Visualizing Data from Time-Dependent Adaptive Simulations

R.–T. Happe, M. Rumpf, M. Wierse
Universität Freiburg

K. Polthier
TU Berlin

Abstract

Many interesting effects in scientific computing, such as shocks, flames, or vortex cores, are local in space and move in time. Recent numerical methods resolve these fine scales by adaptive meshes. We present a visualization approach to the handling of time-dependent data on grids with varying zones of refinement. This includes interactively working on arbitrary interpolated time cuts and an appropriate framework for vector field integration and structures for sceneries of icons representing tensor information at moving points of interest. We especially emphasize the relations between the numerical algorithms and visualization tools.

This text picks up, as a starting point, the concept developed by the authors in a previous paper and discusses applicability and extensions in the context of adaptive simulations.

1 Introduction

In PDE models of natural phenomena, most of the interesting effects correspond to certain nonlinearities. Many of them, such as shocks, cracks, flames, melting zones, vortex cores, are local in space and move in time. Numerical methods which are based on a fixed spatial discretization meet with serious difficulties in resolving all the important scales of the solution. Therefore, especially in the last decade, powerful adaptive methods on unstructured grids (e.g. triangular or tetrahedral meshes) have been developed [2, 11, 20], that, by use of local refinement and coarsening [4, 5, 27], capture local phenomena with a moderate total amount of unknowns.

When we try to understand visualization results, debug a numerical code visually, or want to extract interesting features of the global geometry of the calculated solution, advanced visualization comes into play. Here the flexible and interactive handling of numerical data sets is an essential ingredient of an efficient visualization environment [3, 14, 26, 38, 39]. A variety of widespread visualization tools is currently in use to visualize simulation data [1, 10, 19, 34]. Object oriented concepts are taken into account to structure visualization environments [12, 32, 36]. Unfortunately, for numerical methods which adapt the underlying grid and the time step width, the support from most of the well-known visualization packages is quite limited. One has obtained faster and more efficient simulations with better error control, but at the price of complex adaptive meshes where discretization changes with time. At the first glance it seems to be unclear how to set up an effective interface for the visualization of the latter type of data. We have tried to fill this gap and have developed a concept to handle arbitrary time-dependent processes in an interactive graphical environment. Such a process is no longer taken for a more or less dense list of keyframes, each containing the solution at a distinct time. We deal with the entire process as one *time–object* [30]. In the sense of OOP, *time–object* is an abstract expansion of any type of stationary *object*, i.e. a subclass of some stationary class. It represents a continuous one parameter family of objects of the initial type.

Adaptive simulation data we mainly consider here also form such a family of objects and the result can be mapped to a specific *time–object*. In general, the data base consists of a number of time steps. With an appropriate interpolation scheme, the simulation process can be evaluated at any time in between. This scheme is closely related to the adaptive numerical calculation and takes into account changes in discretization. However, the concept of *time–objects* is not restricted to handling the whole simulation process. Popular CFD expedients like particle traces [7, 17, 24, 28], stream surfaces [18, 42, 41] or moving test sets can again be regarded as time-dependent processes which fit well into our setting. It then depends on the display style for the specific *time–object* whether we visualize the moving particles, curves and surfaces, or their traces. In [25], local characteristics of the flow are visualized by placing specific icons at points of interest. Icons are also used by [13, 16] to visualize critical points together with some information on the underlying tensor fields. A set of icons moving in time and undergoing modification will be an additional example of extracted features forming dynamic processes.

Furthermore it should be possible, although it's still a lot of work, to handle topological information contained for instance in hyperstreamlines [9], vortex cores [6] or the boundaries of topological regions [15, 16] in a similar manner.

Our concept has been implemented in GRAPE [33], an object oriented environment, developed at the SFB 256 at Bonn University and at the Institute for Applied Mathematics at Freiburg University [31, 40]. And yet, this presentation should underline that the methodology introduced here can easily be ported to other environments. Although the object oriented approach is the essential ingredient, we shall not go into details, but hope for an intuitive understanding of message passing, late binding and inheritance in the background.

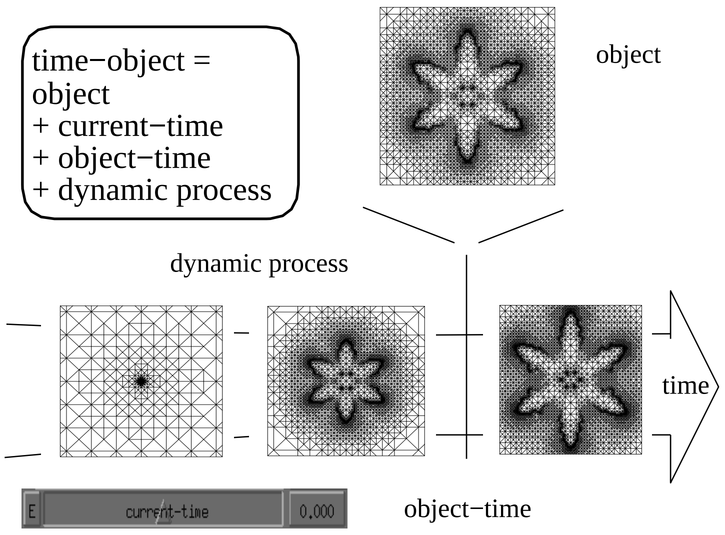


Figure 1: A sketch of a *time–object* and the underlying dynamic process representing the simulation data. Here the growing of a dendrite is calculated on an adaptive triangular grid (A. Schmidt). The grid structure depicts the moving transition zone between the two phases.

