Advanced Real-Time Depth of Field Techniques

Aaron Lefohn

(with additional material from
AMD, NVIDIA, and DICE)

Overview

- History
  - Post-processing was earliest use of GPGPU in games
  - Started out life in pixel shaders (still used today)
  - ComputeShader/OpenCL open up new optimization possibilities due to shared memory and synchronization

- This lecture looks at one post-processing effect, depth-of-field, esp. looking at how they use the programming models
  - Gather method
  - Spreading
  - Simulated diffusion model

Questions to Consider…

- When to use Compute/OCL versus pixel shaders?
- How are the problems parallelized?
  - Data-parallel algorithms (scan, reduce, …)?
  - Hybrid parallel-serial algorithms?
  - Other forms of parallelism?

- Caveat
  - This just scratches the surface. The more you optimize these techniques, the muddier the distinction becomes between data- and task-parallelism

Credits / Disclaimer

- My own slides from Pixar tech report and dissertation
  - Kass, Lefohn, Owens, "Interactive Depth of Field Using Simulated Diffusion on a GPU", Pixar Technical Report, 2006. http://graphics.pixar.com/library/DepthOfField/
  - Lefohn, "Glift: Generic Data Structures for Graphics Hardware," Ph.D. Dissertation, UC Davis, 2006. http://graphics.idav.ucdavis.edu/~lefohn/work/dissertation/
- Justin Hensley, researcher at AMD
  - Ladybug demo http://developer.amd.com/samples/demos/pages/atiradeonhd5800seriesrealtimedemos.aspx
- NVIDIA GDC 2010 presentation: Oles Shishkovtsov (4A Games) and Ashu Rege (NVIDIA)
  - Metro 2033 http://developer.download.nvidia.com/presentations/2010/gdc/metro.pdf
- DICE M.S. thesis: Per Lönroth and Mattias Unger
  - "Advanced Real-Time Post-Processing using GPGPU Techniques" http://publications.dice.se/publications.asp?show_category=yes&which_category=Thesis

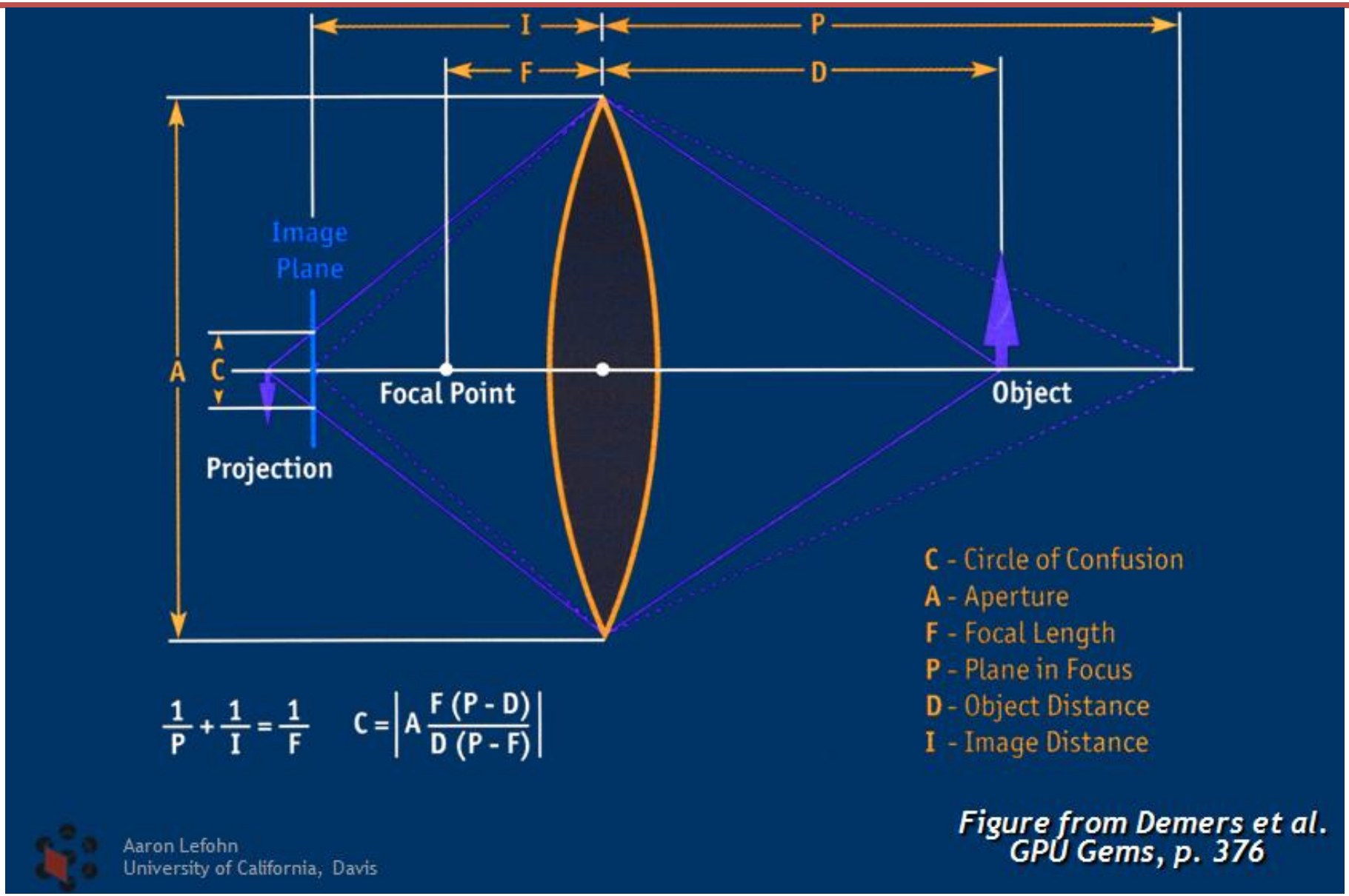
Depth of Field

Image from www.dpchallenge.com

Depth of Field

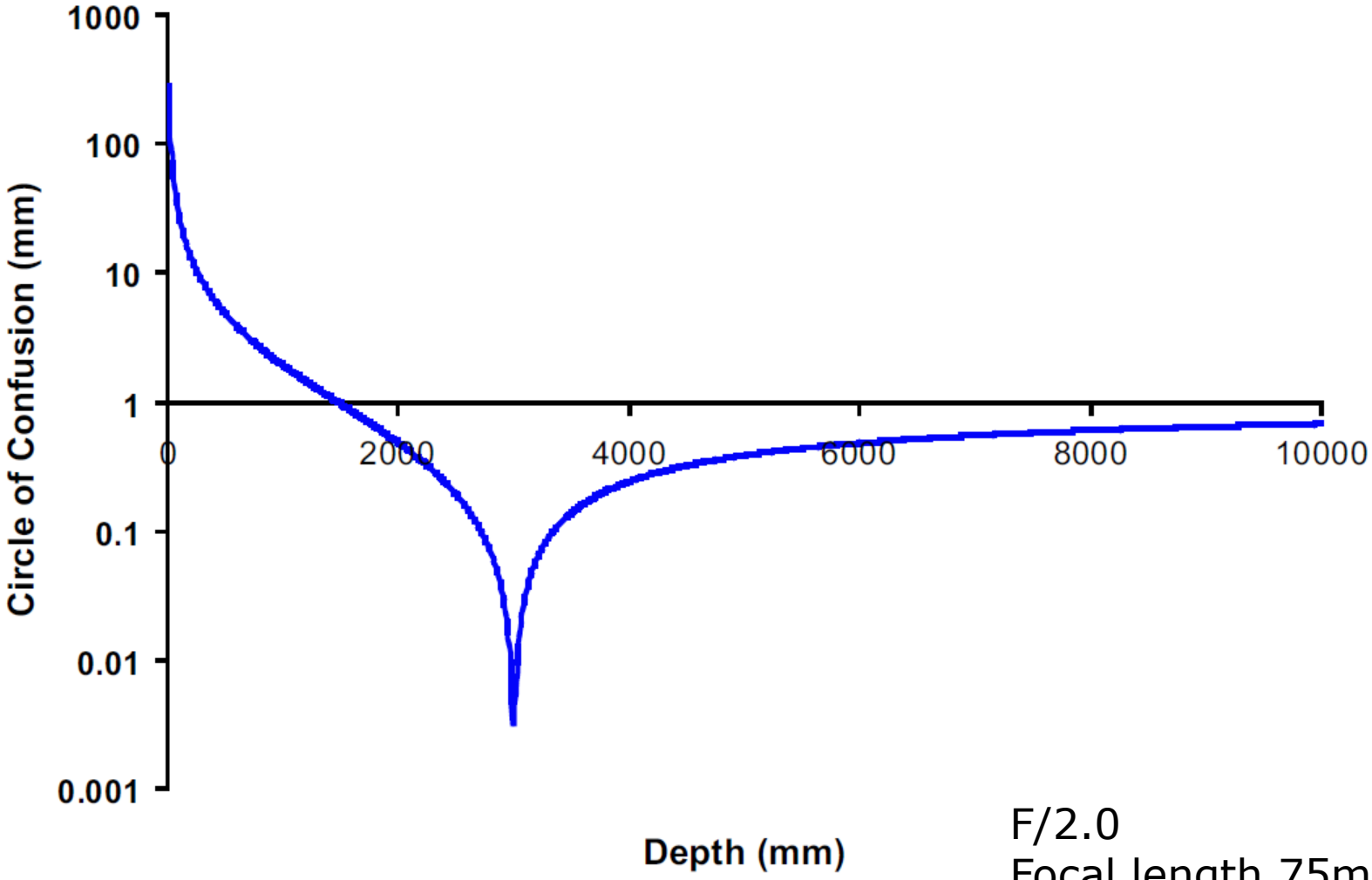
Image from Ren Ng

Circle of Confusion (CoC)



Aaron Lefohn
University of California, Davis

Circle of Confusion as Function of Depth



F/2.0
Focal length 75mm

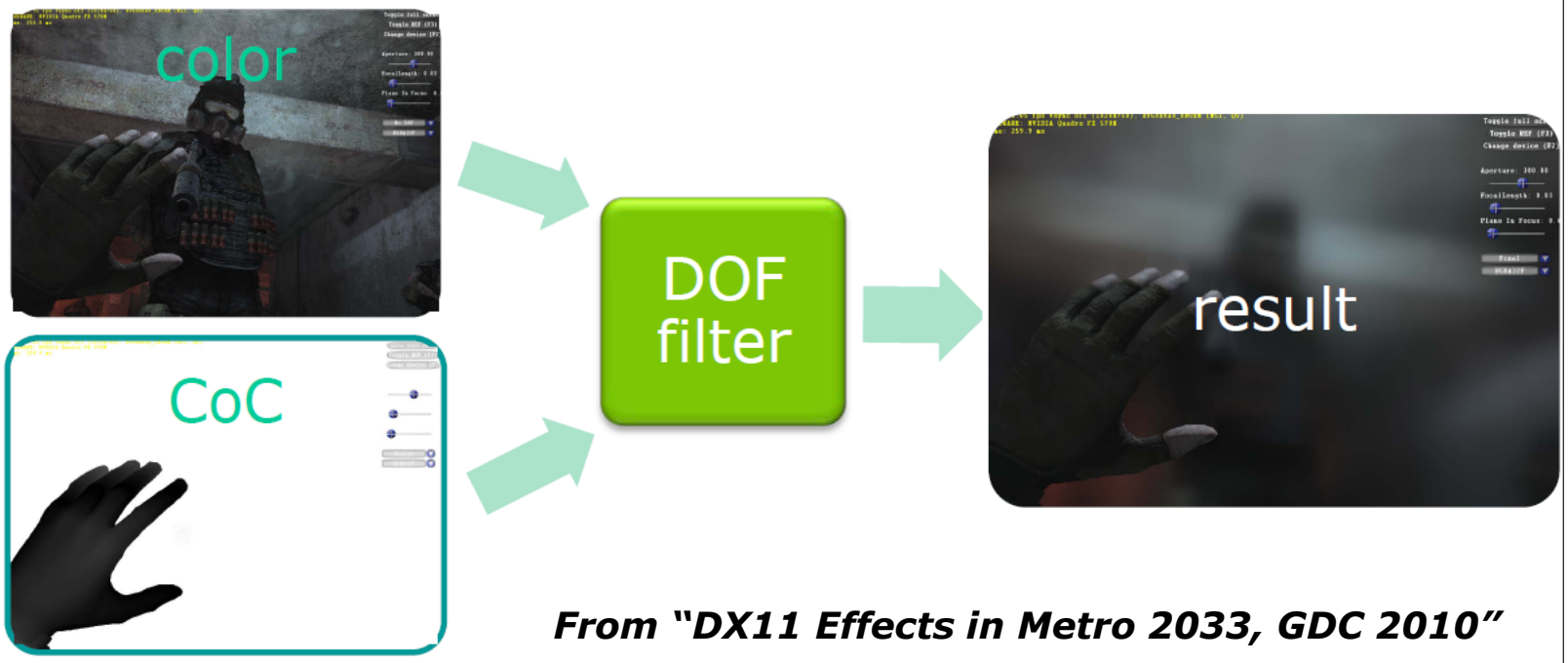
Depth of Field (DOF) in Graphics

- Accurate simulation
  - Sample from multiple positions on “lens”
  - Offline rendering technique (slow). Researchers working on real-time rendering pipelines that can do real DOF, but lots of work still to do.

