

Interactive manipulation of voxel volumes with free-formed voxel tools

Jörg Ayasse, Heinrich Müller

University of Dortmund
Faculty of Computer Science, LS 7
D-44221 Dortmund

Email: ayasse@ls7.cs.uni-dortmund.de

Abstract

Manipulation of high-precision voxel volume at interactive speed is still not immediately possible by known algorithms on today's workstation hardware. We propose to decompose a voxel-based modeling system into layers. The concept opens the possibility to apply less precise but fast algorithms of manipulation and rendering, but also to produce voxel-models at the higher precision possibly required by the application. With this concept in the background, we present a fast, though slightly imprecise algorithm for the problem of updating a voxel workpiece touched by a voxel tool interactively moved by the user along a path with six degrees of freedom. On an SGI Octane MXE, interactive speed is achieved for workpieces up to 400^3 voxels and tools up to 80^3 voxels.

1 Introduction

In voxel-based modeling as we treat it in the following, a geometric model is represented by 1-labeled 3D-cells of a regular grid which fills a box-shaped spatial region comprehending the model. The 3D-cells not belonging to the model are labeled by 0. The labeled grid can be represented by a three-dimensional binary array where the indices correspond to canonical integer coordinates of the grid cells. This representation of geometry can be seen as 3-D extension of binary images to space. The elements of the binary image are the well-known pixels, in the 3-D version they are called "voxels".

On binary images drawing tools can be applied which set or erase the color of pixels. Of course the same is possible in space. Figure 1 gives an illustration with a ball-shaped tool which is moved along a line. On the left side, the tool traverses a cube-shaped model and removes material by setting

all those voxels to 0 which are touched on its path. On the right side, the tool traverses the empty space, and applies material to all those voxels touched on its path by setting them to 1.

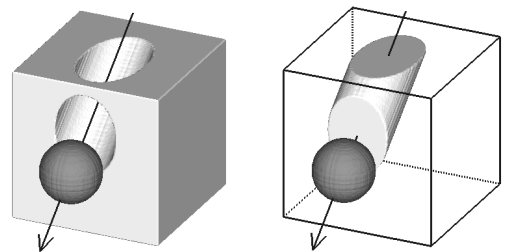


Figure 1: Removal (left) and application (right) of material by a ball moved along a straight-line path in a voxelized space.

There already exists a series of work on this topic. Meagher [12] has introduced an octree-based modeling approach, as a compressed version of voxel based modeling. This approach has been worked out and extended later by [4]. Hui [8] presented methods for simulating three- and five-axes milling processes with a dixel model, which can be understood as a run-length compressed voxel-model. A fast solution for simulating three axis milling with a dixel model on a digital height field is described by Friedhoff et al. [6] and Müller et al. [14]. Galyean and Hughes [7] and Krause and Lüddemann [9, 11] use full voxel models which are manipulated by tools. While Galyean and Hughes use voxelized tools, in the latter work predefined geometric tools are used to remove material. Raviv and Elber [15] use zero sets of scalar trivariate functions for representing the workpiece and the tool in a sculpting process. An approach of model deformation by moving material in the voxel space was

suggested by Takai et al. [16]. A quite recent survey is given in a book on volume graphics edited by Chen et al. [1].

While for binary images the drawing operations can be performed with interactive speed on today's computers, manipulation at interactive speed in the 3D-case is still not immediately possible for high-precision models. In this contribution we focus on this problem and present an approach which makes voxel-based modeling more practical in 3D, too, on standard workstations. It demonstrates that today's and future processor speed and memory size make voxel-based solid modeling to an attractive alternative to other representations of solid geometry.

In order to cope with the problem we propose to decompose an interactive voxel-oriented modeling system into layers. The basic purpose of the layers is to separate the high precision model representation from a model representation for interactive manipulation. This concept opens the possibility to apply less precise but fast algorithms of model manipulation and rendering which meet interactive requirements.

