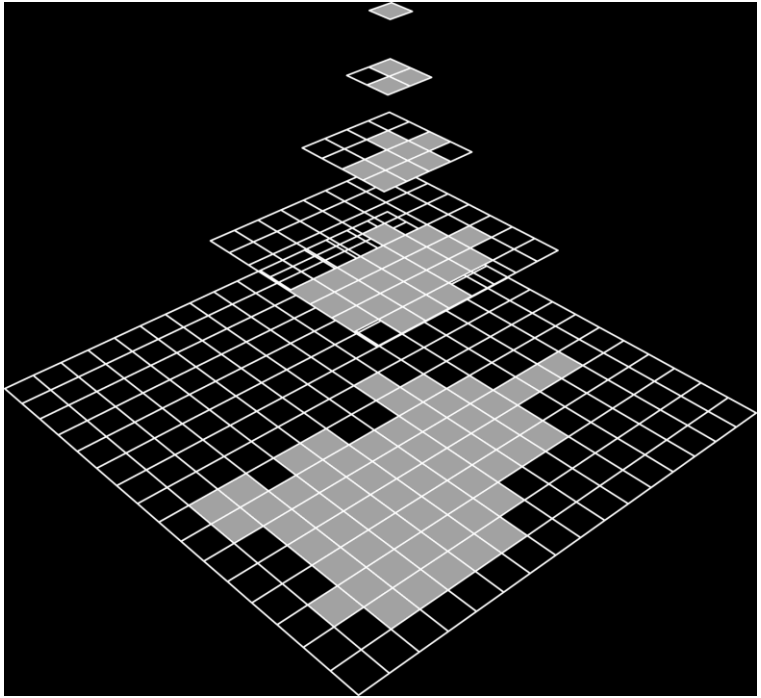
Real Virtual Texturing – Taking Advantage of DirectX11.2 Tiled Resources

Cem Cebenoyan
Developer Technology, NVIDIA

GAME DEVELOPERS CONFERENCE 2014
SAN FRANCISCO, CA
MARCH 17–21, 2014
EXPO DATES: MARCH 19–21

Overview

- Background
- API Overview
- Example Walkthrough
  - Sparse shadow maps



Background

- Virtual texturing techniques useful
  - eg Megatexture
- Suffer from a number of problems
  - Difficulty with filtering
  - Needs borders
  - Performance problems

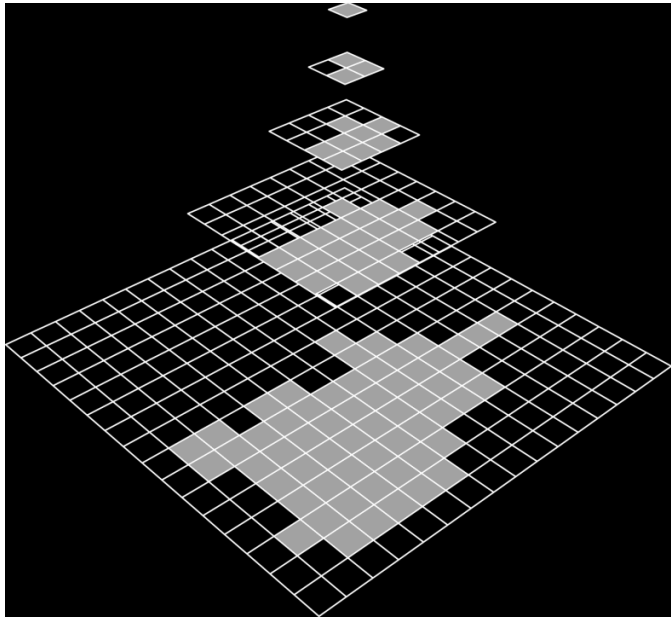
Enter Native HW Support

- But GPUs have had virtual memory for years!

- We can leverage that directly to support tiled / virtual GPU resources

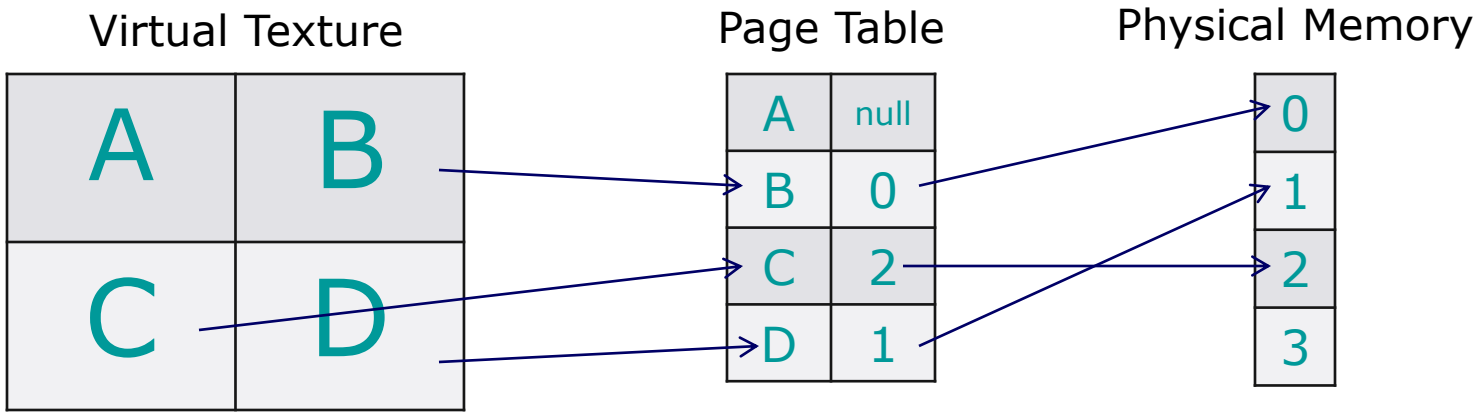
Tiled Resources

- Subdivide texture into a grid of tiles, allow some tiles to be “missing”
  - No physical memory is allocated for missing tiles
- Applications control tile residency
  - Can “map” and “unmap” tiles at run-time
  - Multiple concurrent mappings
- Implemented using virtual memory subsystem
  - Tiles correspond to VM pages



