

# Visualization and Pre-processing of Independent Finite Element Meshes for Car Crash Simulations

N. Frisch, D. Rose, O. Sommer, T. Ertl

Visualization and Interactive Systems Group  
Department of Computer Science  
University of Stuttgart  
Breitwiesenstr. 20-22  
70565 Stuttgart, Germany

<http://wwwvis.informatik.uni-stuttgart.de>  
{frisch, rose, sommer, ertl}@informatik.uni-stuttgart.de

## ABSTRACT

The recent transition from meshing the whole model towards the independent meshing of the assembly parts introduced new challenges also in the task of visualizing and pre-processing the finite element meshes. One part of our work is the visualization and removal of mesh penetrations and perforations, a frequently encountered problem when assembly parts are meshed independently. Further on, we focus on the visualization and the interactive definition of mesh-independent connections between car components represented by finite element meshes. We present methods to automatize the flange recognition, as connections are usually placed along flanges. There are basically three types of connections to visualize and edit: point links, line links and surface links.

**Keywords:** visualization, car crash simulation, finite elements, pre-processing, CAD

## 1 INTRODUCTION

The increasing pace at which new industrial products are brought to the market requires appropriate software tools. Automotive companies aim to place new, enhanced car models on the market in short time, in order to be competitive. Software has a key role in car development; several software tools are involved. Regarding car body design the passenger's safety must be considered. Crash tests are necessary and obligatory. Before performing a real crash test, hundreds of crash worthiness simulations are computed and analyzed. The number of real tests is reduced to a minimum, saving time and money.

In the car development process, the CAD data from the construction department are transformed into finite elements, a process called meshing. Meshing can be thought as a kind of tessellation. Most finite elements are quadrilaterals for numerical reasons. The mesh is the input for the finite element solver which computes the crash simulation. Mesh quality is a precondition for the

correctness of the simulation result. Therefore, the mesh is verified and corrected if necessary.

In the last years, a transition took place from meshing the car body as a whole towards independent meshing of the car's components. This transition required special link elements to be introduced. Linking the car body components with special elements like spotwelds permits an independent meshing of each component. Otherwise, each component's mesh would need to match the neighbouring part's meshes at the contact areas. Changing, modifying or adding an assembly part would require to re-mesh the adjacent car body parts, too. In the past, this often entailed expensive work in the mesh generation process, especially when the mesh data descended from inaccurate CAD models (see related work [1]). The new bonding elements reduce these shortcomings and, therefore, the development time.

A prototype named `crashViewer` was developed at

the University of Stuttgart within the BMBF<sup>1</sup> supported *AutoBench* project. The prototype can be used for improving finite element meshes as well as for visualizing the crash simulation input and output data, see also [9]. Visualizing output data in the so-called post-processing stage is necessary for analyzing the simulation results.

The prototype is based on the OpenGL Optimizer [8] high-level graphics API from Silicon Graphics. It also has a CORBA and Java based interface for the software integration platform CAE-Bench as described in [4]. The integration platform facilitates the control and data exchange between the different applications involved in car body development, leading to a significant increase of productivity.

Based on the *crashViewer* prototype we implemented various new pre-processing features which we describe in this article. These features are useful for making corrections and improvements to the simulation input deck. First we describe the detection, visualization, and removal of initial perforations and penetrations. We also present our texture based technique for interactive geometry clipping, which depends on node based values, particularly with regard to the visualization of potential flanges. Then we introduce effective features which allow the engineers to interactively define, examine, and modify car body part connections by means of welded joints and adhesive bondings.

## 2 INITIAL PENETRATIONS

By means of the meshing step parametric surfaces of the CAD data will be transformed into a discretized finite element mesh. Since the whole car body model consists of hundreds of independently meshed car body parts, this process may introduce '*initial perforations/penetrations*'. Figure 1 points out the initial penetrations as points, where one discretized surface is closer to another surface than the specified material thickness (left and right circle). Areas where elements intersect each other are called perforations (mid circle). Initial penetrations will cause initial forces in the simulation task, and this will falsify the simulation results. Therefore, it is important to detect and remove initial perforations/penetrations during the pre-processing of the simulation input data deck.

In order to detect those vertices which are positioned too close to an element of another car

component, the minimal distance of each vertex to each finite element has to be calculated. This task can only be efficiently solved by using some kind of hierarchical substructuring.

Gottschalk et al. [5] presented bounding volume hierarchy algorithms which have been 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. Their results have shown that object oriented bounding boxes perform better for collision detection than axis-aligned boxes because they need quite less interference tests.

Since we have to calculate point-to-polygon distances, which is cheaper than polygon-to-polygon tests, we use an axis-aligned bounding box hierarchy for the detection of initial penetrations. This requires less time for the bounding volume tree generation and saves any transformations of point coordinates during testing.

First of all we specify the maximum distance of interest which should be at least as thick as the maximal car component thickness. In the initialization phase 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 neighbours at the top level of the bounding

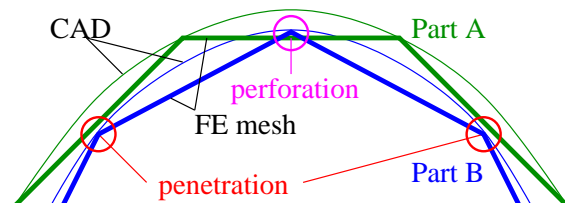


Figure 1: The assembly of independently meshed car body parts may cause '*initial penetrations*' which will influence the simulation results in an undesirable way. Even perforations (mid) could occur where the finite element meshes interpenetrate each other.