With this concept in the background, we present a fast though slightly imprecise algorithm for the problem of updating the voxels of the workpiece which are touched by a tool on a path provided interactively by the user, and of displaying the result of manipulation on-line. The workpiece and the tool are of arbitrary shape, and both are in voxel-representation. Voxelized tools open the possibility to model arbitrary tools with the system itself, starting with a small set of standard tools. In fact, every modeled workpiece can take over the role of the tool. The tool moves along a path with six degrees of freedom. Updating means that an affected workpiece voxel gets or loses material, depending on the tool mode which can be "applying" or "removing".

The algorithm increases the resolution of the voxel spaces which can be handled interactively by the combination of three ingredients:

- (1) reduction of the set of all tool voxels to a small set of affecting voxels,
 - (2) decomposition of the motion of an affecting voxel into small motions over which the voxel is replaced with a hull,
 - (3) rendering by an incremental marching cubes modification on a reduced filtered grid.
- (2) and (3) may cause a slight deviation of the vi-

sualized model from the correct model. This small error, however, is acceptable in many applications of interactive modeling, and, due to the layer concept, does not affect the precision of the resulting model.

Galyean and Hughes [7] have used voxelized tools, too. In contrast to [7] where the tool moves just translational, we allow motions with six degrees of freedom, that is arbitrary rotations are included. The sculpting model introduced by [15] allows six degrees of freedom too, but in contrast to [15] we use classical voxel models.

Section 2 describes the proposed layer concept for voxel-based modeling systems. Section 3 is devoted to the fast algorithm of model manipulation. In section 4 results of an empirical investigation of the computational performance of the algorithm are presented.

2 The Layer Concept

We propose to decompose a voxel-based modeling system into three layers: the layer of the high-precision *application model*, the layer of a usually less precise *interaction model*, and the layer of *rendering*.

The interface between the application model layer and the interaction model layer is given by the sequence of interactive modeling operations applied by the user. The sequence of operations is logged on-line. The result is an exact description of the modeled shape because the sequence in particular represents the motion of the entire tool, independent of the voxel model. By applying the recorded sequence to a voxel representation of a resolution which meets the requirements of the application, a voxel model of any desired precision can be generated.

If the initial model is given as application model, the reduced interaction model has first to be initialized.

The calculation of the high-precision model can be performed completely off-line from the achieved symbolic representation after the modeling session. Another possibility is to split off a separate process of computation during the session. In a multi-processor environment, this process may be executed on a processor different from that responsible for interaction. In a single-processor environment, idle times occurring frequently during interaction

may be used by the second process.

The logging of the executed operation may also be used for implementing an undo-functionality. For this purpose, in addition the current state of the voxel model has to be stored, too. If an undo of a certain number of operations has to be performed, the two stored consecutive states are identified into which the last accepted operation falls. From the older one of the two states, the sequence of operations from this state up to the last accepted calculation are re-executed in order to get the model at that state.

The advantage of introducing the interaction layer is that the resolution of the interaction model can be adapted to the available computation power.

A difficulty with this approach is that the sequence may not be the most efficient description of the model. The reason is that the sequence of operations the user has applied might not be optimal. For example, there might have been trial and error phases in the design process which are logged, too. An interesting question is for the calculation of a reduced equivalent sequence from the given one.

The separate rendering layer allows to apply different rendering techniques for visualization. Alternatives are polygon-based rendering and direct volume rendering. For the traditional polygon-based rendering, the interface between the layer of the interaction model and the rendering layer consists in a function which extracts a polygonal representation of the surface of the voxel model.

The rendering layer also may allow an adaptation of the level of precision to the available computation power of the rendering engine. In section 3.3 a simple example is given for surface-oriented rendering.

