SIGGRAPH2011

Rendering in Cars 2

Chris Hall

Christopher.Hall@disney.com

Rob Hall

Robert.W.Hall@disney.com

Dave Edwards

David.W.Edwards@disney.com

Advances in Real-Time Rendering in Games

SIGGRAPH2011

Cars 2 Motivation

- Different gameplay demands different technology

Differences

Toy Story 3	Cars 2
Platforming	Racing
2 player split screen	4 player split screen
Average 30fps	Essential to maintain 30fps

Toy Story 3	Cars 2
All dynamic lighting	Lightmaps, Light probes, limited dynamic
Dynamic cascaded shadow mapping	Simplified shadowmaps for dynamic only
SSAO, Depth of Field, Glow, God Rays, Sparkle, Bloom, Deferred Ambient	HDR, Bloom, Motion Blur, Color Correction

Outline

- Light Probes
- HDR color precision
- Early stencil shadow culling
- PS3 Post Processing

Light Probes

Chris Hall

Advances in Real-Time Rendering in Games

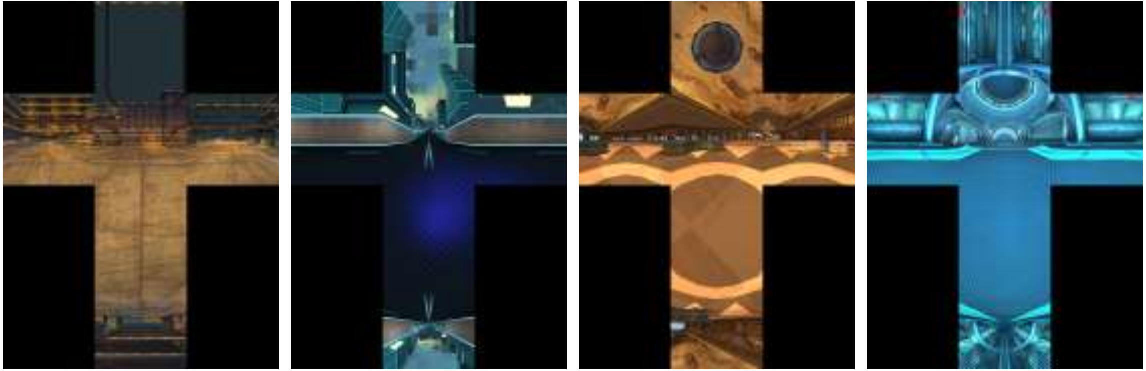
SIGGRAPH2011

Motivation

- 4 Players
- Lightmaps for all world geometry
- Real-time lighting didn't match

Light probes

- Capture light from a point in space
- Bounce lighting
- Environment Mapping



Global probe

- Used for bounce lighting for outdoors
- Either artist defined light rig or captured probe

Environment Map

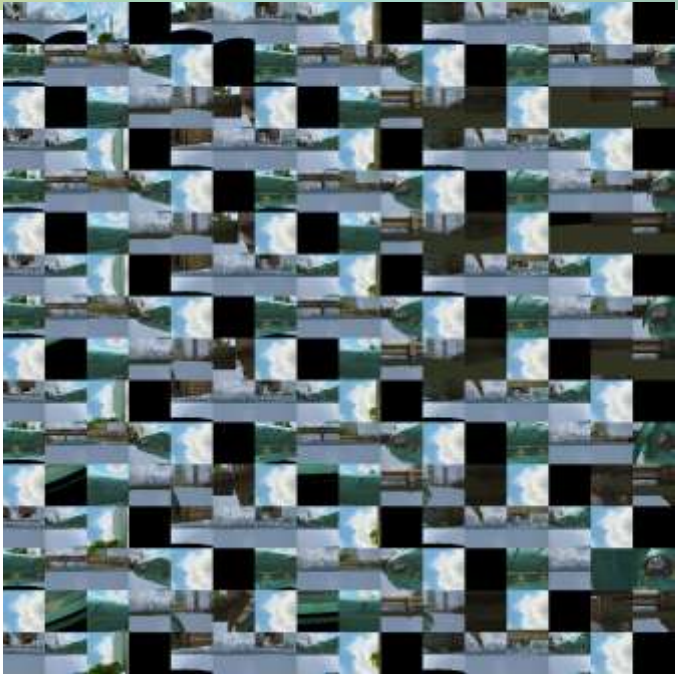
Irradiance Map (SH)

Bounce lighting data

- Store as spherical harmonics
- Order 3 SH = 108 bytes per probe
- Can pack in direct lighting for free

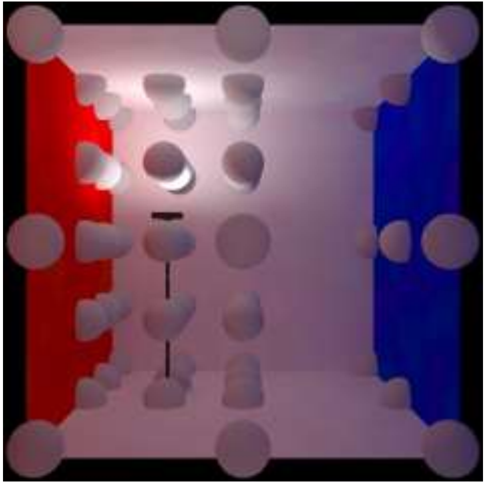
Probe Capture

- Render cubemaps on GPU
- Save as 16F for HDR
- Atlas for speed
- Bounced lighting
  - Cubemap to SH projection [Sloan], DirectX SDK



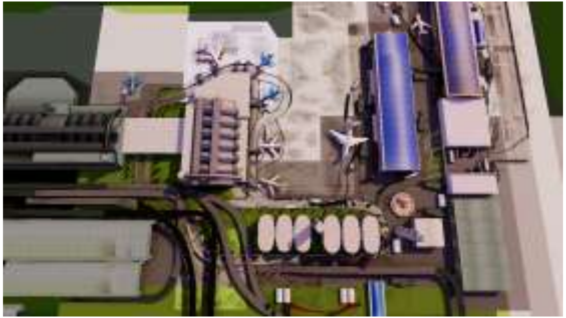
Irradiance Volume

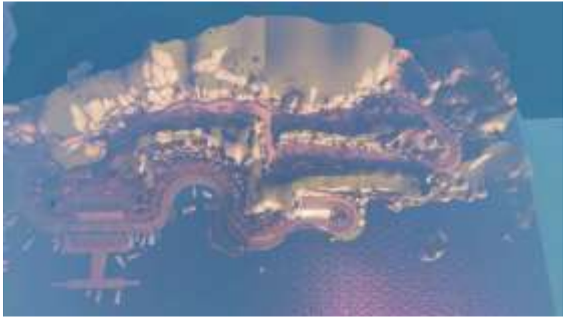
- Volume with a bunch of light probes
- Allows for varied bounced light throughout the world
- Very popular to use with lightmaps
  - [Greger 1998]
  - [Tatarchuk 2005]
  - [Mitchell, McTaggart and Green 2006]



Volume choice

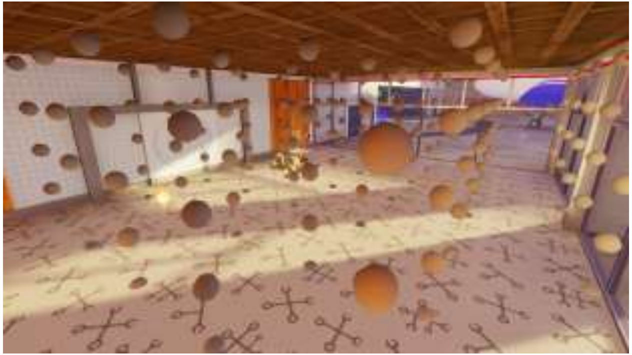
- Racing game
- 2-5 miles of track per world
- Mostly outside
- Lots of thin, curvy areas
- Coverage isn't essential





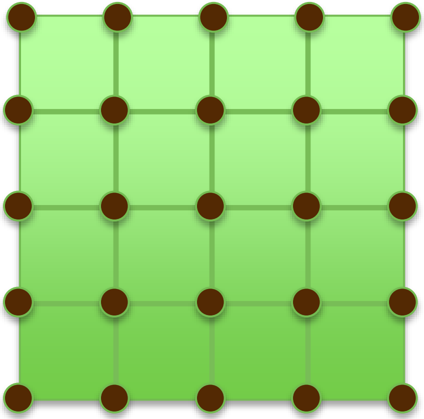
Uniform grid volume

- Box Volume
- Can be rotated and scaled to fit anywhere
- Box split with variable amount of slices (density x/y/z)
- Probes placed along the slices of the volume



Grid Analysis

- Simple structure and easy to implement
- Entire data is saved into a continuous array
- Sample with box intersection tests
  - Can access each probe by an offset
  - O(1) to sample inside the grid
  - Cost is only spent inside volumes
- Wastes space

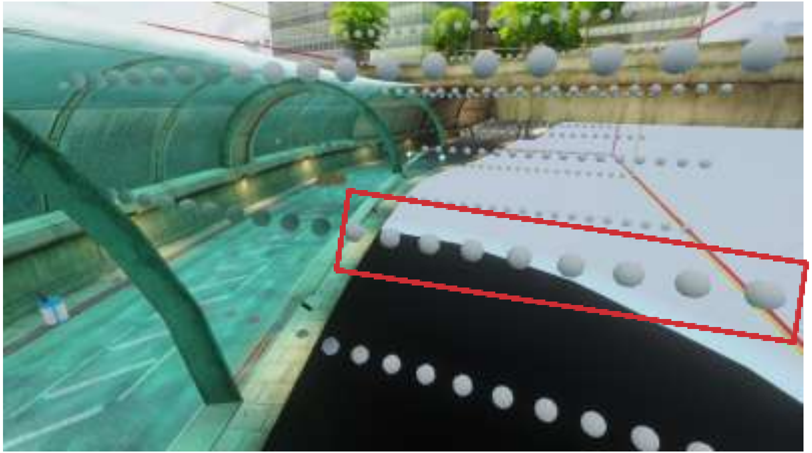


Fading Regions

- Blend between global probe and volume lighting
- Outer Fading Volume
- Inner Fading Volume
- Fading amount

Invalid points

- Probes outside world have incorrect lighting
- How to detect
- Replace with correct lighting
- [Kontkanen and Laine]

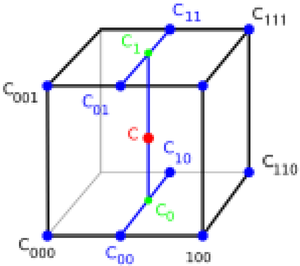


Volume Lookup

- We need some way to light our objects with them
- CPU Based
  - Assign/blend closest SH per mesh
  - Pass SH data through to GPU
- GPU Based
  - Per pixel or per vertex
  - Sample probes on the GPU

CPU Assignment

- For each mesh, sample the lighting at the mesh center
- Intersection tests
  - Create an OBB for each volume
  - Check if center point of mesh is inside
- Trilinear Interpolation

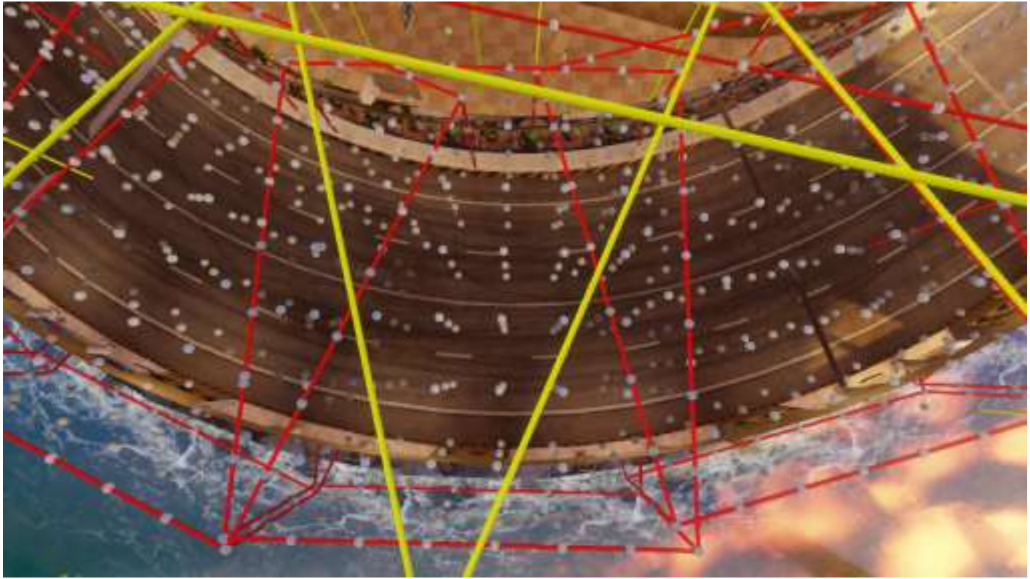


Shader constants

- Per instance shader constants
- Calculate color in shader
- Breaks down for large objects
  - Can break mesh apart
    with vertex color blending
  - Same problem for world lighting

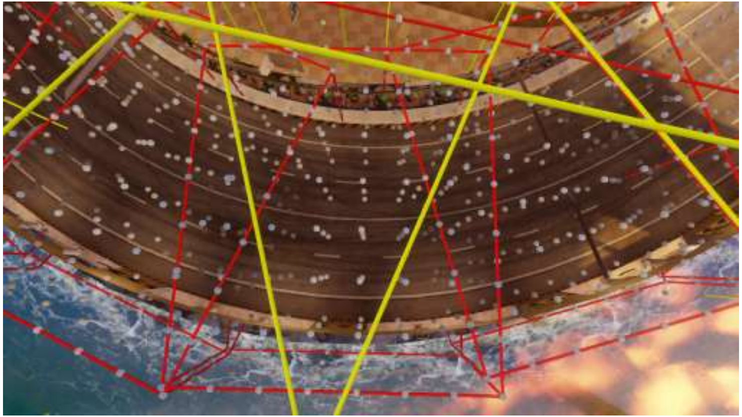
Overlapping Volumes

- Blending is challenging



Time Averaging

- Blend % of the last frame's SH into the current SH
  - Artist adjustable per world
- Trilinear filtering substitute
- Avoid for first frame
- [Mitchell, McTaggart and Green]



Environment Maps

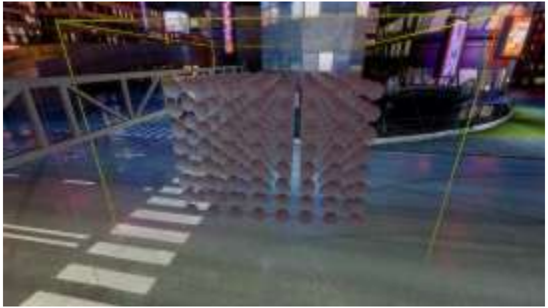
- Probe without a volume
- Worked really well for road reflections
- Assign environment map probes to volumes

Assigning Environment Maps

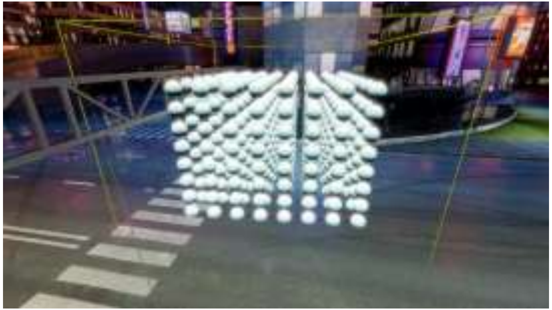
- Use volume’s environment map if inside a volume
- Otherwise, use the global probe
- Switch based on the fading region
- Switch was a pop
- Overlapping volumes avoid pops by sharing cubemaps

Direct Lighting

- Pack direct lighting into probes
  - Can evaluate lighting in SH and add to bounce
- No extra performance cost
- Dependent on grid density



Bounced Lighting



Direct Lighting

Lighting Overrides

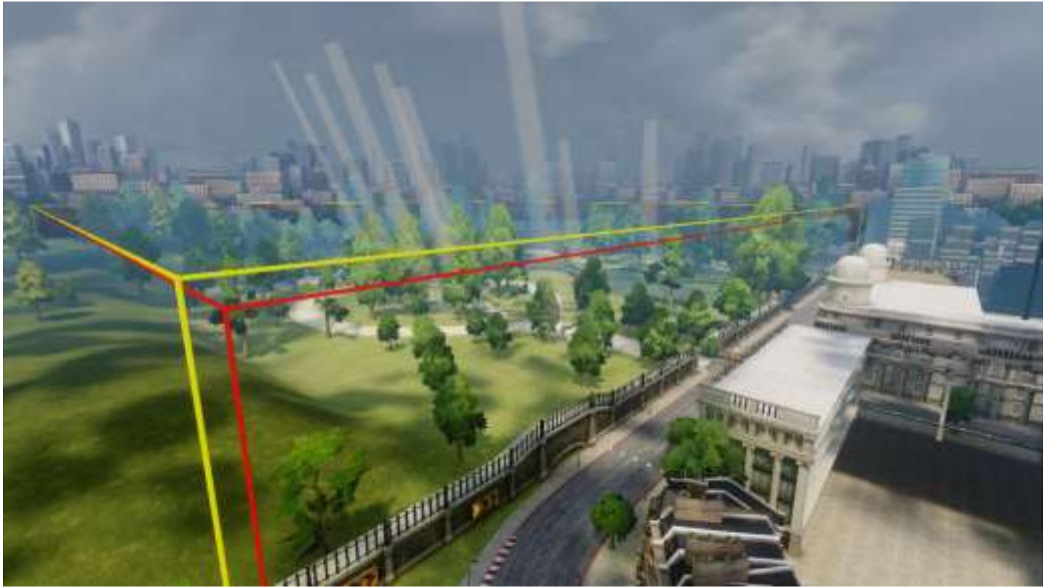
- Directional and ambient lights added if inside volume
- Allows artists to control lighting
- 2d volume
- 1d volume
- Area lights

Artist tricks

- 2d volume

Artist Tricks

- Single point volume



Artist Tricks

SIGGRAPH2011

- Area lights

Ending Thoughts

SIGGRAPH2011

- Uniform grid is easy and fast
- Used little memory and scaled well for 4 players
- Lots of flexibility with artists
- Future ideas
  - GPU lookup of light probes

Trimming the GPU Pipeline

Rob Hall

Advances in Real-Time Rendering in Games

SIGGRAPH2011

Overview

- Reduce cost for HDR rendering
- Reduce shadow cost
  - Scale shadow maps for 4 player split screen
  - Use multi-resolution rendering for deferred shadow mask

HDR requirements

- If possible, use a 32 bits per pixel target format
- Support all hardware alpha blend states
- Limited range from [0,32] is acceptable
- Reduce banding as much as possible

