The Abalone Interpolation
A Visual Interpolation Procedure for the Calculation of Cloud Movement

Elke Hergenröther
Fraunhofer Institute for Computer Graphics
hergenro@igd.fhg.de

Antonio Bleile
ask – Innovative Visualisierungslösungen GmbH
bleile@ucar.edu

Don Middleton
The National Center for Atmospheric Research
don@ncar.ucar.edu

Andrzej Trembilski
Fraunhofer Institute for Computer Graphics
trembils@igd.fhg.de

Abstract

For the production of smooth animation of flow simulation out of only several data steps many interpolated intermediate data steps are needed. Especially for the meteorological visualization of moving clouds for TV broadcasting purposes such interpolations are of high importance. The traditional method for the animation production is to fade one dataset out and to fade the next dataset in. The visual result is not very realistic if the clouds should move quickly and the distances between them are too great, because blending produces no clouds movements. In this paper we present the Abalone Interpolation for the calculation of visually realistic cloud movements. The presented solution is not only suitable for meteorological visualization but also for visual preparation of the flow calculation results.

Keywords: Cloud movement, interpolation, simulation.

1. Introduction

Numerical weather simulation models, like these of the National Center for Atmospheric Research (NCAR) or the German Weather Service (DWD), produce twice a day large quantities of simulation data, which must be visualized for the presentation on television. Even if the simulation programs internally generate high amounts of finely resolved data they can only store data blocks lying 30-60 minutes apart because of the extremely high storage requirement and also the long time needed for the storage process itself. On the other hand, for the presentation of weather forecast on television, time-lapse animations of up to 30 seconds are needed.

2. Problem description and related work

Nowadays the standard TV weather visualization software like TriVis [1], [2], [3] uses blending for cloud animation. The clouds of data time step i disappear slowly as clouds of data time step i+1 slowly appear. If in two following time steps the clouds are close to each other, the spectators hardly notice this, but if there are wide cloudless areas in-between, the animation appears quite bad, because clouds are popping up and disappearing again. This method does not take advance of the wind field data, which is always another result of any weather simulation. For the generation of a realistic animation we need moving clouds. The simulation of the cloud movement is the purpose of our Abalone interpolation algorithm. One of the problems is to find the corresponding cloud areas in two following time steps.

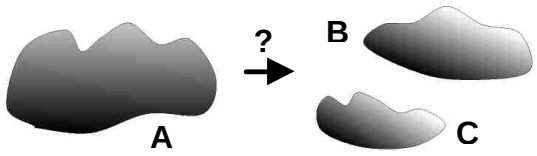

Figure 1. Did clouds B and C develop from cloud A?

Figure 1 shows the initial situation: the first data set (left side) contains only one cloud, in the other data set (right side) there are two clouds different positions. During the animation the spectator will find out if the cloud A corresponds with cloud B, or with cloud C, or even with both of them (after it divided) or maybe with none, because B and C just appeared as a product of a different atmospheric effect.

So the main problem is how to solve this correspondence problem having two data steps being 30-60 minutes apart. A method for solving this problem is to search for some specific features, which can be found in clouds of both data sets. With these features and the wind data set a prediction can be made about the probable movement of the cloud. However, clouds quite quickly change their appearance and the air humidity, which is the base value for extracting a cloud from the simulation data

set. The humidity is always very close to 100%, so the feature extraction from a cloud data set is quite a problem. A single cloud has nothing really special to be easily identified and recognized by an algorithm. We hoped to find a general solution of this, which with some luck could be reused to visualize similar flow field problems. We found an idea for the solution after playing a game of "Abalone". An Abalone player has to build groups of his spherical stones on a playing grid and uses them to move his opponent's (smaller) stone groups out of the grid. Now think of the black stone groups as of clouds and of white stone groups as of the air masses around them. The white stones move the black ones over the grid and vice versa. A wind flow field replaces the human-controlled movements. In this way not only the clouds are considered as particles (though they are our main interest) but also the surrounding air masses. In this way the whole grid is considered as being full of particles and now the solution is much easier.

