

# **GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA**

## **GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA**

**Shuang Zhang, Mike Heck, Patrick Barthelemy**

**Visualization Sciences Group, Mercury Computer Systems, Inc., USA**

### **THEME**

Visualization.

### **KEYWORDS**

Unstructured mesh, Double precision, Field extraction, Feature extraction, GPGPU Computing

### **SUMMARY**

Data extraction is a common task across CAE workflow, particularly in simulation and result post-processing. The performance of this task is critically important to any interactivity during the workflow. In this study, the latest computing technology is employed to address this issue. Impressive performance gain has been obtained on field calculations for large volume of seismic survey data which is hierarchically represented [8]. In this paper, the attention is turned to unstructured solution data of numerical simulations based on requirements drawn from cross disciplinary fields including geophysical modelling, material analysis and CAE.

A software framework is designed where algorithms can be conveniently implemented in a heterogeneous computing environment with one or multiple CPU, one or multiple GPGPU.

Two important algorithms are identified as subjects of this acceleration study,

- Velocity gradient reconstruction [4].
- Extraction of vortex corelines [13].

Performance results are reported for unstructured mesh running in double precision on GPGPU.

### **1: INTRODUCTION**

CFD and FEA software are traditionally known as computation intensive applications. General Purpose (computing on) Graphics Processing Unit

## **GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA**

(GPGPU for short here after), has a hardware architecture that is suitable for massively parallel computing algorithms. Large performance benefit has been demonstrated in many preliminary case studies [8], at a low cost particularly for cluster configuration. GPGPU also out-paced CPU in technology refresh rate [1], and better prepared for future scalability as it will be easier for GPU designers to add transistors for higher arithmetic intensity. The compound benefits provided by GPGPU provided great potential to computation intensive tasks, hence have attracted much attention in scientific computing community.

Despite the promises shown, programmability, management of large unstructured mesh and arithmetic precisions remain to be three major concerns for wider adoption of GPGPU in CFD/FEA.

This paper will examine the performance prospect of an implementation of vortex feature extraction algorithm [12,13], which is vital in flow visualization. Vortex dynamics contributes significantly to turbulence, mixing, and unsteadiness of fluid flow. It in turn impacts, for examples, the design of the turbomachinery blade geometry, the study of fuel efficiency inside a combustion chamber, and the safe distance between two airplanes. After obtaining the integrated parameters (lump parameters) such as inflow and outflow rates, lift and drags, engineers and physicist would like to examine further the details of the 3D flow field. Quantitative measure on the vortex geometry is a very useful tool for this task.

As the data produced from modern CFD or flow measurements is exploding, extraordinary computing power is needed to process these data in a timely manner, including but not limiting to vortex feature extraction. CUDA interface [8] has provided the level of programmability on GPU required by CFD application developers, which is adopted in this work. The impact of floating-point arithmetic precision (single versus double) is examined specifically. The data structure and algorithms are designed as generic as possible for different devices (CPU and GPGPU). Specifically, unstructured mesh with mixed elements is taken into account, as opposed to tetrahedron only mesh assumed by most of existing literature. Shared memory is used for geometry and solution arrays to reduce the memory footprint of the application, as opposed to flattening the unstructured nature of the data and make the elements independent.

It is important to note that GPGPU are not suitable for every algorithm. Berkeley dwarf view [2] is adapted and acceleration is made only when appropriate. Specifically, gradient reconstruction algorithm and the iterative solver for locating vortex core point on a face are studied. The gradient algorithm, based on Green-Gauss theory, is of generic importance to both CFD visualization and solver technology.

# **GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA**

The algorithm is implemented as part of Open Inventor MeshViz XLM, and will be commercially available at both software development kit level and application level through Avizo.

The paper is organized as the following: Section 2 surveys briefly the GPGPU research for unstructured mesh. Section 3 reviews the basic algorithms to be accelerated. Section 4 discusses the implementation details of the GPU acceleration algorithm. Section 5 presents the results. Section 6 highlights directions for future efforts.

