2 Description of Complex Objects from Multiple Range Images Using an Inflating Balloon Model

Yang Chen and Gérard Medioni

We address the problem of constructing a complete surface model of an object using a set of registered range images. The construction of the surface description is carried out on the set of registered range images. Our approach is based on a dynamic balloon model represented by a triangulated mesh. The vertices in the mesh are linked to their neighboring vertices through springs to simulate the surface tension, and to keep the shell smooth. Unlike other dynamic models proposed by previous researchers, our balloon model is driven only by an applied inflation force towards the object surface from inside of the object, until the mesh elements reach the object surface. The system includes an adaptive local triangle mesh subdivision scheme that results in an evenly distributed mesh. Since our approach is not based on global minimization, it can handle complex, non-star-shaped objects without relying on a carefully selected initial state or encountering local minimum problem. It also allows us to adapt the mesh surface to changes in local surface shapes and to handle holes present in the input data through adjusting certain system parameters adaptively. We present results on simple as well as complex, non-star-shaped objects from real range images.

2.1 Introduction

The task of surface description using 3-D input can be described as finding a fit of a chosen representation (model surface) to the input data. This process can be formalized in a number of ways involving the minimization of a system functional that explicitly or implicitly represents the fit of the model to the input data. Another very important aspect of such a system is to construct a *mapping* or *correspondence* between the surface of an object and the structure of the model. This mapping exists because the surface of the model and the surface of the object are topologically equivalent, considering genus zero type of objects. Therefore there exists a one-to-one mapping between the model structures and the object surface elements. Previous researchers have studied such mappings in a variety of ways using different representation schemes and model fitting methods. Examples of these approaches include the dynamic system using energy minimization in [4] and the dynamic mesh in [12] and [13]. The drawbacks of these approaches is that they must rely on an initial guess of the model structure which is relatively close to the shape of the object. The reason is that, in the absence of mapping or correspondence information, some other approxi-

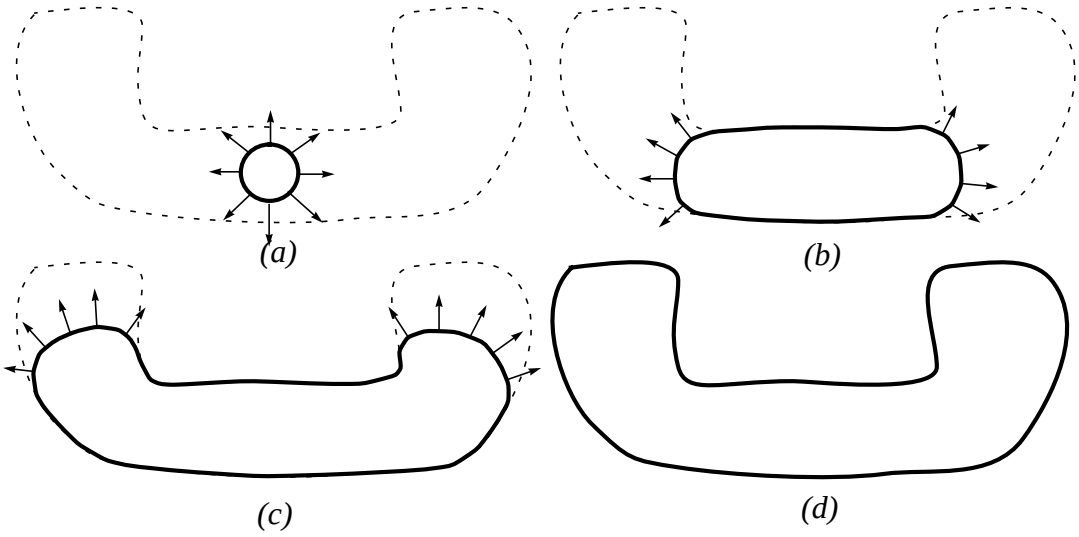


Figure 2.1 The inflating balloon model as illustrated in a 2-D case: (a) the initial state, (b) and (c) intermediate states, (d) the final state.

mations have to be used, such as the nearest data point to the model [13] in order for the system to converge to the desired results under the attraction force between the correspondence points in the model and the data. Such an approximation would have problems in cases where there are data points that are not the closest to their true corresponding points of the model, and thus inevitably lead the system towards a sub-optimal situation (local minimum). This is why those approaches can only deal with star-shaped objects.

In this paper we present a new approach for surface description using a dynamic balloon model represented by a triangular mesh. We start with a small triangulated shell placed inside the object and apply a uniform inflation force on all vertices in the direction normal to the shell's surface. The vertices are also linked to their neighboring vertices through springs to simulate the surface tension and to keep the shell smooth. The applied inflation force moves the vertices towards the object surface until they "land" on it. This process is similar to that of blowing up a balloon placed inside the hollow object until it fits the shell of the object. Thus the goal of mapping the model to the object surface is achieved through the physics of a growing balloon in a very natural way (see Figure 2.1). Of course, we also need to handle noisy data, and holes (lack of data), as described later.

Our system is not based on global minimization methods, and it can make decisions based on local information about the shape of the object surface. During the process of the growth of the triangular mesh, the triangles will be subdivided dynamically to reduce spring tension and to allow the mesh surface area to increase in order to cover the larger object surface. As the mesh expands and the vertices start to reach the object surface, the entire mesh surface is gradually subdivided into pieces

of connected triangular regions, which allows us to treat the surfaces in a local context by tailoring the parameters, and possibly strategy, of the system in dealing with each region separately based on local information.

Another aspect of our approach is its role in data integration. Most of the previous research make use of *scattered* 3-D points or a *single range image* as input. Very few researchers (e.g. [10]) try to use *multiple* densely sampled range images. There are two difficulties in using these range images. First the images must be precisely registered. We have previously presented a method [1] to register multiple range images, which is used to register the range images used in this paper. The second is the issue of integration. Integration can not be performed without an appropriate representation for the integrated data. While star-shaped objects can be simply mapped onto a unit sphere, and the integration can then be performed easily [1]. This is not true for complex objects, since it is difficult to find an integrated representation. Thus finding a suitable representation is very important. While it is not the main theme of this paper, we believe that our approach leads to a good solution to combine and take advantage of multiple range images in surface description for complex objects in terms of integration.

In the following sections, we first review some of the related previous work and then present our balloon model in detail. Section [2.3] describes our surface model and how it works, Sections [2.4] and [2.5] define the dynamics of the system. Section [2.6] explains the adaptive mesh subdivision scheme. In Sections [2.7] and 8, we give an algorithmic description of our system and describe how to set system parameters. Section Chapter 2 discusses issues on how to adapt the parameters locally and dealing with noise. Several test results from real range images, from both simple and complex objects, are presented in Section [2.10]. The conclusion follows in Section [2.11].

2.2 Related Work

Dynamic mesh models have been proposed by previous researchers for shape description. [12] introduced a shape reconstruction algorithm using a dynamic mesh that can dynamically adjust its parameters to adapt to the input data. [13] then extended this approach by introducing an attraction force from the 3-D input for shape description. [6] also proposed a similar system with dynamic nodal addition/deletion for shape description and nonrigid object tracking. [4] proposed a deformable model with both internal smoothness energy and external forces from both the input data and features. There exist other deformable model approaches that differ in the representation schemes of the model and in the approaches to solving the system [5][11].

