

- [52] Sarvajit S. Sinha and Brian G. Schunck "Discontinuity Preserving Surface Reconstruction," *Proceedings of Computer Vision and Pattern Recognition 1991*, pp.229-234, Hawaii, 1991.
- [53] Franc Solina and Ruzena Bajcsy "Recovery of Parametric Models from Range Images: The Case for Superquadrics with Global Deformations," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol.12, NO.2, February 1990, pp.131-147.
- [54] Stoer, J., and Bulirsch, R. 1980, in *Introduction to Numerical Analysis* (New York: Springer-Verlag).
- [55] D. Terzopoulos, and D. Metaxas, "Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 7, July 1991, pp. 703-714.
- [56] Demetri Terzopoulos and Manuela Vasilescu "Sampling and Reconstruction with Adaptive Meshes," *Proceedings of Computer Vision and Pattern Recognition 1991*, pp.70-75, Hawaii, 1991.
- [57] .D. Terzopoulos, A. Witkin, and M. Kass, "Constraints on deformable models: Recovering 3D Shape and Nonrigid Motion," *Artificial Intelligence*, Vol. 36, 1988, pp. 91-123.
- [58] M. Vasilescu and Demetri Terzopoulos. "Adaptive Meshes and Shells: Irregular Triangulation, Discontinuities, and Heretical Subdivision," *Proceedings of Computer Vision and Pattern Recognition 1992* pp. 829-832. Urbana-Champaign, IL, June, 1992.

- [38] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle "Surface Reconstruction from Unorganized Points," *Computer Graphics*, 26, 2, July 1992, pp.71-78.
- [39] W. C. Huang and D. B. Goldgof "ADaptive-Size Meshes for Rigid and Nonrigid Shape Analysis and Synthesis," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 6, June 1993.
- [40] Jacob, David A.H., ed. 1977, in *The State of the Art in Numerical Analysis* (London: Academic Press), pp.259-262.
- [41] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes: Active Contour Models" , in *International Journal of Computer Vision*, January 1988, pp.321-331.
- [42] C. W. Liao, and G. Medioni "Representation of Range Data with B-spline Surface Patches," *Proceedings of International Conference on Pattern Recognition 1992*, pp.745-748, Hague, Netherland, August, 1992.
- [43] Tim McInerney and Demetri Terzopoulos "A Finite Element Model for 3D Shape Reconstruction and Nonrigid Motion Tracking," *Proceedings of International Conference on Computer Vision 1993*, pp.518-523, Berlin, Germany, May, 1993.
- [44] Sylvie Menet, Philippe Saint-Marc, and Gérard Medioni, "B-snakes: implementation and application to stereo," in *Proceedings of Image Understanding Workshop 1990*, pp.720-726, Pittsburgh, September, 1990.
- [45] Shigeru Muraki "Volumetric Shape Description of Range Data using 'Blooby Model'" *Computer Graphics*, Volume 25, Number 4, July 1991, pp.227-235.
- [46] Chahab Nastar and Nicholas Ayache "Fast Segmentation, Tracking, and Analysis of Deformable Objects," technical report, No. 1783, INRIA, France
- [47] Alex Pentland, and Stan Sclaroff, "Closed-Form Solutions for Physically Based Shape Modeling and Recognition" *IEEE trans. on Pattern Analysis and Machine Intelligence*, Vol. 13, No.7, July 1991, pp.715-729.
- [48] Polak, E. 1971, in *Computational Methods in Optimization* (New York: Academic Press).
- [49] William H. Press, Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling, in *Numerical Recipes in C, The Art of Scientific Computing* (Cambridge), Chapter 10.
- [50] Francis J.M. Schmitt, Brian A. Barsky, and Wen-Hui Du "An Adaptive Subdivision Method for Surface-Fitting from Sampled Data" *ACM SIGGRAPH 86*, pp.179-188.
- [51] Sarvajit S. Sinha and Brian G. Schunck "Surface Approximation using Weighted Splines," *Proceedings of Computer Vision and Pattern Recognition 1991*, pp.44-49, Hawaii, 1991.

2. We need a more systematical way to determine the number of the iterations needed. Now the number of iterations is specified by the user. In our experiment, we iterate three times for the complex objects and twice for the simpler objects in our experiments. According to our experiment, reasonable results can be come by in three iterations. Yet, we would like the system to differentiate the complex objects from simple ones by itself, so it can determine the number of iterations.

3. We also would like to find a way to set reasonable  $ERROR_{\text{threshold}}$  and  $RATIO_{\text{ext-to-int}}$  directly by the computer. Now, they are set by the user. Under some situation, we may not know much about the input data, so we would like our system to be able to set these two parameters reasonably under this situation.

4. The self-intersection of the surface is a potential problem, even though we have not observed it in practice. Theoretically, this can be solved by adding an energy term  $E_{\text{intersection}}$ , which is zero when there is no self-intersection, and infinite when the self-intersection occurs. We can rule out this possibility of the self-intersection if there is no self-intersection on the initial surface, because Powell can guarantee that the function value of the result is always less than or equal to the that of the initial guess. The problem with this approach is it is too expensive. It is expensive to check if there is a self-intersection on the surface.

## 4.7 References

- [31] Acton, Forman S. 1970, in *Numerical Methods That Work* (New York: Harper and Row), pp. 464-467.
- [32] Blum, H. "A Transformation for Extracting New Descriptors of Shape," *Proceedings of Symposium on Models for Perception of speech and Visual Form*, W. Whaten-Dunn (ed.), MIT press, Cambridge, Massachusetts, 1967.
- [33] Brent, Richard P. 1973, in *Algorithms for Minimization without Derivatives* (Englewood Cliffs, N.J.: Prentice-Hall), Chapter 7.
- [34] Isaac Cohen, Laurent D. Cohen, and Nicholas Ayache, "Introducing Deformable Surfaces to Segment 3D images and infer differential structures," *Proceedings of Computer Vision and Pattern Recognition 1991*, pp.738-739, Hawaii, 1991.
- [35] H. Delingette, M. Hebert, K. Ikeuchi, "Shape Representation and Image Segmentation Using Deformable Surfaces," *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.467-472, Maui, HI, June 1991.
- [36] Andre Guezic "Large Deformable Splines, Crest Lines and Matching," *Proceedings of International Conference on Computer Vision*, 1993, pp.650-657, Berlin, Germany, May, 1993.
- [37] Song Han, Dmitry B. Goldgof, and Kevin W. Bowyer "Using Hyperquadrics for Shape Recovery from Range Data," *Proceedings of International Conference on Computer Vision* 1993, pp.492-496, Berlin, Germany, May, 1993.

and then the space and time complexities can be reduced significantly. We separate the surface into several strips, so Powell is always called with a limited number of variables. For example, if the fitting surface has  $M \times N$  control points, the maximum number of variables sent to Powell is around  $3 \times (N-4)$ . Only the bad parts of the strips and the meridians are tuned by Powell. So, in practice, the number of variables is far below  $3 \times (N-4)$ . The caps only have  $(N+5)$  variables, which is also low.

4. We reduce the weight of the internal energy implicitly as the iteration goes on, because we have more confidence in the fitting surface after each iteration. This way, the discontinuities of the data can be well preserved.

5. When dealing with an open surface, which is much easier, we just skip the first stage (for the cap).

6. Powell can handle many kinds of functions. This gives us flexibility to define the energy for any specific property. Powell may also be replaced by other numerical methods. Compared to gradient descent, Powell is more accurate and reliable, and in our application Powell is not necessarily slower than gradient descent. From our experiments, we know our algorithm has significant tolerance against a bad initial surface. We attribute this stability and tolerance to minimization algorithm.

7. Through the coarse-to-fine approach, both the global structure and the detailed information on the fine parts of the object can be acquired at different iterations. The global information, which might be applied to recognition and database search, is obtained in the first iteration, and the details of the object can be obtained in the later iterations.

8. Due to the independency among the caps and meridians, our algorithm could run in parallel, so the computational time could be further reduced significantly by parallel processing.

9. This system is easy to control because there are only two global parameters to adjust. In all of our experiments, the same values were used.

10. Overall, this system is reasonable in robustness, accuracy, time complexity, and space complexity. Its output surface can be easily taken by the other surface generating algorithm to construct a smooth surface.

There are also some problems left:

1. We cannot handle the object with deep cavities, especially the through ones. This problem is alleviated by the short-distance external energy, but it is not solved yet. Furthermore, we also cannot differentiate more than two objects when they are very close to one another, and we might mistake them for one object. We are developing some algorithms in 2D to tackle these problems, and we believe they can be extended to 3D in the future.

areas which leads to these artifacts. In the other places, the errors are reasonable. In

**Table 5: Errors**

	size of the external energy cube	final average error in voxels	[0,3)	[3,6)	[6,9)	[9,12)	[12,15]	>15	total
peanut	200X200X200	0.224	4239	310	25	85	0	0	4659
ellipse	200X200X200	0.5	1926	143	54	47	5	0	2175

(c) 4659 surface points are sampled. 4239 points have an error between  $0^\circ$  and  $3^\circ$ , 310 points between  $3^\circ$  and  $6^\circ$ , 25 points between  $6^\circ$  and  $9^\circ$ , 85 points between  $9^\circ$  and  $12^\circ$ , and no points have an error more than  $12^\circ$ . In (g) 2175 surface points are sampled. 1926 points have an error between  $0^\circ$  and  $3^\circ$ , 143 points between  $3^\circ$  and  $6^\circ$ , 54 points between  $6^\circ$  and  $9^\circ$ , 47 points between  $9^\circ$  and  $12^\circ$ , 5 points between  $12^\circ$  and  $15^\circ$ , and no points have an error more than  $15^\circ$ . This information is also shown in Table 5:.

## 4.6 Discussion

There are several important aspects in this paper:

1. We do not use the traditional adaptive approach for this application because it is almost impossible to determine whether a patch is good or not correctly. Our new scheme is a coarse-to-fine approach. It divides all patches after each iteration. It is still efficient because if a patch is really good, then the only operation applied to it in the future is just sub-division, which costs very little. This scheme also preserve the rectangular structure of the surface after each sub-division, which makes generating smooth surface easier and cheaper. This approach is free from the degenerate patch problem because a rectangular patch is always divided into 4 rectangular ones.

We prefer the rectangular mesh to the triangular mesh because it is much easier to construct a smoother surface from the rectangular mesh, and the properties, such as derivatives, are much easier to obtain.

2. We use linear B-splines. On the one hand, it is the cheapest, and on the other hand, it might divide the surface into independent strips. If higher degree B-splines are employed, the relationship between the control points and the patches is more complex.

3. There is always a large matrix associated with the minimization algorithm, and the size of the matrix is in proportional to the square of the number of the variables. This might result in the memory explosion if there are many control points to handle at a time. Also, the numerical method goes extremely slow under this situation. We break a 3 dimensional problem down into several 2 dimensional problems,

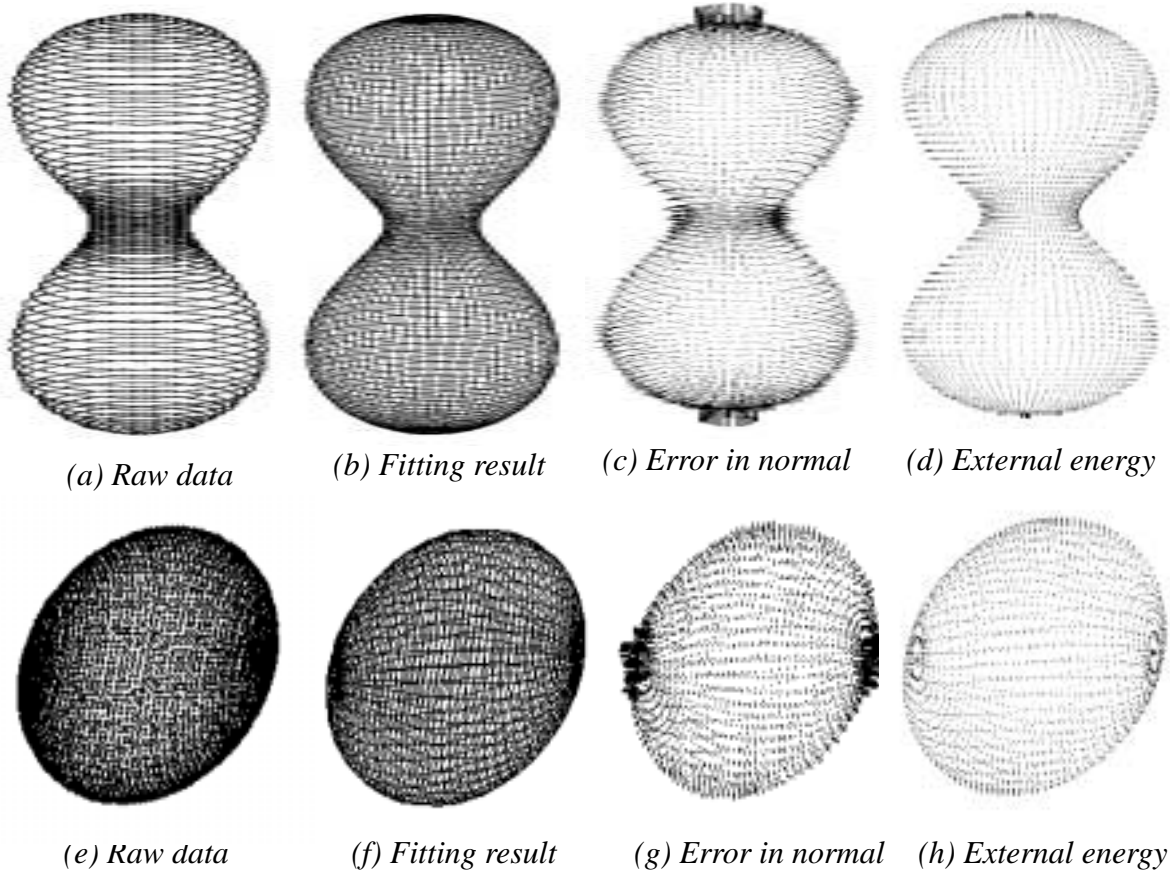


Figure 4.19 Estimates of the error in terms of the normal.

