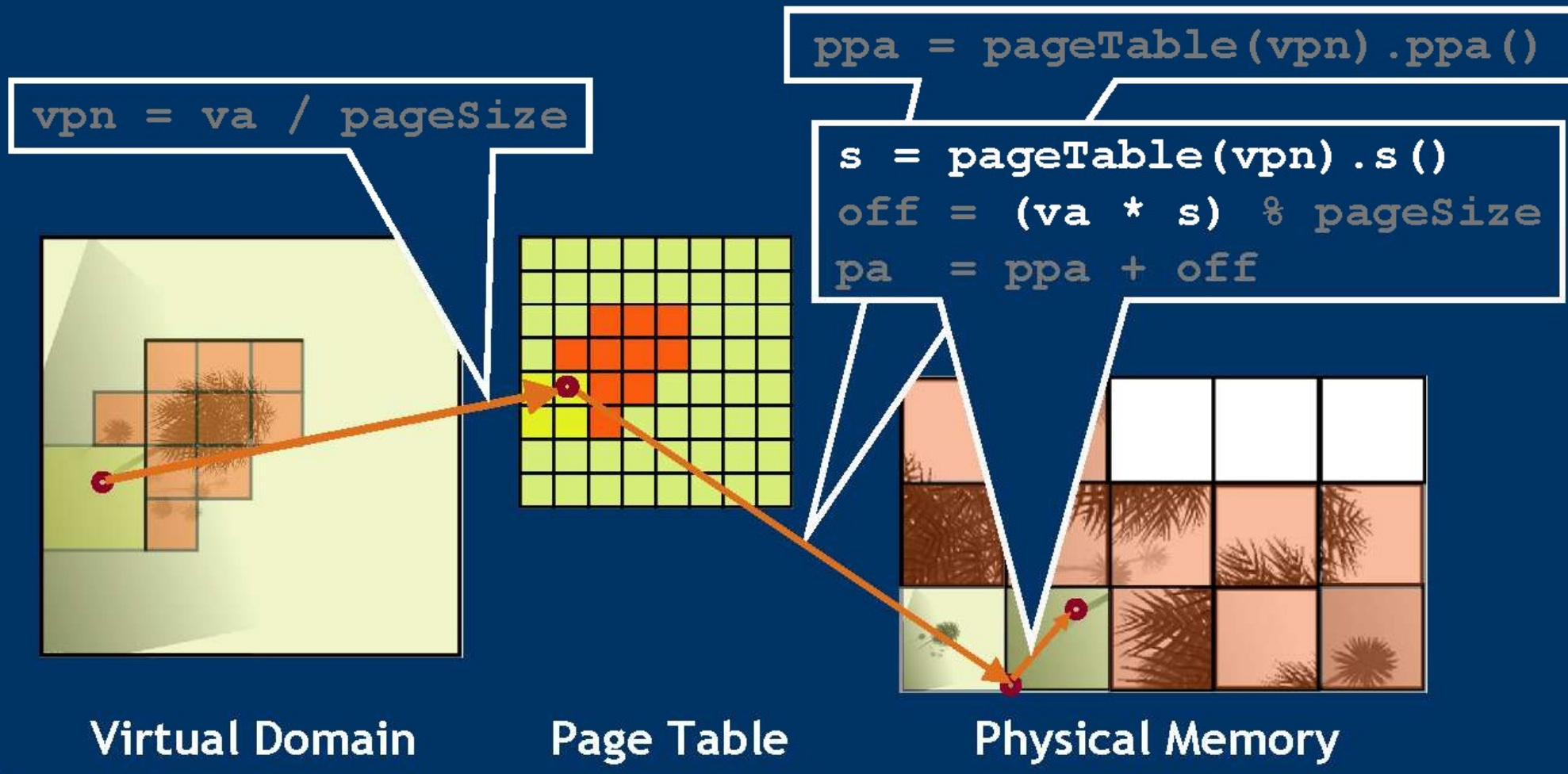
- Page table example




Aaron Lefohn
University of California, Davis

# ASM Data Structure (Adaptive Shadow Maps)

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




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

