EXAMPLES OF GRID GENERATION WITH IMPLICITLY SPECIFIED SURFACES USING GridPro^{TM}/az3000 [1]: FILLETED MULTI-TUBE CONFIGURATIONS

Zheming Cheng and Peter R. Eiseman

Program Development Corporation
300 Hamilton Avenue, Suite 409
White Plains, NY 10601

SUMMARY

With examples, we illustrate how implicitly specified surfaces can be used for grid generation with **GridPro/az3000**. The particular examples address two questions: (1) How do you model intersecting tubes with fillets? and (2) How do you generate grids inside the intersected tubes? The implication is much more general. With the results in a forthcoming paper which develops an easy-to-follow procedure for implicit surface modeling, we provide a powerful means for rapid prototyping in grid generation.

INTRODUCTION

The theory for implicit surface formulations has its roots in algebraic geometry and asymptotics. The implicit form, as opposed to explicitly defined parametric surfaces, is given by a function U of the embedding space variables: the Cartesian x, y, and z. The surface definition is then a level surface of the function $U(x, y, z)$. A level surface is just the set of points for which the specified function is a constant. Level surfaces, for example are commonly used to graphically display the contour lines of some variable. Since we are interested in only one level surface, there is no loss of generality in forming $U(x, y, z)$ in such a way that our desired surface appears when $U(x, y, z)$ is 0. This can always be made to happen for if the desired surface appeared with a value of U equal to another constant C, then we would merely replace U by $U - C$. Geometric modeling with implicit surfaces has been considered by [2].

As with contour lines in graphics, it is clear that the surface can have a richer topology. In the case of intersecting tubes, which serves as an example to illustrate the use of implicit surfaces, the topology is such that the surface has holes at the openings of each tube. Unlike the parametric constructions, this surface cannot be smoothly contracted to a point: it has non-trivial homotopy and cohomology groups. The parametric constructions such as NURBS surfaces are not general enough. They can only be applied in a local piecewise sense: not a global sense.

To explore fully the potential of implicitly specified surfaces in grid generation, there are two important aspects need to be addressed: 1). How can a given implicit surface be used by a grid generator. This is the topic of this paper. 2). How can a user setup implicit surfaces for his/her particular application with ease. This aspect falls in the general area of what can be called implicit surface modeling. This will be addressed in a forthcoming paper, where a well defined and easy-to-follow procedure for general implicit surface modeling is developed.

Altogether, implicit surfaces represent a very useful and powerful facility to specify geometries.

In many situations, it is more advantageous than the explicitly (parametrically) defined surfaces. Often, a simple function can represent a rather complex geometry as is the case shown in this paper. It can be used to define generic shapes of surfaces. Therefore, a grid generator that accepts implicit surfaces is very well suited to perform rapid prototyping in grid generation.

The general implicit surface definitions by level surfaces is one of the surface types that can be directly taken by the grid generator, **GridPro/az3000** [3, 4], for either global or local surface specifications. In fact, for any grid generator to be considered really advanced must be able to deal with implicit surfaces efficiently.

GridPro/az3000 is the multiblock grid generator with automatic zoning. Aside from surface geometry, the primary user input is the pattern of grid lines or surfaces. This pattern is referred to as the grid topology. The definition of grid topology is given by a topology input language (TIL). This is also used to specify the number of grid points and clustering criteria. Once the TIL code is in place, the grid is generated as a batch job in a manner which is steered by a dynamic schedule file. The latter file allows the user a great deal of flexibility. For example, rather than just run the case and write out the results, the user may desire to start with fewer grid points or lesser clustering and then to gradually increase at latter steps in the evolution toward convergence. A common motivation is for more efficiency when only modest computational resources are available for the size of problem at hand.

Our discussion will start with the implicit geometry modeling for filleted intersecting tubes and will continue with the development of TIL code for the generation of high quality (smooth and nearly orthogonal) grids inside the multi-tube configurations. Examples will be given to show the variety of configurations, the grid and surface quality, and the ease with which these items can be created and presented for analysis. The multiplely branched tubes are frequently encountered surfaces in geometry modeling and grid generation for a variety of different fields. As a natural consequence, the mathematical formula for the surface model should be of considerable interest on its own right.

MULT-TUBE GEOMETRY MODELING

The question of modeling the geometry of tube intersections has been addressed before in a piecemeal fashion. The typical approach is to first find the intersection between the tubes. Once the intersection is known, the next step is to move away from that curve of intersection to create displaced curves on each tube at some reasonable distance from the intersection. This displacement is most often formed moving a fixed distance along geodesic paths that emanate orthogonally from the curve of intersection. With these displaced curves along each tube, the fillet is formed by a curve of interpolation between them. This starts on one tube with the displaced position and available geodesic tangent direction and ends on the other with the same conditions. When taken together, a collar type surface is formed. The collar connects the tubes with first derivative continuity. This process requires the solution of a surface surface intersection problem, the computation of geodesic distances to produce the displaced curves, and the act of interpolation to construct the collar surface. For this work, there are three surfaces required to define the intersection of two tubes in a T-type configuration. The result is only first derivative continuous. For each application of this technique, a collar is added. For example, with full tubes along the each coordinate axis, there would be 12 surfaces (the tubes for x, y, z, $-x$, $-y$, and $-z$ plus 6 collars). Moreover, should a tube aspect ratio or squareness be added, then the associated problems of construction would be more intensive.

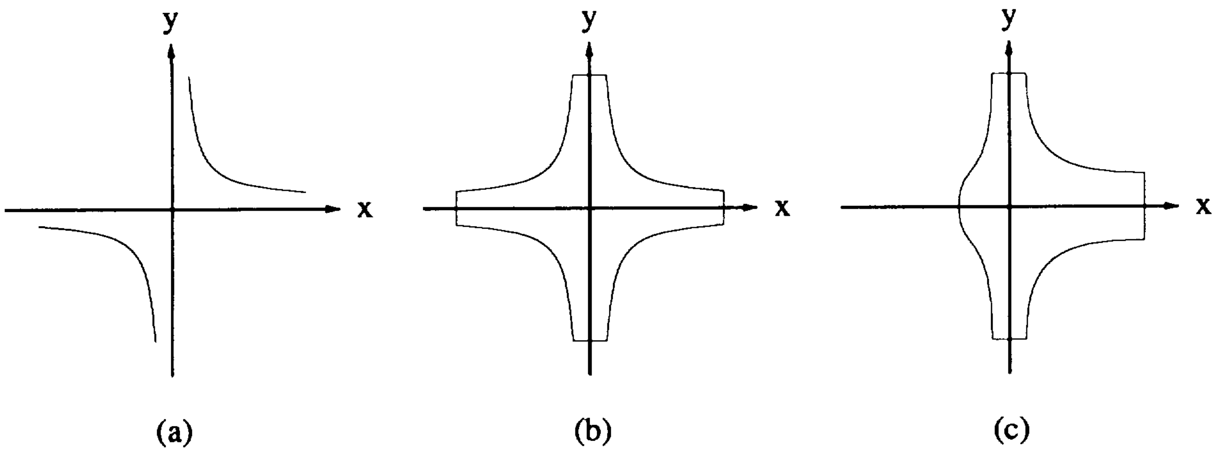


Figure 1: (a). The zero set of $U = 1 - xy$ is shown. (b). The merge of all axial channels are shown. The curved walls are given by either $U = 1 - x^2y^2$ or the more general $U = a + bx^2 + cy^2 - x^2y^2$. (c). The coefficient b is blended from -1 along $x < 0$ to 1 along $x > 0$.

