SIGGRAPH2012

The **39th** International **Conference** and **Exhibition**
on **Computer Graphics** and **Interactive Techniques**

Intersecting Lights with Pixels

Reasoning about Forward and Deferred Rendering

Overview

SIGGRAPH2012

- Real-time rendering with “many” lights
  - 10s to 10,000s of lights, ~2-10s affecting a given pixel
  - Which lights have non-negligible effect on which pixels?
  - Ideas in this talk apply more generally
- Intersecting lights and pixels
- Forward and deferred rendering

Visibility

SIGGRAPH2012

- Visibility is a collision detection problem
  - Intersecting triangles with pixels
  - Rasterization and ray casting compute these intersections
- Can use a variety of data structures and algorithms
  - Hierarchical rasterization, binning (TBDR)
  - Uniform and projected grids, bounding volume hierarchies
  - Etc…

Light Culling

- Light culling is also a collision detection problem
  - Intersecting lights with pixels
  - Can use many of the same tools as for visibility
- Granularity of culling should balance cost of lighting
  - i.e. don’t spend more time getting fancy with culling than it would have taken to just compute the light attenuation
  - Cost of lighting related to visible light size in *image space*

Efficient Light Culling

SIGGRAPH2012

- In general to intersect pixels and lights:
  - Put pixels in some acceleration structure
  - Put lights in some acceleration structure
  - Intersect the two structures
- Let's look at some examples

Conventional Deferred Rendering

SIGGRAPH2012

- Pixels: effectively no accelerator
  - Semi-hierarchical due to HiZ hardware
- Lights: represented as triangles
- Use rasterizer to intersect the pixels and triangles
  - Z-test is single direction, want both min/max
  - Intersections (“coverage”) too big to store per-pixel
  - Forced to accumulate lighting (bandwidth inefficient)

Tiled Culling

- Pixels: projected uniform grid accelerator (“tiles”)
- Lights: no accelerator
- Intersect via software conservative rasterization
  - Use binning to generate light lists per tile
  - Loop over lights in a tile and shade all at once
- Tiled deferred and tiled forward (aka “Forward+”)
  - Significantly better than conventional techniques

Software Rasterization?

- Hardware rasterizer doesn’t do what we need
  - Need conservative rasterization (for tiles)
  - Need double-sized Z
  - Rasterizing shapes like spheres, cones, cylinders, …
  - Need to append to lists as “raster operation”
- Don’t be scared to use software rasterization when the hardware pipeline is inappropriate!

Culling Inefficiencies

SIGGRAPH2012

- Redundant intersection tests
- Inflated tile frusta at discontinuities
- Light acceleration structures
- Culling other terms

Inefficiencies: Redundant Tests

- Testing every light against every tile
  - Massive redundancy of tests… not cool