<sup>1</sup>German Ministry for Education and Research

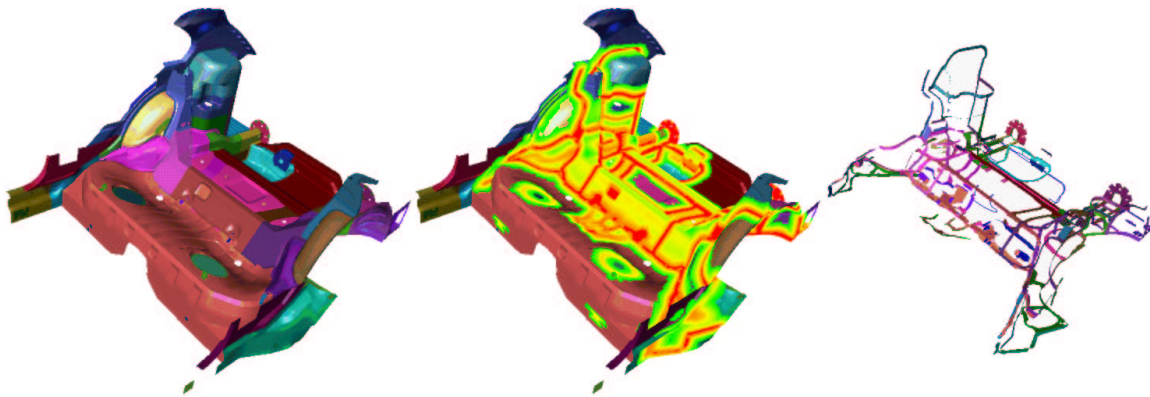


Figure 2: These images show parts of the back compartment of a car. The illustration in the middle visualizes the minimum distance from each node to the closest surface of another car body part up to 50 mm. On the right the same values are mapped to hide all geometry where this distance is more than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

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 Campagna [2] which considers each projection case and computes values only if they are needed for that particular case. For example, the computation of the minimal point-to-polygon distances for a car model with more than 600 components consisting of about 200,000 elements/nodes takes 17 seconds on an SGI R12k at 300MHz. Afterwards, the values are mapped into coordinates of a one-dimensional texture that is used for distance visualization [9].

After detecting all initial penetrations the engineer can mark car parts as '(un-)modifiable' before the removal algorithm is started which moves each node of the modifiable meshes along the calculated force vector in a number of iterations until the initial force vanishes. The selection of modifiable car parts is very important for the replacement of individual components by variants — here, the node coordinates of the variants should be adopted while the rest of the car body model stays fixed.

### 3 INITIAL PERFORATIONS

Initial perforations in the simulation input deck represent an even larger problem than initial penetrations. Initial perforations, in contrast to penetrations, cannot be eliminated by the finite element solver software which computes the crash simulation. Therefore we developed an algorithm

for detection, visualization and removal of perforations.

Like in section 2, the bounding volume hierarchy is used for efficient detection of potential perforations. Only if two bounding volumes of the undermost level overlap, each element edge of one volume is checked for intersections with each element of the other volume. Perforating elements are then marked with a one-dimensional RGBA texture which colors by default the perforating nodes to red (see Figure 3). The environment of the perforating node is also tinted by the texture, so that the perforating region can be discovered when looking from either side of the perforated plate. By modifying the alpha component of the texture like in section 2 the car components can be faded out so that only the areas around the perforations are visible. This feature gives a good overview about where the perforations are located.

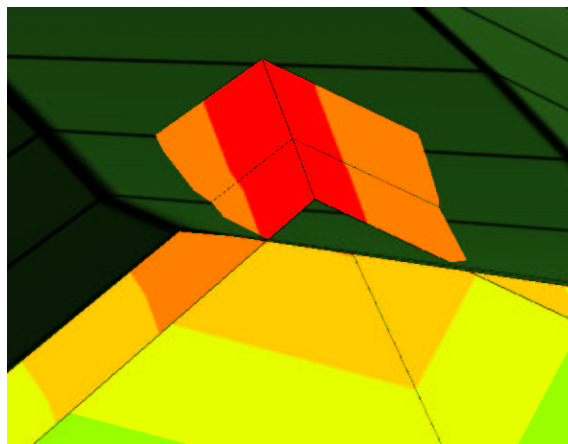


Figure 3: Example of a perforation

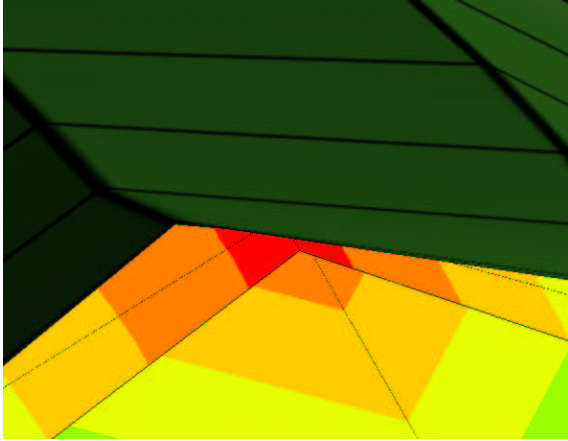


Figure 4: Removed perforation. The color marking is preserved here for illustration only.

In a next step, the perforations are removed (see Figure 4). This is an iterative process. Each perforating node of an intersecting edge is projected on the perforated surface and moved some small amount above that surface. If there exist edges which are completely, that means with both ends on the wrong side of the perforated surface, these will be transformed in some step into intersecting edges. The iteration is complete when no removable perforations are left. Sometimes, a node perforates two elements, by means of two different edges. Then, the decision on which of the two possible planes the node is projected is left to the user.

After the removal of initial perforations, the finite element mesh might be distorted. For best numerical results, the finite elements should be mostly equilateral and of similar size. This goal can be achieved through an iterative relaxation algorithm which is optionally started after the perforation removal. Like in a mass-spring model we consider the nodes being connected by springs instead of edges. Then we move the nodes iteratively by small amounts in the direction of the resulting force with the constraint that each node stays on the initial surface. Corners are not moved, while nodes at the edges or nodes at significant angles are moved only along that edge.

More precisely, in each iteration step the nodes are moved a small amount toward the center of all their neighbour nodes. Then, they are projected onto the initial surface or edge they belong to. In this way the shape of the relaxed car component stays mostly the same throughout the relaxation process. After the relaxation, and also before, the penetration removal process of section 2 can be started.

## 4 VISUALIZATION OF POTENTIAL FLANGES

During the assembly of a simulation input deck it is important to properly define the constraints between car parts, for example, with spotwelds or adhesive bondings. Generally, such contacts are placed at flanges. Since the simulation models become more and more complex it is helpful for the engineer while connecting adjacent components to restrict the visualization of the car body to those flanges. In *crashViewer* this can be done interactively without generating new geometry in the underlying scene graph API.

After the minimal point-to-closest-element distance has been computed for each mesh node, a previously specified distance range is mapped into the texture coordinate range  $[0.0, 1.0]$ . If these coordinates are used together with a one-dimensional  $(RGB)\alpha$ -texture map, the visibility of geometry may be influenced in correspondence to the mapped parameters.

