Sketch-based Modeling of Parameterized Objects

by

Chen Yang

M.E., Tsinghua University, P.R.China, 2003

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Computer Science)

The University of British Columbia

April 2006

Abstract

This thesis presents a modeling system that constructs 3D models of particular object classes, such as cups, airplanes, and fish, from 2D sketches. The core of the system is a sketch recognition algorithm that seeks to match the points and curves of a set of given 2D templates to the sketch. The matching process employs an optimization metric that is based on curve feature vectors. The search space of possible correspondences is restricted by encoding knowledge about relative part locations into the 2D template. Once a best-fit template is found, a 3D object is constructed using a series of measurements that are extracted from the labelled 2D sketch. The sketch-recognition and modeling algorithms are applied to sketches of cups and mugs, airplanes, and fish. The system allows non-experts to use drawings to quickly create 3D models of specific object classes.

Table of Contents

List of Tables

List of Figures

Acknowledgements

First, I would like to thank my supervisor, Michiel van de Panne, for his guidance and support in this thesis project. Michiel has been super supportive of my work and his novel ideas always inspired me greatly. I would not be here without his encouragement and inspiration. Many thanks also to Tamara Munzner for the time she has devoted to reading my thesis and the valuable feedbacks she has given.

Second, I would like to thank Dana Sharon for her help on papers and other conference submissions. Also I would like to thank many friends that helped and supported me in the project. Here's a randomized list of them: James Slack; Vlad Kraevoy; Abhijeet Ghosh; Yongying Zhu; Kang Yin; Ciarán Llachlan Leavitt; Dave Burke; Dan Julius; Qiang Kong;Fred Kimberley; Matt Trentacoste; Dan Archambault; Jason Harrison; Dan Xiao; Peng Zhao; Long Li.

Finally, but of course most importantly, I must give the biggest shout-out to my girlfriend, Dan Liu. Her never-ending love enabled me to complete this work.

CHEN YANG

The University of British Columbia

April 2006

This is for my love

Chapter 1

Introduction

1.1 Motivation

Consumers now have the ability to render 3D content on almost every computer and a growing number of portable devices. However, the capability for non-experts to create 3D content has not kept pace. Even those with in-depth knowledge of computer graphics will often go to considerable lengths to find existing 3D models before resorting to building a 3D model of their own. This can be seen as an interface problem — if we were to imagine describing the shape of this year's SIGGRAPH conference mug to a colleague, we could imagine using hand gestures or a hand-drawn sketch to communicate the object shape, to a first approximation. However, developing 3D modeling systems to support this type of input is a daunting task. For example, inferring the 3D shape of objects from line drawings is a well-studied and largely unsolved computer vision problem. Sketch-based modeling holds the promise of making 3D modeling accessible to a significantly wider audience than current modeling tools.

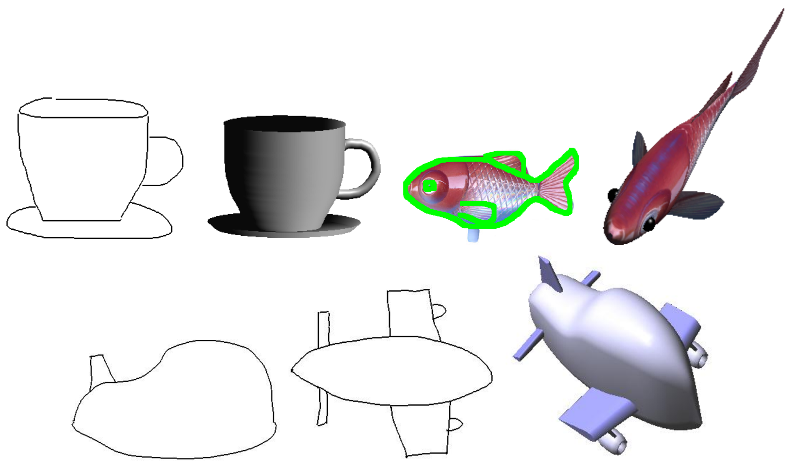


Figure 1.1: Example input sketches and output 3D models. All the objects are drawn from canonical views. The fish is drawn by tracing over the photograph which is then lifted as texture. The airplane uses both the side view and the top view to construct the 3D model.

1.2 A Simple Example

The system we present builds 3D models from hand-drawn sketches. To make the problem tractable, we make a number of assumptions. We equip the system with a priori knowledge about particular classes of objects that we expect to be drawn using the system. This knowledge is encoded in terms of a flexible 2D template that can be matched to the sketch an a procedure for building the 3D model from the 2D template. We also require that the user draw in a fashion that is not overly sloppy, and require that the user have familiarity with the general structure of the underlying template.

Our system does not solve the general problem of building 3D models from arbitrary hand-drawn sketches. Significantly, however, our system allows fast and flexible sketch-based modeling of particular classes of 3D objects with minimal training. These particular classes include cups, airplanes, and fish. We choose them because of their coverage of design space. A 3D cup can be determined by one sketch from the side view while an airplane usually takes two sketches, one from side view and one from top view, to make the model more accurate. A 3D model of fish is similar to that of an airplane but it only uses the side view in most cases. This is because most of the fish have their fins attached to their bodies with almost the same angles. Thus those angles can be set to default to help save the efforts from users. A 3D model of fish can be approximated from the side view. Several examples of the input and output of our system are shown in Figure 1.1.

1.3 Contributions

Our specific contributions are twofold: (1) a sketch-recognition algorithm that is tailored to the needs of sketch-based 3D model construction, and (2) a proof-of-concept system that demonstrates the potential of a new class of sketch-based modeling tools.

Our system assumes that it is worthwhile designing highly parameterized 3D-objects for specific applications. This would allow for user-driven content creation for games, such as a flight simulator game where the game lets the user sketch their own particular airplane design. Automotive design is another potential application that falls into this category. With a sufficient number of object classes added to the system, it opens the door to more general ’draw-it-as-you-see-it’ modeling.

1.4 Publications

This thesis is based in part on a sketch [38] and a paper [39], jointly authored with Michiel van de Panne and Dana Sharon. The work described here was led by Chen Yang, with the exception of generating 3D parameterized planes from labelled sketches in Section 6.3.2 which was done by Dana Sharon.

1.5 Organization

The remainder of this thesis is organized as follows. Chapter 2 discusses related work in 3D-modeling, sketch-recognition, and computer vision. An overview of the various components of our system is given in Chapter 3. Chapter 4 describes several preprocessing techniques which have been applied. Chapter 5 describes the construction of the 2D templates and details the sketch recognition algorithm. Chapter 6 presents how to construct 3D models from 2D recognized sketches. Chapter 7 presents results obtained with the system and discusses several common questions that arise with respect to our approach. Finally, Chapter 8 presents conclusions and future work.

Chapter 2

Related Work

The problem we are trying to solve, sketching a shape, can be deemed as inverse NPR (non-photorealistic rending) [25]. The human visual system uses silhouettes as the first index into its memory of shapes, making everyday objects recognizable without color, shading or texture, but solely by their contours. A few strokes which describe the silhouette can suffice to sketch the main features of a shape. This is why people still prefer using pen and paper to convey the shape. A sketch-based modeling system for 3D shapes should use the very same sketches from the users as input. Our sketch-based modeling system use sketches as input, and produce models which use the sketch curves as feature lines.

There exists a large body of literature in the broad areas related to sketch-based modeling. Sketch-based modeling can be decomposed into sketch recognition and 3D modeling.

2.1 Diagrammatic Sketch Recognition

A first research thread looks at the recognition of diagrammatic sketches such as hand-drawn circuit diagrams [1, 12], flowchart drawings [13], math notation [18], drawings with engineering symbols [17], military planning drawings and other types of diagrams with symbols and connectors. A variety of algorithms are exploited within this domain, often including a mix of top-down and bottom-up procedures that are informed by some domain knowledge. Statistical information can also be based on the observed stroke order [33] or the interpretation assigned to neighboring strokes [29]. Forms of graph isomorphism have been exploited [24], as have dynamic bayesian networks [1]. The algorithms for this class of sketch recognition problem and their comparative evaluation [26] still leave many unresolved issues in terms of building robust-yet-flexible systems. Like most of the previous work, the sketch recognition component of our problem can also be thought of as being "diagrammatic". However, our 2D templates have features that are specifically tailored to our problem domain, such as the notion of part hierarchies, optional parts, and one-of-N part selection.

2.2 Handwriting Recognition

Handwriting recognition is the task of transforming language represented in its spatial form of graphical marks into its symbolic representation [28]. The input of handwriting recognition is either from scanning the writing on paper (which is known as offline) or from writing on a tablet (which is known as online). We consider handwriting recognition as a distantly-related problem. First, handwriting has a set of icons, which are known as characters or letters, that have certain basic shapes. Also, there are rules for combining letters to represent shapes of higher level linguistic units. For example, there are rules for combining the shapes of individual letters so as to form cursively written words in the Latin alphabet.

2.3 2D Shape Matching

Sketch recognition can also be formulated as a 2D *shape matching* problem, where parts of a sketch can be identified and labelled by finding 2D templates that can match parts in the sketch while allowing for some class of deformations. A survey of shape matching can be found in [21] while examples of recent shape matching work can be found in [3, 5, 36, 9, 32]. Much of this work assumes a single template that can undergo arbitrary scaling and rotation, as well as small non-affine deformations. We wish our templates to be more flexible in terms of handling large localized changes of scale, such as a small plane with large wings, and sub-parts that may or may not exist. The performance of shape contexts [3] in such situations is unclear. Shape contexts also require first point sampling all the strokes, which throws away useful information because a good set of segmented curves is readily obtained from a hand-drawn sketch. Thus, we choose to work with curves as a base primitive rather than the point sets required by shape contexts. However, a modified version of shape context is used in our case to help constrain locations of key points in Section 5.4.2.

The notion of 'pictorial structure' is used in a variety of vision algorithms to encode predictive spatial relationships between sub-parts or features of an object [27, 6, 8]. A similar idea is presented in the agent-based sketch interpretation system in [23]. We use 2D templates that encode similar predictive information regarding the location of various parts. Our template features are curve based and not image-based, and our goal is not only to identify a part but also to identify key points and curves on the part, as is necessary for the informed 3D construction of the object from the matched 2D template.

2.4 Interpreting line drawings of 3D polygonal objects

Interpreting line drawings of 3D polygonal objects is a problem that has attracted considerable interest in the past [19, 2, 7, 16], and continues to be the subject of ongoing work [20, 34]. However, these techniques are limited to polygonal objects and they work best with objects having many 90-degree corners or sets of symmetric angles.

2.5 3D Modeling

A variety of constructive, gestural 3D modeling techniques have been developed for pen-based interaction. These use pen strokes to incrementally develop the shape of a 3D object. SKETCH[40] and Teddy [15] represent seminal work in this area. Also in [31], a sketch modeling tool that uses a BlobTree as the underlying shape representation has been developed. Complex BlobTree models can be constructed with a spatial caching technique. Our work has been inspired by these systems, and also by recent work on sketching clothing[37]. The sketch-based system that we present assumes more knowledge about the objects to be drawn, but, in exchange, it supports drawings that can directly represent the final object shape. We see these approaches as being largely complementary in their application.

Also relevant to our work are ideas related to sketch-based retrieval of objects from a 3D database[11, 10, 4]. If there are a large number of sub-parts to an object, each of which can be parameterized in its own way, an approach that matches complete models requires a database to contain the cross-product of all variations, which does not scale effectively. A last interesting application of databases is the work of [35], where a database of airplane images is used to help simplify the design of 3D-curve networks, although the end product is not a 3D model and there is no general part recognition scheme.

Work in model-based vision[22] is also related to our problem, although the models used are usually only parameterized in very limited ways. More distantly related is the work of [30], which fits parameterized generative models to 3D range data.

The latest work in 3D modeling includes the system of automatic photo pop-up [14], which attempts to automatically construct a simple 3D model from a single image by learning a statistical model of geometric classes from a set of training images. Instead of trying to explicitly extract geometric parameters from a single image, their approach is to rely on training data to get statistical information.