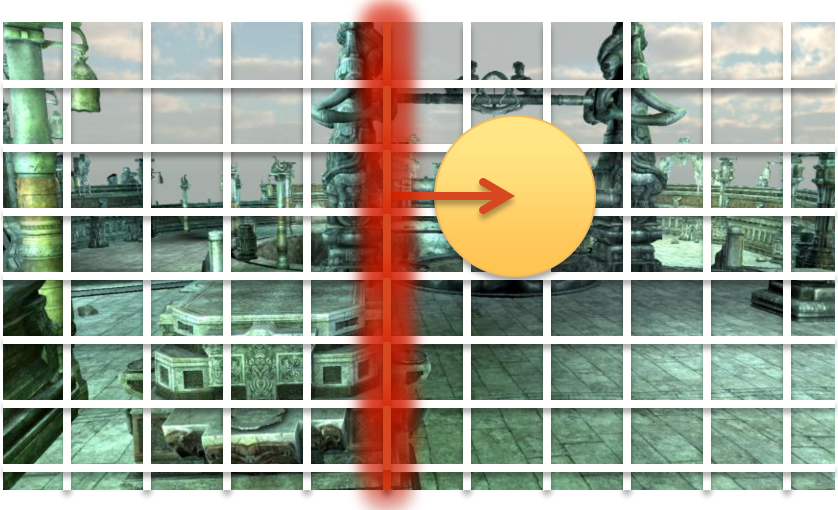


Image Space Quad-Tree

- Build a quad-tree of lights in image space
  - Intersect lights vs. split planes and tile depth bounds
  - Recurse (spawn w/ work stealing) until max light cutoff
  - Shade, output light lists, etc. at leaves (tiles)
- GPU programming model not currently suitable
  - No spawn, work stealing, etc.
- Free optimized CPU implementation in ISPC package
  - http://ispc.github.com

