# Introduction to Data-Stream-Based Processing on GPUs

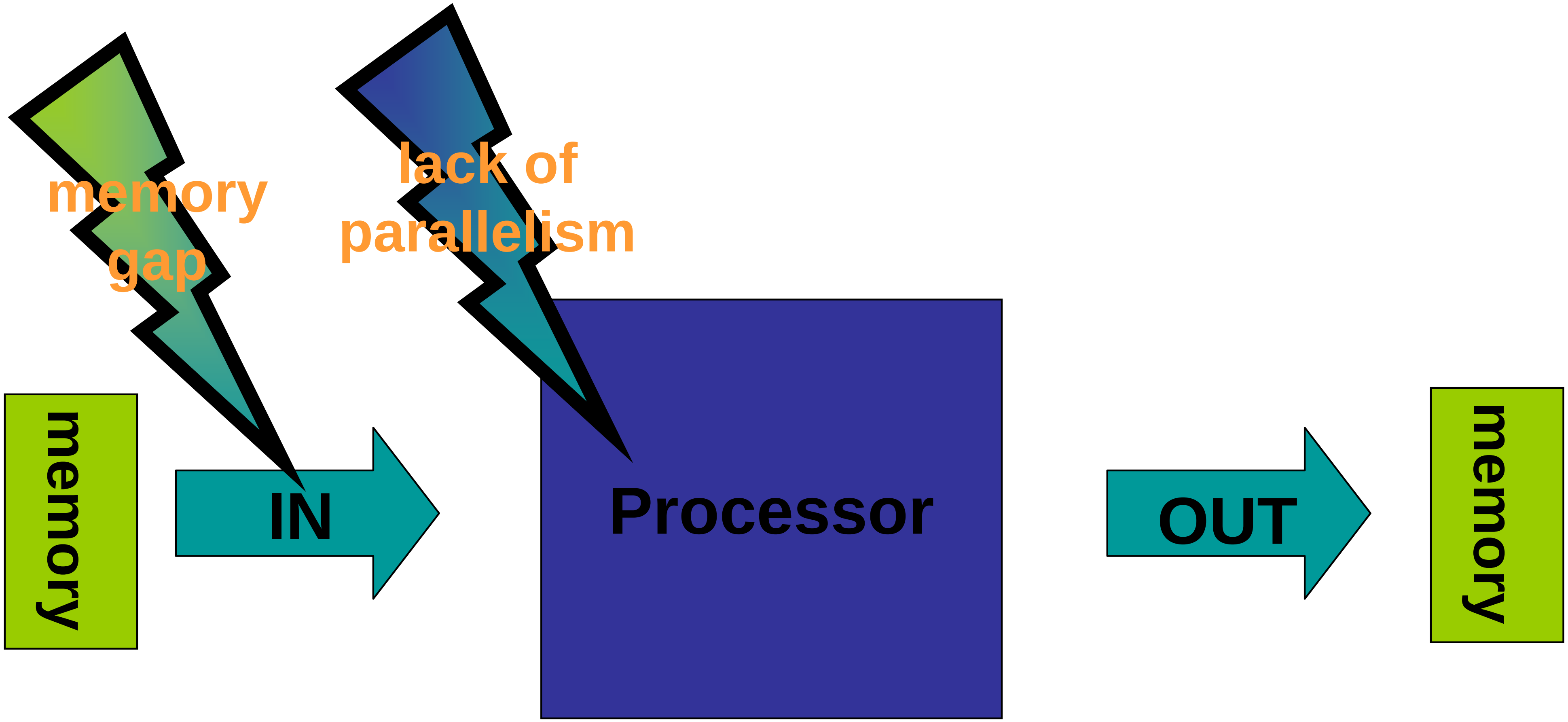
Robert Strzodka

caesar research center
Bonn, Germany

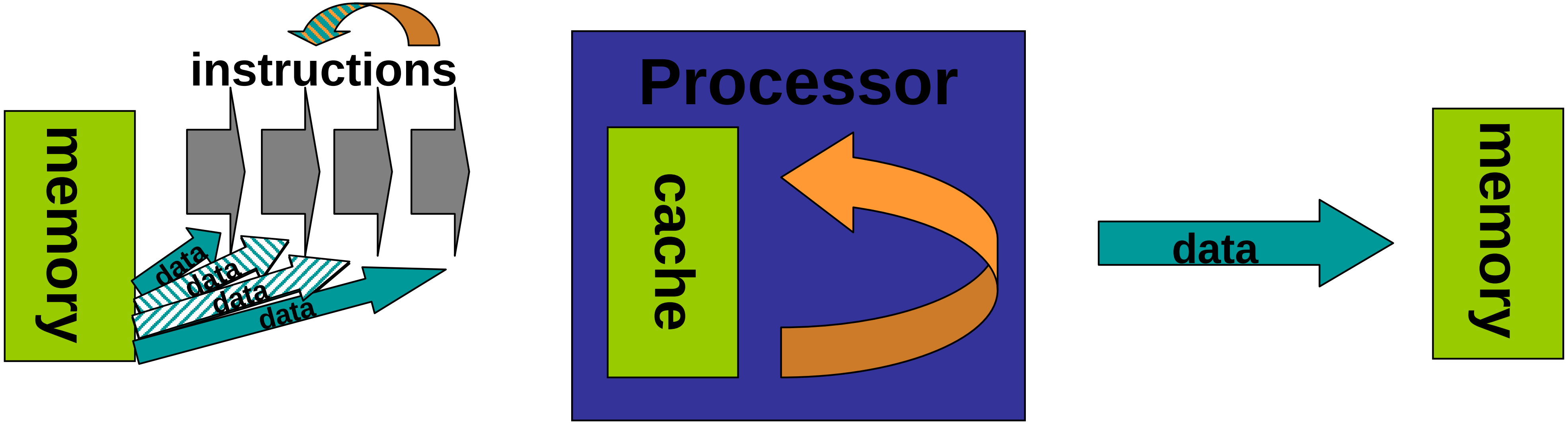
# Overview

- Graphics Processor Unit (GPU)
  - Data-Stream-Based Processing
  - Functionality of GPUs
  - “Hello World” on GPUs
- Scientific Computing
  - Partial Differential Equations
  - Gather and Scatter Operations
  - Matrix Vector Product
- Further GPU - PDE Topics
  - Quantization
  - Discretization Grids
  - Discretization Schemes

