Recovering 3D Shape from Concept and Pose Drawings

by

Mikhail Bessmeltsev

B.Sc., Novosibirsk State University, 2008
M.Sc., Novosibirsk State University, 2010

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

The Faculty of Graduate and Postdoctoral Studies

(Computer Science)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

September 2016

Abstract

Modern tools to create 3D models are cumbersome and time-consuming. Sketching is a natural way to communicate ideas quickly, and human observers, given a sketch, typically imagine a unique 3D shape; thus, a tool to algorithmically interpret sketches recovering the intended 3D shape would significantly simplify 3D modeling. However, developing such tool is known to be a difficult problem in computer science due to multitude of ambiguities, inaccuracies and incompleteness in the sketches. In this thesis, we introduce three novel approaches in CAD and character modeling that successfully overcome those problems, inferring artist-intended 3D shape from sketches.

First, we introduce a system to infer the artist-intended surface of a CAD object from a network of closed 3D curves. Second, we propose a new system for recovering a 3D model of a character, given a single complete drawing and a correspondingly posed 3D skeleton. Finally, we introduce a novel system to pose a 3D character using a single gesture drawing. While developing each system, we derive our key insights from perceptual and artist literature, and confirm our algorithmic choices by various evaluations and comparisons to ground truth data.

Preface

A version of Chapter 3 has been published in ACM Transactions on Graphics [11]. Most of the ideas originated in discussions between myself and Alla Sheffer. I carried out most of the implementation and testing, assembled the video, and presented the paper at SIGGRAPH ASIA 2012. Caoyu Wang contributed to the ideas and implementation of the final stage of the algorithm (Section 3.3) and the implementation of the stable matching algorithm. Karan Singh mostly helped with writing, designing validation procedures and overall framing of the story. Alla Sheffer and Karan Singh wrote most of the manuscript, I edited and wrote parts of the text, and created all the figures.

A version of Chapter 4 has been published in ACM Transactions on Graphics, 2015 [10]. Most of the ideas originated in discussions between myself and Alla Sheffer. I carried out most of the experiments, implementation, testing, evaluations, and video assembly. Will Chang contributed ideas to Section 4.4.1, overall Section 4.4, and helped with its implementation, along with implementation of various rendering tasks. Alla Sheffer, Karan Singh, and Nicholas Vining wrote most the manuscript, I and Will Chang edited and wrote parts of the paper. I also created most of the figures in the manuscript. Nicholas Vining and Karan Singh also helped with shaping the story, designing and conducting evaluations, and video narration.

Finally, a version of Chapter 5 has been conditionally accepted to SIGGRAPH ASIA 2016 conference. Most of the ideas originated in discussions between myself and Alla Sheffer. I carried out most of the experiments, implementation, testing, evaluations, and video assembly. Nicholas Vining contributed ideas to Section 5.6, helped with its implementation, carried out some experiments in Section 5.7, helped with some evaluations and video narration. I wrote the initial version of the paper; Alla Sheffer and Nicholas Vining wrote the final version of the paper, I edited the text, and created all the figures.

Table of Contents

Abstract ii

Preface iii

Table of Contents iv

List of Tables vii

List of Figures viii

1 Introduction 1
- 1.1 Overview of Contributions 3
- 1.2 Recovering Artist-Intended Surfaces from 3D Curves 4
- 1.3 Recovering Character 3D Model from a Cartoon Drawing . . 5
- 1.4 Recovering 3D Character Pose from a Gesture Drawing . . . 6

2 Previous Work 8
- 2.1 Sketch-based 3D Modeling 8
  - 2.1.1 Incremental Approaches 8
  - 2.1.2 Shape and Pose Reconstruction from Photographs and Video 9
  - 2.1.3 Character Shape Reconstruction from Complete Drawings 11
  - 2.1.4 Skeleton-based 3D Modeling 14
- 2.2 Posing Characters 14
  - 2.2.1 Adding 3D Effects to 2D Drawings 14
  - 2.2.2 3D Character Posing 15
- 2.3 Surface Reconstruction from 3D Curve Networks 16

3 Design-Driven Quadrangulation of Closed 3D curves . . . 20
- 3.1 Introduction 20
- 3.2 Quadrangulating a Closed 3D Curve 24

3.2.1 Segmentation and Matching . . . 25
3.2.2 Quadrangulation . . . 32
3.3 Processing Curve Networks . . . 36
3.4 Results . . . 38
3.5 Conclusions . . . 41

4 Modeling Character Canvases from Cartoon Drawings . . 43
4.1 Introduction . . . 43
4.2 Framework Overview . . . 47
4.3 Part Segmentation . . . 50
4.4 Canvas Modeling . . . 57
4.4.1 Computing a 3D Curve-Skeleton and 3D Contours . . 58
4.4.2 Canvas Connectivity . . . 61
4.4.3 Canvas Surfacing . . . 62
4.5 Perceptual and Design Validation . . . 66
4.5.1 Creating Overlaid 3D Skeletons . . . 66
4.5.2 Comparison to Ground Truth and Artist Drawings . 69
4.5.3 Perceived Contour Segmentation . . . 69
4.6 Results . . . 72
4.7 Conclusions . . . 77

5 Gesture3D: Posing 3D Character via a Gesture Drawing . 78
5.1 Introduction . . . 78
5.2 Related Work . . . 81
5.3 Parsing Gesture Drawings . . . 81
5.4 Framework Overview . . . 84
5.5 Character-Contour Correspondence . . . 86
5.5.1 Solution Space . . . 87
5.5.2 Unary Assignment Cost . . . 88
5.5.3 Assignment Compatibility . . . 89
5.5.4 Global Consistency . . . 91
5.5.5 Solver Mechanism . . . 93
5.6 2D Pose Optimization . . . 94
5.7 Full Pose Optimization . . . 94
5.8 Validation . . . 98
5.9 Results . . . 100
5.10 Conclusions . . . 104

6 Discussion and Conclusion 105
6.1 Discussion 105
6.2 Future Work 106
6.3 Conclusions 107

Bibliography 108

List of Tables

3.1 Algorithm statistics for different curve networks. 41

List of Figures

1.1 Graphical interface of a commercial industry-standard 3D modeling package. 2
1.2 A brief overview of Chapter 3. 4
1.3 A brief overview of Chapter 4 5
1.4 A brief overview of Chapter 5. 6

2.1 Stick figure drawings (a), lines of action (b), and outer silhouettes (c) allow for multiple perceptually valid pose interpretations. (d) Poor view selection results in highly foreshortened contours leading to loss of pose information (e.g bends on the left arm or the curved spine). Gesture drawings, consciously drawn from descriptive views (e) effectively convey the intended pose. 10
2.2 Character drawings do not conform to the assumptions made in previous work. (top) The contours of a surface of revolution whose axis is not in the view plane are typically not planar.(bottom) The contours of a typical character include numerous occlusions; a single contour curve can consist of multiple part outlines (see left arm and torso outline in (a)) and as shown by the side view (b) the contour curves are far from planar or view aligned. Our method introduced in Chapter 4 (c) successfully handles such inputs generating a character model similar to the ground truth input (b). 12
2.3 (a) Lacking part information, character shape reconstruction can at best exploit overall shape smoothness, e.g [68, 95]; (b) by using a skeleton to facilitate contour partition and part modeling, we generate a more believable character shape. . . 13
2.4 Using Laplacian diffusion (b) or Thin-Plate Splines [40](c) to surface a four-sided cycle leads to unintuitive results. (d) In contrast the flow lines on an interpolating Coons patch, by construction, bridge opposite cycle sides. 19

2.5 (top) Using a purely topological approach and applying midpoint subdivision (forming either four or six sides) generates a quad mesh with poor flow line layout (left and center). Our method in Chapter 3 (right) uses geometry driven segmentation and matching to generate smooth flow lines and a predictable surface. (bottom) On a concave cycle, parameterization onto a convex domain (a rectangle) leads to foldovers (left), our method automatically segments the cycle into convex quadrilaterals leading to a fair surface (right). 19

3.1 Steps to quadrangulating a design network of closed 3D curves (a) : Closed curves are independently segmented (b) and iteratively paired and refined to capture dominant flow-lines as well as overall flow-line quality (c); final quadrangulation in green and dense quad-mesh (d); quadrangulations are aligned across adjacent cycles to generate a single densely sampled mesh (e), suitable for design rendering and downstream applications (f). 21
3.2 Closed 3D curves: ambiguous hexagonal 3D curve (top) compared to complex curves with a clear design intent (bottom). 22
3.3 Artist designed interpolating quad-meshes. 23
3.4 After the initial segmentation (a), we alternate matching and refinement steps to obtain a pair-based curve segmentation which is converted into a quadrilateral network (c) . To minimize T-junction count (d) we compute global interval assignment, and use it to sample iso-lines on discrete Coons patches. 24
3.5 Iterative segmentation refinement: (a) initial segmentation where the matching highlights correct dominant side matches. The match quality is drastically improved by segmenting the bottom curve (b), and repeating the process (c) to obtain an even segment count. Further refinement has no real impact on matching cost. 27
3.6 Estimated bridge curvature for different segment layouts measured as angle (red) between bridge direction t_i and $p-p'$ (at a point p). The dashed lines visualize representative intersecting flow-lines (a). Shape similarity and distance cost terms (b). 28
3.7 Initial bridge direction t_i of segment i is determined by adjacent segment flow directions f_m, f_n and its normal. 30

3.8 A disconnected dual graph (left) does not allow for a valid primal quad mesh. Splitting the cycle into two by a temporary curve segment (dashed) generates valid graphs for both parts which combined together induce a valid primal quad mesh (right). 33
3.9 Two intersection orders induce different quad connectivity, with the one on the right inducing a better quad shape, and consequently a smoother flow. 33
3.10 We first position interior vertices (left) and then use the chain-long quads to position the interior curves (center). Finally, the resulting quad cycles are quad-meshed using discrete Coons patches (right). 34
3.11 Our distance based weighing (right) generates smoother flow line evolution than topology based one [94]. 34
3.12 Removing interior vertices: (Left) initial match (top) and induced quadrangulation (bottom); (Right) the final match with purple and green pairs flipped (top) has a slightly higher cost but the induced quadrangulation (bottom) has no interior vertices, leading to smoother flow-lines. 35
3.13 Separately processed cycles (a) introduce T-junctions. We first resolve the T-junctions across pairs of neighbouring patches by propagation (b), generating a well defined hierarchy of matching primary segments. We then use integer programming to compute interval assignments (c) that minimizes the number of T-junctions, typically leading to a watertight mesh (d). 36
3.14 Quadrangulation and meshing of closed curves. 38
3.15 Quad meshes of complex closed curves including interior cycles. 39
3.16 Artist generated meshes (left) and ours (right) exhibit very similar flow-line patterns. 39
3.17 Quadrangulation and meshing of curve networks. The stars indicate the network locations of the highlighted complex regions. 42

4.1 Character drawings (c) are traditionally anchored around a skeleton (a), surrounded by generalized surfaces of revolution (b). We use the drawn character contours (d) and a corresponding 3D skeleton (red-to-blue coloring reflects near-to-far skeleton depth variation), to automatically compute a 3D canvas, employed to freely manipulate the character in 3D (e). 43
4.2 Character contours alone (left) frequently do not provide sufficient information to conclusively determine 3D shape both on occlusion free (top) and partially occluded (bottom) inputs. A 3D skeleton, shown in the insets, resolves much of the ambiguity present in contours alone facilitating plausible shape interpretation. 44
4.3 The canvas (center) of the catwoman in Fig. 4.1: (left) thick black line shows reconstructed 3D contour curves, (right) insets visualize representative trajectories and profiles. 45
4.4 (a) Lacking part information, character shape reconstruction can at best exploit overall shape smoothness, e.g [68, 95]; (b) by using a skeleton to facilitate contour partition and part modeling, we generate a more believable character shape. . . 47
4.5 (a) Perceptual studies indicate that viewers group curves that can be smoothly joined together ((a), right), seeing those as a continuation of one another; while treating those with non-smooth transition ((a), left) as distinct; viewers tend to prefer interpretations that balance part simplicity (b) against contour persistence, preferring interpretations that preserve contour shape under small view changes (c). 48
4.6 Canvas construction: Given a sketch and a skeleton (shown in side view) we first segment the input contours into sections associated with skeletal bones (a), correspondences shown by matching color), correctly resolving occlusions; we use the segmentation to replace the straight-line skeleton by a curved-skeleton optimized for symmetry (b); and finally generate maximally simple body part geometries around this new skeleton while maintaining contour persistence with respect to the input drawing (c). 49

4.7 Skeleton-driven segmentation of a simple contour (a) must match skeletal topology (b) and reflect bone proximity. Proximity alone does not guarantee skeleton matching segment topology (c). A more topologically consistent segmentation (d) may need to be refined by bisector rotation to avoid segment overlap (e). Boundaries are then adjusted to best align with negative curvature extrema (f). 51
4.8 A character drawing with inter-part occlusions contains multiple contour curves and the left and right outlines of a body part may now contain multiple Gestalt continuous segments (a); thus 2D proximity based segmentation is no longer adequate (b). Taking into account skeletal depth as well as 2D proximity but neglecting Gestalt continuity leads to better, but still perceptually wrong results (c,d). Our framework accounts for both considerations resulting in the desired segmentation (e). 52
4.9 Segmentation algorithm: iterating between a z-ordering based pass and consistency validation. 53
4.10 Possible scenarios of contour intersections (filled circles) for rays bounding a mini-bone. Empty circle means the ray has no associated contour intersection. 56
4.11 Curve skeleton computation: (a) user posed straight-line skeleton with the initial trajectory centers and their corresponding trajectory contour points marked; (b,c) front and side views of curve skeleton and 3D contours; (d) final surface with contours highlighted. 59
4.12 Canvas connectivity (a) with close-ups of quad strips between trajectories (b) and triangulated terminal trajectories (c). . . 61
4.13 Connectivity across joints: (a) visually continuous parts; (b) Discontinuous parts; (c) the top part is deemed continuous with both lower ones, while the two bottom parts are deemed discontinuous since their shared contour curve has a cusp between them. 61
4.14 Given the input sketch (a), contour persistence indicates that side view contours (b,c) significantly differing from front-view ones are undesirable. Viewers similarly do not anticipate extreme foreshortening (d). Our result (e) is persistent with the front view contours. 62
4.15 We constrain the profile angle to the range between the ideal profile slope given by the two ring radii and the axis direction. 64