Chapter 3

System Overview

An overview of the system is given in Figure 3.1. It takes three steps to produce a 3D model from a sketch:

- The input pen strokes are first preprocessed in order to produce a graph structure.
- The sketch graph is fitted with a series of 2D templates to find the best match.
- Given a best-fit object template and all the instanced part templates that participated in the fit, a 3D model is constructed from the labelled 2D sketch.

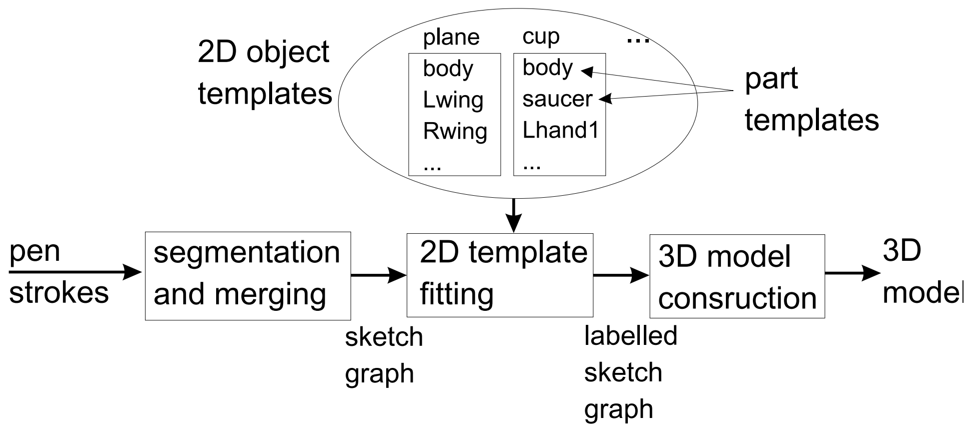


Figure 3.1: System Overview.

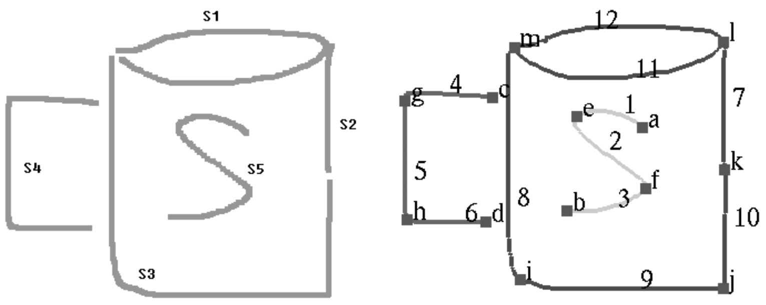
The system works in two modes. The more effective mode of use for our system is one in which the user can tell the system which class of object is being draw, and thus the system knows in advance which object template to match against the drawn sketch. The other mode that we support is to have the system do a linear search through each of the available object templates in turn and find the best-fit object template.

3.1 Preprocessing

The input pen strokes are first preprocessed in order to produce a graph structure. The preprocessing steps consist of:

- Eliminate irregularities from input strokes and convert the sketch to a normalized coordinate system.
- Create a graph structure from the preprocessed data through segmentation and merging. As shown in Figure 3.2, segmentation requires that nodes are added at the start and end points and points with local curvature maximum while close nodes are merged in the next merging step.

The resulting sketch graph will not necessarily be fully connected because parts such as airplane windows and fish eyes can be drawn in isolation. We also choose not to segment strokes at T-junctions; these typically occur when sub-parts are attached to main object parts and our sketch recognition does not require graph connectivity for the recognition of such sub-parts. Figures 3.2(a) and (b) show the strokes of an input sketch and the resulting sketch graph.

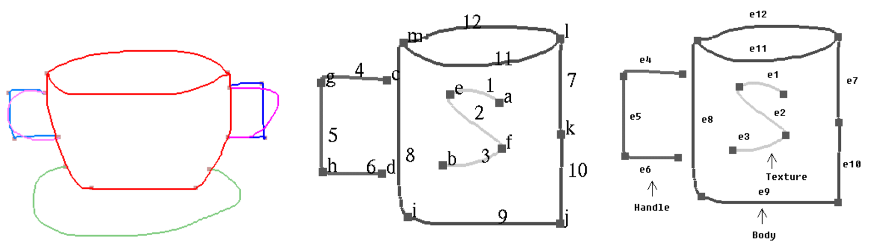


(a) (b)

Figure 3.2: The preprocess. (a) Original sketch, consisting of 5 strokes. (b) Graph computed from sketch strokes.

3.2 2D Template Matching

Given a sketch graph, the next step is that of fitting a series of 2D templates to the sketch graph. This will result in all the parts of the sketch graph being assigned meaningful part labels. The matching happens at two levels:



(a) (b) (c)

Figure 3.3: The matching process. (a) A cup template. (b) Sketch graph of a cup. (c) Labelled sketch graph of a cup.

- Multiple object templates are fitted to the sketch graph and scored for their fit. Only the best-fit labelling is ultimately retained.

- At a lower level, each object template consists of multiple part templates, arranged in a hierarchy. As the part hierarchy is traversed, attempts are made to match part templates to the drawn strokes.If a sufficiently good match is found for a part template, it is instanced, and the search continues by also considering the children of the given part.

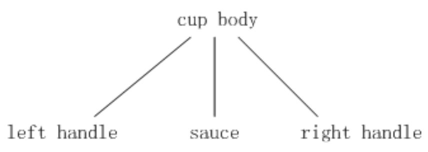


Figure 3.4: Part-template hierarchy for a cup .

The object template supports a flexible instantiation of the part templates. Parts may be deemed as mandatory or optional. A choose-one-of-N option may also be specified. For example, a cup template can specify that it may have an optional right handle and that the system should choose between a rounded handle part template and a square handle template, depending on which one fits best. The ability to support this type of information distinguishes our recognition approach different from most other shape recognition approaches that support only a flat hierarchy, i.e., given N fixed templates, find the best-fit template. Character recognition or object recognition problems with well-detailed sets of objects typically use such flat hierarchies. However, the available variations of the objects we wish to sketch and model precludes this type of “classification” approach.

Both object and part templates are represented as graphs with pre-specified node locations. Just as in a sketch graph, template graph edges are sketched curves. The actual process of matching part templates is thus accomplished using a search over node correspondences, which are then scored using a curve-matching metric on the best-fit curve correspondences that the node correspondences induce. A more detailed description will be

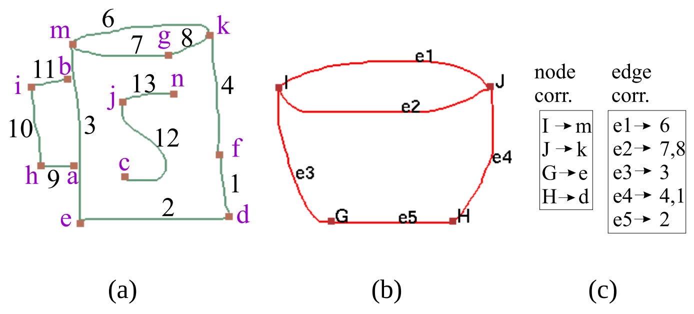


Figure 3.5: Matching a part. (a) Sketch graph of a mug. (b) Cup-body template graph. (c) Computed best-fit correspondences of body part.

given shortly in Chapter 5.

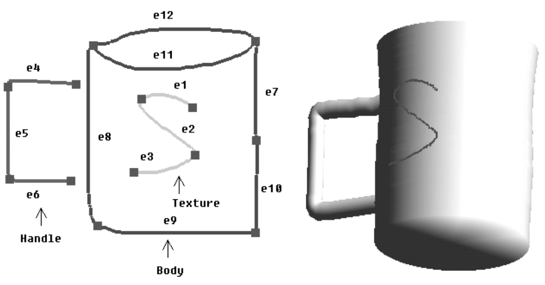
Figure 3.5(b) shows a cup-body template and Figure 3.5(c) lists the best-fit correspondences that were computed for it. These correspondences thus define the labelling of the sketch graph.

3.3 3D Modeling

Given labelled 2D sketch, the last step is that of constructing a 3D model from the labelled 2D sketch.

3D modeling is done by extracting measurements from the 2D sketch, such as wing length or cup height, as well as by fitting spline curves to particular stroke segments. For example, we use a multi-segment spline curve to smoothly approximate the sides of the airplane fuselage and the shape of the rounded cup handle. Given a priori knowledge about the object class and the extracted measurements that describe this particular desired instance, a 3D model can be constructed.

Adding a new class of object to the system thus consists of designing a 2D object template that consists of multiple part templates, as well as developing a procedural means



(a) (b)

Figure 3.6: 3D modeling. (a) Labelled sketch graph of a cup. (b) A 3D cup from 2D sketch.

of constructing the 3D shape from the labelled sketch.

Chapter 4

Preprocessing

In this chapter, we describe how the input strokes from users are preprocessed to produce a sketch graph.

The data we collect from the users are time-stamped (x, y) points. In order to compare the input data with a predefined template, we produce a "sketch graph" structure from the collected points. The preprocessing step consists of two stages. The first stage is to smooth and normalize the input data. The second stage is to extract key points from the preprocessed data.

4.1 Data Preprocessing

Factors such as noise during digitization, irregularity of pen motion, and variations in drawing style can all contribute to low quality of the data, as we show in Figure **??**.

To overcome these problem, we use several preprocessing techniques: first normalization, then resampling, and finally smoothing. Normalization refines the sketch in a normalized 2D coordinate system $x \in [0,1], y \in [0,1]$. Resampling can reduce the amount of redundant sample points and also add points where necessary through interpolation. Smoothing can eliminate noise and irregularities.

4.1.1 Sketch Data

We present the sketch as a number of strokes, which is an ordered sequenced of points $\{p_i\}_{i=1}^{n}$, where $p_i = (x_i, y_i, t_i)$. The points p_1 and p_n correspond to the first and last points of the stroke.

4.1.2 Normalization

Because some steps in our recognition algorithm are not scale-invariant, different sizes of sketches need to be normalized. As shown in Figure 4.1, the screen coordinates $(x_i, y_i)$of point p_i will replaced by

$$x_i^* = \frac{x_i - x_{min}}{L_{max}},$$
$$y_i^* = \frac{y_i - y_{min}}{L_{max}},$$

where $L_{max} = \max\{(x_{max} - x_{min}), (y_{max} - y_{min})\}$, and $[x_{min}, x_{max}, y_{min}, y_{max}]$ is the bounding box of the input sketch. This normalization is simple, but can make the matching free of scale.

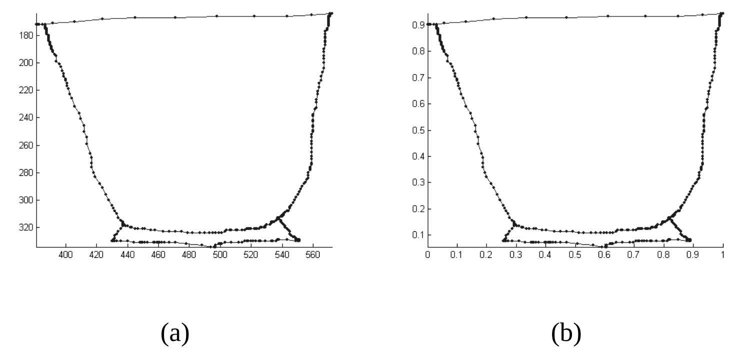


Figure 4.1: Normalization of preprocessing. (a) The original sketch. (b) The normalized sketch.