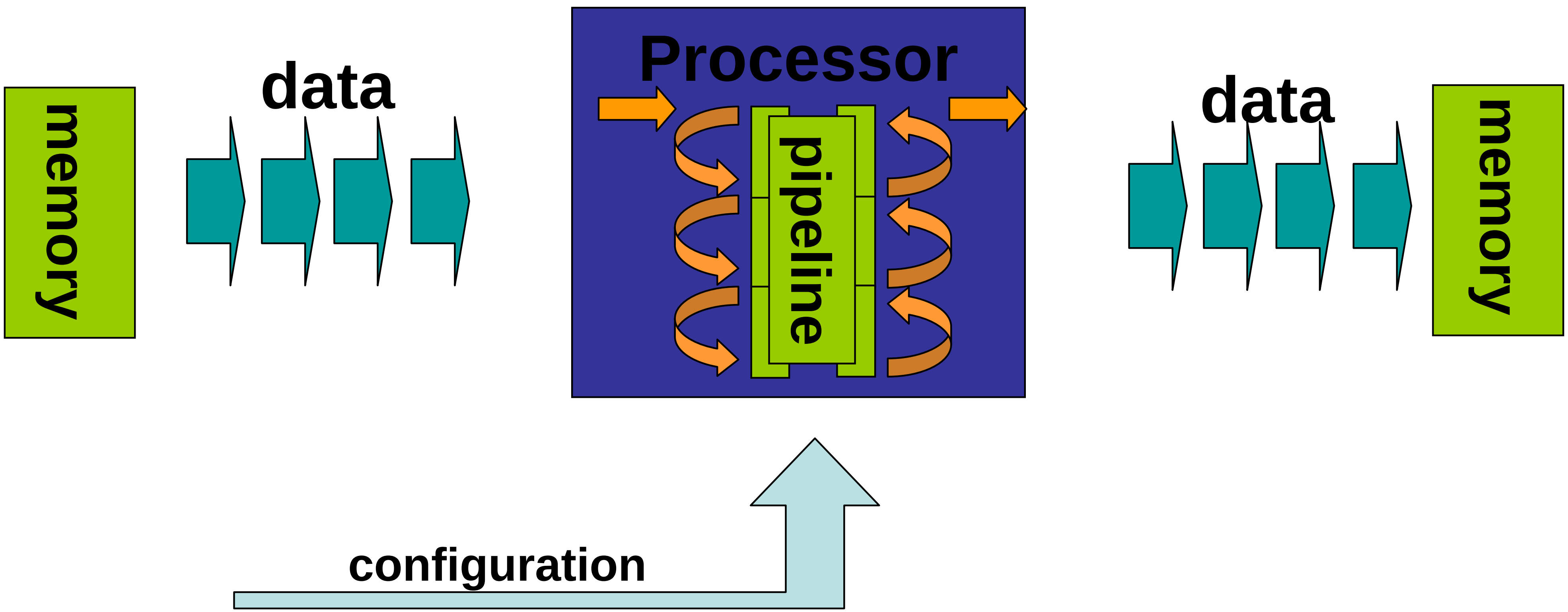
# Data Processing in General



# Instruction-Stream-Based Processing



# Data-Stream-Based Processing



# Instruction- and Data-Streams

Addition of nodal 2d grid vectors: **C= A + B**

| | |
|---|---|
| **instuction stream** processing data | `for(y=0; y<HEIGHT; y++)`<br>`for(x=0; x<WIDTH; x++) {`<br>`  C[y][x]= A[y][x]+B[y][x];`<br>`}` |
| **data streams** undergoing a gather operation | `inputStreams(A,B);`<br>`outputStream(C);`<br>`kernelProgram(OP_ADD);`<br>`processStreams();` |

# Data-Stream-Based Architectures

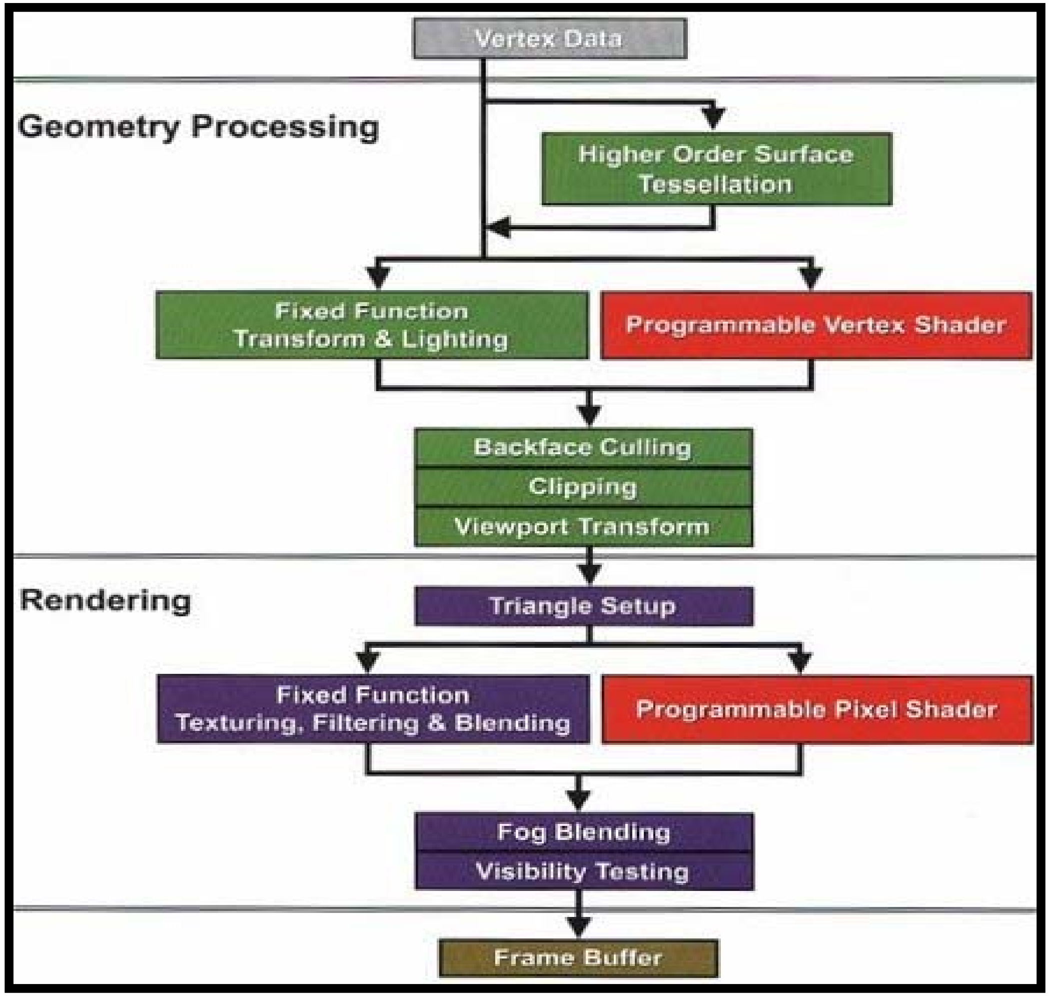
- RL – Reconfigurable Logic, e.g. FPGAs
  - very flexible bit level parallelism
  - high transistor and design costs for total reconfigurability
- RC – Reconfigurable Computing, e.g. XPP (PACT)
  - flexible word (4-32 bit) level parallelism
  - various architectures with different pros and cons
- PIM – Processor-in-Memory, e.g. FlexRAM (Illinois)
  - extreme data parallelism
  - restricted inter-chip communication
- SP – Stream Processors, e.g. Imagine (Stanford)
  - intensive data reusage in hierarchical stream caches
  - performs best for high computational intensity