sonable using our approach. We use the following two implicit functions to generate the synthetic data points:

$$(x^2 + y^2 + (z-a)^2) \cdot (x^2 + y^2 + (z+a)^2) = b^4 \quad \text{where } a=67.68, \text{ and } b=68.96.$$

and

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = 1 \quad \text{where } a=80, b=40, \text{ and } c=60.$$

First we sampled points from these two functions as shown in (a) and (c). A 200x200x200 cube is used to store the external energy. (b) and (f) are the fitting results, respectively. (c) and (g) show difference in the surface normal between the fitting surface and the real surface. (d) and (h) show the distribution of the external energy, respectively, and the length of the vector reflect the magnitude of the external energy at this point. The average external energies of (d) and (h) are 0.224 voxels and 0.5 voxels. The length of each vector in (c) and (g) reflects the difference between the two corresponding normals of a surface point. These vectors are exaggerated a little bit so we can see the distribution of the error easier. Please notice high error points concentrate at the poles. It is because we enforce the planar constraint at the polar

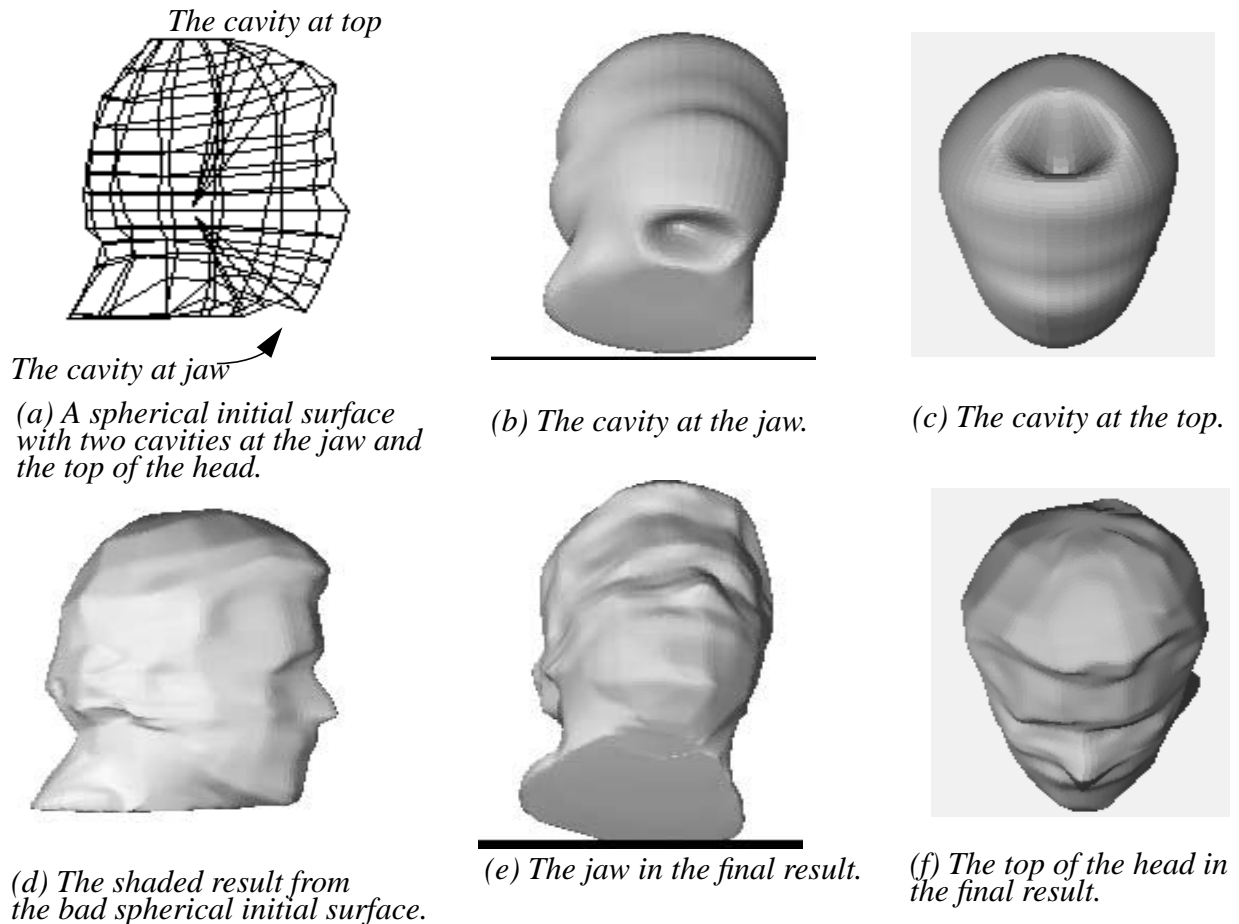
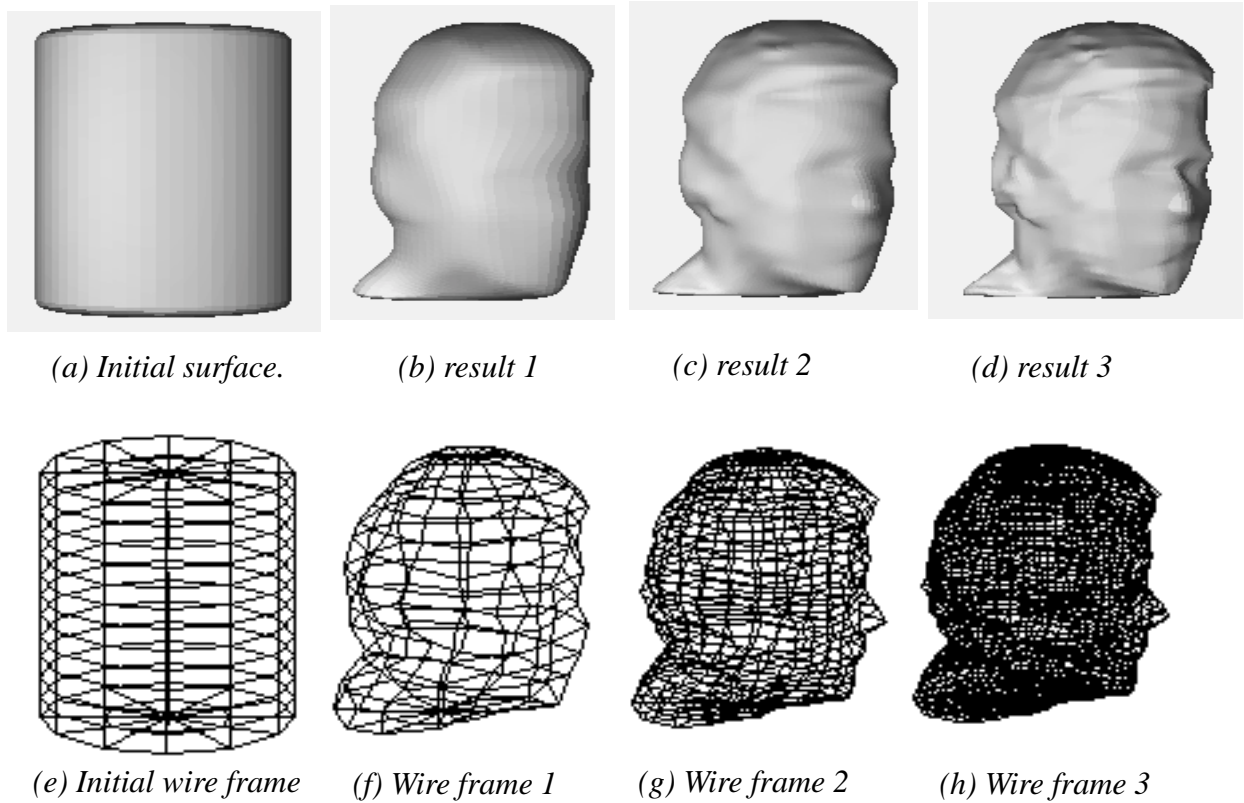


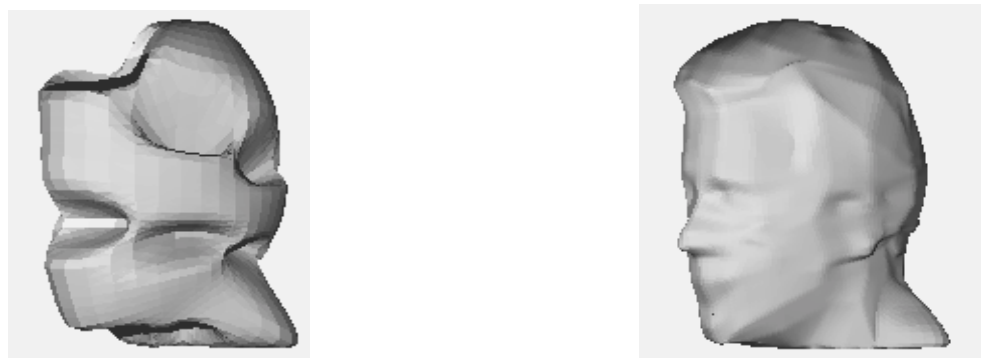
Figure 4.17 A test of the robustness of this algorithm

In head1 and head2 experiments, there are discontinuities at the bottom (for both experiments) and at the top (for head1 experiment). They are flat at those parts. From the results shown so far, we can see those discontinuities come out well because we reduce the importance of the smoothness constraint as the process goes on, and we can handle discontinuities.

In Figure 4.19 we show the difference in surface normal between the fitting surface and the underlying synthetic surface. We conduct this experiment because the normal and the shape of the surface are highly correlated. The normals of the fitting surface and the underlying object might be different while they are physically very close to each other. This experiment is to show how faithful our algorithm is in terms of the normal. Synthetic data are used so we can calculate the normal of the underlying object. It turns out after this experiment that the difference in the normal is rea-



**Figure 4.16** The evolution of the experiment of head 1. (a) is the initial surface (cylinder). (b), (c), and (d) are the deformed results for each iteration. The surface has been sub-divided twice. (e), (f), (g), and (h) show the wire frames of (a), (b), (c), and (d).



(a) One view of the defective initial surface.      (b) The corresponding view of (a) of the final surface.

**Figure 4.18** A test of the robustness on the defective surface with 18 deep cavities. This shows the robustness of this algorithm because it works well when good initial surface is unavailable at the expense of longer computational time.

