THE DEVIL IS IN THE DETAILS

IDTECH 666

Tiago Sousa
Lead Renderer Programmer

Jean Geffroy
Senior Engine Programmer

Render the Possibilities
SIGGRAPH2016

Initial Requirements

- Performance: 60hz @ 1080p
- Speed up art workflow
- Multi-platform scalability
- KISS
  - Minimalistic code
  - No shader permutations insanity: ~100 shaders, ~350 pipe states
- Next Gen Visuals
  - HDR, PBR
  - Dynamic and unified lighting, shadows and reflections
  - Good anti-aliasing and VFX

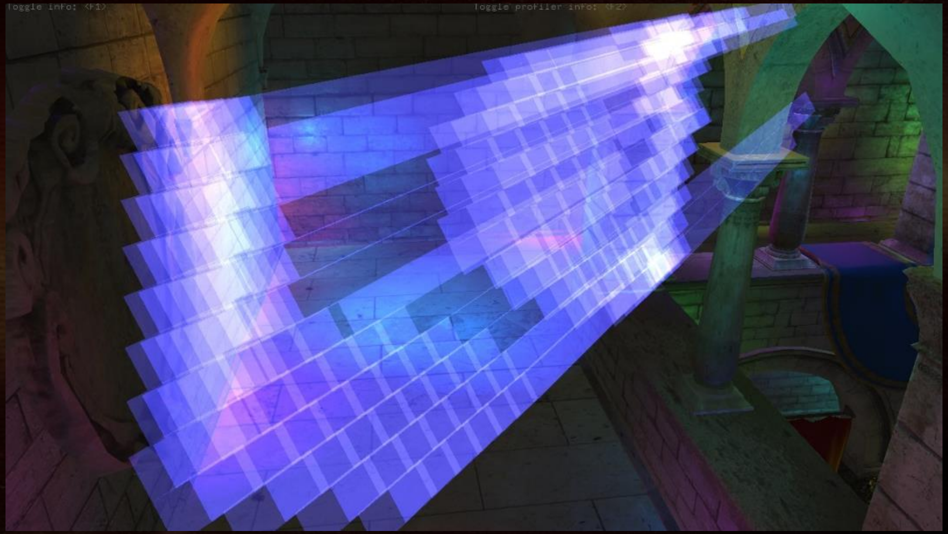
Anatomy of a Frame

Frame	Cost
▪ Shadow Caching	~3.0 ms
▪ Pre-Z	~0.5 ms
▪ Opaque Forward Passes ▪ Prepare cluster data ▪ Textures composite, compute lighting ▪ Output: L-Buffer, thin G-Buffer, feedback UAV	~6.5 ms
▪ Deferred Passes ▪ Reflections, AO, fog, final composite	~2.0 ms
▪ Transparency ▪ Particles light caching, particles / VFX, glass	~1.5 ms
Post-Process (Async)	~2.5 ms

Data Structure for Lighting & Shading

- A derivation from
  - "Clustered Deferred and Forward Shading" [Olson12]
  - "Practical Clustered Shading" [Person13]

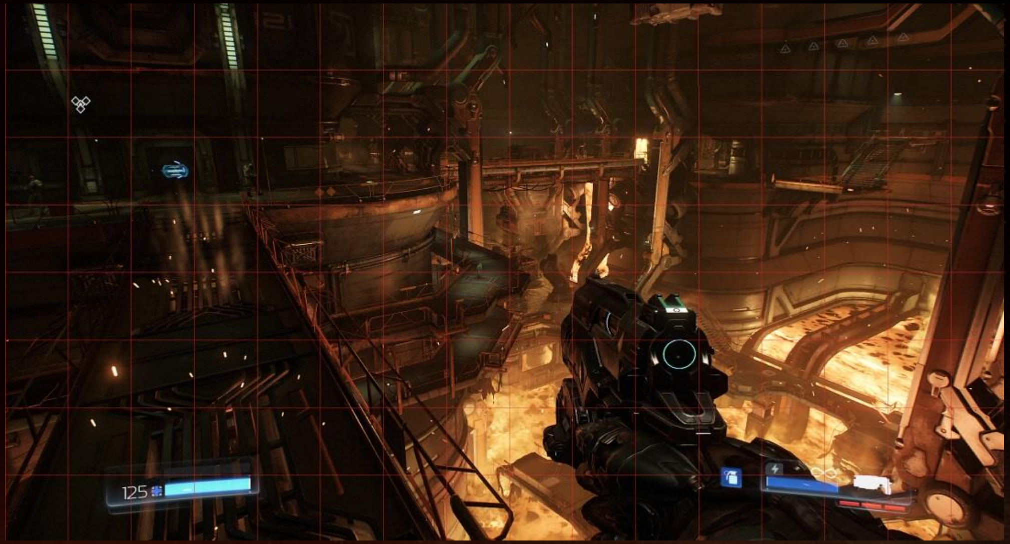
- Just works ™
  - Transparent surfaces
    - No need for extra passes or work
  - Independent from depth buffer
  - No false positives across depth discontinuities
  - More Just Works ™ in next slides



Olson12

Preparing Clustered Structure

- Frustum shaped voxelization / rasterization process
  - Done on CPU, 1 job per depth slice
- Logarithmical depth distribution
  - Extended near plane and far plane
  - $ZSlice = Near_z \times \left(\frac{Far_z}{Near_z}\right)^{\frac{slice}{num\ slices}}$
- Voxelize each item
  - An item can be: light, environment probe or a decal
  - Item shape is: OBB or a frustum (projector)
  - Rasterization bounded by screen space min_{xy} max_{xy} and depth bounds



Preparing Clustered Structure

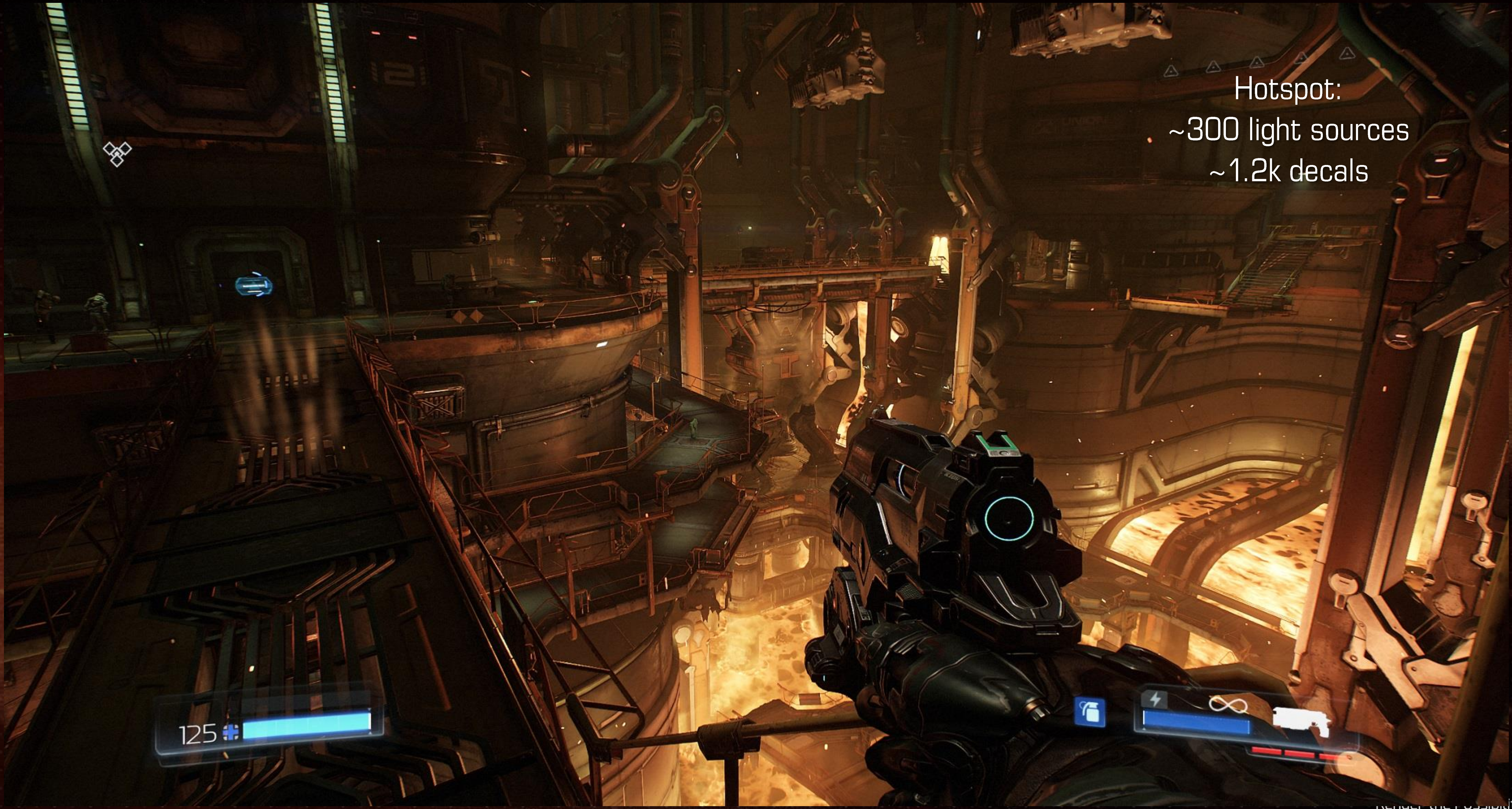
- Refinement done in clip space
  - A cell in clip space is an AABB
  - N Planes vs cell AABB
  - OBB is 6 planes, frustum is 5 planes
  - Same code for all volumes
  - SIMD

```
//Pseudo-code - 1 job per depth slice ( if any item )
for ( y = MinY; y < MaxY; ++y )     {
     for ( x = MinX; x < MaxX; ++x ) {
                 intersects = N planes vs cell AABB
                 if ( intersects ) {
                         Register item
                 }
     }
}
```

