Comparative Visualization of Instabilities in Crash-Worthiness Simulations

Ove Sommer and Thomas Ertl

University of Stuttgart, IfI, Visualization and Interactive Systems Group
{sommer, ertl}@informatik.uni-stuttgart.de
http://wwwvis.informatik.uni-stuttgart.de

Abstract. Since crash-worthiness simulations get more and more important as part of the car development process in order to reduce the cost of development, enhance the product quality, and minimize the time-to-market, the reliability of the simulation results plays a decisive role concerning their significance. Recently the simulation departments of several automotive companies started investigating the quantity and reason for deviations during a number of simulation runs on the same input model.
In this case study we discuss different measurements for instability and present a texture-based visualization method which allows the engineers to efficiently explore the simulation results by interactively hiding finite element structures with nearly constant crash performance. Furthermore, we describe those parts of our prototype which use a CORBA layer for providing the same view on a set of simulation results and allowing the visual comparison by using the marker functionality.

1 Introduction

In recent years simulation has become more and more important for the development of new cars. It supports the testing with hardware-prototypes by delivering simulation results which are close to test results. This makes the reduction of hardware-prototype tests possible and therefore allows the development at a lower price. Furthermore, the shorter cycle of simulation allows the evaluation of more iterations of variants, and thus better or safer car body parts which improves the product quality.

In the field of crash simulation the continuously increasing CPU power of high-end simulation servers and the parallelization of the simulation software leads to

- **models of finer mesh resolutions**. Finer models map the crash-worthiness of a car body more exactly.
- **extensive tracking of more model parameters**. The chance to correlate the temporal behavior of different simulation variables by using state-of-the-art visualization techniques allows the engineers to come to a deeper understanding.
- **more simulations runs**. The more iterations can be computed the more improvements can be done to the structure of car body parts.

Recently the stability of the simulation process is investigated in order to ensure the reliability of simulation results. For this purpose one and the same model will be simulated several times and the results are compared to each other.

In this case study we describe the statistical methods that are used to compare the simulation output and to evaluate the achievable stability. We will discuss two categories of comparison functions. We present a visualization method which allows the engineers to detect regions of instability even in complex models. And finally, the direct visual comparison of multiple simulation runs using marker functionality together with a CORBA connection layer will be presented.

2 Stability calculation

Today a finite element mesh of a whole car body model consists of about 500.000 elements and nodes. For crash-worthiness simulations the first 120 milliseconds of an impact are computed and the coordinates are stored in 60 time steps (*states*) together with tracked variables like velocities, forces or strains, which takes about 50 hours on 6 CPUs of a modern simulation server. 2000 simulation iterations are calculated before the next state is appended to the result file. During simulation several kinds of ramifications will cause differing results. They originate as well from the limited precision of the numerical process as from the structure of the finite element mesh. For example, if one shell element A is pressed against another element B which has a normal that lies in the plane of A then the simulator has to determine the direction in which A will slide on B. Those ramifications are called the *instability* of a simulation process.

In order to evaluate the effects of the replacement of any car body part by a variant it is absolutely necessary to be able to reduce the impact of instabilities caused by the design of the meshed car body model. At the time the engineers try to spot those regions which are responsible for unstable crash dynamics. Multiple runs for the same model with the same boundary conditions are computed. The simulation results are compared against each other by using appropriate evaluation functions.

The set of evaluation functions can be split in different classes: if they use the output data directly or proceed on a previously computed measure, if they represent a local or a global criterion, and if they use one- or multi-dimensional data. In the following three examples of geometric comparison functions are illustrated.

2.1 Global measurement functions

The function $\mathrm{V}(p,t,r) = p(t,r) - p(0,r)$ measures the displacement of the mesh node p in the t'th state from its original position in the initial state for one simulation run r. This is done per component or Euclidean. The length of the displacement vector is compared over all simulation runs R.

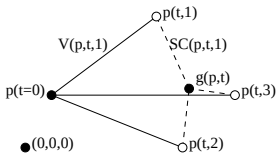


Fig. 1. The outline shows the position of node p in the initial state ($p(t = 0)$) and after t time steps of three different simulation runs. The solid lines mark the displacement function $\mathrm{V}(p,t,r)$, the dashed lines the scatter function $\mathrm{SC}(p,t,r)$.

