10-12 JULY 2012
develop
in brighton

Join the conversation

@developconf #developconf

facebook.com/DevelopConference

Solving Some Common Problems in a Modern Deferred Rendering Engine

Jose Luis Sanchez Bonet

Tomasz Stachowiak

/* @h3r2tic */

Deferred rendering – pros and cons

- Pros *(some)*
  - Very scalable
  - No shader permutation explosion
  - G-Buffer useful in other techniques
    - SSAO, SRAA, decals, …
- Cons *(some)*
  - Difficult to use multiple shading models
  - Does not handle translucent geometry
    - Some variants do, but may be impractical

Reflectance models

- The BRDF defines the look of a surface
  - Bidirectional Reflectance Distribution Function

$$L_o = L_e + \int_\Omega L_i \cdot \boldsymbol{f_r} \cdot \cos\theta \cdot \delta\omega$$

- Typically games use just one (Blinn-Phong)
  - Simple, but inaccurate
- Very important in physically based rendering
  - Want more: Oren-Nayar, Kajiya-Kay, Penner, Cook-Torrance, …

BRDFs vs. rendering

- Forward rendering
  - Material shader directly evaluates the BRDF
    - Trivial
- Deferred rendering
  - Light shaders decoupled from materials
  - No obvious solution

BRDFs vs. deferred – branching?

- Read shading model ID in the lighting shader, branch
- Might be the way to go on next-gen
- Expensive on current consoles
  - Tax for branches never taken
    - Don't want to pay it for every light

Platform	1 BRDF	2 BRDFs	3 BRDFs
360	1.85 ms	2.1 ms (+ 0.25 ms)	2.35 ms (+ 0.5 ms)
PS3	1.9 ms	2.48 ms (+ 0.58 ms)	2.8 ms (+ 0.9 ms)

Three *different* BRDFs, only one used
(branch always yields the first one)

BRDFs vs. deferred – LUTs?

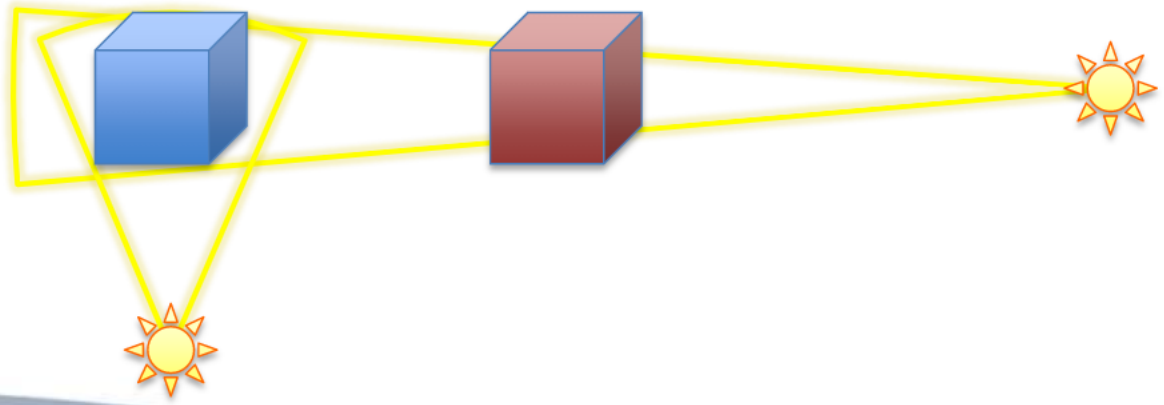
- Pre-calculate BRDF look-up tables
- Might be shippable enough
  - See: S.T.A.L.K.E.R.
- Limited control over parameters
  - Roughness
  - Anisotropy, etc.
- BRDFs highly dimensional
  - Isotropic with roughness control → 3D LUT

BRDFs vs. deferred – our approach

- One default BRDF
  - Others a relatively rare case
- Shading model ID in stencil
- Multi-pass light rendering
- Mask out parts of the scene in each pass