# Graphics Processor

The graphics pipeline consists of:

• **Parameter** controlled processing units

• Processing units **programmable** with high level languages



# Fragment Processor Functionality as seen from a High Level Language

- Float data types:
  - 16-bit & 32-bit (NVIDIA), 24-bit (ATI)
- Vectors, structs and arrays:
  - float4, float vec[6] , float3x4, float arr[5][3], struct {}
- Arithmetic and logic operators:
  - +, -, *, /; &&, ||, !
- Trignonometric, exponential functions:
  - sin, asin, exp, log, pow, …
- User defined functions
  - max3(float a, float b, float c) { return max(a,max(b,c)); }
- Conditional statements by predication, unrollable loops:
  - if, for, while, dynamic branching in PS3
- Arbitrary texture positions can be accessed

# “Hello World“ GPGPU Example

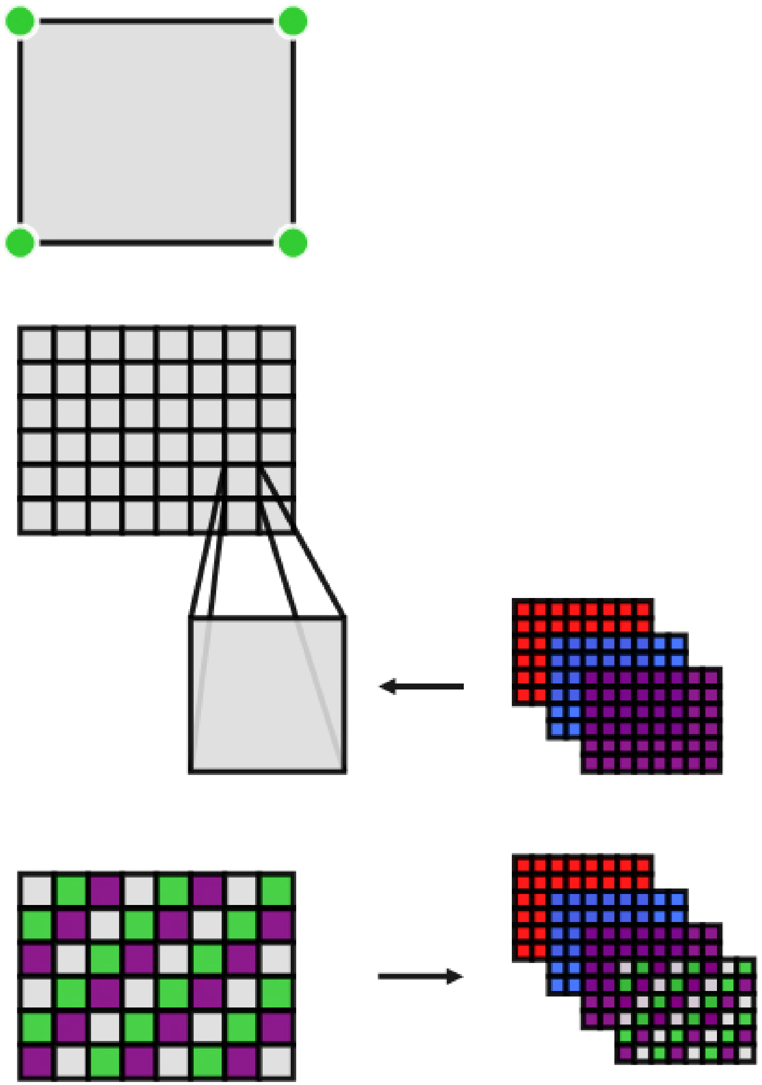
- 3 x 3 Image processing convolution
- CPU version

```
image = loadImage( WIDTH, HEIGHT );
blurImage = allocZeros( WIDTH, HEIGHT );

for (x=0; x < WIDTH; x++)
  for (y=0; y < HEIGHT; y++)
    for (i=-1; i <= 1; i++)
      for (j=-1; j <= 1; j++)
        float w = computeWeight(i,j);
         blurImage[x][y] += w * image[x+i, y+j];
```



slide courtesy of Aaron Lefohn

# Computing by Drawing



- Drawing a large quad replaces the outer x,y loops
- The loop body is executed in parallel for the different x,y indices (fragments)
- The loop body reads data from textures (images)
- The result can be used in as input in next operation