The scatter function $\mathrm{SC}(p,t,r)$ expresses the distance of node p to the centroid $g_c(p,t)$ which is calculated as $g_c(p,t) = \frac{1}{R}\sum_{r=1}^{R} p(t,r)$, where R is the number of runs. Here, the projection to one main axis could also be investigated to focus on differences in the specified dimension.

The drawback of both, the displacement and the scatter function is, that they are global measures. For example, during a front-crash simulation an instability at some finite elements of the engine mount will influence the values of a wide area of adjacent car body parts and will even force deviations in the rear part of the car. The largest differences between simulation runs occur in regions of intense deformation as Fig. 2 shows. The deviation of corresponding mesh coordinates becomes less for larger distances from the center of most buckling.

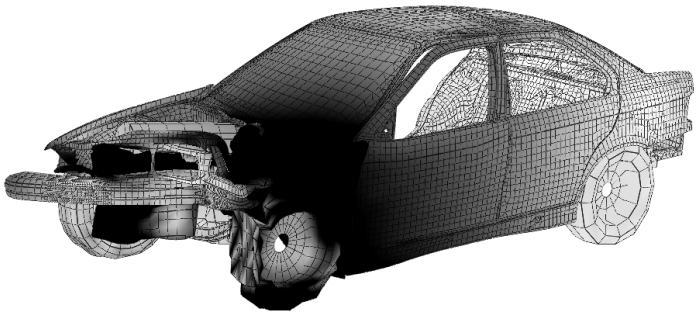


Fig. 2. The grey-scale (see Appendix for color version) maps the length of difference vectors of corresponding nodes. After 80 milliseconds the largest deviation can be found in the left front side (dark regions) and smaller values in the rear. However, using a global measurement it is hard to spot regions where different crash behavior originates.

In order to spot the origin of instability the engineers need another criterion because these global measurement functions detect large regions without determining if the deformation of a set of finite elements is the reason or the result of instability. A more adequate measurement is provided by a local criterion.

2.2 Local measurement function

At the Institute for Algorithms and Scientific Computing (SCAI) of the German National Research Center for Information Technology (GMD) a local deformation criterion [2] has been developed within the Autobench project, a research project driven by some of the leading automotive industry companies and financed by the German Bundesministerium für Bildung und Forschung [1]. This criterion considers the displacement of a node at time step t with regard to its neighborhood nodes in comparison to the initial state, and thus the deformation of its adjacent finite elements (Fig. 3).

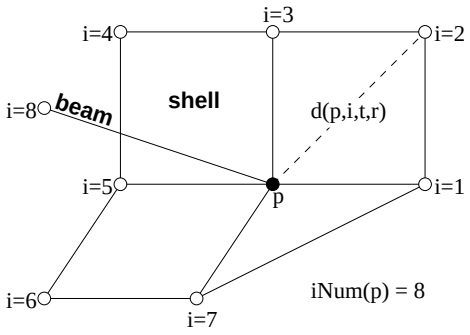


Fig. 3. $d(p,i,t,r)$ is computed as the Euclidean distance (see Eq. (1)) of node p to its *neighbor node* i in state t of the r'th simulation run.

The mesh deformation $\mathrm{DNM}(p,t,r)$ around node p having $\mathrm{iNum}(p)$ neighbors is calculated as the averaged sum of all distance differences $d(p,i,t,r)$ to their initial distances $d(p,i,0,r)$ which are the same for each simulation run r (Fig. 3):

$$d(p,i,t,r) = \left\| \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - \begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} \right\|_{t,r} \quad , \quad \begin{array}{l} t : \text{time step index} \\ r : \text{simulation run index} \end{array} \tag{1}$$

$$\mathrm{DNM}(p,t,r) = \frac{1}{N} \sum_{i=1}^{N} \left| d(p,i,t,r) - d(p,i,0,r) \right| \quad , \quad N := \mathrm{iNum}(p) \tag{2}$$

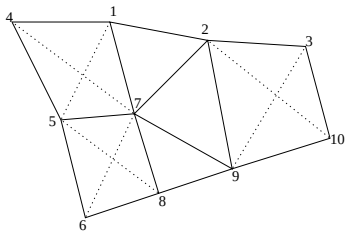
The deformation $\mathrm{DNM}(p,t,r)$ is calculated for each node p in each time step t and over all simulation runs r. This scalar quantity is only influenced by the local neighborhood which contrasts to the global measurement functions. Now, the expected value of the deformation

$$\mathrm{DNMAV}(p,t) = \frac{1}{R} \sum_{r=1}^{R} \mathrm{DNM}(p,t,r) \quad , \quad R : \text{\# simulation runs} \tag{3}$$

over all R simulation runs is determined and its standard deviation is evaluated as a measurement for the local instability of the simulation around node p.