The main difference between our method and those used by previous researchers is that we do not *explicitly* introduce a data force into our model, as discussed in details in the following sections. Our model is driven by an inflation force introduced inside the balloon. Balloon models have been used by [3] and [7], but in these ap-

proaches, the introduced balloon force is used mostly to overcome noise in the data so that the system can converge to the desired results more easily.

2.3 The Inflating Balloon Model

Our balloon model is represented by a shell of triangulated patches. The initial triangulated shell is an icosahedron. A triangulated shell can be either considered as a mesh consisting of triangular patches or a mesh consisting of vertices (or nodes) connected to their neighbors. In the following discussion, a mesh element may refer to either a vertex or a triangle patch. But in this paper, we mainly explore the properties of the vertices.

When placed inside the object, and under the influence of the inflation force, the shell grows in size as the vertices move along the mesh surface normal in the radial direction, maintaining an isotropic shape, until one or more vertices reaches the object surface. During the process of inflation, the triangles may be subdivided adaptively, which also creates new vertices. Once it reaches the surface, a vertex is considered *anchored* to the surface and thus can no longer move freely. The remaining vertices, under the influence of the anchored vertices, will gradually change their course of movement, until finally reaching their corresponding surface point. As can be seen from this process, the movement of the vertices is not influenced directly by any force from the surface of the object. This seems to be bad from the viewpoint of a fitting process, which tries to minimize some distance measure between the object surface and the model. But this is important to us because our main concern is to find the mapping between the mesh elements and the object surface. Not using any attraction force from the surface data allows us to avoid incorrect mappings, which is a similar situation to the local minimum problem in energy minimization approaches. An example of the growing balloon is shown in Figure 2.2 .

2.3.1 The Correspondence Problem

So far, we have not discussed how to test whether the mesh has reached the surface of the object. This is the key difference between our approach and other dynamic model systems or energy minimization systems. In order to test whether a vertex has reached the object surface, one must measure the distance between the mesh surface and some point on the object surface. Ideally this point on the object surface should be the corresponding point of the vertex, which is not possible before the vertex reaches the surface. Previous researchers have used the closest point on the object surface to an mesh element as an alternative, but it may provide incorrect information. In [7] the distance from the data points on the surface to the nearest model point is used instead, which is an improvement over the above approach. This approach, however, is not practical when there is a large number of surface sample points from the object, as in our case.

In our approach, we look for potential corresponding points only in the direction normal to the mesh surface. This is the best knowledge locally available to the points

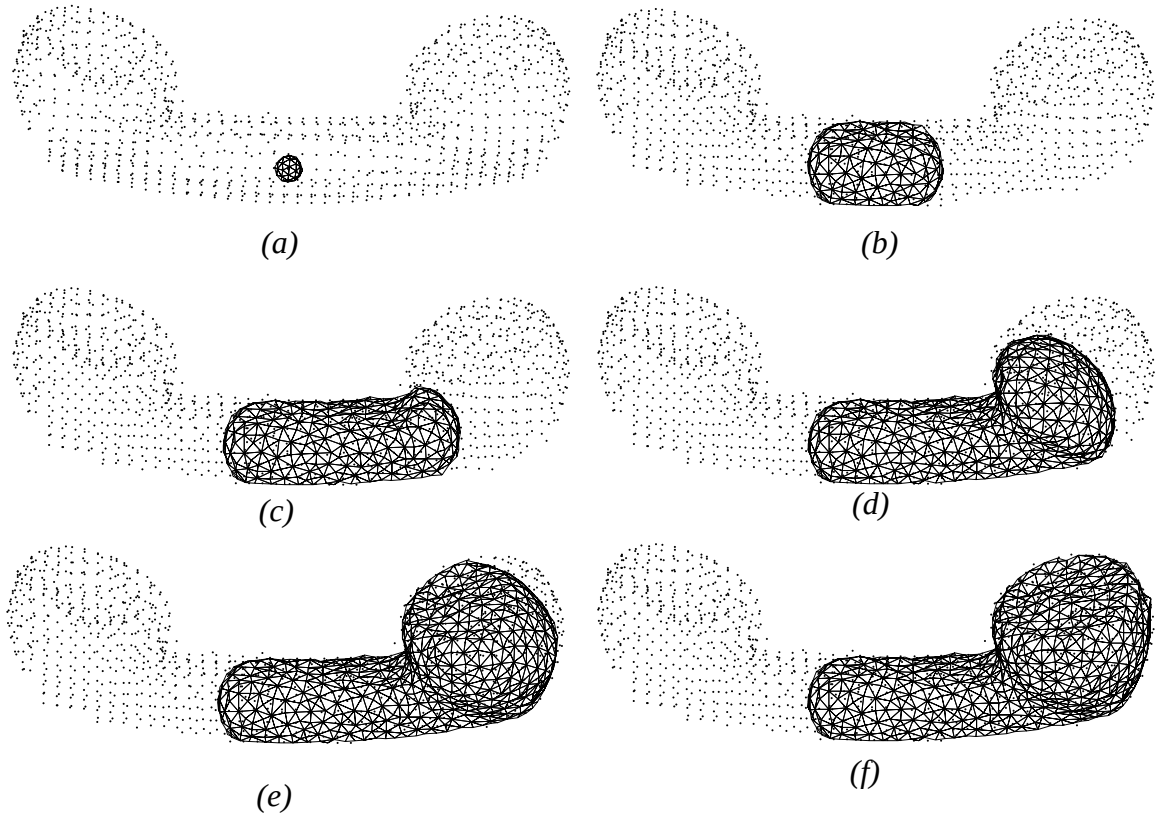


Figure 2.2 Stages of an inflating balloon inside the Phone, showing the movement of the right front only (see Section Chapter 2). The wireframes are superimposed with sample points from the used range images.

on the mesh surface at any time during the mesh's growing process, because the equivalent mesh surface movement in the neighborhood of a mesh element is only in the direction normal to the mesh surface. So it is only natural to look for corresponding point from the object surface in the direction of the normal, which also changes in the process of inflation. In our implementation, this is done by computing the prospective correspondence point *P* (Figure 2.3), the closest intersection of a line in the normal direction and the object surface represented in range images (see Appendix for details). Once the intersection is found, the distance from the mesh surface to the intersection can be used as a measure of whether the mesh has reached the object surface.

