GPU-multicore programming

Tor Dokken
SINTEF IKT
tor.dokken@sintef.no

www.sintef.no/math
www.sintef.no/gpgpu
www.sintef.no/Parallel3D

Performance and hardware evolution of standard hardware

- **1980** - Mini-computers - 1980
  - Digitals VAX 780, (1977), 0.5 MIPS
  - Floating point operations taking multiple clock-cycles
- **1990** - Moving to UNIX-workstations
  - HP 734, (1989) 14 MIPS
- **2000 -** moving to PCs running Windows or Linux
  - Pentium III (1999), 1354 MIPS at 500 MHz
  - Pentium 4 Extreme Edition (2003), 9 726 MIPS at 3.2 GHz
  - Multiple floating point operations in a clock-cycle

- Due to the hardware evolution of standard hardware the performance of sequential programs has increased by at least four orders of magnitude in 3 decades

$$2^{13} = 8192 \text{ and } 16384,\ 2^{14} = 16384$$

Moore’s law (1965)

- *“The number of transistors on an integrated circuit for minimum component cost doubles every 24 months*”
  - Seems to hold true
  - Transistor density roughly proportional to processor performance

- Growth it core clock frequency stagnated
  - Until recently (2004) increases in processor performance relied heavily on the increase of core clock frequency

- Stagnation of growth in performance of sequential code
  - Until now industry has experienced increasing value from their existing code base, with relatively little effort.
  - The *Beach law** now obsolete – your sequential code will probably not run faster next year

**Until recently one way of doubling the performance of your code was to wait 1 ½ years (go to the beach) and then buy a new computer.*

Growth in core frequency has stagnated

- **Memory speeds**
  - Memory speeds have not increased as fast as core frequencies.
  - This decreases the value of increasing the core frequency.
- **Longer instruction pipelines**
  - Longer instruction pipelines allow for higher core frequencies.
  - However, long pipelines increase the cache-miss penalty and lower the number of completed instructions per cycle.
- **Power consumption**
  - When the core frequency increases, the power consumption increases disproportionately.
  - The dependence between core frequency and the power consumption is often said to be quadratic but can for some cases be almost cubic.

Integrate more cores in the processor.

- Can take advantage of the continuous increase in transistor density. (Moore’s law)
- Increase performance with no increase in power consumption
  - Doubling the number of cores and halving the frequency delivers roughly the same performance, while the power consumption can be reduced by a factor of four (cubic case).
  - Hence, by introducing additional cores it is possible to get higher performance without increasing the power consumption.
- Demand for parallelization of software
  - Unless your software is able to utilize multiple cores, you may actually observe that your new multi-core PC runs slower than your high GHz computer.

From multi-core to heterogeneous systems – to further increase performance

- How to use the transistors
  - A majority of the transistors on a traditional CPU core is not used for arithmetic, but for things like flow control and cache.
  - To achieve maximal floating point performance on a given transistor and power consumption budget, it is not necessary for every core to be general-purpose.
  - Instead one can design smaller and simpler computational units, which can very effectively perform floating-point operations on a large set of data.
- These units need to be controlled and fed data by a CPU.
  - These computational units are complementing and not replacing traditional CPUs.
- We denote these simpler units *data stream accelerators*

The two most important present-day variants of *data stream accelerators*

- GPU (Graphics Processing Unit)
  - The major GPU vendors are now selling the same architecture for gaming and for high-performance computing.
  - NVIDIA Tesla computing processor, 2 GiByte memory and offers more than 500 Gflops (singel precision).
  - AMD FireStream 9170, 2 GiByte memory, double precision and offers more than 500 Gflops (single precision)(320 cores, clock rate 775-800 MHz). (Announced November 2007)
- Cell BE (Cell Broadband Engine Architecture): The Cell is an architecture jointly developed by IBM, Sony and Toshiba.
  - It is used in Sony's Playstation 3, as stand-alone blade servers, and as add-in cards for use in desktop computers.
  - Current systems provide roughly 200 Gflops (single precision)

Parallelization will be a must

- Most computer programs existing today are programmed according to a sequential paradigm.
  - Education dominantly addresses sequential programming
  - Traditional use of parallelization has been within HPC - High Performance Computing.
- Until a few years ago, commodity computers were almost solely based on single-core CPUs, and the steady growth in CPU frequency automatically increased the performance of sequential programs.
- In the last years, this growth has slowed down, and the trend is for computers to increase the number of cores at the expense of clock frequency.
  - Hence, the performance of sequential programs does not increase as fast as before and parallelization is necessary to effectively utilize new CPUs.

Consequences for algorithm use

- The performance of algorithms programmed according to a sequential paradigm will not benefit from the current growth in parallel computational resources.
- To exploit the computational power of multi-core CPUs and data stream accelerators we have to:
  - Understand the specificities of *data stream accelerators*
    - They are not optimal for all types of algorithms.
  - Rethink our algorithms to fit heterogeneous resources
    - Replace complex logics in programs by simplistic computational demanding algorithms
    - Replace single pass algorithms by multi-pass algorithms
  - Look for challenges that until now have been regarded as too resource consuming

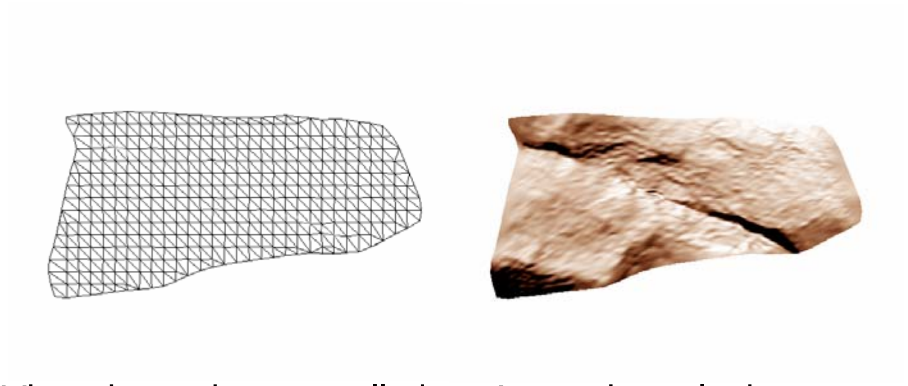
Programming a challenge

- The variation of performance and balance between different computational resources will be extensive on PCs
  - CPUs with different number of cores
  - Wide variation of graphics cards
- Programming tools in its infancy. To develop efficient programs the developers have to take explicit control over:
  - algorithmic partitioning
  - memory hierarchies
  - processor inter-communication.
- This contradicts the dominant trend in the last decade:
  - where object-oriented programming techniques have been used to abstract away detailed knowledge of the underlying hardware
- EU has addressed this challenge for embedded systems in the ICT-program of fp7 Call 1 April 2007.

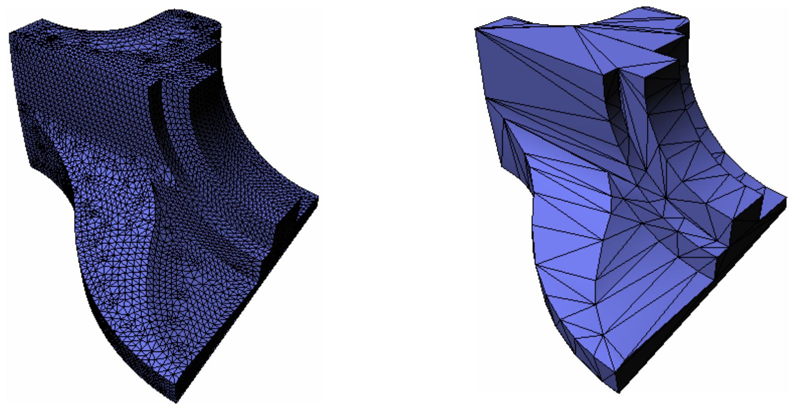
How have this been addressed in SINTEF

- *GPGPU - Graphics hardware as a high-end computational resource.* Project funded by the Research Council of Norway (RCN) (2004-2007)
  - Geometric modeling, Partial Differential, Image Processing, Linear algebra Visualization, 2 PhDs, 1 Post Doc, master students >15
- *Parallel3D - Massive multi-type 3D datasets in parallel Environments.* Project funded by RCN (80%) and Kongsberg SIM (20%) (2007-2010)
  - Scene graphs, Viewers, Multi-resolution representation, 2 PhDs
- "*Heterogeneous Computing*" project application to RCN (2008-2010)
  - Generic heterogeneous computing, Partial Differential Equations, Geometry Processing, 1 PhD
- The "*Heterogeneous Computing*" group in SINTEF ICT
  - 5 research scientists
  - 5 PhD – fellows
  - Many master students
- Patenting

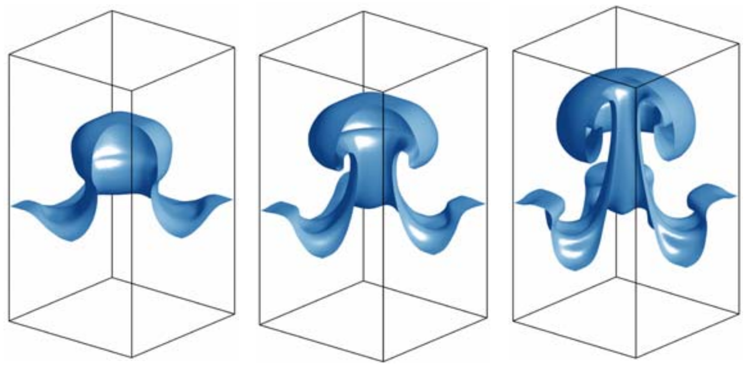
GPGPU Activities in SINTEF Applied Mathematics



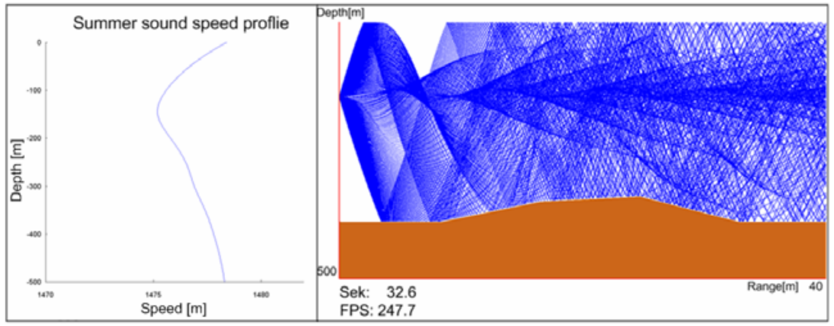
View-dependent tessellation: Approximately the same size triangles when projected on the screen



Preparation of finite element models (cooperation with INPG, France)