## **2: LITERATURE**

GPU computation on unstructured dataset are mainly seen in volume visualization [10], represented by variants of projected tetrahedra (PT) algorithm [8,11]. The performance gains mainly benefited user interactivity of the application. More recently, Maximo et. al.[2] proposed a GPU based ray casting algorithm for unstructured volume rendering, which employed CUDA API. Connectivity of the cells, very relevant to general CFD/FEA data processing, was taken into account. Moreover, it focused on reducing memory footprint of the application, which makes it very valuable toward handling practical size CFD/FEA datasets. Important assumptions made are a). tetrahedron only cells, b). fixed patterns of cell to node connectivity which implicitly define face to node connectivity, and c). coherency of the casted rays which yields more predictable memory access patterns. This study shows a dataset with 1.9M nodes, 22.1M faces and 11.2M cells rendered with 426MB of memory and 3081 ms wall clock time.

While many efforts were made on GPU accelerated CFD solver [2], GPU work on unstructured CFD solution is relatively sparse. In [4], an elements-surrounding-elements data structure gives the connectivity on a per-element basis. For single precision, tetrahedron only solver, [4] reported an impressive 68 times speed-up with a Tesla box. On the precision front, the work is mainly represented by [6] where a mixed-precision FEM solver is studied extensively.

## **3: BASIC ALGORITHM AND SOFTWARE FRAMEWORK**

In this section, the feature extraction algorithm is reviewed. The assumption of the readily available velocity gradient tensor made in [13] is removed in this implementation, which makes gradient reconstruction the first and foremost task.

### **3.1: Gradient reconstruction**

Reconstructing gradient tensor from velocity vector field in CFD solutions is mandatory to any vortex identification or extraction algorithms, hence is the

# GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

first step in this work. An unstructured gradient reconstruction algorithm is employed based on Green-Gauss theorem [4], where the cell-valued gradient is computed from the fluxes on all its facets.

The algorithm can be applied to both scalar and vector field, on any polyhedron. When applying to a vector field, velocity in this case, the algorithm is implemented as illustrated by the pseudo-code in Figure 1,

```
Initialize an empty 3x3 tensor array;
Retrieve grid connectivity and velocity at cell center;
For all cells:
  For each facet that belongs to the cell
    Compute facet area;
    Get velocity value for neighbour cells;
    Compute velocity value for current facet;
    Compute gradient contribution of current facet;
    Accumulate the current contribution to tensor array;
  Endfor
  Compute cell volume
  Divide tensor array element by cell volume
EndFor
```

**Figure 1: Pseudo-code for gradient reconstruction**

Facet area vector will be computed twice in exchange of the memory efficiency, achieved by removing facet to node connectivity array.

## 3.2: Vortex Coreline

Once the velocity gradient tensor field is reconstructed, it is used, in conjunction with velocity field and the geometry of the unstructured mesh, to construct a face based data structure that contains, at each node of the face, two vector values and a coordinate. The two vectors could be any of velocity, vorticity, real-valued eigenvector vector, or swirl eigenvector [12]. Then a Newton-Raphson iteration is triggered for each face qualified by a pre-filter mechanism, to find the location where the two vectors are parallel to each other. Once each of the faces that contains a point is identified, the algorithm will qualify the points again with a post-filter, and connect all the points into lines.

This work focus on accelerate the Newton-Raphson iteration on faces, illustrated in Figure 2 as pseudo-code.

```
Retrieve grid connectivity, velocity and velocity gradient on nodes;
For all faces:
  For all nodes:
    Get coordinates;
    Get velocity values;
    Get velocity gradient and derive vorticity values;
  EndFor
  If face is qualified:
    Trigger Newton-Raphson iteration;
  EndIf
EndFor
```

# GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

**Figure 2: Pseudo-code for locating points on vortex coreline.**

The post-processing of the points extracted is not identified as a good candidate for acceleration due to its low arithmetic intensity. Interested readers are encouraged to refer to [13] for the detail of that part of the algorithm.

### 3.3: Open Inventor MeshViz XLM

The software framework that this work is based upon is Open Inventor MeshViz XLM. Open Inventor by Mercury is the most widely deployed commercial visualization toolkit. MeshViz XLM is a new Open Inventor extension for data extraction and visualization. It defines a generic mesh access interface which abstracted away mesh topology (structured or unstructured) and field type (single precision or double precision). A suite of data extraction routines are provided, such as isosurfacing and streamline integration. Memory management and performance are significantly improved over its predecessors, to allow processing large and complex datasets.

This work expands MeshViz XLM's data extraction offering, and lays down the foundation of the acceleration infrastructure for performance critical components. This infrastructure is based on many years of GPU computing experience at Mercury Visualization Sciences Group (formerly known as TGS Inc.), early with OpenGL's shader extensions and more recently with CUDA. Multi-core CPU or GPGPU acceleration will be triggered in a manner transparent to the application. Depending on device constraint or future hardware implementation, data mapping/transfer will be optimized to use best available techniques, without customer code changes. This infrastructure has been successfully deployed to the field extraction and visualization of volumetric data set, commonly seen in geoscience and medical fields.