When there are holes in the input data (parts of the object surface not covered by the input data), we will not be able to find the intersections described above. In such cases, there are no prospective correspondence points for the affected vertices and thus there is no reason to continue applying inflation force. This kind of decision,

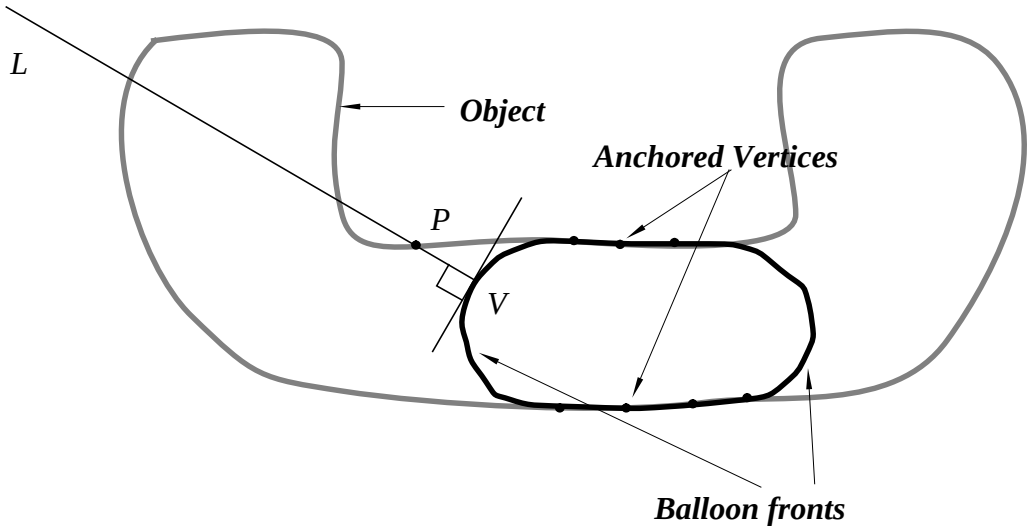


Figure 2.3 Line-Surface intersection: searching for a correspondence point in the normal direction.

however, should not be made locally for each vertex point. We will discuss the handling of holes when we discuss our algorithm in details later, in Section [2.8].

2.4 A Simplified Dynamic Model

The motion of any element i on the model surface can be described by the following motion equation [12]:

$$m_i \ddot{\boldsymbol{x}}_i + r_i \dot{\boldsymbol{x}} + \boldsymbol{g}_i = \boldsymbol{f}_i, \quad i = 1 \ldots N \tag{2.1}$$

where $\boldsymbol{x}_i$ is the location of the element, $\dot{}_i$ and $\ddot{}_i$ are the first and second derivatives with respect to time, m_i represents the mass, r_i is the damping coefficient, $\boldsymbol{g}_i$ is the sum of internal forces from neighboring elements due to, e.g., spring attachments and $\boldsymbol{f}_i$ is the external force exerted on the element. Because of the nonlinear nature of the forces $\boldsymbol{g}_i$ and $\boldsymbol{f}_i$ involved, the systems of ordinary differential equations in Equation (2.1) can be solved using explicit numerical integration [12].

The dynamic system will reach the equilibrium state when both $\dot{}_i$ and $\ddot{}_i$ become 0, which can take a very long time since it is usually an exponential process. A simplified system can be obtained if we make $m_i = 0$, and $r_i = 1$ for all i, in which case Equation (2.1) reduces to

$$\dot{\boldsymbol{x}} = \boldsymbol{f}_i - \boldsymbol{g}_i, \quad i = 1 \ldots N \tag{2.2}$$

There are several reasons for this simplification. First, a zero-inertia system is simpler and easier to control. Second, since there is no inertia, the system will evolve faster in general. Although we are not seeking an equilibrium state for the entire sys-

tem, a simplified system will help speed up reaching local equilibrium states and therefore accelerate the overall dynamic process. Third, a simplified system involves less computation. Also, since we do not intend to have a special treatment for any particular elements in the mesh at this time, all r_i should be equal, in which case we can normalize the parameters so that $r_i = 1$. The set of first-order differential equations in Equation (2.2) has a very simple explicit integration form as follows:

$$\boldsymbol{x}^{t+\Delta t} = \left(\boldsymbol{f}_i^t - \boldsymbol{g}_i^t \right) \Delta t + \boldsymbol{x}^t \tag{2.3}$$

2.5 Spring Force and Inflation Force

The spring force exerted on vertex *i* by the spring linking vertex *i* and *j* can be expressed as [12]:

$$\boldsymbol{s}_{ij} = \frac{c_{ij} e_{ij}}{\left\| \boldsymbol{r}_{ij} \right\|} \boldsymbol{r}_{ij} \tag{2.4}$$

where c_{ij} is the stiffness of the spring, $e_{ij} = \left\| \boldsymbol{r}_{ij} \right\| - l_{ij}$ is the spring deformation, $\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i$, $\| \ \|$ is the vector length operator and $_{ij}$ is the natural length of the spring. The total spring force $_i$ a vertex receives is the vector sum of spring forces from all springs attached to it.

The inflation force a vertex receives takes the form of:

$$\boldsymbol{h}_i = k \hat{\boldsymbol{n}}_i \tag{2.5}$$

where *k* is the amplitude of the force and $\hat{\boldsymbol{n}}_i$ is the direction normal to the local model surface. In our implementation, the normal at a mesh vertex is estimated from the vector sum of the normal vectors of the surrounding triangles:

$$\hat{\boldsymbol{n}}_i = \frac{\boldsymbol{n}_i}{\left\| \boldsymbol{n}_i \right\|}, \qquad \boldsymbol{n}_i = \sum \frac{(\boldsymbol{n}_{ij} + \boldsymbol{n}'_{ij})}{\left\| (\boldsymbol{n}_{ij} + \boldsymbol{n}'_{ij}) \right\|}. \tag{2.6}$$

where $\boldsymbol{n}_{ij}$ is the direction normal to the *j*th triangle $T_j \in \{T_j\}$ surrounding the vertex, and $\boldsymbol{n}'_{ij}$ is the direction normal to triangle T'_j that is the neighbor of T_j but $T'_j \notin \{T_j\}$. This estimation is more stable than the one we get when only the triangles in $\{T_i\}$ are used.

2.6 Subdivision and Adaptation of the Triangular Mesh

In a simulated physical system, during the process of the growth of the mesh model, the mesh triangles increase in size, and tensions due to the spring force also build up, which eventually stops the movement of the mesh, as the inflation force and the spring tension equalize. This is not desirable in our system since we do not consider force from the input data, so that an equilibrium state does not mean a good fit. In order to keep the balloon growing, we can keep the inflation force unchanged

(which actually means to keep on inflating) and at the same time reduce the spring tension by subdividing the triangles in the mesh into smaller triangles. Alternatively, we can increase the inflation force and allow the spring tension to increase. But increasing the inflation force also has the side effect of increasing the maximum displacement. As can be seen from Equation (2.3), the spring force $\boldsymbol{g}_i$ usually acts as a balance force to the inflation force $\boldsymbol{f}_i = \boldsymbol{h}_i$ (assuming a convex local structure), thus the maximum displacement is directly related to the inflation force once a time step is chosen. So we choose not to increase the inflation force in our system, but to subdivide the mesh instead.

The purpose of subdividing triangles is twofold. Once a triangle is subdivided, the sides of the triangles becomes shorter and if we keep the natural length and stiffness of the springs constant, the spring tension is reduced. Also, subdividing the triangles helps maintain an evenly distributed mesh. Subdividing triangles in a certain region, as will be discussed later, also allows the mesh surface to adapt to the local object surface geometry without affecting other parts of the mesh surface.

Before introducing the details of the triangle subdivision process, we first define some terms.