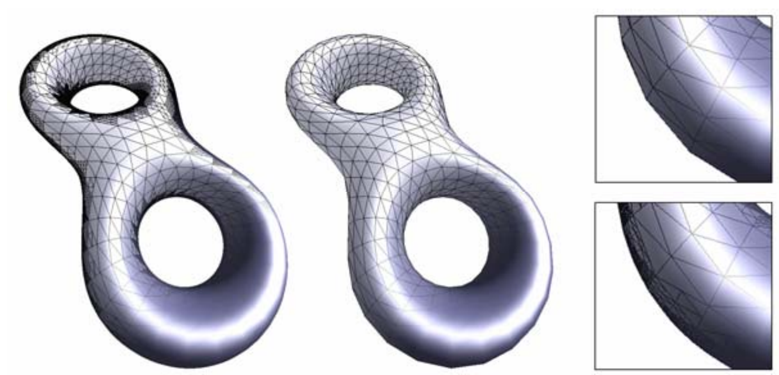


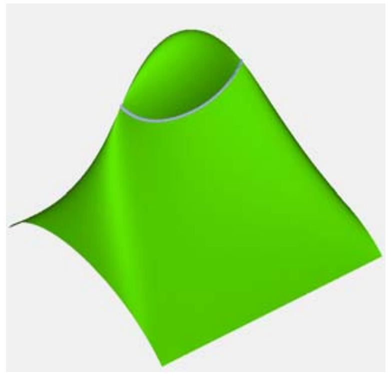
Solving partial differential equations

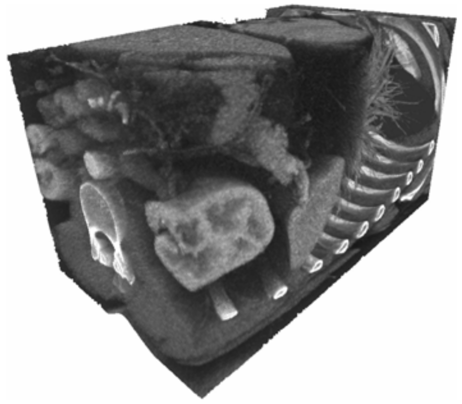


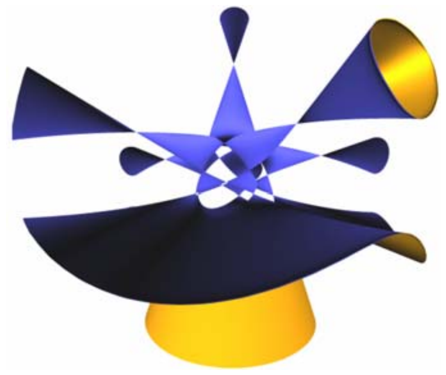
Marine acoustics (cooperation with the Norwegian University of Science and Technology

GPU Activities at SINTEF Applied Mathematics

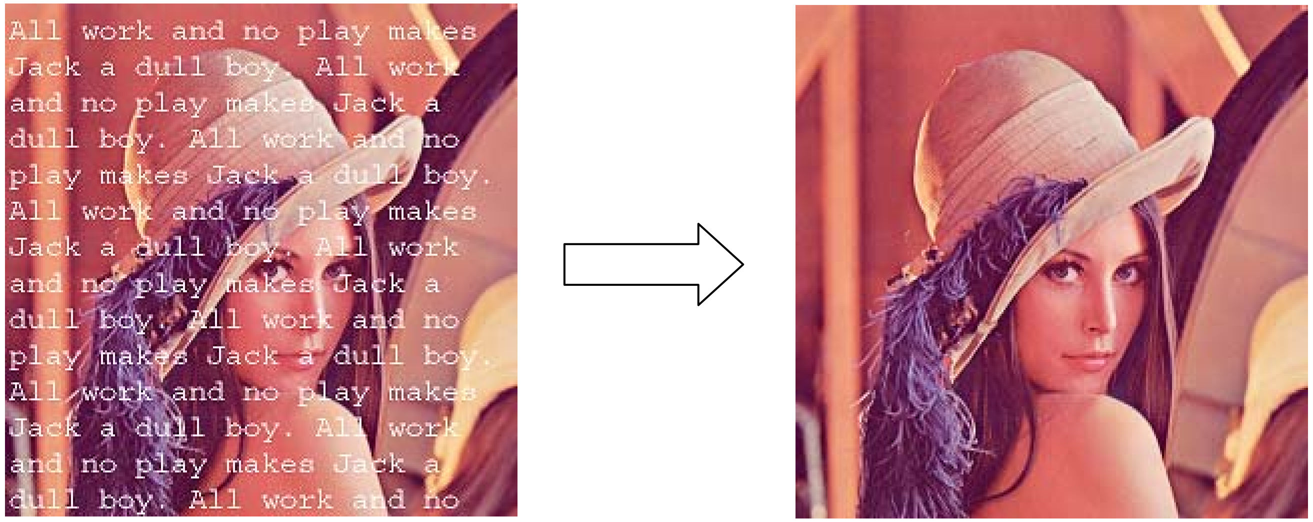

Silhouette refinement
(Cooperation with CMA, University of Oslo)


Intersection acceleration

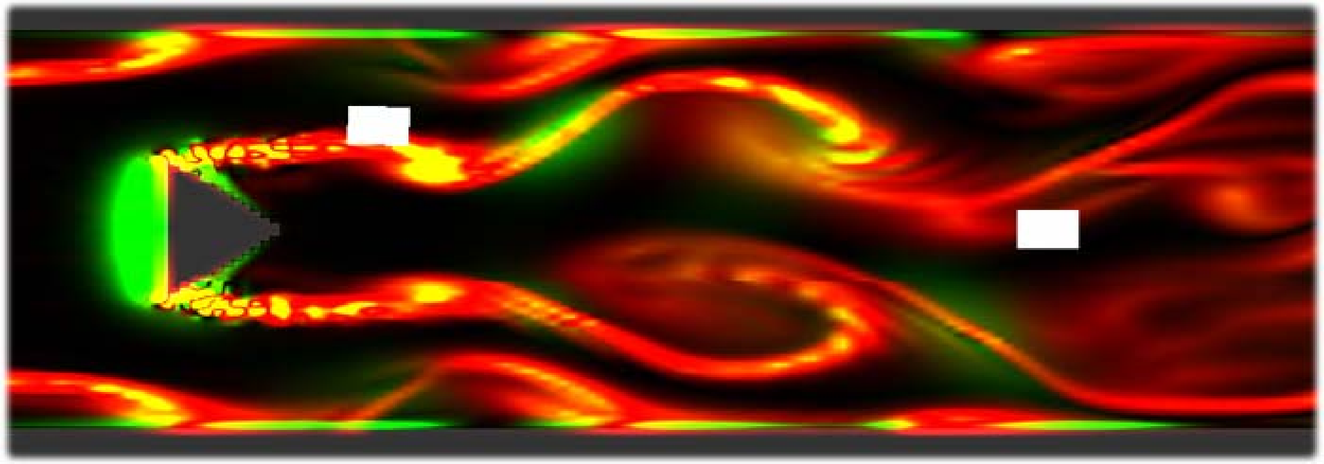

Registration of medical data
(Cooperation with SINTEF Health Research)


Visualization of algebraic surfaces
(Cooperation with CMA, University of Oslo)

GPU Activities at SINTEF Applied Mathematics

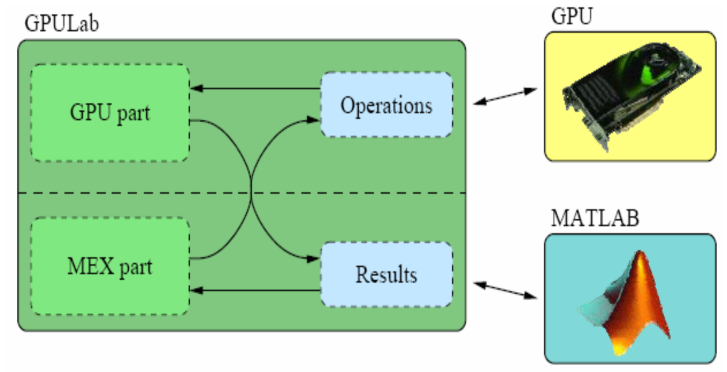

All work and no play makes
Jack a dull boy. All work
and no play makes Jack a
dull boy. All work and no
play makes Jack a dull boy.
All work and no play makes
Jack a dull boy. All work
and no play makes Jack a
dull boy. All work and no
play makes Jack a dull boy.
All work and no play makes
Jack a dull boy. All work
and no play makes Jack a
dull boy. All work and no

Inpainting

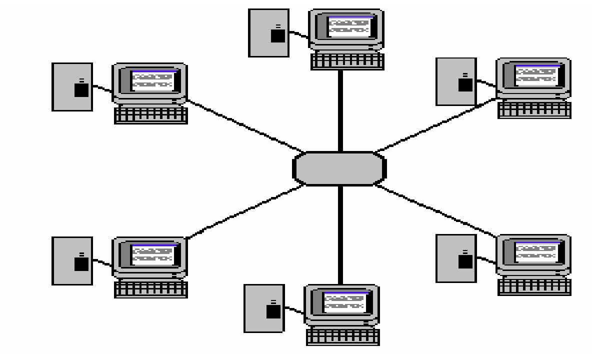


Navier-Stokes: Fluid dynamics

GPU Activities at SINTEF Applied Mathematics



Matlab Interface to the GPU



Cluster of GPU’s

$$\begin{bmatrix} b_1 - a_{-\frac{1}{2}} & -a_{\frac{1}{2}} & 0 & 0 & 0 & \cdots & 0 \\ -a_{\frac{1}{2}} & b_2 & -a_{\frac{3}{2}} & 0 & 0 & \cdots & 0 \\ 0 & -a_{\frac{3}{2}} & b_3 & -a_{\frac{5}{2}} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -a_{n-\frac{5}{2}} & b_{n-2} & -a_{n-\frac{3}{2}} & 0 \\ 0 & \cdots & 0 & 0 & -a_{n-\frac{3}{2}} & b_{n-1} & -a_{n-\frac{1}{2}} \\ 0 & \cdots & 0 & 0 & 0 & a_{n-\frac{1}{2}} & b_n - a_{n+\frac{1}{2}} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{n-2} \\ c_{n-1} \\ c_n \end{bmatrix}$$

Linear algebra / load balancing CPU - GPU

3 examples from geometry processing

- Preparation of finite element models
- Intersection acceleration
- Visualization of algebraic surfaces

GPU-Accelerated Shape Simplification for Mechanical-Based Applications

- Work partly supported by the EU through NoE AIM@SHAPE.
  - Jon Hjelmervik, SINTEF, Norway
  - Jean-Claude Leon, INPG, France
  - Presented at SMI'07

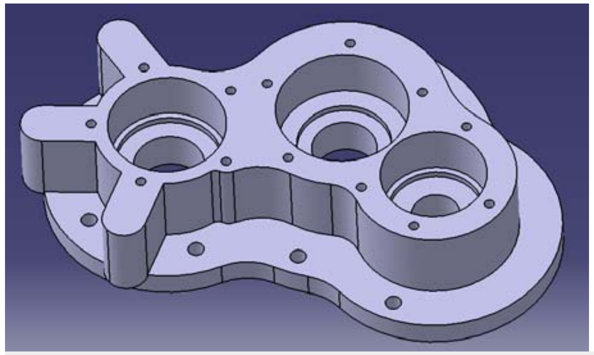
- CAD models contain shape details that are “too small to contribute to mechanical simulations”
- Overly complex models cause high running times in FEA
- Simplification process is often time consuming
- FE model preparation requires more objective criteria

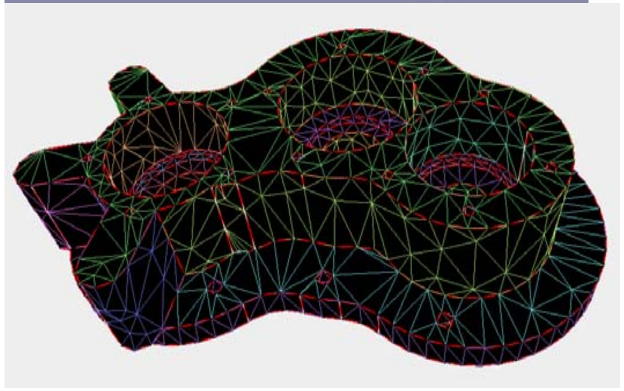
FEA preparation

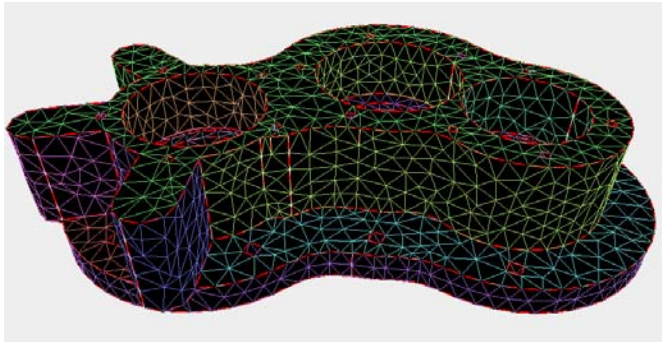
Generate initial shape
(triangulation)