- Post-process approximation
  - Compute DOF from RGBZ image
  - Spread pixel color out over area based on CoC

Depth of field effect

- Post-processing input color layer by using depth layer to calculate CoC (circle of confusion)



From "DX11 Effects in Metro 2033, GDC 2010"

Post-Process DOF

- Gather: Blur each pixel by size of CoC
  - Lots of GPU tricks that do this
  - Lots of problems
- Scatter: Splat each pixel with size based on CoC
  - Used in software film preview, not interactive
    - Sort all pixels
    - 32-bit floating-point blending
    - Hard to conserve energy
  - Justin Hensley et al @ AMD have found a way to approximate this approach using DX11 ComputeShader

Gather Depth of Field

- Basic algorithm
  - For each pixel
    - Compute final color by computing the convolution over the filter size determined by the circle of confusion (CoC)
- Challenges
  - How avoid edge bleeding?
  - How compute variable width blurs efficiently?
  - How compute large blurs efficiently?

Gather 1: Poisson Disk Blur

- Key idea
  - Use fixed number of samples for all filter sizes
  - Sample full-res and low-res image and interpolate based on CoC
  - Conditionally reject samples if cross depth discontinuity

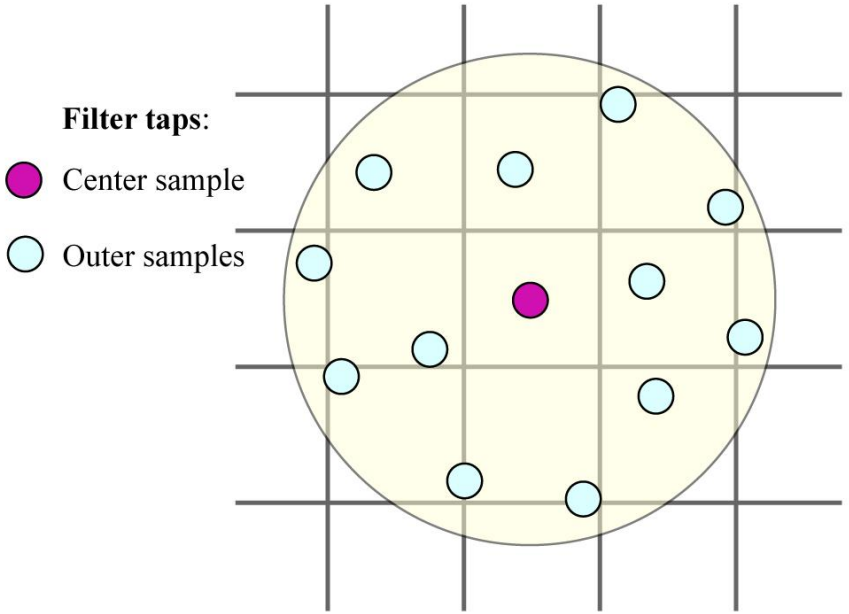
(commonly used and many citations: ShaderX, GPU Gems, etc)

- Solves
  - Fast variable-width blur
  - Partially addresses edge bleeding

- Problems
  - Aliasing caused by under-sampling large filters
  - Edge bleeding

Gather 1: Poisson Disk Blur

- Used fixed number of samples for all filter sizes (aliases but fast)
- Use one downsampled image to cheaply increase blur radius
- Inspect depth of each dample to avoid some edge bleeding

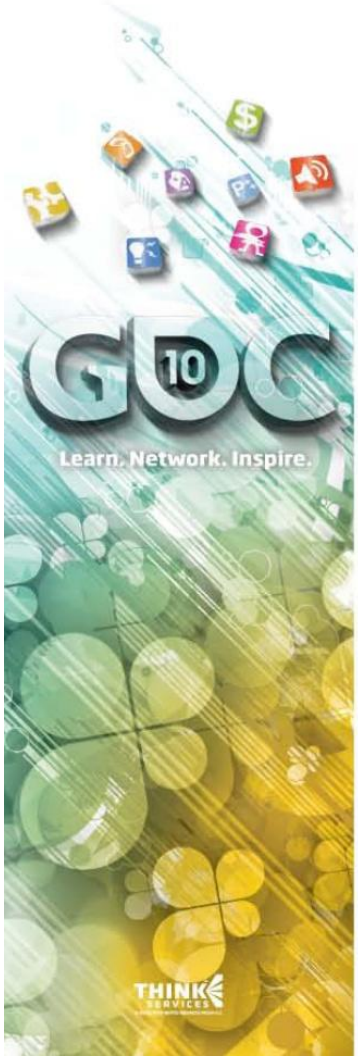


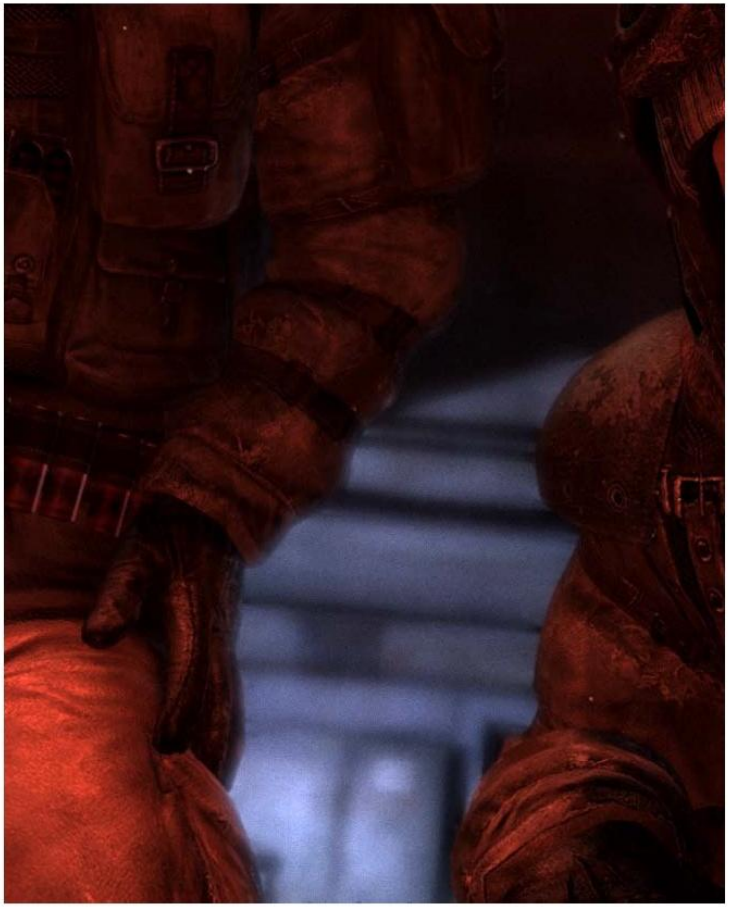
Pictures from DICE research
(Per Lönroth, Mattias Unger)
http://publications.dice.se/publications.asp?show_category=yes&which_category=Thesis

Gather 1: Compute requirements

- Not much
  - Works in DX9
  - Down-sample image (pixel shader, low-res quad)
  - Apply Poisson filter (pixel shader, full-screen quad)

Bleeding artifacts

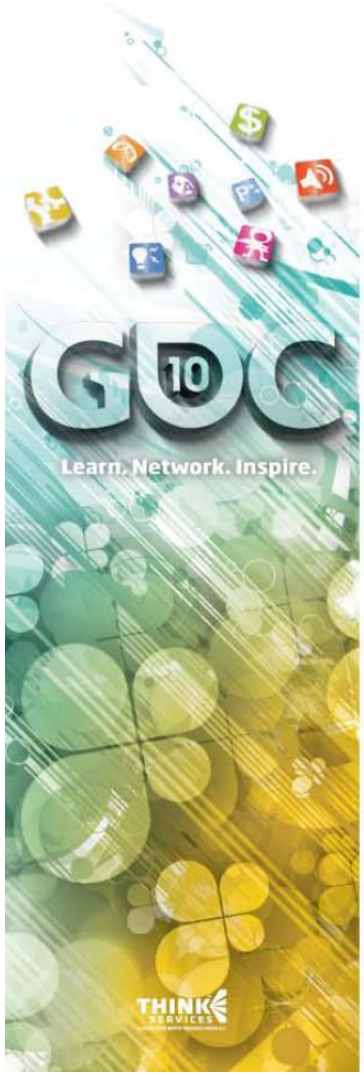




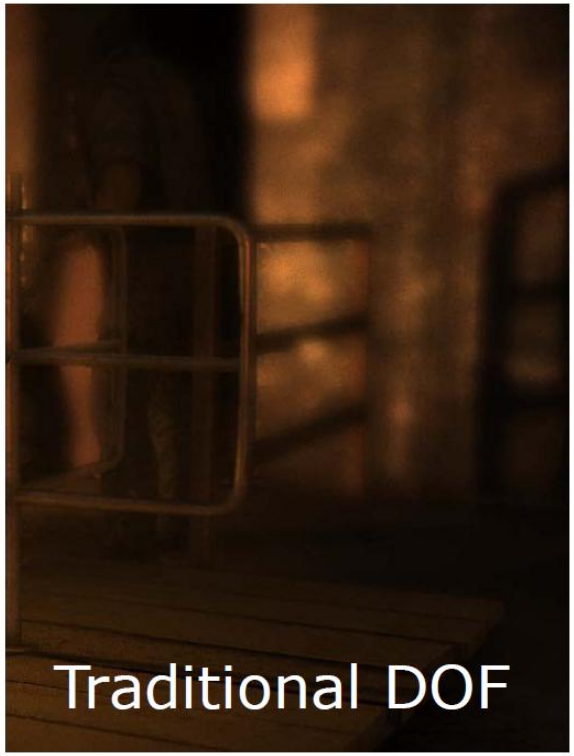
From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

Benefits



- No color bleeding



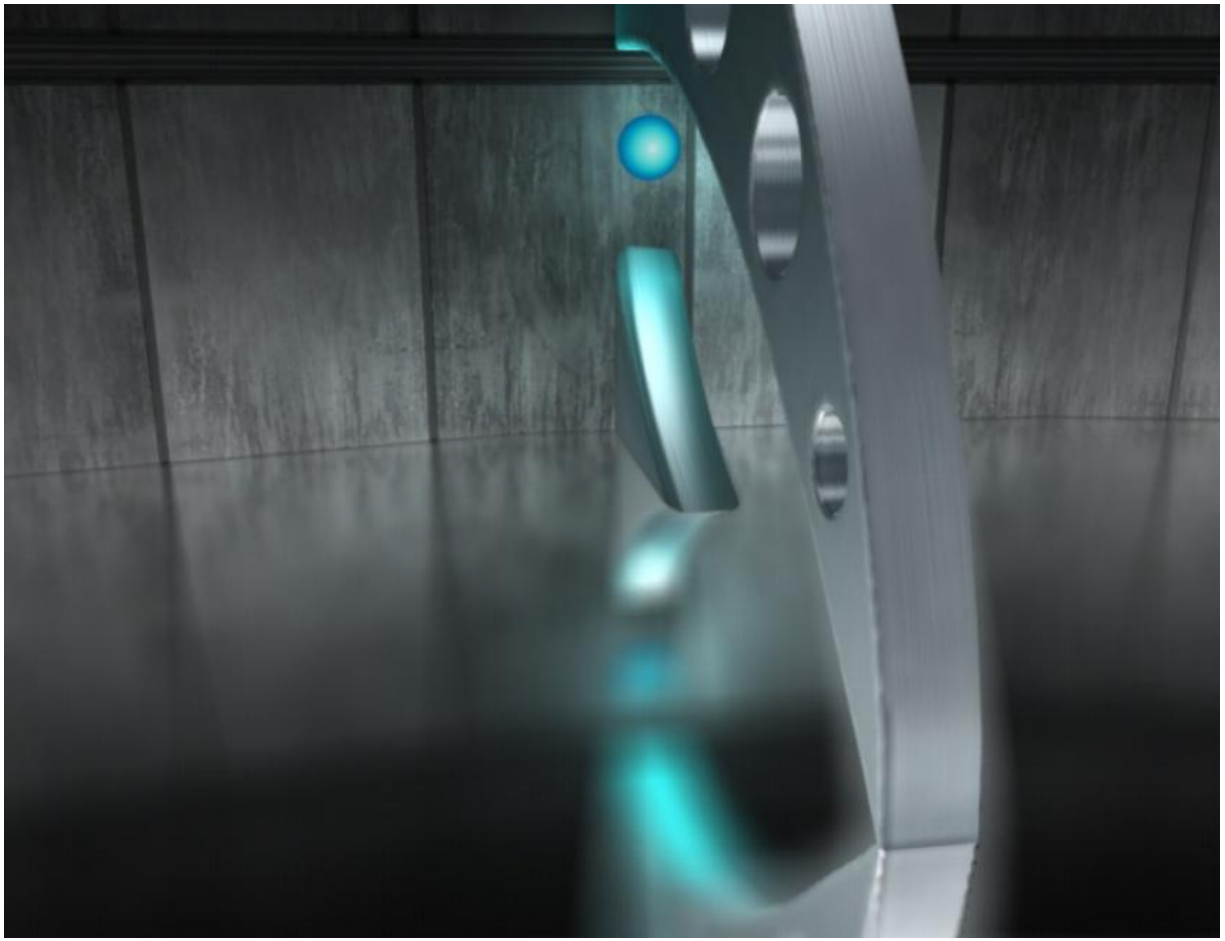
From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

