

Geometry and Rendering Optimizations for the Interactive Visualization of Crash-Worthiness Simulations

Ove Sommer and Thomas Ertl

University of Stuttgart, IfI, Visualization and Interactive Systems Group
Breitwiesenstr. 20-22, 70565 Stuttgart, Germany
<http://wwwvis.informatik.uni-stuttgart.de/>

ABSTRACT

Today's car body models used for crash-worthiness simulations consist of around half a million finite elements. The interactive visualization of these large scale time-dependent geometries with constant topology on workstation platforms requires a variety of modeling and rendering optimizations. We present a memory efficient scene graph design, an algorithm to concatenate the mainly four-sided elements into optimal quadrilateral strips, and a simplifier which generates an approximating triangle mesh by using the one-sided Hausdorff distance as an error measure. Furthermore we describe a technique to explore scalar data mapped onto complex scenes by hiding geometry with values outside an interactively specified range of interest. These optimizations allow for the first time interactive visualization of a full car crash on medium range graphics workstations. They are embedded in an application which is based on *Cosmo3D / OpenGL Optimizer*. It was developed in close cooperation with the BMW Group and it is in productive use.

Keywords: scene graph design, quadrilateral stripping, texture mapping, mesh simplification, bounding box hierarchy, distance mapping, flange visualization, automotive, crash-worthiness simulation

1. INTRODUCTION

In crash-worthiness simulations the amount of data is still rising. The reasons for that are the ability of the applications, which can simulate and track more and more physical attributes, and the requirement of the engineers to build the model as realistic as possible. For example, a car model nowadays consists of 250.000 to 500.000 mainly four-sided finite elements and even more nodes. The size of the model depends on the development stage of the car – in later development phases of a new car the model becomes more complex. During a crash-worthiness simulation the first 120 ms after the crash are computed in 1 μ s-steps. The coordinates of the deforming mesh are saved at 2 ms intervals resulting in 60 time steps. For all the nodes and for each element several parameters can be tracked. Those simulation results are stored in a file which often contains more than 1.5 GB of data.

The graphics application programmer's interface (API) OpenGL Optimizer¹ has been announced by SGI in 1997 as the standard API of the future for large model visualization. Among other things it provides tools for the processing of CAD data, a MP harness that supports multi processing, and several algorithms to optimize the underlying Cosmo3D scene graph and the represented geometry in order to accelerate the rendering speed. Actually, Cosmo3D was meant as a temporary solution – originating from Java3D – and should have been replaced this year by the scene graph layer of the Fahrenheit project. But for a couple of months now, the Fahrenheit project is suspended and Cosmo3D/OpenGL Optimizer has been put in maintenance mode. Thus it is not clear how the future scene graph API looks like. However, the results presented here are transferable to other scene graph APIs with similar characteristics.

Models as large as the described ones not only require an efficient scene graph design but also some sort of preparation in order to optimize the data for the following processing by the graphics subsystem. Furthermore, the complexity of the visualized data enforces the development of new techniques which enable the engineer to manage the investigation of simulation results. This paper describes an efficient scene graph design, two techniques to obtain higher frame rates by reducing the number of vertices, and a mechanism that utilizes texture maps combined with the alpha test to restrict the view to dedicated regions of interest.

Further author information:

E-mail: Ove.Sommer@informatik.uni-stuttgart.de and Thomas.Ertl@informatik.uni-stuttgart.de

2. EFFICIENT SCENE GRAPH DESIGN

Several graphics APIs, such as Iris Performer or OpenGL Optimizer, have been developed to take advantage of recent progress in compute server and workstation architecture with multiprocessing hardware in mind. Since those APIs are usually scene graph based, the car model can be optimized in a pre-processing step during scene graph creation. While traversing the scene graph, various culling mechanism can be used by multiple processes to enhance frame and interaction rates. The huge amount of time-dependent geometry with constant topology, which represents the FE model, requires an efficient scene graph design in order to handle the complex data interdependencies and to achieve high rendering speed.

To visualize such large meshes over all time steps on a midrange graphic workstation it is not possible to store all the information including the mesh topology for each time step separately. We developed a scene graph design where the topology is stored only once and shared for each time step.² Our design shares vertex coordinates and normals for adjacent polygons to render Gouraud shaded meshes (Fig. 1).

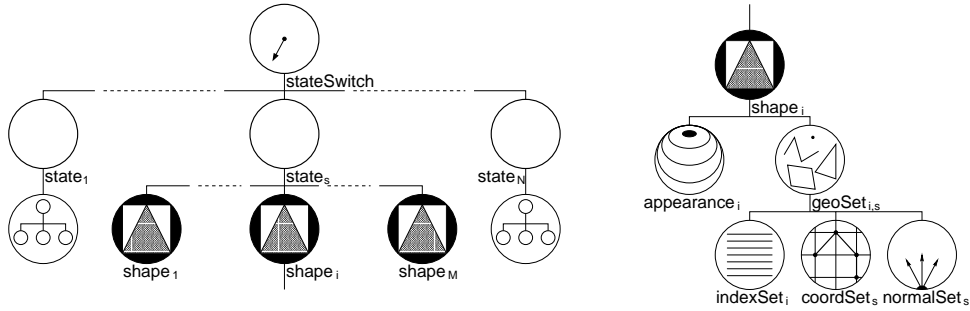
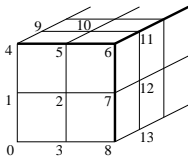