4.1.3 Resampling

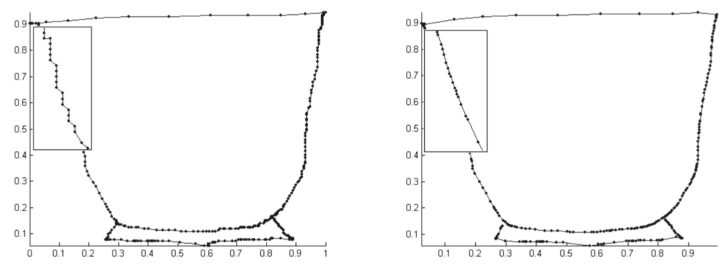
Resampling achieves a more regular spatial separation between the sampled points. Resampling consists of first averaging with neighboring points in a predetermined vicinity, then removing repeated points, and finally adding new points that are interpolated in regions of low density. In each of the following steps, the points to be processed are denoted as p_i before processing and p_i^* after being processed. However, in this notation, a point p_i to be processed at a current step may be some point p_i that has been processed by the previous step.

Averaging Points

In order to smooth strokes and help remove some small features which are usually made by mistake such as hooks in the beginning or end of a stroke, a point p_i is constituted by the average of the temporally and spatially neighboring points. As shown in Figure 4.2, a new averaged point

$$p_i^* = \frac{1}{|V_r(p_i)|} \sum_{p \in V_r(p_i)} p$$

where $|V_r(p_i)|$ denotes the cardinality of $V_r(p_i)$, and $V_r(p_i) = \{p \in s : \| p_i - p \| \leq r\}$.



(a) (b)

Figure 4.2: Average points. (a) The input sketch. (b) The output sketch.

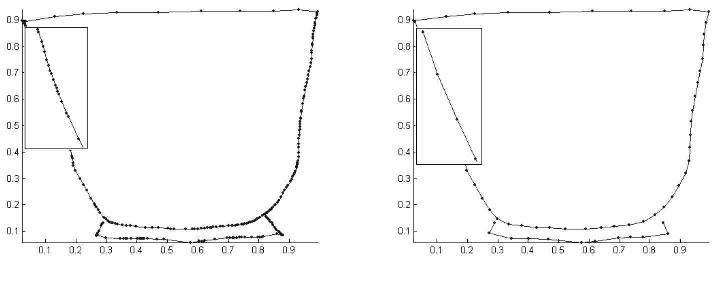
Reducing Points

People tend to slow down or even stop at corners or when they are thinking about what to draw next. Time-based sampling will produce many closely-spaced points or repeated points at these moments. As shown in Figure 4.3, we remove points where spatial density is too high.

Algorithm 1 Reduce Points

```
1: $p_c = p_0$ $\{p_{i(i \in 0,1,\ldots,n-1)}$ are points of a stroke$\}$
2: for $i = 1$ to $n-1$ do
3:     if $\| p_i - p_c \| < r$ then
4:         remove $p_i$
5:     else
6:         $p_c = p_i$
7:     end if
8: end for
```

From the start point p_1 of a stroke, p_2 will be reduced if it is in a predefined vicinity r. Thus we get the next new point p_2^* which is far enough from p_1 . Starting from p_2^*, the next new point p_3^* can be determined which is far enough from p_2^* and so forth. We use $r = 1/25$ after normalization. After removing all the redundant points and renumbering the points , $\| p_{i+1}^* - p_i^* \| \geq r$ holds for all points.



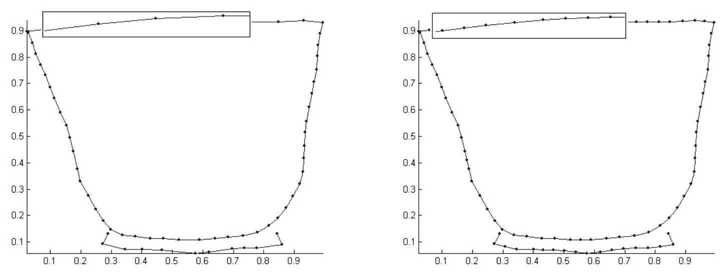
(a) (b)

Figure 4.3: Reduce points. (a) The input sketch. (b) The output sketch.

Adding Points

Sometimes we need to add points when users draw fast strokes and as a result, only few points are sampled. The procedure of adding points is simple. As shown in Figure 4.4, we check the distance between every two neighboring points first. If the distance of two points is larger than a predefined threshold, the same as the threshold for reducing points, we add appropriate number of points between them. For example, p_i and p_{i+1} are two neighboring points. If $d > \alpha * r$, where $d = \| p_{i+1} - p_i \|$, r is the threshold distance, α is a predefined constant (we use α = 1.5), then $n = int(d/r) - 1$ new points will be inserted into p_i and p_{i+1} using equal spacing and linear interpolation.

After the stages of reducing and adding points, $0.75r \leq \| p^*_{i+1} - p^*_i \| \leq 1.5r$ holds for all points.



(a) (b)

Figure 4.4: Add points. (a) The input sketch. (b) The output sketch.

4.1.4 Smoothing

Low-pass filtering is applied to remove pen jitter, which can be the source of false corner detection. The filters are typically a weighted average of neighboring points in the stroke:

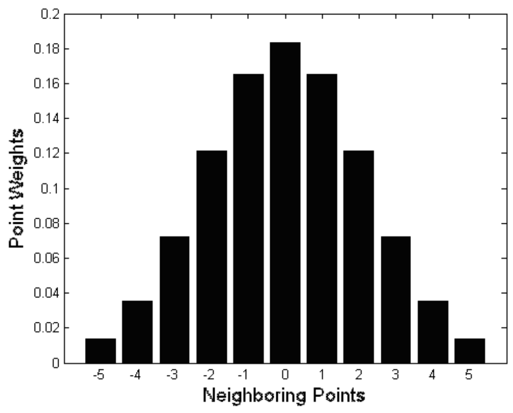
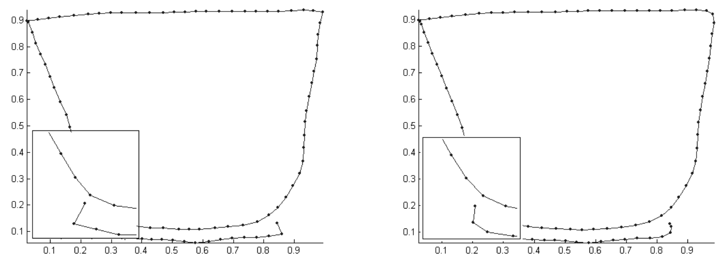


Figure 4.5: Coefficients obtained by a discrete approximation of a Gaussian distribution.

$$p_i = \sum_{k=-n}^{n} \alpha_k p_{i+k},$$

where $\sum_{k=-n}^{n} \alpha_k = 1$, and n is the size of the filter window. We use $n = 5$.

The most commonly-used coefficients are $\alpha = 1/(2n+1)$, which defines a simple box filter. In Figure 4.5, coefficients are generated by a discrete approximation of a Gaussian distribution. In Figure 4.6, an input sketch is smoothed with the given filter.



(a) (b)

Figure 4.6: Smooth points. (a) The input sketch. (b) The output sketch.

4.2 Segmentation and Merging

Given a series of preprocessed input strokes, the next step turns them into a stroke graph, as shown in Figure 4.7. This step is called segmentation and merging. Segmentation divides a long stroke into smaller segments at points of high curvature. Lastly, we need to build the sketch graph. This is done by adding nodes to start and end of each segment. At the merging phase, the nodes which are very close to each other are merged to make a sketch graph.

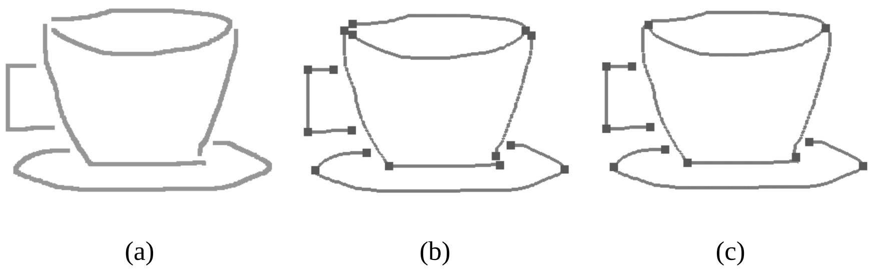


Figure 4.7: Segmentation and merging. (a) The input strokes. (b) Long strokes are divided into small segments. (c) Segments are connected by merging close nodes.

4.2.1 Segmentation

Given a stroke, we find corner points by looking for points of high curvature. To avoid false positives from noise, we use the technique of average-based filtering to get the global extrema of regions above a computed threshold.

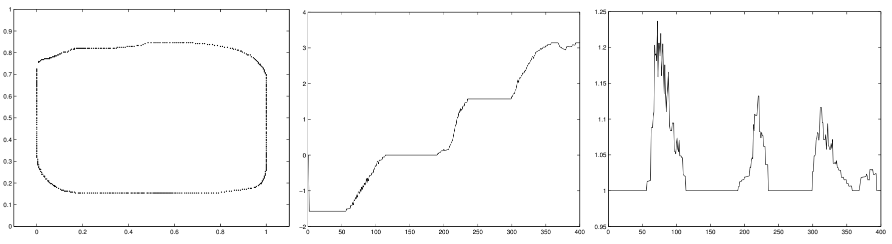
Definitions

For a given stroke, as shown in Figure 4.8, the direction and curvature for each point are computed as

$$\phi_n = \arctan(\frac{y_{n+1} - y_n}{x_{n+1} - x_n})$$

$$c_n = \frac{\Sigma_{i=n-k}^{n+k-1} D(d_{i+1}, d_i)}{D(n-k, n+k)}$$

ϕ_n and c_n represent the direction and curvature of the *n*-th stroke point respectively. k is the neighborhood size around the *n*-th point. We set k = 2 empirically as a tradeoff between the suppression of noise and the sensitivity of vertex detection. $D(n-k, n+k)$ is the straight line distance between the $n-k$ and $n+k$ points. We use $\phi \in [-\pi, \pi]$.



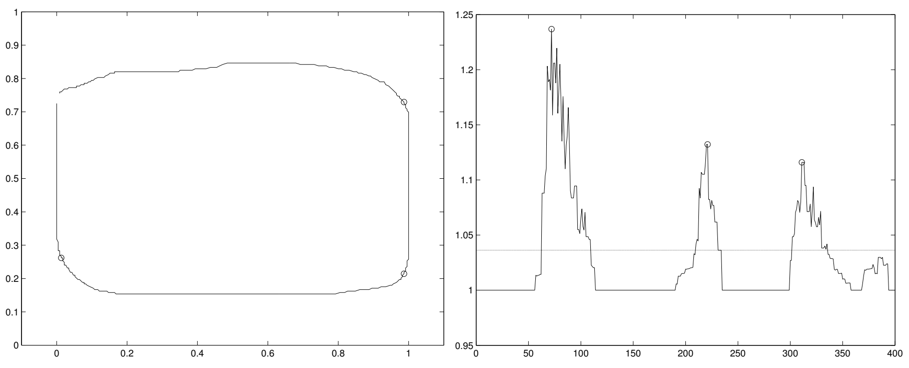
(a) (b) (c)

Figure 4.8: (a) Stroke in normalized coordinates. (b) Direction. (c) Curvature.

Average based filtering

Extrema in curvature typically correspond to corner points while noise in the data introduces many false positives. To deal with this, average-based filtering is applied here.

To accomplish average-based filtering, we select extrema above a threshold. This threshold is computed by scaling the mean of all the curvatures. The scale factor is empirically set to 0.9 in our system. We use this threshold to separate the data into regions where it is above/below the threshold. As shown in Figure 4.9, the global extremum within each region is selected to be the corner vertex. Note that if the stroke turns smoothly, we can simply pick the middle point of each region to be corner point. We use the global extremum within each region in our system because it works for most cases.



(a) (b)

Figure 4.9: Average based filtering is used to find the corners. (a) The corner points of a square are found (circled). (b) Curvature graph for the square with the threshold dividing it into regions. Global extrema in each region (circled) corresponds to the corner point found in (a).

