An Object-Oriented Tool for Visualization and Debugging of Finite Volume Methods

Thomas SCHMIDT, Roland RÜHLE

Computing Center (RUS)
University of Stuttgart
Allmandring 30, 70550 Stuttgart
Germany
E-Mail: {schmidt,ruehle}@rus.uni-stuttgart.de

Abstract

Visualization tools are not only useful for displaying results of computational simulations; they also may serve as a graphical debugger for the development of numerical simulation applications. The object-oriented visualization tool presented here takes the additional requirements into account. It handles arbitrary polygonal and polyhedral grids. It is possible to visualize the internal solution data structures of the simulation program exactly. For finite volume methods this means the possibility to display higher-order reconstruction functions on the control volumes. Therefore, a simple interface allows the easy integration of the simulation data structures into the presented system. It consists of a predefined set of methods which can be defined by using the internal data structures of the simulation program. The visualization of additional debugging information is realized through the introduction of special user objects. Two examples are presented.

1 Introduction

The development and implementation of numerical methods for computational simulation are getting more and more complex. The target is to simulate real-life problems like the air flow around a complete airplane or inside a car (heating) as fast as possible. Bigger, faster computers are needed, but also new, more sophisticated simulation methods have been introduced during the last decade. Former methods worked only with structured grids, which are much easier to handle than unstructured grids. Here, a computational space is decomposed into a set of triangles in 2D space and into a set of tetrahedra in 3D, respectively. A further extension is to use a small set of predefined polygonal or polyhedral types. Some finite volume methods even use arbitrarily unstructured grids [SVD89]. Also adaptive grid refinement is frequently used. This leads to the problem that, during the development of a computer simulation application, not only the production of correct results must be checked, but also features like grid database, adaptive refinement and automatic grid distribution for parallel applications have to be checked.

The presented visualization system deals with finite volume methods on 2D or 3D arbitrarily unstructured grids. A 3D grid of this kind is defined as a set of arbitrarily-shaped polyhedra which build the computational domain. The polygons which build the polyhedra may also be non-convex. The example computational cell in Figure 1 consists of seven polygons, two of them are pentagons, the other five are quadrangles.

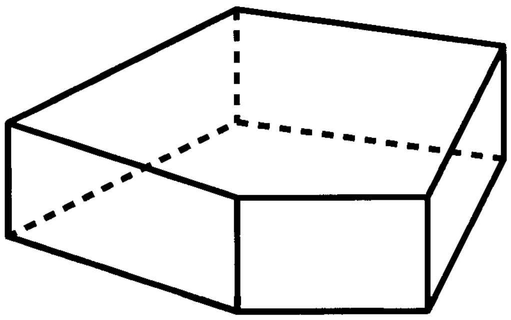


Figure 1: Example cell of a 3D arbitrarily unstructured grid

Some advantages of arbitrarily unstructured grids are: simplifies grid refinement and recoarsening, integration of the grid generation into the simulation process, no additional effort for complex geometries, and no grid degenerations in critical regions [HK94].

The presented system supports the verification of numerical simulation packages during the development process. Therefore, the numerical data structures can be integrated into the visualization system. The user defines evaluation functions for these data entities. The resulting scalars and vectors can then be displayed. Additionally, user-defined data can be visualized in order to identify possible numerical or adaptive grid refinement problems. Hence, this tool is one step towards a graphical debugger for the development of computational simulation systems.

The following sections give an overview about the object-oriented visualization system including a description of the data model. Then examples are presented, followed by a comparison with other approaches and a conclusion.

2 Data Model

For structured grids only few parameters must be stored, e.g. for an ashlar grid the width of the cells in each dimension. The data model for the results of arbitrarily unstructured grid methods is much more complex. This formal model consists of the description of the complex geometry and a set of solution functions for quantities.