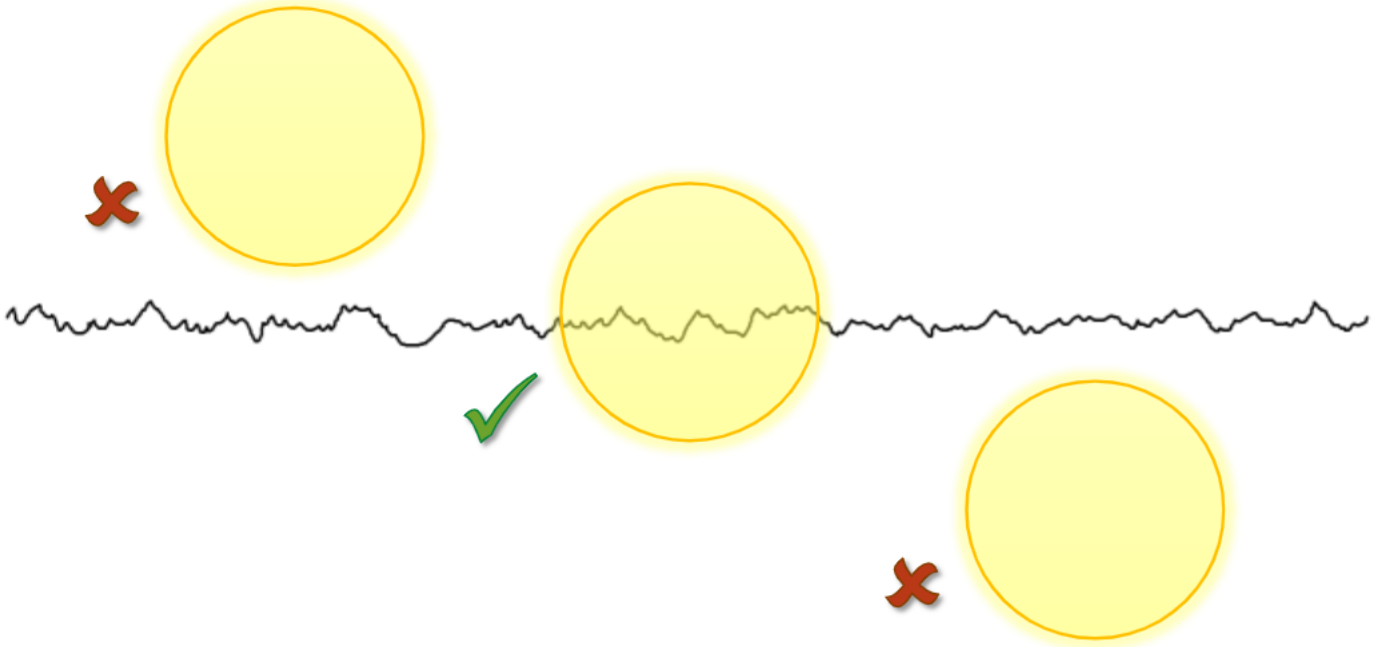
Multi-pass – tax avoidance

- For each light
  - Find all affected BRDFs
  - Render the light volume once for each model
- Analogous to multi-pass forward rendering!
- Store bounding volumes of objects with non-standard BRDFs
  - Intersect with light volumes



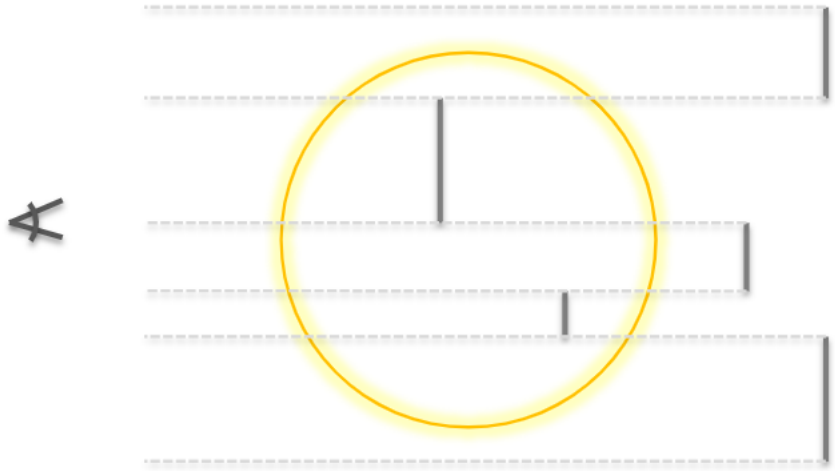
Making it practical

- Needs to work with depth culling of lights
- Hierarchical stencil on 360 and PS3



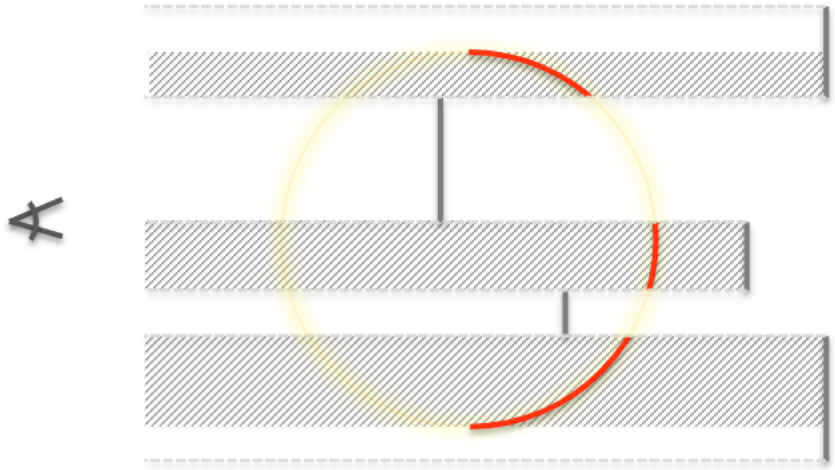
Depth culling of lights

- *Assuming viewer is outside the light volume*
- Start with stencil = 0
- Render back faces of light volume
  - Increment stencil; no color output
- Render front faces
  - Only where stencil = 0; write color
- Render back faces
  - Clear stencil; no color output



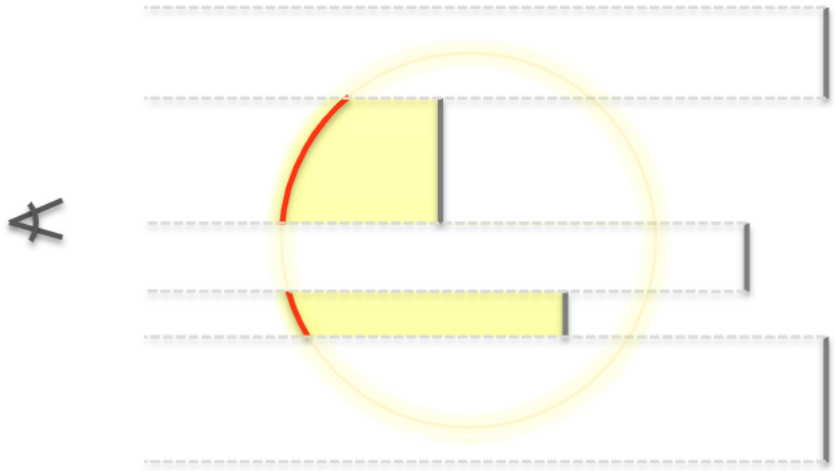
Depth culling of lights

- *Assuming viewer is outside the light volume*
- Start with stencil = 0
- Render back faces of light volume
  - Increment stencil; no color output
- Render front faces
  - Only where stencil = 0; write color
- Render back faces
  - Clear stencil; no color output