2.3 Efficient calculation

Since large data sets consisting of about half a million finite elements and nodes have to be handled, it is impossible to store all the data in main memory. For an efficient calculation of $\mathrm{DNMAV}(p,t)$ we generate a table which represents the neighborhood of each node. This table holds entries of index pairs. One points to the neighbor node and the other one to the corresponding field in the node distance array. The table structure makes sure that the distance of each node pair is computed just once and it provides quick access to the pre-calculated node distances $d(p,i,t,r)$ of the current and the initial time step.



Node	Node/Index pair list
1	**(2,0) (4,1) (5,2) (7,3)**
2	(1,0) **(3,4) (7,5) (9,6) (10,7)**
3	(2,4) **(9,8) (10,9)**
4	(1,1) **(5,10) (7,11)**
5	(1,2) (4,10) **(6,12) (7,13) (8,14)**
6	(5,12) **(7,15) (8,16)**
7	(1,3) (2,5) (4,11) (5,13) (6,15) **(8,17) (9,18)**
8	(5,14) (6,16) (7,17) **(9,19)**
9	(2,6) (3,8) (7,18) (8,19) **(10,20)**
10	(2,7) (3,9) (9,20)

Fig. 4. This example outlines the structure of the node/index pair table which contains a list (row) for each node. The list stores pairs of an adjacent node and an index to an array where the corresponding node distance is stored.

First of all the node/index pair table (Fig. 4) is initialized evaluating the mesh connectivity which is constant over all states. Then the node distances for the initial state are computed and stored in an array which is used for all simulation runs. Starting with the first simulation run the node distances of the current state are calculated and the difference to the corresponding pre-calculated distance of the initial state is stored in a second array. Each time the table is traversed from bottom to top and the lists in the rows are traversed from tail to head as long as the node index of the entry is larger than the node index of the current row (bold entries in Fig. 4). Now, $\mathrm{DNM}(p,t,r)$ is the sum of each referenced value in row p divided by the number of entries in the row. A third array holds the accumulated sum of $\mathrm{DNM}(p,t,r)$ in order to get the expected value $\mathrm{DNMAV}(p,t)$ at the end of all runs.

After all states of one simulation run have been processed the $\mathrm{DNM}(p,t,r)$ is temporarily written to disk. After we have generated this file for each simulation run and divided the values in the third array by the number of simulation runs, the values are read back in and the standard deviation of the local deformation can be computed and stored to disk as a measure for instability. Later on the instability can be mapped onto the geometry of one simulation run using the technique described in the next section.

Furthermore, the span between the minimum and maximum deformation is of interest. Hence, for each state t and each node p the extreme values of $\mathrm{DNM}(p,t,r)$ are stored together with the index r so that the most different simulation runs can be determined later on.

3 Visualization using index texture maps

The advantage of mapping scalar data as colors directly onto geometry is that the data is visualized where it appears and thus the causal relationship between geometry and mapped data is more comprehensible. In the field of CAE flat shading can be used for element-based data visualization. As the data is node-based in the majority of cases Gouraud shading will not lead to meaningful images because the colors are assigned to vertices and interpolated in RGB color space inside the polygon during rasterization. Instead the visualization could be enhanced by adding geometry and assigning appropriate colors to the subdividing vertices, but that will increase the load of the graphics pipeline.

The best way to visualize node-based data is the utilization of a one-dimensional texture which is defined as color band. Each vertex is combined with a texture coordinate. During rasterization the texture coordinate is evaluated at every pixel and then the color is looked up in the texture. Hence, high deviation of mapped values inside the same polygon will result in color-bands without the need of additional geometry [12].

In complex models with many occluding parts in the scene it is difficult to spot regions with critical values. This problem can be solved by using a four channel texture map. The additional alpha channel provides the opportunity to restrict the data mapping or the geometry rendering depending on the texture environment setting in the context of OpenGL [11]. If GL_DECAL is used, the resulting color is composed as $C_{out} = (1 - \alpha_{tex})C_{frag} + \alpha_{tex}C_{tex}$ while the transparency is not modified by the texture ($\alpha_{out} = \alpha_{frag}$). Provided we set the α component of each texel either to 0.0 or 1.0, the data visualization is only visible for those values, where the corresponding texel has an α component of 1.0. Otherwise the geometry is rendered in the original color.