The whole data model is dimension independent because the geometry is recursively defined. First, a node n_i is a vector of dimension dim and at the same time an element of the lowest hierarchy level H_0.

$$\begin{aligned} n_i &= (c_1, \ldots, c_{dim}) \\ H_0 &= \{n_1, \ldots, n_{nodes}\} \end{aligned} \tag{1}$$

Further hierarchy levels H_i are defined as a set of subsets M_k of the previous level.

$$\begin{aligned} M_k &= \{m_1, \ldots, m_l\} \text{ with } m_j \in H_{i-1} \\ H_i &= \{M_1, \ldots, M_{elements}\} \end{aligned} \tag{2}$$

A n-dimensional grid G is defined by a tuple of $n+1$ different hierarchy levels H_i.

$$G = (H_0, \dots, H_n) \tag{3}$$

The solution values (scalars and vectors) are obtained by an arbitrary number of evaluation functions. For a cell centered finite volume method all quantities are assigned to the centers of the elements of the highest hierarchy level. A reconstruction function f_i is used to calculate a quantity q at the position x within a cell C_i. The solution f on the complete computational domain D is composed with the functions of all n cells.

$$f(x) := \begin{cases} f_1(x) & \text{if} \quad x \in C_1 \\ \vdots & \\ f_n(x) & \text{if} \quad x \in C_n \end{cases} \quad \text{with} \quad x \in D \quad \text{and} \quad D = \bigcup_{i=1}^{n} C_i \tag{4}$$

A quantity may be a scalar or a vector. It follows that the solution over the whole computational domain must not be continuous because there may be discontinuities on cell borders. In Figure 2 a 2D example is shown, two neighbouring cells with their solution functions.

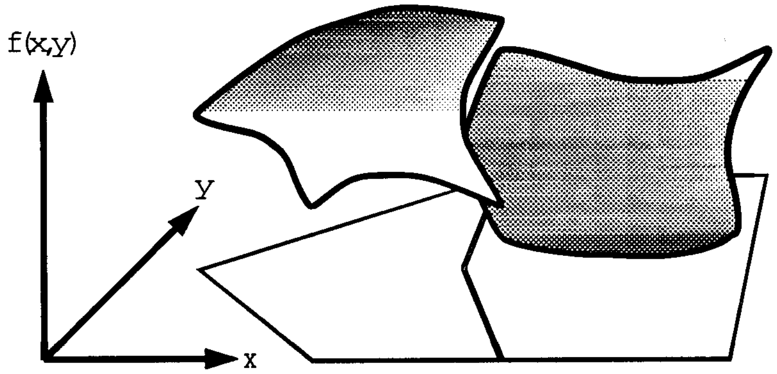


Figure 2: 2D example of a discontinuous solution

A model for the representation of solutions of finite element methods and the management of time-dependent data was presented in [SR95].

3 The Visualization System

3.1 System Architecture

The presented visualization system VIZARD (VIsualiZer of ARbitrary grid Data) was designed to meet the following demands:

- efficient handling of arbitrarily unstructured grids
- management of time-dependent data with changing grids
- visualization of solutions from finite volume methods
- support of piecewise continuous solutions
- integration of online visualization
- support of additional user objects for debugging purposes

- interactive solution steering
- easy connection of simulation and visualization

To fulfill these demands the architecture as shown in Figure 3 was chosen to connect the simulation and the visualization system.