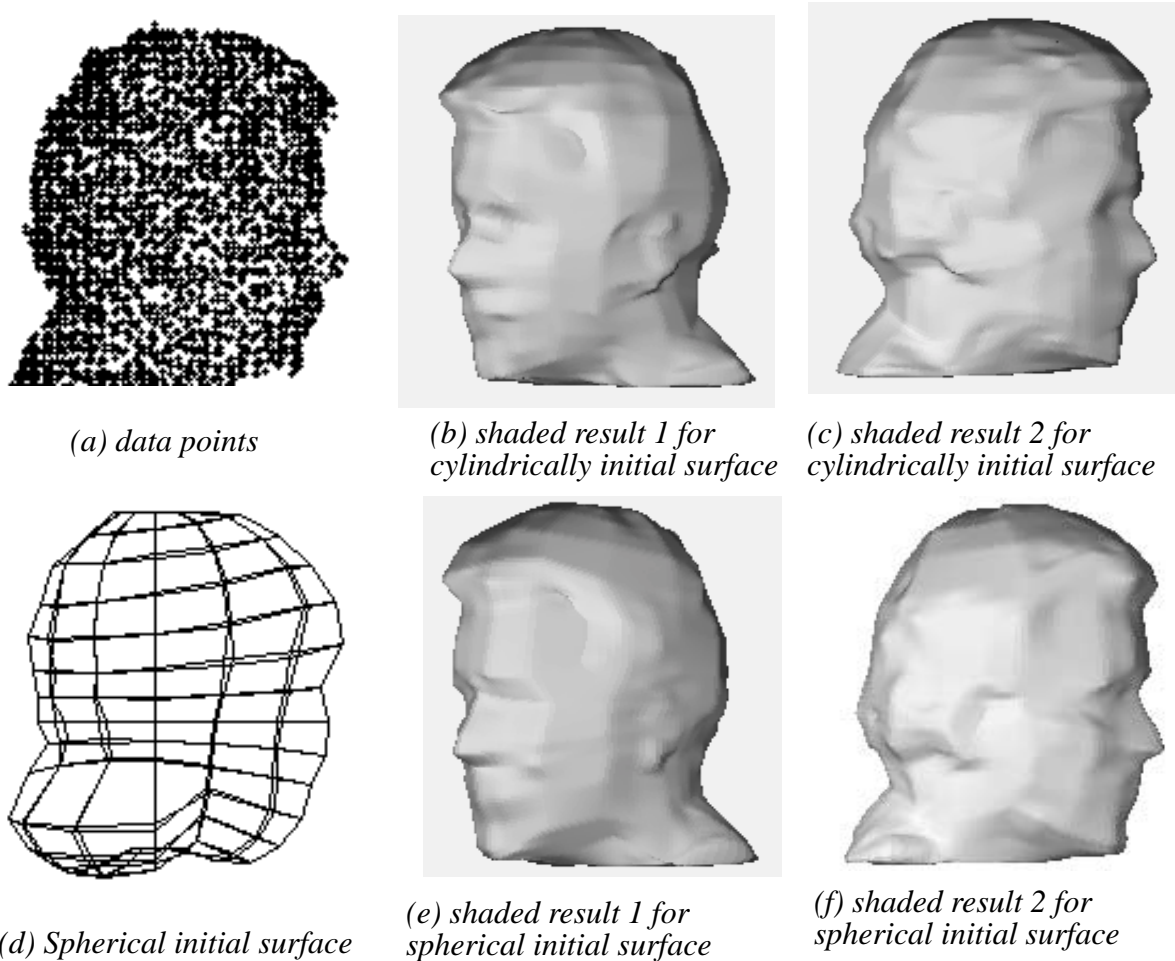


Figure 4.15 Head2

ical initial surface with two cavities at the top of the head and jaw. (b) and (c) show the cavities at the jaw and the top, respectively. (d) is the shaded result after the energy minimization. (e) and (f) show the counterparts of the cavities in (b) and (c), respectively. The cavities have been filled, and the final surface in (d) is fine. In (e) and (f), we can also see the poles are well handled, and the poles on the constructed surface, which are at the top and bottom, are very smooth. The computation is around 24 minutes for the experiment. Based on our experiments on data points head2, this algorithm can tolerate up to 18 cavities, and we obtain similar results in around 24 minutes. Figure 4.18 shows the experiment on the initial surface with 18 deep cavities. (a) shows the defective initial surface, and (b) shows the corresponding views of the final surface. It is evident that all the cavities have been handled. This indicates strong resistance against the bad initial surface.

From the above experiments, we can see that there is not much difference in the quality for these two different initial surface schemes, and this algorithm can also tolerate, to some extent, the error in the initial surface. The main difference is the com-

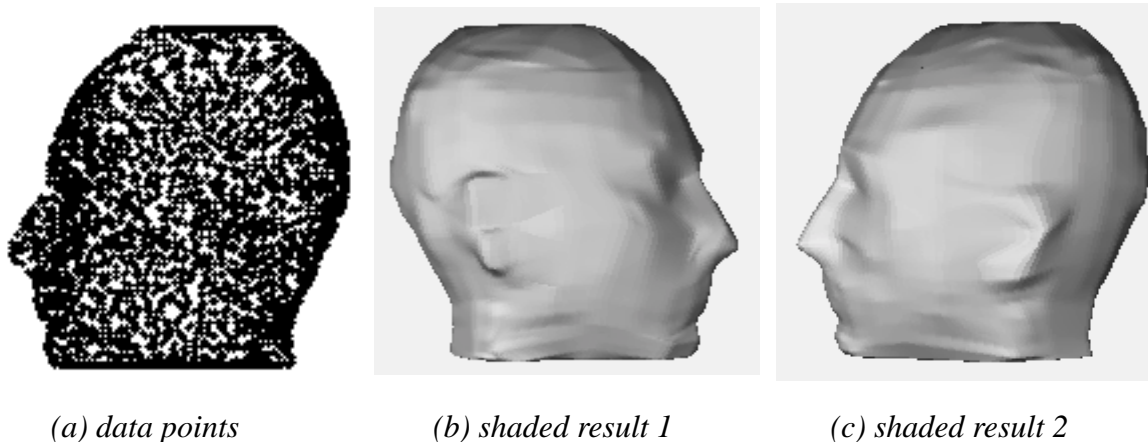


Figure 4.14 Head1 (Carol).

el after the second iteration (one sub-division has been done), and 0.78 voxel after the third iteration (two sub-divisions have been done). It is interesting to note that our number of vertices (2632) is significantly lower than the one reported by others. In Guezic's [36] experiment on this data,  $256 \times 128 (=32768)$  control points are used, and in Nastar's [46] experiment, 11130 nodes are used. One more difference between our result and theirs is we form a closed  $C^1$  surface with the top and bottom closed.

In the head2 experiment, we start with 45514 data points. A  $300 \times 300 \times 300$  cube is used to store the external energy. This set of data is from the Media Lab, MIT. For the same reasons as in the head1 experiment, the number of data points here is different from that of the original one. We tried both algorithms for the initial surface. The running time is around 31.5 minutes for the cylindrically initial surface, and 24 minutes for the spherical one. There are around 2632 control points used on the resultant surface. Figure 4.15 shows the result. In Figure 4.15, (a) shows the data points, and (b) and (c) are the results from the cylindrically initial surface, (d) shows the initial spherical wire-frame surface, and (e) and (f) show the result from the spherical initial surface. Figure 4.16 shows the evolution of the fitting surface with the cylindrically initial surface, and both the shaded and wire-frame results are shown. The average external energy of the surface point is initially 20.15 voxels, 1.43 voxels after the first iteration, 1.21 voxels after the second iteration (one sub-division has been done), and 1.18 voxels after the third iteration (two sub-divisions have been done). In this experiment, we can tell that the result from the spherical initial surface is a little bit better, and can be obtained faster. It is because the spherical initial surface has already captured some geometrical properties, and thus simplifies the calculation to some degree. The surface is subdivided twice, which means there are three iterations.

We also performed an experiment to try the robustness of this algorithm by giving a bad spherical initial surface. The data points here are the same as those of head2 experiment. Suppose there are two directions in which we cannot sample and data point, and the interpolation scheme fails, then there are two deep cavities on the initial surface as shown in Figure 4.17. In (a), we shows the wire frame of the bad spher-

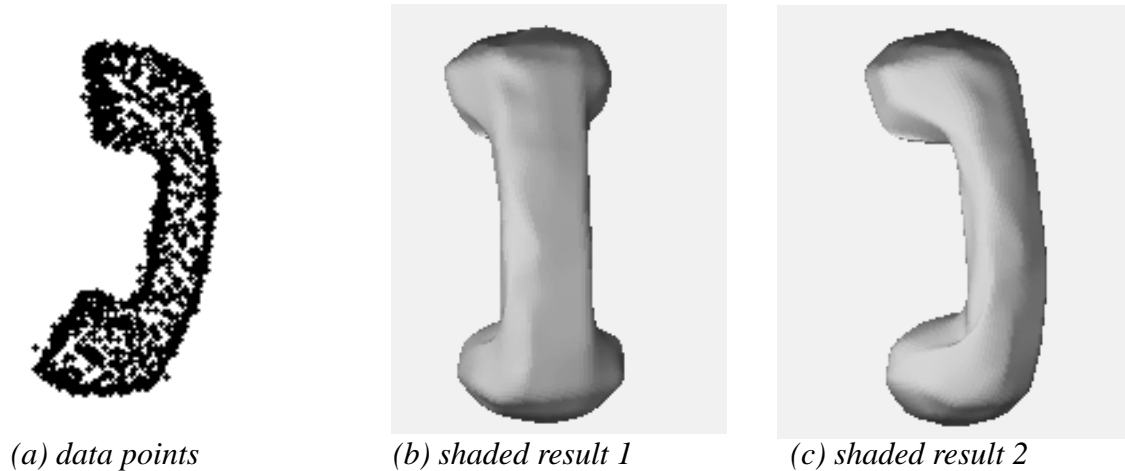


Figure 4.13 Phone

and 0.27 voxel after the second iteration (one sub-division has been done). The distribution of the external energy is shown in (c). The length of each vector in (c) is in proportion to the external energy of the associated surface point.

In the phone experiment, we start with 15776 data points. A  $150 \times 150 \times 150$  cube is used to store the external energy. The running time is around 4.5 minutes. Around 247 control points are used for the resultant surface. The result is in Figure 4.13. The initial surface is a cylinder. The other initial surface construction algorithm cannot be applied because the center of mass falls outside the object and we cannot sample any points in many directions, so the system switches to the cylinder scheme for the initial surface. The surface is subdivided once, which means there are two iterations. The two poles are at the top and the bottom, respectively. The average external energy of the surface point is initially 5.62 voxels, 0.45 voxel after the first iteration, and 0.39 voxel after the second iteration (one sub-division has been done).

In the head1 experiment, there are 136082 data points. A  $300 \times 300 \times 300$  cube is used to store the external energy. This set of data is from INRIA, by courtesy of professor Ayache. We project the data points onto 3D space and fill up the top and bottom of the head, which are hollow originally. We consider points in the same voxel one point, so the number of data points here is different from that of the original one. We also tried both initial surface algorithms. The running time is around 21.5 minutes for the cylindrical initial surface, and 14 minutes for the spherical one. Around 2632 control points are used on the resultant surface. The results from these two different initial surfaces are much alike, so we just show the result in Figure 4.14 from the cylindrical initial surface, which is harder. Figure 4.14 a shows the data points, and Figure 4.14 b and Figure 4.14 c are the results. The computational time is reduced by spherical initial surface because there is already some geometrical information in it. The surface has been subdivided twice, which means there are three iterations. The two poles are at the top and the bottom, respectively. The average external energy of the surface point is initially 18.00 voxels, 1.45 voxels after the first iteration, 0.88 vox-