We implemented a color editor which allows us to interactively modify the transfer function of one or multiple channels of the one-dimensional  $(RGB)\alpha$ -texture. Furthermore, it provides access to the OpenGL texture function. So after we have mapped the distance values into texture coordinates and we defined the texture map as an opaque colorband (each texel has an *alpha*-value of 1.0), the `GL_DECAL` function visualizes the minimal node-element-distances by mapping corresponding colors onto the geometry. Figure 2 (middle) shows the distance visualization of the back compartment. The texels which correspond to higher distances have been made transparent, therefore, the color of the original car component is shown in those regions.

If we only keep texels opaque which correspond to small distance values by setting the alpha channel of the rest of the texture to 0.0 and switching to `GL_MODULATE` only those regions will become visible where the opaque texels will be mapped onto the geometry. In order to avoid z-buffer pollution while rendering the scene without depth sorting we enable the alpha test. Using these settings just potential flanges will be rendered. (Figure 2, right)

If the user has found a satisfying threshold and wants to restrict the rendering to the corresponding regions, *crashViewer* determines which nodes of the finite element mesh fulfill the specified range limitation. Then an indexed geometry is generated that includes all elements which reference at least one of those nodes. The indexed

geometry is used to share the coordinate set with the original scene graph in order to minimize memory allocation. This reduces the load for the graphics pipeline and enables the user to navigate interactively even through complex car body models to define connections between the independently meshed car body parts.

## 5 SPOTWELDS

### 5.1 Setting

Spotwelds are the prevalent link between car body components. Spotweld information can be defined in the CAD data. In an early development stage, spotweld information is not fully available. Additional spotwelds need to be defined and some spotwelds eventually need to be moved or deleted also in advanced stages. The ability to define spotwelds directly on the finite element data reduces the working time for the crash simulation setup. The detour through the CAD department is cut short during the iteration loop.

The visualization of spotwelds by means of small, scalable red cuboids has proven good in practice. Erroneous spotwelds, e.g. spotwelds with missing or inappropriately positioned assembly parts, are visualized with different color and/or geometry in function of the error type, see Figure 5. Since a car contains thousands of spotwelds, it

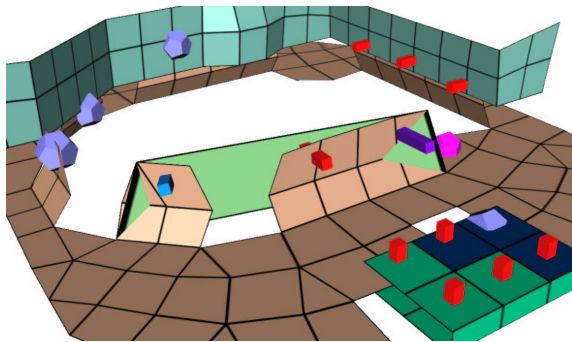


Figure 5: Various valid and erroneous spotwelds, visualized as cuboids and dodecahedra.

can be tedious to edit spotwelds one by one. For this reason, crashViewer provides the facility to define an entire spotweld line at once, by specifying the start and end point of the line with the mouse pointer. This generates a set of spotwelds along a straight line. The spotwelds are equidistantly positioned on this line. If the assembly part is curved, the straight line is projected onto the surface to find spotweld positions.

### 5.2 Curved Spotweld Lines

Obviously, not all spotweld lines in a car will be straight, neither can each curved line be obtained by projection of a straight line onto a car component. Therefore, a feature for generating curved spotweld lines was implemented. A first idea was to define curved lines by means of interactive spline curves. A drawback of this approach is that spline-curves are hard to position by the user exactly on the middle line of the flange. In the time the user positions the spline control points he could also position the spotwelds. Therefore, we used another approach: As spotwelds are usually positioned along flanges, a flange recognition algorithm was developed.

In Section 4 we describe an approach for fast visualization of potential flanges based on distances between components. The visualization gives the user a hint where flanges could be, but it is not an accurate flange detection. Flanges are special regions on plate components with the purpose of enhancing the connection between parts by enlargement of the contact area.

The flange detection algorithm for curved spotweld lines and bondings is done on a per-element basis. Each finite element either is a flange element or it is not, in function of the distance and the angle of this element with regard to the nearest element of the corresponding component. Distances are computed using the bounding box hierarchy described in Section 2. Finally, we observe that flanges have a considerable length but only a limited width. To achieve a curved spotweld line, the following steps are performed: first, the finite elements containing the start and end point of the desired line are determined. The next task is to find a shortest element path between those two. For each element on this path, the left and right flange borders are sought in order to find the mid-line. Finally, the spotweld positions on the mid-line are computed in function of the desired distance between spotwelds. The less trivial steps are finding the

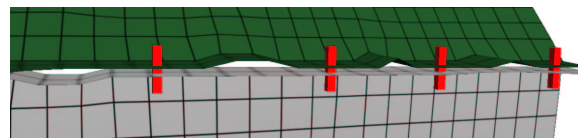


Figure 6: Irregularities like corrugations make flange detection difficult.

path and detecting the mid-line. For path finding we adapted an iterative algorithm from [7] originally designed for graph traversal. A special goal is to find a path even if the flange is interrupted by small gaps and corrugations (Figure 6). The gaps and corrugations can split the

flange area and make it impossible to find a path consisting exclusively of flange elements. In this case, the path should lead over or beneath the obstacles. The algorithm performed good in our tests, with respect to result quality and computing time. Computing time is not noticeable by the user even for large flanges. The following pseudo code describes the path finding procedure:

```

/** method findPath */
list<Elements>
findPath(start, target) {
    Element start, target, element;
    fifoQueue<Elements> fifo;
    list<Elements> reserve, result;
    bool targetReached = false;
    fifo.add(start);
    while (not targetReached) {
        element = fifo.retrieve();
        for (all neighbour of element) {
            if (element == target) {
                targetReached = true;
                neighbour.previous = element;
                break;
            }
            if (neighbour.unvisited) {
                neighbour.markVisited();
                // remember the path we came
                neighbour.previous = element;
                if (neighbour.isFlange)
                    fifo.add(neighbour);
                else
                    reserve.add(neighbour);
            }
        }
        if (fifo.isEmpty)
            fifo.consume(reserve);
    }
    // collect saved elements
    for (element = target;
        element != start;
        element = element.previous)
        result.append(element);
    return result;
}

```

The algorithm above terminates in linear time  $O(N)$  in function of the number of elements  $N$  to be checked. Each element is treated at most once. The search is a breadth-first search of the finite element mesh, searching with increasing radius around the start element. The problem of the small gaps and corrugations is also solved. When the fifo queue runs empty, this means we have checked all connected flange elements without reaching the target. In this case we have to jump over non-flange elements in order to reach the target. Therefore, a list of the encountered non-flange elements is built. The search continues from each of these non-flange elements when all other possibilities are exhausted. The short-

est path we get in this case therefore also contains non-flange elements. These will be properly treated later when determining the mid-line.