4.16 Ground truth (green) and 3D skeletons created by 3 animators overlaid on two ground truth 2D character drawings (a), (b), also shown from an alternate view overlaid on the ground truth 3D canvas. The skeletons in (b) shown individually (c). The purple and maroon skeletons, created by manipulating an overlaid 2D skeleton have differences in 3D limb length between symmetric limbs. The maximum difference for each skeleton, 14% and 33%, is marked on the longer limb. The brown skeleton was created by animator #2 mimicking the workflow of animator #3. The angular deviation between the corresponding bones on the ground truth and artist skeletons is dominated by control bones (hips and shoulders) which have no impact on the result geometry. The maximal deviations without (and with) control bones are: 24° (31°) for the purple skeleton, 24° (46°) maroon, 32° (44°) brown , and 30° (40°) blue. Average angle differences are 13°, 15°, 15°, and 18° respectively. 67
4.17 Comparing our results to ground truth data: Left to right: contours and skeletons of ground truth (GT) models; GT (blue) and our (yellow) models rendered from alternate views. 68
4.18 Left: Given the same input sketch, small variations in skeleton posing (green and purple Figure in 4.16) lead to minor changes in character shape. Right: significant change in bone slope and location for a symmetric contour leads to larger shape difference. 68
4.19 Comparison of our results (b) to sketches produced by artists (a) for the same view and pose. 70
4.20 Overlaid user segmentations (left) for both the elephant and the scientist are qualitatively similar to the algorithmic results (right). 70
4.21 Canvases and alternate view renders generated using our system from the inputs on the right. 71
4.22 A variety of re-posed renders generated automatically from the inputs on the right. 71
4.23 The explicit cylindrical parameterization of our canvases allows for a range of advanced texturing effects, hard to achieve without a an underlying 3D structure. 72

4.24 Given a single drawing and a posed skeleton we generate qualitatively similar results (b,d) to those created by multi-drawing systems which employ manually specified curve correspondences between drawn curves: [106] (a) and [74] (c). . . 74
4.25 An example of the qualitative evaluation questionnaire. . . . 75
4.26 Our ability to plausibly recover character shape is limited by the descriptive power of the inputs. Without cues to the contrary we generate round bird wings, instead of anatomically correct ones (a). Since we use a standard mesh representation, the canvas can be easily edited to correct the wings or add extra features (beak) using standard tools (a, right). Geometries not-well represented by generalized surfaces of revolution, such as loose clothing (b, pink cape) must be modeled by other means. While some fine details can be captured by using skeleton refinement (c), alternate editing tools are likely to achieve this goal faster. 76

5.1 Gesture3D: gesture drawings (b,e) of an input character model (a); estimated 2D skeleton projections (c,f) and new poses automatically computed from the drawings (d,g). 78
5.2 Stick figure drawings (a), lines of action (b), and outer silhouettes (c) allow for multiple perceptually valid pose interpretations. (d) Poor view selection results in highly foreshortened contours leading to loss of pose information (e.g bends on the left arm or the curved spine). Gesture drawings, consciously drawn from descriptive views (e) effectively convey the intended pose. 81
5.3 Portion of a gesture drawing with annotated joint (blue) and part (red) contours. 82
5.4 Contour-skeleton correspondences, with Gestalt continuous contours connected by dashed lines. 82
5.5 Implausible bone locations that violate (a) adjacency, (b) orientation, or (c) crossing cues; consistent placements (d). . . 83
5.6 Depth ambiguity 84
5.7 Occlusion types. 84
5.8 Less natural (b) and more natural (c) interpretations of a drawn pose (a) (leg bent sideways vs forward). 85
5.9 Overview: (a) algorithm input; (b) discrete 2D joint embedding; (c) optimized 2D embedding; (d) 3D skeleton (color visualizes depth) and posed model. 86

5.10 Joint cost visualization. Here the color shows the matching cost on a scale from red (poor match) to blue (good). 87
5.11 Full solutions: (a) contains overlaps; (b) poor coverage; (c) preferred. 92
5.12 Comparing our results to GT data and artist modeled poses. We use as input the projected contours of the posed GT models combined with their bind posed originals (Figure 5.14) to automatically create poses qualitatively similar to both GT and artist results. 99
5.13 Overlays of viewer created skeleton embeddings (lines removed for clarity) and our results on same inputs. 99
5.14 Typical two-stage processing results. Left to right: input model, drawing, 2D skeleton fitting, output model. 101
5.15 (center) 3D posing using only drawing conformity, (right) full 3D solution. 102
5.16 Impact of different bind poses. 102
5.17 (right) Davis et al.[33] trace stick figures over gesture drawings and then pose characters semi-automatically. (left), We use the original drawings to automatically pose characters. . 103
5.18 Extreme mismatch in proportions between model and drawing (a) can lead to poor depth reconstruction (b); correcting the proportions in the drawing (c) corrects the reconstruction. (d) Ambiguous drawings using highly oblique views can cause our 2D pose estimation to fail. 103

Chapter 1

Introduction

3D modeling is used ubiquitously in manufacturing, entertainment, construction, and its influence is rapidly expanding to health care, education, and other industries. It is next to impossible to find a new building, a new car, or a top-selling movie made without 3D models. At the same time, creating 3D models in modern software packages is time consuming, cumbersome, and requires expert 3D modeling skills. Typically 3D modeling software relies on the use of specialized complex 3D modeling tools (Figure 1.1), which to regular artists are far from natural [83].

Sketching is a natural way to communicate ideas quickly [98]. Sketching is often a first step in making fine art, creating illustrations or cartoons, 3D modeling, etc. The expressive power of sketching along with its accessibility make it a method of choice to visually convey an idea.

A tool for automatically interpreting sketches and creating 3D models, envisioned by the designer of the sketch, could be the Holy Grail of modeling [98]. Such a tool may benefit numerous applications, enabling rapid prototyping, creating animations or models quickly and, perhaps even more importantly, enabling more people to use 3D modeling.

To embark on a quest for the tool, one may look at the core components of a sketch, which are its curves. They are the first elements to be drawn before shading or color, and are essential to conveying geometric information [49]. Interpretation of a sketch is largely through interpretation of its curves.

Unfortunately, while curve drawings are a natural and well understood representation for humans, they are hard to interpret algorithmically [80]. The complexity of an algorithmic interpretation roots in the complexity of the human perception of such drawings, which we are yet to understand. The fundamental problem of algorithmic interpretation is that mathematically, each 2D curve has an infinite number of 3D interpretations [86]. Even if curves are embedded in 3D, there is an infinite number of surfaces passing through the curves. Moreover, sketches typically contain occlusions, ambiguities, and distorted proportions that complicate the matter even further.

Nevertheless, as we discuss in Chapters 3-5, for artist-drawn 2D sketches, human observers tend to imagine unique 3D curves, successfully overcoming

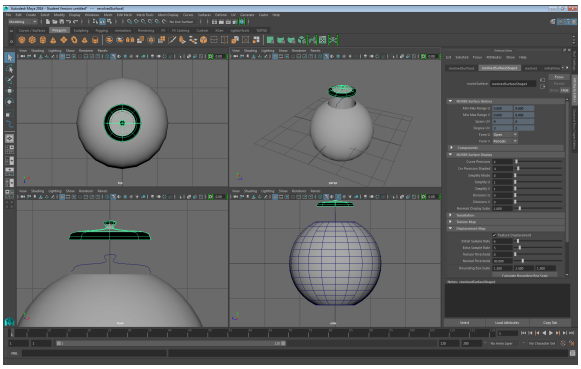


Figure 1.1: Graphical interface of a commercial industry-standard 3D modeling package.

ambiguities and inaccuracies. Furthermore, given such imagined or artist modeled 3D curves, human observers tend to imagine unique 3D shape. The key to understanding why, we believe, is the intent of sketches: they are drawn to convey shape. This intent drives artists to draw a sufficient number of significant curves, unambiguously depicting the object from a non-accidental view. Such an intent is the crucial trait for modeling and perception research with the purpose of discovering the mathematical nature of sketch curves, and therefore give a hope of algorithmic sketch interpretation. Particularly,

- **Significant curves**. Depending on the context, 2D sketches and 3D curve networks typically include ridge/valley lines [26], curves depicting sharp features, and the lines of principle curvature [42, 119]. 2D sketches also include occlusion contours in the selected view.
- **Sufficient number of curves**. While arbitrary 2D curve drawings or 3D curve networks may have vastly different interpretations, artist created sketches typically have enough curves to uniquely convey the imagined 3D object.
- **Non-accidental view**. For 2D sketches, designers tend to choose non-accidental views with few occlusions and least foreshortening [93]. This suggests that observers interpret 2D geometric properties as strongly correlated with 3D geometry rather than being caused by a particular choice of viewpoint [138].

Such observations are the cornerstones of our research, serving as a

base for our algorithms. Additionally, we derive cues on sketch interpretation from well-studied features of human perception. Those features, by large introduced by Gestalt psychologists, include anticipation of symmetry and simplicity (Good Gestalt Principle), alignment and regularity, *et cetera* [132]. In our algorithms, we aim to mimic such behavior.

Apart from information in the sketch itself, human perception of curves also depends on context and prior knowledge [102, 121]. Correctly interpreting sketches requires knowledge of the context and the relevant priors. In our research, each project's scope of applicability defines such context and provides the necessary priors for interpreting curve shape. For instance, an observation that body parts of most characters can be represented as surfaces of revolution, provides missing necessary information when interpreting a character sketch (Section 4). In other words, prior information of this kind allows us to approach problems that otherwise would be ill-defined.

To summarize, this thesis is focused on using specific prior knowledge, findings in perception research, and insights from artist literature to algorithmically interpret sketched objects, in the areas of direct practical applications. To validate our results, we compare them with the ones manually created by professional artists, and, whenever possible, compare our algorithmic choices with human ones.

1.1 Overview of Contributions

The contributions of the dissertation can be split into two main categories. First, in each chapter we distill the artistic knowledge of the area, along with cues drawn from perceptual evaluations, into a set of principles that guide the interpretation of curve drawings. And second, we use those principles to build systems allowing us to algorithmically interpret curve drawings within a particular domain. We then confirm our analysis and intuition with various evaluations and direct comparison with artists' results.

Thus, we introduce three novel systems for algorithmically interpreting sketches and recovering the depicted 3D shape. Our work encompasses two separate shape domains: CAD objects (Chapter 3) and characters (Chapters 4 and 5).

- In Section 1.2 and Chapter 3, we introduce our first contribution, an approach to automatically generate 3D surfaces of CAD objects from artist-drawn 3D curve networks. Such an approach complements 3D curve sketching interfaces such as ILoveSketch [7] or systems for lifting 2D sketches into 3D [138] to rapidly create CAD models.

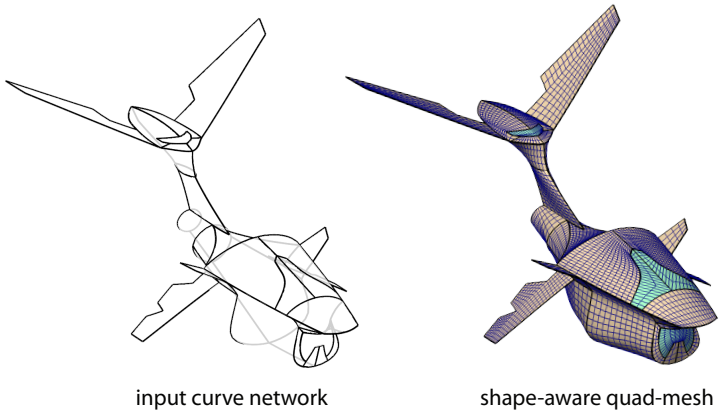


Figure 1.2: A brief overview of Chapter 3.

- In Section 1.3 and Chapter 4, we introduce a new technique to recover 3D character shape from a 2D cartoon drawing and a correspondingly posed 3D skeleton. We demonstrate that this system can be used to create believable 3D models from a single drawing and a 3D skeleton, sidestepping the tedious 3D modeling step.
- In Section 1.4 and Chapter 5, we introduce a system to pose a 3D character directly via a gesture drawing. This system is intended to replace the tedious and cumbersome rig-based posing process in a traditional 3D modeling package.

1.2 Recovering Artist-Intended Surfaces from 3D Curves

Advances in sketching interfaces enable artists to directly draw early concept sketches in 3D while following their pen-and-paper drawing style, creating 3D curve networks [7]. Such curve networks are known to effectively convey complex 3D shape [87], and, if drawn by an artist, typically convey the shape unambiguously. In Chapter 3 we introduce the first solution to constructing the imaginary surface interpolating a general 3D design curve network, consistent with artist intent.

To approach the informal notion of artist intent when defining a surface, we derive our insights from ideas in 3D modeling and perception literature. Namely, we observe, based on design literature, that the artist-drawn 3D

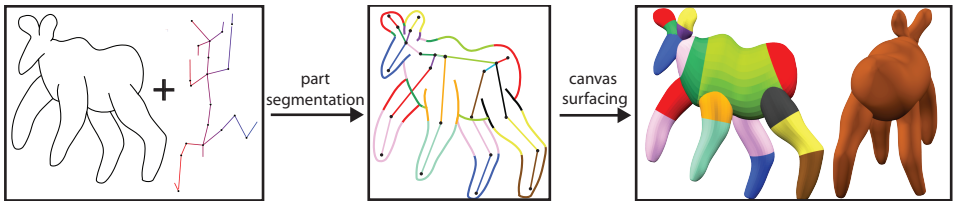


Figure 1.3: A brief overview of Chapter 4

curves can be treated as *representative flow-lines*, an input-derived sparse set of curves, correlated with lines of curvature. We further observe that viewers complete the intended shape by envisioning a dense network of smooth, gradually changing, flow-lines that interpolates the input curves. Components of the network bridge pairs of input curve segments with similar orientation and shape.