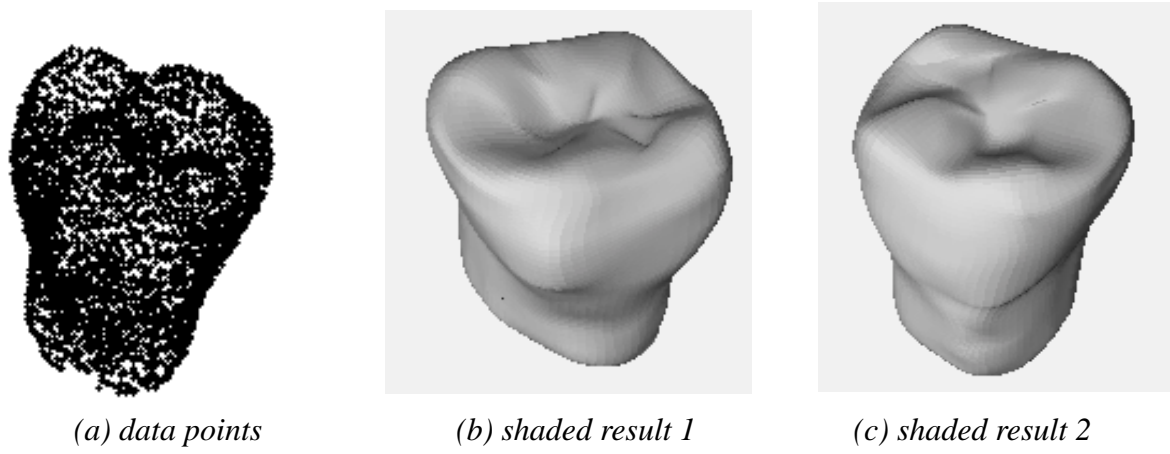


Figure 4.11 Tooth

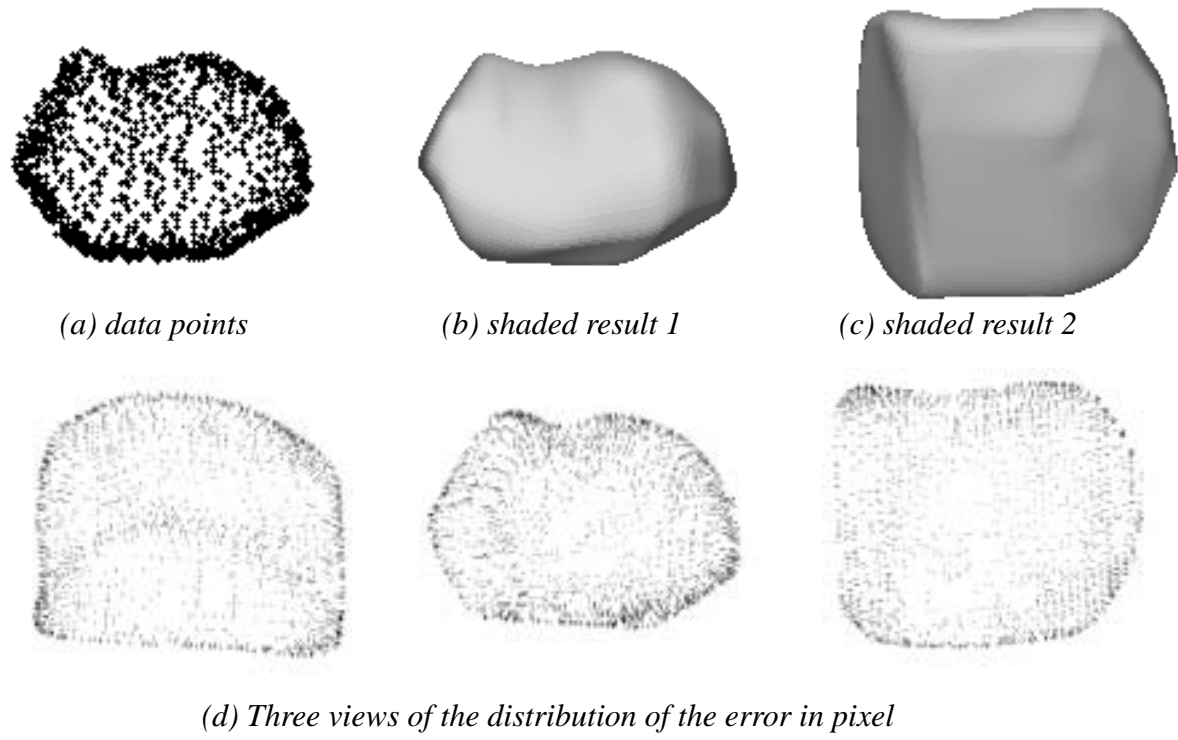


Figure 4.12 Wood block

In the wood experiment, we start with 5562 data points. A 64x64x64 cube is used to store the external energy. The running time is around 1.5 minutes. There are around 212 control points used on the resultant surface. The result is in Figure 4.12 . The initial surface is a cylinder in this experiment, we also try the other algorithm for the initial surface, and it dose not make significant difference because the object itself is not too complex. The surface is subdivided once, which means there are two iterations. The two poles are on the left and right sides, respectively. The average external energy of the surface point is initially 3.0 voxels, 0.41 voxel after the first iteration,

based on the  $C^1$  B-spline surface, which is constructed directly from the control points obtained (in fact, these control points are for  $C^0$  B-spline surface). All surfaces shown here are  $C^1$  and closed, including the two poles.

The general information of the first five experiments is listed in Table 3: and Table 3: Because the cap of the surface is represented by a set of parameters rather than control points, the numbers of control points are approximated numbers..

**Table 4: Performance Summary (cont.)**

	size of the external energy cube	initial average error in voxels (cylinder initially)	final average error in voxels (cylinder initially)
tooth	150X150X150	4.42	0.6
wood	64X64X64	3.0	0.27
phone	150X150X150	5.62	0.39
head1-Carol	300X300X300	18.00	0.78
head2	300X300X300	20.15	1.18

**Table 3: Performance Summary**

	number of data points	number of control points	run time (cylinder initially)	run time (sphere initially)	number of subdivisions
tooth	11841	263	4 mins	3 mins	1
wood	5562	212	1.5 mins	1.0 mins	1
phone	15776	247	4.5 mins	N/A	1
head1-Carol	136082	2632	21.5 mins	14 mins	2
head2	45514	2632	31.5 mins	24 mins	2

In the tooth experiment, we start with 11841 data points. A 150x150x150 cube is used to store the external energy. The running time is around 4 minutes. There are around 263 control points used on the resultant surface. The result is in Figure 4.11 . The initial surface is a cylinder in this experiment, we also tried the spherical initial surface, and it does not make any significant difference, because the object itself is not very complex. The surface is subdivided once, which means there are two iterations. One of the poles is on the top in this experiment, and we can see the top of the tooth is smooth (the other pole is at the bottom). The average external energy of the surface point is initially 4.42 voxels, 1.0 voxel after the first iteration, and 0.6 voxel after the second iteration (one sub-division has been done).

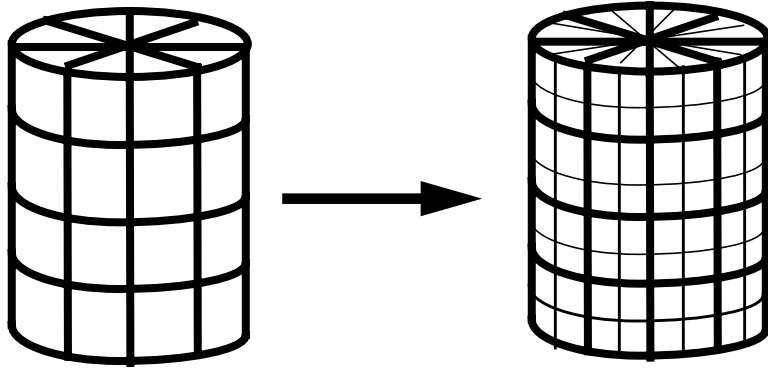


Figure 4.10 Subdivision. Notice that no row is added on the cap

For all  $G_m$ 's on all even(resp. odd)-indexed meridians, we call Powell to minimize  $E_{total} (= W_{int} * E_{int} + W_{ext} * E_{ext})$  of  $G_m$  by adjusting the corresponding  $3 * H_m$  variables.

#### Stage 4: Sub-division

At this step, every rectangular patch is divided into four, except those on the caps. We do not add a new row to the cap because these triangles around the pole are small already, so adding a new row to the cap might lead to degenerate triangles. This is illustrated in Figure 4.10 .

These four stages in the flowchart correspond to those in the previous paragraph. If the cap is already close to the real object, that is, it has low external energy, then we can skip this stage. The following two stages are to do the curve and mesh fittings. We have an inner loop for these two stages in the first two iterations. In this inner loop, we choose the odd-indexed and even-indexed meridians in turns in these two stages for the sake of fairness. The fourth stage is to do the subdivision. We switch from the long-distance energy field to the short one after the first iteration.

## 4.5 Experiments

We now show results from eight experiments on real data. Five of them are on the tooth, wood, phone, head1 (Carol), and head2, respectively. One shows how the fitting surface of the head1 experiment evolves, and the last two show the robustness of this algorithm by giving a very bad initial surface. Then we show one more experimental result on synthetic data, which shows the difference in the surface normal between the fitting surface and the underlying synthetic surface.

In all of these eight experiments, we start with the same crucial parameters to show the robustness and stability of this algorithm.  $ERROR_{threshold}$  is 1.0.  $RATIO_{ext-to-int}$  is 10.

These experiments are performed on a Sun Sparc-10 workstation. For each experiment, we show the original data and two shaded results. The shaded surface is

At the beginning of this stage, the even(resp. odd)-indexed meridians are the same as the corresponding meridians at the previous iteration (or, for the first iteration, the corresponding meridians of the initial sphere) except for those control points shared with the caps.

The responsibility of the selected meridians in this stage is to fit each strip to the data (target) by mesh fitting (We assume that the odd(resp. even)-indexed meridians have done a good job in stage 2 here).

For each strip, there are  $2*(M-1)$  patches and an even(resp. odd)-indexed meridian. Because those four patches shared with the caps are supposed to be good by default, we only need to consider  $2*(M-3)$  patches. Let  $S$  be one of those strips. Now we need to determine the goodness of those  $2*(M-3)$  patches in  $S$ . A patch is bad if its average external energy is greater than  $ERROR_{threshold}$ . We find the connected bad patches (these bad patches need to be processed simultaneously), and this way the bad patches in strip  $S$  can be classified into several groups.

Let  $G_m$  be one of those groups, and  $S_m$  be the intersection of  $G_m$  and the even(resp. odd)-indexed meridian in the strip.  $S_m$  is itself a small snake, and the internal energy of  $G_m$  is calculated through  $S_m$ . An example is in Figure 4.9 . In this example, let the 3 dotted rectangular patches be  $G_m$ , and the thick solid line segments composed of  $P_{21}$ ,  $P_{31}$ , and  $P_{41}$ , be  $S_m$ . The dotted and the solid line segments show the control points involved when calculating the internal energy (13 control points are involved), and the diagonal patches together with the dotted ones are those patches involved when we calculate the external energy of the even(resp. odd)-indexed meridian (there are 8 patches involved). In this example, the control points tuned are  $P_{21}$ ,  $P_{31}$ , and  $P_{41}$ , and thus there are 9 variables injected to Powell for this example.

The internal and external energies for the example in Figure 4.9 . are as below:

$$E_{int} = \left( \sum_{i=1}^5 \|P_{i+1,1} + P_{i-1,1} - 2P_{i,1}\|^2 + \sum_{i=2}^4 \|P_{i,0} + P_{i,2} - 2P_{i,1}\|^2 \right)$$

$$E_{ext} = \sum_{i=1}^4 \sum_{j=0}^1 A_{i,j} \cdot E_{i,j}$$

Where  $E_{i,j}$  is the average external energy of rectangular mesh  $P_{i,j}P_{i,j+1}P_{i+1,j+1}P_{i+1,j}$ , and  $A_{i,j}$  is the approximate area of the rectangular mesh.

The way to compute the internal energy is exactly the same as that for odd(resp. even)-indexed meridian in stage 2, and the way to estimate the area of a patch and the average external energy is explained in the previous section for the cap.

Suppose  $S_m$  contains  $H_m$  control points, and then there are  $3*H_m$  variables to be tuned by Powell.

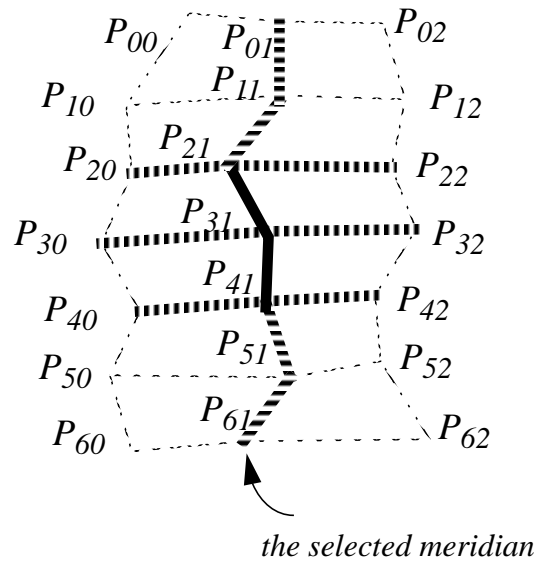


Figure 4.8 The odd(resp. even)-indexed meridian

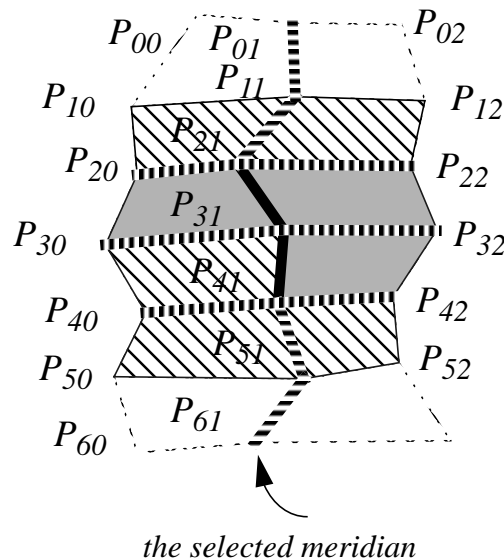


Figure 4.9 The even(resp. odd)-indexed meridian.