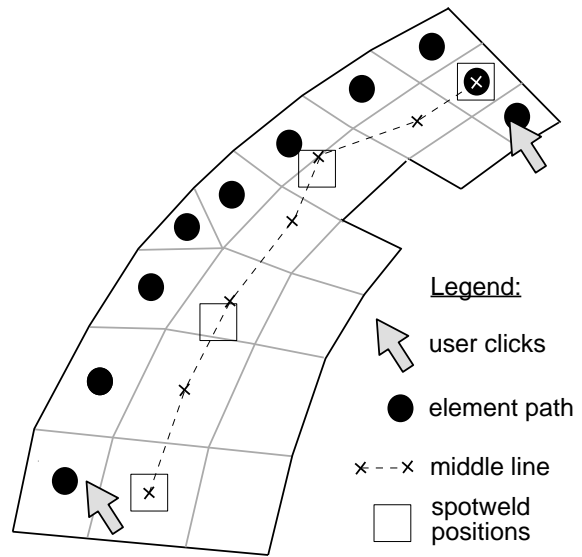


Figure 7: Path and mid-line example.

Once a shortest path has been found, it is necessary to compute the flange's mid-line in order to place spotwelds on it (see Figure 7). Some algorithms for finding mid-lines of polygons can be found in the literature, see [3] and the references there. However, none of them seems to be perfect for our needs, either because it handles just rasterized, or just planar polygons, and they require the mapping of the flange area to a proper polygon. Furthermore, these mid-line algorithms often return a branched mid-line, which is less suitable for our needs.

Instead, we developed another, more straightforward approach: For each element of the previously found path, search the left and right flange border closest to the current element. Then add the midpoint between the left and right border as a new point of the polygon line that will be the medial axis - or at least a sufficiently close approximation.

It is not enough to find the two nearest flange borders. The distance has to be measured on the flange surface, which is not necessarily a plane. Furthermore, the nearest two borders do not always define the mid-line, for example at flange corners. And last but not least, parts of the path may run cross to the flange and mid-line direction. Hence, for each quadrilateral flange element we search for the border in all directions: left and right, forward and backward. Then we add the border distance of the opposite directions and find the minimum.

Since quadrilateral finite elements on flanges are mostly aligned along flange direction, the minimum above is the flange width, the maximum would be the flange length. In exceptional cases, quadrilateral elements (which are un-aligned to the flange direction) are encountered. These can be handled by setting a limit for the maximal flange width measured by the number of elements traversed. Another special case originates from the few triangular elements and the flange corners containing triangular elements, as these elements introduce irregularities in the mesh. As these cases are not frequent, we can skip those regions and interpolate the mid-line. Furthermore, we skip non-flange elements of the previously found path, since the mid-line is undefined there. Interpolating such areas using a straight line gave good results in practice.

The ability to set several spotwelds at once implies the need for extended spotweld deletion features. Besides the ability to delete the spotweld beneath the cursor, the user can undo the last spotweld line. The flange and mid-line detection algorithm can also be used to mark all spotwelds in a given flange area. These spotwelds are colored white and can be deleted by one key press. Furthermore, the user can mark for deletion all spotwelds connected to a given assembly part or a pair of parts.

### 5.3 Enhanced Flange Detection

The curved spotweld lines described in subsection 5.2 enable the user to define a large number of spotwelds at once. In this way the assembling of independently meshed car components is accelerated. The user just has to set a start and an end point of the curved line on the flange, and a second assembly part to be connected. Observing that usually the goal is to weld the whole flange, we developed an algorithm to automatically detect the ends of the flange. Until now, we had a path searching algorithm to find a shortest path of adjacent finite elements leading from a given start point to a given end point. Now we consider one start point on the flange and we want a shortest element path leading from one end of the flange to the other. We call this path a bidirectional path because it leads from the start point in two opposite directions – except when the start point is at one end of the flange or the flange is circular. If we have only one direction, the given start point is one of the two ends of the bidirectional path.

The idea is to split the flange in two regions by

a more or less straight strip of quadrilateral elements (Figure 8). For the sake of robustness we demand that the given unique start point has to be on a quadrilateral element, and also the other elements of that transversal strip should be quadrilaterals. This is not a severe shortcoming because most elements are quadrilaterals. From the quadrilateral start element we search for the borders in all the four directions up to an adjustable maximum flange width. As the finite elements are almost always aligned with the flange borders, we get two strips crossing at the start element. The shorter of the strips is the transversal strip, which splits the flange in two regions. The search of a transversal and a longitudinal element strip crossing at a given element is similar to the procedure for finding the flange’s middle line by looking for the nearest opposite borders.

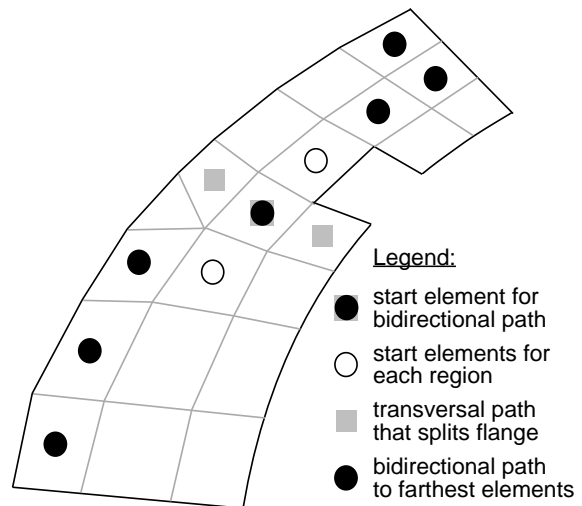


Figure 8: A given start element is used to split the flange in two regions for searching the flange ends.

The initial start element of the bidirectional path search, being a quadrilateral, has up to four edge-adjacent elements. We take the two adjacent elements which do not lie on the transversal strip as start elements for the two regions. As the path search algorithm described in subsection 5.2 performs a breadth-first search for the end point, we can use it with some modifications here, too. Now we do not look for a given end element, but for the flange element farthest away from the start element, yet connected by a path of flange elements with the first one. In this case the search algorithm terminates when the FIFO queue of flange elements is empty. The last flange element of the FIFO queue is the one we looked for because it is one which is farthest away from the start element. Usually, there are several elements equally distant, but choosing an arbitrary element, respec-

tively the last one before the FIFO queue runs empty, will do fine.

