

# Deformation Of Finite Element Meshes Using Directly Manipulated Free-Form Deformation

Norbert Frisch

Thomas Ertl

University of Stuttgart, Germany  
{frisch, ertl}@informatik.uni-stuttgart.de

## ABSTRACT

*CrashViewer* [5, 18] is a tool for visualizing car crash simulation input and output data consisting of finite element meshes. For a shorter workflow, a feature for local deformation of the car components represented by FE meshes is desired. This feature allows to quickly make minor corrections and enhancements directly on the FE mesh. The roundtrip through the CAD department and the remeshing of the CAD representation is avoided. The crash simulation can be started immediately with the modified car component(s).

In order to achieve this goal, we investigate a variety of approaches related to free-form deformation (FFD). We base our implementation on the directly manipulated free-form deformation (DMFFD) [8, 7], adding various enhancements to this approach. We improve the deformation behaviour to be more uniform and consistent over time and space, and thus more intuitive and user friendly. Further, we present ways for increasing the efficiency of the computations necessary for mesh deformation. Finally, we specify a simple user interface which is easy to use while providing the necessary precision.

## Categories and Subject Descriptors

I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling

## General Terms

Algorithms, Performance

## Keywords

free-form deformation, finite elements, CAD

## 1. INTRODUCTION

Product cycles in the automotive industry like in other industries are becoming shorter. In order to be competitive,

the companies need to optimize the product development process. One aspect of the car body development is the passenger safety. A certain set of real crash tests is obligatory for the new car model to be approved. Crash simulations based on finite elements help to reduce to the minimum the number of expensive prototypes being crashed. Crash simulation results are available in a small fraction of the time a real prototype needs to be built.

It is well known that car prototypes are designed with a CAD system, which deals with curved surfaces, parametrically represented by various spline types. In order to perform a finite element calculation like a crash simulation, the car components need to be meshed, i.e. transformed into a finite element mesh. For numerical reasons, the finite elements are preferably quadrilaterals and just a few elements are triangles. Just a few years ago the transition towards independently meshed car components has been done, with the advantage of being able to modify a car component without requiring anymore to re-mesh the whole car model.

After the data from the CAD department have been meshed, the mesh usually cannot be input directly into the simulation. Discretisation of the continuous surfaces can lead to inconsistencies like penetrations, perforations and undesired gaps. Also, during the development process, some car component data might be available only in a deprecated version, or a version newer than the rest might be tested regarding crash behaviour. In all these cases, some minor deformations of geometry are necessary to resolve the inconsistencies. It is desirable, and sometimes the only way (e.g. in the case of discretisation artefacts) to perform those deformations on the finite element data instead of requesting a change from the CAD department.

In the *CrashViewer* tool for visualization of finite element data an automatic detection and removal of perforations and penetrations already exists, which is described in [5]. But sometimes, as in the case of different component versions, a more flexible, explicit geometric deformation is desirable. In this paper we describe a solution suitable for the specific needs of finite element deformation, which can also be used in other areas where meshes need to be deformed. Special attention is given to the efficiency of the algorithms in order to achieve interactivity. Further, a simple and intuitive is provided to facilitate efficient work without much training.

## 2. RELATED WORK

Basically, there are two types of deformation approaches, given an arbitrary polygonal mesh: the spline-based free-form deformation and the direct transformation of some

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SM'02, June 17-21, 2002, Saarbrücken, Germany.  
Copyright 2002 ACM 1-58113-506-8/02/0006 ...\$5.00.

mesh region in combination with the discrete fairing of the adjacent regions [11, 10, 12].

We have chosen the spline based free-form deformation, because the meshes we want to modify are discretized spline surfaces from a CAD system. This approach automatically preserves mesh details and does not require the computation of a resolution hierarchy. However, it is a good idea to give the engineer several tools at hand, therefore we may in the future further examine the suitability of the discrete fairing approach for deforming car components given as finite element meshes.

Free-form deformation (FFD) [17] and extended FFD [2] both embed the whole object into a tensor product volume. The volume can be deformed by means of spline control points while the embedded object is deformed accordingly. Usually Bézier splines or B-splines are used for their properties like local control, affine/projective invariance, continuity etc. Unfortunately, the handling of such splines is not very intuitive. Especially the large number of control points for a tensor product volume is confusing and the actual object is more or less obscured by the control points.

A more natural way to deform a surface is to shape it directly, like it would be done with a real object. Direct manipulation of FFD [7] can satisfy this requirement. One or several arbitrary points can be moved while the control points are computed such that the constraints are satisfied in a least squares sense.

Hu et al. [8] computed the control point displacements in a more efficient way, using Lagrange optimization for the least squares fitting. They also have shown that a multiple points constraint is equivalent to some sequence of single-point constraints. This means that every task of moving several points simultaneously can be solved as well by moving each point sequentially by a certain amount.