Figure 1. This Cosmo3D scene graph design reduces redundant data storage by taking advantage of indexed geometries and by sharing instances of scene graph nodes. The `stateSwitch` node switches between simulated time steps. `groupOfStates` is the root node of the subgraph representing the entire geometry of time step s . The car model consists of up to 1000 components – each one is described in a `shape` node which holds a reference to the component appearance and geometry. `geoSeti,s` refers to the corresponding `indexSeti`, representing the topology of car component _{i} ; and to `coordSets` and `normalSets` storing all the coordinate and normal vectors of `states`.

In order to determine the vertices with multiple normals an edge detection algorithm is applied to the meshes of the first and the last time step. The combined results of those two time steps specify the set of Gouraud edges for all time steps. For each vertex v_j the number N_j of different normals is known, the normals $n_{v_j,1} \dots n_{v_j,N_j}$ are appended to `csNormalSet`, and the coordinates are stored N_j times into `csCoordSet` as outlined in Fig. 2. Finally, we initialize the `csIndexSet` to represent the time-invariant mesh topology. Four consecutive indices refer to the coordinates and normals of a quadrilateral. Three-sided shell elements are rendered as degenerated quadrilaterals where the last index is used twice.



<code>csIndexSet</code>	[0,3,2,1]	[1,2,6,4]	[2,11,8,6]	[3,13,11,2]	[5,7,16,15]	...
index:	0 1 2 3 4	5 6 7 8	9 10 11 12	13 14 15	16 17 18 19	20 ...
<code>csCoordSet</code>	0, 1, 2, 3, 4, 4, 5, 5, 6, 6, 6, 7, 7, 8, 8, 9, 10, 11, 11, 12, 13, ...					
<code>csNormalSet</code>	0, 1, 2, 3, 4 ^a , 4 ^b , 5 ^a , 5 ^b , 6 ^a , 6 ^b , 6 ^c , 7 ^a , 7 ^b , 8 ^a , 8 ^b , 9, 10, 11 ^a , 11 ^b , 12, 13, ...					

Figure 2. This example outlines how the `csIndexSet` can be shared to reference both coordinates and normals. The Gouraud edge is marked as the thick line which means that the vertices 4, 5, 7, 8, and 11 correspond to two different normal vectors, vertex 6 even corresponds to three different ones. Therefore, the coordinates for those vertices are duplicated in the `csCoordSet` for each varying normal. Duplicating vertex coordinates lying on Gouraud edges is more memory efficient than referencing coordinates and normals by separated `csIndexSets`.

3. RENDERING ACCELERATION BY MESH OPTIMIZATION

The rendering speed is limited by the slowest per-frame operation. Such per-frame operations are the

- scene graph traversal which is done by the CPU. During this stage the represented data will be generated and prepared for the following steps in the graphics pipeline.
- transformations. This stage is limited by the rate at which the graphics hardware can process vertices. Therefore, those limits can be influenced by the data only.
- rasterization which is limited by the rate at which the graphics hardware can update the frame buffer. Here, one deciding factor is the size of the viewport.

Thus, for large polygonal meshes we have to alleviate the bottleneck resulting from the transform stage. For example, for a single lighting source, the transform stage for one vertex takes approximately 100 floating-point operations.¹

In the following sections we discuss two alternatives how to reduce the number of vertices that has to be sent to the graphics subsystem in order to speed-up the rendering: First, we present a quadrilateral stripper that eliminates data redundancies by concatenating adjacent mesh elements. Thereafter, we describe how to implement a simplification algorithm that uses the one-sided Hausdorff distance to modify the mesh topology iteratively and how to combine the results in a memory efficient way.

3.1. STRIPPING OF QUADRILATERALS

One way to speed-up the rendering of polygonal scenes by reducing the number of vertices that have to be processed by the graphics hardware is the concatenation of adjacent primitives. For triangle meshes several approaches have been implemented generating triangle strips.^{1,3,4} A triangle mesh represented by an optimal strip is defined by $n + 2$ vertices instead of $3n$ vertices for n individually rendered triangles.

But in the area of crash-worthiness simulations using finite element models the engineers demand the visualization of the model discretization which is mainly given by quadrilaterals. Though Kuschfeldt et al.⁵ presented a method how 2D texture maps can be utilized for discretization of four-sided elements rendered as triangle strips, this technique does not work in wireframe mode. While in wireframe mode the mesh is visualized as it is built; thus, all four-sided elements are broken in two triangles.