A vertex is said to be *anchored* if it has reached the object surface and has been marked as such. A triangle is said to be anchored if all of its vertices are anchored. At any time in the mesh growing process, the triangles in the mesh can be classified into anchored triangle regions, consisting of anchored triangles, and unanchored triangle regions, consisting of movable triangles, called *front*. Each front is a connected component of triangles, in which two triangles are said to be connected iff they share an edge.

2.6.1 Adaptive Triangle Mesh Subdivision

Triangle subdivision is carried out only on the front, since anchored triangles are not allowed to move. This allows the triangular mesh to adapt to the object surface better without globally adjusting the position of all vertices. A good subdivision scheme is one that yields an evenly distributed mesh and produces few degenerate (i.e. long and thin) triangles. The algorithm that we use in this paper first selects a set of triangles that needs to be subdivided through bisection. Then, after these triangles are bisected on their longest edges, adjacent triangles are also bisected or trisected to make the triangles *conforming*, the state in which a pair of neighboring triangles either meet at the a vertex or share an entire edge. In our implementation, only those triangles that exceed certain size limit are subdivided first. The algorithm presented below is adapted from Algorithm 2 (local) in [8], which is developed for refining triangular mesh for finite element analysis.

2.6.2 Algorithm 1

Let τ_{f0} be the set of the triangles from a given front, and $\tau_0 \subset \tau_{f0}$ is the selected set of triangles to be subdivided.

1) Bisect T by its longest edge, for each $T \in \tau_0$.
2) Find $R_1 \subset \tau_0$ the set of non-conforming triangles generated in step 1. Set $\leftarrow 1$.
3) For each $\in R_k$ with non-conforming point $\in T$ (mid-point on the non-conforming edge): (a) bisect T by the longest edge; (b) if P is not on the longest edge of the T, then join P with the midpoint of the longest edge.
4) Let τ_0^k be the triangulation generated in step 3. Find $_{k+1} \subset \tau_0^k$ the set of non-conforming triangles generated in step 3.
5) If $_{k+1} = \{0\}$, stop, the subdivision is done. Else, set $\leftarrow k+1$ and go to step 3.

This subdivision algorithm has the feature that the subdivision is only propagated towards large triangles from the longest edge of a subdivided triangle. It is also proven that the resulting triangles' smallest inner angle is lower-bounded by half of the smallest inner angle of the original triangles [8].

This algorithm, however, does not guarantee that the triangles on the boundary areas of a front conform with the rest of the triangles in the triangulation. Hence, after the algorithm terminates, we must bisect the affected non-conforming triangles accordingly. Thus we have:

2.6.3 Algorithm 2

1) Carry out Algorithm 1 on the set of triangles $\tau_0 \subset \tau_{f0}$.
2) For each non-conforming triangle $T \notin \tau_{f0}$ but connected to τ_{f0}, bisect T by its non-conforming edge.

An example of the result from this algorithm is shown in Figure 2.4 , where triangle *A* is to be subdivided and *C* does not belong to the region (front). As can be seen from the figure, the subdivision is propagated to *B*, and finally *C* is bisected to make the triangles at the region boundary conforming (step [2] above).

2.6.4 Local Mesh Adjustment

The above algorithm works very well under most circumstances, but degenerate triangles that are long and thin may still occur. These triangles are undesirable since they do not represent local surface shape well and are often the cause of self-intersection of the mesh surface. Currently, we use a simple algorithm that checks for pairs of such triangles and rearrange the triangle configuration locally. After each subdivision, we check for triangles that are thin and long, and if two such triangles share an

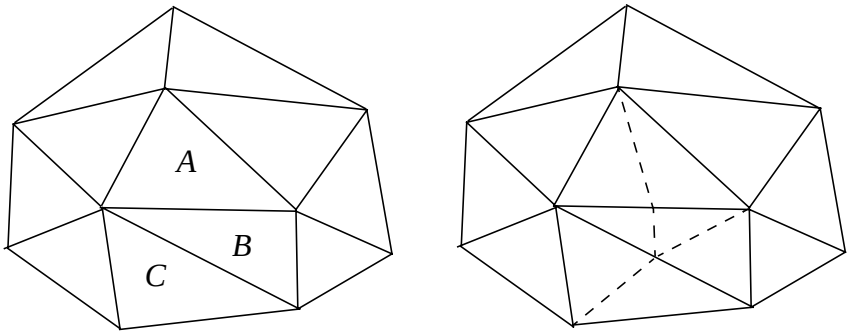


Figure 2.4 Subdivision of triangle mesh. (a) before A is subdivided, (b) after A is subdivided and the subdivision is propagated to both B and C.

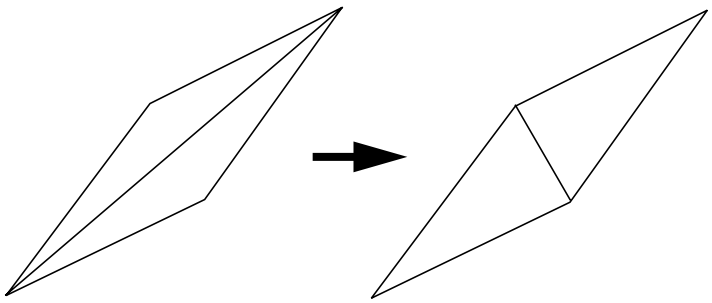


Figure 2.5 Rearranging the local configuration to eliminate long and thin triangles

edge that is the longest for both triangles, then we simply switch the cross edge as shown in Figure 2.5 .

2.7 Description of the Algorithm

In this section we give a brief description of the entire algorithm of our approach. A discussion on how to set the system parameters will follow. We assume that registered range image views of the object to be modeled are available, although we believe the algorithm can be adapted to other types of 3-D input.

We start with selecting an initial point inside the object and constructing an icosahedron shell [13] at this location. The selection process is currently done by hand and the size of the shell should be small enough so that it is completely inside the object. Since the algorithm does not depend on the actual location of the initial shell, as

long as it is inside the object, an alternative to manually selecting the initial position is to choose a smooth patch in any range image and place the shell under the patch. The system algorithm can be described as follows.

2.7.1 Algorithm 3

Let all the triangles on the initial mesh be front F_0 and push it into the front queue Q, then until queue is empty, do the followings repeatedly:

1) $F \Leftarrow$ top of the queue Q, pop the queue.
2) Subdivide the triangles in F if appropriate (see next section).
3) For each vertex $_i \in F$, whose 3-D coordinates at time t is $\boldsymbol{v}_i^t$, do
   a) compute the internal force $\boldsymbol{g}_i$ and external force $\boldsymbol{f}_i = \boldsymbol{h}_i$ based on Equations (2.4) and (2.5).
   b) compute the new vertex location $\boldsymbol{v}_i^{t+\Delta t}$ for the current iteration according to Equation (2.3).
   c) compute prospective correspondence point of $\boldsymbol{v}_i$, which is the intersection $\boldsymbol{w}_i$ of the surface and the line through $\boldsymbol{v}_i$ and in the direction of the mesh normal at $\boldsymbol{v}_i$ (see below).
   d) if $\left\|\boldsymbol{v}_i^{t+\Delta t} - \boldsymbol{v}_i^t\right\| > \left\|\boldsymbol{w}_i - \boldsymbol{v}_i^t\right|$, then $\boldsymbol{v}_i^{t+\Delta t} \Leftarrow \boldsymbol{w}_i$ and mark vertex V_i anchored.
4) For each $_i \in F$, update its position with the corresponding new positions $\boldsymbol{v}_i^{t+\Delta t}$.
5) Discard triangles from F that have thus become anchored (section [2.6]).
6) if $F = \{\varnothing\}$ then go to 1.
7) recompute connected triangle regions in F and push them into Q. Go to 1.