Silver [4], [5], [6], [7], [8], [9], [10] found a solution to the problem of correspondence of a flow field features in two following time steps. She extracts objects from the flow field and follows their movements, developing a strategy for the reduction of memory usage. The detailed form of the object was not interesting for the special problem investigated by Silver. The aim was not to find time-interpolated data steps but the recognition of given features in given following time data steps. In contrast to this our purpose is to interpolate the given cloud data steps in time to get more data steps to produce a smooth animation of clouds. For a smooth and realistic animation for broadcasting purposes a highly detailed data is of high interest. In fact for the interpolation of detailed objects morphing could be a suitable method. Most of the morphing algorithms can be classified into two classes: parametric correspondence method or implicit function interpolation method. The parametric correspondence method algorithms search for corresponding points on the boundary of the bodies to be transformed into each other. Examples for this kind of algorithms are the Sedeberg method [11] and the method of Alexa [12]. For the function based method a function is developed for the start and the target body. Then the start function is transformed into the target function in a possibly smooth way. There are two kinds of functions used for this: the inside/outside function or the characteristic function. The inside/outside function is binary, which means that ‘0’ means the actual position is outside of the object and ‘1’ means it is on the object boundary. The signed distance function is more explicit, it gives the euclidic distance between the actual position and the next point on the object boundary. Morphing algorithms based on this type of function generate very plausible results already when using very simple interpolations. After Türk et.al.[13] the parametric method is mostly quicker and needs less memory than the function-based method. The drawback is that the transformation of objects with different topology is more complicated than with the function-based method. The function-based method has also less problems with self-penetrating faces. Türk et al. [13] combined the parametric method with the function-based method and developed a smooth transformation for objects in every dimension. For using one of these methods you need to know for which objects the interpolation should be calculated. We do not know this (see figure 1) therefore morphing is not the best solution of our problem. But one part of the morphing-algorithms is the solution of the correspondence problem and we have a correspondence problem. The solution of our problem is not the computation of correspondence of polyhedral bodies but the analyze the whole volume of the model. The Abalone Interpolation computes the correspondence of two volume data sets of a fluid simulation.

3. Abalone interpolation algorithm

The Abalone Interpolation Algorithm is subdivided into two sub-algorithms, which is comparable with one of Silver's tracking procedure [10]. First, the movement of the clouds and the air is traced over the time steps, which have to be interpolated. Then the time steps have to be re-traced to find the corresponding clouds. This is done indirectly by interpolating the (scalar) humidity values of the particles. In the following we discuss these two parts in more detail.

3.1. Particle tracing

To follow the movement of the particles through the interpolated data steps, the wind flow fields have to be interpolated through the time. Therefore we compute the flow simulation using Runge-Kutta's 4-th order interpolation, which is the best trade-off between computation time and correctness of the result [14].

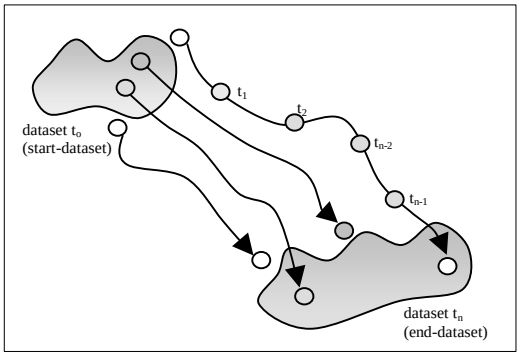

Figure 2.: Particle tracing through all interpolated flow fields.

Knowing the interpolated wind flow fields we can start the computation of the particle movement. We start putting a particle on every grid point of the given data set. In fact, there are only two kinds of particles: cloud particles and non-cloud particles. We now trace their movement through all interpolated time steps, starting with the data set t_0. Figure 2 shows possible ways of particles starting from t_0 to t_n which is the second given meteorological data set. The intermediate data sets $t_1 - t_{n-1}$ are the interpolated ones.