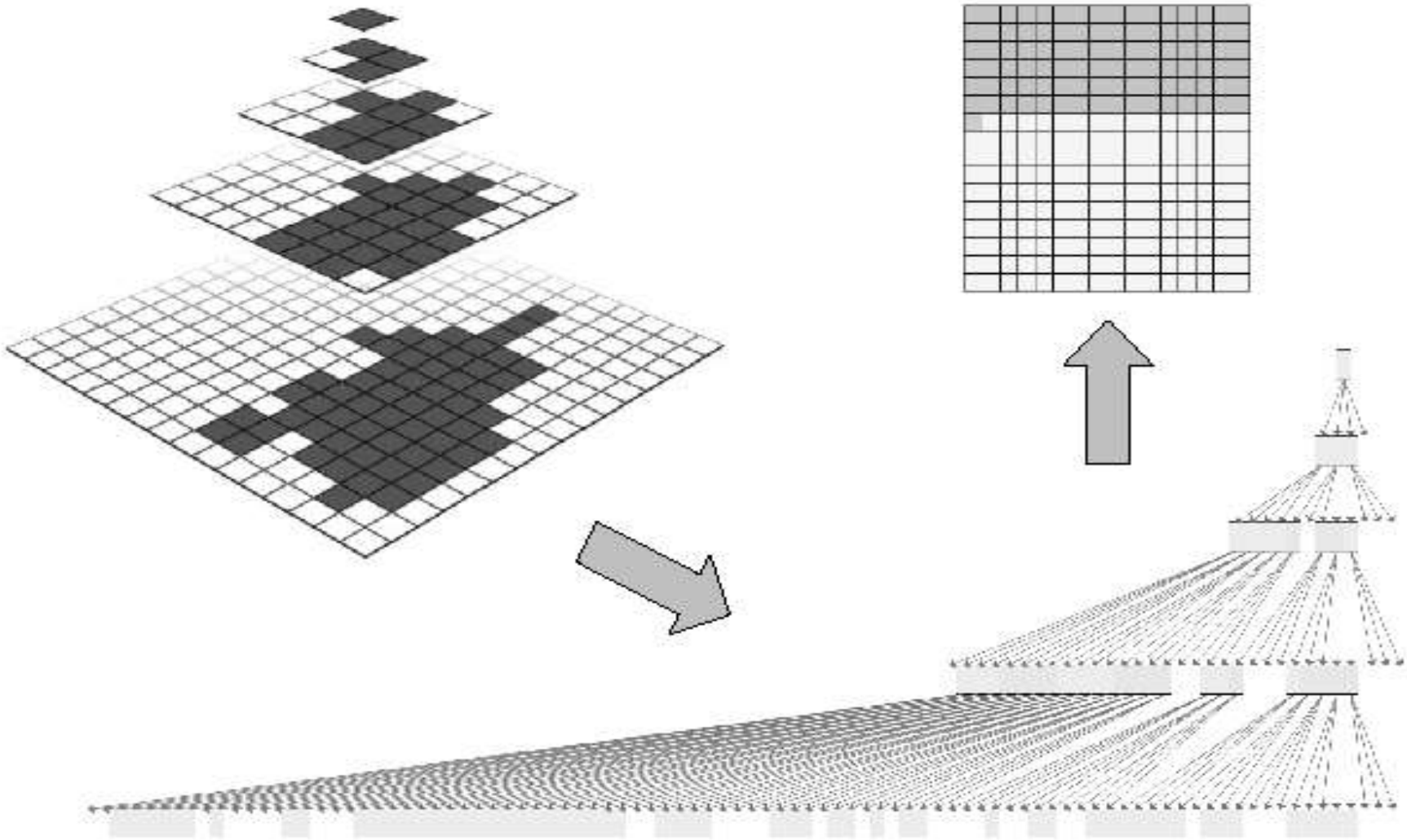
# Virtual Texturing

# Virtual Texturing

# Virtual Texturing

Texture Pyramid with Sparse Page Residency

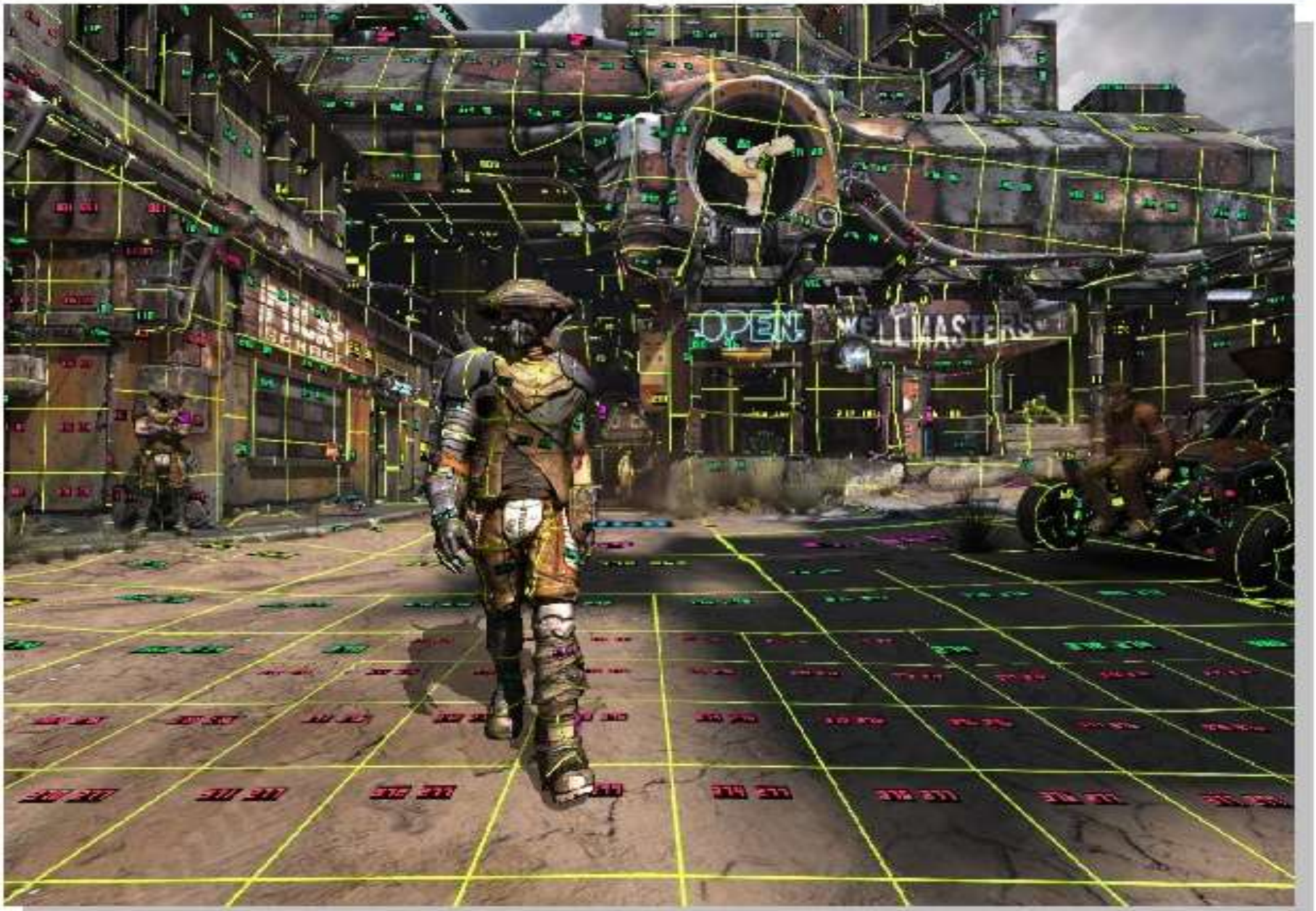
Physical Page Texture



Quad-tree of Sparse Texture Pyramid

# Virtual Texturing

# Virtual Texturing



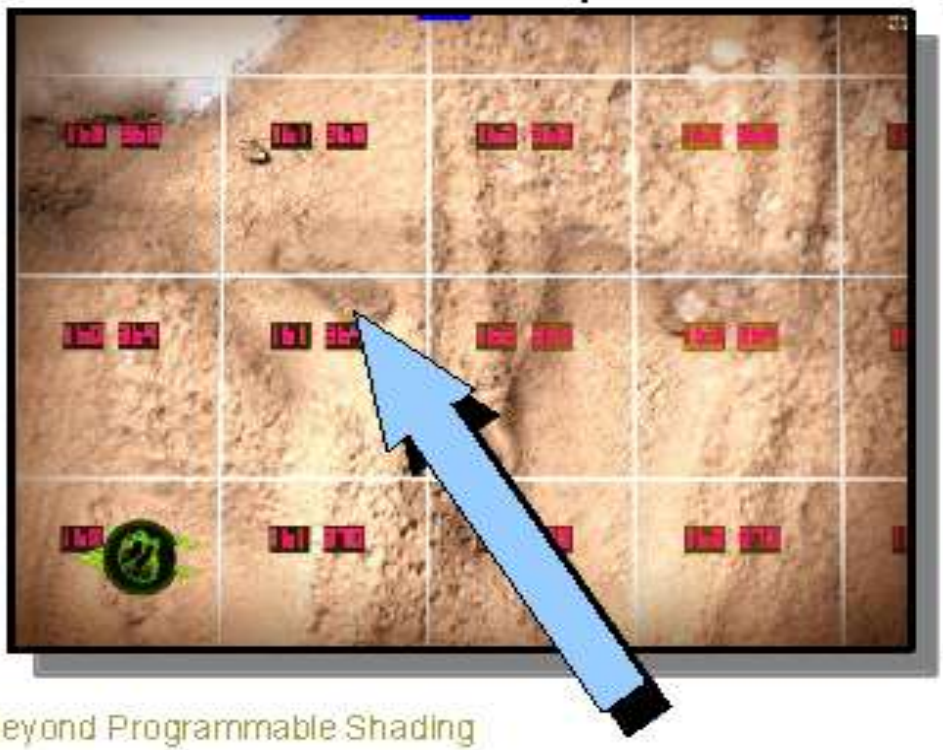