In order to retain the element discretization also in wireframe mode and to reduce the number of vertices that have to be transformed we developed an OpenGL Optimizer based quadrilateral stripping module. Our approach is based on the techniques presented by Evans et al.⁴ Their algorithm processes arbitrary polygonal meshes which do not need to be pre-triangulated. They describe several strategies concerning the

- triangulation of more than three-sided polygons
They distinguish the static triangulation as a pre-processing step from the more flexible dynamic triangulation which is done during the strip generation phase.
- tie-breaking methods in order to continue strips
Five different approaches are described how to proceed if two or more neighbors of the encountered triangle have the same lowest degree, which is the number of remaining neighbors that are still not connected in strips.
- determination and processing of rectangular 'patches' of quadrilaterals
In a global step, which is done before the local algorithms are applied, the mesh will be searched for strips of quadrilaterals in two directions. Afterwards, the discovered regions will be stripped by one 'full-patch strip' or by a set of 'row or column strips'.

We followed their ideas regarding the two-pass stripping since the car models in crash-worthiness simulations mainly consist of four-sided finite elements. But in contrast to the generation of triangle strips where adding a swap (by duplicating the last but one vertex and therefore building a degenerated triangle) in order to continue the current strip is cheaper than starting a new one, this practice makes no sense for quadrilateral strips. A swap inside a quadrilateral strip costs four additional vertices as shown in Fig. 3. This means that starting a new quadrilateral strip is cheaper than extending it by swaps. Therefore, in the global phase we do not look for 'patches' of quadrilaterals

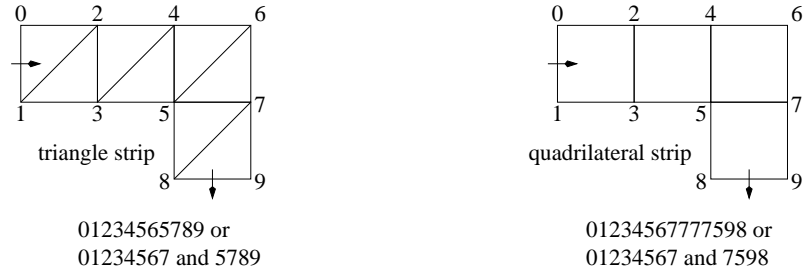


Figure 3. This simple mesh cannot be rendered with a single strip without any swaps. While the triangle strip on the left can be continued by adding one swap (5,6,5) – which is cheaper than starting a new triangle strip – the quadrilateral strip on the right should be rendered as two separate strips because here the swap is more expensive than a new start: the same vertex has to be stored four times in order to change the strip direction.

but bands of maximal length. Those bands have to be at least as long as a specified 'cutoff size'. After determining the longest band all its quadrilaterals will be removed from the set of primitives that have to be connected.

We extended Cosmo3D by a new scene graph node called `csQuadStripSet`. The quadrilateral stripping module was implemented by sub-classing from `opDFTravAction`. On a call to its method `apply()` it traverses the scene graph starting at the given node. Each `csQuadSet` is converted as described and replaced by the resulting `csQuadStripSet`. Figure 4 shows how the bandification strategy works – for the displayed car component it results in 54 vertices less than the row-column-patchification.

```

Mark all quadrilaterals as 'active'
while (found any quad-strip of length >= cutoff) {
  Mark all 'active' quadrilaterals as 'available'
  Empty listOfMaximalStrips
  for (all quadrilaterals marked as 'active' AND 'available') {
    curMaxStrip := maximal strip starting from quadrilateral i
    if (curMaxStrip.length() >= cutoff) {
      Mark all quadrilaterals of curMaxStrip as 'unavailable'
      Insert curMaxStrip in sorted listOfMaximalStrips
    }
    else
      Mark quadrilateral i as 'inactive'
  }
  while (listOfMaximalStrips contains any strip) {
    curMaxStrip := longest strip of listOfMaximalStrips
    if (curMaxStrip.length() is still unchanged)
      Mark all quadrilaterals of curMaxStrip as 'inactive'
    else
      break
  }
}

```

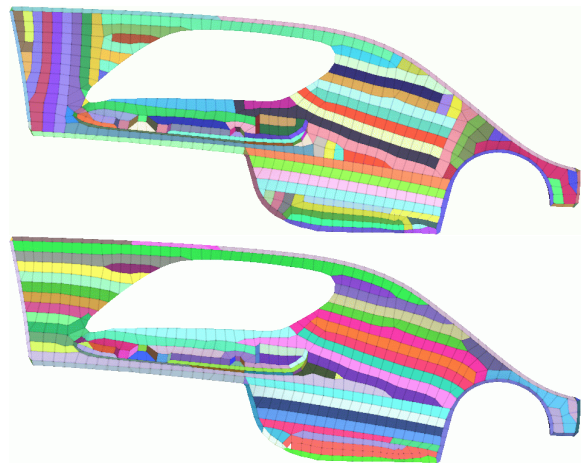


Figure 4. The pseudo code on the left shows the bandification algorithm. The images on the right outline the difference between the row-column-patchification (top) and the bandification (bottom) strategy applied for quadrilateral stripping. Using the bandification method results in less strips – thus less vertices have to be processed by the graphics pipeline.