Gather 2: Summed Area Tables

- Key idea
  - Pre-integrate the image so that you can quickly perform convolutions of arbitrarily (and variably) sized axis-aligned rectangular regions
- Solves
  - Fast variable width blur
  - Fast large blurs
- Problems
  - Edge bleeding
  - Precision problems (repeated FP add over large image)

Dynamic Summed Area Tables



"Fast Summed-Area Table Generation and its Applications,"
Hensley et al., Eurographics 2005

Summed Area Tables

- Goal
  - Filter over arbitrary rectangular regions of a texture in constant time
  - More flexible variable-width filtering than mipmaps
    - Non-square filter regions
    - Filter region sizes not limited to power-of-two LODs

- Idea
  - Create a texture where each texel contains the sum of all elements above and to the left of the original texture
  - Subtract SAT value at lower-right corner from SAT value at upper-left and divide by area

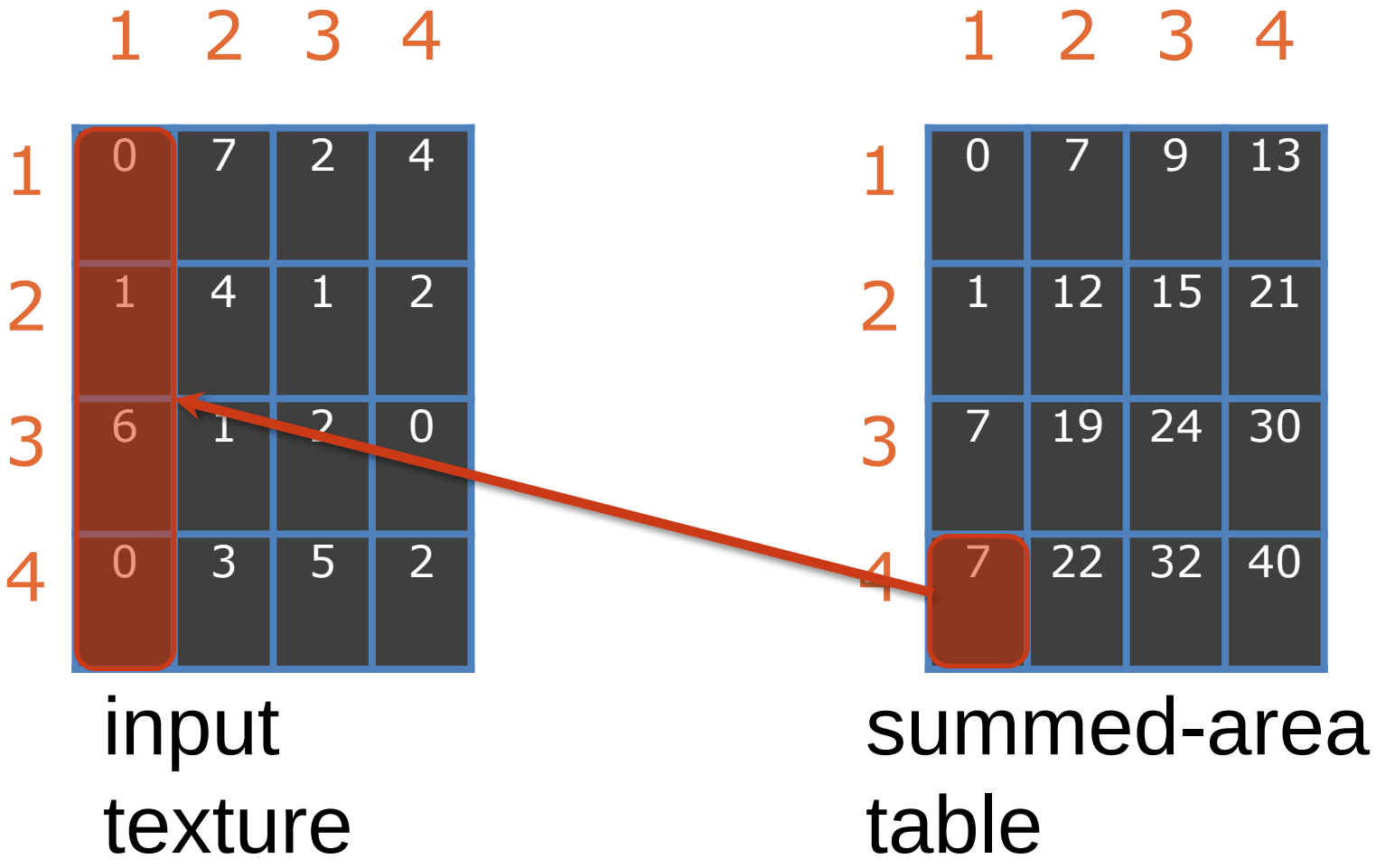
"Summed-area tables for texture mapping," Crow, SIGGRAPH 1984

Summed-Area Tables (SATs)

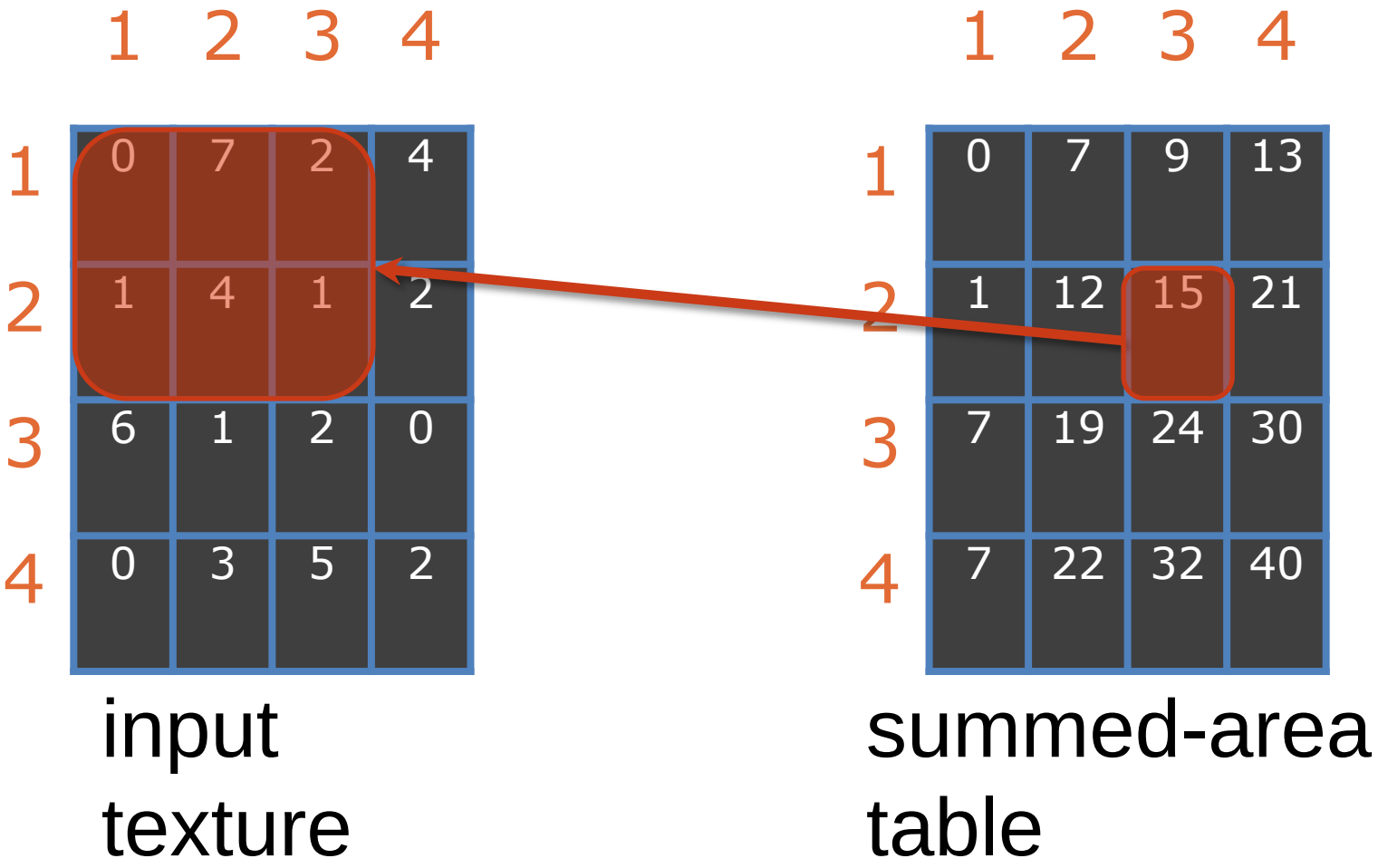
- For a left-handed coordinate system, each element Smn of a summed-area table S contains the sum of all elements above and to the left of the original table/texture T [Crow84]

$$S_{mn} = \sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij}$$

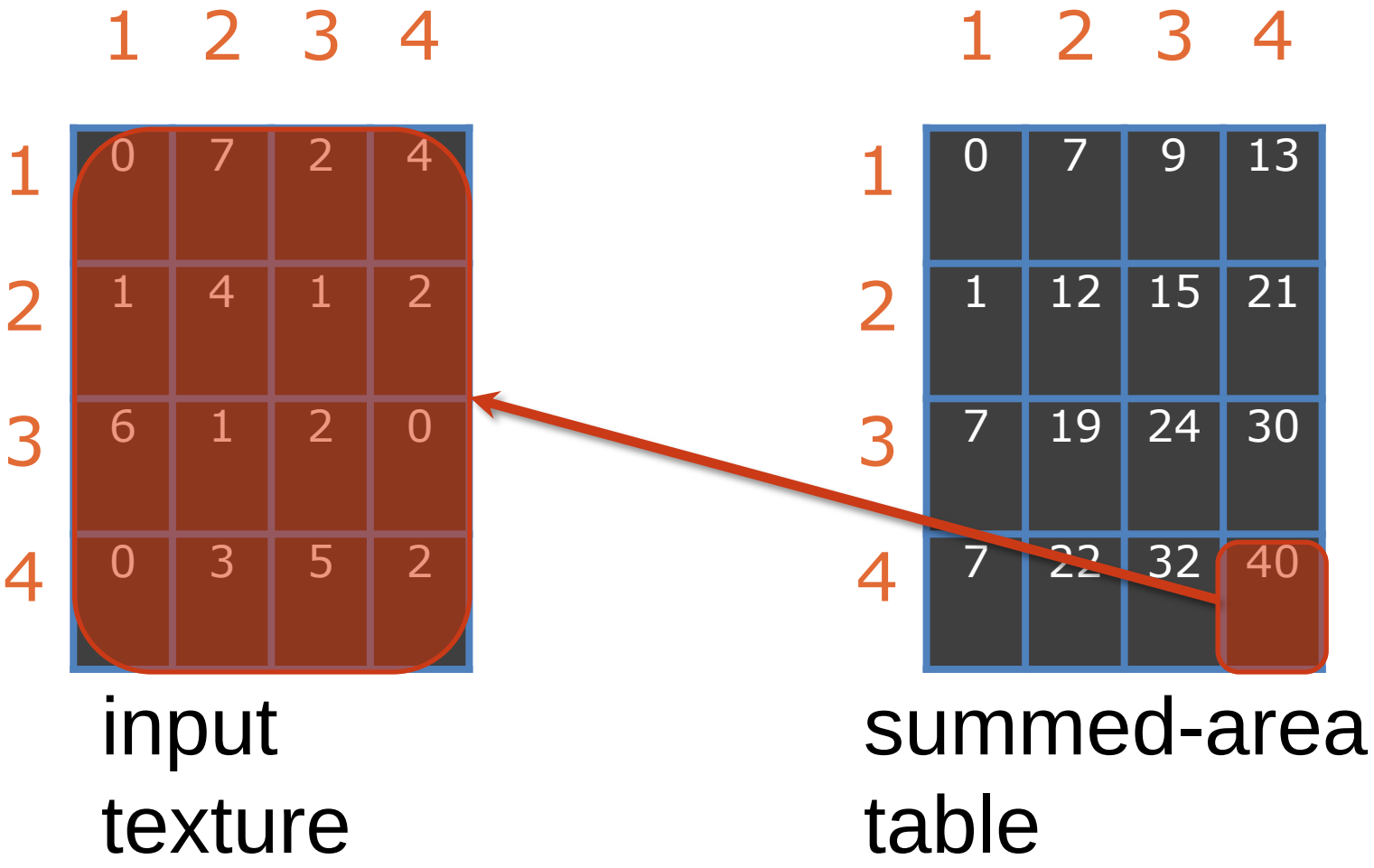
Summed-Area Tables (SATs)



Summed-Area Tables (SATs)