Consequently, we introduce the novel algorithm that mimics this behavior by iteratively segmenting and matching the input curves, and then uses the matching to effectively construct an interpolating surface consistent with artist intent (Fig. 1.2).

1.3 Recovering Character 3D Model from a Cartoon Drawing

Traditional 2D cartoon characters are a mainstay of computer animation. Viewers appreciate the feel of hand-drawn art, while animators enjoy the flexibility and explicit control offered by this medium. This flexibility, unfortunately, comes with the tedium of drawing numerous individual frames, and the cumbersome burden of managing view and temporal coherence. Recent research [112] and practice [104] advocate the use of an underlying 3D model to enable easy 3D control over the view, pose, deformation and painterly rendering effects of cartoon characters. In current animation practice, such models are manually constructed using 2D cartoon drawings as a visual reference, and are then manually rigged to suitably designed skeletons for posing and animation. In chapter 4 we introduce a novel technique for the construction of a 3D character model directly from a 2D cartoon drawing and a user-provided correspondingly posed 3D skeleton, enabling artists to directly articulate the drawn character in 3D.

We observe that traditional cartoon characters are well approximated by a union of generalized surface of revolution body parts, anchored by a

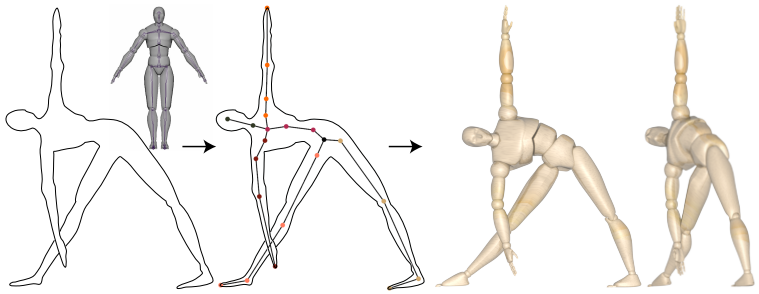


Figure 1.4: A brief overview of Chapter 5.

skeletal structure, which motivates our choice of input. We also observe that while typical 2D character contour drawings allow ambiguities in 3D interpretation, our use of a 3D skeleton eliminates such ambiguities and enables the construction of believable character models from complex drawings. We analyze and distill the insights on the nature of character sketches from perception and art literature, namely, we explore the principles of Gestalt continuity, simplicity, and contour persistence.

Beyond this analysis, our contribution is the method based on those insights that enables generating 3D character models from curve drawings. The core of the method consists of two algorithms: first, the algorithm of body part delineation that segments the input 2D contours into sections outlining individual body parts and resolves inter-part occlusions; and second, the algorithm that imbues the outlined contours for each body part with depth and creates the 3D model, balancing perception cues, image fidelity, and shape priors (Fig. 1.3).

1.4 Recovering 3D Character Pose from a Gesture Drawing

While posing 3D characters is a common task in digital media production, performing it using traditional 3D interfaces is time consuming and requires expert 3D knowledge. Alternative approaches which use stick figures or lines of action as a posing reference are problematic, since these representations are inherently ambiguous even to human observers.

In contrast to these representations, gesture drawings - rough yet expressive sketches of a character's pose - are designed by artists to facilitate a single perceptually consistent pose interpretation by viewers. Artists are

skilled at quickly and effectively conveying poses using such drawings, and use them ubiquitously while storyboarding. Actual animation, however, is typically done by manual manipulation of the skeleton, and those drawings are often used only as a reference [83]. In Chapter 5 we introduce the first method to pose a 3D character directly via a single vectorized gesture drawing as the only input.

The contribution of the chapter is two-fold: we formulate the properties of effective gesture drawings, bringing together insights from various fields, such as psychology, art, and computer graphics, highlighting key perceptual cues which enable viewers to perceive the artist intended character poses; we then use these observations to introduce the first gesture drawing based algorithm for posing 3D characters. Our method enables artists to directly convert their ideated posed character drawings into 3D character poses, and supports complex drawings with occlusions, variable body part foreshortening, and drawing inaccuracies (Fig. 1.4).

Chapter 2

Previous Work

2.1 Sketch-based 3D Modeling

Algorithmic sketch interpretation has a long history rooted in computer vision and artificial intelligence research [25, 59]. Those early works focused on providing semantic descriptions of elements in the drawing, rather than a complete 3D model [79, 129]. For instance, Huffman [59] and Clowes [25] studied the problem of line-labeling of polyhedra drawings. Waltz [129] extended the line labels idea to include shadows, grouping lines into bodies. Mackworth's work [79] deals with 'naturalistic' drawings of sketch maps, classifying lines into shorelines, rivers, roads, and mountains.

The research question of how to infer 3D model from a line drawing emerged not long after 3D modeling itself [86]. The review literature [98] often name 3D Paint by L. Williams [135], Teddy by Igarashi et al. [60], and SKETCH by Zeleznik et al. [140] the first sketch-based 3D modeling systems.

2.1.1 Incremental Approaches

Those early methods were incremental, i.e. complex 3D shapes were modeled via a sequence of simple operations. They did not aim to interpret natural drawings; instead, they provided interfaces with a set of sketch-based modeling operations to create a limited variety of shapes: geometric primitives in SKETCH [140] or 'inflation' surfaces in Teddy [60].

As incremental approaches developed and matured, the range of surfaces they could model expanded [90], and their input method gradually transitioned closer to the pen-and-paper drawing process [99].

We can categorize the more recent methods into single- and multi-view incremental approaches. A more extensive review of related literature can be found in the survey by Olsen et al. [98], here we focus on the works most relevant to our thesis.

Multi-view. Many systems employ a multi-view approach: users build models by drawing contour strokes in views where they are expected to be parallel to the screen, or to project with little foreshortening onto the evolving geometry [95, 123]. Frequent view changes and incremental drawing order are critical when modeling characters using such approaches, as it is next to impossible for all contours of a 3D model to be entirely flat (Fig. 2.2). Borosan et al. [17] proposed simultaneously creating and rigging 3D characters using an interface where body parts are added incrementally, one at a time, and the associated skeleton is generated on the fly. Our systems introduced in Chapters 4 and 5 are independent of drawing order; they allows artists to freely sketch the characters they envision and to interpret legacy sketches without need for oversketching.

The general issue with incremental approaches, however, is that the result strongly depends on the order in which the artists draw strokes, which makes interpretation of complete natural drawings highly unlikely. Similarly, choice of views to draw may not be an easy task for a user. Moreover, they often require additional user input, such as annotations, correct drawing order, etc.

Single-view. Single-view incremental modeling approaches, such as the ones by Cherlin et al., Gingold et al., Shtof et al., and Chen et al. [21, 23, 46, 118] rely on additional information to facilitate modeling of complex shapes. Namely, some methods [99], for the purpose of modeling smooth shapes from existing drawings and photographs, leverage contour drawing order and user annotation; others, such as Shtof et al. [118] and Chen et al. [21] snap parameterized primitives to input contours via a combination of optimization and user annotation, and rely on user assistance to generate 3D models from annotated sketches and photographs respectively. Gingold et al. [46] interactively place tubular and elliptical 3D primitives to match artist drawn contours; as they note, their system does not directly use the 2D image. Cherlin et al. [23] treat each new pair of contour strokes as 2D profiles defining a new generalized surface of revolution part, whose trajectory is either circular or manually defined.

2.1.2 Shape and Pose Reconstruction from Photographs and Video

In the meantime, in computer vision, a very relevant problem was being solved – shape and pose reconstruction from monocular images and video (for historical review see [3]). Compared to sketch-based modeling, much

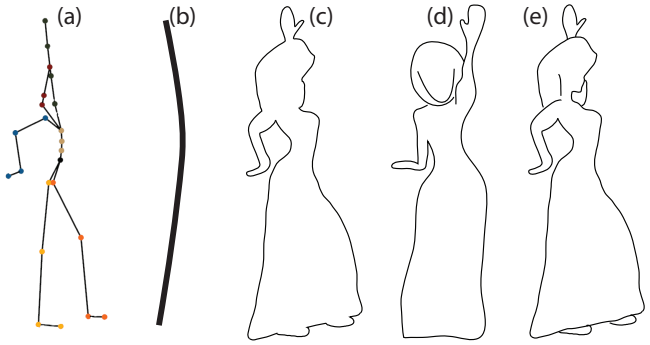


Figure 2.1: Stick figure drawings (a), lines of action (b), and outer silhouettes (c) allow for multiple perceptually valid pose interpretations. (d) Poor view selection results in highly foreshortened contours leading to loss of pose information (e.g bends on the left arm or the curved spine). Gesture drawings, consciously drawn from descriptive views (e) effectively convey the intended pose.

more data is accessible here, though the range of shapes, poses, and motions is severely limited compared to non-realistic animation demands.

A variety of recent multi-view reconstruction methods model human subjects from silhouettes, potentially aided by skeletons [110, 139]. They work with large collections of silhouettes captured from a range of views and poses [92]. The main drawback of this class of methods is that a large number of input drawings is very unlikely to be produced by hand; instead, those methods aim at reconstructing from a series of photographs or video sequences.

Reconstruction from video aims to capture a continuous motion, where the pose in each frame is very close to a previously reconstructed pose in the preceding frame, and heavily relies both on this existing previous pose and on fine image-level correspondences between frames (e.g. [34, 44, 125]). Our thesis has more in common with pose estimation from a single frame, or *pose initialization*, where no such priors are available (e.g. [22, 43, 61, 111]). As we show in Chapter 5, both outlines and incidental-view occlusion contours (Figure 2.1c,d) are insufficient to deduce a pose; single-frame pose estimation methods therefore frequently combine this information with textural and shading cues which are unavailable in our setup.

Recent posing approaches (e.g. [61]) predict the most likely 3D pose by learning from large databases of real and synthetic human motion data. Such databases bias the results toward more frequent poses and can be difficult

to obtain for non-humanoid or non-realistic characters, or for extreme/non-physical poses. The frameworks introduced in the current thesis overcome the lack of extensive anatomic pose priors, and allow recovery of atypical character shapes and poses by leveraging the descriptive cues artists provide when creating gesture and concept drawings.

For a more complete recent review of this area please refer to [125] and the references therein.

2.1.3 Character Shape Reconstruction from Complete Drawings

Incremental approaches allow users to provide guidance and control over the modeling, and, in general, allow for more varied user input. However, in an incremental framework, the input method is often not natural, and reusing existing drawings is hard, if even possible. In contrast, methods for recovering 3D shape from complete drawings are aimed at using existing natural art as input. These methods allow us to preserve the standard 3D model development process that often starts with sketches.

These methods can use either a single drawing to recover 3D shape, or multiple drawings. Using multiple drawings, artists have the freedom to specify some details or shape features invisible from a single view. At the same time, using these methods typically requires more user input to specify correspondences between curves on different drawings. Single-drawing methods are aimed at quick modeling and so typically don't require extra user input; instead, they often rely on simplifying assumptions about the depicted 3D shape.

Using Multiple Complete Drawings. A number of methods, such as the ones by Fiore et al., Rivers et al., Jain et al., and Levi and Gotsman [41, 65, 74, 106] use collections of vector character drawings taken from different views to create a 3D shape proxy or enable direct rendering from in-between views. The biggest problem this approach has is finding the correct correspondences between curves in different drawings. Those methods rely either on user-annotated dense curve correspondences in between the drawings [41, 106], or manually specified correspondence between each individual drawing and a user-positioned 2D [65] or 3D [74] skeleton. The methods require at least three strategically posed drawings to achieve acceptable results. In contrast, our method in Chapter 4 generates reposed character renders that are qualitatively comparable to renders produced by

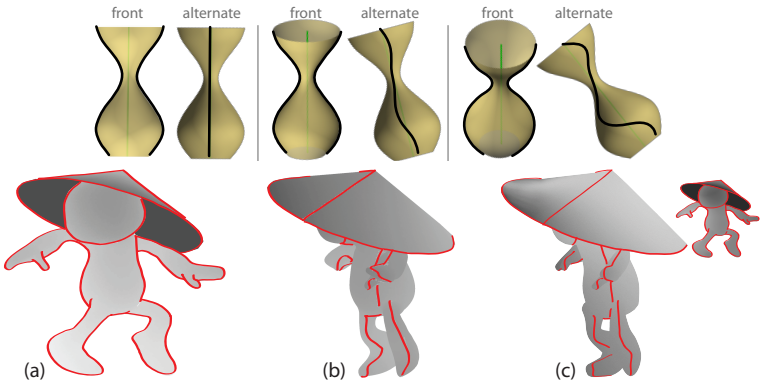


Figure 2.2: Character drawings do not conform to the assumptions made in previous work. (top) The contours of a surface of revolution whose axis is not in the view plane are typically not planar.(bottom) The contours of a typical character include numerous occlusions; a single contour curve can consist of multiple part outlines (see left arm and torso outline in (a)) and as shown by the side view (b) the contour curves are far from planar or view aligned. Our method introduced in Chapter 4 (c) successfully handles such inputs generating a character model similar to the ground truth input (b).

these methods from a single, descriptive drawing and a matching skeleton with no additional annotation (Fig. 4.24).

A related line of work is in the area of inbetweening [134], where the task is to interpolate the motion of a character between given frames. The geometry of a character is assumed to stay roughly the same throughout the animation, and only the view and the pose might change. For complex cases, these approaches also rely on user-annotated curve correspondences between different frames.