Depth culling of lights

- *Assuming viewer is outside the light volume*
- Start with stencil = 0
- Render back faces of light volume
  - Increment stencil; no color output
- Render front faces
  - Only where stencil = 0; write color
- Render back faces
  - Clear stencil; no color output



Culling with BRDFs

- Pack the culling bit and BRDF together
- Use masks to read/affect required parts
- Assuming 8 supported BRDFs:

Unused				BRDF ID			Culling bit
7	6	5	4	3	2	1	0

```
culling_mask = 0x01
brdf_mask    = 0x0E
brdf_shift   = 1
```

Light volume rendering passes

- Render back faces
  - Increment stencil; no color output
- For each BRDF:
  - Render front faces
    - hi-stencil ref = brdf_id << brdf_shift
    - no color out; write hi-stencil
  - Render front faces using the light shader
    - test hi-stencil
- Render back faces
  - Zero stencil; no color output

Handling miscellaneous data in stencil

- Stencil value may contain extra data
  - Used in earlier / later rendering passes

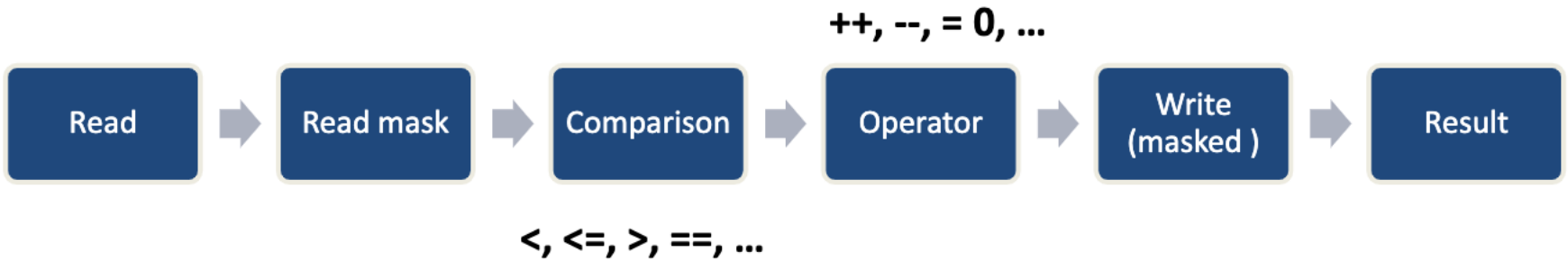
Garbage				BRDF ID			Culling bit
7	6	5	4	3	2	1	0

  - Need to ignore it somehow
  - Stencil read mask?

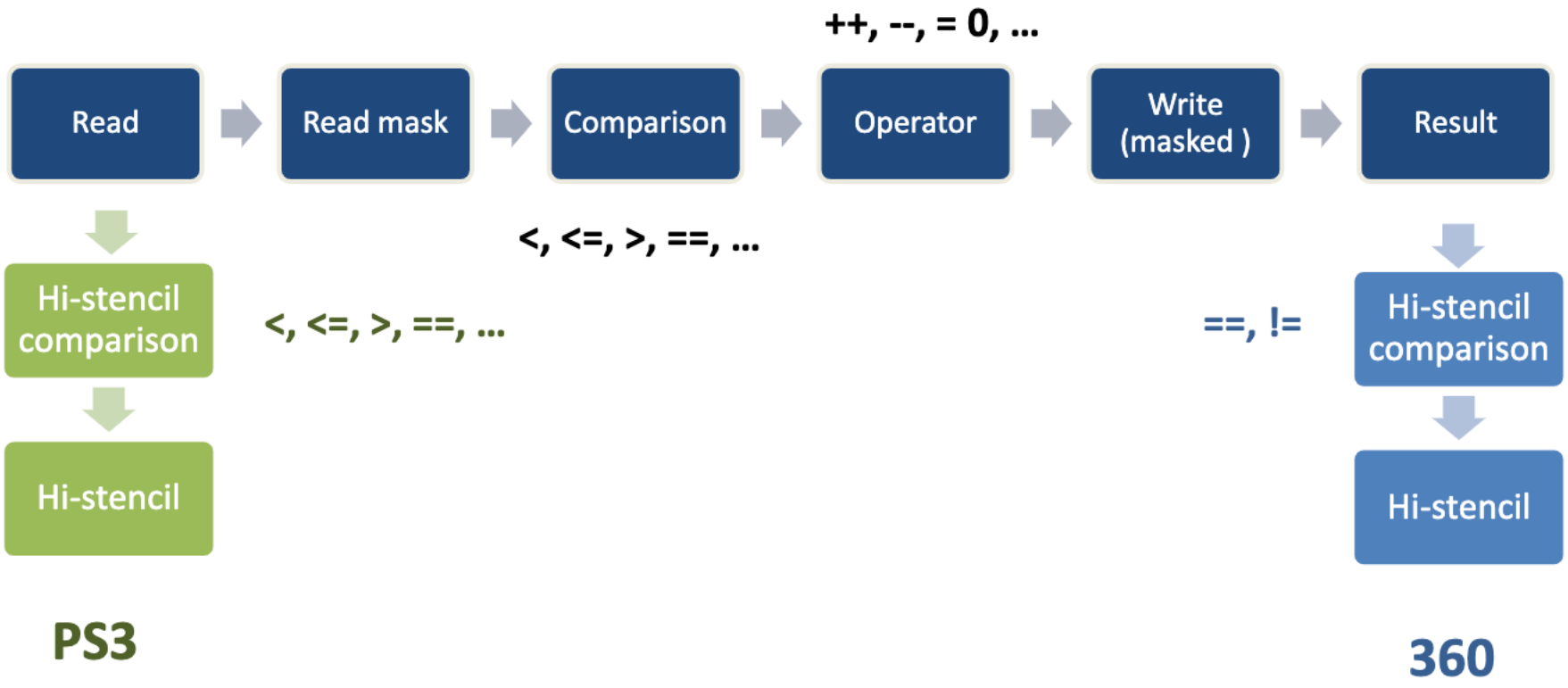
For each BRDF:
Stencil reference = brdf_id << brdf_shift
Stencil read mask ~= garbage_mask ???