It is also noteworthy that all functions available at Open Inventor SDK level to developers, are also exposed in the desktop application framework Avizo and available to end users.

## 4: IMPLEMENTATION DETAIL

A common nature among the two algorithms in Figures 1 and 2 are that the computation is highly independent at element level. For gradient reconstruction, the computation is done for each cell. For vortex core point extraction, the computation is done for each face.

This nature makes them good candidates for massively parallel computation.

CUDA supports a C-like interface, hence the core of the two algorithms studied in this work needs to be mapped to a C data structure. As the GPU

# GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

hardware just started supporting double precision, our implementation supports both single and double precision floating point calculation.

## 4.1: CUDA gradient reconstruction

For gradient reconstruction, the coordinates array of the nodes, the velocity vector of nodes, and cell-to-node connectivity are all placed in the GPU's global memory. Then, one computation thread is assigned to each and every cell.

## 4.2: CUDA vortex core point extraction

Likewise, one computation thread is assigned to each and every face.

## 4.3: Test data

To facilitate the validation of GPU results, the following linear flow is introduced,

$$\mathbf{u} = \begin{bmatrix} x - y \\ x + z \\ y \end{bmatrix} = (\nabla \mathbf{u}) \mathbf{x} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \mathbf{x} \quad (\text{Eq. 1})$$

where the gradient tensor is a constant matrix. Vorticity vector is consequently constant  $[0,0,2]$  as well. Hence the solution of vortex coreline algorithm in Figure 2 is the point where  $x=y=-z$ . The first two components velocity vector is zero along this line, and parallel to vorticity vector.

The flow field is of less importance in this performance study, hence the application of this work on industrial CFD/FEA solutions is deferred to another project. No assumptions were made in the data structure about the topology of the mesh, however, to make the benchmarking task easier particularly with existing studies, all meshes used in the following section is treated as tetrahedron only.

## 5: Results

As our work is targeted directly toward commercial applications, the acceleration strategy is measured specifically against the production code, developed in C++ and optimized to our best capability.

Realizing the facts that not all graphics hardware is supporting double precision (DP hereafter) and that MeshViz XLM is computing DP, a strategy is

## GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

implemented to convert the data to single precision (SP hereafter), and included this conversion time to the performance measurements.

All results are measured under Microsoft Windows operating system. The code is compiled with Microsoft Visual Studio C++ 2005, using CUDA 2.0. The results are obtained by executing the code in release mode.

Table 1 summarize the main features of the benchmarking hardware that will have an impact on our performance measurements. The CPU used for benchmarking is an AMD Opteron 250 Processor, with 3.5GB of RAM. The GPGPUs are connected with PCI-Express x16.

System	1	2
Nvidia GPUs	Quadro FX1700	GeForce GTX 260
Core clock (MHz)	460 MHz	576MHz
MEM (MB)	512	896
Max Memory Bandwidth (GB/S)	12.8	111.9
Compute capability	1.1	1.3 (DP support)
Parallel Cores	32	192

**Table 1: Summary of hardware systems used.**

### 5.1: Gradient calculator speed-up on different hardware

Gradient calculation results are benchmarked in Table 2. Figure 3 shows the speedup measured on two hardware systems in Table 1.

SP calculation has been the dominant mode in the literature, hence is our first point of discussion. On GTX260, an 8x speedup, measured against single CPU C++ product code, is observed, while the speedup is about 5x on FX1700.

The GPU acceleration with SP needs to go through the following process. First, the data from the C++ production code needs to be converted to SP. Second, the data on CPU host needs to be transferred to GPU device. Third, GPU calculation is performed. The first and second process have a reverse correspondence, that is, once the GPU computation is done, the data need to be transferred back to CPU host, and then converted back to DP.