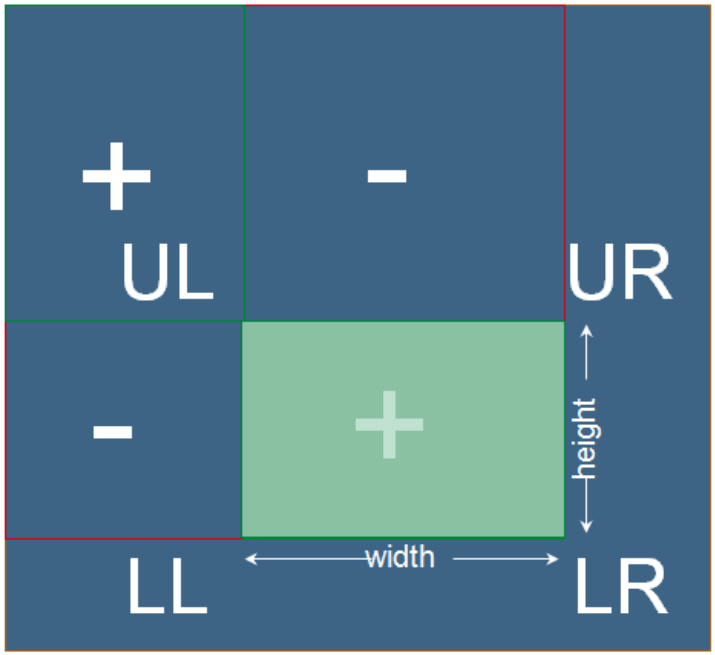


Summed-Area Tables (SATs)



Using a Summed-Area Table

- Summed-area tables enable the averaging rectangular regions of pixel with a constant number of reads



$$average = \frac{LR - LL - UR + UL}{width * height}$$

Building Summed Area Tables

- Implementation
  - Step-efficient 2D parallel-prefix "scan" operation

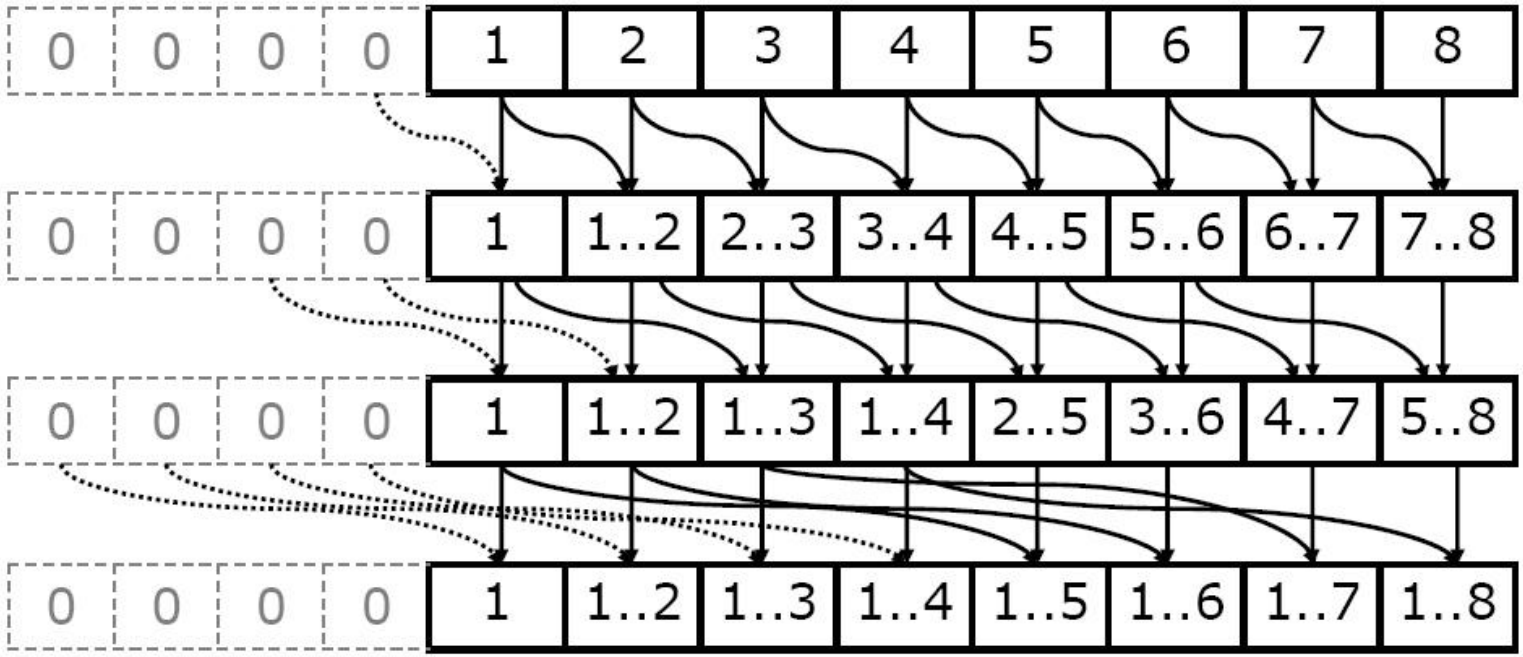
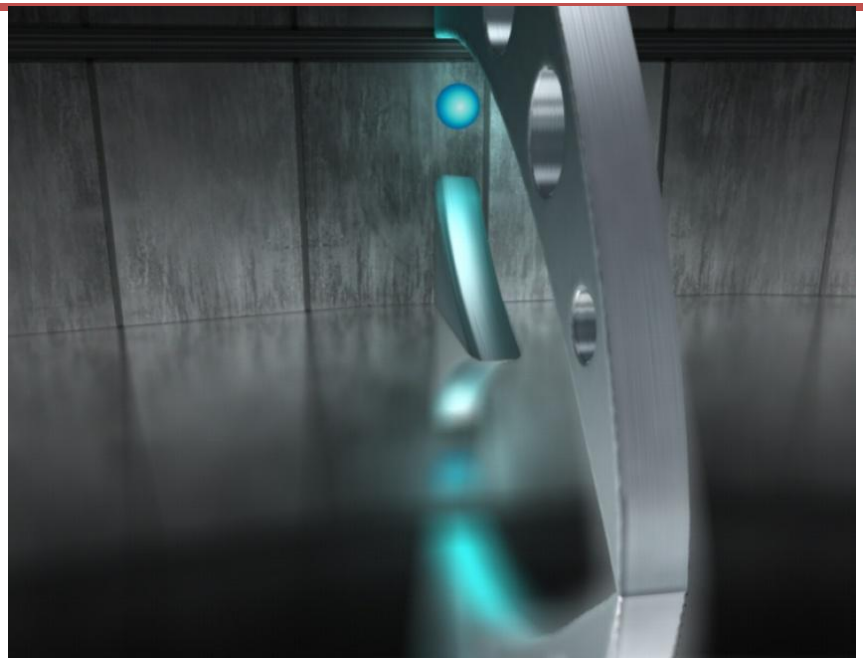
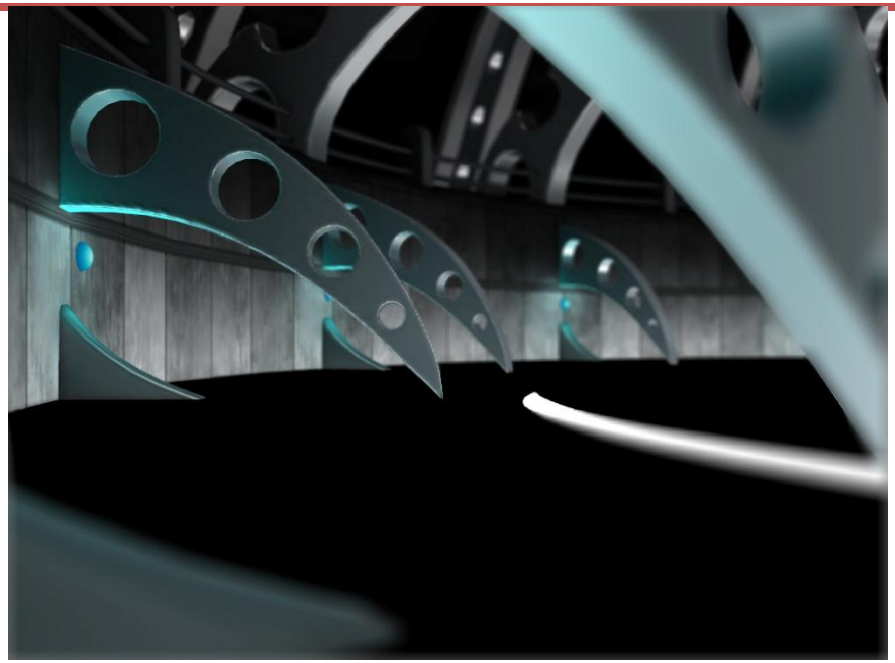


Figure from "Summed Summed-Area Tables Area Tables And Their Application to Dynamic Glossy Environment Reflections," Scheuermann, Game Developer's Conference, 2005

Dynamic Summed Area Tables



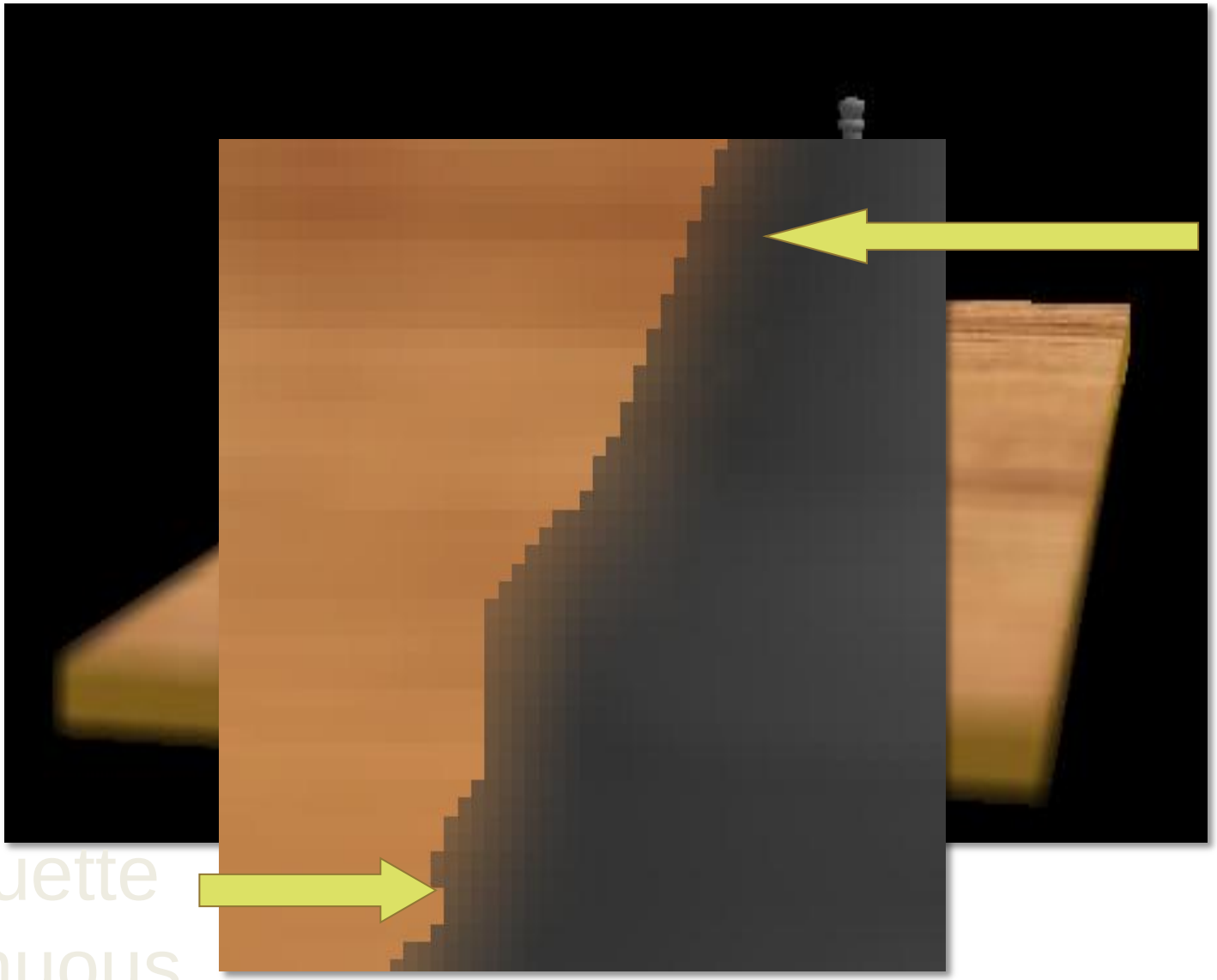
Glossy reflection



Depth-of-field

Images from "Fast Summed-Area Table Generation and its Applications
Hensley et al., Eurographics 2005
- Used in ATI's "Ruby" Demo

Gather Depth of Field Errors



Scatter Instead of Gather?