4.2.2 Merging

As a last step, nodes which are very close to each other are merged, as shown in Figure 4.7 . The result is a sketch graph, where the nodes correspond to corners and end points of strokes, and the edges correspond to one or a set of stroke segments.

Chapter 5

Sketch Recognition

A sketch recognition algorithm is at the core of our sketch-based modeling system. In this chapter, we first describe the 2D templates and their construction. We then describe how template graphs are matched to the sketch graph using a search over possible correspondences. We explain the curve feature vector which is used to evaluate the correspondences. Lastly, we give further details about the hierarchical structure of the templates.

5.1 Overview of the Recognition Process

5.2 Template Construction

Templates can be described at two different levels. The most basic unit is a curve. Template parts are constructed from multiple curves.

5.2.1 Template Curve

Curves are the basic unit of a template. To describe a template curve, we define a curve feature vector $\mathbb{F}$. Figure 5.1 denotes the key parameters that we use to build the feature vector. It is defined as $\mathbb{F} = [f_1 \quad f_2 \quad f_3]^T$, where $f_1 = \theta,\ f_2 = a/d,\ f_3 = A/d^2,\ \theta$ is the angle of the straight line between the curve endpoints with respect to the horizontal, a is the arclength of the curve, d is the straight-line distance between the endpoints, and A is the signed area between the curve and the straight-line. Feature f_1 encodes preferences for desired angles, as our sketch recognition scheme is not rotation-invariant. Features f_2 and f_3 encode information about the type of curve that connects the endpoints and provide useful distinctions between a curve that passes above or below the straight line (f_3), as well as how much the curve meanders (f_2).

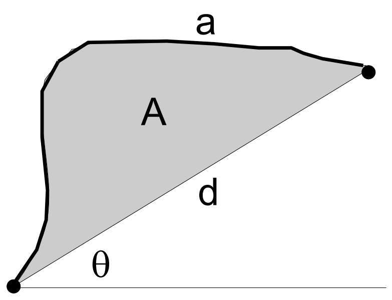


Figure 5.1: Key parameters used to compute the curve feature vector: arclength a, straight-line distance d, angle θ, and area A.

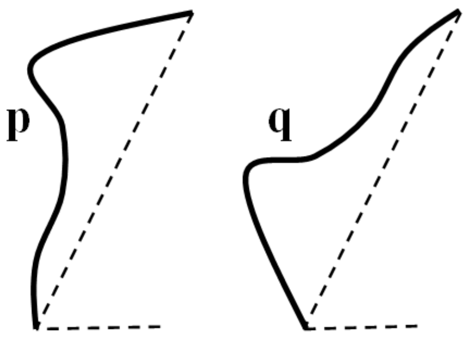


Figure 5.2: Curve p and q have same curve feature vectors.

We note that a limitation of the curve feature vector is that two curves that are symmetric with respect to the perpendicular bisector of the endpoints will have the same curve feature vector, as shown in Figure 5.2. Also, the curve feature vector is potentially problematic for closed curves. For this reason, closed curves are automatically segmented into open curves. An axis-aligned box is computed for the stroke. If $w > h$, the curve is segmented at the points where x_{min} and x_{max} occur. Otherwise, the curve is segmented where y_{min} and y_{max} occur.

5.2.2 Template Part

Individual part templates are subgraphs of the global template graph. A simple graphical user interface supports their construction. Using a mouse, the user selects a set of edges that constitute a desired part template. The nodes and edges of this subgraph are then saved as the part template definition. Also, a hierarchy is established among the parts by designating an edge of an existing part template to serve as the *parent edge* of the new part template. This serves the dual purpose of introducing a hierarchical order for search and instantiating the model parts, as well as a means to encode knowledge about the relative location of parts.

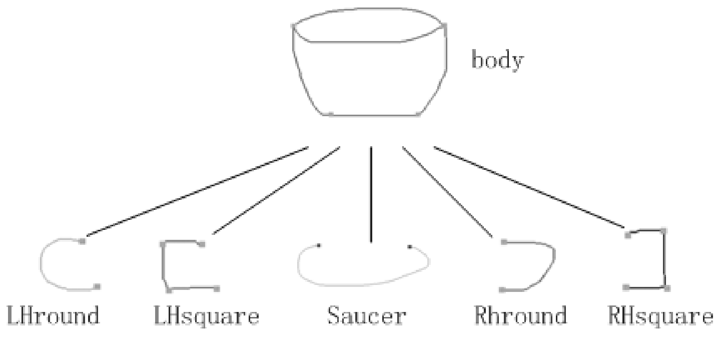


Figure 5.3: Hierarchy of Template Parts of a Cup.

5.2.3 Template Object

An object template is a graph, consisting of nodes which represent key points on the object, and edges, which represent curves of particular shapes that are expected to be found in a sketch. An example of an object template for a cup is shown in Figure 5.4. An object template is itself constructed using a sketch, where each stroke becomes an edge in the resulting template graph.[1] Nodes are added at the start and end of each stroke and then merged with nearby neighbors. Figure 5.4(a) shows the curve templates. The curves are grouped to be part templates as showin in Figure 5.4(b). The resulting template graph is shown in Figure 5.4(c).

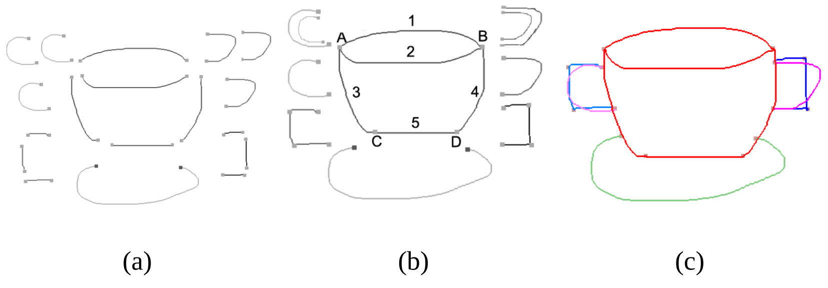


Figure 5.4: Construction of the cup template. (a) Template curves. (b) Template parts by grouping template curves. For example, the cup-body template consists of 5 curves. The left-side handles, right-side handles, and saucer are parented by the cup-body curves 3, 4, and 5, respectively. (c) Template object.

Creating a 2D template for a new class of objects requires careful thought, although its actual construction can be done in about 15-20 minutes using the GUI. The template should represent a stereotypical example of the desired class of object. The template parts also need to provide the information necessary for the appropriate 3D reconstruction of the various parts. Although it is rather easy to add a new template itself, it takes much longer time to create a procedure to generate a parameterized 3D model of the given object class.

[1]Stroke segmentation is turned off when drawing the template graph.

5.3 Matching Process

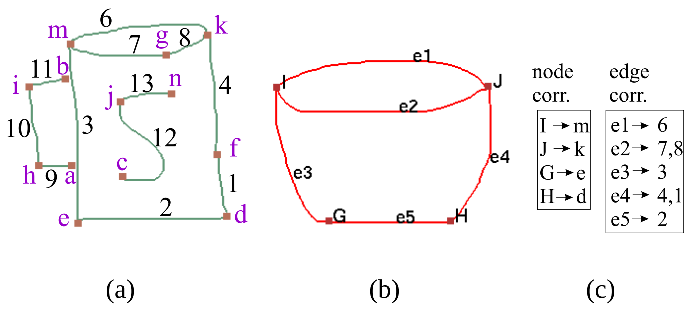
5.3.1 Matching a Curve

In order to determine if a given path through the sketch graph is similar in shape to a desired edge of the template that we seek to locate in the sketch, the matching function that compares two curve feature vectors is defined as $\mathbb{M}(F_a, F_b) = k \sum_i w_i g(\sigma_i, f_{i_a} - f_{i_b})$, where $g(\sigma, x) = e^{-0.5(x/\sigma)^2}$ and $k = 1/\sum_i w_i$. The w_i values provide a relative weighting of the feature vector components: f1 for angle preference, f2 for curve meandering and f3 for direction. We use $w_1 = 2, w_2 = 1, w_3 = 1$ to favor more of the relative positions of the two curve endpoints. The matching function provides a maximum score of 1 for curves that are highly similar, and a score of zero for those that are very dissimilar. σ_i provides a means of scaling the feature vector elements (or rather their differences) to provide a meaningful level of sensitivity. We empirically choose $\sigma_1 = 22^\circ, \sigma_2 = 0.25, \sigma_3 = 0.1$.

5.3.2 Matching a Part

The process for matching a part consists of searching the sketch graph for best-fit correspondences of the template graph nodes and edges. A best-fit solution consists of (1) a corresponding sketch graph node for each template graph node; (2) a correponding *path* of one or more sketch edges (curves) for each template edge; and (3) a score that denotes the quality of the fit. As described in Algorithm 2, the match scores of all curves that make up a part template are summed in order to yield a total match score for the part template.

The matching process is summarized in Algorithm 2 and consists of an iterative process that begins by assigning a particular set of corresponding sketch-graph nodes to the template-graph nodes (lines 4–6). From this assumed node correspondence, a best-fit correspondence is computed for each template edge (lines 8–12). To understand this



node corr.	edge corr.
I → m	e1 → 6
J → k	e2 → 7,8
G → e	e3 → 3
H → d	e4 → 4,1
	e5 → 2

Figure 5.5: Matching a Part. (a) Sketch graph of a mug. (b) Cup-body template graph. (c) Computed best-fit correspondences of body part.

process, we note that a given template edge, such as the left-hand side of the cup body, may correspond to a *path* in the sketch graph that traverses multiple sketch-graph edges. This is because the sketch strokes may be over-segmented when constructing the sketch graph, or because a particular feature may have been drawn using multiple strokes. Both of these cases can be observed in the example shown in Figure 5.5. For example, template edge e is matched to sketch graph segments 1 and 4, which are originally two separate strokes.

There may furthermore be multiple paths between a given pair of nodes in the sketch graph, and it needs to be determined which of these paths best corresponds to the given template edge. The *pathset(n_a, n_b)* function determines a set of possible paths, $\mathbb{P}$, between nodes n_a and n_b in the sketch graph using a depth-first graph traversal. In order to bound the number of possible paths, which can become large for a highly connected sketch graph, we bound the search to paths whose ratio of arclength to straight-line distance is less than twice this same ratio as computed for the template edge that we are seeking to match.

Each path $p \in \mathbb{P}$ is tested for similarity with the given template edge that we seek to match (line 10). This is accomplished by computing a *curve feature vector*, $\mathbb{F}$, for the template curve e_t and for each path p, and computing a match score $\mathbb{M}(\mathbb{F}(e_t), \mathbb{F}(p))$ based upon these feature vectors. The details of how $\mathbb{F}$ and $\mathbb{M}(\mathbb{F}_1, \mathbb{F}_2)$ are computed are given in

Algorithm 2 Part Matching

```
1: input $\mathbb{S}(N_s, E_s)$ { Sketch Graph}
2: input $\mathbb{T}(N_t, E_t)$ { Template Graph}
3: for $i = 1$ to $N_{iter}$ do
4:     for all $n_t \in N_t$ do
5:         $C[n_t] \leftarrow$ select( $N_s$ )
6:     end for
7:     score $\leftarrow 0$
8:     for all $e_t \in E_t$ do
9:         $\mathbb{P} \leftarrow$ pathset( $C[n_1(e_t)], C[n_2(e_t)]$ )
10:        $p \leftarrow$ bestMatchPath( $e_t, \mathbb{P}$ )
11:        score $\leftarrow$ score + $\mathbb{M}(e_t, p)$
12:    end for
13:    if score $>$ bestScore then
14:        update bestScore and best correspondences
15:    end if
16: end for
```

Section 5.3.1.