Cua and Neumann [1] presented a hardware architecture for accelerating the computation of free-form deformation (but not for computing the control point displacements in DM-FFD). Basically, this is an extension of the surface evaluators in OpenGL. We cannot say if such hardware will be available on the market one day. But it is well-known that graphics hardware in general becomes more programmable and thus more flexible. For OpenGL 2.0 more flexibility is planned in order to support programmable graphics hardware. The white paper found at [14] would allow the implementation of free-form deformation acceleration in the hardware vertex processor in a custom way. In this paper, we also present an approach for using OpenGL's surface evaluators (available today) to help computing free-form deformations.

### 3. DIRECT MANIPULATION OF FFD

Let  $Q(u, v, w)$  be a tensor product spline volume:

$$Q(u, v, w) = \sum_{i,j,k=0}^{l,m,n} P_{i,j,k} R_{i,j,k}(u, v, w) \quad (1)$$

where  $R_{i,j,k}(u, v, w) = B_i(u)B_j(v)B_k(w)$  are basis functions and  $P_{i,j,k}$  the control points of this tensor volume.  $l+1$ ,  $m+1$  and  $n+1$  are the number of control points respectively the number of basis functions in each dimension. Without restriction of generality we let  $0 \leq u, v, w \leq 1$ . The minimal displacements of the control points  $P_{i,j,k}$  in order to achieve the movement of a point  $S$  (start point) inside the volume  $Q(u, v, w)$  to an arbitrary point  $T$  (target point) is given by

([8]):

$$\delta_{i,j,k} = \frac{R_{i,j,k}(u_S, v_S, w_S)}{\sum_{i',j',k'=0}^{l,m,n} R_{i',j',k'}^2(u_S, v_S, w_S)} (T - S) \quad (2)$$

with  $Q(u_S, v_S, w_S) = S$ .

The tensor product spline volume  $Q'(u, v, w)$  with the new control points  $P'_{i,j,k}$  can be written:

$$\begin{aligned} Q'(u, v, w) &= \sum_{i,j,k=0}^{l,m,n} P'_{i,j,k} R_{i,j,k}(u, v, w) \\ &= \sum_{i,j,k=0}^{l,m,n} (P_{i,j,k} + \delta_{i,j,k}) R_{i,j,k}(u, v, w) \\ &= \sum_{i,j,k=0}^{l,m,n} P_{i,j,k} R_{i,j,k}(u, v, w) \\ &\quad + \sum_{i,j,k=0}^{l,m,n} \delta_{i,j,k} R_{i,j,k}(u, v, w) \\ &= Q(u, v, w) + \sum_{i,j,k=0}^{l,m,n} \delta_{i,j,k} R_{i,j,k}(u, v, w) \quad (3) \end{aligned}$$

### 4. THE BÉZIER SPLINE APPROACH

Our goal was to provide a flexible and simple to use mechanism for little deformations of finite element meshes. It should be fast enough to allow interactive use. The deformation characteristics should neither depend on the previously performed deformations, nor on the location of the region being deformed. The deformed regions and their boundaries should provide an adjustable level of continuity.

In our first implementation we used Bézier splines which worked fine at the first glance. The control points are placed on a regular grid, aligned with the bounding box of the car component. An advantage of Bézier splines is that they contain the control points at the ends, so that the convex hull of the control points is initially equal to the tensor volume, i.e. the image of  $Q(u, v, w)$ .

After displacing the control points, the new shape of the car component has to be computed. Firstly, for each vertex  $V$  of the finite element mesh, we need to compute the spline parameters  $u_V, v_V, w_V$ ,  $0 \leq u_V, v_V, w_V \leq 1$  such that  $Q(u_V, v_V, w_V) = V$ . Then, the new position of  $V$  is equal to  $Q'(u_V, v_V, w_V)$ , according to (3). The calculation of the parameters  $u_V, v_V, w_V$  such that  $Q(u_V, v_V, w_V) = V$  normally implies the solving of an equation of degree  $n$ , where  $n$  is the degree of the splines we use [7]. In our case a  $G^2$  continuous modeling should be possible, which implies splines of degree 3.

The solving of a third degree equation can be avoided, if the splines fulfill the *linear precision* property. Linear precision here means that

$$\sum_{i=0}^n \frac{j}{n} B_i^n(t) = t \quad (4)$$

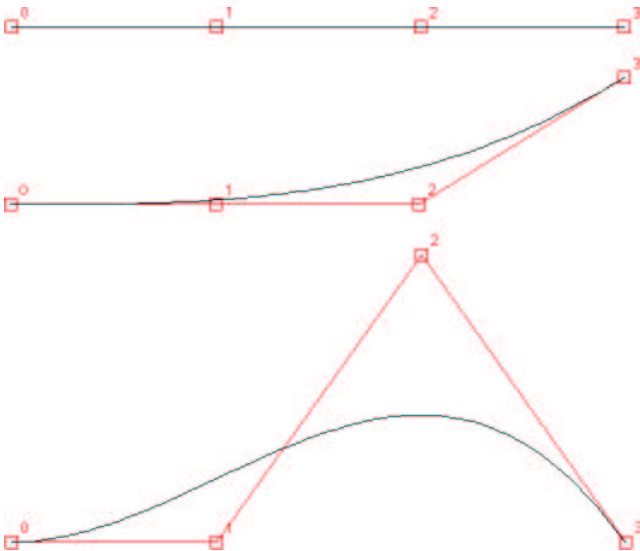
for degree  $n$  basis functions  $B_i^n$  with parameter  $t$  (see also [4]). Linear precision implies that, if all the control points of a spline are positioned equidistantly on a line, the parameter  $t$  for which a given spline value is reached can be obtained by linear interpolation. The spline function value moves uniformly. Some sources state that linear precision just means

that the spline curve becomes a straight line whenever the control points are collinear. This latter definition we don't use here.