Format Chart

SIGGRAPH2011

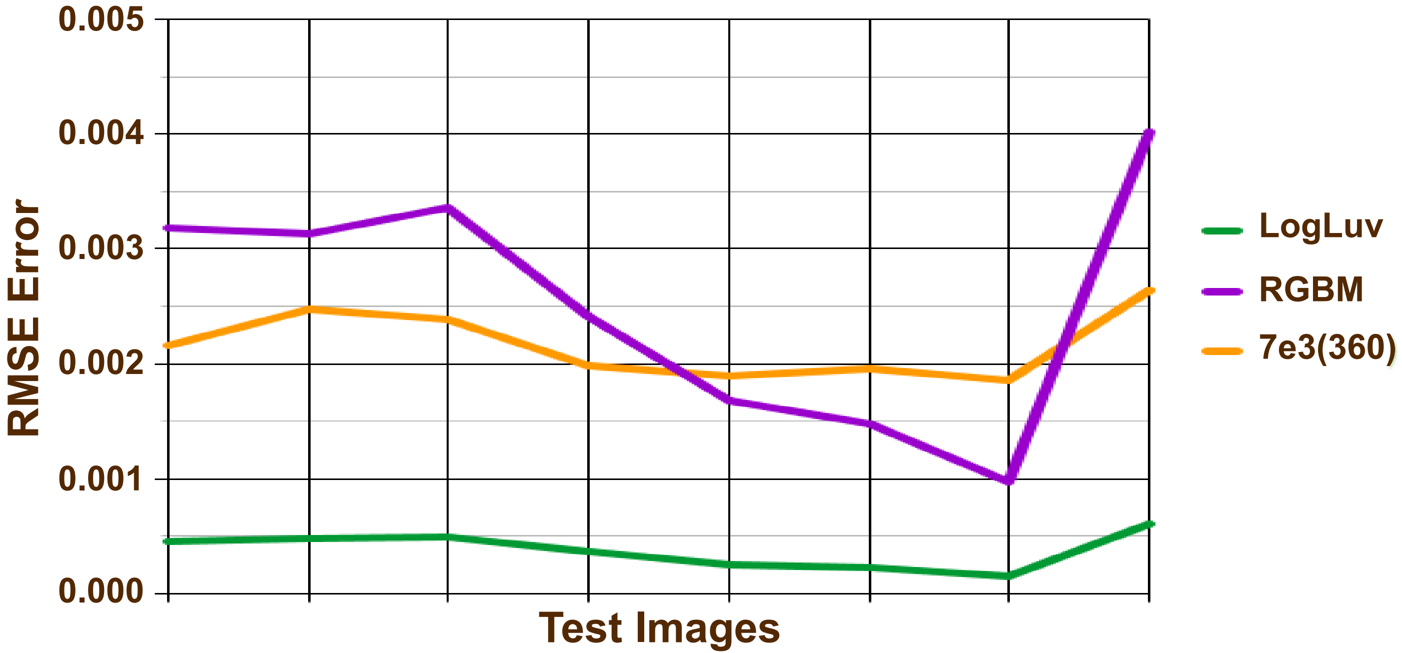
Format	Range	Alpha Blend	Bilinear Filtering	Perf cost
sRGB	[0,1]	Yes	Yes	-
LogLuv [Larson]	[10^19, 10^19]	Limited	No	ALU
RGBM [Karis]	[0,6]	Limited	No, but works OK	ALU
7e3	[0, 31 7/8]	Yes	Yes	Alpha states double fill rate
R11G11B10	[0, 2^16]	Yes	Yes	-
16F	[-2^16, 2^16]	Yes*	Yes*	Double fill rate

* Except Xbox 360

Sample Image 16F linear

And other 16F linear images

HDR format error



Format Chart

Format	Range	Alpha Blend	Bilinear Filtering	Perf cost
sRGB	[0,1]	Yes	Yes	-
LogLuv [Larson]	[10^19, 10^19]	Limited	No	ALU
RGBM [Karis]	[0,6]	Limited	No, but works OK	ALU
7e3	[0, 31 7/8]	Yes	Yes	Alpha states double fill rate
R11G11B10	[0, 2^16]	Yes	Yes	-
16F	[-2^16, 2^16]	Yes*	Yes*	Double fill rate

Xbox 360

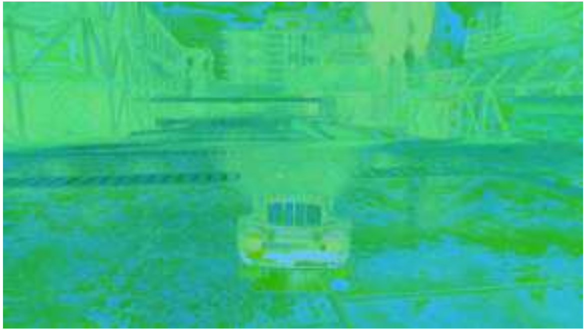
PS3

* Except Xbox 360

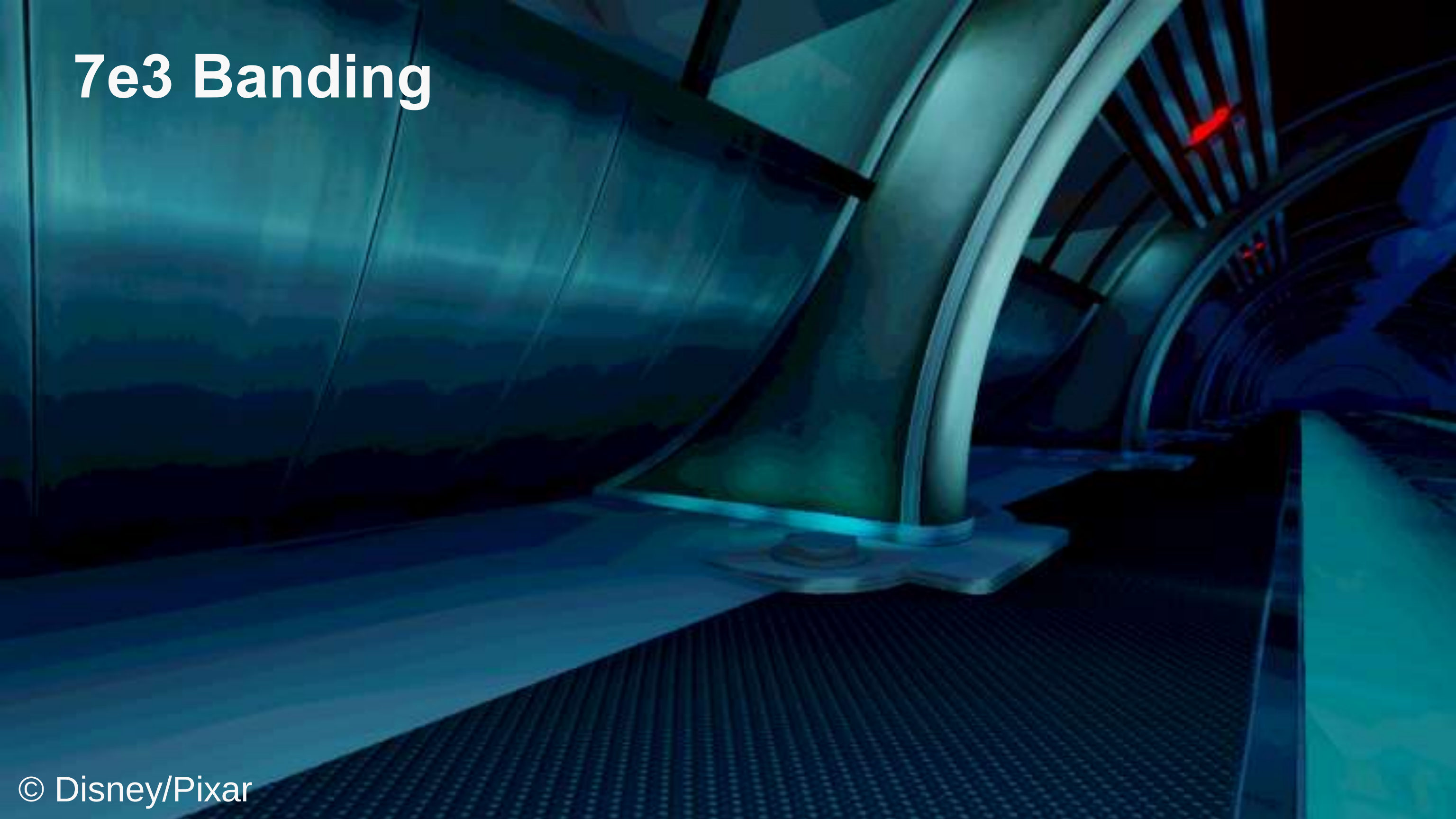
PS3 Luv

SIGGRAPH2011

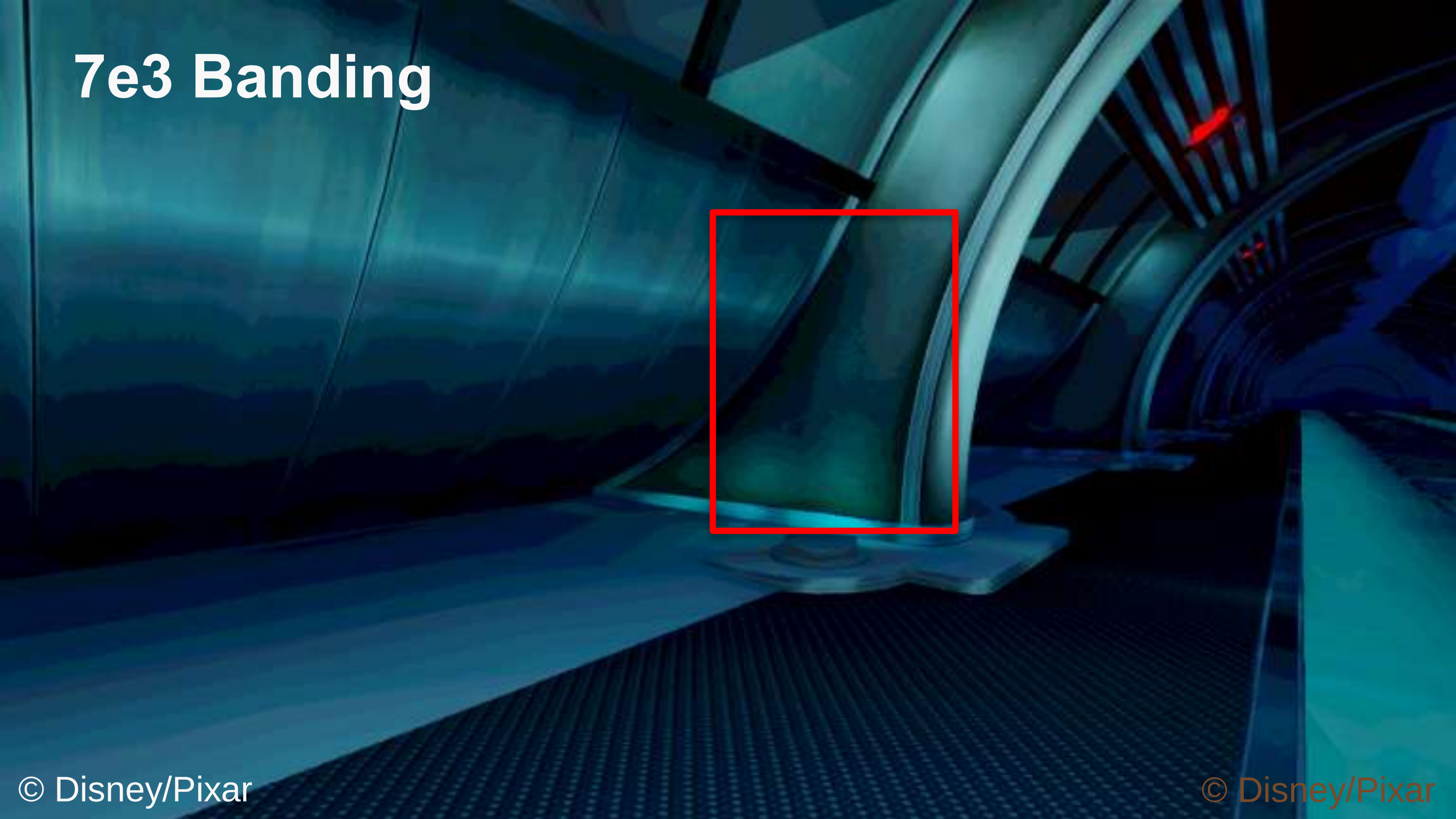
- Similar to LogLuv - compresses 64 bpp to 32bpp
- Encode luminance with a sqrt instead of a log to avoid a costly exp2 operation on the SPU
- Store luminance in 16 bit fixed point, 3 int 13 frac format
- Range is [0,~64] $[0, \left(7 + \frac{8191}{8192}\right)^2]$
- Code sample in Appendix A



7e3 Banding

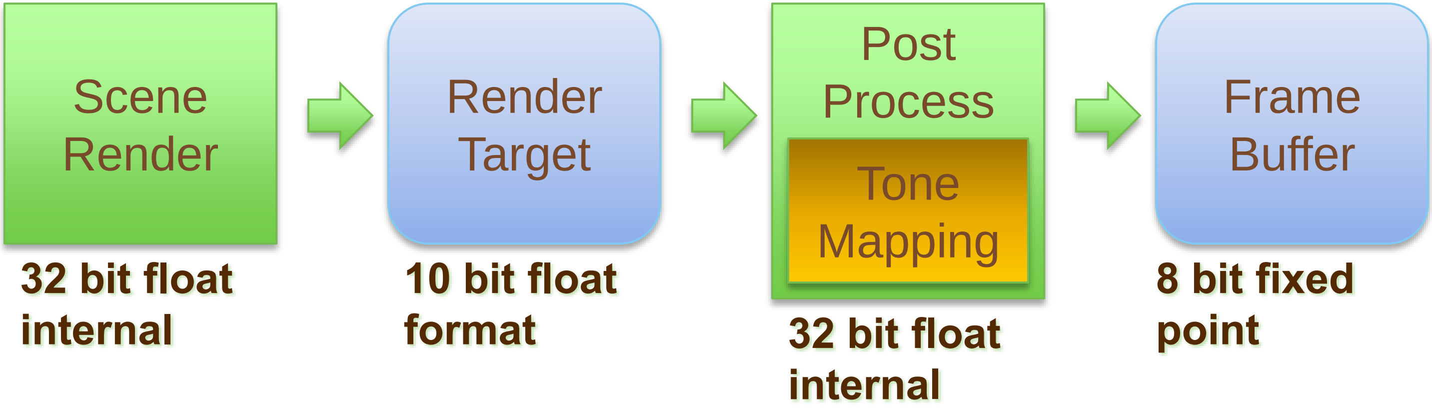


7e3 Banding

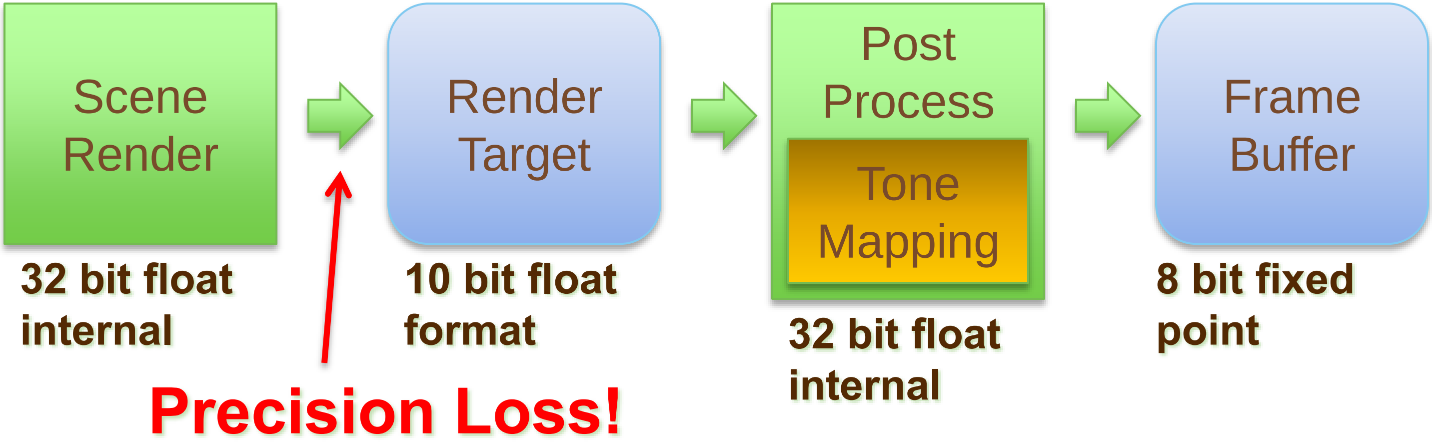


© Disney/Pixar

What’s causing this?



Tone Mapping



Tone Mapping Components

Tone Mapping

- Exposure
- Operator

Exposure

```
Target_Color = Scene_Color
*(Expected_Exposure /
Prev_Frame_Avg_Luminance)
```

Simplified Operator [Hable] [Reinhard]

```
Tone_Mapped_Color =
Target_Color / (1.0f +
Target_Color)
```

