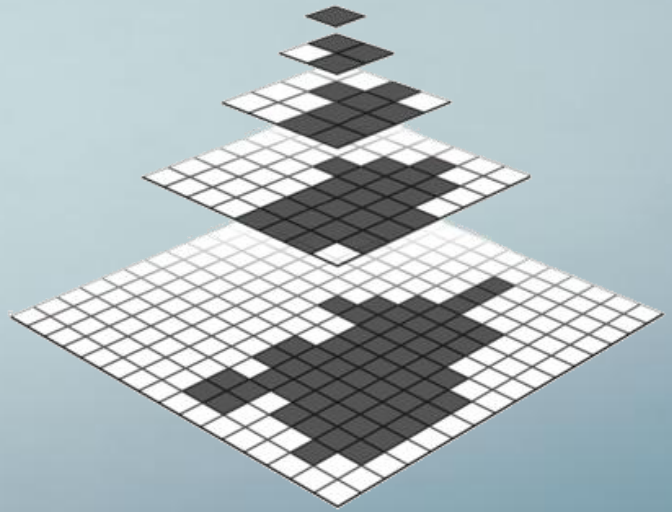
High Quality Software and Hardware Virtual Textures



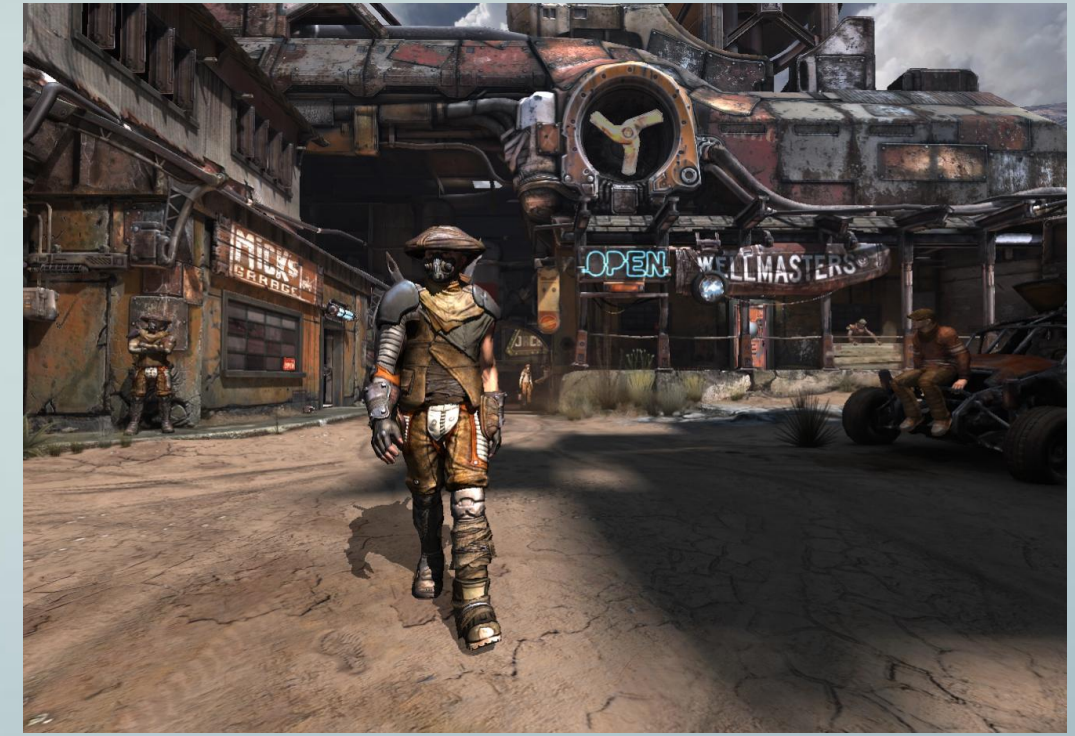
J.M.P. van Waveren
Lead Technology Programmer
id Software

