Fall 2008

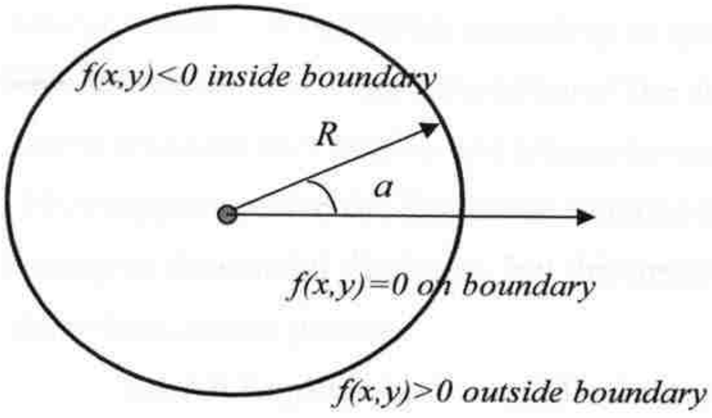
Virtual prototyping with surface reconstruction and freeform geometric modeling using level-set method

Weihan Zhang

Department: Mechanical and Aerospace Engineering

Recommended Citation

Zhang, Weihan, "Virtual prototyping with surface reconstruction and freeform geometric modeling using level-set method" (2008). *Doctoral Dissertations*. 1984.
https://scholarsmine.mst.edu/doctoral_dissertations/1984



A circle:

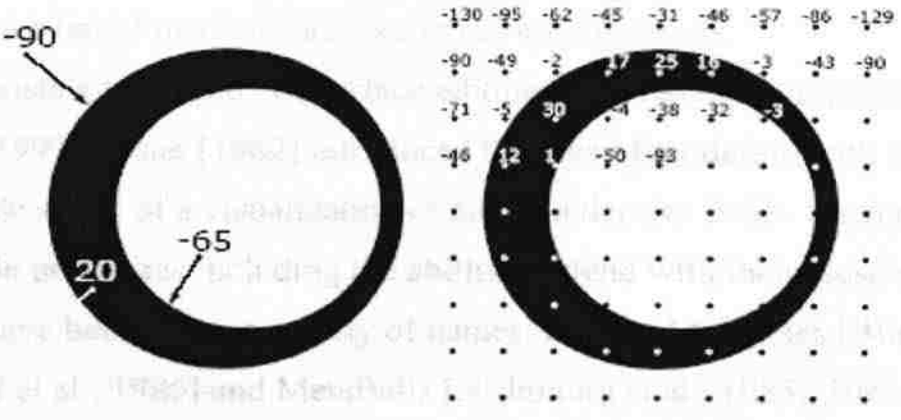
- Parametric representation:

$f(x,y)= (R\cos(a), R\sin(a)),\ a \in [0, 2PI]$

- Implicit representation:

$f(x, y)= x^2+y^2-R^2$

Figure 1.5. A Circle in the Implicit Representation



(a) (b)

Figure 1.6. Sampled Distance Field Data

The implicit surface models can be represented by the discretized distance field data where the distance information is the shortest Euler distance from the grid point to the implicit surface. The available algorithms for computing the distance field from common surface representation includes hierarchical organization and characteristic method [Jones et al., 2006]. Generally a brute force method is used to compute the distances from a grid point in the space to every boundary triangle of M and select the shortest one. To reduce the computation, the shortest distance can be calculated only to a

$$f(r) = a\left(1 - \frac{4r^6}{9R^6} + \frac{17r^4}{9R^4} - \frac{22r^2}{9R^2}\right) \tag{2}$$

where $f(0)=1, f(R)=0, f'(0)=0, f'(R)=0$.

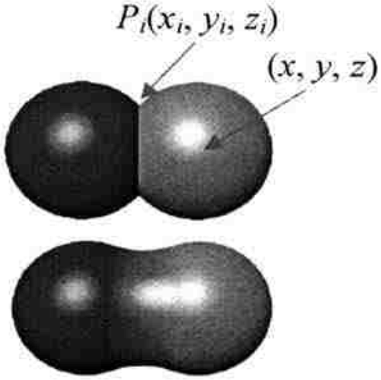


Figure 1.7. The Blobs Model

Given a point in 3D space, the implicit model has the benefit of finding the relation between the point and the surface $f(x, y, z)$. And it is easy to describe the topologically changed surface using, for example, merging, bifurcation, absorbing, etc. Different operations, such as scaling, twisting and CSG operations, are also easy to be defined on a given implicit surface model. Suppose given implicit functions $f(x, y, z) = 0$ and $g(x, y, z) = 0$, according to the definition of the implicit function, different operations on the implicit surfaces such as scale, shear, taper, twist, bend, etc. can be easily defined.

The Blobby models employ local basis functions, so they are often more intuitive to work with than algebraic surfaces [Blinn, 1982]. However, dials or sliders have to be used to adjust the position and radius of each blobby center which is an art work to arrive at the desired surface [Beier, 1993]. Bloomenthal and Wyvill [1990] developed techniques to define/manipulate the skeleton of several shapes, define/adjust the implicit function defined for each skeletal element, and define a blending function to weight the individual implicit functions. By manipulating the skeletal ellipsoids, the user can produce complex models, and the blending and offsetting operations are controlled by a procedural implicit function which permits a greater degree of localized control as compared to a simple blend of implicit primitives in which each primitive potentially has a global affect on the surface as shown in Fig. 1.8[Wyvill et al., 1999]. But this procedural implicit function is awkward to the end users.

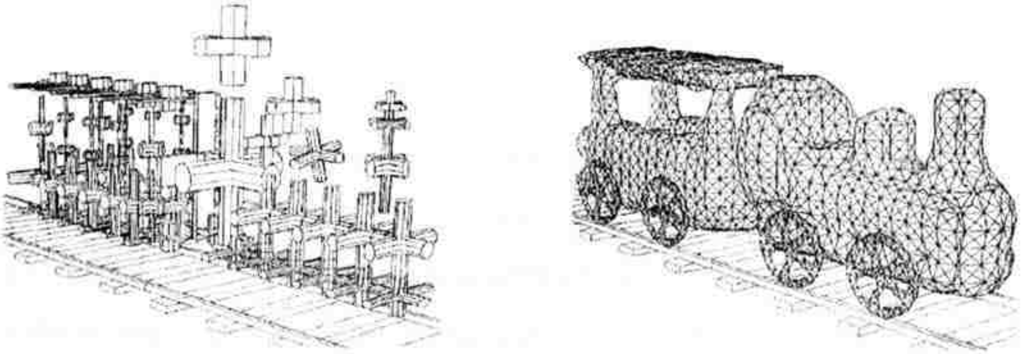


Figure 1.8. Skeletal Elements for the Train and the Surface of the Train after Blending

Wyvill et al. [1999] used tree structure to represent the set theoretical Boolean expression between solid models having half spaces as the primitives. They described techniques for performing blending, warping and Boolean operations on skeletal implicit surfaces called "*Blob Tree*". Galin et al. [1999] addressed the metamorphosis of the Blob-Tree by proposing an original technique that solves the correspondence process and creates an intermediate generic Blob-Tree model whose instances interpolate the initial and final shapes. Besides constructing the model by a larger number of elements arranged in a tree structure, the model can also be defined as single source points with one or several additional curves that control the shape of the field function. Several operations are defined, such as freeform definition curves, rotational, translational or general sweep, twist or interpolation of cross-section, etc. Users can use the splines to control the swept trajectories to generate complex shapes as shown in Fig. 1.9 [Crespin et al., 1996].

Figure 1.9. A Model Defined by Sweeping Primitives

Besides the blobby model based methods, there is another type of implicit function based modeling method called the Control point based methods. Physically based particles provide an interactive means to sample implicit surface functions whereby points on the surface are determined by heuristics, such as the use of the implicit function gradient. Witkin and Heckbert [1994] used particles to sample the implicit surface and applied simple constraint to lock the movement of particles onto a surface while the particles and the surface move. Then, those particles are moved by the user to control the implicit surface. However, it is found that the implicit surfaces are slippery when one attempts to move them using control points. Several approaches [Crossno and Angel, 1997; Rosch et al., 1996] have been proposed to enhance the original Witkin-Heckbert technique by adapting the particle distribution to the local curvature of the surface. Turk and Brien [2002] attacked this problem by using an interpolating implicit surface model and let users directly create and move the boundary constraints to change the shape of the interpolating implicit surface. This provides an intuitive control for interactive sculpting of implicit surfaces which can only accommodate a limited amount of details since at most a few thousand coefficients can be employed in real-time.

Level-set method is a set of numerical methods developed to model the implicit distance field data. Applying level-set methods in interactive geometric modeling is relative new. It started from the work of Museth et al. [2002]. The computational complexity was reduced in their follow-on work [Museth et al., 2005; Nielsen and Museth, 2006]. Museth et al. [2002] developed surface editing techniques like copy, remove and merge level-set models and automatically blend the intersection regions. Their editing operators act on surfaces that happen to have an underlying volumetric

Figure 1.10. Examples of Curve Skeletons of Different 3D Objects

1.3.3.1 Surface reconstruction from dexel model. Dexel data is view-dependent because it only records the geometric information of a 3D object from one viewing direction. In the practice of dexel-based NC simulation, researchers were only able to produce a limited number of views from certain directions for the simulation, without the generation of a surface model that can be viewed from any directions. To solve the view-dependent problem, Huang and Oliver [1995] briefly described a contour tracking technique but without detailed development of an algorithm. They visualized the boundary of the object by simply displaying sets of contours extracted from the dexel data. Konig and Groller [1998] described an algorithm to create a surface representation from dexel data for 3-axis milling simulation. But the algorithm could fail easily in the virtual sculpting process where dexel data are modified in arbitrary directions. Zhu and Lee [2005] presented a visibility sphere marching algorithm for constructing polyhedral models from dexel data for their virtual sculpting research. When the algorithm was applied to complex 3D objects, there could be some cracks and holes in the generated mesh due to topology related issues [2003]. The Marching Cube Algorithm [Lorensen and Cline, 1987] has been used to generate an approximate triangular surface from tri-dexel data [Benouamer and Michelucci, 1997] and from voxel data. But this algorithm requires huge memory storage and suffers from some ambiguity, and it can not be applied to dexel data generated in a single direction. Muller et al. [2003] implemented the point-based rendering method developed by Pfister et al. [2000] for their online sculpting system. However, it was difficult to interface the sculpted models with CAD/CAM/CAE systems for further design and analysis.

contour. For example, the dexel spaces between points 4 and 5, between points 10 and 11, and between points 16 and 17 in Fig. 2.1(b) are an open set. The connections of an open set of dexel spaces are: filling the top or the bottom dexel space, depending on which is covered by a dexel above or beneath, and connecting the boundary dexel points on the same side of the dexel spaces accordingly.

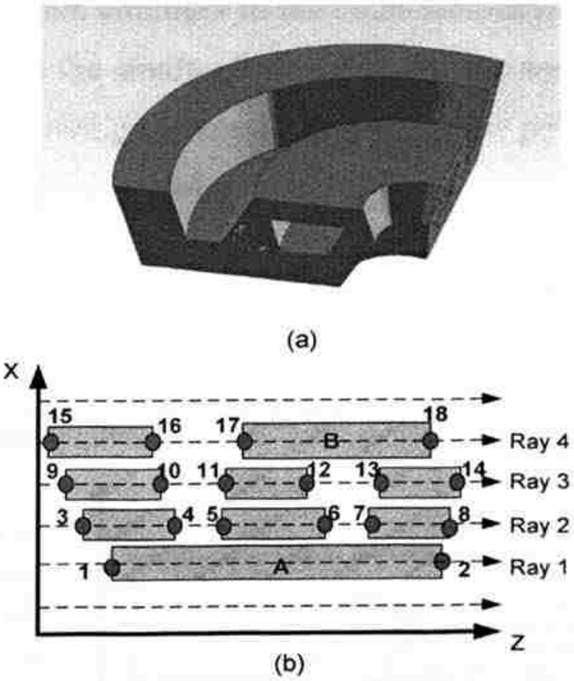
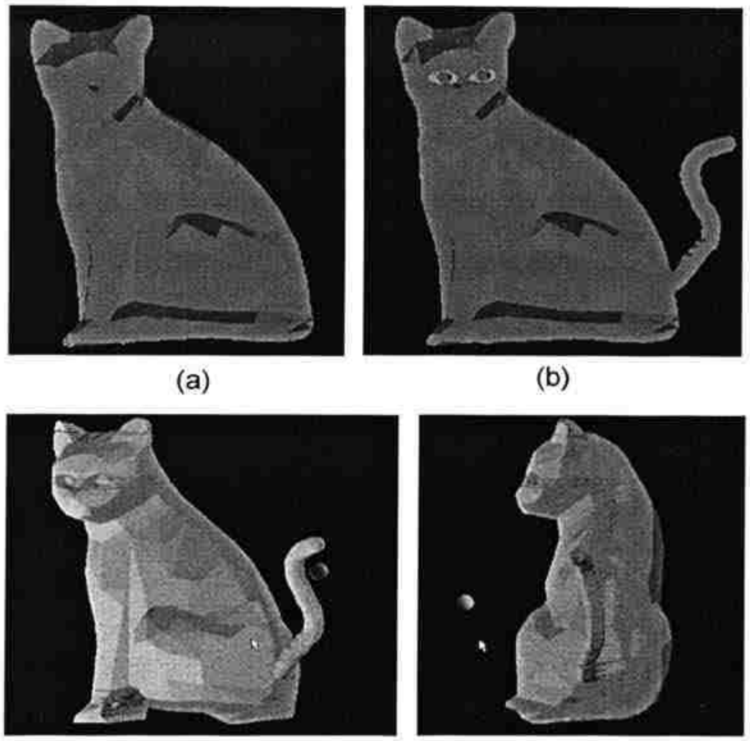


Figure 2.1. Example of the Contour Generation Algorithm. (a) 3D Model (b) One Slice of the 3D Model on XZ Plane

By using the grouping criterion, a four-step contour generation algorithm has been developed. The algorithm first categorizes the dexels on two adjacent rays into different groups by using a "grouping" criterion. The dexel points in the same group are connected using a set of rules to form sub-boundaries. After checking the connections among all the dexel points on one slice, a connection table is created and used to obtain the points of connection in a counterclockwise sequence for every contour. Finally, the contours on all the parallel slices are tiled to obtain triangular facets of the boundary surface of the 3D



(a) (b)

(c) (d)

Figure 2.3. Modeling Example of a Cat Model. (a) The Imported Cat Model Created from a CAD System (b) Eyes and Tails Created by Virtual Sculpting (c) and (d) Viewing the Modified Cat Model in Different Directions

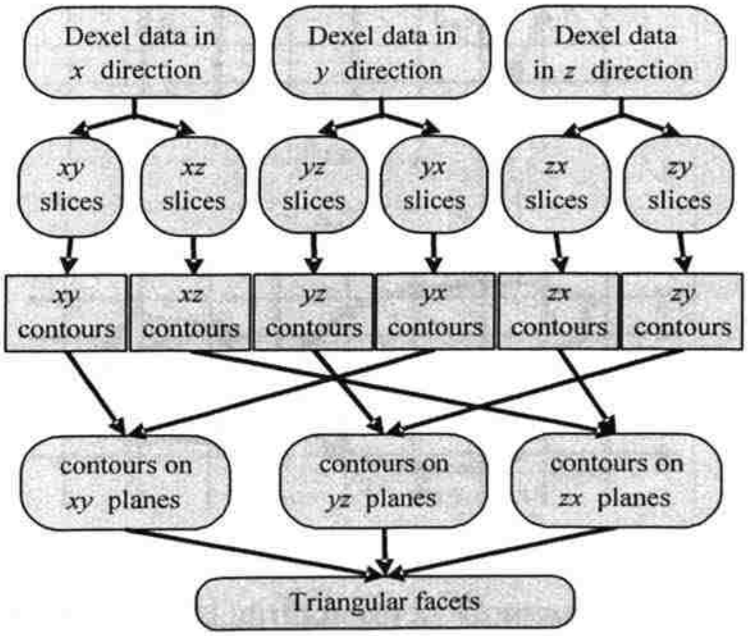


Figure 2.4. Proposed Method of Surface Reconstruction from Triple-Dexel Data

than the reconstructed surface from the single-dexel data when using the same ray resolution. In addition, to benchmark the performance of the developed method, numerical experiments are conducted to compare using triple-dexel data vs. voxel data in terms of the surface reconstruction time and the associated surface error. The test result shows that, under the same resolution, the surface reconstructed from the triple-dexel data has a smaller surface error in comparison with the surface reconstructed from the voxel data. This is because the triple-dexel based method utilizes actual positions of the intersection points between rays and the object's boundary surface as the vertices of the reconstructed surface model, while the voxel based method approximates the positions of these vertices by voxel interpolation.