3.2. Interpolation of the particle values

After the particle tracing is done it has to be checked, if they changed their status after the flight. This can be done, by checking if they landed in the same medium (air or cloud) as they started. If they did not, their values have to be interpolated. Otherwise the particles keep their value. In our sample implementation of the Abalone algorithm we do not have a binary case. We use the particle humidity, which is a scalar value produced by the meteorological simulation. In Figure 3 the four cases are shown which can occur:

1. A non-cloud particle lands in a cloud
2. A cloud particle lands out of a cloud
3. A cloud particle lands in a cloud
4. A non-cloud particle lands out of a cloud

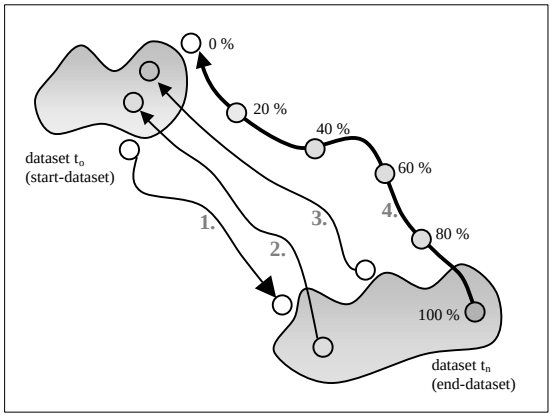

Figure 3. The Interpolation of the particle values

In cases 3 and 4 the scalar humidity value will approximately stay constant. In cases 1 and 2 the particle value has to be interpolated during the flight. We use a linear interpolation for this computation. In Figure 3 the interpolation needed is visualized.

4. " Proof of concept"

The presented algorithm can be divided into two parts. At first a particle with its own scalar value is set at every data grid point. These particles are traced through the interpolated wind flow fields. The second part of the algorithm compares the starting values of the particle and the scalar value of the data grid point where the particle lands and – if needed - interpolates the scalar value of the particle for the intermediate data sets t_1 - t_{n-1}. The method is simple and usable for visual interpolation of given data sets. What we still need is a proof that it works and meets the specification. This proof can be found in the next section.

4.1. Analysis of the Abalone-interpolation

Abalone Interpolation is a general algorithm for visual interpolation of flow-dependent data sets. It should be used for visual processing of the data, which should not be altered by its usage. As the algorithm should only be used for visual analysis of data, physical correctness is not provided. This feature provides the advantage of independency of the original fluid problem and results in a general usage of the algorithm as a visualization tool. We start with given meteorological simulated humidity data sets m_0 ... m_u and the corresponding wind data sets. We call the mi data set t_0 and the m_{i+1} data set t_n during the abalone interpolation. This section discusses the assumptions we made, describes every single module of our procedure and shows that the result is correct as long as the assumptions are right.

4.1.1. Correspondence condition

We assume that there is a direct coherence between the wind and the movement of the clouds, hence if a particle was in a cloud in the data set t_0 and lands again in a cloud in a data set t_n then we assume it was in a cloud for the whole time. We assume a correspondence between the clouds from the data sets mi and m_{i+1}. The existence of this correspondence is a basic condition, which has to be met for the correctness of the procedure. If this correspondence condition is not met, the error can not be found directly, but can be detected using a higher time resolution of the given data set m.

4.1.2. Interpolation of the cloud movement of $m_0 \rightarrow ... m_i \rightarrow m_{i+1} \rightarrow ... \rightarrow m_n$

If the correspondence condition is met it becomes clear, why the movement of the clouds from m_i to m_{i+1} can be computed. We can now also compute how clouds divide into several parts. Indirectly the dissolving of old clouds and the development of new clouds can be

computed, too. In case 2 (see section 2) a cloud particle from data set t_0 lands out of any clouds in date set t_n. If the correspondence condition is met, we can assume that the part of a cloud, which was represented by the particle was dissolved. The cloud existing in the meteorological data set mi disappears in m_{i+1}, which could be for instance caused by rain. In the same way a development of a new cloud can be computed. Of course, if the interpolation works between mi and m_{i+1} it works for the whole set of the given simulation data m_0,..., m_i, m_{i+1}, ..., m_n .