3 The Algorithm of Model Manipulation

The input of the algorithm consists of a voxel workpiece, a voxel tool, and a path with six degrees of freedom. Its output is a continuous update of voxels of the workpiece touched by the tool, under simultaneous visualization of the result. The tool path is given by a sequence of discrete locations $\mathbf{l} = (x, y, z, \phi_x, \phi_y, \phi_z)$ which are interpolated e.g. linearly. (x, y, z) is the location of a reference point of the tool, and $\phi_x, \phi_y, \phi_z \in [0, 2\pi)$ are the rotations around the x -, y -, and z -axis of a reference coordinate system centered at the ref-

1. Calculate the display list of triangles representing the surface of the given workpiece;
2. **Repeat**
3. get the next location P_{i+1} of the tool path;
4. decompose the motion from the previous location P_i into a sequence of small motions;
5. **for** every small motion **do**
6. determine the set of affecting tool voxels;
7. **for** every affecting tool voxel **do**
8. determine the workpiece voxels affected by the tool voxel and update them;
9. update the display list of triangles and redraw.

Figure 2: A summary of the algorithm of manipulation of a voxel workpiece with a free-formed voxel tool.

erence point. The rotations have to be performed in order to bring the tool into the described location. Updating means to replace a voxel value by 1 in the material application mode, and by 0 in the case of the material removal mode. Visualization is performed by extracting a triangular representation of the surface of the workpiece and sending it to a standard polygon rendering engine.

Figure 2 summarizes the main steps of the algorithm. The algorithm processes the sequence of discrete locations iteratively. Assume that it has already processed the motion up to location \mathbf{l} . In order to continue, it gets the next location \mathbf{l}' from the tool path. The motion between \mathbf{l} and \mathbf{l}' is decomposed into so-called *small motions*. A small motion is defined by two consecutive locations of a sequence of sampling locations from the interpolated motion between \mathbf{l} and \mathbf{l}' . The small motions have to be so short that an approximation of them performed later in the algorithm does guarantee a reasonably small error.

Next, for every small motion, the algorithm determines a set of so-called *affecting voxels* of the tool. The background of this step is the observation that many voxels of the tool never come in contact with the workpiece during motion. In fact, only voxels on the boundary of the tool can touch the workpiece, and even many of them are excluded

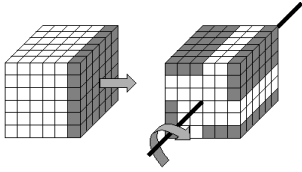


Figure 3: A cubic tool moving to the right (left image), and rotating around an axis (right image), respectively. The grey color indicates the probably affecting voxels.

because they move in the shadow of other voxels. Figure 3 depicts a translational and a rotational motion for which the non-affecting voxels are drawn in white. The calculation of the set of affecting voxels is described in section 3.1.

The affecting voxels are processed individually. For each of them, the workpiece voxels affected by it are determined and updated. These *affected voxels* are calculated as those which intersect a *motion hull* which gives an upper bound of the region which can be touched by the affecting voxel during execution of the small motion. The details are described in section 3.2.

After all affecting voxels have been processed, the surface mesh used for visualization is updated. The calculation of the surface mesh and the efficient local update according to the modifications caused by the affecting voxels during a small motion are described in section 3.3.

3.1 Calculation of the affecting voxels

The set of affecting voxels is calculated as union of some precalculated sets of affecting voxels for the given tool. Each precalculated set belongs to one of twelve elementary motions:

- three orthogonal translations in positive direction
- three orthogonal translations in negative direction
- three orthogonal rotations in positive direction
- three orthogonal rotations in negative direction.

For a given small motion of the tool defined by two consecutive locations \mathbf{l} and \mathbf{l}' , the corresponding elementary motions are obtained by considering the sign of the difference $\Delta \mathbf{l} := \mathbf{l}' - \mathbf{l}$ where the calculation for the angles is performed modulo 2π . The three orthogonal translational motions cor-

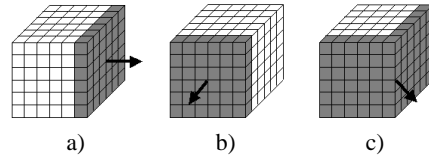


Figure 4: The set of possibly affecting voxels (drawn in grey) of a diagonal translation (c), as union of the precalculated sets of affecting voxels of its two elementary motion components (a,b).

respond to the first three entries of $\Delta \mathbf{l}$, the three orthogonal rotations to the second three entries. The sign of an entry defines the direction of the motion. If an entry is 0, the corresponding elementary motion is not a component of the motion. The union of the precalculated sets of the detected elementary motions defines the set of affecting voxels of the motion.