The outline of the paper is as follows. First we will introduce the concept of *time–objects* and describe how to work with them in an interactive environment and with respect to animation purposes. The next section is concerned with the structure of the dynamic process supervised by a *time–object* when the underlying data is an adaptive time-dependent simulation. Next, it will be pointed out that the integration techniques on CFD data fit into this framework, and an appropriate integration scheme will be presented that takes into account the properties of an adaptive solution on an unstructured grid. Finally, we consider local probing at possibly moving positions in the simulation domain.

2 Adding Time-Dependency to Objects

In this section we will briefly introduce our concept of *time–objects*. More details can be found in [30]. Let us first refer to a simple but fundamental observation. Besides an appropriate projection and shading, it is mainly the interactive rotation, which makes an object appear three-dimensional on a flat screen. This can be transferred to time-dependent objects. The more flexibly we can manipulate and animate time and study arbitrary time cuts of dynamic processes, all the better we will understand the process as a whole. So it seems natural

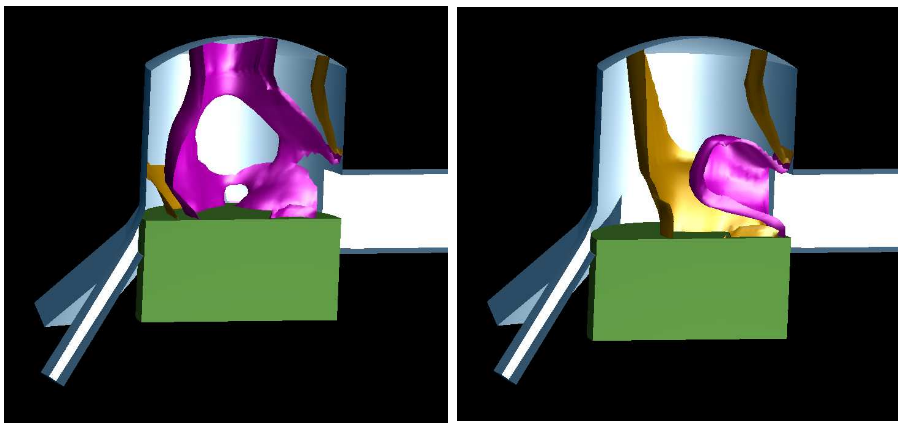


Figure 2: Isosurfaces of the pressure of compressible gas flowing in a two stroke engine with a moving piston at two arbitrary times. The calculation is based on an adaptive and moving tetrahedral finite volume mesh (M. Wierse).

to build this flexibility directly into the corresponding structures, which we will call *time–objects*. In the frame discussed here, objects which depend on time may be points, curves and surfaces (for an application in the context of simulation data see section 4), or functions of finite element or finite volume type on an unstructured adaptive grid, for instance a tetrahedral mesh. Any of these objects can be extended to a *time–object*: We add a reference to a *dynamic* (process), an *object-time* and a *current-time* and define thereby a subclass of the object's class (cf. Fig.1).

The object (or ancestral part) in such a *time–object* no longer represents a stationary data set, but a time cut of the underlying process *dynamic* at a time which is given by *object-time*. The variable *current-time* will in general be linked to some interactive control mechanism, for instance to a slider in a control panel. A typical structure for the process data *dynamic* (which is up to now a black box) will be inspected in the next paragraph. By the inheritance mechanism of OOP any previously implemented display method on the original object will immediately be available on the corresponding *time–object*. We achieve the necessary synchronization of the *object–time* with the *current–time* by the following simple update mechanism, which is invoked every time a message is sent to a *time–object*. It guarantees that always the current time cut of the dynamic process will be displayed.

```
if (current–time != object–time) {
      object = (dynamic ← "get–object")(current-time)
      object–time = current–time;
}
```

where *(dynamic $\leftarrow$ "get–object")* stands for the sending of the message *"get–object"* to the process *dynamic*.

At the level of the *time–object* there is only one thing essential for a correct working of this concept: The method *get–object* has to be implemented on the particular type of *dynamic*.

In GRAPE [33], our concrete environment, objects can be organized in a hierarchical structure so as to build a scene. The subscenes of this structure may represent time-dependent data. Therefore the underlying class *scene* has been enlarged to a *time–scene* as described above. The different *current–time* variables can now be controlled separately or can be synchronized. For that purpose, *time–scenes* additionally hold information on whether to synchronize with a global time or not. The actual synchronization is done by sending a synchronization message to the hierarchy. Then its *time–scenes* may align their respective *current–times* with the global time.

We've got two display strategies on a *time–object*. Applying some inherited method working on stationary data, we can flexibly display time cuts of the process (cf. Fig. 2). In addition, display messages can directly be sent to the dynamic process itself. For some important examples we refer to section 4.