4.1.3. Extrapolation of particle values

For the implementation of the algorithm we should take into consideration that the particles do not flow exactly to a given grid point but in most cases land somewhere between the grid points. It means that for every interpolated data set the extrapolation of the particle values has to be computed (see figure 4). The extrapolation of the particle values correspond to the trilinear interpolation of the grid point values.

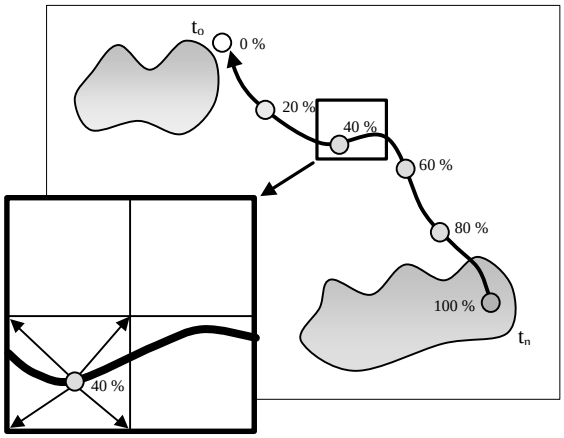

Figure 4. The particle value extrapolation

4.1.4. Interpolation of the cloud movement of $m_n \rightarrow$ ... $m_{i+1} \rightarrow m_i \rightarrow$ ... $\rightarrow m_0$

Until now we only showed that the interpolation works in the direction from m_0 to m_n.. We did not determine if it works in the reverse direction $m_n \rightarrow m_0$. If the Abalone Interpolation was a bijective function, the correctness could be shown easily. Unfortunately, during the interpolation more than one particle of t_0 can land in one voxel of the grid t_n. Other voxels of the data set t_n get no particles at all. The interpolation is not bijective. In figure 5 the result of the algorithm (as presented so far) is shown.

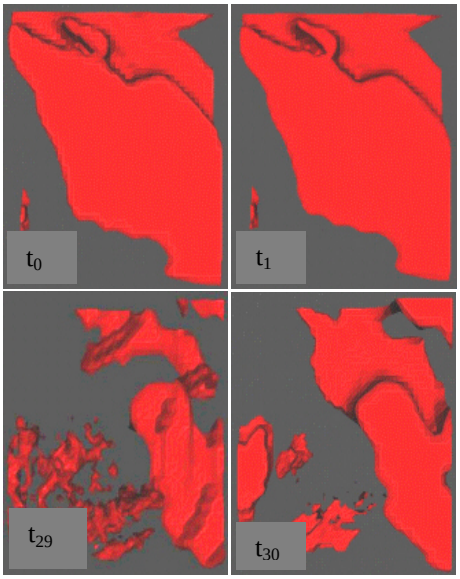

Figure 5: The last iteration of Abalone (t_{29}) differs from the next original data set (t_{30})

The data set interpolation is very smooth. However the last computed interpolated data set t_{29} is very different from the given meteorological data set t_{30}. We guessed this difference is resulting from particles moving into and out of the voxel mesh. Until now we were not speaking about the particles moving into or out of the analyzed area in t_i ($0<i<n$). The particles leaving the area are automatically considered. Even if the value of a particle getting out of the area at t_n is not known, we can make some assumptions about it, which make it possible to compute its value. We found the following assumption working best: If a particle leaves the grid area at t_i ($0<i<n$) it keeps it last value. On the other hand particles, which started outside the area of the dataset cam move into the area by the wind. They can also develop from ascending vapor. A quite obvious but memory intensive method to solve this problem is to introduce a “backward interpolation”. The starting data set is declared to be the final one and vice versa. The flow field is also reverted. In this way the incoming particles are integrated into the Abalone computation. During the backward interpolation they are treated as particles moving out. Finally the forward and backward interpolated data sets are weighted and combined. In this way both kinds of particles are integrated into the Abalone Algorithm.