Unlike the traditional means for fillet creation, we offer a technique which does it all with one infinitely differentiable surface. Moreover, each tube can be assigned a cross-sectional squareness and aspect ratio.

Two dimensional models

Our examination of the geometry modeling process will start with two dimensions and will lead by intuition. This should provide simplicity and clarity to the discussion of the implicit methodology. In keeping with this objective, our first problem is to attempt to define the geometry of four channels which follow the (x, y) coordinate axes and come together at the origin. If we examine the function $y = 1/x$, then immediately we have two curves which asymptote to the axes. As we go towards plus and minus infinity, it approaches the four semi-axes. Unfortunately, there are only two curves associated with this function. One is in the first quadrant while the second is in the third quadrant. Thus, only one half of each channel can be bounded. In order to correct this deficiency, we note that general shape is close to what we want, but that we need to also get the same curves on the opposite side of each semi-axis. This can be readily accomplished by simply taking the square of the equation that we started with. This means that both the original equation $y = 1/x$ as well as the additional equation $y = -1/x$ are satisfied when we use $x^2 = 1/y^2$. It is the additional equation that inserts the previously missing curve components for the second and fourth quadrants. The result is symmetric as is the equation. These are illustrated in Fig 1 (a) and (b).

The above equations between x and y can also be readily expressed in the form of $U(x, y) = 0$. In particular, the symmetric equation is defined with $U(x, y) = 1 - x^2y^2$. Alternatives are provided by a multiplication with any strictly positive or negative quantity. Such multiplication's will not change the 0 set of $U(x, y)$. That is, the level curves at constant 0 as defined by $U(x, y) = 0$ will be the same. With the current choice, the value along each axis will be unity. This will decay as the axial regions are departed. The decay will continue through the level curves that bound the channels and be negative outside of them. Because of this continuous decay, the gradient of $U(x, y)$ will be pointing into the channels from each point on the channel walls. As we will see, a requirement for the surface definition is that it be oriented in such a way that the wall normal vectors point into the region to be gridded. Thus, this choice of sign, will at least, possess the convenience of providing the gradients with the preferred orientation. To complete the definition of the physical region, each channel must have an end. This is easily accomplished by specifying four straight lines which successively cap off each channel with a perpendicular slice and with normals that are oriented into the physical region. In level surface form, a straight line with normal (a, b) is given by $U(x,y) = ax + by + c$ (Fig 1 (b)). Should a surface be given where the normal is not in the correct direction, then a reversal of orientation can also applied in the TIL code with the flag "-o".

As one might wish to make the channels arbitrarily long with constant cross-section, a deficiency immediately appears. That is because the channel width is decreasing at a rate inversely proportional to the distance away from the origin. A partial solution to this problem is to insert a parameter a into the level surface definition to arrive at $U(x,y) = a - x^2y^2$. Then, at least, the value of a can be chosen relative to the distance from the origin. This will assure that the cross-section at the cap is not too close to 0. However, it will also mean that the channels may be too fat at closer distances. Thus, a more substantial correction is needed. For this purpose, the function U is generalized to admit a more desirable asymptotic behavior. It is given by

$$U(x,y) = a + b \cdot x^2 + c \cdot y^2 - x^2 \cdot y^2 \tag{1}$$

The strategy, here, is to add pure quadratic terms in both x and y so that as either x or y becomes large near an axis, one of the new terms will become significant to the same order of the last term in the expression. Since each of the 4 channels behaves in the same manner, we need to examine only one to see what is happening. Thus, we will focus our attention on the positive x-axis. Specifically, if the position vector $\mathbf{r} = (x, y)$ is moved away from the origin but stays in the region near the positive x-axis, then the only significant term in $U(x,y)$ is $x^2(b - y^2)$. Thus, the channel "radius" is nearly $y = \sqrt{b}$ which means that the channel cross-section is nearly twice that. In a more analytical sense, we can rearrange the equation $U(x,y) = 0$ to cast it into the form

$$y^2 - b = \frac{a + c \cdot y^2}{x^2} \tag{2}$$

Upon examination, the numerator on the right hand side is clearly bounded since the position vector lies near the x axis which in turn implies that y cannot become arbitrarily large. However, the denominator does become arbitrarily large as we go arbitrarily far along the positive x-axis. As a consequence, the right hand side becomes progressively smaller as we travel along the positive x-axis. With this clear observation, we see that the equation rapidly reduces to something like that of setting the left hand side to 0. The error term (call it E) in this process is then the right hand side. It tells us how much deviation there is from our desired asymptotic lines. The asymptotic lines are at $y = \sqrt{b}$ and $y = -\sqrt{b}$. As the channel boundaries emerge from the juncture near the origin, they approach the asymptotic lines from the outside. To insure that they stay on the outside, E must be positive for the actual channel "radius" is $y = \sqrt{b + E}$. This positivity condition then becomes a condition on the parameters which is that $a + bc$ must be positive. With $\sqrt{b}$ and $\sqrt{c}$ given as the radii of the channels along the respective x and y axes, the condition really becomes a condition upon the choice of a (Fig 1 (b)).

At this stage, we have successfully constructed a level surface function for the 2D case of four channels which smoothly come together at the origin. The next step is to see if it can be generalized to a system of fewer channels. For this purpose, we consider the basic form of $U(x,y)$ that was already seen to be successful. Upon examination of the last equation where the right hand side was considered to be a positive error term E, the prior solution would not exist if b were negative rather than positive. With a constant negative value for b, the left hand side is strictly positive and bounded away from 0 while the right hand side approaches 0 with larger values of x. This means that the equation cannot be satisfied for large x. Thus, it must be bounded in x. In fact, by solving for x as a function of y we get

$$x = \pm\sqrt{\frac{a + cy^2}{-b + y^2}} \tag{3}$$

which is certainly well defined when b is negative. This function gives the contour on either side of the y-axis. It is symmetric about the x-axis and crosses it with a value of $\pm\sqrt{\frac{a}{-b}}$. The asymptotic lines are located at $\pm\sqrt{c}$. Thus, the contour will bulge out a distance of $\sqrt{\frac{a}{-b}} - \sqrt{c}$ as it crosses the x-axis. If $a = -bc$, then there is no bulge. This can also be seen by a direct substitution into the above equation since it then reduces to $x = \pm\sqrt{c}$. The result is that the channel walls are exactly the asymptotes.

As we have just witnessed, a negative value of the parameter b means that there is no tube in the corresponding x-axis directions. Unfortunately, it kills off the channels on both the positive and negative x-axis at once. This leaves the rather uninteresting case of a single straight channel. To get to the next level of interest, we need to have a channel on one side but not the other. We shall thus consider the case for a channel along the positive x-axis which fillets into a channel along the entire y-axis. This will require b to be negative along the negative x-axis and positive along the positive x-axis. It also means that b must depend upon x as opposed to its previous role as a constant. Since it still represents the square of the channel radius, it must, at the very least, approach a constant value with increasing x. This will then assure us of a channel with fixed cross-section. To achieve that fixed state within the physical region under consideration, it is important to have the approach be sufficiently rapid. With these motivational facts, we need to use an asymptotic blending function with a rate which can be controlled. An ideal candidate for this purpose is the hyperbolic tangent. To allow all possibilities, we shall write b in the form

$$b = p \cdot f(x) + q \cdot [1 - f(x)] \tag{4}$$

where $f(x) = [1 + \tanh(wx)]/2$

The allowed constants along the respective positive and negative x-axis are respectively p and q. To consider the case of interest, the value of q is negative while the value of p is positive. This means that we will only have a channel of radius $\sqrt{p}$ along the positive x-axis. As the blending function $f(x)$ leaves the origin, it approaches 0 along the negative x-axis and 1 along the positive x-axis. The rapidity with which it approaches each value is controlled by the damping factor w. (see Fig 1 (c)). While the hyperbolic tangent construction is convenient, it is certainly possible to consider alternatives which can serve the same purpose.