In interactive visualization, control panel components such as sliders, steer display parameters, positioning and rotation. Time-dependent geometric objects demand parameters which are time-dependent as well, i.e. a function over time. For example, cutting planes in a three-dimensional geometry with moving local phenomena such as vortex cores should move in accordance to the moving effects. We suggest to use *time–sliders*: By editing a spline representation of the attached function, we can modify the slider value over time. Visualization methods request for the current parameter value by sending a *"get–object"* message to the *time–slider*, just as in the case of geometric objects (cf. Fig.3). The synchronization of control panels and geometry finally results in an arbitrary animation which can be adjusted interactively.

3 Interpolating Adaptive Data

Let's now turn to the question of how to handle dynamic processes consisting of adaptive simulation data.

Numerical simulation is in general a calculation of a list of successive time steps. On each time step the solution is given as a function on a mesh, which is evaluated best by a call $u(e, c)$, where e singles out an element, and c is a vector of local coordinates. For triangular or tetrahedral meshes we use barycentric coordinates as local coordinates, because most of the finite element functions, notably those of Lagrangian and Hermitian type, can easily be expressed in these coordinates [8, 22].

The above mentioned *"get–object"* message requests for an interpolated solution at a certain time t in between two time steps t_1 and t_2. With the mesh

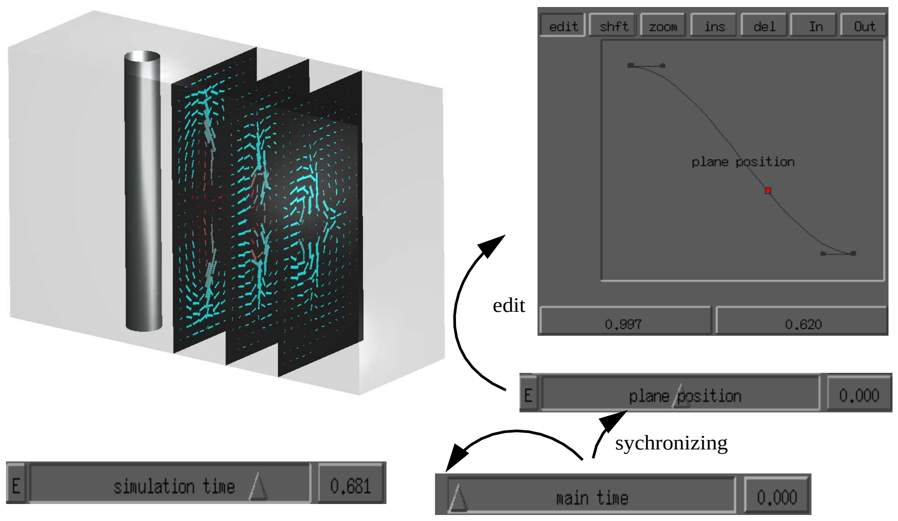


Figure 3: A time–slider for the simulation time and a time–slider for the position of the cutting planes are synchronized with a main–time. The dependency on the main–time is described by an editable spline. Here the non-stationary viscous flow behind a cylindric obstacle (E. Bänsch) is visualized by plotting the velocity field on cutting planes. The colour represents the normal component.

fixed, it is obvious how to interpolate the two functions u_1 and u_2 that represent the numerical solution in t_1 and t_2:

$$u(e,c) = \frac{t_2 - t}{t_2 - t_1} u_1(e,c) + \frac{t - t_1}{t_2 - t_1} u_2(e,c)$$

In the case of an adaptive strategy, the domain discretization will change from time step to time step. Therefore the function argument (e,c) no longer corresponds to the same point on the two meshes, and their seems to be no straightforward interpolation mechanism. To bridge this gap, let us first briefly review the basics of an adaptive numerical strategy [21]. In a more abstract setting the partial differential equations governing the physical process under consideration turn out to be an ordinary differential equation in function spaces [29]

$$\frac{\partial}{\partial t} u(t) \quad = \quad A(u(t), t)$$

In addition u has to fulfill appropriate initial and boundary conditions. Here A is a nonlinear operator depending on the specific problem; e.g. applying the projection onto divergence–free functions Q to the Navier-Stokes system $\partial_t v + (v \cdot \nabla) v + \nabla p - \Delta v = f$, $\nabla \cdot v = 0$ for fluid velocity v and pressure p with the fixed force term $f = f(x,t)$, one just has to let $u = Qv$, $A(u,t) = Q\Delta - Q(v \cdot \nabla v) + Qf(\ ,t)$.