Figure 6 demonstrates that the introduction of the backward interpolation the particles moving into and out of the grid are integrated into our method. The interpolation results change smoothly from data set t_0 to t_n.

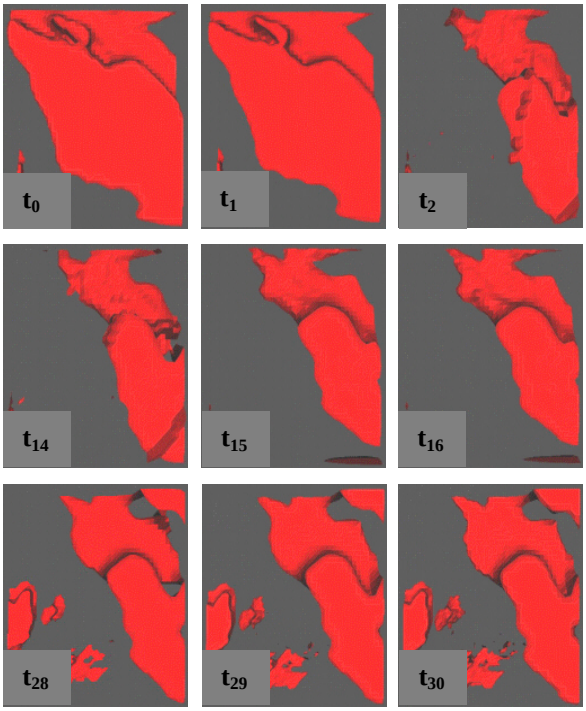

Figure 6: Results of the combined backward and forward interpolations.

5. The compiled Abalone Interpolation:

The different steps of the Abalone-Interpolation can be compiled to one procedure as follows:

- Interpolation of the wind vector data for every frame
- Computation of the forward interpolation with $m_i = t_0$ and $m_{i+1} = t_n$ and for the backward interpolation with $m_{i+1} = t_0$ and $m_i = t_n$. These computation include the following steps:
  - Tracing the particles through all the frames t_0 to t_n.
  - Interpolation of the humidity value of the traced particles
  - Extrapolation of the humidity value of a particle at the surrounding grid values.
- Weighted combination of the data sets computed by the forward and backward interpolation.
- Isosurface generation or raytracing

The described procedure for visual interpolation of flow-dependent data needs quite much memory: 3n*resX*resY*resZ. The computation time is linear with the number of data steps, O(n), but there is a high constant, depending on the number of interpolation steps and the size of the data grid. Thus an optimization of the procedure is a needed.

6. Optimization with "Divide and Conquer"

As described so far, the particle values are interpolated first and then extrapolated on the data grid. During the visualization we found that the clouds in the interpolated frames loose some of their volume. These losses are caused by the interpolation and extrapolation. By inevitable numerical errors noise is introduced into the procedure, which is perceived as a volume loss of the clouds (see also figure 6). To avoid the volume losses we had to leave out the extrapolation. At first, as in the not-optimized version, the wind vector data sets are interpolated. However in the next step the particles are not traced any more but a divide and conquer method is used (see figure 7). For every grid point of data set $t_{n/2}$ we find the starting position of a particle from t_n which lands at this grid point. We also look for the further way of the particle and find the grid position where the actual particle is going to land in t_n. The humidity values of the grid in t_0 and t_n are well known, so the value of the particle can be interpolated as before. This calculation step has to be repeated recursively. In this way all grid values of t_i ($0<i<n$) can be calculated. In this way the extrapolation can be left out and we save also 50% of the memory compared with the original procedure, because the particle positions do not need to be buffered for all the computed frames. Only the wind fields have to be still held in memory. However even the interpolated winds can be computed on the fly for a further memory optimization, but only on the cost of higher computation time in the interpolation step.