Using a Single Complete Drawing. A range of methods attempt to recover character models from single view sketches with no extra input [20, 28, 37, 68]. However, in doing so they by necessity enforce a range of strong simplifying assumptions. In the domain of character models, Buchanan et al. [20] lift an occlusion-free 2D contour into 3D by placing circular arcs along a 2D geometric skeleton; they assume the entire contour to be planar and near-perpendicular to the view direction. Cordier et. al. [28] lift contour drawings of reflectively symmetric and symmetrically posed characters into 3D. They expect every part contour to be planar, and expect each part to be represented as a separate curve in the drawing. Karpenko

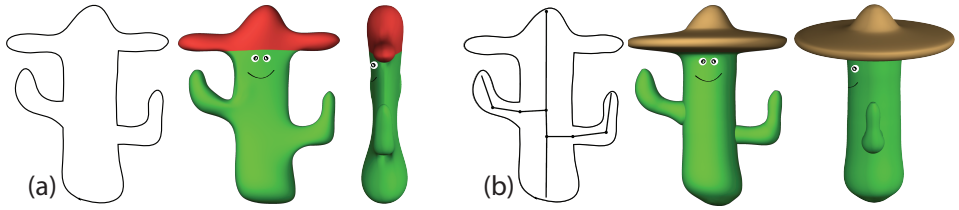


Figure 2.3: (a) Lacking part information, character shape reconstruction can at best exploit overall shape smoothness, e.g [68, 95]; (b) by using a skeleton to facilitate contour partition and part modeling, we generate a more believable character shape.

and Hughes [68] successfully process character drawings containing partial occlusions and asymmetric poses, but assume that each contour curve is planar and perpendicular to the view direction. Lacking part structure, they cannot leverage geometric priors on individual body-part shape and use surface inflation (Fig. 2.3) to generate the outputs. Entem et al. [37] model animals from a canonical side-view sketch and rely on T-junctions to segment the contours into separate part outlines. They assume all contour curves to be planar and perpendicular to the view direction, and only handle local T-junction type occlusions between immediately adjacent body parts.

The assumptions listed above do not hold for the vast majority of articulated character drawings: these drawings frequently contain general inter-part occlusions, individual contour curves frequently extend across multiple body parts, these parts are rarely perfectly symmetric, and part contours are rarely planar (Fig. 2.2).

Research in this category is related to our work presented in Chapters 4 and 5, however, we do not impose such simplifying assumptions on the 3D shape.

By leveraging the additional information provided by the 3D skeleton, our method in Chapter 4 successfully relaxes all of these assumptions and is able to handle inputs such as 'sneaky' (Fig. 2.2 (bottom)) or the catwoman (Fig. 4.1, 4.3) which repeatedly violate them.

Sýkora et al. [122] use user annotation to recover a bas-relief with approximate depth from a single sketch which they use to illuminate a 2D drawing. Their method makes no effort to create a 3D canvas that is plausible from all views; as they note, their proxy meshes "expose their approximate nature when rendered from sideviews using a perspective camera".

In CAD domain, a variety of methods, e.g. one of Xu et al. [138] or one of Lipson and Shpitalni [78] (see [138] for full references) infer 3D curves from

a 2D sketch of a CAD model. Our research in Chapter 3 complements these works by inferring the artist-indended surfaces, thus allowing the creation of a complete 3D model.

2.1.4 Skeleton-based 3D Modeling

Another alternative approach to modeling 3D shapes is skeleton-based modeling. Organic 3D forms created using implicit functions defined around interactively manipulated skeletal primitives have existed for at least two decades [13]. Recently, [8, 17] proposed to simultaneously create 3D shapes and corresponding skeletons as a means to integrate skeletal deformation and interactive shape sculpting. While those methods do not require a skeleton as an input, neither framework can incorporate a complete 2D drawing into the modeling process.

2.2 Posing Characters

Posing characters can be done either with explicit 3D model, or via 3D-like effects on 2D sketches.

2.2.1 Adding 3D Effects to 2D Drawings

Existing 2D animation tools support a limited range of 3D effects. They enable occlusions via explicit layering [2, 4, 58] and approximate out-of-plane deformation using non-uniform scaling that mimics foreshortening [64]. These approaches use a fixed 2D contour topology and are inherently unsuitable for generic 3D manipulation which requires topological changes in character contour and reveals *a priori* occluded geometry (see Fig. 4.1(e)).

Recent industry-driven research (e.g. [104]) aims to enhance hand-drawn animation with 3D effects such as volumetric textures [9, 113], or cloth simulation [65], by utilizing separately created 3D models or proxies in the background. In Chapter 4, we produce the underlying 3D proxy required by these techniques using a single 2D cartoon frame and an appropriately posed 3D skeleton as input. Our problem formulation is a novel intersection of skeleton-driven 3D modeling, sketch-based single-view modeling, and 3D character construction.

2.2.2 3D Character Posing

Sketch Based Articulation Rather than creating a model from scratch, methods such as [71] deform a 3D character template to fit a contour drawing. They either expect the template and drawn poses to be aligned, or expect users to manually specify coarse template-drawing correspondences. They then use local shape compatibility between the input outlines and the corresponding 3D geometry to obtain dense correspondences. Since contours in gesture drawings are approximate and highly abstracted, local shape compatibility cannot be used as a reliable criterion in Chapter 5. Despite this extra challenge, our method does not require manual correspondences nor expects the drawn pose to resemble the input bind one.

Character Posing Interfaces In most industry setups, characters are posed via 3D skeleton manipulation. Users either manually adjust joint angles, or use Inverse Kinematics (IK) based tools to place bone end-points at specific locations [141]. While IK-based frameworks relieve some of the tedium of adjusting individual joints, they still require experience with 3D modeling systems and non-trivial posing time.

Recent research demonstrates effectiveness of alternative posing approaches, such as handles [63], selected regions and exterior cages [66, 130], or animation devices [47]. Handle and cage based approaches focus on expanding the space of possible deformations, while animation devices focus on reducing amount of work needed. However, when artists ideate their desired poses they prefer to use pen and paper. Using these ideation drawings as-is to create 3D poses saves artists time and effort.

Hahn et al. [51] and Guay et al. [48] propose incremental, multi-view, sketch-based posing interfaces. Lines of action, imaginary lines running down a character's spine or other major bone chains (Figure 5.2b) are used by artists for coarse pose communication [48]. Guay et al. use line-of-action strokes to pose characters by placing user-specified corresponding bone-chains along these strokes.

This input allows multiple pose interpretations for body parts not directly present on the line of action or its continuation, and requires an incremental multi-view interface to pose non-coplanar bone-chains. Hahn et al. [51] propose an interface where a user poses characters one limb at a time, by first drawing a stroke along a limb in the current pose and then drawing a corresponding stroke depicting its new pose. The system then poses the limbs by aligning them to the strokes. It assumes uniform foreshortening along the posed limbs, and requires multiple stroke pairs and view

changes to generate complex poses. Our work in Chapter 5 complements these approaches by providing a single-view drawing-based posing mechanism, allowing artists to directly use their gesture and keyframe drawings for character posing.

A number of recent methods use stick-figures [33, 54, 77, 82] - 2D projections of the desired 3D skeleton of the posed character (Figure 5.2a) - to compute a corresponding 3D skeletal pose As the authors acknowledge, stick-figures are inherently ambiguous and allow for multiple geometrically valid and perceptually plausible 3D interpretations. Hecker and Perlin [54] and Mao et al. [82] propose users to encode the relative depth of bones and joints via pen pressure or stroke width. Such interfaces become unwieldy for typical characters (e.g. Figure 5.1) which have dozens of bones. Davis et al. [33] resolve ambiguities through user annotation, followed by users selecting the desired character pose from multiple plausible solutions. Lin et al. [77] use stick-figures to pose characters sitting in a chair, and reduce ambiguities by using specific priors relevant only for sitting characters. Wei et al. [131] and Choi et al. [24] use drawn stick-figures to query a database of human poses. Such databases can be difficult to obtain for custom skeletons, especially of non-humanoid or non-realistic characters. Reliance on databases inherently biases the reconstructed poses toward more frequent database instances. In contrast to stick figures, gesture drawings are unambiguous to human observers, motivating our approach. At the same time while matching 2D stick figures to 3D skeletons is straightforward up to inherent ambiguity between symmetric limbs, matching characters to gesture drawings is an open and challenging problems we successfully address for the first time. Small inaccuracies in 2D stick-figures can lead to large changes in the recovered 3D pose [33]. To improve accuracy Davis et al. [33] advise artists to first draw a gesture or bubble sketch of the target posed character, and then use it to assist in positioning the stick-figure (Figure 5.17). Our work in Chapter 5 operates directly on gesture drawings and robustly overcomes artist inaccuracies by balancing image conformity against other perceptual cues (Figure 5.15).

2.3 Surface Reconstruction from 3D Curve Networks

One can create 3D curve models of CAD objects via a variety of tools, e.g. via a sketch-based interface [7], or by lifting a 2D sketch into 3D [138]. Those tools, however, are capable of creating curves only. The problem is therefore

to find the interpolating surface envisioned by the designer, given such curve networks.

Early approaches aimed to reconstruct polyhedra, given 2D or 3D straight line drawings [85, 133]. In contrast, inferring a smooth surface from a set of 2D/3D freeform curves is a more ambiguous and challenging task.

There is a large body of work on interpolating closed curves, or *cycles*, with smooth surfaces, much of it in the context of hole filling [81]. While a small portion of the methods [1, 32, 40, 75, 95, 107] can operate on arbitrarily shaped curves, the majority assume that the curve is pre-segmented into n sub-curves and can be mapped to a planar n-sided polygon with little distortion, e.g. [27, 45, 128].

Fitting to n-sided curve cycles A variety of popular techniques are available for interpolating and approximating networks of regular quad or triangular patches [38], see [100] for a recent sketching motivated approach. These methods, including the well-known Coons patches [27], and their discrete extension [39] provide an effective solution. We show in Chapter 3 that unlike other fitting approaches, these are widely used by modelers and designers as the resulting surfaces closely reflect designer intent.

For cycles with $n > 4$ existing approaches can be classified into single surface fitting, e.g. [45, 128], or subdivision into quad or triangular cycles, e.g. [94, 112]. The first category of methods interpolate the cycles with a single surface patch by utilizing suitable n-sided convex 2D polygons as parameter domains. As acknowledged by Varady [128] the fitted surface quality is strongly dependent on the quality of the 2D parameterization.

Subdivision approaches, e.g. [94, 112], quadrangulate the input cycles, and then use available techniques to interpolate or approximate the resulting quad network. In the basic midpoint scheme a single vertex is placed in the center of a patch and then connected to the middle of each side. To generate a watertight surface across heterogeneous networks, Schaefer et al. [112] and Nasri et al. [94] introduce more sophisticated quadrangulation schemes that maintain a fixed number of intervals, or sub-segments along each side while aiming to control both the number and valence of the added extraordinary vertices [94]. Note that all these approaches require n to be specified by the user, which may not be evident from the topology of the network.

A variety of techniques are available for interpolating and approximating networks of regular quad or triangular patches [38], see [100] for a recent sketching motivated approach. These methods, including the well-known Coons patches [27], and their discrete extension [39] (Figure 2.4, (d)) pro-

vide an effective solution which naturally aligns the surface iso-lines with the flow-line sequences indicated by the boundary curves. Design and perception literature indicate that designers expect the curve cycle boundaries to correspond to representative flow-lines implying surface curvature directions, a behavior captured by Coons interpolation (Figure 2.4, (d)), but not the other fitting approaches. These methods are widely used by modelers and designers as the resulting surfaces closely reflect designer intent.

Surface Fitting to Arbitrary Curve Cycles These more generic approaches typically utilize a diffusion process that optimizes surface fairness. As we show in Chapter 3, those traditional approaches fail to capture designer intent on structured inputs, even if supplied with pre-defined normals along the input curves[75, 89]. Moreover, such normals are not part of a typical curve-based modeler output [7, 95, 115]. Other approaches impose very strict constraints on the inferred surface, e.g. developability, often incompatible with artist intent [107]. This approach is too restrictive for a general modeling setup, where many inputs, including the cycle in Figure 2.4 (a), aim to convey non-developable surfaces.

Quad Meshing: Our work draws on ideas from coarse-to-fine planar meshing approaches, such as sub-mapping [101, 108]. In contrast to those it supports irregular quad connectivity, automatically introducing irregular interior vertices when warranted by the boundary shape (e.g. Figure 3.10). More significantly it operates on 3D curves, without the benefit of a well defined planar domain. While planar meshing methods focus on element quality or shape, our goal is to recover and quadrangulate a surface enclosed by designer-drawn curves.

Many recent publications addresses quad meshing of existing 3D surfaces [14, 15, 30, 67, 76, 84, 127]. These methods aim to align the output quad meshes with the principal curvature directions in anisotropic regions generating smooth orthogonal families of flow-lines. In our setup no underlying surface is available. Instead we aim to align the output meshes with the flow-line directions conveyed by the input designer curves, which as noted above strongly correlate to curvature lines.

As shown by figure 2.5 (top) using the actual shape of the curves to determine the end-point locations and induced topology as done by our method in Chapter 3 can significantly improve both the flow line layout and the resulting surface shape. Contrary to all the approaches above our method can operate on curve cycles with large concavities (Figures 2.5 (bottom)).

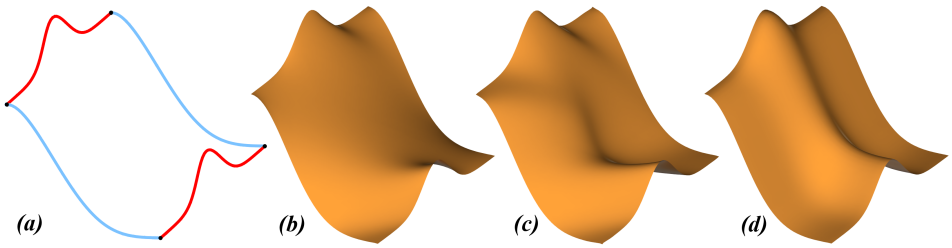


Figure 2.4: Using Laplacian diffusion (b) or Thin-Plate Splines [40](c) to surface a four-sided cycle leads to unintuitive results. (d) In contrast the flow lines on an interpolating Coons patch, by construction, bridge opposite cycle sides.