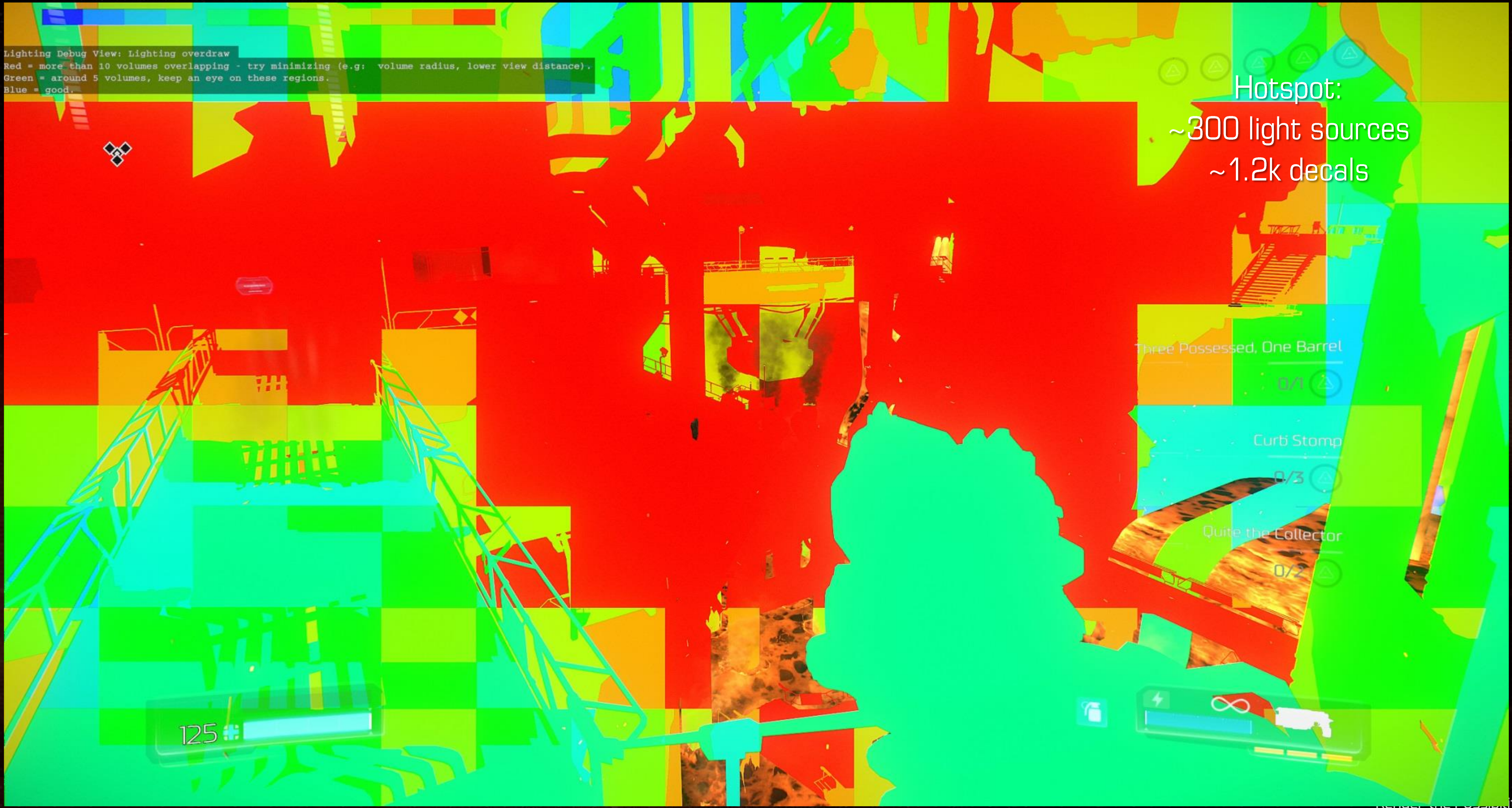
Preparing Clustered Structure

- Structures
  - Offset list:
    - 64 bits x Grid Dim X x Grid Dim Y x Grid Dim Z
  - Item list:
    - 32 bits x 256 x Worst case (Grid Dim X x Avg Grid Dim Y x Grid Dim Z)
- Offset List, per element
  - Offset into item list, and light / decal / probe count
- Item List, per element
  - 12 bits: Index into light list
  - 12 bits: Index into decal list
  - 8 bits: Index into probe list
- Grid resolution is fairly low res: 16 x 8 x 24
  - False positives: Early out mitigates + item list reads are uniform (GCN)

Preparing Clustered Structure

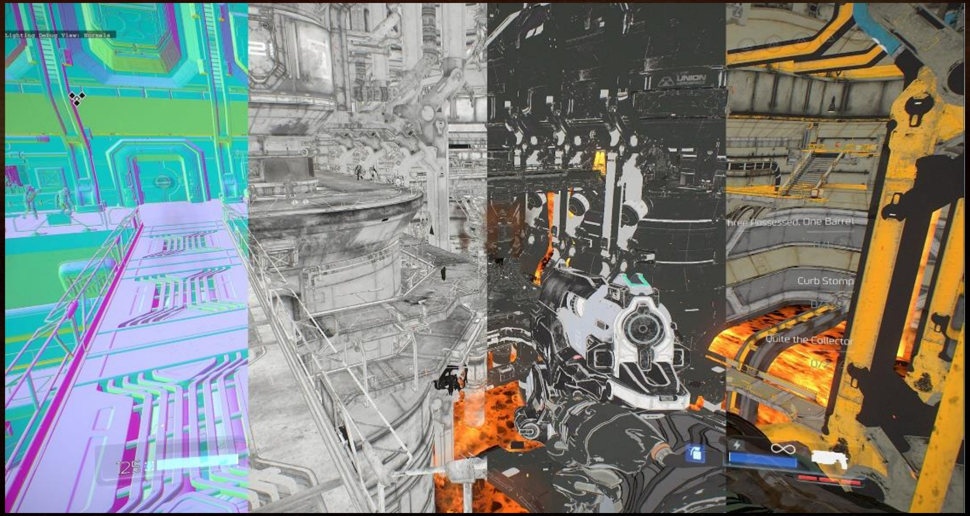


Preparing Clustered Structure



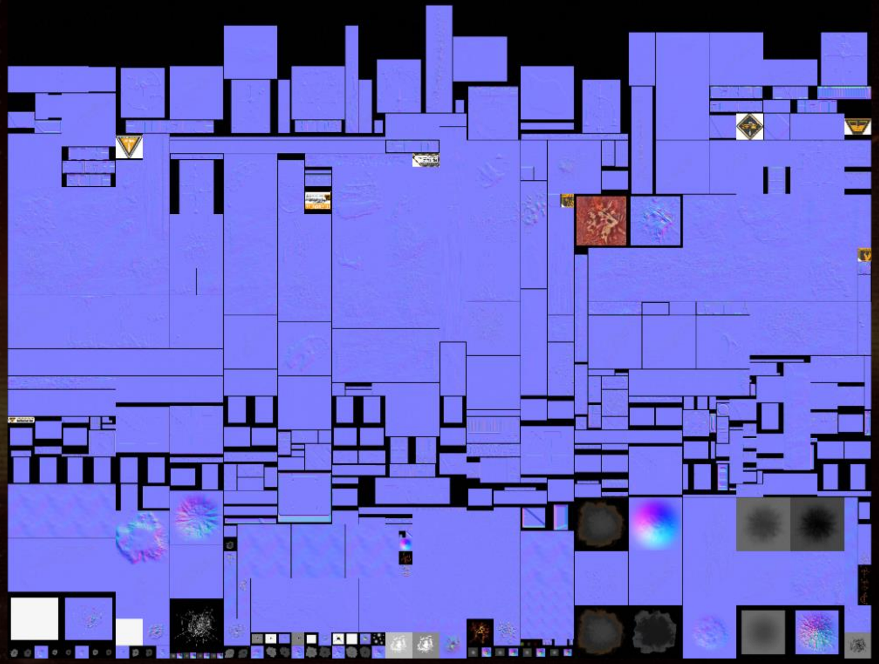
Detailing the World

- Virtual-Texturing[10] updates
- Albedo, Specular, Smoothness, Normals, HDR Lightmap
  - HW sRGB support
  - Baked Toksvig[11,12,13,14] into smoothness for specular anti-aliasing
- Feedback buffer UAV output directly to final resolution
- Async compute transcoding
  - Cost mostly irrelevant
- Design flaws still present
  - E.g. Reactive texture streaming = texture popping



Detailing the World

- Decals embedded with geometry rasterization
- Realtime replacement to Mega-Texture "Stamping"
  - Faster workflow / Less disk storage
- Just Works ™
  - Normal map blending
  - Linear correct blending for all channels
  - Mipmapping / Anisotropy *
  - Transparency
  - Sorting
  - 0 drawcalls
- 8k x 8k decal atlas
  - BC7