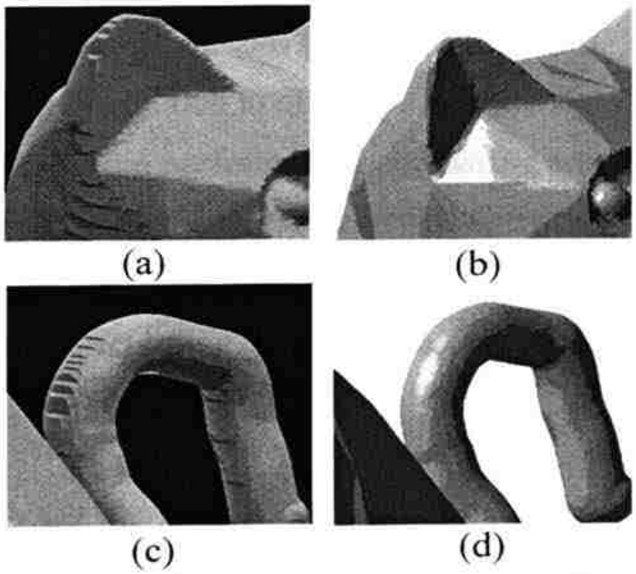


Figure 2.6. Comparisons of Reconstructed Surfaces. (a) and (c) are from Single-Dexel Data, (b) and (d) are from Triple-Dexel Data

The computation complexity of the contour generation, correspondence and combination process using triple-dexel data is O(T) or O(M^2), where M is the number of divisions along each axis. Because the complexity of the volume-based tiling algorithm is

point is an Adjacent Grid Point (AGP) if it is adjacent to any BGP. Next, the sign of the distance value of each BGP and AGP is determined. Third, the surface within each BV is approximated using triangular facets. Finally, the distance value of each BGP and AGP is calculated. A 2D illustration is given in Fig. 2.8, where the gray-colored pixels surrounding the iso-surface are the boundary pixels (i.e., 2D BVs). Each squared point is a BGP and each triangular point is an AGP.

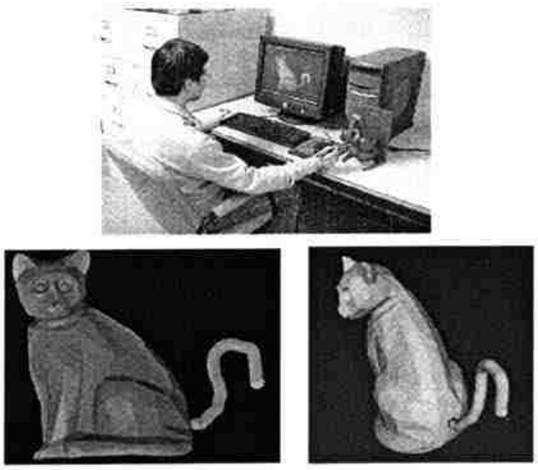


Figure 2.7. A Cat Model Generated Using the Virtual Sculpting System

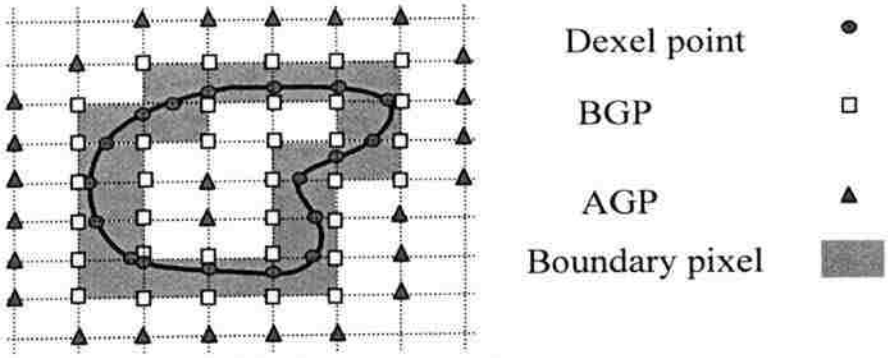


Figure 2.8. Boundary Pixels, BGP, AGP and Dexel Points

If a grid point is between two adjacent dexels along a ray, the distance value of this point is positive. Otherwise, the sign of the distance value is negative. The main idea of this step is to use the Hermite data (i.e., exact intersection points and normals) on the edges of a BV to calculate an additional point inside the BV by minimizing a quadratic function. By connecting this point with other additional points in adjacent BVs, triangular meshes can be generated with a simple patching algorithm to approximate the boundary surface. The Euclidean distance of a BGP of a BV is the shortest distance from the BGP to the local triangles formed by the additional point of this BV. The distances between this BGP and every such triangle are calculated, and the smallest value is the Euclidean distance. As illustrated in Fig. 2.9, the distance of the center grid point is d_2 because $d_2 < d_1$. Based on the same principle, to calculate the distance values of AGPs, such as point p_3 in Fig. 2.9, only triangles formed by the additional points in the adjacent BVs are considered for the distance test.

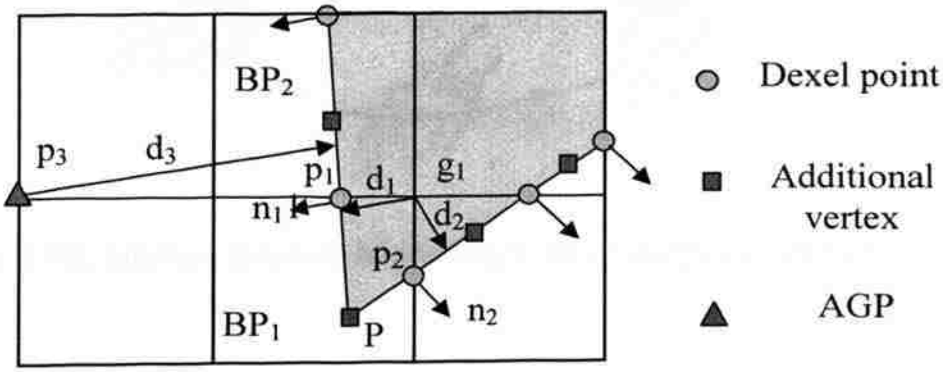


Figure 2.9. Distance Calculation for the Grid Points

2.2.2. Hand Gesture Modeling. A gesture is a form of non-verbal communication made with a part of the body such as the hand. The input of our freeform deformation framework is a series of gestures (i.e., orientations and positions of user's hand), G_i (i=0,…,n), captured from the mouse or 3D input devices such as the 6DOF tracking device. To associate user's gesture inputs with shape changes, the human gesture has been modeled by formalizing a spatial transformation matrix. Then, freeform deformative operations are defined based on the human gesture model. Finally, a mapping method is developed to build connections between the defined operations and the boundary velocity of the surface which enables the level-set method to propagate the shape as desired. The gesture G_i at time t_i is defined by a local coordinate system with origin O_i and three orthogonal directions u_i, v_i, w_i as seen in Fig. 2.10, where $u_i \times v_i = 0$, $v_i \times w_i = 0$ and $w_i \times u_i = 0$. To produce a smooth space warp from input gestures, a B-Spline interpolation has been utilized to calculate the position and orientation of the gesture in between such as G_j in between G_i and G_{i+1} in Fig. 2.10. The gesture at G_j is constructed by the linear combination of translations and rotations around the interpolated origin O_j.

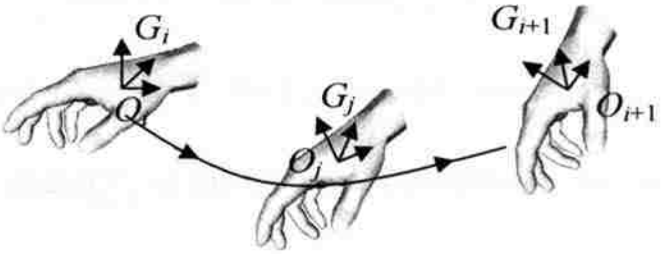


Figure 2.10. Human Gesture Modeling Using Interpolation Method

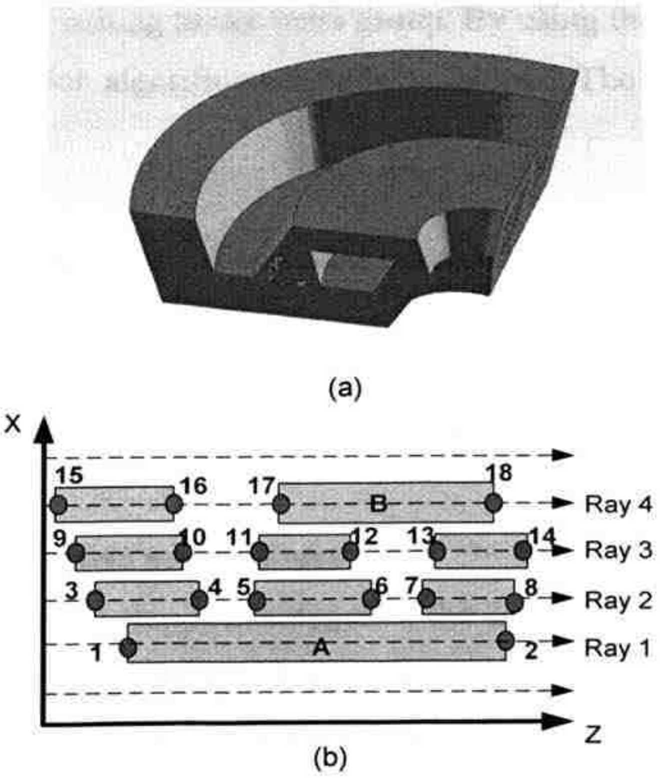
2.2.3. Shape Modeling Using Level-Set Method. Level-set models are deformable implicit surfaces where the deformation of the surface in its normal direction is controlled by a speed function in the level-set partial differential equation [Sethian, 1999]:

$$\frac{\partial F}{\partial t} = -\nabla F \cdot \overline{v} \tag{3}$$

where F(**x**,t) is the Euclidean distance function, **x** is the grid coordinates in Euclidean space R^3, $\overline{v}$ is the speed function of boundary points, ∇ is the gradient and

and 7 and between points 12 and 13 form an inner contour because the top of these overlapped dexel spaces is covered by dexel B and the bottom is covered by dexel A.

According to this observation, if a set of overlapping dexel spaces is covered by both a dexel beneath the bottom and a dexel right above the top, these dexel spaces form an inner contour and are called *a closed set*. For example, the set of dexel spaces between points 6 and 7 and between points 12 and 13 in Fig. 2(b) is a closed set. The connections of a closed set of dexel spaces to form an inner contour are: filling the top and the bottom dexel spaces, and connecting the boundary dexel points on the same side of the dexel spaces accordingly (e.g. connecting point 6 and point 12, and connecting point 7 and point 13 in Fig. 2(b)).