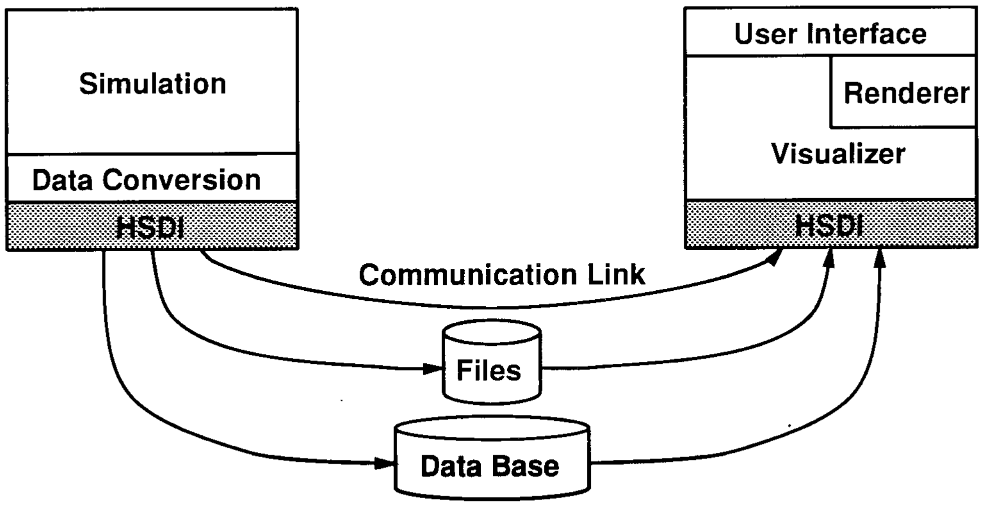


Figure 3: Overview of the system architecture

One part of the complete system is a simulation package which is connected to the visualization system using the Heterogenous Simulation Data Interface (HSDI). The components of the visualization system include a renderer; a visualizer, which performs the visualization methods; and the user interface, which can access the visualization methods and all rendered objects.

The visualization data is organized in a list of time steps. This way, visualization of time-dependent data is possible. Between two time steps linear interpolation of time is used. Other interpolation methods may be suitable for animations. Several time step lists can be managed at the same time. Thus, results of multi-grid methods or distributed grids of parallel applications can be displayed. For a detailed description of the management and the visualization of time-dependent data see [SR95].

Polygon lists generated by visualization methods or user objects created by the simulation program are then transfered to the renderer. Open Inventor™, an object-oriented 3D toolkit developed by Silicon Graphics, was used for the implementation of the renderer. This toolkit is supported by several workstation manufacturers and thus ensures portability.

3.2 The Object Oriented Interface

The Heterogenous Simulation Data Interface is an object-oriented library which was developed in C++. It provides methods for data conversion, transfer and file I/O. Using HSDI makes it possible to send data from the simulation directly to the visualization. Alternatively this data can be archived first and visualized later. Three main base classes exist: HSDI_Geometry, HSDI_Solution, and HSDI_UserData. Details of these classes are descibed in the following sections. Figure 4 shows the inheritance tree for some important HSDI classes. The notation is the one proposed by Booch [Boo94]. Not all interdependences are shown in order to keep the picture clear.