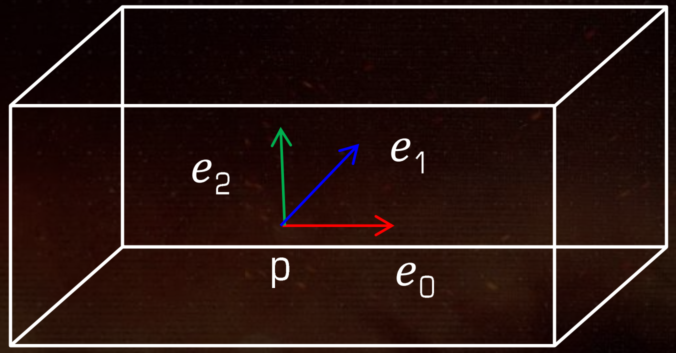
Decal Atlas

Detailing the World

- Box Projected
  - e_0 , e_1, e_2 is OBB normalized extents, p is position

$$M_{decalProj} = M_{scale} \cdot M_{decal}^{-1}$$

$$M_{scale} = \begin{vmatrix} \frac{0.5}{sizeX} & 0 & 0 & 0.5 \\ 0 & \frac{0.5}{sizeY} & 0 & 0.5 \\ 0 & 0 & \frac{0.5}{sizeZ} & 0.5 \\ 0 & 0 & 0 & 1 \end{vmatrix} \qquad M_{decal} = \begin{vmatrix} e_{0_x} & e_{1_x} & e_{2_x} & p_x \\ e_{0_y} & e_{1_y} & e_{2_y} & p_y \\ e_{0_z} & e_{1_z} & e_{2_z} & p_z \\ 0 & 0 & 0 & 1 \end{vmatrix}$$



- Indexing into decal atlas
  - Per decal: Scale & bias parameter. E.g.

```
const float4 albedo = tex2Dgrad( decalsAtlas, uv.xy * scaleBias.xy + scaleBias.zw, uvDDX, uvDDY );
```

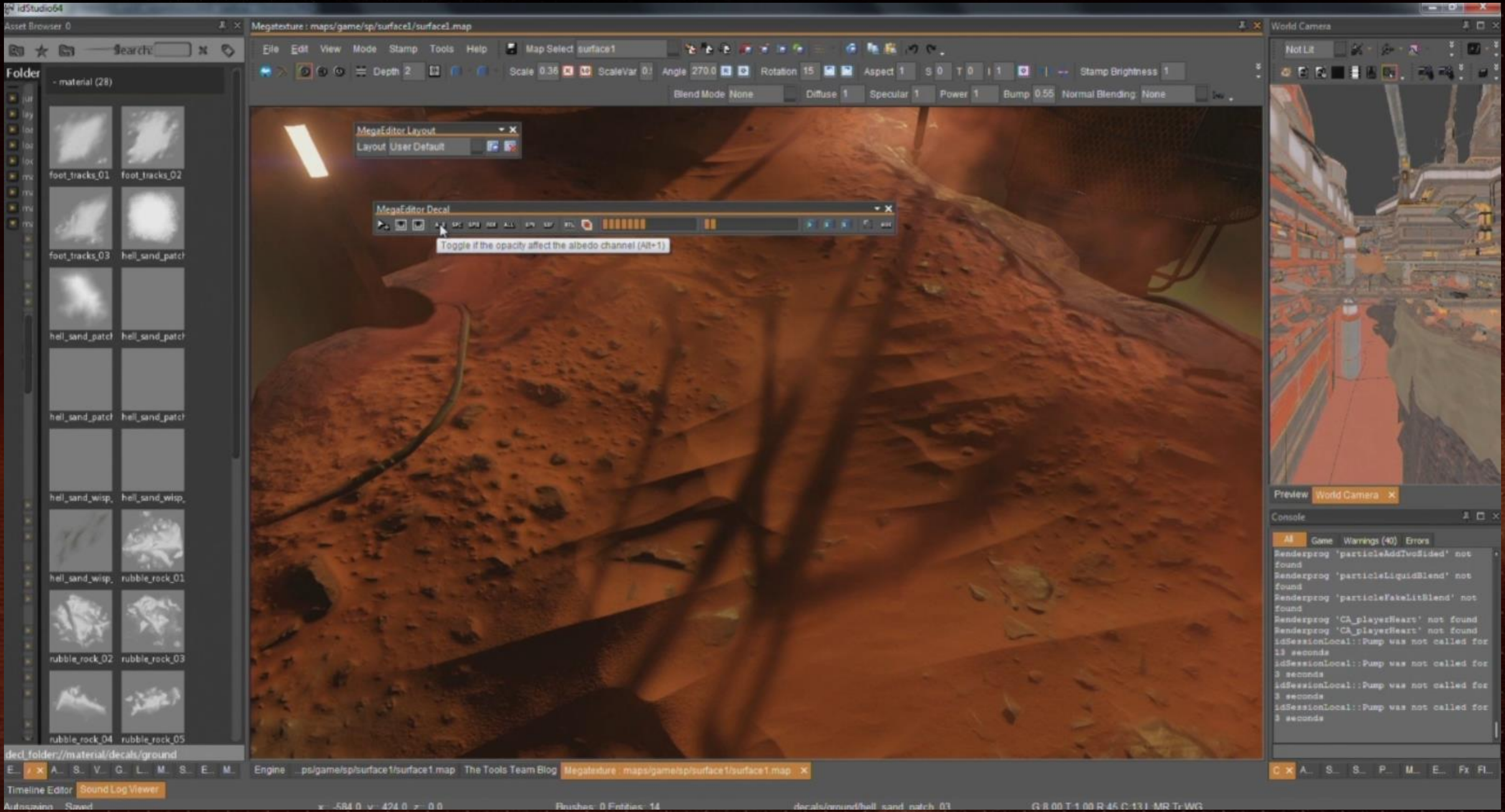
Detailing the World

- Manually placed by artists
  - Including blending setup
  - A generalization for “Blend Layers”
- Limited to 4k per view frustum
  - Generally 1k or less visible
- Lodding
  - Art setups max view distance
  - Player quality settings affect view distance as well
- Works on dynamic non-deformable geometry
  - Apply object transformation to decal