(a)

(b)

Figure 2. Example of the Contour Generation Algorithm. (a) 3D Model (b) One Slice of the 3D Model on XZ Plane

b) Step 2: Connect adjacent dexel points inside each group

The aim of step 2 is to generate connections between dexel points in the same group along every two adjacent rays. Suppose a group of dexels consists of R_i dexels ($D_{i,1}$, $D_{i,2}$, . . . D_{i,R_i}) on Ray *i* and R_{i+1} dexels ($D_{i+1,1}$, $D_{i+1,2}$, . . . $D_{i+1,R_{i+1}}$) on Ray *i*+1, where $R_i \geq 1$ and $R_{i+1} \geq 1$, as illustrated in Fig. 4. $D_{i,j} \rightarrow [h]$ and $D_{i,j} \rightarrow [t]$ are the head and the tail of dexel $D_{i,j}$, respectively. The first two dexel points $D_{i,1} \rightarrow [h]$ and $D_{i+1,1} \rightarrow [h]$ should be connected because they are two adjacent points on the same outer boundary. Likewise, the last two dexel points $D_{i,R_i} \rightarrow [t]$ and $D_{i+1,R_{i+1}} \rightarrow [t]$ are also connected. The points in between should be connected to the adjacent dexel points on the same ray. Thus, the rules of connections within a dexel group are:

①: Connect ($D_{i+1,1} \rightarrow [t]$, $D_{i+1,2} \rightarrow [h]$), . . . ($D_{i+1,R_{i+1}-1} \rightarrow [t]$, $D_{i+1,R_{i+1}} \rightarrow [h]$)

②: Connect ($D_{i,1} \rightarrow [t]$, $D_{i,2} \rightarrow [h]$), . . . ($D_{i,R_i-1} \rightarrow [t]$, $D_{i,R_i} \rightarrow [h]$)

③: Connect ($D_{i,1} \rightarrow [h]$, $D_{i+1,1} \rightarrow [h]$)

④: Connect ($D_{i,R_i} \rightarrow [t]$, $D_{i+1,R_{i+1}} \rightarrow [t]$)

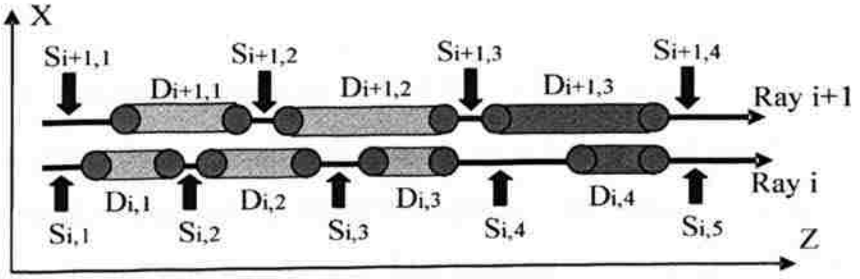


Figure 3. Grouping Process

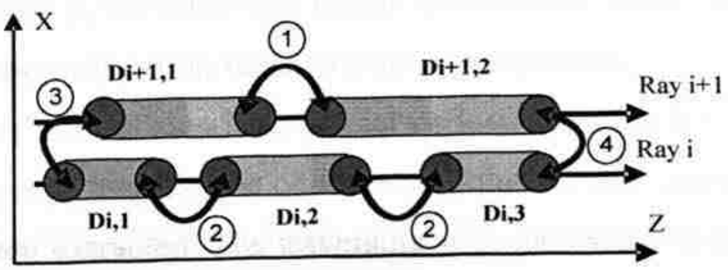


Figure 4. Contouring Algorithm

Special cases exist when one of the two adjacent rays does not intersect with the 3D object, as shown in Fig. 5. The rules of connections for these cases are:

⑤: When $R_i = 0$, connect ($D_{i+1,1}\rightarrow[h]$, $D_{i+1,1}\rightarrow[t]$)

⑥: When $R_{i+1} = 0$, connect ($D_{i,1}\rightarrow[h]$, $D_{i,1}\rightarrow[t]$)

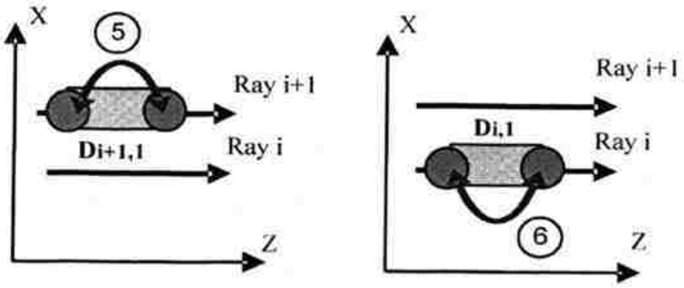


Figure 5. Special Cases of the Contouring Algorithm. (Left) When R_i=0 and (Right) When R_{i+1}=0

c) Step 3: Create a connection table to record the connections

After Step 2, each dexel point has exactly two connected dexel points, which are its adjacent vertices on the contour. In order to separate the points into different contours, a three-column connection table is created. The middle column lists the dexel points in the same sequence as they are generated and read. Their connecting points are stored in the left and right columns separately. In order to generate contours in the counterclockwise sequence, the left column is always filled first. After filling in all the connecting points in Step 2, as shown in Fig. 6, the table will be full without any empty spaces.

d) Step 4: Traverse the connection table to construct contours

The objective of the last step is to extract various contours from the connection table. The basic idea is to traverse the connecting points of one contour at a time, until all the contours have been extracted. The traversing sequence starts from the top to the bottom of the connection table. The starting point of a contour is the first unsearched

original shape in that the two neighboring objects (or holes) have been combined into one. The above problems can be solved by decreasing the distance between adjacent rays, thus, increasing the resolution of dexel data. For example, by decreasing the ray spacing, the generated contours in Figs. 8(c) and 8(f) have captured the original topologies of the models in (a) and (d), respectively.

Connected Point	Dexel Point	Connected Point
2	**1**	3
1	**2**	8
1	**3**	9
5	**4**	10
4	**5**	11
7	**6**	12
6	**7**	13
2	**8**	14
3	**9**	10
4	**10**	9
5	**11**	15
6	**12**	13
7	**13**	12
8	**14**	16
11	**15**	16
14	**16**	15

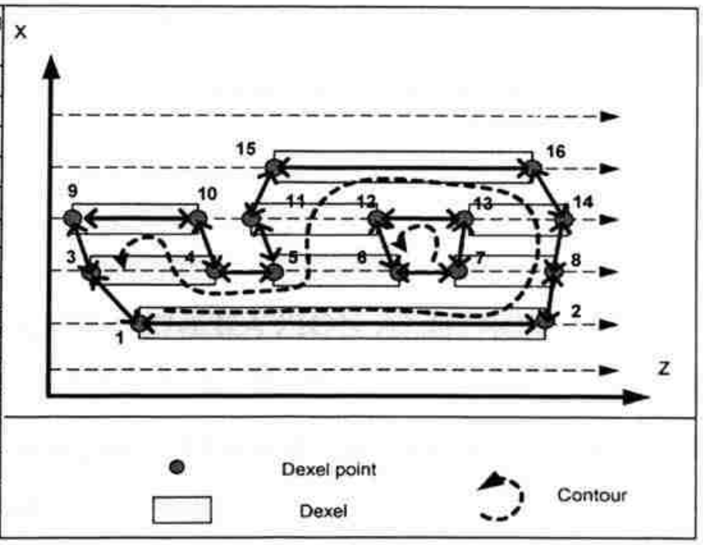


Figure 6. Traversing the Connection Table to Separate Contours

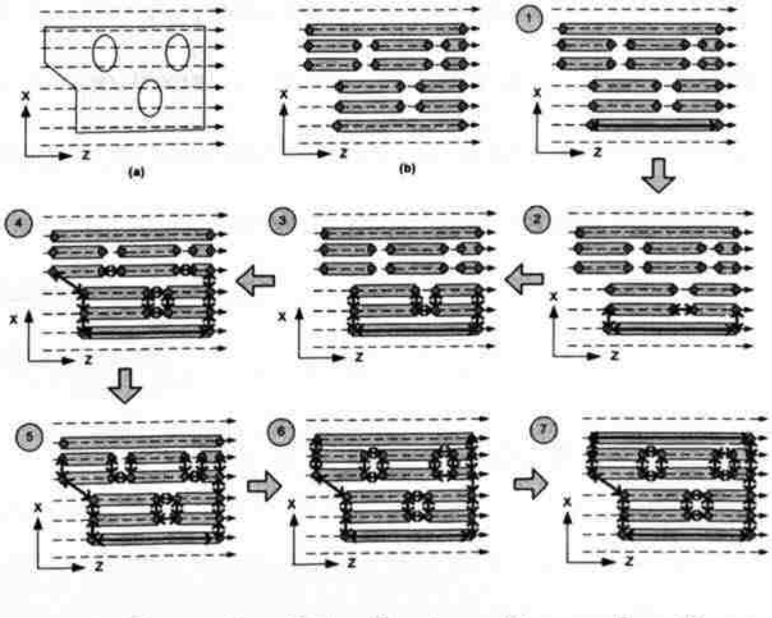


Figure 7. Example of the Contour Generation Process

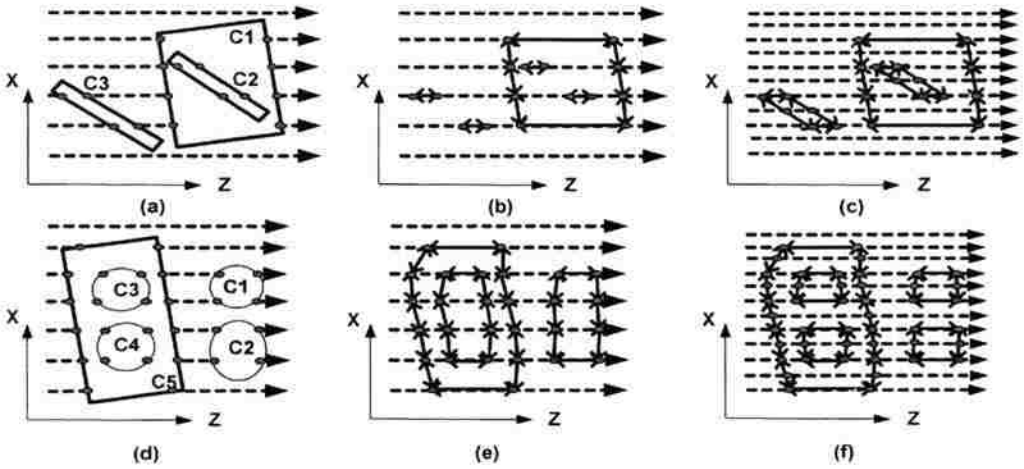


Figure 8. Discussion on the Validation of the Observations

3. ANALYSIS OF THE CONTOUR GENERATION ALGORITHM

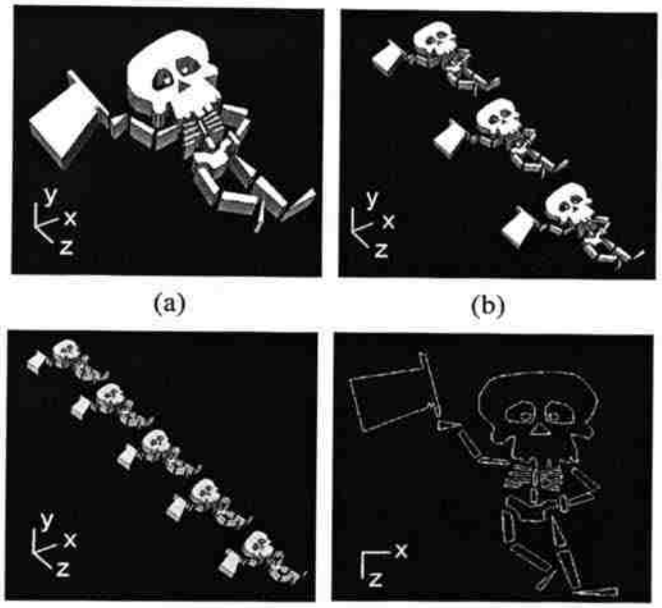
In this section, the computational complexity and storage requirement of the contour generation algorithm are analyzed. Two test cases are used to verify the computational complexity analysis.

3.1. Computational Complexity Analysis

In Step 1 of the contour generation algorithm, the dexel spaces on Ray *i* are compared to the spaces on Ray *i*+1 to separate the dexels into groups. Given n_i dexel spaces on Ray *i* and n_{i+1} dexel spaces on Ray $i+1$, the computation time of comparing these dexel spaces is proportional to $n_i \times n_{i+1}$. Suppose *N* is the number of dexel spaces on a slice, *D* is the number of dexels on the slice, d_i is the number of dexels on Ray *i*, and β is the number of rays intersecting with the object for the considered slice, representing the model's discretization resolution.

Because $n_i = d_i + 1$, thus,

$$N = \sum_{i=1}^{\beta} n_i = \sum_{i=1}^{\beta} (d_i + 1) = D + \beta \tag{1}$$



(a) (b)

(c) (d)

Figure 9. Numerical Experiments. (a)B_1, (b)B_3, (c)B_5, and the generated contour from B_1(d)

Table 1. Computation Results of the Contour Generation Algorithm

	B_1	B_2	B_3	B_4	B_5
No. of dexels (D)	45,666	91,332	136,998	182,664	228,330
Average no. of dexels per ray (α)	3.280	6.559	9.839	13.119	16.398
Contour generation time (sec)	0.161	0.386	0.700	1.116	1.687

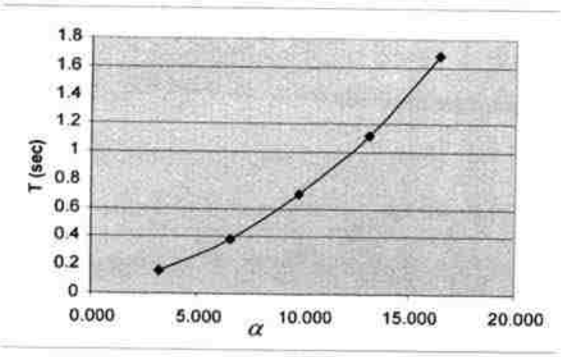


Figure 10. Contour Generation Time (T) vs. Average No. of Dexels Per Ray (α)

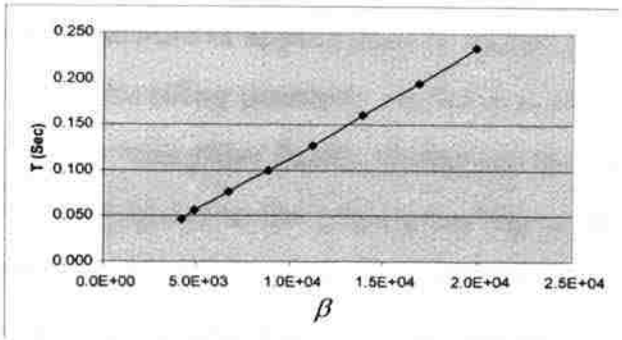


Figure 11. Contour Generation Time (T) vs. Number of Rays (β)

Table 2. Computation Time of the Contour Generation Algorithm

No. of rays (β)	No. of dexels (D)	Average no. of dexels/ray (α =D/ β)	Contour generation time (sec)
4,225	13,650	3.231	0.047
4,900	16,170	3.300	0.056
6,724	22,140	3.293	0.077
8,836	28,764	3.255	0.100
11,236	36,358	3.236	0.127
13,924	45,666	3.280	0.161
16,900	54,600	3.231	0.195
19,881	65,283	3.284	0.235