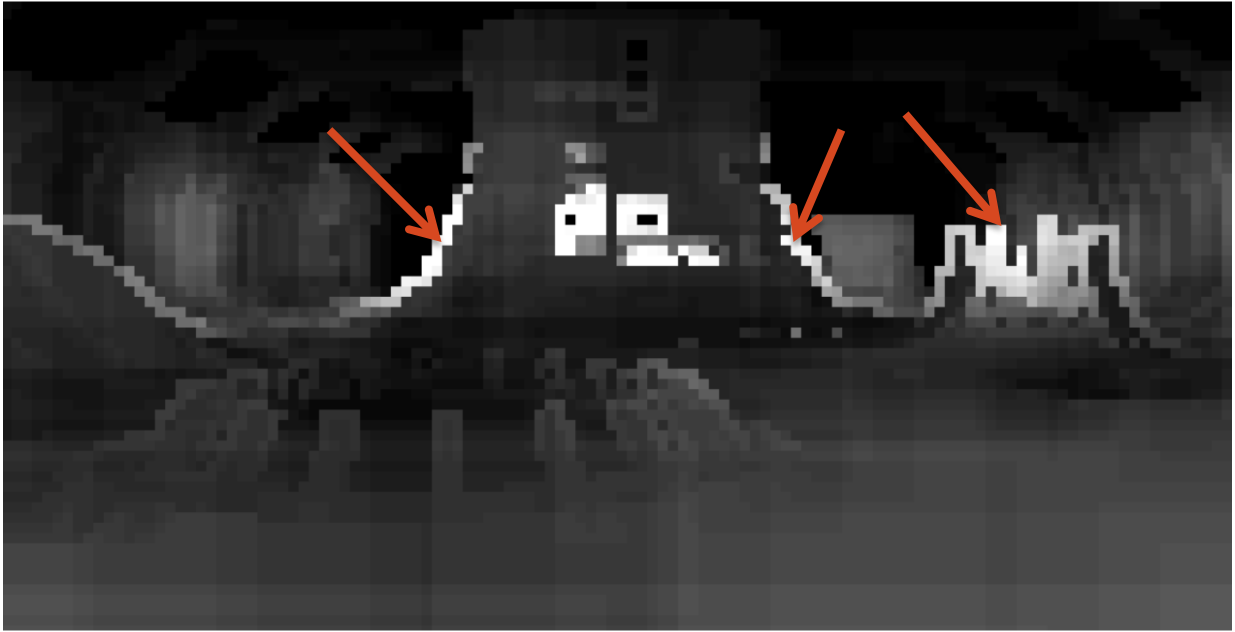
Quad-Tree

SIGGRAPH2012

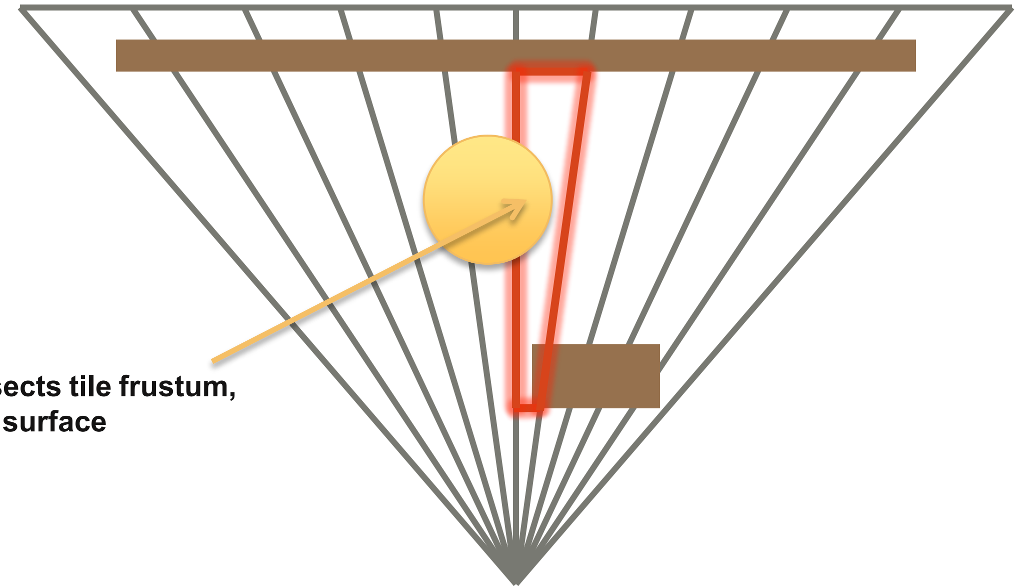
Quad-Tree Performance

- Consider 1080p, 16x16 tiles, 1024 lights
- Brute force (tiled deferred, tiled forward):
  - ~= 33 million x/y plane tests
  - 1.0ms on NVIDIA 680 (~200W)
  - 10.0ms on mobile Intel 4000 (~20W)
- Hierarchical quad-tree:
  - Worst case (all lights hit every tile) ~= ½ million x/y tests
  - 0.8ms on 2nd Gen Core i7 (~100W)

Inefficiencies: Discontinuities



Discontinuities

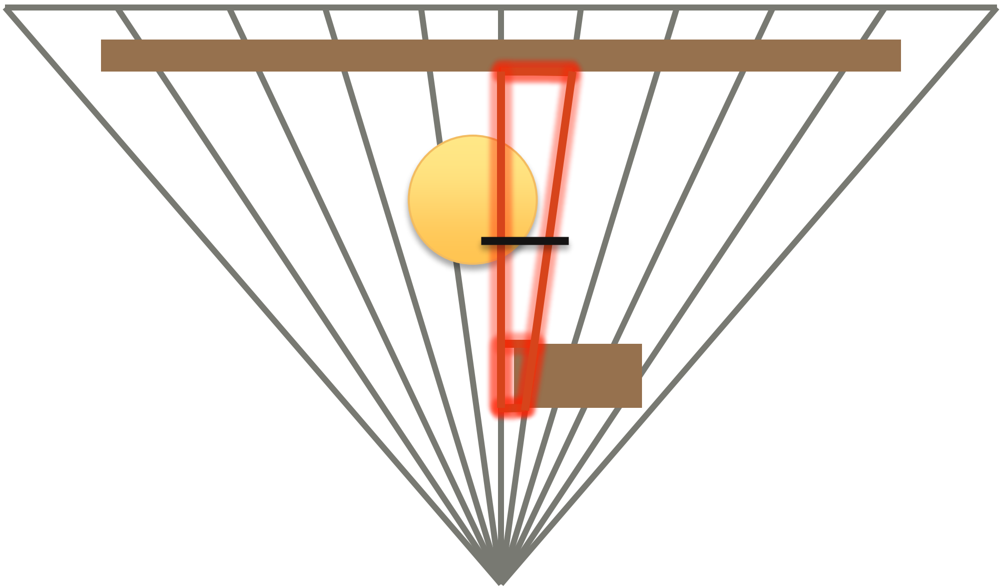


Light intersects tile frustum,
but neither surface

Discontinuity Solutions

- 3D acceleration structure (grid, oct-tree, etc.)
  - Extending tiling/clustering to depth
  - May need this for alpha blended geometry anyways
- Very simple one: bimodal clusters
  - Assume depth distribution over tile is bimodal
  - Split tile in depth half way between extents (min/max)
  - Only adds 2 more plane tests for a total of 8 per tile
  - Handles the majority of problem cases