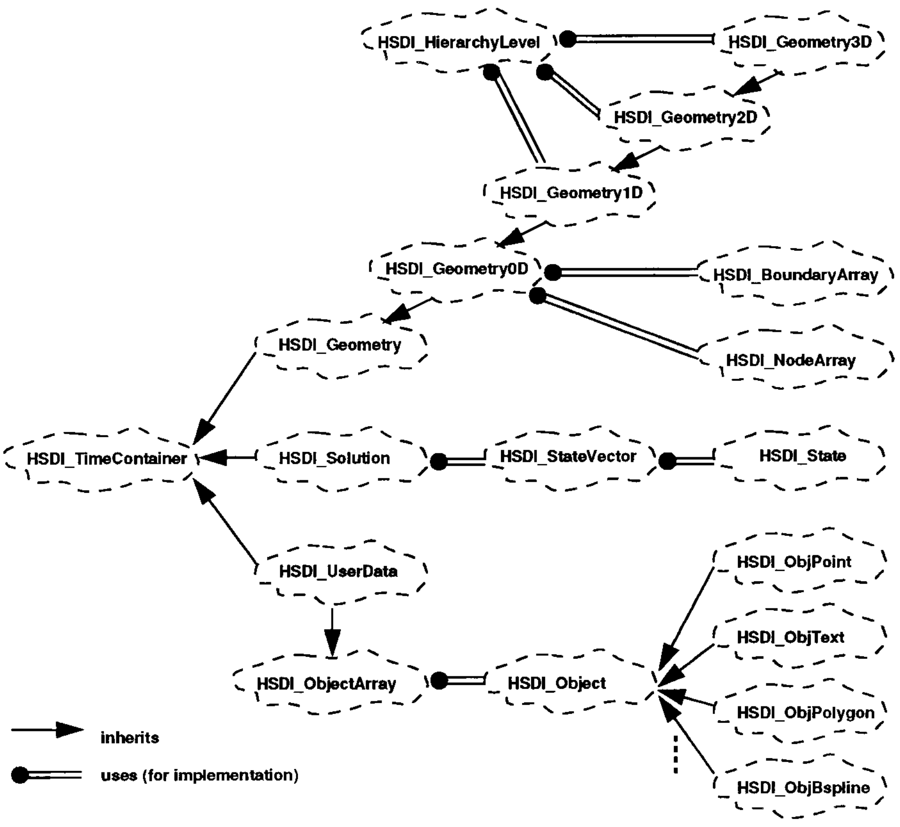


Figure 4: Overview of the main classes of the interface

3.3 Grid Data

The grid is constructed by calling appropriate methods for an object of a derived class of HSDI_Geometry. As derived classes, HSDI_Geometry0D, which contains only nodes and boundary elements, HSDI_Geometry1D, which additionally has edges, HSDI_Geometry2D with polygons, and on top of it class HSDI_Geometry3D with polyhedra are available. Edges, faces and cells are realized by the usage of the class HSDI_Hierarchylevel. Another hierarchy level is always added for the next dimension. If fact, it would be possible to further increase the number of dimensions because this data model is independent of the dimension .

The grid data object is built the following way: at first, all nodes are created by calling the method addNode(), which takes a coordinate vector as a parameter and returns the ID of the new node. If node IDs already exist, then this method must be called for all nodes in the correct order. After all nodes are added, edges can be built; just create a vector of IDs of the nodes which build the edge and pass it as an argument to the method addEdge(). The creation of faces and cells works analogously.

In order to keep the amount of interface data small, the upward connectivity information is created by the visualization system. It is used for efficient implementation of the visualization algorithms. The grid elements are stored in arrays, not as standalone objects, for performance reasons.

3.4 Solution Data

The numerical solutions of finite element methods are administrated by accessing methods of the class HSDI_Solution. Solution values of cells or nodes are added by calling the method addState(). In addition, boundary solution values can be added.

A predefined method interface of the class HSDI_State exists. This way it is possible to display solutions with higher-order reconstructions. The user of the library must provide the implementation of some of the class' methods. These methods should use the solution data structure of the numerical simulation. This is necessary to select evaluation functions and to compute scalar or vector values within a computational cell. These query methods have a position argument, allowing one to compute the value at any position within a grid cell. The needed methods are in specifically:

- new() and delete()
- getNumberOfScalarMethods()
- getNumberOfVectorMethods()
- getNameOfScalarMethod()
- getNameOfVectorMethod()
- setScalarMethod()
- setVectorMethod()
- getScalar()
- getVector()
- read() and write()

3.5 User Data

To support the development process and the debugging of a numerical simulation package, additional user data can be generated by the HSDI. This allows the display of additional information beyond the grid and solution data. Therefore, several derived classes of HSDI_Object exist. At the moment, the following classes of user objects are available:

- HSDI_ObjEmpty for test purposes
- HSDI_ObjPoint for one point
- HSDI_ObjLines for a set of lines
- HSDI_ObjPolygon for one polygon
- HSDI_ObjSphere for a sphere
- HSDI_ObjCone for a cone
- HSDI_ObjText for 2D text
- HSDI_ObjBspline for a b-spline surface

the shape and color of these objects are adjustable and they are displayed at the specified position within the grid by the renderer. This way cells can be marked in which special conditions occur. Flags used by the simulation for the computational cells can be displayed. Text can be used to provide additional information of the solution process. These are only some of the many possible applications.

These user objects will be used to steer the simulation process. They can be selected by clicking on them in the renderer. Then it is possible to send them back to the numerical program. The developer of the computational simulation may, for example, use HSDI user objects to allow the selection of regions for adaptive refinements.

3.6 Output Facilities

Data generated by the numerical simulation must be transfered to the visualization system. This can be done in three ways, by using either files or a database or a direct communication channel. The described interface provides the appropriate classes (see Figure 5).