- Doesn't work with the 360's hi-stencil

Stencil operation



Hierarchical stencil operation



Spanner in the works

Render back faces

- Increment stencil; no color output

For each BRDF:

- Render front faces
  - **hi-stencil ref = brdf_id << brdf_shift**
  - no color out; write hi-stencil
- Render front faces using the light shader
  - test hi-stencil

Render back faces

- Zero stencil; no color output

Breaks if stencil contains garbage we can't mask out

Handling stencil garbage

- Can't do it in a non-destructive manner
  - Take off and nuke the entire site from orbit
  - It's the only way to be sure
- Extra cleaning pass?
  - Don't want to pay for it!
- Do it as we go!
  - `D3DRS_STENCILPASS ← D3DSTENCILOP_ZERO`
  - `D3DRS_STENCILWRITEMASK ← (BYTE)~brdf_mask`

- Save your stencil if you need it
  - Sorry for calling it garbage :`(
  - We were already restoring it later on the 360
  - Don't need to destroy it on the PS3, use a read mask!

Performance

Platform	1 BRDF	2 BRDFs	3 BRDFs
360 (branching)	1.85 ms	2.1 ms (+ 0.25 ms)	2.35 ms (+ 0.5 ms)
360 (stencil)	**1.85 ms**	**1.99 ms (+ 0.14 ms)**	**2.13 ms (+ 0.28 ms)**
PS3 (branching)	1.9 ms	2.48 ms (+ 0.58 ms)	2.8 ms (+ 0.9 ms)
PS3 (stencil)	**1.9 ms**	**2.13 ms (+ 0.23 ms)**	**2.31 ms (+ 0.41 ms)**

		For each BRDF		
Platform	Initial setup	Mask	Render	Cleanup
360	0.03 ms	0.1 ms	>= 0.036 ms	0.022 ms
PS3	0.03 ms	0.1 ms	>= 0.06 ms	0.14 ms

Multi-pass light rendering – final notes

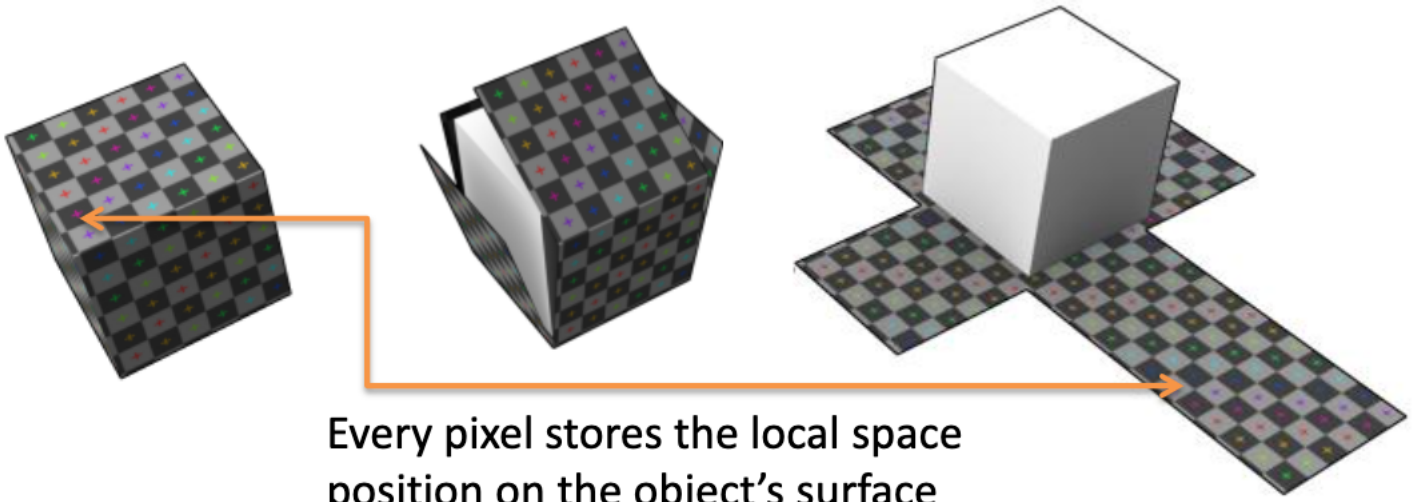
- No change in single-BRDF rendering
  - Use your madly optimized routines
- No need for a 'default' shading model
  - It's just our use case
  - As long as you efficiently find influenced BRDFs
- Flush your hi-stencil
  - `FlushHiZStencil( D3DFHZS_ASYNCHRONOUS );`
  - `cellGcmSetClearZcullSurface( _, CELL_GCM_FALSE, CELL_GCM_TRUE );`
- Tiny lights? Try branching instead.
  - Performance figures only from huge lights!
  - With tiny lights, hi-stencil juggling becomes inefficient

Lighting alpha objects in deferred rendering engines

- Classic solutions:
  - Forward rendering.
  - CPU based, one light probe per each object.
- Our solution:
  - GPU based.
  - More than one light probe.
  - Calculate a lightmap for each object each frame.
  - Used for objects and particle systems.
  - Fits perfectly into a deferred rendering pipeline.