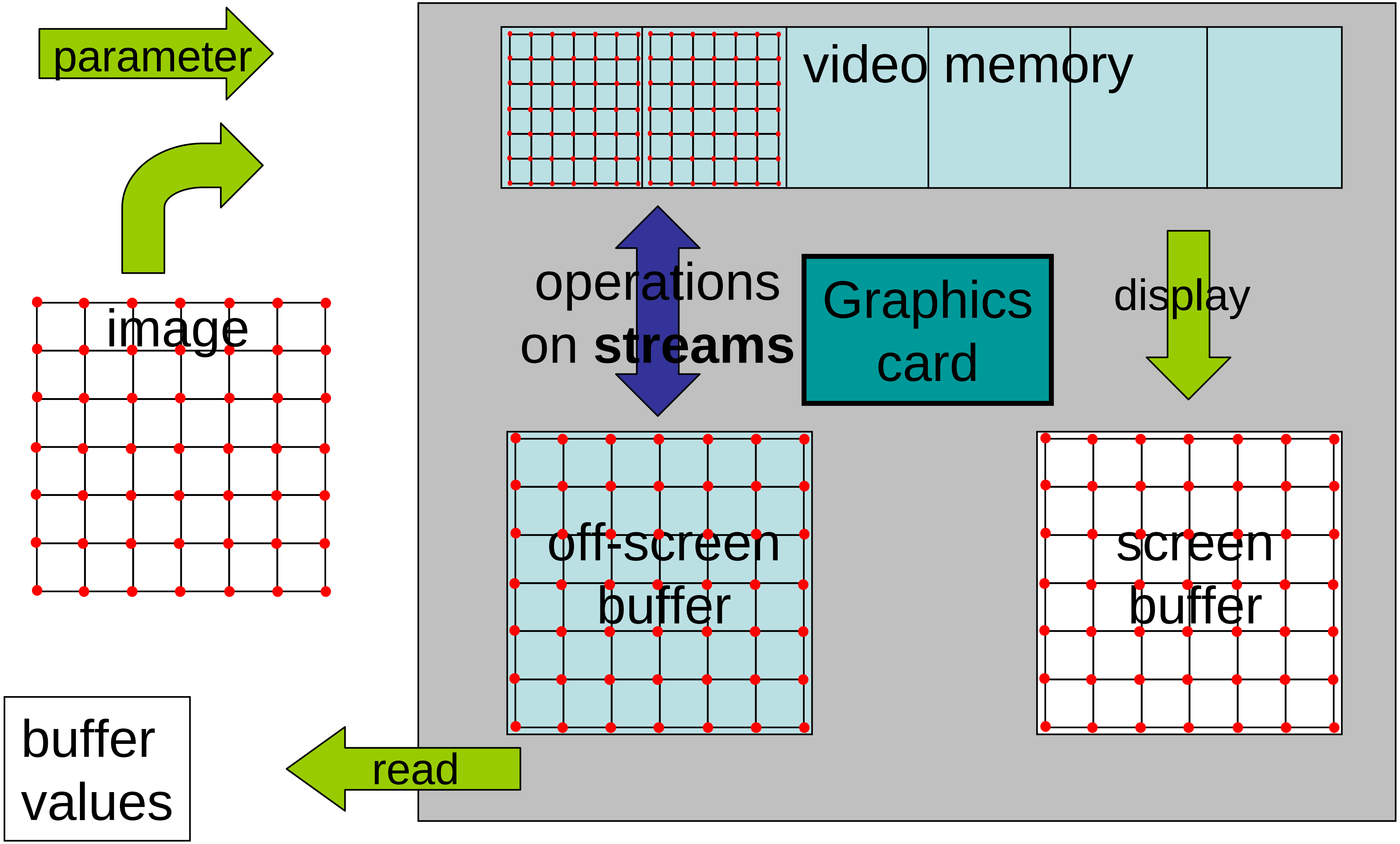
image courtesy of John Owens

# “Hello World“ GPGPU Example

- Fragment program for the loop body in Cg
- GPU Version

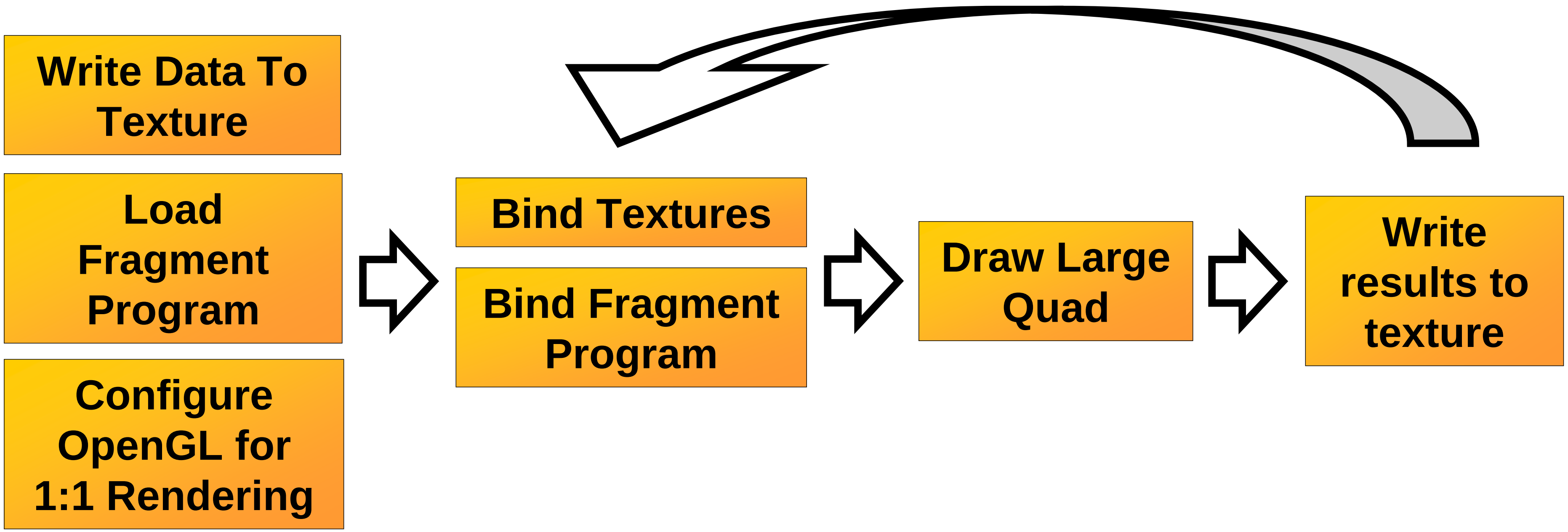
```
float4 blurKernel( uniform samplerRECT image,
                   float2      winPos : WPOS,
                   out float4  blurImage )
{
   blurImage = float4(0,0,0,0);

   for (i=-1; i <= 1; i++) {
     for (j=-1; j <= 1; j++) {
       float2 texCoord = winPos + float2(i,j);
       float  w        = computeWeight(i,j);
       blurImage += w * texRECT( image, texCoord );
     }
   }
}
```