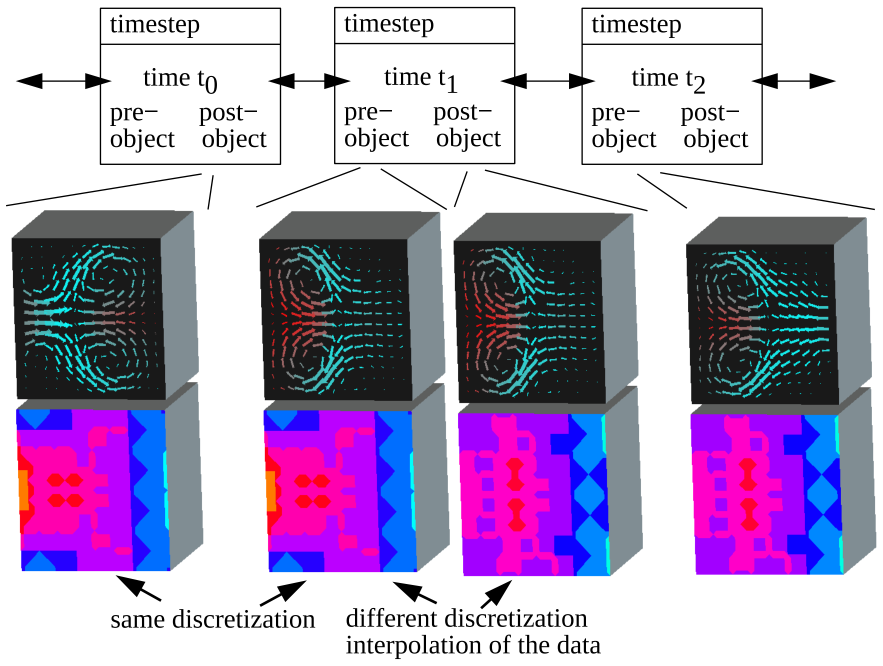


Figure 4: Three time steps of a viscous flow in a rectangular container on an adaptive tetrahedral mesh (E. Bänsch) are depicted. Plane cuts with a mesh of vectors and a colour coding of the degree of refinement are drawn. The mesh width decreases from light blue to yellow.

Adaptive solving now means to apply the following modification of some numerical ODE scheme. Given the post-adaption numerical solution u_{post} on a grid G_{post} at time t, we calculate

$$\begin{aligned}
\Delta t &= \text{TimeStepEstimate}(u_{post}, G_{post}, t)\\
u_{pre} &= \text{OneStep}(t, \Delta t, u_{post}, G_{post})\\
G_{pre} &= G_{post}\\
G_{post} &= \text{AdaptMesh}(u_{pre}, G_{pre})\\
u_{post} &= \text{Projection}(u_{pre}, G_{pre}, G_{post})\\
t &= t + \Delta t
\end{aligned}$$

Here *TimeStepEstimate* returns the new step length; *OneStep* is a numerical scheme for the calculation of one time step, e.g. a backward Euler step [37]; *AdaptMesh* does the refining, coarsening or deforming of the current mesh according to heuristic criteria or a-posteriori error estimates; finally, the solution is transposed onto a new mesh by a *Projection* operator. At each time step this results in two pairs of solution and grid. Let us store the one as *pre–object* and the other as *post–object* in a structure *time–step* representing a time step at a certain *time*. Then we build a doubly linked list of these structures and end up with the appropriate data structure to settle a general interpolation mechanism (cf. Fig.4). We realize that the *post–object* of a time step and the *pre–object* of the successive time step are based on the same mesh, while *pre–object* and *post–object* of a single time step differ in the underlying grid, although the function is the same up to the projection. Therefore, on a list of *time–steps*, the message *"get–object"* causes a search for the right time interval; then the standard interpolation formula is applied in between *post–object* and *pre–object* of the two *time–steps* bounding the specific interval. Now we are able to inspect time dependent adaptive data interactively. In our experience, not the whole sequence of calculated solutions is necessary to achieve a continuous impression of the ongoing physical process. An a lot sparser list of steps will suffice.

Let us finally remark that up to medium sized applications it might be possible to store the list of time steps in memory. For large data sets, one may replace the two objects of each time step by references to files on some disk. Then the "get–object" call will first have to load the requested objects and afterwards start the interpolation procedure. This approach can be improved combining both, direct storing and referencing, by a swapping mechanism for the time intervals currently of interest.

4 A Framework for Vector Field Integration

Integration techniques such as particle tracing or stream surfaces are powerful methods helping to understand global aspects of a CFD simulation. They calculate the transport of certain point sets in the velocity field. Such methods again fit in our concept of dynamic process: We calculate *time–objects*, where objects are single particles, clouds of points, curves or surfaces. The initial objects just examined are released interactively by picking or positioning. The corresponding dynamic processes are lines in space–time or lists of *time–steps* of clouds, curves etc. In general, the step length of the actually stored evolution may be much larger than the step length of the integration scheme.

Depending on the curvature of the transported curves and surfaces, their discretization has to be locally adapted. Now, the integration machinery runs analogue to the numerical scheme presented in section 3. A "get-object" message will cause the extracted processes to procure the transported test sets at the current time. Thus the new processes gained by integration can be examined interactively in quite a similar way as the numerical data itself (see section 2).

