Creating Generative Models from Range Images

Ravi Ramamoorthi
Stanford University*
ravir@graphics.stanford.edu

James Arvo
California Institute of Technology
arvo@cs.caltech.edu

Abstract

We describe a new approach for creating concise high-level generative models from range images or other approximate representations of real objects. Using data from a variety of acquisition techniques and a user-defined class of models, our method produces a compact object representation that is intuitive and easy to edit. The algorithm has two inter-related phases: *recognition*, which chooses an appropriate model within a user-specified hierarchy, and *parameter estimation*, which adjusts the model to best fit the data. Since the approach is model-based, it is relatively insensitive to noise and missing data. We describe practical heuristics for automatically making tradeoffs between simplicity and accuracy to select the best model in a given hierarchy. We also describe a general and efficient technique for optimizing a model by refining its constituent curves. We demonstrate our approach for model recovery using both real and synthetic data and several generative model hierarchies.

CR Categories: I.3.5 [*Computer Graphics*]: Object Modeling—Curve, surface, solid, and object representations, Object hierarchies, Splines; I.4.8 [*Image Processing and Computer Vision*]: Scene Analysis—Object Recognition, Surface Fitting

Keywords: Generative Models, Range Images, Curves and Surfaces, Procedural Modeling

1 Introduction

It has recently become feasible to acquire reasonably accurate point-clouds or *range data* from 3D objects [5, 26]. For graphics applications, these point-clouds are usually transformed into polygonal meshes [5, 12], or spline patches [14]. However, these approaches often provide an unintuitive representation that is difficult to manipulate once generated. In addition, a large amount of data is required since the meshes usually contain thousands of triangles. For modeling many man-made objects, *generative models* proposed by Snyder [21, 22] provide an attractive alternative. A generative model is a generalization of a swept surface in which the generating curve can be continuously transformed by one or more arbitrary curves. For instance, a banana-like shape is specified parametrically by translating a cross-section while scaling and rotating it. The cross-section, scaling function, and rotation are all described by curves. The modeled object is analytically represented by a tree of operators that provides a logical description of its structure. Designers can easily construct, examine, and modify such a model.

*Address: Gates Wing 3B-372, Stanford University, Stanford, Ca 94305.

Figure 1: *Generative models recovered from actual range data. Objects were scanned individually, modeled using our algorithm, then composed in this scene with artificial color and shading. A smooth compact representation was generated for each object from a few simple model hierarchies, despite noisy and incomplete data. See figure 7 for a comparison of range data and recovered models.*

We describe a method for inverting this process to recover generative models from range data or other approximate representations of object geometry. This method requires the user to specify a universe of possible objects by providing a hierarchy of shape operators defining a class of generative models. Given a user-defined hierarchy, such as the ones shown at the bottom of figure 2 and in figure 8, the system automatically selects a parametric model within this hierarchy, and adjusts its parameters to best match data acquired from an actual object. Several models recovered in this way are shown in figure 1.

Our approach for model construction has a number of benefits:

Simplicity: Range data may be obtained by several methods, including the *shadow* approach of Bouguet and Perona [2], that requires only a lamp, pencil and checkerboard apart from the camera.

Robustness: By exploiting redundancies in the data, accurate models can often be recovered from incomplete and noisy data. Our model-based approach is thus much less sensitive to noise than mesh creation. Many of our examples use only one noisy and incomplete range image, while creation of a polygon mesh generally requires many accurate and properly aligned range images.

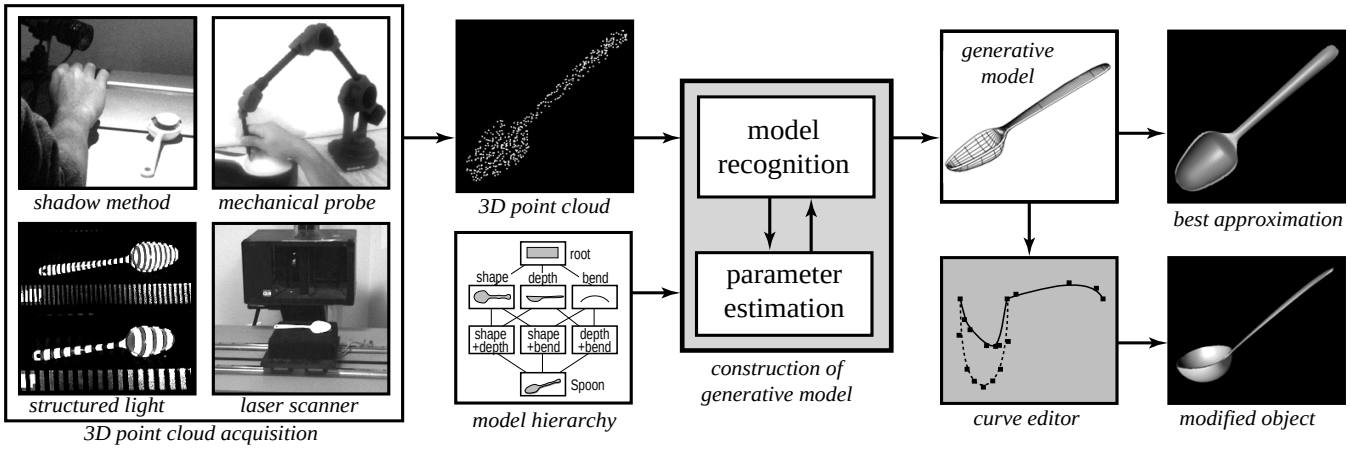

Figure 2: *Overview of generative model creation. The algorithm takes range data in the form of a point-cloud and a generative model hierarchy as input. An appropriate model is then chosen, and parameters are optimized to output an accurate and concise generative model that can subsequently be edited.*

Compactness: Generative models provide a concise representation; we need only store an algebraic model description, and control points of the model's parametric curves. This representation can be orders of magnitude smaller than a triangle mesh.

Intuitiveness: Since the model is expressed in terms of parametric curves corresponding to logical features of the object, it is easy to understand, manipulate, and edit.

Related Work

Many methods have been explored, especially in computer vision, for recovering object shape for specific primitives such as generalized cylinders [1, 15], superquadrics [16, 23] and blended deformable models [7]. Terzopoulos and Metaxas [24] have proposed a computational physics framework for shape recovery in which globally deformed superquadrics model coarse shape and local deformations add fine detail. Superquadrics have also been used for model-based segmentation [9, 11], and for recognition of "geons" using relationships between superquadric parameters [19, 27]. DeCarlo and Metaxas [8] introduced shape evolution with blending to recover and combine superquadrics and supertoroids into a unified model. Debevec et a. [6] considered architectural scenes and developed a system for recovering polyhedral models from photographs.

Our approach is somewhat more general than these previous algorithms in that it is based on a general user-specified generative hierarchy rather than a particular parametric model. This allows automatic construction of more complex and varied shapes, without segmentation, than is possible with current computer vision algorithms. Further, many standard primitives used in computer vision can be recovered using our method since generative models are a superset of traditional shape representations such as globally deformed superquadrics, straight homogeneous generalized cylinders, and blended deformable models, all of which have been recovered using our system. Our method is also automatic, with no user intervention required. However, model-specific algorithms, especially those that allow user-intervention, may out-perform our algorithm on the shapes to which they apply by exploiting model-specific information. For instance, by considering only polyhedral models, and having the user manually specify the edges of interest, Debevec et al. [6] are able to work with only photographs, while we require range data.

This paper deals primarily with shapes represented by a single generative model. At present, we do not add local detail [24], nor address automatic model-based segmentation [8, 9, 11] or image interpretation [3]. However, our results suggest that generative models may be useful for these tasks in lieu of superquadrics or generalized cones.

In contrast to some object recognition methods [19, 27],which estimate specific model parameters to classify an object as a member of some class, our method first determines which degrees of freedom in the model hierarchy are most suitable for the acquired data, and then refines the associated parameters.