# Data Streaming in Graphics Hardware



# GPU as a Data-Parallel Computer

- Data specification → Textures
- Kernel specification → Fragment program
- General execution → Draw single large quad



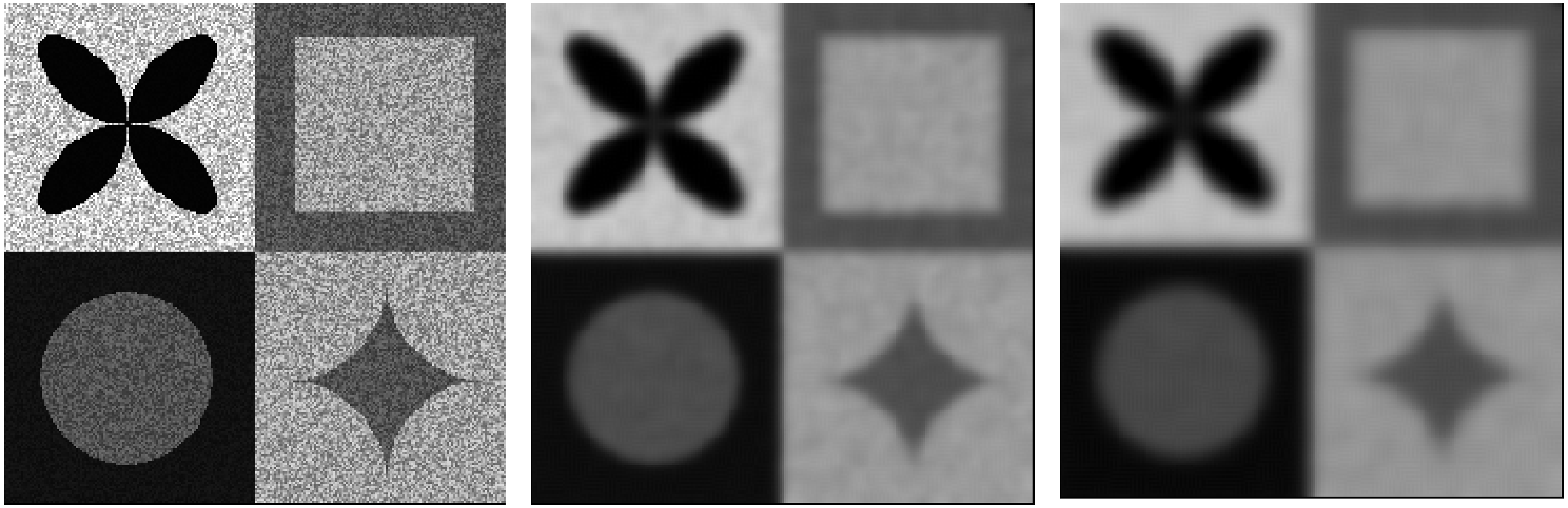
# Overview

- Graphics Processor Unit (GPU)
  - Data-Stream-Based Processing
  - Functionality of GPUs
  - “Hello World” on GPUs

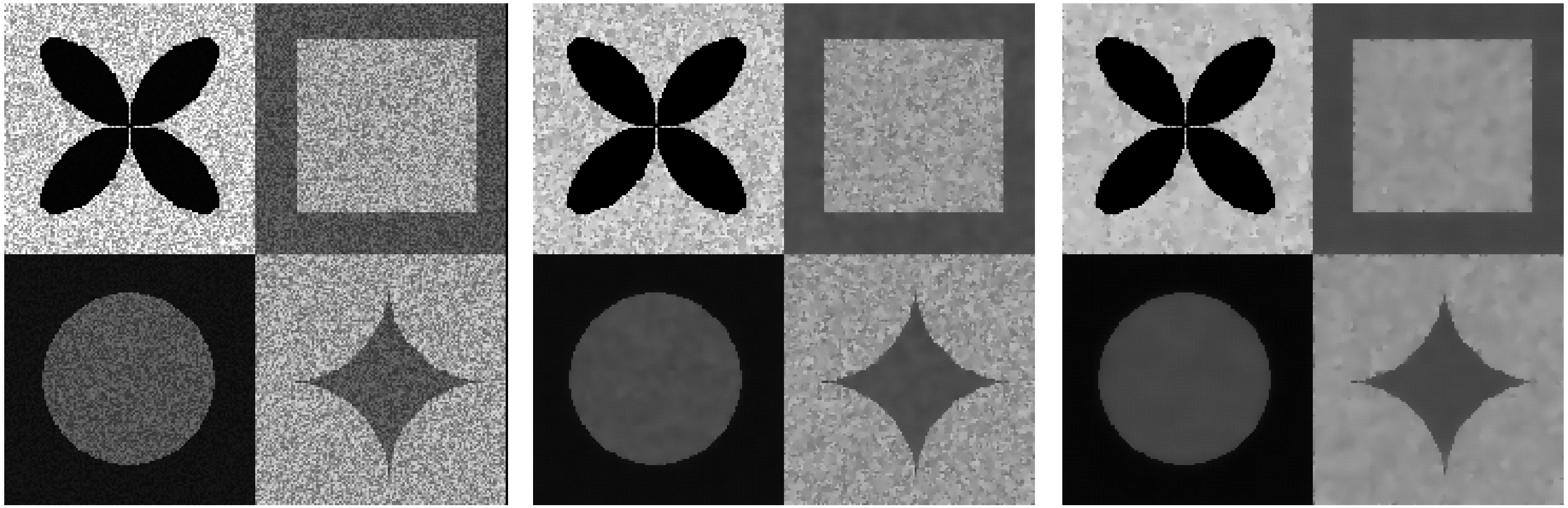
- Scientific Computing
  - Partial Differential Equations
  - Gather and Scatter Operations
  - Matrix Vector Product