Bézier curves have the linear precision property [4]. We could therefore use a Bézier tensor volume with equidistant control points to allow linear interpolation of spline parameters. This approach has some drawbacks. Firstly, if we want more control points, we must increase the degree of the curve. Moving a control point somewhat affects also the distant regions of the volume, we have no local control.

Sometimes it is desirable to deform adjacent car components simultaneously. Then, we would need to embed the whole car in a volume, to avoid discontinuities at volume boundaries. But embedding the whole car in a Bézier tensor volume is not practicable, mainly because of the lacking local control and for performance reasons.

Further, deformations close to the border of the Bézier tensor volume have different characteristics than deformations in central regions. Figure 1 sketches an example of such an inconsistent deformation behaviour. In this example, a point of a surface is displaced along the surface normal. If the point is at the margin of the volume, the surface normal changes when the point is displaced. If the displaced point is located in a central region of the volume, the surface normal at that point does not change and the deformation has another shape than in the previous case. This behaviour is annoying to a user who normally doesn't see the control mesh during direct manipulation and who normally has no background in spline theory.



**Figure 1:** A surface in a Bézier volume deforms differently depending on the initial position of the displaced point. Up: initial setting. Middle: displacement of the surface near control point 3 in normal direction. Down: same displacement vector applied near control point 2.

## 5. THE B-SPLINE APPROACH

In order to achieve local control, a standard approach is the use of B-spline tensor volume. This way, increasing the number of control points does not imply degree elevation.

By adjusting the number of control points, the range of the deformation can be modified in each dimension.

In most of the previous FFD applications, the first and the last control point of a B-spline lies on the spline's ends. This is achieved by knot multiplicity of  $(degree + 1)$  at the ends of the knot vector.

If such a B-spline curve has exactly  $(degree + 1)$  different control points, it is equal to a Bézier curve. With more control points, we have a B-spline with local control. But, considering tensor volumes, we still have the problem of lacking continuity at the volume boundary. Moreover, the linear precision condition is not fulfilled for this kind of B-splines. Linear precision eventually can be achieved through weighting of the control points [3], the spline thus becoming rational. Attaching different weights to the control points unfortunately changes the deformation behaviour in an irregular and undesired way.

Due to these shortcomings of the common B-spline curves, we switched to B-spline curves with uniformly distributed knots. The linear precision property is granted now, so that the spline function parameters  $(u, v, w)$  for the mesh vertices and for the specified start point are efficiently computed through linear interpolation. Furthermore, the deformation behaviour is now uniform.

This new B-spline curve does not reach the terminal control points anymore. As an effect, the tensor volume in which we embed the object for deformation is now smaller than the convex hull of the control points. Anyway, the user does not need to bother with the control mesh, he directly manipulates the embedded object while the control mesh is not displayed.

## 6. UNIFORM DEFORMATION BEHAVIOUR

Other publications on direct manipulation of FFD [7, 8] deal with the embedding of the whole deformable object into the tensor spline volume. Deformations are sequentially applied on the same, more and more distorted volume. This procedure provides the ability to undo a deformation by reversely displacing the start point. It also provides commutativity of the deformation operations [8]. Further, the spline function parameters  $(u, v, w)$  of each mesh vertex need to be computed only once [7].

Keeping the same embedding volume for several deformation operations also has disadvantages: the current deformation behaviour is influenced by previously performed deformations. For the user, who does not want to see the control mesh, the system reacts differently whether he interacts with a previously distorted region or with an untouched one - and he will not remember which is which. And even if he knows that the region is already distorted, it is hard to tell anymore which area will be affected by the new deformation and how the deformation will look like. Furthermore, the deformation behaviour in a global volume also is slightly different depending on where the start point is located.

In *CrashViewer*, like in other software, the user must be able to save the deformed object and continue work later. It is undesirable to save the deformed control mesh to a file, not only because the standard data format does not support this feature, but also because of the need for a deformation behaviour not depending on previous operations. Therefore it is a good idea to provide a new cuboid embedding volume for each deformation operation. This volume actually must not embed the entire object but only the region which is

affected by the distortion. The continuity properties of the B-splines with uniform knot vector provide a  $(G^{degree-1})$  continuous transition between the deformed region and the rest of the object. This locally constrained embedding volume is always centered around the start point, so that the deformation behavior of the object is also independent from the location of the start point. Providing a new, local embedding volume for each deformation step, the deformation behaviour is uniform in time and space.