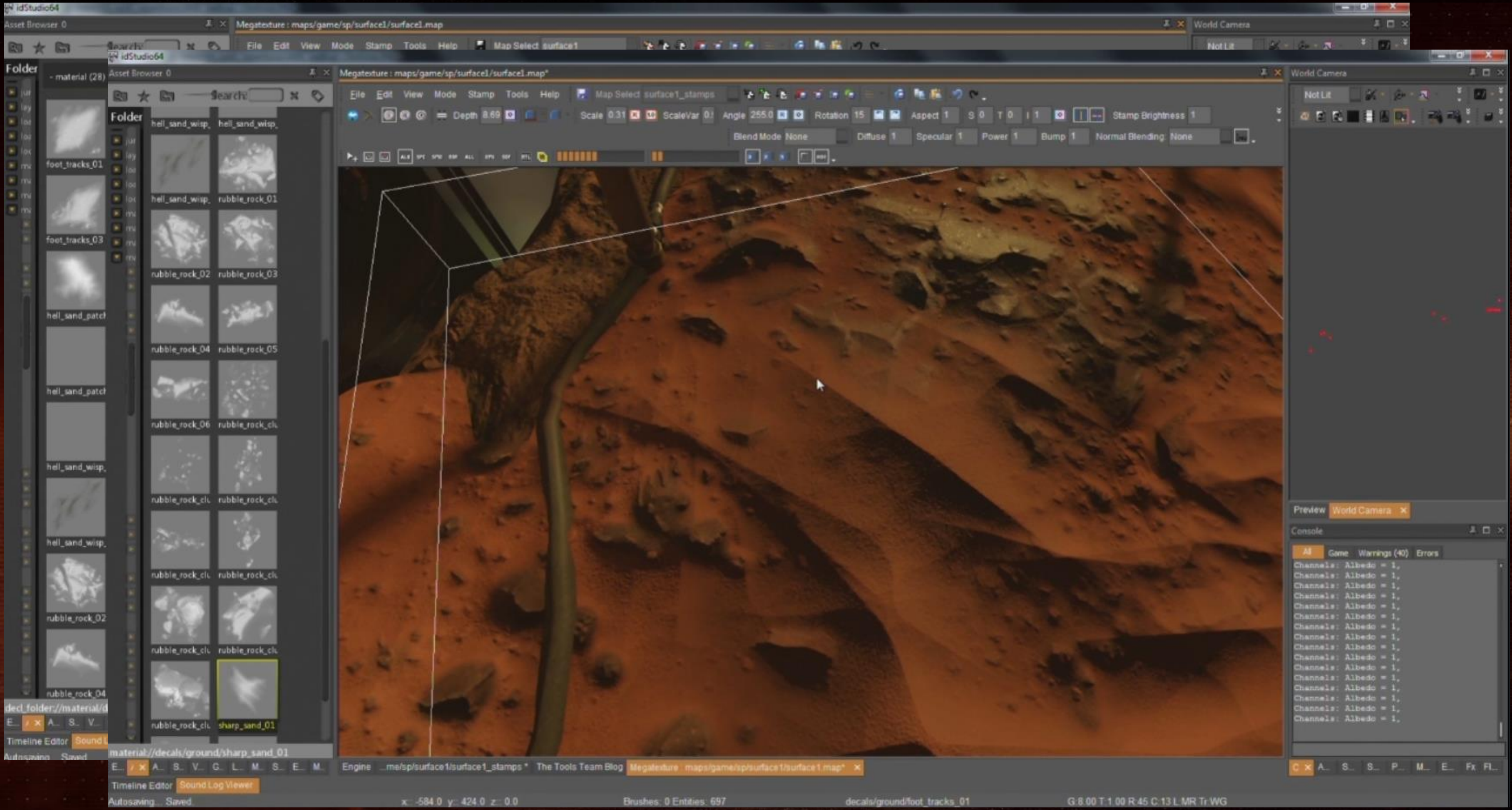
Detailing the World

Detailing the World

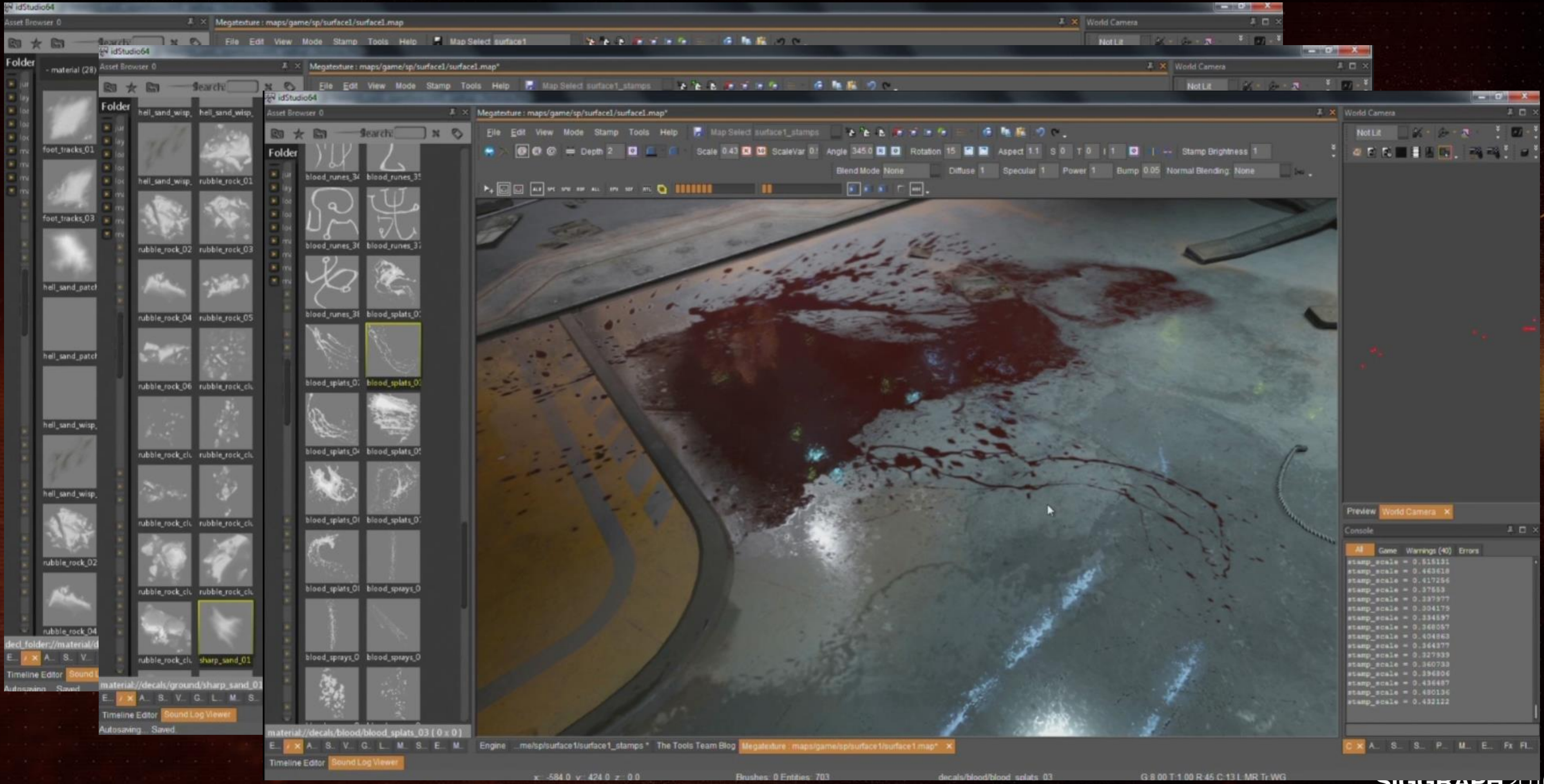
Detailing the World



Detailing the World



Detailing the World



Detailing the World

Lighting

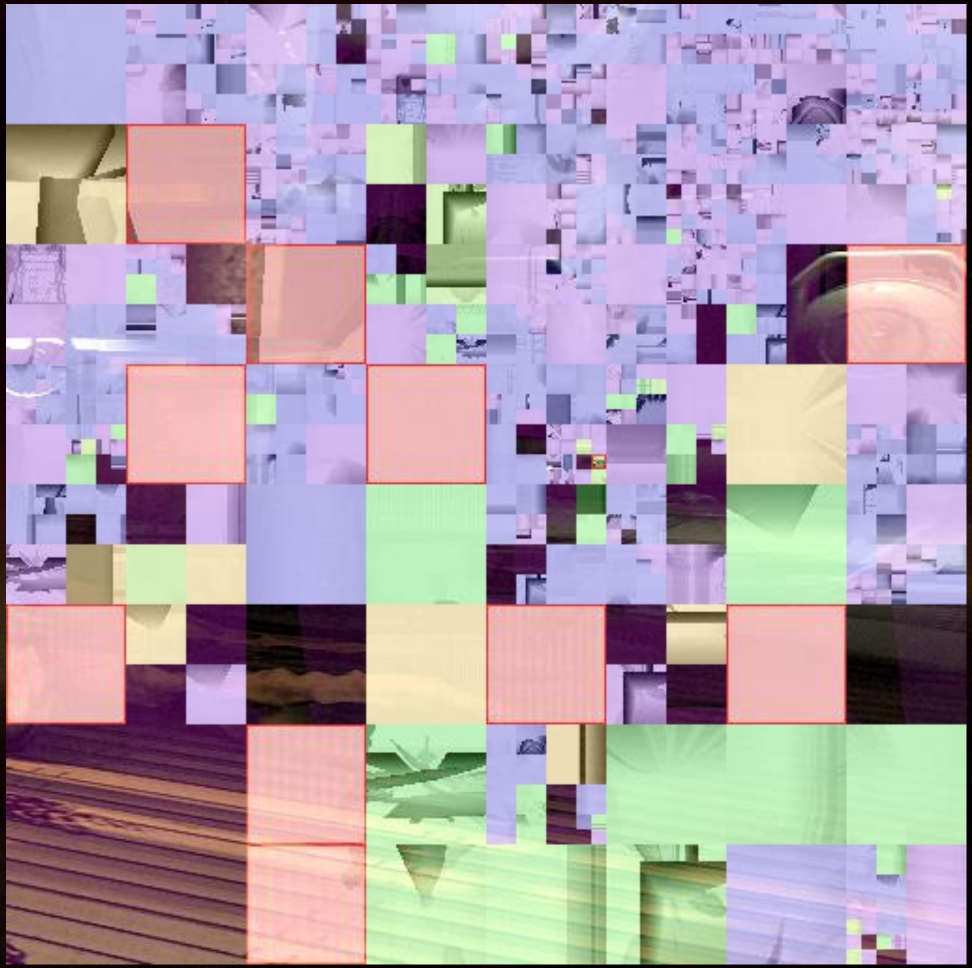
- Single / unified lighting code path
  - For opaque passes, deferred, transparents and decoupled particle lighting (slides 23-27)
- No shader permutations insanity
  - Static / coherent branching is pretty good this days – use it !
  - Same shader for all static geometry
  - Less context switches
- Components
  - Diffuse indirect lighting: Lightmap for static geometry, irradiance volumes for dynamics
  - Specular indirect lighting: Reflections (environment probes, SSR, specular occlusion)
  - Dynamic: Lights & shadows

Lighting

```
//Pseudocode

ComputeLighting( inputs, outputs ) {
    Read & Pack base textures

    for each decal in cell {
        early out fragment check
        Read textures
        Blend results
    }

    for each light in cell {
      early out fragment check
      Compute BRDF / Apply Shadows
      Accumulate lighting
    }
}
```