In step (3)(c) above, an algorithm that computes the intersection between a 3-D line and the object surface in range image form is called for. This algorithm gives the closest intersection of a line, which passes through a given vertex point and is in the direction of the estimated local mesh surface normal at the vertex, and the object surface (point P in Figure [2.3]). This is for the purpose of estimating the distance of the vertex to the prospective corresponding points on the surface of the object (Section [2.3.1]). Details of the algorithm can be found in the appendix.

2.8 Setting up the Parameters

In our current implementation, triangles that have areas larger than a threshold S_t are subdivided at each iteration. S_t is directly related to the precision of the fit of the final mesh to the input surface data. Assuming that our goal is to approximate the object surface to have a triangle fitting error δ for surfaces with maximum curvature of $1/R_t$, S_t can be easily computed by tessellating a unit sphere of radius R_t with equilateral (or near equilateral) triangles of sizes smaller or equal to S_t. This also gives us a sample configuration of an ideal front structure when the maximum

mesh tension is achieved. Let $f_{spring-max}$ be the maximum spring force exerted onto a vertex under such conditions. The inflation force needed to overcome the spring force (in order for the vertices to move) is therefore

$$f_{inflate} > f_{spring-max} \tag{2.7}$$

The inflation force is also constrained by Equation (2.3), since once a time step and a maximum displacement per iteration are set, the allowed inflation force should then be (considering 0 spring force):

$$f_{inflate} \leq \frac{d_{max}}{\Delta t} \tag{2.8}$$

where $d_{max} = \max(\|\boldsymbol{x}^{t+\Delta t} - \boldsymbol{x}^{t}\|)$ is the maximum displacement. Since a large inflation force tends to dominate the mesh's evolution, which is undesirable, we prefer a smaller one. We choose to use the minimal inflation force as shown in Equation (2.7). We can then compute the needed inflation force amplitude k according to Equation (2.5).

Now the whole issue comes down to determining $\boldsymbol{f}_{spring-max}$, d_{max} and the time step Δt. The maximum spring force is determined by the spring natural length l_{ij} and the spring stiffness which are related (Equation (2.4)). In our experiments, we have used $l_{ij} = 0$ and $c_{ij} = 4.0$. d_{max} and Δt are selected to allow the mesh to evolve smoothly and quickly relative to the object size and complexity. For all the tests in this paper, we have used $2mm$ and 0.05 respectively.

Finally, the user needs to select δ and R_t. For the purpose of simplicity, in our experiments, however, we manually set R_t and allow a fixed number of N triangles to fit the sphere with a radius R_t, which gives a nominal approximation error of about $0.6mm$ with $N = 80$ and $R_t = 10mm$.

2.9 Adaptive Local Fitting, Holes and Noise

It is also worth mentioning that our algorithm is parallelizable since the computations on each front in the queue Q are independent of each other. Furthermore, the computation for each vertex within each front is also independent during each iteration.

Another advantage that this computation structure brings us is that we can adaptively adjust system parameters independently for each front based on the information that we gather from the prospective correspondence points of the vertices in the front. For example, if we have detected that the movement of the front is virtually stopped and yet the prospective correspondence points are still certain distance away, this tells us that the preset parameter R_t in previous section is too large and we should adjust it accordingly.

Another example of such adaptation is in handling holes in data. In this case, there exist areas of the object surface that are not covered by any of the input range

images, we will not be able to find prospective correspondence points for some of the vertices in the related front. Eventually, when the rest of the vertices in the front have settled down to their correspondence points, we are left with a front for which none of the vertices have a prospective correspondence point. In such situations, the system automatically sets the inflation force to zero ($k = 0$ in Equation [2.5]), which makes the mesh reach an equilibrium state that interpolates the surface over the hole.

Another important issue is the issue of noise. There are two type of noises that may affect out results. One is the noise introduced by the small misalignment among the range images. The other is the spontaneous outliers from each range image. Our system is very stable with respect to both types of noise. The first one is effectively solved by the weighted sum line-surface intersection algorithm (see Appendix) since the misalignment causes the actual intersections to form a cluster. The second type of noise usually cause the intersection algorithm on the related range image to fail to converge, in which case it does not contribute to the result of the intersection. Even if the noise does produce a wrong intersection, it can easily be filtered out as an outlier that does not belong to the correct cluster.

2.10 Test Results

We now present examples of our system in modeling a telephone handset (Phone) and an automobile part (Renault) using 20 and 24 range image views respectively, along with two examples for simpler objects. The range images are acquired using a Liquid Crystal Range Finder (LCRF) [9], and then registered using the range image registration algorithm described in [1]. Some sample range images used in the experiments are shown in Figure 2.6 . Figure [2.7] shows two examples of the balloon model in fitting two simple objects: the Wood Blob and the Tooth. In Figure 2.8 , the final rendered views of the constructed model of the Phone are shown below the wireframe drawing. Figure 2.9 shows a wireframe and the rendered image of the Renault part. The final model for the Phone has 1694 vertices and 3384 triangles, the Renault part has 2850 vertices and 5696 triangles. The total run time excluding registration on a Sun Sparc-10 running Lucid Common Lisp version 4.0 is 16'17" for the Phone and 32'26" for the Renault. Both the Phone and the Renault part measure about $00mm$ across their longer sides. Note that the wireframe drawings in the presented results are not produced using a hidden-line elimination algorithm, which is the cause of most of the spurious triangles seen in the wireframe drawings, including the "defects" in the middle which actually corresponds to a step at the back of the object.

As can be seen from the results presented above, our algorithm works very well for both simple, compact objects, as well as non-star shaped objects with complex structures. The resulting triangulated model surfaces preserve most of the important geometry feature of the objects with evenly distributed meshes. Our initial guess are all set in the neighborhood of the center of the objects and yet our balloon can successfully grow to cover all parts of the object with complex geometric structures such as the Renault part. Also, it is hard to visualize this, but the data that we use for the