Our solution for alpha objects

- Object space map:



Every pixel stores the local space position on the object's surface

Our solution for alpha objects

- For each object:
  - Use baked positions as light probes
    - Transform object space map into world space
  - Render lights, reusing deferred shading code
  - Accumulate into lightmap
  - Render object in alpha pass using lightmap

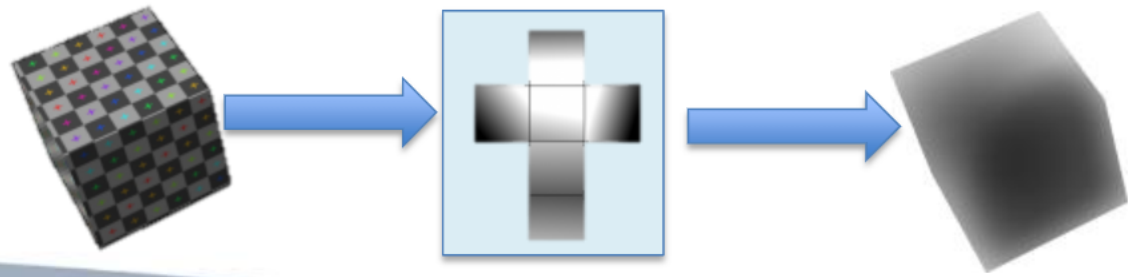
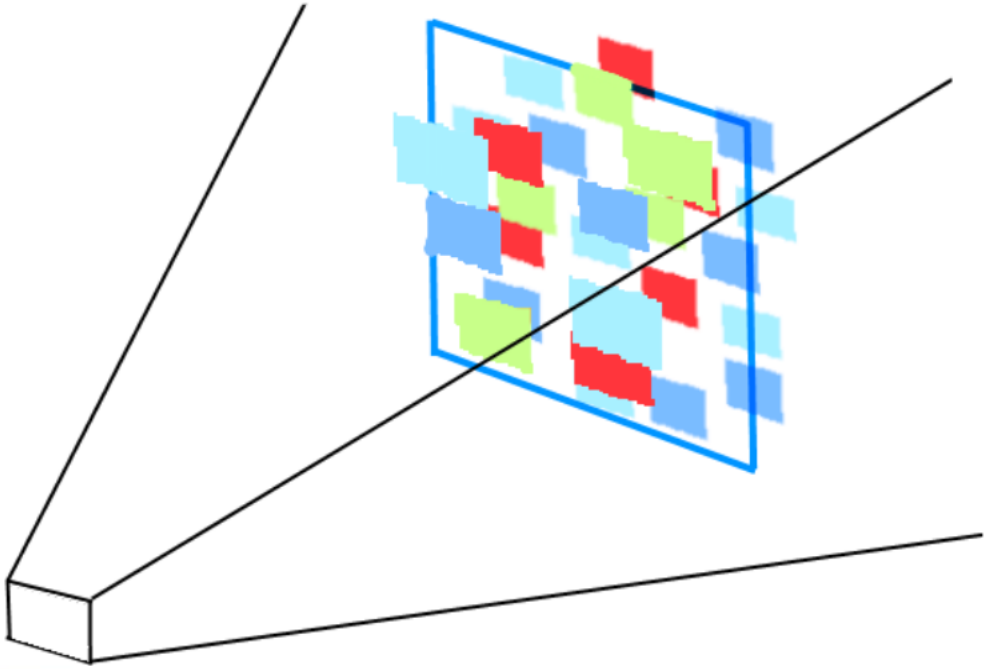


Image attribution: Zephyris at en.wikipedia.

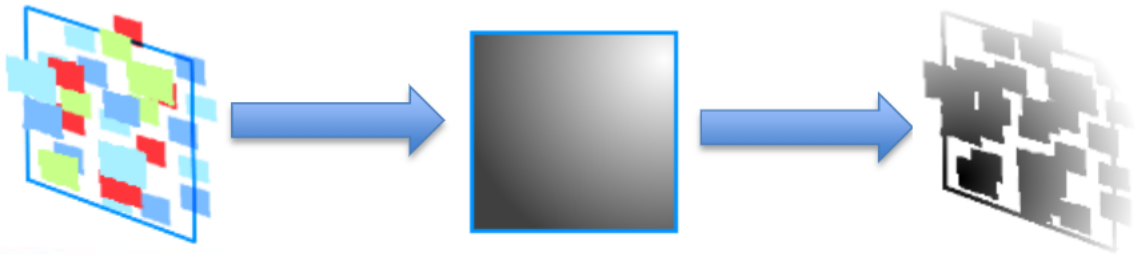
Our solution for particle systems

- Camera oriented quad fitted around and centered in the particle system.