Pre-exposed color

Scene Render
Exposure

32 bit float internal

→

Render Target

10 bit float format

→

Post Process
Tone Mapping Operator

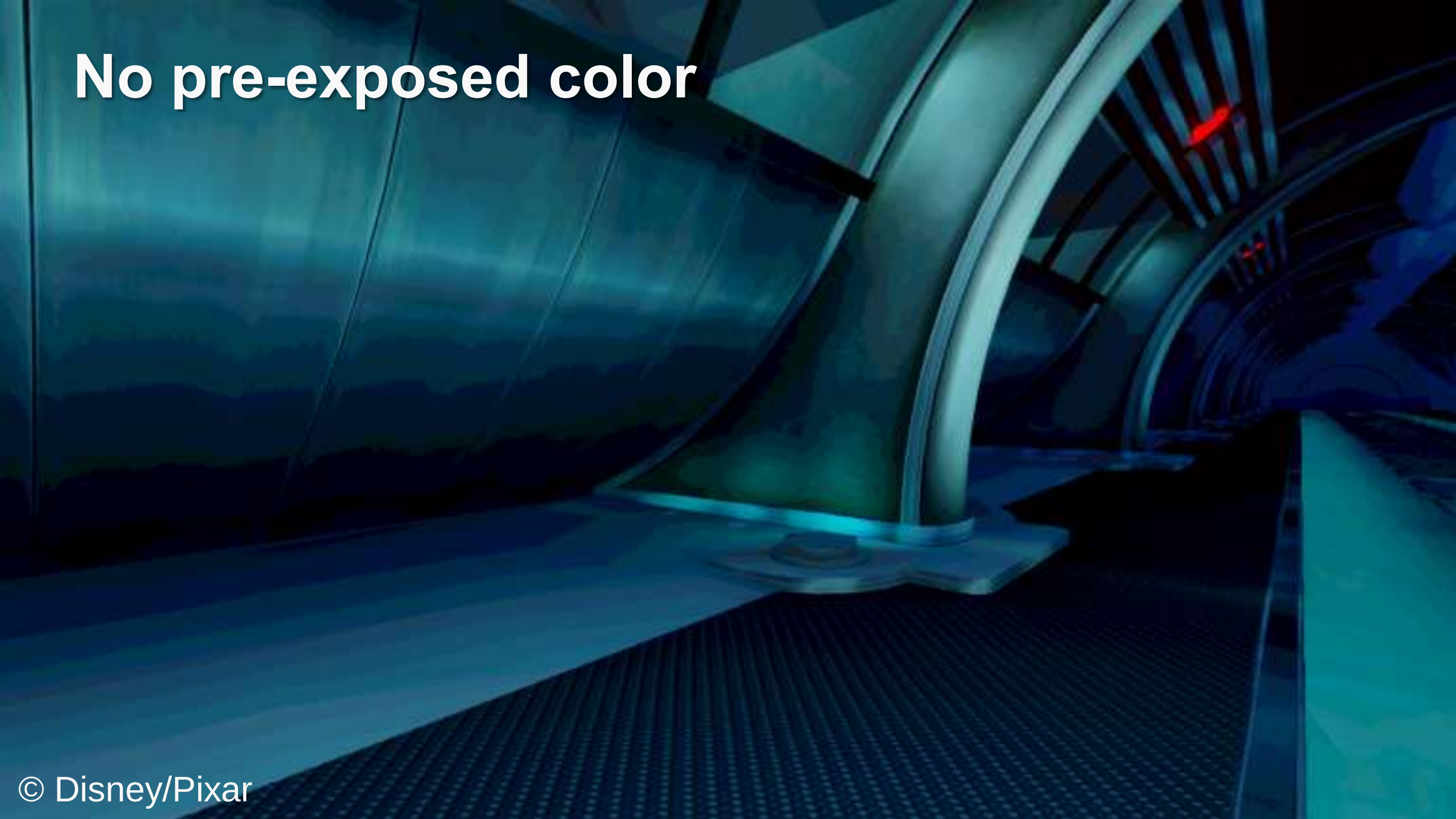
32 bit float internal

→

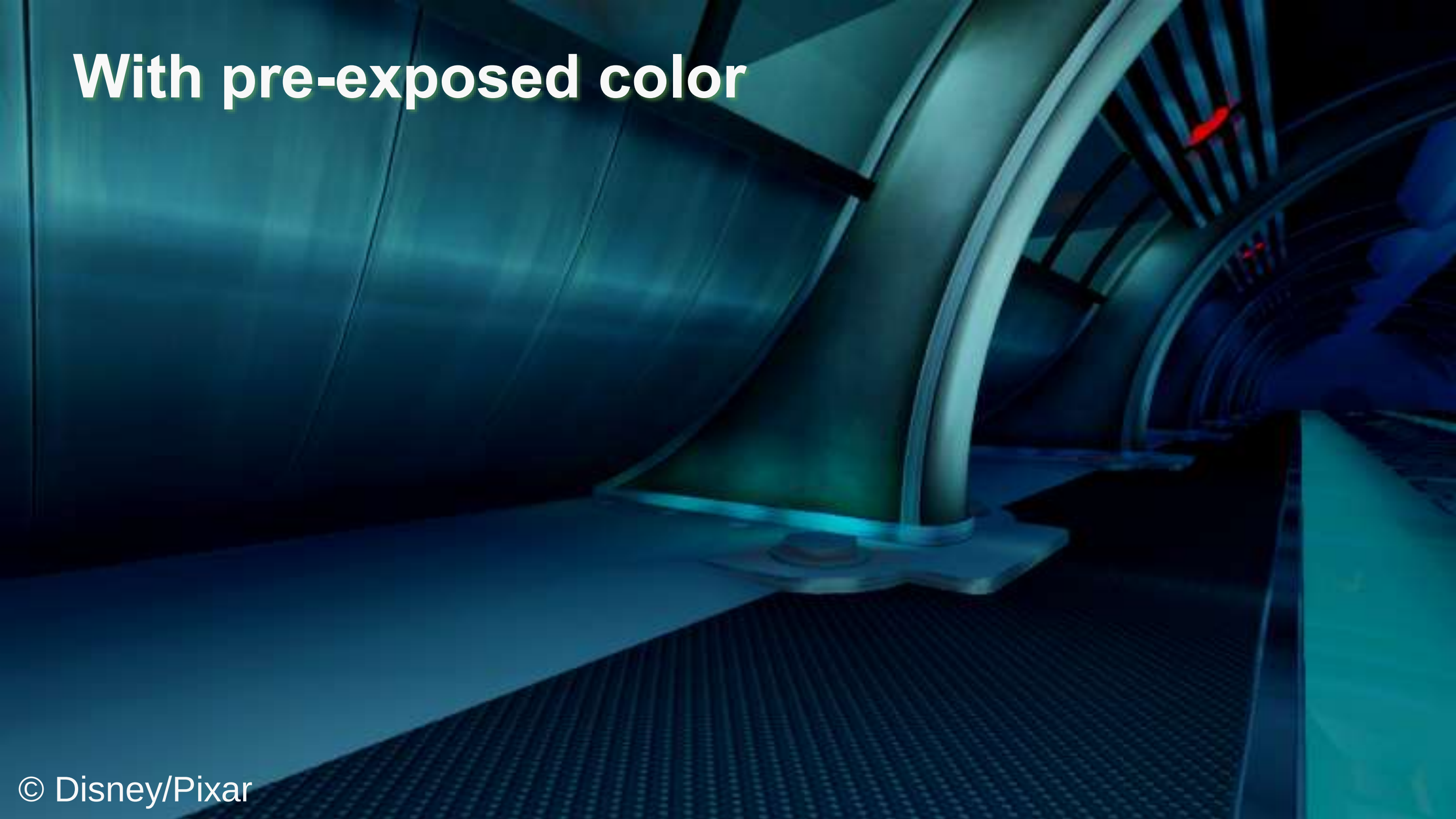
Frame Buffer

8 bit fixed point

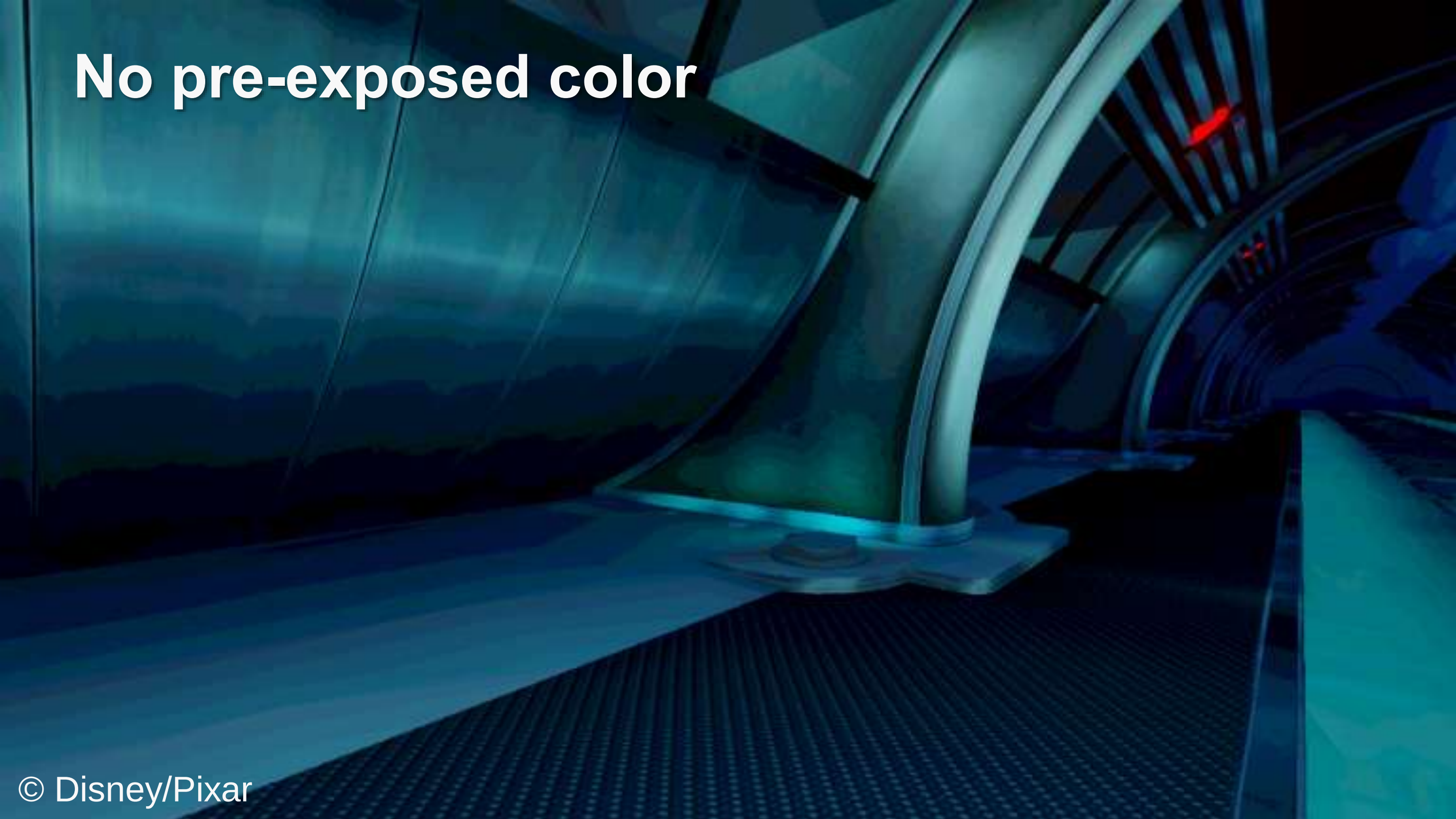
No pre-exposed color



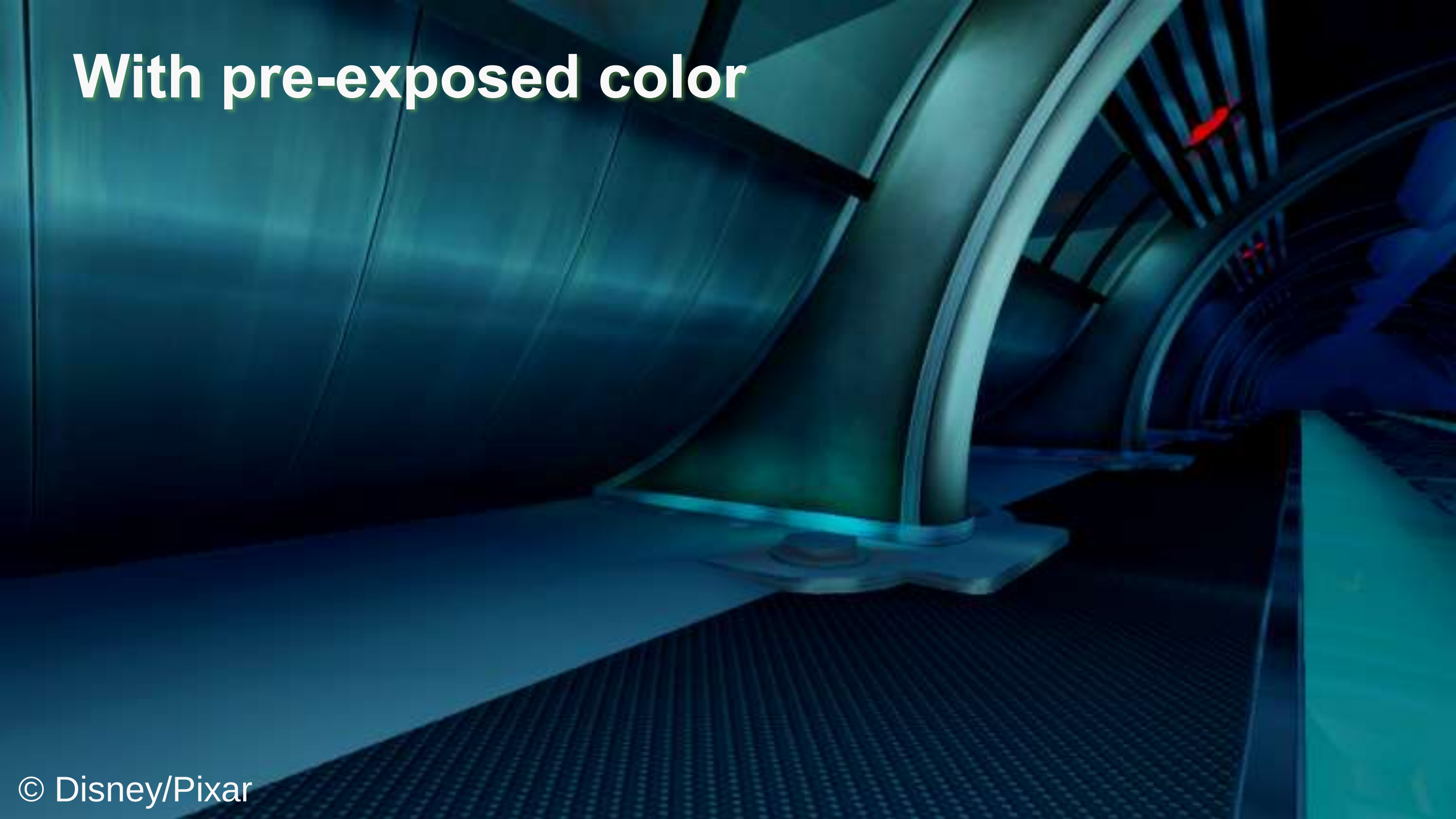
With pre-exposed color



No pre-exposed color



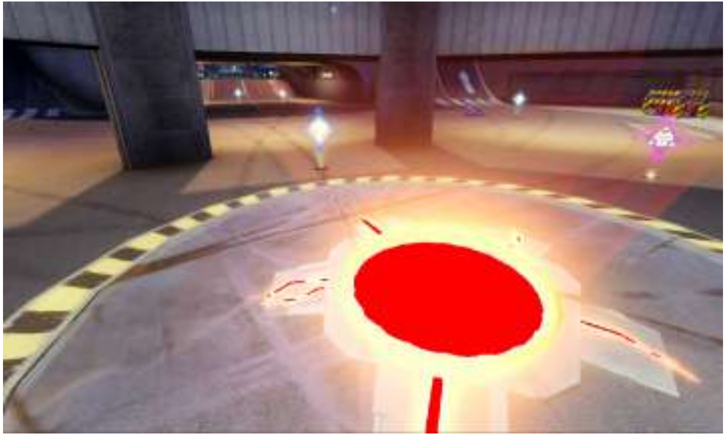
With pre-exposed color



Artifacts

- Overall range is clamped

Clamped Image



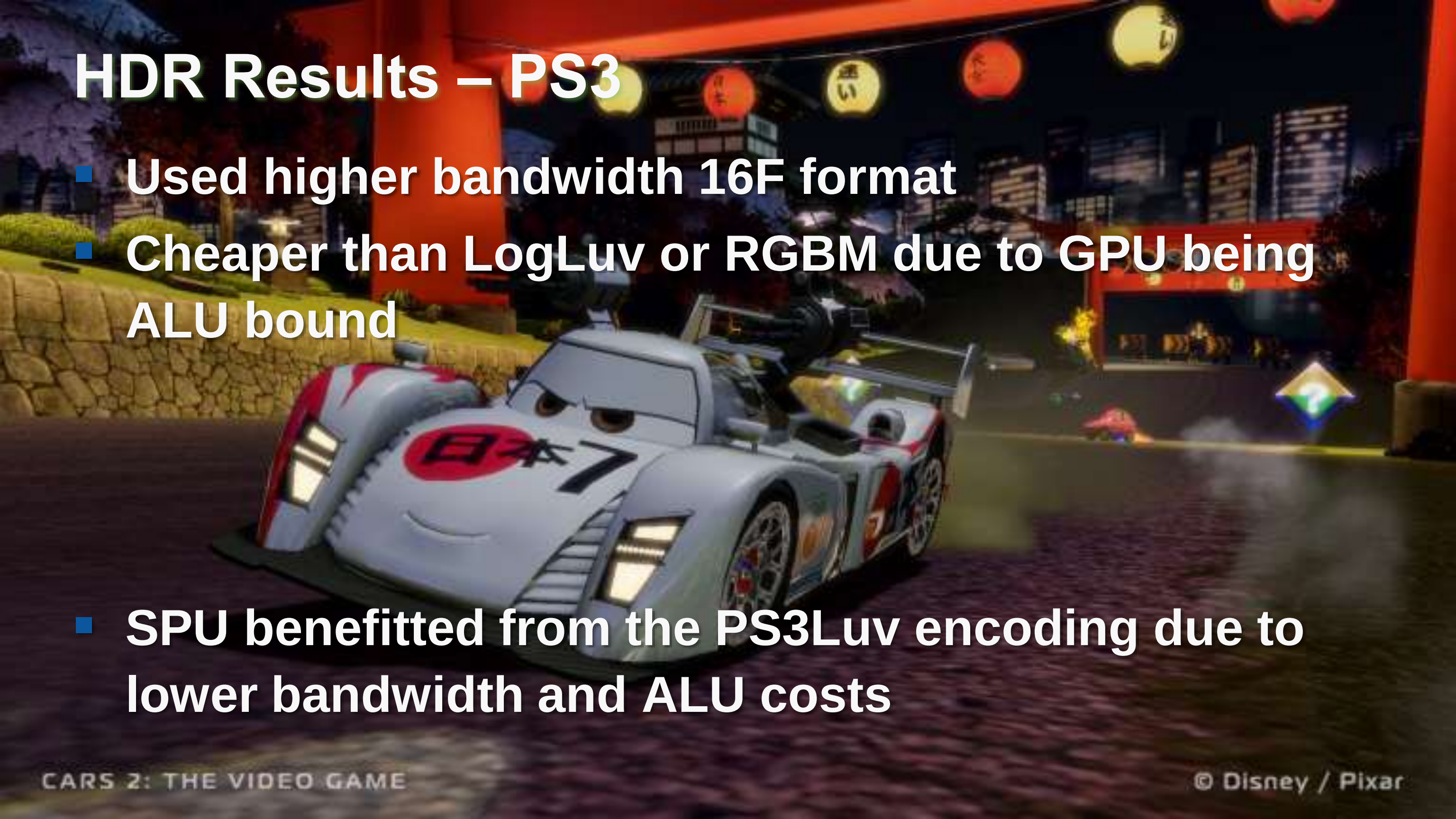
Red = Error in Clamped Image

HDR Results – Xbox 360

- Used 7e3 with pre-exposed color
- No tiling needed on a Non-MSAA 720p target

HDR Results – PS3

- Used higher bandwidth 16F format
- Cheaper than LogLuv or RGBM due to GPU being ALU bound
- SPU benefitted from the PS3Luv encoding due to lower bandwidth and ALU costs

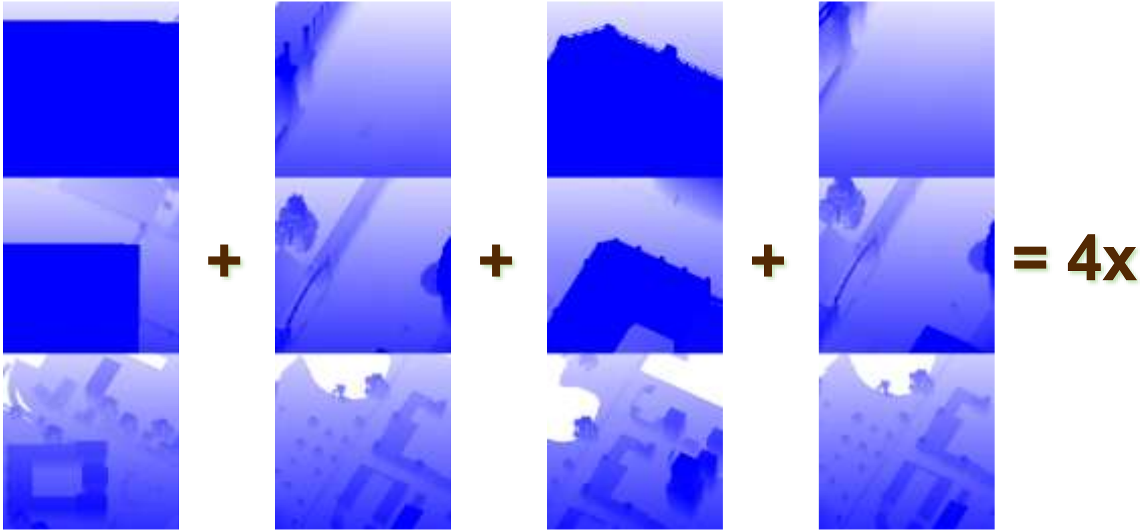


Shadows

Reduce shadow rendering time in half

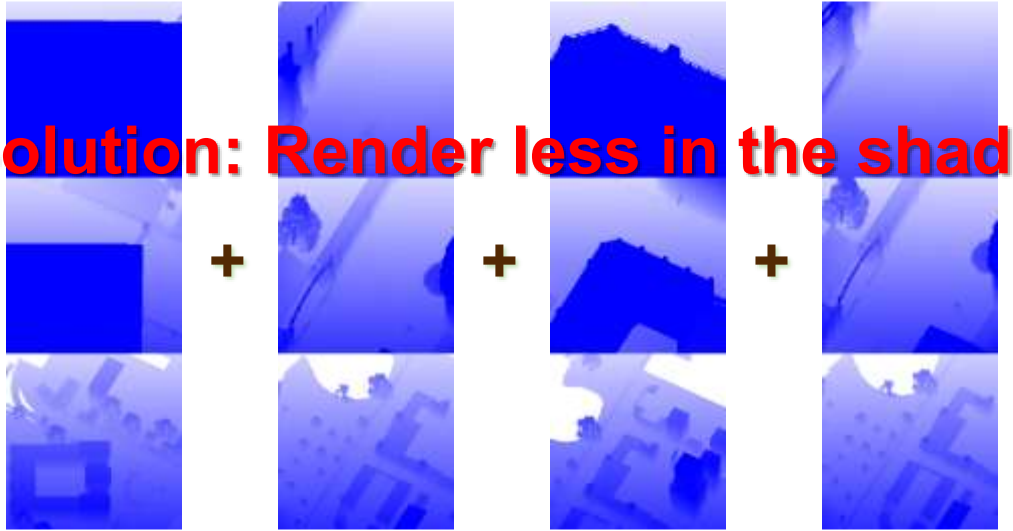
Scaling shadows for multiple viewports

- Four viewports = 4x number of draw calls and geometry
- Cutting resolution only reduces fill rate



Scaling shadows for multiple viewports

- Four viewports = 4x number of draw calls and geometry
- Cutting resolution only reduces fill rate