For searching a farthest element, we do not hold a list of spare non-flange elements as we did when searching a given element. When the FIFO runs empty, the search is over. We do not try to step over gaps and corrugations when searching the farthest flange elements.

Now we have two elements at the opposite ends of the flange. The midpoints of these elements can be taken as the points indicating the start and end point of the curved spotweld line, much in the same way as if the user defined them by mouse clicks. Start and end point are projected later on the flange's mid-line in order to define the points where the first and the last spotweld of the curved spotweld line is going to be set. The rest of the work is done in the same way, mostly by the same program code like in the case where start and end point of the curved spotweld line is specified by the user.

By automatically finding the flange's ends, the precision of mouse positioning can be less accurate: the user points just somewhere on the flange. The definition of a spotweld connection is accelerated significantly compared to the case where start and end point of the connection needs to be specified.

#### 5.4 Automatic Assembling of the Car Body

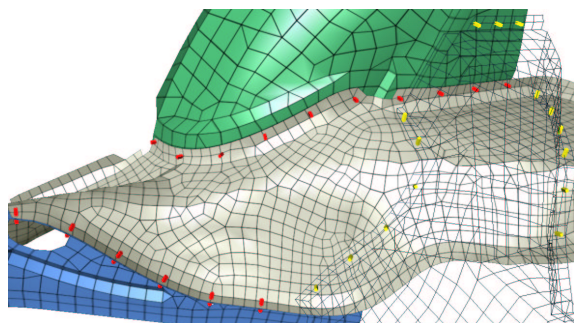


Figure 9: Fully automatic spotweld connection of all components. The spotwelds connected with the wireframed part are marked yellow.

In an early development stage, the assembly units of a car model are not yet connected at all. For a first evaluation it can be useful to connect all car components with their adjacent parts as long as a proper connection area, usually a flange, exists. Figure 9 shows an example of connecting 4 assembly parts at all suitable flanges, and also illus-

trates the visualizing of all spotwelds connected to a given component. With the bounding-volume-based approach for finding adjacent assembly parts (see section 2 and 4) we can efficiently focus on potential flanges for further examination. Once we know that two assembly parts are adjacent, we do not know yet which of the two, if any, has a flange suitable for connection. This information was given by the user in the case of interactive definition described earlier. For full automation we developed a heuristic approach.

We remember that a flange element fulfills the following criteria: it is sufficiently close and sufficiently parallel to the vicinity of the component going to be connected with this flange. Usually several elements of the non-flange part fulfill this condition. The difference is, that the flange elements are aligned with the flange. The corresponding elements that are going to be connected with the flange are not aligned with the flange borders. Therefore it is important to know which of the adjacent car components has a flange and which one does not, in order to obtain best results in flange detection.

One observes that a flange has a limited width but a considerable length, and a rather smooth surface. We developed an algorithm for testing if a finite element is part of a flange or rather resides on the opposite side of the connection. Similar like finding the nearest opposite borders of an element, we start searching in all four directions, but this time not checking the flange criteria. Instead, we follow each direction until a border or a significant angle is encountered. For this test, we need no information about the opposite component. If in this way we get two crossing strips (Figure 10), one of them being longer and the other one being shorter then or equal to the maximum assumed flange width, we can consider the start element, which is common to both strips, to be part of a flange area. Like before, we need to have no triangles in both strips. When encountering triangles, we can skip the check and try checking another element of the potential flange for suitability because we need just one flange element to start a bidirectional path search useful for generating a curved spotweld line.

In some cases, both car components going to be connected have a corresponding flange area, and sometimes none of them has a flange area but they still can be connected, for example if they are plane and overlap. In this cases it doesn't matter on which component the curved spotweld line is computed. In any case and for each potential start point we check if the opposite ele-

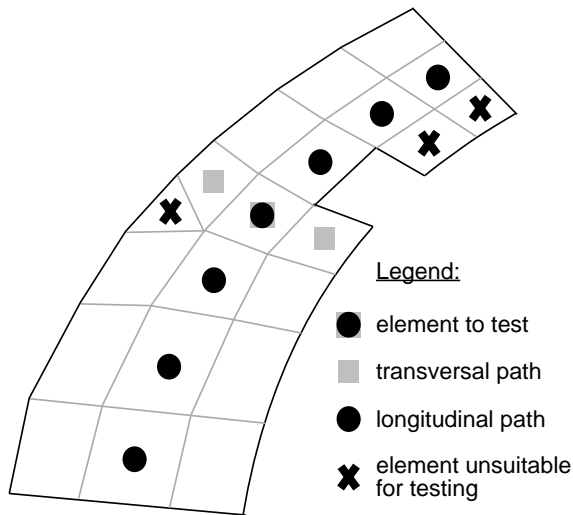


Figure 10: Computing the element strip length in two main directions in order to classify a potential flange element

ment has already been considered for computing a curved spotweld line. If a certain flag on that opposite element tells us that it has been considered, we can skip the current element because we want to set at most one curved spotweld line for each adjacent region of a car component pair.

This approach of checking whether the other part has been already considered does not work in all cases. Actually, we would need to check all the opposite elements during flange detection, but this is expensive and hard to do. A few cases exist where on both adjacent regions spotwelds would be set. Anyway, a distance check is performed, so that no spotweld is set if another spotweld exists at a given distance. This check prohibits the generation of new spotwelds if they would be part of a second spotweld line over the same area.

While the interactive definition of a curved spotweld line works interactively with no noticeable delay, automatically connecting all suitable assembly parts takes a few seconds. For example, on an SGI Octane2 workstation with 400 MHz R12k processor, the connection of all assembly parts by spotwelds required 45 seconds for a basic car body consisting of 248694 finite elements in 285 assembly parts. During this time 6829 spotwelds were generated at 50 millimeter intervals. A considerable amount of processing was necessary for detecting the adjacent regions and checking each potential start element for its suitability. These operations were not necessary when interactively generating each spotweld line because the proper components and the start point was given by the user. Therefore the per-spotweld computation time is larger when

automatically connecting all components of the model.

## 6 ADHESIVE BONDING

A relatively new bonding technique is the usage of adhesives. In contrast to spotweld lines, adhesive bondings are surface links which entail a completely different way of representation of the bonding agent. However, we kept the user interaction as simple as with spotweld lines.