Recent work on simplifying polygonal meshes shares one of our objectives—providing a more compact representation. For example, Hoppe et al. describe techniques to optimize meshes [13], while Eck and Hoppe [10] and Krishnamurthy and Levoy [14] fit spline surfaces to dense meshes. However, mesh-based methods do not yield compact *high-level* models.

The rest of this paper is organized as follows: Section 2 gives an overview of our algorithm and describes our framework for recovering the appropriate model within a user-specified hierarchy. In section 3, we discuss our methods for optimization. Section 4 briefly outlines the various models used in our tests. In section 5, we discuss our results and section 6 presents our conclusions and directions for future work.

2 Algorithm Framework

In this section, we give a high-level overview of the entire algorithm, pictured in figure 2, and describe our method for automatically choosing the appropriate generative model from within a user-defined class. This is essentially a recognition task as it requires the measured data to be classified as one of the models in the user-specified hierarchy. The recognition process is based on a simple tradeoff between accuracy and simplicity. For efficiency, a *greedy algorithm* is employed that starts with the simplest model in the input hierarchy, and then considers more complex models at the next level in the hierarchy. The system selects the model providing the greatest benefit, and repeats the process in a greedy fashion, moving through the hierarchy from simple to more complex models. The process stops when none of the more complex models significantly improves the accuracy, or the most complex model is reached. Although the first model that is fit to the data is trivial, the algorithm then *bootstraps* itself by using information obtained in

fitting previous models, improving at each stage until an accurate and suitably complex model is recovered. For illustrative purposes, we will often refer to the specific hierarchy shown at the bottom of figure 2 and on the left of figure 8, which is inspired by the spoon model created by Snyder [21, p. 83].

The model hierarchy has several levels. For our class of models, the root node of the hierarchy is level zero, which consists of a "half-cylinder" with two global parameters controlling width and depth. This object essentially defines a bounding volume for the data. Deeper levels consist of *refining* one or more of the parameters by representing them as curves instead of global values. In general, one curve is added at each level so the number of curves corresponds to the level. For example, the edge to the "depth" node in figures 2 and 8 corresponds to *refining* the depth parameter by representing it as a curve. The hierarchy can also be thought of as a tree; going from parent to child corresponds to adding a single curve. For instance, the *root* is the *parent*, while the model with refined depth is the *child*. The tree representation implies an order in which curves are added. The curve providing the most benefit is added first, and a child node inherits initial parameter estimates from optimized results for the parent node.

Input to the System: Part of specifying the model hierarchy is to supply functions that perform the following tasks.

- **Initial Guess for Root Model:** Starting values must be supplied for the parameters of the root model, which typically consists of a simple primitive object. The initial values for both the intrinsic parameters and the extrinsic parameters (translation, rotation, and scale) of the root model may be very crude, since they merely provide a starting point for subsequent refinement and optimization. Parameter estimates for more complex models are obtained automatically from those of the parent model in the tree.
- **Model Evaluation:** A function must be supplied to evaluate any model in the input hierarchy at given uv-parameters. Efficient routines can be automatically generated from an algebraic model description.
- **Curve Constraints:** The model hierarchy may optionally contain additional constraints on the curves in the final model, such as fixing their values at specific points, or penalty terms to ensure, for instance, that a particular curve remains positive everywhere.

This information is encapsulated as user-supplied functions along with the code that defines the model hierarchy. Thus, the algorithm may proceed without any manual user intervention.

A summary of our algorithm is shown in figure 3. Below, we discuss each step in detail. The reader may wish to refer to the results in figure 8 and the left of figure 9 for examples of applying the algorithm.

Step 1. Acquire Range Data: We have used a number of different methods to acquire range data for our experiments, including two structured light techniques—a method which uses a sequence of alternating dark and light patterns projected onto the object [25], and the highly portable method described by Bouguet and Perona, in which shape is inferred from the shadow of a rod swept over the object [2]. We have also used a mechanical probe and a laser range scanner. This variety of sources demonstrates that our algorithm is amenable to a wide assortment of data acquisition methods.

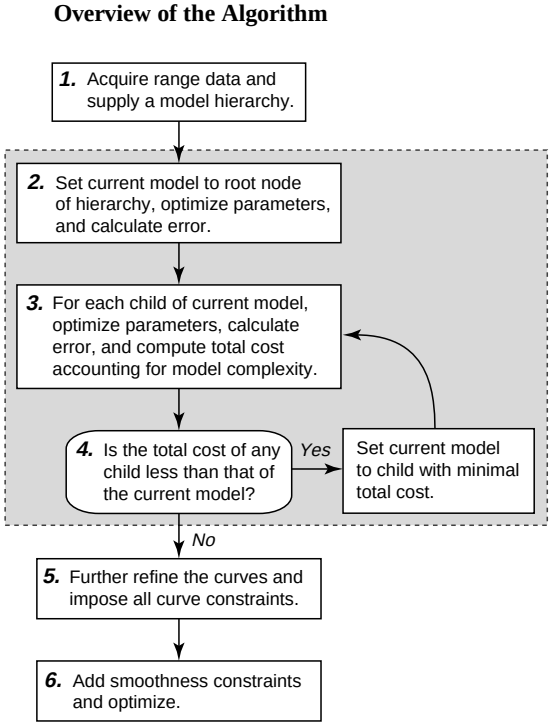

Figure 3: *Overview of the algorithm with the greedy algorithm used for recognition highlighted.*

Step 2. Fit to root model:

- **Initialize:** Let ρ denote our current model—initially the root node of the hierarchy. The root node's intrinsic and extrinsic parameters are initialized using a user-specified function.
- **Optimize:** An error-of-fit function ϕ is computed based on the spatial deviation between the data and the model as defined in equation 4. Optimization is used to adjust the root node to minimize the error-of-fit. Details are given in the next section.
- **Cost:** After minimization, ϕ is used to compute the deviation D which represents the RMS distance between model and data as defined in equation 5. The total cost C is initialized to this deviation.

Step 3. Fit children to Data: For each child of the current model ρ, denoted by $\gamma_i(\rho)$, we calculate the deviation $D(\gamma_i)$ after optimization. For instance, if $\rho = root$, the children are $\gamma_1 = shape, \gamma_2 = depth, \gamma_3 = bend$. We then calculate a cost function for each child:

$$C(\gamma_i) = D(\gamma_i) + \triangle(\gamma_i), \tag{1}$$

where $\triangle$ is a penalty for model complexity. We use a very simple but effective heuristic to assess the complexity of a model: $\triangle(N) = NQ$ where N is the level in the hierarchy of a specific model and Q is a constant that allows the user to control the trade-off between simplicity and accuracy.

In progressing from the current model ρ to a child γ, we add a single curve. The entire process is explained in detail with results in subsection 3.3 on curve refinement. The steps are:

- **Initialize:** Initial parameter estimates for γ are set to parameter values of its parent ρ. The new curve to be added is initialized to a constant inherited from ρ, and control points are added at the ends.
- **Optimize:** A first optimization step yields a coarse curve estimate. In all optimization steps, all model parameters are simultaneously varied to minimize the objective function in equation 4.
- **Refine:** A few interior control points are automatically added based on the heuristic defined in equation 16.
- **Optimize:** Optimization is repeated with the additional control points added, which improves the quality of the new model.
- **Cost:** We compute the deviation D using equation 5 as in step 2, and use equation 1 to compute the total cost of γ.

Step 4. Reset ρ: The previous stage calculated the cost function for all the children of ρ. If the child with lowest cost $C(\gamma_i(\rho))$ has a lower cost than ρ, we make it the new best fit ($\rho = \gamma_i$) and go back to step 3. Otherwise, exit to step 5.