- Further GPU - PDE Topics
  - Quantization
  - Discretization Grids
  - Discretization Schemes

# Denoising by a Linear and a Non-linear Diffusion



linear diffusion



non-linear diffusion

# Solving PDEs on GPUs

We seek a function $u(x,t):(\Omega,\Re^{+})\rightarrow\Re^{m},\Omega\subseteq\Re^{d}$ which satisfies

PDE $\partial_t u + F[u,u] = 0$ in $\Re^{+}\times\Omega$

initial value $u(0) = u_0$ in $\Omega$

boundary $\partial_\nu u = b_N$ or $u = b_D$ on $\Re^{+}\times\partial\Omega$

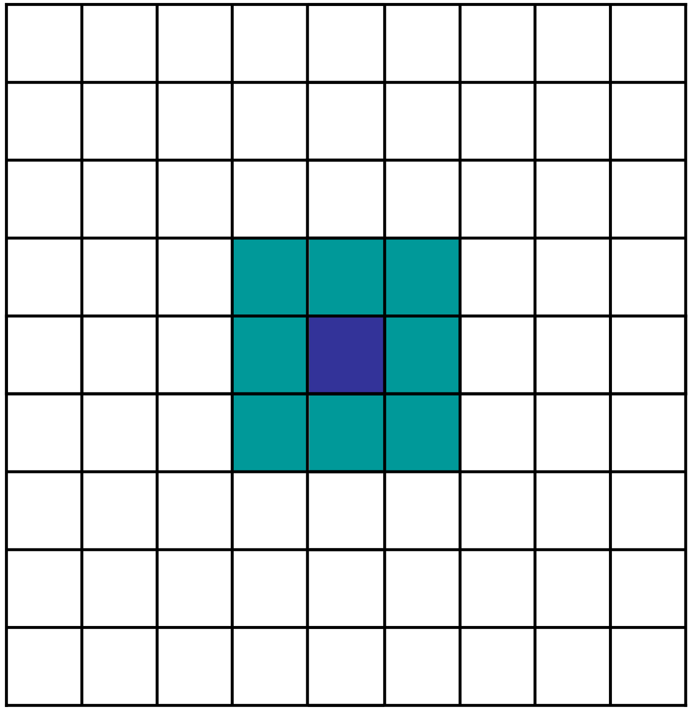
e.g.: non-linear diffusion: $F[u,v] = -\mathrm{div}\left(g(\|\nabla_\sigma u\|)\nabla v\right)$

After discretization in time and space and possibly the use of an iterative linear equation system solver, the main opration required by the algorithm matrix-vector products $A\cdot\overline{V}^{n}$

# Local Gather Operation

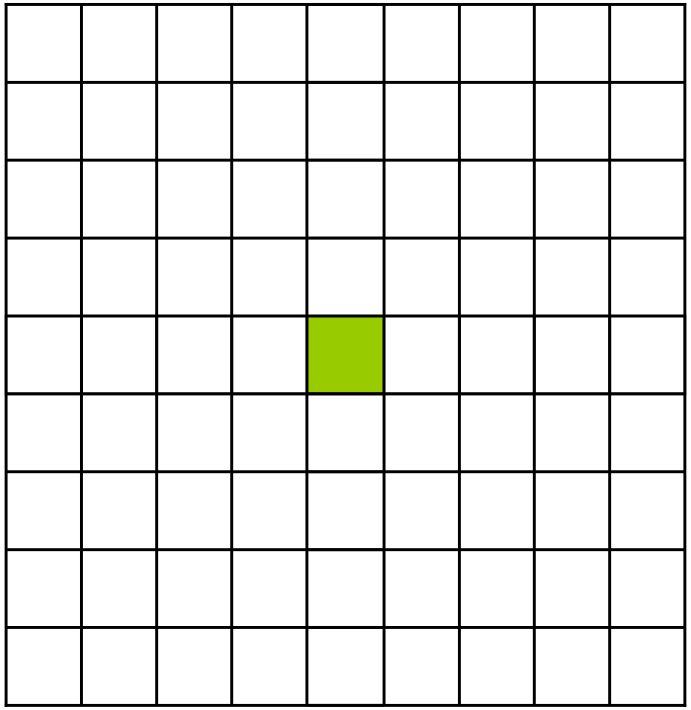
Step n

Step n+1



$$F_h\left(\left(\overline{V}_\beta^{\,n}\right)_{|\beta-\alpha|\le C}\right)$$

$$\sum_{\beta:|\beta-\alpha|\le C} A_{\alpha,\beta}\overline{V}_\beta^{\,n}$$

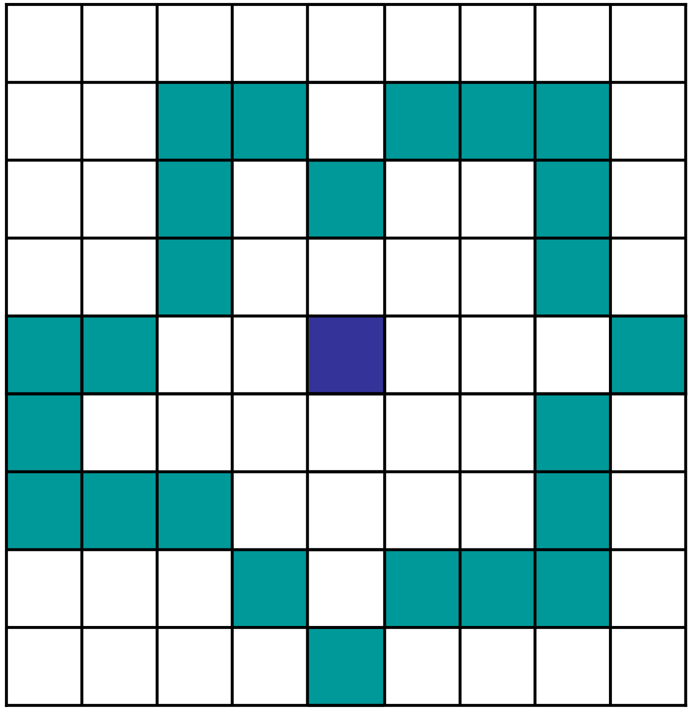


$$\left(\overline{V}_\beta^{\,n}\right)_{|\beta-\alpha|\le C} \quad \overline{V}_\alpha^{\,n}$$