### 6.1 Visualization of Adhesive Strips

The problem of all contact types is that they are hardly visible from outside since they naturally are comparatively small and are placed between two or more assembly parts. There are three different ways to solve this problem, each one with its own pros and cons. The first solution is to use transparent components. Thereby, the bonding agent and the counterpart can be easily seen. Though, with lots of bondings this may be confusing, due to many transparent components, which is a substantial argument against this solution. The second possibility is to render the components partially transparent namely only in the vicinity of the bonding. Unfortunately this operation interferes with parameter mapping. The last way is to illustrate bondings by a thickened representation. The advantage is that the bonding agent can be simply discovered because it sticks out of the bonded component's surface. However, this representation needs to be transparent to see the bonding surface.

All solutions have in common that they use transparency in some form. In doing so, the problem is a shortcoming in the high level graphics API we are using: The render action does not sort transparent objects back to front. A workaround for this problem will be shown below. We have chosen the last possibility: It minimizes difficulties with this deficiency and it yields the best clarity when visualizing many bonding structures.

There are two reasonable ways to thicken adhesive bondings. The first one is to use constant thickness equal to the maximum allowed distance of the assembly parts, and the second way is to employ a variable thickness so that the adhesive representation barely sticks out of the surfaces of the two bonded components. The advantage of the latter way is that we might as well represent the adhesive bond by textures on the assembly

parts stuck together, so we need no extra geometry. On the other hand we cannot use 1D parameter textures (Section 4) anymore, since few workstations support multi-texturing. The first solution has the benefit that the engineer gets a feedback about the distance of the assembled parts which finally led to the decision in favor of the first possibility. There the thickening is achieved by creating two additional surfaces shifted along the averaged node normal vectors.

To improve the conspicuousness of the bonding layer we use a checkerboard look-alike texture alternating opaque white and full transparency. The joined components can easily be seen through the transparent parts. Unfortunately we cannot use the alpha test feature of OpenGL to avoid drawing into the depth buffer as this may result in a performance problem on some machines. Therefore, we need a workaround for the mentioned graphics API transparency problem. We simply append all bonding related shapes at the end of the scene graph, even after the spotweld representations. Thus, it is guaranteed that spotwelds are visible in combination with adhesive bondings, which happens very often.

Figure 11 shows that we closed the outside of the adhesive strip so we get a better impression of the boundaries of the bonding surface which is the topic of the following section.

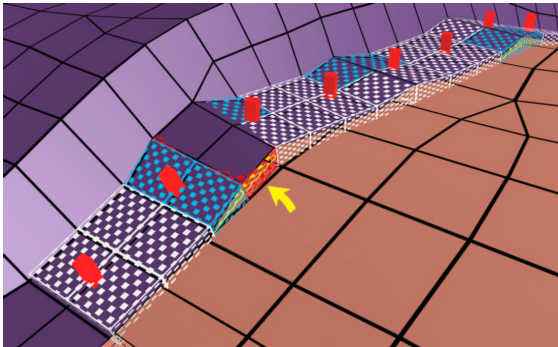


Figure 11: Example of an adhesive bonding in combination with spotwelds. Erroneous parts are color coded, and additionally the actual bonding element is drawn in yellow.

## 6.2 Detecting Boundaries

The search for the boundaries is based on a simple algorithm: For each edge we count the number of conjoint finite elements. In the internal `crashViewer` data structure we have already given the neighborhood relations of the elements and we also know the nodes which build up a finite

element. For each adhesive type element we examine its edges. All edges with two common finite elements are within the material, all edges which belong to only one element lie at an outer boundary (Figure 12). Three or more common

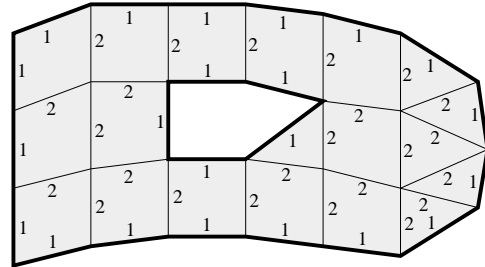


Figure 12: Finding the boundary edges.

elements for one edge are also possible, for example at T-joints (Figure 13). We call this an inner boundary and treat it like outer boundaries of all three (or more) elements. This is both a simple as obvious solution, since we can achieve exactly the same when using independent bondings. It is valid because it produces the same numerical results. Now we have a list of pairs of node IDs:

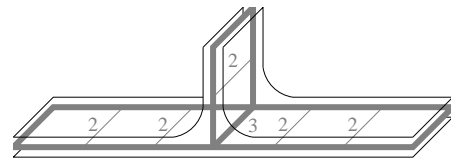


Figure 13: Special boundaries, e.g. T-joint

a stack of edges. We must merge them in order to get a continuous polygon line as a boundary. Of course there can be more than one boundary line. At adhesive strips with a hole, for example, we get two boundary lines, an outer one and another one for the hole (Figure 12).

One can think of the node ID pairs as domino stones. We start with a random stone and add matching stones to the ends of the forming domino queue. If we cannot find matching stones anymore this boundary line is complete. Because we have finite manifold surfaces, the two ends of the queue must match also, we get a closed line stroke. If we used up all stones (no edges are left) the job is done. Otherwise we start another boundary queue with one of the unused 'stones'.

## 6.3 Creation of Adhesive Strips

When creating adhesive strips we use the same techniques as for spotweld lines. Both, user interaction and internal algorithms are similar or

inherited. The engineer picks one assembly part and defines the length of the bonding on the second component with just three mouse clicks overall. The initial width of the adhesive strip is determined by a scalable maximum distance from the prior mentioned mid-line of the flange. All elements within that area meeting the flange criteria are used to build up the mid-surface of the actual bonding agent. Therefore, we project the eligible nodes of the first on the corresponding elements of the second component. The averaged coordinates of original and projected nodes define the mid-surface.

As mentioned in section ??, we use the normal vectors of the adhesive elements to derivate the thickened bonding representation. The normal vectors can be either copied from the shell elements of the first component or they can be recalculated from the node coordinates. In both cases this can result in alternating normal alignments. For lighting of the elements this does not matter because we enabled two sided lighting. However, the normals should point into the same halfspace which is restricted by the adhesive surface because we shift the thickened representation along these normals. If we would allow switching normals, we might get an invalid vector when averaging neighboring element normals at the common nodes. This may lead to an intersecting or too thin visualization of the adhesive strip.

