WP 7.3 Project Deliverable

Conference Papers and Exhibitions at Confereneces

Project Number	IST-2000-29266
Project Title	Virtual Real Time Fire Emergency Simulator
Deliverable Type	Report
Deliverable Class	Internal

Deliverable Number	D7.3
Title of Deliverable	Dissemination through Conferences and Exhibitions
Nature of the Deliverable	Report
Contributing WPs	WP 7
Contractual Date of Delivery	28 February 2004
Actual Date of Delivery	30 August 2004
URL	www.virtualfires.org
Authors	Sascha Schneider (FIGD), Michael Schmidt (FIGD), Volker Luckas (FIGD), Gunther Lenz (SiTu), Thomas Reichl (SiTu), Wilhelm Brandstätter (CD), Christian Redl (CD), Gert Svensson (KTH), Kai-Mikael Jää-Aro (KTH), Alain Bochon (Alpetunnel), René-Michel Faure (CETU), Klaus Schäfer (FDDo), Rainer Koch (FDDo), José Luis Ajuria (EUVE), Christian Redl (CD), Redactor : R.M. Faure
Contact Details	Institute for Structural Analysis / SiTu Research Univ. Prof. Dipl.-Ing. Dr. techn. Gernot Beer Lessingstrasse 25/II 8010 Graz / Austria Tel.: +43 316 8736180 Fax: +43 316 8736185 Email: gernot.beer@ifb.tu-graz.ac.at

Abstract	List of documents shown in journals.
Keywords	Papers, documents, news.

Contents

1 Overview

This report lists all papers that have been published in journals during the project.

2 Conference papers

2.1 FIGD

2.1.1 Parallel Real Time Fluid Simulation and Animation with Fractal Optical Refinements

Thorsten May, Sascha Schneider, Volker Luckas

Fraunhofer Institut für Graphische Datenverarbeitung

Submitted to ESM 2002

2.1.2 The Progressive Grid: Introducing a new Grid Class for efficient CFD Visualization

Thorsten May, Sascha Schneider, Michael Schmidt, Volker Luckas

Fraunhofer Institut für Graphische Datenverarbeitung

Proceedings of the High Performance Computing Conference (HPC), pp.115-120, April 2003

(Orlando)

2.1.3 Parallel architecture of an interactive scientific visualization system for large datasets

Sascha Schneider, Thorsten May, Michael Schmidt

Fraunhofer Institut für Graphische Datenverarbeitung

OpenSG Symposium (2003)

2.1.4 Fast scalar- & vectorfield visualization using a new progressive grid class

Thorsten May, Sascha Schneider, Michael Schmidt, Volker Luckas

Fraunhofer Institut für Graphische Datenverarbeitung

Proceedings of the Simulation and Visualization Conference (SimVis), pp. 89-101, March 2003 (Magdeburg)

2.1.5 Fast & flexible representation of CFD-data: Progressive Grids

T. May, S. Schneider, M. Schmidt and V. Luckas

Department for Animation and Image Communication, Fraunhofer Institute for Computer Graphics, Darmstadt, Germany

Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)

2.2 KTH

2.2.1 Implementing a CFD Steering system for immersive environments

Kai-Mikael Jää-Aro

Department of Numerical Analysis and Computer Science

Royal Institute of Technology

2.2.2 Virtualfires - realtidssimulering och visualisering av brandförlopp i tunnlar

Gert Svensson, Kai-Mikael Jää-Aro

PDC, Kungliga Tekniska Högskolan

VR Forum, 5 November 2003

2.3 SiTu

2.3.1 VIRTUALFIRES a virtual reality simulator for tunnel fires

Gernot Beer, Thomas Reichl and Gunter Lenz

Institut für Baustatik, Graz University of Technology, Austria

Int. Conference on Tunnnel Safety and Ventilation 2002, Graz, 8.-10. April 2002

2.3.2 VIRTUALFIRES A Virtual Reality Simulator for Tunnel Fires

Beer G., Reichl Th., Lenz G.

Institute for Structural Analysis

Graz University of Technology, Austria

Int. Conference on Tunnnel Safety and Ventilation 2004, Graz, 19.-21. April 2004

3 Exhibitions at Conferences

3.1 SiTu

3.1.1 1st International Conference on Safe and Reliable Tunnels

SiTu's Thomas Reichl and Gunther Lenz ran a 3 day demo booth in front of the conference room at Hotel Pyramida, Praha, 4.-6. February 2004

3.1.2 International Conference on Tunnel Safety and Ventilation 2004

SiTu's Thomas Reichl and Gunther Lenz ran a 2 day demo booth in the exhibition room of the Graz University of Technology, 19.-21. April 2004.

4 Appendix: Full Papers

Parallel Real Time Fluid Simulation and Animation with Fractal Optical Refinements

Thorsten May, Sascha Schneider, Volker Luckas

Fraunhofer Institut für Graphische Datenverarbeitung
Fraunhoferstr. 5, 64283 Darmstadt, Germany, Tel. (+49) 6151 155-638, Fax (+49) 6151 155-139
eMail: {tmay,sschneid,luckas}@igd.fhg.de

Abstract

In this paper we describe the parallelization of an unconditional stable solution scheme of the Navier Stokes equation which is suitable for animation purposes. Profiting from its unconditional stability we are able to use simulation time steps wich are larger than the animation timesteps and interpolate between them. Furthermore we present additional improvements in the visualization quality of the simulated output by using fractal textures in the 3D texture renderer post process for optical refinement using the scalar and vector fields as input parameters. Our techniques facilitate real-time animation rates of several examples on current multi-processor workstations and PCs.

1. Introduction

Some of the most fascinating phenomena one can observe in nature are the results of turbulent fluids or gases. According to the rising computer performance *computational fluid dynamics* (CFD) has become a major topic in computer graphics, animation and simulation in the recent years [6, 7, 8, 4, 9, 5, 3, 10]. However, in a graphical animation context the topics "real time simulations" and "optical accuracy" are much more important than in a scientific engineering setting. The simulation can be computed at much lower precision to give still a good impression of a realistic behaviour of the gas or fluid. The result of the simulation has just to be "visually right". Therefore the introduction of an unconditionally stable solution scheme of the Navier-Stokes equation into the area of computer graphics [5] was seen as an important invention by the graphics community. The implicit schemes proposed by Stam for fluid animation allow large step-sizes at the expense of an over-damping of the solution. For the case of the Euler equations, i.e. for systems without viscosity, Stam has shown in recent work how this over-damping can be corrected [6].

For some relatively small but non-trivial 3D examples the sequentially implemented simulation and animation system of Stam was a major step towards the goal of real-time animation. So it is a natural consequence to investigate which speed-ups of the included algorithms can be achieved by extending them through parallelization and running them on multi-processor-systems. In our previous work [12] we showed that this stable solution scheme in principle is well suited for parallelization. We described a parallel implementation that is portable on workstations and PCs running under the UNIX or the WIN32 operating systems. However, one sequential bottleneck remained (in a parallel sub-task) which prohibited the scaling of our parallelization on more than about 3 or 4 processors. In a successive research project the bottleneck was then eliminated by us [13].

Profiting from the unconditional stability of the solution scheme we make use of simulation time steps larger than the size of the frame rendering time step ("frame rate"). By interpolating between the simulation time steps, our parallel environment is able to achieve real-time fluid animations of examples that are by about a factor 2 to 5 larger than previous ones. Analogously we transferred the approach of computing only a 'rough' solution and refining it afterwards, we investigated fractal textures: In the physical simulation we used time steps larger than the frame rate. The information "in between" the calculated time step is automatically interpolated after the performance demanding simulation is done. The same concept was transferred by us from the simulation to the texturing of the scene / volume. The texture informations between the exact calculated scalar values of the dis-

crete simulation grid are computed in a kind of post process operation as well. With the help of automatically calculated fractal textures we are able to enhance the visual quality of the generated output significantly. The parallel fractal texture generation is only based on the scalar and the velocity grids. Furthermore it makes use of hardware accelerated primitives saving precious processor power. There are several papers and books available which introduce the concepts of fractal texture generation [14] we refer to.

In this paper we will first describe the global structure of our program, then going into details of the previous work of Stam [5] which was parallelized by us [12]. Afterwards we explain how our volume renderer works and how we implemented the interpolation [13] and the fractal texture generation on the visualization side. At the end we will present results we achieved with our methods and give an overview of our future work.

2. The parallel architecture

In this section we will give a "top down" description of the parallelized components of our program. It can be subdivided into two main parts (cp. Fig. 1): The simulation code and the solver code communicating over a cyclic *n*-fold buffer.

Since for real time animations a parallelization over a network has a too high latency, we focus on shared-memory architectures, which are nowadays already available in low cost multi-processor PCs or workstations. The general programming paradigm that is available for these platforms is the one of multi-threaded programming. Although quite similar from a programming point of view, the APIs offered by the 32-bit Windows operating systems and the various UNIX derivates differ significantly. However, there are several abstraction classes available on the market with little performance overhead that allow a platform independent access to the thread system (e.g. *Adaptive Communication Environment* (ACE) [15], *OMNI thread* package [16] or the *QT* Library [18]).

Currently, we use the *QT* Library as a platform independent abstraction for the threads, but switching to another one like ACE would be possible with a moderate programming effort.

The windowing classes that we use are also completely written in Qt in order to allow a platform independent implementation. The graphical context of the window is filled with rendered OpenGL graphics [19]. This allows our program to run on all operating systems which support Qt and OpenGL as well. As an improvement to our previous work [12, 13] we have now migrated our thread implementations to Qt.

2.1. Communication and synchronization between renderer and solver components

As mentioned before in our program the renderer (OpenGL/QT window) interacts with the CFD solverthreads using a cyclic *n*-fold buffer. Each buffer contains the velocity field and the entities used for rendering, e.g. the current texture coordinates. For interpolation, $n-1$ parts of the buffer are used by the renderer. After the solver thread has completed one calculation-step and has written its results in the currently unused part of the buffer, the marker in the cyclic buffer, which indicates the last entry to be used by the renderer, is advanced in one position. Thus only a "switch of pointers" has to be locked. A schematic view is given in fig. 1.

3. The simulation algorithm

There is a vast literature on computational fluid dynamics. For general information we refer to recent textbooks [20, 21, 22, 23, 24]. For the specifics of the sequential algorithm that is the basis of our work we refer to the paper of Stam [5].

The simulation program solves the Navier-Stokes equations

$$\nabla \cdot \mathbf{u} = 0 \tag{1}$$

$$\frac{\partial u}{dt} = -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f} \tag{2}$$

for an incompressible fluid, where u is the velocity vector field, v is the viscosity, d the density, p the scalar pressure field and f the external forces acting on the fluid, e. g. buoyancy.

In an incompressible fluid the role of the pressure is to allow continuity to be satisfied. It is therefore possible to write eqn. 1 and eqn. 2 as just one equation using an operator P, which orthogonaly projects a vector field on to the set of divergence free fields. Using this operator the equations 1 and 2 can be written as

$$\frac{\partial u}{dt} = \mathbf{P}(-(\mathbf{u}\cdot\nabla)\mathbf{u} - \nu\nabla^2\mathbf{u} + \mathbf{f}) \tag{3}$$

It is this equation that is solved by the simulation program. A fractional step method consisting of the following parts is used to solve the equation. The method we use for the solution is the one described by Stam [5], to which we refer for more details and a theoretical base for this method. We just sketch the main steps as a reference for the parallelization.

$$\mathbf{w}_0(\mathbf{x}) \overset{\text{addforce}}{\rightarrow} \mathbf{w}_1(\mathbf{x}) \overset{\text{advect}}{\rightarrow} \mathbf{w}_2(\mathbf{x}) \overset{\text{diffuse}}{\rightarrow} \mathbf{w}_3(\mathbf{x}) \overset{\text{project}}{\rightarrow} \mathbf{w}_4(\mathbf{x}) \tag{4}$$

We compute the vector field of velocities and the scalar quantities at discrete time-steps. The simulation of the scalar quantities at time t_n requires the knowledge of the vector field at this time step. However, in a simulation loop the simulation of the vector field for time step t_{n+1} (using the values of the vector field at time step t_n) can be done in parallel to the simulation of the time step of the scalar field at time step t_n (using the value of the scalar field t_{n-1} and the vector field at time step t_n).

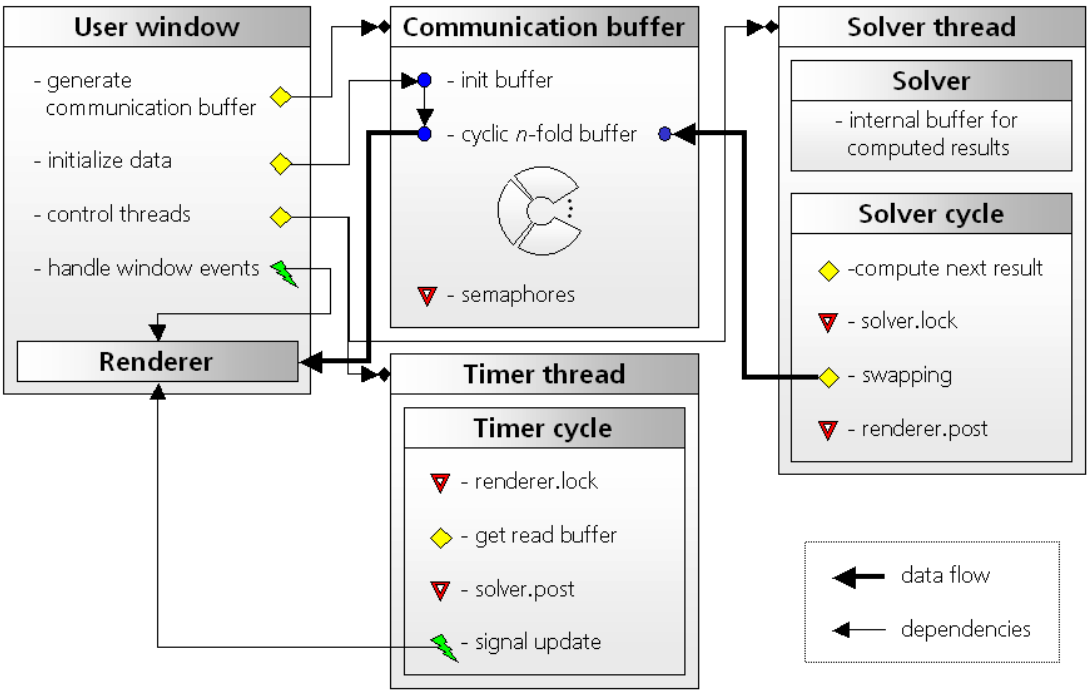


Figure 1: *Communication and synchronization between renderer and solver components.*

Any simulation step of the vector field requires the computation of the sequence *add force* → *transport* → *diffuse* → *project* (and similarly for the scalar quantities). These steps have to be taken sequentially.

3.1. Step 1: Add force step

The effect of external forces is solved using an ordinary explicit Euler scheme.

$$\mathbf{w}_1(\mathbf{x}) = \mathbf{w}_0(\mathbf{x}) + \Delta t \mathbf{f}(\mathbf{x}, t) \tag{5}$$

The *add force* step takes little time but the addition of any force to a cell can be done in parallel.

3.2. Step 2: Transport step

Using a technique based on the method of characteristics the nonlinear transport (advection) part of the equation is solved. This method has several advantages. Two of them are ease of implementation and—as will be shown later—ease of adaption to parallel computing. A point $\mathbf{x}$ is back-traced along a streamline in the old vector field to time $t = -\Delta t$. The velocity at $\mathbf{x}$ in the new vector field is then set to the value at the back-traced point. Velocity vectors not directly on a grid-point are defined using trilinear interpolation from the adjacent grid-points. Using a function $\mathbf{p}(\mathbf{x}, t)$ defined as the streamline passing through point $\mathbf{x}$ at $t = 0$ the step can be written as

$$\mathbf{w}_2(\mathbf{x}) = \mathbf{w}_1(\mathbf{p}(\mathbf{x}, -\Delta t)) \tag{6}$$

The *transport* step can be well parallelized without synchronization requirements: the computation for any grid cell is independent of each other. Using n threads, where n is about four times the number of available processors to reach saturation, we divide the cells in n parts and do the transport computation in the n threads, each responsible for one part. As the computation of transport step roughly takes the same time for any grid cell these threads can be expected to have the same run-time.

3.3. Step 3: Diffuse step

The third step deals with the effect of viscosity. This step calculates the solution of a standard diffusion equation for each cartesian component of the vector field. These equations are solved using an implicit Euler scheme, so the resulting system is

$$(\mathbf{I} - \nu \Delta t \nabla^2)\mathbf{w}_3(\mathbf{x}) = \mathbf{w}_2(\mathbf{x}) \tag{7}$$

where the ∇^2 operator is approximated using finite differences. Several efficient methods for solving such equations exist. Specifically in the implementation described in this paper the `pois3d`-solver from the FISHPAK library is used. This library is available from `http://www.netlib.org`.

Although this routine has a complexity of about $O(n \log n)$ and this equation can be solved theoretically in $O(n)$ using a multi-grid method [25], the practical performance of `pois3d` is much better on the currently used grids.

If the viscosity is 0, an approximation that might be made for air, the diffusion step can be omitted.

The *diffusion* step requires a solution of the corresponding equations in 3 dimensions, which are independent of each other. So we can use 3 threads to do these computations in parallel, which require no synchronization and which have nearly the same run-time. Parallelizing the diffusion computation for each dimension themselves is possible in principle with the technique described below for the project step.

3.4. Step 4: Project step

The vector field produced by the above steps are not ensured to satisfy the continuity equation. The final step is therefore a projection step which makes the field divergence free. The step can be written as

$$\nabla^2 q = \nabla \cdot \mathbf{w}_3 \tag{8}$$

$$\mathbf{w}_4 = \mathbf{w}_3 - \nabla q \tag{9}$$

This Poisson equation is solved using the same subroutine that is used in the diffuse step. Recently we have been able to split the computation of this step into 8 parallel sub-tasks [13]. We will only briefly sketch the main idea of this step.

Mainly because the *projection* step is split into two parts, it becomes clear, that in the best discretization by finite differences for inner grid points of the scalar field - especially one not using auxiliary points which have to be interpolated - no adjacent gridpoint appears. In fact, every gridpoint only depends from gridpoints with even distance, measured by gridcells. Since this keeps true for gridpoints close to the boundary, we are able to partition the grid in eight independent sub-grids. For those the Poisson equation can be solved in parallel. Using a solver with a complexity lower than O(n) this can even result in a theoretical speedup of more than eight.

Technically the partitioning is the disassembly of the right-side-vector of the equation into eight corresponding right-side-vectors for each of the eight equation systems and the remerging of the solution vector. Using the best available Poisson solver, this can still consume up to a quarter of the computation time of the projection step, so consequently this task has to be parallelized as well. Again, the partitioning and eventually the projection itself can be parallelized using domain decomposition since every gridpoint can be handled independently.

3.5. Computing scalar quantities

The evolution of a scalar quantity, e.g. temperature or smoke density in the fluid is computed using a method very similar to the one for the vector field described above. The equation that describes the behaviour of the scalar field is

$$\frac{ds}{dt} = -\mathbf{u} \cdot \nabla a + \kappa \nabla^2 s - \alpha s + S \tag{10}$$

where κ is the diffusion constant, α the dissipation rate and S a source term. All terms are solved using the same steps that are used for the vector field except of *project*, which is not needed. The dissipation term not present in the Navier-Stokes equations is solved using the following equation

$$(1 + \Delta\alpha) s(\mathbf{x}, t + \Delta t) = s(\mathbf{x}, t) \tag{11}$$

The corresponding parallelization scheme for the vector field solver is given in fig. 2, the one for the scalar field solver and the other quantities used for visualization (particles, texture coordinates) are quite similar.

4. The volume renderer

The principal components of the used volume renderer have not changed substantially. We thus refer to our previous work [12] for the details of the used splatting method and the method of using textures for optical refinement of a flow. As this method has also been used by others [5, 26], we refer to this sources for a description of the underlying ideas of this very important part of our fluid animation system.

4.1. Interpolation

Up to now all methods presented aim at the increase of the solver's performance, but the use of modern graphics hardware, which allows accelerated rendering of most complex scenes in realtime, still leaves a significant imbalance favoring the renderer regarding speed and performance. The unconditional stability of the solution scheme gives us a major gain in performance in comparision to explicit solution schemes, as already mentioned in the introduction.

However, for the purposes of real-time-applications 'smooth' transitions between two consecutive frames have to be ensured. These smooth transitions can be realized easily by interpolating between the keyframes provided by the solver. This takes less effort computationally than full simulation steps. The interpolation has been evaluated especially with regards to the quality by which the smallest structures appearing in the flow field are represented. In any given case we could interpolate about five frames without provoking perceptable artifacts and much loss of quality. Given the rather short computation time needed for interpolation in comparision to the time needed for calculating the simulation - this ratio is about 1:10 - we could achieve yet another speed-up of about four to five.

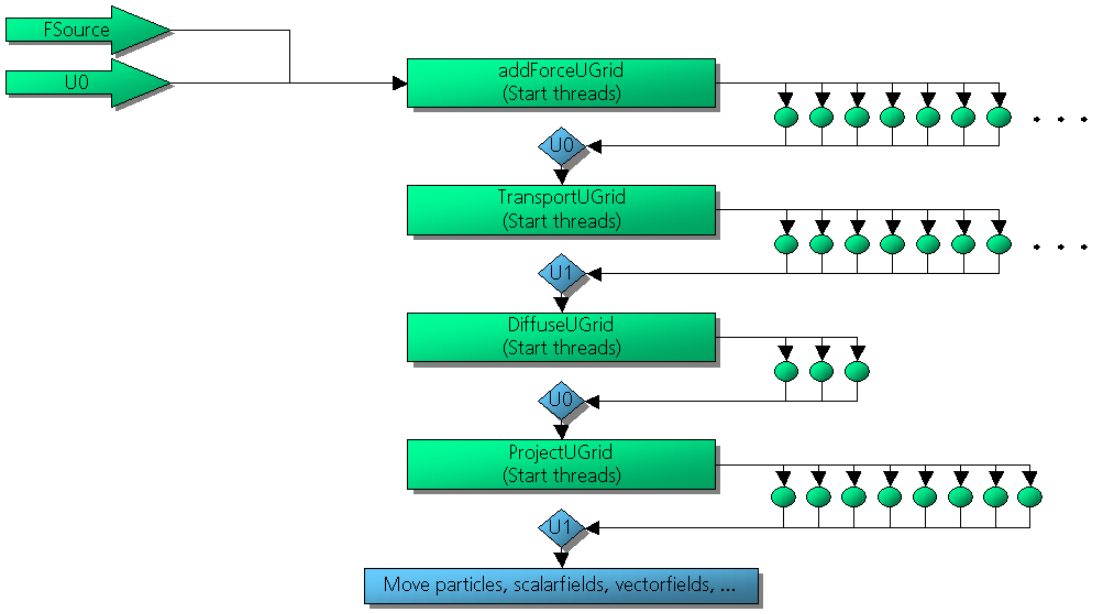


Figure 2: *Parallelization of the vector field solver.*

4.2. Fractal textures

As one can easily imagine, the resolution of the data provided by a realtime simulation can not be increased beyond a certain border because of limited system performance. These requirements collide with the need of a adequate display quality when zooming into the details of a simulation grid. A common method to eliminate this problem is to make use of a set of simple 'precomputed' textures, choosen in a way that their alpha values match the data to be displayed (e.g. a density field). Unfortunately this always gives the resulting image a kind of blobby look. Further enhancements can be made using texture coordinates dependent on the surrounding velocity field, in a way that the texture is distorted according to the conditions in the fluid [5, 12]. In large scales this may be a convenient method. In small scales the resulting cloudlike structures still give an artificial impression.

Because any fluid medium in nature shows fractal structures under the influence of physical forces, a realistic visualization has to take care of that. Other authors have already investigated the relation between fractal properties and fluids (resp. reynolds number) [27]. Therefore fractal textures are choosen by us not only for this reason but for their many possibilities to modifiy their parameters. In essence our textures emerge from a fractal turbulence field blended over a given scalar field [14], which represents the simulation results. The underlying scalarfield is not modified in any way to guarantee data authenticity.

Note: All definitions mentioned are valid for the composition of time dependent 3D-Textures. Adding a fourth dimension just means to add a new component. For better understanding we explain our texturing method for the 2D-case. (Function parameters in angular brackets denote indices, while those in common brackets denote real values resp. vectors.)

Our basic (fractal) turbulence function is designed as proposed in [14]. We start with a discrete periodic noise function:

$$noise[i,j,k] = func(rand()), [i,j,k = 0...size-1] \quad (12)$$

The following conditions are necessary to guarantee that the space can be tiled without inconsistencies:

$$\begin{aligned} noise[size,j,k] &= noise[0,j,k] \\ noise[i,size,k] &= noise[i,0,k] \\ noise[i,j,size] &= noise[i,j,0] \end{aligned} \quad (13)$$

where size is the desired period length, $rand()$ is a function creating equal distributed random values in $[0,1]$ and $func()$ can be an arbitrary mapping from $\mathbb{R}[0,1] \rightarrow \mathbb{R}[0,1]$. Normally $func()$ is set to identity mapping, but the equal distribution of the random values can be modified to create, for example, distortion fields with different distortion policies.