SIGGRAPH2013

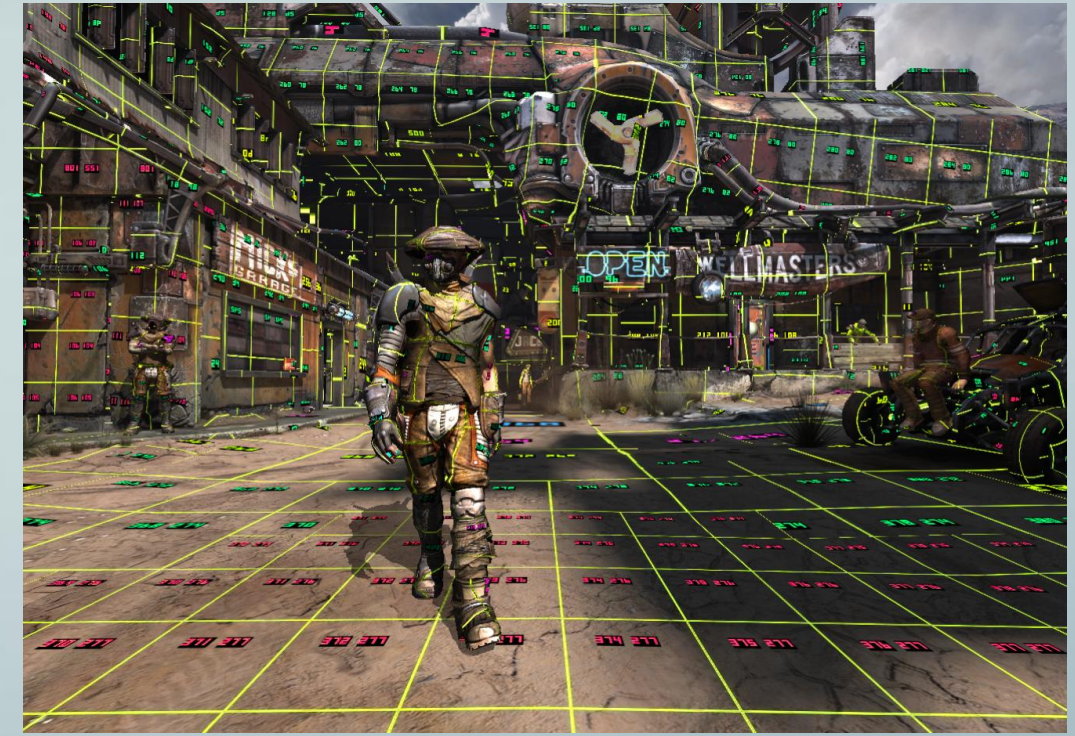
Software Virtual Textures

Solving issues and achieving high quality.