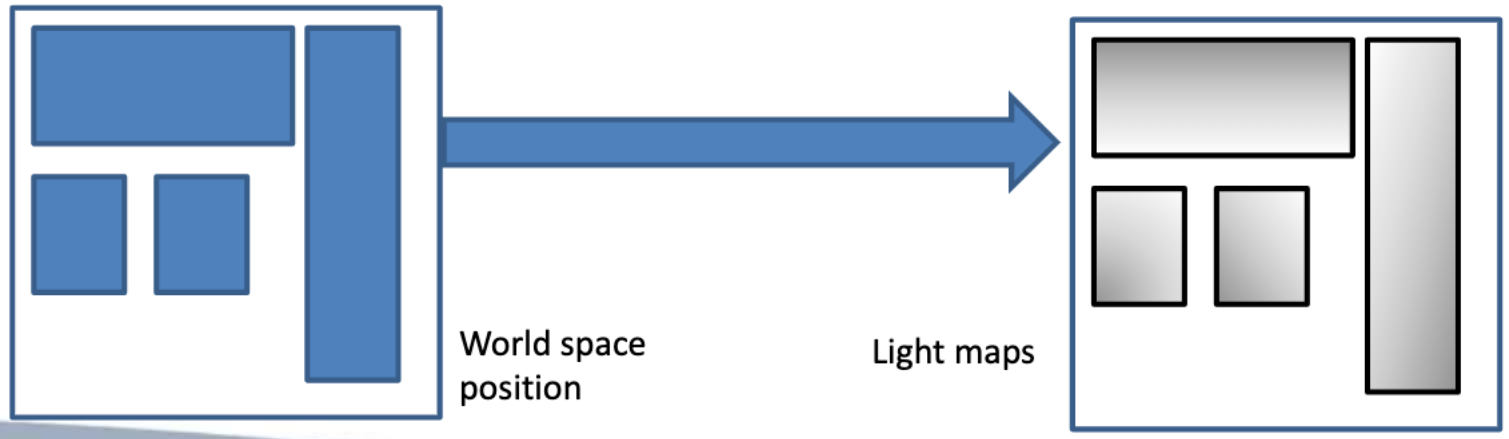
Our solution for particle systems

- For each particle system:
  - Allocate a texture quad and fill it with interpolated positions as light probes
  - Render lights and accumulate into lightmap
  - Render particles in alpha pass, converting from clip space to lightmap coordinates.



Implementation details

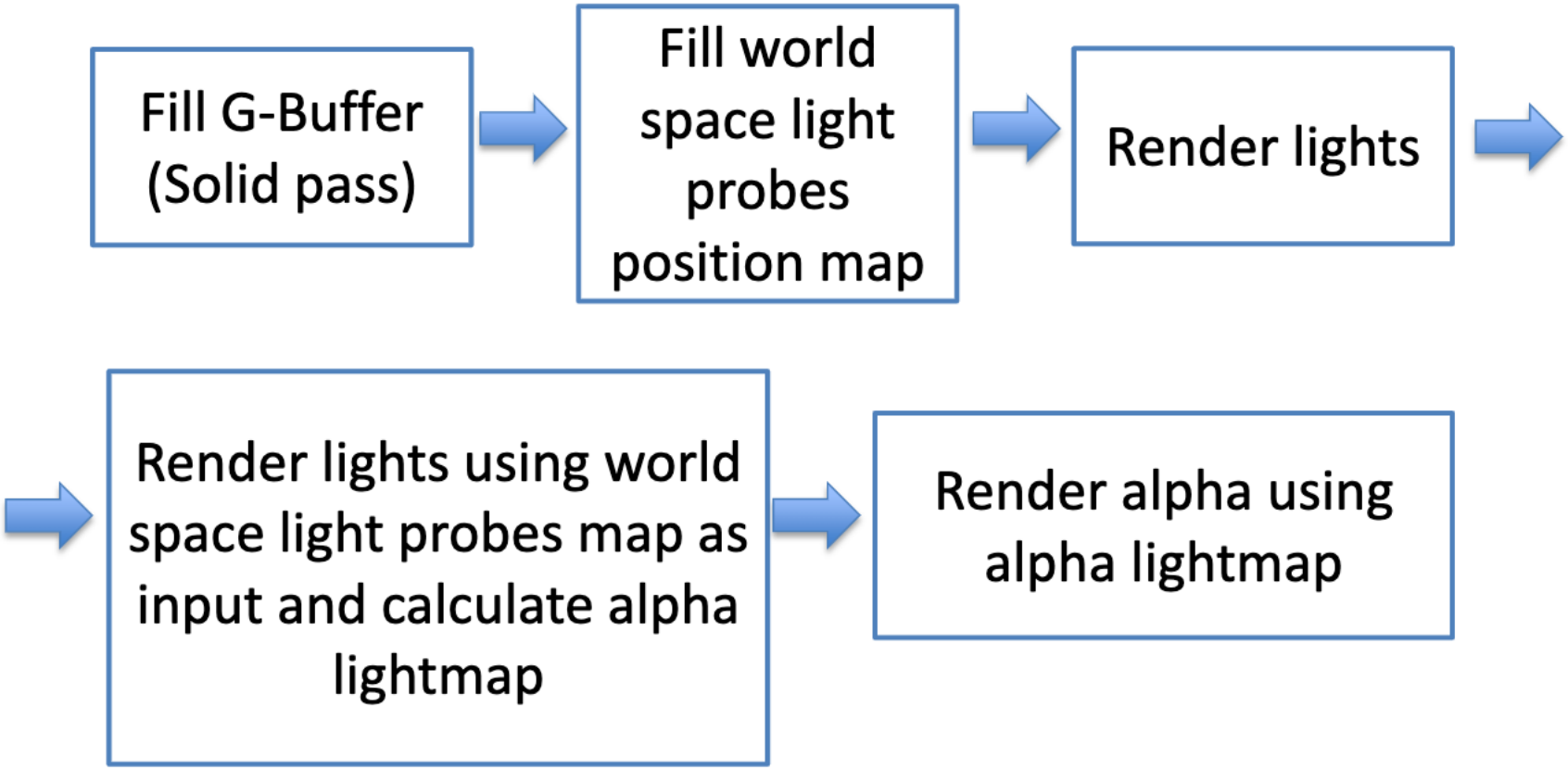
- For performance reasons we pack all position maps to a single texture.
- Every entity that needs alpha lighting will allocate and use a region inside the texture.