Of course, the noise function has to be defined in the continuum and we use a standard trilinear interpolation to achieve that. Where x,y and $t \in \mathbb{R} \cap \mathbb{N}$ we define:

$$noise(x,y,t) = noise[i,j,k], (x = i, y = j, t = k), \quad (14)$$

That way, the noise function is normalized to a base frequency of 1.0.

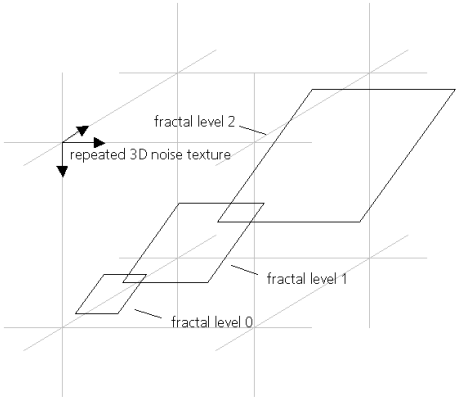


Figure 3: *Noisefield defined as a repeated 3D-Texture. Parallelograms depict the texturecoordinates at different fractal levels for the same patch.*

The central function used to model the fractal turbulence field successively uses the noise function at different magnitudes:

$$turb(x,y,t) = \frac{\left(\sum_{s=0}^{n} \frac{1}{c^s} noise(x \cdot c^s, y \cdot c^s, t \cdot c^s)\right)}{norm} \quad (15)$$

where

$$norm = \sum_{s=0}^{n} \frac{1}{c^s} \quad (16)$$

In this definition c is the fractal scaling factor. It has at least to be set to 1.0 (yielding the plain noise function). According to our experiences the most appealing results can be achieved with values between about 1.3 and 4.0. n can be interpreted as the fractal level of the resulting texture - in effect, this is the level of detail. By manipulating this, one has to keep in mind that the share of the high frequency components of the turbulence function decreases rapidly.

Now, with the basic turbulence function and the scalar field given, one can model the resulting scalar field via interpolation.

$$\begin{aligned} result(x,y,t) &= (turb(x,y,t) - scalar(x,y,t)) \\ &\quad * fact + scalar(x,y,t) \end{aligned} \quad (17)$$

Here $fact$ is the constant interpolation factor used purely as a design parameter to visualize different forms of clouds.

Applying the fractal textures the way we have introduced them the image quality is visually enhanced more than by just relying on the density function alone. Especially the borders of the structures of high density regions (i.e. such with comparably high scalar values) do not simply 'dim' out. Instead they show some fractal microstructures as they can be observed in every natural fluid in motion.

So far these fractal structures only show a certain undirected fluctuation because they just depend on time yet. It would be desirable to make the turbulence field dependent from the surrounding currents as well. Indeed in nature the filigree structures around a cloud are even more exposed to the wind than the clouds itself. Therefore we have modeled the same property for the turbulence field.

Because of that we model a distortion field with the interpolated velocity field v given by the simulation as:

$$\begin{aligned} turb_{new}(x,y,t) &= turb_{old}(\\ &\quad x + s \cdot dist(x,y,t,v).x, \\ &\quad y + s \cdot dist(x,y,t,v).y, t) \end{aligned} \quad (18)$$

where $dist$ generates a vector from the velocityfield and $.x$ resp. $.y$ denotes the corresponding coordinates of the given vector. $dist$ can modeled in several ways.

4.2.1. Integrating the (physically based) velocity field

The easiest way to do this, is to start with the inital velocity for every gridpoint and substract frame by frame any generated velocity vector for this frame:

$$\begin{aligned} dist(x,y,v,t_0) &= v(x,y,t_0) \\ dist(x,y,v,t_i) &= dist(x,y,v,t_{t-1}) - v(x,y,t_i) \end{aligned} \quad (19)$$

s is a scaling factor that describes the degree of influence the fluid velocity has on the filigree. We recommend to adjust this factor regarding the base frequency of the noisefield and the expected average or maximum velocity value. This approach gives us, aside from the enhancing effect, a second benefit when investigating the details of a vectorfield: The filigree in motion can act as a additional visual cue for the local currents in the fluid even when the overall velocity of the density field moving through the fluid is low. The approach of adding a distortion field is not limited with regards to a given velocity field. In fact we use a second distortion field to add a certain undirected brownian motion, which is quite slow compared to the velocities in the current.

Investigating another promising method, we experimented in using the fractal turbulence function as a distortion field rather than as a additional density field. Using this approach we are still able to make this distortionfield dependent on the surrounding currents. This method is slighty more expensive in computation time than the one mentioned above. We achieve results which allows us to depict flames that look quite realistic (see Fig. 6).

4.3. Texture application

While calculating and applying the fractal textures we have to keep in mind, that the overall postprocess effort to enhance to image quality after each simulation step should not be too high compared to the simulation itself. Therefore we choose to utilize the features modern graphics boards

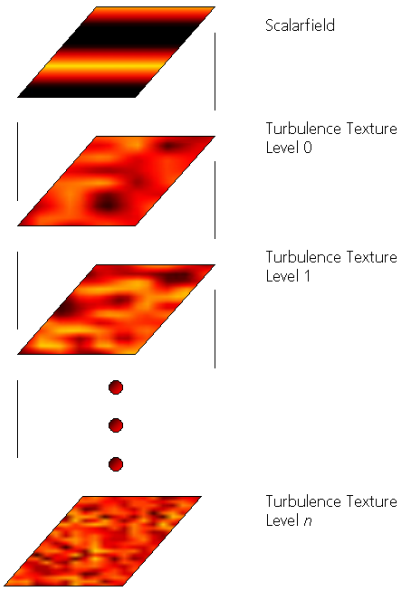


Figure 4: *The composition of the different levels of the fractal textures.*

offer. Especially the interpolations are the most timeconsuming tasks and can easily be passed on hardware. We modeled the noisefield as a 3D texture, so any patch rendered can be multitextured in as many layers as the fractal level of the turbulence field dictates. Each level is applied with a different scaling factor. Following this approach the distortion field becomes a texture coordinate transformation defined on the vertex points of the patch.

5. Results

We measured the speed-up of our parallelization by determining the wall clock times for the sequential and parallel versions as the averages of 10 runs. When determining the speed-ups of the different steps in the vector field computations we disabled the render components in order to avoid disturbances. On a 4 processor SGI Onyx 2 workstation we obtained speed-ups of 2.8–3.1 on grids of sizes 31^3 and 64^3 for project step. On grids of size 31^3 cells the speed-up for the transport step has been 3.1 and the one for the diffuse step has been 1.7. On larger grids (with 49^3 cells or more) the speed-up of the transport step reached the theoretically possible value of 4 up to 5 %. As the speed-ups scale to faster processors, animations on grids of these sizes are possible in real-time using our programming techniques on new multiprocessor systems that consist of GHz-processors. Further performance measurements of our implemented results are presented in our previous work [12].

Some snapshots from parts of our system are given in the Fig. 5 and Fig. 6. Movies showing some further animations are provided as accompanying material and can also by found in the Web: `http://www.igd.fhg.de/igd-a3/projects/physically-based-simulation-and-animation/cfd`

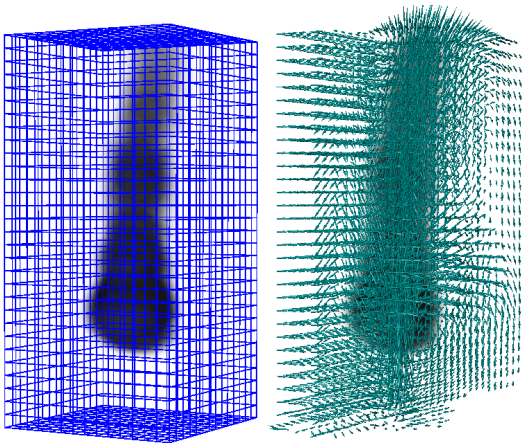


Figure 5: *A scalar field and the corresponding velocity field from one of our smoke simulations.*

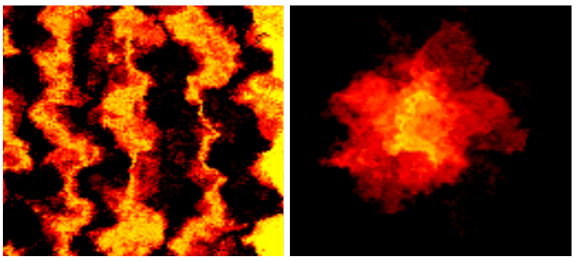


Figure 6: *Some of our fractal textures - 2D example: 2 density fields distorted by fractal turbulence fields.*

6. Conclusion

One further enhancement of our approach can be made introducing even more design parameters for the fractal texture generation. For example it could be useful to add space and time dependent factors to the texture generation equations. Another enhancement could be the development of a second distortion field to depict true microturbulences. It would also be nice to find simple and efficient rules to describe their appearance and character. They should, of course, depend on the velocity field as well. Since turbulent currents indeed show fractal structures it may be interesting to investigate if a fractal function replacing our turbulence function can be used as a distortion field for the texture.

References

1. ACM SIGGRAPH. *SIGGRAPH 99 Conference Proceedings*, Annual Conference Series, Los Angeles, CA USA, August 1997. 8
2. ACM SIGGRAPH. *SIGGRAPH 2000 Conference Proceedings*, Annual Conference Series, New Orleans, LA USA, July 2000. 8
3. Yoshinori Dobashi, Kazufumi Kaneda, Hideo Yamashita, Tsuyoshi Okita, and Tomoyuki Nishita. A simple, efficient method for realistic animation of clouds. In *In SIGGRAPH 2000 Conference Proceedings*, pages 19-28 [2]. 1
4. Nick Foster and Dimitris Metaxas. Modelling the motion of a hot, tubulent gas. In *In SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, pages 181-188, Los Angeles, CA USA, August 1997. ACM SIGGRAPH. 1
5. Jos Stam. Stable fluids. In *SIGGRAPH 99 Conference Proceedings*, pages 121-128, [1]. 1, 2, 4, 5
6. Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *SIGGRAPH 2001 Conference Proceedings*, Annual Conference Series, Los Angeles, CA, USA, August 2001. ACM SIGGRAPH. 1
7. J. Stam and E. Fiume. Turbulent wind fields for gaseous phenomena. *In SIGGRAPH 93 Conference Proceedings*, Annual Conference Series, pages 369-376, Anaheim, CA, USA, August 1993. ACM SIGGRAPH. 1
8. J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *In SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 129-136, Los Angeles, CA USA, August 1995. ACM SIGGRAPH. 1
9. Henrik Weimer and Joe Warren. Subdivision schemes for fluid flow. In *SIGGRAPH 99 Conference Proceedings*, [1]. 1
10. Gary D. Yngve, James F. O'Brien, and Jessica K. Hodgins. Animating explosions. In *SIGGRAPH 2000 Conference Proceedings*, pages 29-36 [2]. 1
11. Davic Baraff and Andrew Witkin. Large steps in cloth simulation. In *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 43-54, Orlando, FL, USA, July 1998. ACM SIGGRAPH.
12. Mattias Bryborn, Reinhard Klein, Thorsten May, Sascha Schneider, and Andreas Weber. A portable, parallel, real-time animation-system for turbulent fluids. In *Parallel and Distributed Computing and Systems (PDCS 2000)*, pages 394-400, Las Vegas, USA, November 2000. International Association of Science and Technology for Development. 1, 2, 4, 5, 7
13. Reinhard Klein, Thorsten May, Sascha Schneider, and Andreas Weber. Real-Time Fluid Animation by Parallel and Stable Solutoin Techniques. In *Festschrift zum 60. Geburtstag von Wolfgang Strasser*, pages 91-107, Tübingen 2001. 1, 2, 4
14. David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing & Modelling* - A Procedural Approach ACADEMIC PRESS, 1998. 2, 5
15. Douglas C. Schmidt. *The ADAPTIVE Communication Environment:*. An object-oriented network programming toolkit for developing communication software. 1993. `http://www.cs.wustl.edu/schmidt/ACE-papers.html`. 2
16. Tristan Richardson. *The OMNI Thread abstaction*. `http://www.lfpt.rwth-aachen.de/Links/GNU/gnu/omniorb/omnithread/omnithread.html`. 1997. Available as part of omniORB. [17] 2
17. AT&T Laboratories Cambridge. *omniORB*. 1999. `http://www.uk.research.att.com/omniORB`. 8
18. Trolltech. *QT 3.0*. 2001. `http://www.trolltech.com/`. 2
19. Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide*. Addison-Wesley, third edition, 1999. 2
20. Donald F. Young, Bruce R. Munson, and Theodore H. Okiishi. *A Brief Introduction to Fluid Dynamics*. John Wiley & Sons, 1997. 2
21. Frank M. White. *Fluid Dynamics*. McGraw-Hill, 3rd edition, 1994. 2
22. Joel H. Ferziger and Milovan Peric. *Computational Methods for Fluid Dynamics*. Springer Verlag, 2nd edition, 1999. 2
23. M. B. Abbott. *Computational Fluid Dynamics: An Introduction for Engineers*. Wiley, 1989. 2
24. A. J. Chorin and J. E. Marsden. *A Mathematical Introduction to Fluid Mechanics*, volume 4 of *Texts in Applied Mathematics*. Springer-Verlag, 2nd edition, 1990. 2
25. W. Hackbusch. *Multi-grid Methods and Applications*. Springer-Verlag, 1985. 4
26. N. Max, R. Crawfis, and D. Williams. Visualizing wind velocities by advecting cloud textures. In *Proceedings of Visualization 92*, pages 179-183, Los Alamitos, CA, USA, 1992. IEEE. 4
27. Uriel Frisch, and A. N. Kolmogorov. *Turbulence*, Cambridge University Press, 1995. 5

The Progressive Grid: Introducing a new Grid Class for efficient CFD Visualization

Thorsten May, Sascha Schneider, Michael Schmidt, Volker Luckas
Fraunhofer Institut für Graphische Datenverarbeitung
Fraunhoferstr. 5, 64283 Darmstadt, Germany
{**tmay,sschneid,mschmidt,luckas**}**@igd.fhg.de**

Keywords: Conversion, compression, scalar-/vector field, Computational Fluid Dynamics, progressive grid

Abstract

We present a new generic progressive data format and framework for CFD-data, which is specialized for scientific data simulation: The progressive grids. While using arbitrary types of cells and partitioning schemes, simulation data can be stored progressively. Hierachically arranged by its level of detail, the data can be adapted efficiently to many systems with different ressources. Maintaining a modular approach, we are able to implement, test and update different kinds of progressive grids by adjusting the corresponding parts of the system.

1 INTRODUCTION

The ever increasing computational power of computer workstations and clusters used within the field of numerical simulation produces a comparable increasing mass of simulation data results. These results are generally accessible by means of scientific visualization. Today, the sheer mass of data makes it impossible to display the data on low-end-systems (e.g. laptops used for presentations) with an acceptable performance. It is essential to use data structures, which allow to trade off accuracy, quality and rendering performance as well as to adapt them to the system resources of the system used. In effect, this means to load and display only a certain part of the simulation data. It is crucial to guarantee that this part contains the most important information: With only a small portion loaded one will be enabled to qualitatively explore the whole simulation domain in real time. The progressive grids developed in this context fulfill this goal.

2 RELATED WORK

Progressive grids are a certain form of hierarchically structured grids. Hierarchically structured grids are well known in the field of computer graphics [1, 2]. In this context they are used to spatially arrange large amounts of geometry fragments in a way that one is able to decide for any given cell which fragments it contains. These grids are adaptive, because the cells are divided if the enclosed data exceeds a certain threshold.

Our progressive grids make use of the technology known from progressive data formats, which are closely related to digital image processing (Wavelet-, JPEG-Compression) [3, 4, 5]. In these formats the data is ordered by its level of detail. So the data associated with a given arbitrary resolution can be extracted efficiently. As a result the amount of data that has to be transmitted and/or processed can be freely adapted to variable system resources or the users requirements. Geometry data, for example, can be transmitted and displayed simultaneously in incremental granularity levels thus the viewer gets an impression of the geometry already with the beginning of the transmission [6].

While using progressive data formats in the context of CFD simulation data, we take advantage of these properties. We arrange the data with respect to the information it contains, beginning with the most important ones. According to this alignment, we build a hierarchy that is represented in a spatial partitioning scheme that has so far been used in computer graphics. We now put it into a new context: the management of CFD simulation data.

The concept of *adaptive mesh refinement(AMR)* is well-known in the field of numerical computation [7] for the solution of partial differential equations using finite differences. The AMR-method [8] is based on the principle of making recursive local refinements to a given coarse grid. The refinements are controlled by an error, which is determined by a suitable error metric. The basic idea of this method is to execute numerical computations only where they are necessary.

Other than in the field of numerical analysis and scientific computation there are only a few related approaches in the area of scientific visualization. In [9] a method is proposed, which decomposes a rectilinear mesh into tetrahedra and octahedra and approximates the original values by using triangular *coon patches*. If the resulting error from the initial decomposition is too large, the partitioning con-

tinues locally. [10] utilizes hierarchically structured rectilinear grids but focusses on the interpolation scheme that avoids discontinuities at the boundaries facing different subdivision levels.

Another approach for the generation of hierarchical meshes is described in [11]. Actually, the authors also build the hierarchy from the coarsest level or coarsest mesh respectively, but the approximation problem is shifted to the *finite element space*. The input data is regarded as a discretization of a smooth function, which will be approximated by an adaptive mesh refinement and error control starting from a discrete function that is defined on a coarse mesh. [12] specifies the associated adaptive mesh algorithm that is based on an adaptive and regular refinement of a coarse start grid by a consistent decomposition into tetrahedra and octahedra.

3 CONCEPT

With our work we adopt the principle of progressive data processing to CFD data resp. discrete scalar- and/or vectorfields. Since any grid which is used in numerical simulation is unable to handle its data progressively, these grids have to be converted first. Afterwards the data is available in a format which is specially enhanced to meet the demands of a visualization system.

The basis for a progressive grid and the input data for the conversion algorithm is a regular discretization of the domain. We can choose if this discretization should originate from the simulation grid itself or a from a regular sample of an irregular or even unstructured grid. In the following we describe how a progressive grid is organized an how the conversion works.

First of all, it has to be stated that our progressive grids are *generic*. They are implemented in a conversion framework. It is thereby possible to choose the topological structure arbitrarily: It is possible to select which cell types or spatial partitioning schemes will be used. They define the topology of the progressive grid and how the given grid data has to be interpreted. The same holds true for the interpolation schemes (defining the error estimation), used to compare the progressive grid and its parent - the numerical grid. This approach even allows us to couple different time steps in a four-dimensional grid to make use of the temporal coherencies for compression.

As stated above, we represent our grid in a spatial partitioning scheme, which in turn means that all cells contained in it are arranged in a hierarchical order. The actual numerical information is either associated with a given cell or with the vertices defining its corners. In any case, the partitioning of a cell introduces a certain number of vertices along with at least two new childcells. Their data will contribute to provide a greater local accuracy.

3.1 The Grid Cells And Their Data

The *cell* is the elemental structure in the progressive grid. In an abstract sense, it stores

- references to the vertices at its corners,
- the information if and how the cell is partitioned,
- references to its 'parent' and 'child'-cells (if any)
- numerical information associated with the cell,
- references to a certain number of neighboring cells (if necessary).

This storage structure can be illustrated in a directed acyclic graph. If possible, the corners of the cells should be vertices of the original regular sample. As there is no need for the progressive grid to be finer than the original one, this helps saving an interpolation step afterwards. For cell types other than boxes, however, this condition may be not satisfied. The vertices contain their coordinates (being of arbitrary dimension) - this defines the location of a cell - and the scalar values derived from the original grid at their position. Furthermore - if the input data is able to provide this information - every vertex has a flag which marks if it is located within the surrounding original grid, which is not necessarily the case (referred to as the *inoutflag*). The reason for a cell to store only references to the vertices is that the single vertex is obviously shared by a number of neighboring cells and by a number of cells belonging to the same hierarchy. A standard grid cell is illustrated in figure 1.

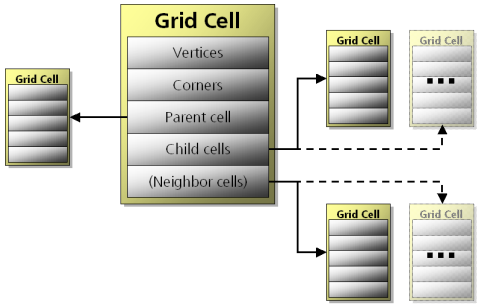


Figure 1: A grid cell.

The information regarding the partitioning of a cell is encoded according to the cell type and the partitioning scheme given.

Most important of the numerical information associated with a cell (further referred to as *volume data*) for the conversion algorithm is the *approximation error*. It describes the approximation quality compared to the original

grid. Additionally, some information related to the different scalar fields (e.g. their minimum, average and maximum values within a cell) is provided.

3.2 The Modular Approach

The topology of a progressive grid cell, the number and connectivity of its corners, edges and faces is an implication of the cell type itself. Together with the partitioning scheme these information can be seen as a convention how to interprete the cell data. Because never stored explicitly in the grid for reasons of memory consumption, all algorithms that operate directly on this data have to fulfill this convention. Each of the generic parameters can be changed virtually independently from each other offering various configuration possibilities. In order to investigate this possibilities efficiently, we encapsulated this convention in a module and made it accessible only via an abstract interface for the converter and a second interface. This interface will be used by the visualization methods afterwards.

4 CONVERSION SCHEME

4.1 The Initial Partitioning

The main task during conversion is to build up the cells hierarchy. This is the most time consuming job. Furthermore the converter has to define the order in which the cell information blocks have to be stored. Since the conversion scheme should remain independent from the given topology or data of the progressive grid, the construction of the cell's hierarchy is handed over to the module mentioned above.

The conversion starts with an initial partition. In the simplest case, using box-type cells, the grid can be based on the bounding box of the simulation grid. In the case of a tetrahedral decomposition, the grid is rooted in a special initial partitioning of the original bounding box into a number of tetrahedrons. However, with a given cell type, we are not restricted to a certain initial decomposition. This allows the generation of more than one 'root'-cell (and an associated hierarchy). To support more than one initial partition of the original domain is advantagous, because we are able to define a level of detail for every single spatial domain (resp. its hierarchy) separately.

4.2 Determining Approximation Quality

Given the partitioning scheme, a hierarchy of grid cells is built up through spatial partitioning. Within a cells boundary the data is interpolated using a selected interpolation scheme (see section 3). For every unpartitioned cell, an approximation error is computed by comparision of values interpolated within the current cell and values given be the original grid at a set of sample points. This approach guarantees that the approximation by the progressive grid is independent from the topology of the original grid. The independency has one drawback, though: As no information from the original grid is presented, except for the values at the sample points, the right choice of the samples points is crucial for the conversion. Just choosing the vertices of the original grid can lead to severe artifacts, because cells of considerable size in the progressive grid may not contain a single vertex of the original one - even if the original grid would intersect the cell. For this reason we choose to predefine a regular sample of the original grid, which is fine enough to cover all its details and compare the interpolation within the current cell with the regular sample.

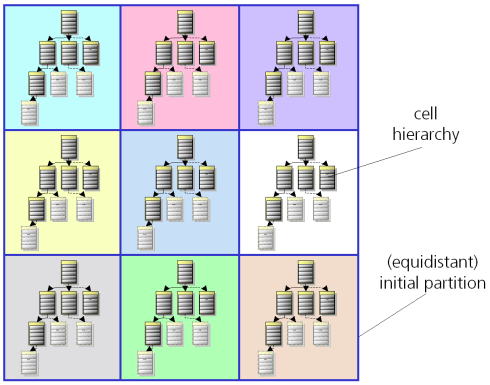


Figure 2: An example initial partition. For every rootcell of the partition a (different) cell hierarchy is created.

We define the *approximation error* at each sample point as a linear combination of the errors made for each value of a scalar- or vectorfield and the *inoutflag*.

The errors can be weighted to control their influence on the resulting grid. As, for example, the remodeling of the original grid's boundaries will have a higher priority than the scalar values (at last the boundaries define the domain where the scalar values are valid), the linear coefficient for the *inoutflag* will usually have a higher value.

4.3 Hierarchy Building And Rearranging

Of all the data contained in a cell information block, the *approximation error* is the only data made accessible to the converter, which has to sort the cells. A cell of the progressive grid is partitioned, if its approximation error is worst compared to all other currently unpartitioned cells. The partitioning generates a number of new unpartitioned 'child'-cells, which can be expected to approximate the original grid more adequate. The process stops, if either the cell size reaches the resolution of the input data (original grid) or a given minimum error tolerance has been

reached. This scheme guarantees that the maximum error in the 'leaf'-cells of the constructed hierarchy decreases as fast as possible. The available number of progression-levels matches the number of partitioned cells. At last the grid cells are stored in the same order as their 'parent'-cells have been partitioned. That way, if the progressive grid (or to be more accurate: a single hierarchy within the progressive grid), has to be loaded up to a certain error tolerance, all cells that are needed constitute one block in the storage format. Furthermore, because the grid locally adapts its resolution to the scalar field data, this circumstance can be exploited for compression in areas where low frequencies predominate.

While progressing through the cells in the way, in which they are sorted, it is guaranteed that the vertex data is arranged in the same manner. The vertex data needed to define the grid up to a certain refinement level is stored in a single block as well. As the final order of every cell and vertex is now known, all what is left to do now, is to rearrange the references for the corners, parents and children of every cell.