# Virtual Texturing

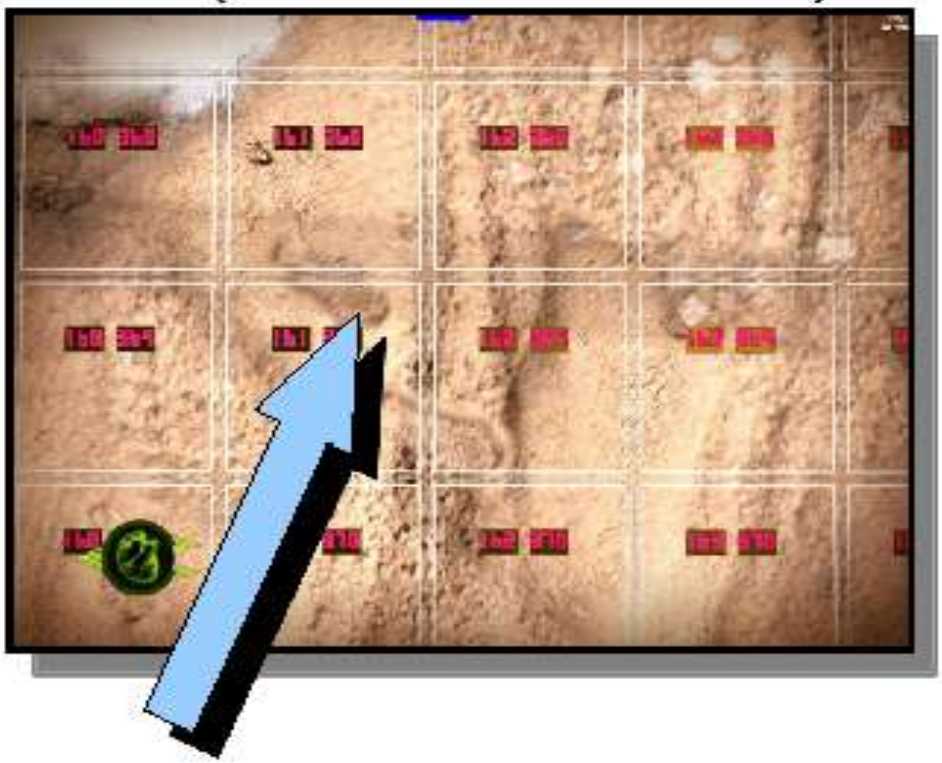
A few interesting issues...
- Texture filtering
- Thrashing due to physical memory oversubscription
- LOD transitions under high latency


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)