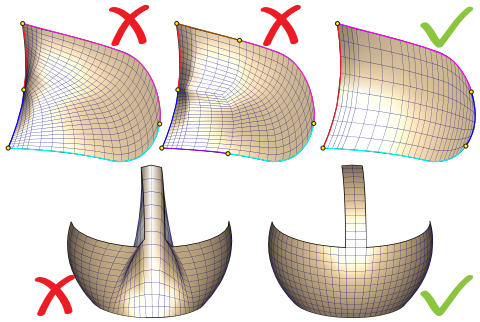


Figure 2.5: (top) Using a purely topological approach and applying midpoint subdivision (forming either four or six sides) generates a quad mesh with poor flow line layout (left and center). Our method in Chapter 3 (right) uses geometry driven segmentation and matching to generate smooth flow lines and a predictable surface. (bottom) On a concave cycle, parameterization onto a convex domain (a rectangle) leads to foldovers (left), our method automatically segments the cycle into convex quadrilaterals leading to a fair surface (right).

Chapter 3

Design-Driven Quadrangulation of Closed 3D curves

3.1 Introduction

In this chapter we present our first contribution, Design-Driven Quadrangulation from Closed 3D Curves. Here we introduce a new approach to creating surfaces interpolating closed 3D curves created by sketch-based or other curve modeling systems. The project has since been published in ACM Transactions on Graphics [11].

Sparse networks of closed 3D curves are the foundation of shape in both traditional CAD modeling [39] and increasingly popular sketch-based modeling interfaces [7, 95, 115]. As we mentioned in the introduction of the dissertation, recent research affirms that such 3D curve networks do effectively convey complex 3D shape [35, 88, 89] (Figure 3.1 (a)). We aim to recover and compactly represent this conveyed shape (Figure 3.1 (f)), for designer-drawn curve networks,such as those generated by Abbasinejad et al. [1] from sketched 3D curves [7].

While arbitrary 3D curve cycles have highly ambiguous interpolating surfaces (Figure 3.2 (top)), designer created curve cycles, even when highly complex, typically convey a uniquely imagined surface (Figure 3.2 (bottom)). These curves are designed to serve as a visual proxy of the 3D object, with the expectation that every element of surface detail is explicitly captured by the network [42]. To this end, design texts repeatedly emphasize the significance of using *representative flow-lines* of the object [16, 42], as curve network elements. While design literature provides no precise mathematical definition of flow-lines, design and modeling references [42, 119] suggest that flow-lines are strongly correlated to sharp features and lines of curvature but allow for artistic license at surface discontinuities, over fine details and in umbilic regions.

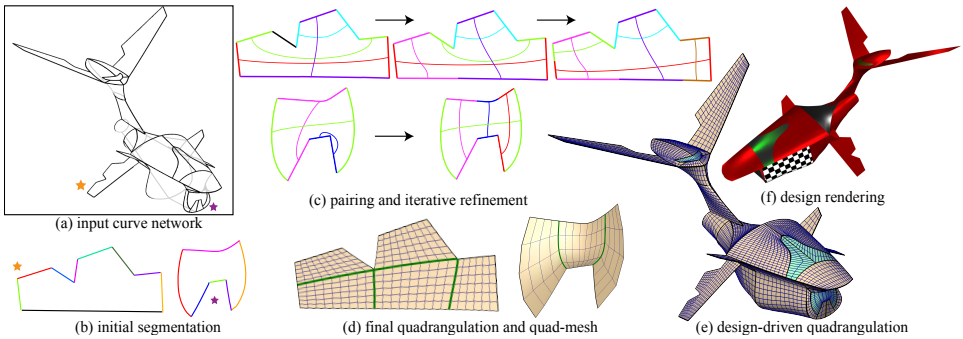


Figure 3.1: Steps to quadrangulating a design network of closed 3D curves (a) : Closed curves are independently segmented (b) and iteratively paired and refined to capture dominant flow-lines as well as overall flow-line quality (c); final quadrangulation in green and dense quad-mesh (d); quadrangulations are aligned across adjacent cycles to generate a single densely sampled mesh (e), suitable for design rendering and downstream applications (f).

These observations, confirmed by perception studies [120], suggest that any additional flow-line on the surface must be expressible as a blend of the explicitly defined flow lines on the designer-created curve cycles. Viewers complete the intended shape by envisioning a dense network of such blended, gradually changing flow-lines. An examination of artist-drawn dense networks (e.g. Figure 3.3) confirms this observation; moreover, artists take advantage of this property by implicitly pairing opposite representative flow-lines, and constructing curve sequences that smoothly evolve from one input flow-line to its mate along the interior surface. The resulting surface is described by the union of these sequences, and forms a quad-dominant mesh. Consistent with this examination, popular CAD tools capture the geometry of four-sided curve cycles using Coons patches [27] (Figure 2.4 (d)), whose iso-lines implicitly define a sequence of flow-lines that bridge the opposite sides of each cycle.

Our surface-fitting algorithm aims to replicate this behavior. Before formally describing the algorithm, we introduce some terminology. A *3D curve network* is a graph of connected 3D curves, where one or more curve cycles have been marked for surfacing by the designer. A *quadrangulated curve network* (left, black) requires that all curve cycles marked for surfacing be four-sided. A single *quad-mesh* can be created from quadrangulated curve network cycles by sampling parametric four-sided patches. The *dual* of a quad network is a graph whose vertices correspond to quad cycles, and whose edges correspond to shared cycle sides. Each dual *poly-chord*, drawn

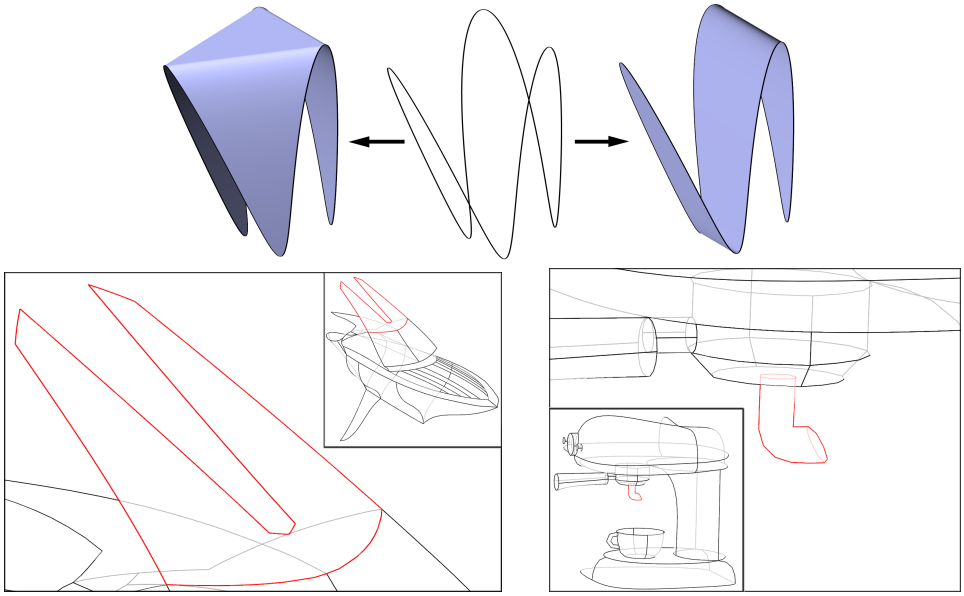
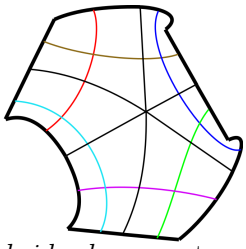


Figure 3.2: Closed 3D curves: ambiguous hexagonal 3D curve (top) compared to complex curves with a clear design intent (bottom).

on the left in a different color, is a sequence of dual edges that corresponds to a chain of quadrilaterals sharing opposite sides [31].



If we can extract the flow-line pairings that artists use, we can then reconstruct the surface in a natural manner using a dual quadrangulation approach, described below. The grand challenge, therefore, is in obtaining a suitable segmentation of a curve cycle into pairs of matching opposite flow-lines. As we expect internal flow lines to change smoothly and gradually, these *bridged* segment pairs should have similar orientation and shape. When examining artist generated flow-networks, we observe that the preference for pairing segments becomes more pronounced as the degree of compatibility between them increases, often at the expense of sub-optimal pairing of other segments. This effect is evident in the highlighted regions in Figure 3.3. On the left, the strong preference for the blue pair enforces the far less obvious red one. On the right, the dominant yellow and blue matches enforce the far less attractive purple one. Such dominant preference order can be formally described as a *stable matching*, where a matching is considered stable when

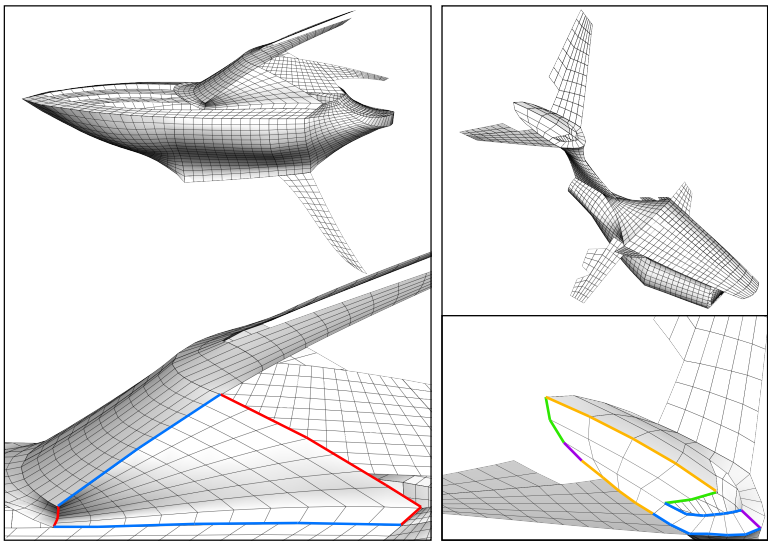


Figure 3.3: Artist designed interpolating quad-meshes.

there are no two elements that prefer each other to their current match [62]. We simultaneously compute the segmentation and its corresponding stable pairing using a tailored discrete optimization strategy which interleaves matching and segmentation steps.

Given the computed segmentation and pairing (Figure 3.1 (c)), we construct a network of quadrilateral cycles (Figure 3.1 (d)) whose dual polychords connect the matched flow-line curve segments and interpolate those with tensor-product surface patches. Using this construction, the iso-lines of the patches naturally align with the matched curves, forming a dense flow-line network conveying the intended surface. An arbitrarily dense quad-mesh describing the target shape is then created by tracing patch iso-lines (Figure 3.1 (f)).

We demonstrate the quad meshes created by our method on a variety of challenging inputs, including both synthetic models and curve networks created by different modeling softwares, comparing our outputs against those manually created by design professionals (Section 3.4).

Contribution:

The main contribution of the chapter is the first solution to constructing the imaginary surface interpolating a general 3D design curve network. We represent this surface using a quad mesh whose iso-lines capture the design flow inherent in the network. Lacking a mathematical model of human per-

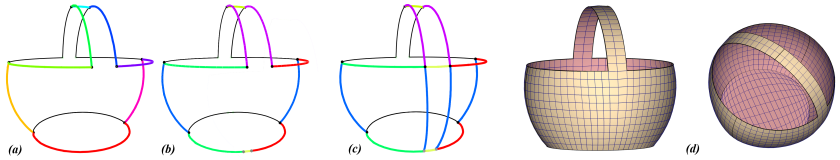


Figure 3.4: After the initial segmentation (a), we alternate matching and refinement steps to obtain a pair-based curve segmentation which is converted into a quadrilateral network (c) . To minimize T-junction count (d) we compute global interval assignment, and use it to sample iso-lines on discrete Coons patches.

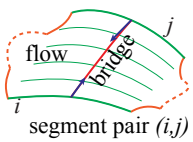
ception, we distill perception studies and guidelines from design literature into a mathematical formulation of flow-line matching and segmentation. We evaluate this formalism by showing results that match both viewer expectation and artist created surfaces.

Our key technical innovation is a simultaneous segmentation and pairing algorithm that locates suitable end segments for the dual poly-chords of the interpolating quad mesh based on analysis of the input curve geometry.

Quad-remeshing techniques often strive to generate rectangular quad elements. We note that our primary objective is to capture flow-lines; since these lines are often related to lines of curvature, we will typically generate well-shaped quads. However, when flow-lines conflict with quad orthogonality, we focus on capturing the flow at the expense of irregularly shaped quads (see Figure 3.1). This ensures that our output is consistent with designer expectations (Figure 3.3).

3.2 Quadrangulating a Closed 3D Curve

This section describes our approach for quadrangulating the interior of a closed curve such that the iso-lines induced by the 4-sided curve cycles capture designer intended flow-lines. The extension of this method to networks of curves is discussed in Section 3.3. We use a dual based quadrangulation approach, where we first compute the dual graph of the quadrangulation (Section 3.1), and then use it to induce the primal quad connectivity and geometry (Section 3.2). This workflow is illustrated in Figure 3.4.



To assemble the dual, we segment the input curve into a small number of matching segment pairs that serve as opposite ends of dual graph poly-chords and corresponding

primal quad-chains. In this respect, paired segments are analogous to river banks that both bound and define the flow between them; the poly-chord represents a bridge across the flow, connecting the paired segments.

Simultaneously computing this segmentation and pairing is an ambitious problem; we want to explicitly minimize the average matching cost, while avoiding outlier matches with very high cost. We consider the average, rather than the sum, so that the cost is not affected by the number of segments. To render this problem tractable, we use a discrete iterative optimization strategy that interleaves matching and segmentation. Given an existing segmentation and an appropriate cost metric, the right pairing strategy is not simply one that minimizes an overall cost, but instead one that prioritizes strongly compatible segment pairs that define dominant flow-lines. As noted in the Introduction, this can be mathematically formulated using the concept of a *stable matching*; we can find such a stable matching using the method of Irving [62].

Once we have obtained such a pairing, we can then refine our segmentation by looking for a subdivision that maximally decreases our average matching cost without increasing the worst match cost (Section 3.1.5). To find the optimal splitting point(s), we examine the pairings in the current stable matching and consider strategies that improve the current high-cost matches. This new segmentation can then be fed back into the matching stage. To generate the desired segmentation and pairing, we start from an initial segmentation and interleave segmentation and matching steps. Since we aim for a compact quadrangulation, we use a coarse to fine segmentation update strategy, starting with the minimal segmentation for which the notion of opposite segments, or bridging directions, is well defined. To avoid over-segmentation we stop the refinement process once the improvement to the average match cost becomes insignificant.