- Film preview uses brute force scatter method $O(n^2)$
  - Best possible result with a single depth layer
  - Correct result if have all depth layers

- Justin Hensley et al at AMD recently came up with a way to approximate scatter DOF using "inverse summed area tables"
  - Does not do true scatter, but instead provides fast higher-quality (higher-order) filter kernels
  - "Ladybug" demo available online (Paper is in review)
    - http://developer.amd.com/samples/demos/pages/atiradeonhd5800seriesrealtimedemos.aspx

Heat Diffusion Depth of Field

- Idea
  - Think of depth-of-field blur as anisotropic heat diffusion
- Solves
  - Very high quality filters
  - O(1) variable width blurs
  - O(1) large blurs
  - Separates sharp edges and blurry regions
- Problems
  - PDEs can hard to control if anything goes wrong
  - High overhead to achieve "constant time" variable width blurs
  - Must perform multiple heat diffusions on background CoCs and foreground CoCs
  - Like all post-processing methods, foreground blurs require capturing all depth layers up to focal plane

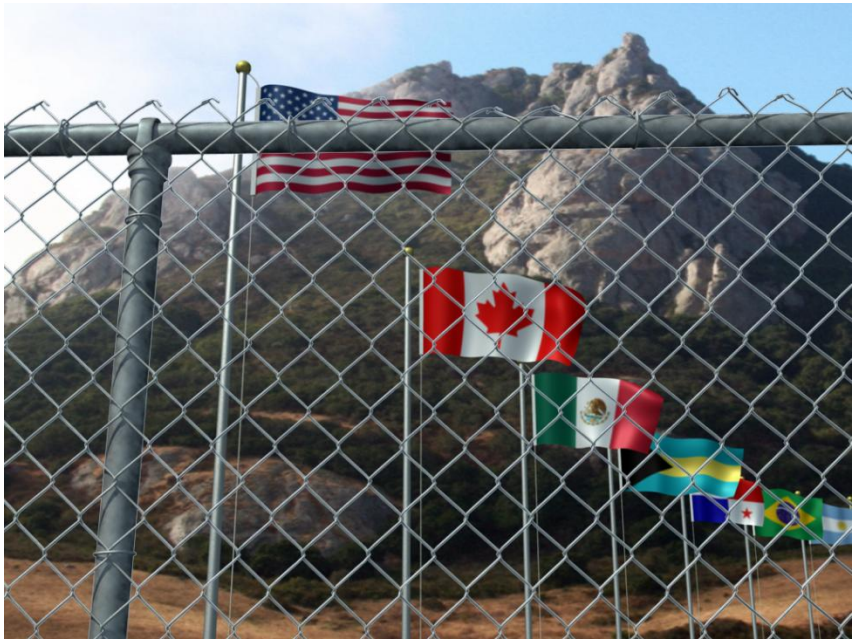
Heat Diffusion Depth-of-Field?

- Idea
  - Cast depth-of-field blur problem in terms of anisotropic heat equation
  - Input image is "initial heat distribution"
  - Define "material model" based on CoC
  - Obtain DOF result by allowing heat to diffuse
- Implementation
  - Solve heat equation with separable implicit solver
  - Implicit solver equivalent to IIR filter
    - Build and solve 1000s of tridiagonal linear systems

"Interactive Depth of Field Using Simulated Diffusion on a GPU," Kass, Lefohn, Owens, Pixar Technical Report, 2006

Infinite Impulse Response (IRR) Filters

- Another method for constant-time, spatially-varying filters
  - Can produce arbitrarily wide blurs in constant cost
  - Also called "recursive filters"



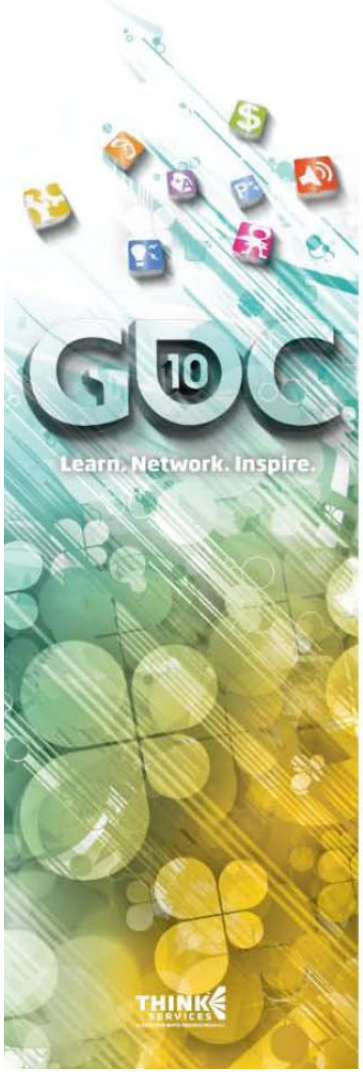
Infinite Impulse Response Filters

- Idea
  - Use value and filtering result of adjacent pixel to determine filtered value of current pixel
    - Enables information to travel across entire image
    - Requires communication between pixels

Heat Diffusion Depth-of-Field

Image from "Interactive Depth of Field Using Simulated Diffusion on a GPU," Kass et al., Pixar Technical Report #06-01, Jan. 2006

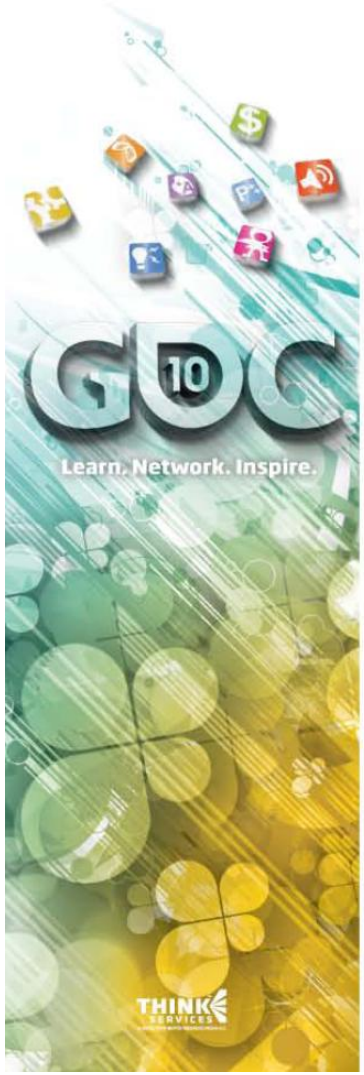
Diffusion DOF in Metro



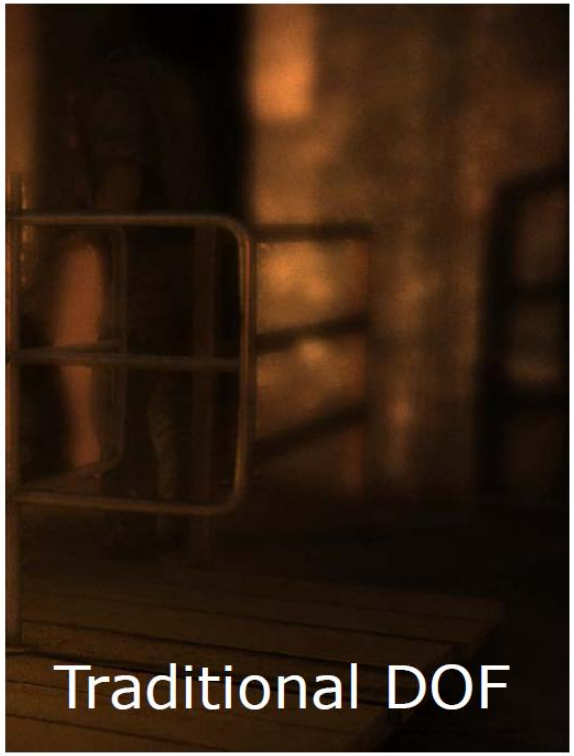
From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

Benefits



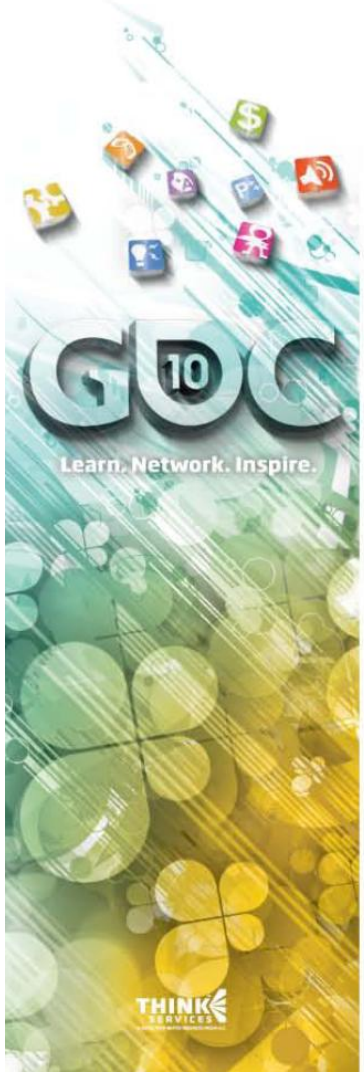
- No color bleeding

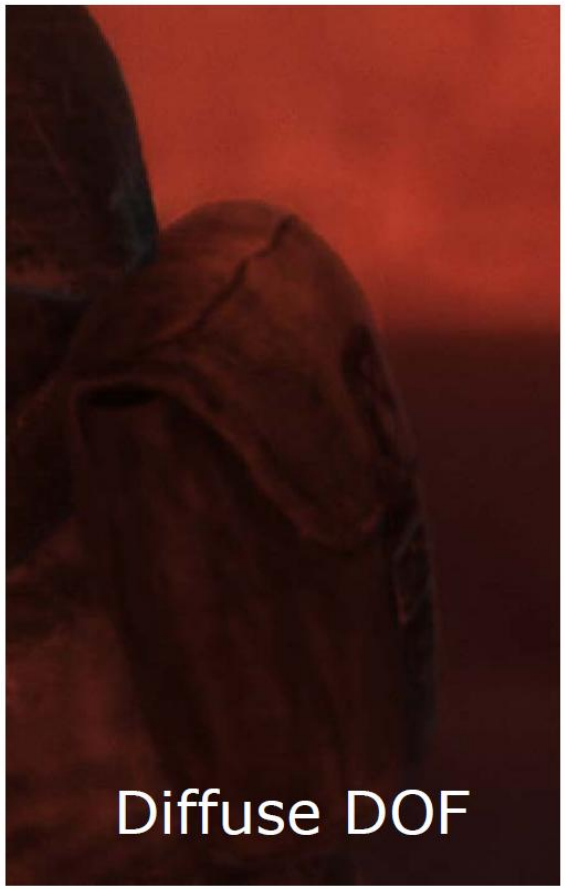


From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

Benefits – detail view

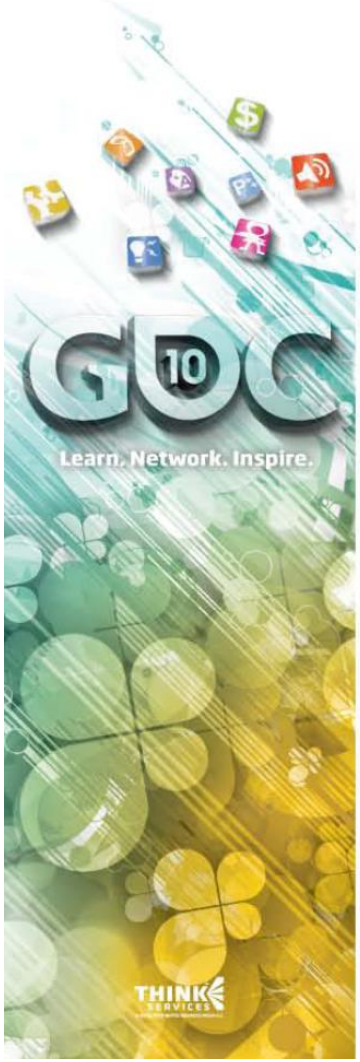




From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

Implementation



- We cast DOF problem in terms of differential equation

$$\frac{\partial u}{\partial t} = \nabla \cdot (\beta \cdot \nabla u)$$

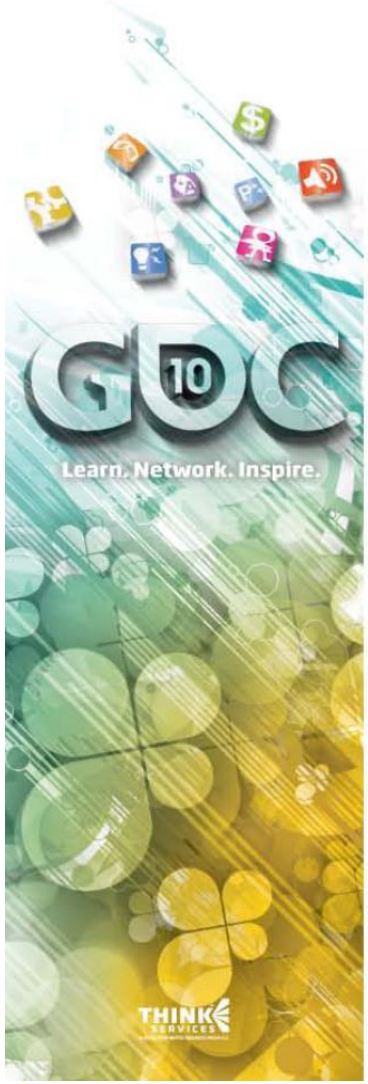
$u(x, y)$ Image color

$\beta(x, y)$ Circle of confusion

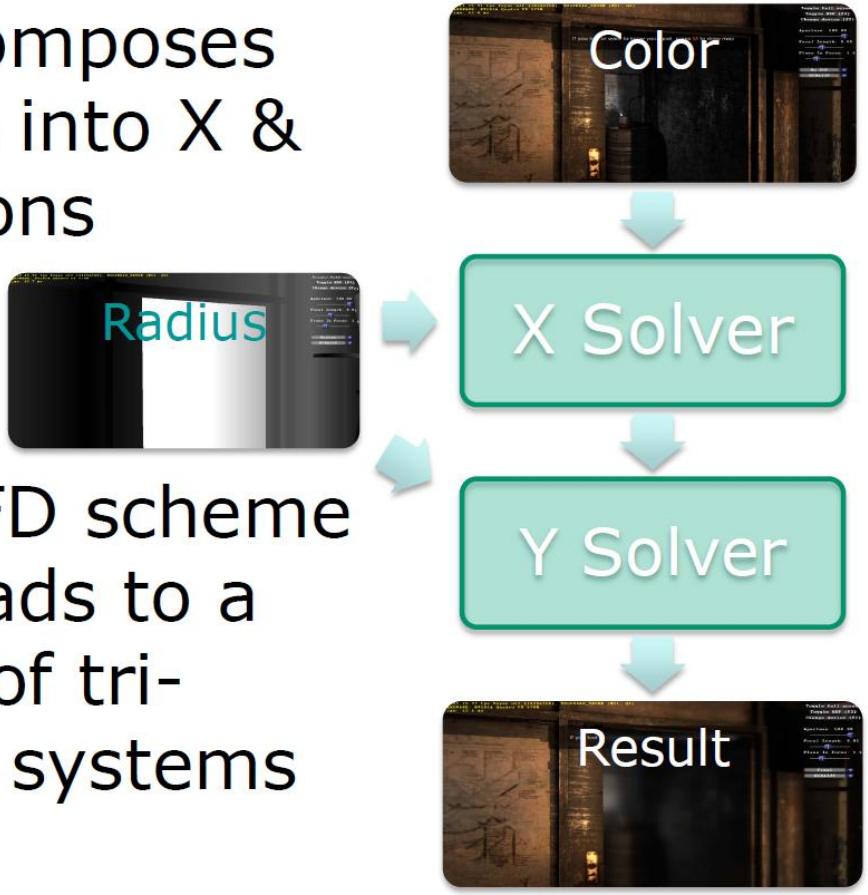
- Using Alternate Direction Implicit (ADI) numerical method