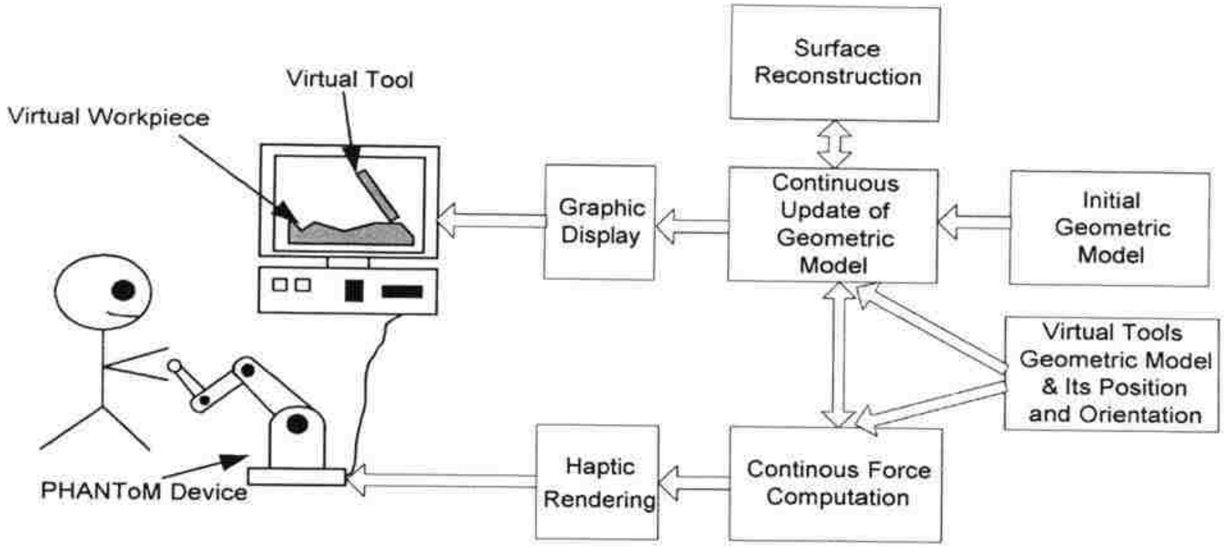


Figure 12. The Virtual Sculpting System Configuration

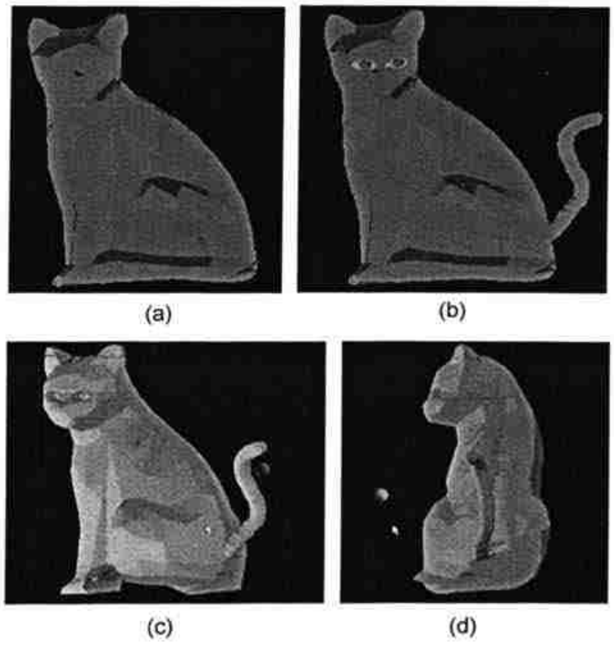


Figure 13. Modeling Examples. (a) The Imported Cat Model Created from A CAD System (b) Eyes and Tails Created by Virtual Sculpting (c) and (d) Viewing the Modified Cat Model in Different Directions

4.3. NC Machining Simulation

The NC machining simulation system has the same geometric modeling engine as the virtual sculpting system. The only difference is that in the NC machining simulation system, the cutter location file is generated by a CAD/CAM system, instead of by the designer/stylist in the case of virtual sculpting. Figure 14 shows a mouse model that is in the midst of NC machining simulation. It demonstrates that the triangular surface can be reconstructed from the dexel data interactively during the animation of simulated machining. The sculpted model can also be rotated in arbitrary angles to provide different views of the model.

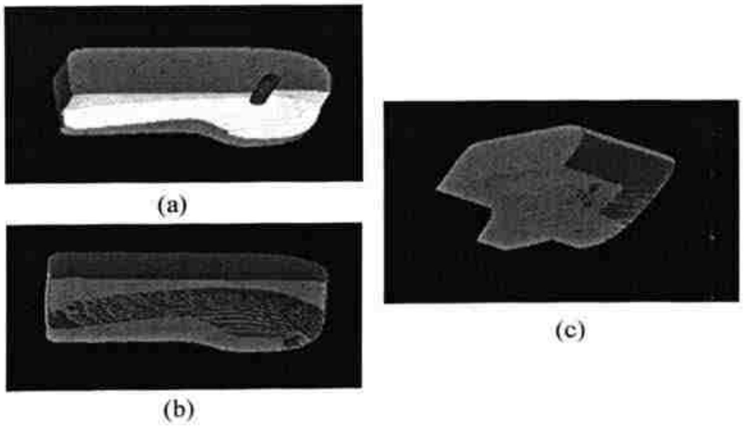


Figure 14. A Mouse in the Midst of NC Machining Simulation. (a), (b) and (c) Show the Generated Mouse Viewed from Two Different Directions after Performing Surface Reconstruction during the Machining Simulation

5. CONCLUSION

This paper has presented the development of a novel method to extract 2D contours from dexel data for the purpose of surface reconstruction for a 3D model. The surface reconstruction process solves the view-dependent problem inherent in dexel-based applications such as virtual sculpting and NC machining simulation. The dexel data are first put into different groups using a grouping criterion. Then the dexel points in the same group are connected using a set of connection rules. A connection table is created

analysis, and automated manufacturing applications. Further, the triangulated 3D model can be viewed in any directions as desired using standard routines of computer graphics software. The surface reconstruction from triple-dexel is also difficult because reconstruction methods have to overcome topological ambiguity, which is usually being dealt through grid based methods.

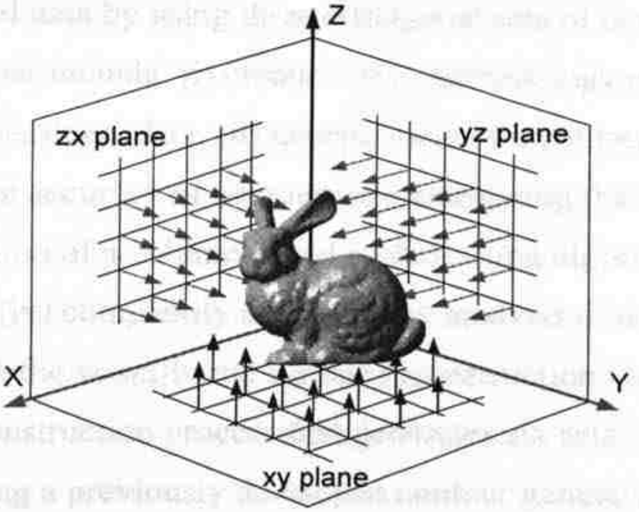


Figure 2. Construction of a Triple-Dexel Model

Benouamer and Michelucci [1997] utilized the marching cube algorithm to generate the triangular surface model from triple-dexel data by first generating voxel data from the triple-dexel data. However, the reconstructed surface may suffer from topology errors and poor approximation of sharp features. In addition, the computations are expensive because the computation complexity is proportional to the number of voxels. In a sense the triple-dexel data can be converted into point clouds and reconstructed using Delaunay triangulation or surface fitting methods available from the existing literature. However, the Delaunay triangulation and surface fitting processes are also computationally expensive. Triple-dexel data can be also converted into parallel slices where triangular surfaces can be reconstructed using surface tilling algorithms [Barequet

and 7 in Fig. 4, on two adjacent rays may form part of an inner contour, such as the connections between points 6, 7, 13 and 12. Thus, the end points of the two overlapped dexel spaces are connected, e.g. point 6 is connected with point 12 and point 7 is connected with point 13. Overlapped dexel spaces on adjacent rays separate dexels into different groups, and within each group the end points of the dexel spaces are connected. Overall, this contour generation algorithm has four steps: first, dexels on every two adjacent rays are categorized into groups according to the grouping criterion. Second, inside each group, adjacent dexel points are connected. Third, a connection table is created to record connections between dexel points. Finally, the connection table is traversed to construct contours in a counterclockwise sequence.

Connected Point	Dexel Point	Connected Point
2	**1**	3
1	**2**	8
1	**3**	9
5	**4**	10
4	**5**	11
7	**6**	12
6	**7**	13
2	**8**	14
3	**9**	10
4	**10**	9
5	**11**	15
6	**12**	13
7	**13**	12
8	**14**	16
11	**15**	16
14	**16**	15

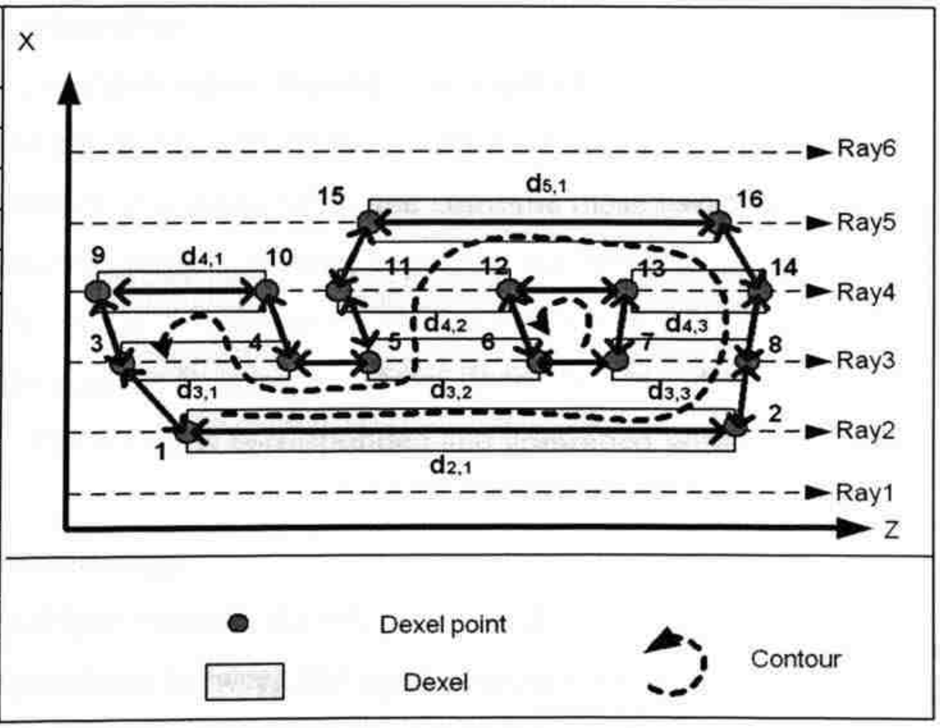


Figure 4. Contour Generation from Single-Dexel Data

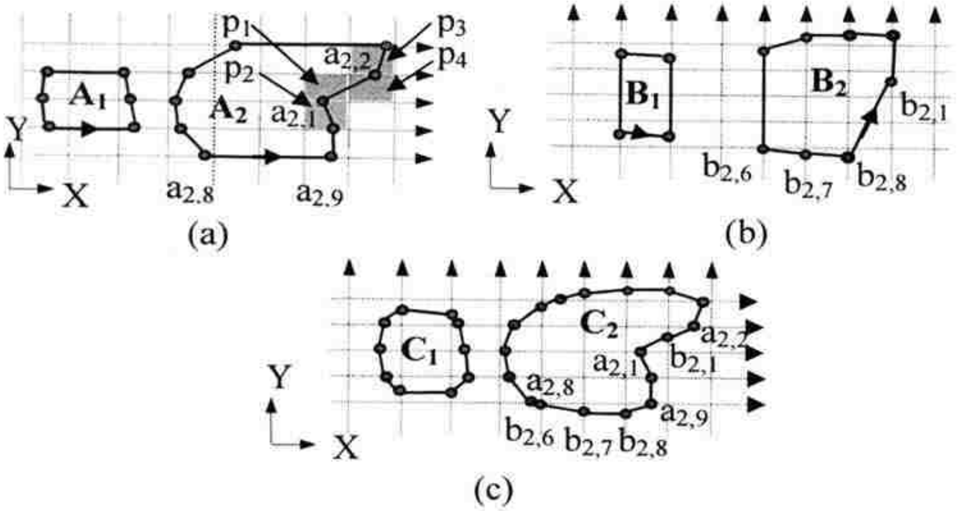


Figure 5. Contour Combination Algorithm. (a) xy Contours, (b) yx Contours and (c) the Combined Contours

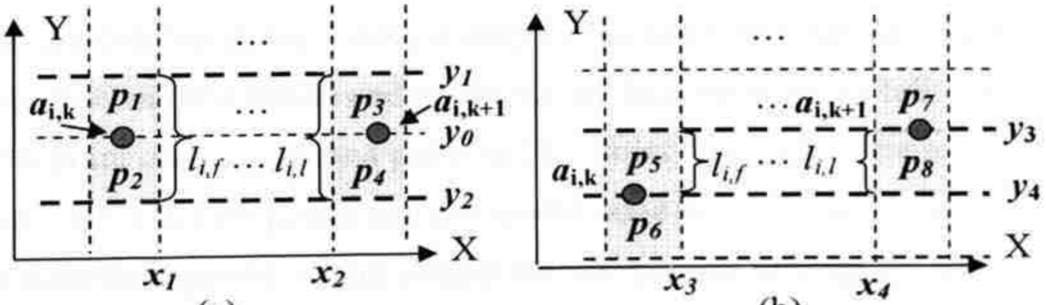
Here a local connection method is developed by realizing both input contours are in the counterclockwise direction. If we scan the points of contour A_i in the counterclockwise sequence, the points of contour B_j should be continuously added to contour A_i in the same counterclockwise sequence. For example, a sequence of three points $b_{2,6}$, $b_{2,7}$, and $b_{2,8}$ are added between points $a_{2,8}$ and $a_{2,9}$ in Fig. 5(c). This implies that if the first ($b_{j,f}$) and the last ($b_{j,l}$) associated points between points $a_{i,k}$ and $a_{i,k+1}$ are correctly selected, it is trivial to find the rest of the associated points in between and insert them into A_i. It also implies that point $b_{j,l+1}$ is the next point to be considered for inserting into contour A_i. So it is only necessary to check the next pair of points of contour A_i to see if $b_{j,l+1}$ is in between. If so, $b_{j,l+1}$ and its following points from contour B_j can be added until the last associated point ($b_{j,o}$) is identified according to the criteria defined in Section 3.2.3. This process is repeated until all the points from contour B_j have been added to contour A_i. This contour combination method is efficient because of using the point sequence information in the input contours.

3.2.2. Contour Correspondence

The contour correspondence problem involves finding which contour from contour set A is to be combined with which contour from contour set B. The overlapping

between them, and that the pixels p_1, p_2, p_3 and p_4 have no more than one ray intersecting point on any of their edges. Note that there could be many pairs of $a_{i,k}$ and $a_{i,k+1}$ that satisfy the requirements. Any of these pairs can be used as the starting pair. However, it is possible that the first pair of points is not found because of an insufficient number of rays. In such case, we can always cast additional rays to increase the ray density as discussed in Sec. 3.2.4.