For efficiency, we use a **greedy** algorithm to choose the best model. We consider only the descendants of the current model ρ to choose the next model. Conversely, a node is considered only if it is linked to the best guess ρ at some point. Thus, curves are added in order of importance, with the one reducing the objective function the most added first. This approach works best when a particular curve is clearly more important than other choices, or when curves may be refined in any order with similar results. In cases where two curves produce similar results early on, but one branch later proves to be clearly superior, a more exhaustive search algorithm with backtracking would achieve better results.

Steps 2-4 constitute the *Recognition Phase* of the algorithm where an appropriate model is chosen by identifying a path through the model hierarchy as shown in figure 8. The goal is to quickly choose the appropriate model in the hierarchy i.e: to identify which curves and deformations are required to model the data. Since this process merely chooses a suitable model, it is appropriate to represent the curves at a coarse level of detail. However, since we wish to eventually recover an accurate final model, we further refine the "recognized" model ρ in the next step.

Step 5. Refine curves further and add curve constraints:

- **Refine:** Using the heuristic in equation 16, control points are automatically added where necessary. More control points are added than in the *refine* phase of step 3 above, as this allows the curve to be represented at a finer resolution; the number of control points used is given by equation 19.
- **Curve Constraints:** We then add user-specified constraints to the curves such as fixing the values at the end-points. Typically, these constraints improve the visual accuracy of the final model, but have little impact on the overall shape.
- **Optimize:** Optimization is used to get a refined model.

Step 6. Smoothness: Finally, we add a term given by equation 21 to the objective function, which enforces smoothness of the curves, and repeat the optimization process. This reduces kinks in the model that result from over-fitting noisy input data.

3 Optimization

In this section, we describe our optimization techniques for fitting a model to measured data. First, we define an objective function to assess the deviation between data and model, and describe a procedure for efficiently computing the gradient of this function. We then describe methods for refining local features of the model.

3.1 Error of Fit

To define a practical measure of fit between a model and the corresponding data, we begin with the notion of such a measure for two arbitrary surfaces. For any point $\mathbf{x} \in R^3$ and subset $S \subseteq R^3$, let

$$\chi(\mathbf{x}, S) = \inf_{\mathbf{y} \in S} || \mathbf{x} - \mathbf{y} ||, \tag{2}$$

where $||\cdot||$ is the Euclidean 2-norm, which is the distance from $\mathbf{x}$ to the surface S. If S_1 and S_2 are two integrable surfaces in R^3, we define a symmetric measure of closeness by

$$\phi(S_1, S_2) = \int_{S_2} \chi^2(\mathbf{x}, S_1)\, d\mathbf{x} \; + \int_{S_1} \chi^2(\mathbf{x}, S_2)\, d\mathbf{x}, \tag{3}$$

which is zero if and only if $S_1 = S_2$. This function imposes an equal penalty for either surface deviating from the other, or for either surface covering too little of the other. If S_1' and S_2' are discrete point sets with unit weight, this becomes

$$\phi(S_1', S_2') = \sum_{\mathbf{x} \in S_2'} \chi^2(\mathbf{x}, S_1') \; + \sum_{\mathbf{x} \in S_1'} \chi^2(\mathbf{x}, S_2'). \tag{4}$$

Moreover, if $S_1' \subset S_1$ and $S_2' \subset S_2$ are sets of discrete samples from the respective surfaces, then $\phi(S_1', S_2')$ can be used to approximate $\phi(S_1, S_2)$. To determine how well a generative model matches an actual object, we employ two point sets in exactly this manner, as equation 3 is typically impossible to evaluate exactly [1]. One point set is obtained from direct measurement, such as a range image, and the other by sampling the parameter space of a model. By optimizing with respect to this objective function, we in effect minimize the RMS deviation of the two surfaces:

$$D(S_1', S_2') = \sqrt{\frac{\phi(S_1', S_2')}{|S_1'| + |S_2'|}}, \tag{5}$$

where $|S|$ denotes the number of elements in the set S.

3.2 Computing the Gradient

Since a generative model is typically a nonlinear function of its constituent curves, we use a general optimization technique to estimate the model's shape parameters. For simplicity and ease of implementation, we use a conjugate gradient method [18] to minimize the functional $\phi(S_d', S_m')$ with respect to the model parameters. Let S_d' denote the fixed data, and S_m' the model samples, which depend on intrinsic shape parameters as well as extrinsic parameters controlling translation, scale, and pose. Specifically,

$$S_m' = S_m'(c_0, c_1, \ldots, c_k, \mathbf{t}, \mathbf{q}),$$

where $c_0, c_1, \ldots c_k$ are model parameters, such as global deformations and the y-components of curve control points, $\mathbf{t} \in R^3$ is the

[1] Simple analytic formulae for χ are seldom known, and numerical evaluation, though precise, is typically too slow for the inner loop of an optimization algorithm.

global translation, and $\mathbf{q}$ is a quaternion encoding global scale and rotation. Thus, optimization is guided by gradients of the form

$$\nabla\phi \;=\; \left[\frac{\partial\phi}{\partial c_0}, \frac{\partial\phi}{\partial c_1}, \ldots, \frac{\partial\phi}{\partial c_k}, \frac{\partial\phi}{\partial \mathbf{t}}, \frac{\partial\phi}{\partial \mathbf{q}}\right]. \tag{6}$$

To compute this gradient, first observe that

$$\frac{\partial\phi}{\partial c_j}(S'_d, S'_m) \;=\; \sum_{i=1}^{|S'_m|} \frac{\partial\phi}{\partial \mathbf{x}_i}\,\frac{\partial \mathbf{x}_i}{\partial c_j}, \tag{7}$$

where $\mathbf{x}_1, \mathbf{x}_2, \ldots$ are elements of S'_m. Because $\partial\phi/\partial\mathbf{x}_i$ and $\partial\mathbf{x}_i/\partial c_j$ are row and column vectors of dimension three, respectively, the summation is over inner products. We next express the partial derivatives of ϕ in equation 7 in terms of the *nearest neighbor function* $\eta_d : S'_m \rightarrow S'_d$, where $\eta_d(\mathbf{x})$ is the element of S'_d nearest to $\mathbf{x} \in S'_m$. The function $\eta_m : S'_d \rightarrow S'_m$ is defined analogously. Then, equation 4 becomes

$$\phi(S'_d, S'_m) = \sum_{\mathbf{x}\in S'_m} ||\,\mathbf{x} - \eta_d(\mathbf{x})\,||^2 + \sum_{\mathbf{y}\in S'_d} ||\,\mathbf{y} - \eta_m(\mathbf{y})\,||^2\,. \tag{8}$$

Since the nearest neighbor functions are piecewise constant, their derivatives are zero almost everywhere. Hence

$$\frac{1}{2}\,\frac{\partial\phi}{\partial\mathbf{x}} \;=\; [\mathbf{x} - \eta_d(\mathbf{x})]^T \;+\; \sum_{\mathbf{y}\in\eta_m^{-1}(\mathbf{x})} [\mathbf{x} - \mathbf{y}]^T\,, \tag{9}$$

for all $\mathbf{x} \in S'_m$, where the inverse relation $\eta_m^{-1}(\mathbf{x})$ denotes the set of points in S'_d whose nearest neighbor in S'_m is $\mathbf{x}$. The nearest-neighbor correspondences are efficiently computed using a kd-tree updated at each major iteration of the conjugate-gradient solver.