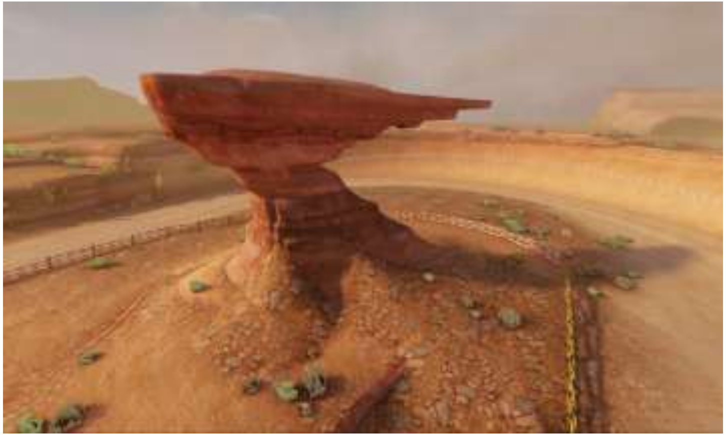


= 4x

Solution: Render less in the shadow maps

Two types of objects

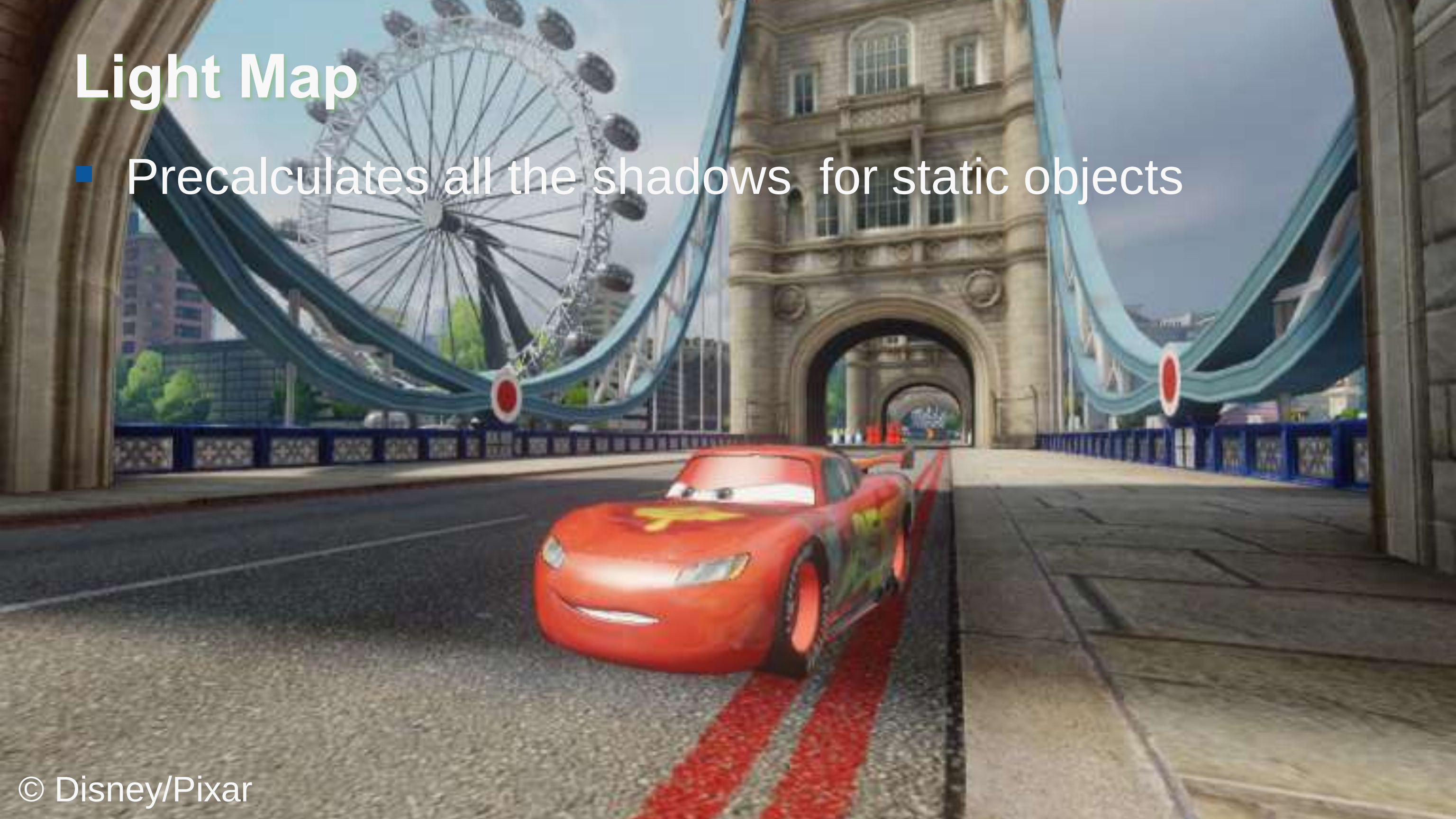
Static Objects – Things that never change position



Dynamic Objects – Those that can

Light Map

- Precalculates all the shadows for static objects

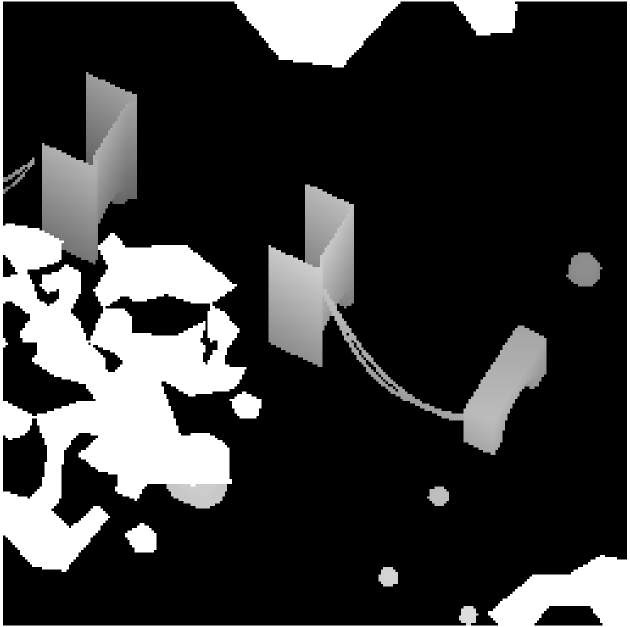


Dynamic objects

- Need to also receive shadows from light map
- Only need coarse transitions when going in and out of shadow

Low Resolution Shadow Map

- Use a 256x256 shadow map
- Super cheap ~0.1ms
- Use simple proxy geometry

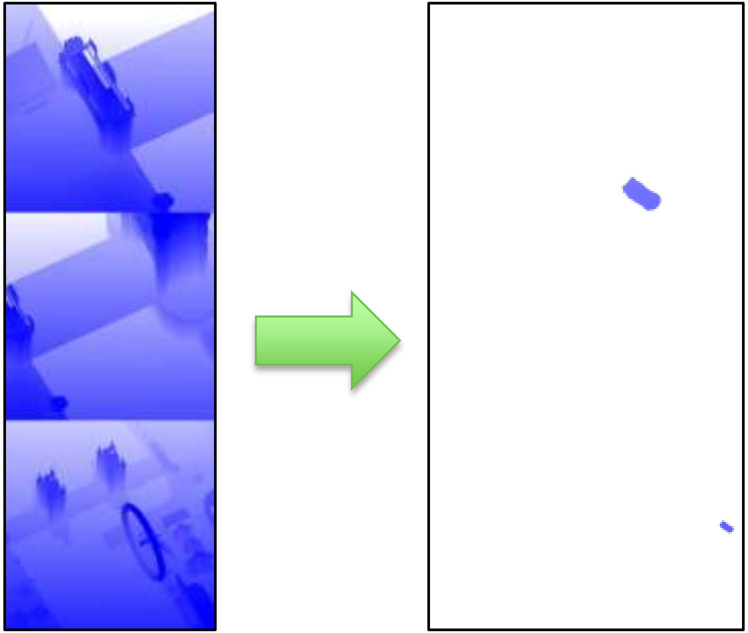


LightMap Only

With Low Resolution Shadow Map

Dynamic Shadows

- Only draw the dynamic objects in two cascades
- Reduced shadow distance
- Reprojection artifacts OK since tracks are on a 2d plane



Lightmap and Low Res Shadow Map

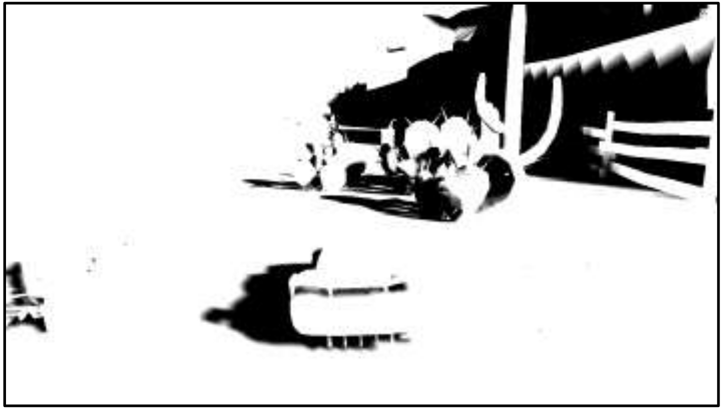
With Dynamic shadows

Reducing deferred shadow mask

- Reduce number of pixels processed

¼ Size Render Target

Bilateral Upsample



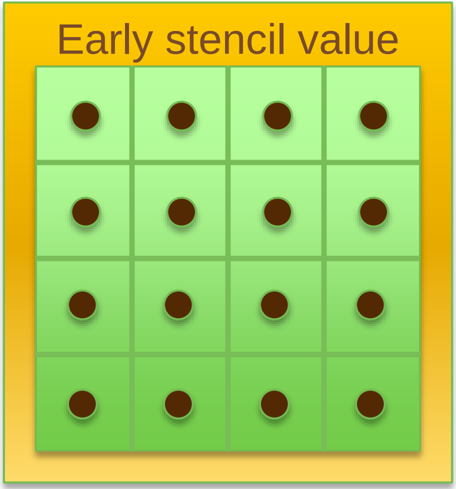
Unavoidable artifacts

- Edge Artifacts
- Lower resolution
- Too visible to ignore

MLAA and Early Stencil Culling [Jimenez]

Early Stencil Culling

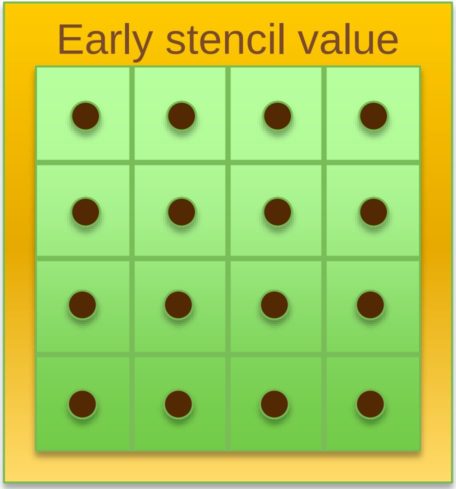
- Culls fragments before hitting the pixel shader
- Supported on PS3, 360, and modern PC graphics cards
- PC is automatic, PS3 and 360 manually controlled
- Latency between writing and testing



4x4 pixel block

Deferred shadows with early stencil

- Render shadows at 1/16 resolution [Hutchinson]
- Fill full resolution early stencil with 1/16 shadow mask
- Re-render shadow edges at full resolution using early stencil test



4x4 pixel block

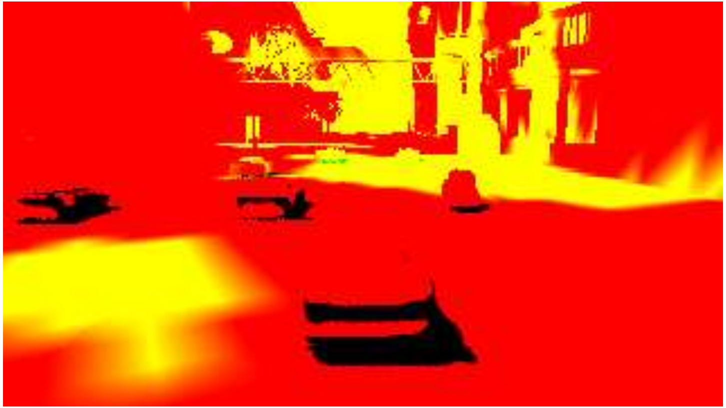
What's good enough at low resolution?

SIGGRAPH2011

- Shadow values that are 0 or 1
- Cascade selection
- Most pixels in cross bilateral filter

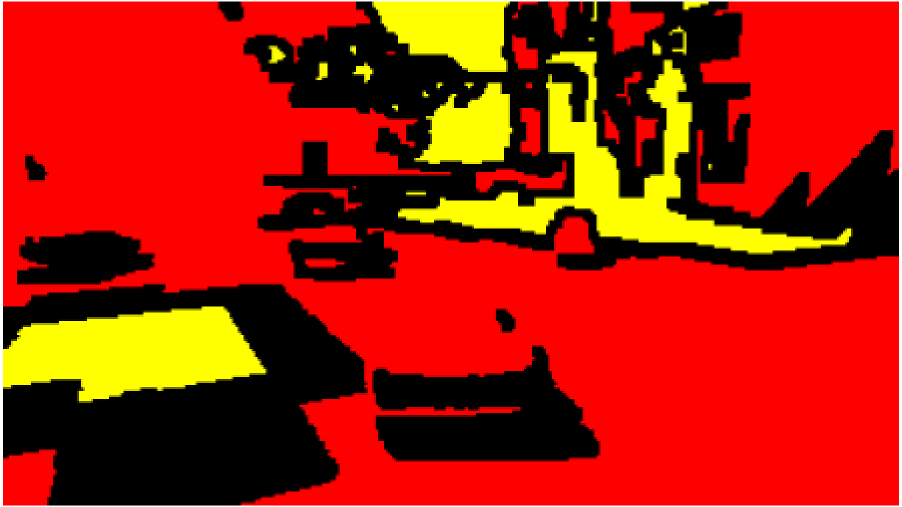
Render at 1/16 size

1/16 Deferred shadow mask → Dilate edges



Render at Full Resolution

- Point sample 1/16 target
- Turn on early stencil writes
- If it is inside the dilate region, then texkill



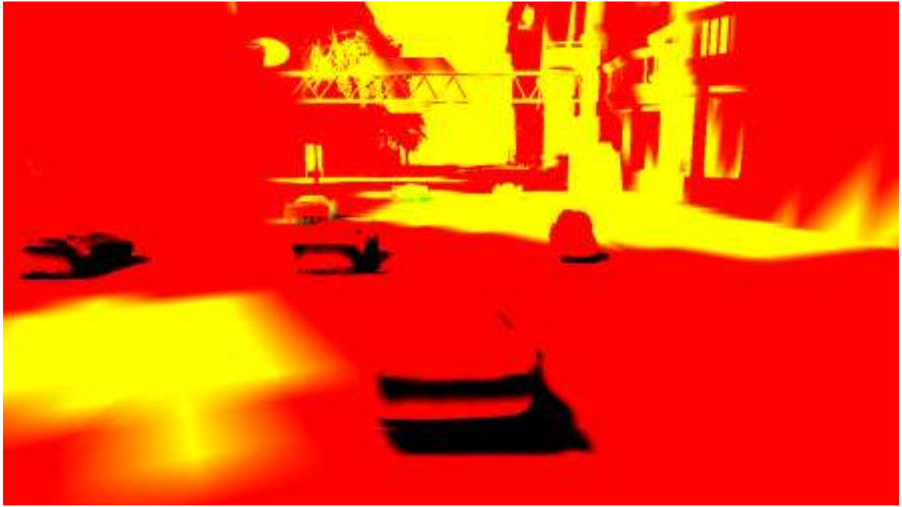
Pixels not yet filled

Out of shadow

In shadow

Render high res shadow mask [Ownby]

- Turn on early stencil test
- Early stencil culls pixels filled in previous pass
- Only renders ~30% of the pixels



Two pass bilateral blur

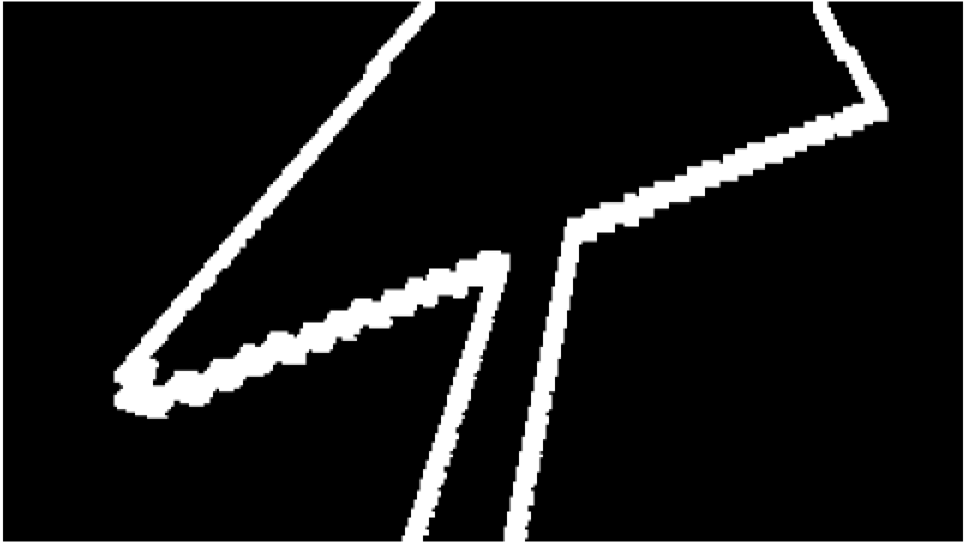
- Keep early stencil test on
- Only blurs ~30% of the pixels

Edge Artifacts

Early Stencil Culling **Without**

Edge Artifacts

- Early stencil is just a mask
- Dilate does not cover the blur regions
- Only happens at extreme closeups with a wide dilate



Conclusion

- Pre-exposed color is very effective when rendering to limited precision targets
- Low res shadow map for dynamic objects is cheap
- Deferred shadow mask rendering time effectively cut in half

SPU Post Processing

David Edwards

Advances in Real-Time Rendering in Games

SIGGRAPH2011

Motivation and Background

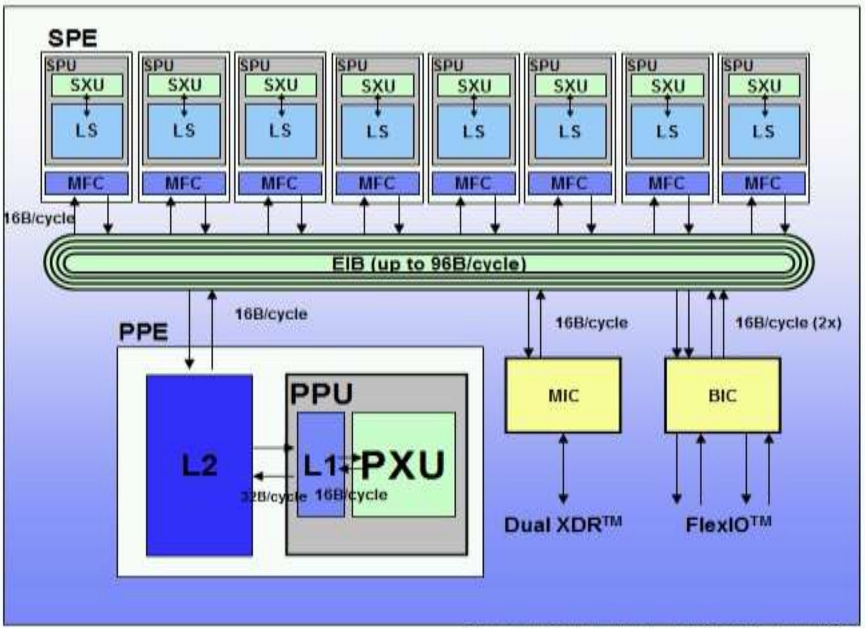
- In Toy Story 3 PS3 GPU performance typically lagged behind the Xbox 360, some effects had to be simplified or dropped on PS3 version
- Nearly half of GPU time in Toy Story 3 on PS3 was related to post processing
- PS3 has a lot of power that wasn't being fully utilized

Cars 2 SPU Post Process Pipeline

- SPU post process executes concurrently with the GPU
- GPU rendering reduced by ~10 ms

Cell Broadband Engine

- 6 SPUs available
- 3.2 GHz
- 128 vector registers
- 256KB Local Store



Source: M. Gschwind et al., Hot Chips-17, August 2005

SPU vs. PS3 GPU Pixel Shading

	SPU	GPU
Execution Speed	4 SPUs * 3.2 GHz = 12.8 GHz	24 pixels * 500 MHz = 12.0 GHz
Instructions	SIMD, General Purpose	SIMD, graphics centric (special math functions, texturing)
Data Model	Must prefetch into local store before use, no filtering	Texture cache, efficient threading model to hide latency, filtering

