ADAPTIVE VIRTUAL TEXTURE RENDERING

IN

FARCERY4

Ka Chen
Technique ArchitectUbisoft
Contents

1. Overview of Virtual Texture Techniques
2. Far Cry 4 Terrain
3. Adaptive Virtual Textures - AVT
4. Virtual Texture Rendering Challenges
5. Results, Performance, Summary
1. OVERVIEW OF VIRTUAL TEXTURE TECHNIQUES
Virtual Texturing

Extremely Large Virtual Texture

Indirection Texture

Physical Texture Cache

Virtual address -> Physical address

- Virtual texturing in games
  - Mega-Textures
  - Procedural Virtual Textures

GDC 2015
Mega-Textures

- Developed by id Software for Rage (*Waveren 2013*)
- Texture data is stored on disk and streamed to memory as required
- Runtime determines the required tiles (pages) and requests them from disk
- Tiles are loaded to a tile cache (physical texture cache) and the page table (indirection texture) is updated
Procedural Virtual Textures

- Used by DICE in Frostbite Engine for Battle Field3 (Widmark 2012)
- Splats terrain rendering into virtual textures at runtime
  - No highly compressed virtual textures from disk
  - Direct render into virtual texture for missing pages
- Leverages frame-to-frame coherency to reduce terrain rendering cost
- Powerful GPU optimization for terrain rendering
2. FAR CRY 4 TERRAIN
Far Cry 4 Terrain

- Cross platform (PS4, Xbox1, PC, PS3, Xbox 360)
- Large world: 10 x 10 KM
- Far terrain (Vista terrain > 300 meters away):
  - Offline baked geometry and textures
- Near terrain:
  - Rendered from height-map
  - 4 detail material layers blended with a mask texture
  - Road and decals add unique detail
  - Target resolution: 10 texels / cm
  - Could use virtual textures
For next gen platforms we want to add a massive number of procedurally placed decals.
• Simple deferred decals would be too expensive in this quantity
• So optimize by baking decals into a virtual texture at runtime
Procedural Virtual textures in Far Cry 4

Virtual texture
512K x 512K

Indirection texture
2K x 2K

Physical texture
9K x 9K

11 Virtual texture mips

11 Indirection texture mips

11 Virtual texture mips

GDC 2015
Indirection Texture Format

- Entry coordinate \((x, y)\):
  - Each entry represents one virtual page
  - Entry coordinate = Virtual page coordinate / virtual page size

- Entry content format: 32 bit integer

<table>
<thead>
<tr>
<th>PageOffsetX</th>
<th>PageOffsetY</th>
<th>Mip</th>
<th>Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

- \(\text{PageOffsetX} = \text{Physical page U Coordinate} / \text{physical page size}\)
- \(\text{PageOffsetY} = \text{Physical page V Coordinate} / \text{physical page size}\)
- Mip: Mip-map level of this page
- Debug: used for debugging only
  - (for example saving a frame counter)
Motivation

- With conventional virtual texturing
  - 512K x 512K Virtual Texture on 10 x 10 Km world

- Another technique is required.
  - **10 million x 10 million Virtual Texture**!

- 0.5 texel/cm resolution
- 10 texel/cm resolution
3. ADAPTIVE VIRTUAL TEXTURES
Adaptive Virtual Textures (AVT)

- Based on procedural virtual textures
- The 10x10KM world is divided into $64\times64$ meter sectors
- Near terrain sectors:
  - Allocate virtual images in the virtual texture
  - Nearer sectors: larger virtual images
    - $64K \times 64K$ ($64K / 64$ meter = 10 texels/cm)
  - Farther sectors: smaller virtual images
    - $32K \times 32K$
    - $16K \times 16K$
    - ...
    - $1K \times 1K$
Adaptive Virtual Textures (AVT)

- Allocate virtual images inside the virtual texture for all close sectors

Visualize the allocation of virtual images for close sectors inside virtual texture
- Each colored square represents one virtual image for each nearby sector

Camera frustum
Multiple sectors slightly further from camera
16K x 16K virtual images

2 sectors nearest the camera
64K x 64K virtual images

6 sectors further from camera
32K x 32K virtual images

Continue until all nearby sectors are allocated in the virtual texture
Upscale virtual image size when camera moves closer

32K x 32K -> 64K x 64K
Downscale virtual image size when camera moves further

$64K \times 64K \rightarrow 32K \times 32K$
Upscale a Virtual Image

- We allocate a larger virtual image in the virtual texture and remove the old one
  - In this example, $32K \times 32K \rightarrow 64K \times 64K$
Upscale a Virtual Image

- Terrain material blending with additional decals
  - Already cached in our physical texture cache

Shift and reuse them!
For all pages that are from mip 1 to mip 10, copy entries of indirection texture from old image to new image while shifting up 1 mip

Old virtual image: 32K x 32K

New virtual image: 64K x 64K
Upscale virtual image of sector 32K → 64 K

- Update all mip 1 entries in indirection texture for new virtual image

Do it for all pages in mip 1 in new virtual image

Update mip 2 – 10 pages in the similar way
Update mip 0 pages in indirection texture

- Need to handle mip 0 pages
  - They haven’t been rendered in the old image
Update mip 0 pages

- 4 mip 0 pages have 1 corresponding mip 1 page
  - Temporarily map to lower mip page
    - Images appear blurred in this frame
    - Will become sharper after correctly updated.
One entry content in this image:
{ PageOffset = (5, 2), Mip = 1 }

Copy entry content to 4 corresponding mip 0 entries
{ PageOffset = (5, 2), Mip = 1 }

Do it for all pages in mip 0 in new virtual image
Update mip 0 pages in indirection texture

- Copy indirection texture entries content

Indirection texture mip 0

Do not shift down mip!