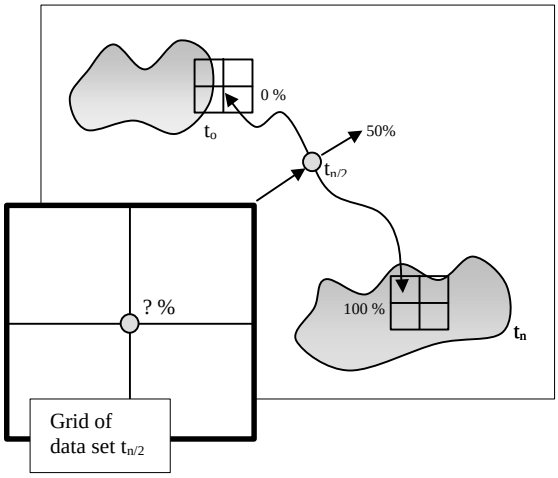

Figure 7. Optimized Abalone Interpolation

The results are presented in three movies. Pictures of the movies (figure 8 and figure 9) are attached on the end of the paper. In the first fire movie (fire.mpg) you can see the visualization of the original datasets. The frames are

repeated. So it is easy to compare the original frames with the interpolated frames (interp.fire.mpg). The cyclone movie shows 37 original time steps with 18 interpolated time steps in between. The grid resolution of the used dataset is 80 x 96 x 35. The calculation of the abalone interpolation takes 45 minutes and for rendering we need 14 seconds per frame on an AMD Athlon 1,2 GHz.

7. Summary and discussion

The most important condition for the proper work of the Abalone Interpolation is the correspondence condition. If the data does not fulfill this condition the interpolation has errors, which are not perceivable in the visualization. For meteorological applications this condition can only be violated if the data set m_i contains a cloud, which is replaced by a completely new cloud in the data set m_{i+1}. If there is no information about the disappearance of the old cloud and the creation of a new cloud, the process will not be detected by our algorithm (and, probably, by no other). This limitation can be overcome by the addition of new meteorological information. In this paper we do not do it, because we wanted to keep the procedure universal. Of course it is possible to extend the algorithm to adjust it for this specifically meteorological problem. The universal usability, an easy implementation and visually convincing results are important advantages of the Abalone Interpolation. The attached file interp.fire.mpg is an example animation of a fire flow simulation and compare them with the original fire simulation data set visualization in file fire.mpg. Some frames from the animations are shown in figure 8. The original goal of our development was the visualization of meteorological simulation data. The introduced procedure can be used to generate more detailed and prolonged animations out of given data sets. In this way details can be visualized in the animation, which would not appear in a shorter film. The optimized procedure needs only the memory for the computation of the data set m_i and m_{i+1}. The computed data can then be stored as a polygonal model of the iso-surface or as a volume grid.

8. Future Work

Generally the Abalone Interpolation can process any fluid simulation data. It would be interesting to find out if the method can be reused for processing of data sets which were not generated by fluid simulations, for example finite element simulation data. Another important step will be the integration of the rendering data into rendering algorithms as these shown in [15], [16], [17], [18].

9. Acknowledgements

The data used for the movies are simulated by Dr. Ying-Hwa Kuo form The National Center for Atmospheric Research (NCAR) in Boulder, Colorado. We want to thank him for providing the data. Without this interesting datasets the presentation of the results would be much less attractive. We also want to thank Hans-Joachim Koppert from German weather services (Deutscher Wetterdienst) in Offenbach, Germany for his helpful suggestions and for the data used in figures 5 and 6. Our sincere thank are given Dr. Florian Schröder from ask – Innovative Visualisierungslösungen GmbH in Messel, Germany. He supported the development, because he always thought Abalone was a good idea.