Shape simplification

Generate FEA mesh
(based on triangulation)

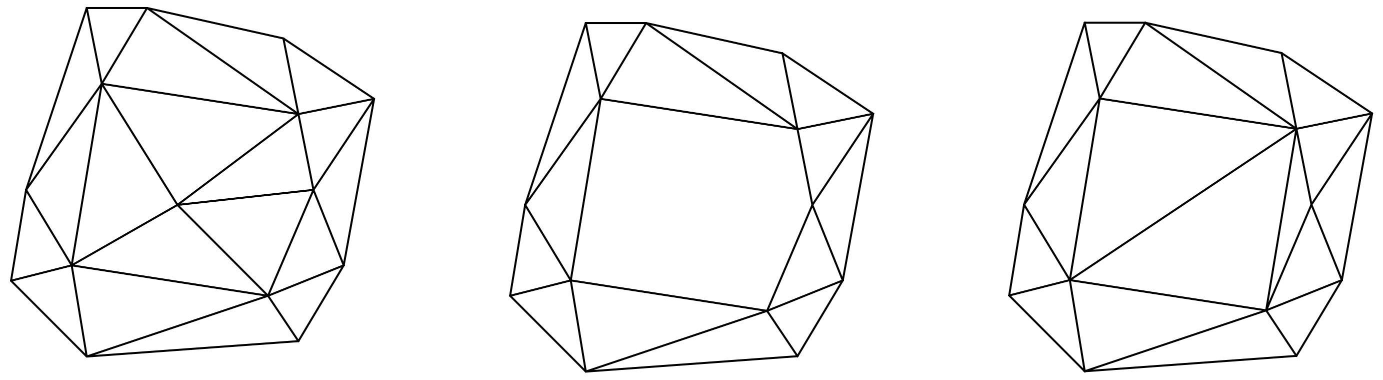






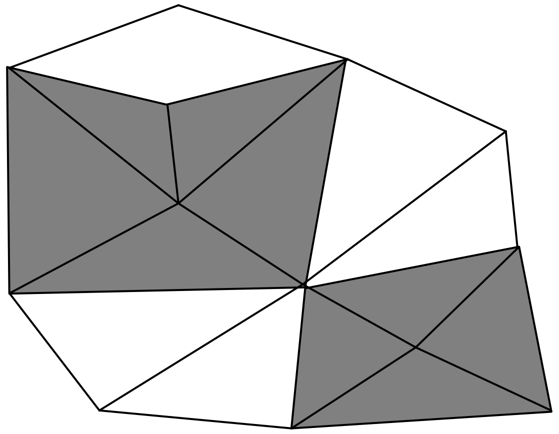
Vertex removal operator

- Assign a cost to each vertex
- Remove the vertex with lowest cost
- Remesh the hole
- Keep the remeshed version if it satisfies given criteria

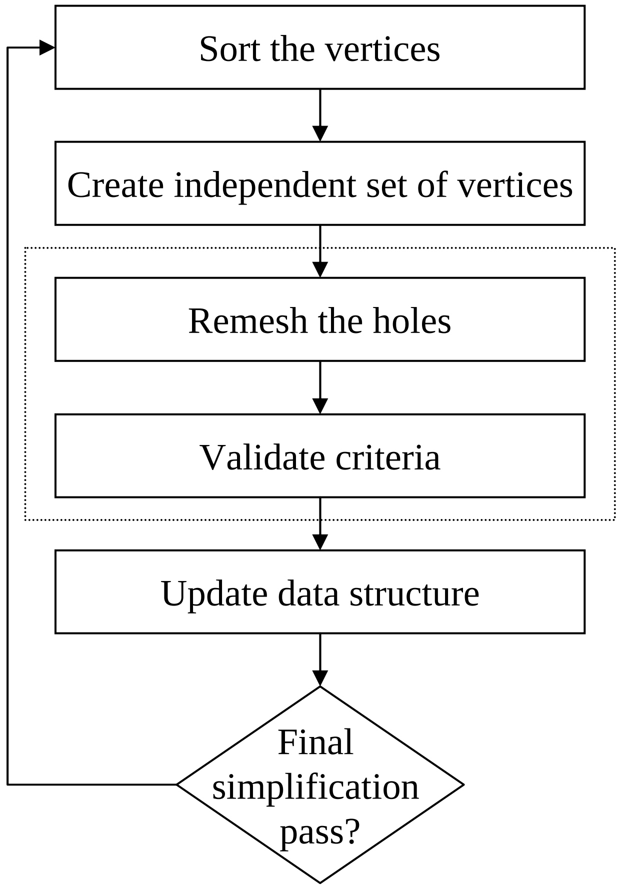


Parallelization requirements

- The ordering of vertex removals is not strict
  - Allows parallelization
- Decimation uses information in the 1-ring neighborhood
  - Information outside this area may be modified by another vertex removal



Algorithm overview

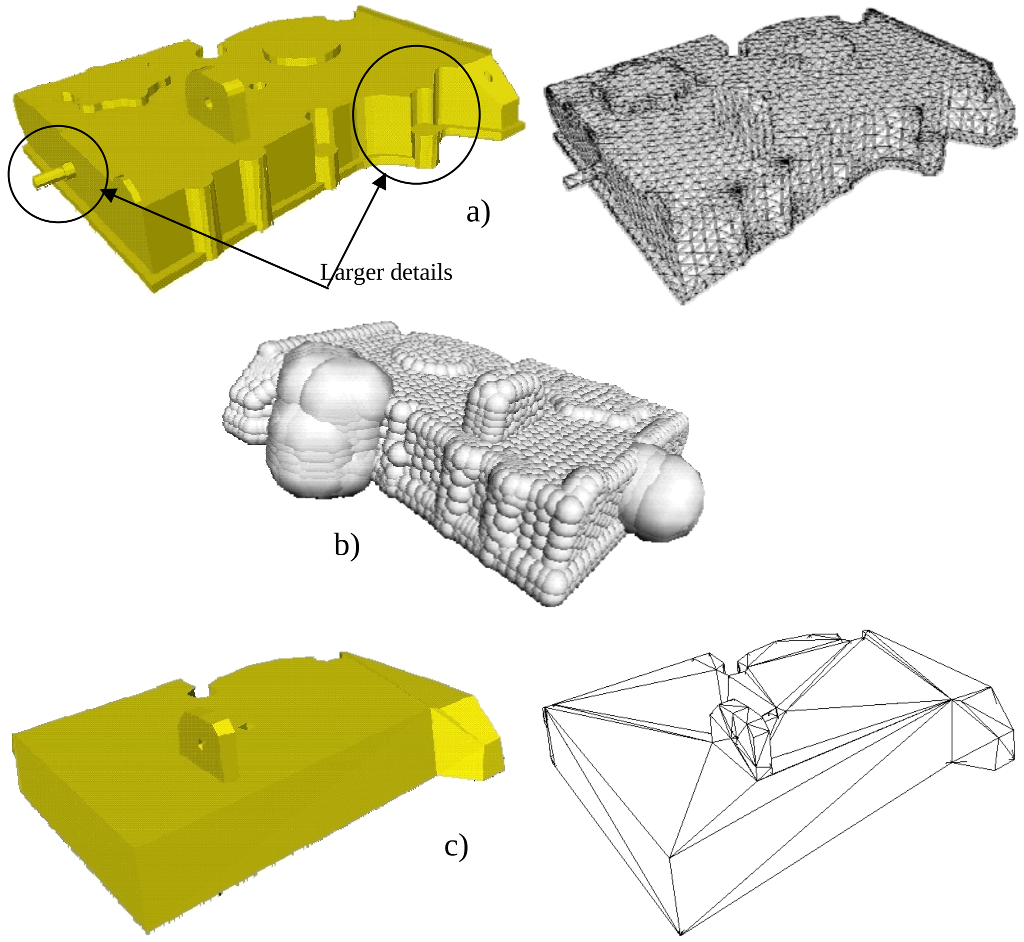


- Algorithm overview The vertices are sorted based on their cost, in order to create an independent set
- Information concerning the potential vertices are transferred to graphics memory
- Vertex removal is operated by GPU
- Result is read back to system memory
- CPU updates triangulation
- (cont)

Remeshing

- General remeshing
  - Provides a broader range of possible shapes
  - Difficult to implement on GPUs
- Half-edge collapse
  - Easy to implement (allows GPU implementation)
  - Provides sufficient freedom in remeshing for the majority of vertex removals

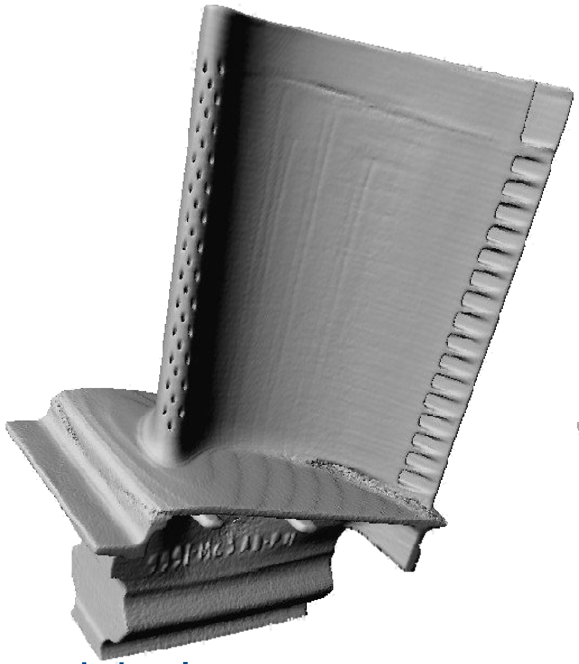
Variable geometric tolerance

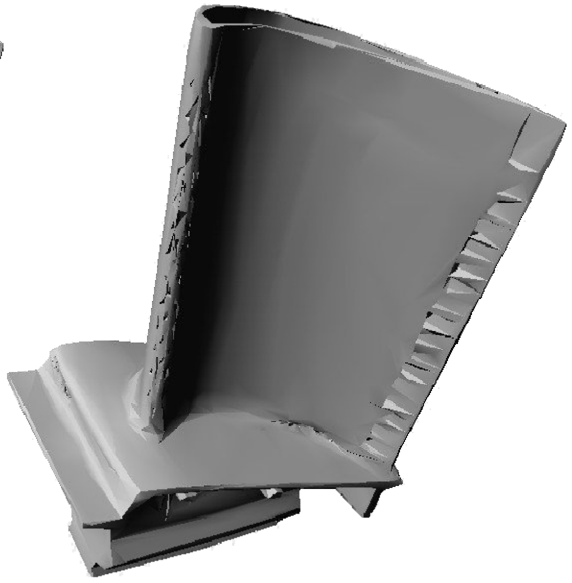


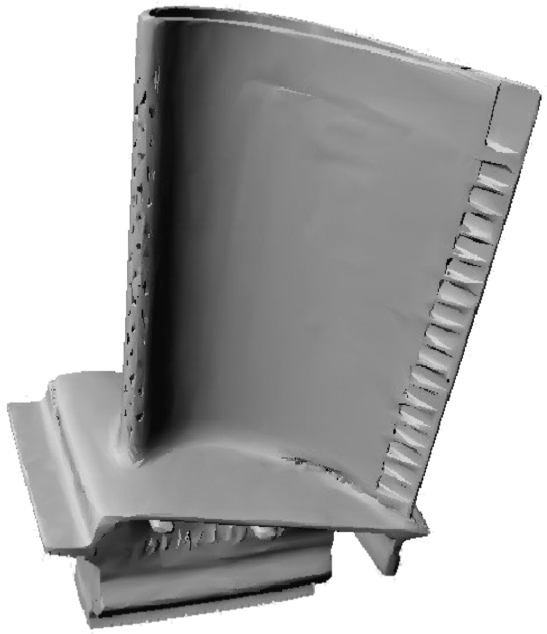
- One error zone (sphere) is attached to each original vertex
- The simplified model must intersect all error zones
- Faster than Hausdorff distance
- Allows varying geometric tolerance