Figure 4 shows the set of affecting voxels of a diagonal translation which is obtained as union of the precalculated sets of affecting voxels of the two orthogonal translations defining the corresponding elementary motions.

Analogously, figure 5 depicts the set of affecting voxels of a rotation of a cube around a semi-diagonally located axis. A closer look shows that the resulting set is in fact a superset of those voxels which really come in contact with the workpiece. The combined motion consists of a simultaneous execution of the two elementary rotations. All affecting voxels of the first elementary motion belong to the calculated set, although they are not all affecting in the semi-diagonal rotation. In general, the calculated set of affecting voxels is a superset of the affecting voxel sets for all concrete motions which are obtained by combining the detected elementary motions at any speed.

The correctness of the observation that the calculated sets are a superset of all combined motions can be seen as follows. The precise set of affecting voxels are those surface voxels whose speed vector is directed to the outside of the tool volume. The speed vector of the combined motion is the sum of the speed vectors of its component motions. The orientation of the sum with respect to the surface depends on the length of the vectors, that is the speed. Because potentially each of these vectors can dominate the sum, a voxel has to be classified as affecting

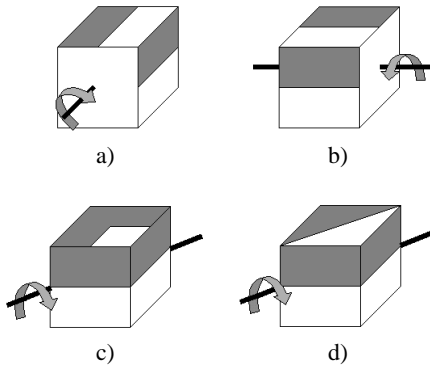


Figure 5: A set of affecting voxels of a rotation around a semi-diagonal axis (c), as union of two elementary motions (a,b), and the precise set of affecting voxels of this rotation (d).

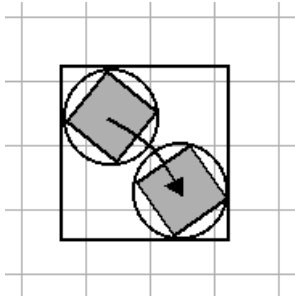


Figure 6: Calculation of the bounding box of one moving voxel.

if at least one of the vectors is directed outside. But that is just achieved by taking the union of the sets of affecting voxels of the component motions.

3.2 Calculation of the affected voxels

The set of workpiece voxels affected by voxel tool in motion is calculated approximately. Figure 6 illustrates the solution. A grey tool voxel is depicted at the first and last location of its motion path. The light grey grid in the background represents the voxels of the workpiece. All workpiece voxels touched by the tool voxel on its path have to be deleted.

The set of affected voxels determined by the algorithm consists of all those workpiece voxels intersected by the grid-line-parallel bounding box which

envelops the minimum surrounding spheres of the voxel in its start and end location. If the translational part of the motion is straight-line, the tool voxel stays completely inside the box during its motion. Thus in this case the algorithm yields a superset of the really affected voxels of the motion.

This form of approximation can be performed very quickly, but it causes an error. For the mode of material removal, the error is visible as a layer of air between the tool and the workpiece. Figure 7 shows a spheric tool traversing a cubic workpiece. A small layer of air between them is visible, but the shape of the new boundary generated by the workpiece is visually the same as without error. The thickness of the layer depends on the length of small motion, and can be kept in the order of about one workpiece voxel for very small motions. Larger small motions cause a thicker layer, but have the advantage of a higher calculation speed.