Integration with deferred rendering

Deferred rendering

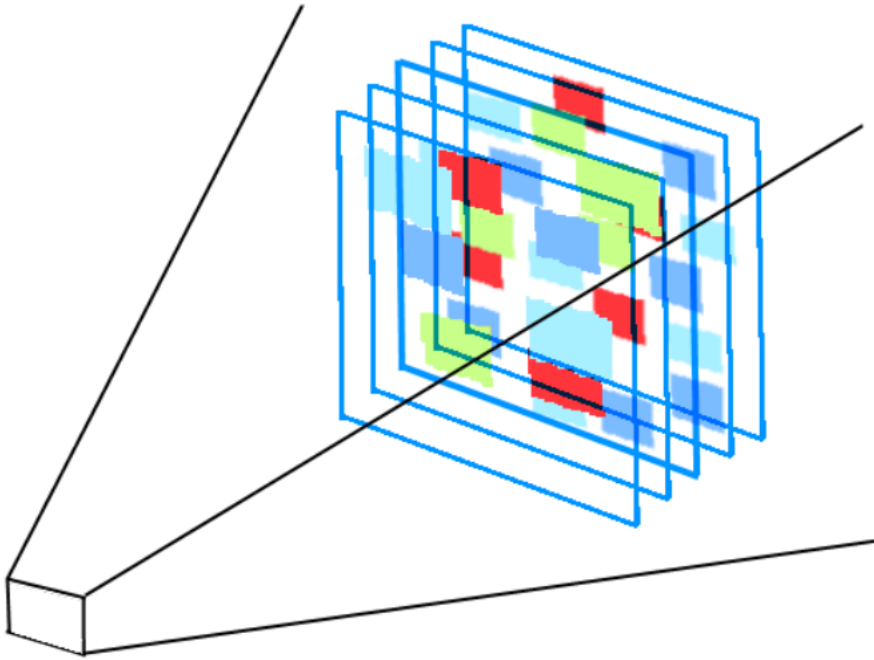
Our solution



Improvements

- Calculate a second texture with light direction information.
- Other parameterizations for particle systems:
  - Dust (one pixel per mote).
  - Ribbons (a line of pixels).
- 3D volume slices for particle systems.
  - Allocate a region for every slice
  - Adds depth to the lighting solution.