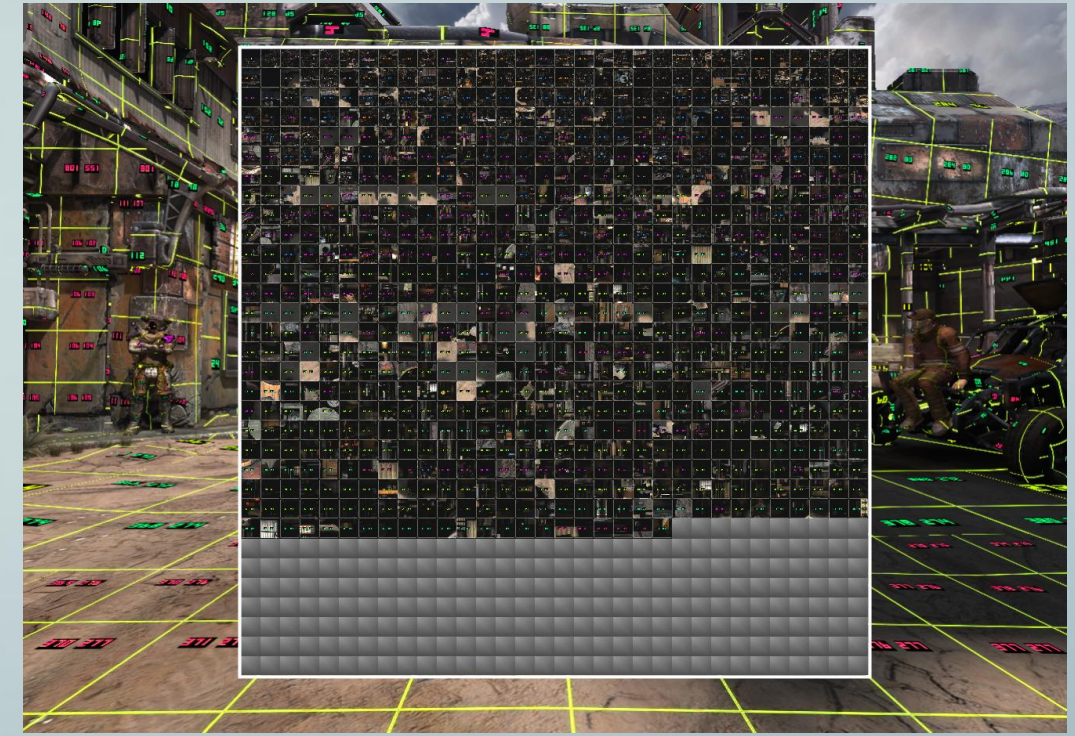
Software Virtual Textures in RAGE



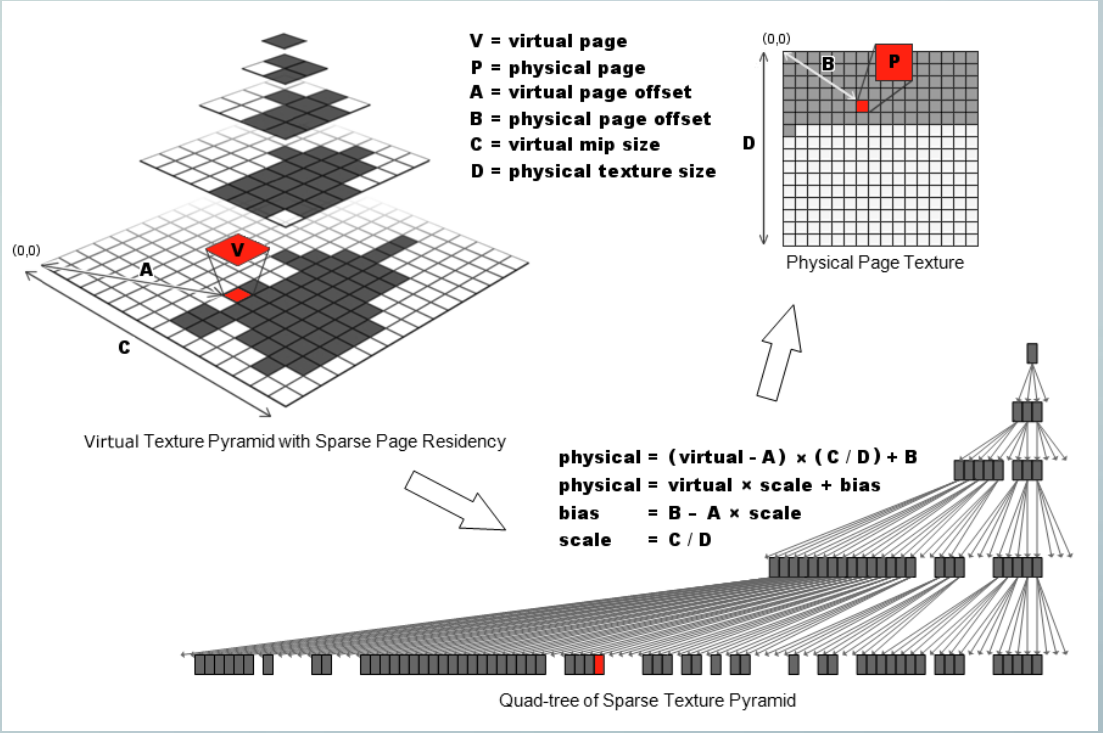
Software Virtual Textures in RAGE



Software Virtual Textures in RAGE



Address Translation



Page Table Texture

- Page table is typically a texture with one texel per virtual page where each texel stores a mapping from a virtual page to a physical page.
- A page table texture effectively stores the complete quad-tree whether pages are resident or not.
- The page table texture stores a mapping to the nearest resident coarser texture page for any virtual page that is not resident.
- By using a FP32x4 texture the page table can store the mapping from virtual to physical space as a simple scale and bias.
- Other, more memory efficient page tables (5:6:5 etc.) are typically used in practice.

Page Table Sampling Issue

- Page table must be point sampled.
  - Blending adjacent but independent mappings makes no sense.
- Hardware page table lookup unaware of anisotropic lookup that follows.
  - Texture LODs for point sampling and anisotropic sampling are different.
- Typically end up with mapping to texture page that is too coarse.
- Not enough texture detail for the anisotropic texture filter.

Page Table Sampling Solution

- RAGE uses fixed page table LOD bias of “- log2(maxAniso)”.
- Results in appropriate detail on surfaces at oblique angle to view with maximized sample footprint (anisotropic).
- Causes aliasing on surfaces orthogonal to view with minimized sample footprint (isotropic).
- Real solution is to calculate correct page table LOD in fragment program based on anisotropy.

Page Table LOD Calculation

```
const float maxAniso = 4;
const float maxAnisoLog2 = log2( maxAniso );
const float virtPagesWide = 1024;
const float pageWidth = 128;
const float pageBorder = 4;
const float virtTexelsWide = virtPagesWide * ( pageWidth - 2 * pageBorder );

vec2 tc = virtCoords.xy * virtTexelsWide;
vec2 dx = dFdx( tc );
vec2 dy = dFdy( tc );

float px = dot( dx, dx );
float py = dot( dy, dy );

float maxLod = 0.5 * log2( max( px, py ) ); // log2(sqrt()) = 0.5*log2()
float minLod = 0.5 * log2( min( px, py ) );
float anisoLOD = maxLod - min( maxLod - minLod, maxAnisoLog2 );
```

Page Table LOD Calculation

- Equivalent to “textureQueryLod()”.
- However, “textureQueryLod()” uses a texture.
- There is no virtual texture.
- Use page table texture instead.
- Page table texture “page payload” times smaller than virtual texture.
- Must bias “textureQueryLod()” result with “log2(page payload)”.

Anisotropic Filtering Issue

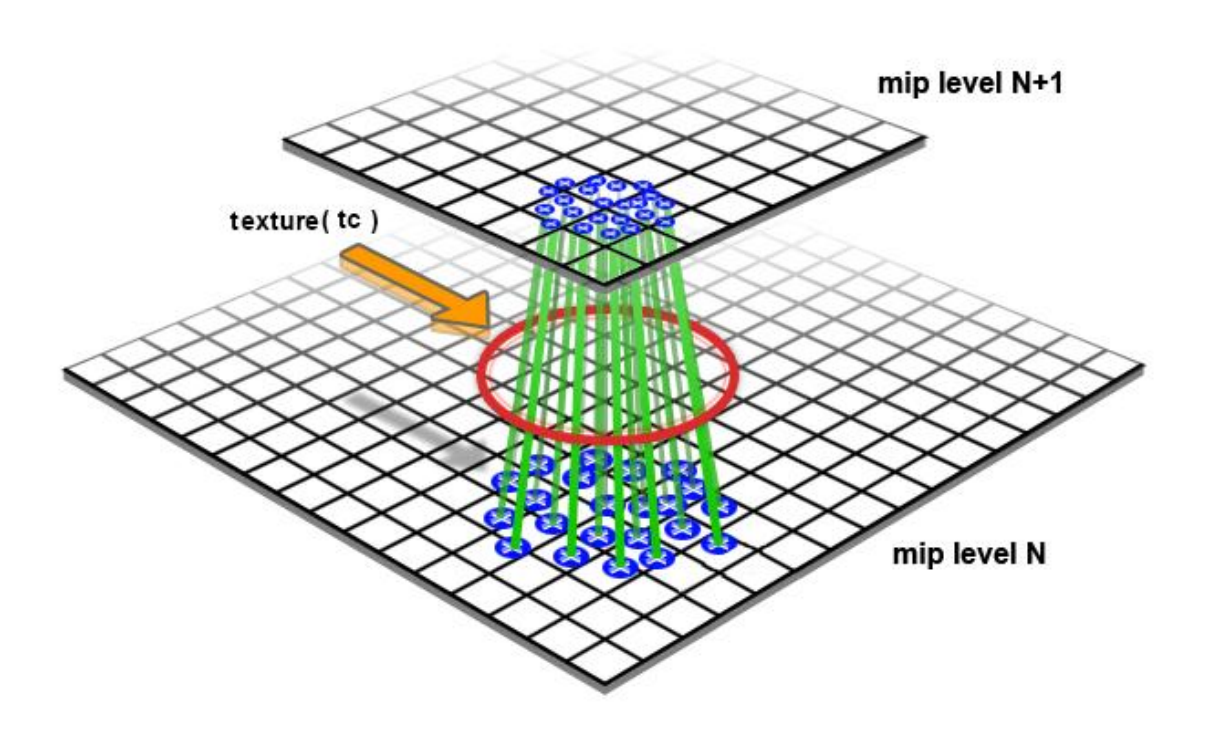
- RAGE uses physical textures without mip maps.
- Anisotropic filter uses single mip level (bilinear samples).
- Results in shimmering / aliasing.
- Does not allow gradually blending in detail when new texture page is made resident.