Suppose now that c is a control point of a curve Γ. Then, to compute $\partial\mathbf{x}/\partial c$, we must account for the composition of nonlinear transformations that may be applied to the curve as part of the current model. For example, suppose that the point $\mathbf{x}$ is obtained by evaluating the model at the parameter values u and v. Then

$$\mathbf{x}(u, v) = \mathbf{F}(u, v, \Gamma(u; c_1, \ldots, c_q)), \tag{10}$$

where, $\mathbf{F}(u, v, \cdot) : R \rightarrow R^3$ is the parametric mapping defined by previous levels in the model hierarchy and applied to curve Γ; we assume Γ to be a function of the parameter u as well as the control points $c_1, \ldots, c_q$. It follows that

$$\frac{\partial\mathbf{x}(u, v)}{\partial c} \;=\; \frac{\partial\mathbf{x}(u, v)}{\partial\Gamma(u)}\,\frac{\partial\Gamma(u)}{\partial c}, \tag{11}$$

where $\partial\mathbf{x}/\partial\Gamma(u)$ is the 3×1 Jacobian matrix of $\mathbf{F}$ at the parameter values u and v. The partial derivative on the right of equation 11 is easily evaluated, given that Γ is simply a spline curve. The partial $\partial\mathbf{x}/\partial\Gamma(u)$ can be evaluated symbolically by differentiating the current model. We have found that numerical approximation by a finite difference is equally effective, and may be simpler to compute. Thus, we can also use

$$\frac{\partial\mathbf{x}(u, v)}{\partial\Gamma(u)} \approx \frac{\mathbf{F}(u, v, \Gamma(u) + h) - \mathbf{F}(u, v, \Gamma(u) - h)}{2h}, \tag{12}$$

where h is a suitably small step. When c is a global parameter, controlling a bending deformation for example, $\partial\mathbf{x}/\partial c$ is obtained directly from the algebraic model specification using either symbolic or numeric differentiation, and the second factor on the right of equation 11 is no longer necessary.

The extrinsic parameters $\mathbf{t}$ and $\mathbf{q}$ map canonical model coordinates, in which the computations are initially performed, into the world-space coordinates of the measured data, in which the gradient $\nabla\phi$ is computed. More precisely,

$$\mathbf{x} \;=\; \mathbf{t} + \mathbf{M}(\mathbf{q})\mathbf{x}_0, \tag{13}$$

where the point $\mathbf{x}_0$ is in canonical coordinates, and M is a scale and rotation matrix parametrized by $\mathbf{q}$. All partials of $\mathbf{x}$ are also transformed by

$$\frac{\partial\mathbf{x}}{\partial\Gamma(u)} \;=\; \mathbf{M}(\mathbf{q})\frac{\partial\mathbf{x}_0}{\partial\Gamma(u)}. \tag{14}$$

Finally, to compute $\partial\phi/\partial\mathbf{t}$ and $\partial\phi/\partial\mathbf{q}$ in equation 6, we substitute partials of $\mathbf{x}$ with respect to $\mathbf{t}$ and $\mathbf{q}$ in place of $\partial\mathbf{x}/\partial c$ in equation 7, where

$$\frac{\partial\mathbf{x}}{\partial\mathbf{t}} = \mathbf{I} \quad\text{and}\quad \frac{\partial\mathbf{x}}{\partial\mathbf{q}} = \frac{d\mathbf{M}}{d\mathbf{q}}\,\mathbf{x}_0. \tag{15}$$

Here $\mathbf{I}$ is the 3×3 identity matrix, and the derivative of each element in the matrix $\mathbf{M}$ with respect to $\mathbf{q}$ is itself a quaternion. Equation 7 becomes a summation over row vectors or quaternions after the respective substitutions.

procedure ComputeGradient

Use new point correspondences.
$\mathcal{P} \leftarrow \emptyset$
for all $\mathbf{x} \in S'_m$ and $\mathbf{y} \in S'_d$ **do**
  add $(\mathbf{x}, \eta_d(\mathbf{x}))$ to $\mathcal{P}$
  add $(\eta_m(\mathbf{y}), \mathbf{y})$ to $\mathcal{P}$
endfor
Compute E, extrinsic & global partials.
$\mathbf{g} \leftarrow \mathbf{0}$; $E \leftarrow \mathbf{0}$
for all $(\mathbf{x}, \mathbf{y}) \in \mathcal{P}$ **do**
  $(u, v) \leftarrow$ parameters of $\mathbf{x}$
  $\mathbf{w} \leftarrow 2(\mathbf{x} - \mathbf{y})^T$ — *Eq. 9*
  $p \leftarrow \mathbf{w}\,[\partial\mathbf{x}/\partial c_0]$ — *Eq. 7*
  $\mathbf{g} \leftarrow \mathbf{g} + [p\;\; \mathbf{0}\;\; \mathbf{w}\;\; \mathbf{w}\dot{\mathbf{M}}\mathbf{x}_0]$ — *Eq. 7, 15*
  $E_u \leftarrow E_u + \mathbf{w}\,[\partial\mathbf{x}/\partial\Gamma(u)]$ — *Eq. 16*
endfor
Fill in partials wrt control points.
for all u samples **do**
  for each c_j affecting $\Gamma(u)$ **do**
    $\mathbf{g}_j \leftarrow \mathbf{g}_j + E_u\,[\partial\Gamma(u)/\partial c_j]$ — *Eq. 17*
  endfor
endfor
return $\mathbf{g}$

Figure 4: *Computing the gradient $\mathbf{g}$ of the objective function ϕ. We assume a single curve Γ with control points $c_1, c_2, \ldots, c_k$, parametrized with respect to u. The parameter c_0 is assumed to be a global shape parameter. The model is sampled at discrete parameter values in u and v.*

Efficiency: From equations 6, 7 and 9, it appears that the gradient computation requires $O(nk)$ time per curve, where n is the number of sample points, and k is the number of shape parameters controlling a curve. By exploiting sparsity, we can reduce the time to essentially $O(n)$ since each sample point of the model is affected by only a small number of shape parameters. In particular, when $c_1, c_2, \ldots$ are control points, their effects are very localized, so each $\partial\Gamma(u)/\partial c$ in equation 11 is nonzero only for a small number of shape parameters c.

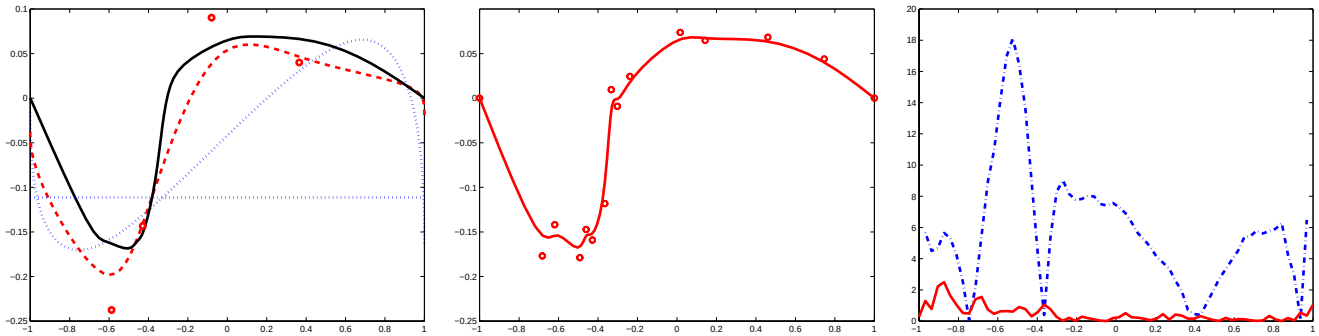

Figure 5: *The process of curve refinement.* **Left:** *Depth curve of spoon model as control points are added in recognition phase. The straight blue dotted line is the global value at root node, and the blue dotted curve is the initial coarse version. The red dashed curve results after adding four control points shown by circles, which greatly improves the approximation. For comparison, the final curve is shown by a black solid curve.* **Middle:** *Further refinement of the depth curve. Constraints force the curve to be* 0 *at the end-points and additional control points increase the accuracy of the curve, but also introduce undesirable kinks. These are essentially eliminated in the smoothed version shown with a solid line on the left.* **Right:** *The blue dash-dot curve indicates the initial error heuristic before any interior control points are added and the red solid line indicates the error after refinement and optimization. The optimized version has a much lower and flatter error according to the heuristic.*