Without loss of generality, in the starting pair of points, $a_{i,k}$ is assumed on the left side of $a_{i,k+1}$. Then the first associated point ($b_{j,f}$) and the last associated point ($b_{j,l}$) from B_j for points $a_{i,k}$ and $a_{i,k+1}$ can be found as follows: $b_{j,f}$ is on the line segment $l_{i,f}$, which is on the y ray immediately to the right of $a_{i,k}$; and $b_{j,l}$ is on the line segment $l_{i,l}$, which is on the y ray immediately to the left of $a_{i,k+1}$. As shown in Fig. 6(a), if $a_{i,k}$ and $a_{i,k+1}$ are on the same ray, each of $l_{i,f}$ and $l_{i,l}$ consists of two pixel edges. If $a_{i,k}$ and $a_{i,k+1}$ are on two adjacent rays, each of $l_{i,f}$ and $l_{i,l}$ consists of one pixel edge as shown in Fig. 6(b). In this case, the points on $l_{i,f}$ and $l_{i,l}$ in (b) are the first and the last associated points, respectively, from contour B_j.

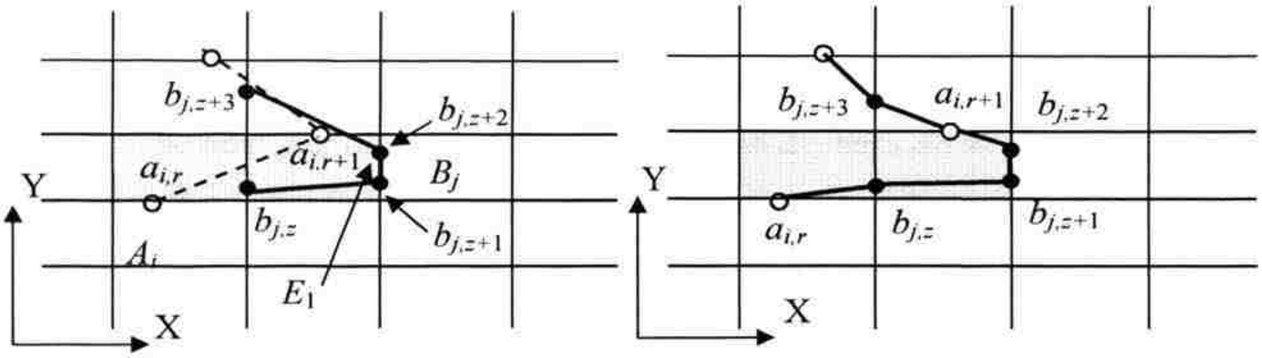


(a) (b)

Figure 6. Locations of the First and the Last Associated Points of Contour B_j. (a) $A_{\mathrm{i,K}}$ and $A_{\mathrm{i,K+1}}$ on the Same Ray and (b) $A_{\mathrm{i,K}}$ and $A_{\mathrm{i,K+1}}$ on Adjacent Rays

To pinpoint the exact first and last associated points from contour B_j when $a_{i,k}$ and $a_{i,k+1}$ are on the same ray as shown in Fig. 6(a), Table 1 is created with four possible

point between points $a_{i,r}$ and $a_{i,r+1}$. After inserting the points, the combined contour of contours A_i and B_j is shown in Fig. 8(b).



(a) (b)

Figure 8. Contour Combination Process. (a) Two Generated Contours from the Contour Generation Process and (b) the Combined Contour

If two adjacent points $a_{i,r}$ and $a_{i,r+1}$ satisfy the following relationship

$$\mathrm{INT}[(a_{i,r} \rightarrow [x])/\Delta x] = \mathrm{INT}[(a_{i,r+1} \rightarrow [x])/\Delta x] \tag{4}$$

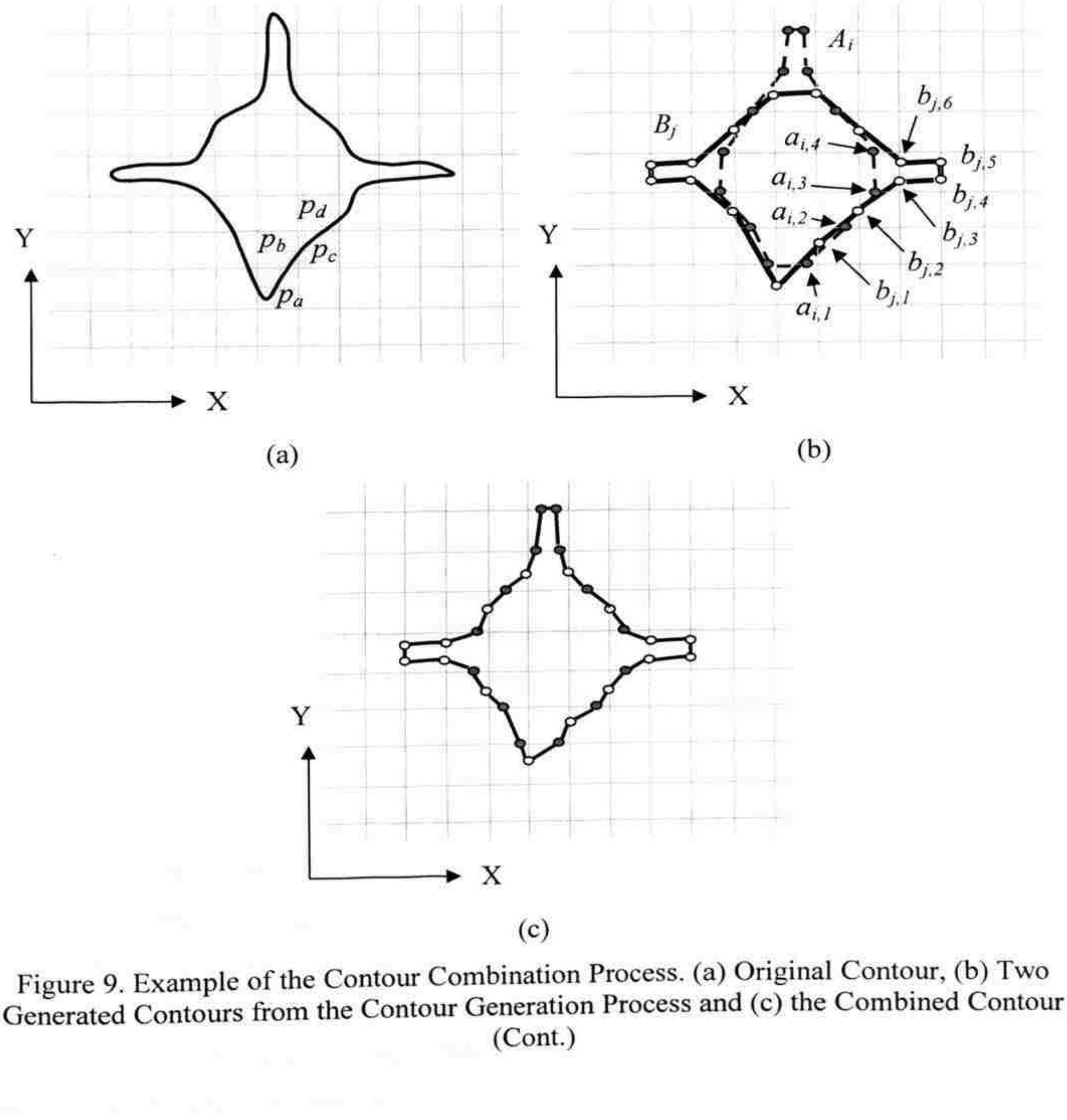
there is still a possibility that $a_{i,r}$ and $a_{i,r+1}$ have associated points from B_j in between. If $b_{j,l+1}$ satisfies the following criteria:

$b_{j,l+1}$ is on one of the edges of the pixel that contains both $a_{i,r}$ and $a_{i,r+1}$

the *y* value of $a_{i,r}$ is between the *y* values of $b_{j,l}$ and $b_{j,l+1}$

then $a_{i,r}$ and $a_{i,r+1}$ will have at least one associated point, and the first one is $b_{j,l+1}$. In this case, the same criteria as those given above can be utilized to find the last associated point ($b_{j,o}$) in between. If $a_{i,r}$ and $a_{i,r+1}$ have no associated point from B_j in between, the next pair of points in A_i, i.e., $a_{i,r+1}$ and $a_{i,r+2}$ will be checked to see if they have any associated point in between according to the above criteria. By repeating Step 2 for the rest of points in A_i until all the points in contour B_j have been added to contour A_i, the combined contour is finally obtained. The pseudo code for *Step 2* is given in the Appendix.

the same topology, our contour combination algorithm can combine the two contours to reconstruct the correct shape as shown in (f).

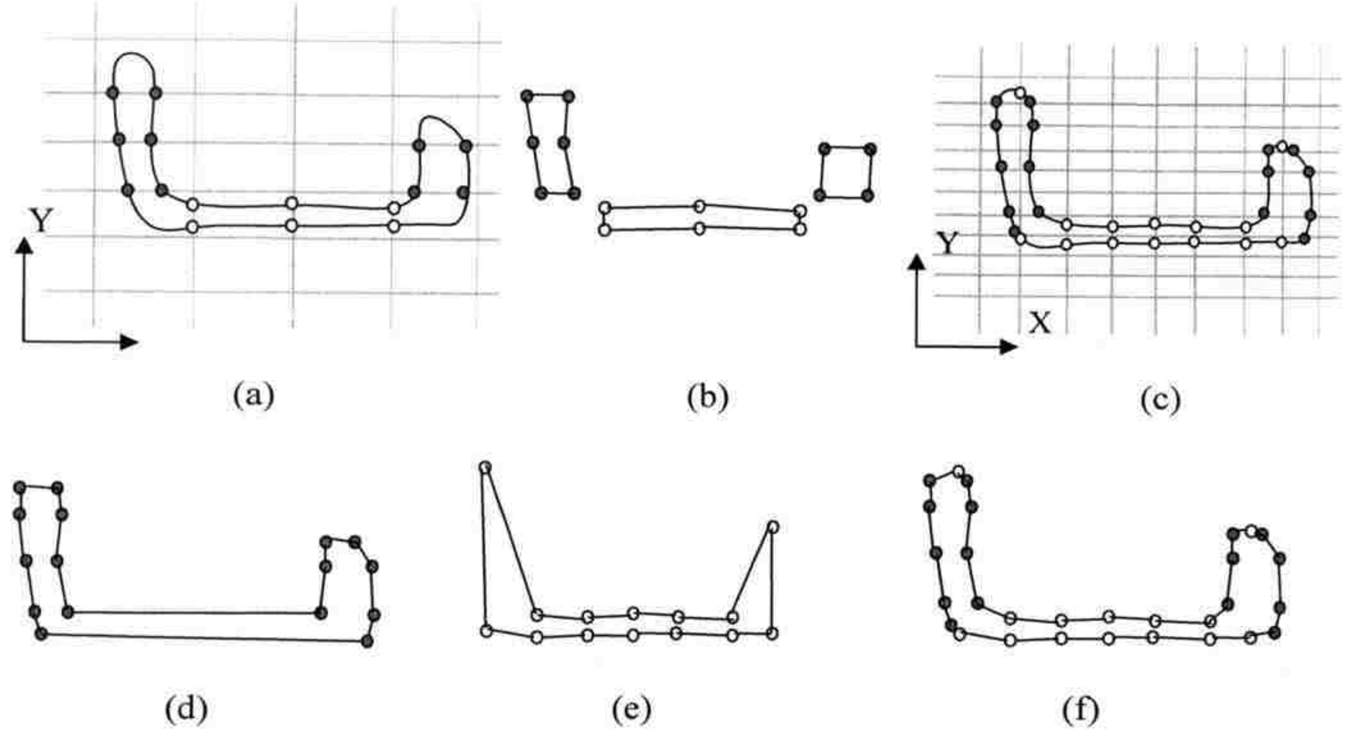


(a)

(b)

(c)

Figure 9. Example of the Contour Combination Process. (a) Original Contour, (b) Two Generated Contours from the Contour Generation Process and (c) the Combined Contour (Cont.)



(a) (b) (c)

(d) (e) (f)

Figure 11. A Case Study of the Contour Combination Process. (a) the Input Contour with X- and Y-Dexel Data, (b) Contours Generated from the X-Dexel Data and Y-Dexel Data, (c) the Dexel Points after Increasing the Number of Rays in X and Y Directions, (d) Contours Generated from X-Dexel Data in (c), (e) Contours Generated from Y Dexel Data in (c), and (f) Combined Contour from (d) and (e)

3.3. Volume-Based Surface Tiling Algorithm

After the contour combination process, three sets of orthogonal slices of contours are generated. The volume-based tiling algorithm of Svitak and Skala [2004] is utilized to reconstruct the boundary surface of the 3D model from these contours. The main idea of this volume-based tiling algorithm is to generate triangular facets within each rectangular box associated with the rays in *x*, *y* and *z* directions. Because the three sets of orthogonal contours contain the positions and connectivity of all triangle vertices, the problem of generating triangular meshes within each box becomes the problem of searching the locations and connection information of the vertices from the three sets of contours that have been generated. Once this information is obtained, it is trivial to generate the triangular facets within each box by using a triangular patching algorithm.

The volume-based tiling algorithm consists of three steps. Given a triple-dexel data with *M*, *N*, and *O* numbers of divisions in the *x*, *y* and *z* axes, respectively, the 3D space is divided into $M \times N \times O$ equal-sized rectangular boxes. The algorithm first identifies the Boundary Sub-Volumes (BSVs) that are the boxes having non-null intersections between their edges and the solid's boundary surface. Second, the three orthogonal sets of contours are searched to find a close loop of vertices within each BSV. Finally, triangular facets are created within each BSV by patching these vertices.

Some details of the algorithm are given in the follows. The algorithm identifies the BSVs by searching the intersection points within the object's boundary surface along the three orthogonal sets of rays. For example, in Fig. 12(a), the intersection point between ray R1 and the object's boundary surface is point *p* and thus, boxes *A*, *B*, *C* and *D* are the BSVs. Within each BSV, the boundary surface of the object forms a close loop. To find the close loop of vertices within a BSV, the algorithm starts from the point on the bottom of the BSV and ends when coming back to the starting point.

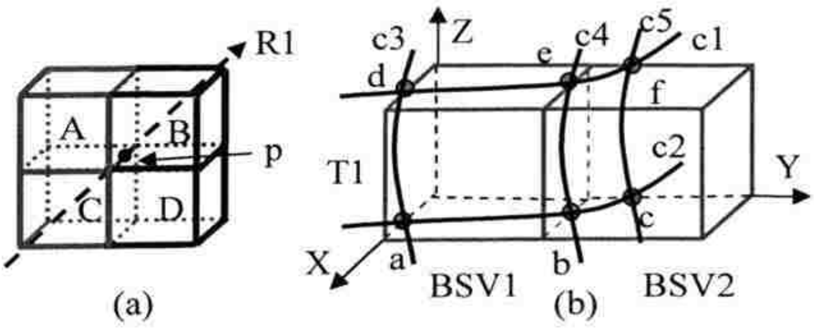


Figure 12. Volume Tiling Algorithm. (a)Identification of Boundary Sub-Volumes and (b) Generation of Surface Patches within Two Boundary Sub-Volumes

Taking Fig. 12(b) as an example, within BSV1, the search starts from point *a* on the bottom. After searching contour *c*2 on the *xy* plane, the next point found is point *b*. Because point *b* is on both contour *c*2 and contour *c*4, thus, contour *c*4 is searched to find

as shown in Fig. 13(b) and Fig. 13(c), from the dexel data in *x* and *y* directions, respectively. The combined contours on *xy* planes are shown in Fig. 13(d).