One attractive alternative is to consider a rational polynomial. Since the level surface function $U(x, y)$ is a polynomial, this would represent the most pure method since it would keep the entire operation within the domain of polynomials. That is because a coefficient like b appears linearly and thus so does $f(x)$. As a consequence, a multiplication by the polynomial denominator of the rational $f(x)$ will produce a new overall equivalent $U(x, y)$ that is entirely a polynomial. In mathematical terms, this means that we remain within the context of what are called algebraic varieties (a term from algebraic geometry).

With this motivation, we proceed. The conditions which must be met are that the blending function go from a constant value of q along the negative x-axis to another constant value of p

along the positive x-axis. Fo: this to happen asymptotically on either side of the origin, the highest power of x in both the numerator and the denominator must match. The asymptotic value in this process is then the ratio of the coefficients for these respective highest powers. This, however, is just one number. But we actually need two different numbers if we wish to match two asymptotic values. Thus, this technique fails.

The only hope to retrieve it is to allow a large growth in negative values as we migrate along the axis in which there is to be no channel. For example, consider b to be the rational function $x^3/(1+x^3)$. This has a singularity (pole) at $x = -1$. Departing the pole in the positive direction, the function increases from arbitrarily large negative values, passes through the origin, and continues along the positive x-axis to approach the asymptotic value of 1 from below. Departing the pole in the negative direction, the function decreases from arbitrarily large positive values and approaches the same asymptotic value from above. Thus, we must use the function on only the positive side of the pole. With the pole of b at -1, we shall consider a channel radius of 0.5 along the y-axis. This will be a safe distance away. We can then expect to have the wall on the negative side of the vertical channel to pass through the the x-axis ($y = 0$) somewhere between a value of the channel asymptote ($x = -0.5$) and the pole of b (at $x = -1$). That offset distance will be determined by the choice of leading parameter a. To give a specific case, we will set $a = 1$. With this form, it is then natural to view the contour as a function x of the variable y. This function is smooth and symmetric about $x = 0$. Upon evaluation at $y = 0$, the level surface function reduces to a fifth order equation in x which has an approximate solution of $x = -0.84$. By separating out y, an evaluation at the nearby $x = -0.8$ yields $y = \pm 0.92$. Thus we see the trend of going from the negative peak at -0.84 at $y = 0$ and dropping steadily to the asymptotic value of -0.5. Here, the offset distance is 0.34. In summary, we have witnessed that, while it is possible to use rational functions for the blending, more care and cleverness is required in its execution and the overall format is much more restrictive. In this paper, we will thus use the flexibility of transcendental functions so that we can conveniently employ multiple asymptotes in a single blending function. In particular, we shall utilize the hyperbolic tangent.

Three dimensional models

Having established the basic properties in two dimensions, the extension into three dimensions can be discussed with more brevity. To begin, the 2D level surface function $U(x, y) = a - x^2y^2$ is replaced by the 3D generalization [5]

$$U(x, y, z) = 1 - (x^2y^2 + x^2z^2 + y^2z^2) \tag{5}$$

where now $\mathbf{r} = (x, y, z)$ and $r = |\mathbf{r}| = \sqrt{x^2 + y^2 + z^2}$. This gives the parallel to the previous 2D curves that decayed at the rate of $1/r$ as each axis was traversed to infinity. For the same reasons the one can be replaced by a parameter "a" to displace the effect of the decay. As before, this does not remove the decay. The correction to permit constant cross-sections is the same as before. This requires second order terms in each of x, y, and z to asymptotically select the appropriate parts of the core quartic term. This gives us the basic function

$$U(\mathbf{r}) = a + bx^2 + cy^2 + dz^2 - (x^2y^2 + x^2z^2 + y^2z^2) \tag{6}$$

where a, b, c and d are constants.

With a simple analysis, we can show that for $a > -\min\{bc, bd, cd\}$, $b > 0$, Eq. 6 defines a surface of 6 tubes intersecting at origin. The separation in x is explicitly seen by consideration of the reordered function

$$U(\mathbf{r}) = x^2 \cdot (b - y^2 - z^2) + a + c \cdot y^2 + d \cdot z^2 - y^2 z^2 \tag{7}$$

As **r** departs the origin along the x-axis, the first term becomes more and more dominant over the others. In the level surface equation $U(\mathbf{r}) = 0$, a division by x^2 will give an equivalent equation where all terms except the first will approach 0 as x grows in the outward vicinity of the x-axis. In the limit, the equation is just the circular cylinder of radius $\sqrt{b}$ about the x-axis.

The axes of the 6 tubes lay on the $\pm x$, $\pm y$ and $\pm z$ axes respectively. Asymptotically, the cross sections of the tubes are circles of radius $\sqrt{b}$, $\sqrt{c}$ and $\sqrt{d}$ for tubes along the $\pm x$, $\pm y$ and $\pm z$ axes respectively. The asymptotical error in radius is,

$$\Delta R < \frac{a + b + c + d}{2r^2} \tag{8}$$

The value of a determines the abruptness of the intersection. A larger a gives a lesser degree of abruptness of the intersection.

The signs of b, c and d determine whether tubes in the corresponding directions will present. If, say, b is negative, the tubes in the $\pm x$ direction will disappear.

The analysis also shows that for the surface implicitly defined by $U(\mathbf{r}) = const$, it is well behaved near $const = 0$.

The function can be generalized to enable the tube branchs to be switched on and off individually:

$$U(\mathbf{r}) = a + Bx^2 + Cy^2 + Dz^2 - (x^2y^2 + x^2z^2 + y^2z^2) \tag{9}$$

where B, C and D are functions of **r**. A set of choices are:

$$\begin{aligned} B &= -1 + s_x^+ \cdot (1 + \tanh(4x)) + s_x^- \cdot (1 - \tanh(4x)) \\ C &= -1 + s_y^+ \cdot (1 + \tanh(4y)) + s_y^- \cdot (1 - \tanh(4y)) \\ D &= -1 + s_z^+ \cdot (1 + \tanh(4z)) + s_z^- \cdot (1 - \tanh(4z)) \end{aligned} \tag{10}$$

Here, a particular choice of B, C and D is selected by setting six switch parameters to 0 or 1. Each of the parameters controls one tube branch and a value of 1 (value of 0) means there is (is not) a tube in the corresponding direction.

$$\begin{array}{llll} \mathbf{s_x^+} & = 0 \text{ or } 1 \text{ for tube along } +x \text{ axis.} & \mathbf{s_x^-} & = 0 \text{ or } 1 \text{ for tube along } -x \text{ axis.} \\ \mathbf{s_y^+} & = 0 \text{ or } 1 \text{ for tube along } +y \text{ axis.} & \mathbf{s_y^-} & = 0 \text{ or } 1 \text{ for tube along } -y \text{ axis.} \\ \mathbf{s_z^+} & = 0 \text{ or } 1 \text{ for tube along } +z \text{ axis.} & \mathbf{s_z^-} & = 0 \text{ or } 1 \text{ for tube along } -z \text{ axis.} \end{array} \tag{11}$$

The functional form can further generalized to allow the cross section of the tubes to be superellipses of different degrees. The further generalized formulas are listed in Appendix A.