For the sake of clarify, the following time components are defined:

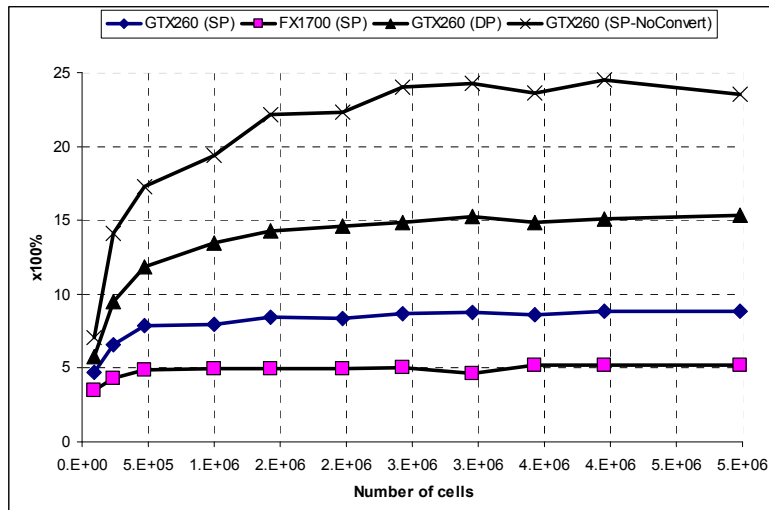
- *Core computing time*: the time GPU device actually performs parallel calculation.
- *Transfer time*: the time consumed in transferring data from host to device before and the reverse transfer after core computing.
- *Convert time*: the time consumed in converting DP to SP before and the reverse conversion after the core computing and transfer are done.

## GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

To understand better Figure 3, the percentage of each time component is plotted in Figure 4.a. It is noticeable that the convert time is a large portion of the total time consumed, and increases to over 60% with increasing cell numbers. Although for all curves, the performance increases with data sizes, due to the amortization of the cost of convert and transfer components, the dominance of the convert time drives largely the flattened speed-up at relatively small dataset size. If the converting component is excluded, the curves labeled “GTX260 (SP-NoConvert)” shows up to 25x speed up.

# cells	C++	GTX260 SP	FX1700 SP	GTX260 DP
93750	0.203	0.0428	0.05872	0.03524
235824	0.512	0.0775	0.11802	0.0541
477042	1.027	0.131	0.21127	0.0867
998250	2.118	0.266	0.4303	0.1571
1429968	3.057	0.3619	0.6127	0.2134
1971054	4.129	0.494	0.8328	0.282
2431344	5.143	0.594	1.0138	0.347
2958234	6.284	0.714	1.3644	0.411
3430722	7.27	0.842	1.3951	0.489
3951018	8.348	0.947	1.6068	0.553
4983504	10.553	1.191	2.019	0.689

**Table 2: Benchmarking data on gradient calculator.**

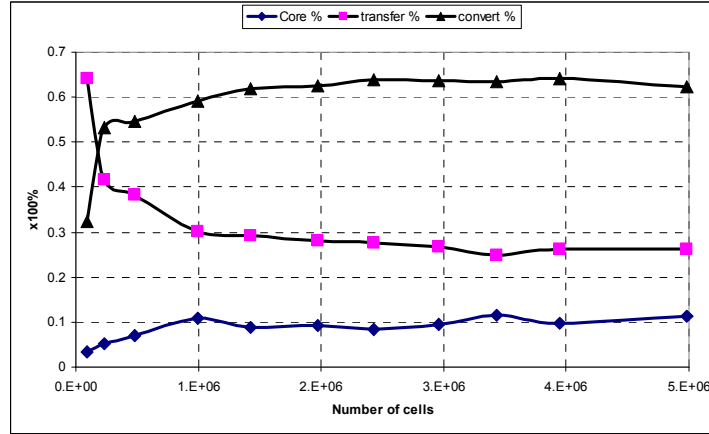


**Figure 3: Speed-up measurements of SP gradient calculator on two hardware systems.**

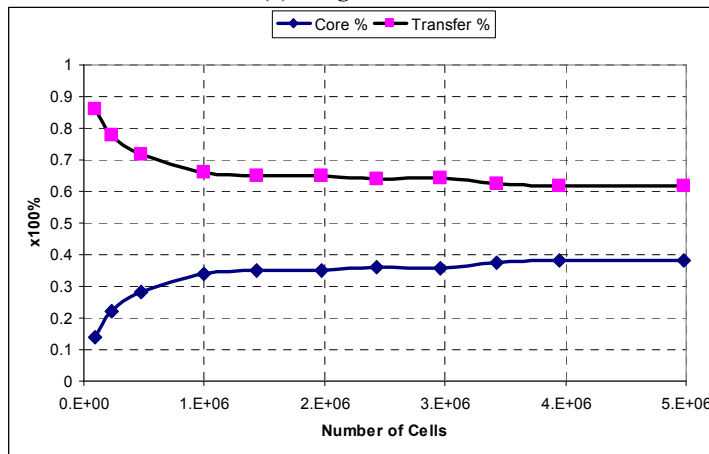
By studying the two hardware, it is concluded that CUDA compute capability 1.1 is not well-suited for unstructured grid calculations. For unstructured data it is almost guaranteed to have uncoalesced memory access. Memory bandwidth will consequently be reduced in an order of magnitude. CUDA compute capability 1.3 has greatly loosened the restrictions for coalesced memory access, which makes unstructured grid calculations effectively a more realistic

# GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

goal. This explains the big difference between two “(SP-NoConvert)” curves comparing to the difference between two “(SP)” curves.



(a). Single Precision



(b). Double precision.

Figure 4: Percentage of different time components on GTX260.

## 5.2: Double Precision

Running in DP improves the fidelity of the numerical solution, which is mandatory to the CAE community. Moreover, as is the case in our study, many numerical software have their data in DP format already. Hence there is no cost associated with the convert time. Figure 5 shows the DP result, speed-ups from 6x for smaller dataset, up to 15x for larger dataset, are observed.

Figure 4.b shows a gradually increasing core time portion and a gradually decreasing transfer time portion with the size of dataset, converged at about 40% and 60% respectively.

# GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

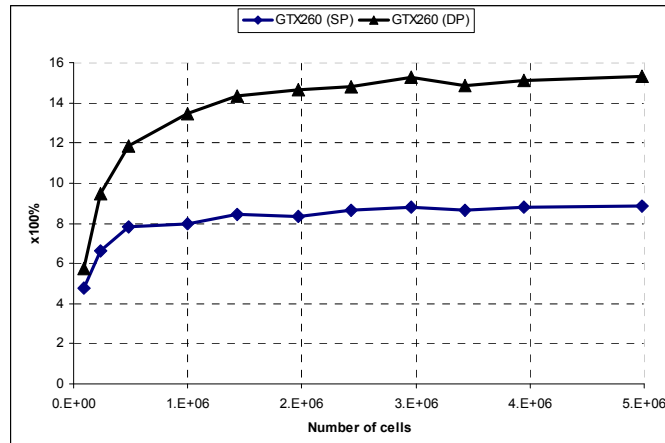


Figure 5: Percentage of different time components on GTX260. SP result from Figure 3 is repeated here for reference.

### 5.3: Vortex coreline algorithm speedup

For performance benchmarking purpose, the filtering mechanism is disabled to trigger Newton-Raphson iteration for every face. It is admitted a theoretical approach as in real applications, the faces that are not qualified should be skipped. The focus here is the acceleration of the Newton-Raphson iteration.

The CUDA version of the algorithm is not yet optimized, but initially we observed a speedup of approximately 3X using double precision computation on the GTX260 board. The performance improvement was relatively constant across the range of mesh sizes studied in this work (up to five million cells). This result suggests that the more complex algorithm may be limited by the double precision performance of current GPU hardware. More work is planned to explore the potential of GPU for this algorithm.

### 6. Concluding remarks and future work

We believe that GPGPU computing will have a significant impact on CAE simulations, based on our experience accumulated on GPU with graphics and visualization tasks. This work further strengthened our belief. Specifically, the speed-up that is achieved on DP in this work is very encouraging. It is reported by NVidia that the theoretical GFlops for DP and SP computations differs greatly. However, due to the random access to the device global memory, the penalty introduced by DP for unstructured applications could be less significant. We keep reasonable optimism that the algorithm designed in this work will scale with better DP GPUs.

## **GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA**

On the other hand, however, for algorithms entering production mode, it is important to take a “realistic” emphasis by considering if the GPU is valuable based on the total cost and effectiveness of off-loading CPU work, rather than just the speed of the computation. It is clear that GPGPU can not yet be used as a general purpose processor. The memory has to be pre-allocated, branching and random access is inefficient. Hence one should not move all the work to the GPU. Hence an acceleration strategy is designed in this work that is specifically aiming to use both the CPU and GPU according to their strengths.