Table 2. Surface Errors of the Reconstructed Bunny Model from Triple-Dexel Data

Resolution	Hausdorff distance (d_H: mm)	Normalized error (e)
30*30	0.003334	1.332%
50*50	0.001989	0.7525%
100*100	0.001445	0.4390%
200*200	0.000789	0.2656%

Table 3. Surface Errors of the Reconstructed Bunny Model

Resolution	Normalized error from the triple-dexel model	Normalized error from the single-dexel model
50*50	0.7525%	1.365%
100*100	0.4390%	0.658%

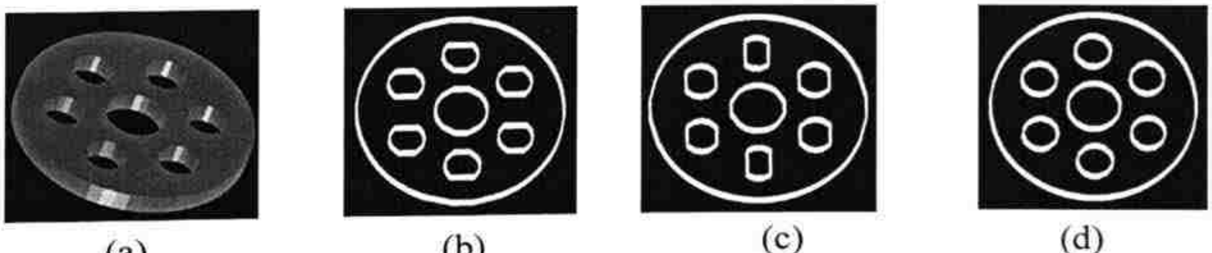


Figure 13. Illustrative Example of the Contour Combination Algorithm. (a) Input Object Model, (b) Contour Generated from X-Dexel Data, (c) Contour Generated From Y-Dexel Data, and (d) Combined Contour from (b) and (c)

After generating three orthogonal sets of contours in the contour combination process as described, the volume-based surface tiling algorithm is utilized to generate the boundary surface of the 3D model. Figure 14 shows the reconstructed surface of a bunny from the obtained contours on 70 slices in each of *xy*, *yz*, and *zx* planes. Figure 15 illustrates the surface improvement from the triple-dexel data over the single-dexel data. Figures 15(a) and (c) show the results of surface reconstruction from single-dexel data, and Fig. 15(b) and (d) show the corresponding results of surface reconstruction from triple-dexel data. These figures clearly show that the generated surface from the triple-dexel data is more accurate than the reconstructed surface from the single-dexel data when using the same ray resolution.

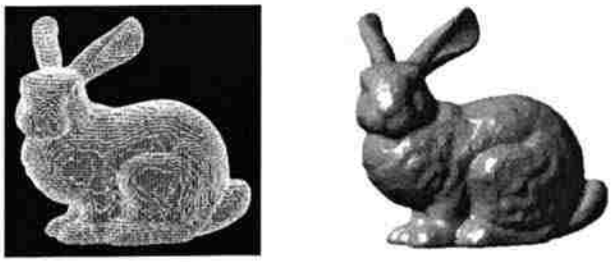


Figure 14. A Bunny Model and the Reconstructed Surface of the Bunny

The developed surface reconstruction process based on the triple-dexel model is incorporated into a virtual sculpting system [Peng and Leu, 2003; Leu et al., 2005; Peng et al., 2006]. The virtual sculpting system is developed on a Microsoft Windows XP workstation. The software is written in C++, and the graphics-rendering component is built on OpenGL and GLUT. The haptics interface is implemented using the PHANToM^{TM} device and the GHOST (General Haptics Open Software Toolkit) SDK software available from SensAble Technologies. This virtual sculpting system enables the user to create and modify 3D freeform objects through interactive sculpting operations and gives the user real-time force feedback during the sculpting process. The tool swept volume between two consecutive sampling times is obtained by the Sweep Differential Equation method [Blackmore and Leu, 1992] and represented by boundary triangular meshes [Peng and Leu, 2003]. The workpiece and the tool swept volumes are scan-converted to obtain their triple-dexel data. Boolean operations on the triple dexels are performed by comparing and merging the dexel data in each of *x*, *y* or *z* directions. The surface reconstruction software is executed during the sculpting process to convert the triple-dexel model to a triangular mesh model. Figure 16 shows the setup of the virtual sculpting system and a cat model created using the system and viewed from two different directions.

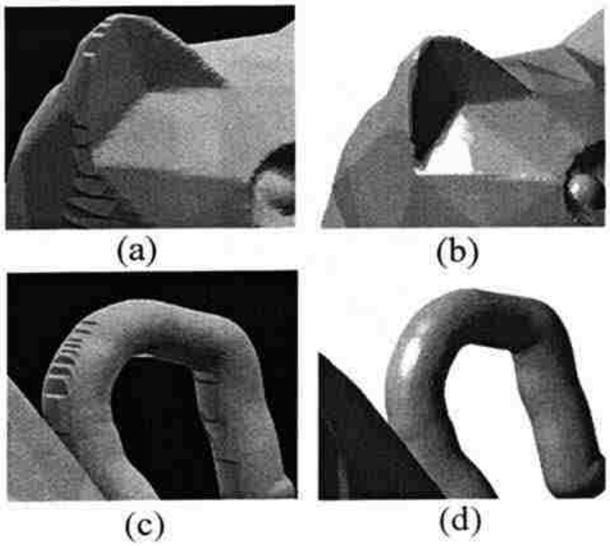

(a)
(b)
(c)
(d)

Figure 15. Comparisons Between Single-Dexel Data and Triple-Dexel Data. (a) and (c) Surfaces Reconstructed from Single-Dexel Data, (b) and (d) from Triple-Dexel Data

6. COMPARISON WITH VOXEL REPRESENTATION

Voxel modeling is a popular representation scheme [Kaufman et al., 1995; Hadwiger et al., 2006]. To benchmark the performance of the developed method, numerical experiments are conducted to compare using triple-dexel data vs. voxel data in terms of the surface reconstruction time and the associated surface error. An impeller and a bunny model, as shown in Fig. 17, are discretized into voxel data in the resolution of 50*50*50, 100*100*100 and 150*150*150 by using a fast voxelization algorithm [Karabassi et al., 1999]. The voxel data is stored in the 3D array structure, with the marching cube algorithm utilized to reconstruct the surface from the voxel data. Meanwhile, the triple-dexel data is stored in the linked list structure and the object's surface is reconstructed using the method developed in this paper. The normalized surface errors of the reconstructed surface are calculated using the Metro comparison tool [Cignoni et al., 1998] and shown in Table 4. The time of the contour generation, correspondence and combination process is compared with the surface reconstruction time from the voxel representation in different resolutions in this table.

Figure 16. A Cat Model Generated Using the Virtual Sculpting System

The test result shows that, under the same resolution, the surface reconstructed from the triple-dexel data has a smaller surface error in comparison with the surface reconstructed from the voxel data. This is because the triple-dexel based method utilizes actual positions of the intersection points between rays and the object's boundary surface as the vertices of the reconstructed surface model, while the voxel based method approximates the positions of these vertices by voxel interpolation.

The computation complexity of the contour generation, correspondence and combination process using triple-dexel data is $O(T)$ or $O(M^2)$, where M is the number of divisions along each axis. Because the complexity of the volume-based tiling algorithm is also $O(M^2)$[Svitak, 2004], the developed surface reconstruction method is more efficient than the voxel-based surface reconstruction method, whose computational complexity is $O(M^3)$. The total computation times for surface reconstruction from the voxel data and from the triple-dexel data of the impeller and the bunny models are plotted vs. the number of divisions along each axis in Figs. 18 and 19. The results in these figures verify that the triple-dexel model is more efficient than the voxel model.

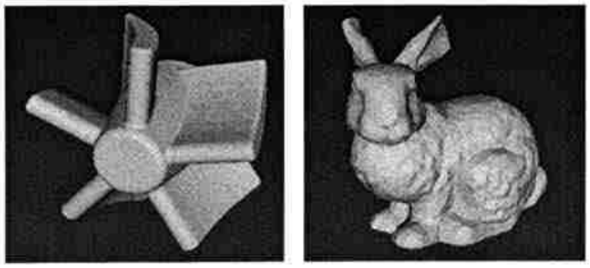


Figure 17. Two Test Cases: Impeller and Bunny

Table 4. The Surface Reconstruction Time and Surface Error

Resolution	Test Model	No. of Dexel Points	Reconstruction Time Using Triple Dexels (s)	Reconstruction Time Using Voxels (s)	Error of Reconstructed Surface from Triple-Dexel Data (%)	Error of Reconstructed Surface from Voxel Data (%)
50*50*50	Impeller	19916	0.1333	0.12843	0.4263	0.9091
	Bunny	14402	0.0985	0.11755	0.7525	1.0272
100*100*100	Impeller	79632	0.4974	0.9836	0.1683	0.4012
	Bunny	57852	0.3631	0.9436	0.4390	0.5063
150*150*150	Impeller	179727	1.1720	3.2290	0.1843	0.2653
	Bunny	130650	0.8230	3.1300	0.2656	0.4299

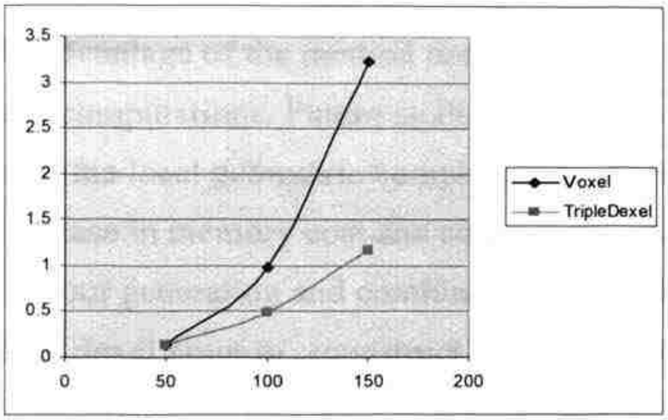


Figure 18. Surface Reconstruction Time vs. Number of Divisions from the Voxel Data and the Triple-Dexel Data for the Impeller

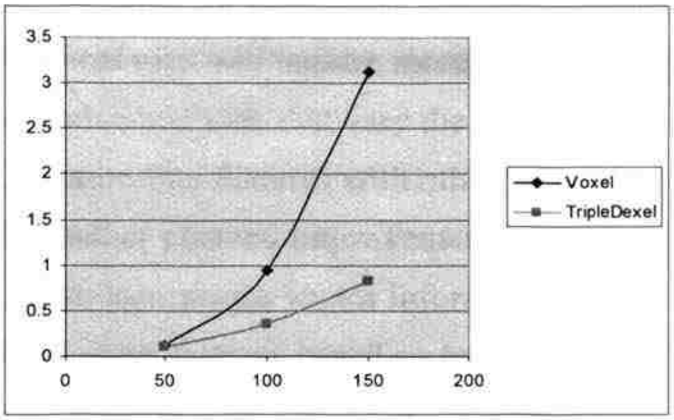


Figure 19. Surface Reconstruction Time vs. Number of Divisions from the Voxel Data and the Triple-Dexel Data for the Bunny

7. CONCLUSIONS

This paper has described a novel method of surface reconstruction from triple-dexel data. Three sets of contours on orthogonal slices are generated from triple-dexel data by a contour generation algorithm and a contour combination algorithm. A volume-based tiling algorithm is then utilized to generate the boundary surface of the 3D object in triangular patches from these contours. The computation complexity and the memory requirements of the developed method are analyzed. Both computation time and memory cost are found to be linearly proportional to the number of dexel points of the triple-dexel model. Comparing with the surfaces reconstructed from single-dexel data and from voxel data with the same resolution, our triple-dexel based method has a higher surface accuracy. Also the described surface reconstruction method is more efficient than the popular voxel-based method. The developed surface reconstruction process has been incorporated into a virtual sculpting system to address the view-dependent problem inherent in triple-dexel modeling. Examples are given to demonstrate the capability of the developed method.

The developed contour combination method requires the same numbers of input contours generated from rays in two orthogonal directions for each slice. In case the numbers of contours are different, the density of rays to scan the object needs to be

using the level-set method [9]. We will apply the level-set method with mean curvature flow to develop surface smoothing operation in Section 4.

3. GENERATING DISTANCE FIELD FROM TRIPLE-DEXEL DATA

In our triple-dexel based virtual sculpting system, the representation of a solid during the sculpting process is by computing intersections between the solid and rays in three orthogonal (e.g., x, y and z) directions. For each ray, the intersection points and the surface normal sampled at each point are stored. Two intersection points in a line segment that is completely inside the solid is defined as a dexel. An illustration is given in Figure 1.

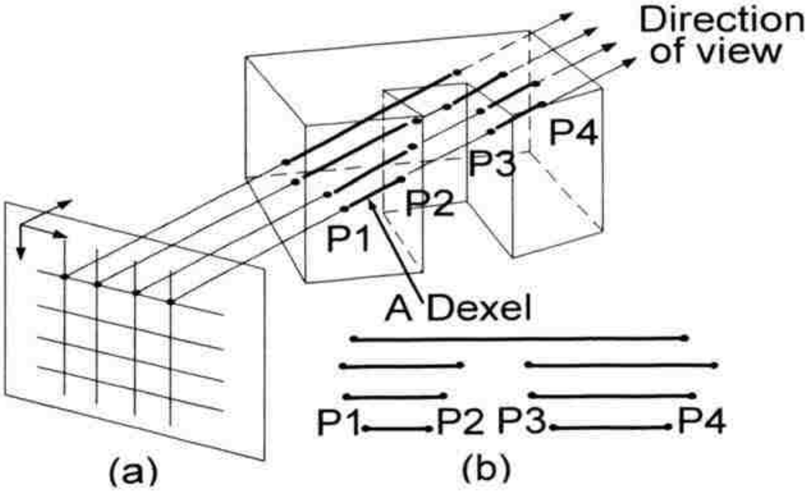


Figure 1. Illustration of the Ray-Casting Process and the Dexel Representation

To simulate the material removal process, Boolean operations are performed between the triple-dexel data of the workpiece and the tool. To visualize the sculpted solid, we have developed a surface reconstruction method from triple-dexel data. The method includes contour generation, contour combination and surface tiling algorithms [5]. Meanwhile, the PHANToM^{TM} manipulator (SensAble Technologies) is used to

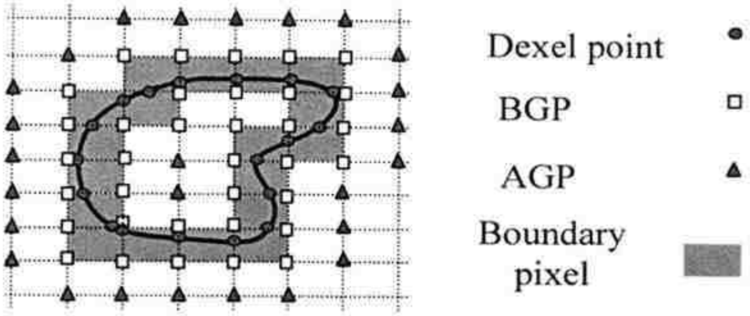


Figure 2. Boundary Pixels, BGP, AGP and Dexel Points

3.3. Approximate Iso-Surfaces Inside BVs

The main idea of this step is to use the Hermite data (i.e., exact intersection points and normals) on the edges of a BV to calculate an additional point inside the BV by minimizing a quadratic function. By connecting this point with other additional points in adjacent BVs, triangular meshes can be generated with a simple patching algorithm to approximate the boundary surface.