GRID GENERATION WITH GridPro/az3000

Now we show how one can generate grids for the region inside the branching tubes defined above using **GridPro/az3000**.

The grid generation process with **GridPro/az3000** starts with the design of a block topology (domain decomposition into hexahedras). Let's first consider the case where all 6 tube branches are present. The surface under consideration is

$$U(\mathbf{r}) = 2 + x^2 + y^2 + z^2 - (x^2y^2 + x^2z^2 + y^2z^2).$$

Here, all 6 tube branches have the same radius, 1. We also decide that the grid will be generated in the region $-5 < x < 5$, $-5 < y < 5$ and $-5 < z < 5$.

In order to have the flexibility of changing configuration easily, we will build the block topology in a component style. A component in **GridPro/az3000** represents a subtopology or a portion of the block topology that is conveniently grouped such that it can be reused in a similar fashion as a subroutine can in, say, Fortran.

How a block topology should be designed and how the various components should be chosen depend on many factors. One of our considerations here is to be able to change the configuration to include a different number of tubes rather quickly.

At this step, we choose to use two top level components, a `center` component for the center intersection and a `tube` for a tube branch. Each of these two components is constructed by one or more copies of the component, `sec`. The `sec` component, in turn, is constructed by properly linking two `loop4` components into a hyperquad.

These topology components are shown in Fig 2. Here, a solid dot is a block corner defined in the component, a solid line is a block edge defined in the component, and a circle is an imported corner into the component. On the corner and link level, the components `sec` and `tube` look the same. They differ mainly in whether the outer quad defines a block face. It is a face for `sec`, and it is not a face for `tube`.

To run **GridPro/az3000**, the topology design or components must be programmed in the Topology Input Language(TIL). The complete TIL code for this case is listed in Appendix B. In the following, we focus on three components to illustrate the general flavor of TIL programming.

The most basic component in the above design is `loop4`. In TIL code it looks like this :

Program 1 *TIL component* `loop4`

```
COMPONENT loop4(sIN surf1,surf2, cIN pos[1..4],corn1[1..4],corn2[1..4])
BEGIN
 c 1 @<pos:1> -s surf1 surf2 -L corn1:1 corn2:1;
 c 2 @<pos:2> -s surf1 surf2 -L corn1:2 corn2:2 1;
 c 3 @<pos:3> -s surf1 surf2 -L corn1:3 corn2:3 2;
 c 4 @<pos:4> -s surf1 surf2 -L corn1:4 corn2:4 3 1;
END
```

This component imported two surfaces `surf1` and `surf2`, and three corner arrays of length 4, `pos[1..4]`, `corn1[1..4]` and `corn2[1..4]`. `surf1`, `surf2`, `pos`, `corn1` and `corn2` are similar to what are termed as dummy variables in Fortran subroutines. They do not introduce new surfaces or corner to the topology. Instead, they only provide a mechanism to refer to existing corners and surfaces defined outside of the component. Corners and surfaces are constructed with corner and surface definition statements in components, and by `INPUT`ing components that have corner and surface definition statements in them. In `loop4`, 4 corners are defined. For every call to this

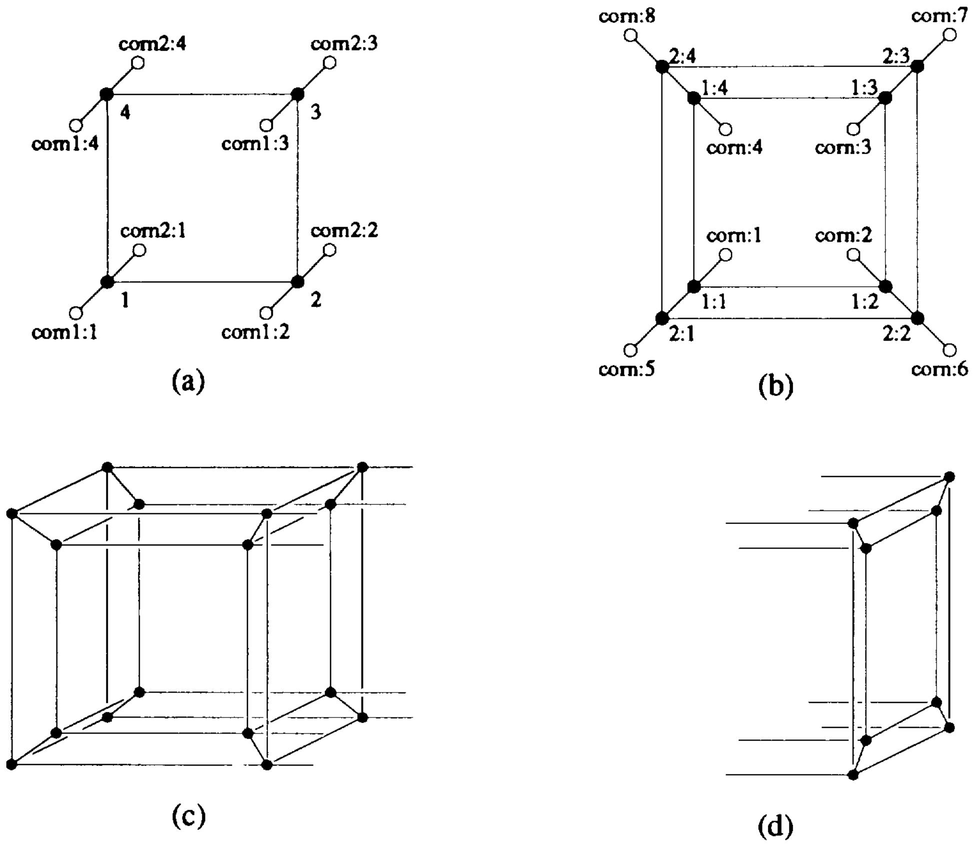


Figure 2: Topology components. (a). The most basic component, **loop4**. (b). The component, **sec**. It is constructed from two **loop4**'s. (c). The component, **center**. It is constructed from two **secs**. Also, 8 open links for connecting to a tube branch are shown. (d). The component, **tube**. It is constructed by a **sec** with certain boundary conditions. Shown are 8 open links for connecting to the **center**.

component, 4 corners will be constructed and inserted into the topology. A corner is generally defined by specifying a corner id, an initial position, a list of surfaces that the corner is supposed to be on, and a list of existing corners to which the current corner should have links. The surface list and link list are optional. In this case, the initial positions are provided by the import variable **pos**. All 4 corners are attached to both **surf1** and **surf2**. In addition, the corners are linked to the imported corners **corn1** and **corn2**. They are also properly linked to each other to form a 4-corner loop. The notation **corn1:3** simply means the 3rd element of array **corn1**.

Two **loop4**'s are used to construct the component **sec**.