Table 1 contains the results for the quadrilateral stripping module compared with those of the `opTriStripper`. Although the quad-stripper is not as flexible as a tri-stripper could be, the reduction-rate of those vertices that have to be processed by the graphics subsystem is about 10% better than that of `opTriStripper` and the mean strip length measured in triangles per strip is about twice as long.

original mesh		opTriStripper			QuadStripper		
#quads	#verts	#tri-strips	#verts	length	#quad-strips	#verts	length
2215	8860	560	5657 (64%)	8.1	231	4848 (55%)	19.0
26284	105136	11708	78370 (75%)	4.7	4085	60365 (57%)	12.8
51473	205892	13138	131450 (64%)	8.0	5806	113308 (55%)	17.5
104749	418996	36203	283374 (68%)	5.8	19667	244244 (58%)	10.4
255641	1022564	74590	678418 (66%)	7.1	35298	575528 (56%)	14.3

Table 1. The three main columns separate the results for the unstriped original quadrilateral mesh, those of opTriStripper, and those of our quadrilateral stripping module. Each row contains the results for another data set. '#verts' are the number of vertices that have to be processed by the graphics hardware. 'length' is the mean number of triangles per strip.

3.2. MESH SIMPLIFICATION

In order to provide a user-friendly steering of the camera during interaction (e.g. while flying through the model) it is important to achieve even higher frame-rates than 10 frames per second. Therefore, for large models it is not enough just concatenating adjacent primitives. The complex mesh has to be simplified first in order to get rid of the larger part of the vertices.

OpenGL Optimizer offers two different simplification tools, however, their focus is on the generation of Level of Detail representations of the input geometry. Since we want to use the simplified variant irrespective of the camera distance to the geometry, the simplifier needs to generate meshes with a minimal number of triangles which do not differ more than a specified tolerance from the original mesh.

The OpenGL Optimizer class opSRASimplify provides methods to triangulate polygonal meshes and to decimate those triangle meshes. However, one of the main drawbacks when using this method is the setting of input parameters. The decimation proceeding is influenced by three weights which determine what to preserve most: the volume, sharp edges, or smooth curvature of the geometry. Furthermore, the simplification goal has to be specified: the percentage of the remaining vertices of the model. Usually, a car body model consists of a large variety of different geometries – some with large smooth regions and others having a lot of sharp edges. Therefore it is very difficult to specify such a quadruple of parameters in order to get an optimally simplified geometry for each case.

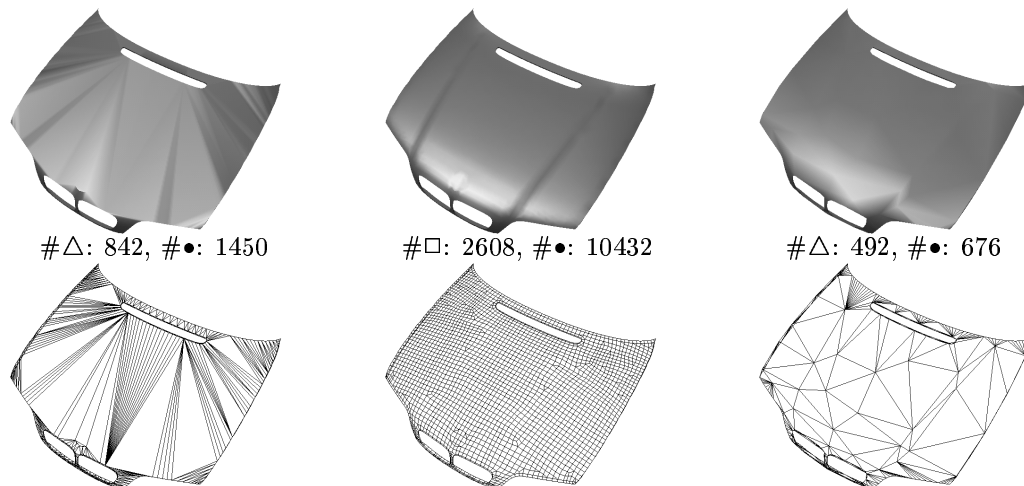


Figure 5. The top row shows the Gouraud shaded engine hood, the bottom row shows the corresponding wireframe model. The original quadrilateral mesh (middle) was simplified using the default settings of opSRASimplify (on the left) and by our triangle decimation module (on the right).

It is much more intuitive to set a criterion which considers the relative size of the geometry that has to be simplified. Hence, we decided to re-implement a variant of progressive meshes^{6,7} which employs the one-sided Hausdorff-distance as an error measure. The error tolerance is specified relative to the size of the original geometry by setting it for each car body part to 0.5% of the diameter of its bounding sphere. This allows larger deviation for large scale parts and lower deviations for small ones. The algorithm successively eliminates triangles by doing edge-collapses until the specified tolerance is exceeded. This means the coordinates remain unmodified – just the mesh topology is changed. This allows us to share the `csCoordSets` storing the vertex coordinates between the original and the simplified mesh by assigning a new `csIndexSet` which represents the simplified topology.

Figure 5 compares the results of the OpenGL Optimizer simplification tool and our implementation of Campagna’s⁷ decimation algorithm. Like for the quadrilateral stripper we extended the tool-set by a traversal action that applies the decimation method to each polygonal `csGeoSet`. For none-triangle sets the `opSRASimplify` is utilized specifying 100% as the simplification goal in order to convert for example a `csPolySet` into a `csTriSet`. The triangles resulting from our decimation tool are not as spiky as those coming out of `opSRASimplify`. Thus the lighting for the solid geometry on the right of Fig. 5 shows less artifacts than the one on the left.