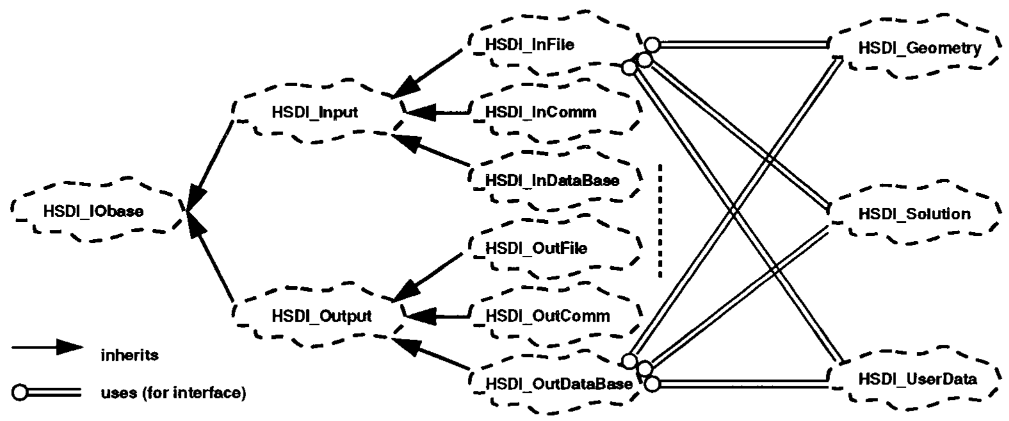
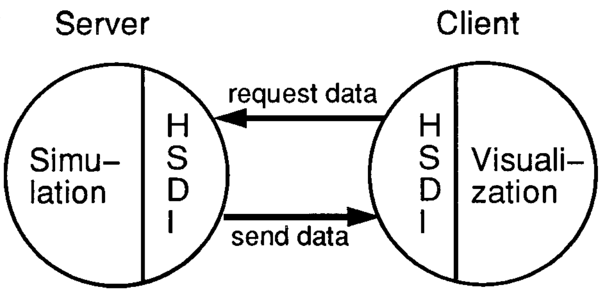


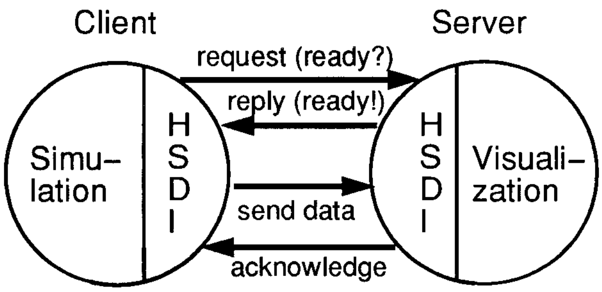
Figure 5: Class diagram showing output organization

An instance of the virtual base class HSDI_Output can either be an object of the class HSDI_OutFile for file output or of the class HSDI_OutPort for direct communication or of the class HSDI_OutDB for database access. This way an easy switch between file output, database access and direct communication is possible. Possible output objects are instances of the classes HSDI_Geometry, HSDI_Solution and HSDI_UserData.

For the online simulation a client/server protocol is supported by appropriate methods of the HSDI. Two modes can be realized as shown in Figure 6. The first one is the *independent* mode. Only if the visualization demands data it is sent by the numerical simulation. The second mode is the *tracking* mode. The simulation may send data at any time if the visualization is ready to receive.



(a) tracking mode



(b) independent mode

Figure 6: Possible communication scenarios for online visualization

3.7 Visualization Methods

As described in [FH94], the visualization methods such as boundary selection, cutting planes, color mapping, vector display and isolines/isosurfaces can be combined. One of the system's purposes is to allow the display of a solution function within a cell. Therefore, and to display isolines/isosurfaces, the grid cells are triangulated. The Delauny triangulation is used. Procedures to gain Delauny triangulations are readily available in the literature (e.g. [LS93]). Within a triangle or a tetrahedron linear interpolation is used to generate isolines/isosurfaces. If a closer look at a cell is desired, it is possible to do further refinement on the triangular/tetrahedral cells. The introduction of an additional grid point requires no interpolation because the available solution function can be used to get numerically exact values of the quantities. In contrast to the use of the marching cube algorithm, no problems with ambiguities occur (e.g. see [Mat94]). A more detailed desciption of the visualization methods can be found in [SR96].