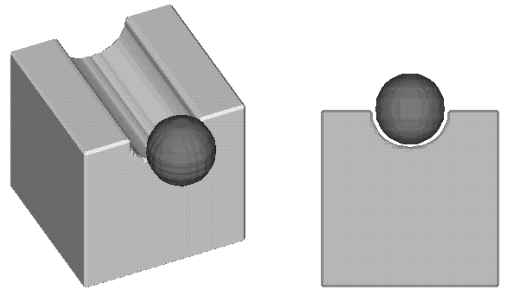


Figure 7: Visualization of the error caused by the approximate calculation of the affected voxels. It can be noticed as small layer of air between the tool and the workpiece.

3.3 Calculation of the surface mesh

The surface mesh is calculated by the marching-cubes algorithm [10, 5, 13]. For surface extraction the marching-cubes algorithm scans the voxel array. In every step eight neighboring voxels defining the vertices of a cube are considered. If all eight voxels are equal to zero or equal to one, then the cube lies completely outside or inside of the geometric model. If both ones and zeros occur, the surface of the model traverses the cube. In this case a configuration of triangles is reported which describes the surface in the cube. The configuration of triangles

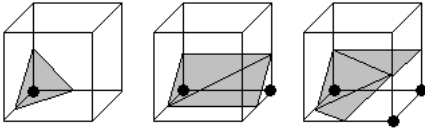


Figure 8: Three typical configurations used by the marching cubes algorithm. The vertices inside the object are indicated by thick points.

depends on the distribution of 0 and 1 on vertices of the cube. Figure 8 shows three typical examples out of the complete list of configurations.

The immediate application of the marching cubes algorithm to binary voxel arrays generally shows rasterization artifacts, see figure 9, left. We diminish this effect by simultaneous low pass filtering and reduction of sampling density. For that purpose, $3 \times 3 \times 3$ cubes of voxels are combined into a super voxel to which the sum of labels of the original voxels is assigned. Figure 10 depicts this approach in two dimensions with 3×3 squares. The smoothed surface is obtained by application of the marching cubes algorithm to the new voxel array. Vertices with a value below a given threshold are treated as outside, the others are inside. The threshold value is chosen as one half of the maximum possible label, of 27. The vertices of the triangles are located on the cube edges according to the ratio of the values at the vertices. Figure 9, right, shows the result of this standard approach when applied to the same binary voxel array. The ball is smooth approximated with even a smaller number of triangles than the left example.

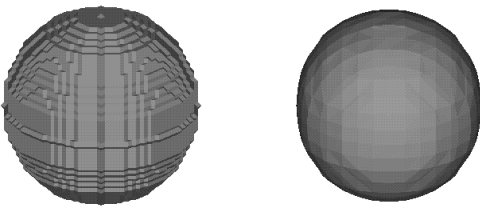


Figure 9: Visualization of a voxel ball with the marching-cubes algorithm. The left mesh is calculated immediately from the binary voxel, the right mesh after averaging cubes of $3 \times 3 \times 3$ voxels.

Instead of 3, any other not too large integer a can

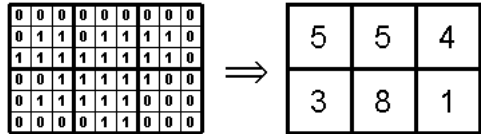


Figure 10: Filtered reduction of a binary raster array by replacing 3×3 squares with a super-square to which the sums of the original voxel values are assigned.

be used for the squares. By the choice of a , a trade-off between accurateness of visualization and rendering speed can be controlled.

At the start of the algorithm, the complete triangular surface mesh of the workpiece is calculated for the chosen level a of resolution. These triangles are stored in a long list. The triangles of every cube of the marching cubes algorithm are arranged in a connected sublist of this list. For every cube, two pointers are stored which refer to the first and the last triangle of the sublist. Because the cubes are virtual, the two pointers of a cube are stored in the voxel array, at the voxel which represents the vertex of the cube with the smallest x -, y -, and z -coordinates. Thus every entry in the reduced voxel data array now consists of an integer value and two pointers.