5 IMPLEMENTATION DETAILS

5.1 General Layout

By setting a certain changing celltype, space partitioning and interpolation, the appearance of the progressive grid will change accordingly. To support the adjustment of these 'parameters' it is possible to replace the corresponding parts of the system. As stated above, this is realized using a modular approach. The conversion consists of a fixed part, which only uses the information that is relevant for the conversion. The second part is a 'plug-in' which manages the cell-type information, its topology and interpolation schemes. Technically this is a class representing a cell of the given type. To describe the interface between these two classes, it is necessary to understand how these modules communicate during the conversion:

Initially the converter contains two main storage pools: A *cell pool* and a *vertex pool* (see 4). Whenever a cell or vertex is used, its reference is taken from the pool and the structure is updated by a routine provided by the cell type class.

The cartesian sample of the original grid, which is the data source for the conversion is stored within an array in the converter. The conversion begins with the definition of an initial partitioning. The cell class exposes to the converter, how many vertices and cells of the given type are needed for the partitioning. Both the vertices and the cells are taken from their respective pools and the partitioning is built without any interference from the converter. According to the actual representation of the celltype, the cells

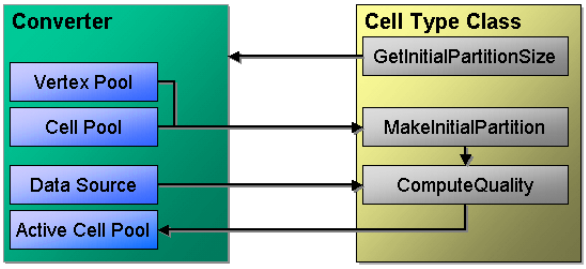


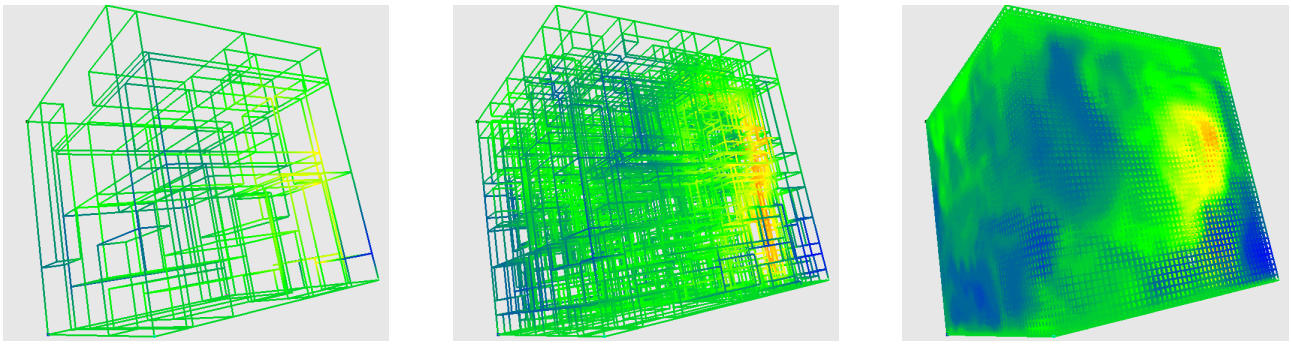
Figure 4: Flow of data between the modules making the initial partition

belonging to the partitioning are arranged in the boundingbox by defining the vertex coordinates and setting the corresponding vertex references of the cell. If necessary all known neighborhood references are set as well. At last the approximation quality of all cells of the initial grid has been defined. Using the interpolation scheme choosen for the cell type, the cell class provides a method to compute the approximation quality as well as methods to compare and query the quality of the cells. The comparision between the original and the interpolated scalar values is defined by the error metric. Because the metric is independent from the cell type, it is implemented in the converter class. Following the definition of the initial grid, the cells and vertices will be transferred back to the converter. The cells are stored in an *active cell pool* (which is realized as a priority queue) sorted by their quality. The vertices used are sorted by their spatial coordinates. In the next step of the conversion the first cell (i.e. the one with the worst quality) is removed from the *active cell pool* and its partitioning is computed and stored within the cell. Aside from using the data from the data source, the converter has no influence on this computation.

Once the partitioning is given, the 'child'-cells have to be defined. Therefore the cell class exposes the number of 'child'-cells to be taken from the pool. Another method defines the 'child'-cells using the data of the current 'parent'-cell - its localization and partitioning - by setting the cell and vertex references accordingly (see 5). If a cell is partitioned and all child cells are defined, it is removed from the *active cell pool*. The approximation quality of the 'child'-cells is computed and they are sorted into the *active cell pool* if they exist.

These steps are repeated until the *active cell pool* is empty.

The final step of the conversion involves the resorting of all cells and vertices defined, and the definition of references which are used in the storage format. This is done by the converter. And at last, the cell and vertex data will be written to the storage format, using methods of both



(a) (b) (c)

Figure 3: Different resolutions of the same progressive grid: (a) Grid structure using 50 cells, (b) using 1000 cells, (c) using the full resolution with 110286 cells. Please note the the appearance of the scalarfield differs due to the increasing resolution as well as to the decreasing 'transparency' of the grid.

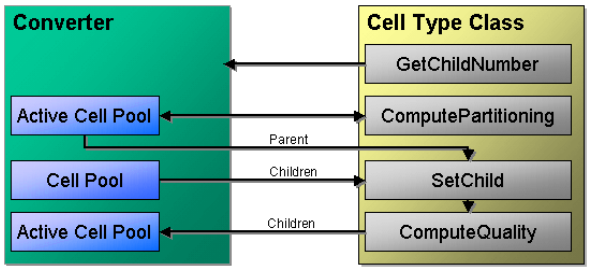


Figure 5: Flow of data between the modules while defining the cell partitionings

the converter (for the vertices) and the cell type class itself, which alone is able to write its representation to the buffer. It has to be noted that the same fileformat is used for different types of progressive grids. For this reason the visualization system has to be divided in similar way: To decouple the visualization methods from the different grid types, another 'plug-in' handles the interpretation of the data optimized for its specific type.

This visualization part of the cell type module is a mirror image of the conversion part. First of all, it provides a method to read its cell information block from the format. The main task for the visualization is to locate an arbitrary point (given in world coordinates) within a given cell, and to do the interpolation. First of all the point has to be located in a cell of the initial partition.

Depending on the type of cell, a hierarchical traversal subsequently identifies the 'child'-cells, in which the point is located in, either up to a given error tolerance or until a cell is referenced, which has not been loaded so far. Actually this makes the grids 'progressive': Every stage of refinement - from the initial partitioning up to the highest resolution grid - maintains its consistency due to this scheme.

But this scheme alone is highly dependent from current hierarchical depth of the grid. As mentioned in 3 many visualization methods compute points, which are closely related to each other. To exploit these correlations, the point location algorithm is able to start at any cell in the grid and it always returns a reference to the grid cell found. If the point lies within the current cell, the algorithm may progress to the 'child'-cells, otherwise it will traverse to the neighbor nearest to the point. If using the neighborhood information, the point location algorithm will traverse only one or two cells.

5.2 The Adaptive Binary KD-Tree

To test our concept for the progressive grids, we implemented a module which treats box-type cells along with an adaptive binary kd-tree decomposition (see fig. 3). In contrast to the likewise well-known octree scheme, we are able to control along which axis the resolution increases. Using cartesian boxes, we identified every corner of a cell with a vertex of the original sample. The data of the four new vertices defining a partitioning plane has a great influence on the quality of the partitioning. Hence we divided a cell not necessarily into two parts equal in size. So we implemented an algorithm, adapting the partitioning with respect to the original data. Our algorithm tests all possible partitioning planes defined by the discrete sample for every dimension to determine the one which is most suitable to approximate the original grid. Compared to a simple bisection algorithm, the accuracy increases more rapidly, when progressing through the levels of detail. Furthermore this improves the overall compression rates for a grid. This partitioning/converting algorithm can be parallelized in a straightforward way (as the partitioning planes

can be tested independently of each other). Even if there is still lots of room for improvement (see below), the progrisive grids lead to promising results for the future, regarding flexibility, efficiency, performance and even compression.

6 CONCLUSION AND FUTURE WORK

We presented a generic progressive format and framework, which serves as highly adaptive representation of CFD-datasets. From the developers point of view, the modular approach guarantees a high flexibility, while different kinds of progressive grids may be implemented and evaluated, without need to rebuild the whole system. From the users perspective, we are able to adapt the CFD-data resolution in different levels to the memory resources of the environment. At the same time, no concessions to the rendering performance of the grid must be made. We converted a number of datasets from fire and CFD simulations. Zooming through the according levels of detail was possible in real time. A visualization of streamlines (including 200.000 vertices) using the progressive grid was comparable in computation speed to one based on the original equidistant grid.

For the future we plan to include further enhancements like

- the evaluation and comparision of different kinds of progressive grids (especially spatio-temporal grids),
- the improvement and parallelization of the partitioning algorithm,
- the test of different error estimation schemes, especially those matching the different character of scalarfield and vectorfield data,
- the implementation of data compression schemes.
- investigate the feasibility and suitability of our format for the progressive streaming of CFD data

REFERENCES

[1] J. Wilhelms and A. van Gelder. "Octrees for faster isosurface generation". *ACM Transactions on Graphics (TOG)*, 11(3), July 1992.

[2] H. Samet. "The quadtree and related hierarchical data structures". *ACM Computing Surveys (CSUR)*, 16(2), June 1984.

[3] L. Freitag and R. Loy. "Adaptive multiresolution visualization of large data sets using a distributed memory octree". In *ACM/IEEE Conference on Super-Computing: Proceedings*, January 1999.

[4] P. Cigoni, C. Montani, E. Puppo, and R. Scopigno. "Multiresolution representation and visualization of volume data". *IEEE Transactions on Visualization and Computer Graphics 1997*, 3(4), 1997.

[5] Z. Zhu, R. Machiraju, B. Fry, and R. Moorhead. "Wavelet-based multiresolutional representation of computational field simulation datasets". In *Proceedings of the Conference on Visualization '97*, October 1997.

[6] H. Hoppe. "Progressive meshes". In *SIGGRAPH '96: Proceedings*, 1996.

[7] E. Blayo and L. Debreu. "Adaptive mesh refinement for finite difference ocean models: first experiments". *Journal of Physical Oceanography*, 29(6):1239–1250, 1999.

[8] J. Berger, M.and Oliger. "Adaptive mesh refinement for hyperbolic partial differential equations". *Journal of Computational Physics*, 53:484–512, March 1984.

[9] D.J. Holliday and G.M. Nielson. "Progressive volume models for rectilinear data using tetrahedral coons volumes". In *Symposium on Visualization*, 2000.

[10] G.H. Weber, O. Kreylos, T.J. Ligocki, J.M. Shalf, H. Hagen, B. Hamann, K.I. Joy, and K.-L. Ma. "High-quality volume rendering of adaptive mesh refinement data". In *Proceedings of 6th International Fall Workshop Vision, Modeling, and Visualization*, pages 21–23, 2001.

[11] R. Grosso, Ch. Luerig, and Th. Ertl. "The multilevel finite element method for adaptive mesh optimization and visualization of volume data". In *Proceedings Visualization '97 Conference*, 1997.

[12] R. Grosso and G. Greiner. "Hierarchical meshes for volume data". In *Proceedings Computer Graphics International '98*, pages 761 – 769, 1998.

OpenSG Symposium (2003)
D. Reiners (Editor)

Parallel architecture of an interactive scientific visualization system for large datasets

Sascha Schneider, Thorsten May †, Michael Schmidt

Fraunhofer Institut für Graphische Datenverarbeitung, Fraunhoferstr. 5, 64283 Darmstadt, Germany

Abstract

In this paper we describe a further development state of our system which is able to compute actual scientific and realistic visualization methods in parallel. This paper is related to the basic work we presented in one of our previous articles [18]. *Our system is capable to integrate easily in modern VR renderers like for example* Open Inventor [19], Coin [4] *and* OpenSG [5]. *Our approach is designed for processing large datasets which usually are the result of physically based simulation algorithms and programs. Using our techniques it is even more feasible to manage similar visualization problems for other large amounts of data (e.g. medicinal CT-scans or large geometries) in the context of displaying interactively.*

1. Introduction

Besides of the simulation of physical phenomena their performant and professional visualization comes more and more into the focus of modern scientific applications. Nowadays there are several powerful physical based simulation programs available on the software market (e.g. *Fluent* [9], *FemLab* [6], *Flovent* [12], *CFX* [3], etc.) which allow the user and developer to simulate and investigate nearly every kind of physical problem in high quality and detail. In the realisation concepts of these products mostly parallel approaches play an important role in the program architecture to gain major increases in perfomance.

Empirically these simulation programs are producing large amounts of result data which are often displayed only roughly or inperformant using simple visualization tools. These tools are mostly already integrated within the simulation programs themselves. There are only few visualization programs available which are completely independent of the underlying simulation system and/or data format, grid and glyph types (e.g. *VTK* [23]). These independent tools offer a good general approach for the visualization problem. On the other hand they often lack in performance to process the large amounts of simulation data in reasonable timeframes.

† Fraunhofer Institut für Graphische Datenverarbeitung, Tel. (+49) 6151 155-638, Fax (+49) 6151 155-139, eMail: tmay@igd.fhg.de

Very large data sets (i.e. datasets much larger than 256^3 grid points) mainly cause problems due to

- data is too large to load into main storage completely
- loading data in the hierarchically ordered memory (hard disk, cache, main memory) takes too much time for qualitative, quantitative and interactive rendering.
- costly calculations for some visualization methods.

For these reasons, techniques from the areas

- data compression (e.g. wavelet methods)
- parallelization of program code (e.g. multi threading, OpenMP, MPI)
- hardware accelerated algorithms (e.g. 3D texturing)
- efficient algorithms / software design (e.g. object oriented programming)
- utilization of efficient software development tools (e.g. C/C++) and libraries (e.g. OpenSG, OpenGL, Qt)
- development of portable, functional, easy usable and extendible software

were developed and steadily enhanced up to this day.

2. Related work

There are many works ([1], [13], [11], [21]) in computer graphics that use methods from some of the above areas. But to our knowledge, there is no comparable and published work that covers all of them. On the contrary, our approach to this topic

applies most advanced methods from all of these areas in order to create a visualization tool for very large scientific data that features high rendering quality in real time, a very good functionality and an easy handling.

On the area of displaying simulation results many visualization methods have established today, like for example iso-surfaces, stream- and time-lines and volume rendering. But compared to the massive parallel simulation program, the corresponding available visualization software is still noticable less performant. The reason for that is that it mostly works either sequentially or it is implemented as a post process which operates on the basis of an offline rendering concept.

The transfer of the principle of parallel programming from the simulation of physical problems to their interactive visualization lets the end user profit of significant accelerations. This approach enables the applications to process even larger amounts of data interactively. Interactive Visualization of scientific data is a classical area of computer graphics that is steadily and rapidly developing due to the increasing requirements on data volume, display quality, program functionality and usability.

Approaches to interactive visualization in the past partly based on completely (data pre-processing and rendering) parallelization of existing visualization algorithms. By utilizing modern graphics hardware (NVIDIA GeForce [16], ATI Radeon [8]) it's possible to shift the rendering process to them. The advantage is a faster rendering on standard PCs, that do not provide many processors for parallel execution.

3. Architecture

The main target of the following paper is to present a general concept for an interactive parallel visualization of scientific simulation data making use of the sophisticated capabilities of modern graphic cards. In this concept a generalized description of visualization methods is defined. Based on this description an extension of the developed system with further visualization methods is easily realizable every time. For that reason the system can be supplemented rapidly with new visualization methods as soon as they appear.

Furthermore our approach is realized platform independently, to satisfy the need for software portability because of different computer system architectures which are available and in use nowadays. The used ingredients (*C++*, *OpenGL* [20], *OpenSG* and *QT* [2]) allow us to fullfill this capability.

A rather new method in the field of scientific CFD visualization is to render scientific CFD data (stream lines, iso surfaces, ...) combined with detailed and textured geometrical data. This could be, for instance, a 3D model of the original scene that is used from the simulation side for generating the simulation mesh. Blending in original 3D models simplifies navigation for the user and is basis for another technical innovation, the application of highly realistic visualization methods within the area of CFD visualization. This means that in addition to visualization of abstract physical quantities like temperature, pressure, etc. with scientific visualization algorithms (cutting plane, glyphs, iso surface, ...), realistic quantities like fluid, gas or fire and smoke can be rendered as they appear in their natural form inside a photo realistic rendered virtual (simulation) environment [22].

4. Scenegraphs APIs

Scenegraphs APIs (e.g. Open Inventor / Coin, OpenSG) are immediate APIs in which the objects and commands that are going to be rendered are not sent directly to the graphics processor (GPU) but are integrated in a (acyclic directed) tree graph based description of the displayed scene. This tree is then traversed and rendered separate from the application that provides the geometry and what shall be rendered. Therefore the scenegraph implements a kind of abstraction layer between the application and the GPU. Typical Objects in a scenegraph are normally derived from a base object („scene graph tree node"): Geometry node, material node, transformation node, group node, light node.

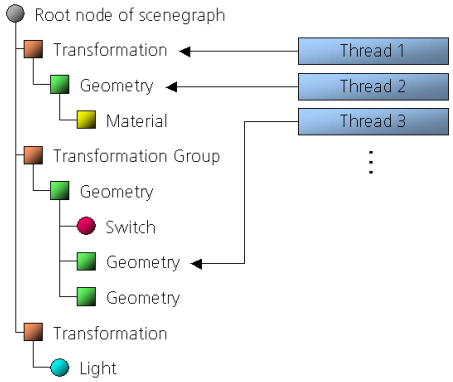


Figure 1: *Example scenegraph with multiple threads modifing it.*

Modern scenegraphs like OpenSG support parallel processing (fig. 1) natively so that it possible for the application to modify parts or nodes of the scenegraph directly from several threads at the same time. As long as every application thread addresses a different node in the graph it is not necessary to lock the whole graph each time a part of it is accessed. In spite of this it is possible that every application thread can work exclusively and in parallel on its part of the scenegraph with no risk of creating an access conflict with other nodes / threads. These scenegraphs are called „thread safe". Our visualization system is capable to use arbitrary scenegraphs for the rendering output because only the functions which produce the scenegraph nodes and the initialisation of the

Figure 2: *Visualization method: Iso-surfaces.*

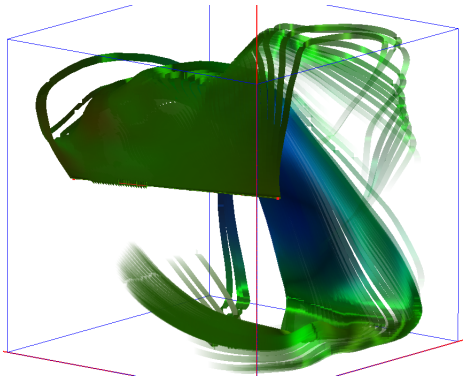


Figure 3: *Visualization method: (Shaded) stream lines.*

whole scene have to be adjusted. At the moment we are experimenting with the OpenSG scenegraph, the Open Inventor / Coin scenegraph and a specialized scenegraph which was developed by us.

Furthermore we store the geometry of the surrounding scene (e.g. a car, an airplane or a tunnel) in the selected scenegraph together with the data necessary for the rendering of the visualization methods. This information is render geometry (triangles, textures, colors, etc.) as well after the corresponding calculations have finished.

5. Visualization

As mentioned in the beginning nowadays many methods for scientic visualization have established (e.g. iso-surfaces, stream lines, time lines, etc. - (see fig. 2 and fig. 3))

As every 3D rendering, these methods can be created in

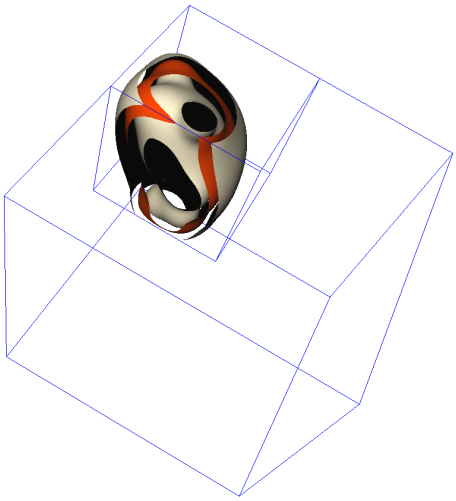


Figure 4: *Iso-surfave probe (small cube) placed in the scene (big cube).*

two ways: through ray tracing/casting on the one hand and through direct rendering using the capabilities of modern graphic cards (e.g. vertex and pixel shaders, shadowing, ...). To provide a parallel approach for the second method it is advisable to make use of thread safe scenegraphs in the first place. By doing so the application can process the calculation for each visualization method in parallel. After the calculation has finished each thread can store its calculated render information in a separate node in the scenegraph. Therefore it is easy to have several visualization methods at the same time in one displayed scene and to process each method in parallel. Furthermore each visualization method itself can be computed in parallel. By doing so, it only has to be assured, that the threaded calculations of one parallelized method itself is brought together at the same time into the scenegraph to avoid flickering effects.

5.1. Probe Concept

To allow the user to restrict visualization methods to certain areas of the datafield/scene we implemented the probe concept [13]. The user can create as many of these probes (fig. 4) and place them in the scene as he wants. Each probe is associated with a cube in VR. In this cube only one certain visualization method is calculated. These probes can be placed arbitrarily and resized within the scene so that it is possible to let the system render the desired visualization method at every place in the dataset where the user wants it to be. To have two or more visualization methods displayed at the same location at the same time it is only necessary to place the corresponding probes together at the same coordinates in VR space.

5.2. Inheritance

To allow an easy extension of the system afterwards we implemented a basic class of a visualization method. Based on this generalized description we implemented all visualization methods we needed. If the user wants to introduce a new (specialised) visualization method to the system, the only thing he has to do, is to inherit from the base (= „*root*“) class and implement its basic functions (calculation, node generation function, ...). Afterwards he only has to make sure to register his new visualization method within the system. Then he can start right off using his new method.

Additionally each visualization method introduces a user interface („panel-window“) to the system together with a list of corresponding actions / commands. Every time a feature of the visualization method is changed (e.g. the user moves a slider in the interface for the color distribution of the method) an action transporting the changed parameters is send through the inter-thread communication framework (by generating events which are collected in queues) to the central scheduler (see section 6.3). Furthermore it is possible to generate these kind of parameter changes not just by hand but by automated control over time for example („display a movie of visualization method parameter changed“).

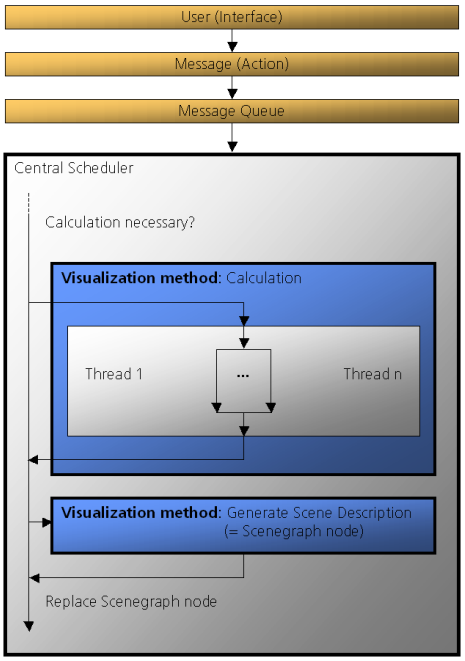


Figure 6: *The central scheduler.*

6. Parallel Concept

In this section the basic system layout (fig. 5) of the parallel visualization system is introduced showing how the different central parts of the software work together. The system is designed so that it is capable to check the capabilities of the hardware platform it is running on (e.g. count the number of available CPUs or the amount of memory). Each visualization method is implemented full scalable so that is possible to adjust the number of used calculation threads automatically by the system.

6.1. System Layout

As one can see in fig. 5 user interacts with the scene and changes the parameters of the visualization. He has influence on the view of the scene (i.e. which part of the scene is rendered from which perspective). Additionally he can create and modify probes each carrying one certain visualization method. On the other side we have the kernel - we call it „*central scheduler*“, which collects all incoming actions / events and processes them (see fig. 6).

6.2. General Visualization Method Object

As mentioned in section 5.2 each visualization method is derived from a basic class which introduces all system necessary functions for the calculation (and the rendering) as virtual functions. The calculation part of the visualization method is implemented using multiple threads to support the intended parallel processing of the system. Every time the calculation part is finished an event is generated and sent to the kernel. For the prototype, we implemented that each visualization method can define individually how many parallel processes/threads it starts for its computation. This can be decided at runtime, which allows us to test the effects of different scalability realizations for each visualization method individually.

6.3. Central Scheduler

The central scheduler is responsible for processing all necessary reactions of incoming user and / or calculation events. Every time a parameter change is generated, by user interaction or by a timer function modifying it for example, this scheduler receives a corresponding event in his incoming event queue. Being a normal thread it is then activated by the operating system having a look at his „*incoming queue*“. Afterwards the scheduler is responsible for initiating and controlling the necessary calculations triggered by the corresponding event. At the end the calculation-threads inform the central scheduler, again using inter-thread event communication, that the calculations have finished. The scheduler then initiates a scenegraph node generation of the corresponding visualization method and controls the exchange of the old node(s) in the current scene description with the new one(s).