Several methods could be used to arrive at appropriate choices of node correspondences to be evaluated (lines 4–6). One choice would be to systematically evaluate all permutations of assignments of the N_t template nodes to the N_s sketch nodes. While this approach is guaranteed to find globally-optimal correspondences, it suffers from a combinatorial explosion of the number of combinations to be considered. Our system currently employs a stochastic local search method, beginning a search by using a uniform random assignment of sketch nodes to template nodes. Randomized local changes to the current set of correspondences are then evaluated by matching score of edge correspondences and accepted or rejected in a greedy fashion. If a local maximum is reached, the current best solution is updated if necessary, and the search is restarted at another randomized point in the space of all possible correspondences. We limit the total number of iterations to $N_{iter} = 2000$.

Although we do not explicitly evaluate a matching function for node correspondences, we encode some knowledge of expected locations of nodes into the template in Section 5.4.1 and also use shape context in Section 5.4.2 to constrain the search over possible node correspondences.

5.3.3 Matching an Object

Object templates are specified in a simple script file and consist of an ordered list of part templates T_j to be matched to the sketch graph, $\mathbb{S}$, as well as an instancing threshold, α_j. The thresholds provide a means to allow parts that have a low best-match score to not be instanced as part of the model. This provides control to the template designer as to whether or not a scribble drawn to the right of the cup should be interpreted as a cup-handle (low threshold), or as a scribble that is to be ignored (high threshold).

Part templates also provide support for one-of-N matching. For example, it may be possible to interpret a cup-handle as either a rounded handle or a square handle, each of which has its own part template and its own associated 3D-model-construction method. A comma-separated list of part-template names in the object template has a one-of-N semantics, meaning that a fit should be attempted of all the templates in the list, but only the best-fitting template should be instanced.

The scores of part templates can be combined to compute an overall object template score. Currently, we compute a score based on the mean scores of the instantiated part templates. The object template scores allow a sketch to be fit with all the possible object templates, with only the best-fit object template being used to instantiate the 3D model.

5.3.4 Matching with Multiple View/Object Templates

Sometimes two or more templates, such as a top view and a side view, may be needed to recognize sketches of the same object from different viewpoints. For such cases, the sketch from one view will be matched to the template from the same view first. This is done automatically by matching templates from both views to the sketch and finding the best match. Then all the labelled sketches from different views are normalized to the same scale according to the correspondences from templates. In this way, 2D measurements of the same part can be meaningfully extracted from different sketches and then combined to produce the 3D model.

Multiple object templates, such as templates of a cup and a plane, are matched in turn in order to recognize the sketched object. At the end of this process, if the highest match score is above threshold, the object is recognized and then all the parts are instanced. Otherwise, sketch is labelled as unrecognized.

5.4 Making the Search Efficient

In order to constrain the search over possible node correspondences, global and local information about part locations is encoded with the object template. In our project, the extracted information includes some knowledge of expected locations of the nodes as well as local information from the shape context of a node in its neighborhood.

5.4.1 Matching Hierarchy

To constrain the search areas of nodes, we encode some knowledge of expected locations into the template and use this knowledge to constrain the search.

The part templates that are at the root of the part hierarchy, such as the body of the plane or the body of the cup, do not have a parent edge. For these parts, the part template explicitly stores an expected location for the graph nodes that comprise the part. By using a normalized coordinate system that is defined by the axis-aligned bounding box of the template sketch, normalized coordinates are computed for each node. The bounding box of a drawn sketch serves to instantiate this normalized coordinate system for the sketch and thereby provides a set of expected locations for the nodes. As shown in Figure 5.6, the sketch recognition algorithm will only search for correspondences for a template-graph node within a given radius of the expected location. This radius is defined in normalized coordinates (we use $r = 0.3$) and thus typically maps to an elliptical region in the sketch.

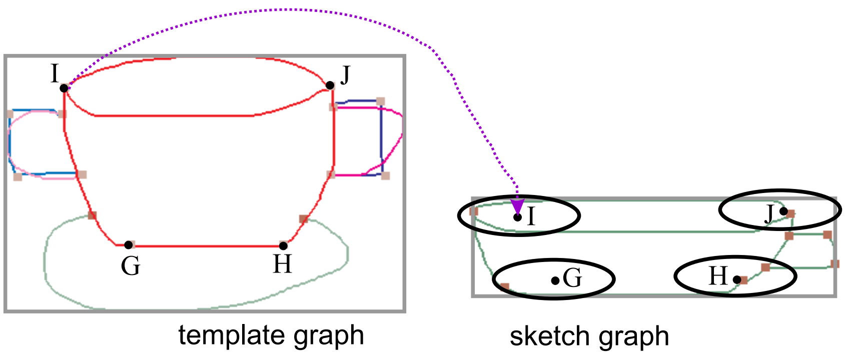


Figure 5.6: Expected locations in the sketch graph are computed for the template graph nodes of the cup body using a normalized coordinate system based on the bounding box.

Part templates that are not at the root of the part hierarchy, such as the wing of the plane or the handles of the cup, are associated with a parent edge. Associated with the parent edge is a *bounding polygon* (BP), which serves to represent the area in which to search for a part template during the sketch recognition. We use a convex quadrilateral to represent the BP. The BP is specified during the design of a part template by dragging its vertices to the desired locations in the template sketch window. The polygon lives in a coordinate system that is defined by the parent edge. One of the two nodes associated with the edge becomes the origin of this coordinate system, and the other node locates the point $(1,0)$. As shown in Figure 5.7, this allows the BP to be transformed to the sketch coordinate system once the parent edge has been labelled in the sketch. If the parent edge is not labelled in the sketch, i.e., the parent part was not instantiated, then there will be no attempts to fit the part templates which use that parent edge. The BP provides the sketch-recognition algorithm with necessary information about the expected relative location of parts and thus serves to greatly constrain the search space of possible correspondences.

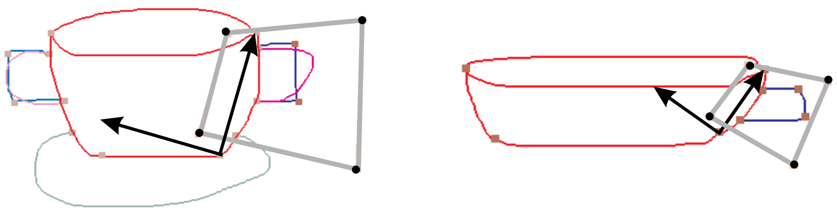


Figure 5.7: Transformation of the bounding polygon for the right-cup handle. The template is shown on the left, and a sketch on the right. The right-hand edge of the cup-body template serves as the parent edge for this part.

5.4.2 Shape Context

Besides the use of the inherent hierarchy of template to restrict the search, more information can be extracted from templates to help constrain the node correspondences.

Original Shape Context

A descriptor which is called the shape context was proposed [3] to find the best matching point q_i on the second shape for each point p_i on the first shape. The shape context is defined as follows. If there are n points on a shape, a full set of vectors originating from one point p_i to all other $n-1$ sample points on a shape can express the configuration of the entire shape relative to the reference point. However, this is much too detailed to use as a shape descriptor. Instead, a histogram of the *distribution* over relative positions can be used as a more compact, yet highly discriminative descriptor. For a point p_i, a coarse histogram h_i of the relative coordinates of the remaining $n-1$ points is computed:

$$h_i(k) = \sharp\{q \neq p_i : (q - p_i) \in bin(k)\}$$

where $\sharp$ is the number of points in one bin, $k \in \{1, 2, ..., K\}$. This histogram is defined to be the shape context of p_i. In order to make the descriptor more sensitive to positions of nearby sample points than to those points far away, bins are uniform in $log - polar$ space.

To compare the similarity of two points p_i and q_j, the χ^2 test statistic is used to define the cost function $C_{ij} = C(p_i, q_j)$ as the following:

$$C_{ij} \equiv C(p_i, q_j) = \frac{1}{2}\Sigma_{k=1}^{K} \frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)}$$

where $h_i(k)$ and $h_j(k)$ denote the K-bin normalized histogram at p_i and q_j respectively. The shape context can be applied to grey-level images, but it is very dependent on gray-scale brightness values. For line drawings such as sketches in our project, it is more applicable.

Localized Shape Context

Inspired by the idea of shape context, we define our own localized shape context. Compared with the shape context [3], our localized shape context differs in two aspects. First, we only compute the shape context within a limited range of p_i instead of over all points. We localize the shape context to avoid the errors which are brought by optional parts. Second, we compute a weight-based shape context instead of dividing points into bins.

As shown in Figure 5.8, axes are used in localized shape context instead of the bins in the original version. Thus instead of putting a neighboring point P_1 in one of the bins we project it to the two closest axes, II and III, with weights. By computing weights from the angles θ_1 and θ_2, the closer axis III gets more weight from P_1. The reasons of using localized shape context can be seen in Figure 5.9.

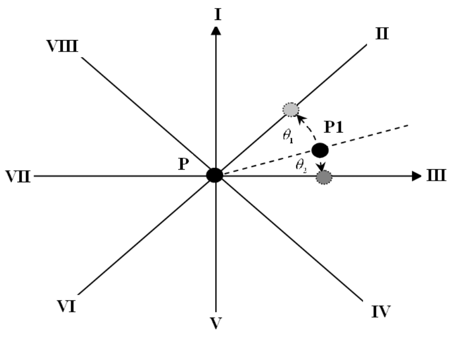


Figure 5.8: Localized shape context computation. Axes are used instead of bins in shape context. A neighboring point P_1 is projected to the closest two axes with weights which are proportional to the angles.

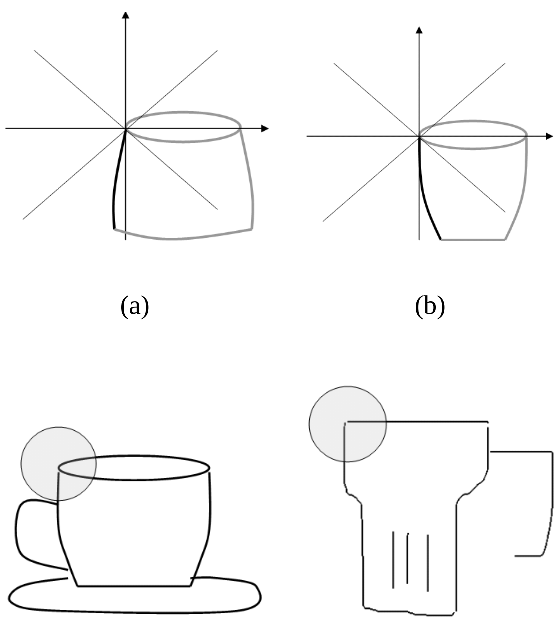


Figure 5.9: Examples of using localized shape context. (a) (b) If we compute the original shape context of the top-left nodes of the cups with the bins divided by the eight axes, the left side strokes of the two cups will be put into completely different bins. Using localized shape context, the left side strokes are projected to the same axis with most weights. (c) (d) With localized shape context in the circled areas, we can avoid effects from other parts of the sketch such as texture and saucer.

Chapter 6

3D Model Construction

Given a set of instanced template parts, the last step is that of constructing the 3D model from the instanced parts. This is done by extracting measurements from the 2D sketch, such as locations, lengths and fitted spline curves. Given a priori knowledge of the object class and the extracted measurements, a 3D model is constructed by using extrusions, surfaces of revolution, and swept shapes.

6.1 2D Measurement Extraction

There are many possible ways to extract 2D measurements from the labelled sketch. We choose spline fitting and the use of bounding boxes. For example, we use a multi-segment spline curve to smoothly approximate the sides of the airplanes fuselage.

6.1.1 Spline Fitting

To approximate a curve where necessary, we use a multi-segment Catmull-Rom spline.

Catmull-Rom Spline

A Catmull-Rom spline is defined by four control points as shown in Figure 6.1. One of the features of the Catmull-Rom spline is that the specified curve will pass through all the control points.