Furtheron, drawing particle paths or stream surfaces is only a specific display method on the extracted processes themselves. Depending on two parameters, an *end time* and a *start time*, the trace of the evolution in between this time interval can be visualized. The method needed for the meshing of a stream sur-

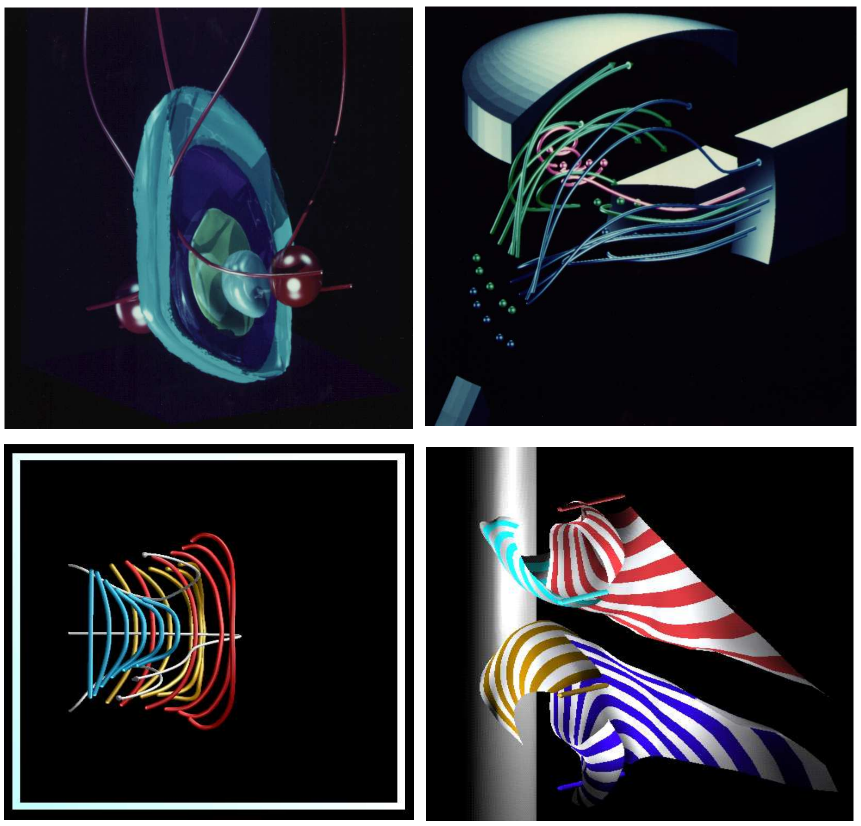


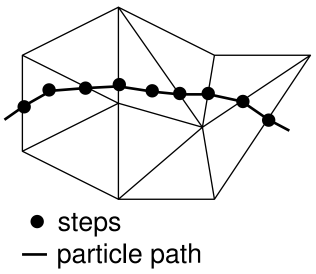
Figure 5: Different extracted dynamic processes with different display styles: The deformation of test spheres starting at two inlets and splashing against each other, particle traces in a two stroke engine, moving curves in a flow volume, stream surfaces displaying the evolution of curves in the flow behind a cylindric obstacle.

face picks up ideas already presented in [18] to generate a triangulation joining the polygon lines of two time steps.

5 Algorithmic Aspects of Integration

On adaptive meshes with changing discretization, care has to be taken when using integration schemes. Let us now focus on several important aspects. Schemes like the Runge–Kutta or the Euler–Cauchy scheme are frequently used for particle tracing and related methods [23, 35]. Assuming smoothness of the data, they are of second or higher order with a consistency error of third order. But in most finite element and finite volume strategies in CFD, the resulting velocity field is at most C^0, or even discontinuous at the boundary faces of the elements; only in the interior of an element is the solution smooth. These features stand out in areas of rapidly changing data, such as shocks in the physical solution, especially if they have been resolved by the CFD algorithm by intensive local refinement. It's difficult to handle them with a standard scheme. Note that even a step size control based on information about the extracted path does not necessarily take accurate account of discontinuities in the data. The following modified Euler–Cauchy scheme for a single integration step overcomes this problem. It first shortens the integration step size Δt such that the time when the discretization changes again won't be crossed. Then it correctly picks up all element boundaries in space:

$$\begin{aligned}
\tilde{x}(s) &= x_0 + sv(x_0, t_0) \\
s^* &= \min\{\Delta t, \{s > 0 | \tilde{x}(s) \in \partial E\}\} \\
\hat{x}(r, s) &= x_0 + rv(x_0, t_0) + \\
&\quad \frac{r^2}{2s}(v(x_0 + sv(x_0, t_0), t_0 + s) - v(x_0, t_0)) \\
r^* &= \min\{\Delta t, \{r > 0 | \hat{x}(r, s^*) \in \partial E\}\} \\
x_1 &= \hat{x}(r^*, s^*) \\
t_1 &= t_0 + r^*
\end{aligned}$$



Here x_0 is the particle position at time t_0 in an element E, $v(x, t)$ the velocity in the current element at a position x and at time t , and x_1 is the particle position at time t_1. At a boundary face the algorithm switches from one element to the adjacent element. A direct but tedious calculation proves that this scheme is consistent of the order $O(\min\{h, \Delta t\}^3)$ where h is the local grid size, and that it integrates fields of piecewise linear velocity accurately. Let us remark, that the case where the velocities in two adjacent elements are of opposite common direction needs some extra effort.