Bimodal Clusters



Inefficiencies: Light Accelerators

SIGGRAPH2012

- All of these techniques still loop over all lights
  - Incapable of accepting or rejecting multiple lights at once
  - Not scalable to very high light counts
- Can be solved by adding a light accelerator as well
  - Typically a bounding volume hierarchy of some sort
  - Intersection then can then efficiently merge the two structures and find overlaps
- See [Olsson 2012] for one example of this

Culling Other Terms

- Can also cull other terms of the attenuation function
  - NdotL is a common one – cull lights that are entirely back-facing from the pixel/tile/cluster
- Don't go overboard…
  - Remember: culling must be cheaper than the cost of computing the term of each pixel
  - Ex. Computing NdotL for a 16x16 tile is 768 ops per light
    - Make sure any normal culling you do is cheaper than that!

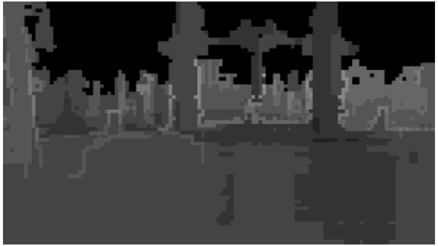
Light Culling Conclusions

SIGGRAPH2012

- Light culling is collision detection, like visibility
  - Pick and acceleration structure or two and intersect ☺
  - Rasterization, binning, tiling, etc. are all just tools to help
  - Ideally a renderer should be able to vary these choices without any programmer intervention
- Always branch per-pixel on your attenuation function
  - Only do fancier culling if it's cheaper than this branch

Tiled Forward or Tiled Deferred?

Compute Pixel Geometry → Intersect Lights with Pixel Tiles → Shade



Tiled Forward

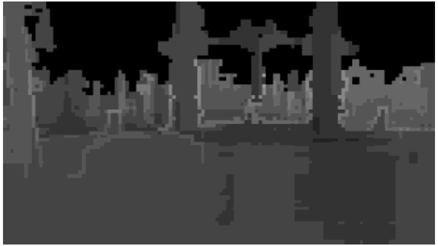
SIGGRAPH2012

Compute Pixel Geometry → Intersect Lights with Pixel Tiles → Shade

- Submit geometry
- Store depth

- Read depth, cull tiles
- Store lights lists

- Submit geometry
- Read light lists
- Shade



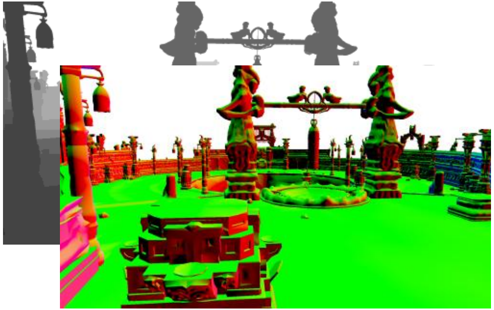
Tiled Deferred

Compute Pixel Geometry → Intersect Lights with Pixel Tiles → Shade

Compute Pixel Geometry
- Submit geometry
- Store surface data

Intersect Lights with Pixel Tiles
- Read depth, cull tiles
- Store light lists (local)

Shade
- Read surface data
- Read light lists (local)
- Shade



What's the Difference?