Results Blade

The models were processed using an
AMD Athlon 4400+ CPU and a NVIDIA 7800GT GPU

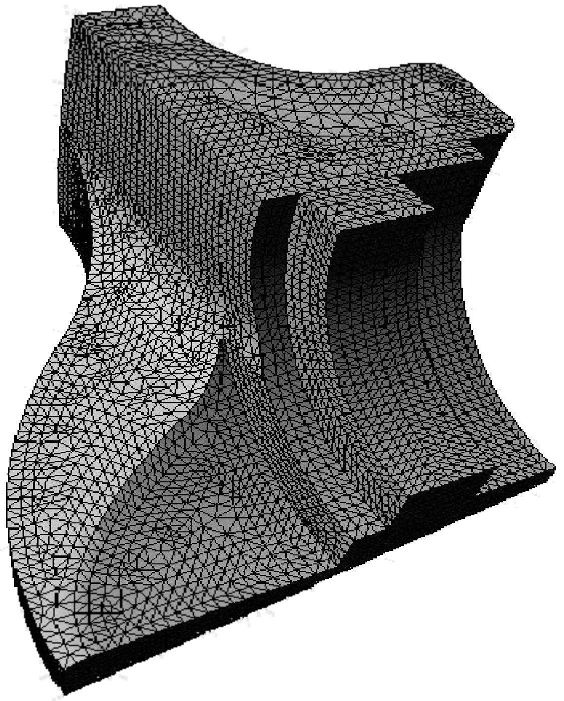

original
model


without
volume criterion

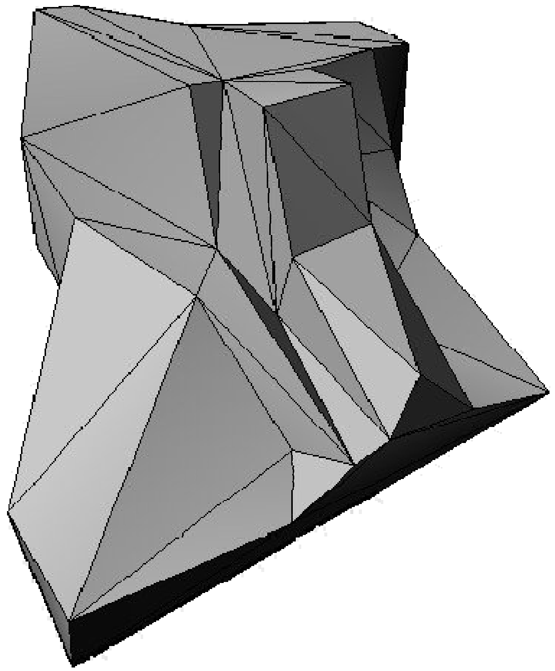

with
volume criterion

Initial #tris	Final #tris	Error zone	Volume tolerance	CPU time(s)	GPU time(s)
1 765 388	25 830	1.3%	1.7e-6%	71.4	9.7
1 765 388	10 818	1.3%	1.7e-5%	87.6	10.8

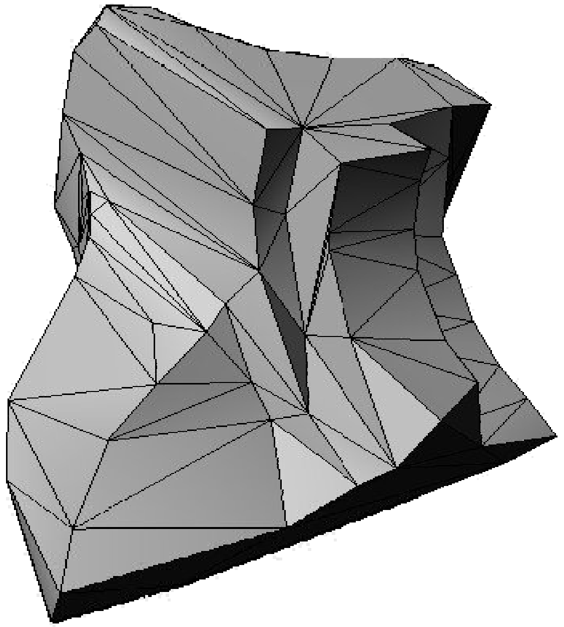
Results Fandisk



original model



without volume criterion



with volume criterion

Initial #tris	Final #tris	Error zone	Volume Tolerance	CPU time(s)	GPU time(s)
12 946	128	1.3%	1.5e1%	0.583	0.21
12 946	148	1.3%	1.5e-3%	0.57	0.21

Experiences: Shape Simplification

- The implemented validity test is computationally intensive, written for CPU, and ported to GPU
- The shaders include *for loop*, and nested *if* statements
- GPU-acceleration increases the performance with a factor of 7 against a single threaded CPU version for large models.
- More GPU-friendly criteria may increase speed up factor

Speedup CAD-type intersection algorithms

- Work by:
  - Sverre Briseid
  - Tor Dokken
  - Trond Runar Hagen all from SINTEF
  - Jens-Olav Nygaard
  - Vibeke Skytt (will address challenges of self-intersection in next talk)

- Surface intersection algorithms based on recursive subdivision are well adapted for parallelization using threading on multi-core CPUs
  - Data structures and data storage strategies have to be modified to support parallel subproblems.
- Massive parallel subdivision can help making more tight:
  - bounding boxes for interference checking
  - normal cones for loop destruction

  to analyze the complexity of an intersection problem to determine the best algorithm approach to use

Comparison of CPU and GPU implementation of subdivision

Initial process for surface self-intersection

- Subdivision into 2^n x 2^n subpatches of a bicubic Bezier patch and the associated (quintic) patch representing the surface normals
- Test for degenerate normals for subpatches
- Computation of the approximate normal cones for subpatches
- Computation of the approximate bounding boxes for subpatches
- Computation of bounding box pair intersections for subpatches

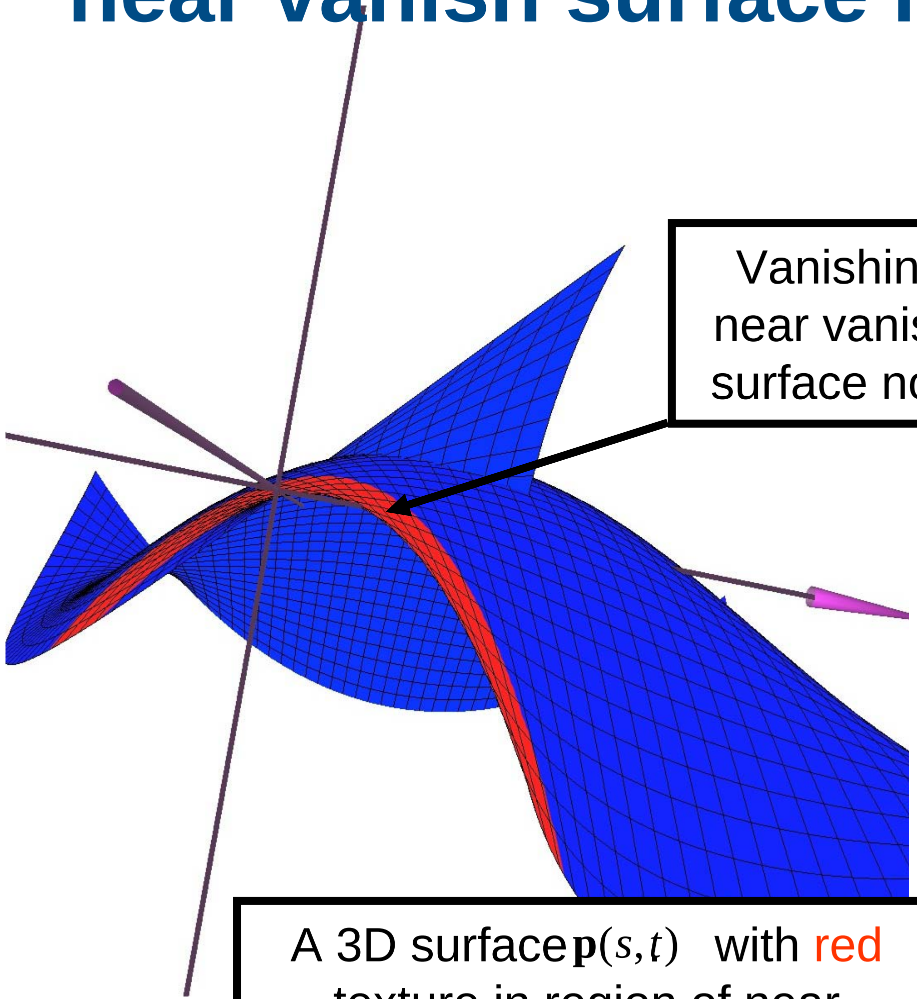
n	Grid	GPU[1]	CPU[2]	Speedup
4	16 × 16	7.456e-03	6.831e-03	0.9
5	32 × 32	1.138e-02	7.330e-02	6.4
6	64 × 64	7.271e-02	1.043e00	14.3
7	128 × 128	9.573e-01	1.607e01	16.8
8	256 × 256	1.515e01	2.555e02	16.9

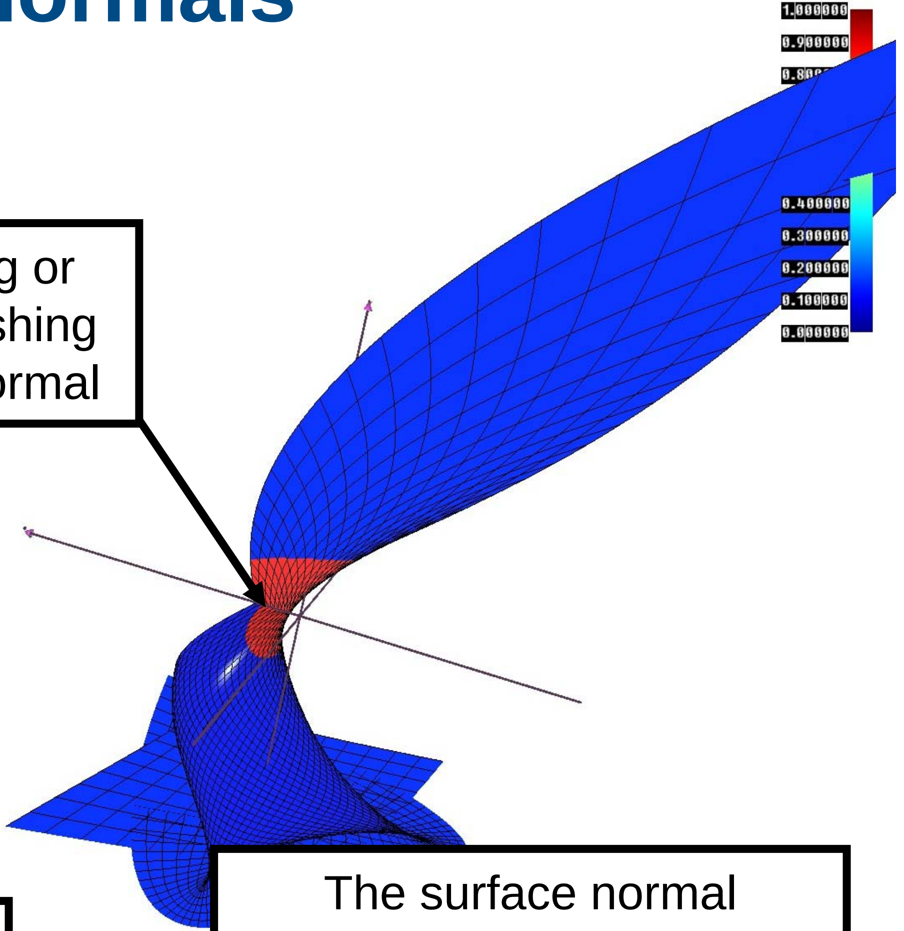
CPU/GPU approach

- For n<5 use CPU
- Refined checks use GPU
- For problems not sorted out use recursive subdivision approach on CPU, or refined subdivision on GPU

1. NVIDIA GeForce 7800GT graphics cards (165 GFlops), (2007: G80 Cards 500 GFlops)
2. On a single core of AMD X2 4400

Simple subdivision to identify regions with near vanish surface normals





Vanishing or near vanishing surface normal

A 3D surface $\mathbf{p}(s,t)$ with red texture in region of near vanishing normal

The surface normal

$$\frac{\partial \mathbf{p}(s,t)}{\partial s} \times \frac{\partial \mathbf{p}(s,t)}{\partial t}$$

described as a surface

Ray-casting of algebraic surface

- Idea triggered by the talk of Charles Loop at the SIAM GD conference (Phoenix 2005) addressing the GPU used for visualization of algebraic curves"
  - Work initiated by Tor Dokken, main work load carried out by Johan S Seland (CMA/SINTEF)
  - Second version work headed by Johan S. Seland in cooperation with Martin Reimers at CMA, University of Oslo.
- Solves the ray surface intersection of ray-casting on the GPU using subdivision
  - Challenge: Intersecting singularities and near singularities!
- Current version works for degrees up to 20, near real time up to degree 12
- Next: Start video from interactive section. (Paper Accepted for Eurographics 2008

Start of the GPGPU-project

- We started out addressing explicit solutions of partial differential equations:
  - Simple grids and data structures.
  - Classical finite-difference methods are very simple.
  - Embarrassingly parallel.
  - Almost perfect speedup expected.