In the case of using triple-dexel data, the dexel points and the surface normals at these points are available from the triple-dexel data. The additional point inside a BV is the intersection point between the tangent elements of the dexel points on the edges of a BV. A 2D example is shown in Figure 3, where the circle points are the dexel points and the square points are the additional points.

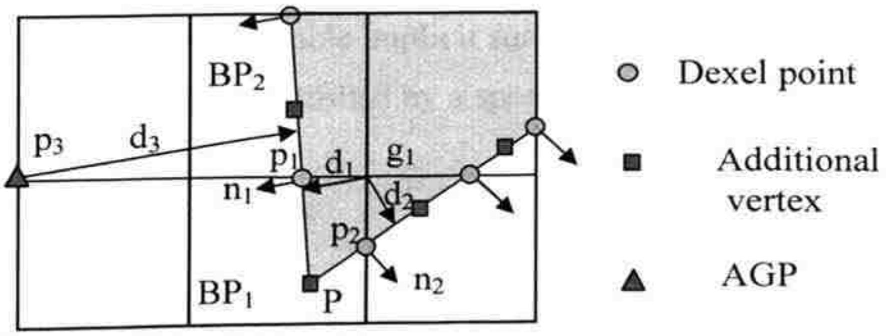
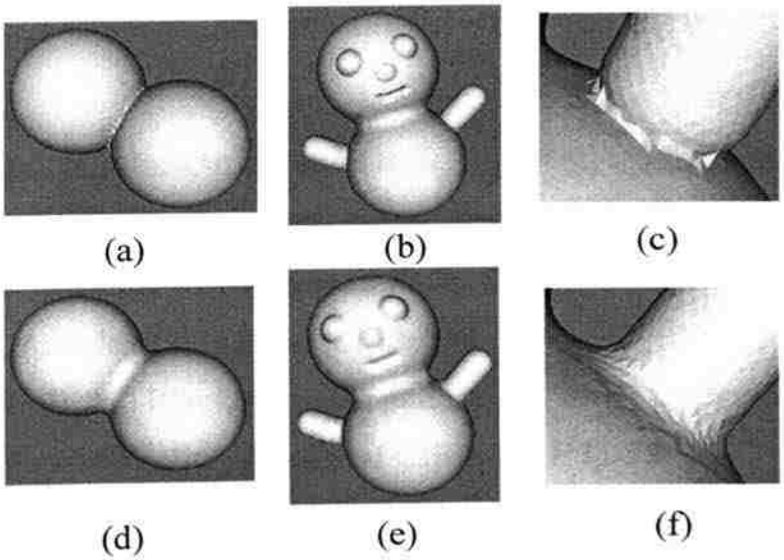


Figure 3. Distance Calculation for the Grid Points

satisfied result has been obtained. Figures 6(a) and (d) show two spheres joined together before and after the smoothing operation. A snowman model is created by adding and removing materials w.r.t. the two spheres model. Figures 6(b) and (c) show the snowman model and the smoothed model is shown in (e) and (f) for comparison.

Figure 5. A Cat Model Created Using the Virtual Sculpting System



(a) (b) (c)

(d) (e) (f)

Figure 6. Modeling Examples. (a) and (d) Two Joined Spheres, (b) and (e) a Snowman Model,(c) and (f) the Boundary with and without Smoothing

origin O_i and three unit vectors u_i, v_i, w_i as shown in Fig. 2, where $u_i \times v_i = 0$, $v_i \times w_i = 0$ and $w_i \times u_i = 0$.

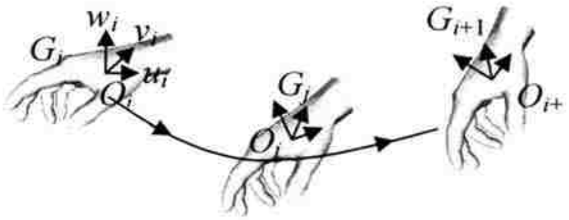


Figure 2. Hand Gesture Modeling by Interpolation

To produce a smooth space warping from the input gestures, a B-Spline interpolation is constructed to calculate the position and orientation of the gesture in between. The B-spline curve passing through (n+1) points, $O_i(x,y,z)$, i=0, … n, is defined as:

$$O_r(x,y,z) = \sum_{i=0}^{n} O_i(x,y,z) N_{i,k}(r) \tag{1}$$

where n+1 is the total number of sampled points from the user's hand input, r is the parameter, k-1 is the degree of the B-Spline curve and $N_{i,k}$ is the B-spline basis function where

$$N_{i,k}(r) = \frac{(r - t_i) N_{i,k-1}(r)}{t_{i+k-1} - t_i} + \frac{(t_{i+k} - r) N_{i+1,k-1}(r)}{t_{i+k} - t_{i+1}} \tag{2}$$

and

$$N_{i,1}(r) = \begin{cases} 1 & if\ t_i \le r \le t_{i+1} \\ 0 & otherwise \end{cases} \tag{3}$$

For a B-spline function, the parameter t_p is calculated as:

$$t_p = \begin{cases} 0 & if \quad p < k \\ p-k+1 & if\ k \le p \le n \\ n-k+2 & if \quad p > n \end{cases} \tag{4}$$

Similarly, the interpolated orientation (u_r, v_r, w_r) is calculated as

$$u_r = \sum_{i=0}^{n} u_i N_{i,k}(r) \tag{5}$$

$$v_r = \sum_{i=0}^{n} v_i N_{i,k}(r) \tag{6}$$

$$w_r = \sum_{i=0}^{n} w_i N_{i,k}(r) \tag{7}$$

The tangent vector of the B-spline curve at point $O_r(x, y, z)$ is calculated as

$$O'_r(x,y,z) = \sum_{i=0}^{n} O_i(x,y,z) N'_{i,k}(r) \tag{8}$$

3.2. The Influence Zone of a Tool

In the modeling method we propose, each virtual tool has a limited local region of space around the tool, defined by the distance field and a user defined region of influence (RI). The distance field $d_s(x)$ is a scalar field that is defined by the minimum distance between every point x in space and a given surface S. Because the tool shape in the modeling process is pre-defined, the tool's distance field can be preprocessed using the closest point transformation algorithm [Mauch, 2003]. The user can change the parameter R to adjust the tool's RI as shown in Fig. 3.

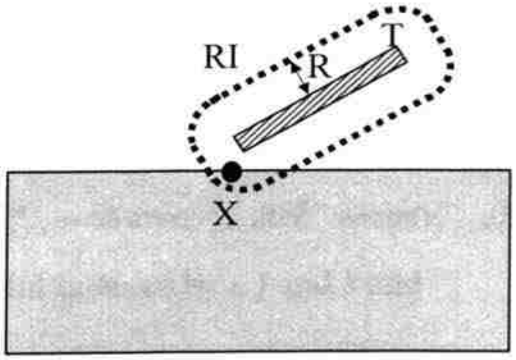


Figure 3. The Influence Zone of a Tool

function is formulated to interpolate the trajectory traversed through these points. This interpolated B-spline curve intersects many grids as shown in Fig. 4. Let g_j be a grid point on one or more of these grids. To calculate the velocity of g_j, we find its closest point p_j on the B-spline curve and calculate its tangent vector by Eq. (8). This is seen as a small dark arrow in Fig. 4. If the magnitude of the velocity is constant, it can be used to define an imprinting operation. If it is associated with a weighting function which varies with the distance from the grid point to the tool boundary, it can be used to define a deformation operation. If it is associated with the curvature of local geometry, it can be used to define a smoothing operation. These operations will be discussed in detail in Section 4.

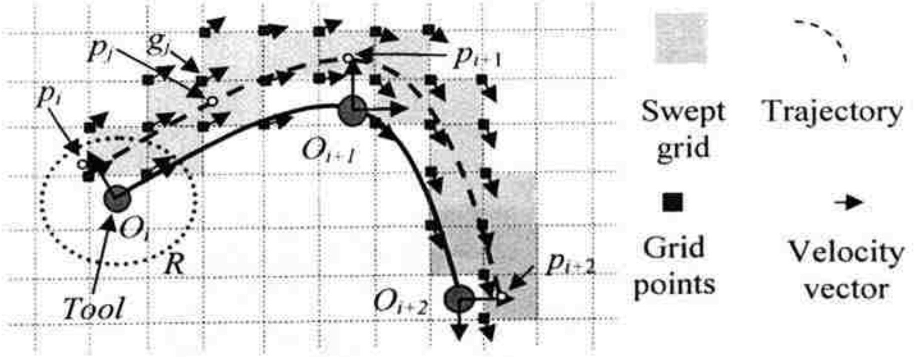


Figure 4. Generation of the Grid-Based Velocity Field

4. FREEFORM MODELING OPERATIONS

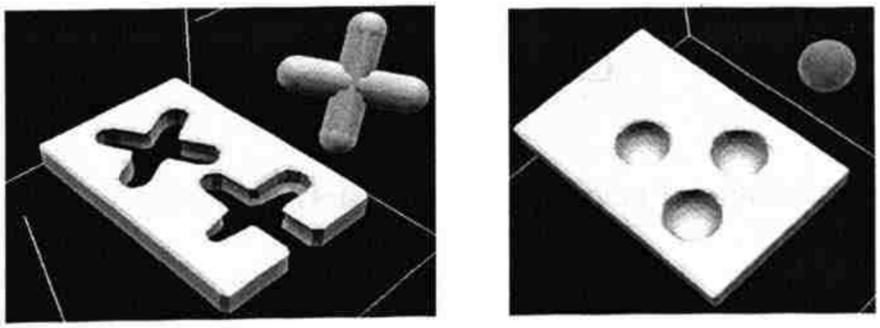
In this section, three freeform modeling operations, i.e. the imprinting operation, the deformation operation, and the smoothing operation, are developed. The modeling results are compared with other virtual sculpting methods to demonstrate the usefulness of the proposed method.

4.1. Imprint Operation

The imprinting operation works as follows: the user selects different types of tools from a tool library, and defines parameters to customize the tool shape. Then, the user grabs the virtual tool using a 3D digital manipulating device such as a space mouse or the Phantom™ haptic device, and applies the imprinting operation onto the initial workpiece model which is updated in real-time. In the imprinting operation, the movement of the virtual tool generates a grid-based velocity field along the path of the virtual tool and the solution of the level-set method changes the boundary of the workpiece. Only the velocity component in the normal direction of the workpiece boundary contributes to the change of this boundary, thus Eq. (10) can be written as

$$\frac{\partial F(x,t)}{\partial t} + k \cdot \| \nabla F(x,t) \| = 0 \tag{16}$$

where k is a user-defined constant to control the speed of the propagation. The gradient $\nabla F(x,t)$ is approximated by the up-wind finite difference scheme described in Sec. 3.3. Figure 5 is an example showing the imprinting operation by using a cross-shaped tool and a spherical tool to modify a rectangular plate.



(a) (b)

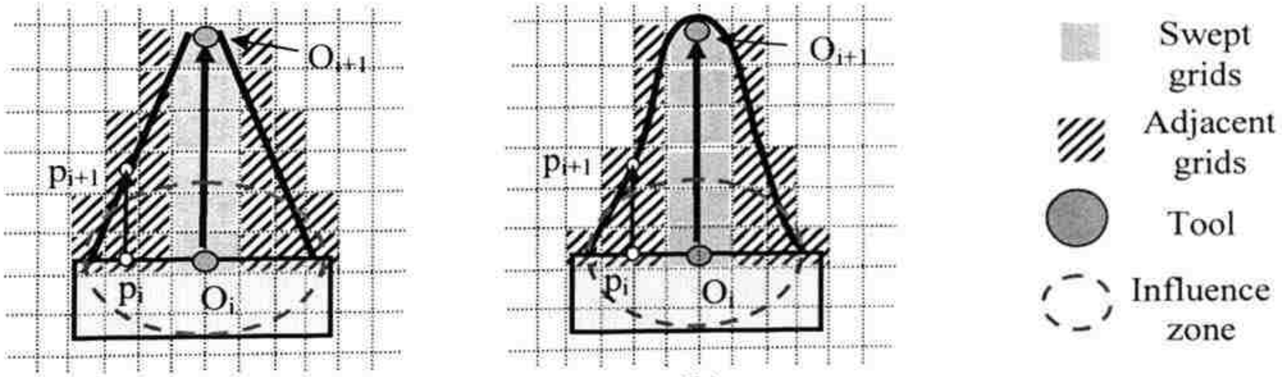
Figure 5. Example of Imprinting Operation. a) Using a Cross-Shaped Tool and (b) Using a Spherical Tool

4.2. Fold-Free Deformation Operation

As discussed in Sec. 3.4, the movement of a virtual tool generates a grid-based velocity field along the trajectory of the tool. To generate the effect of deformation, the original grid-based velocity field is modified by a user-defined weight function as follows:

$$p_{i+1} = p_i + w(p_i) \cdot t(O_i) \tag{17}$$

where O_i is the position of the tool at time i, p_i is a vertex on the boundary of the workpiece inside the tool's region of influence, $w(p_i)$ is a user defined weight function to control the shape of the deformation, and $t(O_i)$ is the transformation matrix. Figure 6 is a 2D illustration example, where the tool moves from point O_i to O_{i+1}. The tool's influence zone is within the dotted ellipse at time i. The point p_i, which is a vertex of the workpiece surface inside the influence zone, is transformed to point p_{i+1} according to Eq. (17).