- Difference *is not*:
  - Ability to handle many/complex materials
  - Ability to use multi-sample anti-aliasing
  - …
- Difference *is*:
  - Store all rasterizer outputs in pre-z pass, or resubmit
  - Hardware vs. software scheduling of shading

G-buffer

- G-buffer simply stores rasterizer outputs
  - Store all interpolated vertex attributes
  - Exception: often better to sample and store albedo
    - Otherwise have to store uv's *and gradients;* albedo is smaller
    - But if you were sampling many textures, just store uv's
  - Anything else, simply store in a separate (constant) buffer
    - Then store a single offset ("material ID") in the G-buffer

Theoretical Bandwidth

- Store (deferred):
  - 8-16 additional bytes * overdraw stored per pixel
  - 8-16 bytes read per pixel
  - (Aside: don’t clear your G-buffers, only depth!)
- Resubmit (forward):
  - Vertex positions, bones, constants, etc. read twice
  - Alpha testing potentially done twice
  - Z-buffer read more due to overdraw in shading phase
- Deferred tends to write more, forward tends to read more

Measured Bandwidth

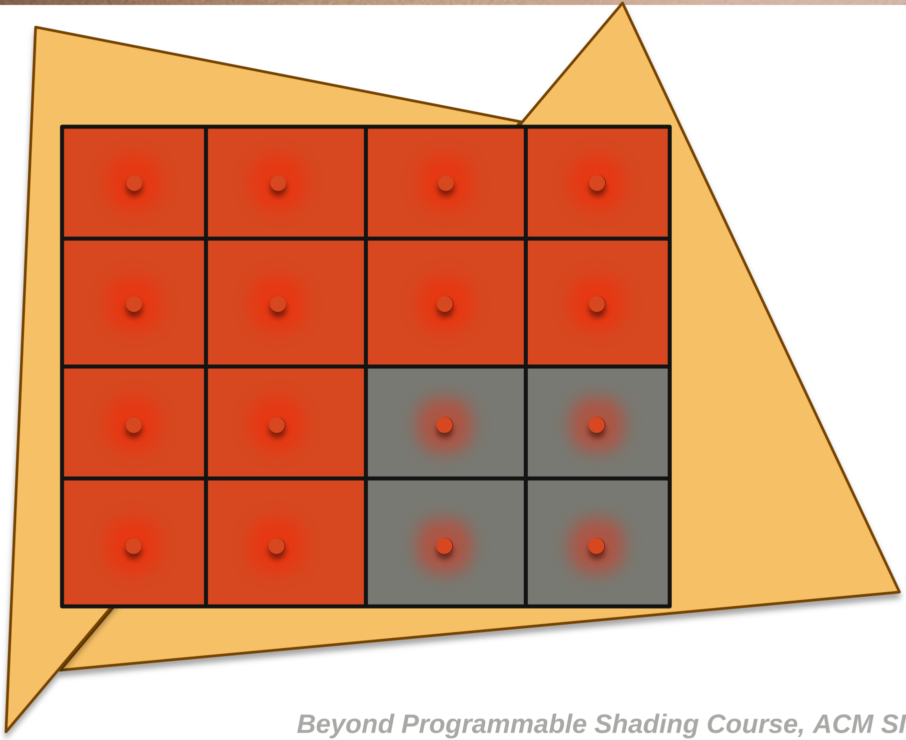
SIGGRAPH2012

- Game scene 1 @ 1080p, ~600k triangles, 1024 lights
  - Tiled Forward: 188MB read, 76MB written, **265MB** total
  - Tiled Deferred: 151MB read, 127MB written, **278MB** total
- Game scene 2 @ 1080p, ~650k triangles, 1024 lights
  - Tiled Forward: 206MB read, 104MB written, **310MB** total
  - Tiled Deferred: 151MB read, 183MB written, **334MB** total
- Not a significant difference in these scenes

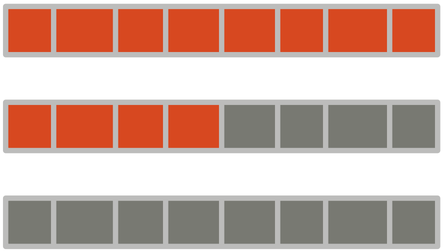
Hardware vs. Software Scheduling

- Rendering pipeline (shaders) is a nice abstraction
  - But to compare forward/deferred, need to think of hardware
  - This is increasing true for all graphics algorithms work
- Scheduling deferred multi-frequency shading
  - Efficient deferred MSAA with software scheduling
  - See [Lauritzen 2010]
- Applies to the general case as well

Pixel Shader Scheduling



SIMD Vectors



24 invocations for 16 pixels

= 33% waste!