Some advantages of keeping the same embedding volume for several deformations can not be directly used anymore. Regarding undo-ability, we can instead save the vertex positions for an undo operation. We also present another solution later on. The lack of commutativity is not a problem in practice. The fact that the spline parameters  $(u, v, w)$  need to be recomputed for each deformation operation is not a problem because we can do this by linear interpolation now.

## 7. PERFORMANCE ISSUES

When the user picks one point for displacement, the point's coordinates  $(u, v, w)$  relative to the embedding volume are computed. From (2) results that  $(degree + 1)^3$  control points are displaced in case of a one point constraint. The reason is, that for a given spline parameter  $u$  at most  $(degree + 1)$  B-spline basis functions  $B_i(u)$  are non-zero [16].

Each of the displaced control points affects a region of  $(degree + 1)$  spline segments in each dimension. Since the control points are placed equidistantly on a straight line, the distance between control points is equal to the length of a spline segment. Control points lie at the segment's boundaries or at their middle, depending on whether the degree is even or odd. The region affected by all the displaced control points in each dimension is  $(2degree + 1)$  spline segments, respectively  $(2degree + 1)$  times the spacing between control points. To reproduce a spline with  $m$  segments, we need  $(m + degree)$  control points. Therefore, the volume of influence is spanned by  $(3 degree + 1)$  control points per dimension, that means a total of  $(3 degree + 1)^3$  control points.

Since the embedding volume is always centered around the start point, only a fraction of the control point displacements must be computed due to symmetry. For a spline degree of 3, only 6 out of 64 control point displacements need to be computed using (2), the others are equal to one of those 6 values.

In the case of a large volume embedding the whole object, the start point parameter for the displacement would have different values for each operation. Sometimes, the start point parameter might be close to a spline knot point, at other times the start point might be somewhere in the middle of a segment. As we can see in Figure 3, the deformation characteristics varies slightly depending on where the start point lies, although all control points are aligned in a uniform manner. This figure also shows the comparison between a second and a third degree spline volume.

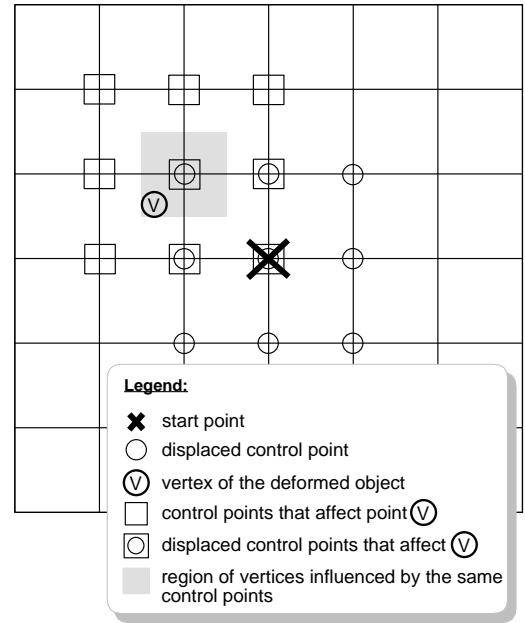
Placing the tensor volume in such a way that the start point's spline parameter  $(u_S, v_S, w_S)$  lies exactly on a knot segment boundary, the number of displaced control points is reduced by one in each dimension. The reason is that only  $degree$  basis functions are different from zero at B-spline segments boundaries, rather than  $(degree + 1)$ . Spanning the volume by means of  $degree^3$  instead of  $(degree + 1)^3$  control points is therefore mathematically correct.

Using only  $degree$  control points reduces the number of control point displacements to calculate. Decreasing the number of control points, however, the displacement characteristics loses some of its smoothness, the curvature distribution becomes less uniform, although the continuity properties are still the same. Therefore, in our application we decided to use  $(degree + 1)^3$  control points in each dimension. Again, regarding the case of a global, persistent control grid, it would depend on the start point position whether the behaviour is that of the first case or that of the second case or something inbetween.

Looking at (2), we observe that actually the computationally intensive displacement factor of the control points

$$\frac{R_{i,j,k}(u_S, v_S, w_S)}{\sum_{i,j,k=0}^{l,m,n} R_{i,j,k}^2(u_S, v_S, w_S)} \quad (5)$$

remains the same unless we change the degree, because we always place the volume in such a way that  $(u_S, v_S, w_S) = (0.5, 0.5, 0.5)$ . Therefore, the displacement factor for each control point can be re-used and needs just be multiplied with the deformation vector  $(T - S)$ .



**Figure 2: Most vertex displacements require just a few bases functions to be computed.**

For deformation regions containing a significant number of object vertices, the computation time is proportional to the number of vertices. For an efficient computation of the mesh deformation we keep in mind that each mesh vertex is only affected by the  $(degree + 1)^3$  control points around him. From these  $(degree + 1)^3$  control points, just a few are displaced. Because the current vertex position is known, only the difference vector must be computed based on 3, so that the basis functions must be computed only for the displaced control points affecting this vertex. The closer a vertex is to the corners, the smaller is the number of displaced control points that affect this vertex and therefore the number of non-zero basis functions for computing its displacement. An example for degree 2 splines is sketched in Figure 2.