In this paper, two algorithms are identified as suitable for GPU acceleration. Good speed-ups are obtained on multiple GPGPUs and in both SP and DP. In the meanwhile, it is worth noting that there are two categories of tasks that could be computation intensive, but not suitable for GPU. One is preparing the data structure suitable for massive parallel computing, the other is small computation on large data. In both cases, the benefit of computation efficiency available on GPU does not justify the cost of transferring data to and from the GPU. These CPU-suited tasks, however, can overlap with the GPU accelerated task and achieve overall performance improvement. This subject is left to a future study.

Moreover, in the future, the shared memory and texture memory available to multiprocessors should be considered to improve memory coalescing by eliminating bank conflicts. For unstructured mesh, it remains a challenge, particularly when a design is trying to avoid keeping separate algorithms and data structures for different devices.

A detailed look at the vortex coreline portion of the work also awaits further investigation.

This work will be available to the CAE community as commercial offering, in terms of both data extraction SDK (Open Inventor MeshViz XLM extension) and desktop visualization application (Avizo Wind Edition).

The authors acknowledge the exclusive support of Mercury Computer Systems, Inc., Visualization Sciences Group on this work. S.Z. is in-debt to insightful discussions with Andrew Corrigan.

# GPGPU ACCELERATION ON FIELD AND FEATURE EXTRACTION FROM UNSTRUCTURED SIMULATION DATA

## REFERENCES

1. AKELEY K - Stanford GPUBench Project, <http://graphics.stanford.edu/projects/gpubench/>, 2007.
2. Berkeley Dwarf Mine. - [http://view.eecs.berkeley.edu/wiki/Dwarf\\_Mine](http://view.eecs.berkeley.edu/wiki/Dwarf_Mine).
3. Brandvik, T & Pullan, G - Acceleration of a 3D Euler Solver using Commodity Graphics Hardware. 46th AIAA Aerospace Sciences Meeting, Reno, NV, Jan 2008.
4. CFD Online Wiki Page – [http://www.cfd-online.com/Wiki/Gradient\\_computation](http://www.cfd-online.com/Wiki/Gradient_computation), 2009.
5. CORRIGAN A., CAMELLI F., LOHNER R., and WALLIN J. "Running unstructured Grid Based CFD Solvers on Modern Graphics Hardware", Private Communication. Also [http://cde.gmu.edu/~jwallin/c702f08/csi702\\_cuda\\_presentation.pdf](http://cde.gmu.edu/~jwallin/c702f08/csi702_cuda_presentation.pdf), 2009.
6. Göddeke D. & Strzodka R. - Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations (Part 2: Double Precision GPUs), *Ergebnisberichte des Instituts für Angewandte Mathematik*, Nr. 370, TU Dortmund, 2008.
7. MAXIMO A., RIBEIRO S., BENTES C., OLIVEIRA A. & FARIAS R. - Memory Efficient GPU-Based Ray Casting for Unstructured Volume Rendering, Volume Graphics (2008), Hege HC & Laidlaw D. (Editors), pp. 1–8, 2008.
8. MERCURY VISULIZATION SCIENCES GROUP. Oil & Gas Application, Innovative 3D visualization solutions for Oil and Gas, NVidia CUDA Zone. Also see other examples and CUDA documents therein. [http://www.nvidia.com/object/cuda\\_home.html#state=detailsOpen;aid=7e814a19-5a09-470d-9752-e3ba25d1d51e](http://www.nvidia.com/object/cuda_home.html#state=detailsOpen;aid=7e814a19-5a09-470d-9752-e3ba25d1d51e)
9. SHIRLEY P & TUCHMAN A. - Polygonal approximation to direct scalar volume rendering, Proceedings San Diego Workshop on Volume Visualization, Computer Graphics, vol. 24(5), pp. 63–70, 1990.
10. SILVA C., COMBA J., CALLAHAN S., & BERNARDON F. - A Survey of GPU-Based Volume Rendering of Unstructured Grids, RITA, Volume X, 2005.
11. WYLIE B., MORELAND K., FISK L. A., CROSSNO P. - Tetrahedral projection using vertex shaders. Proceedings of the 2002 IEEE Symposium on Volume visualization and graphics, pp. 7–12, 2002.
12. ZHANG S. & CHOUDHURY D. - Eigen helicity density: A new vortex identification scheme and its application in accelerated inhomogeneous flows. Physics of Fluids, 18:058104, 2006.
13. ZHANG S., MEREU S., YAN J., CHOUDHURY D. - Extraction of vortical features in a turbomachinery application." 45<sup>th</sup> AIAA Aerospace Sciences Meeting and Exhibits, AIAA07-1285, Reno, NV, 2007.