10. References

[1] Koppert, H.-J., Schröder, F., Hergenröther, E., Lux, M., Trembilski, A. : "3D Visualization in daily operation at the DWD". *Proceedings of the Sixth ECMWF Workshop on Meteorological Operational Systems*, Reading, England, 1998.

[2] Haase, H., Bock, M., Hergenröther, E., Knöpfle, C., Koppert, H.-J., Schröder, F., Trembilski, A., Weidenhausen, J.: "Where Weather Meets the Eye -- A Case Study on a Wide Range of Meteorological Visualisations for Diverse Audiences". *Proceedings of IEEE/EG VisSym'99* (Springer- Verlag), Vienna, Austria, May 1999

[3] Haase, H., Bock, M., Hergenröther, E., Knöpfle, C., Koppert, H.-J.,Schröder, F., Trembilski, A., Weidenhausen, J.: "Meteorology meets computer graphics - a look at a wide range of weather visualisations for diverse audiences". *Computers & Graphics 24 (2000)*, pp. 391-397.

[4] D. Silver, R. Samtaney, N. Zabusky, J. Cao: "Visualizing Features and Tracking Their Evolution," *In: IEEE Computer, Volume 27 Nr.7*, pp. 20 – 27, July 1994

[5] D. Silver, "Object-Oriented Visualization", *IEEE Computer Graphics and Applications*, Volume 3 Nr. 15, pp. 54 – 62, 1995

[6] D. Silver, X. Wang, "Volume Tracking",*R. Yagel and Nielson, editors, IEEE Proc. Visualization '96*, Computer Society Press, pp. 157 – 164, 1996

[7] D. Silver, X. Wang, "Tracking and Visualizing Turbulent 3D Features", *IEEE Transactions on Visualization and Computer Graphics*, Volume 3 Nr 2, pp. 129 – 141, 1997

[8] D.Silver, X. Wang, "Visualizing Evolving Scalar Phenomena", *Invited Paper, Journal of Future Generations of Computer System*, 1998

[9] D. Silver, "Tracking Scalar Features in Unstructured Datasets", *D. Ebert H. Hagen H. Rushmeier, editors, IEEE Proc. Visualization* '98, pp. 79 – 86, Computer Society Press, 1998

[10] T. van Walsum, F. H. Post, D. Silver, F. J. Post, "Feature Extraction and Iconic Visualization", *IEEE Transactions on Visualization and Computer Graphics*, Volume 2 Nr. 2, pp. 111-119, 1996

[11] Sederberg, T., Greenwood, E., "A physically based approach to 2D shape blending", *Computer Graphics*, 26, pp. 25-34, 1992

[12] Alexa, M.; Cohen-Or, D.; Levin, D. “As-Rigid-As-Possible Shape Interpolation”, *ACM SIGGRAPH 00*, New Orleans, USA
[13] Turk, G.; O’Brien, J, “Shape Transformation Using Variational Implicit Functions”, *ACM SIGGRAPH 99*, Los Angeles, CA USA
[14] Frühauf,T. “Graphisch-Interaktive Strömungs-visualisierung", *Dissertation an der TH Darmstadt*, FB Informatik, GRIS, Springer Verlag, 1997.
[15] A. Trembilski: "Two Methods for Cloud Visualisation from Weather Simulation Data", *WSCG 2000*, Plzen, Czech Republic, pp. 192-196
[16] Andrzej Trembilski, Two Methods for Clouds Visualisation from weather simulated Data, *The Visual Computer, International Journal of Computer Graphics*, Volume 17, No 3, May 2001, pp. 179-185
[17] Trembilski, A., Brossler, A., "Transparency for Polygon Based Cloud Rendering", *Proceedings of ACM Symposium on Applied Computing (SAC)*, Multimedia and Visualization Track, Madrid, Spain, March 10-14, 2002, pp. 785-790
[18] Trembilski, A., Brossler, A., "Surface-Based Efficient Cloud Visualization for Animation Applications", *Proceedings of the WSCG'2002 - the 10-th International Conference in Central Europe on Computer Graphics*, Visualization and Computer Vision'2002, University of West Bohemia in Plzen, Czech Republic, February 4-8.th, 2002, pp. 453-460.