Software Scheduling

SIGGRAPH2012

- Group and reorder computation in compute
  - Append work to one or more lists in local memory
  - Sort/compact lists as needed
  - Re-index work items across compact lists
- Very efficient execution of following code
  - But some overhead to doing the repacking

Hardware vs. Software Scheduling

- Pros of hardware scheduling:
  - Dedicated hardware means less overhead
  - Will tend to match memory layout of textures/render targets
- Pros of software scheduling:
  - No wasted “helper pixels” at triangle edges
  - Can schedule based on arbitrary criteria

Forward or Deferred Guidelines

SIGGRAPH2012

- Complex geometry tends to favor deferred
  - Forward does vertex shading, tessellation, alpha test twice
  - Forward wastes cycles on helper pixels with small triangles
- Deferred is more flexible
  - Can combine terms in any manner
  - Avoid needing to have all data resident at once
- MSAA tends to favor forward
  - Overhead in software scheduling in deferred

Forward or Deferred Conclusions

- Relatively orthogonal to choice of light culling
  - Tiled forward and tiled deferred are the same algorithm
- If you use a pre Z pass, you are “deferring” something
  - Only difference is what you store and reconstruct
  - These are “details”… test and do whatever is the fastest
- Again, ideal renderer should just do this transparently
  - Switch forward/deferred even per-object

Conclusions

- Tiled lighting is far better than conventional techniques
  - Tiled deferred and tiled forward are variants
  - Difference is just performance; test several options
- Fixed-function pipeline is not “magic”
  - Think about how algorithms map to hardware
  - … **not** how you implement it in the API!

Acknowledgements

SIGGRAPH2012

- Kayvon Fatahalian for collaborating on this presentation
- Johan Andersson (DICE)
- Aaron Lefohn and Richard Huddy (Intel)
- Various people for game assets to test with

References

- Johan Andersson, *Parallel Graphics in Frostbite – Current and Future*, SIGGRAPH 2009. http://s09.idav.ucdavis.edu
- Takahiro Hirada, Jay McKee, and Jason Yang, *Forward+: Bringing Deferred Lighting to the Next Level*, GDC 2012. http://developer.amd.com/gpu_assets/AMD_Demos_LeoDemoGDC2012.ppsx
- Andrew Lauritzen, *Deferred Rendering for Current and Future Rendering Pipelines*, SIGGRAPH 2010. http://bps10.idav.ucdavis.edu/
- Ola Olsson, Markus Billeter and Ulf Assarsson, *Clustered Deferred and Forward Shading*, HPG 2012. http://www.cse.chalmers.se/~olaolss/main_frame.php?contents=publication&id=clustered_shading
- Ola Olsson and Ulf Assarsson, *Tiled Shading*, Journal of Graphics, GPU and Game Tools 2011. http://www.cse.chalmers.se/~olaolss/main_frame.php?contents=publication&id=tiled_shading
- http://aras-p.info/blog/2012/03/27/tiled-forward-shading-links/
- http://mynameismjp.wordpress.com/2012/03/31/light-indexed-deferred-rendering/

Backup

Conventional Forward Rendering

SIGGRAPH2012

- Intersect light volumes with object bounding volumes
  - Generates a list of lights per-object
  - Assumes “objects” correlates roughly to pixels
- But it doesn’t always… thus inefficient culling
  - Impossible to pick a good culling granularity
  - Near lights/objects are large: need more culling
  - Far lights/objects are small: need less culling

Image-Space Culling

- Intersect light volumes with pixels
- Classic deferred shading
  - Reload BRDF inputs from G-buffer for every light
  - Blend contributions for every light
  - Inefficient use of off-chip memory bandwidth
- Want to evaluate and accumulate all lights at once
  - But with image space culling