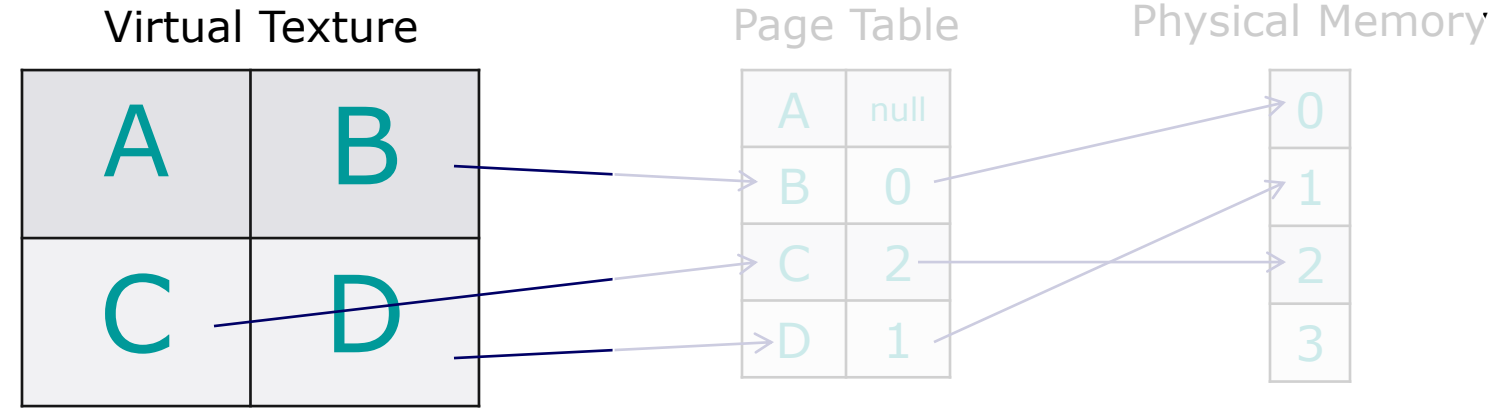
DirectX 11.2 Tiled Resources

- Looks like virtual memory:



Tiled Resources In Practice

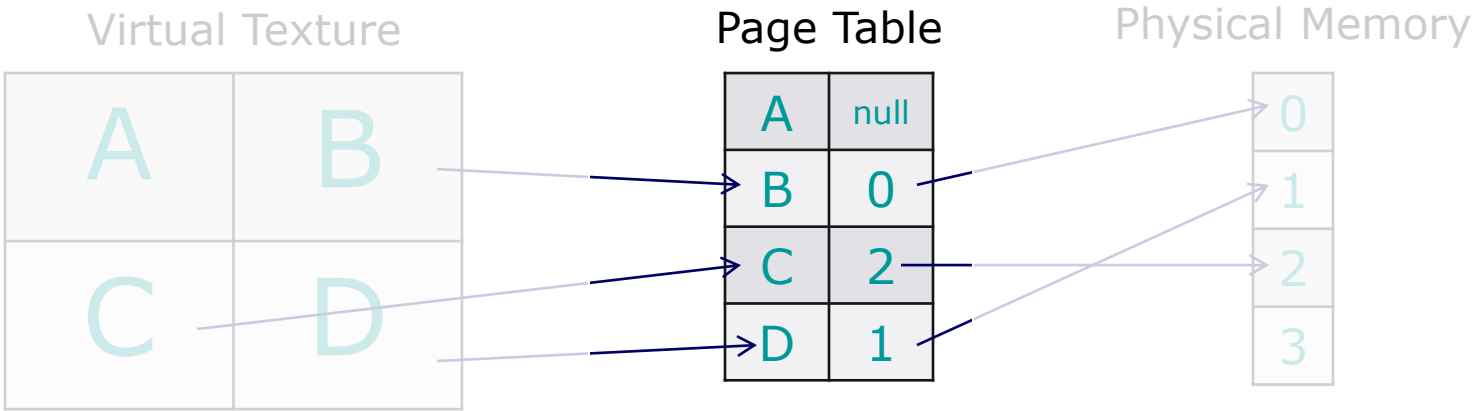
- Virtual texture is a texture or buffer with D3D11_RESOURCE_MISC_TILED flag