$$\overline{V}_\alpha^{\,n+1}$$

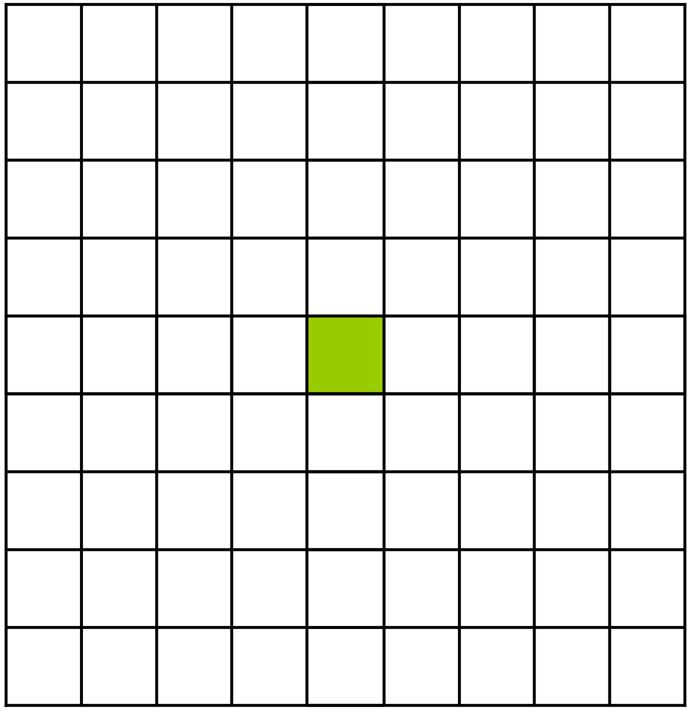
Robert Strzodka
caesar research center

# Global Gather Operation



$$F_h\left(\left(\overline{V}_\beta^{\,k}\right)_{\beta\in\gamma^k}\right)$$

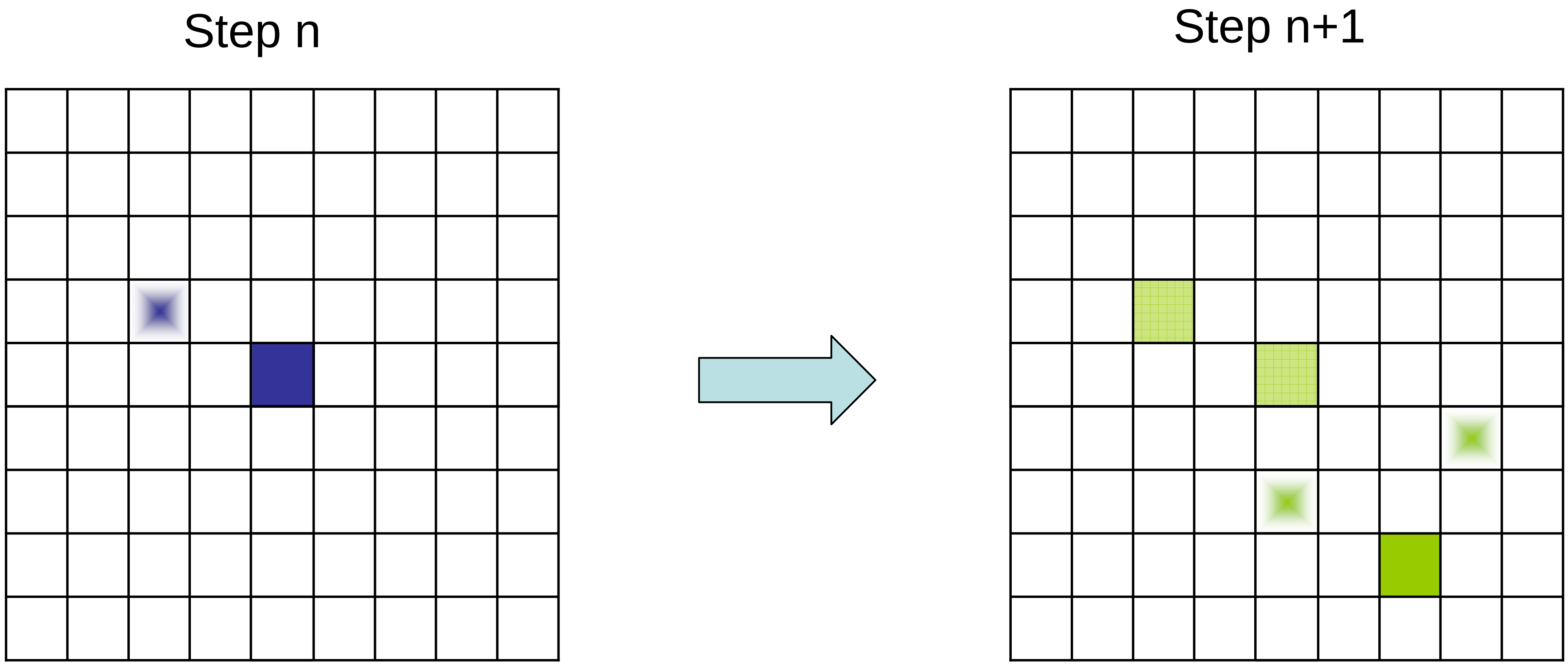
$$\sum_{\beta\in\gamma^k} A_{\alpha,\beta}\overline{V}_\beta^{\,k}$$



$$\left(\overline{V}_\beta^{\,k}\right)_{\beta\in\gamma^k}$$

$$\overline{V}_\alpha^{\,k}$$

# No Parallel Dynamic Scatter




In the loop body the index of the updated node cannot be changed dynamically. Slow remedy: scattering of points.

# Overview

- Graphics Processor Unit (GPU)
  - Data-Stream-Based Processing
  - Functionality of GPUs
  - “Hello World” on GPUs

- Scientific Computing
  - Partial Differential Equations
  - Gather and Scatter Operations
  - Matrix Vector Product

- Further GPU - PDE Topics
  - Quantization
  - Discretization Grids
  - Discretization Schemes

# Quantization

Roundoff examples for the **float s23e8** format

| | | |
|---|---|---|
| additive roundoff | $a = 1 + 0.00000004$ | $=_{fl} 1$ |
| multiplicative roundoff | $b = 1.0002 * 0.9998$ | $=_{fl} 1$ |
| cancellation $c=a,b$ | $(c-1) * 10^8$ | $=_{fl} 0$ |

Cancellation promotes the small error 0.00000004
to the absolute error 4 and an order one relative error.