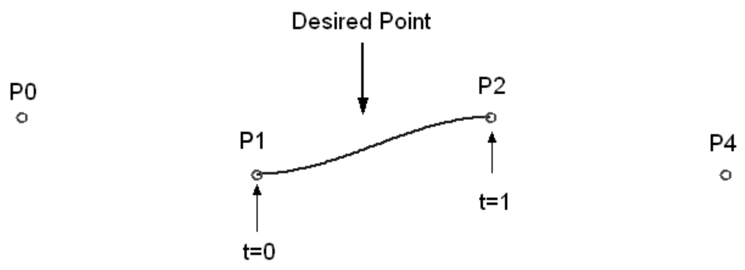


Figure 6.1: Control points of Catmull-Rom spline.

Given four control points P_0,P_1,P_2,P_3, the Catmull-Rom curve is defined as follow:

$$q(t) = TMP$$

where $T = 1+t+t^2+t^3$, $M = \begin{pmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{pmatrix}$, and $P = \begin{pmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{pmatrix}$, t $\in$ [0,1].

Multi-segment Catmull-Rom Spline

A long curve requires multiple spline segments to approximate well. More control points can be added to keep more details from a curve, as shown in Figure 6.2. Control points can be sampled according to curvature. For a smooth curve, control points can also be sampled by distance so that all the control points can distribute evenly throughout the stroke. In our case, most of the strokes that we need to fit with splines are smooth curves, so we sample control points by distance. We adjust the number of segments of a spline by sampling more or less points from the input stroke and using the sample points as control points. Each spline segment is defined by two control points that form the endpoints of the segments and one additional control point on either side of the endpoints. Thus for a given segment with endpoints p_{n-1} and p_n, the segment would be calculated using $[p_{n-2}, p_{n-1}, p_n, p_{n+1}]$. For the first and last segment of the stroke, we duplicate the end point.

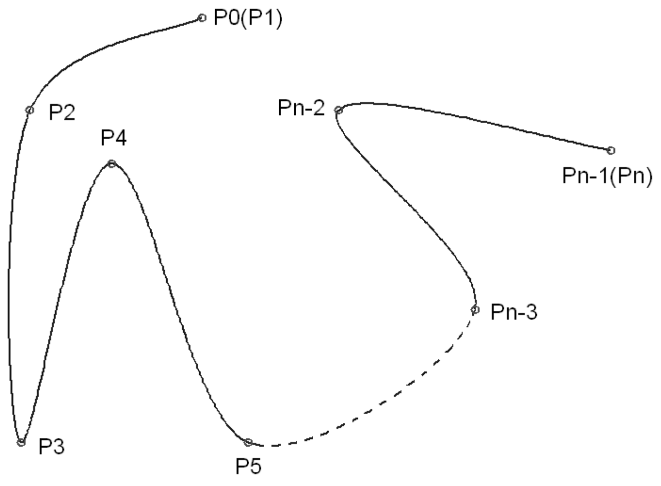


Figure 6.2: Multi-segment Catmull-Rom Spline.

6.1.2 Other Extraction Methods

Other features are also extracted from the 2D curves for use in the 3D construction. Figure 6.3 shows the bounding box used to get the height H and width R of a specified shape. Major and minor axes of a shape can be computed either directly by using principal component analysis, or by sampling the possible angles of rotation to extract the angle that yields the maximum projected length. We use the latter one for its simplicity.

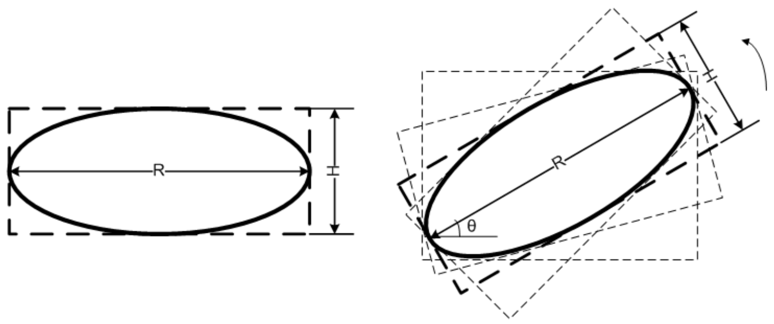


Figure 6.3: Axis-aligned bounding box and oriented bounding box. In the right figure, the major axis yields the maximum projected length.

6.2 3D Mesh Construction

Our framework was designed to accommodate many possible 3D construction methods. We have implemented two: surfaces of revolution and swept surfaces.

6.2.1 Surface of Revolution

A surface of revolution is a surface generated by rotating a two-dimensional curve about an axis. The resulting surface therefore always has azimuthal symmetry. Examples of surfaces of revolution include perfect apples, cones (excluding the base), cylinders (excluding the ends), and spheres.

As shown in Figure 6.4, given a curve $y = f(x)$ and an axis such as $x = a$, a surface can be generated by rotating sample points on the curve around the axis and then triangulating the resulting surface. For one point (x_i, y_i) on the curve $y = f(x)$, the 3D points which can be generated by revolution are:

$$\theta_k = \frac{2\pi k}{n}$$

$$\begin{cases} x_{ik} = \cos(\theta_k)x_i \\ y_{ik} = y_i \\ z_{ik} = \sin(\theta_k)x_i \end{cases}$$

where $k = 1, \ldots, n$, and n is the desired number of angular samples.

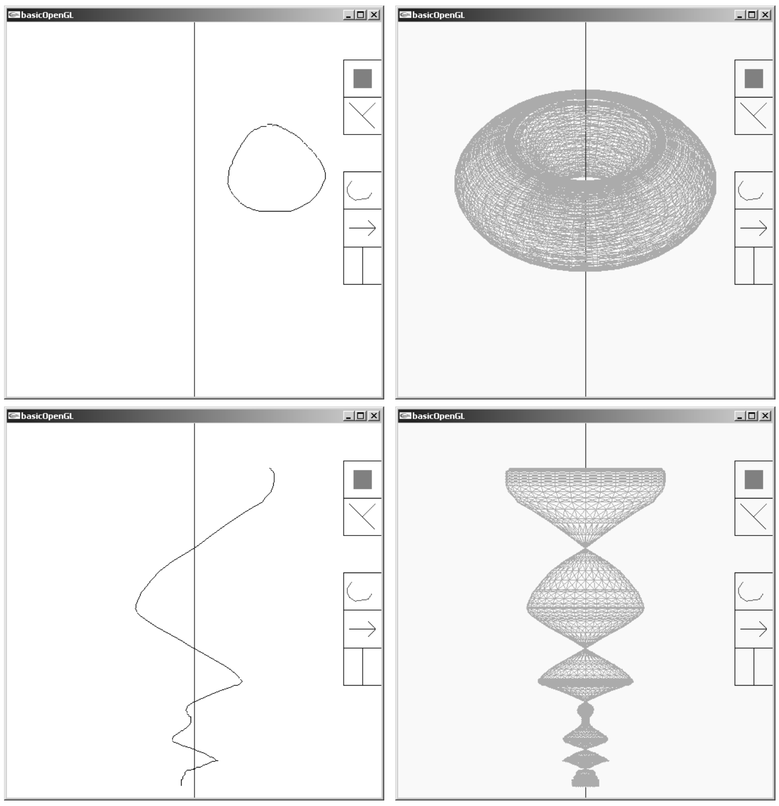


Figure 6.4: Examples of surface of revolution.

6.2.2 Swept Shapes

Surface of revolution can only generate objects with azimuthal symmetry. For 3D shapes such as airplane fuselages or fish bodies, swept shapes are a useful modeling primitive. These objects are constructed by specifying a two-dimensional shape and a path along with the shape is moved. For example, to construct a 3D cup handle, a circle with default diameter is obtained as the cross-sectional shape and then it is moved along the spline which is fitted to the sketch handle.

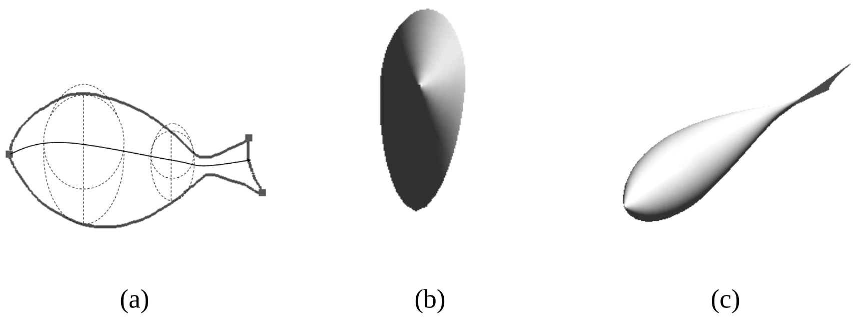


Figure 6.5: Sweep construction of fish body. (a) The input sketch with cross-sectional slices and sweep path. (b) The cross section of the 3D model. (c) Another view of the 3D model.

In Figure 6.5, a 3D fish body is constructed as swept shapes. Here the cross-sectional shape varies when it is being moved along the path. The shape consists of a half circle and a half ellipse both of which are determined by the vertical distance between the point on the path and the point on the sketch strokes. The path line is obtained by connecting points on the vertical lines between points on the upper stroke and the lower stroke.

6.2.3 Texture

As shown in Figure 6.6, strokes that are drawn over the cup body automatically become annotations on the surface of the body. To implement this, we project the strokes to the generated surface. As illustrated in Figure 6.7, tracing over a photograph can also be a powerful way of creating a sketch. At the same time we can lift the texture from the photograph. The resulting texture mapping is not necessarily of high quality because of inaccuracies in the tracing. However, models require only seconds to create.

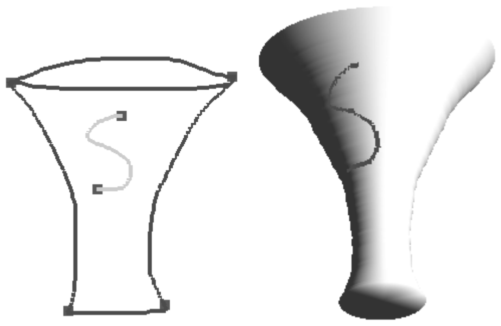


Figure 6.6: Strokes on the cup body are projected to the generated surface.

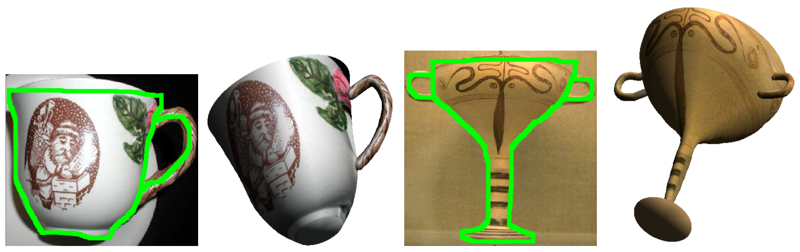


Figure 6.7: Photographs are traced and used as texture for the generated 3D models.

6.3 Specific Examples

We now discuss the specific details for constructing 3D cups, airplanes, and fish. A supplementary video showing our system is available at http://www.cs.ubc.ca/∼cyang.

6.3.1 Cups and Mugs

A variety of cups and mugs modeled using our system are shown in Figures 6.8. The cup template represents the canonical view that is commonly used to draw or photograph a cup. The template supports left and right rounded handles and square handles. A second left and right rounded handle can also be recognized, which can serve to either define inside and outside edges for the handle, or as a second rounded handle. In our implementation, it is the model building process that distinguishes between these two cases.