Lighting

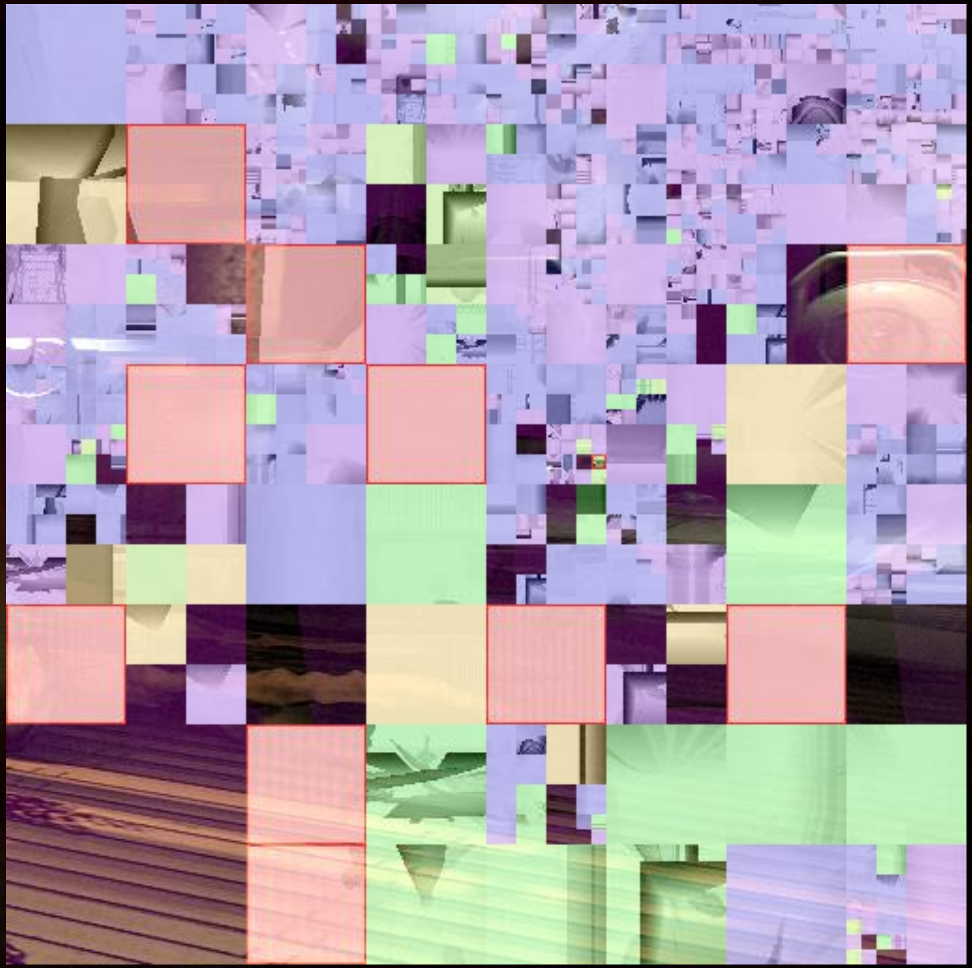
- Shadows are cached / packed into an Atlas
  - PC: 8k x 8k atlas (high spec), 32 bit
  - Consoles: 8k x 4k, 16 bit
- Variable resolution based on distance
- Time slicing also based on distance
- Optimized mesh for static geometry
- Light doesn't move?
  - Cache static geometry shadow map
  - No updates inside frustum ? Ship it
  - Update? Composite dynamic geometry with cached result
  - Can still animate (e.g. flicker)
- Art setup / Quality settings affect all above



Shadow Atlas

Lighting

- Index into shadow frustum projection matrix
- Same PCF lookup code for all light types
  - Less VGPR pressure
- This includes directional lights cascades
  - Dither used between cascades
  - Single cascade lookup
- Attempted VSM and derivatives
  - All with several artefacts
  - Conceptually has good potential for Forward
    - Eg. decouple filtering frequency from rasterization



Shadow Atlas

Lighting

- First person arms self-shadows
  - Dedicated atlas portion. Disabled on consoles to save atlas space

First Person Self-Shadows: On

First Person Self-Shadows: Off

(Notice light leaking)

Lighting

- Keep an eye on VGPR pressure
  - Pack data that has long lifetime. e.g: float4 for an HDR color ⇔ uint, RGBE encoded
  - Minimize register lifetime
  - Minimize nested loops / worst case path
  - Minimize branches
  - 56 VGPRS on consoles (PS4)
    - Higher on PC due to compiler inefficiency ☹ (@ AMD compiler team, pretty plz fix - throwing perf out)
- For future: half precision support will help

- *Nvidia: use UBOs / Constant Buffer (required partitioning buffers = more / ugly code)*
- *AMD: Prefer SSBOs / UAVs*

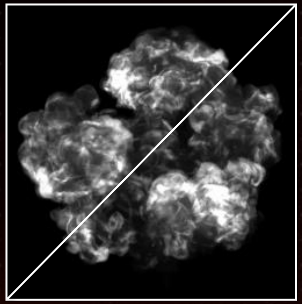
Transparents

- Rough glass approximation
  - Top mip is half res, 4 mips total
  - Gaussian kernel (approximate GGX lobe)
  - Blend mips based on surface smoothness
  - Refraction transfers limited to 2 per frame for performance
- Surface parameterization / variation via decals

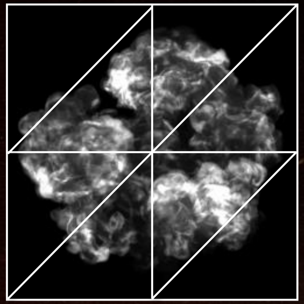
Glass Roughness Variation

Particle Lighting

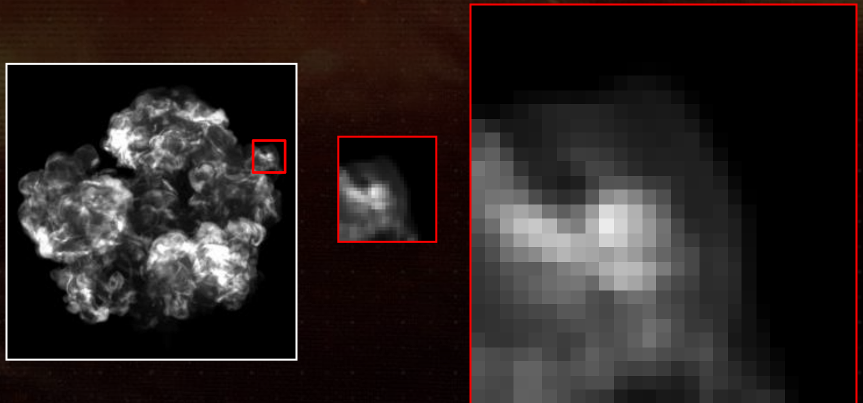
- Per-vertex ?
  - No higher frequency details (e.g. shadows)
- Per-vertex + tessellation [Jansen11]
  - Requires large subdivision level
  - Not good for GCN / Consoles
- Per-pixel ?
  - That's a lot of pixels / costly
- Mixed resolution rendering ?
  - Nguyen04 ? Problematic with sorting
  - Aliasing MSAA target ? Platform specific



Per Vertex



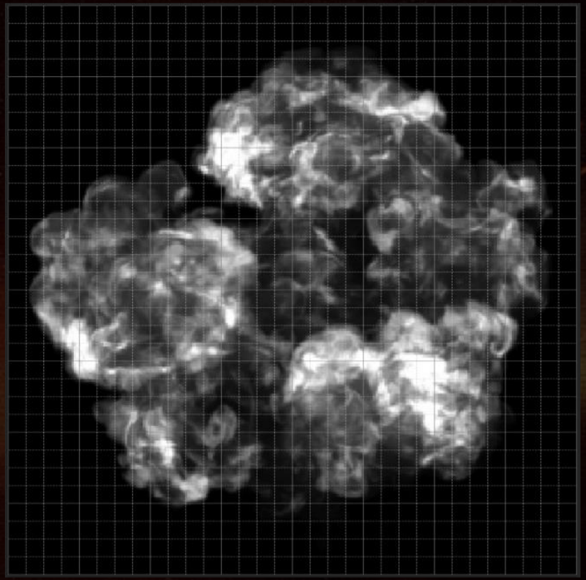
Tessellation



Per-Pixel*

Decoupled Particle Lighting

- Observation
  - Particles are generally low frequency / low res
  - Maybe render a quad per particle and cache lighting result ?
- Decouples lighting frequency from screen resolution = Profit
  - Lighting performance independent from screen resolution
  - Adaptive resolution heuristic depending on screen size / distance
    - E.g. 32x32, 16x16, 8x8
- Exact same lighting code path
- Final particle is still full res
  - Loads lighting result with a Bicubic kernel.