4. HIERARCHICAL SUBDIVISION OF FINITE ELEMENT MODELS

For several tasks in large model visualization it is quite important that the model is geometrically substructured. For example, each geometry representing a scene graph node provided by Cosmo3D (e.g. `csGroup` or `csShape`) stores a bounding volume which contains the geometry of its subgraph. OpenGL Optimizer offers tools to evaluate this data in order to decide which geometry is visible before all the OpenGL primitives are generated and sent down the graphics pipeline. This allows the CPU to cull away large portions of the model by viewfrustum and occlusion culling, especially if the camera is inside a complex model.¹

If the model is represented by a scene graph, as we first proposed in² (see Fig. 1), each car component is held by a `csShape` node and therefore it is not substructured any further. This section discusses a bounding volume hierarchy which subdivides each car component in order to provide efficient per element calculations. We present some new visualization techniques where this hierarchy is required to allow interactive response.

We utilized the bounding volume hierarchy algorithms presented by Gottschalk et al.⁸ which actually were developed to enable real-time collision detection. Their approach compares the effectiveness of different bounding objects and introduces a fast overlap test for oriented bounding boxes. The test if two oriented boxes overlap each other is based on a ‘separating axis’ theorem. After the vertex is determined, which is the closest to the other box, the segment between this vertex and the mid point of its box will be calculated. Those segments of both boxes are successively tested for an overlap in up to 15 projections. If one projection can be found for which two segments do not overlap, the boxes are considered as non-interfering.

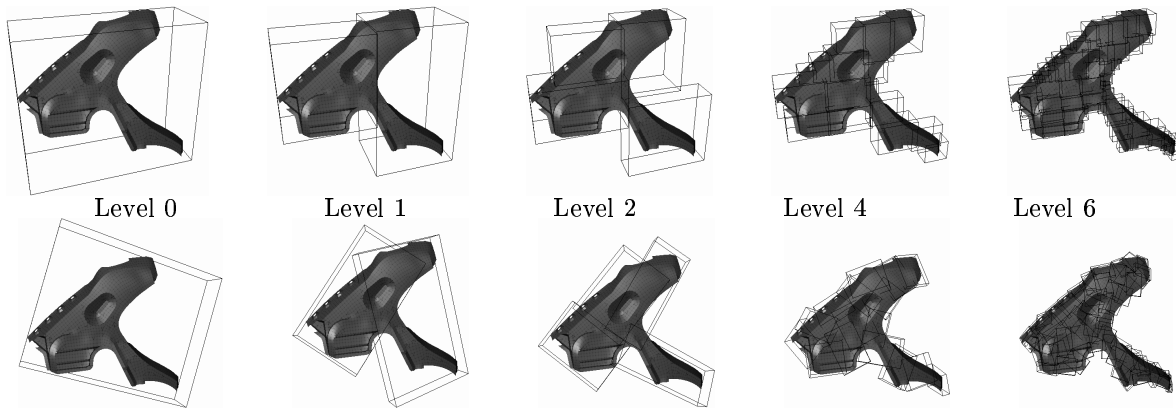


Figure 6. The first levels of an axis-aligned bounding box hierarchy (top) compared with object oriented boxes (bottom). The car component consists of 1608 shell elements.

4.1. EFFICIENT DISTANCE CALCULATION

The transformation of CAD data into a discretized finite element model includes 'initial penetrations' which are points, where one discretized surface is closer to another surface than the specified material thickness. If the simulation will be started with a model containing initial penetrations then initial forces at those points are caused. Thus, the simulation results will be influenced in an undesirable way. Hence, one goal of the pre-processing of the simulation input data deck is the controlled removal of those initial penetrations.

In order to detect those vertices which are positioned too close to an element of another car component, actually the minimal distance of each vertex to each finite element has to be calculated. This task can only be efficiently solved if any hierarchical substructuring is used.

In the initialization phase we specify the maximal distance of interest which should be at least as thick as the maximal car component thickness. This value is stored as the current minimal distance. During the test of one vertex with another sub-mesh first the distance between the vertex and the bounding volume is computed. Only if it is smaller than the currently stored minimal distance, the children of the bounding object will be tested next. A child can be a set of more bounding volume instances or one or more finite elements if the bounding object was a leaf node in the hierarchy.

During the distance calculation this approach eliminates nearly all car components except the direct neighbors at the top level of the bounding volume hierarchy. Just a small number of tests are applied until the point is tested on a per element basis. There the minimal distance is calculated by the slightly modified algorithm proposed by Campagna⁹ which considers each projection case and computes values only if they are needed for that particular case. For example, if the leaf nodes of the bounding volume hierarchy contain more than 5 elements the exact distance computation can be avoided in about 80% by first checking if the point-to-plane distance is less than the current minimal distance. The results – the minimal distance to any other sub-mesh – is stored at each vertex and can be used for distance visualization as presented in section 5.