Anisotropic Filtering Solutions

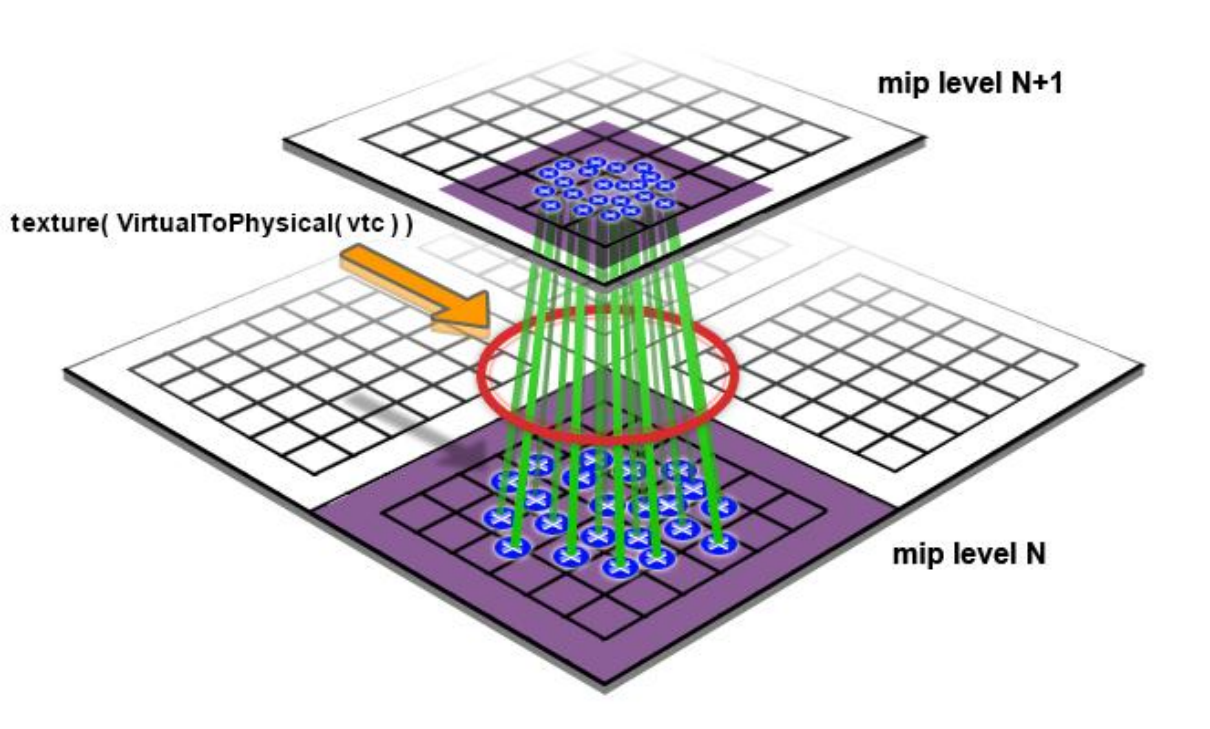
1. Add one mip level to the physical textures.