Program 2 *TIL component* **sec**

```
COMPONENT sec(sIN tube,tube_end,cIN pos[1..4],corn[1..8])
BEGIN
  VECTOR p[1..4],shift;
  INPUT 1 loop4(sIN  (tube),(tube_end),
                cIN  (pos:1..4),(corn:1..4),(-4),
                cOUT (1..4));
  <shift> = 0.05*0.25*(<pos:1>+<pos:2>+<pos:3>+<pos:4>);
  <p:1> = 0.8*<pos:1>+0.2*<pos:3>-<shift>;
  <p:2> = 0.8*<pos:2>+0.2*<pos:4>-<shift>;
  <p:3> = 0.8*<pos:3>+0.2*<pos:1>-<shift>;
  <p:4> = 0.8*<pos:4>+0.2*<pos:2>-<shift>;
  INPUT 2 loop4(sIN  (tube_end),(-1),
                cIN  (p:1..4),(corn:5..8),(1:1..4),
                cOUT (1..4));
END
```

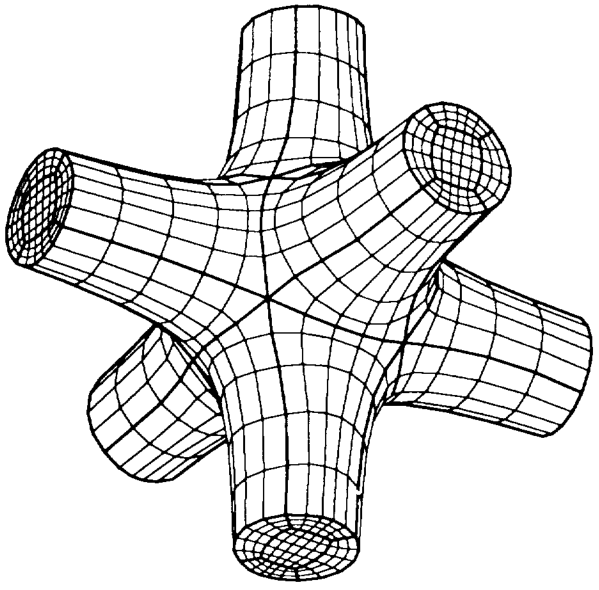
Here, vector operations are used to position the second loop4 relative to the first. The key word cOUT exports some or all corners defined in the inputing component. Other components are similarly constructed.

To complete the topology design, we need to have a head component to assemble the various components together. This component must be the first one in the file. For our case, this component is named Branching_Tubes:

Program 3 *TIL component* Branching_Tubes

```
COMPONENT Branching_Tubes()
BEGIN
 VECTOR cut_x,cut_X,cut_y,cut_Y,cut_z,cut_Z;

  s 1 -implic ''6jxXyYzZ.surf'' ; # 6 tubes
# s 1 -implic ''6jxXyYZ.surf'' ;  # 5 tubes
# s 1 -implic ''6jxXyY.surf'' ;   # 4 tubes
# s 1 -implic ''6jXYZ.surf'' ;    # 3 tubes

 <cut_x> = {-5, 0, 0}; <cut_X> = { 5, 0, 0};
 <cut_y> = { 0,-5, 0}; <cut_Y> = { 0, 5, 0};
 <cut_z> = { 0, 0,-5}; <cut_Z> = { 0, 0, 5};

  INPUT 1 center(sIN (1), cOUT (3:1..48));

  INPUT 2 tube_x(sIN (1),cIN (cut_x),(1:1..48));
  INPUT 3 tube_X(sIN (1),cIN (cut_X),(1:1..48));
  INPUT 4 tube_y(sIN (1),cIN (cut_y),(1:1..48));
  INPUT 5 tube_Y(sIN (1),cIN (cut_Y),(1:1..48));
  INPUT 6 tube_z(sIN (1),cIN (cut_z),(1:1..48));
  INPUT 7 tube_Z(sIN (1),cIN (cut_Z),(1:1..48));
END
```

Here a # symbol introduces comments ending at the end of the line. A TIL statement starting with the key word s defines a surface. Here, one surface is defined. It is an implicit type and the surface specification is in the file, 6jxXyYzZ.surf which contains the surface in Eq. 9..12 with all six


(a)

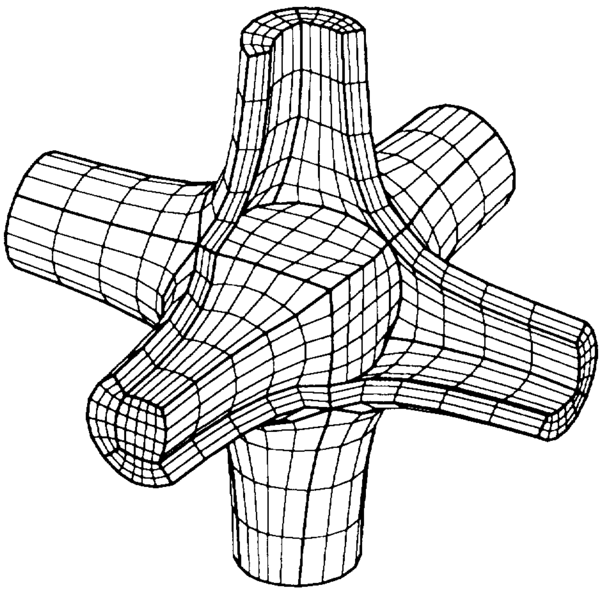

(b)

Figure 3: A grid for the inside of a 6-tube intersection. The thicker lines are block boundaries. (a). an outside view. (b). a cut away view.

tube branches present. The file, 6jxXyYzZ.surf is in the C programming language macro definition style and is listed below.

Program 4 *Surface File* **6jxXyYzZ.surf**

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
               ya=y*y, y1= ya*tanh(y*4), \
               za=z*z, z1= za*tanh(z*4), \
2.0 + xa + ya + za - xa*ya - ya*za - za*xa)
```

For a non-build-in implicit surface, up to 9 pre-named macros must be defined. However, for a fixed surface such as the one we have here, only **FUNCU** is needed. **x**, **y** and **z** are the coordinate variables, and **x1**, **y1**, **z1**, **xa**, **ya** and **za** are intermediate variables. **x1**, **y1** and **z1** are not really used in the final formula. They will be used in a 5-tube surface. They are included here for the reason of easy comparison.

In Program 3, following the surface definition, a **center** component and 6 tube branches components are inputed. Six vectors are used to allow different choices of tube lengths. A high quality grid can be generated by running **GridPro/az3000** on this topology. Fig 3 shows some aspects of the generated grid. It consists of 31 elementary blocks. Grid densities, clustering and other aspects of grids can be readily adjusted by setting proper parameters.

Now, suppose we want to generate a grid for the same surface, but without the tube branch on the -z axis. The head component will be modified to look like this:

Program 5 *Modified TIL component* **Branching_Tubes**

```
COMPONENT Branching_Tubes()
BEGIN
 VECTOR cut_x,cut_X,cut_y,cut_Y,cut_z,cut_Z;

# s 1 -implic ''6jxXyYzZ.surf'' ; # 6 tubes
  s 1 -implic ''6jxXyYZ.surf'' ;  # 5 tubes
# s 1 -implic ''6jxXyY.surf'' ;   # 4 tubes
# s 1 -implic ''6jXYZ.surf'' ;    # 3 tubes

 <cut_x> = {-5, 0, 0}; <cut_X> = { 5, 0, 0};
 <cut_y> = { 0,-5, 0}; <cut_Y> = { 0, 5, 0};
 <cut_z> = { 0, 0,-5}; <cut_Z> = { 0, 0, 5};

  INPUT 1 center(sIN (1), cOUT (3:1..48));