1 elem/BV	# BVs	# point/BV	# point/elem	time
AABB	419,049	17,152,432	25,483	6.3 + 13.93
OBB		14,270,338	5,386	11.0 + 35.64
≤10 elem/BV	# BVs	# point/BV	# point/elem	time
AABB	57,443	16,226,990	2,642,270	3.2 + 14.52
OBB		13,603,798	2,140,013	4.9 + 32.58

Table 2. The upper part shows the results of a hierarchy where each leaf node contains just one element, for the lower part up to ten elements are encapsulated. The number of generated bounding volumes is the same regardless of the bounding object type. The last three columns show the number of applied tests between the current mesh vertex and a bounding box or an element respectively and the time in seconds needed for hierarchy generation and testing.

The adopted algorithms were tested on a SGI Octane with a R12K/300 MHz CPU inside. The model consisted of 627 car components, 209,838 four-sided elements, 197,861 vertices, and the maximum distance of interest was set to 1.5 mm. The results of the performance-test (Tab. 2) shows that for distance-to-point computation using bounding volume hierarchies of axis-aligned bounding boxes is twice as fast as those of object-oriented bounding boxes, although there are fewer tests applied because of the better fitting OOBs (Fig. 6). The reason is the more expensive interference calculation for OOBs. Furthermore, it can be seen that the costs for point-to-element distance calculation is relatively small: OOBs perform better if there is more than one element encapsulated by a leaf node. However, this does not scale to higher values.

In the assembly stage, when the car model is composed from the finite element data sets of the independently meshed car body parts, the engineer has to take care of the correct connectivity definition for adjacent sub-meshes. Furthermore, the contact types have to be specified, otherwise car body parts would interpenetrate each other. For a complex model it is very difficult to keep all those definitions in mind. In order to completely check, if the model was set up correctly, the engineers take connectivity matrices as an aid where each pair of adjacent sub-meshes is

recorded. Here, 'adjacent' means that the elements are closer to each other than a specified distance. Using the described method, those connectivity matrices can be efficiently computed.

4.2. EFFICIENT FORCE-TUBE COMPUTATION

One important result of a front-crash simulation is the information how much force is transmitted by the longitudinal structures and how much is absorbed by those car body parts. We already presented the force flux visualization with force tubes where a tubular element is used to visualize section forces by varying its radius and color.²

Now, we extended this approach to allow the engineer to interactively define a trace line and to select those finite elements which should be taken in account for the force tube calculation. Furthermore, we reimplemented the intersection algorithm in order to take advantage of the bounding volume hierarchy for the decision if an element is contributing or not.

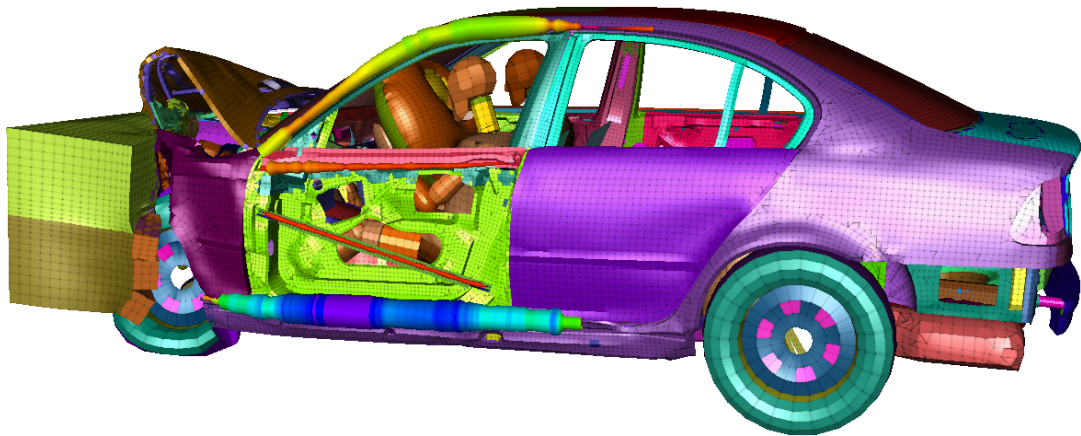


Figure 7. The force flux is visualized by the radius and the color* of those tubular objects, called force tubes. In order to calculate the accumulated section force for one section plane that is positioned orthogonally to the previously specified trace line, the intersected elements have to be determined. This computation is efficiently enabled by the utilization of bounding volume hierarchies.

First, the engineer selects two or more control points which define the trace line. The control points are connected to the longitudinal structure, thus the force tube performs the same deformation. Each segment contains a set of section planes which are equally distributed between the starting and the ending control point orthogonal to the segment. The normals of a small number of section planes between two segments are interpolated to generate smooth transitions and to avoid artifacts. To suppress large varying distances between the car body part and the corresponding trace line the control points should be defined at major inflexion points.