Attachment:

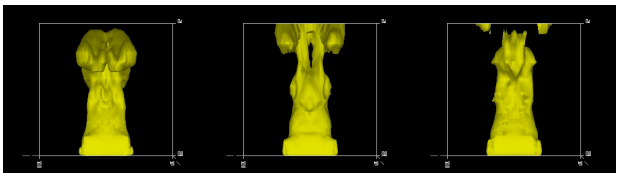

Figure 8: Some frames from a sample visualization of a flow simulation of a fire. See the video file fire.mpg for the whole animation.

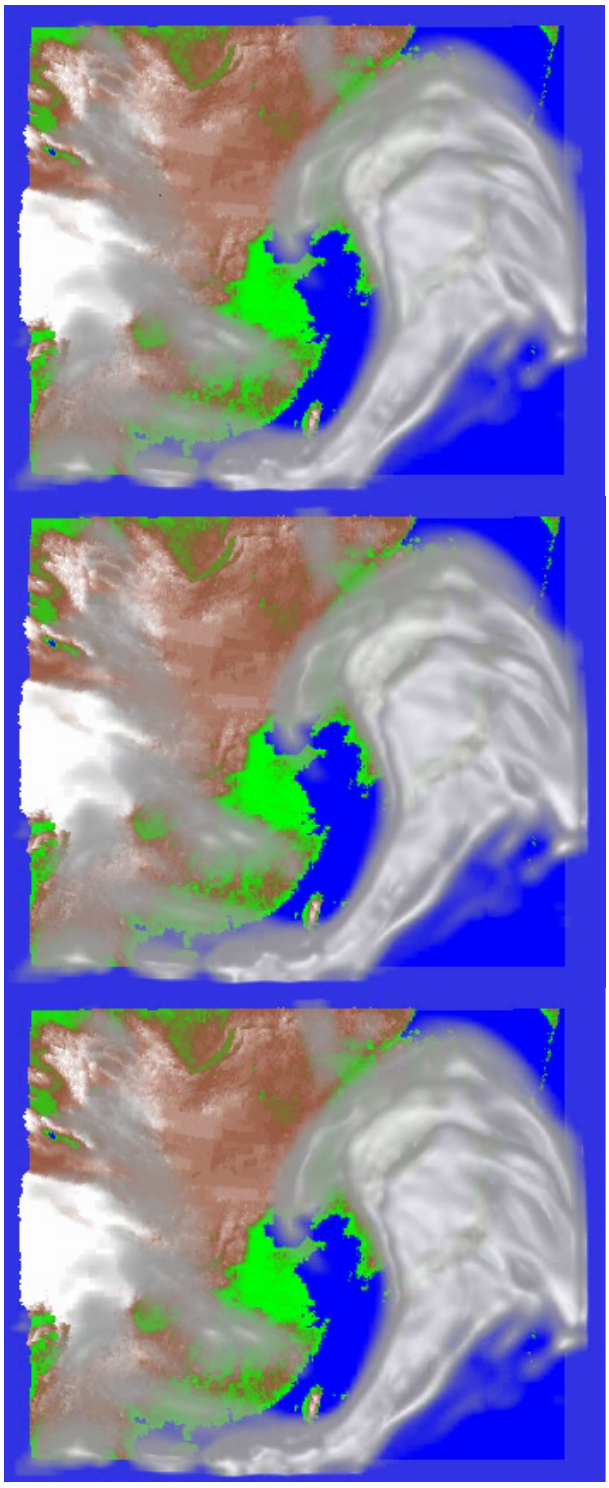

Figure 9: Some frames from a sample Abalone Interpolation used for meteorological visualization. See file cyclone.mpg for the whole animation.