Furthermore, it is useful to keep track not only of the world coordinates but also of the pair (e, c) of element and local coordinates for each particle position. As already mentioned in section 3, this is essential for an efficient evaluation of the numerical velocity field. Passing the boundary face of an element on a fixed mesh, as it occurs in the above scheme, implies a simple shift in the coordinate vector. But each change of the discretization entails a search for the right element and local coordinates on the new mesh. The following rules guarantee that this can still be done efficiently:

- Try to avoid global searching operations on the whole mesh. Start at a guess in position (e, c) on the new mesh and keep track of the elements passed along a line to the current position x.
- For a single particle, start search in position (e, c) relative to the old mesh. If the local adaption area is not met the new pair (e, c) will in general coincide with the old one.
- Advance the points of moving sets such as curves or surfaces step by step as a whole. When switching to a new mesh locate the new position (e, c) of some point in the set. Take this for a guess at the position of its neighbouring points. Having found them, go on recursively.

6 Critical Points and Local Probes

Local properties of, say, a flow field v and the related tensor field Dv may be rendered by geometric symbols, which are placed at points of interest (critical points, vortex cores etc.) [13, 16, 25]. *Icons* assemble the local mathematical data and display–directives; a set of icons is gathered in a linked list. Naturally, a time-dependent field requires mobile icons (cf. Fig. 7) that e.g. drift along particle paths or stay in the center of a moving vortex, that arise, end, split and merge just as the inspected local phenomena do. Such dynamic processes may be represented by *time–icons*: a tree describes the evolution of a flock of *icons*. At each time-step we have a list of icon nodes; each genuine node represents a particular icon at a particular time (cf. Fig.6);
it contains

- the mathematical data t, x, $v(t, x)$, $\nabla v(t, x)$ and $\partial_t v(t, x)$. The time derivative is required for icons based on characteristics of the particle path $p(t + dt) \approx x + dt\, v + \frac{1}{2} dt^2 (v \cdot \nabla v + \partial_t v)$ (cf. [25] for the stationary case);
- directives regarding the render style;
- a link to its immediate temporal successors in the time evolution (if the node represents an icon that splits in t it has several successors). On the other hand, the node may be successor of several previous nodes that merge into it;

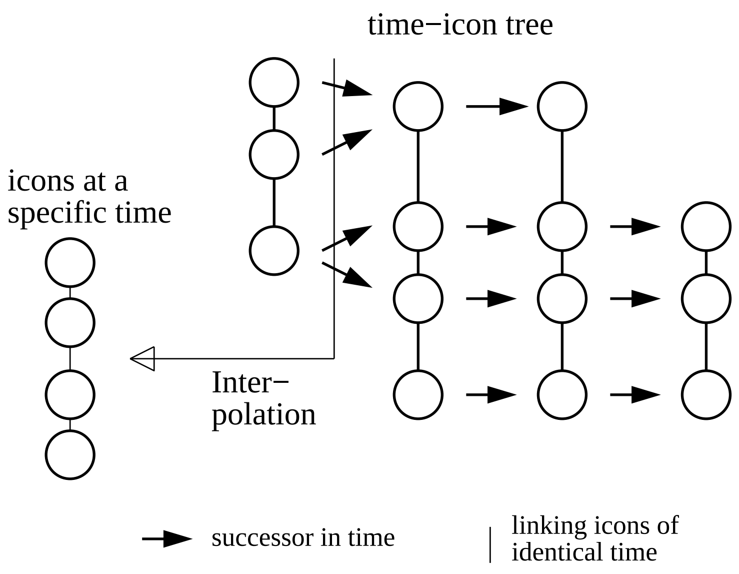


Figure 6: A sketch of the data structure holding information on time-dependent icons. Each node corresponds to a particular position in space and time; columns of nodes form time cuts; horizontally linked nodes correspond to points of the (possibly forked) line in space–time described by an icon.

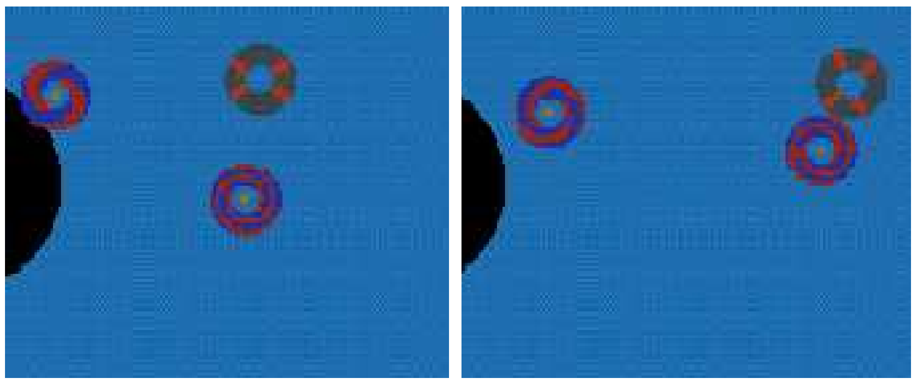


Figure 7: Critical points in the fluid flow behind a circular obstacle. The invariant subspaces of the saddle (indicated by the arrows) converge as the vortex and the saddle come close. The arrows point in the direction of the local flow.