When a new strip is created we define the normal of the first element as fixed. Starting from this element the neighboring elements are checked for a flipped normal direction. Using the sign of the dot product of two neighboring normals  $\vec{n}_1$  and  $\vec{n}_2$  to gain information about the right alignment can be a first improvement. In rare cases this check will fail as you can see in Figure 14b. By means of the tangent vectors  $\vec{t}_1$  and  $\vec{t}_2$  which point from the center  $M$  of the shared edge to the centers  $C_1$  and  $C_2$  of the respective elements (average of all four or three node coordinates) we are able to determine the correct orientation of the normal by checking the sign of  $(\vec{n}_1 \times \vec{t}_1) \cdot (\vec{n}_2 \times \vec{t}_2)$ . If the vector products point into the same halfspace, restricted by the plane that is defined by  $M$ ,  $C_1$ , and  $C_2$ , the dot product will be positive and we must flip the normal  $\vec{n}_2^*$  of the second element. Propagating this test from element to element will result in normals standing on the same side of the surface.

One problem is left: the shape of the adhesive strip around the start and end point. We need to find a rule to achieve a straight termination. Thus, we take the vector  $\vec{p}$  defined by the first

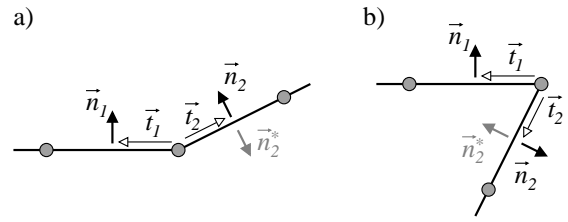


Figure 14: Tangent vectors help to determine correct normal vectors.

two points of the mid-line as initial clue (Figure 15). Then we project the position vectors of all

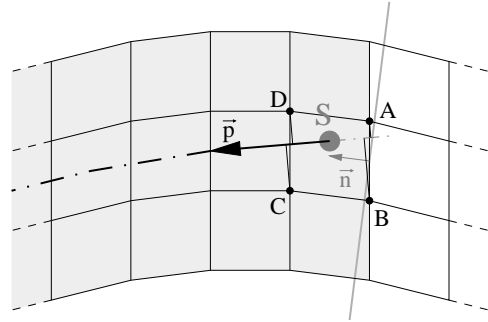


Figure 15: Calculation of the limiting planes.

four (or three) nodes of the nearest element onto this vector. The two nodes which lie farthest from the mid-line (here point A and B have the lowest dot-products) define the terminating edge for this element. To obtain a limiting plane for the whole adhesive strip we simply take the midpoint of this edge and the center of the element to define the normal vector  $\vec{n}$  of this limiting plane. In the same manner we specify the termination of the other end of the bonding strip.

#### 6.4 Error Detection and Validation

When adhesive elements are created or read from a simulation input deck they are scanned for erroneous areas. If those invalid elements were generated by the creation process they will be deleted immediately. If they were already in the data file these areas will be color coded depending on the error case. Failures like missing projection points, too far contact parts, too high normal deviation, or the lack of a compound material can also appear at spotweld links and are therefore indicated using the same corresponding color. Additionally the actual adhesive element is rendered in yellow in the middle of the volumetric bonding representation. This relieves the engineer's decision what to do. Generally, he will just delete the erroneous elements. In Figure 11 you can find adhesive elements with too far bonding counter-

parts marked red (naturally, the bonding surface is partially hidden by one of the counterparts in this case) and too high normal deviation marked blue. The yellow link elements in the middle are pointed out by an arrow.

Since adhesive bondings are surface links, the verification is somewhat more expensive than with spotweld points. We will show it exemplarily for the intersection test which checks for other materials or adhesive elements, not part of the compound, but interfering with it in an illicit way. In Figure 16 the adhesive connects part 1 and 2. The grey rectangle shows the area of influence where the surface link algorithms of the simulation software search for nodes and elements which will be bonded. This area correlates with the thickened representation of the adhesive. It is easy to see that part 4 prevents the successful connection and therefore the appropriate adhesive element must be marked as defective, but how can this be put in an algorithm?

We use the bounding volume hierarchy prior mentioned to preselect the neighboring parts. After that, we use the bounding volumes to gain all elements of the resulting subset in the sphere of influence of the reviewed adhesive element. If only one or no material element is found in the vicinity, either the gap between the counterparts is too big, a compound material is missing or something similar is wrong. These cases are quite easy to handle. When only the two compound materials are found and they lie on different sides of the adhesive surface (checked with the normal vector) the bonding is valid, assuming the normal deviation is in the specified range. If more elements of different parts are found we take the vectors from the center of the adhesive element to the centers of the concerning elements and project them onto the normal of the bonding element. This is a good and fast approximation for the distance, and additionally we can distinguish on which side of the adhesive the part is. Now we can decide if the bonding element is valid, even if another part (part 3) is nearer to it than one of the compound parts (part 2), as shown in Figure 16.

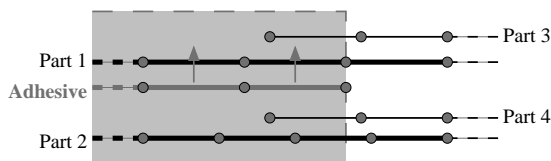


Figure 16: Interfering neighboring parts.

Elements with the same error type are merged

to areas using neighborhood relationships. Then we calculate a camera perspective pointing to the central point and save it into a list. This way the engineer is able to head towards erroneous areas and easily navigate to the next spot.

## 7 WELDED JOINTS, EDGE LINKS

In addition to spotweld lines, welded joints or edge links were introduced to the simulation software. In comparison to the 0-dimensional spotweld point lines and the 2-dimensional surface linking adhesives, these are 1-dimensional linear elements. Three scenarios can be defined for this bonding type: edge-edge-connection, edge-surface-connection, and surface-surface-connection along a line. Up to now we finished the implementation of the visualization and creation of the two first cases, the edge-link types. The latter line-link type is currently under development.

For visualization we use a triangular profile and shift it along the edge link line. Once again we need to gain the normal directions of these elements like we did it for the adhesive strips to avoid a twisted or too thin visualization of the edge link geometry. We use the same colors as for spotwelds to clarify the relationship to that joint type. An example can be seen in Figure 17.