From "DX11 Effects in Metro 2033, GDC 2010"

Implementation



- ADI decomposes equation into X & Y directions
- Applies FD scheme which leads to a number of tri-diagonal systems



From "DX11 Effects in Metro 2033, GDC 2010"

Solving Heat Equation

- Use separable, implicit solver
  - One tridiagonal system for each row/column
  - 100s – 1000s of tridiagonal matrices

$$\begin{pmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & c_3 & \\ & & \ddots & \ddots & \ddots \\ 0 & & & a_n & b_n \end{pmatrix} \begin{pmatrix} h_1 \\ h_2 \\ h_3 \\ \vdots \\ h_n \end{pmatrix} = \begin{pmatrix} h_1^0 \\ h_2^0 \\ h_3^0 \\ \vdots \\ h_n^0 \end{pmatrix}$$

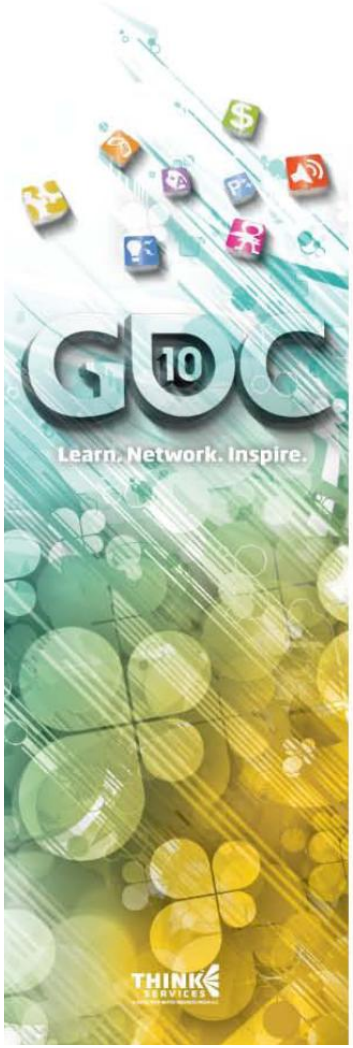
Parallelism Challenge

- Need direct tridiagonal linear solver
  - Standard serial algorithm is LU-decomposition
  - LU-decomp method uses forward- and back-substitution

```
for (int j = n - 2; j >= 0; --j) {
   u[j] -= gam[j+1] * u[j+1];
}
```

  - "Impossible" to parallelize
    - Recurrence relation
    - Parallel prefix (scan) to the rescue

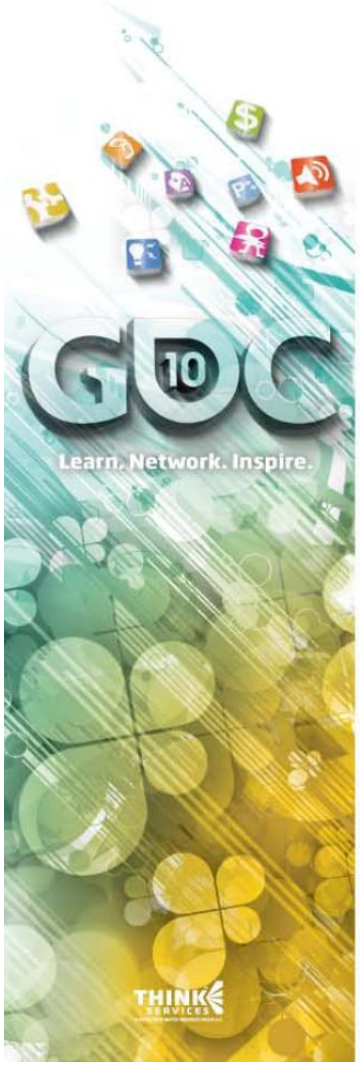
Solving tridiagonal systems



- A number of methods exist:
  - Cyclic reduction (CR)
  - Parallel cyclic reduction (PCR)
  - Simplified Gauss elimination (Sweep)
  - (see references for details)
- We use a new hybrid approach
  - PCR + Sweep

From "DX11 Effects in Metro 2033, GDC 2010"

Solving tridiagonal systems (x direction)



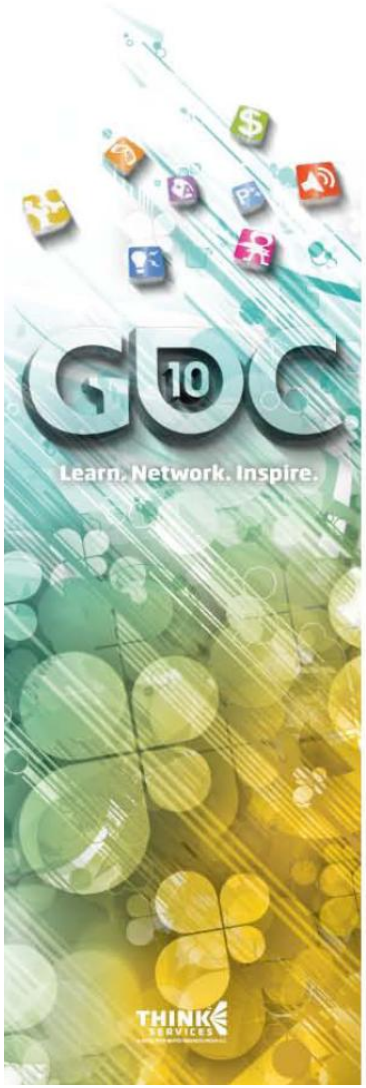
$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

- Here x_i are the per-pixel colors
- Each system is of size == WIDTH
- And # of systems == HEIGHT
- d_i are the initial conditions
- We want a GPU friendly version…

From "DX11 Effects in Metro 2033, GDC 2010"

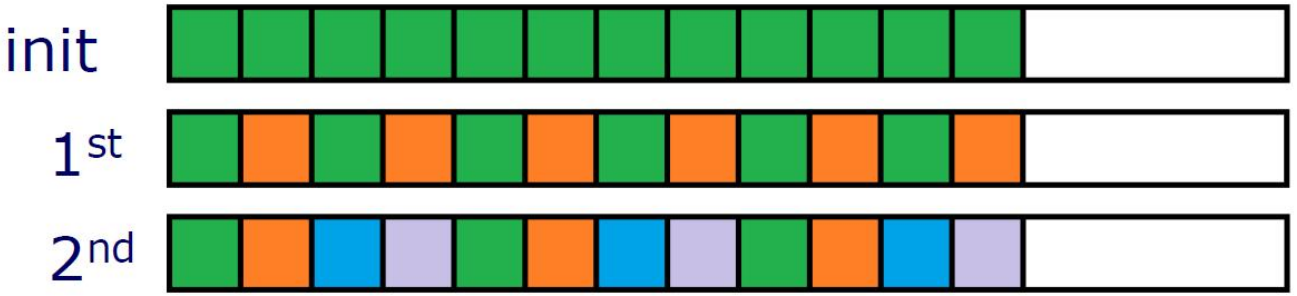
GPU Solution

- Refactor cyclic reduction
  - No scatter (we could scatter but want contiguous writes)
  - Dense iteration/output domain
- Data structure
  - Array of tridiagonal matrices
  - Each matrix is array of (a_i, b_i, c_i)
- Final computation is simply *parallel prefix* (Scan)!



Hybrid tridiagonal solver - PCR

- Few steps of parallel cyclic reduction (PCR)
  - Implemented in *pixel shader*



At each PCR step we double the number of systems and halve the size of each system → More parallel work to fill GPU.

From "DX11 Effects in Metro 2033, GDC 2010"

Work-Efficient Scan Operator

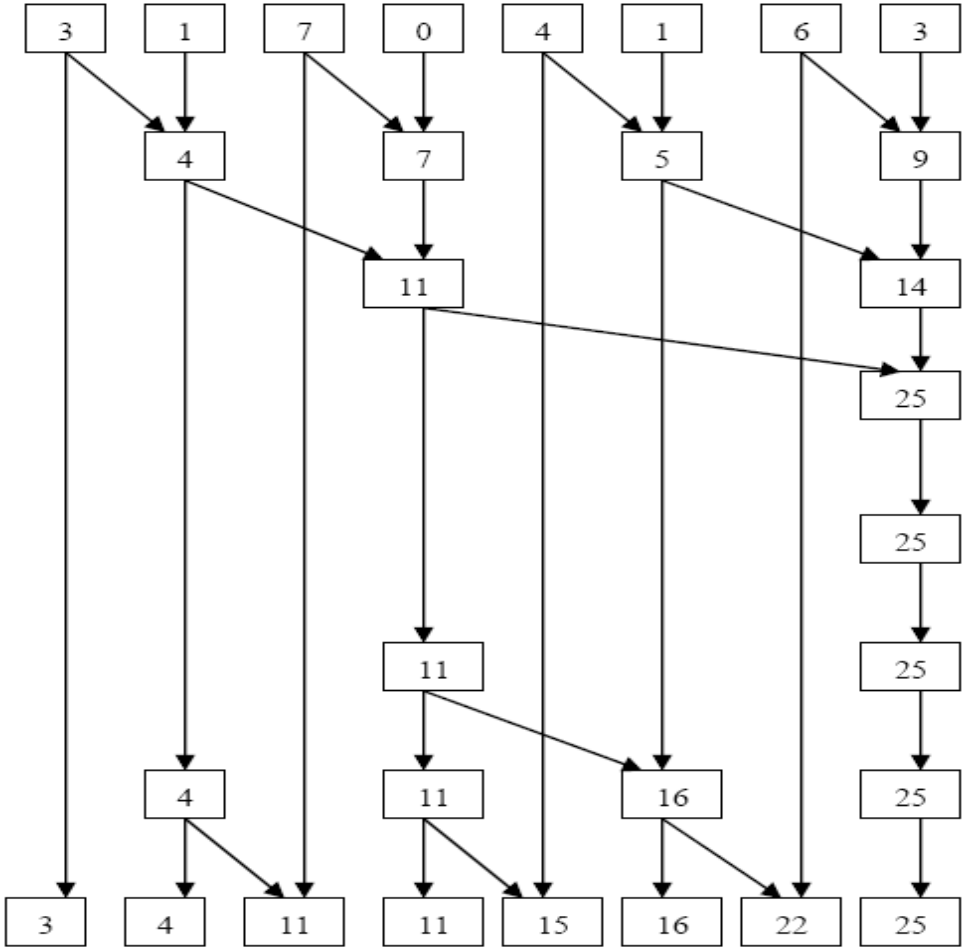
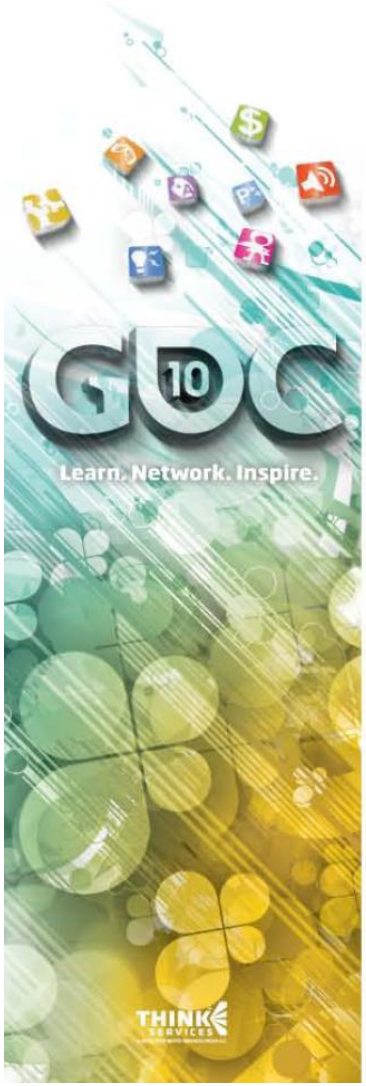