We may say that the responsibility of odd(resp. even)-indexed meridians is to find the frame or profile of the object.

### Stage 3. Mesh fitting

The surface now is separated by caps and odd(resp. even)-indexed meridians into  $K$  independent strips, and each strip can be tuned independently now. We select the rest even(resp. odd)-indexed meridians, and use them to fit all strips to the data (target). These even(resp. odd)-indexed meridians are fixed in stage 2 for the curve fitting. Each strip contains an even(resp. odd)-indexed meridian.

nary snake except that it is also influenced by the control points on the two neighboring Meridians.

The first two and last two control points of the odd(resp. even)-indexed meridian are shared with the caps, so these four points are supposed to be good and won't be tuned under any situation. Each meridian has  $M$  control points and  $(M-1)$  spans. Because the first and the last spans are fixed by the caps, we only need to handle at most  $(M-3)$  spans for each odd(resp. even)-indexed meridian when doing curve fitting.

A odd(resp. even)-indexed meridian is tuned adaptively, and we just deal with the bad spans of the meridian at each iteration. We check the external energy of each span. Those spans with average external energy greater than  $ERROR_{threshold}$  are bad. We find out the bad connected spans (bad connected spans have to be processed simultaneously). This way, we can classify those bad spans into several groups, and each group is independent of each others. Let  $G_s$  be one of the groups ( $G_s$  is itself a miniature snake in some sense). An example is shown in Figure 4.8 . In this example, let the solid thick curve, composed of  $P_{21}$ ,  $P_{31}$ , and  $P_{41}$ , be  $G_s$ . The dotted and the solid line segments show the control points involved when calculating the internal energy. The external and internal energies are as below. In this example, there are 3 control points  $P_{21}$ ,  $P_{31}$ , and  $P_{41}$  to adjust, which leads to 9 variables injected to Powell.

The average external energy can be obtained by sampling a certain number of

$$E_{int} = \left( \sum_{i=1}^5 \|P_{i+1,1} + P_{i-1,1} - 2P_{i,1}\|^2 + \sum_{i=2}^4 \|P_{i,0} + P_{i,2} - 2P_{i,1}\|^2 \right)$$

$$E_{ext} = \sum_{i=1}^4 \frac{\|P_{i,1} P_{i+1,1}\|}{L_i} \cdot E_i$$

here  $E_i$  is the average external energy of the line segment between  $P_{i,1}$  and  $P_{i+1,1}$ .

points, finding the external energy of each sampled point through array  $G$ , and then averaging those external energies. The length of each line segment is considered because the importance of each line segment should be in proportion to its length.

The weights of those two energies are determined in the same way as we do for the cap in the previous stage.

Suppose there are  $H_s$  control points in  $G_s$ , excluding those shared with the caps.

The coordinates of each control point are the three components  $X$ ,  $Y$ , and  $Z$ , so we have  $3 \cdot H_s$  variables.

For all  $G_s$ 's on all odd(resp. even)-indexed meridians, we call Powell to minimize

$$E_{total} (= W_{int} \cdot E_{int} + W_{ext} \cdot E_{ext})$$

of  $G_s$  by adjusting the corresponding  $3 \cdot H_s$  variables.

The external energy for the cap is:

$$E_{ext} = \sum_{i=0}^{N-1} (A_{t,i} \cdot E_{t,i} + A_{r,i} \cdot E_{r,i})$$

Where  $A_{t,i}$  is the area of triangle  $P_p P_{1,i} P_{1,i+1}$ ,

$E_{t,i}$  is the average external energy of triangle  $P_p P_{1,i} P_{1,i+1}$ ,

$A_{r,i}$  is the approximate area of rectangular mesh  $P_{1,i} P_{1,i+1} P_{2,i+1} P_{2,i}$ , and

$E_{r,i}$  is the average external energy of rectangular mesh  $P_{1,i} P_{1,i+1} P_{2,i+1} P_{2,i}$ .

The area factor is introduced in the above formula because the contribution of each patch should be in proportion to its size.

$E_{t,i}$  and  $E_{r,i}$  can be estimated by

- 1) sampling a certain number of points in the patch,
- 2) finding the energy of each point from array G defined in the previous section, and finally
- 3) averaging the external energy.

$A_{r,i}$  can be coarsely estimated by dividing the rectangular patch into four triangles through the central point, and then summing the area of every triangle.

Every time before Powell is called, the weights  $W_{int}$  and  $W_{ext}$  are determined by  $RATIO_{ext-to-int}$ , the current internal energy  $E_{int}$ , and external energy  $E_{ext}$  of the cap. Then, Powell is called to minimize  $E_{total} = (W_{int} * E_{int} + W_{ext} * E_{ext})$  of the cap by adjusting those (N+5) variables. There are two independent caps on the surface, so this step will be executed twice.

### Stage 2. Curve fitting

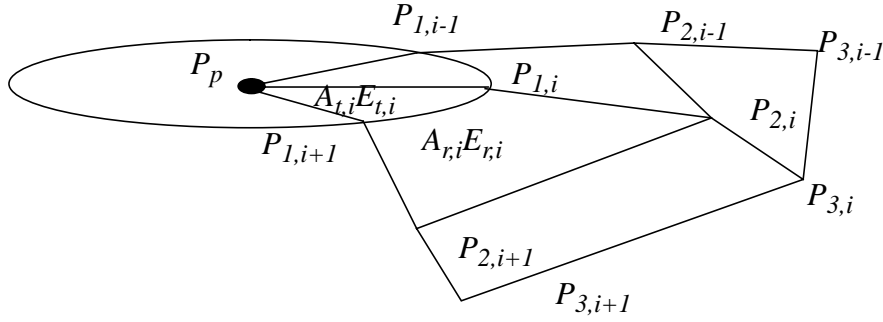
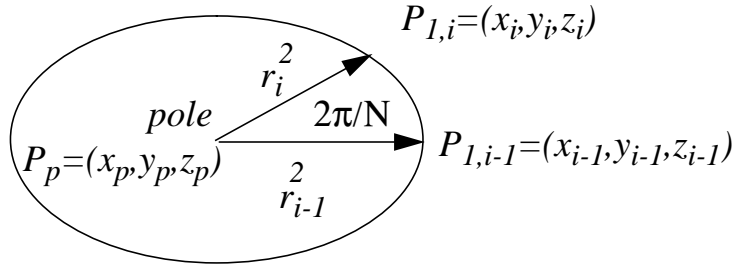
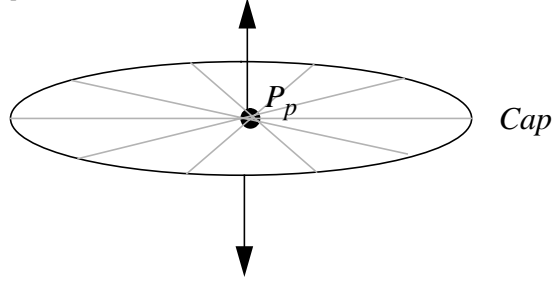
Suppose there are MxN control points on the surface, that is, there are M rows and N columns. Here, N is always even. Let  $N=2K$ . There are N Meridians on this surface.

Now we select the Meridians with odd (or even) index to do the curve fitting (and the rest are for the mesh fitting, explained later), so there are K such meridians.

At the beginning of this stage, these selected odd(resp. even)-indexed meridians are the same as the corresponding meridians at the previous iteration (or, for the first iteration, the corresponding meridians of the initial sphere) except for those control points shared with the caps.

We treat each odd(resp. even)-indexed meridian individually as if they are independent of one another. Each odd(resp. even)-indexed meridian is just like an ordi-

Normal vector determined by two parameters  $\varphi$  and  $\psi$



$P_{1,i}$  and  $P_{1,i-1}$  are two consecutive control points on the row immediately adjacent to the pole.

Figure 4.7 An example of the cap

closed surface (the caps of the initial surface should be planar surely). An example is depicted in Figure 4.7 .

The internal energy  $E_{int}$  for the cap is as below:

Let  $P_{1,i}$  and  $P_{2,i}$ ,  $0 \leq i < N$ , be the two closest rows to the pole ( $P_{1,i}$  is the row immediately adjacent to the pole.).

$$E_{int} = \left( \sum_{i=0}^{N-1} \left( \|P_p + P_{2,i} - 2P_{1,i}\|^2 + \|P_{1,i} + P_{3,i} - 2P_{2,i}\|^2 + \|P_{1,i-1} + P_{1,i} - 2P_{1,i+1}\|^2 \right) \right)$$

On the contrary, if we set  $W_{\text{int}}$  directly and keep it unchanged, then the internal energy tends to dominate at the later iterations because the external energy decreases faster than the internal energy does. This might not lead to a good fit.

$\text{ERROR}_{\text{threshold}}$  and  $\text{RATIO}_{\text{ext-to-int}}$  are 1.0 and 10 in all of our experiments.

#### 4.4 Details of our algorithm

Now, we elaborate on the four stages concerning the cap, curve and mesh fittings, and subdivision.

##### Stage 1. Cap fitting

By treating the pole as 2 or more control points, we can achieve  $C^0$  continuity at the poles on the caps when constructing the fitting surface. For example, for quadratic B-spline surface, we can achieve  $C^0$  simply by duplicating the control points at the two poles. The caps of the surface present problems when we try to upgrade the surface from  $C^0$  to  $C^1$ , because the poles will not be  $C^1$  and are singular. This is an inherent limitation of the rectangular mesh no matter what surface construction algorithms (for example, B-spline and Bezier) are employed. But if the control points are coplanar, then the tangent vectors at the poles along all directions will be coplanar. Based on this, we constrain all the points, including the pole and its adjacent control points to be coplanar. This way the surface constructed can be smooth everywhere.

The cap can be represented by the following formula, which contains  $(N+5)$  variables, where  $N$  is the number of the adjacent points of the pole. Initially,  $\phi$  and  $\psi$  are

$$(x_p, y_p, z_p) = \left( R_i^2 \cos \theta_p, R_i^2 \sin \theta_p, 0 \right) \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \times \begin{bmatrix} \cos \psi & \sin \psi & 0 \\ -\sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix} + (x_p, y_p, z_p), \quad 0 \leq i < N.$$

where  $(x_p, y_p, z_p)$  is the coordinate of the pole,

$N$  is the number of points adjacent to the pole,

$(x_i, y_i, z_i)$  are the coordinates of the  $i$ th point adjacent to the pole,

$q_i = 2\pi * i/N$ ,

$R_i^2$  is the distance from  $(x_p, y_p, z_p)$  to  $(x_i, y_i, z_i)$ ,

$\phi$  is the angle of the rotation around the x axis, and

$\psi$  is the angle of the rotation around the z axis.

zero. The distance from the pole to an adjacent point should be always positive, so we use the square root of the real distance as the variable to guarantee that the distance is positive. The variables in this formula are  $x_p, y_p, z_p, \phi, \psi$ , and  $R_i, 0 \leq i < N$ , so the number of variables is  $N+5$ . Initially, these variables can be set in accordance to the initial

The computational time could also be largely reduced by parallel processing. It is obvious that (1) the two caps are independent of each other, (2) all odd (resp. even)-indexed meridians are independent of one another, and (3) all even (resp. odd)-indexed meridians are independent of one another, so each stage can be performed in parallel.

There are two important parameters,  $ERROR_{\text{threshold}}$  and  $RATIO_{\text{ext-to-int}}$ , set by the user in this algorithm.

$ERROR_{\text{threshold}}$  is used to determine whether or not a patch of the surface or a span of the snake is good. We only process the bad parts of the snakes and the bad patches on the fitting surface during each iteration. At each iteration, a patch (or a span) is good if its average external energy is smaller than  $ERROR_{\text{threshold}}$ ; otherwise, it is bad.

$RATIO_{\text{ext-to-int}}$  specifies the relative importance of the external energy with respect to the internal energy. After setting  $RATIO_{\text{ext-to-int}}$ , two internal parameters  $W_{\text{ext}}$  and  $W_{\text{int}}$ , concerning the weights of the external and internal energies, are set by the system.  $W_{\text{ext}}$  is always 1, and  $W_{\text{int}}$  is set as below:

$$W_{\text{int}} = \frac{E_{\text{current-ext}}}{E_{\text{current-int}} \times RATIO_{\text{ext-to-int}}}$$

where  $E_{\text{current-ext}}$  is the current external energy of the fitting surface or snake, and  $E_{\text{current-int}}$  is the current internal energy of the fitting surface or snake.