2. Use two virtual to physical translations and two physical texture lookups and blend between results.

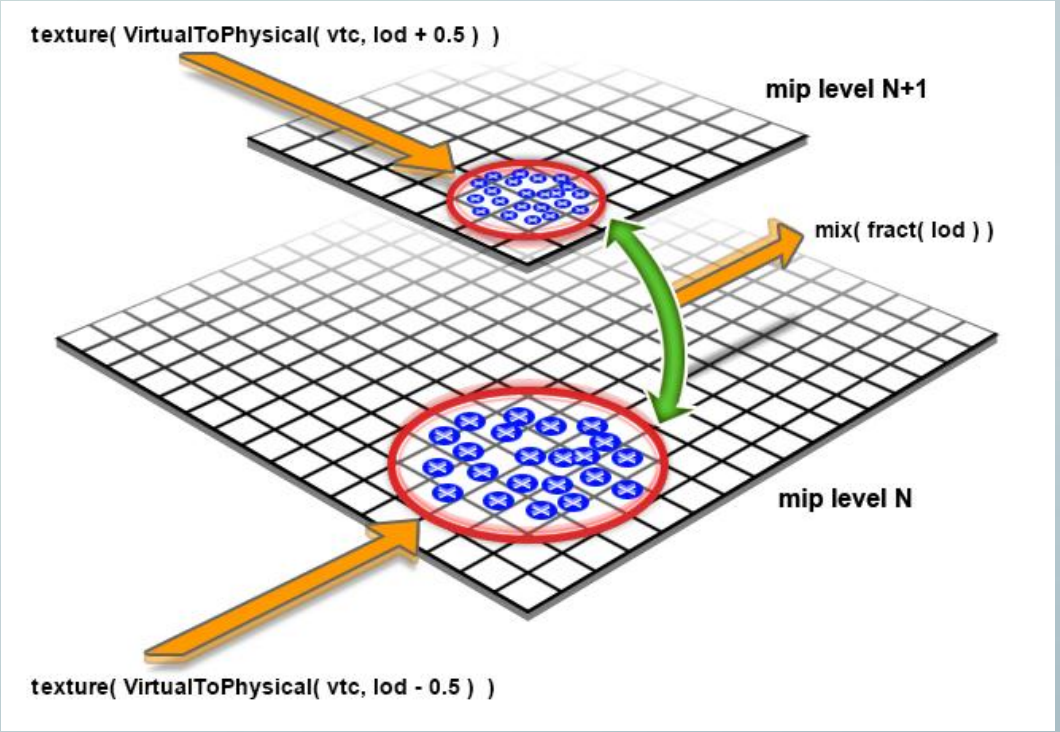
Normal Trilinear Anisotropic



Mip Mapped Physical Texture



Two Anisotropic Physical Lookups



Two Anisotropic Physical Lookups

```
vec4 scaleBias1 = textureLod( pageTable, virtCoords.xy, anisoLOD - 0.5 );
vec4 scaleBias2 = textureLod( pageTable, virtCoords.xy, anisoLOD + 0.5 );

vec2 physCoords1 = virtCoords.xy * scaleBias1.xy + scaleBias1.zw;
vec2 physCoords2 = virtCoords.xy * scaleBias2.xy + scaleBias2.zw;

vec4 color1 = texture( physicalTexture, physCoords1 );
vec4 color2 = texture( physicalTexture, physCoords2 );

color = mix( color1, color2, fract( anisoLOD ) );
```

Texture Popping Issue

- Delay between texture page needed for rendering and residency.
- Even with a highly optimized pipeline a delay may be unavoidable:
  - *Texture data may be streamed from hard disk, optical disk, Internet etc.*
  - *Texture data may need to be transcoded.*
- Unpleasant “pop” when delayed texture page suddenly becomes resident.

Texture Popping Solutions

1. Predict required texture pages well ahead of time.
   - *Hard to predict visible texture data in interactive environment.*
   - *Highly variable delays (optical disk seek times, Internet lag).*
2. Gradually blend in delayed texture pages.
   - *Far less distracting than sudden “pop”.*

Clamp LOD with Min-LOD Texture

```
const float maxVirtMipLevels = 16;

float clampLOD = texture( minLodTexture, virtCoords.xy ).x * maxVirtMipLevels;

anisoLOD = max( anisoLOD, clampLOD );
```

Software Virtual Texture Sampling

```
uniform  sampler2D pageTable;          // RGBA-FP32  - { scaleS, scaleT, biasS, biasT }
uniform  sampler2D minLodTexture;      // R-8        - { minimum-LOD }
uniform  sampler2D physicalTexture;    // RGBA-8     - { red, green, blue, alpha }

in vec4 virtCoords;   // virtual texture coordinates
out vec4 color;       // output color

void main()
{
```

Calculate Page Table LOD

Clamp LOD with Min-LOD Texture

Trilinear Anisotropic Filtering

```
}
```

```
const float maxAniso = 4;
const float maxAnisoLog2 = log2( maxAniso );
const float virtPagesWide = 1024;
const float pageWidth = 128;
const float pageBorder = 4;
const float virtTexelsWide = virtPagesWide * ( pageWidth - 2 * pageBorder );

vec2 tc = virtCoords.xy * virtTexelsWide;
vec2 dx = dFdx( tc );
vec2 dy = dFdy( tc );

float px = dot( dx, dx );
float py = dot( dy, dy );

float maxLod = 0.5 * log2( max( px, py ) );  // log2(sqrt()) = 0.5*log2()
float minLod = 0.5 * log2( min( px, py ) );
float anisoLOD = maxLod - min( maxLod - minLod, maxAnisoLog2 );
```

```
const float maxVirtMipLevels = 16;
float clampLOD = texture( minLodTexture, virtCoords.xy ).x * maxVirtMipLevels;
anisoLOD = max( anisoLOD, clampLOD );
```

```
vec4 scaleBias1 = textureLod( pageTable, virtCoords.xy, anisoLOD - 0.5 );
vec4 scaleBias2 = textureLod( pageTable, virtCoords.xy, anisoLOD + 0.5 );

vec2 physCoords1 = virtCoords.xy * scaleBias1.xy + scaleBias1.zw;
vec2 physCoords2 = virtCoords.xy * scaleBias2.xy + scaleBias2.zw;

vec4 color1 = texture( physicalTexture, physCoords1 );
vec4 color2 = texture( physicalTexture, physCoords2 );

color = mix( color1, color2, fract( anisoLOD ) );
```

Hardware Virtual Textures

Also known as Partially Resident Textures (PRTs).