In D3D: *Tiled Resource (Texture2D or Buffer)*

Tiled Resources In Practice

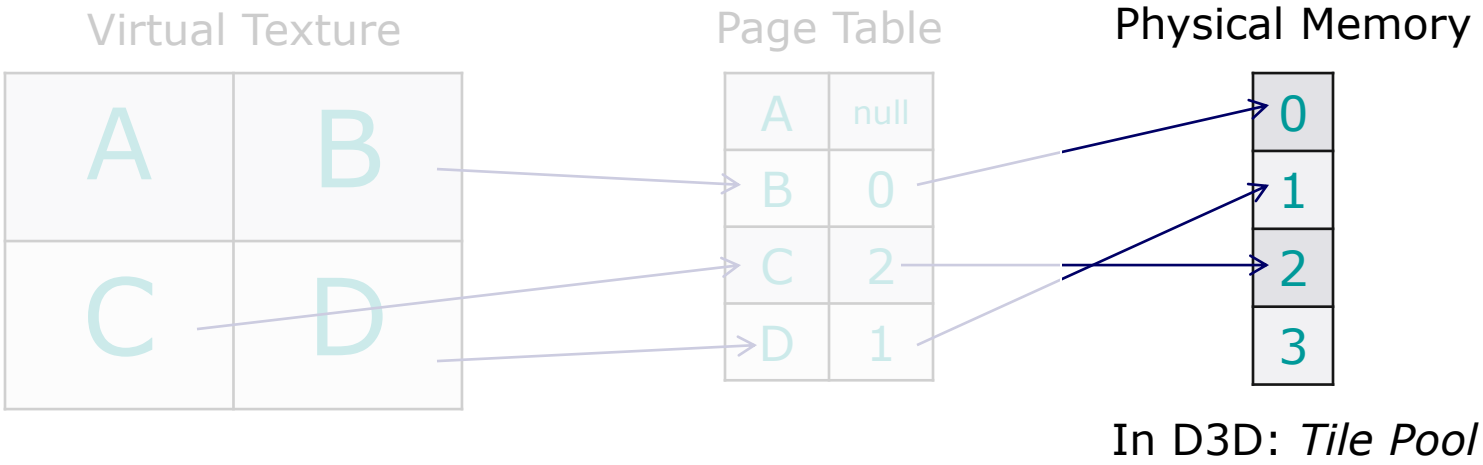
- Page table mappings are managed using UpdateTileMappings().



In D3D: *Tile Mappings*

Tiled Resources In Practice

- Physical memory is the Tile Pool, a buffer with D3D11_BUFFER_MISC_TILE_POOL



Checking Availability

- CheckFeatureSupport()
  - D3D11_FEATURE_D3D11_OPTIONS1 field
  - TiledResourcesTier subfield
  - NOT_SUPPORTED, TIER_1, or TIER_2

TIER_1

- Tiled Resource and Tile Pool creation supported
- Accessing (r/w) NULL mapped tiles has undefined behavior
  - Up to the user to define “default” tile and point all “unmapped” tile mappings to it
- Available on all AMD and NVIDIA hardware from the past few years

TIER_2

- Relaxes some restrictions
- Accessing NULL mapped tiles now defined to return zero
    - Writes to NULL mapped discarded
- Sample instructions for LOD clamp and getting feedback supported
- Available on newest and future hardware

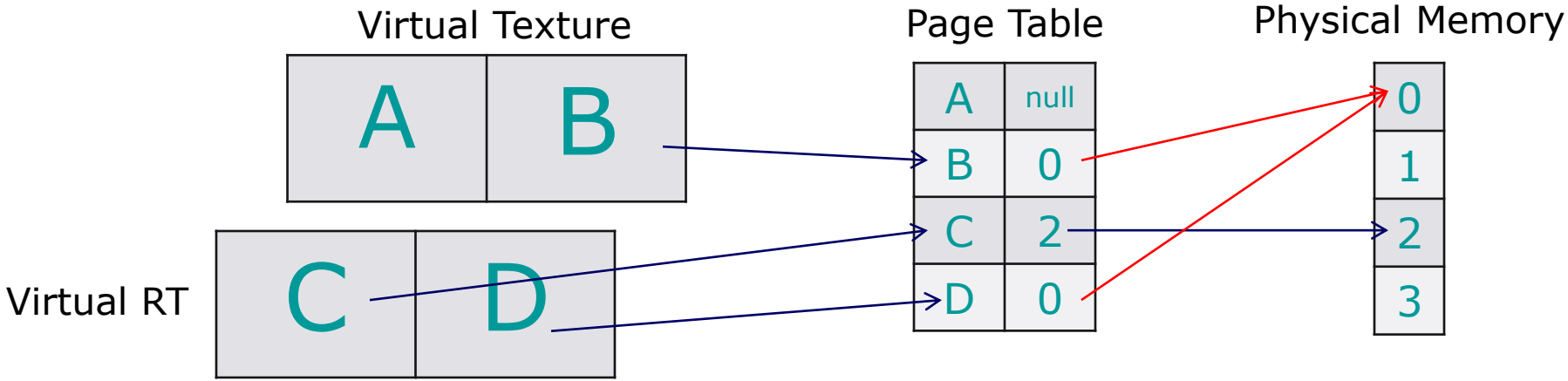
TIER_1 vs. TIER_2

	Tiled Resources	Tile Pool	LOD clamp Sample instruction	Feedback Sample instruction	NULL mapped behavior	Supported on all current hw?
TIER 1	√	√	x	x	undefined	√
TIER 2	√	√	√	√	Zero	x

- In general, almost all algorithms can be mapped to both tiers
  - For example, LOD clamp can be approximated with explicit LOD and gather4
  - Tier 2 generally just an optimization

Other API features

- ResizeTilePool()
  - Non-destructive
- TiledResourceBarrier()
  - Handle this case:



Plus / Minus over SW Solutions

- Plusses
  - All filtering modes just work
  - No borders necessary
  - Fast (virtual->physical translation in hw)
- Minuses
  - HW and OS limitations
  - But note TIER1 is supported by a ton of hw