Every time Powell is invoked,  $W_{\text{int}}$  is re-calculated based on  $RATIO_{\text{ext-to-int}}$  and the current internal and external energies of the fitting surface. So every time, Powell may be called with different  $W_{\text{int}}$ .

The reasons why we set  $RATIO_{\text{ext-to-int}}$  is that  $W_{\text{ext}}$  and  $W_{\text{int}}$  are different measures and thus on different scales.  $RATIO_{\text{ext-to-int}}$  serves to normalize two energies.  $RATIO_{\text{ext-to-int}}$  is always greater than 1; otherwise, the fitting surface is unlikely to conform to the data points, as we now explain.

As we subdivide the surface after each iteration, the fitting surface is approaching the real object. We have more confidence in the fitting surface. So it is suggested that the internal energy weight be reduced as the process goes on. One more advantage of  $RATIO_{\text{ext-to-int}}$  is that the weight of the internal energy tends to decrease as the process goes on, because  $E_{\text{ext}}$  decreases faster than  $E_{\text{int}}$  does when  $RATIO_{\text{ext-to-int}}$  is greater than 1. By setting  $RATIO_{\text{ext-to-int}}$  to be a constant greater than 1, we can diminish the importance of the internal energy after each iteration implicitly, and thus obtain a better fitting surface.

profiles of the target. These selected meridians are not influenced much by the fitting surface (by the internal energy) when moving, so they can detect the object more accurately. Please notice that the external energy of these selected meridians is defined on the curve without considering the surface nearby. The surface is separated by the selected meridians into independent “strips” in a way, and each strip contains an even (resp. odd)-indexed meridian. Each strip is bounded by two odd (resp. even)-indexed meridians.

Next, we fit each strip to the target. We select meridians of the other polarity, even (resp. odd)-indexed, to do the mesh fitting for each strip. We treat them as regular snakes, except that they are tuned to minimize the external energy (error) of the strip. This means the external energy is not only from the curve but also from the area it defines.

The fourth stage is subdivision. If the fitting surface up to now is not satisfactory, we subdivide all rectangular patches into four and then go to stage 1; otherwise, exit.

### 4.3.3 Summary and discussion

In summary, in the first step, we fit the caps. Next, the odd (resp. even)-indexed meridians are used to find the profile or frame of the target, and the surface is divided into strips by these meridians. Then the even (resp. odd)-indexed meridians are applied to fit the strips to the data. Finally in stage four, we subdivide the surface, that is, we divide each rectangular patch into four. We break the 3D surface problem into a set of 2D (linear) B-snake problems in a way. Also notice that, at each step, although we have different ways of calculating the internal energy  $E_{int}$  and the external energy  $E_{ext}$  the basic idea is the same.

We can see that this is a typical coarse-to-fine approach. We start with few control points and large patches, then we increase the number of the patches and control points in later iterations.

Our algorithm overcomes the time and space complexities, and closed surface problems by (1) breaking the 3D surface problems into several 2D snake problems, which is shown in stages 2 and 3, (2) coarse-to-fine approach, and (3) forcing the cap to be planar, which is explained to stage 1. Due to the robustness of Powell, we do not need a good initial guess, and two examples are shown in Figure 4.17 and Figure 4.18. Also, the Powell method guarantees convergence.

Please notice that stages 2 and 3 can be performed very fast when there are few control points. We can take advantage of this to get more reliable global information. We repeat these 2 stages until there is no further improvement in the first 2 iterations (in the first 2 iterations, there are not many control points in the meridians or caps). It is almost impossible to rectify the error from the first 2 iterations, which is considered global, by later iterations. So, in our implementation we add an inner loop to these two stages for the first two iterations.

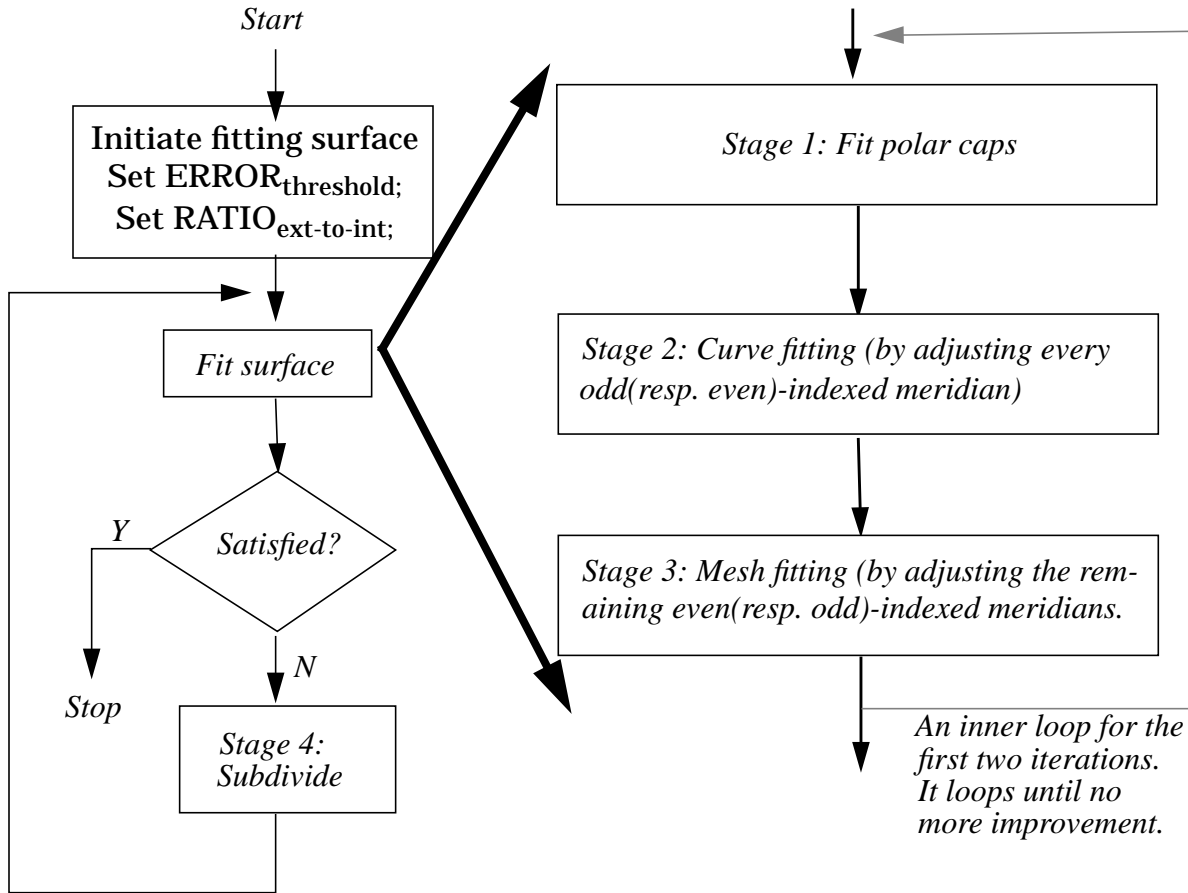


Figure 4.6 The flowchart of our algorithm

vide all patches in four, and repeat the process until some terminating condition is met.

Our algorithm is a 4-stage one, and during the first three stages, Powell is called frequently for energy minimization.

First, we fit the caps to the data, and force the caps to be planar. This way, all tangent vectors in all directions at the pole are coplanar. When fitting, the cap can change its shape, subject to the planar constraint, in order to get the best fit. Thanks to this planar constraint, the constructed surface is smooth even at the poles.

Second, we perform the curve fitting to some meridians to locate the profile of the target, and this is done by applying energy minimization to these meridians. We select the odd (resp. even)-indexed meridians and fit them to the data by treating them as approximating linear B-snake[44]. The only difference between a B-snake and a selected meridian lies in the internal energy. When calculating the internal energy of these meridians, we not only consider their own smoothness but also the smoothness between them and their immediate neighboring even (resp. odd)-indexed meridians (an example is depicted in Figure 4.8 ). Then we let them adapt to find the

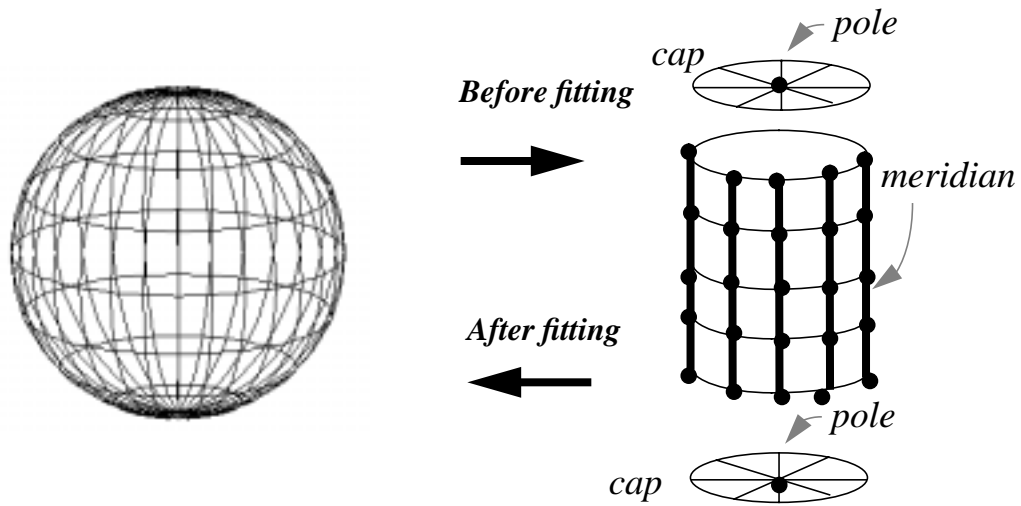


Figure 4.5 The initial closed surface and the definitions of Pole, Meridian, and Cap

- 4) The convergence of the surface to the data points: The convergence of the fitting process should be guaranteed.

Now, we define some terms for further use. We define a Cap to be the triangular patches formed by a Pole and its adjacent control points. So, we always have two Caps. A Meridian (a line of constant  $u$  in parameter space) is defined to be the line connecting the two Poles, as depicted in Figure 4.5 .

### 4.3.2 The algorithm

A flowchart of our algorithm is in Figure 4.6 .

Both the cylindrical and spherical initial surfaces we adopt are topologically equivalent to a sphere. In our algorithm, we consider a sphere as composed of three parts, which are two caps and an open cylinder as shown in Figure 4.5 . These three parts are processed separately in our algorithm.

Instead of injecting all  $M \times N$  control points into the minimization procedure (which is possible but extremely expensive), we decompose the problem into a curve fitting problem followed by a (simpler) mesh fitting problem.

Given that the caps are already in place, we select every other meridian and move their  $(M-4)$  control points, which are not shared with the two caps, to minimize their energy. We then select the remaining meridians and move their  $(M-4)$  control points, which are not shared with the two caps, to minimize the energy of the related patches this time. It is important to note, however, that only the bad segments (patches or curves) are injected into the minimization procedure (after the energy minimization, we can guarantee that the associated energy of the bad segments is reduced, but some segments with higher external energy might still be bad). Then, we subdi-

### 4.2.7 Choice of the initial surface

Now we move on to how the initial surface is set up. Currently, we have two methods for setting up the initial surface.

The first is to build a cylinder that covers all the data points as the initial surface. The second is based on a spherical coordinate representation. We compute the center of mass of the data first as the center of the initial sphere, then sample data points in  $N_\theta \times N_\phi$  directions, and find out the farthest data point in each direction. The radius in each direction is the distance between the center of mass and the corresponding farthest data point. We use this deformed sphere as an initial guess. If we do not find a data point in a given direction, we use the average of the radii in the neighboring sampled directions as the one in this direction. The caps of these two initial surfaces are constrained to be planar.

We always start with the second approach, which can give a better approximation of the data. If we are unable to compute the radius in many (more than 18 for  $12 \times 17$  sampled directions) directions, we abort this choice and revert to the cylindrical initial guess.