- Limitation of GPUs until now:
  - You can read from general locations
  - You can only write a limited number of floats to specific location

Why solve PDEs on GPUs?

- Simple grids and data structures.
- Classical finite-difference methods are very simple.
- Embarrassingly parallel.
- Almost perfect speedup expected.
- Best speed up for advanced schemes

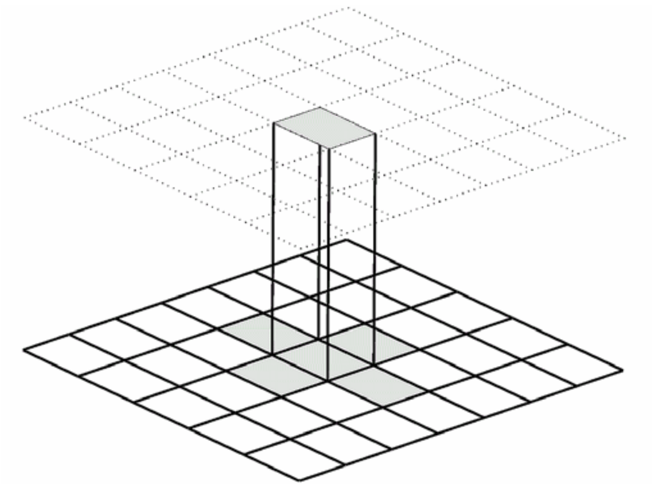
Example: Heat Equation

$$u_t = u_{xx} + u_{yy}$$

Discretication by finite differences over a regular grid:

$$U_{i,j}^{n+1} = U_{i,j}^{n} + \frac{k}{h^2}\left(U_{i+1,j}^{n} + U_{i-1,j}^{n} + U_{i,j-1}^{n} + U_{i,j+1}^{n} - 4U_{i,j}^{n}\right).$$

Each fragment updated as a weighted sum of its nearest five neighbours.

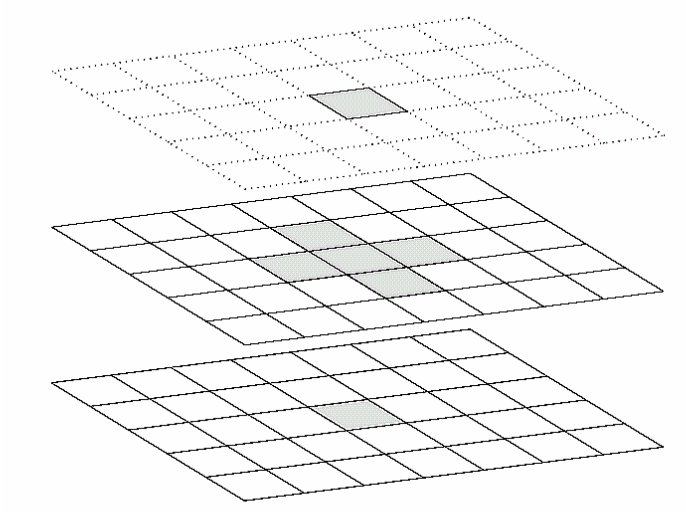


Example: Linear Wave Equation

$$u_{tt} = u_{xx} + u_{yy}$$

Discretication by finite differences over a regular grid:

$$U_{i,j}^{n+1} = 2U_{i,j}^{n} - U_{i,j}^{n-1} + \frac{k^2}{h^2}\left(U_{i+1,j}^{n} + U_{i-1,j}^{n} + U_{i,j-1}^{n} + U_{i,j+1}^{n} - 4U_{i,j}^{n}\right).$$

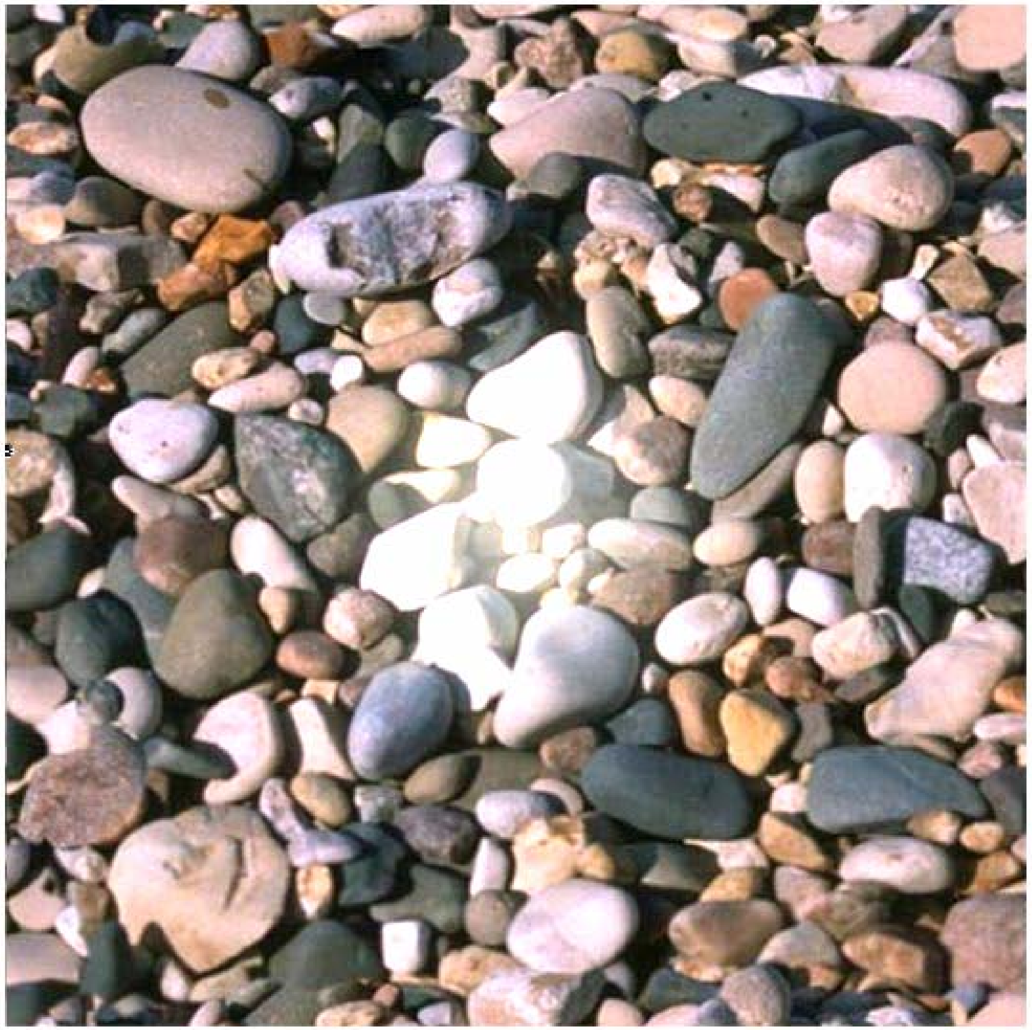


Almost the heat equation, but needs extra texture to store the value at *n-1*

Wave Equation contd.

SINTEF ICT APPLIED MATHEMATICS

Wave Equation contd.



Systems of Conservation Laws

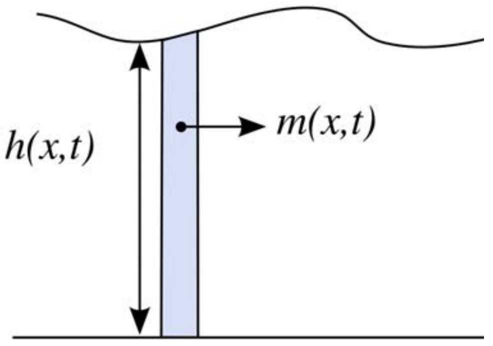
- Fundamental laws of physics: conservation of quantities like mass, momentum and energy.

- In arbitrary space dimension this reads:

$$Q_t + \nabla \cdot f(Q) = 0, \qquad Q(x,0) = Q_0(x)$$

- Example: Shallow water equations

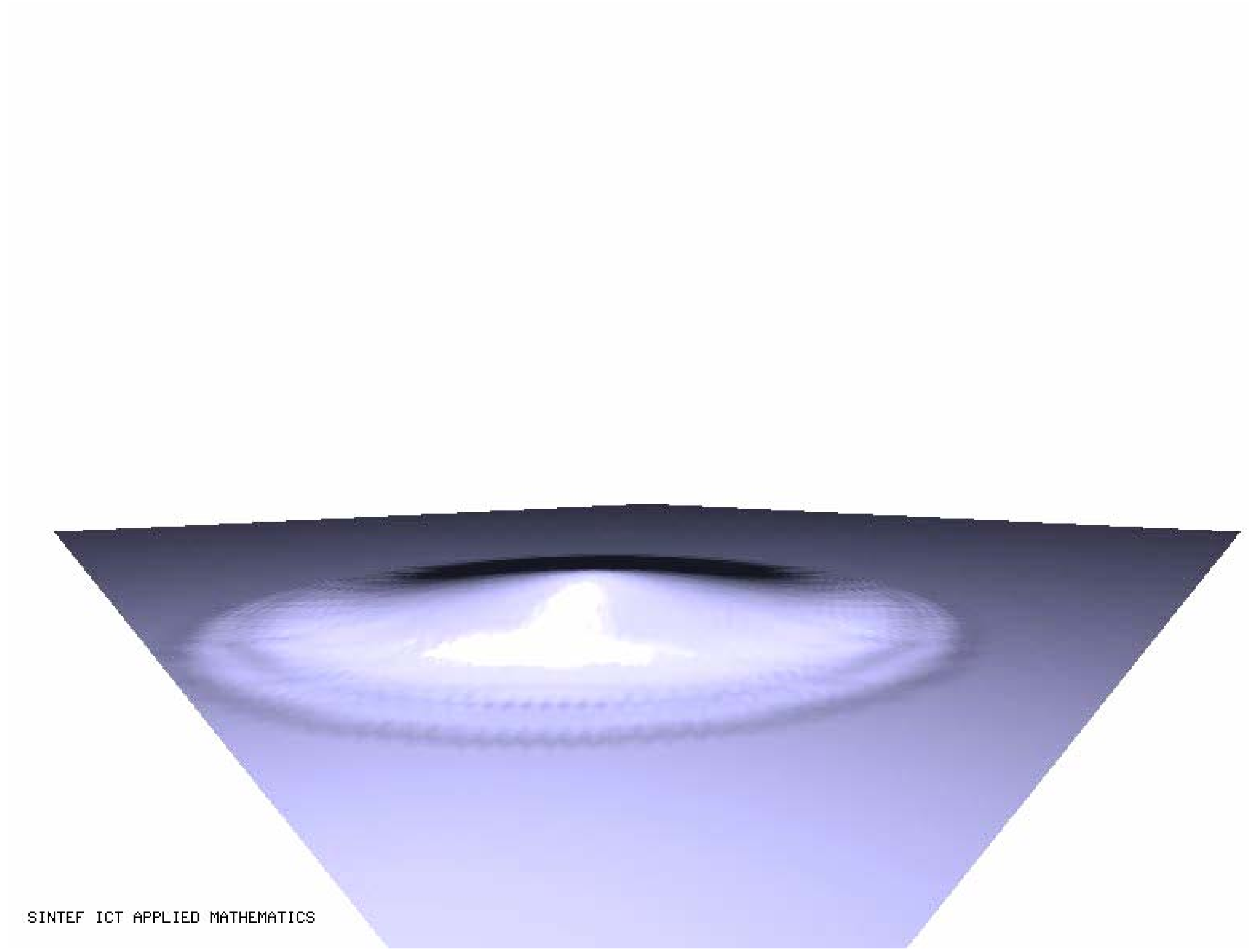
$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$