The additional information is used for the local update of this part of the surface mesh which is influenced by workpiece voxels affected by the affecting voxels for a small motion. The update is first performed in the original binary array, and is then transferred to the reduced voxel array. Then the marching cubes algorithm is applied to just those cubes for which the voxel value of at least one of its vertices has changed. For those cubes the update of the mesh is performed by removing the sublist of triangles belonging to this cube from the long list, if there is any. Then the list of new triangles, if there is any, is appended to the long list, and the pointers are set to its first and last triangle.

4 Performance of the Algorithm

For the experiments presented in the following an SGI Octane with 250 MHz R10000 processor, 384 MByte RAM, and MXE graphics has been used. On this machine it is possible to work interactively with

tool sizes of up to 80^3 voxels and a workpiece sizes of up to 400^3 . This means, that the user can move the tool by hand, using any input device, and gets an immediate visual update on screen.

The speed of calculation depends on the size of the tool, especially on the number of surface voxels. The kind of the motion is important, too, because it influences the number of involved voxels. Measurements with a cubic tool of size 50^3 and different motions have shown a speed of 4.4 - 25 voxels per second. That means, if the tool lies completely inside the workpiece it can be moved with a speed of 4.4 up to 25 voxel lengths per second. The measurements only include the calculating time for the process of deletion. Not included are the times of surface extraction and rendering.

For every tool the voxels involved in the twelve special movements must be precalculated. The calculation for a cubic tool with a size of 50^3 voxels needs 1.86 seconds. For a cubic tool of size 100^3 the time increases to 15.3 seconds. This is fast enough for calculating the involved voxels during program initialization.

The storage requirements depend on the size of the tool and workpiece. A workpiece of size 256^3 consists of 256^3 bits and needs 2 MByte of memory. A cubic tool of size 50^3 needs about 15 KByte memory for its voxels. In addition some amount of memory for the precalculated voxel sets is necessary. The size of the precalculated voxel sets depend on the complexity of the tool surface. For a cubic tool of size 50^3 about 2.1 MByte storage space is needed. The storage requirements for the triangular surface representation depend on the size of the voxel model, the complexity of the surface, and the filtering factor used by the marching cubes algorithm. For a cubic workpiece of 400^3 voxels and a filtering factor of 3 about 7.5 MByte of storage space is needed. The resulting storage requirements should not be a problem for today's RAM sizes.

Figure 11 - 13 give an impression of the various layout possibilities of free-formed tools. All calculations have been performed by the SGI Octane.

5 Concluding Remarks

The work presented in this contribution is part of the development of a modeling system called *Grid Based Modeler (GBM)*. The GBM is designed as an open object-oriented architecture which allows

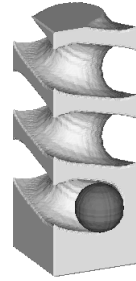


Figure 11: A spheric tool winds around a pillar. The pillar has a size of $100 \times 100 \times 260$ voxels. The tool has a diameter of 50 voxels. The calculating time is 64 seconds.

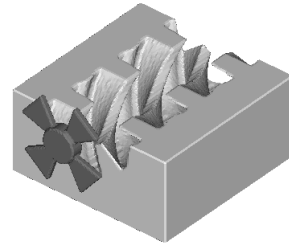


Figure 12: A propeller rotates through a block of material. The block has a size of $200 \times 200 \times 100$ voxels. The propeller consists of $100 \times 100 \times 15$ voxels. The calculating time is 76 seconds.

to integrate independent modeling modules. One emphasis of this project is to find sufficiently efficient algorithmic solutions for different tasks, like the one presented in this paper. Another emphasis is on the design of the user interface. The GBM runs currently in a CAVE-like environment [2]. The interaction is two-handed. Although at a first glance the modeling approach seems to be quite intuitive, it has turned out that with an increasing number of different possible operations it becomes difficult to offer them in a natural way to the user. Furthermore, free-hand modeling makes it often hard to achieve a desired shape, and higher order functions and constraints seem to be mandatory. The danger here, however, is that intuition is lost. Based on the GBM we are currently exploring multi-modal user interfaces in order to find satisfiable solutions. A related usability study has been performed [3], its

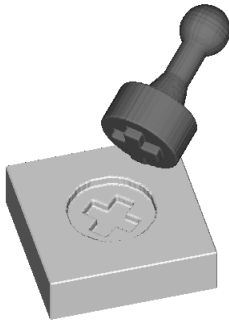


Figure 13: Stamping. The stamp has a size of $200 \times 100 \times 100$ voxels. The workpiece consists of $200 \times 200 \times 60$ voxels. It needs 3 seconds to press the stamp 20 voxels deep into the workpiece.

results are considered in the development.