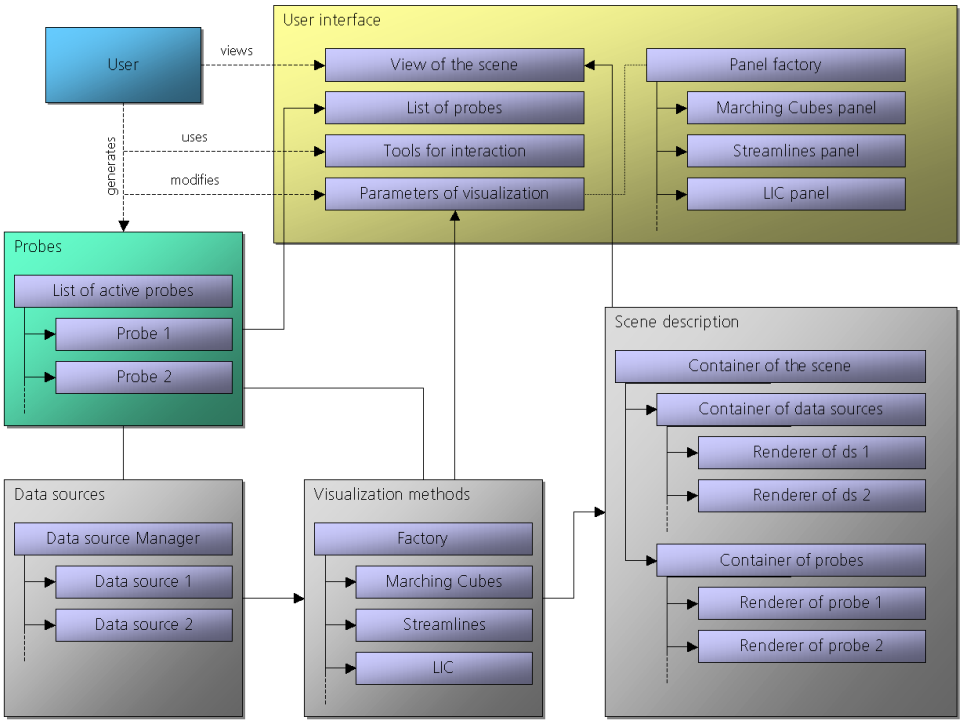


Figure 5: *The basic system layout.*

7. Datasource Management

In order to adapt the data source management to the resources of the system used, we developed data structures which allow us to trade off accuracy, quality and rendering performance: the *progressive grids* [14]. The progressive grids are a special kind of *hierarchically structured grids* [10], which originate from the field of computer graphics. There they are used to spatially arrange large amounts of geometry data. Progressive grids make use of the technology of progressive data formats which are closely related to digital image processing (e.g. *Wavelet-*, *JPEG-Compression*). In these format the data is ordered by its level of detail. So the data associated with a given arbitrary resolution can be extracted efficiently. As a result the amount of data that has to be transmitted and/or processed can be freely adapted to variable system resources or user requirements. Geometry data, for example, can be transmitted and displayed simultanously in incremental granularity levels thus the viewer gets an impression of the geometry already with beginning of the transmission [7]. While using progressive data formats in the context of CFD simulation data, we take advantage of these properties. It makes sense to arrange the data with respect to the information it contains. According to this alignment we built a hierarchy consisting of a spatial partitioning scheme [17] that has so far been used in computer graphics or geometry.

With our work we introduce the principle of progressive data processing to CFD-data. This efficient grid class is able to replace the current unoptimized grid classes used in scientific visualization systems (see fig. 7) completely [15]. Actually no grid used in numerical simulation is able to handle its data progressively. So these grids have to be converted into the progressive format to make use of them in our visualization system. For this conversion we are free to choose which cell types, partitioning schemes or error estimation schemes are used in concrete. All these parameters can be selected independently from each other and this offers a rich repertoire of possibilities. According to this, the converter is divided in two parts: A fixed one and a plug-in, which manages cell type information, its topology, interpolation schemes etc. The plug-in part can be replaced in order to implement different progressive grids (i.e. grids which use different cell types, partitioning schemes etc.).

The conversion itself works in the following way: The bounding box of the original simulation grid becomes the *root cell* of our progressive grid. Using the predefined decomposition scheme, a hierarchy of grid cells is then built up through spatial partitioning. An approximation error is computed through comparision of values interpolated within the current cell and the original grid. (This approximation is independent from the topology of the original grid.) A cell will be further partitioned, if its approximation error is the

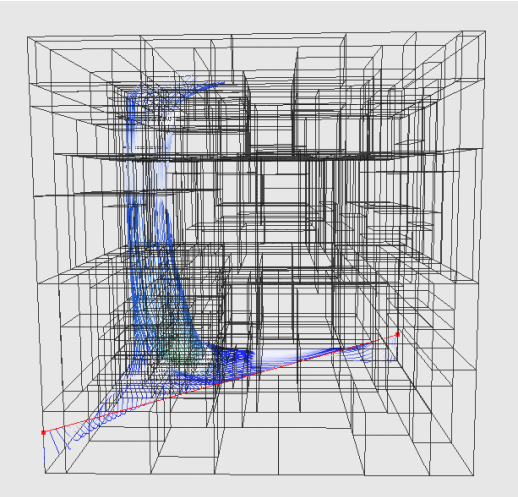


Figure 7: *A Progressive Grid together with a correspondig streamline visualization.*

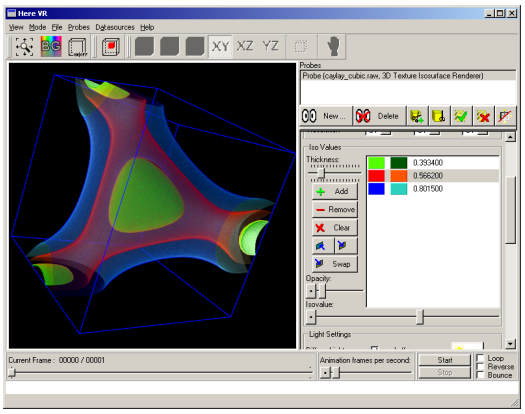


Figure 8: *Screenshot of the running application*

worst compared to all other currently unpartitioned cells . The new cells generated in this way, represent a *„better“* approximation of the original grid. The whole partitioning process stops if a certain error tolerance has been reached.

The progressive grid generated in this way has a number of advantages. The maximum error within the leaf-cells in the hierarchy decreases fast. This minimizes the number of cells to be loaded at a given error tolerance. Furthermore these cells constitute a single block, because they are written in the same order their parent cells have been partitioned. The hierarchical structure of the grid can be exploited for compression in areas where low-frequency portions of the scalar fields predominate. We converted a number of datasets of fire simulations and are able to zoom through the levels of detail in real time, without making concession to performance. A visualization of streamlines (involving 200.000 vertices) using the progressive grid was comparable in speed to the one on the original, equidistant grid.

8. Summary

We presented a new concept of a visualization system which is able to make use of the capabilities of modern graphic cards. This system is scalable and can be easily adjusted to different hardware conditions. Furthermore it is portable and can be easily extended with new visualization methods. Together with its capability to display scientific visualization methods together with realistic rendered it is very attractive to the end user, because he is able to investigate his simulation results within a realistic looking virtual environment. The parallel approach of our system makes it very attractive for processing very large amounts of (simulation) data. Based on the progressive approach it becomes possible to visualise even large datasets on machines which have only little performance only at the cost of losing details in the loaded and displayed data.

For the future is is planned to have a kind of automatic adjustment of how many processes/threads should be started for each visualization method („profiler“) depending on the used hardware capabilities. This idea is well supported by our system design (see section 6.2).

References

1. R. M. Aaron Trott and J. McGinley. Wavelets applied to lossless compression and progressive transmission of floating point data in 3-d curvilinear grids. In *Proceedings IEEE Visualization '96*, pages 385–388. IEEE, 1996. 1

2. T. AS. Qt, c++ toolkit for application development. http://www.trolltech.com/products/qt/. 2

3. CFX. Cfx, cfd software package. http://www.software.aeat.com/cfx/. 1

4. Coin3D. Coin, scenegraph based 3d graphics library. http://www.coin3d.org/. 1

5. J. B. Dirk Reiners, Gerrit Voß. Opensg - basic concepts. http://www.opensg.org/OpenSGPLUS/symposium/Papers2002/. 1

6. Femlab. Femlab, pde solver package. http://www.femlab.com/femlab/. 1

7. H. Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings*, 1996. 5

8. A. T. Inc. Ati radeon, graphic board. http://www.ati.com/. 2

9. F. Inc. Fluent, cfd software package. http://www.fluent.com/software/fluent/. 1

10. A. v. G. J. Wilhelms. Octrees for faster isosurface generation. *ACM Transactions on Graphics (TOG)*, 11(3), 1992. 5

11. R. M. L. Lori A. Freitag. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *Proceedings of SC99: High Performance Networking and Computing*, 1999. 1

12. F. Ltd. Flovent, cfd based software. http://www.flovent.com/. 1

13. W. B. M. Schulz, F. Reck and T. Ertl. Interactive visualization of fluid dynamics simulations in locally refined cartesian grids. In *Proceedings IEEE Visualization '99*, pages 413–553. IEEE, 1999. 1, 3

14. T. May, S. Schneider, M. Schmidt, and V. Luckas. Fast scalar- & vectorfield visualization using a new progressive grid class. In *to be published at the High Performance Computing Conference (HPC)*, 2003. 5

15. T. May, S. Schneider, M. Schmidt, and V. Luckas. The progressive grid: Introducing a new grid class for efficient cfd visualization. In *to be published at the Simulation and Visualization Conference (SimVis)*, 2003. 5

16. NVIDIA. Geforce, graphic processing unit. http://www.nvidia.com/. 2

17. H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2), June 1984. 5

18. S. Schneider, T. May, and M. Schmidt. Rendering large (volume) datasets: A new parallel visualization system. In *Journal of WSCG*, pages 418–424, February 2003. 1

19. SGI. Open inventor, object oriented 3d graphics api. http://www.sgi.com/software/inventor/. 1

20. SGI. Opengl, 3d rendering api. http://www.opengl.org/. 2

21. M. L. Szymon Rusinkiewicz. Qsplat: A multiresolution point rendering system for large meshes. In *Siggraph 2000: Computer Graphics Proceedings*, 2000. 1

22. V. L. Thorsten May, Sascha Schneider. Parallel real time fluid simulation and animation with fractal optical refinements. In *ESM ´02: Proceedings of the 16th European Simulation Multiconference, Modelling and Simulation 2002*, pages 224–228, 2002. 2

23. K. M. M. William J. Schroeder and W. E. Lorensen. The design and implementation of an object-oriented toolkit for 3d graphics and visualization. *IEEE Visualization '96*, http://public.kitware.com/VTK/:93–100, 1996. 1

Fast scalar- & vectorfield visualization using a new progressive grid class

Thorsten May,* Sascha Schneider, Michael Schmidt, Volker Luckas
Fraunhofer Institut für Graphische Datenverarbeitung

Abstract

We present a new generic progressive data format and framework for CFD-data, which is specialized for scientific data simulation: The progressive grids. While using arbitrary types of cells and partitioning schemes, simulation data can be stored progressively. Hierachically arranged by its level of detail, the data can be adapted efficiently to many systems with different ressources. Maintaining a modular approach, we are able to implement, test and update different kinds of progressive grids by adjusting the corresponding parts of the system.

1 Introduction

The ever increasing computational power of computer workstations and clusters used within the field of numerical simulation produces a comparable increasing mass of simulation data results. These results are generally accessible by means of scientific visualization. Today, the sheer mass of data makes it impossible to display the data on low-end-systems (e.g. laptops used for presentations) with an acceptable performance. It is essential to use data structures, which allow to trade off accuracy, quality and rendering performance as well as to adapt them to the system resources of the system used. In effect, this means to load and display only a certain part of the simulation data. It is crucial to guarantee that this part contains the most important information: With only a small portion loaded one will be enabled to qualitatively explore the whole simulation domain in real time. The progressive grids developed in this context fulfill this goal.

2 Related work

Progressive grids are a certain form of hierarchically structured grids. Hierarchically structured grids are well known in the field of computer graphics [WvG92, Sam84]. In this context they are used to spatially arrange large amounts of geometry fragments in a way that one is able to decide for any given cell which fragments it contains. These grids are adaptive, because the cells are divided if the enclosed data exceeds a certain threshold.

*Fraunhofer Institut für Graphische Datenverarbeitung, Fraunhoferstr. 5, 64283 Darmstadt, Germany, Tel. (+49) 6151 155-638, Fax (+49) 6151 155-139, eMail: tmay@igd.fhg.de

Our progressive grids make use of the technology known from progressive data formats, which are closely related to digital image processing (Wavelet-, JPEG-Compression) [FL99, CMPS97, ZMFM97]. In these formats the data is ordered by its level of detail. So the data associated with a given arbitrary resolution can be extracted efficiently. As a result the amount of data that has to be transmitted and/or processed can be freely adapted to variable system resources or the users requirements. Geometry data, for example, can be transmitted and displayed simultaneously in incremental granularity levels thus the viewer gets an impression of the geometry already with the beginning of the transmission [Hop96].

While using progressive data formats in the context of CFD simulation data, we take advantage of these properties. We arrange the data with respect to the information it contains, beginning with the most important ones. According to this alignment, we build a hierarchy that is represented in a spatial partitioning scheme that has so far been used in computer graphics. We now put it into a new context: the management of CFD simulation data.

The concept of *adaptive mesh refinement(AMR)* is well-known in the field of numerical computation [BD99] for the solution of partial differential equations using finite differences. The AMR-method [Ber84] is based on the principle of making recursive local refinements to a given coarse grid. The refinements are controlled by an error, which is determined by suitable error metric. The basic idea of this method is to execute numerical computations only where they are necessary.

Other than in the field of numerical analysis and scientific computation there are only a few related approaches in the area of scientific visualization. In [HN00] a method is proposed, which decomposes a rectilinear mesh into tetrahedra and octahedra and approximates the original values by using triangular *coon patches*. If the resulting error from the initial decomposition is too large, the partitioning continues locally. [WKL$^+$01] utilizes hierarchically structured rectilinear grids but focusses on the interpolation scheme that avoids discontinuities at the boundaries facing different subdivision levels.

Another approach for the generation of hierarchical meshes is described in [GLE97]. Actually, the authors also build the hierarchy from the coarsest level or coarsest mesh respectively, but the approximation problem is shifted to the *finite element space*. The input data is regarded as a discretization of a smooth function, which will be approximated by an adaptive mesh refinement and error control starting from a discrete function that is defined on a coarse mesh. In [GG98] is specified the associated adaptive mesh algorithm that is based on an adaptive and regular refinement of a coarse start grid by a consistent decompostion into tetreahedra and octahedra.

Adaptive visualization methods, rather than data formats, are presented in [OR97, NORS97]. For instance in an isosurface extraction algorithm a trade off between accuracy and performance is achieved be setting an initial threshold for the *visual*

error. A *procedural interface* connecting arbitrary meshes to the visualization methods is proposed in [RSS96]. If the user provides a set of methods to access the grid data, no conversion between the numerical and the visualization grid is necessary.

A hierarchical representation of vector fields is described in [HWHJ99]. First, the vector field is divided into two disjoint clusters. The vectors in each cluster are averaged and approximated by a single vector. For a given point in the field, two streamlines are calculated using a Runge-Kutta-method. The first one is based on the original field and the second one is based on the corresponding approximation. If the deviation of the two streamlines within a cluster is greater than a given threshold the cluster will be subdivided. The method creates a hierarchy of *BSP-trees* and saves memory resources, because it is gridfree. In [GPR+01] a physical clustering model, the *Cahn-Hilliard-Model*, which describes the phase separation is used in order to build a continous multiscale clustering on vector fields.

3 Concept

With our work we adopt the principle of progressive data processing to CFD data resp. discrete scalar- and/or vectorfields. Since any grid which is used in numerical simulation is unable to handle its data progressively, these grids have to be converted first. Afterwards the data is available in a format which is specially enhanced to meet the demands of a visualization system (see figure 1).

Figure 1: The orignal CFD grid is converted into a progressive grid.

The basis for a progressive grid and the input data for the conversion algorithm is a regular discretization of the domain. We can choose if this discretization should originate from the simulation grid itself or a from a regular sample of an irregular or even unstructured grid. In the following we describe how a progressive grid is organized an how the conversion works.

First of all, it has to be stated that our progressive grids are *generic*. They are implemented in a conversion framework. It is thereby possible to choose the topological structure arbitrarily: It is possible to select which cell types or spatial partitioning schemes will be used. They define the topology of the progressive grid and how the given grid data has to be interpreted. The same holds true for the interpolation schemes (defining the error estimation), used to compare the progressive grid and its parent - the numerical grid. This approach even allows us to couple different time steps in a four-dimensional grid to make

use of the temporal coherencies for compression.

As stated above, we represent our grid in a spatial partitioning scheme, which in turn means that all cells contained in it are arranged in a hierarchical order. The actual numerical information is either associated with a given cell or with the vertices defining its corners. In any case, the partitioning of a cell introduces a certain number of vertices along with at least two new childcells. Their data will contribute to provide a greater local accuracy.

3.1 The grid cells and their data

The *cell* is the elemental structure in the progressive grid. In an abstract sense, it stores

- references to the vertices at its corners,
- the information if and how the cell is partitioned,
- references to its ’parent’ and ’child’-cells (if any)
- numerical information associated with the cell,
- references to a certain number of neighboring cells (if necessary).

This storage structure can be illustrated in a directed acyclic graph *(see fig. 3)*. If possible, the corners of the cells should be vertices of the original regular sample. As there is no need for the progressive grid to be finer than the original one, this helps saving an interpolation step afterwards. For cell types other than boxes, however, this condition may be not satisfied. The vertices contain their coordinates (being of arbitrary dimension) - this defines the location of a cell - and the scalar values derived from the original grid at their position. Furthermore - if the input data is able to provide this information - every vertex has a flag which marks if it is located within the surrounding original grid, which is not necessarily the case (referred to as the *inoutflag*). The reason for a cell to store only references to the vertices is that the single vertex is obviously shared by a number of neighboring cells and by a number of cells belonging to the same hierarchy. A standard grid cell is illustrated in figure 2.

Most important of the numerical information associated with a cell (further referred to as *volume data*) for the conversion algorithm is the *approximation error*. It describes the approximation quality compared to the original grid. Additionally, some information related to the different scalar fields (e.g. their minimum, average and maximum values within a cell) is provided.

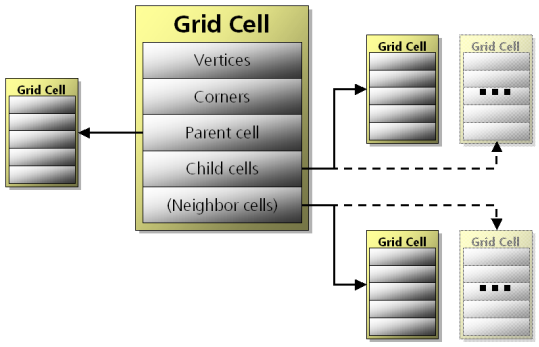


Figure 2: A grid cell.

4 The conversion scheme

4.1 The initial partitioning

The main task during conversion is to build up the cells hierarchy. This is the most time consuming job. Furthermore the converter has to define the order in which the cell information blocks have to be stored. Since the conversion scheme should remain independent from the given topology or data of the progressive grid, the construction of the cell's hierarchy is handed over to the module mentioned above.

The conversion starts with an initial partition. In the simplest case, using box-type cells, the grid can be based on the bounding box of the simulation grid. In the case of a tetrahedral decomposition, the grid is rooted in a special initial partitioning of the original bounding box into a number of tetrahedrons. However, with a given cell type, we are not restricted to a certain initial decomposition. This allows the generation of more than one rootcell (and an associated hierarchy). In this case, the module has to provide the mechanism to identify the rootcell in which a given point is located in. To support more than one initial partition of the original domain is advantagous for at least two reasons: First of all the conversion can be parallelized very easy since the work to be done for every partition can be done in parallel without any overhead due to synchronization. The second and most important advantage is, that while using a progressive grid that is converted into several independent hierarchies, we are able to define a level of detail for every single spatial domain (resp. its hierarchy). For scenes bigger than the viewing frustrum, this supports a view-dependent level-of-detail visualization. In fig. 4 an example initial partition is shown.

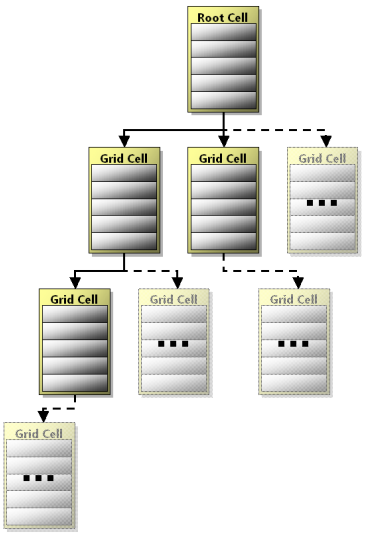


Figure 3: Severaly cells referencing each other in a graph like hierarchy.

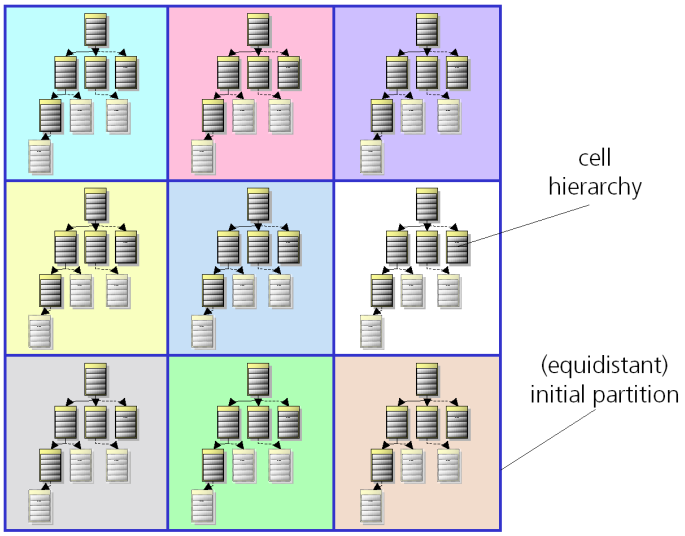


Figure 4: An example initial partition. For every rootcell of the partition a (different) cell hierarchy is created.

4.2 Determining the approximation quality

Given the partitioning scheme, a hierarchy of grid cells is built up through spatial partitioning. Within a cells boundary the data is interpolated using a selected interpolation scheme (see section 3). For every unpartitioned cell, an approximation error is computed by comparision of values interpolated within the current cell and values given be the original grid at a set of sample points. This approach guarantees that the approximation by the progressive grid is independent from the topology of the original grid. The independency has one drawback, though: As no information from the original grid is presented, except for the values at the sample points, the right choice of the samples points is crucial for the conversion. Just choosing the vertices of the original grid can lead to severe artifacts, because cells of considerable size in the progressive grid may not contain a single vertex of the original one - even if the original grid would intersect the cell. For this reason we choose to predefine a regular sample of the original grid, which is fine enough to cover all its details and compare the interpolation within the current cell with the regular sample.

We define the *approximation error* at each sample point as a linear combination of the errors made for each value of a scalar- or vectorfield and the *inoutflag*. It is possible to choose which subset of scalar values actually is written to the resulting grid, as well as the subset which is considered relevant for the error estimation. According to this, a number of grids each adapted to a single scalar value can be generated. However we noticed that the resulting - slightly better - compression rates are not worth the effort of managing all those grids in virtually all cases. The errors can be weighted to control their influence on the resulting grid. As, for example, the remodeling of the original grid's boundaries will have a higher priority than the scalar values (at last the boundaries define the domain where the scalar values are valid), the linear coefficient for the *inoutflag* will usually have a higher value.

4.3 Building and rearranging the hierarchy

Of all the data contained in a cell information block, the *approximation error* is the only data made accessible to the converter, which has to sort the cells. A cell of the progressive grid is partitioned, if its approximation error is worst compared to all other currently unpartitioned cells. The partitioning generates a number of new unpartitioned childcells, which can be expected to approximate the original grid more adequate. The process stops, if either the cell size reaches the resolution of the input data (original grid) or a given minimum error tolerance has been reached. This scheme guarantees that the maximum error in the leafcells of the constructed hierarchy decreases as fast as possible. The available number of progression-levels matches the number of partitioned cells. At last the grid cells are stored in the same order as their parentcells have been partitioned. That way, if the progressive grid (or to be more accurate: a single hierarchy within the progressive grid), has to be loaded up to a certain error tolerance, all cells that are needed constitute one block in the storage format. Furthermore, areas where the low frequencies of the scalar fields predominate can be represented with comparably few cells even at low

error tolerance rates because the grid locally adapts its resolution to the scalar field data as mentioned above. Since typically every CFD dataset contains areas of low frequency, this circumstance can be exploited for compression. This especially holds true for a dataset derived from an adaptive numerical grid as well.

After the sorting of the cell information blocks, an order for the vertex data, which is stored in an independent array for each hierarchy involved, is defined. To maintain the advantages of the progressive grid, a vertex is sorted into this array, as soon as it is referenced as a corner of a cell. If a vertex is referenced repeatedly, only the first occurence will be recorded.

While progressing through the cells in the way, in which they are sorted, it is guaranteed that the vertex data is arranged in the same manner. The vertex data needed to define the grid up to a certain refinement level is stored in a single block as well. As the final order of every cell and vertex is now known, all what is left to do now, is to rearrange the references for the corners, parents and children of every cell.