## 8. SPECIAL MODELING TASKS

Often, a bending in one direction is desired, while the curvature in the orthogonal direction should not change. This can approximately be achieved by extending the tensor volume by a relatively large amount in the direction(s) along which the curvature should not change. The example in Figure 5 has been constructed thisway, with a single point constraint and an embedding volume 10 times larger in x and y direction then in z direction (while the number of control points is still the same in each direction). Though this is only an approximation, it has very good results. For absolute precision, the factor would need to be infinite.

The large amount of deformation in Figure 5 is just for illustration. In practice, the deformations are small, since only minor corrections need to be made on finite element data. Therefore, no significant skew is introduced, which would deteriorate the simulation result.

An undo functionality can be achieved, by aligning the embedding volume with the deformation direction and making it very large in this direction. This way, the spline parameter of the start point is nearly the same as the spline parameter of the end point. The result is, that the shape of the embedded object is restored when the initial position of the start point is restored. With a persistent embedding volume this was obvious, but when each deformation is started with a new embedding volume the composition of two contrary deformations usually leads not to the initial shape. The reason is that the relative positions of the points are not the same whether the deformed volume is kept for the next operation or it is replaced by another, undeformed one.

## 9. GRAPHICS HARDWARE SUPPORT

At the SIGGRAPH conference 2000 Chua and Neumann [1] presented a hardware acceleration approach for free-form deformations. As this approach requires special hardware und software (i.e. OpenGL) extensions not available yet, we have thought of another way to use hardware for the acceleration of free-form deformation. We examined how we could use the already available hardware acceleration of spline surfaces for free-form deformation.

The OpenGL library provides so-called evaluators, which compute the position of a vertex in function of the control points, given a spline parameter value  $u$  for curves or  $(u, v)$  for surfaces. Yet, there is no way to specify a three-dimensional control mesh and three-dimensional parameter values  $(u, v, w)$ .

Based on relation 1, we can write

$$\begin{aligned} Q(u, v, w) &= \sum_{i,j,k=0}^{l,m,n} P_{i,j,k} B_i(u) B_j(v) B_k(w) \\ &= \sum_{i,j=0}^{l,m} \mathcal{P}_{i,j}(w) B_i(u) B_j(v) \end{aligned} \quad (6)$$

where

$$\mathcal{P}_{i,j}(w) = \sum_{k=0}^n P_{i,j,k} B_k(w) \quad (7)$$

$\mathcal{P}_{i,j}$  can be thought of as control points of a surface so that the function of this surface at parameter  $(u, v)$  evaluates to the desired vertex. Expression (7) is computed by the CPU,

the results for all  $(i, j)$  are fed into the graphics hardware in order to evaluate (6). For each vertex with different spline parameter  $w$  the  $\mathcal{P}_{i,j}$  must be recomputed and fed into the graphics hardware.

A problem is that the OpenGL library does cope with Bézier splines only. As a solution, the B-spline volume could be converted into a set of aligned Bézier volumes by introducing new control points *after* the control points are displaced. Then, the Bézier volume each vertex belongs to must be determined. To do this efficiently, the vertices must be in a spatial order. Unfortunately, with finite element meshes this is usually not the case. Further, in many applications, like in *CrashViewer*, it is not enough to draw the deformed object, but the new vertex coordinates must be read back from the hardware for updating the scenegraph and for writing them to file.

Through the optimizations described in section 7 we already have achieved interactive frame rates (better then 3 fps) for deformation regions of 10000 vertices with spline degree 3 on regular workstations like Silicon Graphics Octane with 250 MHz. Therefore, hardware acceleration for FFD was not implemented in the *CrashViewer* context, but the described hardware acceleration might be useful in other environments.

When modifying a car component, the whole scene needs to be refreshed in order to correctly display the modifications. In our case this means that eventually hundreds of car components need to be redrawn if one component is modified.

If the workstation provides a sufficiently fast transfer rate between graphics memory and main memory, the scene rendered without the object under modification can be saved as an image plus z-buffer into memory. Then, several modifications of the same object can be done without redrawing the whole scene. Instead, the saved image and z-buffer is fetched from memory each time and the modified object is rendered on top of it. This approach decouples the frame rate from the scene complexity.

## 10. USER INTERFACE

Any good user interface should be kept as simple and intuitive as possible. Usually the actual coordinates are not known, the user just knows how the object should look like. Though, the user wants to achieve a precise modeling.

A widely-used approach are manipulators, like the ones that come with SGI's OpenInventor library [9]. These manipulators give some information about what actions are possible on the selected object. However, through additional geometry they may confuse the user and obscure parts of the object. Furthermore, their exact positioning is hard to achieve with the mouse, the user's intentions are only approximated.

For the reasons above we preferred another tool for object manipulation. In the real world, a hammer is a widely-used tool for the deformation of a solid yet deformable object. Based on this idea, we implemented a rubberhammer-like functionality with various extensions.