If we switch the texture environment to GL_REPLACE and enable the alpha test the geometry rendering is controlled by the mapped values. In [10] boolean textures were already used to clip geometry during the rasterization stage. We use this clipping functionality of the texture subsystem in correlation with the values simulated at the geometry. While the texture defines the outgoing color, the relation of the texel's α component to the alpha test reference value decides, if the fragment is rendered or not. Thus, this technique can be used to restrict the geometry rendering to interesting data value ranges as already associated with, for example, the visualization of potential flanges [4]. The alpha test has to be enabled to avoid z-buffer pollution; otherwise the invisible geometry could hide other geometry which lies behind the transparent parts and therefore will fail the z-buffer test.

For visual data exploration and analysis the interactive modification of the mapping has turned out as very useful. It allows the engineers to interactively restrict the color mapping or the geometry rendering to the regions of interesting values. In order to provide high interactivity the texture map does not contain RGBα quadruples but indices. These indices are used to reference the color and transparency of the texel in a hardware-supported texture color lookup table. The contents of this table represents a transfer function which can be modified in a color editor dialog.

For the investigation of instability this technique allows an interactive search for regions where different crash behavior originates. First the standard deviation of the element deformation as described in section 2.3 is loaded from disk and mapped to the indices of the texture color lookup table. By switching to the GL_REPLACE/*alpha test* mode and adjusting the alpha transfer function the engineer can hide all geometry that behaves constant or shows only a small standard deviation across all simulation runs. Then the user can zoom into a remaining area, lock the camera to the geometry, and analyze the reason for the instable performance in several simulation runs activating the time animation. A semi-transparent rendering instead of hiding that geometry with low deviation may help to orientate oneself in a complex model (Fig. 5).

The described visualization methods have been integrated into *crashViewer*, a prototype for pre- and post-processing functionality [8, 9] in the area of crash-worthiness simulations using the PAM-CRASH code [5]. The application has been developed in co-

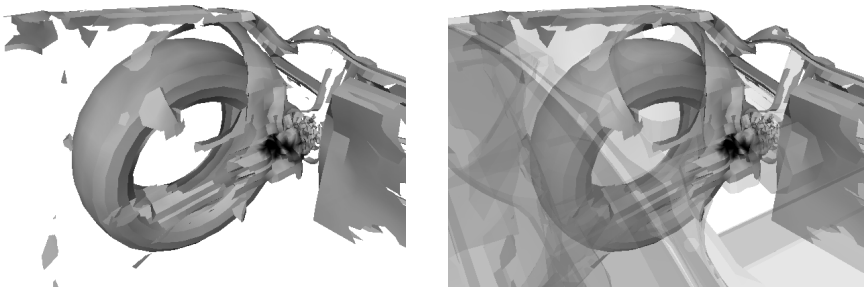


Fig. 5. To detect regions of primary instability it turned out to be very useful if the transfer function of the alpha channel is set lower than the alpha test reference value for values of small deviation. For a better orientation the user can interactively fade in the neighborhood as shown in the right image. (see also color plate in Appendix)

operation with the BMW Group and is in productive use. It uses OpenGL Optimizer [7], a tool set for large model visualization which is based on Cosmo3D [6], a scene graph layer on top of OpenGL.

4 Comparing geometry using synchronized viewers

If the most differing simulation runs have been determined the engineer could get an impression of the real deformation deviation only if it is possible to visually compare both finite element meshes in detail. A CORBA connection layer which has originally been implemented to support collaboration of two or more distant engineers evaluating simulation results [3] can be used for this task to synchronize multiple viewers on the same display.

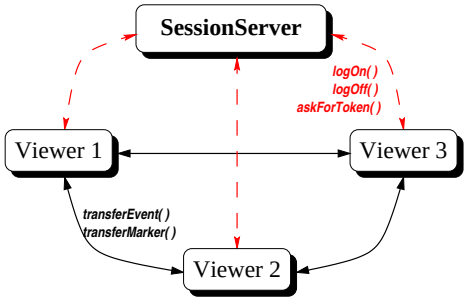


Fig. 6. The session server acts as a controler of a multi-viewer session and controls the master token. The viewer which currently holds the token sends event messages directly to other participating viewers.