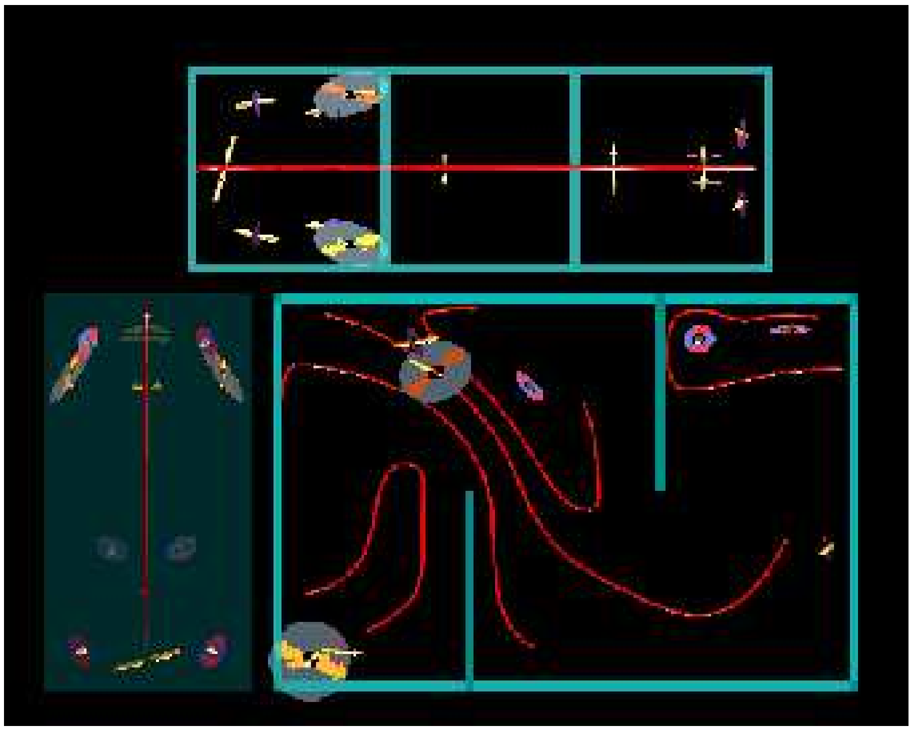


Figure 8: Critical points and particle traces in a fluid container with inner walls. The arrows which are scaled according to the eigenvalues of the gradient indicate the invariant manifolds of the critical point; in the plane of the (grey) disk the point is a node. The coloured disks describe the spiralling flow in the plane spanned by the complex eigenvectors of the gradient.

- a further link; the nodes with identical t form a list just as *icons* independent of time do.

Along each of those links that connect the successive states of an icon an interpolation can be defined. Therefore, we can extract an interpolated list of *icons* at any time in the time interval covered by the tree. And now, if the tree receives the "get-object" message, a linked list of *icons* is generated by interpolation which then may be displayed.

While the movement of icons can be designed interactively, the generation of such a tree may be automated as well. A routine may string the subsequent steps of an icon on a given integral curve, or else one may apply a routine that identifies critical points.

7 Conclusions

We have described a concept to deal with time-dependent simulation data on adaptive meshes, where the discretization changes in time. One aspect we have tried to underline throughout this paper is, that many questions of scientific visualization in this area are closely related to numerical concepts. We think that the visualization of time-dependent adaptive data is a good example for the benefits visualization derives from the numerical background. On the other hand an interactive and efficient handling of this type of simulation data supports visual debugging of recent numerical methods and helps to understand the global geometry and interesting features of the results.

Acknowledgments

The authors gratefully acknowledge numerous helpful remarks and many inspiring discussions especially with A. Schmidt and E. Bänsch and they thank T. Geßner, A. Schmidt and E. Bänsch for providing part of the numerical data.

References

[1] Advanced Visual Systems, Inc.: AVS user's guide, Waltham, 1992

[2] Babuška, I.; Rheinboldt, W. C.: Error estimates for adaptive finite element computations, SIAM J. Numer. Anal. 15, 736–754, 1978

[3] Banerjee, D.; Morley, C.; Smith, W.: The Design and Implementation of the Cortex Visualization System, IEEE Visualization '94, 265–272, 1994

[4] Bänsch, E.: Local mesh refinement in 2 and 3 dimensions, IMPACT Comput. Sci. Engrg. 3, 181–191, 1991

[5] Bank, R. E.; Sherman, A. H.; Weiser, A.: Refinement algorithms and data structures for regular local mesh refinement, Scientific Comput., North-Holland, 3–17, 1983

[6] Banks, D. C.: Singer, B. A.: Vortex Tubes in Turbulent Flows: Identification, Representation, Reconstruction, IEEE Visualization '94, 132–139, 1994

[7] Bryson, S.; Levit, C.: The Virtual Wind Tunnel, IEEE CG&A 12, No. 4, 25–34, July 1992

[8] Ciarlet, P. G.: The finite element method for elliptic problems, Repr. North-Holland, 1987