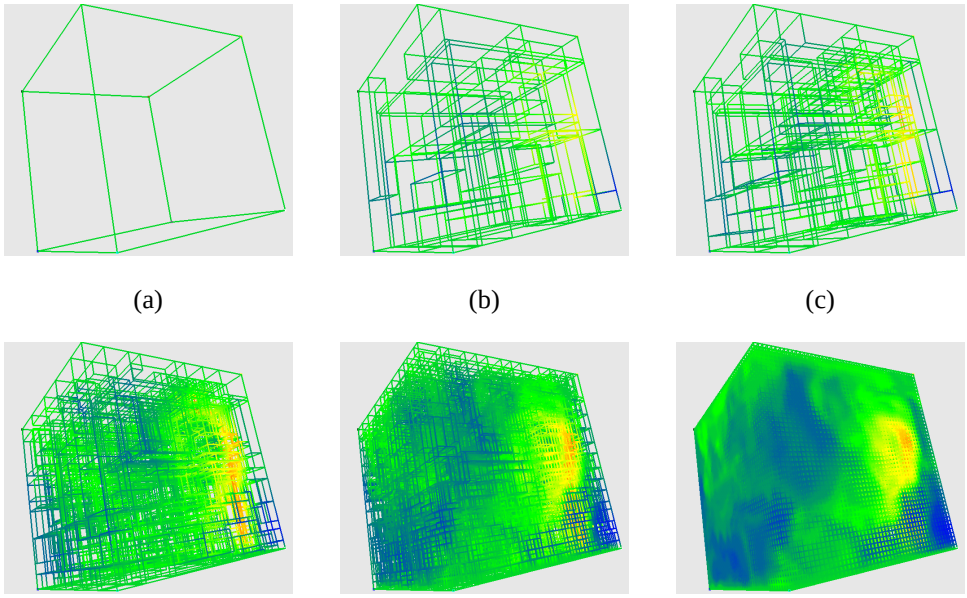
5 Implementation Details

5.1 General Layout

By setting a certain changing celltype, space partitioning and interpolation, the appearance of the progressive grid will change accordingly. To support the adjustment of these „parameters“ it is possible to replace the corresponding parts of the system. This is realized using a modular approach, in which the converter is split into two main parts, each communicating via an abstract interface. The conversion consists of a fixed part, which only uses the information, which is relevant for the conversion. The second part is a „plug-in“ which manages the cell-type information, its topology and interpolation schemes. Technically this is a class representing a cell of the given type. It must be capable to create its partitioning, to write its data to a buffer and to exposes some general information to the converter (e.g. the size of the buffer needed, the size of the initial partitioning, references to its children or the approximation error). It has to be noted that the same fileformat is used for different types of progressive grids. For this reason the visualization system is divided in a similar way: To decouple the visualization methods from the different grid types, another ’plug-in’ (a codec) handles the interpretation of the data optimized for its specific type.

5.2 Cell Types

Two cell types a are worth a closer look: *Boxes* are easy to handle, their corners can be associated with vertices from the original sample and they provide efficient containment



(a) (b) (c)

(d) (e) (f)

Figure 5: Different resolutions of the same progressive grid: (a) grid structure at hierarchy level 0 resp. using 1 cell. (b) using 50 cells. (c) using 100 cells. (d) using 1000 cells. (e) using 10000 cells. (f) using the full resolution with 110286 cells. Please note the the appearance of the scalarfield differs due to the increasing resolution as well as to the decreasing 'transparency' of the grid.

tests and interpolation schemes, which can be crucial for the performance of the visualization system. On the other side, using *tetrahedral* cells leads to a better approximation of irregular grids, since their faces are not confined to be aligned to the axis, which influences the overall compression rates. When using tetrahedral cells, their corners will usually not match to vertices in the regular sample and therefore it has to be guaranteed that they contain at least a single vertex of the original data, which in turn means to prevent the tetraedhrons getting too small and „degenerate".

5.3 The adaptive binary kd-tree

To test our concept for the progressive grids, we implemented a module which treats box-type cells along with an adaptive binary kd-tree decomposition (see fig. 5). In contrast to the likewise well-known octree scheme, we are able to control along which axis the resolution increases. Using cartesian boxes, we identified every corner of a cell with a vertex of the original sample. The data of the four new vertices defining a partitioning plane has a

great influence on the quality of the partitioning. Hence we divided a cell not necessarily into two parts equal in size. So we implemented an algorithm, adapting the partitioning with respect to the original data. Our algorithm tests all possible partitioning planes defined by the discrete sample for every dimension to determine the one which is most suitable to approximate the original grid. Compared to a simple bisection algorithm, the accuracy increases more rapidly, when progressing through the levels of detail. Furthermore this improves the overall compression rates for a grid. As this algorithm does an exhaustive search, it performs rather poor while converting a grid of considerable size. Of course even this partitioning/converting algorithm can be parallelized in a straightforward way (as the partitioning planes can be tested independently of each other). But there is still some more room left for optimization. The idea is not to use the error metrics defining the quality of a cell approximation as above, which includes the computation of an approximation error for each sample point and each partitioning plane. Instead, the subcells in itself can be made as homogenous as possible.

Standard bi- or trilinear interpolation for 2D or 3D data respectively are the most simple schemes for a continous approximation of a given scalar field. Roughly speaking, the accuracy of these approximation schemes depends on the gradient field and the interpolation will be exact if the gradient field is constant within a given cell. For this reason a partitioning is defined so that it gathers vertices with equal or similar gradients in a subcell as big as possible. Usually, if a bigger subcell is chosen, the variation of the gradients increases (as more vertices are collected in the cell). In an abstract sense the homogeneity of a cell is a measure to assess its size against these variation of the gradients. An optimal partitioning will maximize the homogeneity in both subcells. This eliminates the disadvantage of using the approximation error to test different partitionings: Even if the subcells in different partitionings overlap nearly completely, there is no way to correlate the error computation results, because they mainly depend on the values at the subcells corners (at last they are the interpolation coefficients), which can change arbitrarily. Furthermore we gain the opportunity to do some precomputation.

The following formulae refer to 2D space and a partitioning along the first axis, but it can easily be extended to any dimension. Since we use a regular sample of the domain we identify all vertices by their cartesian indices $v_{i,j}^{\rightarrow}$. Let $g_{i,j}^{\rightarrow}$ be the gradient of the scalarfield at the corresponding vertex. A fairly simple yet efficient measure to assess the homogeneity H of gradients within a given cell B (defined by indices $i_{low}, i_{high}, j_{low}, j_{high}$) is the norm of the sum of all affected gradient vectors:

$$H(B) = \left\| \sum_{i=i_{low}}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} g_{i,j}^{\rightarrow} \right\|_2^{\alpha} \tag{1}$$

That way the homogeneity of a region is nearly proportional to its size if the gradients are similar to each other. $\alpha(> 1)$ defines a 'design'-parameter describing a tendency of bigger regions to be more selective when including new gradients. The partitioning will divide the cell along a given axis into two subcells. To evaluate the quality of a cell partitioning p we

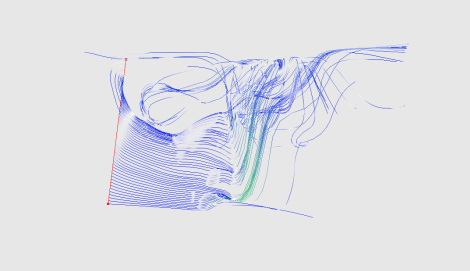
add the homogeneities of the subcells.

$$H_p(B) = \left\| \sum_{i=i_{low}}^{p} \sum_{j=j_{low}}^{j_{high}} \vec{g_{i,j}} \right\|_2^{\alpha} + \left\| \sum_{i=p+1}^{i_{high}} \sum_{j=j_{low}}^{j_{high}} \vec{g_{i,j}} \right\|_2^{\alpha}, p = (i_{low}+1)...(i_{high}-2) \quad (2)$$

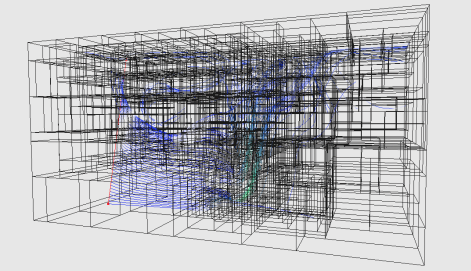
In contrast to the computation of the approximation error, this scheme can be optimized by a preprocessing step which sums up all gradients belonging to the same layer, because the relevant data for the layer can be expressed in a single vector.

$$h_l = \sum_{j=j_{low}}^{j_{high}} \vec{g_{l,j}} \quad (3)$$

The partitioning that maximizes $H_p(B)$ will be chosen to split up the cell into its subcells.



(a)



(b)

Figure 6: Stream line visualization using our progressive grids: (a) Just the calculated stream lines. (b) Stream lines together with the corresponding progressive grid.

6 Conclusion and Future Work

We presented a generic progressive format and framework, which serves as highly adaptive representation of CFD-datasets. From the developers point of view, the modular approach guarantees a high flexibility, while different kinds of progressive grids may be implemented and evaluated, without need to rebuild the whole system. From the users perspective, we are able to adapt the CFD-data resolution in different levels to the memory resources of the environment. At the same time, no concessions to the rendering performance of the grid must be made. We converted a number of datasets from fire and CFD simulations (see fig. 6). Zooming through the according levels of detail was possible in real time. A visualization of streamlines (including 200.000 vertices) using the progressive grid was comparable in computiation speed to one based on the original equidistant grid.

For the future we plan to include further enhancements like

- the evaluation and comparision of different kinds of progressive grids (especially spatio-temporal grids),
- the improvement and parallelization of the partitioning algorithm,
- the test of different error estimation schemes, especially those matching the different character of scalarfield and vectorfield data,
- the implementation of data compression schemes.
- investigate the feasibility and suitability of our format for the progressive streaming of CFD data

References

[BD99] E. Blayo und L. Debreu. *Adaptive mesh refinement for finite difference ocean models: first experiments*. Journal of Physical Oceanography, 29(6):1239–1250, 1999.

[Ber84] J. Berger, M.and Oliger. *Adaptive mesh refinement for hyperbolic partial differential equations*. Journal of Computational Physics, 53:484–512, March 1984.

[CMPS97] P. Cigoni, C. Montani, E. Puppo und R. Scopigno. *Multiresolution Representation and Visualization of Volume Data*. IEEE Transactions on Visualization and Computer Graphics 1997, 3(4), 1997.

[FL99] L. Freitag und R. Loy. *Adaptive Multiresolution Visualization of Large Data Sets using a Distributed Memory Octree*. In: ACM/IEEE Conference on SuperComputing: Proceedings, January 1999.

[GG98] R. Grosso und G. Greiner. *Hierarchical meshes for volume data*. In: Proceedings Computer Graphics International '98, S. 761 – 769, 1998.

[GLE97] R. Grosso, Ch. Luerig und Th. Ertl. *The Multilevel Finite Element Method for Adaptive Mesh Optimization and Visualization of Volume Data*. In: Proceedings Visualization '97 Conference, 1997.

[GPR+01] H. Garcke, T. Preußer, M. Rumpf, A. Telea, U. Weikard und J. van Wijk. *A Phase Field Model for Continuous Clustering on Vector Fields*. In: IEEE Transactions on Visualization and Computer Graphics, S. 230–241, 2001.

[HN00] D.J. Holliday und G.M. Nielson. *Progressive Volume Models for Rectilinear Data using Tetrahedral Coons Volumes*. In: Symposium on Visualization, 2000.

[Hop96] H. Hoppe. *Progressive Meshes*. In: SIGGRAPH '96: Proceedings, 1996.

[HWHJ99] B. Heckel, G. Weber, B. Hamann und K.I. Joy. *Construction of Vector Field Hierarchies*. In: Proceedings of IEEE Visualization '99, S. 19–26, October 1999.

[NORS97] R. Neubauer, M. Ohlberger, M. Rumpf und R. Schwörer. *Efficient Visualization of Large-Scale Data on Hierarchical Meshes*. In: Visualization in Scientific Computing, 1997.

[OR97] M. Ohlberger und M. Rumpf. *Hierarchical and Adaptive Visualization on Nested Grids*. Computing, 59(4):269–285, 1997.

[RSS96] M. Rumpf, A. Schmidt und K. Siebert. *Functions defining arbitrary meshes, a flexible interface between numerical data and visualization routines*. In: Computer Graphics Forum, S. 129–141, 1996.

[Sam84] H. Samet. *The Quadtree and Related Hierarchical Data Structures*. ACM Computing Surveys (CSUR), 16(2), June 1984.

[WKL+01] G.H. Weber, O. Kreylos, T.J. Ligocki, J.M. Shalf, H. Hagen, B. Hamann, K.I. Joy und K.-L. Ma. *High-quality Volume Rendering of Adaptive Mesh Refinement Data*. In: Proceedings of 6th International Fall Workshop Vision, Modeling, and Visualization, S. 21–23, 2001.

[WvG92] J. Wilhelms und A. van Gelder. *Octrees for Faster Isosurface Generation*. ACM Transactions on Graphics (TOG), 11(3), July 1992.

[ZMFM97] Z. Zhu, R. Machiraju, B. Fry und R. Moorhead. *Wavelet-based Multiresolutional Representation of Computational Field Simulation Datasets*. In: Proceedings of the Conference on Visualization '97, October 1997.

Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization (2003)
G.-P. Bonneau, S. Hahmann, C. D. Hansen (Editors)

Fast & flexible representation of CFD-data: Progressive Grids

T. May[1†], S. Schneider[1], M. Schmidt[1] and V. Luckas[1]

[1] Department for Animation and Image Communication, Fraunhofer Institute for Computer Graphics, Darmstadt, Germany

Abstract

We present a new generic progressive data format and framework for CFD-data, which is specialized for scientific data simulation: The progressive grids. While using arbitrary types of cells and partitioning schemes, simulation data can be stored progressively. Hierachically arranged by its level of detail, the data can be adapted efficiently to many systems with different ressources. Maintaining a modular approach, we are able to implement, test and update different kinds of progressive grids by adjusting the corresponding parts of the system.

1. Introduction

The ever increasing computational power of computer workstations and clusters used within the field of numerical simulation produces a comparable growing mass of simulation data results. These results are generally accessible by means of scientific visualization. Today, the mass of data makes it impossible to display all the available data straight forward on low-end-systems (e.g. laptops used for presentations) with an acceptable performance. It is therefore essential to use data structures, which allow to trade off accuracy, quality and rendering performance as well as to adapt them to the resources of the used (hardware) system. In effect, this means to load and display only a certain part of the simulation data. It is crucial to guarantee that this part contains the most important information: With only a small portion loaded the user will be enabled to qualitatively explore the whole simulation domain in real time. The progressive grids developed in this context fulfill this goal.

2. Related work

Progressive grids are a certain form of hierarchically structured grids. Hierarchically structured grids are well known in the field of computer graphics [16, 15]. In this context they are used to spatially arrange large amounts of geometry fragments in a way that one is able to decide for any given cell which fragments it contains. These grids are adaptive, because the cells are divided if the enclosed data exceeds a certain threshold.

Our progressive grids make use of the technology known from progressive data formats, which are closely related to digital image processing (Wavelet-, JPEG-Compression) [5, 12, 17]. In these formats the data is ordered by its level of detail. So the data associated with a given arbitrary resolution can be extracted efficiently. As a result the amount of data that has to be transmitted and/or processed can be freely adapted to variable system resources or the users requirements. Geometry data, for example, can be transmitted and displayed simultaneously in incremental granularity levels thus the viewer gets an impression of the geometry already with the beginning of the transmission [9].

While using progressive data formats in the context of CFD simulation data, we take advantage of these properties. We arrange the data with respect to the information it contains, beginning with the most important ones. According to this alignment, we build a hierarchy that is represented in a spatial partitioning scheme that has so far been used in computer graphics. We now put it into a new context: the management of CFD simulation data.

The concept of *adaptive mesh refinement(AMR)* is well-known in the field of numerical computation [3] for the solution of partial differential equations using finite differences. The AMR-method [1] is based on the principle of making recursive local refinements to a given coarse grid. The refinements are controlled by an error, which is determined by suitable error metric. The basic idea of this method is to execute numerical computations only where they are necessary.

Other than in the field of numerical analysis and scientific computation there are only a few related approaches in the area of scientific visualization. In [4] a method is pro-

† Web: http://www.igd.fhg.de/igd-a3, eMail: tmay@igd.fhg.de

posed, which decomposes a rectilinear mesh into tetrahedra and octahedra and approximates the original values by using triangular *coon patches*. If the resulting error from the initial decomposition is too large, the partitioning continues locally. [7] utilizes hierarchically structured rectilinear grids but focusses on the interpolation scheme that avoids discontinuities at the boundaries facing different subdivision levels. Another approach for the generation of hierarchical meshes is described in [13]. Actually, the authors also build the hierarchy from the coarsest level or coarsest mesh respectively, but the approximation problem is shifted to the *finite element space*. The input data is regarded as a discretization of a smooth function, which will be approximated by an adaptive mesh refinement and error control starting from a discrete function that is defined on a coarse mesh. In [6] is specified the associated adaptive mesh algorithm that is based on an adaptive and regular refinement of a coarse start grid by a consistent decompostion into tetreahedra and octahedra.

Adaptive visualization methods, rather than data formats, are presented in [10, 14]. For instance in an isosurface extraction algorithm a trade off between accuracy and performance is achieved be setting an initial threshold for the *visual error*. A *procedural interface* connecting arbitrary meshes to the visualization methods is proposed in [11]. If the user provides a set of methods to access the grid data, no conversion between the numerical and the visualization grid is necessary.

A hierarchical representation of vector fields is described in [2]. First, the vector field is divided into two disjoint clusters. The vectors in each cluster are averaged and approximated by a single vector. For a given point in the field, two streamlines are calculated using a Runge-Kutta-method. The first one is based on the original field and the second one is based on the corresponding approximation. If the deviation of the two streamlines within a cluster is greater than a given threshold the cluster will be subdivided. The method creates a hierarchy of *BSP-trees* and saves memory resources, because it is gridfree. In [8] a physical clustering model, the *Cahn-Hilliard-Model*, which describes the phase separation is used in order to build a continous multiscale clustering on vector fields.

3. Our approach

With our work we adopt the principle of progressive data processing to CFD data resp. discrete scalar- and/or vector-fields. Since any grid which is used in numerical simulation is unable to handle its data progressively, these grids have to be converted first. Afterwards the data is available in a format which is specially enhanced to meet the demands of a visualization system (see figure 1).

The basis for a progressive grid and the input data for the conversion algorithm is a regular discretization of the domain. We can choose if this discretization should originate from the simulation grid itself or a from a regular sample

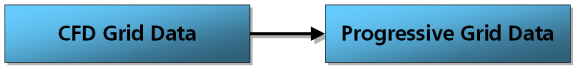


Figure 1: *The orignal CFD grid is converted into a progressive grid.*

of an irregular or even unstructured grid. In the following we describe how a progressive grid is organized an how the conversion works.

First of all, it has to be stated that our progressive grids are *generic*. They are implemented in a conversion framework. It is thereby possible to choose the topological structure arbitrarily: It is possible to select which cell types or spatial partitioning schemes will be used. They define the topology of the progressive grid and how the given grid data has to be interpreted. The same holds true for the interpolation schemes (defining the error estimation), used to compare the progressive grid and its parent - the numerical grid. This approach even allows us to couple different time steps in a four-dimensional grid to make use of the temporal coherencies for compression.

As stated above, we represent our grid in a spatial partitioning scheme, which in turn means that all cells contained in it are arranged in a hierarchical order. The actual numerical information is either associated with a given cell or with the vertices defining its corners. In any case, the partitioning of a cell introduces a certain number of vertices along with at least two new "child"-cells. Their data will contribute to provide a greater local accuracy.

3.1. The grid cells and their data

The *cell* is the elemental structure in the progressive grid. In an abstract sense, it stores

- references to the vertices at its corners,
- the information if and how the cell is partitioned,
- references to its "parent" and "child"-cells (if any)
- numerical information associated with the cell,
- references to a certain number of neighboring cells (if necessary).

This storage structure can be illustrated in a directed acyclic graph *(see fig. 3)*. If possible, the corners of the cells should be vertices of the original regular sample. As there is no need for the progressive grid to be finer than the original one, this helps saving an interpolation step afterwards. For cell types other than boxes, however, this condition may be not satisfied. The vertices contain their coordinates (being of arbitrary dimension) - this defines the location of a cell - and the scalar values derived from the original grid at their position. Furthermore - if the input data is able to provide this information - every vertex has a flag which marks if it is located within the surrounding original grid, which is not

necessarily the case (referred to as the *inoutflag*). The reason for a cell to store only references to the vertices is that the single vertex is obviously shared by a number of neighboring cells and by a number of cells belonging to the same hierarchy. A standard grid cell is illustrated in figure 2.

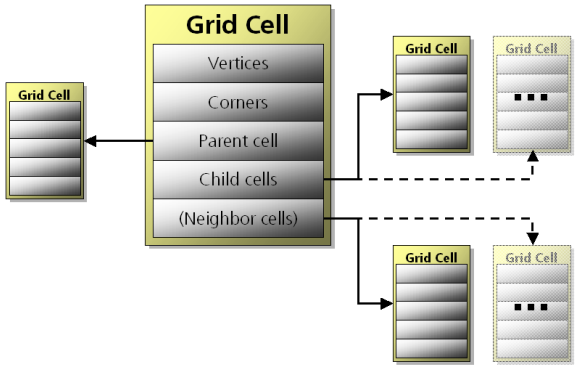


Figure 2: *A grid cell.*

The information regarding the partitioning of a cell is encoded according to the cell type and the partitioning scheme given. If, for example, the cell type is a box, the information needed to represent an octree partitioning reduces to a mere partitioning flag, while the representation of a binary cartesian kd-tree partitioning would include information about the partitioning axis. Any other information need for the partitioning is given implicitly in the "child"-cells.

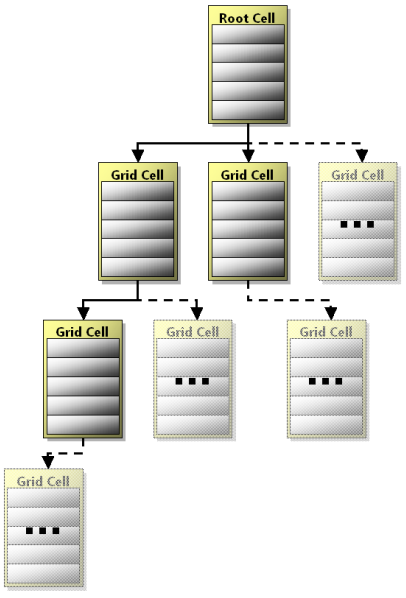


Figure 3: *Severaly cells referencing each other in a graph like hierarchy.*

Most important of the numerical information associated with a cell (further referred to as *volume data*) for the conversion algorithm is the *approximation error*. It describes the approximation quality compared to the original grid. Additionally, some information related to the different scalar fields (e.g. their minimum, average and maximum values within a cell) is provided.

Theoretically the references to the neighboring cells are not necessary for the progressive grid to work properly, as any point in the domain can be located in a certain cell via recursive descent, beginning at an exposed "root"-cell. In practice, though, some algorithms for scientific data visualization work in an incremental way, which means that a step of the algorithm provides some information that should be exploited in the following step (e.g. following a certain path of a streamline through a velocity data volume or the standard *marching cubes* algorithm). Many of those visualization algorithms are following a certain path through the datafield going from cell to cell. So there is a high probability that the next cell processed will be a neighbor to the one currently visited. To precompute and store the neighboring cells helps saving a considerable amount of time during visualization. Moreover, in such cases, the point location algorithm is independent from the current hierarchical depth of the grid.

3.2. The modular approach

The topology of a progressive grid cell, the number and connectivity of its corners, edges and faces is an implication of the cell type itself. Together with the partitioning scheme these information can be seen as a convention how to interprete the cell data. Because never stored explicitly in the grid for reasons of memory consumption, all algorithms that operate directly on this data have to fulfill this convention. Each of the generic parameters can be changed virtually independently from each other offering various configuration possibilities. In order to investigate this possibilities efficiently, we encapsulated this convention in a module and made it accessible only via an abstract interface for the converter and a second interface. This interface will be used by the visualization methods afterwards.

4. The conversion scheme

4.1. The initial partitioning

The main tasks during conversion are to build up the cells hierarchy. This is the most time consuming job. Furthermore the converter has to define the order in which the cell information blocks have to be stored. Since the conversion scheme should remain independent from the given topology or data of the progressive grid, the construction of the cell's hierarchy is handed over to the module mentioned above.