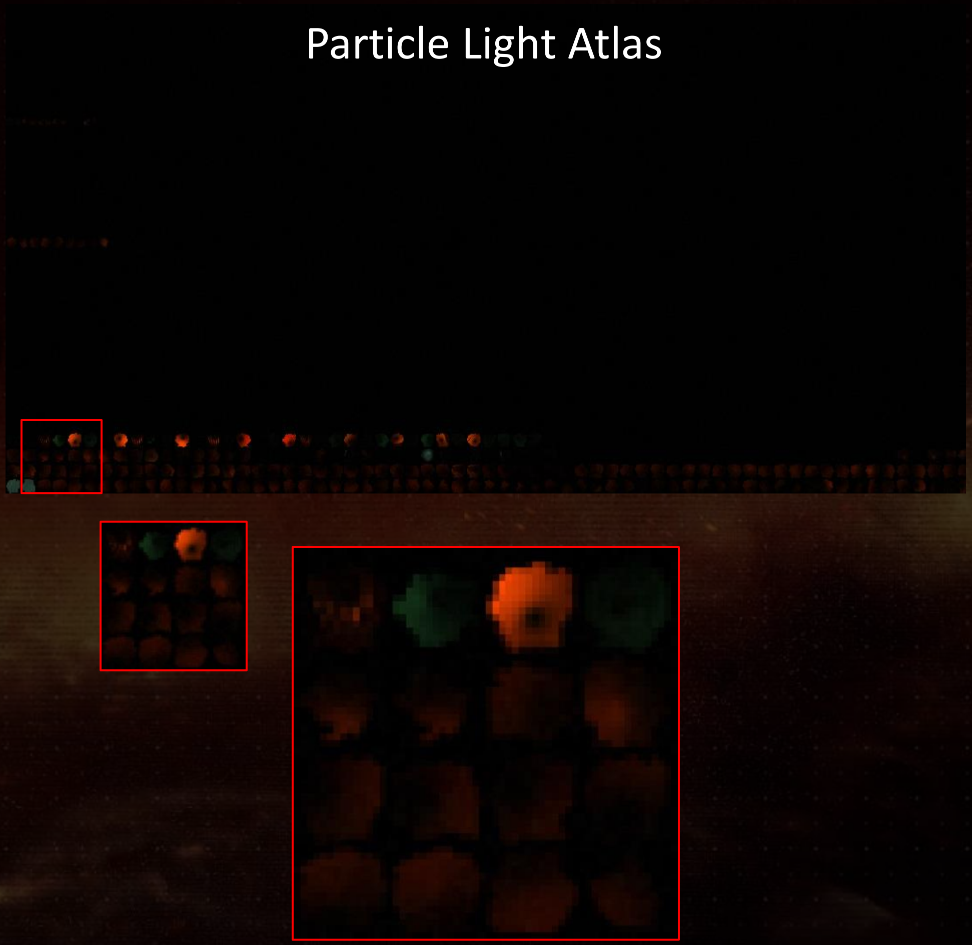
Adaptive resolution

Decoupled Particle Lighting

```
//Pseudocode - Particle shading becomes something like this

Particles( inputs, outputs ) {
    …
    const float3 lighting = tex2D( particleAtlas, inputs.texcoord );
    result = lighting * inputs.albedo;
    …
}
```

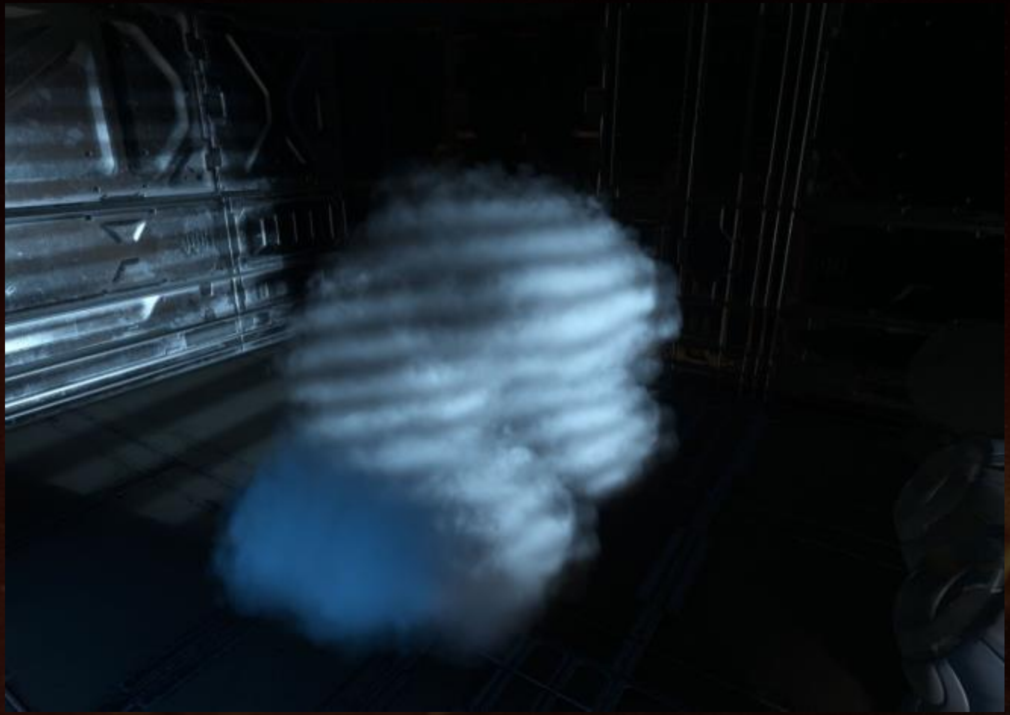
Decoupled Particle Lighting

- 4k x 4k particle light atlas
  - Size varies per-platform / quality settings
  - R11G11B10_FLOAT
- Dedicated atlas regions per-particle resolution
  - Some waste, but worked fine – ship it
- Fairly performant: ~0.1 ms
  - Worst cases up to ~1 ms
  - Still couple orders magnitude faster
  - Good candidate for Async Compute