  INPUT 2 tube_x(sIN (1),cIN (cut_x),(1:1..48));
  INPUT 3 tube_X(sIN (1),cIN (cut_X),(1:1..48));
  INPUT 4 tube_y(sIN (1),cIN (cut_y),(1:1..48));
  INPUT 5 tube_Y(sIN (1),cIN (cut_Y),(1:1..48));
# INPUT 6 tube_z(sIN (1),cIN (cut_z),(1:1..48));
  INPUT 7 tube_Z(sIN (1),cIN (cut_Z),(1:1..48));
END
```

We used a different surface specification file "6jxXyYZ.surf", which has only 5 tube branches on $\pm x$, $\pm y$ and $+z$ axes respectively. To accomplish this in "6jxXyYZ.surf", the **FUNCU** macro is defined as:

Program 6 *Surface File* 6jxXyYZ.surf

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
               ya=y*y, y1= ya*tanh(y*4), \
               za=z*z, z1= za*tanh(z*4), \
        2.0 + xa + ya + z1 - xa*ya - ya*za - za*xa)
```

The second thing we did in Program 5 is to comment out **INPUT 6**, which builds the sub-topology for a tube branch on the $-z$ axis. Running **GridPro/az3000** on this topology yields the grid shown in Fig 4. Similarly, we can define a surface that has 4 tube branches or 3 tube branches, and appropriately modify the component **Branching_Tubes** for the corresponding topologies. The grids are shown in Fig. 5.

CONCLUSION

We have now witnessed an application of implicit surfaces in the full cycle of geometry modeling and grid generation using **GridPro/az3000**. We have noted the high quality of both the geometry model and the generated grid therein. We have also noted the ease with which changes can be made

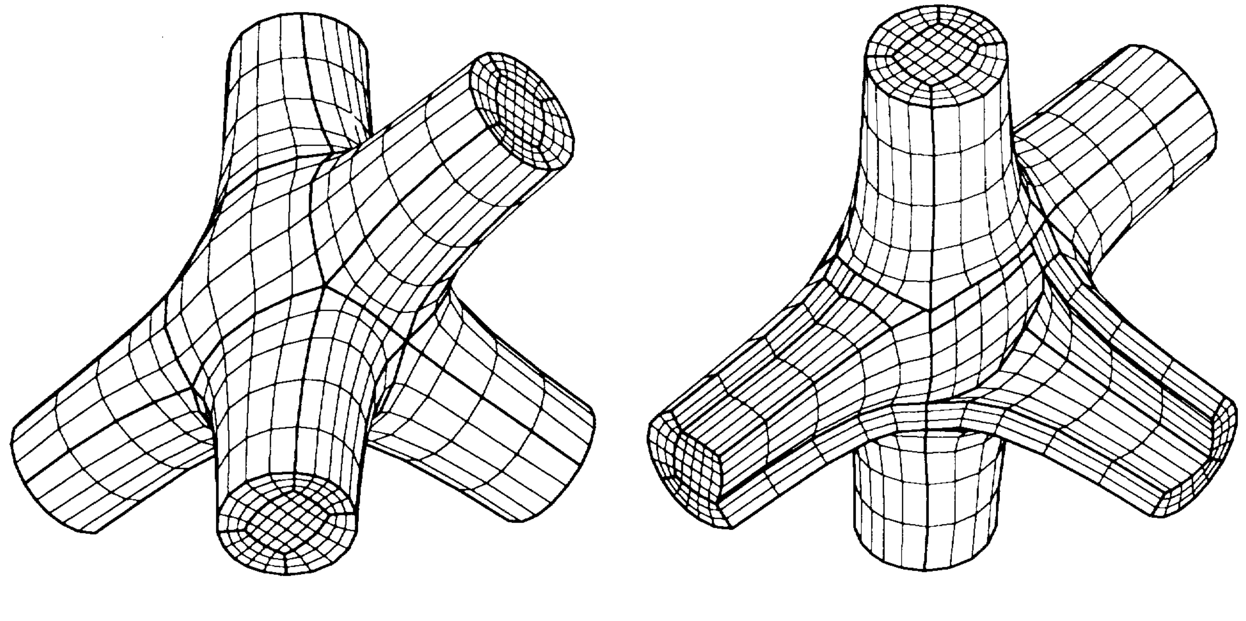


Figure 4: A grid for the inside of a 5-tube intersection. The thicker lines are block boundaries. (a). an outside view. (b). a cut away view.

to both aspects. The example class of multi filleted tubes has demonstrated a general philosophy in a rather concrete setting which helps to establish a basic understanding.

In a forthcoming paper, we will develop a well-defined and easy-to-follow procedure for implicit surface modeling which allows the user to efficiently assemble simple implicit surfaces (e.g cylinders) into a complex whole (e.g intersecting cylinders). With these results, we have provided a powerful means to address many analysis applications, and moreover, have opened a path in the direction of rapid prototyping in grid generation.