Therefore a small control application (*SessionServer*, Fig. 6) is started which links the participating viewer instances together and assigns the master token to them. After the session server has been started, it stores a CORBA reference to disk. Using this reference a *crashViewer* instance can register itself to the session. The registration is propagated to the other viewers by the session server. Any event message will be transmitted from the master viewer (which is the one that currently holds the token) to the slave viewers directly without involvement of the session server. Each slave viewer can claim for the token by sending a message to the session server. After the current master has released the token, it will be transfered to the next claiming slave.

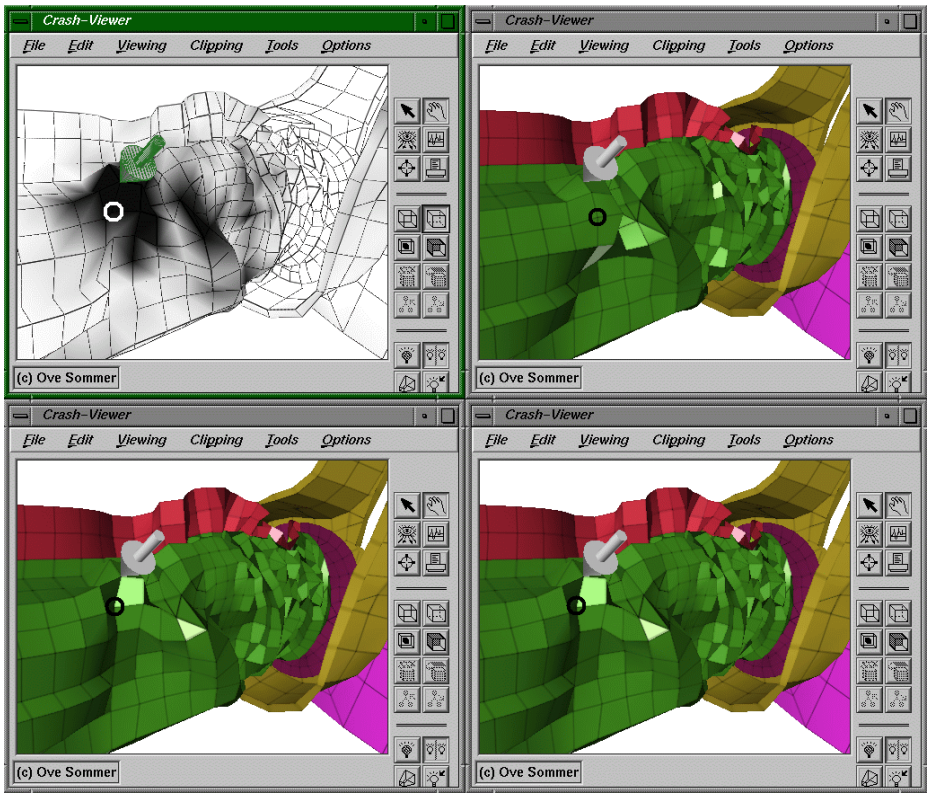


Fig. 7. Our prototype *crashViewer* is started four times and shows different simulation results of the same input deck. The camera movement is synchronized using a CORBA connection. The 3D arrow marks the same global coordinate while the circle tags the same finite element mesh node. The mapped standard deviation in the upper left window points out the different deformation behavior.

The camera position is sent by the master as a transformation matrix. Furthermore, markers can be inserted into the scene to define reference points. Fig. 7 shows four

different simulation results. The upper left viewer visualizes the standard deviation of all results. The 3D arrow marks the same global coordinate in each viewer and points out the geometric difference between the simulation runs. The upper right window shows a completely different deformation around the circled node which marks the same mesh node in each window.

Of course this functionality can also be used to compare the crash performance of variants when the reasons for instability have been removed. This would enhance the car development process significantly because the differences between constructive variants and their effects to the whole model regarding crash dynamics would directly be visible.

5 Results

The calculation of a measure for the instability of crash-worthiness simulations is construed to be time and memory efficient. The test data set that can be shown here contains about 60.000 shell elements and nearly 55.000 nodes. Each of the 15 result files store 81 simulated time steps of the same source model. On a SGI Octane with one R12k/300MHz CPU the standard deviation of the local deformation as described in section 2.2 is computed for all result files in 4.5 minutes. The process needs about 45MB main memory. The memory consumption depends on the number of nodes and the number of states but it is independant of the number of simulation runs. Simulation results with 500.000 nodes over 60 time steps should require less than 500MB. For larger models or result files with more time steps it is possible to make the memory consumption also independant of the number of states which would lower the performance.