Figure courtesy of Shubho Sengupta at UC Davis

Hybrid tridiagonal solver - Sweep



- Finish with simplified Gauss elimination (Sweep)
  - Implemented in *compute shader*
- Each thread solves one system
  - Forward elimination
  - Backward substitution
  - Complexity O(N)

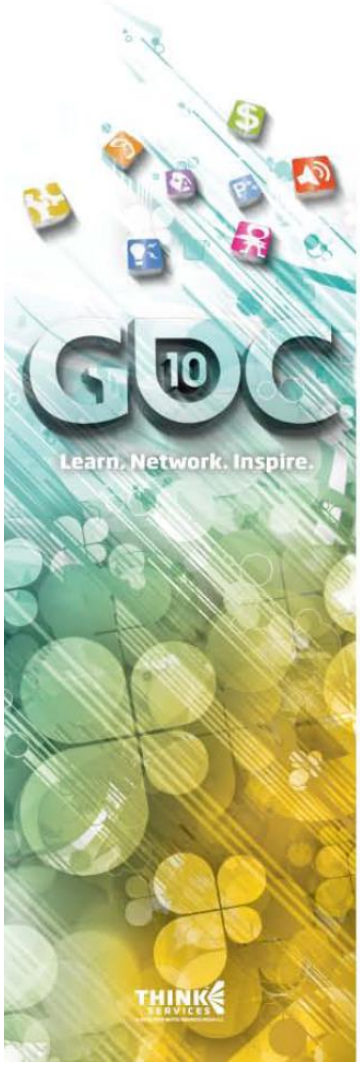
From "DX11 Effects in Metro 2033, GDC 2010"

Hybrid solver: parallelism considerations

- Use data-parallel solver when more elements than available ALUs
- Use many serial (CPU-style) solvers when problem size fits available compute

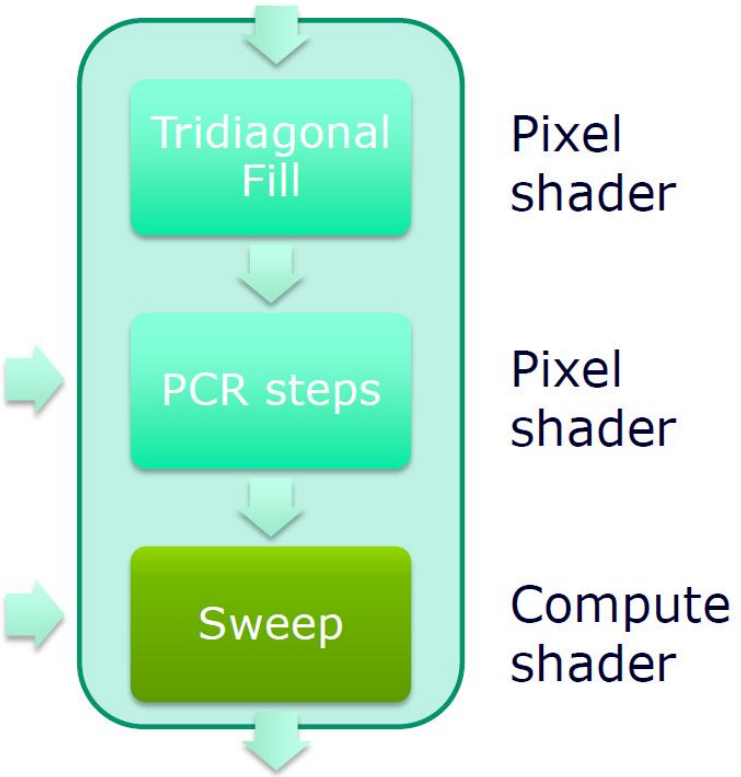
  - "Tridiagonal solvers on the GPU and applications to fluid simulation" Nikolai Sakharnykh, GTC 2009
  - "Fast tridiagonal solvers on the GPU" Yao Zhang, Jon Cohen, John D. Owens, PPoPP 2010

Tridiagonal solver in DX11



PCR steps = 3

Num systems	System size
Height	Width
Height*8	Width/8



From "DX11 Effects in Metro 2033, GDC 2010"

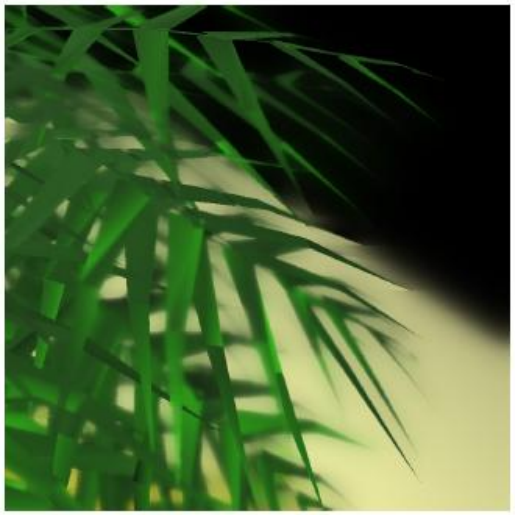
Question: Pixel Shader vs Compute Shader?

- Why did Metro 2033 use pixel shader for cyclic reduction (scan) stages and ComputeShader for the sweep?

- The best point to switch from cyclic reduction to LU-decomposition is architecture-dependent

More Diffusion DOF Details (and some problems)