## 4.3 Overview of our algorithm

### 4.3.1 Issues

Several problems come with the algorithms dealing with deformable model:

- 1) Huge computational time and space: Assuming  $M \times N$  control points on the fitting surface, there are  $3MN$  variables for this surface. Theoretically, we can just inject these  $3MN$  variables into a minimization algorithm to minimize the energy of the fitting surface. This approach turns out to be impractical due to the unbearable computational time when  $3MN$  is large. Furthermore, most minimization algorithms need a matrix of size  $(3MN) \times (3MN)$ , which results in huge space complexity also. An adaptive approach can just alleviate these problems, but cannot avoid these problems in the worst case, especially when all control points or patches are bad.
- 2) Constructing a smooth closed surface from the B-spline control points: This is no problem if the surface is constructed from the triangular mesh, but we choose rectangular mesh, and we have already given the reasons in the previous section. The problem is that the poles are not smooth if we try to construct the closed smooth surface, which is topologically equivalent to a sphere, directly from the control points. It is because these two poles are shared by many degenerate patches (triangular patches) around them.
- 3) A good approximation of the data as the initial surface: The quality of the fitting result depends on the initial surface, and we would like the result to be invariant to the initial surface as long as the initial surface is not too bad.

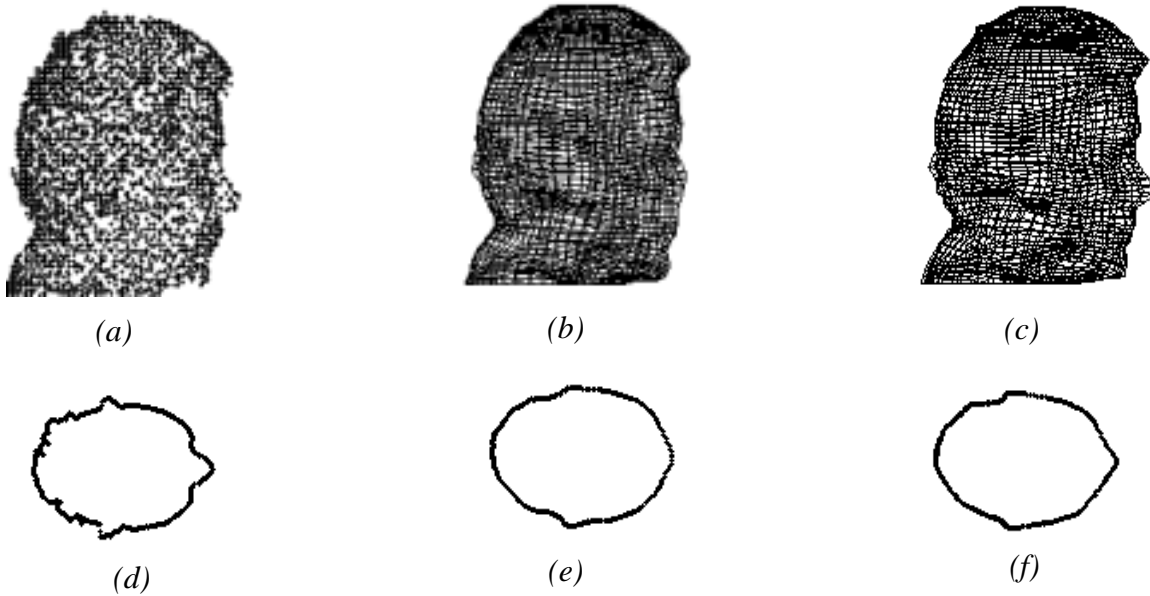


Figure 4.4 An example of the effect of the short-distance external energy

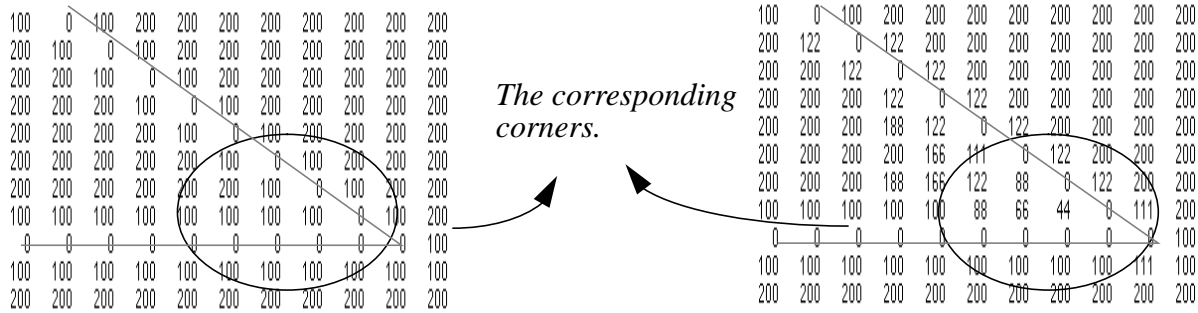
#### 4.2.6 Internal energy $E_{int}$

The internal energy should be computed as a measure of the curvatures on the surface. Here instead, we measure it as the sum of the digital curvatures on some curves drawn on the surface. The choice of these curves will be explained later.

For a B-spline surface, parameterized by  $u$  and  $v$ , with  $M \times N$  control points, we can construct  $M$  (resp.  $N$ ) snakes, each of which contains  $N$  (resp.  $M$ ) control points, directly from the control points along the  $u$  (resp.  $v$ ) direction. We use the second derivative of these snake to represent the internal energy. Due to the high similarity between the shapes of the control points and the fitting surface, we can just directly use the control points to estimate the internal energy. The way we estimate the internal energy brings down the computational time since the internal energy is calculated very frequently. Suppose we have a curve composed of points  $P_i$ ,  $0 \leq i < N$ , the internal energy is defined as below:

$$E_{int} = \sum_{i=1}^{N-1} \|P_{i-1} + P_{i+1} - 2P_i\|^2$$

The reason why we define the internal energy in such a simple way, which has no arc length as the denominator, is that the length of the snake (curve) cannot change much when the fitting surface is close to the fitted object. We always place much more weight on the external energy than internal energy, so we don't expect the snake length to be influenced much by the internal energy. This formulation favors coplanar and equidistant point arrangements.



(a) the long-distance external energy.      (b) the short-distance external energy.

Figure 4.3 An example of the long-distance external energy field and its corresponding short-distance external energy field

Every voxel is initialized to a very large number, except the data voxels which are zero. We then put a one in any voxel for which any of its neighbors contains a 0, and so on.

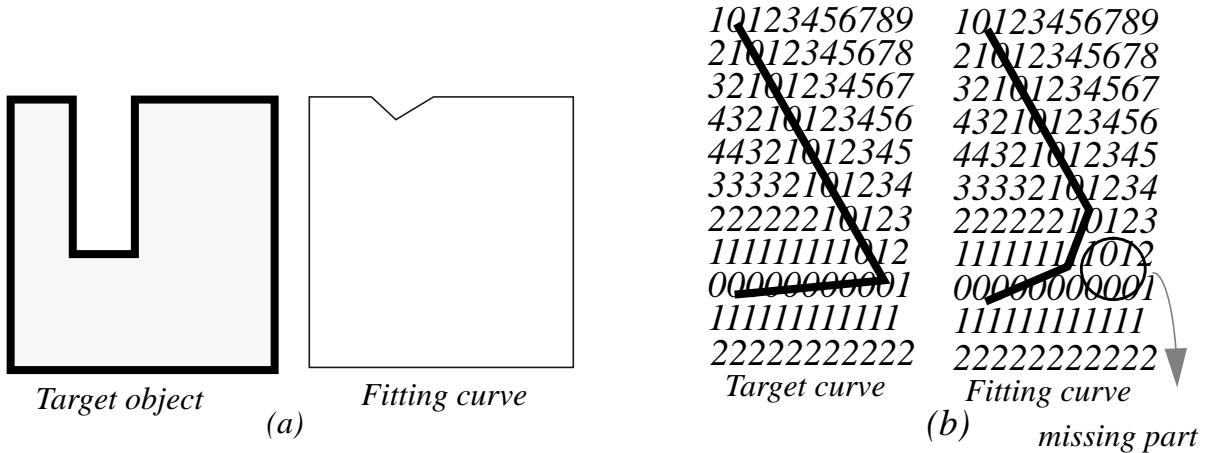
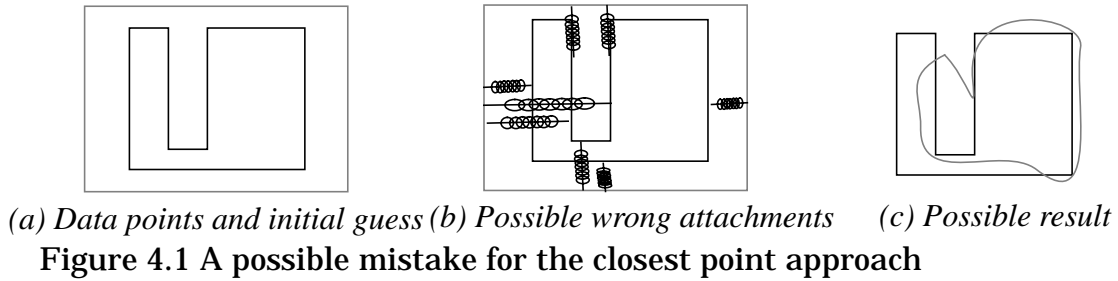
Short-distance external energy field:

Once the surface is close enough (the distance is smaller than  $H$ , typically 3), we redefine the energy field by averaging the values of the original field in an  $H \times H \times H$  neighborhood. Of course, we leave the 0 values (data) unchanged. This is equivalent to interpolating the original values. Figure 4.3 shows an example illustrating the difference between the long-distance and the short-distance external energies. It should be clear to the reader that the corner in the short-distance energy field is more salient than in the long-distance energy field.

Figure 4.4 shows an example of the effect of the short-distance external energy field. In this example, (a) is the initial data, which is a head, (b) is the fitting result applying the long-distance external energy field only, and (c) is the fitting result applying both the long-distance and the short-distance external energy field. (d), (e), and (f) are cross-sections of (a), (b), and (c), respectively. We can clearly see that we obtain a better fit if the short-distance external energy is applied. The use of short-distance external energy can also bring down the computational time to some degree.

We could combine these two external energies into one. The problem is that we use one byte to store the external energy of each voxel in our implementation, and the span of the energy value of the voxel for short-distance external energy field is much smaller, so we can get better quantization (resolution) for the short-distance external energy field, and thus obtain a better result. This is why we have two separate external energy fields here. We switch to the short-distance external energy field after the first iteration in our implementation.

In summary, the long-distance external energy field is coarse and brings the surface close to the data, and the refining work is left to the short-term external energy field.



ergy at a low cost, and the space complexity, which is in proportion to the number of voxels in the cube, is reasonable.

The problems with this definition of distance, as the ones commonly encountered with morphology operations, is that it rounds corners and cannot handle cavities. These two problems are illustrated in Figure 4.2 below. In (a), the resultant curve only gives a dimple instead of going deep into the cavity. (b) shows the distance field and the target curve, and the resultant fit, which rounds the corner. The numbers in this example indicate the external energy at each point, and the pixel with energy zero is the data point. Any algorithm based on the shortest distance to the data points cannot be immune to these problems. To palliate this, we define two external energy fields, a long-distance one and a short-distance one. The long-distance external energy field is rather coarse, and measures the distance from each voxel to the nearest data point. Its main purpose is to quickly pull the fitting surface towards the data. The short-distance one is a more accurate measure.

At the beginning, we use the long-distance external energy, and when the fitting surface is close to the fitted object as measured by the long-distance external energy, we switch to the short-distance external energy to improve the results.

#### Long-distance external energy field:

The long-distance field is computed by a 3D Blum Medial Axis Transform [32] in a digitized cube  $G(X, Y, Z)$  typically  $150 \times 150 \times 150$ . This algorithm is straightforward:

es, good and bad, based on their associated external energy (error fit). Only the control points belonging to the bad patches are adjusted to minimize the energy (error) of the fitting surface, and then all patches are subdivided into four sub-patches after the fitting process. In our current implementation, the user specifies the number of the iterations. So, the process terminates after we have iterated a certain number of times. The numbers of the control points and patches increase as the iteration goes on. Presently, we iterate at most three times, which means that we only go through at most two subdivisions. The coarse-to-fine approach helps improve the performance in time very much. The global information can be acquired in the first 2 iterations, and the following iteration is used to highlight the surface details.

#### 4.2.5 External energy $E_{\text{ext}}$

The external energy is a potential energy which attracts the fitting surface toward the data points. The result of the energy minimization highly depends on the external energy field. In our implementation, we define the external energy of a point on the fitting surface to be the distance to the nearest data point. The fitting surface can approach the data points when the external energy of the fitting surface is being minimized.