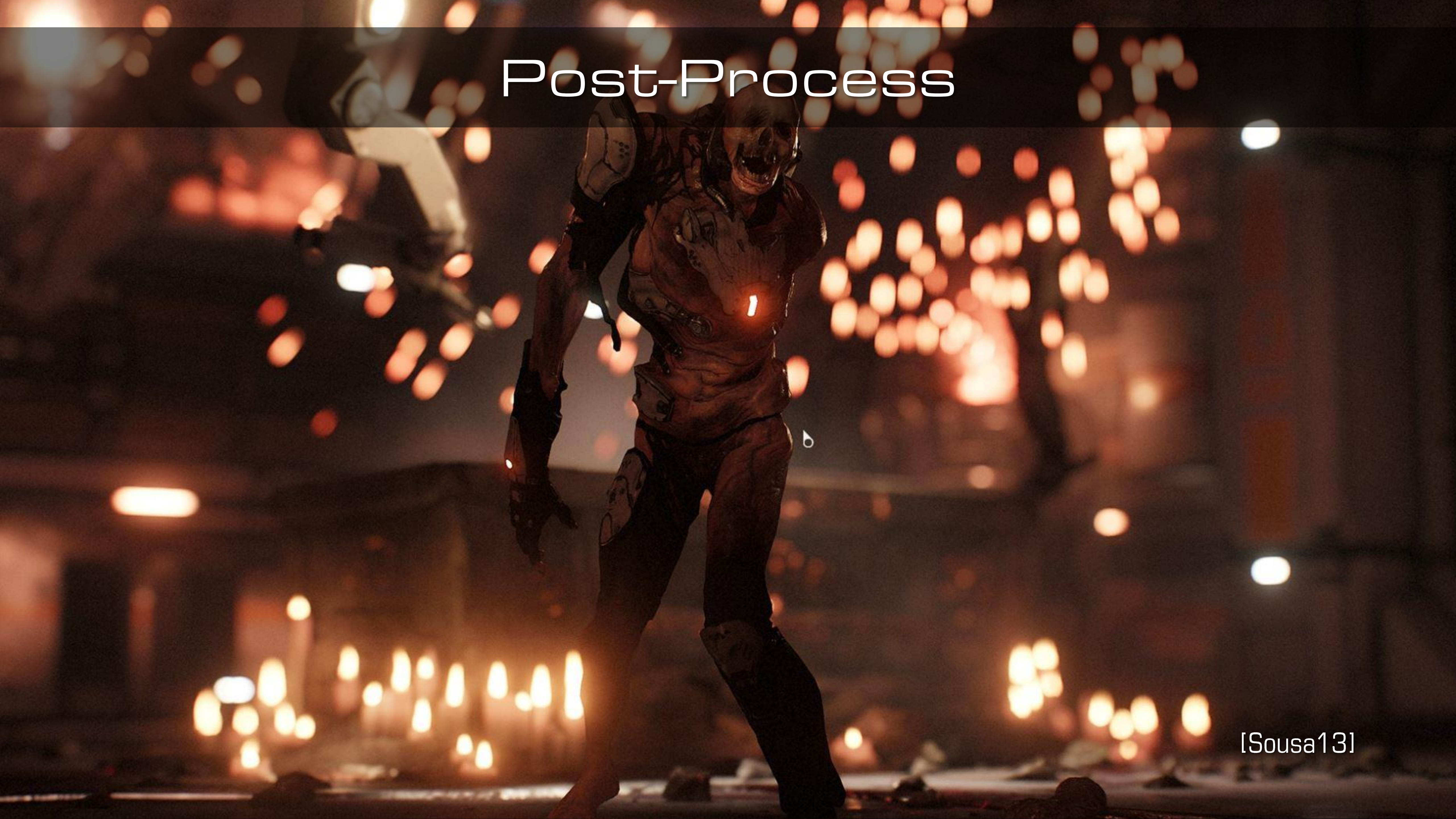


Decoupled Particle Lighting

- Results



Post-Process



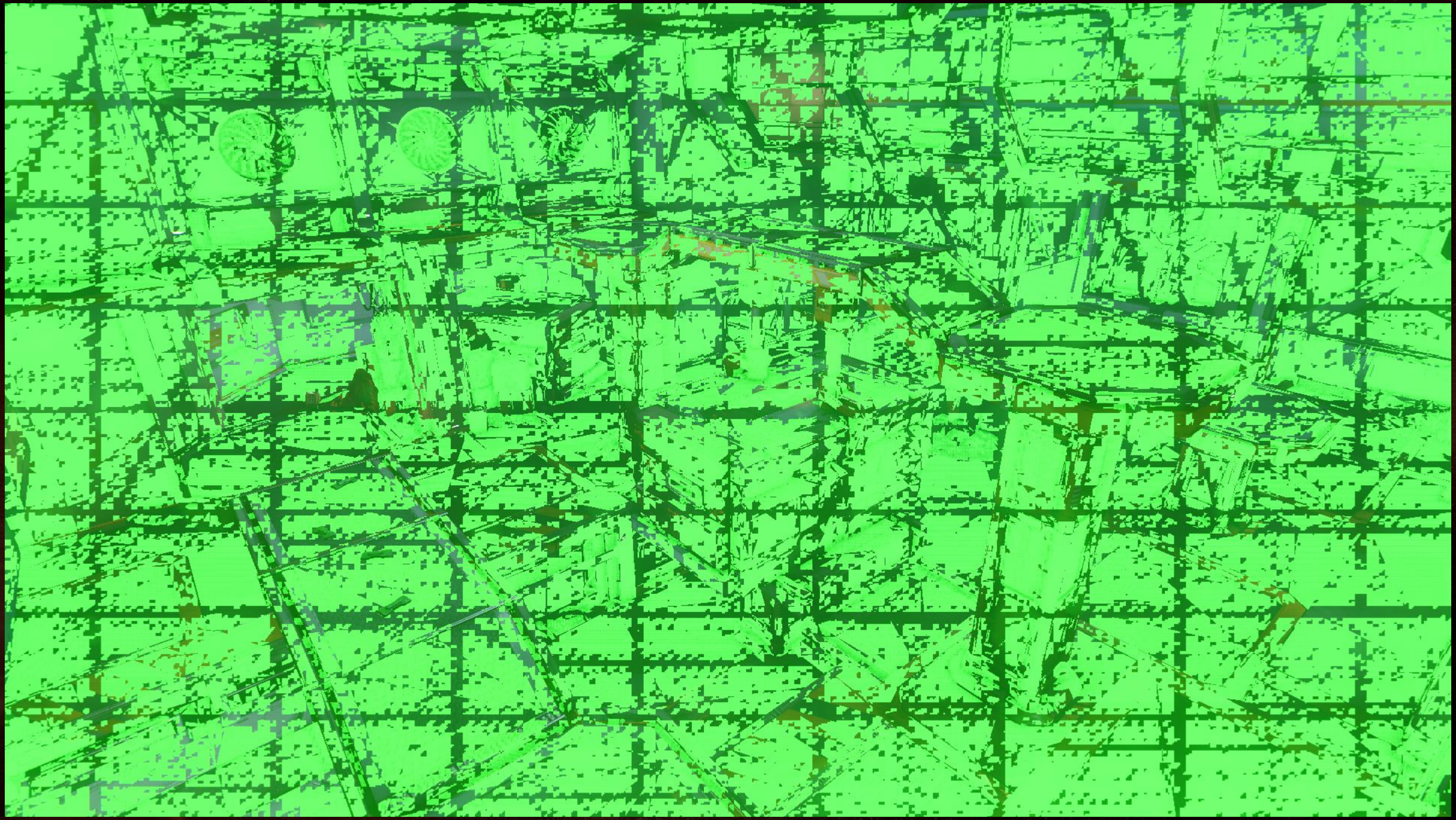
[Sousa13]

Optimizing Data Fetching (GCN)

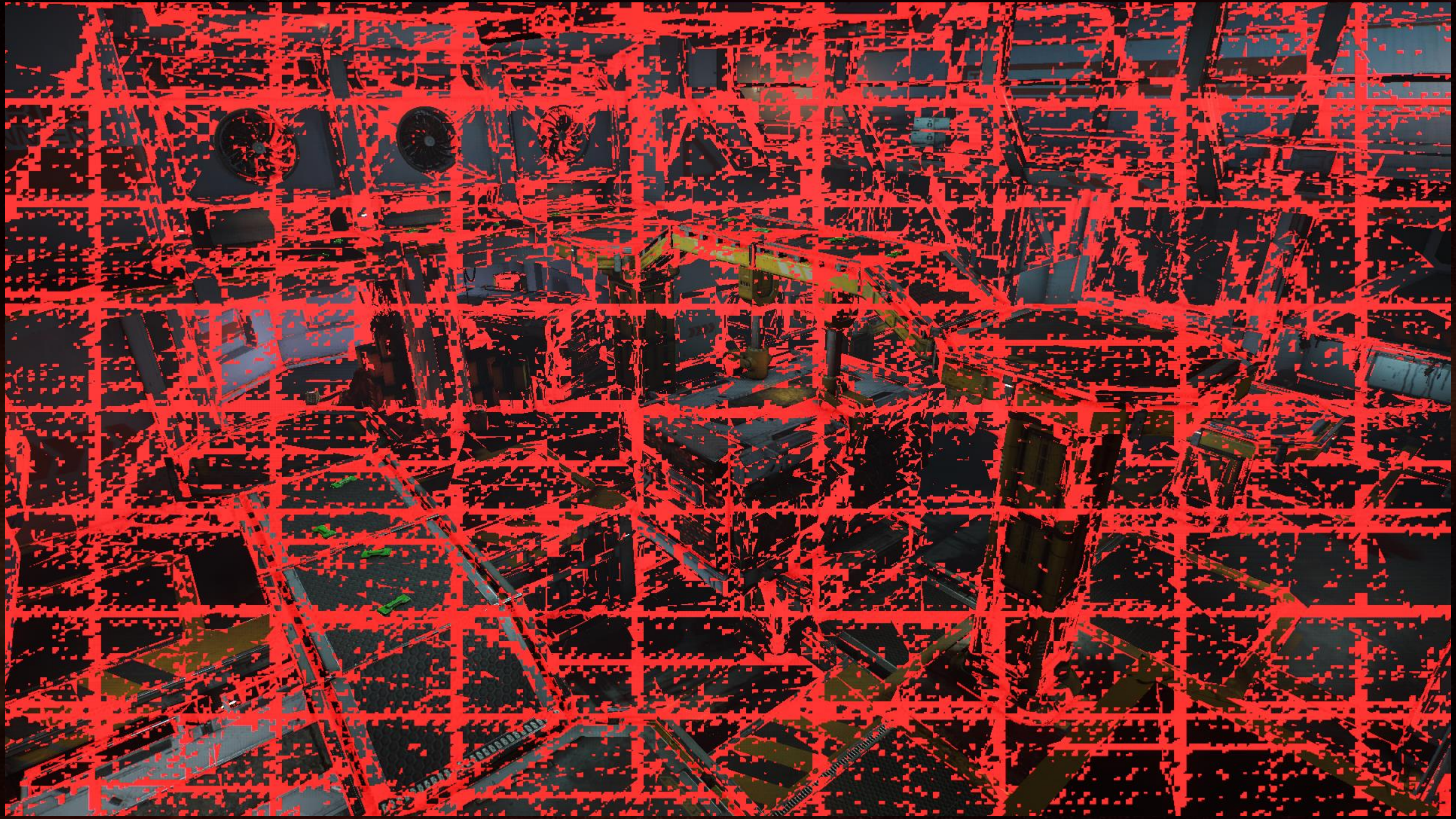
- GCN scalar unit for non-divergent operations
- Great for speeding up data fetching
  - Save some VGPRs
  - Coherent branching
  - Fewer instructions (SMEM: 64 Bytes, VMEM: 16 Bytes)
- Clustered shading use case
  - Each pixel fetches lights/decals from its belonging cell
  - Divergent by nature, but worth analyzing