For single-curve handles, a default handle width and cross-section is assumed. For

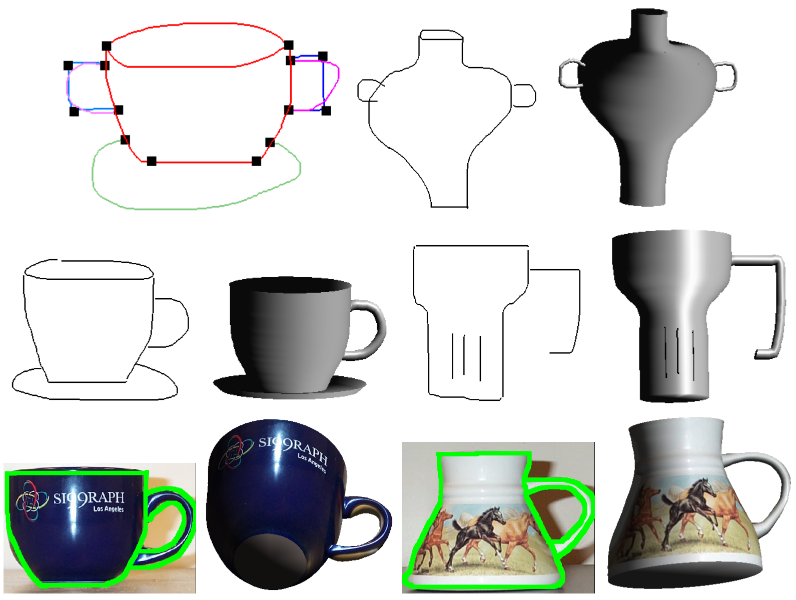


Figure 6.8: Cup template, followed by sketches of cups and wine glasses, shown together with the synthesized 3D models.

double-curve handles, a rounded-rectangular cross-section is assumed that has fixed depth but varies in width according to the drawn curves. The handles and sides of the cup are both automatically fitted with spline segments that serve to smooth the sketched curves. The cup body is constructed using a surface of revolution that is developed from only one of the edges, although a computed ‘average’ based on the expected symmetry about the centerline could also be used. The saucer is also constructed from a surface of revolution for a spline that is fitted to the corresponding stroke. The 3D handle is constructed by sweeping a predefined 3D cross-section along the spline fitted to the sketched handle. Adjustments are made to the endpoints so that the cup handles are attached to the body even if there is a gap in the sketch. Unrecognized strokes that are drawn over the region of the cup body are projected onto the cup body as annotations, as shown in Figure 1.1.

6.3.2 Airplanes

Our system can reconstruct 3D airplane models from sketches. The work described in this section was carried out by Dana Sharon.

For sketching airplanes, the traditional orthographic views (top, side, and front) are typically the easiest to sketch and also provide the most information. We employ templates for the top-view and the side view. These two views are recognized as separate sketches, each having their own template. The information from both labelled sketches is then combined in the model building process. The system also accepts a top-view in isolation, in which case default assumptions are made about the fuselage and the tail.

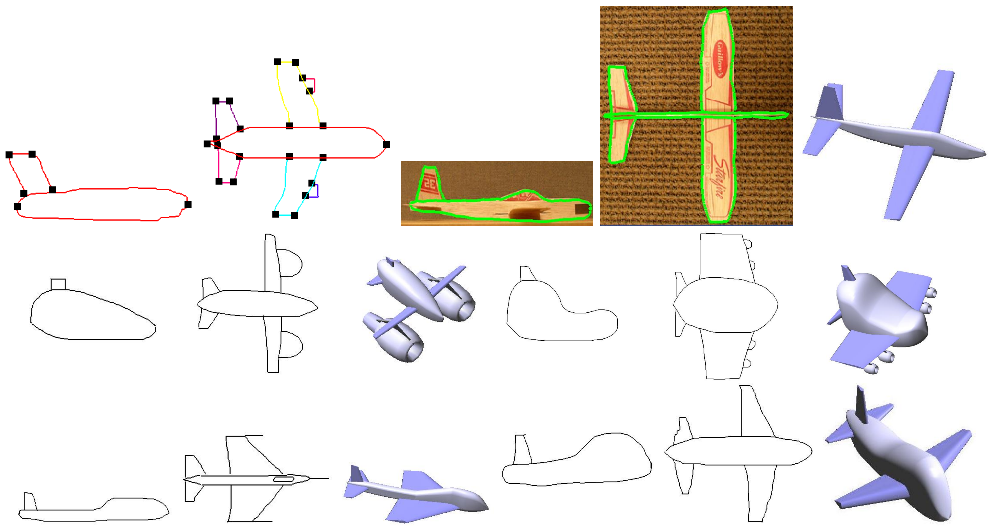


Figure 6.9: Airplane modeling.

The fuselage has a default cross-sectional shape that is allotropically scaled as it is swept along the spline curves defined by the fuselage side, top, and bottom curves. The wings and horizontal stabilizers are constructed using a swept-airfoil cross-section. The tail fin has a similar swept construction. Engines are instanced as copies of a single 3D engine model, scaled and translated to match the top-view sketch. Left-right symmetry is enforced in the final model. The model building process must also resolve any conflicts between the top view and the side view, such as the length of the fuselage. This is handled by scaling one of the views so that the fuselage lengths matches the other view. Figures 1.1 and 6.9 show a number of sketches and the resulting 3D airplane models.

6.3.3 Fish

Fish are modeled by tracing the appropriate features from a side-view photograph. Alternatively, a top-view template could be added to provide this information, as with the airplane.

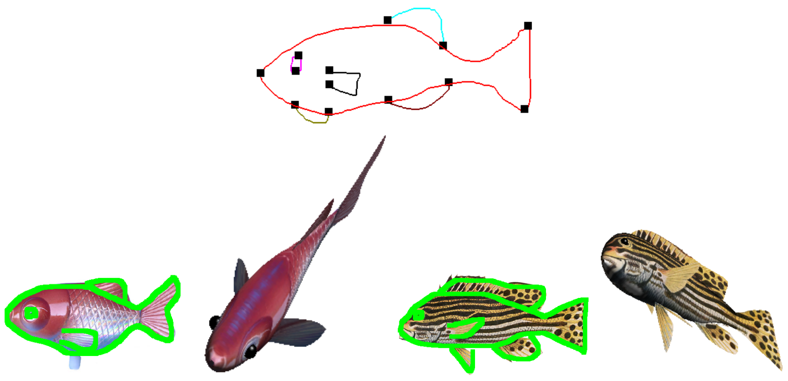


Figure 6.10: The fish template and two fish models that were created by tracing over photographs.

The 3D shape of the body is determined from a default elliptical cross-sectional shape. The fins are polygons and are placed at default angles with respect to the fish body.

Chapter 7

Results

7.1 Experiment

Some of the user data are shown in Figure 7.1. In the experiment, users were asked to draw several sketches of cups or glasses with no further constraints being given. Instead of showing them real cups, we let them draw from memory. Notice in the figure that although users were not required to draw from any specific viewpoint, almost all of them draw the cups from a side view with most of the right handles. The 2D templates and the procedure for building the 3D models from the 2D templates are informed by the observation of the user data.

7.2 System in Use

Our system is tested on a desktop PC with standard mouse and keyboard configuration, a tablet, and SMART Board. Users can use the mouse, stylus, or their fingers on the touch-screen of SMART Board to draw a sketch. One drawing is normally done in less than one minute. Once the sketch is drawn, the reconstructed 3D model pops up in less than one second.

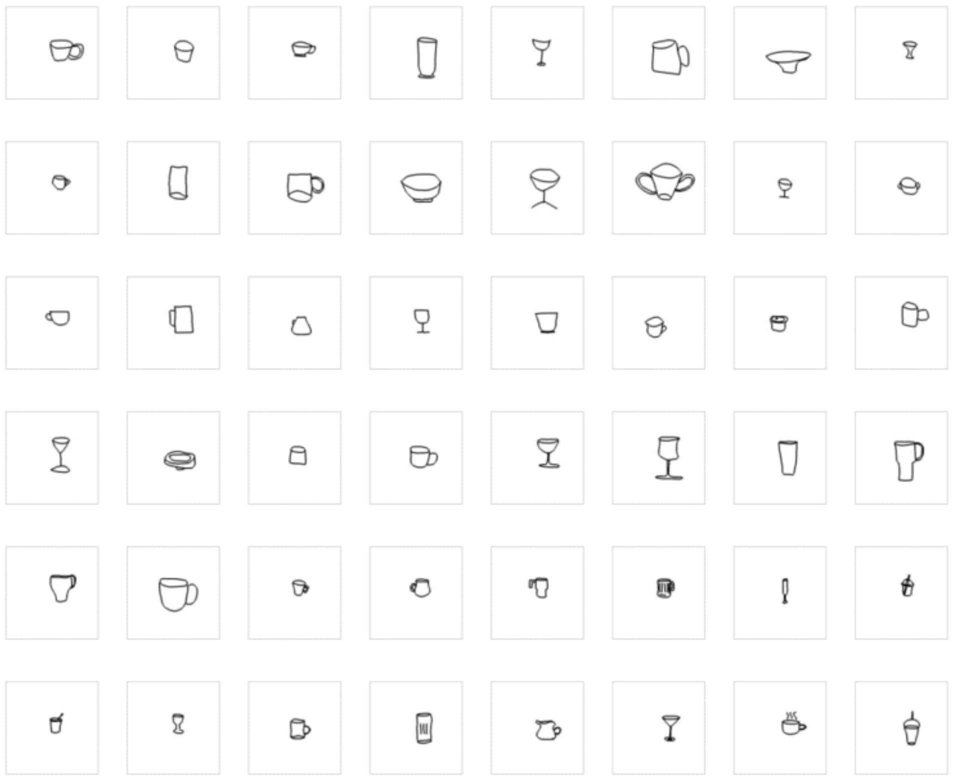


Figure 7.1: Some of the user sketches of a broad range of cups and mugs.

7.3 Results

We have tested the system using 3 classes of objects: cups, planes, and fish. Table 7.1 gives the list of part templates used for each object template. We have found that developing the script or code that builds the 3D model from the labelled 2D sketch is generally the most time-consuming part of adding a new object class.

The current sketching interface consists of a sketch area, an area for displaying the resulting 3D model, and a button panel. The user draws their desired sketch in the sketch area and hits a ’Recognize’ button. The most effective mode of use for our system is one in which the user can tell the system which class of object is being drawn, and thus the system knows in advance which object template hierarchy to match against the drawn sketch. The

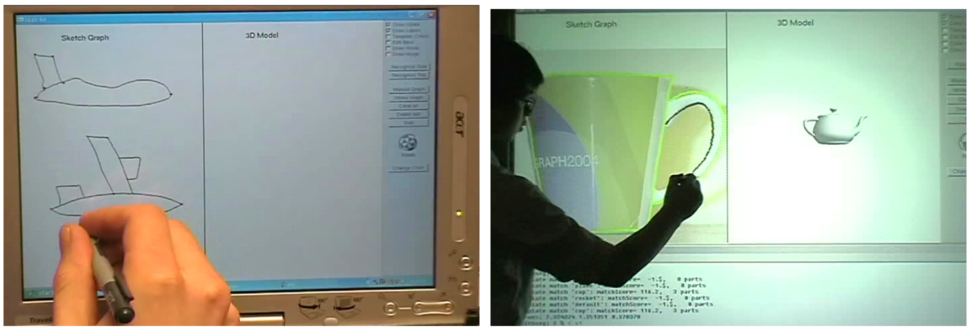


Figure 7.2: Our system is run on a tablet (left) and a SMART Board (right).

plane	*n*	*e*	**cup**	*n*	*e*	**fish**	*n*	*e*
body	2	2	body	4	5	body	3	3
Lwing	4	3	saucer	2	1	topfin	2	1
Rwing	4	3	RHround1	2	1	midfin	2	1
Ltail	4	3	RHround2	2	1	eye	2	2
Rtail	4	3	LHround1	2	1	botfin1	2	1
Lengine	2	1	LHround2	2	1	botfin2	2	1
Rengine	2	1	RHsquare	4	3			
Lengine2	2	1	LHsquare	4	3			
Rengine2	2	1						

Table 7.1: List of template parts used to build the cup, plane, and fish templates. *n* and *e* denote the number of nodes and edges in the part template graphs, respectively.

current recognition process requires on the order of one second to match an object template hierarchy against a sketch, as measured on an 800 Mhz TabletPC.

Another mode that we support is to have the system do a linear search through each of the available object templates in turn and then instantiate the 3D object corresponding to the best-fit object template. If there are many templates loaded, the recognition can become slow and it also becomes more prone to recognition errors, such as potentially interpreting a flat cup as a plane body. In the future, statistically-based object-class recognition algorithms borrowed from the computer vision literature could be used to help identify likely object classes and to therefore avoid a linear search through all object templates.