- For most effects SPU near same performance as GPU
- But runs concurrently, so, theoretically, we could nearly double performance of GPU

SPU vs. PS3 GPU Pixel Shading

GPU Code

```
FLOAT3 colorGradedColor = tex3D( samColorGradeLut, ( OUT.Color1.rgb * lutScale ) + lutOffset ).rgb;
```

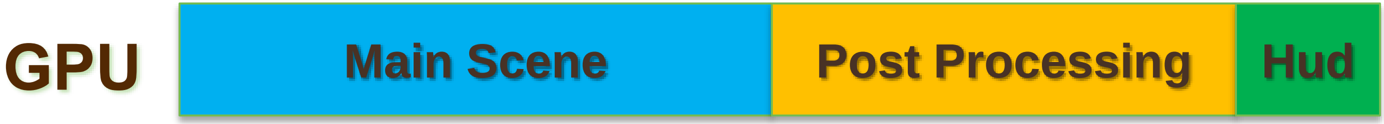
SPU vs. PS3 GPU Pixel Shading

SIGGRAPH2011

```
.macro LERP(result, a, b, t)

    .localreg   diff
    fs          diff, b, a
    fma         result, t, diff, a

.endmacro

.macro VOLUME_FETCH_INTERP(outColor, x0, x1, x2, x3, x4, x5, x6, x7, rInterp, gInterp, bInterp)

    .localreg   c0, c1, c2, c3, c4, c5, c6, c7
    lqx         c0, x0, colorGradeFp
    lqx         c1, x1, colorGradeFp
    lqx         c2, x2, colorGradeFp
    lqx         c3, x3, colorGradeFp
    lqx         c4, x4, colorGradeFp
    lqx         c5, x5, colorGradeFp
    lqx         c6, x6, colorGradeFp
    lqx         c7, x7, colorGradeFp

    .localreg   c01, c23, c45, c67, c0123, c4567

    LERP(c01, c0, c1, bInterp)
    LERP(c23, c2, c3, bInterp)
    LERP(c45, c4, c5, bInterp)
    LERP(c67, c6, c7, bInterp)

    LERP(c0123, c01, c23, gInterp)
    LERP(c4567, c45, c67, gInterp)

    LERP(outColor, c0123, c4567, rInterp)

.endmacro

.cfunc void r_spuPostColorGrade(uint8_t * color, vec_float4 * colorGradeFp, uint32_t width)

    .reg        kShufR, kShufG, kShufB
    il128       kShufR, "000B000F000J000N"
    il128       kShufG, "000C000G000K000O"
    il128       kShufB, "000D000H000L000P"

    .reg        kShufXXXX, kShufYYYY, kShufZZZZ, kShufWWWW
    il128       kShufXXXX, "AAAA"
    il128       kShufYYYY, "BBBB"
    il128       kShufZZZZ, "CCCC"
    il128       kShufWWWW, "DDDD"

    .reg        kHalf, kShufABab
    ilf32       kHalf, 0.5f
    il128       kShufABab, "ABab"

    .reg        kShufColorCombine
    il128       kShufColorCombine, "1HLP1hlp00000000"

    .reg        kLutScale
    ilf32       kLutScale, 15.0f/255.0f

loop:
    ai          width, width, -4
    ai          color, color, 0x10

    .reg        val
    lqd         val, -0x10(color)

    .reg        r, g, b
    shufb       r, val, val, kShufR
    shufb       g, val, val, kShufG
    shufb       b, val, val, kShufB
    cuflt       r, r, 0
    cuflt       g, g, 0
    cuflt       b, b, 0
    fm          r, r, kLutScale
    fm          g, g, kLutScale
    fm          b, b, kLutScale

    .reg        r0, g0, b0, r1, g1, b1
    cfltu       r0, r, 0
    cfltu       g0, g, 0
    cfltu       b0, b, 0
    ai          r1, r0, 1
    ai          g1, g0, 1
    ai          b1, b0, 1

    .reg        rInterp, gInterp, bInterp
    cuflt       rInterp, r0, 0
    cuflt       gInterp, g0, 0
    cuflt       bInterp, b0, 0
    fs          rInterp, r, rInterp
    fs          gInterp, g, gInterp
    fs          bInterp, b, bInterp

    .reg        rInterp0, gInterp0, bInterp0
    .reg        rInterp1, gInterp1, bInterp1
    .reg        rInterp2, gInterp2, bInterp2
    .reg        rInterp3, gInterp3, bInterp3

    shufb       rInterp0, rInterp, rInterp, kShufXXXX
    shufb       gInterp0, gInterp, gInterp, kShufXXXX
    shufb       bInterp0, bInterp, bInterp, kShufXXXX

    shufb       rInterp1, rInterp, rInterp, kShufYYYY
    shufb       gInterp1, gInterp, gInterp, kShufYYYY
    shufb       bInterp1, bInterp, bInterp, kShufYYYY

    shufb       rInterp2, rInterp, rInterp, kShufZZZZ
    shufb       gInterp2, gInterp, gInterp, kShufZZZZ
    shufb       bInterp2, bInterp, bInterp, kShufZZZZ

    shufb       rInterp3, rInterp, rInterp, kShufWWWW
    shufb       gInterp3, gInterp, gInterp, kShufWWWW
    shufb       bInterp3, bInterp, bInterp, kShufWWWW

    .reg        col0, col1, row0, row1, depth0, depth1
    shlqbii     col0, r0, 4
    shlqbii     col1, r1, 4
    shlqbyi     row0, g0, 1
    shlqbyi     row1, g1, 1
    shlqbii     depth0, b0, 4
    shlqbii     depth1, b1, 4
    shlqbyi     depth0, depth0, 1
    shlqbyi     depth1, depth1, 1

    .reg        rd00, rd01, rd10, rd11
    a           rd00, row0, depth0
    a           rd01, row0, depth1
    a           rd10, row1, depth0
    a           rd11, row1, depth1

    .reg        x00, x10, x20, x30, x40, x50, x60, x70
    .reg        x01, x11, x21, x31, x41, x51, x61, x71
    .reg        x02, x12, x22, x32, x42, x52, x62, x72
    .reg        x03, x13, x23, x33, x43, x53, x63, x73

    a           x00, col0, rd00
    a           x10, col0, rd01
    a           x20, col0, rd10
    a           x30, col0, rd11
    a           x40, col1, rd00
    a           x50, col1, rd01
    a           x60, col1, rd10
    a           x70, col1, rd11

    rotqbyi     x01, x00, 4
    rotqbyi     x11, x10, 4
    rotqbyi     x21, x20, 4
    rotqbyi     x31, x30, 4
    rotqbyi     x41, x40, 4
    rotqbyi     x51, x50, 4
    rotqbyi     x61, x60, 4
    rotqbyi     x71, x70, 4

    rotqbyi     x02, x00, 8
    rotqbyi     x12, x10, 8
    rotqbyi     x22, x20, 8
    rotqbyi     x32, x30, 8
    rotqbyi     x42, x40, 8
    rotqbyi     x52, x50, 8
    rotqbyi     x62, x60, 8
    rotqbyi     x72, x70, 8

    rotqbyi     x03, x00, 12
    rotqbyi     x13, x10, 12
    rotqbyi     x23, x20, 12
    rotqbyi     x33, x30, 12
    rotqbyi     x43, x40, 12
    rotqbyi     x53, x50, 12
    rotqbyi     x63, x60, 12
    rotqbyi     x73, x70, 12

    .reg        color0, color1, color2, color3
    VOLUME_FETCH_INTERP(color0, x00, x10, x20, x30, x40, x50, x60, x70, rInterp0
    VOLUME_FETCH_INTERP(color1, x01, x11, x21, x31, x41, x51, x61, x71, rInterp1
    VOLUME_FETCH_INTERP(color2, x02, x12, x22, x32, x42, x52, x62, x72, rInterp2
    VOLUME_FETCH_INTERP(color3, x03, x13, x23, x33, x43, x53, x63, x73, rInterp3

    fa          color0, color0, kHalf
    fa          color1, color1, kHalf
    fa          color2, color2, kHalf
    fa          color3, color3, kHalf

    cfltu       color0, color0, 0
    cfltu       color1, color1, 0
    cfltu       color2, color2, 0
    cfltu       color3, color3, 0

    .reg        c01, c23, c0123
    shufb       c01, color0, color1, kShufColorCombine
    shufb       c23, color2, color3, kShufColorCombine
    shufb       c0123, c01, c23, kShufABab

    stqd        c0123, -0x10(color)

    brnz        width, loop : $mintrip<4>

.endfunc
```

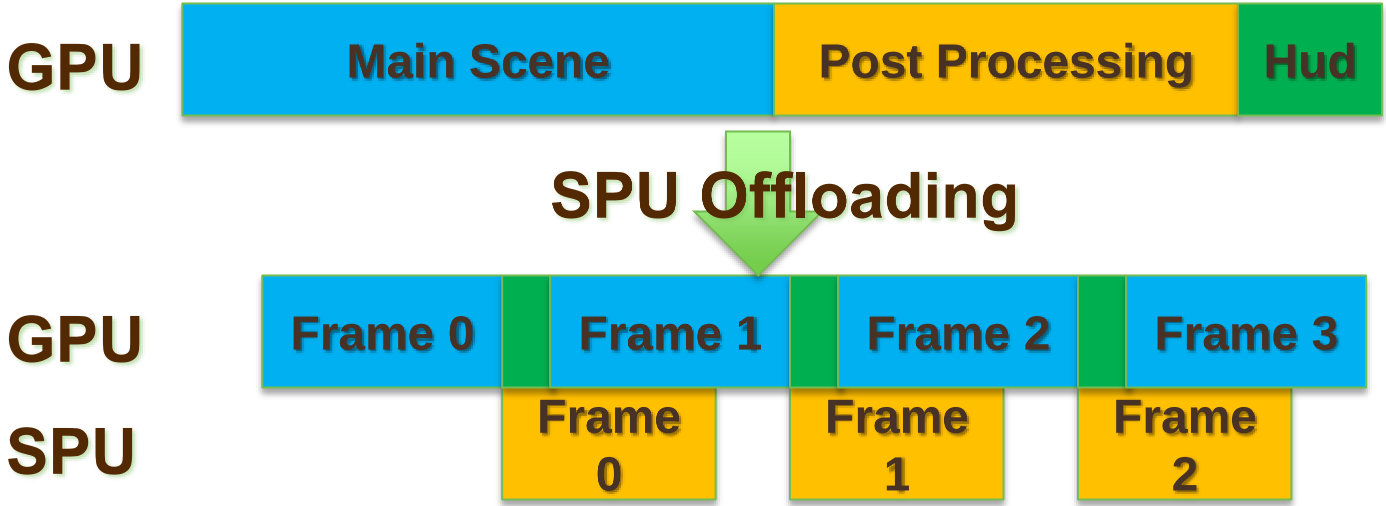
SPU Code

SPU Post Processing Implementation



SPU Post Processing Implementation

SIGGRAPH2011



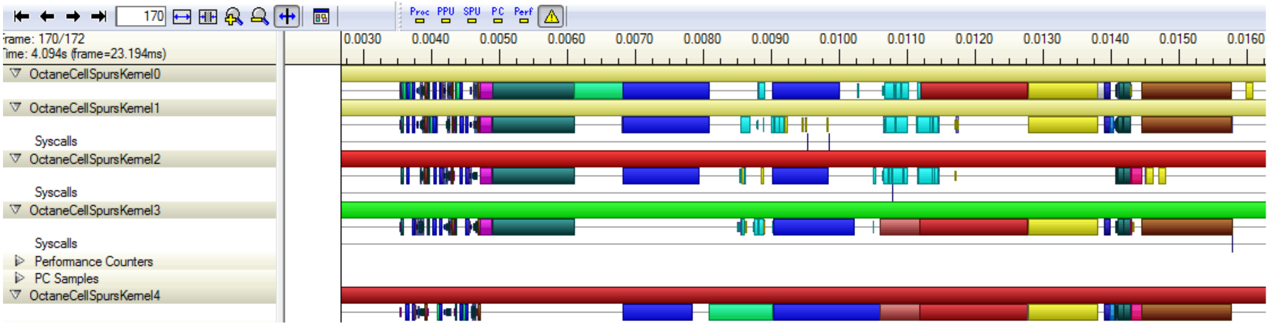
SPU Post Processing Implementation

- SPU post processing adds some overhead
  - Source textures must reside in main memory
    - SPU can't (realistically) read from VRAM
    - Adds nearly 10 MB to main memory
    - Adds about 1.5 ms of GPU overhead
- Either SPU or GPU can copy back to VRAM so use whatever is not bottlenecked

SPU Post Processing Effects

- Scene average log luminance
- Tonemapping
- Morphological antialiasing (This saves main scene GPU time too!)
- Motion blur
- Downsamples / Upsamples
- Highpass
- Gaussian Blurs
- Color correction
- Stereo 3D
- Screen space ambient occlusion (not used in Cars 2)