4 Visualization Example

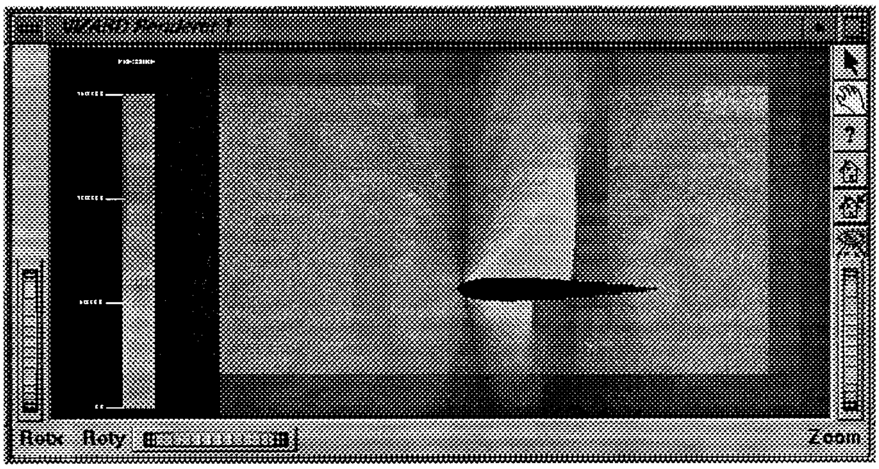
In [HBK95], a parallel euler-solver for highly unstructured grids based on arbitrary control volumes (CEQ) is presented. The results of this solver are transfered to the presented visualization system by using the HSDI. This is the reason why in a first step the visualization system was developed for the needs of visualizing fluid flows. As a special feature (CEQ) works on computational cells which may contain holes. Hence, it was necessary to implement a conversion method which disassembles these cells into cells without holes.

The visualization examples given are Figure 7 is from an inviscid flow around the NACA 0012 airfoil at mach 0.8 with an incident of 1.25 degrees (2D). In Figure 7 (a), the color-encoded pressure values are shown (dark - high, light - low). A detail of the grid is shown in Figure 7 (b). The numbers were generated by the simulation program as HSDI user objects. They indicate the refinement level on which the edges were created.

5 Other Approaches

Commercial visualization systems are designed for the visualization of results of finite element methods. Within IRIS Explorer™, for example, the pyramid data type allows one to create arbitrary control volumes, but the computed quantities are connected with the nodes. New quantities may be calculated from them. Within the cells interpolation takes place. This may lead to differences between the correct numerical values and the displayed ones, such that no visualization of finite volume methods is possible.

In contrast to an object-oriented approach, a functional one is also possible. A very good example is GRAPE (GRAphics Programming Environment) [GR91]. It is a flexible tool for the visualization of finite element methods on triangular and tetrahedral grids. Predefined but arbitrary grid elements were introduced in [RSS95]. Handling of unsteady simulations is integrated. Management of time-dependent data as described in [PR94] influenced the presented work. The possibility of online visualization was recently introduced using the net-manager concept [OP95].