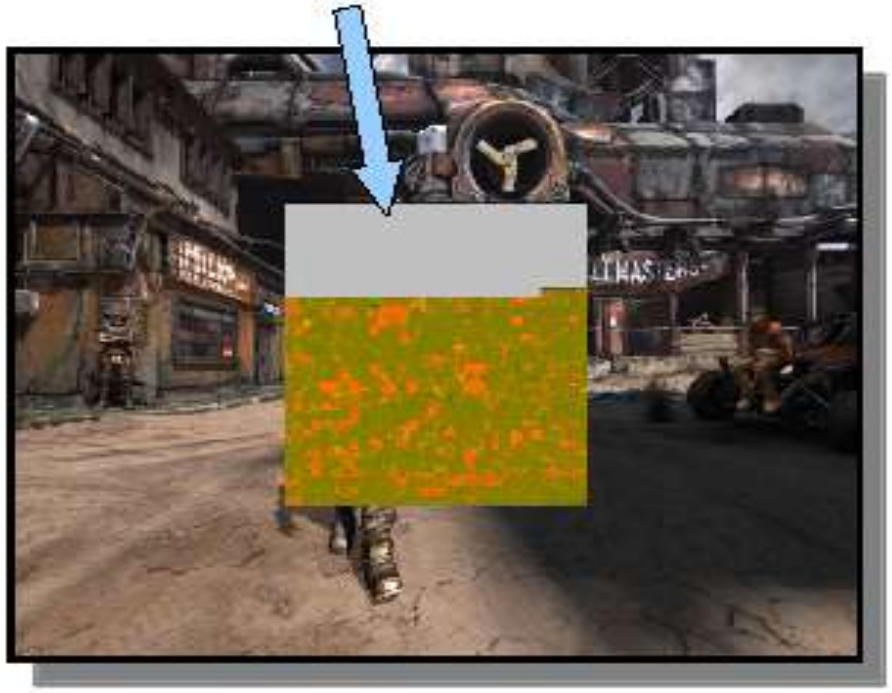




# Virtual Texturing - Thrashing

- 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



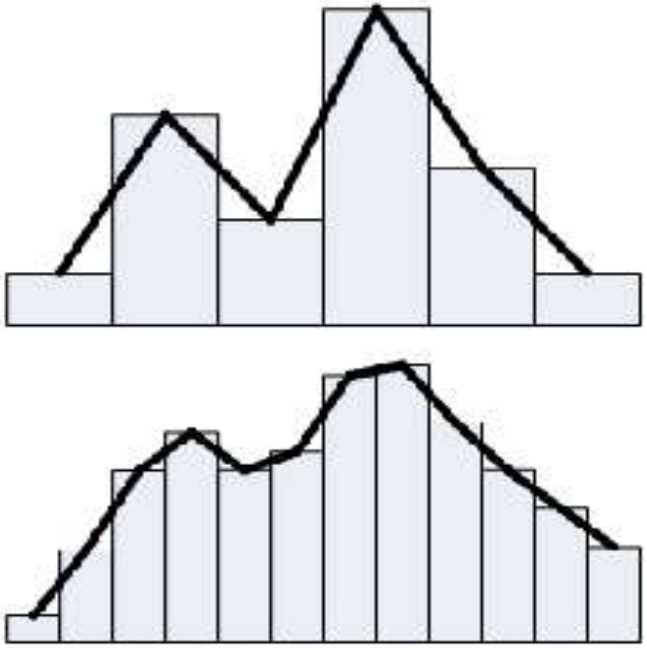
1024 Physical Pages

8x8 pages

64 Physical Pages

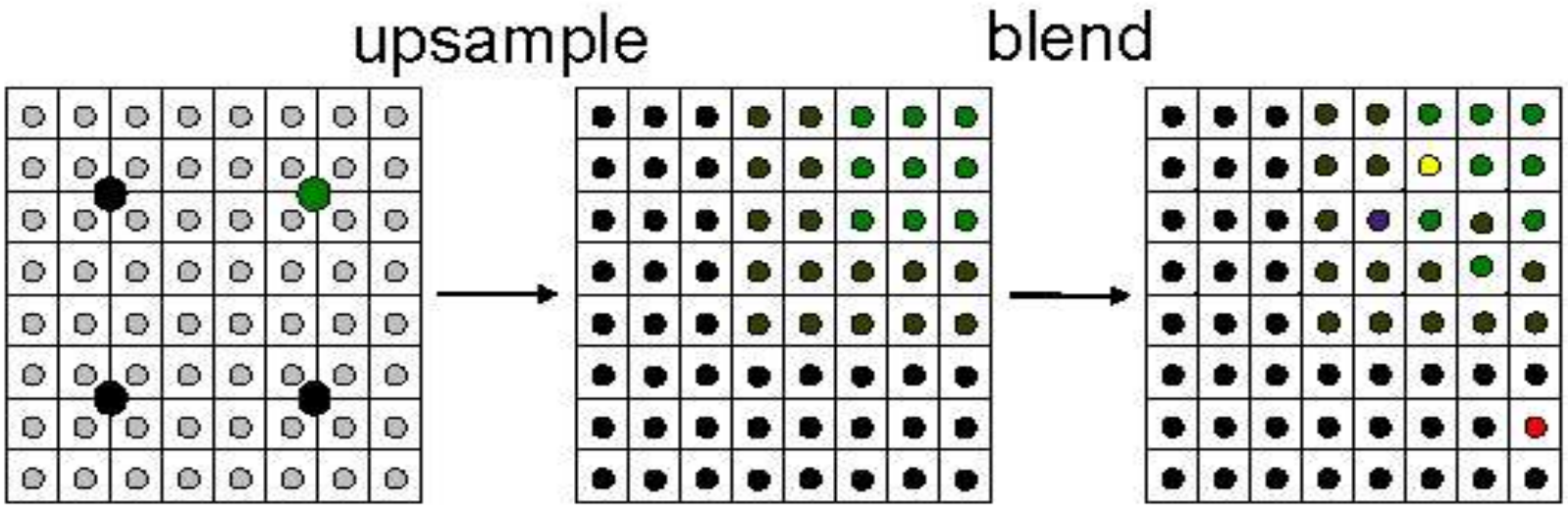
# Virtual Texturing – LOD Snap

- 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



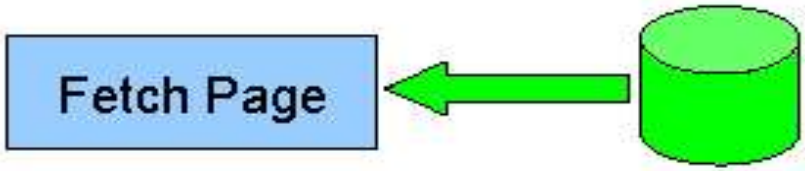
# Virtual Texturing – LOD Snap

- 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



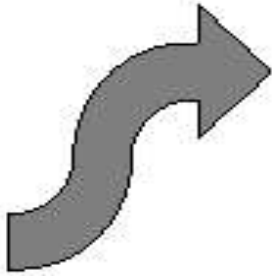

- But this is a real-time app... so no blocking allowed
- Cache handles hits, schedules misses to load in background
- Resident pages managed independent of disk cache
- Physical pages organized as quad-tree per virtual texture
- Linked lists for free, LRU, and locked pages