The conversion starts with an initial partition. In the simplest case, using box-type cells, the grid can be based on the

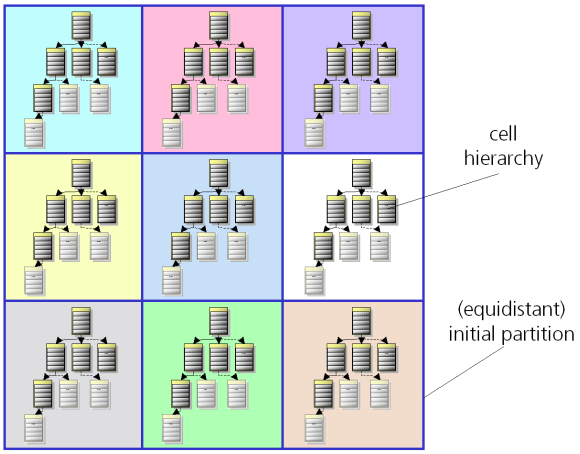


Figure 4: *An example initial partition. For every "'root"' cell of the partition a (different) cell hierarchy is created.*

bounding box of the simulation grid. In the case of a tetrahedral decomposition, the grid is rooted in a special initial partitioning of the original bounding box into a number of tetrahedrons. However, with a given cell type, we are not restricted to a certain initial decomposition. This allows the generation of more than one "root"-cell (and an associated hierarchy). In this case, the module has to provide the mechanism to identify the "root"-cell in which a given point is located in. To support more than one initial partition of the original domain is advantagous for at least two reasons: First of all the conversion can be parallelized very easy since the work to be done for every partition can be done in parallel without any overhead due to synchronization. The second and most important advantage is, that while using a progressive grid that is converted into several independent hierarchies, we are able to define a level of detail for every single spatial domain (resp. its hierarchy). For scenes bigger than the viewing frustrum, this supports a view-dependent level-of-detail visualization. In fig. 4 an example initial partition is shown.

4.2. Determining the approximation quality

Given the partitioning scheme, a hierarchy of grid cells is built up through spatial partitioning. Within a cells boundary the data is interpolated using a selected interpolation scheme (see section 3). For every unpartitioned cell, an approximation error is computed by comparision of values interpolated within the current cell and values given be the original grid at a set of sample points. This approach guarantees that the approximation by the progressive grid is independent from the topology of the original grid. The independency has one drawback, though: As no information from the original grid is presented, except for the values at the sample points, the right choice of the samples points is crucial for the conversion. Just choosing the vertices of the original grid can lead to severe artifacts, because cells of considerable size in the progressive grid may not contain a single vertex of the original one - even if the original grid would intersect the cell. For this reason we choose to predefine a regular sample of the original grid, which is fine enough to cover all its details and compare the interpolation within the current cell with the regular sample.

We define the *approximation error* at each sample point as a linear combination of the errors made for each value of a scalar- or vectorfield and the *inoutflag*. It is possible to choose which subset of scalar values actually is written to the resulting grid, as well as the subset which is considered relevant for the error estimation. According to this, a number of grids each adapted to a single scalar value can be generated. However we noticed that the resulting - slightly better - compression rates are not worth the effort of managing all those grids in virtually all cases. The errors can be weighted to control their influence on the resulting grid. As, for example, the remodeling of the original grid's boundaries will have a higher priority than the scalar values (at last the boundaries define the domain where the scalar values are valid), the linear coefficient for the *inoutflag* will usually have a higher value.

4.3. Building and rearranging the hierarchy

Of all the data contained in a cell information block, the *approximation error* is the only data made accessible to the converter, which has to sort the cells. A cell of the progressive grid is partitioned, if its approximation error is worst compared to all other currently unpartitioned cells. The partitioning generates a number of new unpartitioned "child"-cells, which can be expected to approximate the original grid more adequate. The process stops, if either the cell size reaches the resolution of the input data (original grid) or a given minimum error tolerance has been reached. This scheme guarantees that the maximum error in the "leaf"-cells of the constructed hierarchy decreases as fast as possible. The available number of progression-levels matches the number of partitioned cells. At last the grid cells are stored in the same order as their "parent"-cells have been partitioned. That way, if the progressive grid (or to be more accurate: a single hierarchy within the progressive grid), has to be loaded up to a certain error tolerance, all cells that are needed constitute one block in the storage format. Furthermore, areas where the low frequencies of the scalar fields predominate can be represented with comparably few cells even at low error tolerance rates because the grid locally adapts its resolution to the scalar field data as mentioned above. Since typically every CFD dataset contains areas of low frequency, this circumstance can be exploited for compression. This especially holds true for a dataset derived from an adaptive numerical grid as well.

After the sorting of the cell information blocks, an order

for the vertex data, which is stored in an independent array for each hierarchy involved, is defined. To maintain the advantages of the progressive grid, a vertex is sorted into this array, as soon as it is referenced as a corner of a cell. If a vertex is referenced repeatedly, only the first occurence will be recorded.

While progressing through the cells in the way, in which they are sorted, it is guaranteed that the vertex data is arranged in the same manner. The vertex data needed to define the grid up to a certain refinement level is stored in a single block as well. As the final order of every cell and vertex is now known, all what is left to do now, is to rearrange the references for the corners, parents and children of every cell.

5. Realization

5.1. General Layout

By setting a certain changing celltype, space partitioning and interpolation, the appearance of the progressive grid will change accordingly. To support the adjustment of these "'*parameters*"' it is possible to replace the corresponding parts of the system. As stated above, this is realized using a modular approach, in which the converter is split into two main parts, each communicating via an abstract interface. The conversion consists of a fixed part, which only uses the information, which is relevant for the conversion. The second part is a "plug-in" which manages the cell-type information, its topology, interpolation schemes. Technically this is a class representing a cell of the given type. It must be capable to create its partitioning, to write its data to a buffer and to exposes some general information to the converter (e.g. the size of the buffer needed, the size of the initial partitioning, references to its children or the approximation error). It has to be noted that the same fileformat is used for different types of progressive grids. For this reason the visualization system is divided in a similar way: To decouple the visualization methods from the different grid types, another 'plug-in' (a codec) handles the interpretation of the data optimized for its specific type.

5.2. Cell Types

Two cell types a are worth a closer look: *Boxes* are easy to handle, their corners can be associated with vertices from the original sample and they provide efficient containment tests and interpolation schemes, which can be crucial for the performance of the visualization system. On the other side, using *tetrahedral* cells leads to a better approximation of irregular grids, since their faces are not confined to be aligned to the axis, which influences the overall compression rates. When using tetrahedral cells, their corners will usually not match to vertices in the regular sample and therefore it has to be guaranteed that they contain at least a single vertex of the original data, which in turn means to prevent the tetraedhrons getting too small and "'degenerate"'.

5.3. The adaptive binary kd-tree

To test our concept for the progressive grids, we implemented a module which treats box-type cells along with an adaptive binary kd-tree decomposition (see fig. 5). In contrast to the likewise well-known octree scheme, we are able to control along which axis the resolution increases. Using cartesian boxes, we identified every corner of a cell with a vertex of the original sample. The data of the four new vertices defining a partitioning plane has a great influence on the quality of the partitioning. Hence we divided a cell not necessarily into two parts equal in size. So we implemented an algorithm, adapting the partitioning with respect to the original data. Our algorithm tests all possible partitioning planes defined by the discrete sample for every dimension to determine the one which is most suitable to approximate the original grid. Compared to a simple bisection algorithm, the accuracy increases more rapidly, when progressing through the levels of detail. Furthermore this improves the overall compression rates for a grid. As this algorithm does an exhaustive search, it performs rather poor while converting a grid of considerable size. Of course even this partitioning/converting algorithm can be parallelized in a straightforward way (as the partitioning planes can be tested independently of each other). But there is still some more room left for optimization. The idea is not to use the error metrics defining the quality of a cell approximation as above, which includes the computation of an approximation error for each sample point and each partitioning plane. Instead, the subcells in itself can be made as homogenous as possible.

To measure the homogeneity of a given cell, a wavelet analysis of the domain is applied as a preprocessing step. If this is done only once, it significantly increases the performance of the conversion algorithm. Comparing the wavelet coefficients of the regions of interest (i.e. those covering the current cell), we are able to evaluate the partitioning without involving the original data repeatedly. The wavelet representation of a dataset serves us as a "proposition" of how to build up the hierarchy, because homogenous regions (identified be the lack of high frequencies) in the wavelet-representation will be preserved. This will release ressources needed for the representation of inhomogenous regions within the progressive grid.

6. Conclusion and Future Work

We presented a generic progressive format and framework, which serves as highly adaptive representation of CFD-datasets. From the developers point of view, the modular approach guarantees a high flexibility, while different kinds of progressive grids may be implemented and evaluated, without need to rebuild the whole system. From the users perspective, we are able to adapt the CFD-data resolution in different levels to the memory resources of the environment. At the same time, no concessions to the rendering performance of the grid must be made. We converted a number of datasets

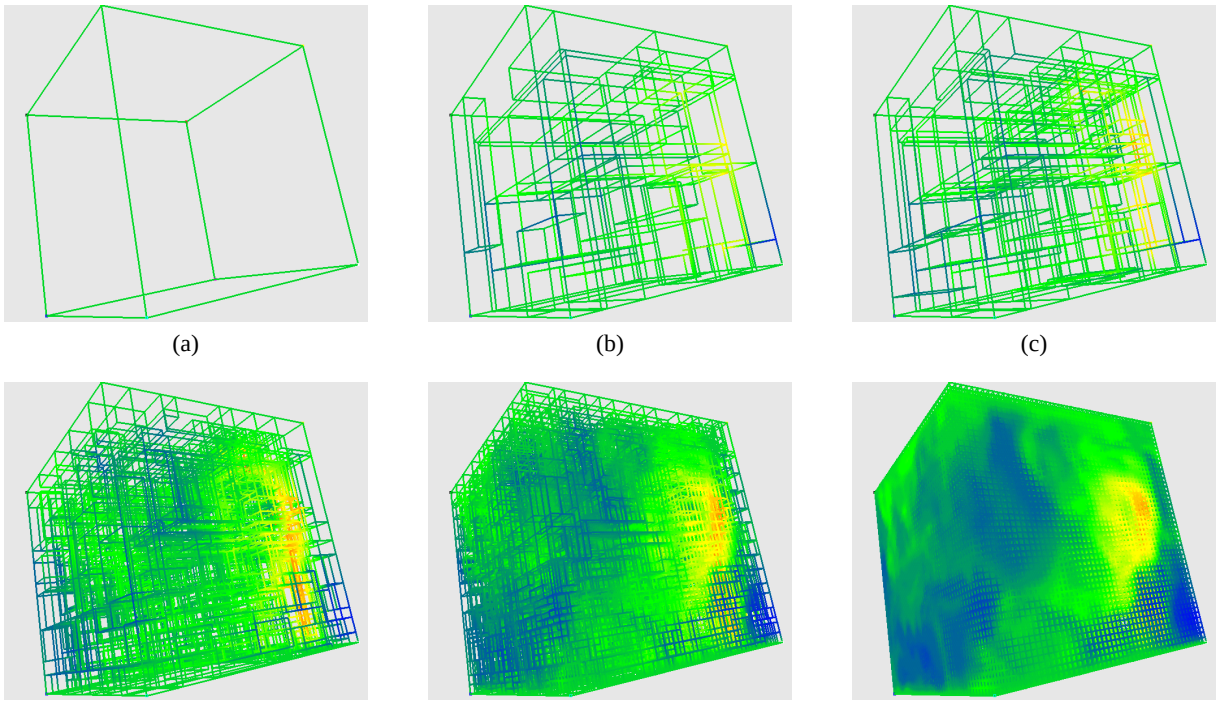


Figure 5: *Different resolutions of the same progressive grid: (a) grid structure at hierarchy level 0 resp. using 1 cell. (b) using 50 cells. (c) using 100 cells. (d) using 1000 cells. (e) using 10000 cells. (f) using the full resolution with 110286 cells.*

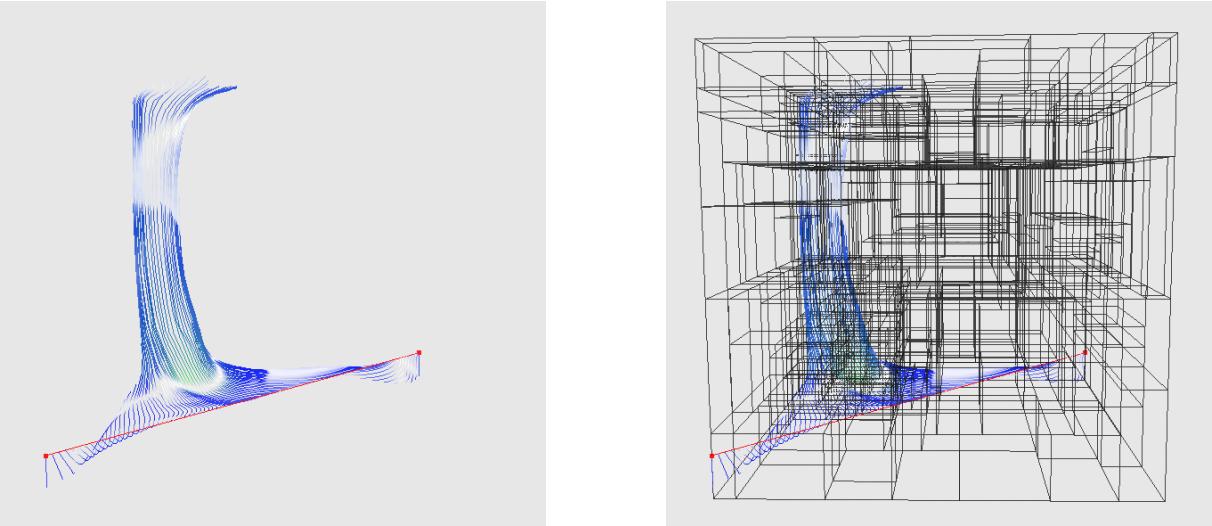
from fire and CFD simulations (see fig. 6). Zooming through the according levels of detail was possible in real time. A visualization of streamlines (including 200.000 vertices) using the progressive grid was comparable in computiation speed to one based on the original equidistant grid.

For the future we plan to include further enhancements like

- the evaluation and comparision of different kinds of progressive grids (especially spatio-temporal grids),
- the improvement and parallelization of the partitioning algorithm,
- the test of different error estimation schemes, especially those matching the different character of scalarfield and vectorfield data,
- the implementation of data compression schemes.

References

1. Marsha Berger and Joseph Oliger. Adaptive mesh refinement for hyperbolic partial differential equations. *Journal of Computational Physics*, 53:484–512, March 1984.
2. Bernd Hamann Kenneth I. Joy Bjoern Heckel, Gunther Weber. Construction of vector field hierarchies. In *Proceedings of IEEE Visualization '99*, pages 19–26, October 1999.
3. E. Blayo and L. Debreu. Adaptive mesh refinement for finite difference ocean models: first experiments. *Journal of Physical Oceanography*, 29(6):1239–1250, 1999.
4. Gregory M. Nielson David J. Holliday. Progressive volume models for rectilinear data using tetrahedral coons volumes. In *Symposium on Visualization*, 2000.
5. L. Freitag and R. Loy. Adaptive multiresolution visualization of large data sets using a distributed memory octree. In *ACM/IEEE Conference on SuperComputing: Proceedings*, January 1999.
6. R. Grosso and G. Greiner. Hierarchical meshes for volume data. In *Proceedings Computer Graphics International '98*, pages 761 – 769, 1998.
7. Terry J. Ligocki John M. Shalf Hans Hagen Bernd Hamann Kenneth I. Joy Kwan-Liu Ma Gunther H. Weber, Oliver Kreylos. High-quality volume rendering of adaptive mesh refinement data. In *Proceedings of 6th International Fall Workshop Vision, Modeling, and Visualization*, pages 21–23, 2001.
8. M. Rumpf A. Telea U. Weikard H. Garcke, T. PreuSSer and J. van Wijk. A phase field model for continuous clustering on vector fields. In *IEEE Transactions on*



(a) (b)

Figure 6: *Stream line visualization using our progressive grids: (a) Just the calculated stream lines. (b) Stream lines together with the corresponding progressive grid.*

Visualization and Computer Graphics, pages 230–241, 2001.

9. H. Hoppe. Progressive meshes. In *SIGGRAPH '96: Proceedings*, 1996.
10. M. Rumpf M. Ohlberger. Hierarchical and adaptive visualization on nested grids. *Computing*, 59(4):269–285, 1997.
11. A. Schmidt M. Rumpf and K. Siebert. Functions defining arbitrary meshes, a flexible interface between numerical data and visualization routines. In *Computer Graphics Forum*, pages 129–141, 1996.
12. E. Puppo P. Cigoni, C. Montani and R. Scopigno. Multiresolution representation and visualization of volume data. *IEEE Transactions on Visualization and Computer Graphics 1997*, 3(4), 1997.
13. Ch. Luerig R. Grosso and Th. Ertl. The multilevel finite element method for adaptive mesh optimization and visualization of volume data. In *Proceedings Visualization '97 Conference*, 1997.
14. M. Rumpf R. Schwörer R. Neubauer, M. Ohlberger. Efficient visualization of large-scale data on hierarchical meshes. In *Visualization in Scientific Computing*, 1997.
15. H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2), June 1984.
16. J. Wilhelms and A. van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics (TOG)*, 11(3), July 1992.
17. B. Fry Z. Zhu, R. Machiraju and R. Moorhead. Wavelet-based multiresolutional representation of computational field simulation datasets. In *Proceedings of the Conference on Visualization '97*, October 1997.

Implementing a CFD steering system for immersive environments

Kai-Mikael Jää-Aro

Department of Numerical Analysis and Computer Science
Royal Institute of Technology

Contents

- Background – the VIRTUALFIRES project
- Current implementation
- Conclusions for the future

VIRTUALFIRES partners

- http://www.virtualfires.org/
  - SiTu, TU Graz, Austria
  - CD, Uni Leoben, Austria
  - PDC, KTH, Sweden
  - FIGD, FhG, Germany
  - EUVE, Spain
  - FDDo, Germany
  - LTF, France
  - CETU, France

VIRTUALFIRES – the aims

- Real-time numerical simulation of tunnel fires
- Real-time steering of simulation parameters
- Immersive visualisation
- Safety studies of future tunnels
- Scenario training for fire fighters
- (Support partners' pet projects)

Desired interface

- Users should be able to set boundary conditions interactively in the displayed tunnel geometry
- Users should be able to interactively place fire loads, fire-fighting equipment and other items in the tunnel
- Users should be able to indicate events happening at future times in a scenario (ventilation turned on/off, fire extinguishers used, etc)

Further desirables

- Multiplatform – PC to supercomputer
- Ideally use identical user interfaces in CAVE, in HMD and on desktop
- Support non-expert users
- Easily extensible interface

Choices made

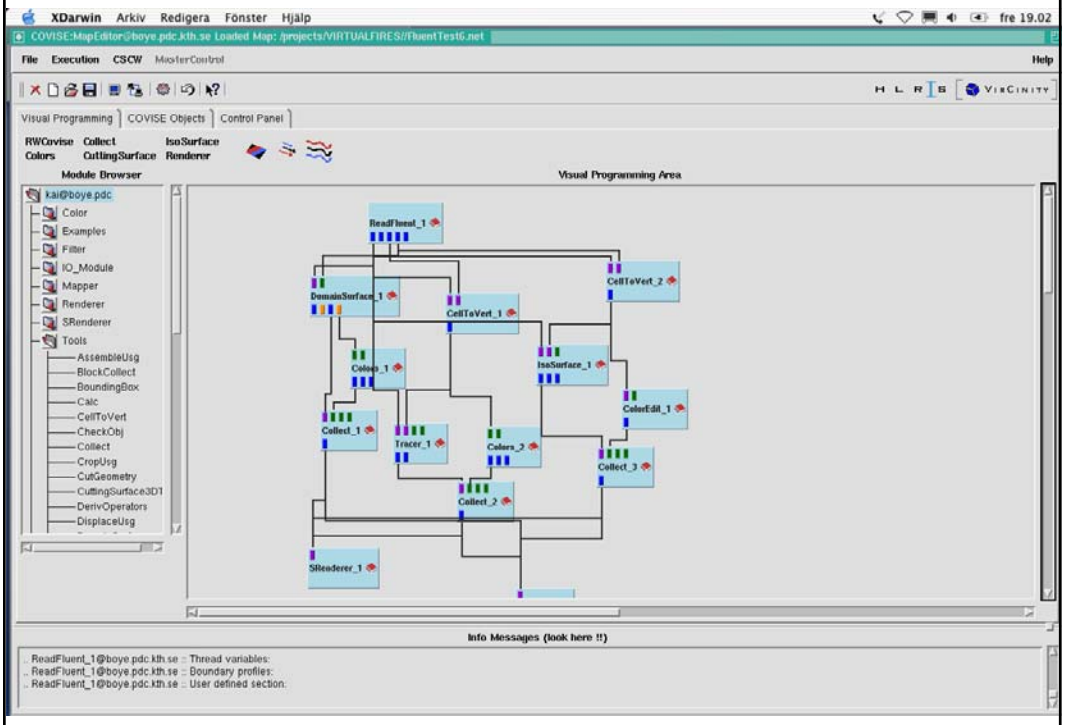
- Small-size GUI
  - Displayable on PDA for CAVE version
  - Fits in HMD view
  - Can be used on desktop screen
- Use COVISE as visualisation platform

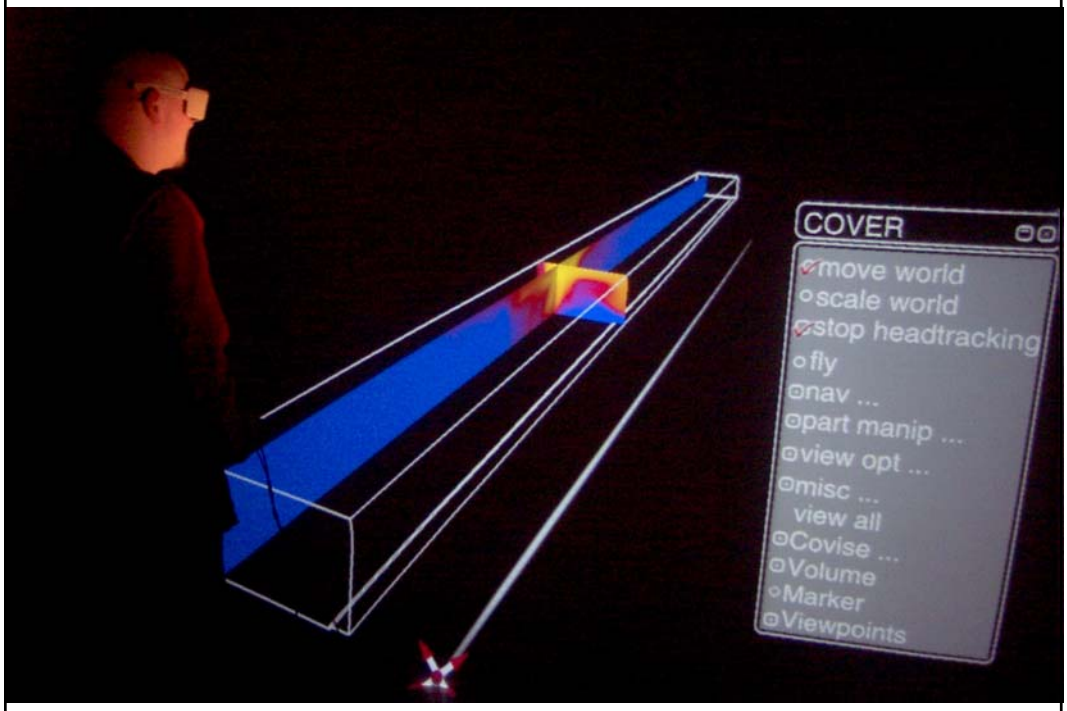
COVISE

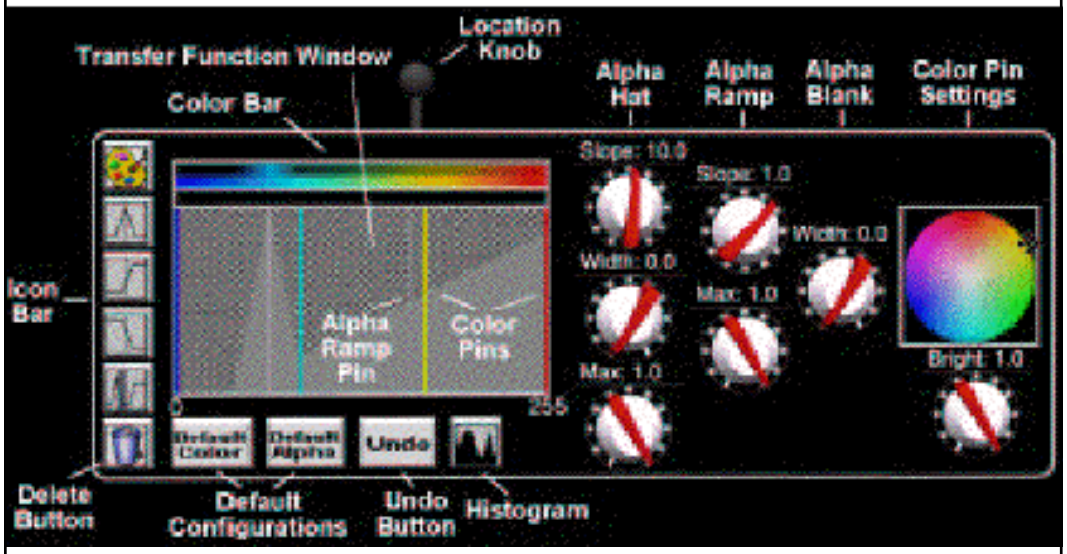
- `http://www.vircinity.de/`
- Modular visualisation system
- Graphical programming language
- Distributed system
  - Modules on different machines
  - Support for remote collaboration
- Immersive rendering module – COVER
- User-extensible