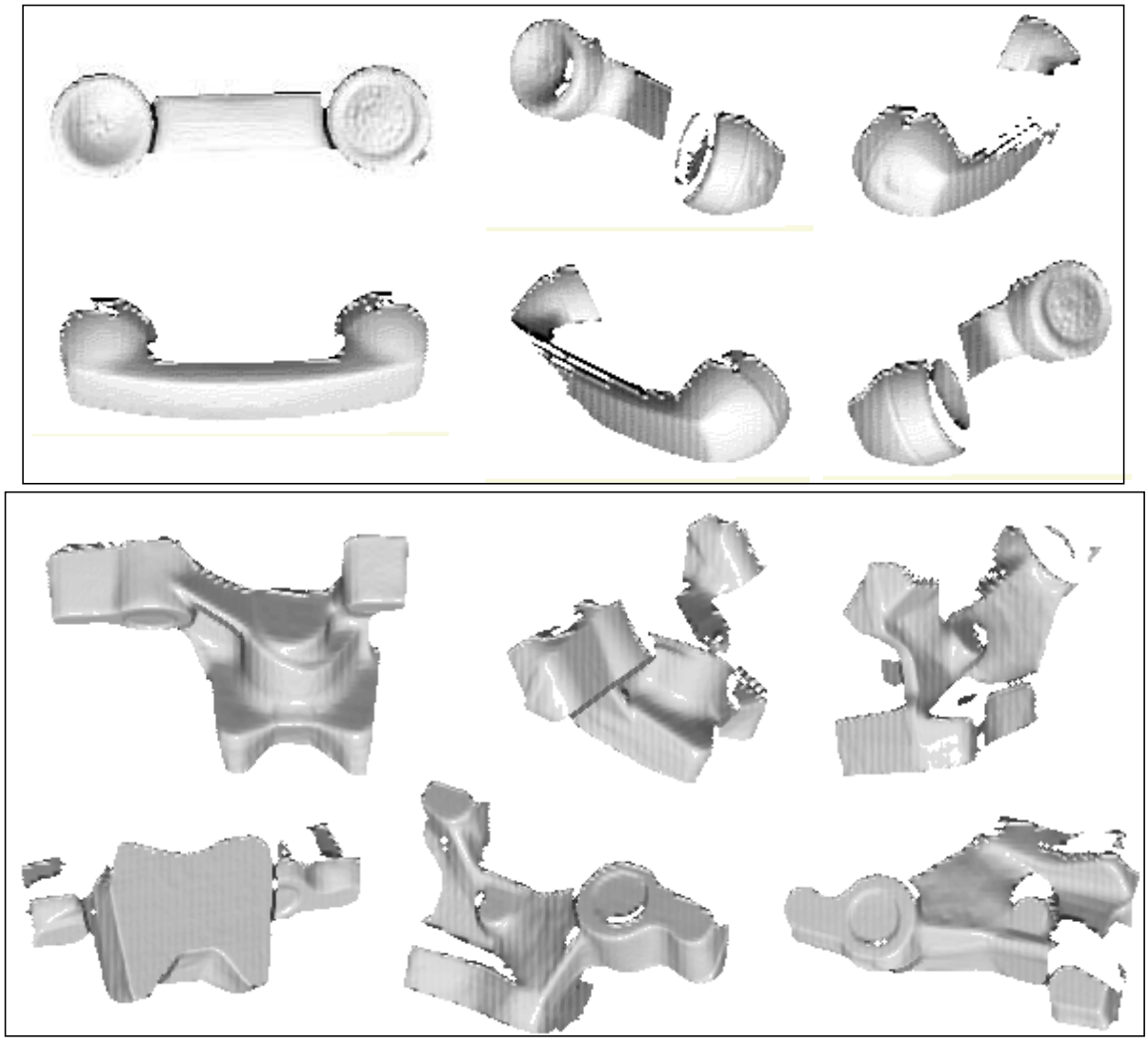


Figure 2.6 Sample range images used in constructing the Phone model and Renault model in Figures [2.8] and [2.9], shown here as shaded intensity images.

Renault part contain holes on top of both of the arms, and the resulting mesh was able to interpolate them very well. There is, however, a defect under the right arm of the Renault part, as can be seen in the wireframe drawing. It is a small opening in the mesh that tends to self-intersect which is caused by a small narrow ridge section (about 8*mm* thick, much smaller than 2 times R_t, where $_t = 10\,mm$). We believe that this can be solved by examing and identifying local surface changes more closely and adjusting system parameters accordingly in that area.

2.11 Conclusions and Future Research

We have presented a surface description method based on a dynamic balloon model using a triangular mesh with springs attached to the vertices. The balloon

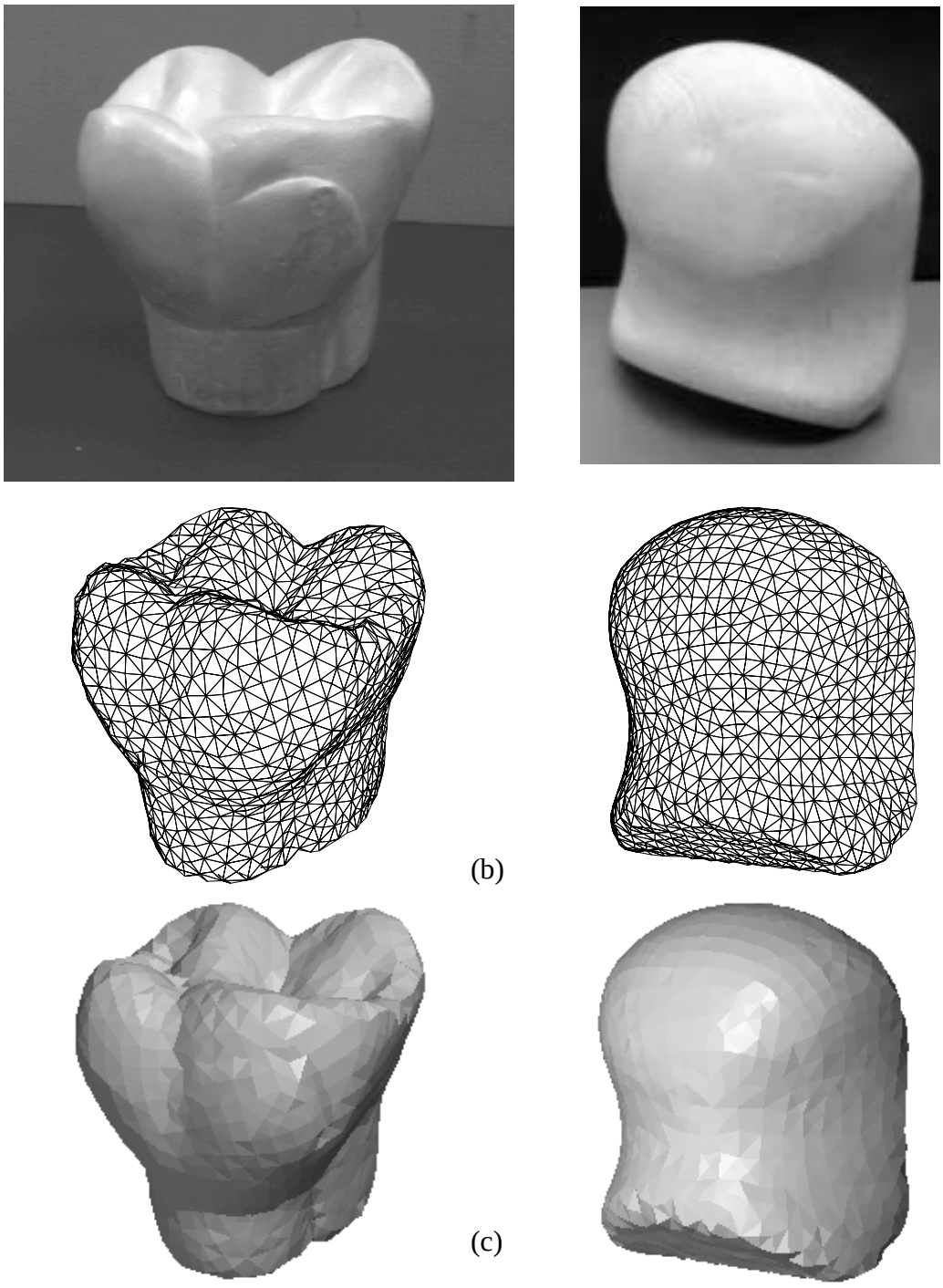
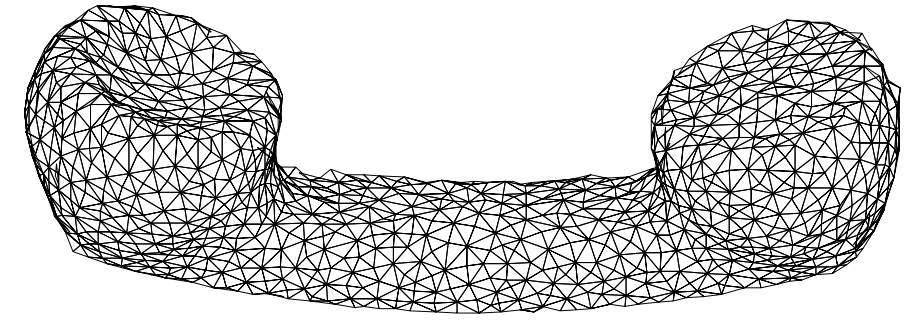