(a) (b)

Figure 6. Example of the Shape Deformation. (a) Linear Interpolation and (b) Cubic Interpolation

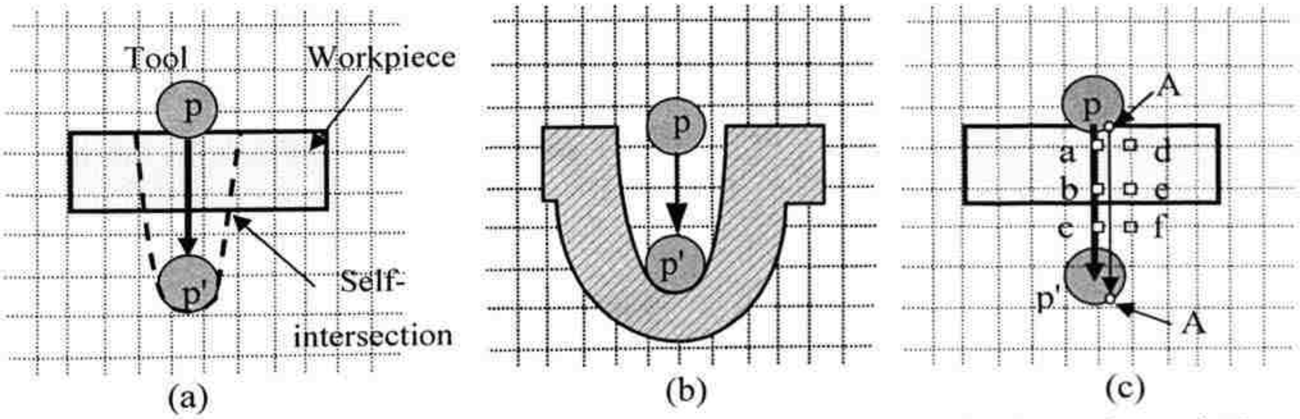
The weight function can be defined as a linear interpolation or a cubic interpolation as follows:

$$w(x) = 1 - d(p) \tag{18}$$

$$w(x) = 1 - d^2(p)(3 - 2d(p)) \tag{19}$$

By using the above weight function, the top boundary of the workpiece can be deformed into different shapes as shown in Fig. 6. The grid-based velocity field is generated using the same procedure as given in Sec. 3.4.

As mentioned before, mesh-based spatial deformation method may generate fold-over of the ambient space and self-intersection of the object as shown in Fig. 7(a), where the upper boundary of the workpiece is deformed by the movement of tool from *p* to *p*' and intersected with the lower boundary of the shape. The deformed upper boundary is represented by the dotted lines.

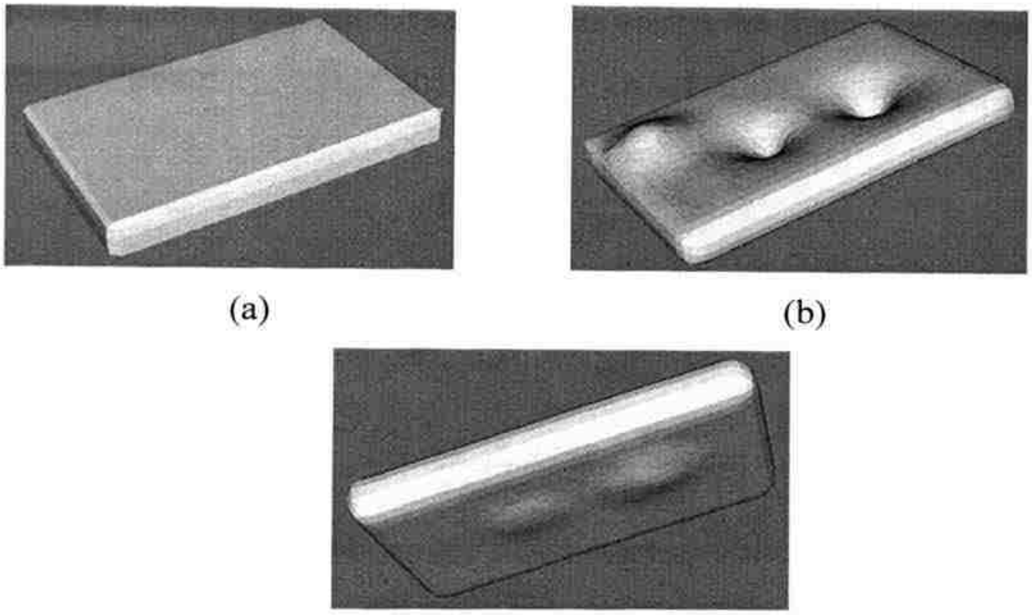


(a) (b) (c)

Figure 7. The Deformation Operation. (a) Self-Intersection, (b) the Deformed Shape Without Self-Intersection, and (c) Boundary Propagation by Defining the Velocity for the Boundary Grids

To solve the self-intersection problem and generate the deformed shape as shown in Fig. 7(b), we can calculate the grid-based velocity field not only according to the movement of the tool and the user-defined weight function, but also to the grid point's inside/outside information. We propose the following folder-free deformation algorithm consisting four steps:

- Step1: Identify the workpiece's boundary vertices inside the influence zone of the tool and their transformed points according to the input vector and a user defined



(a) (b) (c)

Figure 8. The Freeform Deformation Operation. (a) A Plate Model before Deformation (b) the Front Side of the Model after Deformation and (c) the Backside of the Model after Deformation

4.3. Smoothing Operation

In the smoothing operation, we assign the magnitude of the velocity at each grid point proportional to the curvature of the shape as follows:

$$\frac{\partial F(x,t)}{\partial t} - bH(x,t)\|\nabla F(x,t)\| = 0 \tag{20}$$

where *b* is a user-defined constant and *H*(x,*t*) is the mean curvature of the boundary surface at point x, which is the average of the principal curvatures (κ_1 and κ_2), i.e.

$$H = (\kappa_1 + \kappa_2)/2 \tag{21}$$

For a surface in 3D space defined as *F*(*x*,*y*,*z*), the mean curvature at a grid point is

$$H = \frac{(F_{yy}+F_{zz})F_x^2 + (F_{xx}+F_{zz})F_y^2 + (F_{xx}+F_{yy})F_z^2 - 2(F_xF_yF_{xy} + F_xF_zF_{xz} + F_yF_zF_{yz})}{2(F_x^2+F_y^2+F_z^2)^{3/2}} \tag{22}$$

where the differential terms can be approximated using the first-order, central finite difference as follows:

$$F_x = \frac{F_{i+1,j,k} - F_{i-1,j,k}}{2\Delta x} \tag{23}$$

$$F_{xx} = \frac{F_{i+1,j,k} - 2F_{i,j,k} + F_{i-1,j,k}}{\Delta x^2} \tag{24}$$

$$F_{xy} = \frac{F_{i+1,j+1,k} - F_{i+1,j-1,k}}{4\Delta x\Delta y} + \frac{F_{i-1,j-1,k} - F_{i-1,j+1,k}}{4\Delta x\Delta y} \tag{25}$$

According to Eq. (20), the part of the boundary with a larger curvature moves faster along the surface normal direction than the part of the boundary with a smaller curvature. This movement results a smoothing operation as illustrated in Fig. 9, where the top of a cylindrical shape is smoothed by the developed smoothing operation.

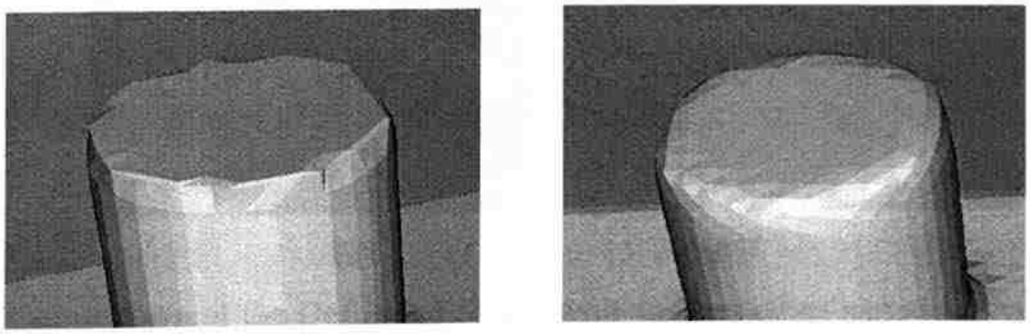
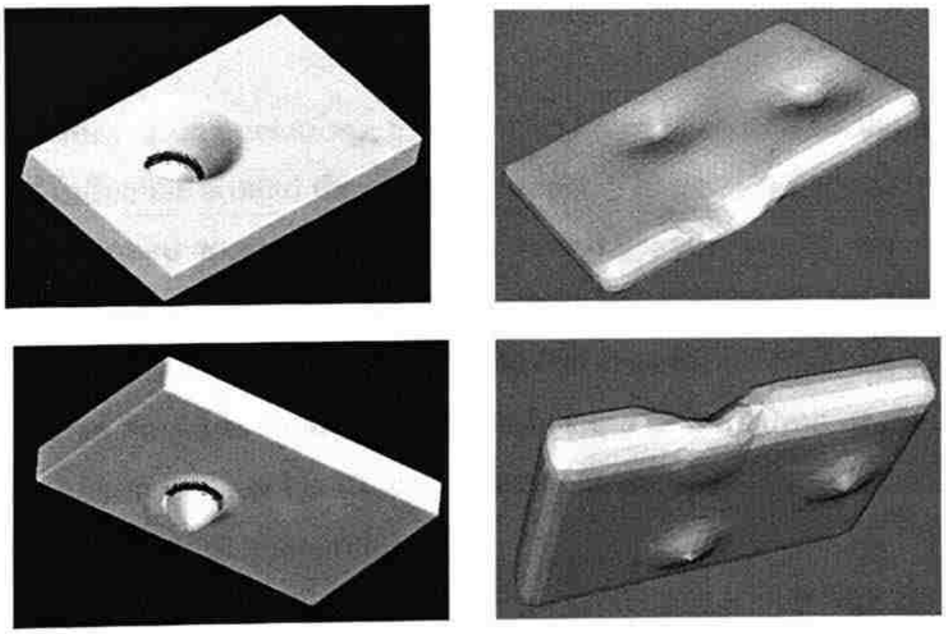

(a) (b)

Figure 9. Example of a Smoothing Operation on the Top of a Cylindrical Shape. (a) before Smoothing and (c) after Smoothing

4.4. Advantage of the Modeling Method

To demonstrate the advantage of our level-set based freeform modeling method with the same operation available from an existing commercial package, which is the FreeForm™ modeling system (v8.1) from SensAble Technology [2008], a thin rectangular block is deformed to generate a dent area as shown in Fig. 10. In the FreeForm system, the deformed top surface intersects with the unreformed bottom surface and this self-intersection of the boundary of the rectangular block creates a non-manifold object with two separate geometric entities as seen in Fig. 10(a). In contrast, by using the level-set method, the entire shape is deformed without producing multiple parts, thus remaining a manifold, as seen in Fig. 10(b).



(a) (b)

Figure 10. Comparison of the Deformed Shape with the Same Deformation Operation. (a) by the FreeForm™ System and (b) by Our System

5. IMPLEMENTATION

Our freeform modeling system runs on a Microsoft Windows XP workstation equipped with a 1.6 GHz CPU and 1 GB RAM. The software is written in C++, and the graphics-rendering component is built upon OpenGL and GLUT libraries. The setup of the modeling system is shown in Fig. 11.

Figure 11. The Virtual Shape Modeling System Setup

To apply deformation operations, a pre-defined tool is chosen by the user to select a certain region of influence around the sculpted model. Then the workpiece within selected region is deformed according to the user's hand gesture inputs. The surface modification process can be stopped at any time once a satisfied result has been obtained. Figures 12(a) and (b) show two spheres joined together before and after the smoothing operation. A snowman model is created by smoothing and deformation on the two-sphere model and the result is shown in Fig. 12 (c). Figures 12(d) and (e) show a part of the snowman model before and after smoothing.

To evaluate the performance of the described method, a smoothing operation is performed on a shape. The number of grid points, the time of calculating distance values, and the time of updating the lists are given in Table 1. It can be seen from the table that about a 11.7Hz refresh rate can be achieved by updating 28,260 grid points in each iteration.

6. CONCLUSION

This paper presents the development of a spatial warping method using the implicit distance field data representation and the level-set method for shape modeling. The trajectory of the user's hand is interpolated and utilized to define a grid-based velocity field. The solution of the level-set method propagates the boundary of the workpiece with the external velocity field, resulting different freeform modeling operations such as imprinting, deformation, smoothing, etc. The developed modeling operations are intuitive and easy to use for freeform modeling. Compared with the mesh-based spatial warping methods, the triangular meshes generated using the described spatial warping method are free of the self-intersection problem.

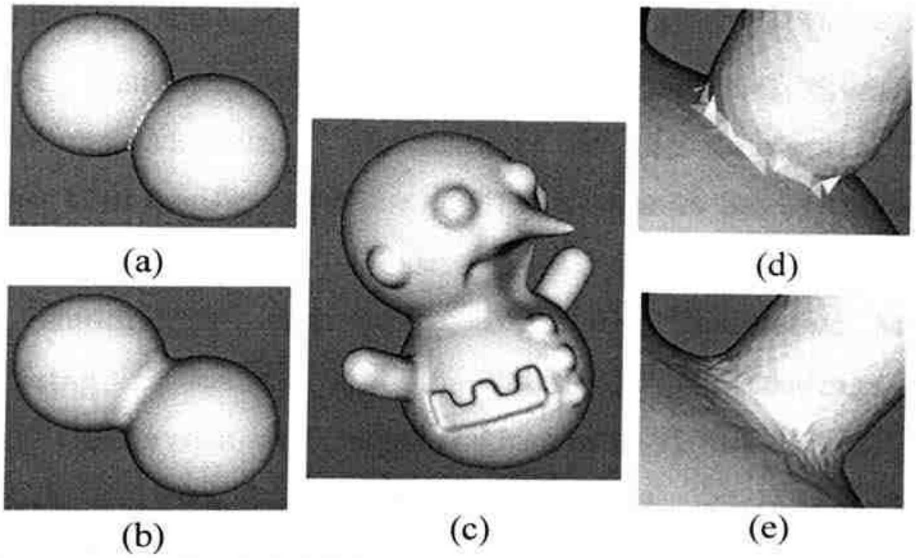


Figure 12. Modeling Example. (a) and (b) Two Joined Spheres and the Smoothed Shape, (c) the Snowman Model after Deformation and Smoothing, (d) & (e) Part of the Snowman Model before and after Smoothing

Table 1. Test Results of the Level-Set Method

No. of grid points	Time of calculating the distance values (s)	Time of updating the lists (s)	Total time (s)
202,592	0.4637	0.1631	0.6268
156,702	0.3675	0.0973	0.4648
149,942	0.3680	0.0902	0.4582
101,788	0.1754	0.0897	0.2651
28,260	0.0746	0.0108	0.0854
23,217	0.0638	0.0108	0.0746