After specifying the trace line the engineer can restrict the set of elements that are considered during the section force calculation. On one hand this can be done by pre-selecting car body parts. But for several situations it turned out that the restriction inside a sub-mesh (e.g. a side frame) is essential to evaluate the force flux. Thus we implemented a selection tube whose mid line is defined by the previously specified trace line. Its radius and position can be interactively modified. We utilize the RGBz image technique of OpenGL Optimizer in order to guarantee high frame rates. Before we insert the selection tube into the scene graph the car model has been rendered into a RGBz image. During modification of the selection tube for each frame only this RGBz image has to be copied into the frame buffer and the updated selection tube will be rendered afterwards.

The bounding volume hierarchy is used for both the determination of contributing elements and the efficient intersection calculation between the mesh and the section plane. As for the distance calculation we first traverse the bounding volume hierarchy for those bounding objects that are intersected by the current plane. The vertices of encapsulated elements are classified regarding the normal direction of the section plane. The section forces are

*Colored images can be found on our WWW server.

accumulated from those vertices on the normal side of the section plane which belong to elements referring to vertices on the opposite side.

Finally, the amount of the accumulated section force is mapped onto the radius and the color of the corresponding ring. The topmost force tube in Fig. 7 consists of about 170 connected rings and was calculated in 90 seconds for 61 time steps. In order to accelerate the analyzing process those force tubes can be pre-calculated and stored by a batch tool. Thus, the engineer does not have to wait for the calculation of standardized load cases.

5. SELECTIVELY HIDING GEOMETRY USING RGB_α TEXTURE MAPS

The main problem in post-processing of crash-worthiness simulations is the analysis of tracked data. The aim of visualization applications is to aid the user in understanding the data, for example, by mapping scalar values as colors directly onto the underlying geometry. But for particular tasks the models are too complex to spot all the regions with critical values at once. This means that an application should provide a mechanism to restrict the visualization for the 'interesting' areas.

One way would be the use of clipping planes which can be interactively positioned inside the model. But this mechanism hides the geometry regardless of its corresponding values. We wanted the geometry rendering to depend on the mapped variables. Therefore, we employ α -texture maps and the alpha test to influence the visibility of geometry in correspondence to the mapped parameters. This technique is used in several scenarios:

- **Visualization of oscillating parameters**

During crash-worthiness simulation parameters can be tracked which heavily deviate for adjacent elements. For example, only a few elements contain critical values and they are distributed all over the model. The only way to check, where those elements are positioned, is to hide all the geometry pertaining to values inside the tolerance range.

- **Comparison of different variants**

Due to the introduction of independently meshed car body parts it is now easier to exchange several parts by variants. The effects of replacement can only be evaluated in detail if it is possible to visualize the comparison, for example, of coordinates or parameter values of corresponding elements. Peaks of deviations are easily detectable by using the presented mapping method.

- **Flange visualization**

Another aim of this texture mapping and alpha testing technique is the visualization of potential flanges. Combined with distance visualization the engineer is able to restrict the rendering to that geometry which has a previously specified distance to another surface. (Fig. 9)

The parameters are tracked for each time step either per vertex or per element. Two examples are shown in Fig. 8. The engineer specifies a range for the chosen variable and the number of colors that should be used for visualization. The values are read or calculated and then mapped into the range $[0.0, 1.0]$. The results are used as texture coordinates into a one-dimensional texture map which represents the color scale. Regarding to memory efficient scene graph design we have to distinguish two cases:

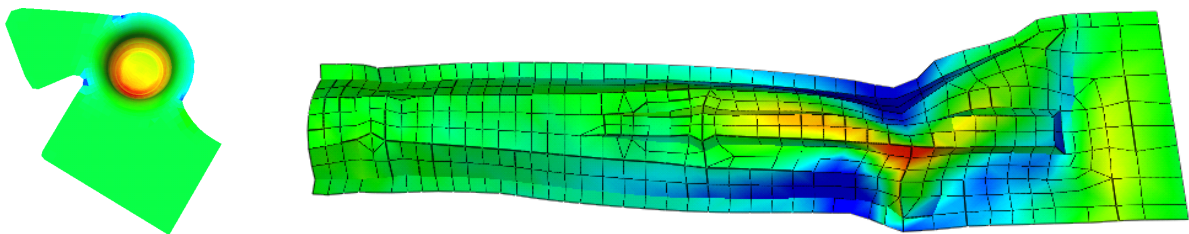


Figure 8. On the left the thinning of a car component during stamping is mapped per element. The buckling rate of a longitudinal structure during rear crash is visualized on a per vertex basis in the right image.*