[9] Delmarcelle, T.; Hesselink, L.: Visualizing Second–Order Tensor Fields with Hyperstreamlines, IEEE CG&A 13, No. 4, 25–33, July 1993

[10] Dyer, D. S.: A dataflow toolkit for visualization, IEEE CG&A 10, No. 4, 60–69, 1990

[11] Eriksson, K.; Johnson, C.: Adaptive finite element methods for parabolic problems I: A linear model problem, SIAM J. Numer. Anal. 28, 43–47, 1991

[12] Favre, J. M.; Hahn, J.: An Object Oriented Design for the Visualization of Multi–Variable Data Objects, IEEE Visualization '94, 318–325, 1994

[13] Globus, A.; Levit, C; Lasinski, T.: A Tool for Visualizing the Topology of Three–Dimensional Vector Fields, IEEE Visualization '91, 33–40, 1991

[14] Haber, R. B.; Lucas, B.; Collins, N.: A data model for scientific visualization with provisions for regular and irregular grids, IEEE Visualization '91, 1991

[15] Helman, J. L., Hesselink, L.: Surface Representation of Two– and Three–Dimensional Fluid Flow Topology, IEEE Visualization '90, 6–13, 1990

[16] Helman, J. L.; Hesselink, L.: Visualizing Vector Field Topology in Fluid Flows, IEEE CG&A 11, No. 3, 36–46, May 1991

[17] Hin, A. J. S.; Post, F. H.: Visualization of Turbulent Flow with Particles, IEEE Visualization '93, 46–52, 1993

[18] Hultquist, J. P. M: Constructing Stream Surfaces in Steady 3D Vector Fields, IEEE Visualization '92, 171–178, 1992

[19] IBM, Inc.: IBM AIX Visualization Data Explorer, user's guide, IBM Publication SC38–0081

[20] Johnson, C: Adaptive finite element methods for diffusion and convection problems, Comput. Methods Appl. Mech. Engrg. 82, 301–322, 1990

[21] Johnson, C.; Hansbo, P.: Adaptive Finite Element Methods in Computational Mechanics, Preprint, Chalmers University of Technology, Göteborg, 1992

[22] Kardestuncer, K.: Finite element handbook, McGraw-Hill, New York, 1987

[23] Lane, D. A.: UFAT – A particle Tracer for Time-dependent Flow Fields, IEEE Visualization '94, 257–264, 1994

[24] Lane, D. A.: Visualization of Time-Dependent Flow Fields, IEEE Visualization '93, 32–38, 1993

[25] Leeuw, W. C. de; Wijk, J. J. van: A Probe for Local Flow Field Visualization, IEEE Visualization '93, 39–45, 1993

[26] Lucas, B.; et. al. : An architecture for a scientific visualization system, IEEE Visualization '92, 1992

[27] Mitchell, W. F.: A comparison of adaptive refinement techniques for elliptic problems, ACM Trans. Math. Software 15, 326–347, 1989

[28] Pagendarm, H.–G., Walter, B.: Feature Detection from Vector Quantities in a Numerically Simulated Hypersonic Flow Field in Combination with Experimental Flow Visualization, IEEE Visualization '94, 117–123, 1994

[29] Pazy, A.: Semigroups of Linear Operators and Applications to Partial Differential Equations, Springer, 1992

[30] Polthier, K.; Rumpf, M.:A Concept for Timedependent Processes, in: M. G"obel; H. M"uller, B. Urban (eds.): Scientific Visualization, Springer, 1995

[31] Rumpf, M.; Geiben, M.: Visualization of finite elements and tools for numerical analysis, Advances in Scientific Visualization, Eds. F. Post, A. H. Hin, Springer, 1993

[32] Schroeder, W. J.; Lorensen, W. E.: Visage: An object-oriented scientific visualization system, IEEE Visualization '92, 219–225, 1992

[33] SFB 256, University of Bonn: GRAPE manual, http://www.iam.uni-bonn.de/main.html, Bonn 1995

[34] Silicon Graphics Computer Systems, Inc.: IRIS Explorer, Tech. Report BP–TR–1E–01, 1991

[35] Smith,M. H.: Analysis and Visualization of Complex Unsteady Three-dimensional Flow, AIAA Paper 89–0139

[36] Strauss, P. S.; Carey, R.: An Object–Oriented 3D Graphics Toolkit, Computer Graphics 26, No. 2, 341–349, 1992

[37] Temam, R.: Navier-Stokes equations: theory and numerical analysis, Repr., 1985

[38] Treinish, L. A.: Data structures and access software for scientific visualization, Computer Graphics 25, 104–118, 1991

[39] Upson, C.; et. al.: The Application Visualization System: A computational environment for scientific visualization, IEEE CG&A 9, No. 4, 30–42, 1989

[40] Wierse, A.; Rumpf, M.: GRAPE, Eine objektorientierte Visualisierungs– und Numerikplattform. Informatik Forschung und Entwicklung 7, 145–151, 1992.

[41] Wijk, J. J. van: Flow Visualization with Surface Particles, IEEE CG&A 13, No. 4, 18–24, July 1993

[42] Wijk, J. J. van: Implicit Stream Surfaces, IEEE Visualization '93, 245–252, 1993