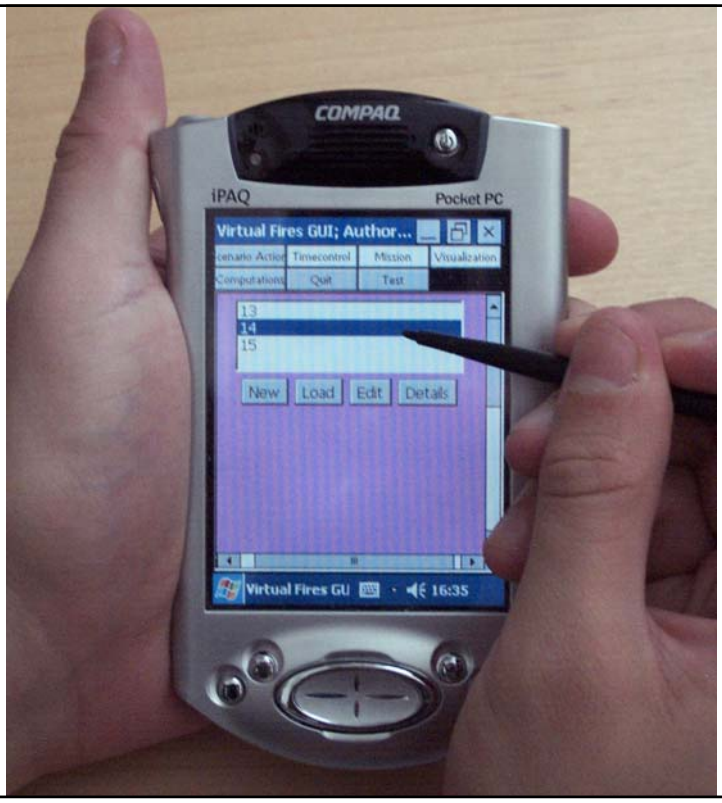
COVISE programming model

- Fairly strict dataflow, but modules can attach messages to data
- Plugins in COVER can intercept these messages and send responses to the originator
- Plugins can send messages to each other
- Plugins can access the scene graph

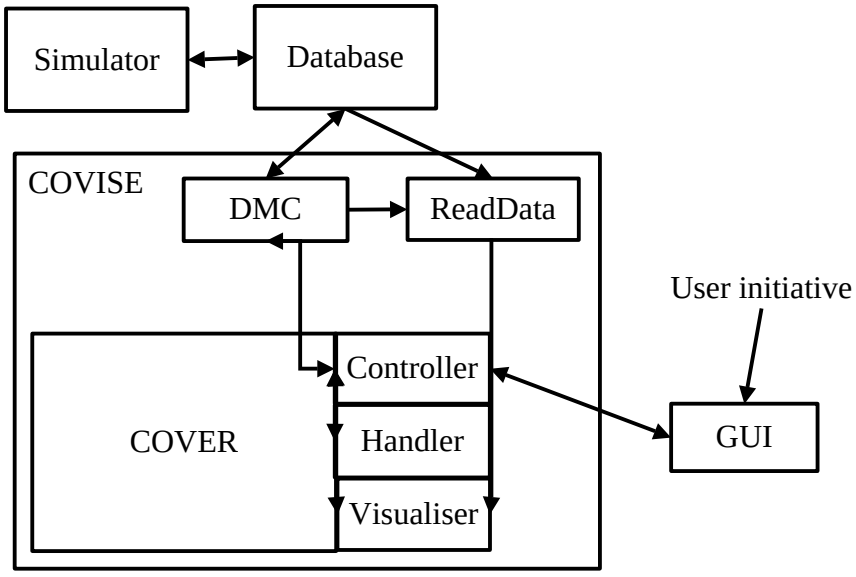



COVER
move world
scale world
stop headtracking
fly
nav ...
part manip ...
view opt ...
misc ...
view all
Covise ...
Volume
Marker
Viewpoints


Transfer Function Window
Location Knob
Color Bar
Alpha Hat
Alpha Ramp
Alpha Blank
Color Pin Settings
Slope: 10.0
Slope: 1.0
Width: 0.0
Width: 0.0
Max: 1.0
Max: 1.0
Bright: 1.0
Icon Bar
Alpha Ramp Pin
Color Pins
255
Undo
Delete Button
Default Configurations
Undo Button
Histogram



Current system design



Experiences

- Using COVISE did not work out very well
  - Most of the programming model had to be dropped
  - Much functionality had to be reimplemented
- Yet, COVISE is not a bad system
  - What happened?

Analysis

- We were locked in by an application framework. COVISE expects data to come in at the top, be processed and sent to a renderer.
  We want to use graphics as *input* to the system as well as output from it.
  - COVISE *did* allow us to implement extensions
  - But, very little of the functionality of COVISE is left

COVISE-specific problems

- The behaviour of COVISE modules is set through module *parameters*
  - Parameters can be interactively set by the user, by editing values in a popup window and/or by adding parameters to a separate *Control panel* window
  - A module can, through an explicit message, allow a COVER plugin to modify its parameter values immersively, but only a few modules send this message, and the rest cannot be modified, only reimplemented.
- System in principle allows customisation, but does not fully support it in practice.

- In our application many operations have to communicate with each other to allow interaction

  ® use plugins instead of modules

  But, plugins are invisible in the programming interface

  ® the logic of the program is hidden in parallel code

  ® programs are not interpreted but have to be compiled

COVISE-specific remedies

- Input ports instead of “hidden” parameters give more flexibility when programming
- Sub-classable modules diminishes need for reimplementation
- Input ports made available for plugins
- Plugins should not be hidden in the visual program, they need to be made public, showing their connections to other program elements

Better yet

- We should not be constrained by a framework
- Yet we want to use as much existing functionality as possible from a visualisation system
  - We should be able to link in visualisation functions in our program without prejudice
  - Caveat: This is (currently) difficult to do in a purely graphical programming system

The case for open source

- There are very few general visualisation tools for immersive environments:
  AVS Express/MPE, COVISE, vGeo
  - It's non-trivial to test commercial systems and get them to work
  - They are difficult to extend beyond their framework
- We need an open source platform for immersive visualisation

Alternatives

- Currently there are two major open visualisation packages: VTK and Open DX
- Both can be linked into external programs
- VTK has already been used for immersive visualisation through VtkActorToPF module, but is not as easy to program
- Open DX has graphical user interface, but no immersive renderer/interaction module

My suggestion

- I would prefer an immersive extension to Open DX, as it is easier for non-experts to program and has more complete functionality

Virtualfires - realtidssimulering och visualisering av brandförlopp i tunnlar

Gert Svensson, Kai-Mikael Jää-Aro,
PDC, Kungliga Tekniska Högskolan

VIRTUAL FIRES

VR FORUM, 5 November 2003

Parallelldatorcentrum, PDC

- Nationellt akademisk center
- Simulering
- Visualisering

VIRTUAL FIRES

VR FORUM, 5 November 2003

Bakgrund - tunnelolyckor

- Mont Blanc 1999, 39 döda
- Tauern 1999, 12 döda
- Kaprun 2000, 155 döda
- Gleinalm 2001, 5 döda
- St Gotthard 2001, 11 döda

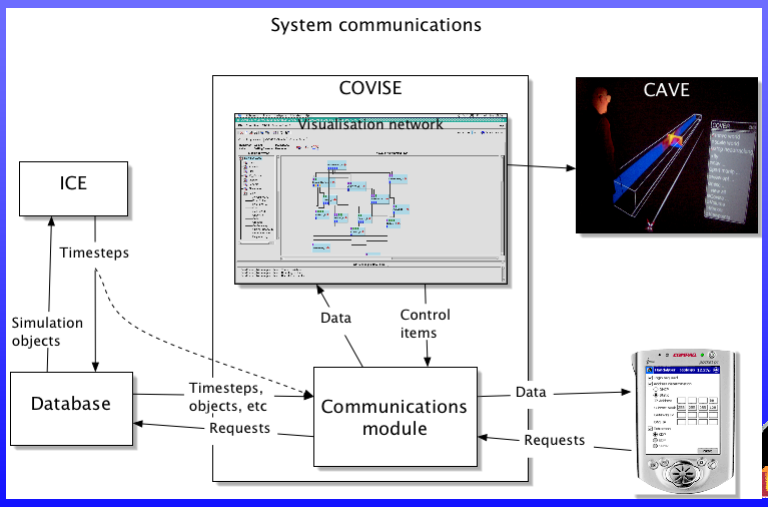
EU-projektet VirtualFires

- Simulering och visualisering av bränder i tunnlar
- Visualisering i VR eller platt skärm
- Samtidig simulering och visualisering
- Träningsverktyg för brandchefer

Funktionalitet

- Utplacering av brännbara objekt
- Definition av atmosfäriska storheter
- Styra ventilation under branden
- Styra sprinklers
- Styra släckinsatser
- Gå fram och åter i förloppet
- Definiera alternativa förlopp

Arkitektur



Simulering

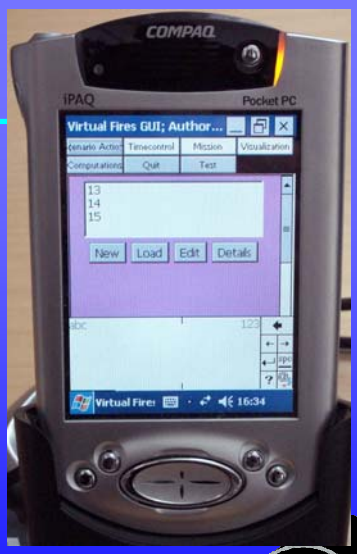
- Samtidigt som visualisering
- Lattice Bolzmann metod
- Kubisk grid
- Direkt metod
- Enkel att parallellisera
- Vi hoppas på ung. Realtid
- Dock begränsad volym 10-100 m

Visualisering

- Vanlig PC skärm
- HMD
- Cave
- Vetenskaplig visualisering
- Realistisk visualisering (rök och eld)

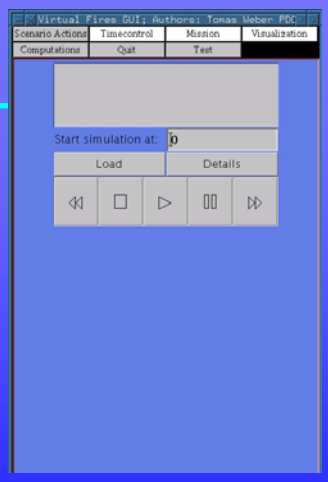
Användarinterface

- Samma interface för Cave, HMD och PC
- Java GUI som kan köra på handhållen dator
- Försedd med lägesgivare i VR-miljöer



Användarinterface …

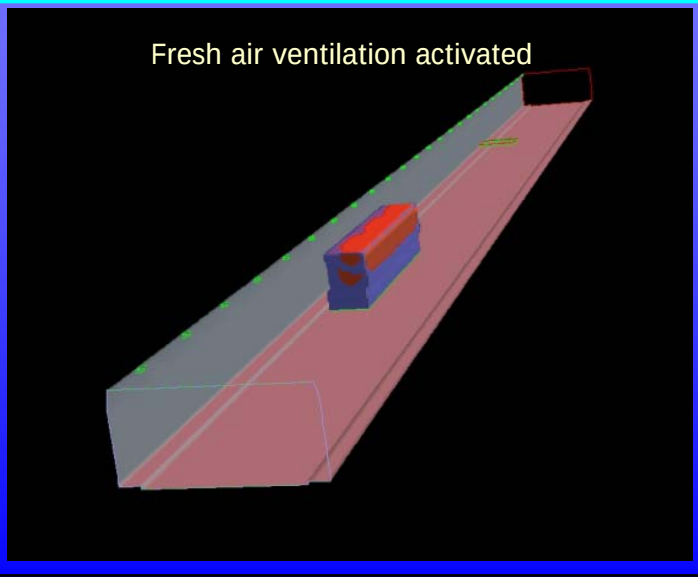
- Videoinspirerat interface
- Delvis datadrivet interface via XML-objekt



Några exempel på simuleringar

VIRTUAL FIRES

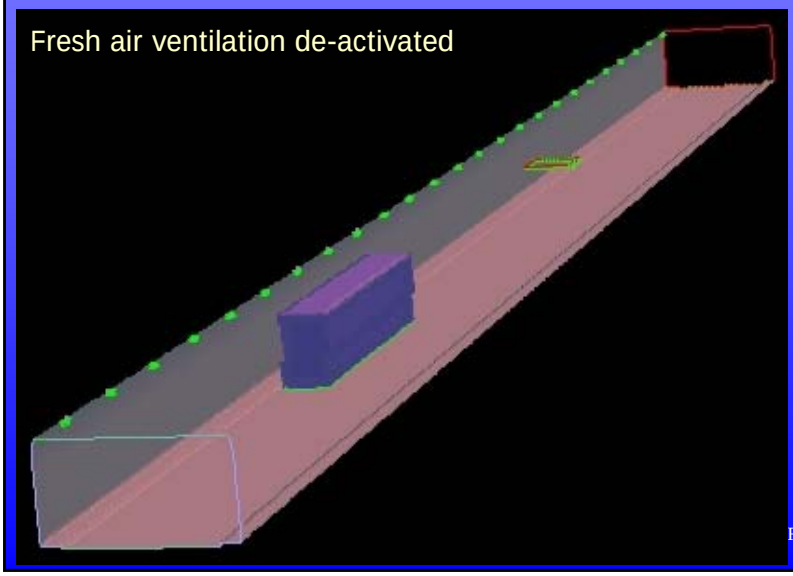
Lastbil i Gleinalm-tunneln



Isoyta 400 K

VIRTUAL FIRES

Fresh air ventilation de-activated



VIRTUAL FIRES

FORUM, 5 November 2003

Partners

KTH VETENSKAP OCH KONST

Fraunhofer Institut Graphische Datenverarbeitung

Alpetunnel

ACT

SITU Simulation In Tunnelling

EUVE

VIRTUAL FIRES

VR FORUM, 5 November 2003

Mer information

www.virtualfires.org

www.pdc.kth.se

Gert@pdc.kth.se

VIRTUALFIRES a virtual reality simulator for tunnel fires

Gernot Beer, Thomas Reichl and Gunter Lenz
Institut für Baustatik, Graz University of Technology, Austria

ABSTRACT
The project VIRTUALFIRES, which is being funded by the European Commission and involves eight partners from five European countries, is presented. The aim of the project is to develop a simulator that allows to train fire fighters in the efficient mitigation of fires in a tunnel, using a computer generated virtual environment. This will be a cheap and environmentally friendly alternative to real fire fighting exercises involving burning fuel in a disused tunnel. The simulator can also be used to test the fire safety of a tunnel and the influence of mitigating measures (ventilation, fire suppression etc.) on it's fire safety level.
Key words: Visualisation, Virtual Reality, Tunnels, CFD

1. INTRODUCTION

Recent serious fire accidents in tunnels have highlighted the problems that currently exist with respect to the prevention of serious fire incidents and with respect to fire mitigation once the fire has started. There is a need for action on a European scale with respect to

- Ascertaining the safety level of existing tunnels and retrofitted tunnels (i.e. Mont Blanc tunnel).
- The specification of the required safety features and installations for new tunnels.
- Training of rescue personnel in order to increase the efficiency of fire and smoke mitigation procedures.
- Training of drivers with respect to correct behaviour in the case of a fire emergency.

With respect to ascertaining the safety level of existing and retrofitted tunnels much reliance is still placed on real tests using fire/smoke pans or vehicles set on fire. Such tests have been recently performed for example in the refurbished Mont Blanc Tunnel. Fire fighting exercises are usually carried out on a regular basis using burning vehicles or cold smoke generators.

The disadvantages of real tests are that they are expensive, can only be carried out at certain times and are not environmentally friendly since toxic smoke is produced. The main aim of the VIRTUALFIRES project is to develop an alternative to real tests by replacing them with virtual tests. In an virtual test the tunnel and the fire emergency only exists in computer memory. Using computational fluid dynamics (CFD) computations, the spread of fire and smoke in a particular tunnel is calculated.

The tunnel including the safety installations, traffic signs, vehicles etc. is visualised together with the results of the CFD calculations using the method of virtual reality. Under the term virtual reality we mean total immersion in a three-dimensional data set. The effect should be very close to reality. The simulator can be used as a training tool for fire fighters and for assessing the safety level of existing or retrofitted tunnels. It may also be used to check the design of a planned tunnel.

The simulator will be equipped with a user friendly program for the input of data which comprise the shape of the tunnel, ventilation characteristics, safety installations, vehicles, emergency exits etc. Two versions of the simulator will be developed: One where the CFD

simulations are carried out prior to the visualisation (pre-calculated scenario) and one where the CFD simulations are carried out concurrent with the visualisation. The advantage of the second type would be that ventilation characteristics may be changed during the visualisation (i.e. one may check the effect of reversing the ventilation on the spread of smoke and fire). The first type of system can be used for the training of fire fighters and drivers and for checking the fire safety of existing tunnels.

2. DESCRIPTION OF SIMULATOR

1.1. Hardware

The simulator will be implemented on two hardware platforms: a portable version consisting of a Laptop PC with a head-mounted display connected to it for off-line simulation and a more powerful and interactive version running in a distributed computing environment and a CAVE as the visualisation front-end.
A headmounted display (HMD), like the one shown in figures 1 & 2, contains 2 liquid-crystal displays and a 3DOF tracking sensor. Due to the possibility of displaying images from different viewpoints to the each eye, the user gets a threedimensional impression of the scene. The tracking sensor, which delivers the rotation-angles around the 3 principal axis in realtime, allows for synchronisation of the users head movement with the viewing direction of the rendered scene.

Figure 1: User wearing a headmounted display

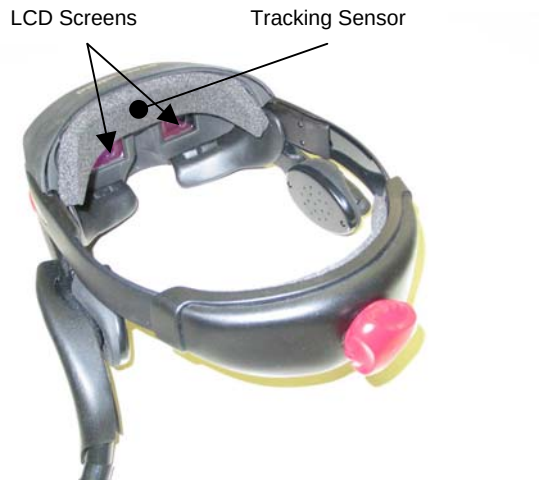


Figure 2: interior view of a HMD

A CAVE is a multi-person, room-sized, high-resolution, 3D video and audio environment. Graphics are backprojected in stereo onto the walls, the floor and the ceiling, and viewed with shutter glasses (see figure 3). As a viewer wearing a position sensor moves within the display boundaries, the correct perspective and stereo projections of the environment are updated in realtime by the rendering system, and the images move with and surround the viewer. Hence stereo projections create 3D images that appear to have a continuous presence both inside and outside the projection room. To the viewer with stereo glasses, the projection screens become transparent and the 3-D image space appears to extend to infinity.

Figure 3: User in a CAVE wearing shutter glasses

Both display-systems have some advantages and drawbacks.
HMD:

+ portable and lightweight device
+ moderate computing power for image rendering: only 2 different images of the scene need to be generated at the given framerate
- user not fully immersed into the scene due to the limited field of view, impression is more like watching the scene through divers-glasses
- low resolution of the display

CAVE:

+ wide field of view, so the user is fully immersed into scene
+ very high resolution of the rendered scene
- stationary installation
- high computing power needed for rendering: dependent on the number of backprojected walls 10 or 12 images are required at the given framerate

Due to the limited computing power of the mobile version of the system there will probably be no possibility of changing parameters of the simulation in realtime. This system is more thought of an interactive threedimensional movieplayer for the precalculated simulation results. Nevertheless it will be possible to change parameters at any given timestep and to continue the simulation from this point with the new values.

1.2. Software

Since CFD computations of fire and smoke spread are already at a fairly high level of sophistication the emphasis of the project is on the further development of methods of visualisation and virtual reality. New developments are only envisaged in concurrent CFD calculations using massive parallel computers.

There are two aspects that need to be considered in the visualisation:

Size of data: The amount of data produced by a CFD program is quite large. Depending on the length of the tunnel being considered in the simulation and the time span modelled they are in the order of several Gigabytes. Before such a large amount of data can be displayed in real time one must apply data compression/optimisation techniques.

Feeling of reality: The simulator will only find acceptance if the simulation is realistic, i.e. gives a feeling of reality. A realistic display of the tunnel and its safety installations for example is important, as is the use of realistic textures. Questions to be answered are: how does one visualise fire and smoke, temperature and toxicity etc.

1.3. Rendering system and user interface

Not all of the potential users (i.e. fire fighters, rescue personnel, etc.) are familiar with VR-Equipment, so the userinterface must be simple and easy to understand in a straightforward manner. The planned System will be capable of defining missions prior to the simulations as well as interactive interaction during the execution of such a mission, like "freezing" the scenario, changing some parameters, i.e. start the ventialtion, or move forward or backward in time.

3. SYSTEM SPECIFICATIONS

The specifications of the system capabilities are still being worked out but can be summarised as follows:

- VR-System

Tunnel geometry and safety installations will be visualised with realistic textures. Traffic signs, emergency exits, escape tunnels, vehicles must be displayed very realistically for the evaluation of safety features.

- CFD calculation

Calculation of fire and smoke spread after a vehicle has been set on fire. Level of detail sufficient for realistic visualisation (higher level close to observer, lower further away) and for calculation (using 1D or 3D-Models). Type of output expected: Flame spread, smoke density, temperature and toxicity. Time step may be governed by stability criteria but updating of transient data for visualisation is only required every 4 ms (equal to 25 frames per Second) to ensure smoothness of display.

- Display of CFD data

The rendering of the simulation results needs to fulfill two different criteria: For those tasks where a visual evaluation of the scene is necessary, like in checking the readability of traffic signs, the display of the smoke distribution data must be as realistic as possible, including changing fog density and turbulent flow (see figure 4). On the other hand the rendering of temperature and toxicity-values needs a meaningful and rapidly interpretable representation like the one shown in figure 5, where colour is used to discriminate the different levels of danger (blue: safe area, red: dangerous). To observe the velocitiy and direction of the flow of the smoke the system will allow traces of particles to be shown.

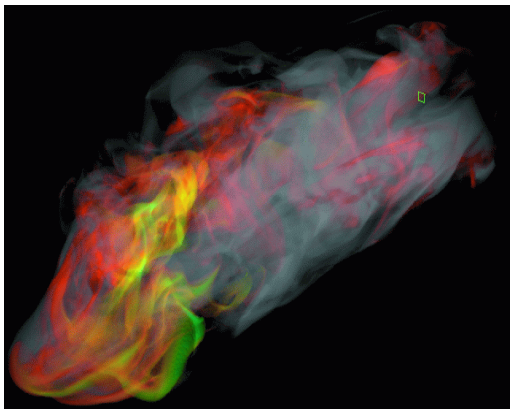


Figure 4: smoke visualisation

büdl isosurf

Figure 5: volume rendering of toxocity values

büdl colourplot

Figure 6: tracelines of a particle

- User navigation

User moves through virtual space with a three-dimensional input device (space ball or hand held input device). Speed of movement is controlled to simulate walking, running. Head movement is detected by positioning devices. Collision detection is implemented to prevent users from going through tunnel walls. Pull down virtual menus are used to select type of display.

4. EXAMPLES OF APPLICATION

1.4. Mission planning for firefighters

In this application is designed to allow a fireman to evaluate the effectiveness of a planned mission in a specific emergency-scenario. After defining the necessary mission-parameters like number of firemen involved, type of equipment and direction of attack, the simulation provides some quantities for ranking the success of the mission like set up time for equipment, time to extinguish, level of impact of the fire or toxic load exposure of the firemen.

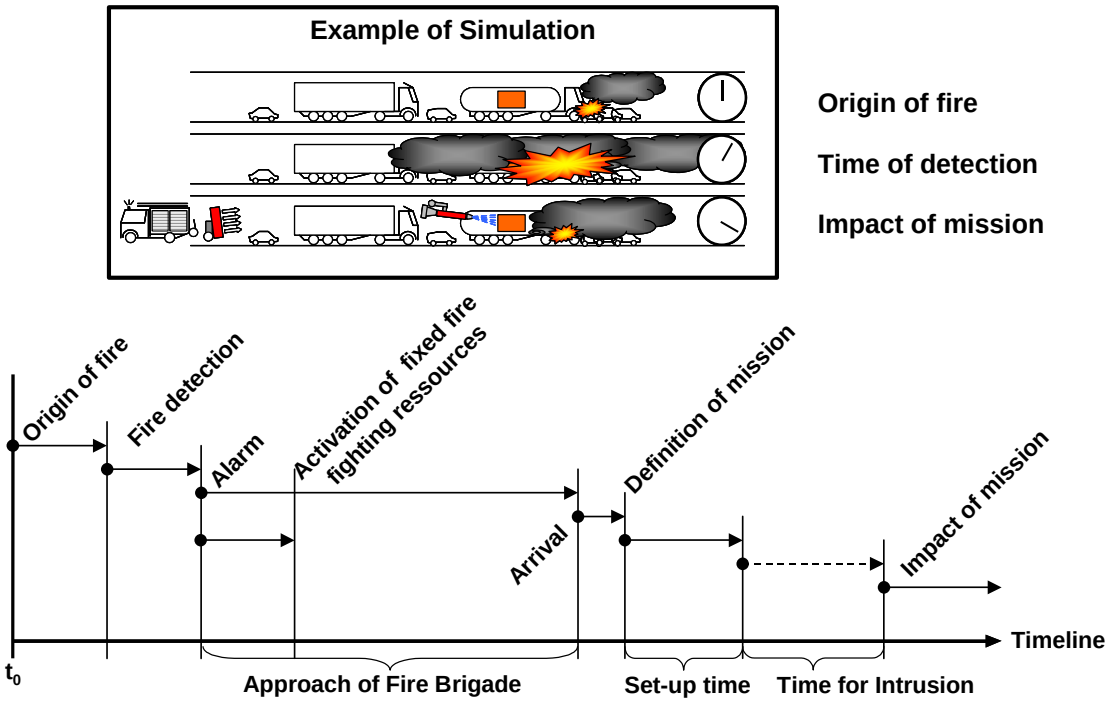


Figure ??: Resulting timeline of a simulated mission
(pic from FDDo/c.l.k. Report WP 2.4)

1.5. Evaluation of tunnel safety equipement

The authority or planner of the tunnel infrastructure can use this tool to objectively evaluate the functionality and usability of the safety equipement. This task includes checking the visibility of road-signs in a smoke environment, testing the amount of fresh air supplied to the emergency cabins or evaluating the performance of the ventilation system.