The pseudo-code in figure 4 summarizes the gradient computation. Here, $\mathbf{g}$ denotes the gradient, which is a row vector, $\mathbf{w}$ is a contribution to the row vector $\partial\phi/\partial\mathbf{x}$, as defined in equation 9, and

$$E_u = \frac{\partial\phi}{\partial\Gamma(u)} = \sum_v \frac{\partial\phi}{\partial\mathbf{x}(u,v)} \frac{\partial\mathbf{x}(u,v)}{\partial\Gamma(u)} \tag{16}$$

is a scalar associated with the parameter value u. The summation in equation 16 is over the discrete samples of v. Finally, we have

$$\frac{\partial\phi}{\partial c_j} = \sum_u E_u \frac{\partial\Gamma(u)}{\partial c_j}, \tag{17}$$

where the summation is over discrete samples of the u parameter. The computation is broken down in this way for efficiency. Note that the first element of the gradient vector $\mathbf{g}$ is a partial with respect to global parameter c_0, while the next k elements of the gradient are partials with respect to curve control points. The construct $[p \;\; \mathbf{0} \;\; \mathbf{w} \;\; \mathbf{w}\dot{\mathbf{M}}\mathbf{x}_0]$ in line 11 of the pseudocode denotes the concatenation of the elements into a single row vector, where the zero vector $\mathbf{0}$ has k entries, and $\dot{\mathbf{M}}$ is the derivative of the scale and rotation matrix with respect to the quaternion $\mathbf{q}$.

In this algorithm, we recompute the point correspondences using a kd-tree[2]. We then loop through all the point pairs computing the contributions to E_u, the partial with respect to the global parameter c_0, and the partials with respect to the extrinsic parameters $\mathbf{t}$ and $\mathbf{q}$. Rather than explicitly computing the set $\eta_m^{-1}(\mathbf{x})$, as shown in equation 9, we simply accumulate the contribution of each data point in a single pass through the point set.

Discussion: The objective function ϕ described above is fairly crude as it depends on the distribution of sample points, and the nearest neighbor function may fail to make the most meaningful correspondences. Further, we do not interpolate between sample points for greater accuracy, nor do we use connectivity information available in the model. While physically-based heuristics such as momentum and inertia can improve the situation somewhat [24], our simple objective function suffices to guide the traversal of the model hierarchy for the examples described in this paper.

[2] Actually, this is done earlier as a result of evaluating the objective function.

3.3 Curve Refinement

Each curve is adaptively refined during optimization. Refinement in this context differs from previous curve-fitting approaches in several ways. First, generative models are composed of spline curves with local control, whereas optimization techniques typically used in computer vision entail global shape parameters. Secondly, the curves of a generative model describe characteristics of a 3-D surface rather than approximating sets of 2-D points [17]. Our approach to curve refinement most closely resembles the techniques for optimizing trajectories used in animation [4, 20].

When a curve is added during the recognition phase in step 3 of figure 3, it is initialized to a constant with a global parameter inherited from the parent node in the model hierarchy. Multiple control points are added at the ends to construct a valid spline curve. After the first optimization pass, four or five additional interior control points are added and optimization is repeated. See figure 5. During recognition, a coarse approximation of the curve suffices, as it is used only to choose a suitable model in the hierarchy.

Our approach to curve refinement is to add control points gradually, and only where needed. We use E_u, as defined in equation 16, as an error term for a curve Γ, and insert new control-points that equidistribute the error; that is, we select the points so segments between them have equal net error.

Since $\nabla\phi = 0$ at a minimum, $\partial\phi/\partial c = 0$ for all c. As the number of control-points increases, E_u approaches $\partial\phi/\partial c$ for some control point c, and is therefore 0 when the curves are represented exactly. A large $|E_u|$ indicates that the approximating curve needs refinement. We can also use a variational argument. Since ϕ at a minimum should be 0 to first order for variations in the curve Γ, $|E_u|$ is equivalent to the Euler-Lagrange [20, 28] error, which should be 0 when the curve is represented precisely. The error ϕ by itself does not necessarily indicate where curves need refining, as the model may be fundamentally incapable of representing the data; this is especially so for simpler models in the hierarchy.

Producing the Final Curves: In step 5 of the algorithm shown in figure 3, we refine the curves further and add constraints. First, we compute the total error normalized by the size of the data set separately for each curve:

$$E = \frac{\sum_u |E_u|}{|S'_d \cup S'_m|} \tag{18}$$

Based on this total error, we calculate the number of control-points we wish to add to each curve using the heuristic

$$N_c = \frac{E}{\epsilon}, \tag{19}$$

which adds one control point for each ϵ of error. We have found that a value of $\epsilon = 10^{-3}$ is suitable for objects of unit dimensions. The control points are again added to equidistribute the error. Then, user-specified constraints such as curve end conditions are added. Refer to the middle of figure 5 for an example. The objective function is minimized again to yield a high-accuracy solution satisfying the constraints.

In the final stage, step 6 in figure 3, we ensure that noise in the data does not lead to extraneous kinks in the model. We do this by introducing a penalty based on the integrated curvature β given by

$$\beta = \int_u |\kappa(u)| \, du, \tag{20}$$

where $\kappa(u)$ is the curvature of Γ at the parameter value u. We estimate β using numerical quadrature and approximate the derivatives of β with respect to the curve control points using finite differences. These values are summed over all curves in the model. The objective function is then augmented with an extra term

$$\hat{\phi} = \phi + a\beta, \tag{21}$$

where a is a positive weight chosen so that the smoothness term and the original objective function are of approximately the same magnitude. Thus,

$$a = K\frac{\phi_0}{\beta_0}, \tag{22}$$

where the subscript 0 denotes the value after step 5 in the algorithm framework, but before optimizing with respect to the augmented objective function. Here, K is a constant that controls the relative importance of the two terms, which was fixed at 10 in our tests.

4 Model Hierarchies Used

This section briefly reviews the model hierarchies used in our experiments. The hierarchy for the spoon is patterned after the model given by Snyder [21, p. 83]. The generative modeling equation is

$$\textsc{spoon}(u,v) \;=\; \begin{bmatrix} \mathrm{S}_x(v) \\ \textsc{arc}_x(\mathrm{S}_y(v), \mathrm{D}_y(v), u) \\ \mathrm{B}_y(v) + \textsc{arc}_y(\mathrm{S}_y(v), \mathrm{D}_y(v), u) \end{bmatrix},$$

where S is the width or shape curve, D is the depth curve, and B is the bend, parametrized so $\mathrm{S}_x = \mathrm{D}_x = \mathrm{B}_x$. $\textsc{arc}(a,b,u)$ defines a parametric arc passing through $(-a,0)$, $(0,b)$, and $(a,0)$ in the xy-pane. The above equation is obtained at the deepest level of the hierarchy, regardless of the path, since all the operators commute. In the root model, S and D are global parameters and B is 0. As more complex models are reached, these constants are replaced by curves. Curves are constrained for this model so that S and D are 0 at the end-points, where S is also perpendicular to the x-axis; since the model is symmetric about the x-axis, this last constraint avoids introducing a kink. To complete the representation, we also require a thickness. Since the thickness is typically small and difficult to discern from range data, we use a constant value derived from the projection of the acquired data in the yz-plane.