Solving Hyperbolic Conservation Laws on GPUs (or Cell processors) - Motivation

- Most high-resolution schemes for conservation laws are explicit.
- Explicit schemes are embarrassingly parallel.
- Single precision is not a problem due to numerically stable algorithm.
- Vector of unknowns in each cell $\longrightarrow$ easy to utilize vector operations.
- Complex schemes $\longrightarrow$ high number of arithmetic operations per memory operation.
- Finite speed of wave propagation $\longrightarrow$ easy to decompose computational domain into subdomains. Benefits:
  - Overcomes lack of memory on GPUs.
  - Potential for cluster implementations.

Classical scheme: Lax-Friedrich



Speedup - *Lax-Friedrichs*

- Runtime per time step and speedup factor for the CPU versus GPU implementation of *Lax-Friedrichs*

N	CPU* ms	GPU** ms	Speedup
128x128	2.22	0.23	9.53
256x256	9.09	0.46	19.8
512x512	37.10	1.47	25.2
1024x1024	148.00	5.54	26.7

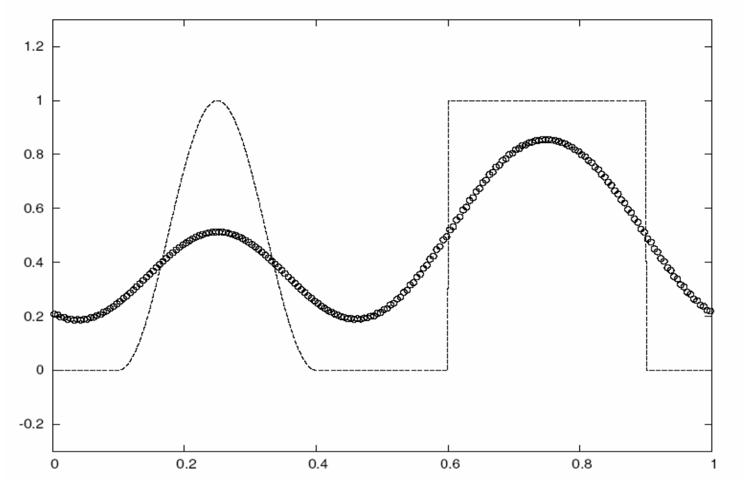
* 2.8 GHz Intel Xeon (EM64T)

** GeForce 7800 GTX (450 MHz)

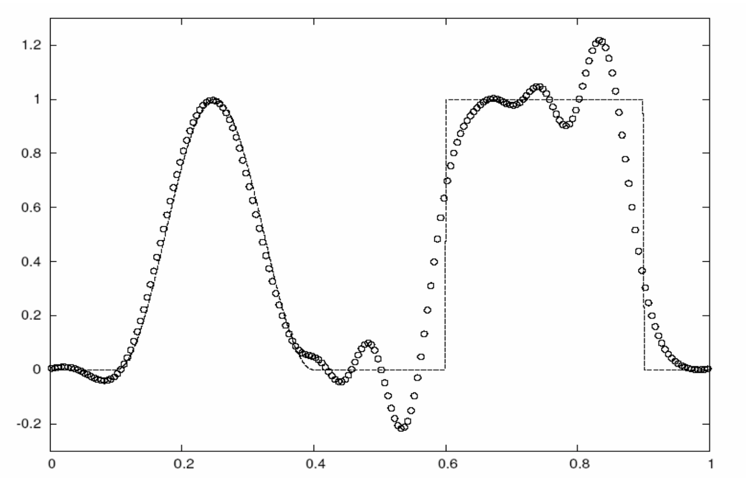
Linear Advection

- Models various transport phenomena of (passive) quantities. Simple equation, but difficult to compute solutions correctly.

1st order scheme: Lax-Friedrich



2nd order scheme: Lax-Wendroff



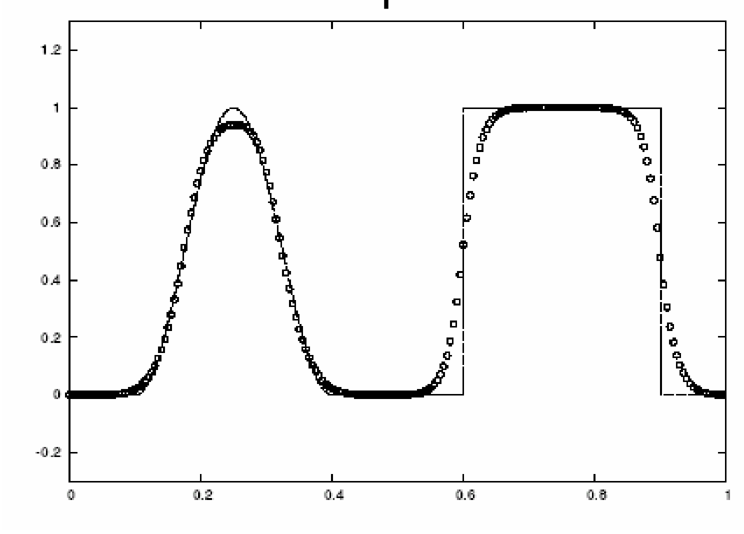
- Classical schemes: excessive smearing or spurious oscillations.

  $\longrightarrow$ Need for more sophisticated schemes.

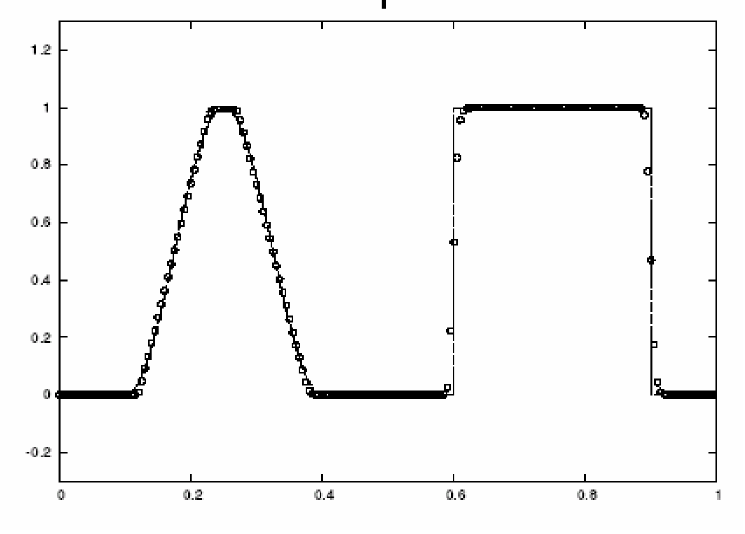
Linear Advection contd.

- Modern high-resolution schemes:

Dissipative



Compressive



- Higher-order approximation of smooth parts and no spurious oscillations at discontinuities.

Semi-Discrete High-Resolution Schemes

- Evolution of cell averages described by ODEs

$$\frac{d}{dt}U_{ij}(t) = \frac{\left(F_{i-1/2,j}(t) - F_{i+1/2,j}(t)\right)}{\Delta x} + \frac{\left(G_{i,j-1/2}(t) - G_{i,j+1/2}(t)\right)}{\Delta y}$$

- Steps in the algorithms:
  - Reconstruction of piecewise polynomials from cell averages
  - Evaluation of reconstruction at integration points
  - Numerical computation of edge fluxes
  - Evolution by Runge-Kutta scheme

Speedup – 2nd order high-resolution

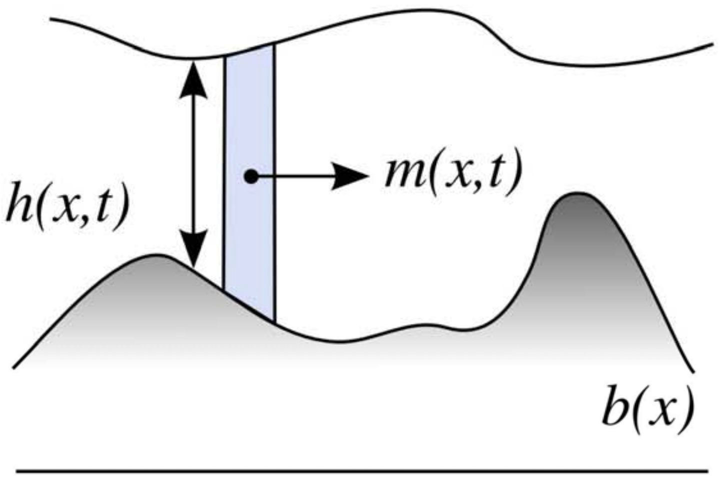
- Runtime per time step and speedup factor for the CPU versus the GPU implementation of *bilinear interpolation* with modified *minmod* limiter for the shock-bubble problem of gas dynamics. The results relate to *second-order Runge-Kutta* time stepping.

N	CPU*	GPU*	Speedup
	ms	ms	
128x128	30.6	1.27	24.2
256x256	122	4.19	29.1
512x512	486	16.8	28.9
1024x1024	2050	68.3	30.0

* 2.8 GHz Intel Xeon (EM64T)

** GeForce 7800 GTX (450 MHz)

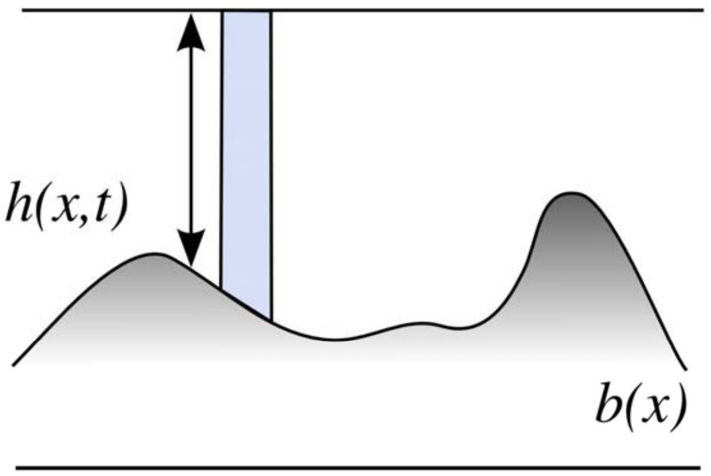
Variable bottom topography



$$h_t + m_x = 0,$$

$$m_t + [\frac{m^2}{h}]_x + gh(b+h)_x = 0,$$

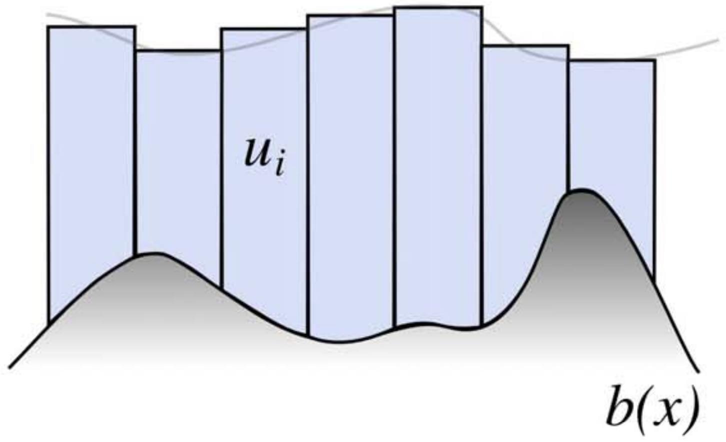
Well balanced schemes



$$
\begin{aligned}
h_t + m_x &= 0, \\
m_t + [\frac{m^2}{h} + \frac{1}{2} g h^2]_x &= g h b_x,
\end{aligned}
$$