Tile Shapes

- Tile size is fixed in *bytes*, not *texels*
  - Texture format determines tile shape in texels
  - Address mapping designed to keep tiles roughly square
- GPU pages are *64KB*
  - Implications for residency granularity

Texel format	Bytes per texel	Tile shape for 64KB pages, texels
RGBA8	4	128 x 128
RGBA16F	8	128 x 64
DXT1	0.5	512 x 256

Sparse Shadowmaps

- Ubiquitous shadow rendering technique
  - Used in virtually every game

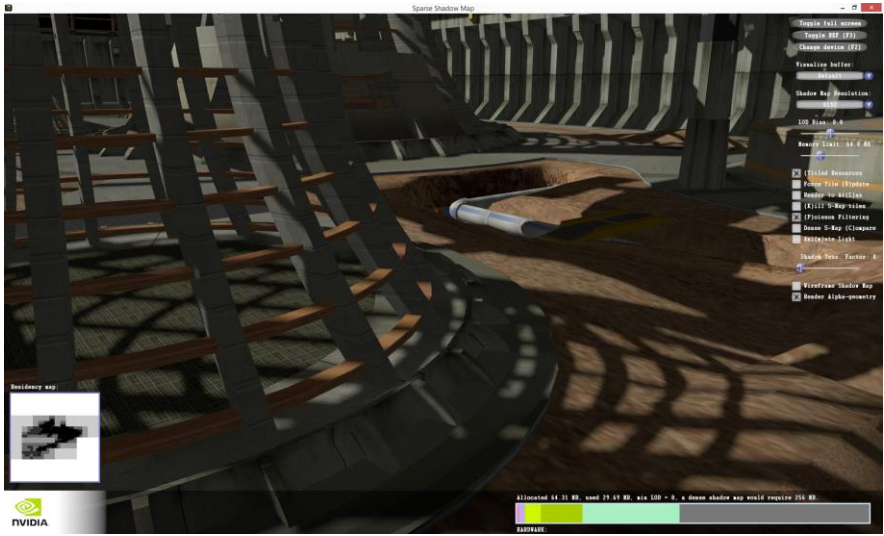
- Major problem: mismatch in sampling rates between image space and light space
  - Source of most aliasing problems

Existing Solutions

- Existing solutions
  - Creative transformations of the shadow map (PSM, TSM)
  - Divide-and-Conquer (CSM)
  - Exotic: resolution-matched shadowmaps, irregular Z-buffer

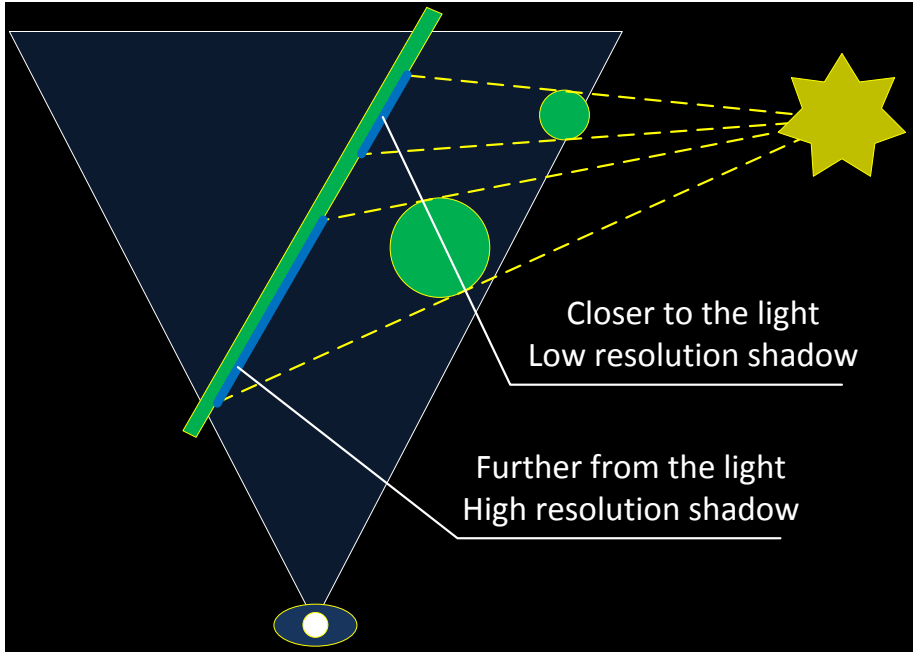
Sparse Shadowmaps

- Tiled texture support allows defining sparsely populated textures
  - Texture residency is controlled per-tile
- Can view mip-mapped sparse texture as a ***variable-resolution*** representation
  - Tiles missing at some level implies the data is presented at coarser LODs
- Provides finer-grained resolution control for shadow mapping