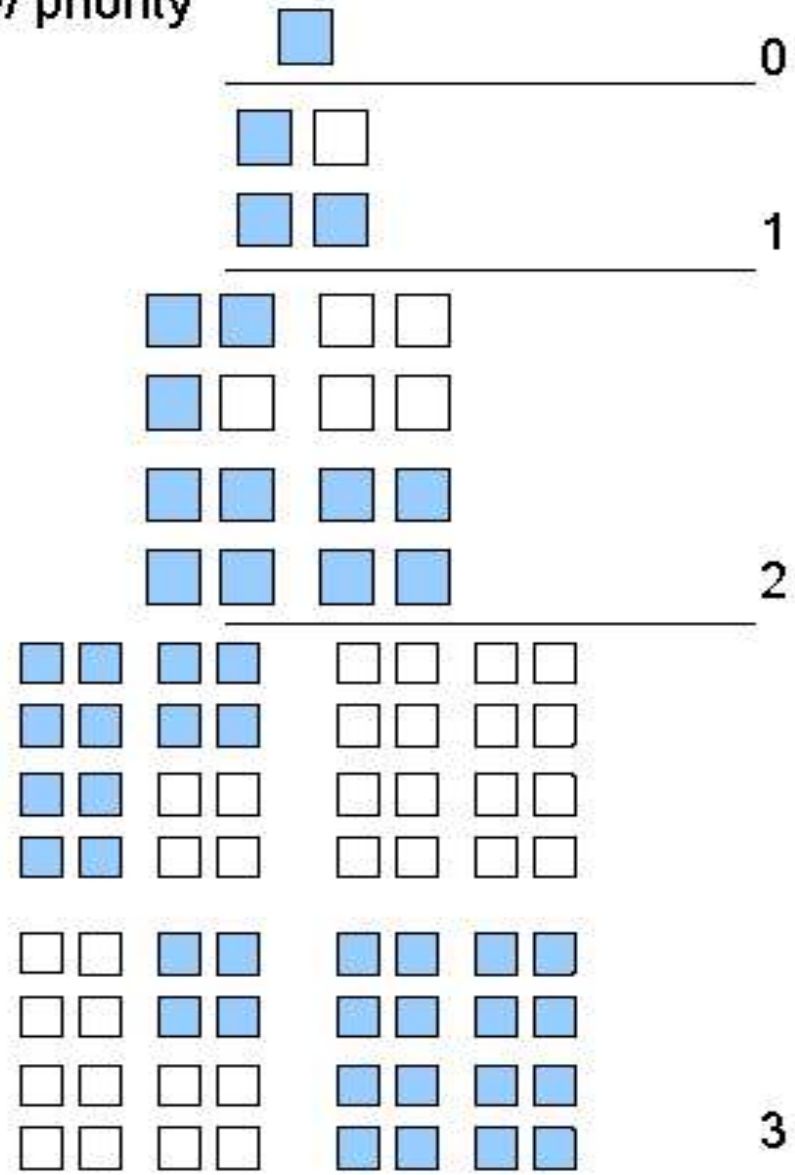
# Virtual Texturing - Feedback

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

Color Buffer

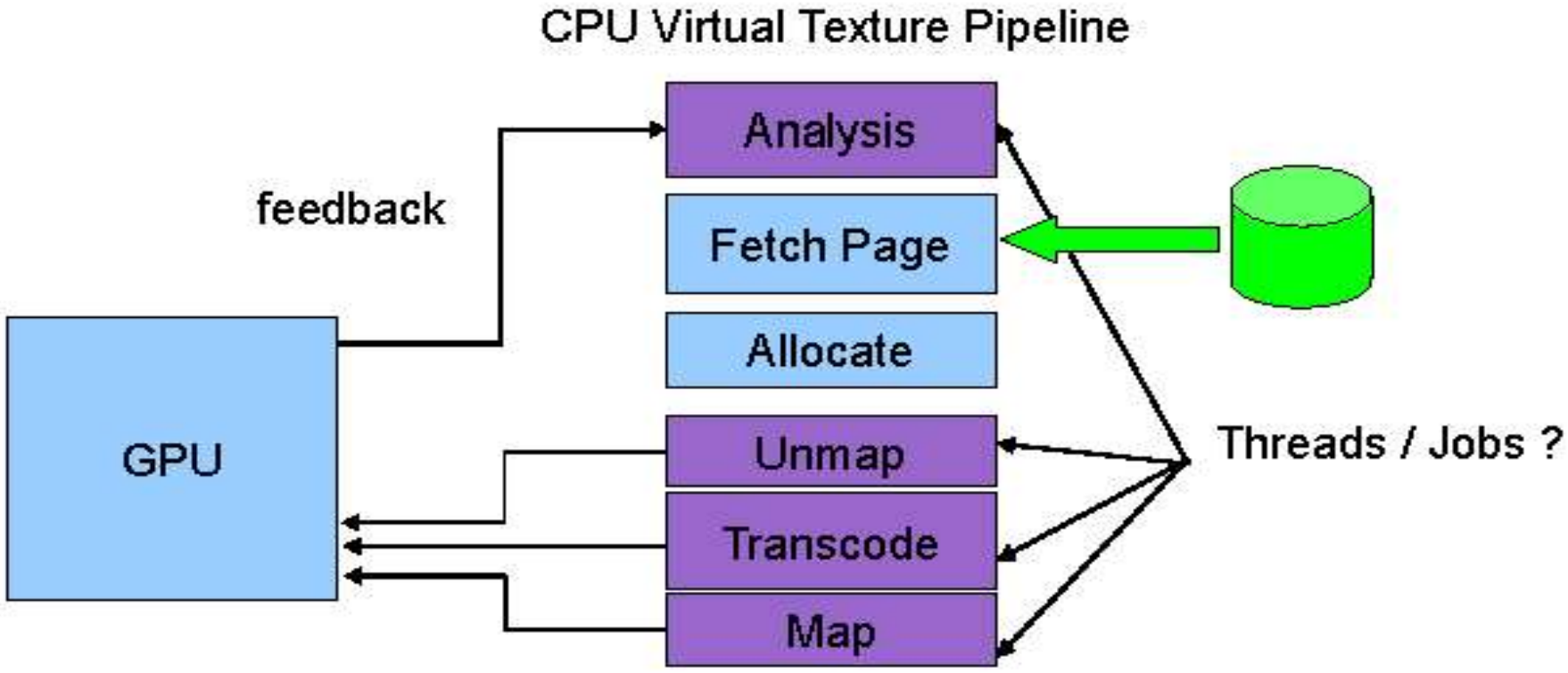
Feedback Buffer





# 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:**

**Unreal Engine 5 Virtual Texturing**

`https://dev.epicgames.com/documentation/en-us/unreal-engine/virtual-texturing-in-unreal-engine`

`https://dev.epicgames.com/documentation/en-us/unreal-engine/streaming-virtual-texturing-in-unreal-engine`