The terms in red should balance exactly

Finite-Volume Scheme



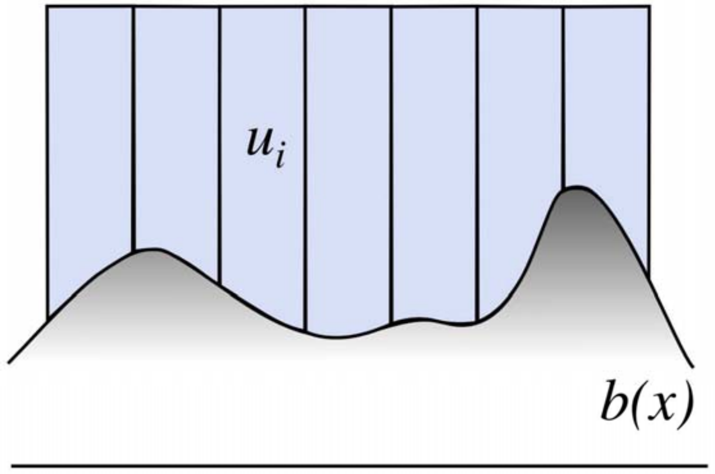
Simple conservative finite-volume scheme.

$$\frac{\partial u_i}{\partial t} + \frac{1}{\Delta x}[F^n_{i+1/2} - F^n_{i-1/2}] = B_i,$$

$$F_{i+1/2} = F(u_L, u_R).$$

The scheme is determined by the choice of F and the approximation of u_L and u_R

Finite-Volume Scheme

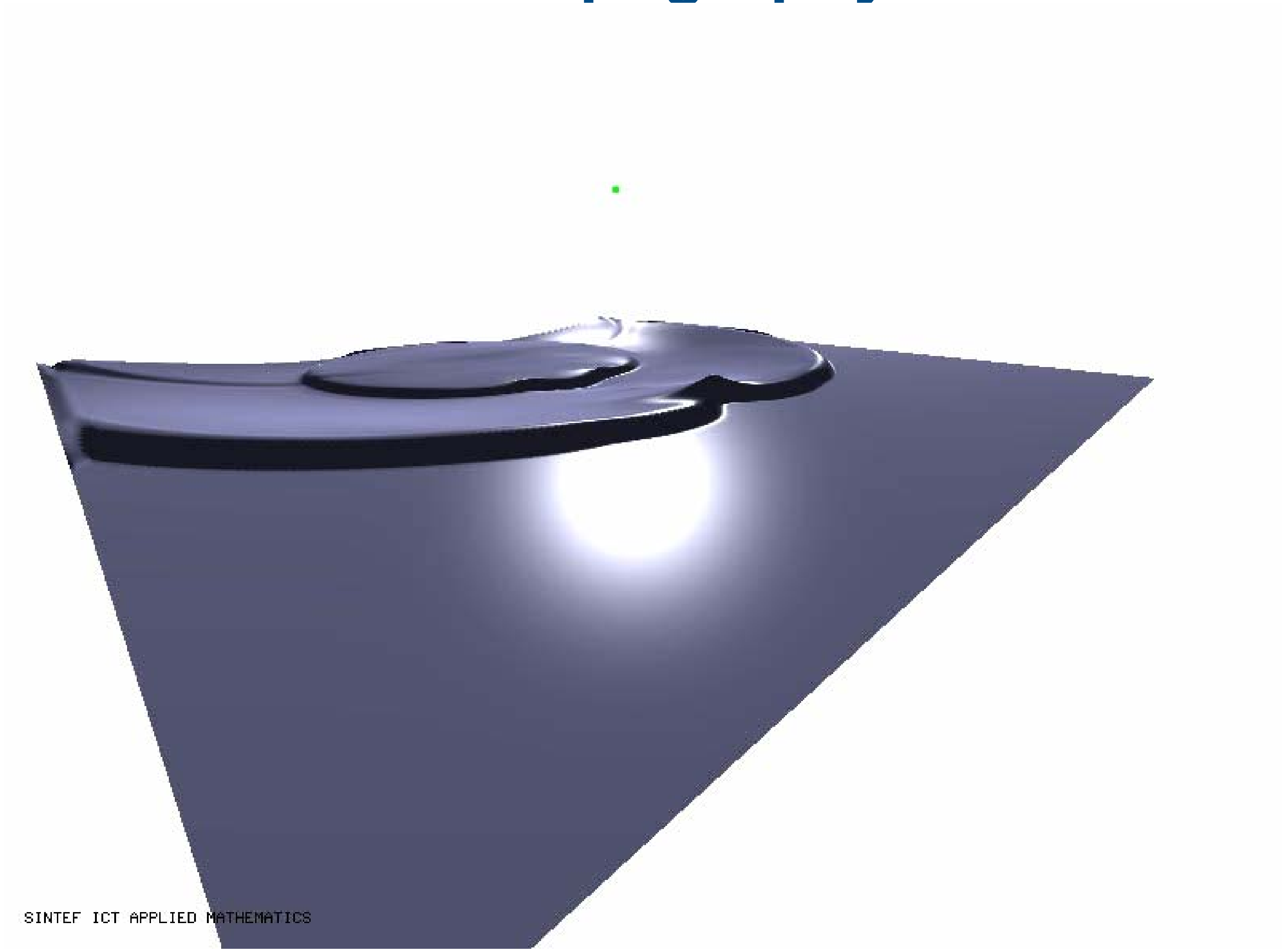


When $b + h = C$ and $m = 0$, we get

$$\frac{1}{\Delta x}[F^n_{i+1/2} - F^n_{i-1/2}] = B_i$$

Truncation error must vanish at equilibrium!

2nd order central upwind scheme
- Variable bottom topography



Example – Shallow Water Waves

- Free-surface flow over a variable bottom topography under the influence of gravity can be modeled by the shallow-water equations

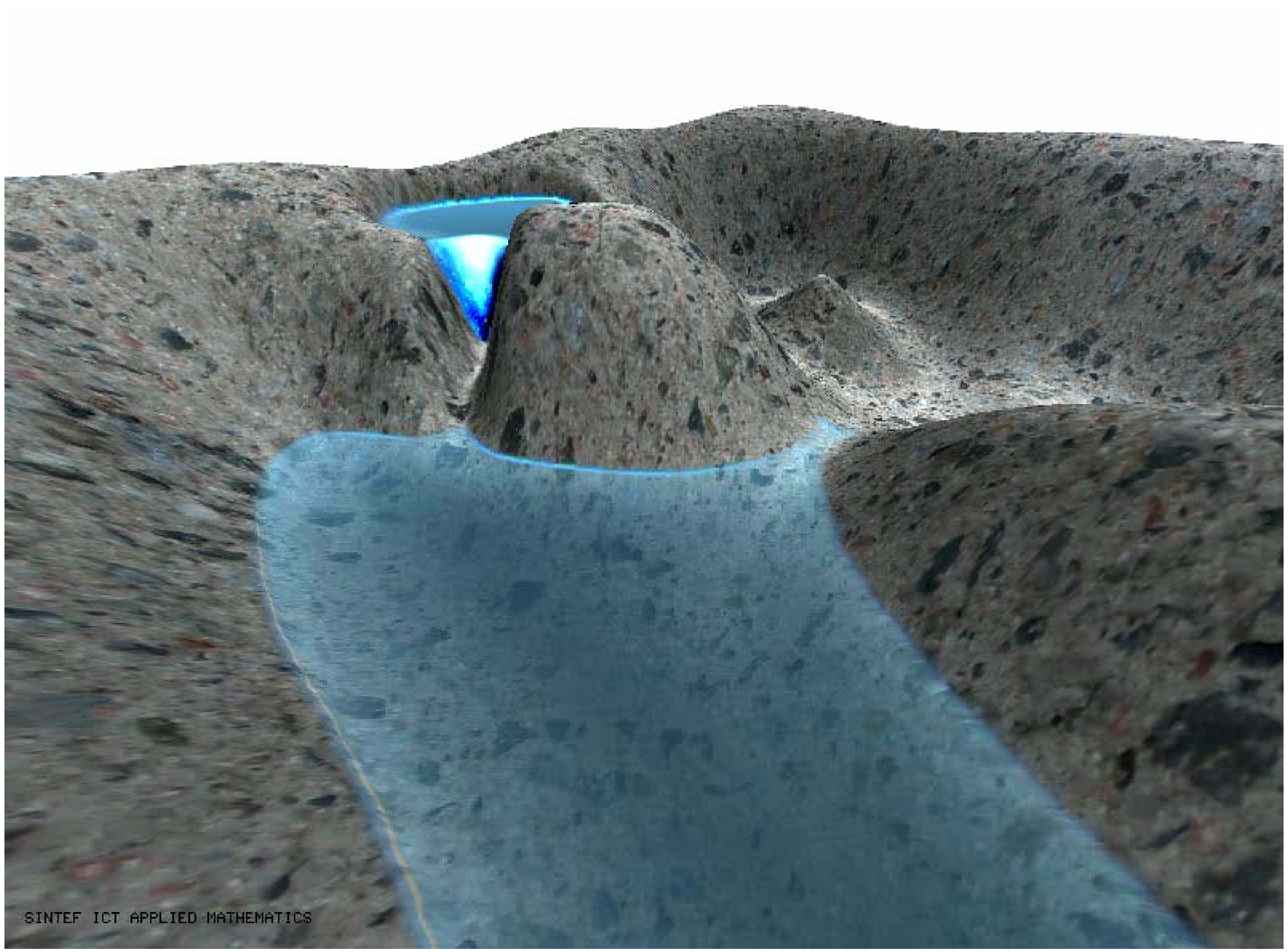
$$\begin{bmatrix} h \\ hu \\ hv \end{bmatrix}_t + \begin{bmatrix} hu \\ hu^2 + \frac{1}{2}gh^2 \\ huv \end{bmatrix}_x + \begin{bmatrix} hv \\ huv \\ hu^2 + \frac{1}{2}gh^2 \end{bmatrix}_y = \begin{bmatrix} 0 \\ -gh\frac{\delta B}{\delta x} \\ -gh\frac{\delta B}{\delta y} \end{bmatrix}$$

- Here B(x,y) is the bottom topography, h(x,y,t) is the distance from the bottom to the (wavy) surface, [u,v] is the depth-averaged velocity, and g is the gravitational acceleration.

Runtimes per time step in seconds of the dam-break problem on a N x N grid simulated using a dry-states shallow-water scheme.