Figure 2.7 Examples of the balloon model for fitting simple objects: (a) the original intensity images of the objects, (b) the wireframes of the obtained balloon models and (c) the rendered shaded images of the models.

model is driven by an applied inflation force towards the object surface from inside of the object, until all the triangles are anchored onto the surface. The model is a physically based dynamic model and the implementation of the algorithm is highly paral-



(a)

(b)

(c)

Figure 2.8 The final balloon model for the Phone: (a) wireframe (b), (c) smoothly shaded.

lelizable. Furthermore, our system is not a global minimization based approach and

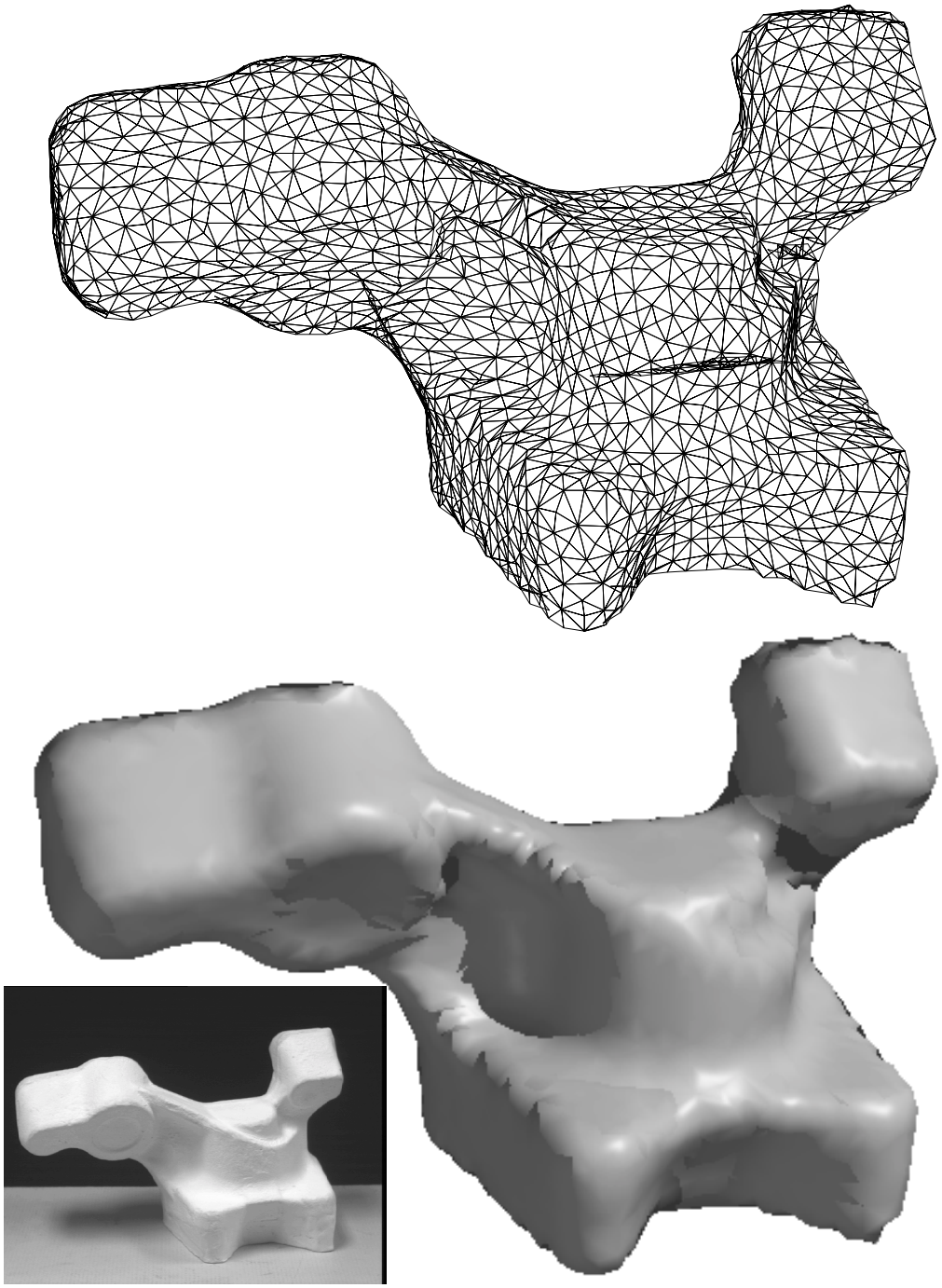


Figure 2.9 The wireframe and the rendered image of the reconstructed model for the Renault automobile part. The inserted picture is the intensity image of the actual object. Note that the wireframe is not produced by a hidden-line removal algorithm (see Section Chapter 2 on page 17).

can allow the model to adapt to local surface shapes based on local measurements. Tests showed very good results on complex, non-star-shaped objects.

As stated earlier, our goal is to achieve a correct mapping of a triangulated mesh to the surface of an object, hence there are still many ways to improve the resulting model we have achieved, including using the algorithms presented in [2] to improve triangle fitting errors, or the method in [10] to merge small triangles into larger ones without affecting the fitting error for constructing a hierarchical representation. Local smooth patches can also be constructed for high level surface property analysis. Alternative surface models, such as a smooth finite element surface model ([7]), can also be used so that the implementation of the system elements can be made more precisely. In addition, our future research consists of detecting and avoiding possible self-intersections of the mesh surface.

2.12 References

[1] Y. Chen and G. Medioni, "Object Modelling by Registration of Multiple Range Images," *International Journal of Image and Vision Computing*, 10(3):145–155, April 1992.

[2] Y. Chen and G. Medioni, "Surface Level Integration of Multiple Range Images", in *Proceedings of the Workshop on Computer Vision for Space Applications*, Antibes, France, September 1993.