- Physical UV is calculated according to the mip in the entry

Code snippet in terrain pixel shader

```c
scale = (virtual texture size / physical texture size) >> mip
bias = physical_page_offset - virtual_page_offset * scale
physical_uv = virtual_uv * scale + bias
```

- Physical page offset and mip are the entry content
Downscale a virtual image

- We allocate a smaller virtual image in the virtual texture and remove the old one
  - In this example, 64K x 64K -> 32K x 32K
- Reverse the steps of upscaling virtual image
Old virtual image: 64K x 64K
Mip 0

New virtual image: 32K x 32K
Mip 0

GDC 2015
4. Virtual Texture Rendering Challenges
Reduce memory for virtual page id buffer

- **Strategy**
  - Output page IDs and MIP levels
    - To a Read Write Buffer
    - During G-Buffer pass

- **Buffer format (32 bits)**
  - PageID X: 12
  - PageID Y: 12
  - Mip: 4
  - Size: 4

  - PageID XY = Virtual UV / Virtual Page Size
  - Size = log2 (Virtual Image Size)

- **Buffer size:** 1/8 of resolution of MRTs
Limit per frame rendering cost

- Caching virtual textures can be slow
  - Camera moves fast -> need to render a lot of pages
    - When driving a vehicle
    - Flying
Limit per frame rendering cost

- **Solution:** Distributed rendering
  - Sort required pages by mip levels
    - low -> high
  - Distribute the rendering of pages into multiple frames
Generate massive number of decals

- Artists want to generate many decals efficiently
- Solution: Procedural content generator
  - Automatically attach decals to specified objects
    - Leaves, stones, roots under trees
    - Cracks, dirt on road
    - Much more...
• Procedurally generated decal:
• Procedurally generated decal:

Attach to other procedurally generated content
For example generate fallen leaves under trees
Anisotropic Filtering

- Support 8x anisotropic in Far Cry 4
  - Texels in neighbor pages are not adjacent in world space
    - This could cause color bleeding
- Solution: Add 4 texels border to physical texture pages
  - Physical texture page: 264 x 264
  - Render page with 264 x 264 viewport
• Physical texture page: 264 * 264
• Can support 8x anisotropic filtering
Support trilinear filtering

- Only bilinear filtering
  - See seams where mip level is changing
- Far Cry 4 Solution: software trilinear filtering
  - Fetch virtual textures twice with \text{mip}(x) \text{ and } \text{mip}(x+1)
  - Calculate linear blending of fetched colors in shader
- Other solution: hardware trilinear filtering
  - Create ¼ size \text{mip} 1 \text{ cache}
  - Render into \text{mip} 1 \text{ cache} too
  - Hardware handle the blending between mips
  - 25% more cache memory
5. RESULTS, PERFORMANCE, SUMMARY
# Rendering Performance

<table>
<thead>
<tr>
<th>Performance (PS4)</th>
<th>CPU (rendering thread)</th>
<th>GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static scene (Cache primed)</td>
<td>0.2 ms</td>
<td>0.1 ms</td>
</tr>
<tr>
<td></td>
<td>Analyze PageID buffer</td>
<td>Write PageID buffer</td>
</tr>
<tr>
<td>Dynamic scene (Caching virtual textures)</td>
<td>0.5 – 1 ms</td>
<td>0.5 – 1 ms</td>
</tr>
<tr>
<td></td>
<td>Setup time for render</td>
<td>Render terrain and decals into virtual textures and compress to BC format textures</td>
</tr>
</tbody>
</table>
## Memory Consumption

<table>
<thead>
<tr>
<th>Memory</th>
<th>PageID Buffer</th>
<th>Indirection Texture</th>
<th>Physical Texture</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>0.4 MB</td>
<td>16 MB</td>
<td>202 MB</td>
<td>220 MB</td>
</tr>
</tbody>
</table>
Summary

- Procedural virtual texture is a good fit for terrain rendering
- Using AVT we can increase the resolution of the results
- Great when drawing a massive number of decals on Far Cry 4
Thanks to:

- Far Cry 4 rendering and arts team
- GDC reviewer : Mark Cerny
References

1. Software Virtual Textures (J.M.P. van Waveren 2013)
2. Advanced Virtual Texture Topics (Martin Mittring)
3. Terrain in Battlefield 3 (Mattias Widmark 2012)
4. Virtual Texturing in Software and Hardware
5. Virtual Texturing
References

1. Software Virtual Textures (J.M.P. van Waveren 2013)

2. Advanced Virtual Texture Topics (Martin Mittring)

3. Terrain in Battlefield 3 (Mattias Widmark 2012)

4. Virtual Texturing in Software and Hardware

5. Virtual Texturing

Thanks!