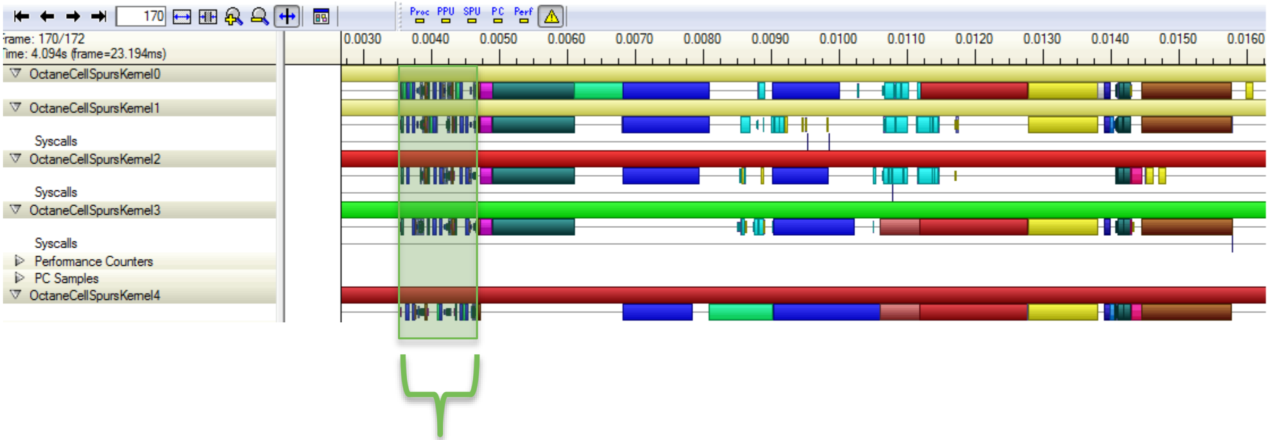
SPU Post Process Performance



$\underbrace{\hspace{10em}}$

12.25 ms Total

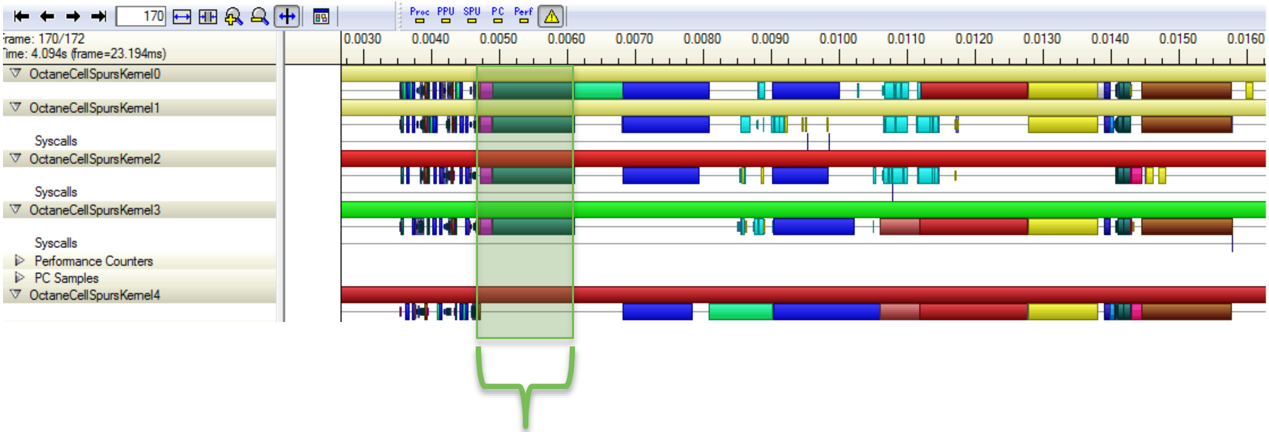
SPU Post Process Performance



1.15 ms – three picture in pictures

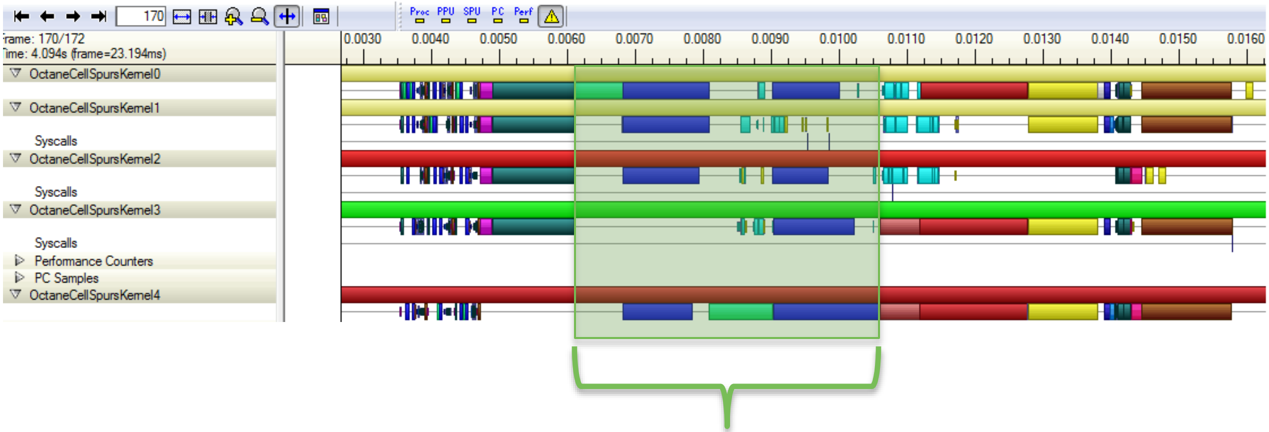
SPU Post Process Performance

SIGGRAPH2011



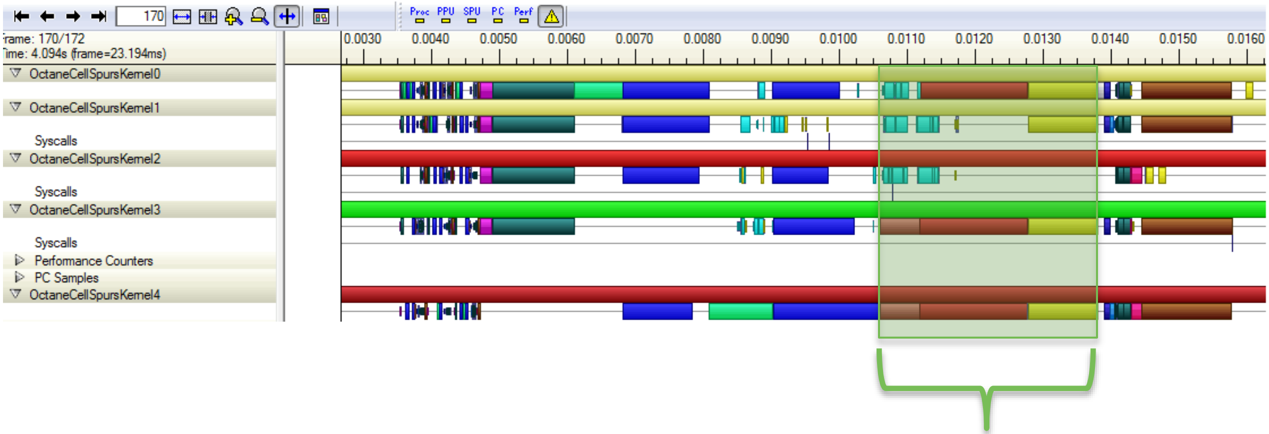
1.4 ms – Average Log Luminance/Tonemap

SPU Post Process Performance



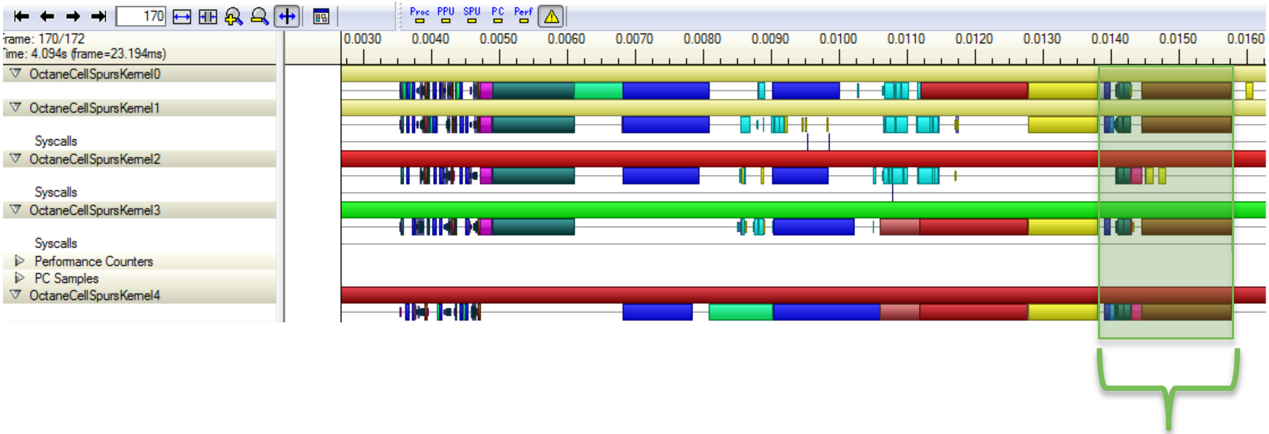
4.5 ms – Morphological Anti-aliasing

SPU Post Process Performance



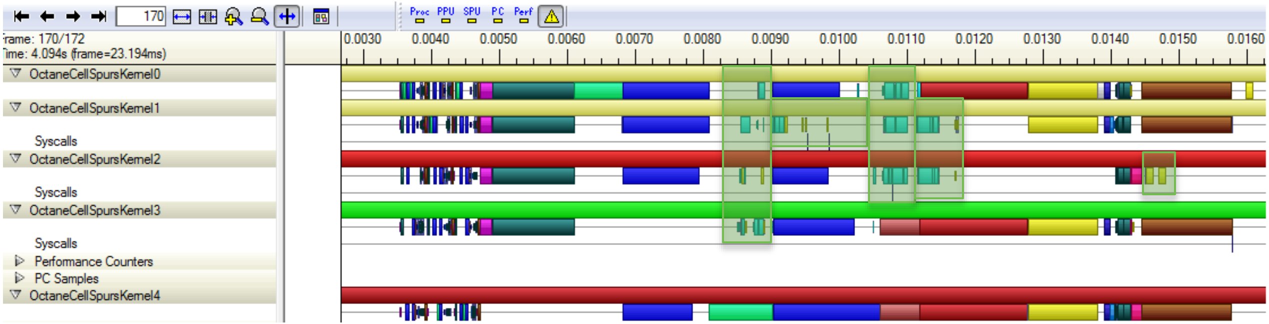
3.2 ms – Downsample/Motion Blur/Composite

SPU Post Process Performance



2.0 ms – HDR Bloom/Composite

SPU Post Process Performance



Other SPU Jobs mixed in with Post Processing

Stereo 3D

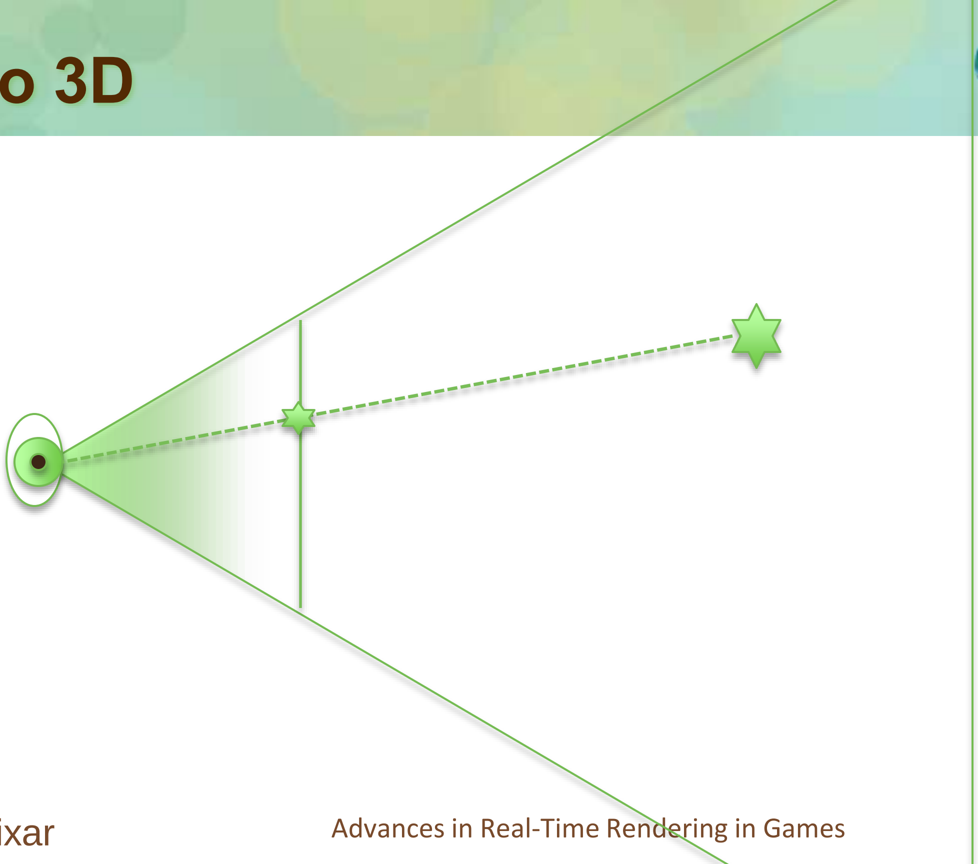
Advances in Real-Time Rendering in Games

SIGGRAPH2011

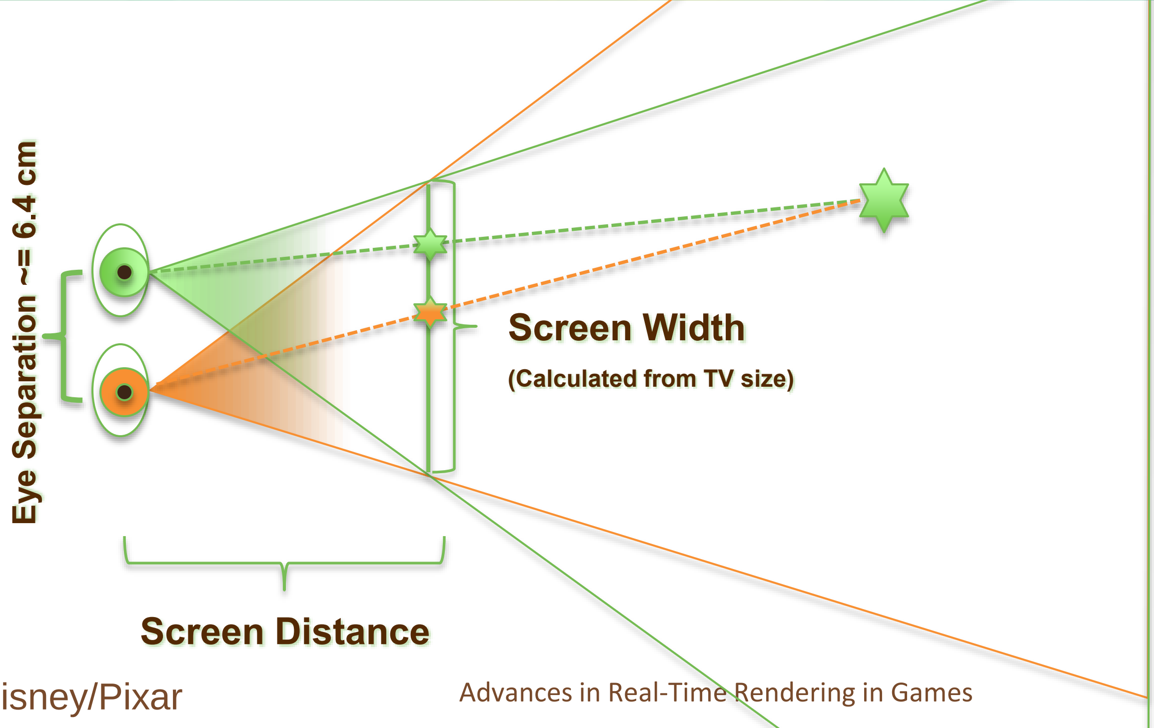
Cars 2 Stereo 3D

- Implemented as part of SPU post processing pipeline
- And hence was free!!
- No reduction in performance, graphics content, effects, and/or resolution

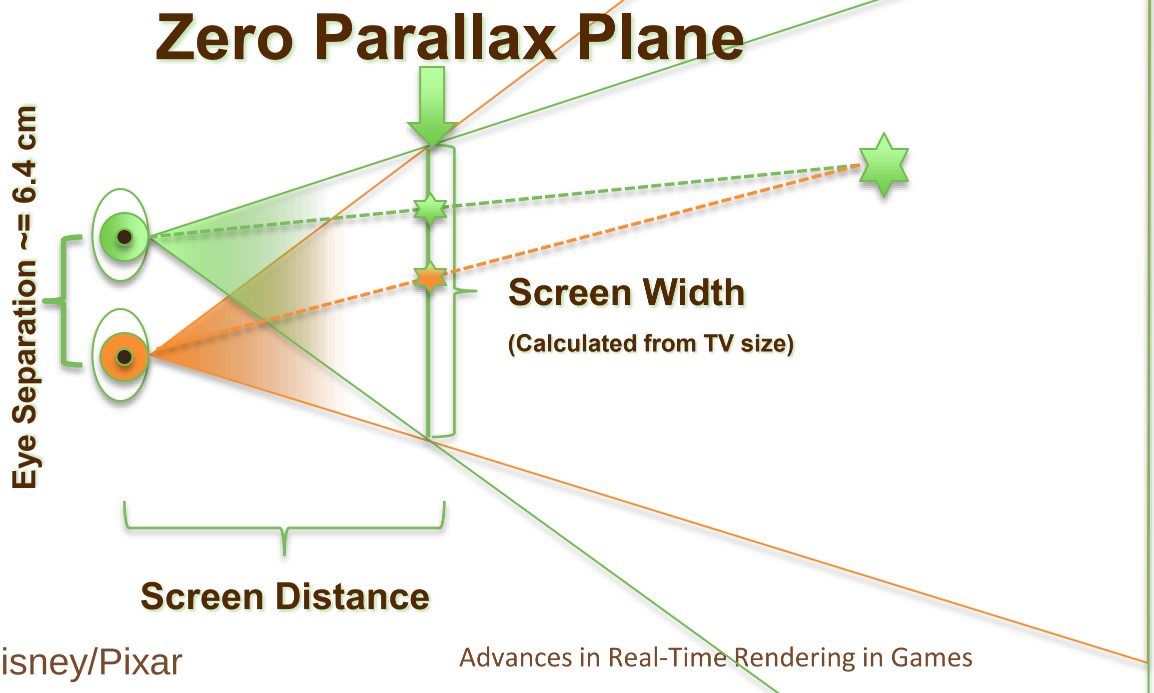
Mono 3D



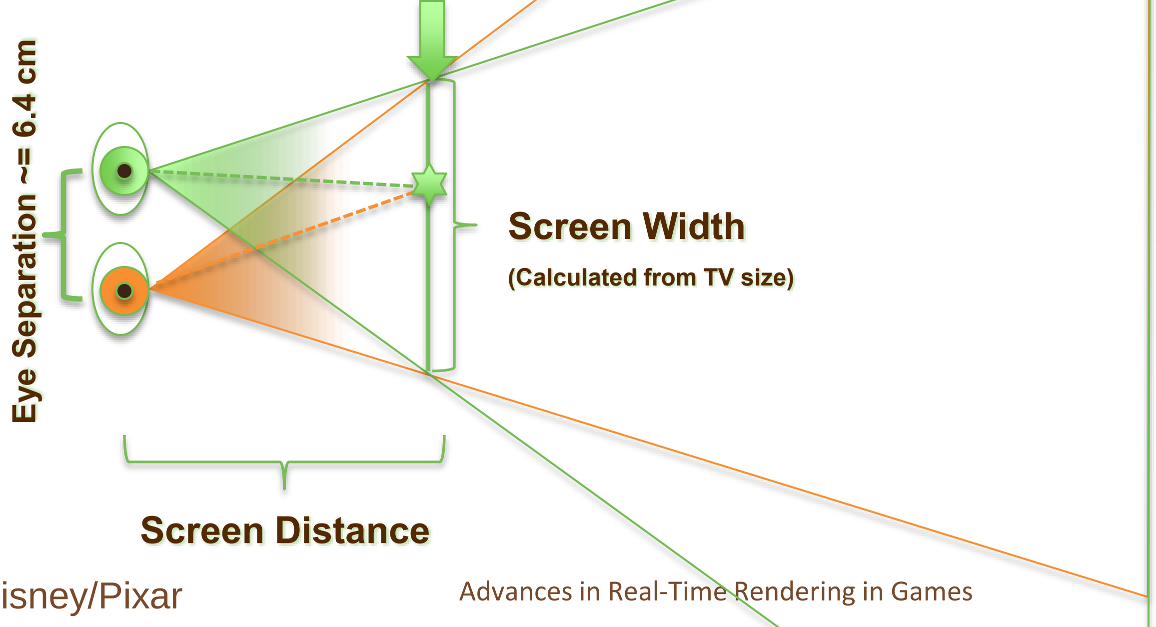
Stereo 3D


Eye Separation ~= 6.4 cm
Screen Width
(Calculated from TV size)
Screen Distance

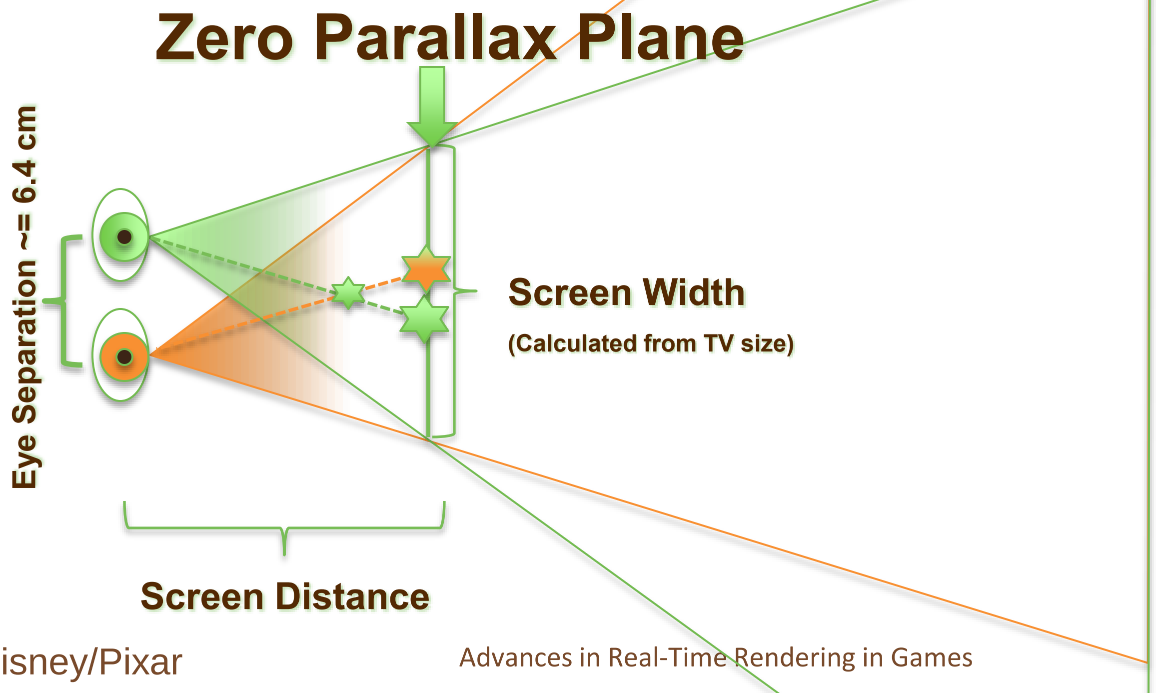
Stereo 3D


Zero Parallax Plane
Eye Separation ~= 6.4 cm
Screen Width
(Calculated from TV size)
Screen Distance

Zero Parallax Plane



Stereo 3D



Zero Parallax Plane
Eye Separation ~= 6.4 cm
Screen Width
(Calculated from TV size)
Screen Distance

Traditional Stereo 3D

SIGGRAPH2011

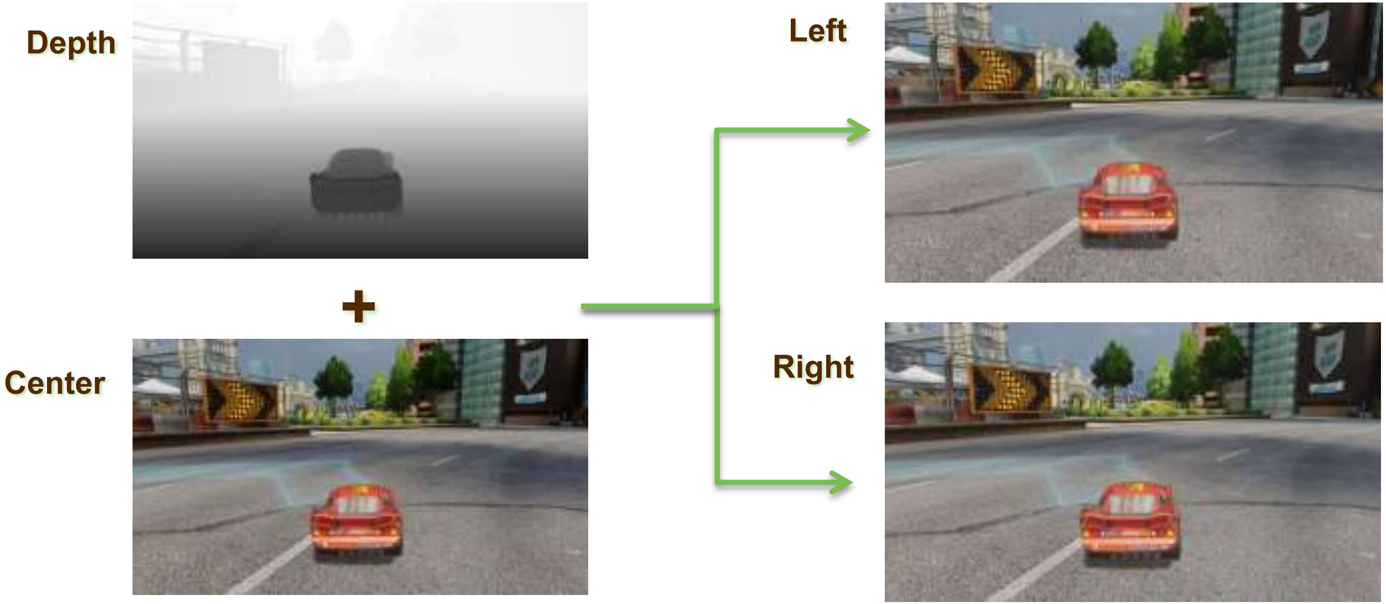
- Render both stereo pairs fully including post processing effect.
- Performance Cost is 2x
- Can be optimized
  - Scene cull once, share results with both eyes
  - Reuse shadow map
  - Use lower resolution target, PS3 has hardware upscaling modes for 3D
  - Reduce content or effects