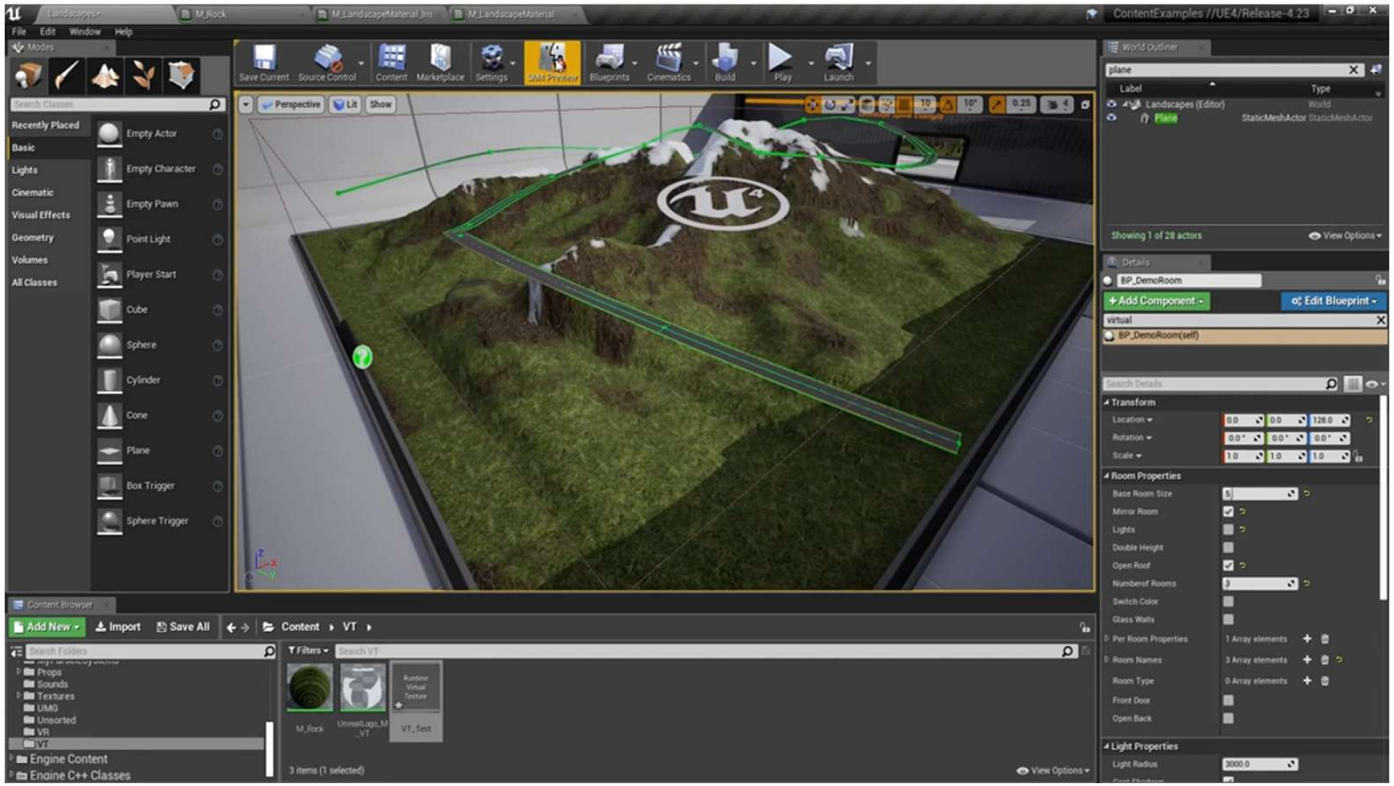
`https://dev.epicgames.com/documentation/en-us/unreal-engine/virtual-texture-memory-pools-in-unreal-engine`

`https://dev.epicgames.com/documentation/en-us/unreal-engine/virtual-texturing-settings-and-properties-in-unreal-engine`

# Unreal Engine 5 Virtual Texturing

## Run-Time Virtual Texturing

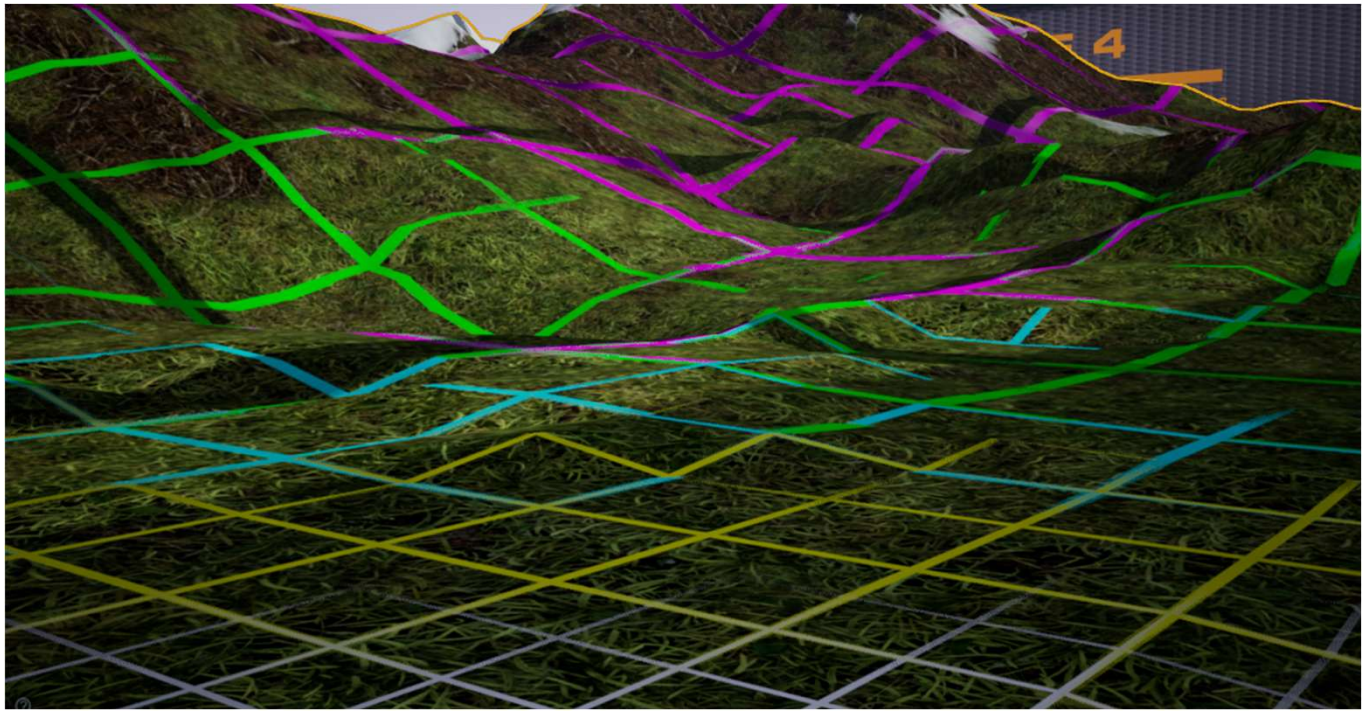
- Procedurally generate hires texture content; materials, etc.