AMD_sparse_texture

Integration, solving issues and achieving high quality.

Hardware Virtual Texture Sampling

```
uniform  sampler2D virtualTexture;   // RGBA-8    - { red, green, blue, alpha }

in vec4 virtCoords;   // virtual texture coordinates
out vec4 color;       // output color

void main()
{
      int code = sparseTexture( virtualTexture, virtCoords.xy, color );
}
```

Fall Back To Resident Texture Data

```
if ( !sparseTexelResident( code ) ) {
    float sampleLOD = textureQueryLod( virtualTexture, virtCoords.xy ).x;
    for ( sampleLOD = ceil( sampleLOD ); sampleLOD <= 8.0; sampleLOD += 1.0 ) {
        code = sparseTextureLod( virtualTexture, virtCoords.xy, sampleLOD, color );
        if ( sparseTexelResident( code ) ) {
            break;
        }
    }
}
if ( sampleLOD > 8.0 ) {
    color = vec4( 0, 0, 0, 0 );
}
```

Hardware Virtual Texture Sampling

```
uniform  sampler2D minLodTexture;    // R-8        - { minimum-LOD }
uniform  sampler2D virtualTexture;   // RGBA-8     - { red, green, blue, alpha }

in vec4 virtCoords;   // virtual texture coordinates
out vec4 color;       // output color

void main()
{
```

Calculate Desired LOD

Clamp LOD with Min-LOD Texture

Trilinear Anisotropic Texture Fetch

Fall Back To Resident Data

```
}
```

```
float anisoLOD = textureQueryLod( virtualTexture, virtCoords.xy ).x;
```

```
const float maxVirtMipLevels = 16;
float clampLOD = texture( minLodTexture, virtCoords.xy ).x * maxVirtMipLevels;
anisoLOD = max( anisoLOD, clampLOD );
```

```
int code = sparseTextureLod( virtualTexture, virtCoords.xy, sampleLOD, color );
```

```
if ( !sparseTexelResident( code ) ) {
   for ( sampleLOD = ceil( sampleLOD ); sampleLOD <= 8.0; sampleLOD += 1.0 ) {
      code = sparseTextureLod( virtualTexture, virtCoords.xy, sampleLOD, color );
      if ( sparseTexelResident( code ) ) {
         break;
      }
   }
}
if ( sampleLOD > 8.0 ) {
   color = vec4( 0, 0, 0, 0 );
}
```

Borderless Texture Pages

- Software virtual textures perform virtual to physical address translation before sampling.
  - *Software virtual texture pages need borders because texture unit samples a single page.*
- Hardware virtual textures perform virtual to physical translation during sampling.
  - *Hardware virtual texture pages do not need borders because texture unit can sample from multiple pages.*

Borderless Texture Page Issue

- RAGE stores texture pages with borders on disk because it significantly simplifies the pipeline.
- Need to support both software and hardware virtual textures because not all hardware supports PRTs.

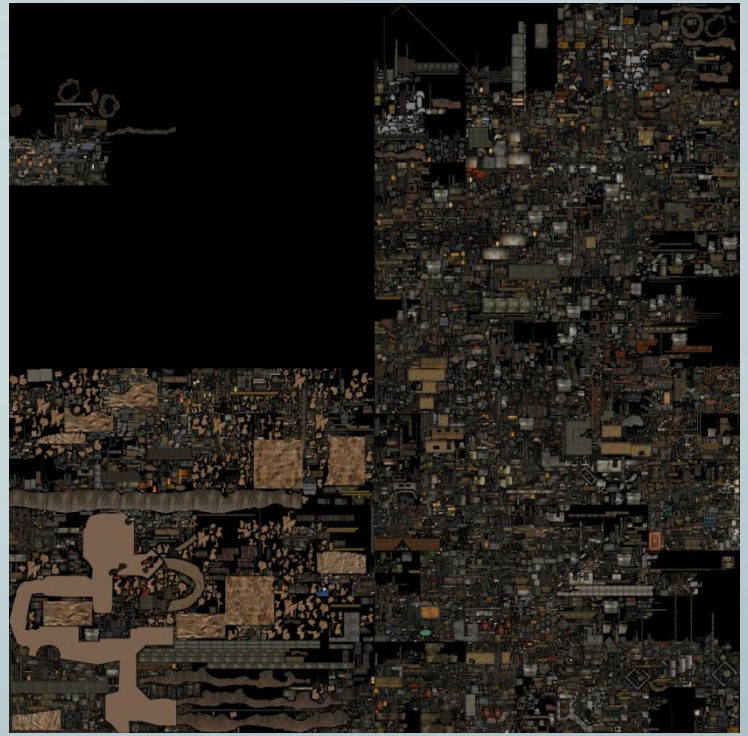
Borderless Texture Page Solutions

1. Ship two virtual textures, one with and one without borders.
   That's a whole lot of data.
2. Strip borders at run-time by upsampling 120 payload to 128.
   Non-integer up-sampling ratio causes noticeable blurring.
3. Composite borderless pages from multiple pages with borders at run-time (or vice versa).
   Complicates the pipeline and introduces significant overhead.
4. Convert virtual textures with borders to ones without borders at install time.
   De-re-compressing texture data may introduce additional compression artifacts.