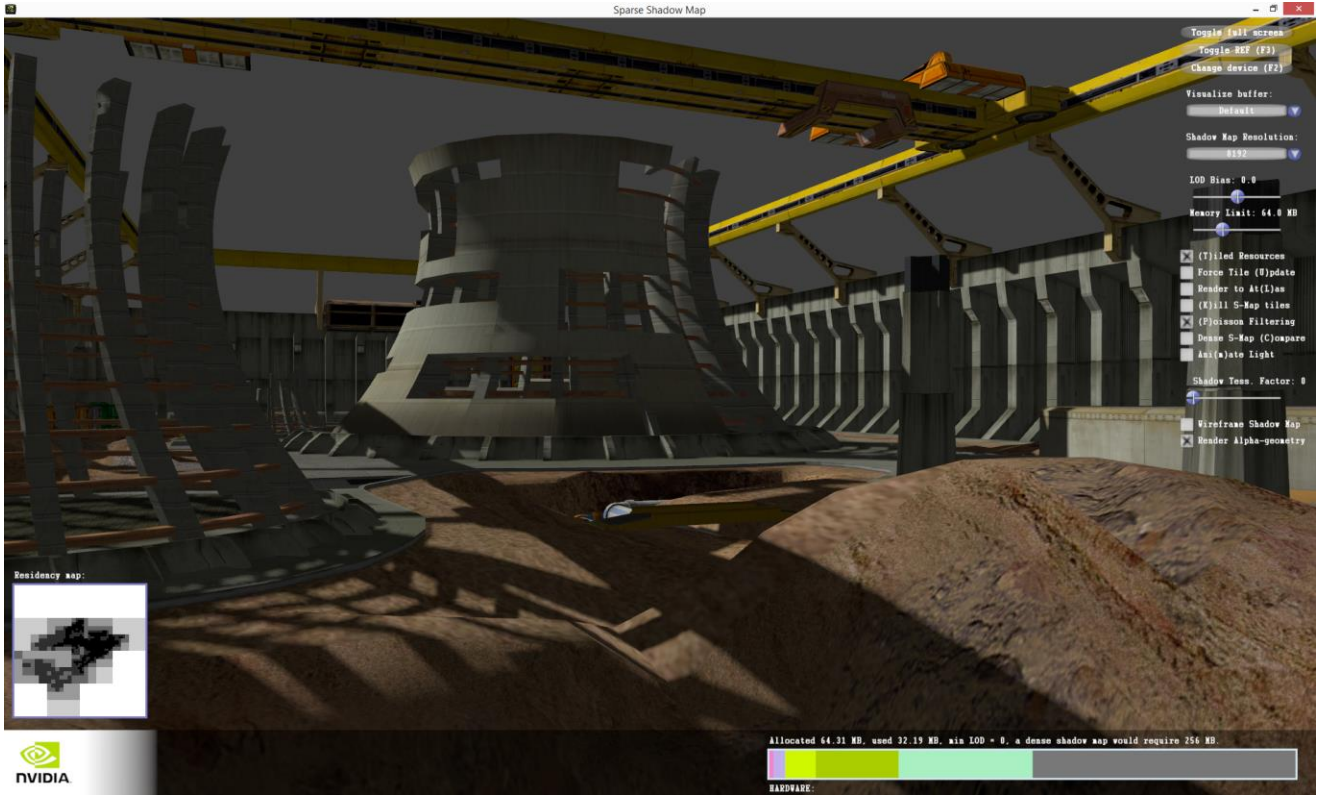


Sparse Shadow Maps

- Render the shadow map with non-uniform resolution
  - Resolution allocated dynamically, depending on the current frame needs
  - Shadow map represented by *sparsely populated MIP-chain*



Sparse ShadowMaps Demo!