# Unreal Engine 5 Virtual Texturing

Streaming Virtual Texturing

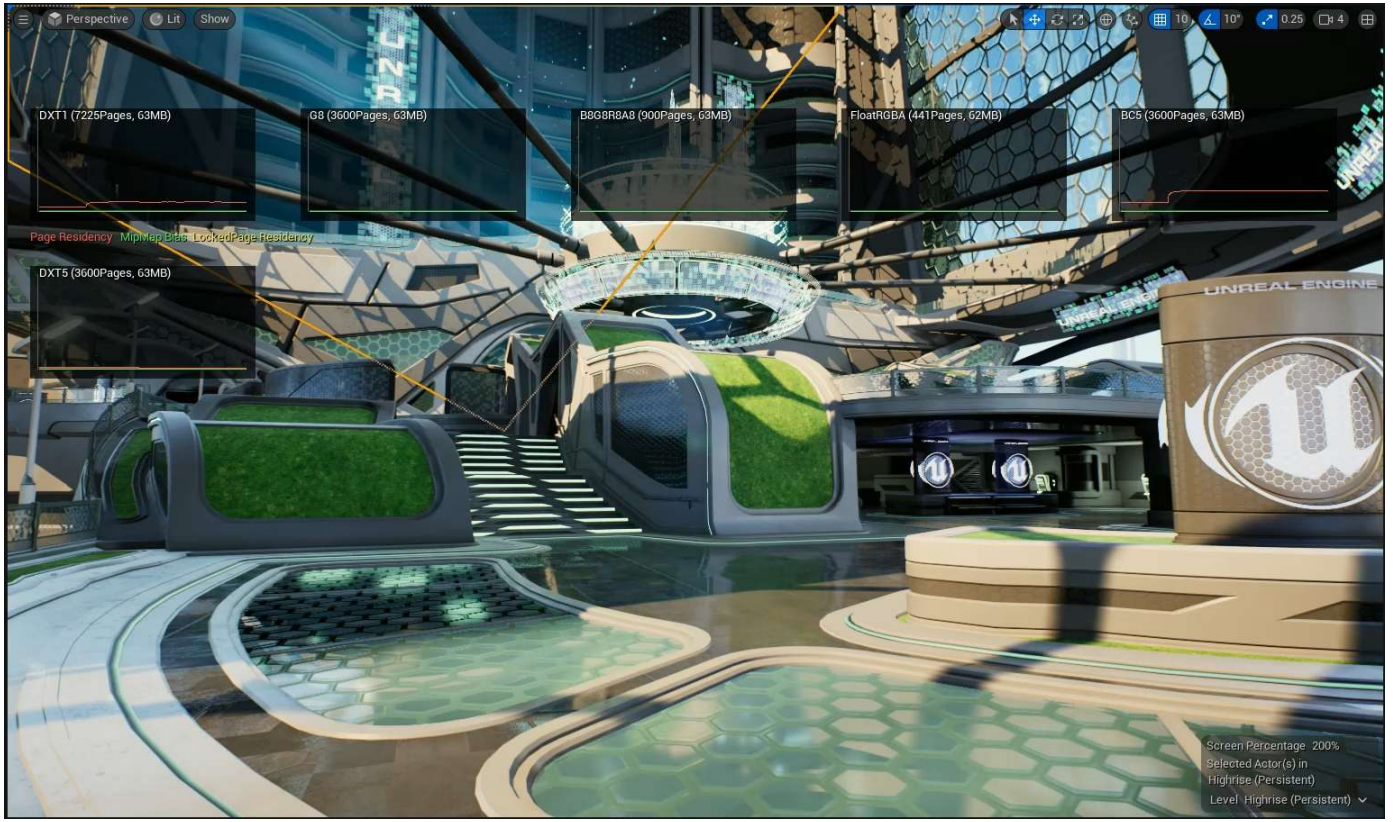
- Stream texture data from disk; default tile size 128, up to 4K tiles
- Also: pre-baked hires light maps, etc.



# Unreal Engine 5 Virtual Texture Memory Pools

One pool for virtual textures of each type/format

- Can use *residency mipmap bias* (mipmap LOD bias)