When a plate is hit by a hammer, it gets a bump. This is equivalent with some displacement perpendicular to the surface. Therefore, we take the normal as displacement direction by default. Sometimes this is not desired, for example on corrugations, so that we provide the possibility of picking another point on the same or on another object for

using that normal as deformation direction. Sometimes it is necessary to stretch or shrink a surface, then a direction within that surface is obtained, either by projection of the current deformation direction onto that surface, or by picking two points of the surface. To inform the user about the current deformation direction, we optionally place a bi-colored, three-dimensional arrow near the cursor. Besides "pushing", the user can also "pull" the surface.

The amount of displacement is easily determined. The user hits different keys, like key '1' to key '0' in order to move the point under the cursor by a fixed amount like 1 to 10 millimeters. This is fast and precise, the user can quickly employ several hits to change the shape as desired. Before changing the shape, it is useful to save the old vertex positions so that several undo levels are possible.

Perforations between parts are a frequent inconsistency when meshing CAD data to obtain finite element meshes. Figure 4 shows how a perforation is removed with just one displacement operation.

## 11. CONCLUSIONS

In this paper we presented a practical application of free-form deformation with direct manipulation. We pointed out problems of previous approaches and we found solutions for this problems. The algorithms have been optimized for computational efficiency. By using a new embedding volume for each deformation, the deformation behaviour has become more uniform and therefore more intuitive.

When resolving inconsistencies of finite element meshes, relatively small deformations are performed. Therefore, some problems with FFD like self-intersection [6], element skew and distortion do not appear in this context.

A fast and simple user interface has been designed and already used with success by users being engineers from the automotive industry. The integration into a daily-used tool proved the practicability of the results.

Through changing the influence radius, i.e. the embedding volume size, the user has good control over his operations. He can modify the full range from a single vertex up to all objects at once. The alternative methods for penetration and perforation removal described in [5] provide a supplementary facility for precisely aligning objects to each other.

Though we implemented the presented features for the deformation of car components represented by finite elements, most results can be used for deformation of arbitrary objects. We think that the approach can be adapted for the handling of voxel data, too, leading to some kind of warp like it is often performed on images [13, 19]. We plan to further examine the application to voxel data, because volumetric data sets also have increasing importance in the automotive industry.

In the near future, we plan to develop a mostly automatic removal of inconsistencies from finite element meshes. Iterative application of direct manipulation seems a promising approach.

## 12. REFERENCES

- [1] Clint Chua and Ulrich Neumann. Hardware-Accelerated Free-Form Deformation. In *Proceedings of SIGGRAPH*, 2000.
- [2] Sabine Coquillart. Extended Free-Form Deformation: A Sculpting Tool for 3D Geometric Modelling. In *Proceedings of SIGGRAPH*, 1990.
- [3] G. Farin and D. Jung. Linear precision of rational Bezier curves. *Computer Aided Geometric Design*, 12, 4 1995.
- [4] Gerald Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, 4th edition, 1997.
- [5] Norbert Frisch, Dirc Rose, Ove Sommer, and Thomas Ertl. Pre-processing of Car Geometry Data for Crash Simulation and Visualization. In *Proceedings of WSCG'01*, pages 25–32, 2001.
- [6] James E. Gain and Neil A. Dodgson. Preventing self-intersection under free-form deformation. In *IEEE Transactions on Visualization and Computer Graphics Vol. 7, No. 4*, pages pp. 289–298, 2001.
- [7] William M. Hsu, John F. Hughes, and Henry Kaufman. Direct Manipulation of Free-Form Deformations. In *Proceedings of SIGGRAPH*, 1992.
- [8] Shi-Min Hu, Hui Zhang, Chew-Lan Tai, and Jia Gua. Direct manipulation of FFD: efficient explicit solutions and decomposable multiple point constraints. *The Visual Computer*, August 2001. Springer.
- [9] OpenInventor Architecture Group/Silicon Graphics Inc. *The Inventor Mentor*. Addison-Wesley, 1994. <http://www.sgi.com/software/inventor/>.
- [10] Leif Kobbelt, Thilo Bareuther, and Hans-Peter Seidel. Multiresolution Shape Deformations for Meshes with Dynamic Vertex Connectivity. In *Proceedings of EuroGraphics*, 2000.
- [11] Leif Kobbelt, Sven Campagna, Jens Vorsatz, and Hans-Peter Seidel. Interactive Multi-Resolution Modeling on Arbitrary Meshes. In *Proceedings of SIGGRAPH*, 1998.
- [12] Leif Kobbelt et al. Geometric Modeling Based on Polygonal Meshes. In *EuroGraphics tutorial*, 2000.
- [13] Apostolos Leros, Chase D. Garfinkle, and Marc Levoy. Feature-based volume metamorphosis. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 449–456. ACM Press, 1995.
- [14] 3Dlabs Inc. Ltd. OpenGL 2.0 white papers. <http://www.3d labs.com/support/developer/ogl2>.
- [15] Robert A. Noble and Gordon J Clapworthy. Direct Manipulation of Surfaces using NURBS-Based Free-Form Deformations. In *Proceedings of the International Conference on Information Visualization*, 1999.
- [16] Les Piegl and Wayne Tiller. *The NURBS Book*. Springer, 2nd edition, 1997.
- [17] Thomas W. Sederberg and Scott R. Parry. Free-Form Deformation of Solid Geometric Models. *Computer Graphics*, August 1986.
- [18] Ove Sommer and Thomas Ertl. Geometry and Rendering Optimization for the Interactive Visualization of Crash-Worthiness Simulations. In *Proceedings of the Visual Data Exploration and Analysis Conference in IT&T/SPIE Electronic Imaging*, pages 124–134, January 2000.
- [19] George Wolberg. *Digital Image Warping*. IEEE Computer Society, 1990.