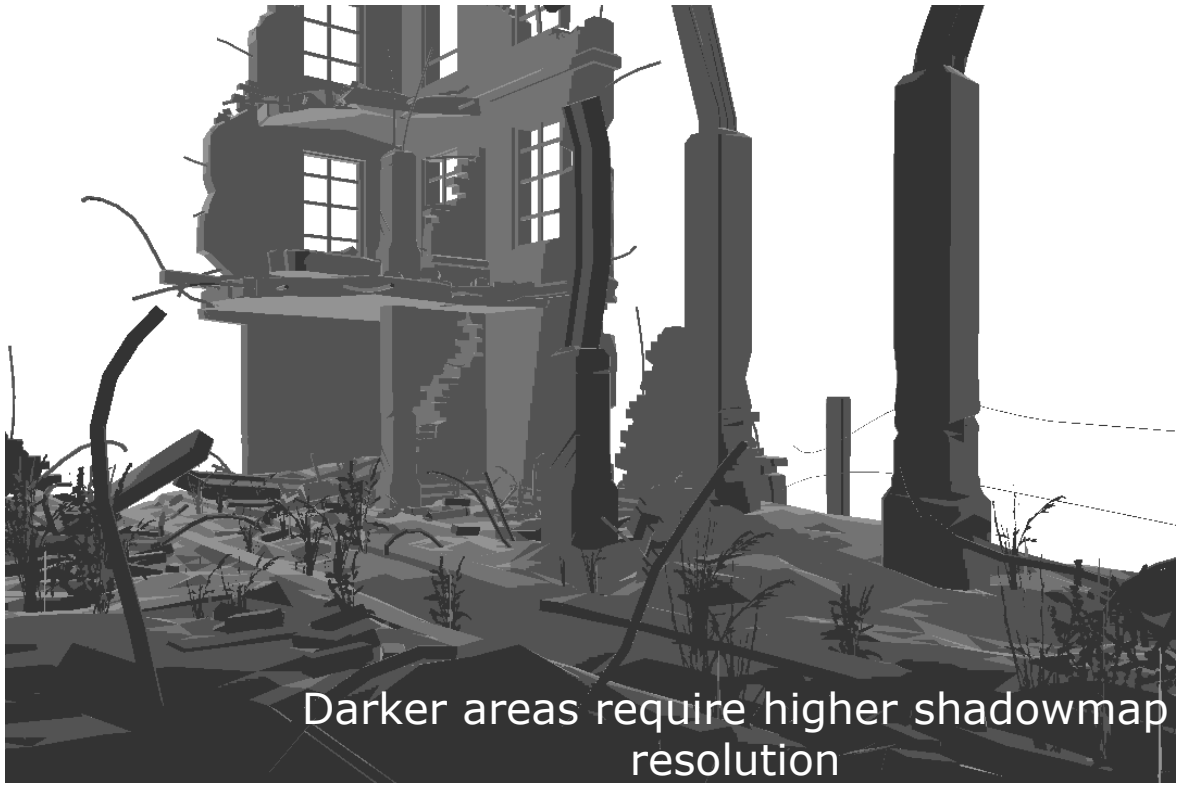
Algorithm Overview

1. Render pre-pass, determining shadow map LOD at each pixel
   - E.g. a separate channel in the G-buffer may be used to store the LOD
2. Build the *min LOD map* in shadow map space
   - Project screen-space per-pixel LODs to light space, compute min LOD per-tile
3. Create a sorted list of *tile allocation requests*
   - Sorted from coarse to fine LODs
4. Remap tiles from the tile pool
   - First N tiles from the *request queue,* N is the size of the pool
5. Render to the sparse shadow map
   - Broadcast geometry to multiple MIP levels, writes to unmapped tiles ignored
6. Shade using the sparse shadow map
   - Equivalent to other sparse texture usage

Algorithm Overview

1. Render pre-pass, determining shadow map LOD at each pixel
   - E.g. a separate channel in the G-buffer may be used to store the LOD
2. Build the *min LOD map* in shadow map space
   - Project screen-space per-pixel LODs to light space, compute min LOD per-tile
3. Create a sorted list of *tile allocation requests*
   - Sorted from coarse to fine LODs
4. Remap tiles from the tile pool
   - First N tiles from the *request queue,* N is the size of the pool
5. Render to the sparse shadow map
   - Broadcast geometry to multiple MIP levels, writes to unmapped tiles ignored
6. Shade using the sparse shadow map
   - Equivalent to other sparse texture usage