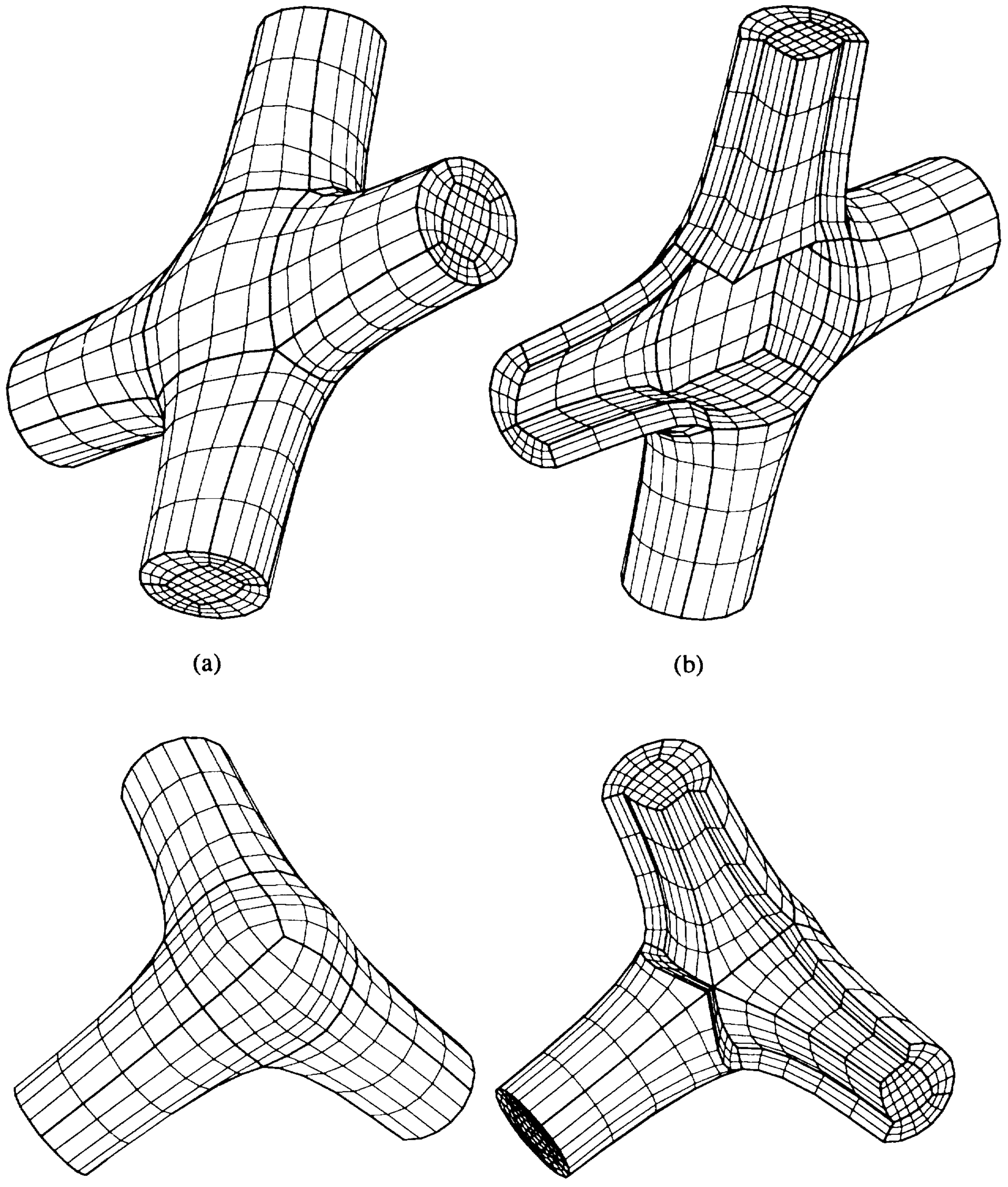


Figure 5: Grid for the inside of a 4-tube and 3 tube intersections. The thicker lines are block boundaries. (a). an outside view for the 4-tube case. (b). a cut away view for the 4-tube case. (c). an outside view for the 3-tube case. (d). a cut away view for the 3-tube case.

APPENDIX A

To enlarge the scope of the geometry modeling for tube intersections, the function is generalized to allow the cross section of the tubes to be superellipses of different degrees that can be switched on and off individually. This generalization is given by,

$$U(\mathbf{r}) = A + BX^2 + CY^2 + DZ^2 - (X^2Y^2 + X^2Z^2 + Y^2Z^2)$$

where A, B, C, D, X, Y and Z are functions of $\mathbf{r}$.

$$X = |\frac{x}{S_x(\mathbf{r})}|^{n(\mathbf{r})} , \quad Y = |\frac{y}{S_y(\mathbf{r})}|^{n(\mathbf{r})} , \quad Z = |\frac{z}{S_z(\mathbf{r})}|^{n(\mathbf{r})} .$$

$$\begin{aligned} n(\mathbf{r}) = \frac{1}{r^2}\{ \quad & [n_x^+ \cdot \frac{1+\tanh(4x)}{2} + n_x^- \cdot \frac{1-\tanh(4x)}{2}] \cdot x^2 \\ + \quad & [n_y^+ \cdot \frac{1+\tanh(4y)}{2} + n_y^- \cdot \frac{1-\tanh(4y)}{2}] \cdot y^2 \\ + \quad & [n_z^+ \cdot \frac{1+\tanh(4z)}{2} + n_z^- \cdot \frac{1-\tanh(4z)}{2}] \cdot z^2\} . \end{aligned}$$

$$\begin{aligned} S_x(\mathbf{r}) = \frac{1}{r^2}\{ x^2 \quad + \quad & [a_{xy}^+ \cdot \frac{1+\tanh(4y)}{2} + a_{xy}^- \cdot \frac{1-\tanh(4y)}{2}] \cdot y^2 \\ + \quad & [a_{xz}^+ \cdot \frac{1+\tanh(4z)}{2} + a_{xz}^- \cdot \frac{1-\tanh(4z)}{2}] \cdot z^2\} . \end{aligned}$$

$$\begin{aligned} S_y(\mathbf{r}) = \frac{1}{r^2}\{ \quad & [a_{yx}^+ \cdot \frac{1+\tanh(4x)}{2} + a_{yx}^- \cdot \frac{1-\tanh(4x)}{2}] \cdot x^2 + y^2 \\ + \quad & [a_{yz}^+ \cdot \frac{1+\tanh(4z)}{2} + a_{yz}^- \cdot \frac{1-\tanh(4z)}{2}] \cdot z^2\} . \end{aligned}$$

$$\begin{aligned} S_z(\mathbf{r}) = \frac{1}{r^2}\{ \quad & [a_{zx}^+ \cdot \frac{1+\tanh(4x)}{2} + a_{zx}^- \cdot \frac{1-\tanh(4x)}{2}] \cdot x^2 \\ + \quad & [a_{zy}^+ \cdot \frac{1+\tanh(4y)}{2} + a_{zy}^- \cdot \frac{1-\tanh(4y)}{2}] \cdot y^2 + z^2\} . \end{aligned}$$

$$\begin{aligned} A &= constant(\text{say}, 1.2) \\ B &= -1 + s_x^+ \cdot (1 + \tanh(4x)) + s_x^- \cdot (1 - \tanh(4x)) \\ C &= -1 + s_y^+ \cdot (1 + \tanh(4y)) + s_y^- \cdot (1 - \tanh(4y)) \\ D &= -1 + s_z^+ \cdot (1 + \tanh(4z)) + s_z^- \cdot (1 - \tanh(4z)) \end{aligned}$$

These functions are controlled by a set of parameters. They are,

$\mathbf{s_x^+}$	$= 0$ or 1 for tube along $+x$ axis.	$\mathbf{s_x^-}$	$= 0$ or 1 for tube along $-x$ axis.
$\mathbf{s_y^+}$	$= 0$ or 1 for tube along $+y$ axis.	$\mathbf{s_y^-}$	$= 0$ or 1 for tube along $-y$ axis.
$\mathbf{s_z^+}$	$= 0$ or 1 for tube along $+z$ axis.	$\mathbf{s_z^-}$	$= 0$ or 1 for tube along $-z$ axis.

n_x^+ – power of super ellipse for tube along $+x$ axis.
n_x^- – power of super ellipse for tube along $-x$ axis.
n_y^+ – power of super ellipse for tube along $+y$ axis.
n_y^- – power of super ellipse for tube along $-y$ axis.
n_z^+ – power of super ellipse for tube along $+z$ axis.
n_z^- – power of super ellipse for tube along $-z$ axis.

a_{xy}^+ – length of x-semi axis for tube on $+y$ axis.	a_{xz}^+ – length of x-semi axis for tube on $+z$ axis.
a_{xy}^- – length of x-semi axis for tube on $-y$ axis.	a_{xz}^- – length of x-semi axis for tube on $-z$ axis.
a_{yx}^+ – length of y-semi axis for tube on $+x$ axis.	a_{yz}^+ – length of y-semi axis for tube on $+z$ axis.
a_{yx}^- – length of y-semi axis for tube on $-x$ axis.	a_{yz}^- – length of y-semi axis for tube on $-z$ axis.
a_{zx}^+ – length of z-semi axis for tube on $+x$ axis.	a_{zy}^+ – length of z-semi axis for tube on $+y$ axis.
a_{zx}^- – length of z-semi axis for tube on $-x$ axis.	a_{zy}^- – length of z-semi axis for tube on $-y$ axis.

APPENDIX B

Files used to generate the grids with **GridPro/az3000**.