For a ladle-like shape, the arcs are translated by the bend only after first rotating them about the y-axis by an angle equal to that of the bend curve from the horizontal. This forces the arcs to remain perpendicular to the bend curve. A shape suitable for a cup handle is obtained by making the circular cross-section ARC rectangular. [3]

We also used a *rotating generalized cylinder*—the "banana" model defined by Snyder [21, p. 69]—to represent many different objects. This model rotates a cross-section while scaling and translating it, and generalizes common primitives in computer vision known as profile products or simple homogeneous generalized cylinders. The parametric equation is

$$\textsc{banana}(u,v) \;=\; \textsc{yrot}(R(v)) \begin{bmatrix} \mathrm{S}(v)\,\mathrm{C}_x(u) \\ \mathrm{S}(v)\,\mathrm{C}_y(u) \\ \mathrm{Z}(v) \end{bmatrix}$$

where $\textsc{yrot}(\theta)$ is a rotation of θ about the y-axis, R is the parametric rotation angle, S the scale, and C the 2D cross-section. The root model is a right circular cylinder with unit radius and no rotation. The curves R, S, and C are added at more complex levels in the hierarchy. This class of models can also be used to represent surfaces of revolution, such as the bowl example that is shown.

As the previous model hierarchy demonstrates, our approach subsumes some common primitives used in computer vision such as simple homogeneous generalized cylinders. We have also recovered globally deformed superquadrics with our approach.

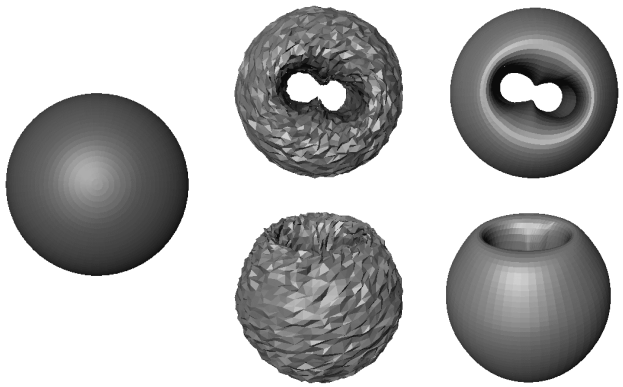

Figure 6: *Fitting of a blended model [7] to synthetic randomly perturbed samples of a sphere/torus blend with variable cross-section. This example shows how blended models fit directly into our framework, with the blend curve being treated like any other generative curve. It also shows that non-spherical and variable topology can be handled. The model eliminates most of the noise in the data without introducing significant errors. The leftmost image is the initial model, which is simply a sphere. To the right are two views of the rough input data and the smooth final model.*

DeCarlo and Metaxas [7] present a method for using blended deformable models. Models of variable topology can be created using their method. Their ideas fit directly into our framework since the *blend curve* is just another curve in the generative model. Since our objective function does not directly consider topology at all, no effort is needed to incorporate variable topology. We note that DeCarlo and Metaxas require certain constraints on the blending function to obtain a consistent model, which are naturally incorporated into the curve constraints phase of our algorithm. The general formula for a blended model [7] is

$$\textsc{blend}(u,v) = \Gamma(u)\mathrm{B}_1(u,v) + (1-\Gamma(u))\mathrm{B}_2(u,v),$$

[3] The actual handle is not exactly rectangular, leading to the overly *squared* results. The data is also too sparse to reliably estimate the cross-section from scratch.

where Γ is the blending curve that can be treated just as any other curve in the generative model. Here B_1 and B_2 are simply constituents of the generative model, which can be any shape at all, including as a special case, the superquadrics and supertoroids used in [7]. Figure 6 shows an example of recovering blended models with our approach.

Although this paper deals primarily with shapes that can be represented by a single generative model, we have carried out some experiments on simple articulated objects. The watering-can in figure 1 was modeled by first manually segmenting it into body, handle and spout, and then fitting a rotating generalized cylinder to each part. Minor imperfections are primarily due to errors in segmentation. We may also define a single composite generative model by combining existing parts, each controlled by separate extrinsic and intrinsic parameters. The cup in figure 1 is an example. If the extrinsic transformations of the individual parts are left free—and not constrained to ensure correct connectivity—the user may need to make some minor adjustments at the end to ensure the parts connect properly. While the user-supplied initial guess function can still be crude, it needs to be more accurate for an articulated object since a data point can otherwise be incorrectly associated with the wrong model part.

5 Results

Parameters: For our tests, the complexity constant Q defined below equation 1 was set to a deviation of .01 after scaling the models to have a major axis range from -1 to $+1$. Thus, Q corresponds to a percentage error of approximately $.5\%$. The results demonstrate that the algorithm is not very sensitive to the precise value of Q. We report the percentage error using a 64×64 tessellation of the model. For illustrative purposes only, both range images and the model were meshed to create the final images for display and colors are artificial. Snyder [21] describes alternate methods to image generative models without mesh creation, but at the cost of loss of interactivity.

Data: The range data used in our experiments was obtained from a variety of sources. Of the objects in figure 7, the spoon and bowl data are single range images obtained using structured light [25], while 6 cylindrical scans are aligned for the cup data. The ladle is a single range image obtained using the method of Bouguet and Perona [2]. Data for the banana and candle-holder were obtained using a mechanical probe, and the watering-can data is a cylindrical scan obtained from a laser range-scanner. For the data obtained from the probe, connectivity information was not available, so the meshes for the figures were obtained using the approach of Hoppe et al. [12]. Our algorithm operated directly on the range data, and the results demonstrate the benefit of recovering a model as opposed to a mesh, especially in cases of noisy and incomplete data.

Recognition Trees: Figure 8 shows recognition trees for two objects—a spoon acquired using structured light, and synthetic data for a banana-like object. Data on errors are given in figure 9. The root models are trivial, and the user-supplied initial guess functions need not specify accurate initial estimates; nonetheless the algorithm is able to *bootstrap* itself to produce an accurate final model. Paths that are not ultimately selected can sometimes produce strange and interesting results as a curve is trying to adjust to match data that it is incapable of matching. This effect will be especially noted in the tree for the banana.

Accuracy and Robustness: A visual comparison indicates that the method produces a good match to the data, even when the data is noisy and/or incomplete. As a confirmation of the accuracy of the method, on the synthetic data of the banana shown in figure 8, the technique produces results accurate to within $.4\%$. As shown in the left of figure 10, even if the input hierarchy is unable to adequately represent the object, the algorithm does the best it can, producing a simple model that conveys some of the dominant aspects of the shapes.

Finally, we demonstrate the robustness of the technique by running it on a sparse sampling of the spoon data; after removing 90% of the spoon data, a visually appealing reasonably accurate model is still obtained as shown in figure 9.

Compactness: Our models typically had fewer than a hundred parameters, primarily curve control points. This is at least two orders of magnitude smaller than the corresponding meshes.

Editing: An example of editing a recovered spoon model into a ladle-like shape is shown on the right of figure 10, demonstrating how easily new models can be constructed by simple and intuitive curve editing from shapes already recovered.

Computation Time: The entire algorithm took between 20 and 30 minutes on a 150 MHz SGI MIPS R4400, depending on model complexity and the size of the data set. Each iteration of the conjugate gradient took 1-2 seconds, with each optimization pass taking about 50 iterations. The process was entirely automatic; no manual intervention was required. The total number of points (range data and tessellated model) was typically about 15000.

6 Conclusions and Future Work

We have presented a new method for creating concise generative models from incomplete range data, given a user-supplied model hierarchy. Advantages of our approach are simplicity, robustness to noise, and creation of an intuitive compact model. We extend traditional computer vision algorithms for recovery of specific shapes in that curves of a user-supplied generative model are estimated; the user can supply a model of their choice and immediately obtain an automatic recovery algorithm.

Our work currently has many limitations. The fits obtained are not perfect, especially when the model inadequately describes the real object. Even for the synthetic banana-like data used in figure 8, there is some residual error. Our method also does not preserve local detail, and there may be artifacts from under- or over-smoothing, such as the squaring near the ends of the spoon model. Further, the algorithm requires the user to specify an appropriate model hierarchy, and currently does not allow different hierarchies to be combined. If the wrong hierarchy is input, a simple model that mimics the original to the extent possible will be output as shown on the left of figure 10, but those results may not always be useful. Also, while the models shown are complex compared to single parametric models previously used in computer vision, they are still fairly simple for graphics as we do not provide automatic segmentation for recovery of complex articulated objects.