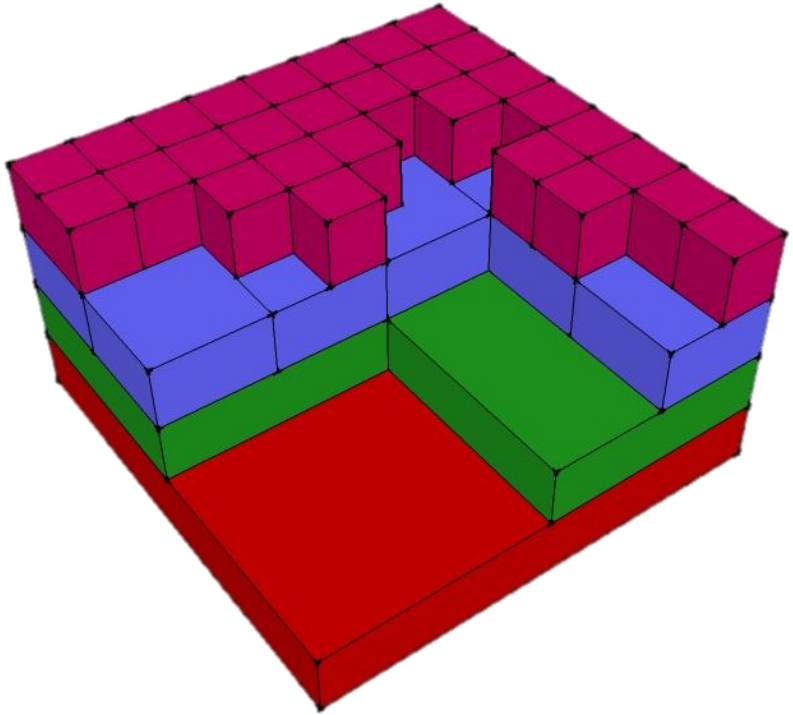
Required Shadowmap LOD



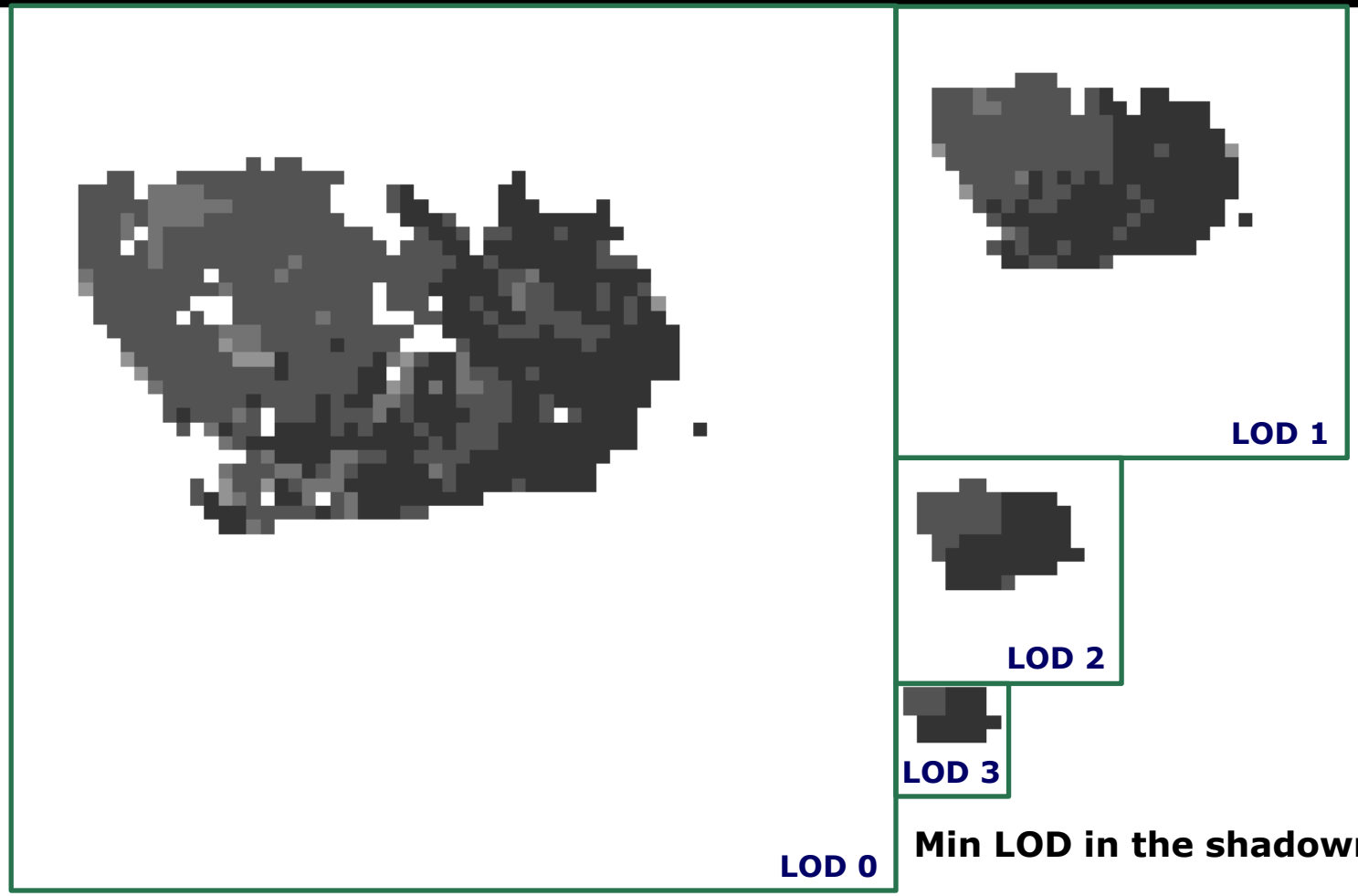
Algorithm Overview

1. Render pre-pass, determining shadow map LOD at each pixel
   - E.g. a separate channel in the G-buffer may be used to store the LOD
2. Build the *min LOD map* in shadow map space
   - Project screen-space per-pixel LODs to light space, compute min LOD per-tile
3. Create a sorted list of *tile allocation requests*
   - Sorted from coarse to fine LODs
4. Remap tiles from the tile pool
   - First N tiles from the *request queue,* N is the size of the pool
5. Render to the sparse shadow map
   - Broadcast geometry to multiple MIP levels, writes to unmapped tiles ignored
6. Shade using the sparse shadow map
   - Equivalent to other sparse texture usage

Sparse texture and min LOD map



0	0	0	0	0	0	0	0
0	0	0	1	0	0	1	1
0	0	1	1	2	2	2	2
0	0	0	1	2	2	2	2
0	0	0	0	3	3	3	3
0	0	0	1	3	3	3	3
0	0	1	1	3	3	3	3
0	0	1	1	3	3	3	3