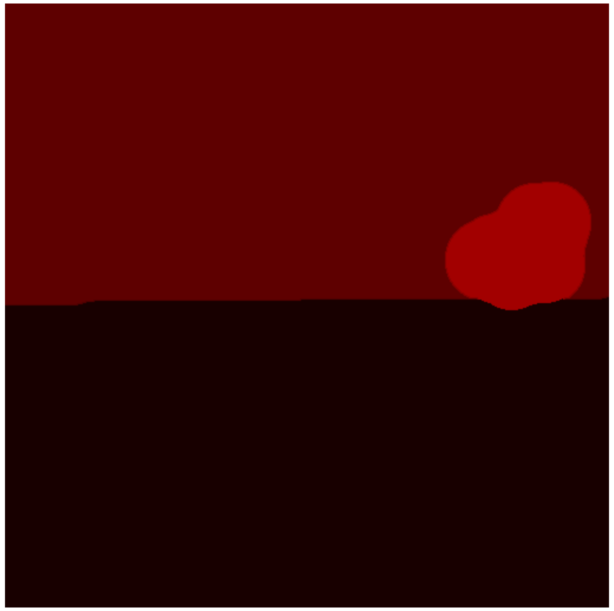
N	Intel Xeon 2.8 GHz	GeForce 7800 GTX	Speedup
256	1.43e-1	8.09e-3	17.7
512	5.99e-1	3.19e-2	18.8
1024	3.27e-0	1.42e-1	23.0

2nd order central upwind scheme
- Well balanced and with dry states

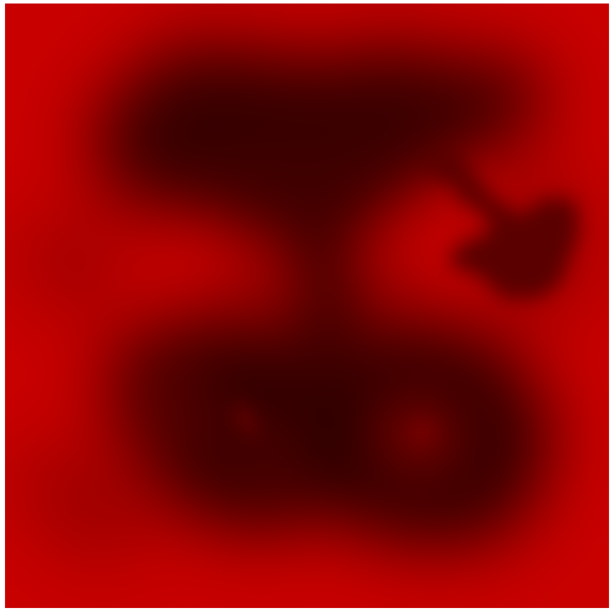


2nd order central upwind scheme
- Well balanced and with dry states

Initial wave map



Initial bottom map



Euler equations

- The two dimensional Euler equations model the dynamics of compressible gasses:

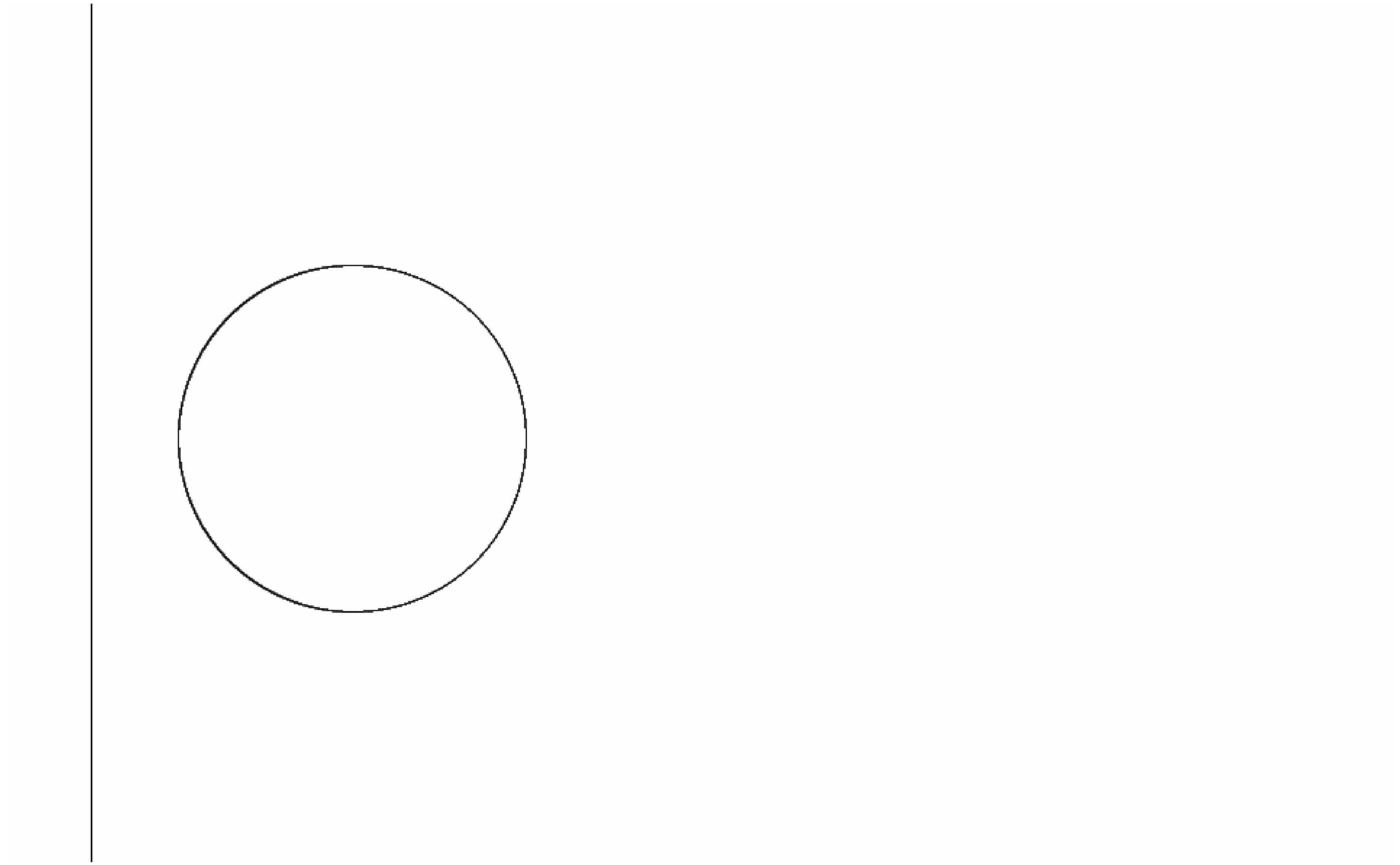
$$\begin{bmatrix} \rho \\ \rho u \\ \rho v \\ E \end{bmatrix}_t + \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ u(E+p) \end{bmatrix}_x + \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ v(E+p) \end{bmatrix}_y = 0$$

  ρ denotes density, *u* and *v* velocity in x- and y- directions, *p* pressure and *E* the total energy.

- The three dimensional

$$\begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho w \\ E \end{bmatrix}_t + \begin{bmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uw \\ u(E+p) \end{bmatrix}_x + \begin{bmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vw \\ v(E+p) \end{bmatrix}_y + \begin{bmatrix} \rho w \\ \rho uw \\ \rho vw \\ \rho w^2 + p \\ w(E+p) \end{bmatrix}_z = \begin{bmatrix} 0 \\ 0 \\ 0 \\ g\rho \\ g\rho w \end{bmatrix}$$

Example: 2D Euler Equations - Interaction of a low-density bubble with a shock.



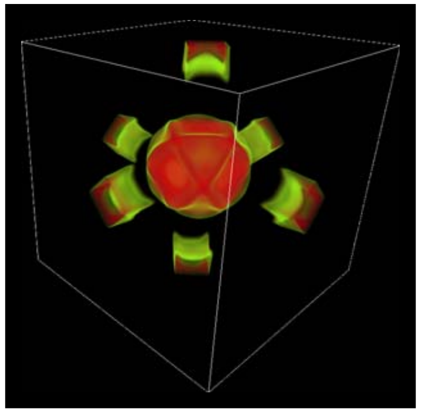
Speedup of 2D shock-bubble on NxN cells

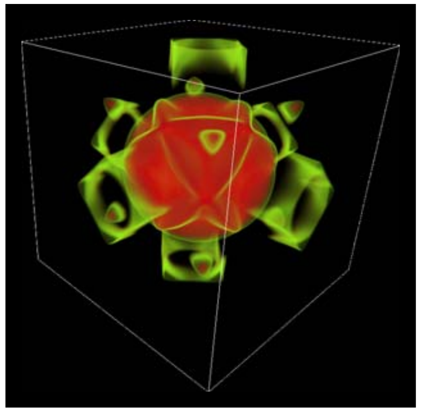
Bilinear reconstruction						
N	**Intel**	**6800**	**speedup**	**AMD**	**7800**	**speedup**
128	4.37e-2	3.70e-3	11.8	1.88e-2	1.38e-3	13.6
256	1.74e-1	8.69e-3	20.0	1.08e-1	4.37e-3	24.7
512	6.90e-1	3.32e-2	20.8	2.95e-1	1.72e-2	17.1
1024	2.95e-0	1.48e-1	19.9	1.26e-0	7.62e-2	16.5

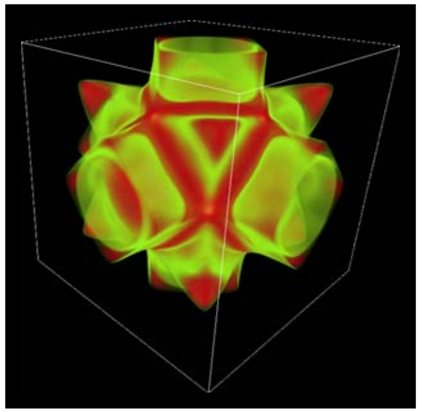
CWENO reconstruction						
N	**Intel**	**6800**	**speedup**	**AMD**	**7800**	**speedup**
128	1.05e-1	1.22e-2	8.6	7.90e-2	4.60e-3	17.2
256	4.20e-1	4.99e-2	8.4	3.45e-1	1.74e-2	19.8
512	1.67e-0	1.78e-1	9.4	1.03e-0	6.86e-2	15.0
1024	6.67e-0	7.14e-1	9.3	4.32e-0	2.99e-1	14.4

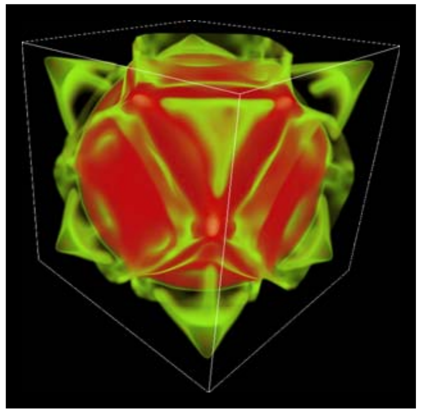
Example – 3D Euler Equations contd.

The images below shows a circular explosion inside a cubic container.









Runtimes per time step in seconds of the Rayleigh – Taylor instability on a N x N x N grid simulated both on a GPU and a CPU.

N	AMD X2 4400+	GeForce 7800 GTX	Speedup
49	5.23e-1	4.16e-2	12.6
64	1.14e-0	8.2e-2	13.9
81	1.98e-0	1.72e-1	11.5

Semi-Discrete High-Resolution Schemes

- Evolution of cell averages described by ODEs

$$\frac{d}{dt}Q_{i,j}(t) = \frac{\left(F_{i-1/2,j}(t) - F_{i+1/2,j}(t)\right)}{\Delta x} + \frac{\left(G_{i,j-1/2}(t) - G_{i,j+1/2}(t)\right)}{\Delta y}$$

- Steps in the algorithms:
  - Reconstruction of piecewise polynomials from cell averages
  - Evaluation of reconstruction at integration points
  - Numerical computation of edge fluxes
  - Evolution by Runge-Kutta scheme

Future plans

- We hope/expect the GPGPU-project to be extended for the period 2008-2010 under the name “Heterogeneous Computing”
- We plan to continue work within 3 work packages
  - Generic heterogeneous computing
  - Partial Differential Equations
    - Parallelization on Cell BE
    - Parallelization on GPU-clusters
  - Geometry processing
    - Visualization of Scalar and Vector fields
    - Intersection and collision detection
    - FEM model preparation