Solving the above problems defines some important directions for future work. Improvements could also be made in using more complex objective functions and minimization algorithms, more flexible tradeoffs between accuracy and simplicity, and more exhaustive non-greedy methods for traversing the input hierarchy. Finally, the model hierarchy used could be learned from examples or created automatically from a model database.

While many challenges remain, we believe that algorithms for recovering high-level models are an important direction of research for both computer vision and computer graphics.

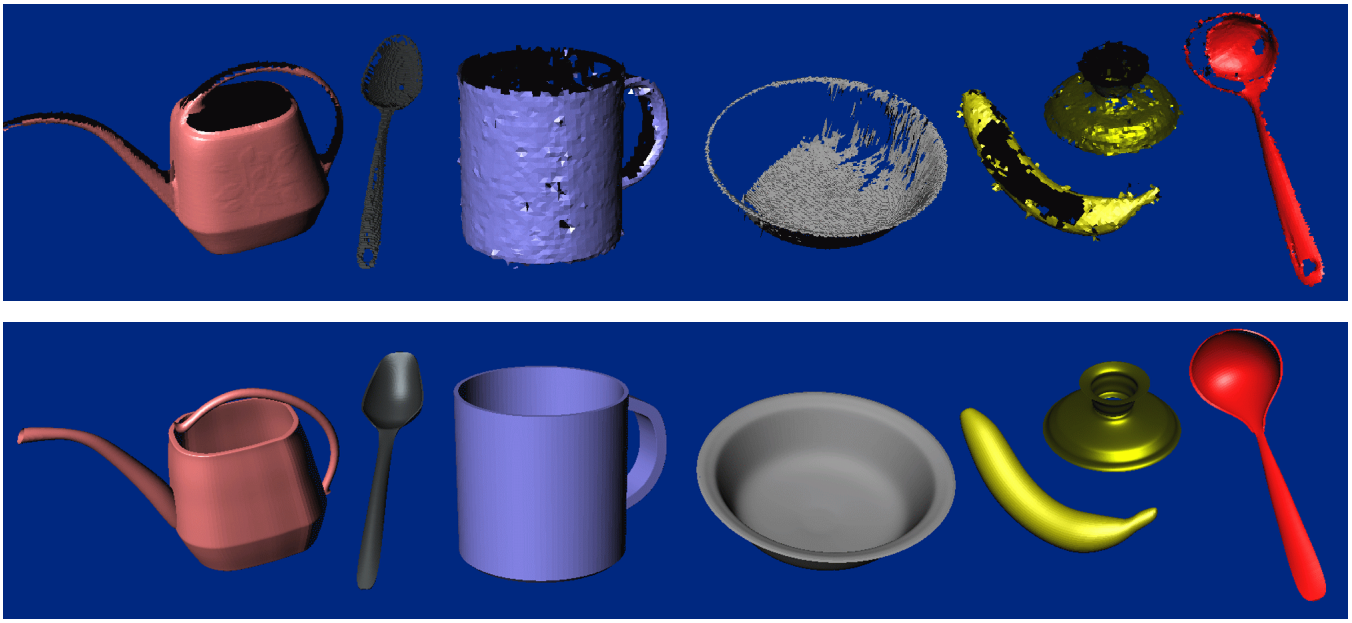

Figure 7: Data (above) and models (below) for the objects in the scene of figure 1. The models are robust to noise and incomplete data, and are a smooth compact representation.

Acknowledgements: Special thanks to Jean-Yves Bouguet for reviewing early drafts, and for help with data acquisition. Preliminary discussions with Al Barr were of immense help. We are also grateful to the anonymous Siggraph reviewers (especially #2) and committee for their helpful comments, and to members of the graphics groups at Caltech and Stanford, for their support.

This work was supported by the NSF Science and Technology Center for Computer Graphics and Scientific Visualization (ASC-8920219), an Army Research Office Young Investigator award (DAAH04-96-100077), the Alfred P. Sloan Foundation, and a Reed-Hodgson Stanford Graduate Fellowship. All opinions, findings, conclusions or recommendations expressed here are those of the authors only and do not necessarily reflect the views of the sponsoring agencies and individuals.

References

[1] T. O. Binford. Visual perception by computer. In *Proceedings of the IEEE Conference on Systems Science and Cybernetics*, 1971.

[2] Jean-Yves Bouguet and Pietro Perona. 3D photography on your desk. In *ICCV 98 proceedings*, pages 43–50, 1998.

[3] R. A. Brooks. Model-based three-dimensional interpretations of two-dimensional images. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 5(2):140–150, 1983.

[4] M.F. Cohen. Interactive spacetime control for animation. In *SIGGRAPH 92 proceedings*, pages 293–302, 1992.

[5] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH 96 proceedings*, pages 303–312, 1996.

[6] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *SIGGRAPH 96 proceedings*, pages 11–20, 1996.

[7] D. DeCarlo and D. Metaxas. Blended deformable models. *PAMI*, 18(4):443–448, Apr 1996.

[8] D. DeCarlo and D. Metaxas. Shape evolution with structural and topological changes using blending. *PAMI*, 20(11):1186–1205, Nov 1998.

[9] S.J. Dickinson, D. Metaxas, and A. Pentland. The role of model-based segmentation in the recovery of volume parts from range data. *PAMI*, 19(3):259–267, Mar 1997.

[10] Matthias Eck and Hugues Hoppe. Automatic reconstruction of B-Spline surfaces of arbitrary topological type. In *SIGGRAPH 96 proceedings*, pages 325–334, 1996.

[11] F. Ferrie, J. Lagarde, and P. Whaite. Darboux frames, snakes, and super-quadrics: Geometry from the bottom up. *PAMI*, 15(8):771–784, 1993.

[12] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Surface reconstruction from unorganized points. In *SIGGRAPH 92 proceedings*, pages 71–78, 1992.

[13] Hugues Hoppe, Tony DeRose, Tom Duchamp, John McDonald, and Werner Stuetzle. Mesh optimization. In *SIGGRAPH 93 proceedings*, pages 19–26, 1993.

[14] Venkat Krishnamurthy and Marc Levoy. Fitting smooth surfaces to dense polygon meshes. In *SIGGRAPH 96 proceedings*, pages 313–324, 1996.

[15] R. Nevatia. *Structured Descriptions of Complex Curved Objects for Recognition and Visual Memory*. PhD thesis, Stanford, 1974.

[16] A. Pentland. Toward an ideal 3-D CAD system. In *Proc. SPIE Conf. Machine Vision Man-Machine Interface*, 1987. San Diego, CA.

[17] Michael Plass and Maureen Stone. Curve-fitting with piecewise parametric cubics. In *SIGGRAPH 83 proceedings*, pages 229–239, 1983.

[18] W. Press, S. Teukolsky, W. Vetterling, and P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing (2nd ed.)*. Cambridge University Press, 1992.

[19] N. Sriranga Raja and A. K. Jain. Obtaining generic parts from range images using a multi-view representation. *Computer Vision, Graphics, and Image Processing. Image Understanding*, 60(1):44–64, July 1994.

[20] R. Ramamoorthi and A. H. Barr. Fast construction of accurate quaternion splines. In *SIGGRAPH 97 proceedings*, pages 287–292, 1997.

[21] J. Snyder. *Generative Modeling for Computer Graphics and CAD*. Academic Press, 1992.

[22] John M. Snyder and James T. Kajiya. Generative modeling: A symbolic system for geometric modeling. In *SIGGRAPH 92 proceedings*, pages 369–378, 1992.

[23] F. Solina and R. Bajcsy. Recovery of Parametric Models from Range Images: The Case for Superquadrics with Global Deformations. *PAMI*, 12(2):131–147, February 1990.