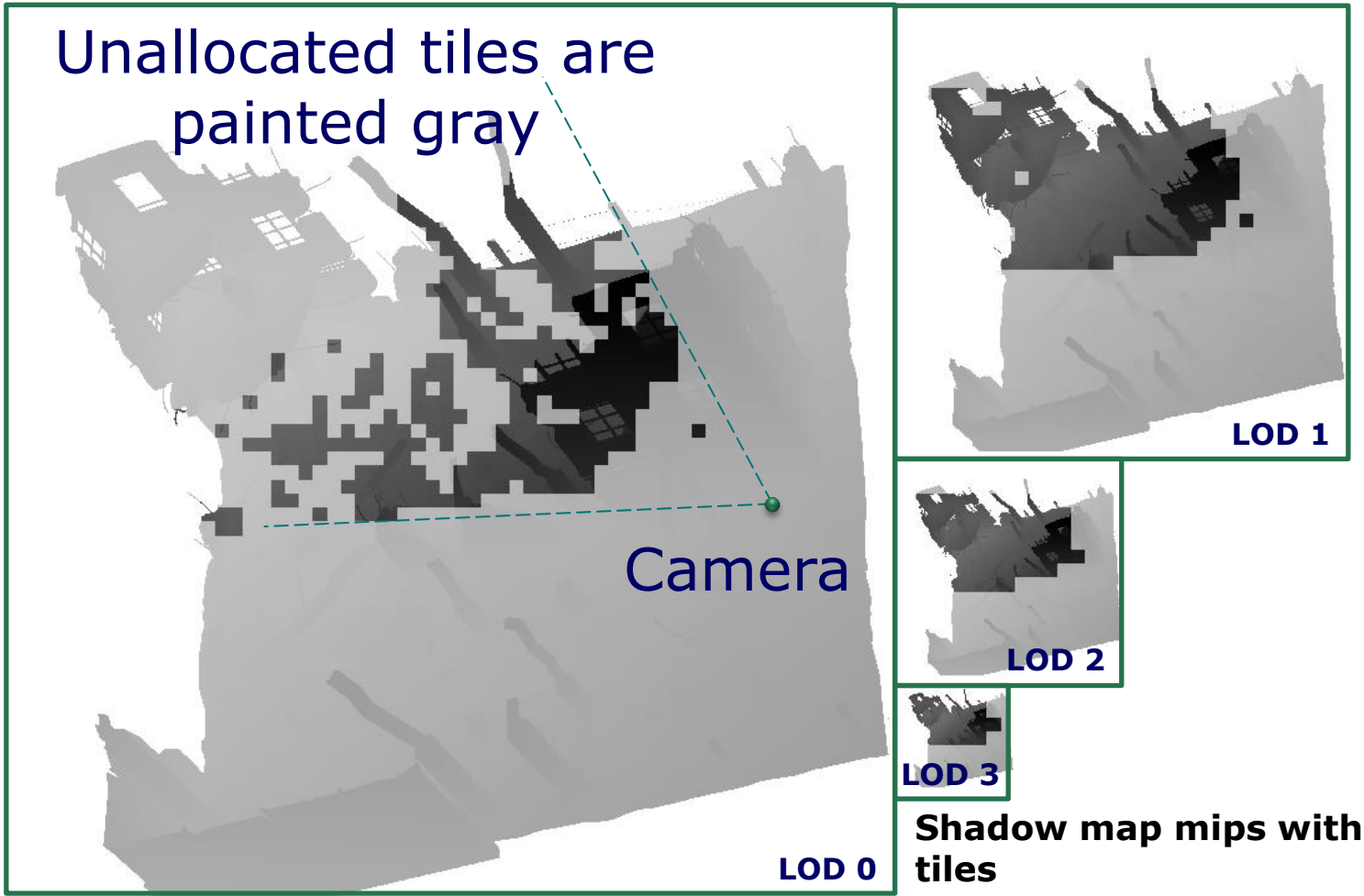
Min LOD in the shadowmap space

Algorithm Overview

1. Render pre-pass, determining shadow map LOD at each pixel
   - E.g. a separate channel in the G-buffer may be used to store the LOD
2. Build the *min LOD map* in shadow map space
   - Project screen-space per-pixel LODs to light space, compute min LOD per-tile
3. Create a sorted list of *tile allocation requests*
   - Sorted from coarse to fine LODs
4. Remap tiles from the tile pool
   - First N tiles from the *request queue,* N is the size of the pool
5. Render to the sparse shadow map
   - Broadcast geometry to multiple MIP levels, writes to unmapped tiles ignored
6. Shade using the sparse shadow map
   - Equivalent to other sparse texture usage

Algorithm Overview

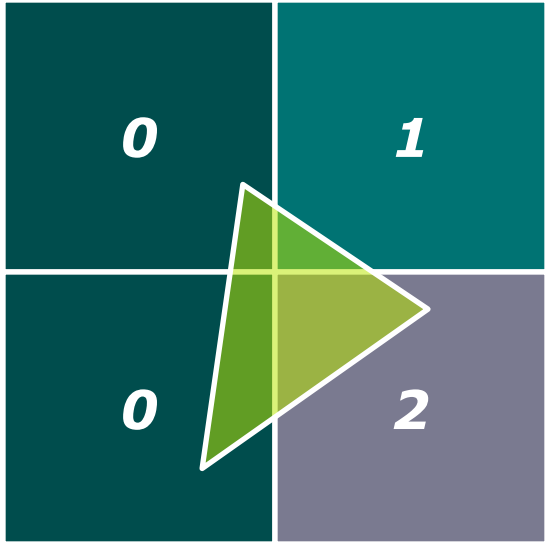
1. Render pre-pass, determining shadow map LOD at each pixel
   - E.g. a separate channel in the G-buffer may be used to store the LOD
2. Build the *min LOD map* in shadow map space
   - Project screen-space per-pixel LODs to light space, compute min LOD per-tile
3. Create a sorted list of *tile allocation requests*
   - Sorted from coarse to fine LODs
4. Remap tiles from the tile pool
   - First N tiles from the *request queue,* N is the size of the pool
5. Render to the sparse shadow map
   - Broadcast geometry to multiple MIP levels, writes to unmapped tiles ignored
6. Shade using the sparse shadow map
   - Equivalent to other sparse texture usage



Shadow map mips with allocated tiles

Rendering to the sparse shadowmap

- Geometry intersecting multiple tiles need to be replayed to appropriate LODs
  - GS sends triangle to finest level that has tiles mapped, and all coarser levels
  - Can use instanced GS for efficiency
- Writes to unmapped tiles are dropped



Need to render the triangle at LOD 0, 1, 2

Algorithm Overview

1. Render pre-pass, determining shadow map LOD at each pixel
   - E.g. a separate channel in the G-buffer may be used to store the LOD
2. Build the *min LOD map* in shadow map space
   - Project screen-space per-pixel LODs to light space, compute min LOD per-tile
3. Create a sorted list of *tile allocation requests*
   - Sorted from coarse to fine LODs
4. Remap tiles from the tile pool
   - First N tiles from the *request queue,* N is the size of the pool
5. Render to the sparse shadow map
   - Broadcast geometry to multiple MIP levels, writes to unmapped tiles ignored
6. Shade using the sparse shadow map
   - Equivalent to other sparse texture usage

Shading pass

- Use the shadowmap as any other sparse texture
  - Use the min LOD map to determine the LOD
  - Feed that into either LOD clamp or direct LOD texture sampling
  - Can also do a speculative lookup and replay

Fi

Questions?

- cem@nvidia.com

- For more info:
  - *Massive Virtual Textures for Games: Direct3D Tiled Resources*, Matt Sandy, Microsoft
  - http://channel9.msdn.com/Events/Build/2013/4-063

- Special thanks to Alexey Panteleev and Yury Uralsky