Traditional Stereo 3D

SIGGRAPH2011

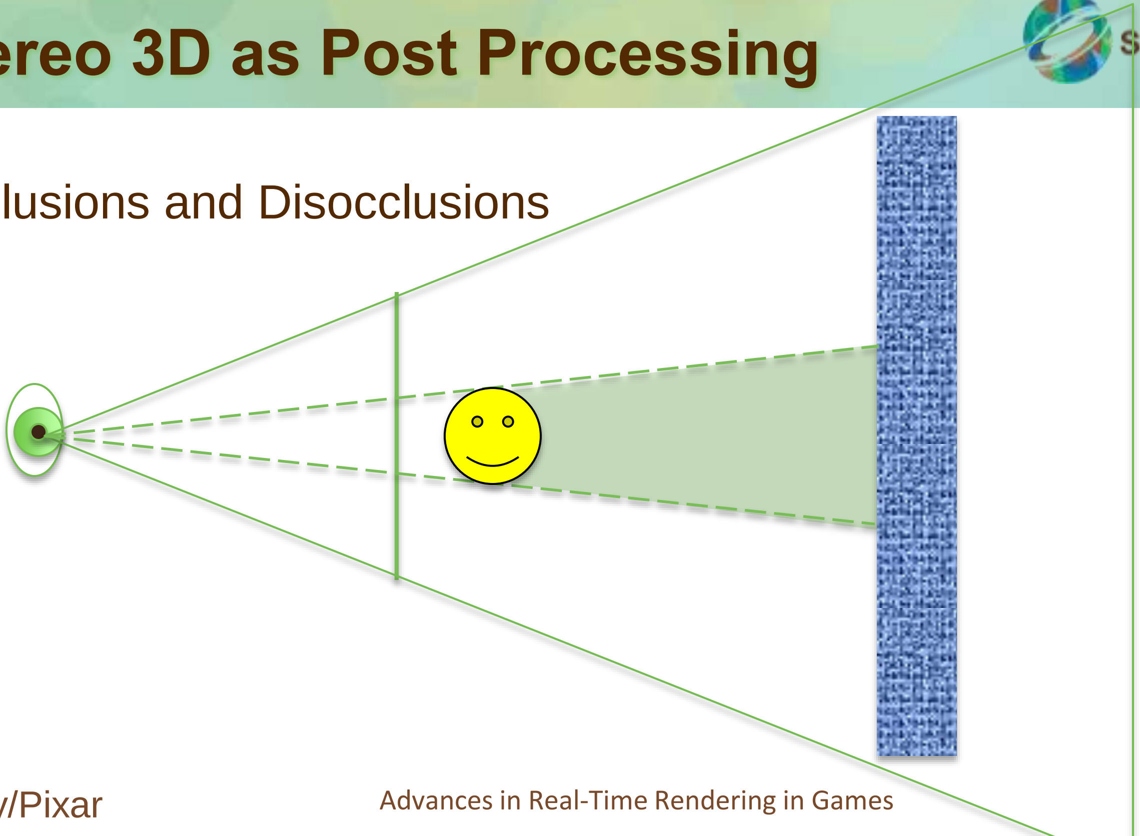
- Traditional Stereo 3D in 4 player split screen game
  - Very high geometry cost (8x)
  - Reduced resolution in 4 player split screen would not be acceptable

Stereo 3D as Post Processing



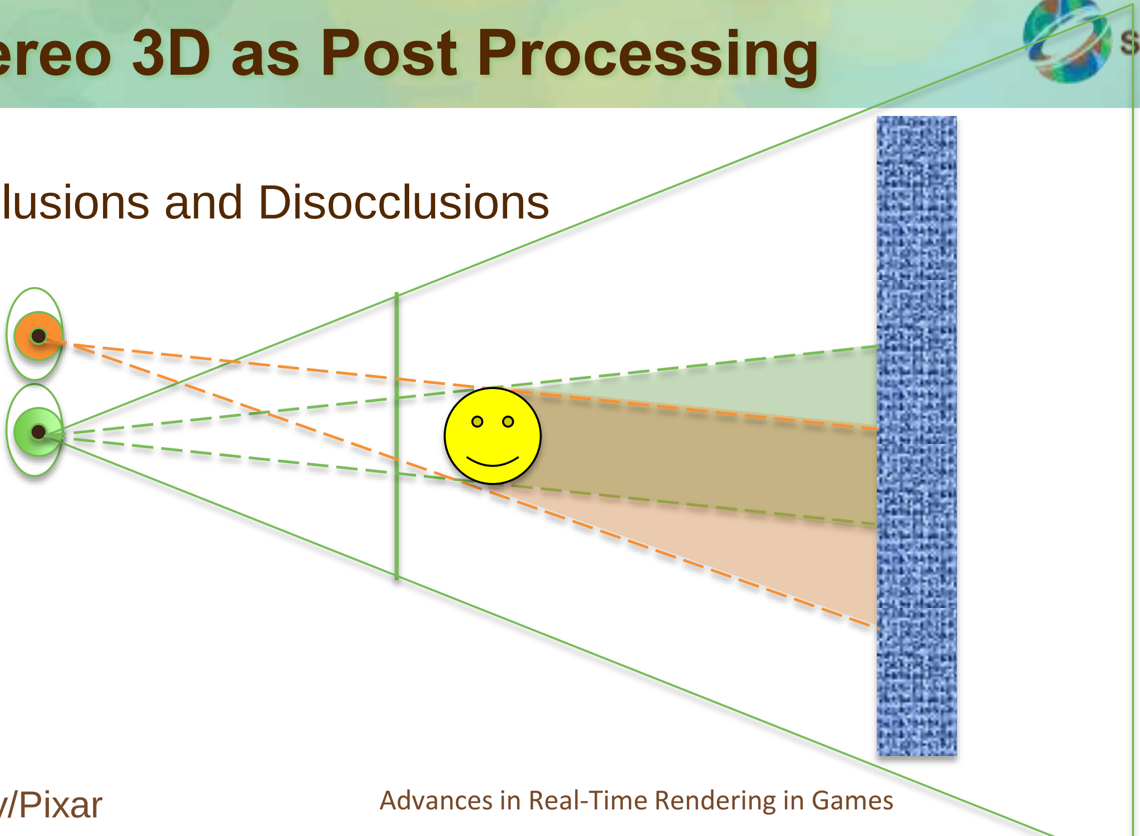
Stereo 3D as Post Processing

Occlusions and Disocclusions



Stereo 3D as Post Processing

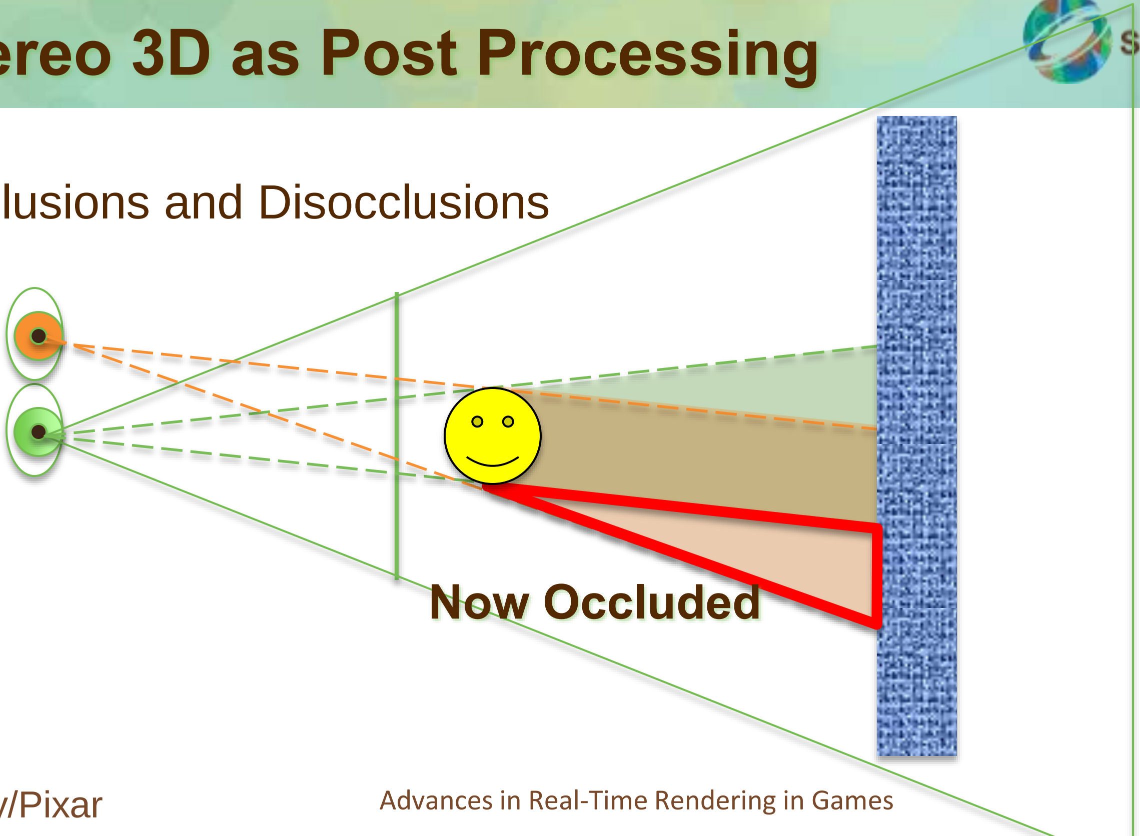
Occlusions and Disocclusions



Stereo 3D as Post Processing

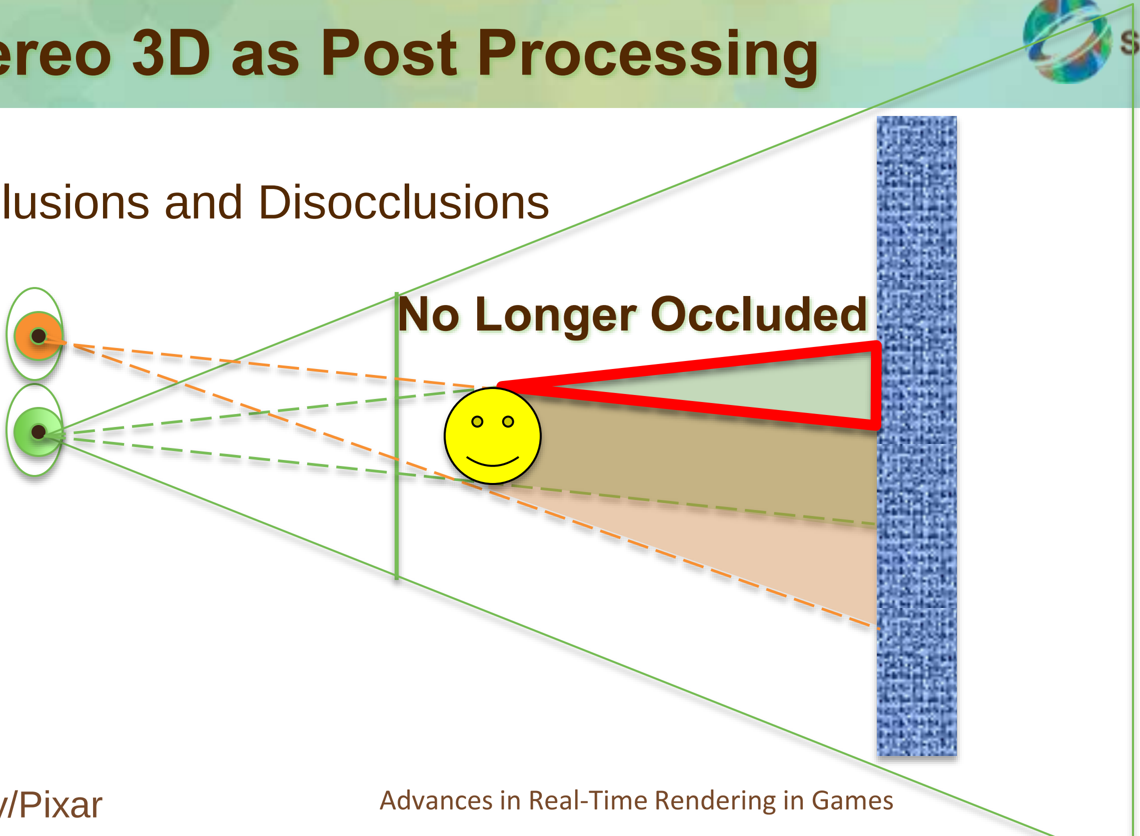
SIGGRAPH2011

Occlusions and Disocclusions



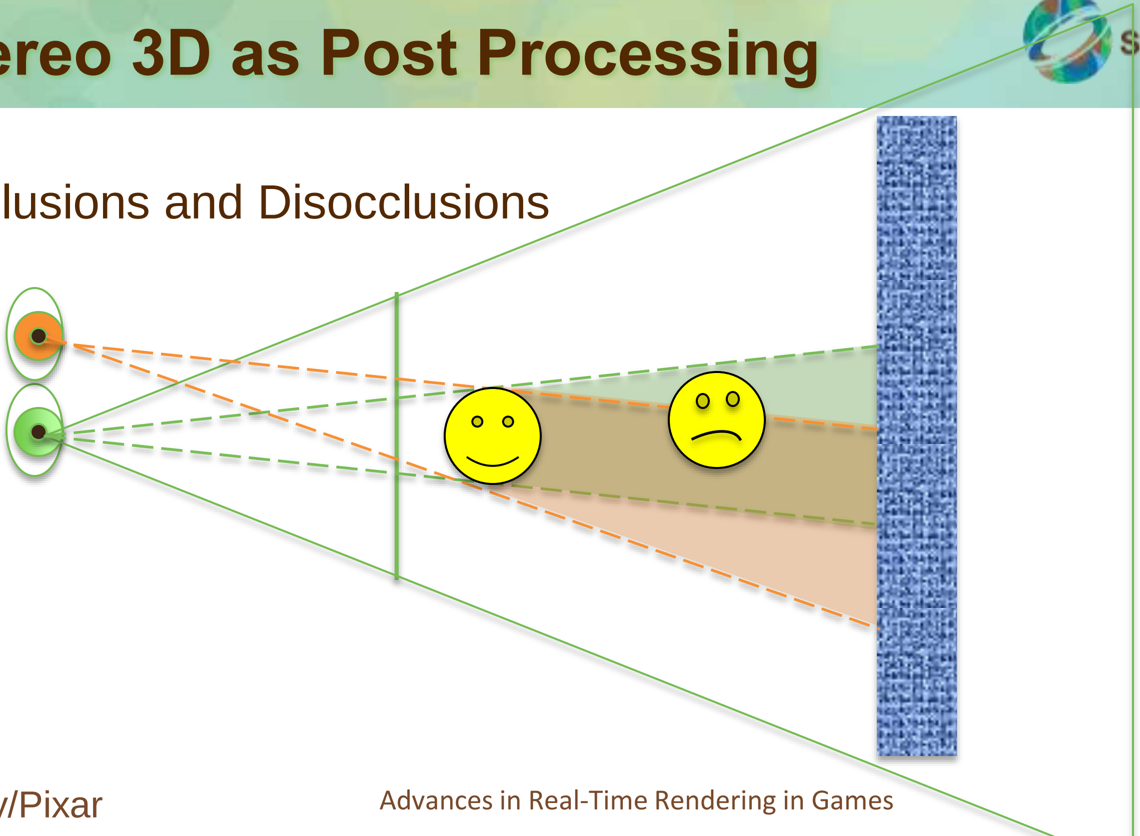
Stereo 3D as Post Processing

Occlusions and Disocclusions



Stereo 3D as Post Processing

Occlusions and Disocclusions



Stereo 3D as Post Processing

- Other Issues
  - View dependent lighting, and reflections
  - Translucent objects (no depth value written)

Stereo 3D as SPU Post Effect

SIGGRAPH2011

- SPU has some advantages over GPU for stereo 3D

	GPU	SPU
Gathered Reads	Really Good	Good
Scattered Writes	Complex/Inefficient	Good

- Scattered Writes is inverse of Gathered Reads
  - Gathered Reads: What texel should I sample for the current pixel?
  - Scattered Writes: What pixel should I write to for the current texel?

Stereo 3D as SPU Post Effect

- Given a depth value, we can reproject the location for each left and right eye, allowing us to write the color value to the new location
- This is Scattered Writes, which can be done efficiently on the SPU
- However, the GPU cannot do this efficiently
  - A Gathered Reads approach ends up using approximated reprojections

Source

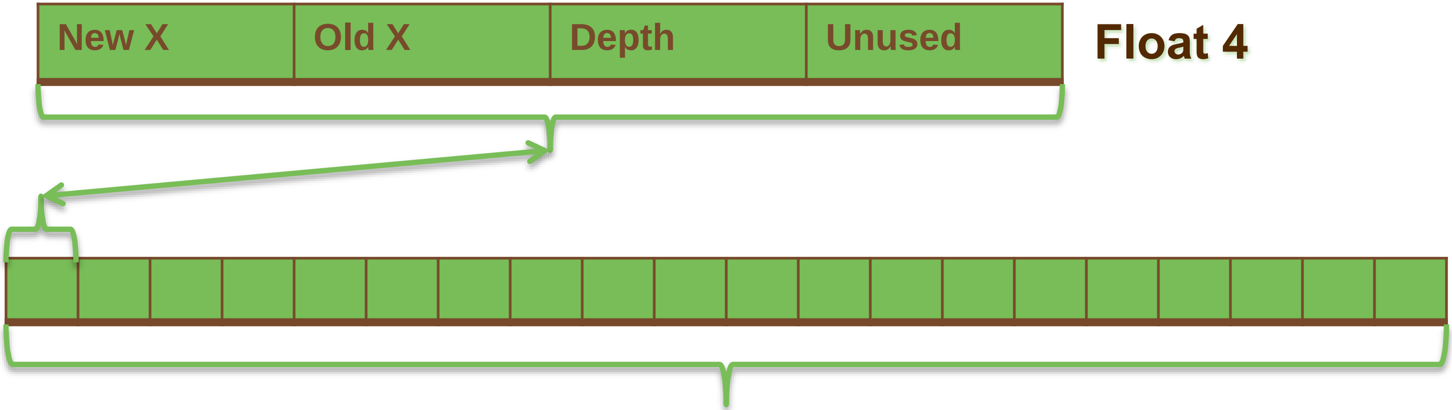

Left
© Disney/Pixar


allinol
Right

Stereo 3D Implementation

- Human eyes are horizontal
- Stereo reprojection only shifts to the left or right
- We can process each scan line independently

Item Buffer



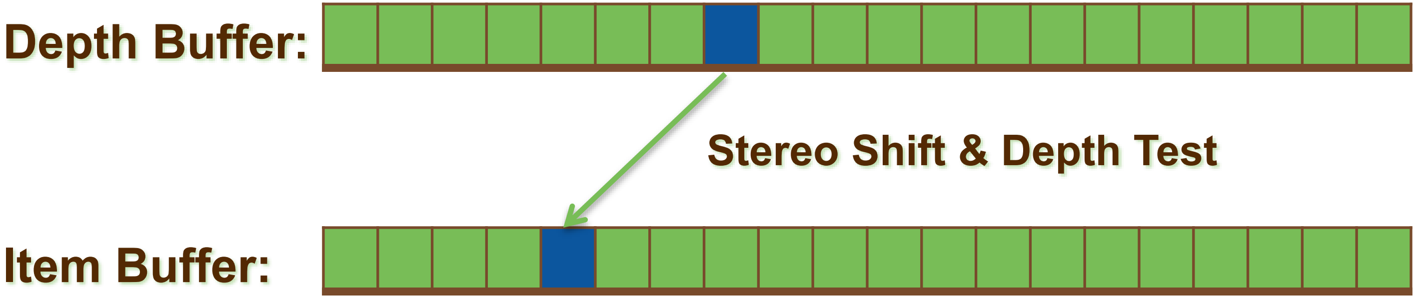
N-Items where N = width

Stereo 3D Implementation

SIGGRAPH2011

Step 1: Clear depth of each item in item buffer

Step 2: Iterate over depth buffer and fill item buffer



Stereo 3D Implementation

Step 3: Hole Filling

Item Buffer:

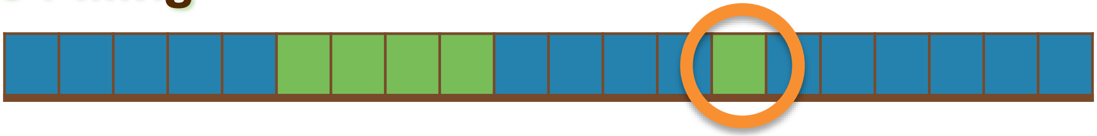
Two kinds of holes

Stereo 3D Implementation

SIGGRAPH2011

Step 3: Hole Filling

Item Buffer:



Rounding Holes

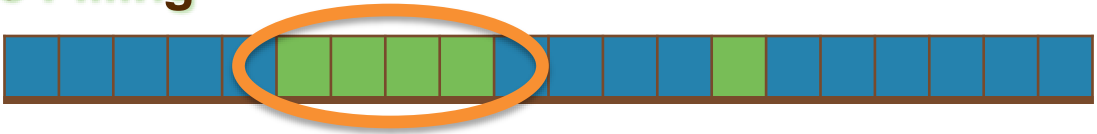
Two kinds of holes

Stereo 3D Implementation

SIGGRAPH2011

Step 3: Hole Filling

Item Buffer:



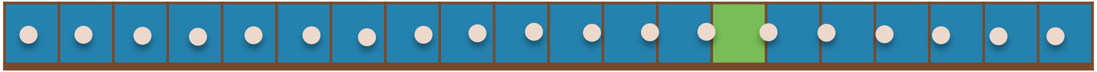
Disocclusion Holes

Two kinds of holes

Stereo 3D Implementation

Rounding Holes

Item Buffer:

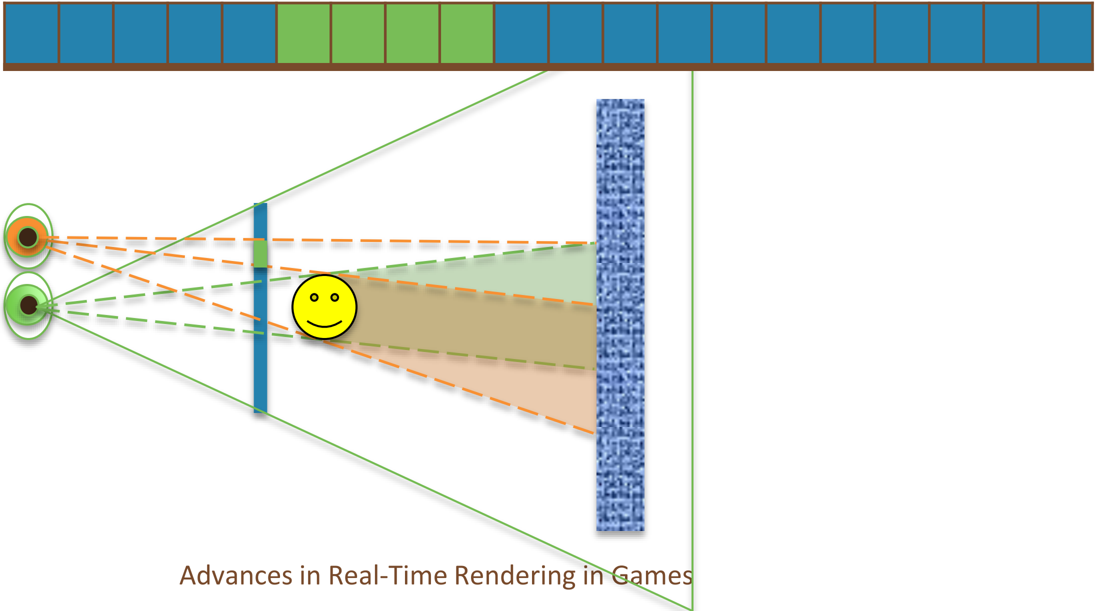


- These are not real holes in the scene
- Just fill in by interpolating the before and after items

Stereo 3D Implementation

Disocclusion Holes

Item Buffer:



Stereo 3D Implementation

SIGGRAPH2011

Disocclusion Holes

Item Buffer:

	R	O	A	D	H	O	L	E	L	I	G	H	T	N	I	N	G		

Item Buffer:

	R	O	A	D	R	O	A	D	L	I	G	H	T	N	I	N	G		


Source
© Disney/Pixar

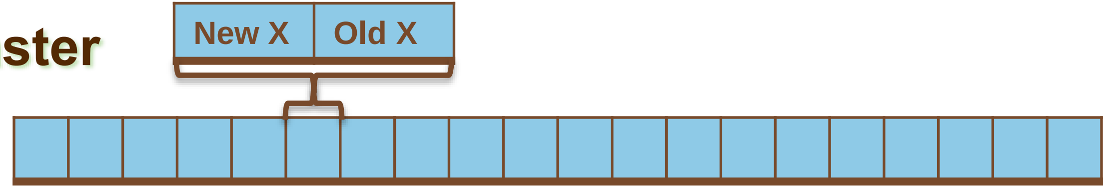

Left Eye Showing Disocclusions

Left Eye With Disocclusions Filled

Stereo 3D Implementation

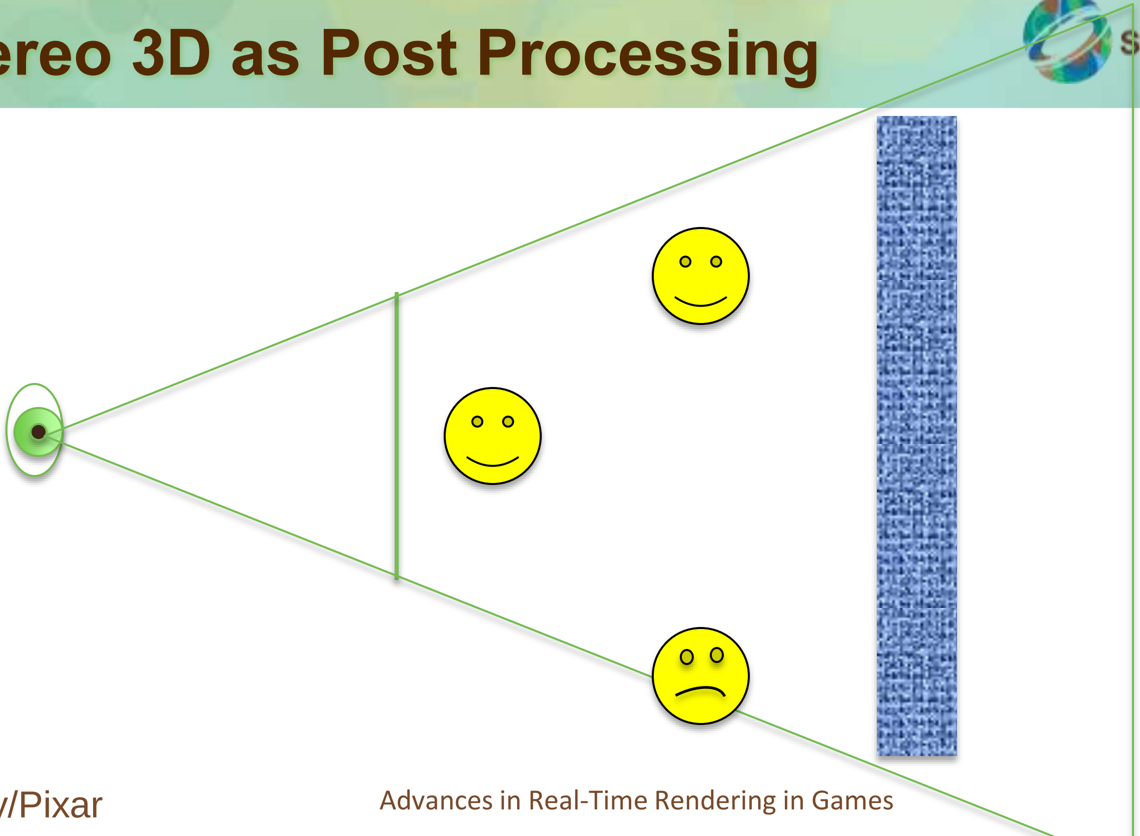
Step 4: Raster

Item Buffer:



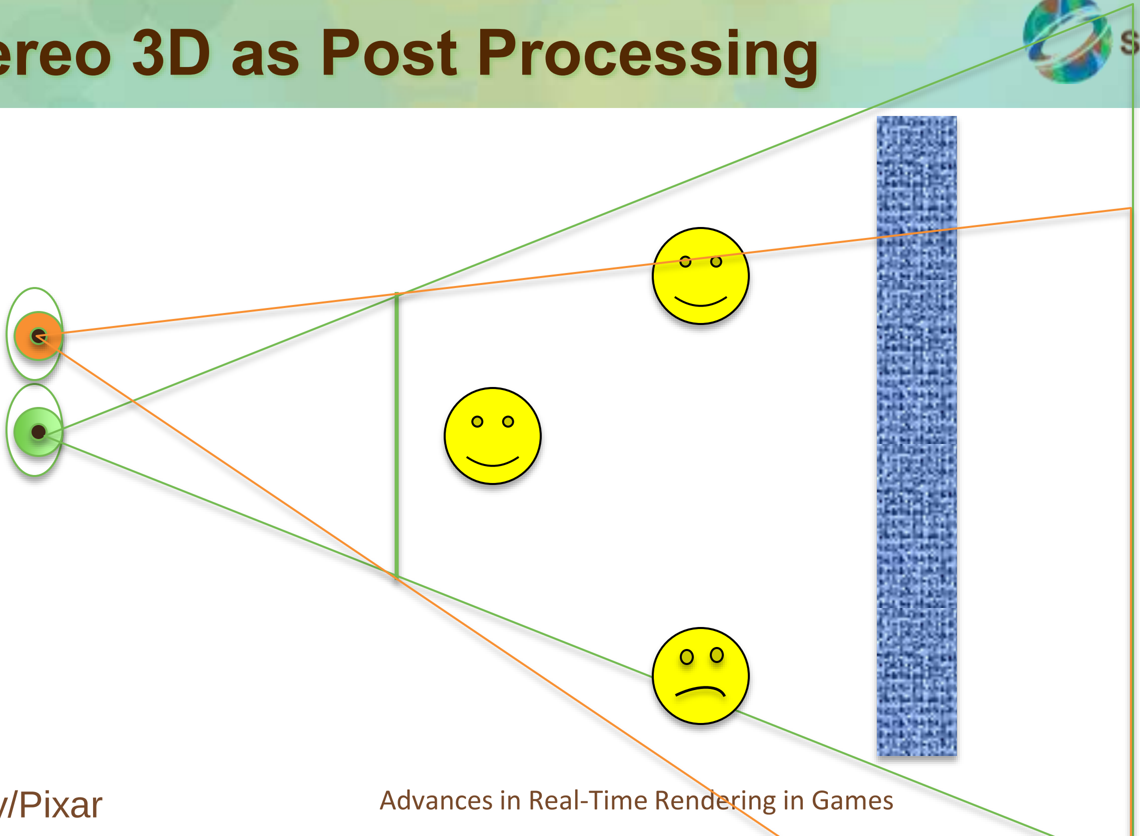
- For each output pixel
  - Interpolate item at each pixel center using the New X value
  - This gives us an interpolated Old X value
  - Use the interpolated Old X value to linear filter into the color buffer and output

Stereo 3D as Post Processing



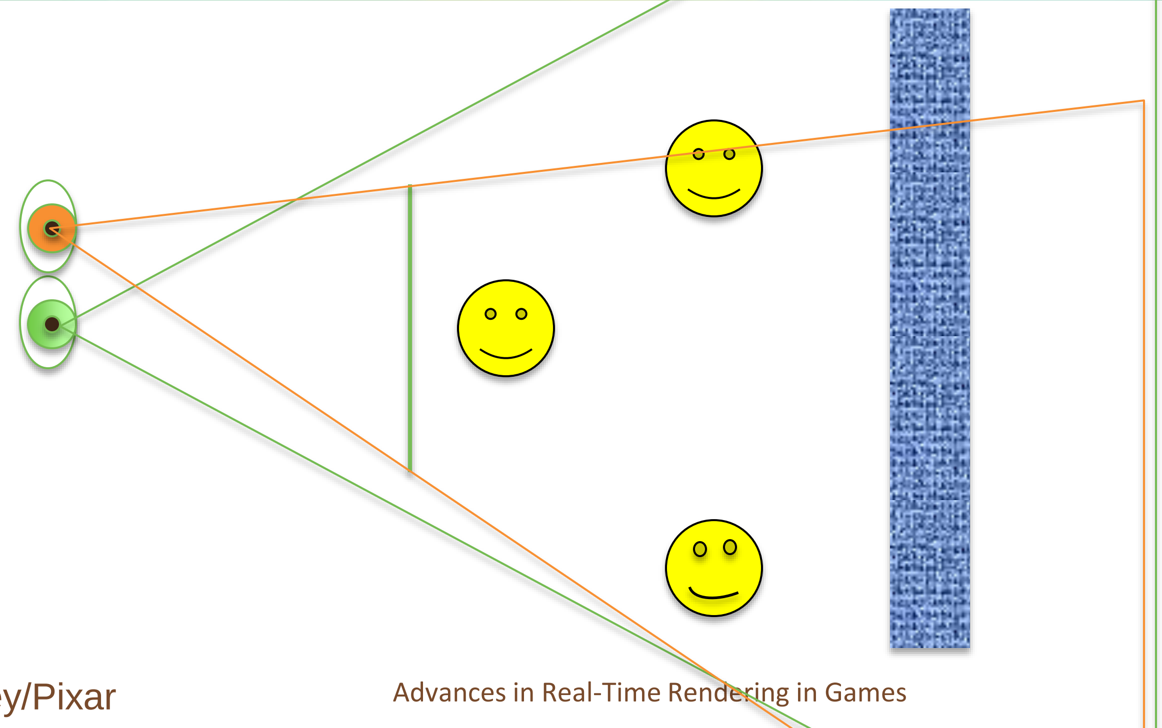
Stereo 3D as Post Processing

SIGGRAPH2011



Stereo 3D as Post Processing

SIGGRAPH2011



Source


Left Eye With Screen Disocclusion


Source

Source with tweaked aspect ratio to widen view


Left Eye Final
© Disney/Pixar

Conclusion

- Moving post processing onto SPU was a big win
- It also gave us Stereo 3D for free
- SPU is a good fit for Stereo 3D post processing

Questions?

References

- 3D Stereo Design Guide , PlayStation®3 Library Documentation
- 3D Stereo Programming Guide, PlayStation®3 Library Documentation
- Castaño, I. 2011. Hemicube Rendering and Integration. Retrieved from http://the-witness.net/news/2010/09/hemicube-rendering-and-integration/
- Greger, G., Shirley, P., Hubbard, P., Greenberg, D. 1998. The Irradiance Volume.
- Hable, J. 2010. Uncharted 2: HDR Lighting, In GDC 2010.
- Hutchinson, N., Knight, B., Ritchie, M. Parrish, G., and Moore, J. Screen space classification for efficient deferred shading. In SIGGRAPH 2010
- Jimenez, J., Masia, B., Echevarria, J., Navarro, F., and Gutierrez, D. 2011. GPU Pro 2, ch. Practical Morphological Anti-Aliasing.
- Karis, B. 2009. RGBM color encoding. Retrieved from http://graphicrants.blogspot.com/2009/04/rgbm-color-encoding.html
- King, G. 2005. *GPU Gems 2*, ch. Real-Time Computation of Dynamic Irradiance Environment Maps.
- Kontkanen, J. and Laine, S. 2006. Sampling Precomputed Volumetric Lighting. “Journal of Graphics, GPU, and Game Tools”. Vol. 11, #3 pp 1-16.

References

- Larson, G., Encoding for full-gamut, high-dynamic range images. “Journal of Graphics Tools”. Vol 3, #1, Mar 1998.
- Mitchell, J., McTaggart, G. and Green, C. 2006. Shading in Valve’s Source Engine. In *SIGGRAPH Courses, Advanced Real-Time Rendering in 3D Graphics and Games* 2006.
- Ownby, J.-P. Hall, R., and Hall, C. Rendering techniques in Toy Story 3. In SIGGRAPH Courses, Advances In Real-Time Rendering in 3D Graphics and Games 2010
- Perthius, C. 2010. MLAA in God of War 3. In Sony PS3 DevCon 2010.
- Reinhard, E., Stark, M., Shirley, P., and Ferwerda, J. 2010. Photographic tone reproduction for digital images. In SIGGRAPH 2002.
- Rorke, J. 2010. Lighting Volumes. In GameFest 2010
- Sloan, P.-P. 2008. Stupid Spherical Harmonics Tricks. In *GDC* 2008.
- Tatarchuk, N. 2005. Irradiance Volumes for Games. In GDC Europe 2005.
- Tchou, C. 2006. HDR the Bungie Way. In GameFest 2006.
- Williams, M. 2010. Hustle Kings Lighting. Retrieved from http://www.voofoostudios.com/?p=156

Extra Slides

Stereo 3D Post Processing Gotchas

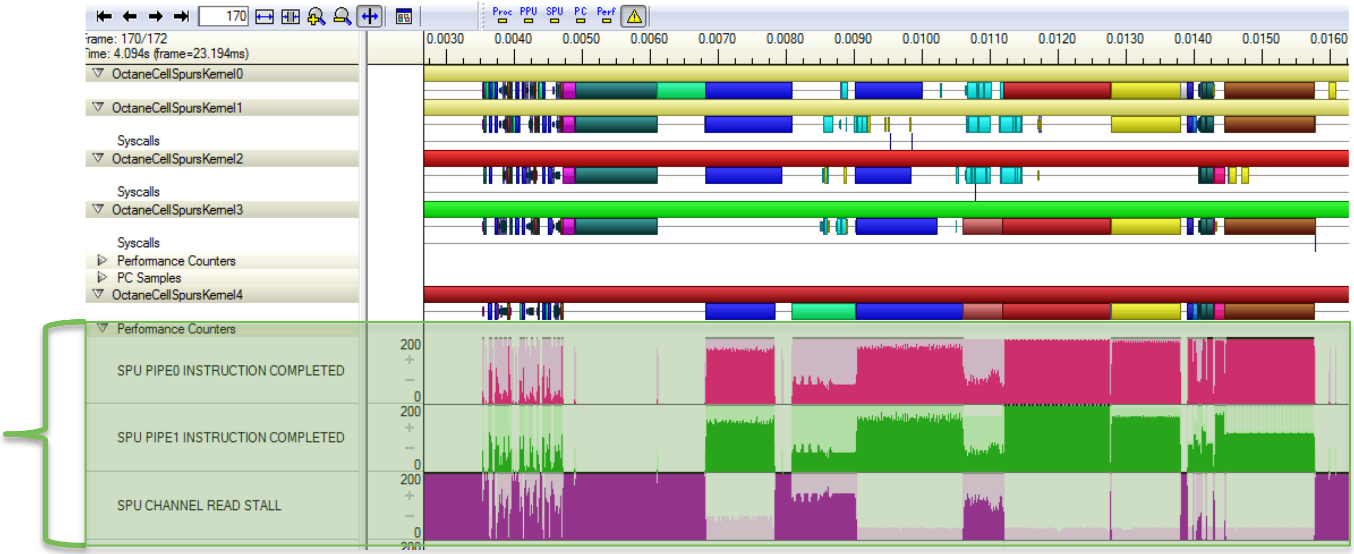
SIGGRAPH2011

- Pixel separation can get high when coming out of screen, hence more disocclusion artificats
  - So clamped min Z
- Small TVs = Higher Pixel separation
  - Clamped distortion to ensure max pixel separation stayed within decent limit

SPU Tips and Tricks

- Use SPA (SPU Pipelining Assember)
  - Optimizes loops by pipelining, to achieve maximum instruction throughput
  - In some cases this nearly speeds it up 2x
- Double buffer SPU input and output buffers, most SPU effects can run at near 100% CPU utilization, without any memory stalls

SPU Post Process Performance



Execution vs. DMA stalls

Stereo 3D issues

- Out of screen 3D is difficult
  - Windows violation
- Uncomfortable eye positions/movements
  - 2D focuses only on screen (comfortable)
  - 3D focuses in front and behind
    - Can’t focus on the whole screen at once
    - Quick depth focus change is hard
    - Going cross-eyed is uncomfortable

© Disney/Pixar

Appendix A - PS3 Luv sample code

SIGGRAPH2011

GPU Encode:
Modified function from LogLuv [Karis]:

```
const static float fx16Scale = 8192.0 / 65535.0;
vResult.zw = unpack_4ubyte( pack_2ushort( sqrt(Xp_Y_XYZp.y) * fx16Scale )).xy;
```

SPU Decode:
cuflt = Convert Unsigned Integer To Float
fm = Floating Point Multiply

```
cuflt lum, lum, 13
fm lum, lum, lum
```