The final segmentation induces a poly-chord graph, which we use to generate quad network connectivity. The generated interior curves are positioned using an extension of the quadrangulation scheme of Nasri et al. [94] (Figure 3.4 (c)). A mesh of the entire network is then computed as discussed in Sections 3.2 and 4 (Figure 3.4,(d,e)).

3.2.1 Segmentation and Matching

The pseudocode below describes the flow of our iterative segmentation and matching process. Every iteration, we subdivide one or more segments to maximally reduce the average matching cost, without increasing the worst-

match cost (Section 3.1.4). While the number of curve segments, at intermediate steps of the algorithm may be odd, each iterative refinement increments the number of segments, typically by one, admitting a perfect segment matching after one or two iterations. We continue to iterate until there is no significant drop in the average matching cost, rolling back to the last even segmentation when significant improvement is no longer possible (Figure 3.5). While this algorithm does not guarantee a globally minimal average match cost, it captures our design goals admirably in that it finds and preserves dominant segment pairs early and then refines segments as necessary to reduce the matching cost of poorly paired segments.

Notation: The above steps are described succinctly using notation and pseudo-code as follows: Given a curve segmentation $\sigma = 1, .., n$, we refer to (i, j) as a distinct segment pair with a matching cost $c_{i,j}$ ((i, j) and $c_{i,j}$ are symmetric). $c_{i,j}$ captures the compatibility of any two curve segments to form opposite sides of dual poly-chord in our target quadrangulation. $M(\sigma)$ is a perfect matching of σ, where each segment is uniquely paired, barring a solitary unmatched segment when the number of segments $||\sigma||$ is odd. We define the average cost of a matching $M(\sigma)$ as $cost(M, \sigma) = (\sum_{(i,j)\in M(\sigma)} c_{ij}^2)/(2 \cdot \lfloor||\sigma/2||\rfloor)$. A constant $drop = 1.25$ captures the factor of average cost reduction below which the iterative algorithm terminates.

σ= initial segmentation (Sec. 3.1.1);
$M(\sigma)$= stable matching of segment pairs (i, j) using match cost $c_{i,j}$ (Sec. 3.1.3);
$U_b^* = \infty$;
$cost^* = \infty$;
repeat
  if then$||\sigma||$ is even:
    $\sigma^* = \sigma; M^* = M; cost^* = cost(M, \sigma)$
    $U_b^* = \max_M c_{i,j}$;
  end if
  σ'=refine σ (Sec. 3.1.4);
  $M'(\sigma')$= stable matching of σ'; $\sigma = \sigma'$;
until ($||\sigma'||$ is even) and ($drop * cost(M', \sigma') > cost^*$ or $U_b^* < \max_{M'} c_{i,j}$);
create internal quadrangulation curves from poly-chord graph of $M^*(\sigma^*)$;

We now elaborate on the rationale and details of each step.

Initial Segmentation

As described in the Introduction we expect the flow-lines induced between any pair of segments to be smooth. Motivated by this continuity property of flow-lines, we can use any robust corner finding technique, such as

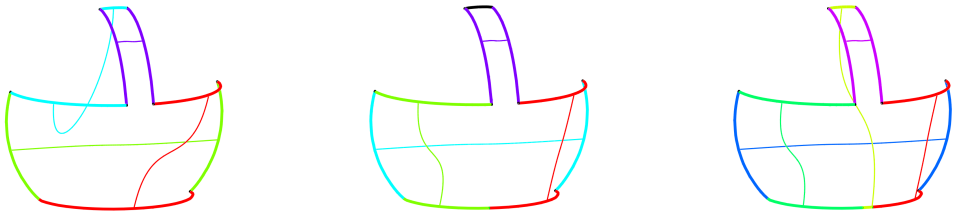
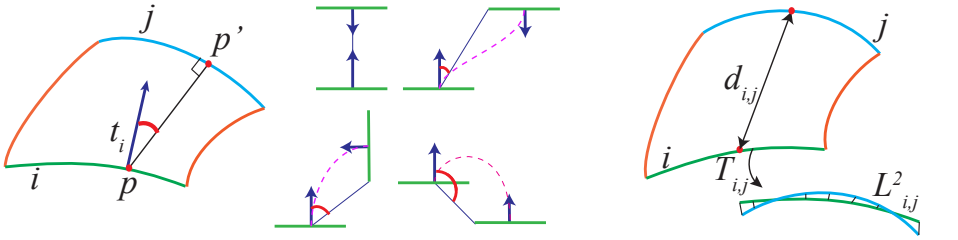


Figure 3.5: Iterative segmentation refinement: (a) initial segmentation where the matching highlights correct dominant side matches. The match quality is drastically improved by segmenting the bottom curve (b), and repeating the process (c) to obtain an even segment count. Further refinement has no real impact on matching cost.

computing discontinuities of discrete curvature along the curve [87], for our initial segmentation. We further refine this segmentation to ensure that the line segments connecting curve end-points are near linear using a technique similar to [88]. This property helps define coherent bridge directions for matching cost evaluation, described next.

Segment Pairing Cost

Paired segments have a two-fold impact on the final flow-line network. They explicitly define the sequence of flow-lines evolving from one segment to its mate. They also impact the family of flow-lines intersecting this sequence. Since the pairing defines a chain of quadrilaterals in the final quad network, these intersecting flow lines connect the two segments by evolving from one pair of end-points to another (see Figure 3.6 (a)). To generate the designer-expected flow-line network, the matching cost must satisfy the following criteria. First, to minimize the variation of flow-lines that evolve from one segment to the next we aim for the segments to be *similar*. Matching impacts the shape of the intersecting family of curves, or *bridge*, which in general we want to be as straight as possible, minimizing its *curvature*. Internal flow-lines should reflect input curve geometry, thus we would like the bridge to be aligned with intersecting flow lines evolving from input curve chains connecting the two segments, or, since these chains can be very complex, to at least *align* with intersecting sequences evolving from neighboring segments. Lastly, to best capture the general correlation between flow-lines and lines of curvature of the imagined surface, we expect intersecting sequences of flow lines to be *orthogonal*. We capture the last two requirements through a



(a) bridge curvature $b_{i,j}$ using bridge directions t_i (b) distance $d_{i,j}$, similarity $s_{i,j}$

Figure 3.6: Estimated bridge curvature for different segment layouts measured as angle (red) between bridge direction t_i and $p - p'$ (at a point p). The dashed lines visualize representative intersecting flow-lines (a). Shape similarity and distance cost terms (b).

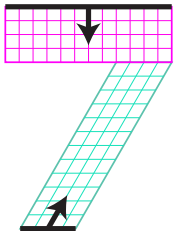
per-segment preferred *bridge direction*, which depends on the segment and its two neighbors. We use these directions to define bridge curvature $b_{i,j}$. Our matching cost combines bridge curvature, a term measuring similarity between the segments $s_{i,j}$, and a weak distance term $d_{i,j}$ used to prioritize more close-by matches

$$c_{i,j} = w_b b_{i,j} + w_s s_{i,j} + w_d d_{i,j}.$$

As the segments typically have fairly similar shape, bridge curvature dominates the cost with $w_b = 0.8$ and $w_s = w_d = 0.1$.

Bridge Curvature: To estimate the curvature of the anticipated intersecting flow lines, or bridge, between segments i and j, we use the predicted bridge directions t_i and t_j for both ends of the bridge. As illustrated in Figure 3.6, the flow-line shape depends both on these directions and the relative location of the segments. As start and end positions, plus directions, allow for fitting of multiple flow-line curves, explicitly evaluating flow-line curvature is problematic. Instead we use an angle based curvature predictor defined as follows. Let p be a point on the segment i, and let p' be the point where the angle between the vectors t_i and $p - p'$ is minimal on the segment j, i.e. $p' = \text{argmin}_{x \in j} |\angle(t_i, x - p)|$. Then, for a given point, the angle $\angle(\overline{t_i}, \overline{p' - p})$ measures the angular difference between the shortest bridge between the segments and the one taken when using the estimated bridge direction t_i. To compute deviation across the segment i, $a_{i \to j}$, we average the angular difference over all points. Finally, we set the bridge curvature to the maximum of the per-segment deviations, namely $b_{i,j} = \max(a_{i \to j}, a_{j \to i})$.

Bridge directions: The bridge direction t_i is the predicted optimal tangent direction for the flow-lines intersecting the segment i. As such, it depends both on the segment orientation, and on the bridge directions at neighboring segments.



The initial bridge direction t_i, for any segment i, is estimated from the initial segmentation (Figure 3.7(a-c)) and then refined in every subsequent algorithmic iteration (Figure 3.7(d)). The initial bridge direction $t_i = n_i$, is set to capture a direction orthogonal to the segment and lying on the imaginary surface emanating from it. Specifically, we define n_i as the perpendicular to the straight line f_i connecting its end-points, in the best-fit plane of the segments i and its neighbors. Neighboring segments can also strongly influence bridge direction. An adjacent segment m is considered to influence the bridge direction of i if it is of reasonable arc-length l ($1.5 * l_m > l_i$), and if its general flow direction f_m is likely to form flow-lines intersecting those emanating from i ($\angle(f_m, f_i) \leq 135\,^\circ$). The bridge direction t_i is refined to be the average f of its influential neighbours (Figure 3.7(a)(b)), or left as n_i if none exist (Figure 3.7(c)).

Then, at every algorithmic iteration, we update bridge directions (Figure 3.7(d)), using *dominant pairs*, i.e. pairs (i, j) such that $c_{ij} < dom$, where $dom = 0.15$. First, we refine the bridge direction of the dominant pairs. We update t_i and t_j of all dominant pairs (i, j) to their current average (thus implicitly lowering their bridge curvature estimate $b_{i,j}$). Next, for any segment i that is not dominantly matched but has a neighbor m that is part of a dominant pair, we use t_m to update t_i. Specifically, we attempt to set t_i to either align, or to be orthogonal to, t_m. if the angle between t_i and t_m is less than $135\,^\circ$, we set t_i to be orthogonal to t_m in the plane defined by n_i. If $135\,^\circ \leq \angle(t_i, t_m) \leq 225\,^\circ$, we set $t_i = t_m$. If t_i has two dominant neighbors, we use the one with lower matching cost for the update. The remaining bridge directions are left unchanged in this iteration.

Distance and Similarity: The distance $d_{i,j}$ is simply the Euclidean distance between the segment centers (Figure 3.6b). Given two curve segments i, j, we measure their similarity in terms of shape and scale. We measure scale as the difference in curve length $\|l_i - l_j\|$.To compare shape, we first compute a best-fit affine transform $T_{i,j}$ from i to j. We do this by resampling the curves by arc-length using the same number of points, 50 for all our experiments. We then use a linear least squares formulation to find the affine transformation which minimizes the L_2 distance between the two point-sets.

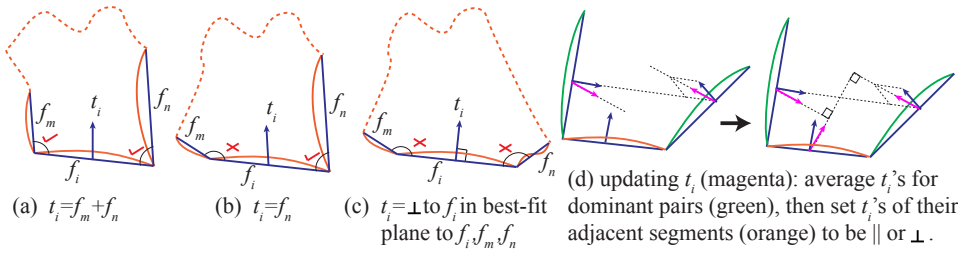


Figure 3.7: Initial bridge direction t_i of segment i is determined by adjacent segment flow directions f_m, f_n and its normal.

We use a generic affine transform instead of a rigid one to allow for non-uniform scale and shear. We then measure similarity as the L_2 closest-point distance between the transformed curve and its mate $L_{i,j}$. All distances are normalized by the *diameter* of the processed curve, i.e. by the maximal distance between two points on the curve. Similarity between curves is then set to $s_{i,j} = 0.5\|l_i - l_j\| + 0.5(1 - e^{-L_{i,j}^2/\sigma^2})$. The second term measures the affine invariant shape difference of two curve segments. Specifically, we define a function that is zero if the curves are identical and 1 if they are maximally different. We achieve this mapping using a Gaussian fall-off function applied to the L_2 distance between the curves segments. Normalizing this distance by the diameter of the curve loop and setting $\sigma = 1/3$, set using the three-sigma rule, results in the desired shape difference function.

Stable matching of segment pairs

Given a curve segmentation and a cost of pairing any two curve segments to form opposite sides of a poly-chord, this step aims to match segment pairs in a manner that maximally satisfies the dominant pairing preferences producing a stable matching.

The standard algorithm for computing a stable matching [62] consists of two phases. First, each segment "proposes" to all other segments in order of pairing preference, continuing to the next segment if and when its current proposal is rejected. A segment rejects a proposal if it already holds, or subsequently receives, a proposal from a segment it prefers. In our setup, since matching costs are symmetric, if the number of segments is even this step ends with each segment holding a proposal from another segment. If the number of segments is odd, one segment is left out by the process and is ignored by the subsequent step.

Held proposals form a set S of ordered segment pairs (i, j), where i holds a proposal from j (j is i's current favorite). S is a stable matching if $(j, i) \in S$ whenever $(i, j) \in S$. A second phase of repeated co-rotations, described below, transforms S into a stable matching. Suppose that $(i, j) \in S$, but not (j, i). For each such i we identify the current second favorite to be the first successor of j in i's preference list who would reject their held proposal in favor of i. A rotation relative to S is a sequence $(i_0, j_0), (i_1, j_1), ..., (i_{k-1}, j_{k-1})$ such that $(i_m, j_m) \in S$ for each m, and j_{m+1} is i_m's current second favorite (all indices are modulo k). A co-rotation replaces pairs (i_m, j_m), with (i_m, j_{m+1})in S.