[24] D. Terzopoulos and D. Metaxas. Dynamic 3D models with local and global deformations: Deformable superquadrics. *PAMI*, 13(7):703–714, July 1991.

[25] Marjan Trobina. Error model of a coded-light range sensor. Technical Report BIWI-TR-164, ETH, Zurich, 1995.

[26] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *SIGGRAPH 94 proceedings*, pages 311–318, 1994.

[27] K. Wu and M. D. Levine. Recovering parametric geons from multiview range data. In *CVPR*, pages 159–166, June 1994.

[28] D. Zwillinger. *Handbook of Differential Equations*. Academic Press, 1989.

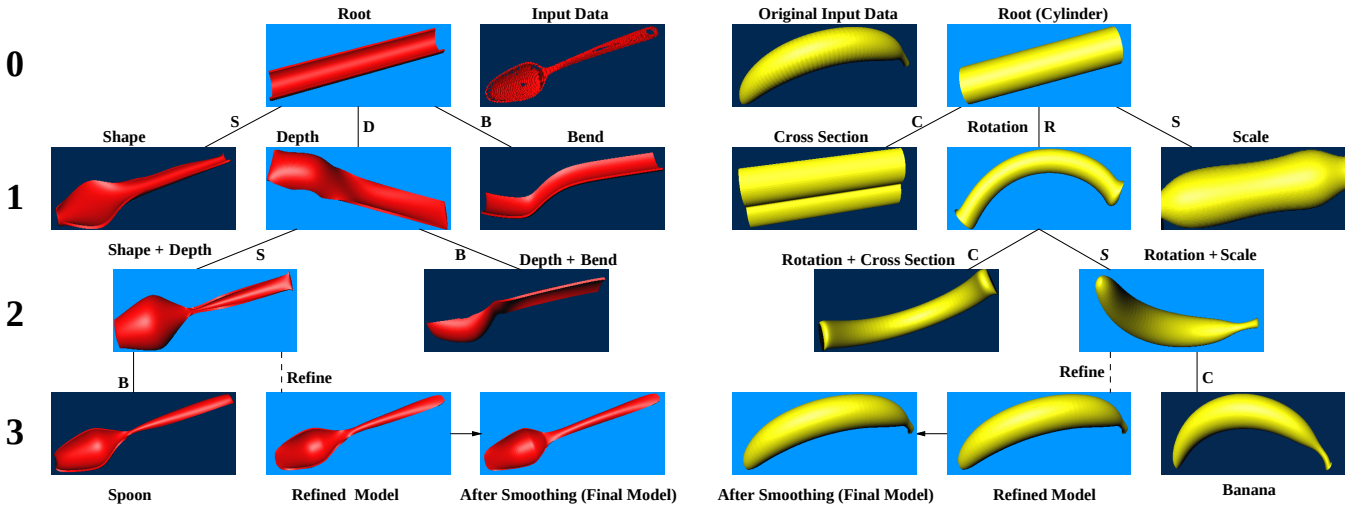

Figure 8: Recognition trees for the spoon (left) and banana (right). Highlighted nodes (lighter background) indicate the best guess at some level, and only nodes reachable from a highlighted node are generated. The algorithm is able to *bootstrap* itself, starting from very crude initial conditions, improving at each stage and finishing with an accurate model.

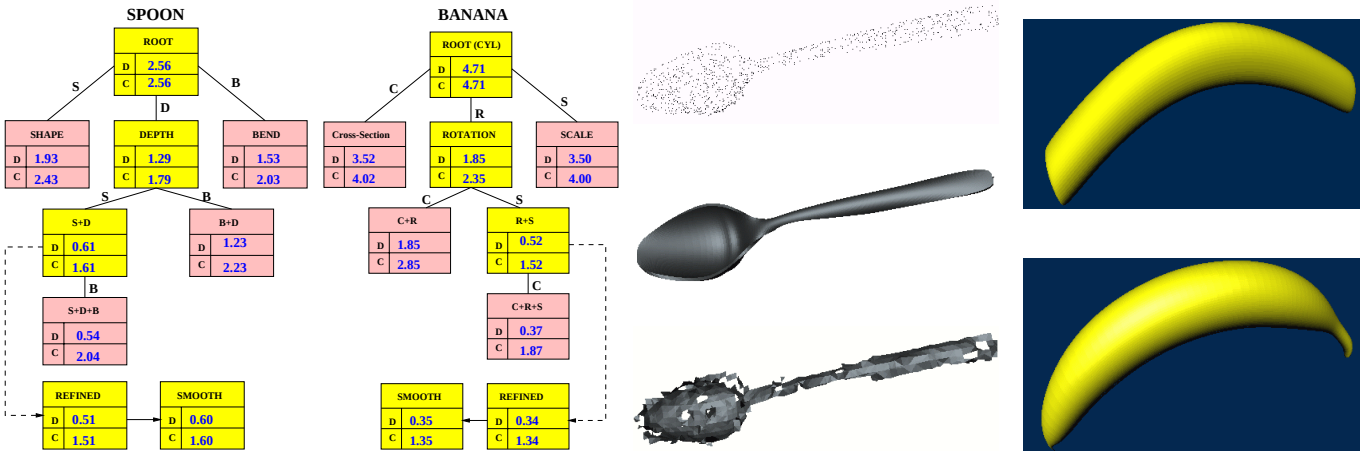

Figure 9: **Left:** Percentage deviation errors (D) and total costs (C) for the spoon (left) and banana (right). **Middle:** Fitting of model to very sparse data. On top is a pointcloud with fewer than 900 points. The middle shows the recovered model while the bottom is a mesh obtained from Hoppe's [12] algorithm on the same data. A comparison indicates the robustness of our approach. **Right:** The top shows a superquadric fit to the banana data while the bottom shows our model, indicating the benefit of generative models.

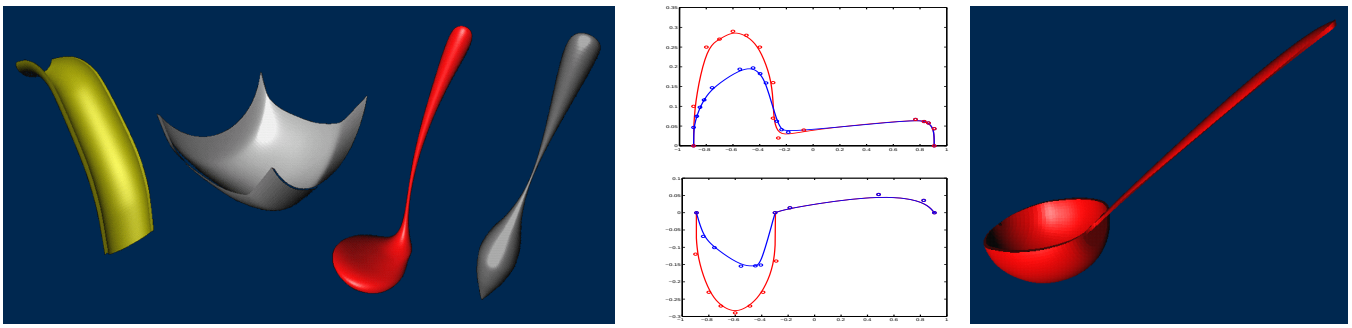

Figure 10: **Left:** Models recovered using a model hierarchy that was a poor match for the actual data since it did not possess the appropriate degrees of freedom to adequately represent the object. The two objects on the left are the banana and bowl recovered using the spoon hierarchy, while the two objects on the right are the ladle and spoon recovered using the rotating generalized cylinder hierarchy. The algorithm did the best that it could, and managed to convey some of the dominant aspects of the shapes. **Middle:** Editing a recovered spoon model into a ladle. Only a few control points need to be moved to get a radically different shape. *Middle Top:* Edited shape curve before (blue) and after (red) editing. Control points are shown as circles. *Middle Bottom:* A similar plot for the depth curve. **Right:** A view of the edited model.