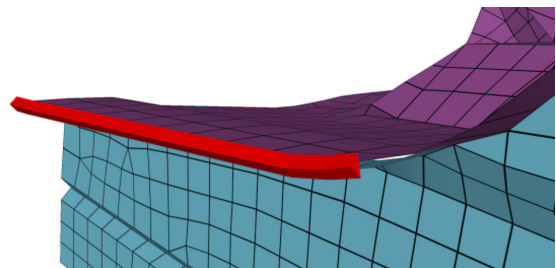


Figure 17: Visualization of edge link elements.

When creating a new edge link line, the user interaction is solved analogous to spotweld lines and surface links. The underlying algorithm is different. We do not need to do the complicated flange detection because these elements are restricted to the boundary of one or both compound parts. We can apply the previously introduced boundary detection algorithm (see Section 6.2) to the two counterparts. This procedure is simpler and faster than the flange detection, though both algorithms are running in the range of milliseconds to tenth of seconds and are therefore highly interactive. In the vicinity of the selected start

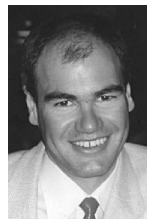
and end coordinate the nearest boundary point of both material parts is searched. The edge link elements are gained by projecting the boundary nodes between the start and end point on the second compound material or its boundary line, whatever is nearer. Hereby the shorter of the two possible paths is chosen. As with the other bonding types the actual edge link positions are calculated by averaging the corresponding original and projected coordinates. Finally the elements distance and normal deviation are verified and illegal edge links are removed.

## 8 CONCLUSIONS

We introduced a set of techniques which reduces the workflow paths in the car development process. The transition to independently meshed car body parts required efficient algorithms for the interactive definition, modification, and deletion of assembly part connections like spotwelds and adhesive bondings. The presented visualization of such constraints and the rendering restriction to potential flanges supports the engineer in the pre-processing step. Furthermore, the algorithms for the detection and the controlled removal of initial penetrations allow the testing of multiple component variants. The described tools have been developed in cooperation with the BMW Group and they are in productive use at the crash simulation department.

## REFERENCES

- [1] G.Barequet, S.Kumar: *Repairing CAD Models*, in *IEEE Visualization '97 Conference Proceedings*, pages 363–370, IEEE Computer Society Press
- [2] Swen Campagna: *Polygonreduktion zur effizienten Speicherung, Übertragung und Darstellung komplexer polygonaler Modelle*, PhD thesis, University of Erlangen-Nuremberg, Germany, 1998.
- [3] F. Chin, J. Snoeyink, and C. A. Wang: *Finding the Medial Axis of a Simple Polygon in Linear Time*, Proc. 6th Ann. Int. Symp. Algorithms and Computation (ISAAC 95), Lecture Notes in Computer Science 1004, pp. 382-391, 1995.
- [4] N.Frisch, T.Ertl: *Embedding Visualization Software into a Simulation Environment*, in *Proceedings of the Spring Conference on Computer Graphics*, pp. 105-113, Bratislava, 2000
- [5] Stefan Gottschalk, Ming Lin, and Dinesh Manocha: *OBB-Tree: A hierarchical structure for rapid interference detection*, in Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 171–180. ACM SIGGRAPH, Addison Wesley, August 1996, held in New Orleans, Louisiana, 04-09 August 1996.
- [6] R. Sedgewick: *Algorithms in C++, Parts 1-4*, Addison-Wesley 1998.
- [7] Silicon Graphics, Inc.: *OpenGL Optimizer Programmer's Guide: An Open API for Large-Model Visualization*, at [http://www.sgi.com/software/optimizer/tech\\_info.html](http://www.sgi.com/software/optimizer/tech_info.html)
- [8] O. Sommer, T. Ertl: *Geometry and Rendering Optimizations for the Interactive Visualization of Crash-Worthiness Simulations*, in Proc. of SPIE: Visual Data Exploration and Analysis VII, vol.3960, pp. 124-134, January 2000.



**Norbert Frisch** is a PhD student at the University of Stuttgart, Germany. His research concerns the interactive manipulation of finite element representations and also the embedding of visualization software into integration environments. He completed his studies of computer science at the University of Erlangen, Germany, in 1996.



**Dirc Rose** is a PhD student at the University of Stuttgart, Germany. His main interests include mesh simplification and refinement, as well as the interactive manipulation of finite element representations. He finished his studies of aero and space technologies at the University of Stuttgart, Germany, in 1999.



**Ove Sommer** will finish his PhD thesis this year at the visualization and interactive systems group of the University of Stuttgart, Germany. His research interests include hardware accelerated volume rendering and interactive visualization and manipulation of data coming from structural mechanics. He received his MS in computer science in 1997 from the University of Erlangen, Germany.



**Thomas Ertl** is a professor of visualization and interactive systems in the computer science department of the University of Stuttgart. His research interests include volume rendering, flow visualization, multi-resolution analysis, parallel and hardware accelerated graphics, large datasets and interactive steering. He received an MS in computer science from the University of Colorado at Boulder, and a PhD in theoretical astrophysics from the University of Tübingen, Germany.

## LIST OF FIGURES

1	The assembly of independently meshed car body parts may cause 'initial penetrations' which will influence the simulation results in an undesirable way. Even perforations (mid) could occur where the finite element meshes interpenetrate each other. . . . .	2	12	Finding the boundary edges. . . . .	10
2	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 2 mm using the texture subsystem and the alpha test. The rendered geometry shows potential flanges. . . . .	3	13	Special boundaries, e.g. T-joint . . . . .	10
3	Example of a perforation . . . . .	3	14	Tangent vectors help to determine correct normal vectors. . . . .	11
4	Removed perforation. The color marking is preserved here for illustration only. . . . .	4	15	Calculation of the limiting planes. . . . .	11
5	Various valid and erroneous spotwelds, visualized as cuboids and dodecahedra. . . . .	5	16	Interfering neighbored parts. . . . .	12
6	Irregularities like corrugations make flange detection difficult. . . . .	5	17	Visualization of edge link elements. . . . .	12
7	Path and mid-line example. . . . .	6			
8	A given start element is used to split the flange in two regions for searching the flange ends. . . . .	7			
9	Fully automatic spotweld connection of all components. The spotwelds connected with the wire-framed part are marked yellow. . . . .	8			
10	Computing the element strip length in two main directions in order to classify a potential flange element . . . . .	9			
11	Example of an adhesive bonding in combination with spotwelds. Erroneous parts are color coded, and additionally the actual bonding element is drawn in yellow. . . . .	10			