The external energy, which derives from the data points, can decide the success of the fitting process. A good definition of the external energy should have reasonable time and space complexities when the external energy of the fitting surface is being calculated. One possible solution to this is to define the external energy also from the data point's viewpoint. In addition to the energy field, we might define one more external energy based on the distance between each data point and the fitting curve. This might bring about another serious problem. How do we determine the corresponding point on the fitting curve for each data point? Usually, this information is not available. We might define the corresponding point to be the closest point on the fitting curve. But in reality the nearest points are not necessarily the corresponding points, and it is possible for one point on the fitting curve to correspond to more than one data point as shown in Figure 4.1 when the target is concave. In (a), the concave polygon, composed of solid lines, is the data points, and the dotted rectangle is the initial fitting curve. (b) shows possible wrong attachments based on the closest point criterion. Data points belonging to different parts of the target might be attached to the same part of the fitting curve, and, in consequence, the fitting curve could not capture the profile of the target faithfully as shown in (c). One more problem with this approach is that it is quite expensive, as we need to calculate the closest point for each data point every time the external energy is evaluated.

Our proposed method is efficient in the time and space complexities. We use an energy field represented by a cube composed of voxels. Each voxel contains the distance to nearest data point, and this distance is the energy of the voxel. The external energy of the fitting surface is calculated by sampling points on the surface and getting the energies of the voxels they fall in. This way, we can compute the external en-

### 4.2.3 The Powell minimization procedure

What we are performing now is simply the minimization of energy  $E_{total}$  by adjusting the points on the mesh. Gradient descent is not very reliable. For our application, an eligible numerical method should meet the following requirements:

- 1) It should be able to handle discontinuous functions,
- 2) it should work when the derivative information is unavailable,
- 3) it should be reliable and accurate,
- 4) the time complexity should be reasonable, and
- 5) convergence should be guaranteed.

Powell is a good candidate, though might be slower than gradient descent. But, in our application, it is not really the case because (1) we are using a coarse-to-fine approach and only the bad patches are handled at each iterations and (2) the weight  $W_{int}$  of smoothness constraint is changing implicitly as the process goes on. So, the gradient descent procedure, if applied, has to inverse a completely different matrix every time it is invoked even if the number of variable is the same (due to different  $W_{int}$ ). It is impractical to pre-compute all inverse matrices ahead of time because there are infinite number of possible matrices. In contrast, Powell does not have to invert a matrix. Inverting a huge matrix, which might happen fairly often in our application, is time consuming. So, here the whole process should not be slowed down much by Powell.

The Powell algorithm is itself a direction set method for function minimization. Assume the function to be minimized has  $N$  variables. With an initial guess, which is an  $N$ -tuple vector, Powell can work by producing spontaneously mutually conjugate (non-interfering) directions, and searching along these directions sequentially for the (local) minimum. Because of the power of Powell, we can define almost any kind of energy we need. If we have any *a priori* about our target, we can define it in terms of the energy and apply it to Powell. For instance, if the approximate area  $AREA$  of the object is known, we can define an energy,  $E_{area} = |AREA - AREA_{fitting\ surface}|$ , using this information. According to our experiments, Powell is efficient, compared to the regular gradient descent approach, on the aspect of the number of control points needed to fit the data points. For more information, we refer the reader to [49,33,31,40,48,54].

If too many control points need to be handled simultaneously, Powell can become very slow. In our algorithm, this issue is addressed by a partitioning scheme, explained later, and thus Powell is always called with a limited number of variables. and we can get reasonable results in our experiments in a few minutes for simple objects.

### 4.2.4 Coarse-to-fine approach

Our approach is iterative, and we start this fitting process with a small number of control points (patches). At each iteration, we categorize the patches into two class-

property of B-spline function, we can also use these  $C^0$  control points to construct a  $C^1$  or  $C^2$  B-spline surface without significant error when there is a large number of control points and these control points are close to one another.

#### 4.2.1 Global parameters

In our system there are only two global parameters set at the beginning, which makes the whole system easy to control.

The first is  $ERROR_{\text{threshold}}$ , which is used to determine the goodness of the patch on the mesh. The second is  $RATIO_{\text{ext-to-int}}$ , which is used to set the initial respective contributions of the internal and external energies. These two global parameters will be explained in details in the following section. It is worth noting that all our data sets were processed with the same parameter values.

#### 4.2.2 Surface Representation

First, how do we represent a surface? The most common primitives are triangular and rectangular meshes. The triangular mesh is more general, but suffer from the following drawbacks:

a) Eventually, we would like to construct a smooth surface, but it takes high degree polynomials to construct a  $C^1$  or  $C^2$  surface from the triangular mesh, which is expensive. In most cases, the algorithms for this purpose require the gradient information of each point, which does not come with the triangulation, and needs to be estimated. The rectangular mesh can be upgraded it to  $C^1$  or  $C^2$  easily through B-spline or Bezier functions.

b) For energy minimization, we need a method to estimate the smoothness of the surface. It is not easy to estimate the smoothness of the triangular mesh, compared to the rectangular mesh, whose derivative information can be evaluated with simple mathematics. In a way, each patch on the triangular mesh is parameterized by different parameters, which makes the estimation of the smoothness difficult.

In our algorithm, we need to sub-divide some patches on the mesh after each iteration. Sub-dividing the patches might lead to degenerate patches, which are points or lines. This might cause serious numerical problems later. We avoid this by always dividing the patch into four sub-patches at the center point.

The rectangular mesh, of course, presents problems of their own: it is harder to construct a closed smooth surface from the rectangular mesh because of the poles. But we can get around this problem, and then upgrade the closed rectangular mesh to  $C^1$  closed surface.

Here, we currently use a Linear B-spline surface for its efficiency in computation time and some geometric properties we need. For a Linear B-spline surface, each control point only affects its four neighboring patches. This makes it very easy to separate the whole surface into independent strips.

tion, are employed to adapt the superquadrics to the data points. Han *et al.*[37] introduce hyperquadrics, which is a generalization of superquadrics, for shape recovery from range data. Their model can represent any arbitrary convex shapes. Huang and Goldof [39] develop an algorithm similar to Vasilescu and Terzopoulos's. Their algorithm differs mainly in the way the meshes are subdivided. Instead of adjusting the stiffness of the spring to fit the local properties of the data, they update the patch size adaptively by adding or deleting nodes appropriately. A new node is added between two neighboring nodes if they are far from each other, and two neighboring nodes are merged if they are very close to each other.

To summarize, most of the algorithms described above either require many parameters, or cannot guarantee convergence (partially due to the use of gradient descent to minimize the energy). Our proposed algorithm can deal with these problems appropriately. In our approach, we apply a tested numerical method which guarantees convergence, and through an coarse-to-fine approach and a partitioning scheme, the computational time is kept in check.

## 4.2 Description of our approach

Most of the algorithms described earlier are suffering from long computation time and large space complexity. Furthermore, sometimes due to the instability of the numerical methods, such as gradient descent, the result might be bad because of over-shooting. It is not easy to detect over-shooting, let alone to backtrack and tune the stepsize. Most algorithms attacking this problem are based on gradient descent. Here we adopt Powell, a much more accurate and stable method. Through some mechanisms in our algorithm, the computation time is kept in check. The formalism we are about to establish amounts to deforming the initial surface to conform as closely as possible to the given 3D data points. This is achieved by defining an attraction force field around these data points to bring the initial surface closer to them. The initial surface is updated by a function minimization algorithm, Powell. Currently, the surface consists of  $C^0$  rectangular mesh. In a word, we treat the whole process as minimization problem – given an initial guess, which may be a cylinder, we find the local minimum of the energy function by tuning the variables (the positions of the control points).

The total energy of the fitting surface is defined as below:

$$E_{total} = W_{int} * E_{int} + W_{ext} * E_{ext}$$

$E_{ext}$  expresses the distance between the fitting surface and the data points.  $E_{int}$  depends on the constraints, such as smoothness. The definition of  $E_{int}$  is subject to change when different constraints are applied.  $W_{int}$  and  $W_{ext}$  are the corresponding weights. Without loss of generality,  $W_{ext}$  is always 1 in our system.

Once a  $C^0$  surface is obtained, it is also possible to upgrade it to  $C^1$  using existing algorithms [50], although we do not address this point here. Due to the convex-hull

data. A deformable model, which can give an analytical surface representation over a cloud of 3D data points, is a good candidate for this purpose.

The idea of fitting data by a deformable model can be found in the work of Kass *et al.*[41] in 2D. Such models are generalized in 3D by the same authors [57] for a surface of revolution.

Recently, Cohen *et al.*[34] have expressed the surface fitting problem as a functional minimization problem. They enhance performance and numerical stability using a variational approach and the Finite Element Method. Terzopoulos and Metaxas[55] present a physically based approach to fitting complex 3-D shapes using a class of dynamic models which can deform locally and globally, and satisfy the conflicting requirements of shape reconstruction and shape recognition. Based on the elastic properties of real materials, Pentland and Sclaroff [47] propose a closed-form, physically based solution for recovering a 3-D solid model from collections of 3D surface measurements. A closed-form solution can be obtained in their system using Modal Dynamics. Nastar and Ayache's approach [46] is similar to Pentland's, but they delete a nonlinear term in the physics governing equation, and then the computation is simplified. This way they can speed up the fitting process. Terzopoulos and Vasilescu [56], motivated by concepts from numerical grid generation, use adaptive meshes that could sample and reconstruct intensity and range data. In their recent work [58], they develop some algorithms to handle the discontinuity problem, and by subdividing the adaptive meshes, reasonable results can be obtained. Delingette *et al.*[35] model an object as a closed surface that is deformed subject to attractive fields generated by input data points and features. Features affect the global shape of the surface while data points control local shape. Sinha and Schunck [51,52] use weighted bicubic splines, which are able to interpolate data with discontinuity without much distortion, as a surface descriptor. A regularized least square fit with the addition of an adaptive mechanism in the smoothness functional is applied in order to make the solution well behaved. Minimizing the energy by handling the fitting B-spline surface independently along parameters  $u$  and  $v$  and interpolating the external energy field, Gueziec [36] obtains good fitting results efficiently in terms of time and space complexities. McInerney and Terzopoulos [43] apply a dynamic balloon model and finite element method to reconstruct a 3D object. Their model can give the information to measure the differential geometric properties of the fitted surface. Muraki [45] uses a "Blobby" model for the shape description. In his approach, a potential energy field is constructed through the "primitive," realized as an implicit function  $F(P)=T$ . By splitting one selected bad primitive into two at a time, and fitting them to the data, the shape of the object can be retrieved. Hoppe *et al.*[38] propose an algorithm based on the estimated tangent plane of each sampled data point, and then a Riemannian Graph is constructed using EMST (Euclidean Minimum Spanning Tree). The contour of the object can be derived from this graph. Solina's approach [53] is based on superquadrics, and several functions concerning bending, tapering, and cavity deforma-

# 4 Surface Approximation of a Cloud of 3D Points

**Chia-Wei Liao and Gérard Medioni**

We present an implementation of deformable models to approximate a 3-D surface given by a cloud of 3D points. It is an extension of our previous work on “B-snakes” [44] and [42], which approximates curves and surfaces using B-splines. The user (or the system itself) provides an initial simple surface, such as a closed cylinder, which is subject to internal forces (describing implicit continuity properties such as smoothness) and external forces which attract it toward the data points. The problem is cast in terms of energy minimization. We solve this non-convex optimization problem by using the well known Powell algorithm which guarantees convergence and does not require gradient information. The variables are the positions of the control points. The number of control points processed by Powell at one time is controlled. This methodology leads to a reasonable complexity, robustness, and good numerical stability. We keep the time and space complexities in check through a coarse to fine approach and a partitioning scheme. We handle closed surfaces by decomposing an object into two caps and an open cylinder, smoothly connected. The process is controlled by two parameters only, which are constant for all our experiments. We show results on real range images to illustrate the applicability of our approach. The advantages of this approach are that it provides a compact representation of the approximated data, and lends itself to applications such as non-rigid motion tracking and object recognition. Currently, our algorithm gives only a  $C^0$  continuous analytical description of the data, but because the output of our algorithm is in rectangular mesh format, a  $C^1$  or  $C^2$  surface can be constructed easily by existing algorithms.

## 4.1 Introduction

Range sensing is a mature technology, and there are many methods, such as time of flight and MRI, collect 3D data based on this technology. In addition to this, 3D data can also be obtained in passive ways like stereo and shape from X methods. The data obtained from the above sources is in the form of points.

But, in computer vision, what we need are some properties such as the curvature, normal, and principal directions. These quantities relate to the underlying surface, which is not made explicit in the original data. Furthermore, it is even more difficult if some ordering relation among the data points is not known. This happens mainly when we gather data points from various sources. Analytical surface construction of a cloud of points (boundary points of the object) becomes important because it is much easier to extract the features from an analytical surfaces. So, we need some tools to construct an analytical description (for example, surface) for the collected 3D