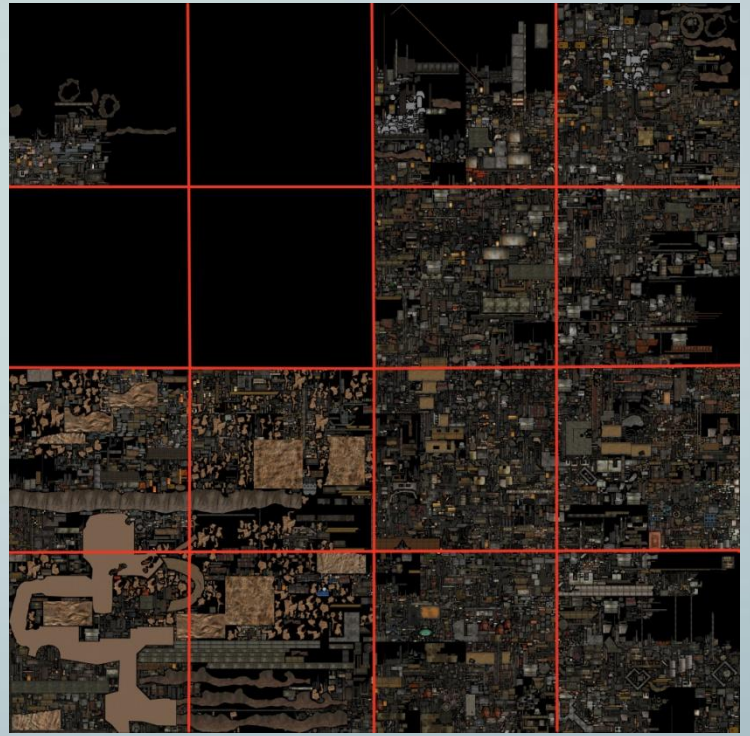
Hardware Virtual Texture Size Limit

- Floating-point precision limits software virtual texture sizes but they can go up to 256k x 256k texels (and beyond).
- Hardware virtual textures are currently limited to 16k x 16k texels.
  - DirectX limits textures to 16k x 16k texels.
  - DirectX requires 8-bits of sub-texel and sub-mip precision on texture filtering.

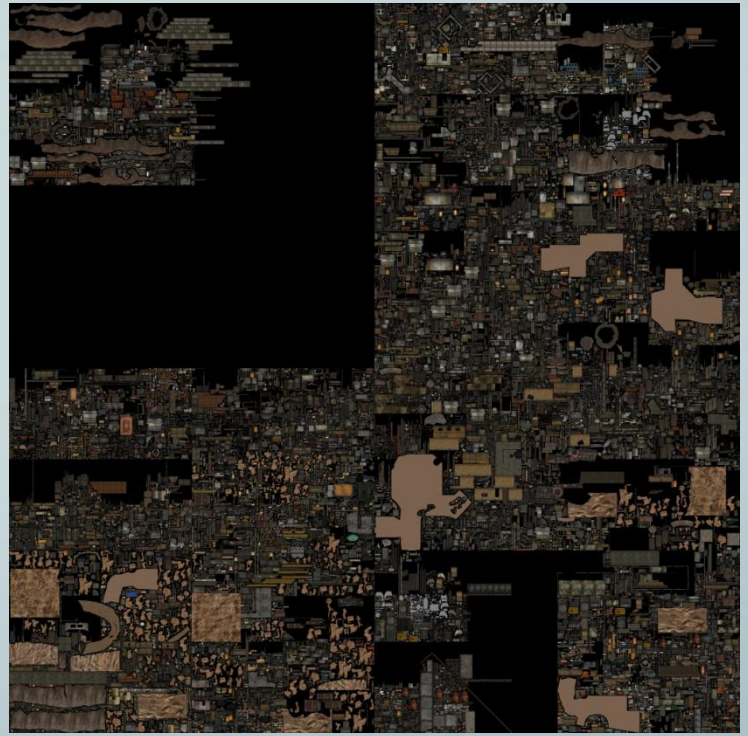
Example 64k x 64k Virtual Texture



Partially Resident Texture Array



Split Texture Islands > 16k



?

Texture Array Coordinate Calculation

```
// convert 120-texel + border texture coordinates to 128-texel borderless ones
vec2 borderlessCoords = virtCoords.xy * 120.0 / 128.0;

// scale coordinates such that the fractional part addresses a single layer of the texture array
const float widthInPRTs = 4;
float2 layerCoords = borderlessCoords.xy * widthInPRTs;

// split the coordinates into a texture array index and layer coordinates
vec3 arrayCoords;
arrayCoords.xy = fract( layerCoords.xy );
arrayCoords.z = floor( layerCoords.y ) * widthInPRTs + floor( layerCoords.x );
```

Hardware Virtual Texture Array Sampling

```
uniform  sampler2D minLodTexture;         // R-8          - { minimum-LOD }
uniform  sampler2DArray virtualTexture;   // RGBA-8   - { red, green, blue, alpha }

in vec4 virtCoords;   // virtual texture coordinates
out vec4 color;       // output color

void main()
{
```