The standard method [62] is proven to provide a stable match for an even number of participants, unless an *odd party* is found [124], i.e. a rotation such that k is odd, and $p_i = q_{i+(k+1)/2}$ for all i. In that case no stable matching exists. In the rare case of an odd party, we have an odd-length cycle of segments with equal pairwise costs, e.g. an equilateral triangle or three perfectly symmetric curves (Figure 3.10). This case can be seen as a generalization of the standard midpoint splitting, and is resolved by splitting each segment in the cycle into two. Once the split is performed, a clear difference in cost emerges and the matching is repeated.

Segmentation Refinement

The refinement process looks for a segment, or segments, to subdivide so as to maximally decrease the average matching cost. Our refinement examines two segmentation strategies, first searching for a single edge refinement and then a global mid-edge split. Since the number of segments is typically very small, a stable matching computation is practically instantaneous. Using the first approach, we quickly iterate over all segments, segmenting each one and evaluating the cost of the match computed with the refined segmentation. We then select the segmentation that maximally lowers the cost. Using this strategy, the one question we need to address is where to place the split, as the location can impact the subsequent segmentation cost.

The basic strategy of splitting the segment in half is tested first, then a more targeted strategy that leverages the computed matching is applied to the currently matched segments. Given a current segment i which is matched to j we search for all segments k that are either unmatched, or that prefer to be matched to i rather than their current mate l, i.e. $c_{k,i} < c_{k,l}$. In such situations, for instance the bottom curve on the basket (Figure 3.5), splitting the curve strategically into i_1 and i_2 can often satisfy this preference by generating matches (i_1, j) and (i_2, k). To minimize the cost of (i_1, j) and

(i_2, k) we break i into two possible subdivisions i_1, i_2 based on arc-length (l) ratio, where $l_{i_1}/l_{i_2} = l_j/l_k$ or $l_{i_1}/l_{i_2} = l_k/l_j$, and $l_{i_1} + l_{i_2} = l_i$, and test the matches induced by these segmentations.

While theoretically more comprehensive or global segmentation refinement strategies may exist, we found our approach to work well in practice. It preserves dominant pairs and improves poor matches as intended by our subdivision heuristic.

3.2.2 Quadrangulation

Once we have an acceptable perfect stable match whose cost cannot be reduced by further segmentation, we use this segmentation and matching and its induced poly-chord graph (see Figure 3.4), to construct a quadrangulation.

Extracting Quad Connectivity: Using standard dual notations [31] we say that two poly-chords (i, j) and (k, l) *intersect* in the graph theoretic sense if and only if their corresponding curve segments are interleaved on the closed curve. For instance, the purple and red segments on Fig. 10 are interleaved, resulting in intersecting poly-chords. To generate a valid quadrangulation we require that the poly-chord graph be connected . This is easily accomplished by adding curve segments connecting end-points of common segments of components of the poly-chord graph and turning each graph component into a smaller closed curve, for which our algorithm can be re-run (Figure 3.8). To avoid T-junctions we disallow the newly added segments from being further refined. To make the quad layout more compact, we merge adjacent poly-chord (i, j) and $(i+1, j-1)$ when the transition between the consecutive segments is smooth.

An intersection between two poly-chords corresponds to a quadrilateral in the final network. Connectivity between these quads is determined by the intersection order, e.g. determining the top-down order of the intersections of the green poly-chord with the blue and red ones in Figure 3.9. We define the quad connectivity by incrementally embedding poly-chords into the layout of cells, or regions, bounded by input boundary segments and previously added poly-chords. Given the graph whose vertices are these cells and whose edges connect adjacent cells, we embed a poly-chord by computing the shortest path in this graph between the two vertices or cells, corresponding to the boundary curve segments connected by the poly-chord. This path minimizes the number of intersections between the new poly-chord and those already embedded. This choice minimizes the number of dual graph

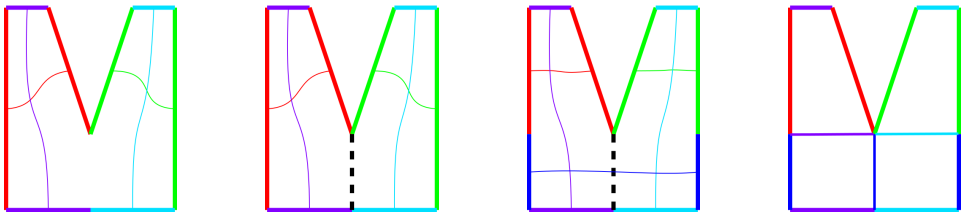


Figure 3.8: A disconnected dual graph (left) does not allow for a valid primal quad mesh. Splitting the cycle into two by a temporary curve segment (dashed) generates valid graphs for both parts which combined together induce a valid primal quad mesh (right).

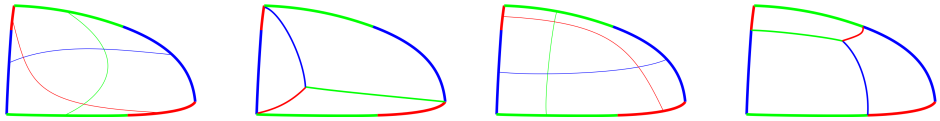


Figure 3.9: Two intersection orders induce different quad connectivity, with the one on the right inducing a better quad shape, and consequently a smoother flow.

cycles. Such cycles correspond to interior primal quadrangulation vertices adding which, as discussed below, can reduce flow smoothness. Given two equal length choices, we prefer one that induces better shaped quadrilaterals, where quality is measured as the scaled Jacobian [19] (Figure 3.9).

Extracting quad geometry: The dual graph defines the connectivity of our quadrangulation. To position the interior vertices and curves we use a two step process which leverages the quad topology to generate interior curves best reflecting the flow directions. Specifically we note that each chain of quads can be seen as a four-sided uv patch interpolating two flow-line end segments. Associating the v coordinate with the end segments, we expect the patch u-isolines to smoothly interpolate them. Our geometry computation builds on the geometry construction in [94] which shares the same goal. We first compute the interior vertex positions that best satisfy our requirements, using a global optimization of a per-vertex formulation [94], that sets each vertex G to a weighted sum of vertex positions in neighboring quads:

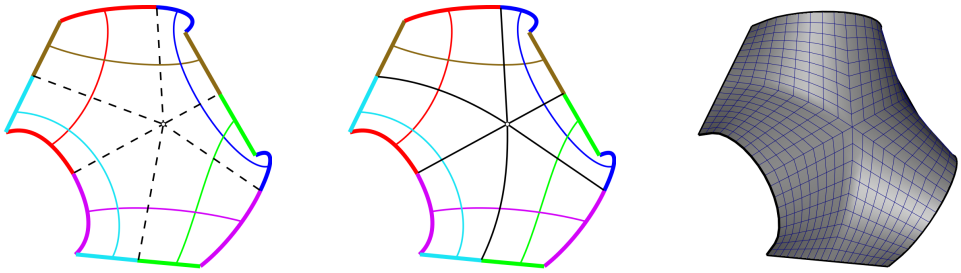


Figure 3.10: We first position interior vertices (left) and then use the chain-long quads to position the interior curves (center). Finally, the resulting quad cycles are quad-meshed using discrete Coons patches (right).

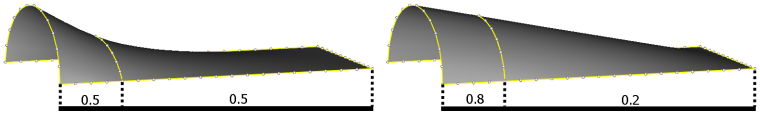
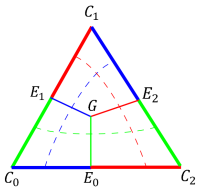


Figure 3.11: Our distance based weighing (right) generates smoother flow line evolution than topology based one [94].



$$G = \frac{\sum_{i=1}^{n}(E_i + E_{i-1} - C_i)/a_i}{\sum_{i=1}^{n} 1/a_i} \tag{3.1}$$

where E_i are quad network vertices that share side curves with G, C_i are the diagonal quad corners between E_{i+1} and E_i, and $a_i = \|E_i - C_i\|\|C_i - E_{i+1}\|$ is an estimate of the area of the corresponding quad (see inset). We then generate straight-line edges connecting these and boundary vertices as an intermediate approximation of the quadrangulation. Using this initial network each interior curve is now computed as a u-isoline on the quadrilateral patch containing two bounding flow-lines and the curve paths connecting them, using a discrete Coons formulation [38] (Figure 3.10). This formulation takes into account the distance of the new curve from the bounding flow lines, improving on the original formulation of Nasri et al [94] (Figure 3.11).

Meshing: To fit a surface in the interior of each quad-patch we can use any number of methods. The examples shown in this chapter use a quad mesh sampled on a discrete bicubic Coons surface [109]. This construction provides continuity across shared boundaries when the cross tangents are

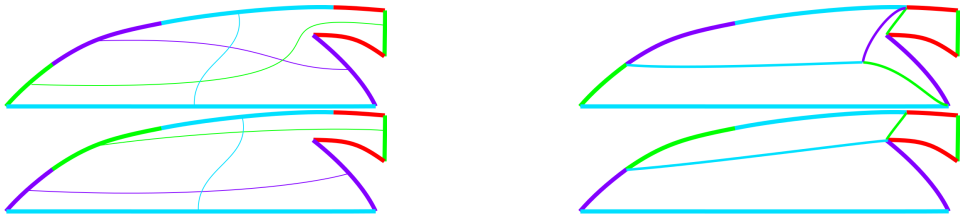


Figure 3.12: Removing interior vertices: (Left) initial match (top) and induced quadrangulation (bottom); (Right) the final match with purple and green pairs flipped (top) has a slightly higher cost but the induced quadrangulation (bottom) has no interior vertices, leading to smoother flow-lines.

continuous. More sophisticated fitting tools which provide better cross-patch continuity can be used as well.

Minimizing Flow Dislocation: The segmentation and pairing algorithm optimizes the cost of the individual flow-line matches, does not explicitly consider the impact of the quad patch connectivity on the final flow. Specifically, at the matching stage it is hard to predict the impact of the introduction of interior patch vertices on the smoothness of the flow lines. In some cases these vertices are essential to forming a good surface such as on the top of the espresso machine (Figure 3.17), but in other cases removing them can improve the flow (Figure 3.12). Thus, given a quadrangulation, we test if removing any of the interior vertices can improve the surface quality. Recall that each such vertex corresponds to a cycle in the dual graph. We thus attempt to break cycles in the dual graph if the quadrangulation quality improves and the increase in the overall matching cost is acceptable. Specifically, for each edge $\langle (i,j),(k,l) \rangle$ of a cycle in the poly-chord graph we evaluate the consequence of swapping segment pairs to (i,l) and (k,j), or (i,k) and (l,j). A swap is valid if the following three criteria are satisfied: the quad quality, measured using the scaled Jacobian, is improved, no new cycles are introduced into the graph and the cost of the matches after the swap is no greater than the worst match cost before it. We thus perform a valid swap for the poly-chord edge of the cycle with the minimum increase in matching cost (Figure 3.12).

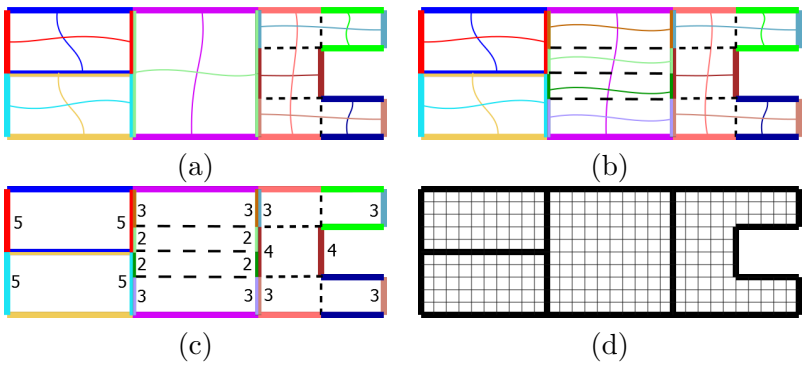


Figure 3.13: Separately processed cycles (a) introduce T-junctions. We first resolve the T-junctions across pairs of neighbouring patches by propagation (b), generating a well defined hierarchy of matching primary segments. We then use integer programming to compute interval assignments (c) that minimizes the number of T-junctions, typically leading to a watertight mesh (d).

3.3 Processing Curve Networks

Up until now, we have only considered the meshing of a single curve cycle. The reason for this is that in curve networks, the majority of vertices adjacent to two or more cycles define corners that induce our initial segmentation. The remaining vertices form T-junctions that should not bias the flow-lines within cycles where the incident curves are continuous. Once the individual cycles have been quadrangulated however, we must ensure that the geometry is watertight across the common boundary of adjacent cycles. For a quad-mesh fitting this requires the sampling, or interval count, along shared boundaries to be the same on both sides. This goal is easy to achieve for a conforming quad-patch layout, such as those generated inside each input cycle, using a fixed number of intervals per boundary curve. Special care is needed though, when meshing curve networks where cycle segmentation creates T-junctions.

We optimize interval assignment using two modifications to the basic cycle quadrangulation algorithm described above. The first stage, performed after segmentation and matching process, described above, for each cycle, resolves the initial, *primary*, T-junctions between pairs of neighbouring cycles. A T-junction occurs when one curve has a segment end-point, or vertex, at a boundary point and another curve does not. Given a T-junction, we first

attempt to resolve it by merging adjacent vertices based on a threshold distance, while keeping in place both sharp corners and T-junctions present in the original artist input. Throughout our experiments, we set our threshold to 5δ, where δ is the minimum Euclidean distance between adjacent samples of the input polylines. Intuitively, the finer the initial sampling, the more precise the algorithm is, the smaller the merging threshold we need.

For any T-junctions that we cannot resolve in this manner, we split the adjacent segment and its matching segment in the corresponding cycle. We then refine the matching accordingly. This process resolves all the primary T-junctions, but in turn introduces *secondary* T-junctions where the matching segments are split (Figure 3.13, (b)). These T-junctions are further reduced using another iteration of threshold based merging.