An interesting extension would be the use of force-feedback devices. The voxel-oriented representation is well-suited to that purpose.

References

[1] Chen, M; Kaufman, A.E.; Yagel, R. (eds.); Volume Graphics, Springer-Verlag, London, 2000

[2] Cruz-Neira, C.; Sandin, D.J.; DeFanti, T.A.; Kenyon, R.V.; Hart, J.C.; The CAVE: Audio-Visual Experience Automatic Virtual Environment, Communications of the ACM (1992) 35:67-72

[3] Deisinger, J.; Wesche, G.; Müller, H.; Towards Immersive Modeling - Challenges and Recommendations: A Workshop Analyzing the Needs of Designers, in: Proceedings ISATA '2000, Dublin, Ireland, September 25-29, 2000

[4] Dietz, P.; A computer model based on cubic volumes for the approximation of arbitrarily sculptured volume objects (in German). Doctoral thesis, University of Dortmund, 1994

[5] Duerst, M. J.; Additional Reference to Marching Cubes, In: Computer Graphics 22(4): 72-73 (Letters)

[6] Friedhoff, J.; Müller, H.; Weinert, K.; Efficient Discrete Simulation of 3-Axis Milling, in: "Product Engineering – Research and Development in Germany". Annals of the WGP,

Volume III/2, Carl Hanser-Verlag, Munich, Germany, 1996

[7] Galyean, T.A.; Hughes, J.F.; An interactive volumetric modeling technique, Proceedings SIGGRAPH Conference 1991, 267–274

[8] Hui, K. C.; Solid Sweeping in image space - application in NC simulation, The Visual Computer (1994) 10:306-316

[9] Krause, F. L.; Lüddemann, J.; Virtual Clay Modeling, Proceedings IFIP WG 5.2, Geometric Modeling for CAD, Airlie Mai 1996, M. Pratt, London, Chapman and Hall, 1996

[10] Lorensen, E. W.; Cline, H. E.; Marching Cubes: A High Resolution 3D Surface Construction Algorithm, In: Computer Graphics, Volume 21, Number 4, July 1987

[11] Lüddemann, J.; Virtual Clay Modeling for Sculpturing by Sketching in Industrial Design (in German). Doctoral thesis Dissertation, IWF of the Technical University of Berlin 1996

[12] Meagher, D.; Geometric Modeling Using Octree Encoding. Computer Graphics and Image Processing 19; 1982

[13] Montani, C.; Scateni, R.; Scopingo, R.; A modified look-up table for implicit disambiguation of Marching Cubes, In: The Visual Computer (1994) 10:353-355

[14] Müller, H.; Albersmann, F.; Weller, F.; Zabel, A.; Efficient Direct Rendering of Digital Height Fields, Proceedings of IFIP TC5/WG5.10 and CSI International Conference on Visual Computing (ICVC'99), Goa, India, 23-26 February 1999, 1999; also available as Research Report No. 669, University of Dortmund, Germany

[15] Raviv, A.; Elber, G.; Three-dimensional freeform sculpting via zero sets of scalar trivariate functions, Computer-Aided Design 32 (2000) S. 513-526

[16] Takai, Y.; Takai, N. K.; A deformable object in the active voxel space, Proceedings of the IASTED International Conference, Computer Graphics and Imaging, October 25-27, 1999, Palm Springs, California, USA

[17] Wang W. P.; Wang, K. K.; Geometric Modeling for Swept Volume of Moving Solids, IEEE Computer Graphics Applications 6 (1986) 8-17