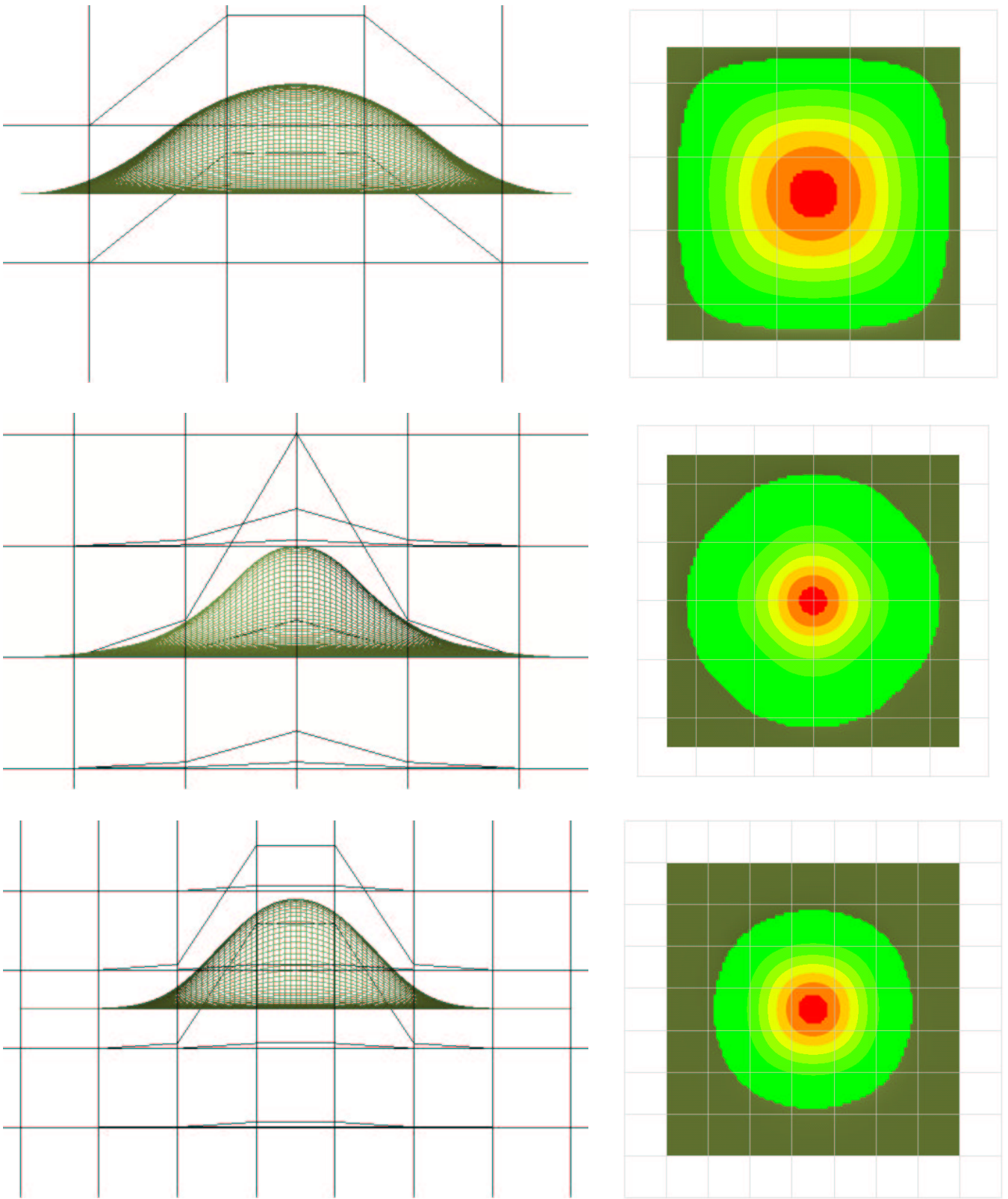


Figure 3: Comparison between different degrees and control point arrays. Up: second degree, 6 control points per dimension. Middle: second degree, 7 control points. Down: third degree, 10 control points. Left side shows a side view, right side shows the color mapped displacement characteristics for displacements larger than 1%.