Clustered Lighting Access Patterns

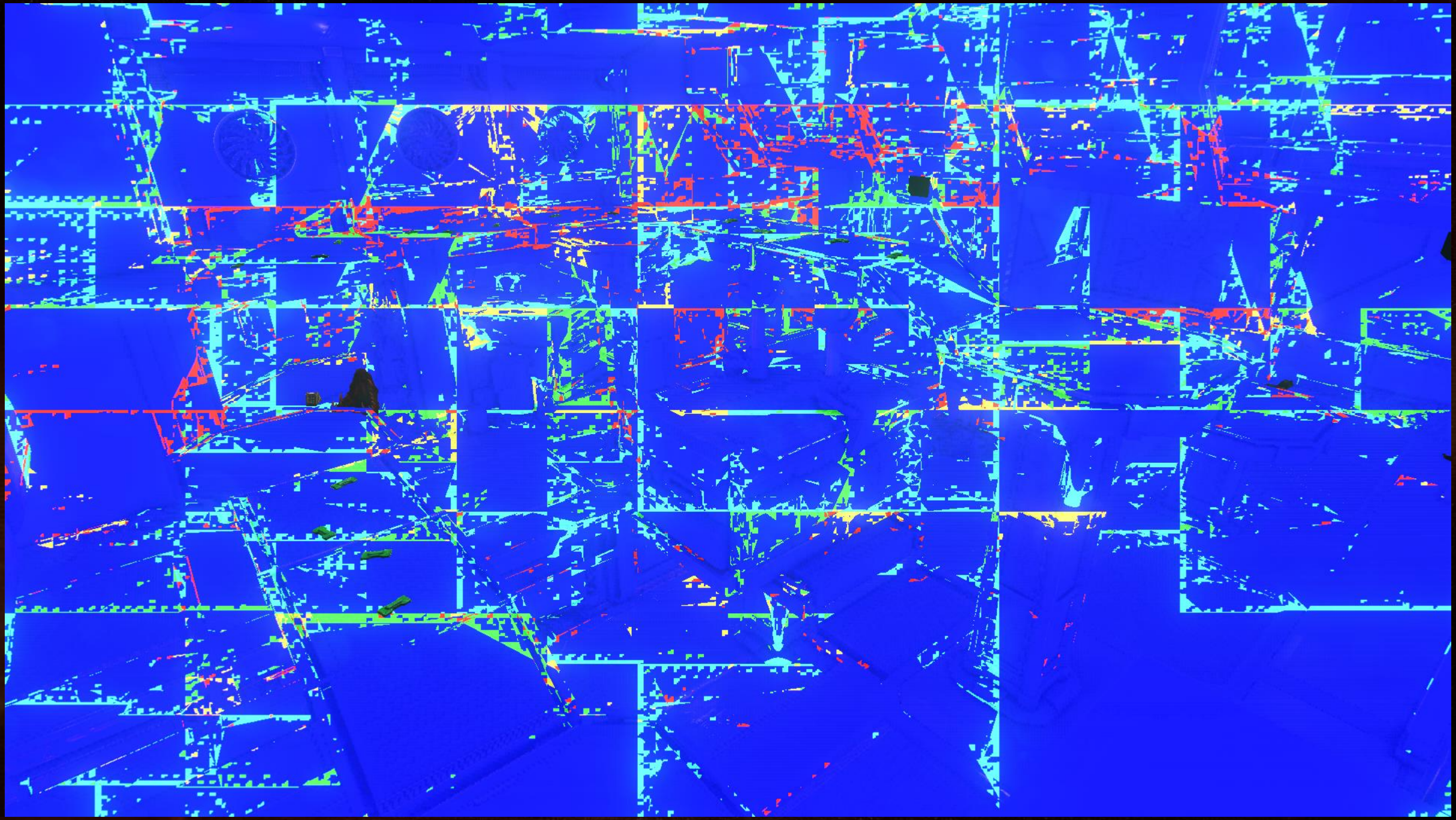
Clustered Lighting Access Patterns



Clustered Lighting Access Patterns



Clustered Lighting Access Patterns



Analyzing the Data

- Most wavefronts only access one cell
- Nearby cells share most of their content
- Threads mostly fetch the same data
- Per-thread cell data fetching not optimal
  - Not leveraging this data convergence
- Possible scalar iteration over merged cell content
  - Don’t have all threads independently fetch the exact same data

Leveraging Access Patterns

- Data: Sorted array of item (light/decal) IDs per cell
  - Same structure for lights and decals processing
  - Each thread potentially accessing a different node
  - Each thread independently iterating on those arrays
- Scalar loads: Serialize iteration
  - Compute smallest item ID value across all threads
    - ds_swizzle_b32 / minInvocationsNonUniformAMD
  - Process item for threads matching selected index
    - Uniform index -> scalar instructions
    - Matching threads move to next index

Divergent

Thread X	A	B	D	
Thread Y	B	C	E	
Thread Z	A	C	D	E

Serial

Scalar	A	B	C	D	E

Special Paths

- Fast path if touching only one cell [Fuller15]
  - Avoid computing smallest item ID, not cheap on GCN 1 & 2
  - Some additional (minor) scalar fetches and operations
- Serialization assumes locality between threads
  - Can be significantly slower if touching too many cells
  - Disabled for particle lighting atlas generation
- Opaque render pass, PS4 @ 1080p
  - Default: 8.9ms
  - Serialized iteration only: 6.7ms
  - Single cell fast path only: 7.2ms
  - Serialized iteration + fast path : 6.2ms

Dynamic Resolution Scaling

- Adapt resolution based on GPU load
  - Mostly 100% on PS4, more aggressive scaling on Xbox
- Render in same target, adjust viewport size
  - Intrusive: requires extra shader code
  - Only option on OpenGL
- Future: alias multiple render targets
  - Possible on consoles and Vulkan
- TAA can accumulate samples from different resolutions
- Upsample in async compute

Async Post Processing

- Shadow & depth passes barely use compute units
  - Fixed graphics pipeline heavy
- Opaque pass not 100% busy either
- Overlap them with post processing
  - Render GUI in premultiplied alpha buffer on GFX queue
  - Post process / AA / upsample / compose UI on compute queue
  - Overlap with shadows / depth / opaque of frame N+1
  - Present from compute queue if available
    - Potentially lower latency

GCN Wave Limits Tuning

- Setup different limits for each pass
  - Disable late alloc for high pixel/triangle ratio
- Restrict allocation for async compute
  - Avoid stealing all compute units
  - Mitigate cache thrashing
- Worth fine tweaking before shipping
  - Saved up to 1.5ms in some scenes in DOOM!

GCN Register Usage

- Think globally about register and LDS allocation
  - Do not always aim for divisors of 256
  - Bear in mind concurrent vertex / async compute shaders
- Fine tweaking to find sweet spot
- Example: DOOM opaque pass
  - GFX queue: 56 VGPRs for PS, 24 for VS
  - Compute queue: 32 VGPRs for upsample CS
  - 4PS + 1CS/VS or 3PS + 2CS + 1VS
  - Saves 0.7ms compared to a 64 VGPRs version


D1


RESEARCH

What's next ?

- Decoupling frequency of costs = Profit
- Improve
  - Texture quality
  - Global illumination
  - Overall detail
  - Workflows
  - etc

Special Thanks

- Code
  - Robert Duffy, Billy Khan, Jim Kejllin, Allen Bogue, Sean Flemming, Darin Mcneil, Axel Gneiting, Michael Kopietz, Magnus Högdahl, Bogdan Coroi, Ivo Zoltan Frey, Johnmichael Quinlan, Greg Hodges
- Art
  - Tony Garza, Lear Darocy, Timothee Yeremian, Jason Martin, Efgeni Bischoff, Felix Leyendecker, Philip Bailey, Gregor Kopka, Pontus Wahlin, Brett Paton
- Entire id Software team
- Natalya Tatarchuk

We are Hiring !

- Various openings across Zenimax studios
- Please visit https://jobs.zenimax.com

Thank you

- Tiago Sousa
  - tiago.sousa@idsoftware.com
  - Twitter: @idSoftwareTiago
- Jean Geffroy
  - Jean.geffroy@idsoftware.com

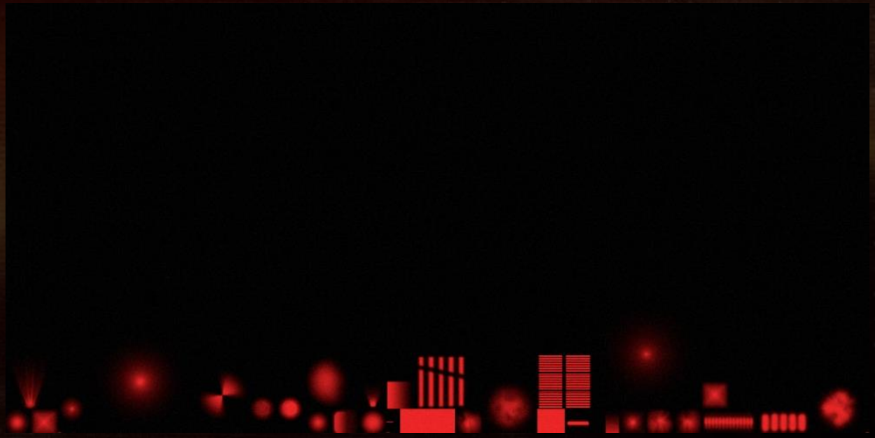
References

- [1] "Clustered Deferred and Forward Shading", Ola Olson et al., HPG 2012
- [2] "Practical Clustered Shading", Emil Person, Siggraph 2013
- [3] "CryENGINE 3 Graphics Gems", Tiago Sousa, Siggraph 2013
- [4] "Fast Rendering of Opacity Mapped Particles using DirectX11", Jon Jansen, Louis Bavoil, Nvidia Whitepaper 2011
- [5] "Fire in the Vulkan Demo", H Nguyen, GPU Gems, 2004
- [6] "Lost Planet Tech Overview", http://game.watch.impress.co.jp/docs/20070131/3dlp.htm
- [7] "GPU Best Practices (Part 2)", Martin Fuller, Xfest 2015
- [8] "Southern Island Series Instruction Set Architecture", Reference Guide, 2012
- [9] "GCN Shader Extensions for Direct3D and Vulkan", Matthaeus Chajdas, GPUOpen.com, 2016
- [10] "id Tech 5 Challenges", J.M.P. van Waveren, Siggraph, 2009
- [11] "Mipmapping Normal Maps", Toksvig M, 2004
- [12] "Real-Time Rendering, 3rd Edition", Moller et al., 2008
- [13] "Physically-based lighting in Call of Duty: Black Ops", Dimitar Lazarov, Siggraph 2011
- [14] "Specular Showdown in the Wild West", Stephen Hill, 2011

Bonus Slides

Lighting

- Light types
  - Point, projector, directional (no explicit sun), area (quad, disk, sphere)
  - IBL (environment probe)
- Light shape
  - Most lights are OBBs: Acts as implicit “clip volume” to help art preventing light leaking
  - Projector is a pyramid
- Attenuation / Projectors
  - Uses art driven texture at this point
  - Stored in an atlas, similar indexing as decals
  - Art sometimes uses for faking shadows
  - BC4
- Environment Probes
  - Cube map array, index via probe ID
  - Fixed resolution, 128 x 128
  - BC6H



Projector Atlas

Deferred Passes

- Wanted dynamic and performant AO & reflections
  - Decoupling passes helps mitigate VGPR pressure
- 2 extra targets during forward opaque passes
  - Specular & smoothness: RGBA8
  - Normals: R16G16F
- Allows compositing probes with realtime reflections
- Final Composite
  - SSR, environment probes, AO / specular occlusion , fog