The interface supports a progressive workflow if this is desired. The recognition and model building can thus be invoked at any time for whatever parts that have currently been drawn. One caveat is that parts such as the handles of the cup cannot be recognized in isolation. The parent part must first be successfully instantiated.

Images can be loaded into the background to use as a reference for tracing a particular airplane, cup, or fish. At the same time, the model building can extract a texture map from the background image if this is desired. This thus supports quick-and-dirty construction of models from photographs. In the future, we foresee the possibility of using current-generation image segmentation algorithms to reduce or eliminate the tracing step, thus further automating the knowledge-based construction of 3D models from particular classes of photographs.

7.4 Discussion

An obvious question that arises from our template-driven system is: ”Is it not simpler to have users directly edit the template curves to get the object they want?”. Our answer to this is twofold. First, drawing is often easier than template manipulation. Consider the case of the cup template, which has 4 handles and a saucer, as shown in Figure 5.4. A great many control points would need to be repositioned to create the tall mug shown in Figure 1.1, whereas it can be drawn in a matter of seconds. The user also has to understand what control points are, why there are two left handles and two right handles, has to delete 3 of these, and then move the control points on the remaining handle in order to achieve the squarish right handle that is finally desired. A similar case can be made for drawing airplanes instead of editing an airplane template. All the optional template parts introduce considerable visual clutter and need to be explained to the user. A sketched curve that represents a particular feature, such as the side of a cup, can be fit with a dynamically-chosen number of spline segments, whereas a template would typically be constructed with a fixed number of spline segments. Second, and perhaps most importantly, a drawing interface allows for the kind of transparency in use that we strive for in this work. The user should be required to know as little as possible about the underlying representation and assumptions that will be used to construct the 3D model.

A second question that one can ask of our system is: ”Will it scale to large numbers of object classes?”. Our solution is scalable with the addition of more templates. To be efficient with a large number of templates would require a separate object classification step in order to identify a small set of candidate object classes to which the full template match could then be applied. While our system still has robustness issues, it can support large within-class variations. For example, our cup template can model a large variety of cup or mug types and shapes (Figures 1.1 and 6.8). Similarly, our airplane template supports a significant variety of shapes (Figure 6.9). With respect to the use of templates, is is clear that some form of prior knowledge is essential for sketch or line-drawing recognition, and we choose to embody this prior knowledge in our 2D templates.

How familiar do users need to be with the templates in order to use the system? This project began with us collecting sketch data on a TabletPC, giving purposely-vague instructions by asking users (with no knowledge of templates) to “draw a cup,” or “draw a plane,” in a wizard-of-oz experiment. More than half of the cup sketches can be recognized by our system. Most of the airplane sketches cause errors. About half of the time the errors are serious enough that the object is not recognized. Fish can always be recognized without finding all of the fins. Thus, some familiarity with the template is generally required.

Chapter 8

Conclusions and Future Work

We have presented a system that supports quick-and-dirty creation of 3D models based upon a sketching interface. Many recent sketch-based systems apply a gestural or context-sensitive interpretation to drawn strokes. Instead, we propose a template-based recognition algorithm which treats strokes as a sparsely-populated image and then drives the instantiation of a flexibly-paramaterized 3D model.

8.1 Limitations

Our sketching system has a number of known limitations. As currently defined, the object templates do not encode information about the expected scale of parts because the curve feature vectors are scale-invariant. The curve feature vector currently has no notion of the smoothness of the curve, and thus smooth template curves may be matched to jagged paths through the sketch graph without penalty, assuming that the remaining curve features match well. The system is not nearly as robust as we would like. The most common reason for sketches not being recognized is because of some sloppiness in sketches that results in disconnected strokes. Increasing the threshold distance within which sketch-graph nodes are merged allows for more of such gaps to be bridged, but this comes at the expense of losing further detail in the drawn parts because strokes which were intended to be distinct may be merged together. The key-points in the curves that become the nodes of the sketch graph are not always extracted as desired.

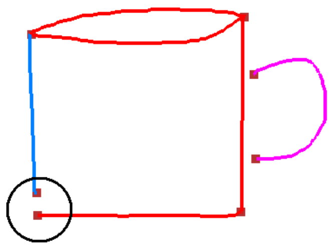


Figure 8.1: One failure example because of disconnected strokes.

8.2 Future Work

There are numerous interesting directions for future work. Our sketch recognition method allows for recognition with only one example, namely the object template, but a statistically-based approach based on a set of hand-labelled sketches would potentially provide a more informed approach. This could be applied to modeling the space of expected part shapes as well as the sketching tolerances that should be accomodated. Currently, the sketch recognition process is driven in a top-down fashion. A hybrid top-down, bottom-up approach has the potential to be significantly more efficient and more robust. User testing under controlled conditions would provide useful data with respect to robustness and determining the extent to which familiarity with the template structure is helpful.

Bibliography

[1] Christine Alvarado and Randall Davis. Sketchread: a multi-domain sketch recognition engine. In *UIST '04 ACM symposium on User interface software and technology*, pages 23–32, 2004.

[2] H. G. Barrow and J. M. Tenenbaum. Retrospective on interpreting line drawings as three-dimensional surfaces. *Artificial Intelligence*, 59:71–80, 1993.

[3] S. Belongie and J. Malik. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(24):509–522, April 2002.

[4] D.-Y. Chen, X.-P. Tian, Y.-T. Shen, and M. Ouhyoung. On visual similarity based 3d model retrieval. *Computer Graphics Forum (Eurographics 2003)*, 22(3), 2003.

[5] James M. Coughlan and Sabino J. Ferreira. Finding deformable shapes using loopy belief propagation. In *ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part III*, pages 453–468. Springer-Verlag, 2002.

[6] C. M. Cyr and B. K. Kimia. 3d object recognition using shape similarity-based aspect graph. In *Proceedings of ICCV*, 2001.

[7] L. Eggli, C. Hsu, B. Bruderlin, and G. Elber. Inferring 3d models from freehand sketches and constraints. *Computer-Aided Design*, 29(2):101–112, 1997.

[8] P. F. Felzenszwalb and D. P. Huttenlocher. Pictorial structures for object recognition. *Intl. Journal of Computer Vision*, 61(1):55–79, January 2005.

[9] R. Fergus, P. Perona, and A. Zisserman. A visual category filter for Google images. In *Proceedings of the 8th European Conference on Computer Vision, Prague, Czech Republic*, May 2004.

[10] Thomas Funkhouser, Michael Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David Dobkin. Modeling by example. *ACM Trans. Graph.*, 23(3):652–663, 2004.

[11] Thomas Funkhouser, Patrick Min, Michael Kazhdan, Joyce Chen, Alex Halderman, David Dobkin, and David Jacobs. A search engine for 3d models. *ACM Trans. Graph.*, 22(1):83–105, 2003.

[12] L. Gennari, L. B. Kara, and T. F. Stahovich. Combining geometry and domain knowledge to interpret hand-drawn diagrams. In *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, 2004.

[13] T. Hammond and R. Davis. Automatically transforming symbolic shape descriptions for use in sketch recognition. In *AAAI*, pages 450–456, 2004.

[14] Derek Hoiem, Alexei Efros, and Martial Hebert. Automatic photo pop-up. In *SIGGRPAPH '05*, volume 24, pages 577–584, Los Angeles, CA, USA, 2005.

[15] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: a sketching interface for 3d freeform design. In *SIGGRAPH '99*, pages 409–416, 1999.

[16] T. Kanungo, R. M. Haralick, and D. Dori. Understanding engineering drawings: A survey. In *Proceedings of First IARP Workshop on Graphics Recognition*, pages 217–228, 1995.

[17] L. B. Kara and T. F. Stahovich. An image-based trainable symbol recognizer for sketch-based interfaces. In *AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural*, 2004.

[18] Joseph J. LaViola and Robert C. Zeleznik. Mathpad2: a system for the creation and exploration of mathematical sketches. *ACM Trans. Graph.*, 23(3):432–440, 2004.

[19] Yvan G. Leclerc and Martin A. Fischler. An optimization-based approach to the interpretation of single line drawings as 3d wire frames. *Int. J. Comput. Vision*, 9(2):113–136, 1992.

[20] H. Lipson and M. Shpitalni. Correlation-based reconstruction of a 3d object from a single freehand sketch. In *AAAI Spring Symposium on Sketch Understanding*, pages 99–104, 2002.

[21] S. Loncaric. A survey of shape analysis techniques. *Pattern Recognition*, 31(8):983–1001, 1998.

[22] David G. Lowe. Fitting parameterized three-dimensional models to images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(5):441–450, May 1991.

[23] G. Mackenzie and N. Alechina. Classifying sketches of animals using an agent-based system. In *Proceedings of the 10th International Conference on Computer Analysis of Images and Patterns*, pages 521 – 529, 2003.

[24] J. V. Mahoney and M. P. J. Fromherz. Three main concerns in sketch recognition and an approach to addressing them. In *AAAI Spring Symposium on Sketch Understanding*, March 2002.

[25] Andrew Nealen, Olga Sorkine, Mare Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. In *SIGGRPAPH '05*, Los Angeles, CA, USA, 2005.

[26] Michael Oltmans, Christine Alvarado, and Randall Davis. Etcha sketches: Lessons learned from collecting sketch data. In *Making Pen-Based Interaction Intelligent and Natural*. AAAI Fall Symposium, 2004.

[27] Marcello Pelillo, Kaleem Siddiqi, and Steven W. Zucker. Matching hierarchical structures using association graphs. *ECCV '98*, 1407, 1998.

[28] R. Plamondon and S.-N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Transations on Pattern Analysis and Machine Intellegence*, 22(1):63–85, 2000.

[29] Yuan Qi, Martin Szummer, and Thomas P. Minka. Diagram structure recognition by bayesian conditional random fields. In *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[30] Ravi Ramamoorthi and James Arvo. Creating generative models from range images. In *SIGGRAPH '99*, pages 195–204, New York, NY, USA, 1999.

[31] Ryan Schmidt, Brian Wyvill, and Mario Costa Sousa. Sketch-based modeling with the blobtree. In *SIGGRAPH '05 Technical Sketch*, Los Angeles, CA, USA, 2005.

[32] T. S. Sebastian, P. N. Klein, and B. B. Kimia. Shock-based indexing into large shape databases. In *EECV*, volume 3, pages 731–746, 2002.

[33] Tevfik Metin Sezgin and Randall Davis. Hmm-based efficient sketch recognition. In *Proceedings of the International Conferences on Intelligent User Interfaces (IUI'05)*, 2005.

[34] A. Shesh and B. Chen. Smartpaper: An interactive and user friendly sketching system. *Computer Graphics Forum (Eurographics Proceedings)*, 23(3), 2004.

[35] S. Tsang, R. Balakrishnan, K. Singh, and A. Ranjan. A suggestive interface for image guided 3d sketching. In *Proceedings of CHI 2004*, pages 591–598, 2004.

[36] Z. Tu and A. L. Yuille. Shape matching and recognition using generative models and informative features. In *Proceedings of ECCV'04*, 2004.

[37] Emmanuel Turquin, Marie-Paule Cani, and John Hughes. Sketching garments for virtual characters. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling*, 2004.

[38] Chen Yang, Dana Sharon, and Michiel van de Panne. Sketch-based modeling of parameterized objects. In *SIGGRAPH '05 Technical Sketch*, Los Angeles, CA, USA, 2005.

[39] Chen Yang, Dana Sharon, and Michiel van de Panne. Sketch-based modeling of parameterized objects. In *2nd Eurographics Workshop on Sketch-based Interfaces and Modeling*, Dublin, 2005.

[40] R. C. Zeleznik, K. P. Herndon, and J. F. Hughes. Sketch: an interface for sketching 3d scenes. In *SIGGRAPH '96*, pages 163–170, 1996.