1.6. Evaluation of fire countermeasures performed by tunnel operators

As the operators sitting in the control-rooms are the first persons to respond to an emergency, these people need to be trained for specific situations. With the help of the simulator an operator can monitor the consequences of his set actions.

1.7. Training of drivers for firesituations in tunnels

Several incidents in road tunnels have shown, that the majority of the drivers reacts totally wrongly due to panic and missing information about the things that can happen. The simulator can give a trainee the experience of an accident of a lorry with flammable goods and he/she can try different strategies of escape from the tunnel to a safe environment without the danger of getting harmed by a wrong reaction.

5. SUMMARY AND CONCLUSIONS

The project aims at the building of an interactive and high-performance simulation environment for fire incidents in tunnels. This development will lead to an objective tool for evaluating the safety infrastructure of a tunnel and provides a cheap alternative for verifying the effectiveness of fire fighting missions and for training of drivers and tunnel-operators in the case of an emergency.

VIRTUALFIRES A Virtual Reality Simulator for Tunnel Fires

Beer G., Reichl Th., Lenz G.
Institute for Structural Analysis
Graz University of Technology

ABSTRACT

The VIRTUALFIRES (Virtual fire emergency) simulator is presented, that allows to train fire fighters in the efficient mitigation of fires in a tunnel, using a computer generated virtual environment. This is a cheap and environmentally friendly alternative to real fire fighting exercises that are currently carried out and that involve burning fuel in a disused tunnel. The simulator can also be used to test the fire safety of a tunnel and to ascertain the influence of mitigating measures (ventilation, fire suppression etc.) on the fire safety level. The simulator is developed with financial support from the European community under the IST (Information society technology) program and combines the simulation of fires using advanced CFD software and the visualization of smoke, toxicity levels and temperature. Two versions are being developed: One using a head mounted display and a laptop, the other using a CAVE virtual environment together with a supercomputer. Both display systems have some advantages and disadvantages. The presented HMD version requires moderate computing power and can show realistic fire and smoke distribution, but the user is not fully immersed in the scene due to the limited field of view. The CAVE version gives a realistic impression of emergency situations in tunnels, because of its room-sized high resolution 3D video and audio environment. The VIRTUALFIRES simulator can be tested by the participants during the conference.
Key words: tunnels, virtual reality, CFD simulation

1. INTRODUCTION

A Virtual Real Time Fire Emergency Simulator (VIRTUALFIRES) has been developed using techniques of virtual reality. In the simulator, the observer is able to visualise the fire and smoke development and the transport of heat and toxic combustion products inside a tunnel and walk or run through the virtual structure in the same way as through a real tunnel. The simulator uses and accesses a database, which contains the results of three-dimensional transient combustion (Computational Fluid Dynamics - CFD) simulations for particular tunnel geometries with associated safety installations, particular fire hazard scenarios, etc.
CFD-results can be displayed as a fixed installation in a CAVE virtual environment and as a portable installation using a PC and a head-mounted display (HMD). Two systems are developed: one where the CFD simulation is pre-calculated, stored into a database and then displayed and another where it is carried out in parallel to the visualisation. In the first system the user will be able to move through the data but will not be able to change the characteristics of the simulation, for example the ventilation characteristics. In the second system the user may change the properties of the simulation while the data are displayed.
The VIRTUALFIRES system will be a unique system that can be used for assessing the fire safety of tunnels, for training of rescue personnel and for planning rescue scenarios and will be able to replace or supplement real fire tests. The end users of this system will be rescue organisations such as the fire brigade and police, tunnel operators and government organisations concerned about tunnel safety. The system can be used for making an objective assessment of the fire safety of existing European tunnels. It can also be used for training drivers on how to behave in the case of a fire emergency in a tunnel.

2. DESCRIPTION

The layout of the software is depicted in Figure 1. At the heart of the system is the CFD simulation software ICE, which uses the Lattice Boltzmann method [1] to compute the air velocity, temperature, pressure and smoke density at a cell point due to a fire.

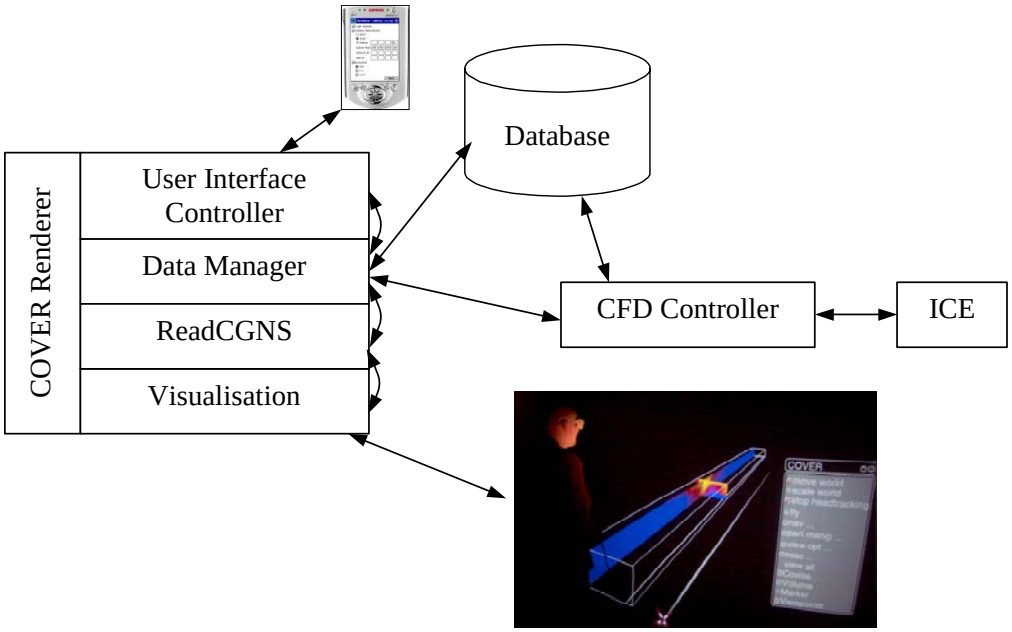


Figure 1: Layout of the VIRTUALFIRES software

The "smoke density" is an artificial quantity, which varies between 0 and 1. The smoke production is taken to be proportional to the CO and CO2 standardized production curves provided as input. It must be pointed out that smoke is a result of fuel rich combustion and the modelling via standardized curves is only an approximation. However, a real combustion model requires input data, which are normally not available and the calculation is very time consuming.

The storage and retrieval of the calculated CFD-data and the states of all objects that are involved in a simulation-run is handled by the Database Manager module. This component serves as the communication layer between the simulation front end and the database server back end. Currently it transparently supports the MySQL 4.0.15 open source SQL server, but is adaptable to any other SQL server.

The communication between the CFD solver and the storage layer is done by the data manager Controller module. This module has been integrated into the Covise VR-environment [2] and also handles all requests from the user interface. As there are normally limited interaction capabilities inside a CAVE environment, a new PDA-based graphical user interface has been developed. This GUI allows the user to specify the mission he/she wants to examine, change simulation parameters and restart a simulation. The major advantage of this solution is that this navigation tool can also be used outside the CAVE with the PC-based VR environment without any changes to the simulator, because it is integrated into the network communication layer inside the simulator.

Within the project also some new visualization techniques have been developed. This was necessary as currently available ones where not sufficient for the system, mainly for 2 reasons:

1. They were to slow to handle the amount of the data produced by the CFD to update the rendering in real time
2. They were not capable of rendering photo realistic fire and smoke

These visualization methods where integrated as plugins for the Cover renderer and can be managed from the user interface. The photo realistic rendering of smoke is done by a fast volume rendering approach which takes advantage of the availability of programmable shader functions on modern graphics boards. This way frame rates around 25fps for the volume rendering of the CFD-results are possible on normal PC hardware. To achieve a photo realistic rendering of fire a fractal 3D texture is applied to the regions of the flames. As CFD results are to coarsely spaced compared to the fast visual fluctuations of a flame front, this behaviour is interpolated by the fractal texturing process until the availability of the next CFD result.

3. CAPABILITIES

3.1. Visualisation

At the current development stage the simulator is able to perform simulation runs for predefined missions. The results can be visualized in 3D on the HMD or the CAVE environment. Navigation in space is supported by a space mouse device and also navigation in time is possible by a simple “VCR-like” graphical user interface. Within this user interface the user can create and define new missions, edit existing ones and start new calculations. The visualization system shows these new results as soon as they are available on the database server.

The following visualization methods are already available on most platforms:

1. Line integral convolution (Figure 2)
2. Streamlines (Figure 3)
3. Isosurfaces (Figure 4)
4. Bill boarding method for realistic smoke visualization (Figure 5)

Figure 2: Line Integral Convolution

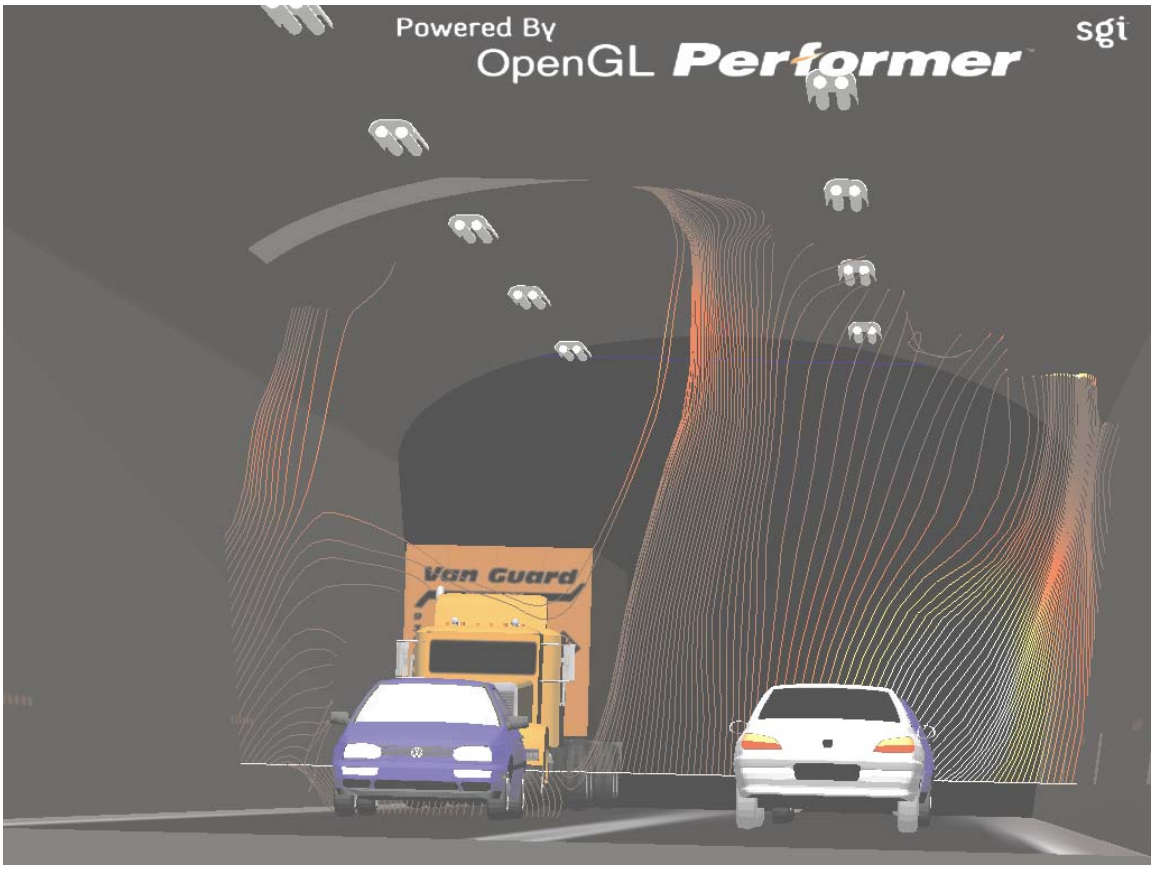


Figure 3: Streamlines

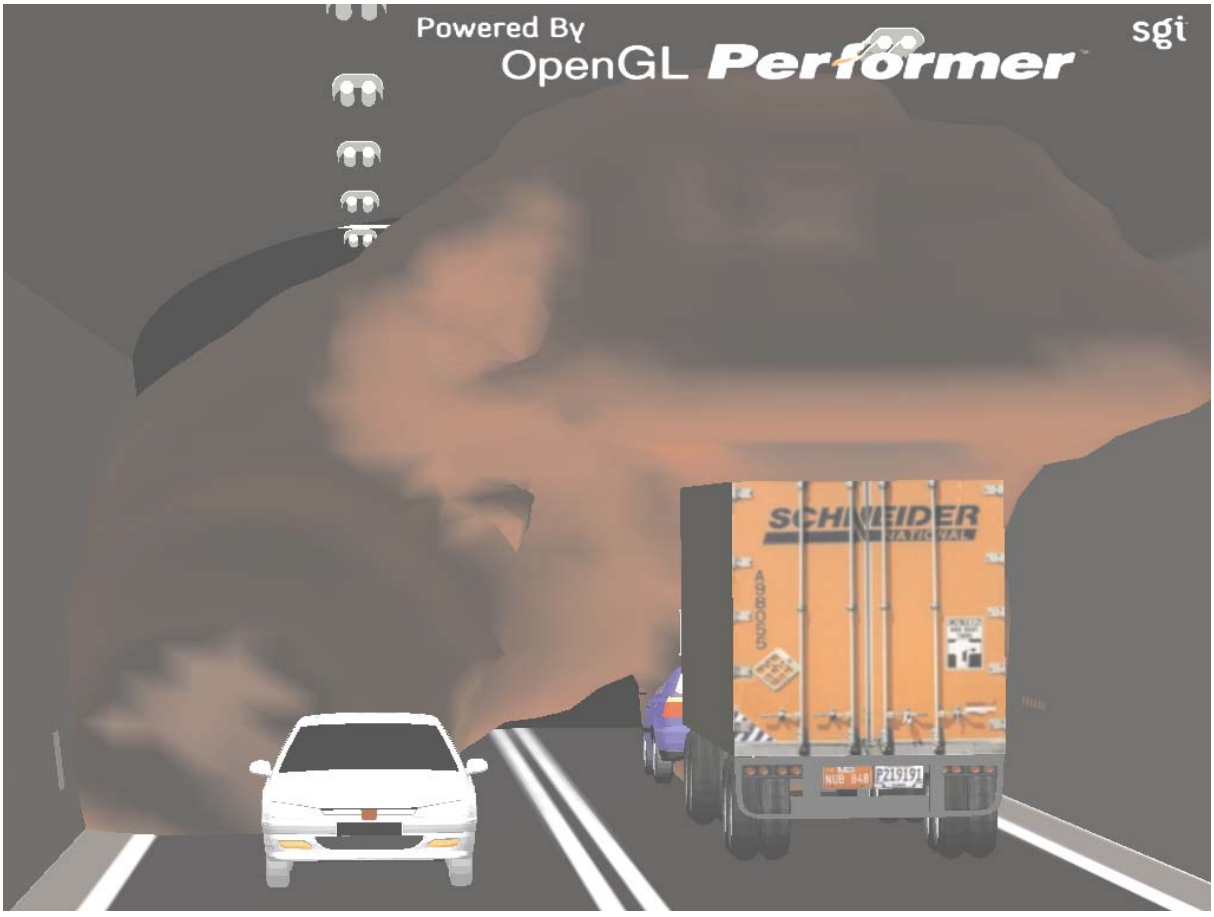


Figure 4: Isosurfaces

Figure 5: Bill Boarding

3.2. Real time CFD Simulation

Computer programs are usually sequential meaning that their execution is done by only one processor. The computational time varies extremely from a few seconds up to several weeks. Especially in the second case speeding up the computation is of particular interest. One can rely on the steady increasing performance of processor technology or try to parallelise computer codes. The idea of parallelisation is to spread the workload to several processors and therefore speed up computation. The ultimate objective of the Virtual Fires system is to perform real time simulations of tunnel fires in a concurrent VR environment. Since in general CFD calculations are very CPU demanding, it is not possible to perform this on today‘s single processor systems. Therefore, the parallelisation of the Lattice Boltzmann code ICE is a must to achieve the aforementioned objective.

There mainly exist two classes of parallel programming paradigms:

- The shared memory paradigm consists of sharing data through a common memory by using compilation directives. This paradigm allows using parallel machines without major changes to the sequential code, but becomes inefficient for larger numbers of involved processors due to memory bandwidth limitations or in inhomogeneous environments.
- The distributed memory paradigm consists in distributing the data to the processors to share the work load. Processors requiring information located in another processor have to communicate through messages. As the messages are sent over a network connecting the processors the amount of communication should be minimal. The distributed memory model requires much more programming effort but leads to more efficient codes and can be even used for cluster solutions.

In view of the available hardware and the requirements to the program it was decided to use a distributed memory parallelisation. Distributed memory systems are characterised by a high scalability and the large physical memory available normally. The performance is influenced by the balance between CPU speed and network speed and depends heavily on the programmer. In the MPMD (multiple program - multiple data) programming model each processor executes its own program. The communication between the processors is performed by sending and receiving messages. A subroutine library which contains functions for sending and receiving messages. The data distribution and communication must be explicitly defined by the programmer. Although it is probably the most difficult approach to parallel programming it is selected due to the following reasons:

- It promises the highest performance on distributed memory systems with a large number of processors.
- It is the most portable approach .
- The programmer has all freedom to optimise communication.

There are a few message passing libraries available, but for VirtualFires the Message-Passing-Interface (MPI), the de facto standard, is used for portability. A full description of the MPI standard is given in[3]and [4]. Up-to-date information can be found on the web site of the MPI forum [5]. The VirtualFires project uses a Linux Cluster located at KTH in Sweden. The performance of the code is tested with two different configurations:

1. Test case A consists of 60.000 computational cells and is the most representative for the current use of ICE within the Virtual Fires project
2. Test case B consists of 250.000 cells

Figure 6 shows the speedup factor for the different configurations. For large problems (Test case B) the speedup factor increases linear to the number of processors. For small problems (Test case A) the performance breaks down if more than 8 processors are involved due to the relative increase of communication compared to calculation. Investigations are being performed to optimise the communication so that a real time simulation can be achieved.

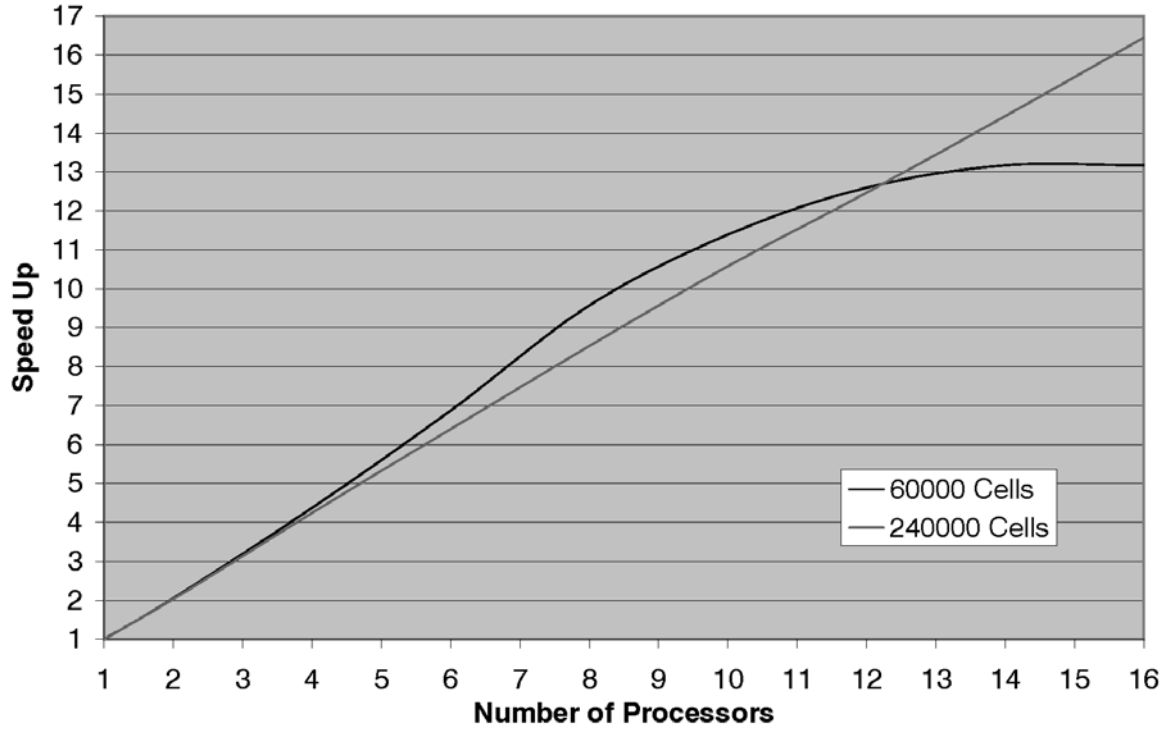


Figure 6: MPIce on Lucidor

4. EXAMPLES

For demonstration purposes some popular fire incidents have been calculated with the simulator. These datasets also serve as a base for the verification of the system. The calculated dataset consists of different ventilation scenarios for the Mt. Blanc tunnel in France and the Gleinalm tunnel in Austria. Both tunnels where examined with their former ventilation system and also with the improved ones after the reopening. Example of fire simulations is given in Figure 7. Also a typical subway station in Dortmund has been analysed. In Figure 8 the temperature distribution inside the station during a fire incident on a subway train is shown. Together with the calculation of smoke spread this kind of simulation is important for the fire fighters to plan their missions inside these stations and to verify that their strategies are efficient.

Figure 7: Mt. Blanc Tunnel

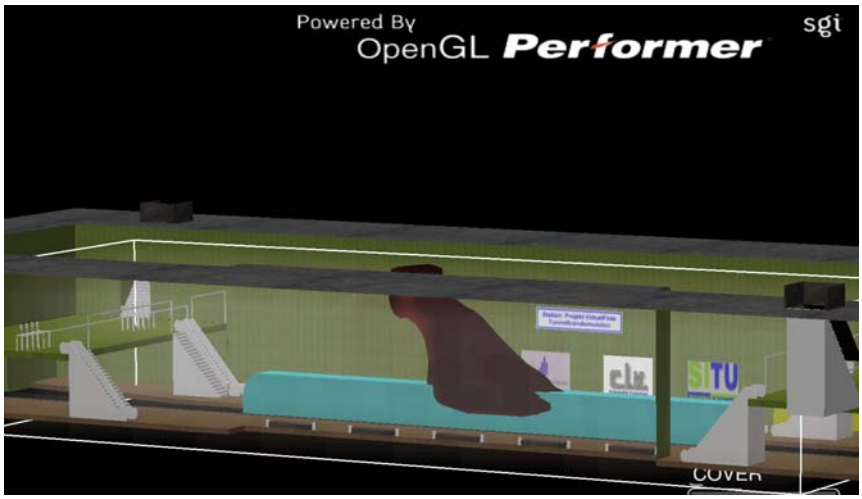


Figure 8: Subway Station Dortmund

5. CONCLUSIONS

A simulator was presented which allows fire men to perform virtual training exercises with a head mounted display or in a CAVE environment. The simulator also allows to assess the fire safety of existing tunnels and can be used as a tool for designing new tunnels. A prototype of the system will be available in May 2004 and it is expected that the system will be marketed world wide.

6. ACKNOWLEDGEMENTS

The work reported here was supported by the European Community under the 5th framework (IST) program.

7. REFERENCES

[1] D'Humieres, D., Ginzburg, I., Krafczyk, M., Lallemand, P., Luo, L.-S., (2002) Multiple-relaxation-time lattice Boltzmann models in three dimensions, Phil. Trans. R. Soc. Lond. A 360, p. 437 – 452.

[2] COVISE, Vircinity GmbH, Nobelstraße 15, 70569 Stuttgart Germany, http://www.vircinity.com

[3] Snir, M., Otto, S., Huss-Ledermann, S., Walker, D., Dongarra, J., (1996) MPI: The complete Reference, MIT Press, 336 pp.

[4] The MPI Forum, (1997) MPI-2: Extensions to the Message-Passing Interface, University of Tennessee, 362 pp.

[5] http://www.mpi-forum.org