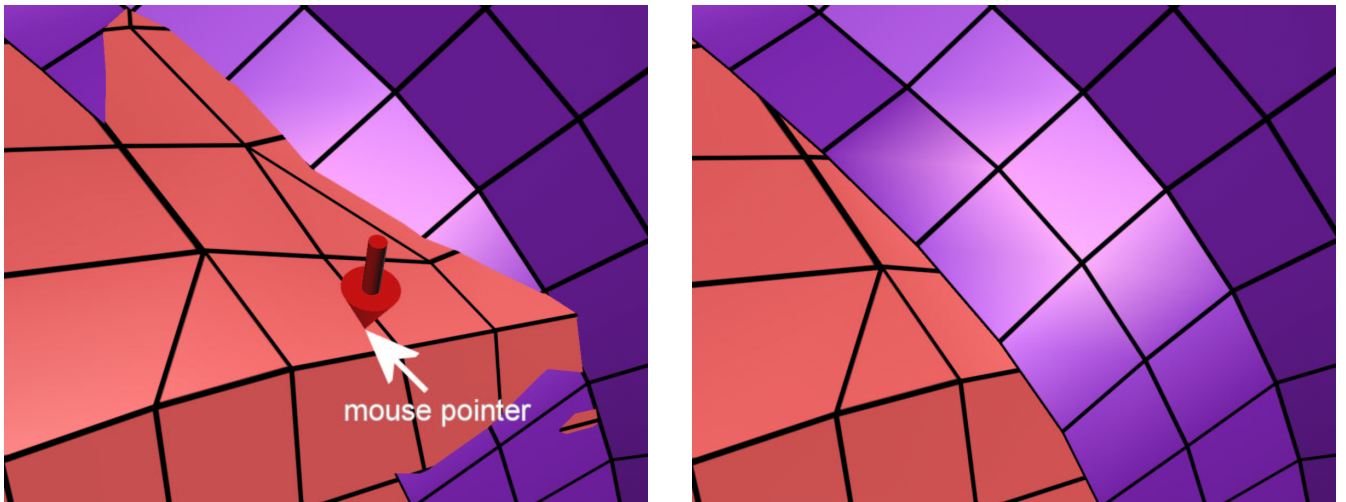


Figure 4: Left: Perforation between two finite element meshes. The red 3D-Pointer shows the deformation direction, which is automatically determined in this example based on the local surface normal and the viewing direction. Right: Perforation removed using a single-point-constrained DMFFD operation.

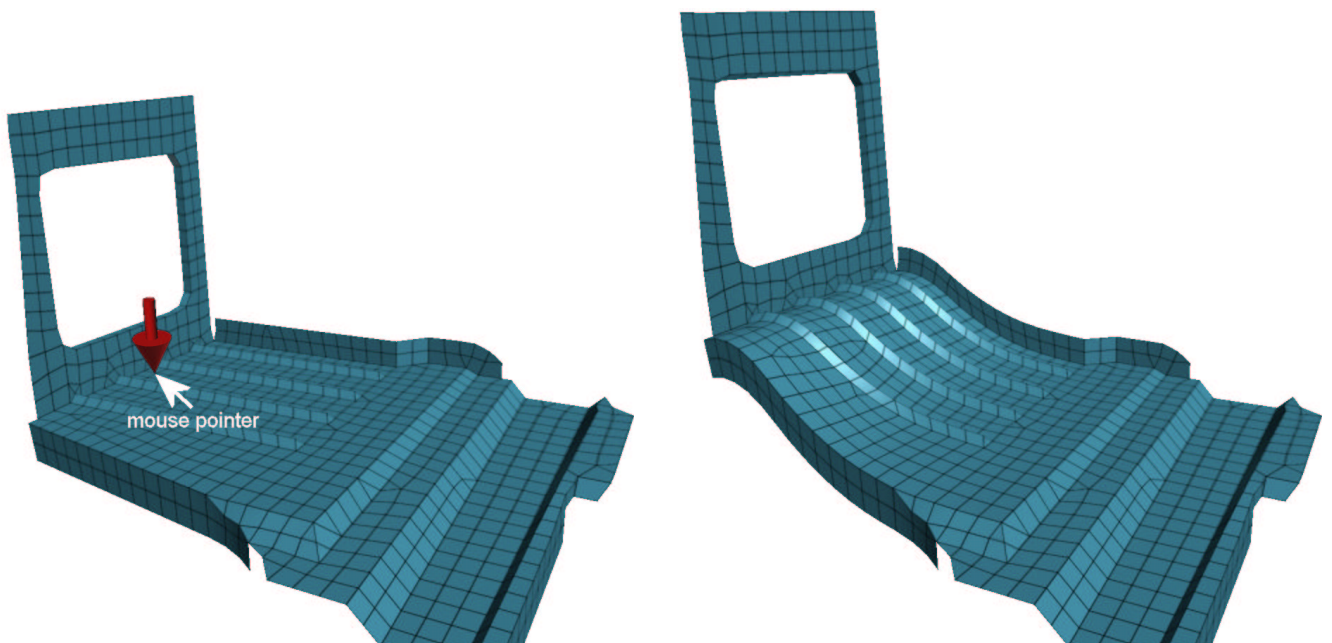


Figure 5: Shaping a car component by means of a single-point constraint. Local details like corrugations are still present. Due to a volume very large in two dimensions (x and y axes directions), the component was deformed just along the third dimension (z axis direction).