(a) Color encoded pressure values

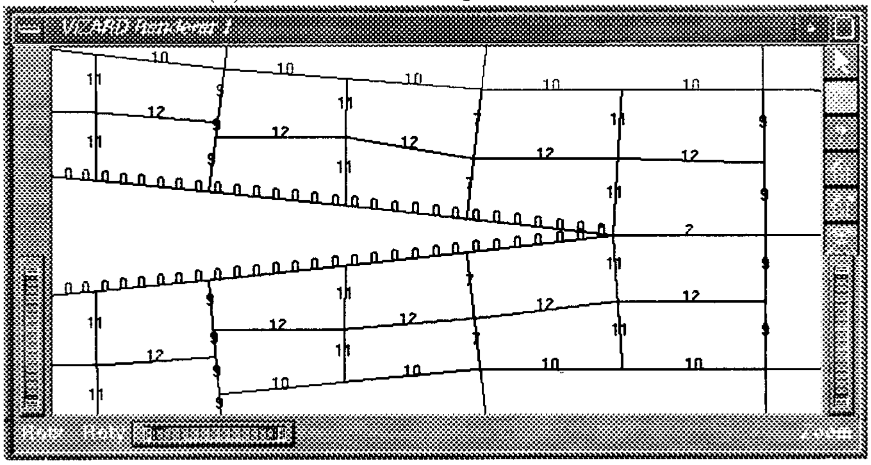

(b) Detail with text user objects

Figure 7: Inviscid flow around the NACA 0012 airfoil

6 Conclusions

An object-oriented visualization tool for finite volume methods on arbitrary unstructured grids was presented. The solutions may be only piece-wise continuous – no other visualization system is able to handle this kind of data. Using the **Heterogenous Simulation Data Interface (HSDI)**, an easy connection of a numerical simulation and the visualization system can be made. This allows the integration of the visualization and a simulation application. Also mechanisms for interactively steering the simulation are suggested. Examples were presented.

References

[Boo94] Grady Booch. *Object-Oriented Design With Applications, 2nd Edition.* The Benjamin/Cummings Publishing Company, 1994.

[FH94] James M. Favre and James Hahn. An object oriented design for the visualization of multi-variable data objects. In Harold P. Santo, editor, *Proccedings of the Visualization '94*, pages 318–325, Washington, D.C., October 1994. IEEE

Computer Society Technical Committee on Computer Graphics in cooperation with ACM/SIGGRAPH, IEEE Computer Society Press.

[GR91] M. Geiben and M. Rumpf. Visualization of finite elements and tools for numerical analysis. In *Proccedings of the Second Eurographics Workshop on Visualization in Scientific Computing*, Delft, Netherlands, April 1991.

[HBK95] C. Helf, K. Birken, and U. Küster. Parallelization of a highly unstructured euler-solver based on arbitrary polygonal control volumes. In N. Satofuka, J. Periaux, and A. Ecer, editors, *Proceedings of the Parallel CFD '95 Conference*, Pasadena, USA, 1995. Elsevier.

[HK94] C. Helf and U. Küster. A finite volume method with arbitrary polygonal control volumes and high order reconstruction for the Euler equations. In S. Wagner, E.H. Hirschel, J. Périaux, and R. Piva, editors, *Proceedings of the Second European Computational Fluid Dynamics Conference*, Stuttgart, Germany, 1994. Wiley & Sons.

[LS93] Michael K. Loze and R. Saunders. Two simple algorithms for constructing a two-dimensional constrained delauny triangulation. *Applied Numerical Mathematics*, 11:403–418, 1993.

[Mat94] S. V. Matveyev. Approximation of isosurface in the marching cube: Ambiguity problem. In *Proccedings of the Visualization '94*, pages 288–292, Washington, D.C., October 1994. IEEE Computer Society Technical Committee on Computer Graphics in cooperation with ACM/SIGGRAPH, IEEE Computer Society Press.

[OP95] B. Oberknapp and K. Polthier. A simple concept for distributed computing in computer graphics. In D.W. Fellner, editor, *Modelling - Virtual Worlds - Distributed Graphics*, pages 26–35. Infix, 1995.

[PR94] K. Polthier and M. Rumpf. A concept for time-dependent processes. In *Workshop Papers of the Fifth Eurographics Workshop on Visualization in Scientific Computing*, Rostock, Germany, May/June 1994.

[RSS95] M. Rumpf, A. Schmidt, and K.G. Siebert. On a unified visualization approach for data from advanced numerical methods. In *Proceedings of Sixth Eurographics Workshop on Visualization in Scientific Computing*, Chia, Italy, May 1995.

[SR95] T. Schmidt and R. Rühle. On–line visualization of arbitrary unstructured, adaptive grids. In R. Scateni, J. van Wijk, and P. Zanarini, editors, *Visualization in Scientific Computing '95*, pages 25–34. Springer, Wien, New York, 1995.

[SR96] T. Schmidt and R. Rühle. Visualization of finite-volume methods on arbitrary unstructured grids. In Harold P. Santo, editor, *Compugraphics '96 Conference Proceedings*, Bonneuil-sur-Marne, France, December 1996. GRASP.

[SVD89] R. Struijs, P. Vankeirsbilck, and H. Deconinck. An adaptive grid polygonal finite volume method for the compressible flow equations. Meeting Paper AIAA-89-1959-CP, American Institute of Aeronautics & Astronautics, 1989.