Order of operations can be crucial:

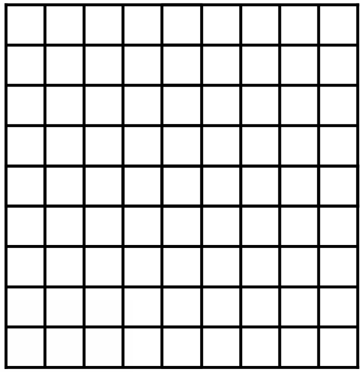
$$1 + 0.00000004 - 1 =_{fl} 0$$

$$1 - 1 + 0.00000004 =_{fl} 0.00000004$$

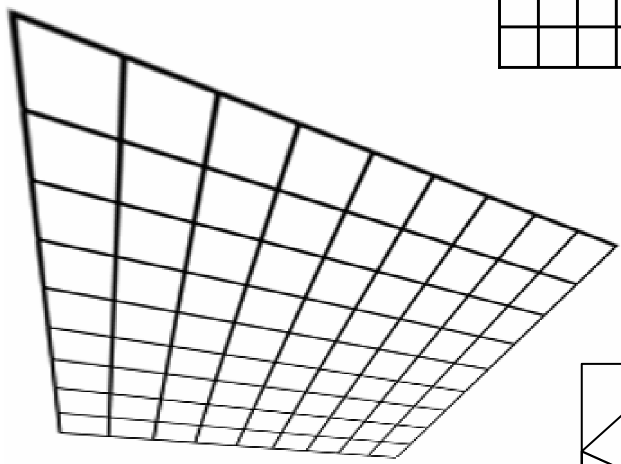
But cancellation cannot be avoided automatically!

# Discretization Grids on GPUs

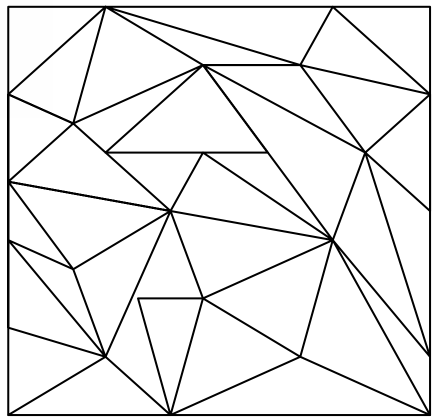
- An equidistant grid
  - Easy to implement
  - One texture holds all values



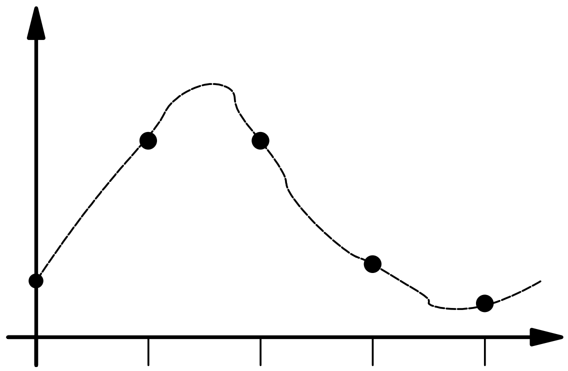
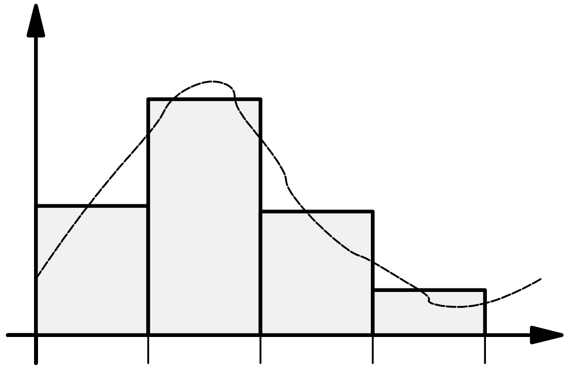
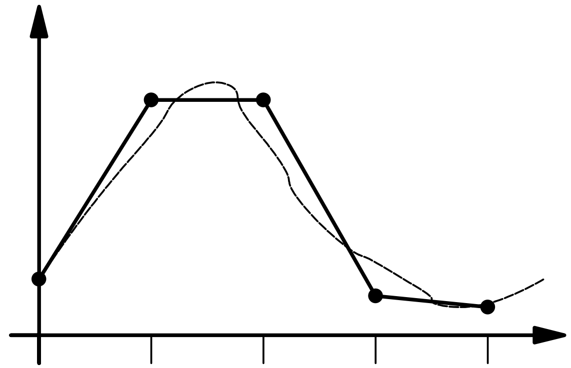
- Deformed tensor grid
  - Parallel dynamic updates
  - One texture for values second for deformation



- Unstructured grid
  - Good performance for static, poor for dynamic grid topology
  - Several indirections are needed



# Discretization Schemes

- Finite Differences
  
  - Interpolative approach: simple and fast
  - Usually interaction with direct neighbors
- Finite Volumes
  
  - Volumetric approach: mass conservation
  - Good at discontinuities, less for smooth data
  - Interaction over element boundaries
- Finite Elements
  
  - Approximative approach: error minimization
  - Good handling of deformed, unstructured grids
  - Interaction of basis functions (all neighbors)

# Conclusions

- Many problems expose a lot of data parallelism.
- GPUs perform well as inexpensive parallel devices for this kind of processing.
- Floating point number support and programming with high level languages facilitates access to this functionality.
- Future GPUs will be even more flexible and powerful but focus on data-stream-based processing will remain.

# Interested in GPU Programming?

- **GPGPU** = General Purpose Computations on GPUs
- Visit the GPGPU base: papers, code, news, links, people
  - **www.gpgpu.org**
- Site contains extensive material of two full-day tutorials
  - SIGGRAPH 2004
  - Visualization 2004
- Code examples
  - Commented ‘Hello GPGPU’
  - Contributed applications
  - Utility libraries