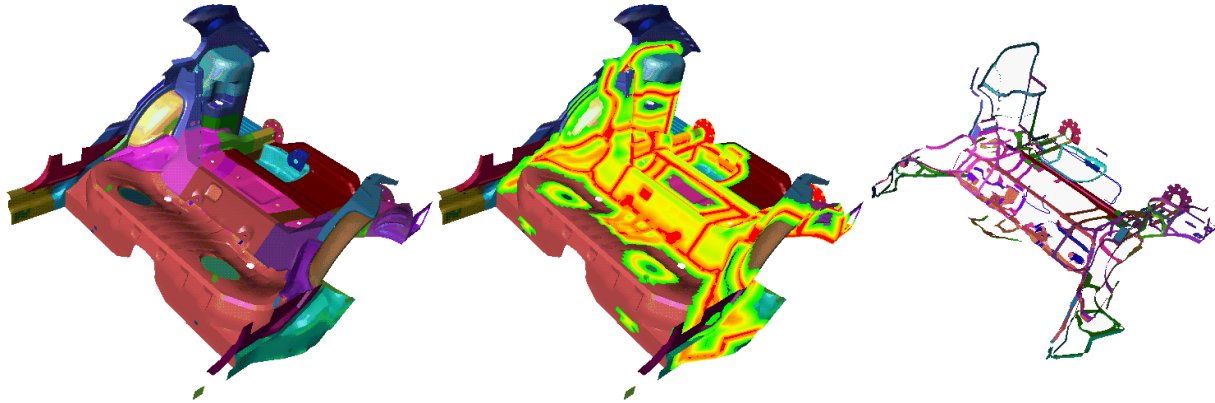


Figure 9. These images show parts of the back compartment of a car. The illustration in the middle visualizes the minimum distance from each node to the closest surface of another car body part up to 50 mm. On the right the same values are mapped to hide all geometry where this distance is more than 2mm using the texture subsystem and the alpha test. The rendered geometry show potential flanges.

- Vertex based parameters can be visualized by storing the mapped texture coordinates and duplicating them for those vertices which lie on a Gouraud edge like we did for the `csCoordSet` (Fig. 2). Here, the `csIndexSet` referring the texture coordinates can be shared with normals and coordinates.
- Element based variables require a separate `csIndexSet`. Its length is four times the number of elements and it contains quadruples each of four times the same index. In the element based case the `csTexCoordSet` stores just one value per element.

The advantage of texture mapping for vertex based scalar visualization as opposed to vertex coloring is the rendering of contour lines crossing the elements. If we additionally utilize the alpha channel by applying a four channel texture map to the geometry employing the texture environment `GL_DECAL` we are able to restrict the parameter visualization to those areas of interesting values. For example, if initial penetrations should be visualized, the colored texture representing the minimal vertex distance to an adjacent car body part is visible only for a small tolerance range. For larger distance values the texture map is totally transparent, $\alpha = 0$ (see also Fig. 9 mid).

Changing the texture environment to `GL_MODULATE` combined with the alpha test enabled allows the restriction of geometry rendering. Then the α -component of the texture map influences the appearance of the fragment: $\alpha_{out} = \alpha_{frag} \cdot \alpha_{tex}$. Those fragments assigned to an α -value below the alpha test reference value will not be rendered.

In order to modify the restriction we apply an index texture map in combination with a texture color lookup table. The implemented texture color table editor allows for the interactive justification of the transfer function for each channel separately. The scene graph data remains unmodified and the user gets immediate response.

6. CONCLUSIONS

We introduced a scene graph design that allows to organize the results of crash-worthiness simulations effectively. Thus the visualization of a full car crash can be performed even on midrange workstations using scene graph APIs like Open Inventor or Cosmo3D. All the presented techniques are modularly implemented in just one Cosmo3D / OpenGL Optimizer based visualization application. The hierarchical substructuring of the finite element model enables the efficient computation and preparation of visualization data. Only the combination of advanced rendering techniques and exploiting graphics hardware allows an innovative visualization application which is now in productive use at BMW.

REFERENCES

1. "Opendgl optimizer™ programmer's guide: An open api for large-model visualization." Silicon Graphics Inc., IRIS Insight Library, 1998.
2. S. Kuschfeldt, O. Sommer, and T. Ertl, "Efficient visualization of crash-worthiness simulations," *IEEE Computer Graphics and Applications* 18, pp. 60–65, July/August 1998.
3. K. Akeley, P. Haeberli, and D. Burns, "tomesch.c." C Program on SGI Developer's Toolbox CD, 1990.
4. F. Evans, S. Skiena, and A. Varshney, "Optimizing triangle strips for fast rendering," in *Proc. IEEE Visualization '96*, Yagel and Nielson, eds., pp. 319–326, 1996.
5. S. Kuschfeldt, T. Ertl, and M. Holzner, "Efficient visualization of physical and structural properties in crash-worthiness simulations," in *Proc. IEEE Visualization '97*, Yagel and Hagen, eds., pp. 487–490, 583, IEEE Computer Society Press, October 1997. ISBN 1-58113-011-2.
6. H. Hoppe, "Progressive meshes," in *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, ed., Annual Conference Series, pp. 99–108, ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.
7. S. Campagna, L. Kobbelt, and H.-P. Seidel, "Efficient decimation of complex triangle meshes," Tech. Rep. 3, University of Erlangen-Nürnberg, Germany, 1998.
8. S. Gottschalk, M. Lin, and D. Manocha, "OBB-Tree: A hierarchical structure for rapid interference detection," in *SIGGRAPH 96 Conference Proceedings*, H. Rushmeier, ed., Annual Conference Series, pp. 171–180, ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.
9. S. Campagna, *Polygonreduktion zur effizienten Speicherung, Übertragung und Darstellung komplexer polygonaler Modelle*. PhD thesis, University of Erlangen-Nuremberg, Germany, 1998. ISBN 3-89675-480-7.