With the described methods integrated our prototype *crashViewer* allows for the first time the comparative visualization of instability in crash-worthiness simulations. The interactive modification of transfer functions used by the index texture map provides value-based geometry clipping. The engineer is visually guided to regions in the finite element model where different crash behavior originates. The detailed investigation of such areas is supported by several functions like the camera locking mechanism. (See Appendix for additional color plate.)

6 Conclusion

We introduced a method to determine and visualize the instability across multiple crash-worthiness simulations of the same source model. The integration of the presented techniques into our prototype *crashViewer*, allows the engineers of the crash simulation department to explore the origins of instability. Finally, the use of multiple synchronized viewers displaying different simulation results makes a direct comparison possible. Only the combination of advanced rendering techniques and exploiting graphics hardware allows an innovative visualization application which is in productive use at BMW.

7 Acknowledgements

We thank the Institute for Algorithms and Scientific Computing (SCAI) of the German National Research Center for Information Technology and the crash department of the BMW Group for providing the simulation results. This work was partially funded by the Bundesministerium für Bildung und Forschung in the context of the Autobench project.

References

1. Autobench – An Integrated Construction Environment for Virtual Prototypes in Automotive Industry. http://www.autobench.de, 1998–2001.
2. Jürgen Bendisch and Hartmut von Trotha. Stabilitätsuntersuchungen mit Mitteln der Statistik. Internal report of GMD, Autobench project, April 2000.
3. Klaus Engel, Ove Sommer, and Thomas Ertl. A Framework for Interactive Hardware Accelerated Remote 3D-Visualization. In *Proc. of EG/IEEE TCVG Symposium on Visualization VisSym 2000*, pages 167–177,291. Springer Wien/New York, May 2000.
4. Norbert Frisch, Dirc Rose, Ove Sommer, and Thomas Ertl. Pre-processing of Car Geometry Data for Crash Simulation and Visualization. In Vaclav Skala, editor, *WSCG 2001 - The Ninth International Conference in Central Europe on Computer Graphics and Visualization*, pages 25–32, February 2001.
5. E. Haug, A. Dagba, J. Clinckemaillie, F. Aberlenc, A. Pickett, R. Hoffman, and D. Ulrich. Industrial Crash Simulations using the PAM-CRASH code. In *Supercomputing in Engineering Structures*, pages 171–196. Computational Mechanics Publications, 1989.
6. Silicon Graphics Inc. Cosmo3D™ Programmer's Guide. Silicon Graphics Inc., IRIS Insight Library, 1998. http://techpubs.sgi.com/.
7. Silicon Graphics Inc. OpenGL Optimizer™ Programmer's Guide: An Open API for Large-Model Visualization. Silicon Graphics Inc., IRIS Insight Library, 1998. http://techpubs.sgi.com/.
8. Sven Kuschfeldt, Thomas Ertl, and Michael Holzner. Efficient Visualization of Physical and Structural Properties in Crash-Worthiness Simulations. In Yagel and Hagen, editors, *Proc. IEEE Visualization '97*, pages 487–490,583. IEEE Computer Society Press, October 1997. ISBN 1-58113-011-2.
9. Sven Kuschfeldt, Ove Sommer, and Thomas Ertl. Efficient Visualization of Crash-Worthiness Simulations. *IEEE Computer Graphics and Applications*, 18(4):60–65, July/August 1998.
10. William E. Lorensen. Geometric Clipping Using Boolean Textures. In *Proceedings Visualization '93*, pages 268–274. IEEE, 1993.
11. Ove Sommer and Thomas Ertl. Geometry and Rendering Optimizations for the Interactive Visualization of Crash-Worthiness Simultations. In *Proceedings of IT&T/SPIE Electronic Imaging, Visual Data Exploration and Analysis VII*, volume 3960, pages 124–134, January 2000.
12. Michael Teschner and Christian Henn. Texture Mapping in Technical, Scientific, and Engineering Visualization. http://www.sgi.com/chembio/resources/texture/index.html, 1995. Technical Report, Silicon Graphics Inc.