Calculate Texture Array Coordinates

Calculate Desired LOD

Clamp LOD with Min-LOD Texture

Trilinear Anisotropic Texture Fetch

Fall Back To Resident Data

```
}
```

```
vec2 borderlessCoords = virtCoords.xy * 120.0 / 128.0;

const float widthInPRTs = 4;
float2 layerCoords = borderlessCoords.xy * widthInPRTs;

vec3 arrayCoords;
arrayCoords.xy = fract( layerCoords.xy );
arrayCoords.z = floor( layerCoords.y ) * widthInPRTs + floor( layerCoords.x );
```

```
float anisoLOD = textureQueryLod( virtualTexture, arrayCoords.xy ).x;
```

```
const float maxVirtMipLevels = 16;
float clampLOD = texture( minLodTexture, borderlessCoords.xy ).x * maxVirtMipLevels;
anisoLOD = max( anisoLOD, clampLOD );
```

```
int code = sparseTextureLod( virtualTexture, arrayCoords.xyz, sampleLOD, color );
```

```
if ( !sparseTexelResident( code ) ) {
   for ( sampleLOD = ceil( sampleLOD ); sampleLOD <= 8.0; sampleLOD += 1.0 ) {
      code = sparseTextureLod( virtualTexture, arrayCoords.xyz, sampleLOD, color );
      if ( sparseTexelResident( code ) ) {
         break;
      }
   }
}
if ( sampleLOD > 8.0 ) {
   color = vec4( 0, 0, 0, 0 );
}
```

Hardware Virtual Texture Page Sizes

- PRT pages do not have a fixed size in texels.
- PRT pages have a fixed size in memory.
- On current AMD hardware the PRT pages are always 64 kB.

Format	Size in Texels
uncompressed RGBA-8	128 x 128 texels
DXT5/BC3 compressed	256 x 256 texels
DXT1/BC1 compressed	512 x 256 texels

Supporting Different Page Sizes

- Support for uncompressed and compressed PRTs is desirable.
- Virtual texture page size on disk is 128 x 128.
- Maps directly to an uncompressed PRT page.
- Integer multiple of on disk pages used to create a compressed PRT page.

PRT Page Management

- TexSubImage used to both simultaneously upload texture data and update page tables.
- Need the texture data before calling TexSubImage.
- Only after TexSubImage was called you know whether physical memory was available.
- Getting the texture data ready may require significant effort (streaming, transcoding etc.) only to find out no more physical memory is available.

PRT Page Management

- Undesirable to drop page or free up memory last minute after TexSubImage fails.
- Need to know if physical memory is available first so memory can be freed up early on.
- Extensions to separate page table update from texture page allocation + population are being worked on.

Are PRTs worth the trouble?

- PRTs do not need pages with borders.
  - *Simplifies things everywhere.*
- The number of resident PRT pages is not limited by the size of a physical textures.
  - *All available video memory can be used.*
- PRTs support proper high quality texture filtering.
  - *The anisotropic footprint not limited by page border size.*

Increasing Texture Detail

- Uniquely textured worlds require a lot of storage and bandwidth.
- Can only reasonably ship so much detail to consumer (DVD, BluRay, Digital-download).

Solutions for More Texture Detail

1. Stream detail at run-time over the Internet.

   Must be online to experience virtual world.

2. Enhance detail with detail textures.

   Specialized form of texture compression.

   Limited variety and creation, selection, run-time cost.

3. Programmatically enhance texture detail.

   Virtual textures are well suited for efficient programmatic texture enhancement.

Virtual Texture Upsampling

- Allocate a software virtual texture or PRT much larger than the virtual texture stored on disk.
- Upsample a coarser texture page to populate a texture page for which no original content is stored on disk.
- As opposed to upsampling (or magnifying) for every pixel in a fragment program the cost is amortized by upsampling once when a page is made resident.
- Can use various interesting upsampling algorithms. (bicubic, Sinc, sharpening, edge enhancing etc.)

Virtual Texture Upsampling Example

Virtual Texture Upsampling Example

Upsampling Avoids Bilinear Magnification

- Not relying on standard bilinear magnification because new texture data is generated as the viewer approaches a textured surface.
- Need less bits of sub-texel precision on texture filtering.
- This frees up bits that can be used to support larger textures.

Populating Virtual Textures

- Populating a virtual texture currently faces significant API overhead.
- In RAGE uploading texture data through the graphics driver may cost more than 6 milliseconds of CPU time per frame.
- Texture updates are also synchronized with rendering when instead they could happen completely asynchronously.

Direct Texture Access

- Unified memory architectures are the future.
- On the consoles we have had direct access to texture memory for years.
- Direct access allows virtual texture pages to be updated asynchronously.
- GPU texture caches may not be coherent with texture memory when directly writing to memory.
- Page table and texture updates can be spaced far apart and texture caches are typically flushed frequently enough.
- Extensions for direct texture access are being worked on.

Tiled Texture Formats

- Tiled formats are used for improved memory access patterns and performance.
- For direct texture access either use linear (non-tiled) or use known specified tiled format.
- Direct texture access does not allow arbitrary tiling changes for new hardware/drivers.
- Need standardized tiled formats.

More Information

- Course website
  - http://cesium.agi.com/massiveworlds/
- Contact us
  - Patrick Cozzi (@pjcozzi, pcozzi@agi.com)
  - Kevin Ring (kring@agi.com)
  - Emil Persson (@_Humus_, http://www.humus.name/)
  - Graham Sellers (@grahamsellers, graham.sellers@amd.com)
  - Jan Paul van Waveren (http://www.mrelusive.com)
- Come grab us right now!