3D volume slices



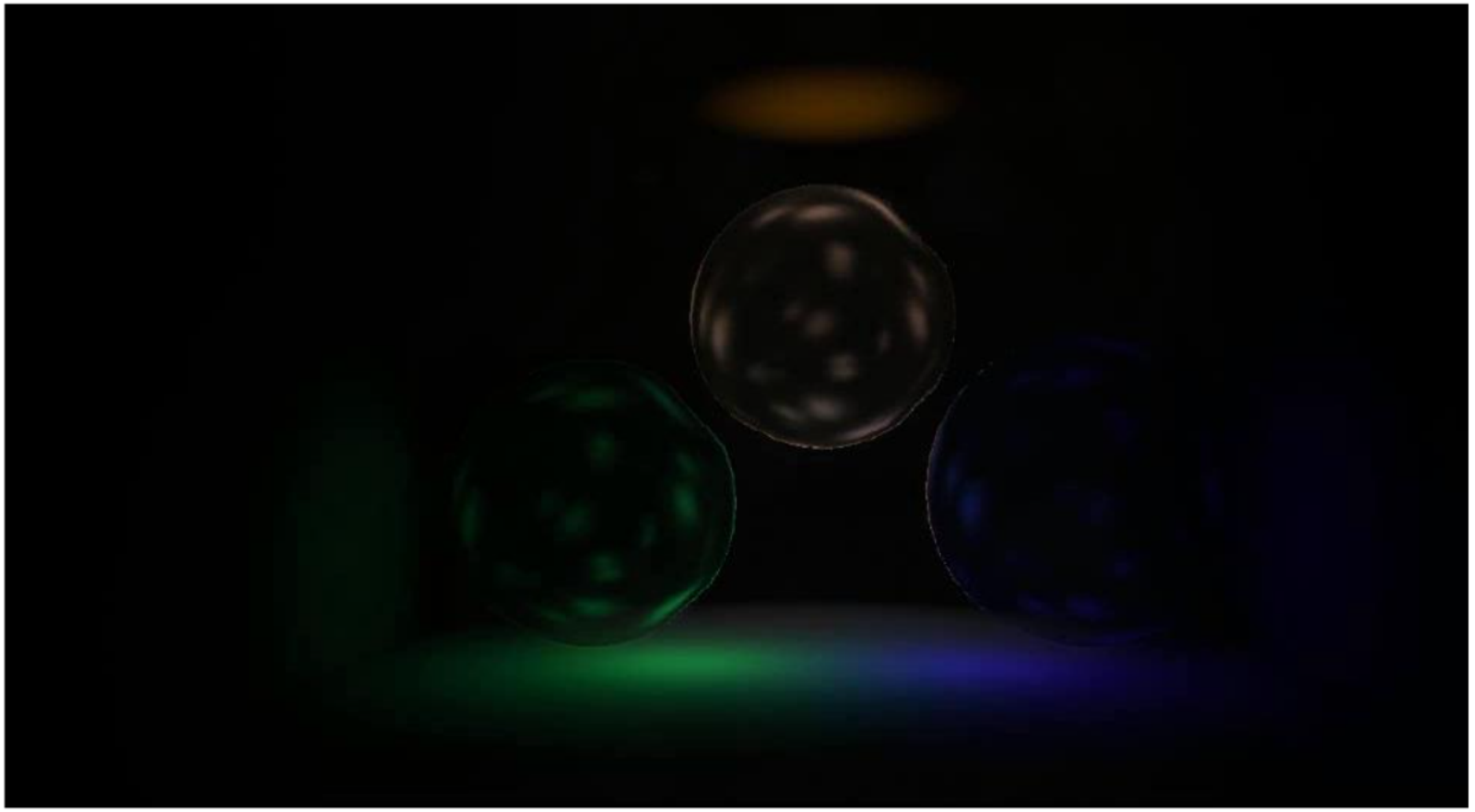
For each slice we allocate one region

Slice 0 map

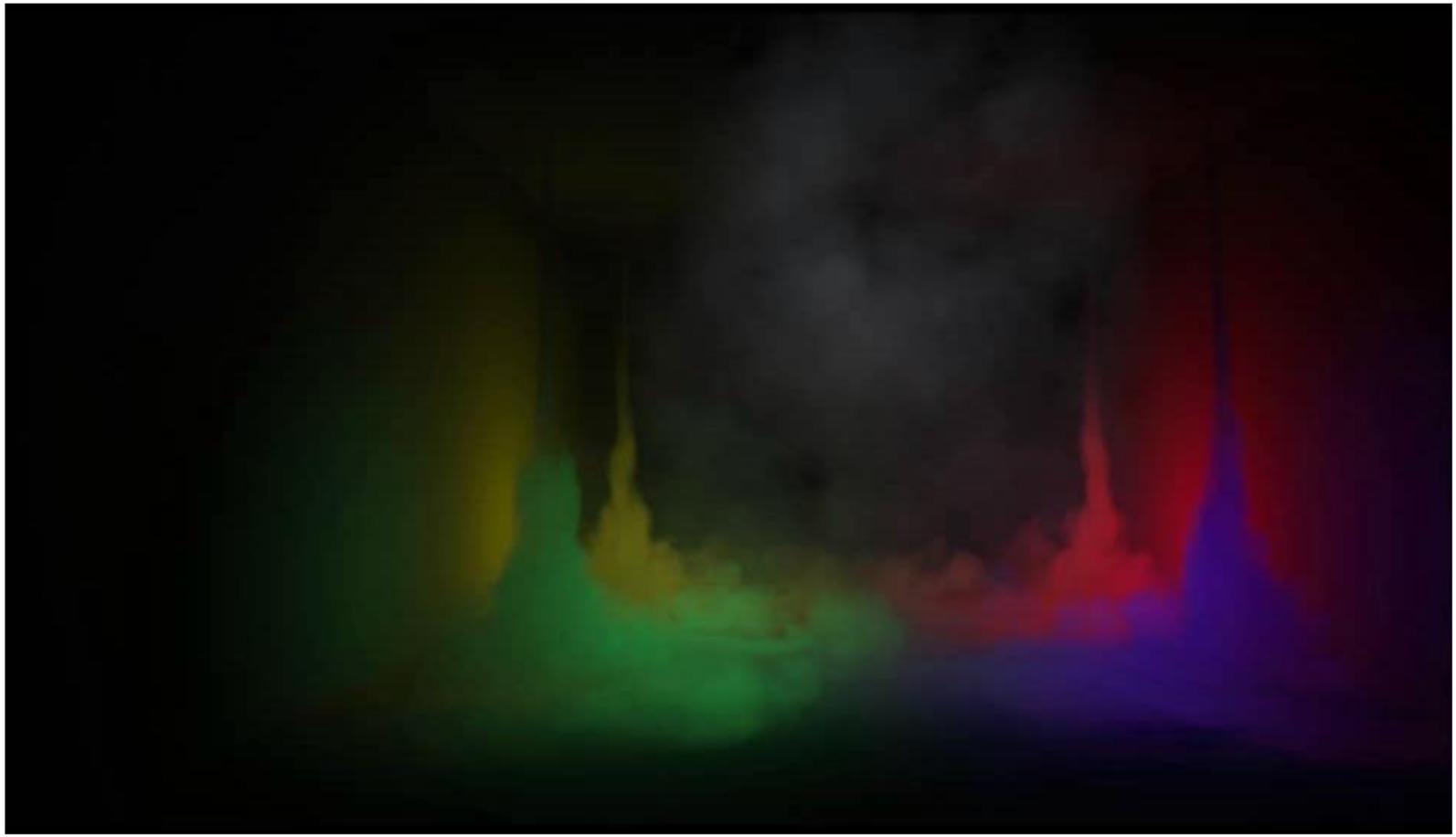
.
.
.

Slice n map

Demo



Demo



WE ARE HIRING!

http://www.creative-assembly.com/jobs/

the CREATIVE ASSEMBLY

SEGA®

Questions?

Jose Luis Sanchez Bonet
jose.sanchez@creative-assembly.com

Tomasz Stachowiak
tomasz.stachowiak@creative-assembly.com
twitter: h3r2tic