[3] L. D. Cohen and I. Cohen. Finite-element Methods for Active Contour Models and balloons for 2-D and 3-D Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:1131–1147, 1993.

[4] H. Delingette, M. Hebert, K. Ikeuchi, "Shape Representation and Image Segmentation Using Deformable Surfaces," *CVPR 1991* pp.467-472.

[5] Andre Gueziec, "Large Deformable Splines, Crest Lines and Matchings", *Proceedings of the International Conference on Computer Vision*, pp. 650-657, Berlin, Germany, May 1993.

[6] W.-C. Huang and D. B. Goldgof. Adaptive-Size Meshes for Rigid and Nonrigid Shape Analysis and Synthesis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(6):611–616, June 1993.

[7] T. McInerney and D. Terzopoulos, "A Finite Element Model for 3D Shape Reconstruction and Nonrigid Motion Tracking", *Proceedings of the International Conference on Computer Vision*, pp. 518-523, Berlin, Germany, May 1993.

[8] M. Cecilia Rivara, "Algorithms for Refining Triangular Grids Suitable for Adaptive and Multigrid Techniques", *International Journal for Numerical Methods in Engineering*, Vol. 20, pp. 745-756, 1984.

[9] K. Sato and S. Inokuchi, "Range-Imaging System Utilizing Nematic Liquid Crystal Mask," In *Proceedings of the IEEE International Conference on Computer Vision*, pages 657–661, London, England, June 1987.

[10] M. Soucy and D. Laurendeau, “Multi-resolution surface modeling from range views,” In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 348–353, Urbana-Champaign, IL, June 1992.

[11] D. Terzopoulos and D. Metaxas. Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(7):703–714, July 1991.

[12] D. Terzopoulos and M. Vasilescu, “Sampling and Reconstruction with Adaptive Meshes” *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pp.70-75, Maui, HI, June 1991.

[13] M. Vasilescu and D. Terzopoulos, “Adaptive Meshes and Shells: Irregular Triangulation, Discontinuities, and Hierachical Subdivision,” *Proceedings of the Conference on Computer Vision and Pattern Recognition,* pp. 829-832. Urbana-Champaign, IL, June 1992

2.13 AppendixThe Line-Surface Intersection Algorithm

We describe an algorithm for computing the intersection of a directed line (a ray) and the object surface represented by a set of registered range images. This algorithm is based on the line-surface intersection algorithm in [1], which computes the intersection between a line and a digital surface represented by a single range image.

We begin with a brief description of the original algorithm. As shown in Figure 2.10 , we are given a directed line $\boldsymbol{l}$ that passes through a certain point $\boldsymbol{p}$, the intersection, $\boldsymbol{q}$, of $\boldsymbol{l}$ and the surface Q can be computed as follows. We first project $\boldsymbol{p}$ onto Q in its image space and find the tangent plane of Q at the projection. Then we compute the intersection $\boldsymbol{q^0}$ of the plane and $\boldsymbol{l}$, which becomes the first approximation of the intersect we are looking for. We repeat the process by projecting the intersection approximation $\boldsymbol{q^k}$ onto Q at each iteration. When this process converges, the resulting intersection is taken as $\boldsymbol{q}$. Note that in this approach, we also have a directional constraint for the found intersection, which states that the local surface normal at the computed intersection point must be within 90° of the direction of the ray, which is the direction of the mesh surface normal in this paper.

When we have more than one range images, the intersection of the line with all the range images are computed. Let $\{Q_i\}$, $i = 1 \ldots m$, be the set of range images and $\boldsymbol{l}$ be the line in consideration. The intersection of $\boldsymbol{l}$ and the surface $\{Q_i\}$ can be defined as the weighted sum of all the intersections:

$$\boldsymbol{q} = \frac{\sum_{i=1}^{m} a_i w_i \boldsymbol{q}_i}{\sum_{i=1}^{m} a_i w_i}, \qquad w_i = (\hat{\boldsymbol{n}}_{q_i} \cdot \boldsymbol{n}_s) \tag{2.9}$$

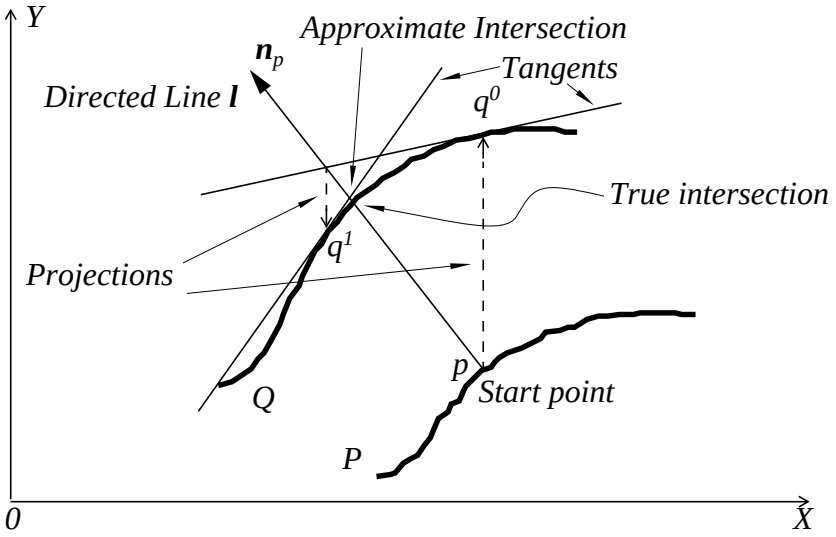


Figure 2.10 Intersecting a line with a digital surface illustrated in a 2-D case.

where $\boldsymbol{q}_i$ is the intersection of $\boldsymbol{l}$ and Q_i, $\hat{\boldsymbol{n}}_{q_i}$ is a unit vector normal to Q_i at $\boldsymbol{q}_i$, $_s$ is the vector pointing towards the sensor and a_i is a binary number depending on the intersection of $\boldsymbol{l}$ and Q_i, and

$$a_i = \begin{cases} 1, & \text{if the intersection exists} \\ 0, & \text{if there is no intersection} \end{cases}$$

The reason behind taking a weighted sum of the intersection points is that the sensor measurement of the position of a surface point is less reliable if the local surface is facing away from the sensor. The weight w_i is a reflection of this heuristic. In general, the sensor direction information $_s$ is available from the range image sensor setup and calibration. For a Cartesian range image (depth map) without sensor information, we can simply take $\boldsymbol{n}_s = (0, 0, 1)^\tau$.

If there are more than one intersections between $\boldsymbol{l}$ and the object surface, we can perform a clustering to separate the intersections into groups corresponding to each real intersection, and choose the closest cluster to compute the above weighted sum. In practice, we have not had to use such a clustering algorithm. This is because the result of the intersection algorithm depends on an initial point $\boldsymbol{p}$ (which is the vertex point in this paper). When the vertices are far from the object surface, $\boldsymbol{q}_i$ can be from any clusters. But we are less concerned with the actual location of the intersection at the time. As the mesh grows and the vertex get closer to the surface, almost all the intersections computed are from the closest cluster, since it is the closest local minimum when considering the intersection process as a minimization. For robustness purpose, we have implemented a simple filtering scheme to eliminate gross outliers in the intersections based on the distribution of the intersections found.