# Virtual Texturing

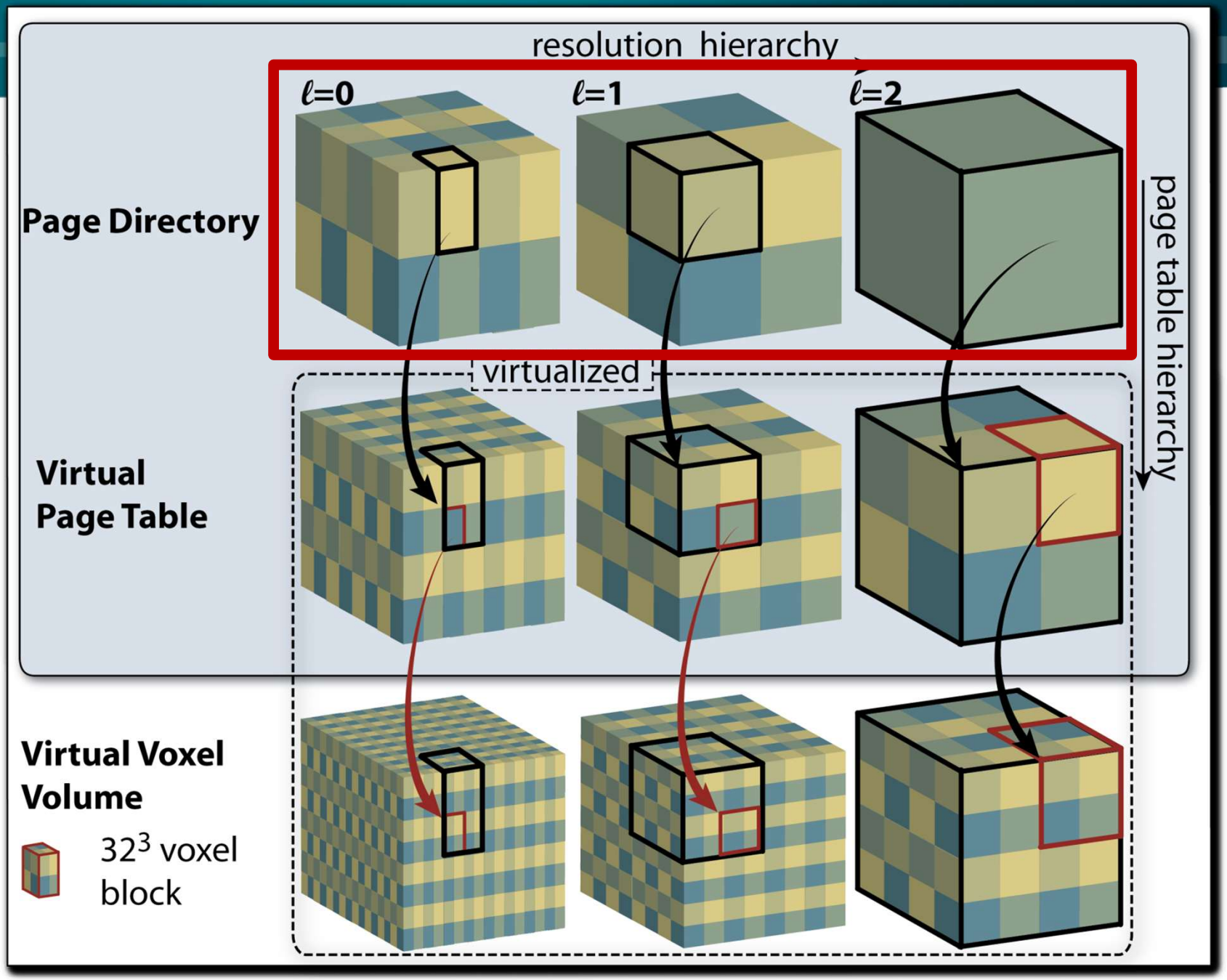
**Example #5:**

**Petascale Volume Rendering**

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

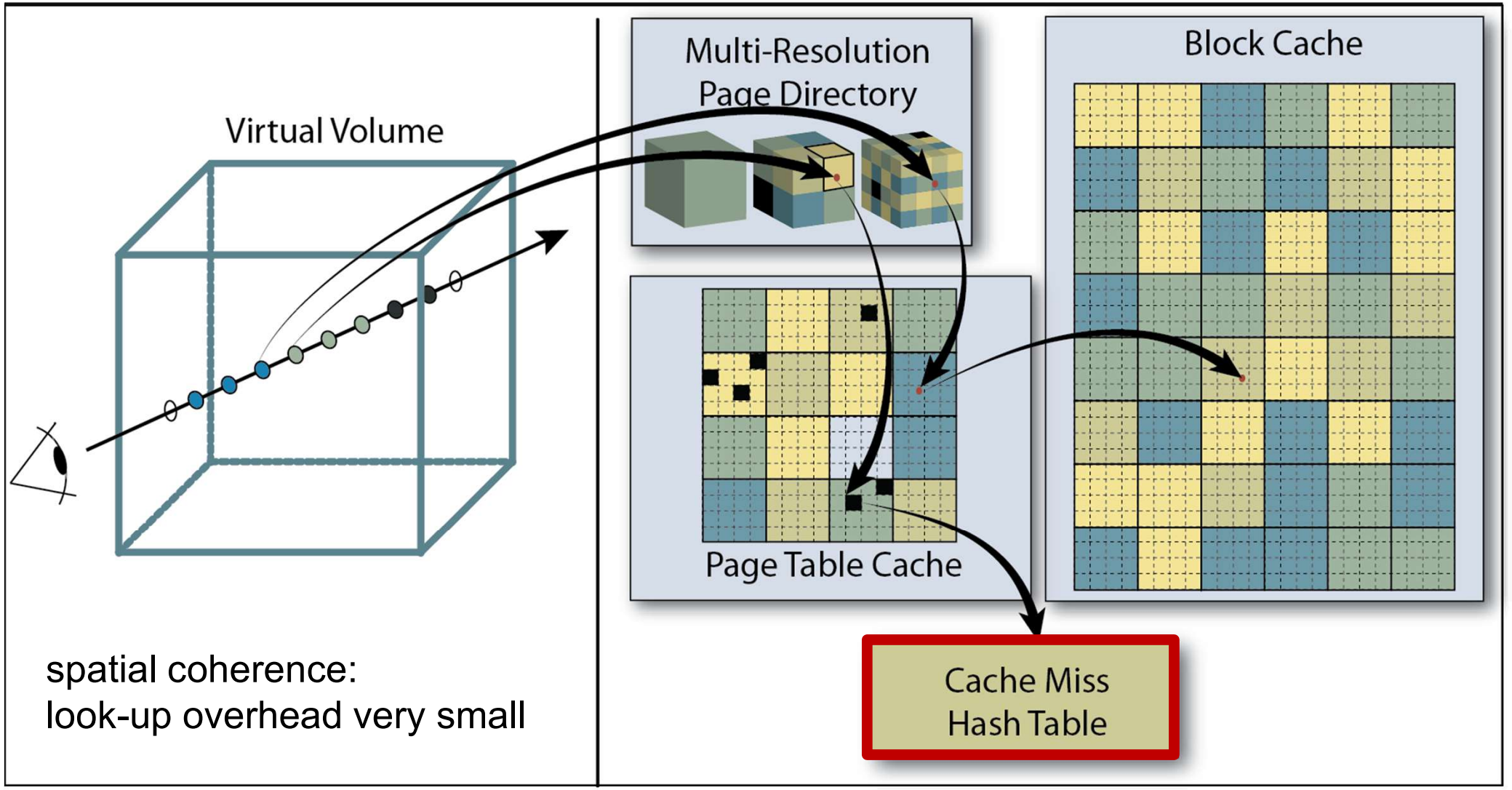
**`http://dx.doi.org/10.1109/TVCG.2012.240`**

# Petascale Volume Rendering




multi-resolution
page directory

# Petascale Volume Rendering




spatial coherence:
look-up overhead very small

Thank you.