Program 7 *Topology File* `6joint.fra`

```
SET GRIDDEN 6

COMPONENT Branching_Tubes()
BEGIN
 VECTOR shift_x,shift_X,shift_y,shift_Y,shift_z,shift_Z;

  s 1 -implic "6jxXyYzZ.surf" ; # 6 tubes
# s 1 -implic "6jxXyYZ.surf" ;  # 5 tubes
# s 1 -implic "6jxXyY.surf" ;   # 4 tubes
# s 1 -implic "6jXYZ.surf" ;    # 3 tubes

 <shift_x> = {-5, 0, 0}; <shift_X> = { 5, 0, 0};
 <shift_y> = { 0,-5, 0}; <shift_Y> = { 0, 5, 0};
 <shift_z> = { 0, 0,-5}; <shift_Z> = { 0, 0, 5};

 INPUT 1 center(sIN (1), cOUT (3:1..48));

 INPUT 2 tube_x(sIN (1),cIN (shift_x),(1:1..48));
 INPUT 3 tube_X(sIN (1),cIN (shift_X),(1:1..48));
 INPUT 4 tube_y(sIN (1),cIN (shift_y),(1:1..48));
 INPUT 5 tube_Y(sIN (1),cIN (shift_Y),(1:1..48));
 INPUT 6 tube_z(sIN (1),cIN (shift_z),(1:1..48));
 INPUT 7 tube_Z(sIN (1),cIN (shift_Z),(1:1..48));
END
```

```
# lines below are hidden from novice users
COMPONENT tube_x(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:1..8)); END

COMPONENT tube_X(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:9..16)); END

COMPONENT tube_y(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:17..24)); END

COMPONENT tube_Y(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:25..32)); END

COMPONENT tube_z(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:33..40)); END

COMPONENT tube_Z(sIN tube,cIN shift,corn[1..48])
BEGIN INPUT 1 tube(sIN (tube),cIN (shift),(corn:41..48)); END

COMPONENT center(sIN tube)
BEGIN
  VECTOR pos[1..4],x,y,z;
  <x> = {1.5,0,0}; <y> = {0,1.5,0}; <z> = {0,0,1.5};
  <pos:1> =  <x> + <y> + <z>; <pos:2> = -<x> + <y> + <z>;
  <pos:3> = -<x> - <y> + <z>; <pos:4> =  <x> - <y> + <z>;
  INPUT 1 sec(sIN  (tube),(-1),
      cIN  (pos:1..4),(-8),     cOUT (1:1..4 2:1..4));
  <pos:1> =  <x> + <y> - <z>; <pos:2> = -<x> + <y> - <z>;
  <pos:3> = -<x> - <y> - <z>; <pos:4> =  <x> - <y> - <z>;
  INPUT 2 sec(sIN  (tube),(-1),
              cIN  (pos:1..4),(1:1..8), cOUT (1:1..4 2:1..4));

  INPUT 3 shuffle(cIN (1:2 2:2 2:3 1:3 1:6 2:6 2:7 1:7
       1:1 2:1 2:4 1:4 1:5 2:5 2:8 1:8
       1:4 1:3 2:3 2:4 1:8 1:7 2:7 2:8
       1:1 1:2 2:2 2:1 1:5 1:6 2:6 2:5
       2:1..8 1:1..8),
                 cOUT (corn:1..48),
       tube_x0(corn:1..8),  tube_x1(corn:9..16),
                       tube_y0(corn:17..24),tube_y1(corn:25..32),
                       tube_z0(corn:33..40),tube_z1(corn:41..48));
  g 1:1 1:5 3;
  LABEL SHELL = e(1:1 1:5);
END

COMPONENT shuffle(cIN corn[1..48]) BEGIN END

COMPONENT tube(sIN tube,cIN shift,corn[1..8])
```

```
BEGIN
  VECTOR pos[1..4],norm;
  <norm> = [-<shift>];

  s 1 -plane @(<norm>, <shift>);    # tube end

  <pos:1> = 0.7*(<corn:1> + <shift>); <pos:2> = 0.7*(<corn:2> + <shift>);
  <pos:3> = 0.7*(<corn:3> + <shift>); <pos:4> = 0.7*(<corn:4> + <shift>);
  INPUT 1 sec(sIN  (tube),(1),
      cIN  (pos:1..4),(corn:1..8), cOUT (1:1..4));
  x f 1:1 1:3 corn:1 corn:3;
END

COMPONENT sec(sIN tube,tube_end,cIN pos[1..4],corn[1..8])
BEGIN
  VECTOR p[1..4],shift;
  INPUT 1 loop4(sIN  (tube),(tube_end),
        cIN  (pos:1..4),(corn:1..4),(-4),
        cOUT (1..4));
  <shift> = 0.05*0.25*(<pos:1>+<pos:2>+<pos:3>+<pos:4>);
  <p:1> = 0.8*<pos:1>+0.2*<pos:3>-<shift>;
  <p:2> = 0.8*<pos:2>+0.2*<pos:4>-<shift>;
  <p:3> = 0.8*<pos:3>+0.2*<pos:1>-<shift>;
  <p:4> = 0.8*<pos:4>+0.2*<pos:2>-<shift>;
  INPUT 2 loop4(sIN  (tube_end),(-1),
cIN  (p:1..4),(corn:5..8),(1:1..4),
        cOUT (1..4));
END

COMPONENT loop4(sIN surf1,surf2,cIN pos[1..4],corn1[1..4],corn2[1..4])
BEGIN
 c 1 @<pos:1> -s surf1 surf2 -L corn1:1 corn2:1;
 c 2 @<pos:2> -s surf1 surf2 -L corn1:2 corn2:2 1;
 c 3 @<pos:3> -s surf1 surf2 -L corn1:3 corn2:3 2;
 c 4 @<pos:4> -s surf1 surf2 -L corn1:4 corn2:4 3 1;
END
```

Program 8 *Schedule File* `6joint.sch`

```
step  1:  -S 100 -v

write -a -D 3 -f grid.tmp
```

Program 9 *Surface File* `6jxXyYzZ.surf`

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4) \
               ya=y*y, y1= ya*tanh(y*4) \
                za=z*z, z1= za*tanh(z*4) \
2.0 + xa + ya + za   -xa*ya- ya*za -za*xa)
```

Program 10 *Surface File* `6jxXyYZ.surf`

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
                ya=y*y, y1= ya*tanh(y*4), \
                za=z*z, z1= za*tanh(z*4), \
2.0 + xa + ya + z1   -xa*ya- ya*za -za*xa)
```

Program 11 *Surface File* `6jxXyY.surf`

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
                ya=y*y, y1= ya*tanh(y*4), \
                za=z*z, z1= za*tanh(z*4),\
2.0 + xa + ya - za   -xa*ya- ya*za -za*xa)
```

Program 12 *Surface File* `6jXYZ.surf`

```
#define FUNCU (xa=x*x, x1= xa*tanh(x*4), \
                ya=y*y, y1= ya*tanh(y*4), \
                za=z*z, z1= za*tanh(z*4), \
2.0 + x1 + y1 + z1   -xa*ya- ya*za -za*xa)
```

References

[1] **GridPro**^{TM}**/az3000** is a software product of the Program Development Corp., 300 Hamilton Ave. Suite 409, White Plains, NY 10601.

[2] Bajaj C. L.; and Ihm I.: Algebraic Surface Design with Hermite Interpolation. *ACM Trans. on Graphics.*, Vol. 11. No. 1, Jan. 1992, pp. 61.

[3] Eiseman P .R.; and Cheng Z.; and Hauser J.: Applications of Multiblock Grid Generations with Automatic Zoning. *Proceedings of the* 4th *International Conference* held at Swansea, Wales 1994, pp. 123.

[4] Program Development Corporation: **GridPro**^{TM}**/az3000** User's Guide and Reference Manual. 1993-1995.

[5] Gray A.: Modern Differential Geometry of Curves and Surfaces. published by CRC Press,Inc. 1993, pp. 226.