Original Image



Single Layer Heat Diff. DOF

Background-only Heat Diff. DOF

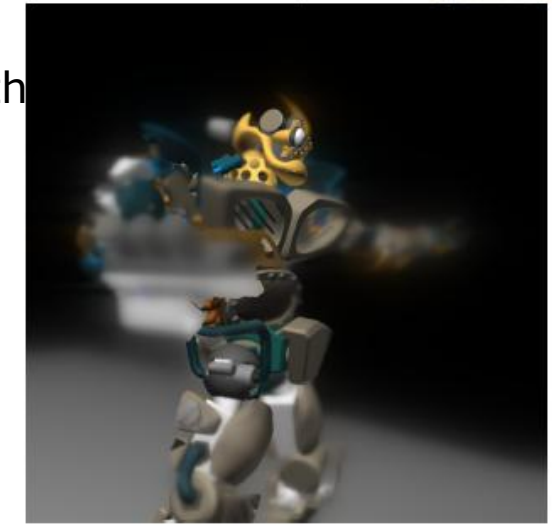
2-Layer Heat Diff. DOF

From Lefohn Ph.D. Dissertation

Foreground Blurs Require Multiple Depth Images

Original Image

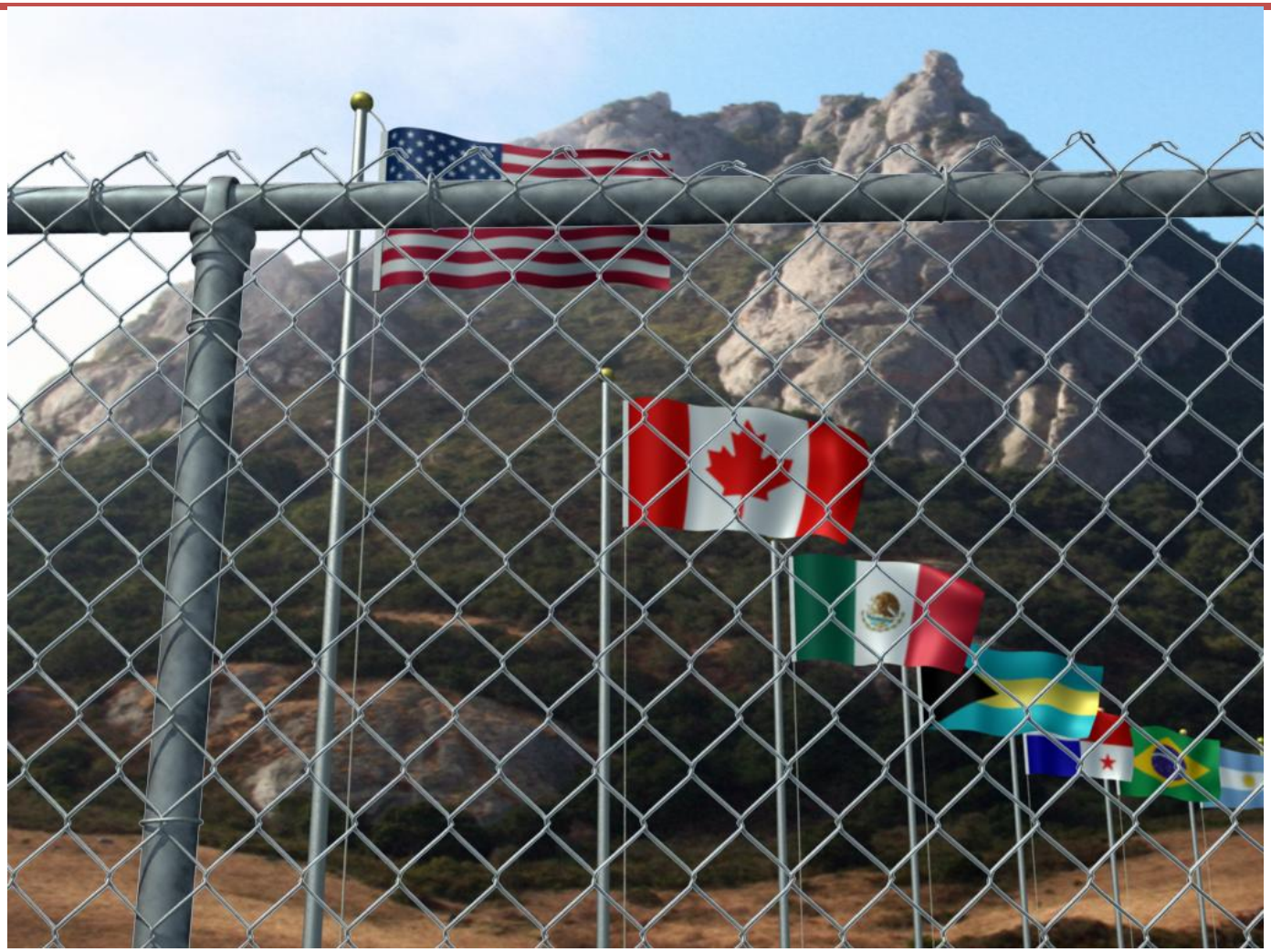
Single depth image



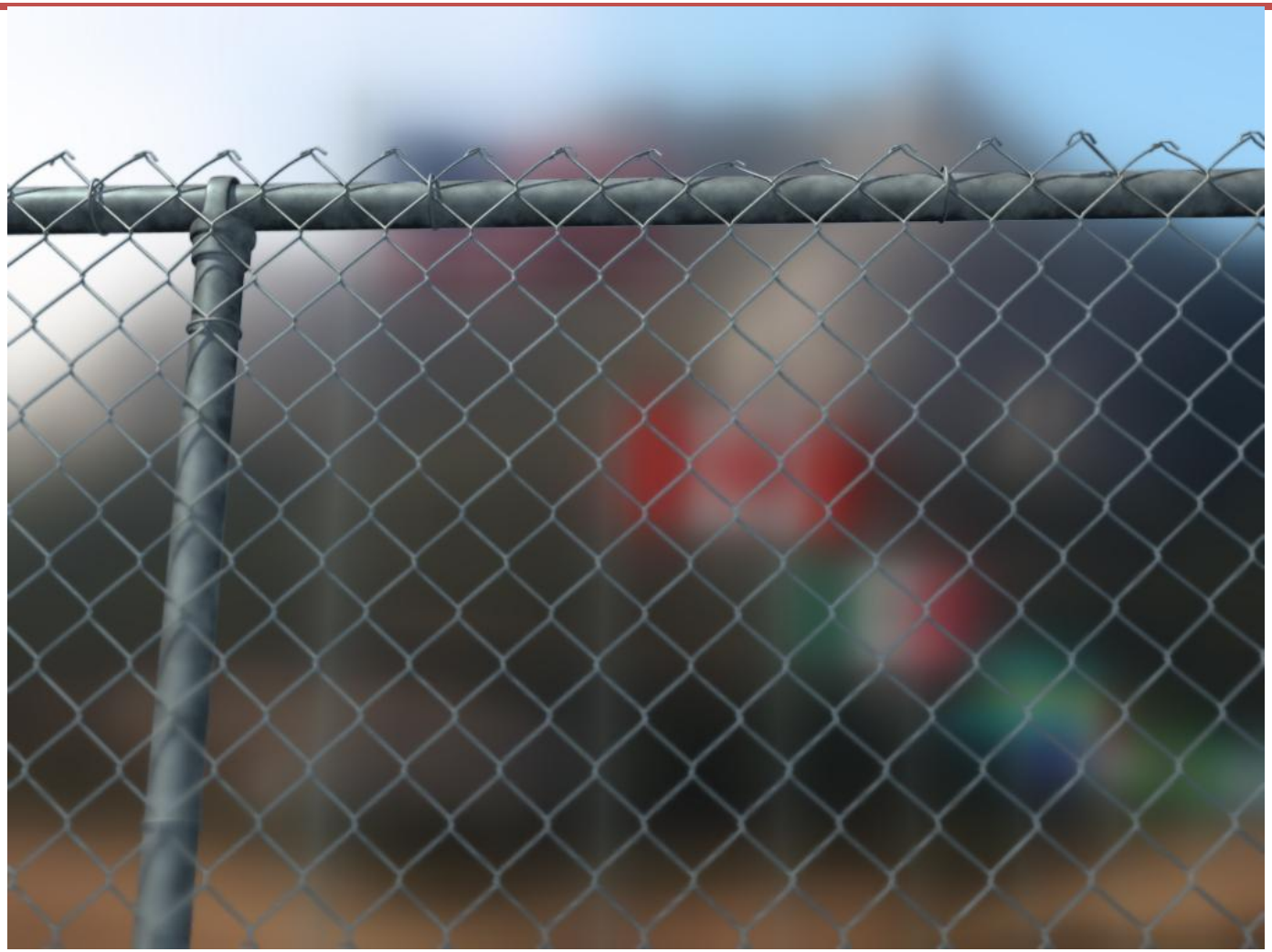
Three depth images

From Lefohn Ph.D. Dissertation

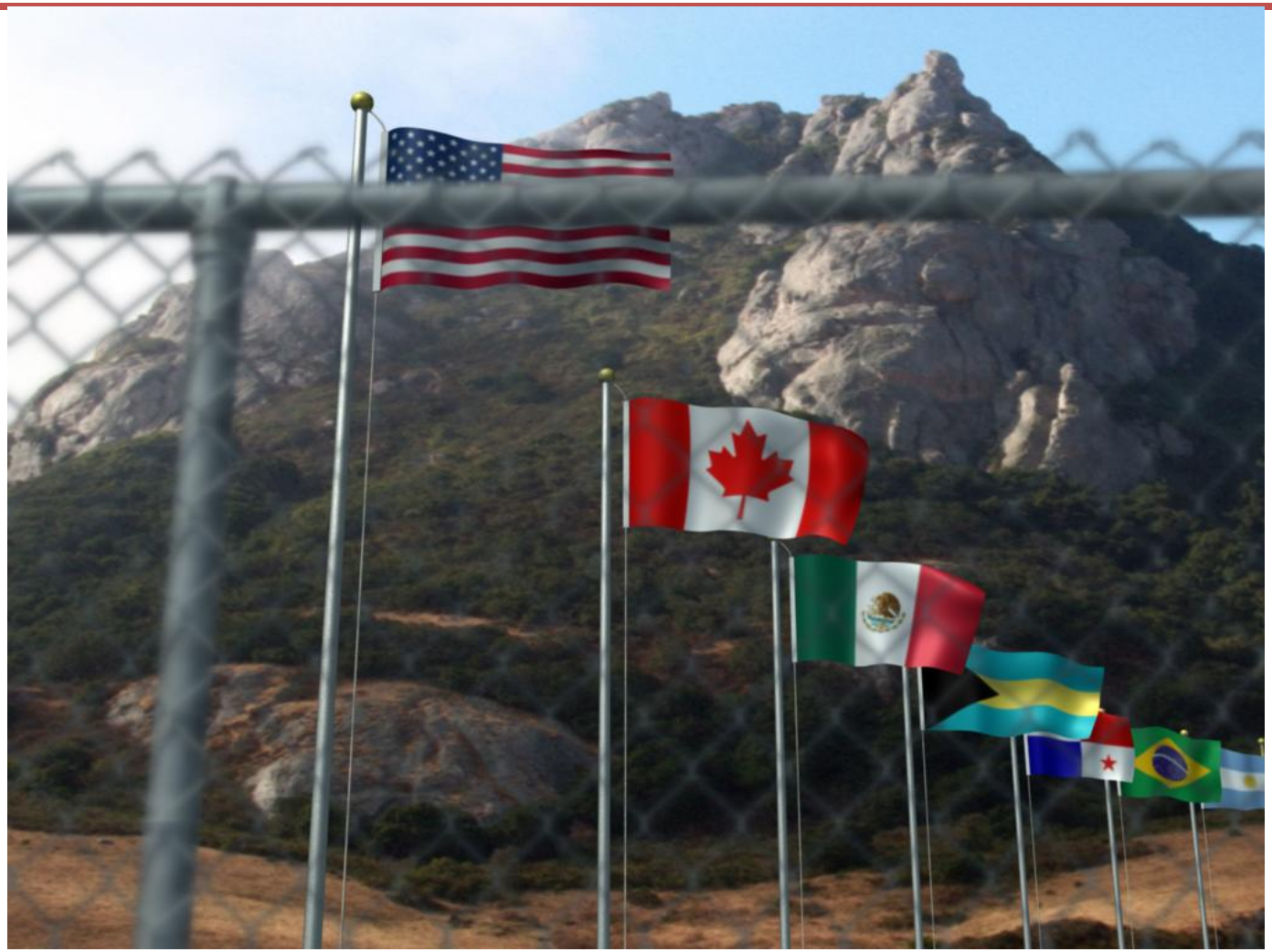
Interactive Depth of Field



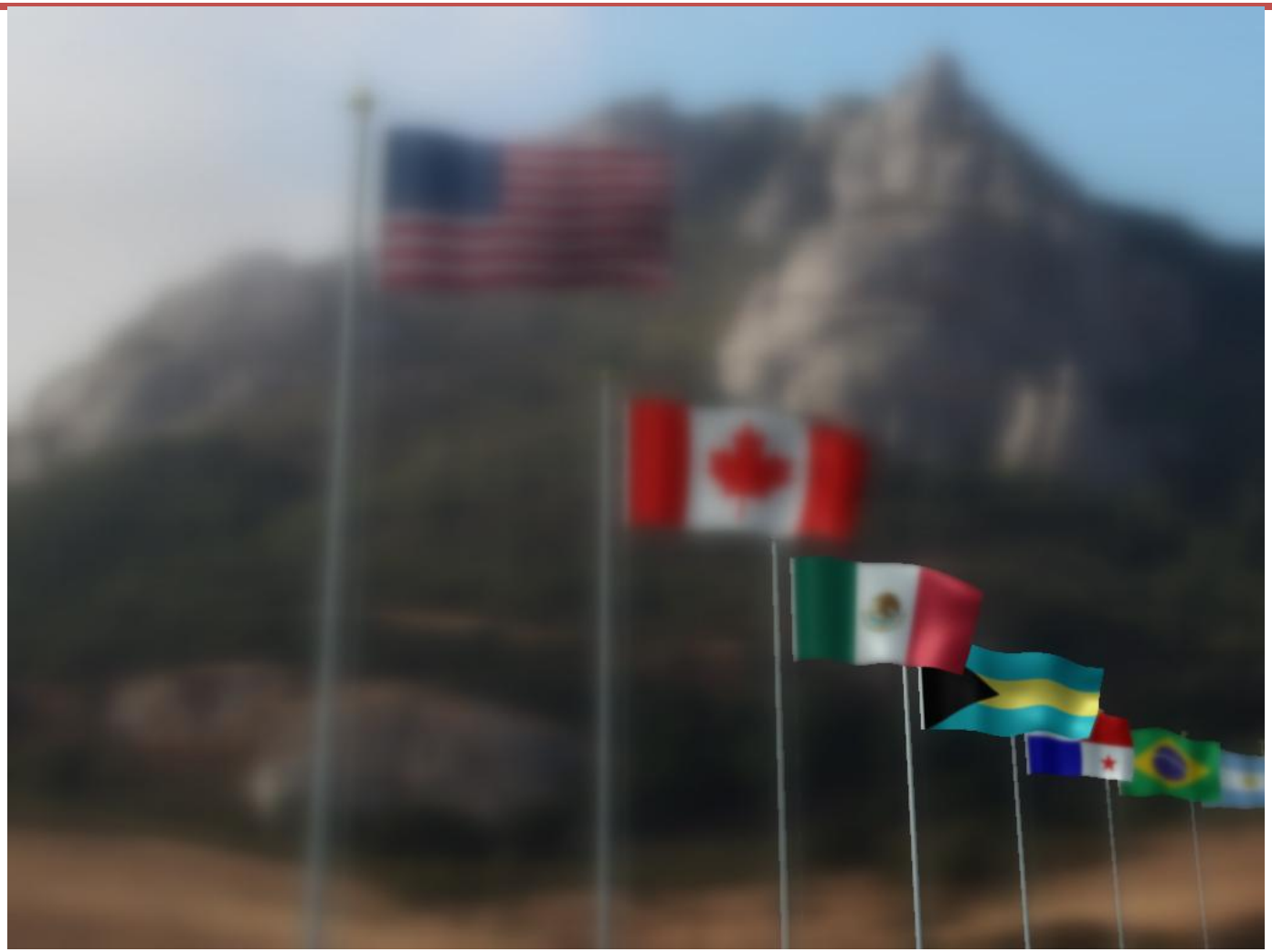
Interactive Depth of Field



Interactive Depth of Field



Interactive Depth of Field



Results

- Separable simulated diffusion

Pictures from DICE research

DICE Research

- Separable simulated diffusion

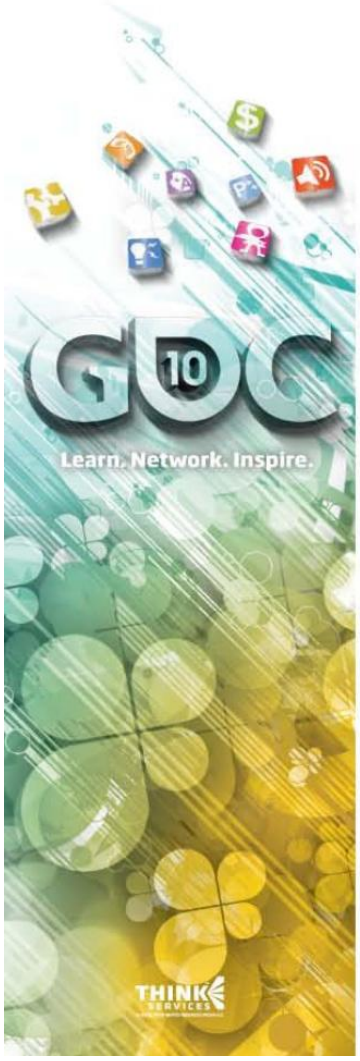
Pictures from DICE research

Results

- Lens parameter settings

Pictures from DICE research

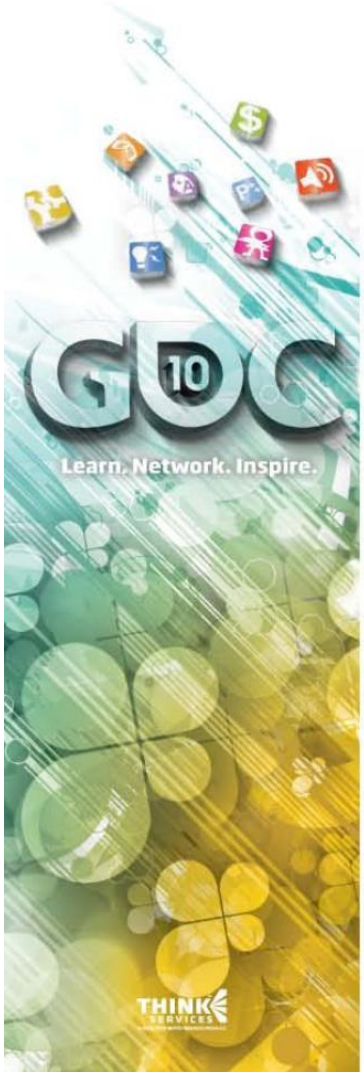
Diffusion DOF in Metro



From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

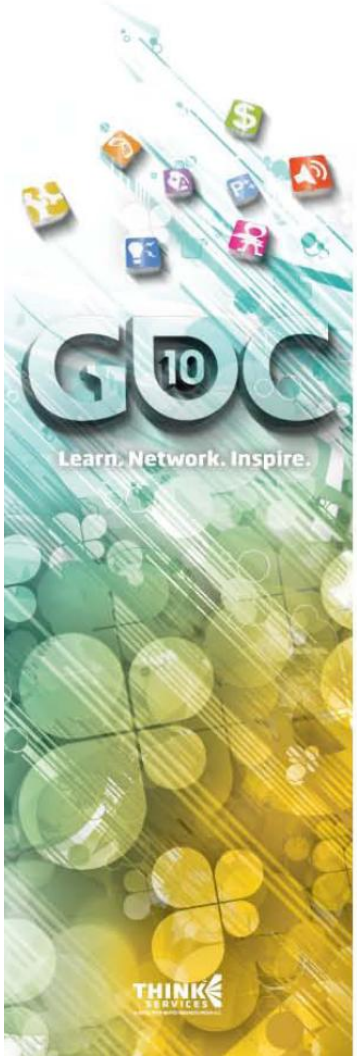
Diffusion DOF in Metro



From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

Benefits



- Clear separation of sharp in-focus and blurred out-of-focus objects

From *Metro 2033*, © THQ and 4A Games

From "DX11 Effects in Metro 2033, GDC 2010"

Conclusions

- Post-process depth-of-field is hard to do well
  - Spread-based method looks promising
  - Still unsolved problems in post-process DOF
    - Will splatting from multiple depth layers become fast enough that the hacks can just go away?
    - Worth trying a brute force solution again in DX11?
  - Will post-process DOF eventually go away when we get real-time renderers that can do it right and affordably?

- When to use a pixel shader vs ComputeShader?
  - Shared memory and synchronization
  - If no shared memory usage, pixel shader will give better performance