Contrary to primary T-junctions, the secondary T-junctions are guaranteed to be contained in *primary* segments that share clearly defined *primary* vertices (Figure 3.13, (b)), a property we take advantage of in the final interval assignment stage. At this point, the network is converted to quad-patch topology using the method of Section 3.2. In the final step, when generating the per-patch meshes, we need to assign a consistent interval count to each segment. For a given primary segment, we require that the number of intervals on both sides of the segment are equal. We further require that each secondary segment (one bounded by primary or secondary vertices) and its matching segment have the same number of intervals. Finally, we wish to minimize the total interval count while enforcing a minimum number of intervals per edge based on its length.

If we formulate all of these requirements as a wishlist, as shown by Mitchell [91], there may exist configurations where no valid assignment exists. We therefore relax our watertightness requirement, which allows us to reformulate this problem in terms of a minimization. Consider a pair of adjacent primary segments L and R. By virtue of the first step, we know that L and R share common endpoints; however, they may each contain a differing number of secondary segments. If l is a secondary segment on L and r is a secondary segment on R, let $n_{l,R}$ and $n_{r,R}$ represent the number of intervals that the secondary segments l and r are divided into, respectively. We can then express our minimization condition as the following function:

$$\min\ f(x) = w \sum_{(L,R)} (\sum_{l \in L} n_{l,L} - \sum_{r \in R} n_{r,R})^2 + \sum_{L,l} (n_{l,L})$$

The first term in this equation seeks to minimize the number of mismatched interval counts along a given pair of adjacent primary segments. The second term seeks to minimize the total number of intervals for the entire mesh. We

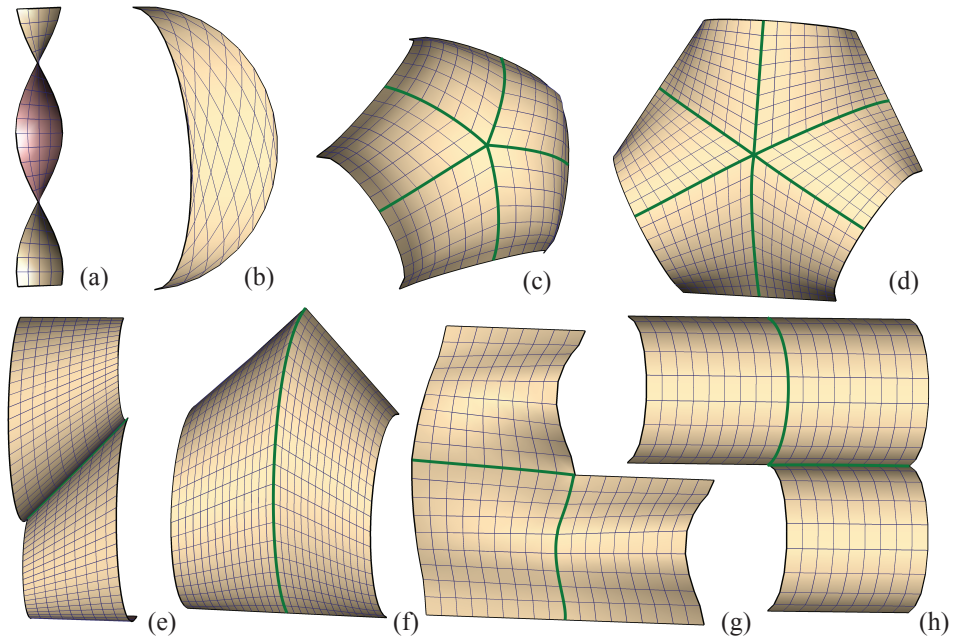


Figure 3.14: Quadrangulation and meshing of closed curves.

use $w = 1000$ to minimize the number of mismatches as much as possible. This minimization is subject to a number of constraints. We require that opposite segments of each quad patch have the same number of intervals. Additionally, we require that the number of intervals on a given secondary segment does not fall below a specified minimum. This minimum is determined by dividing the secondary segment length by a user-specified desired (local or global) interval length. Together, the minimization function and constraints form a quadratic, mixed-integer programming problem, which we solve using Tomlab /CPLEX. This approach lead to valid assignments for all the inputs we tested. The assigned intervals are used to optimize the positions of the secondary T-junctions and generate the final meshes.

3.4 Results

Closed Curves: We generated a number of synthetic test inputs to evaluate the behavior of our method on a variety of closed curves with different side configurations demonstrated in Figures 3.14 and 3.15. These included a variety of convex regions (Figure 3.14 (a-f)) with different degrees of pla-

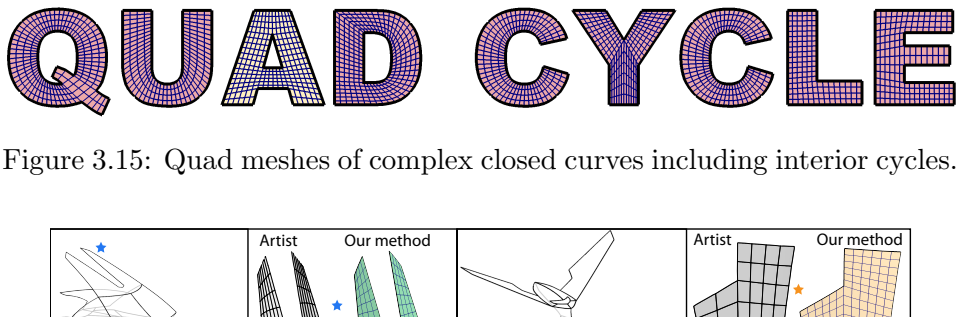


Figure 3.15: Quad meshes of complex closed curves including interior cycles.

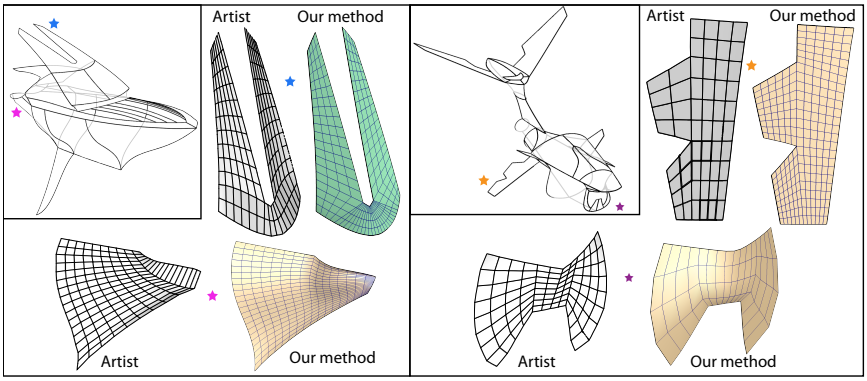


Figure 3.16: Artist generated meshes (left) and ours (right) exhibit very similar flow-line patterns.

narity and different number of boundary discontinuities. For some of the inputs the expected surface shape, is best captured by introducing an extraordinary interior vertex (c,d). For other regions with $n > 4$ sides such as (e,f) a regular connectivity better captures the intended shape. Our method makes the appropriate choice based on analyzing the relationships between the input curve segments,and the degree of parallelism between them. This is in contrast to purely connectivity methods, e.g. [94], where the choice is strictly based on the number of segments. Figure 3.14 (b) shows an a-typical two sided region, nevertheless reasonably fittted by our method, while (g,h) show non-convex regions, where the optimal pairing is found automatically through refinement of initial segments. The letters in Figure 3.15 show the robustness of the method in the presence of complex non-convex curves as well as processing of faces with interior loops. To handle such models, we first locate a pair of matching segments on different loops with minimal matching cost and introduce the shortest straight segment connecting those. The method then proceeds as usual on the resulting single cycle.

Curve Networks: We tested our method on a variety of input curve networks (Figure 3.1, 3.4, 3.17) generated by different modeling systems

[7, 107, 115]. As demonstrated by the figures these networks contain a variety of complex, non-convex cycles. Our method successfully captures the designer intent conveyed by the networks generating predictable and smoothly flowing quad-meshes interpolating the input curves. While the airplane (Figure 3.17) was created using a classical CAD modeling system, many of the other inputs (car, espresso maker, submarine, starcraft) (Figure 3.17) were generated using sketching tools, which easily introduce noise and inaccuracies that hamper traditional surfacing. Our method is robust to such artifacts.

We compare our outputs on the boat and starcraft to those generated by an artist (Figures 3.3 3.16). The flow-line structure of our meshes is largely identical to artist generated one, with only minor differences, such as flow on the side of the boat cabin, where both our and artist interpretations are feasible (our outputs contain a few extra cycles not present in the artist models).

Quantitative Evaluation: On an Intel i7 CPU 870 2.93GH machine our method takes on average two seconds to quadrangulate a single curve cycle (most of the time spent on matching), making it amenable for interactive surfacing in a sketch based modeler like ILoveSketch [7]. The most time consuming regions are the front of the car (166s) and the top of the speaker (66s) (Figure 3.17). Intervals assignment is practically instantaneous, taking 0.1s for a an average network and taking 2s to process the largest model (plane). The quad statistics for the models we tested are summarized in Table 3.1 and include numbers of input cycles, number of output quad cycles, mesh size(s), and the number of added extraordinary vertices, All the generated meshes are watertight.

Limitations: Our approach has three broad limitations which can be addressed by future research.

Global context: The biggest limitation of our method is lack of global context. Our flow-line analysis for each input cycle in a network is independent. In practice however, most adjacent cycles meet at sharp corners, typically resulting in a similar segmentation and flow across shared curve segments. The context of adjacent cycles could be useful in enforcing flow line continuity across cycles and predicting the flow within an individually ambiguous cycle.

Failure cases: While our algorithm works well on design curve inputs from a variety of sources, it may not provide meaningful results for arbi-

	input cycles	output quad cycles	mesh size	interior vertices
Sphere Bag	3	9	987	0
Boat	30	82	4464	9
Spaceship	41	94	5008	6
Car	26	70	5020	13
Espresso	54	75	6904	5
Speaker	13	42	8548	1
Plane	140	192	10705	10
Submarine	39	103	16600	31

Table 3.1: Algorithm statistics for different curve networks.

trarily shaped curve cycles with no perceptible flow-lines. The absence of corners on a completely smooth curve cycle will not provide us a meaningful initial segmentation to refine. In such cases we can impose an initial segmentation based on curvature maxima and arc-length of the input curve.

Algorithmic complexity: While our central idea of flow-line segmentation and matching is conceptually clear, various aspects of our implementation could be streamlined. For example, while most of the parameters used by the method were derived based on clear algorithmic goals, a few such as *drop* in Section 3.2.1) are based on trial-and-error, and could be learned from designer quadrangulated examples.

3.5 Conclusions

We presented the first, to our knowledge, method for quadrangulating general designer specified closed 3D curves and curve networks. Our results show the approach to robustly process complex curve networks, generating interpolating quad meshes consistent with designer intent. Our key insight is an interleaved segmentation and matching algorithm, that pairs dominant flow-lines and uses poor matches to guide segmentation refinement, computing a poly-chord graph that captures user-intended bridging directions across a closed curve. We advocate the use of stable matching as the principled way to formulate our quadrangulation goals and anticipate it to be well-suited to other problems relating to shape matching or coherence, where both dominant components and their correspondence is sought.

Our work points to a number of future directions. Rather than restrict our input to a constrained geometric definition of a design curve network, we attempted to quadrangulate any 3D curve network as a designer would,

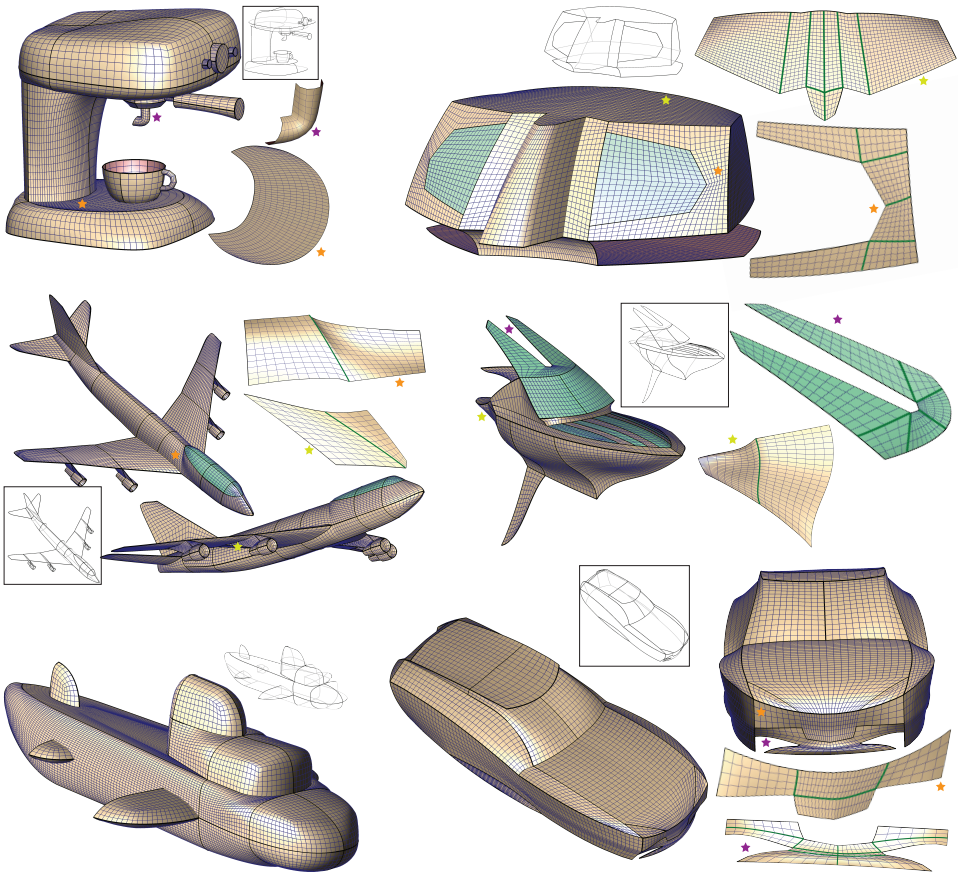


Figure 3.17: Quadrangulation and meshing of curve networks. The stars indicate the network locations of the highlighted complex regions.

using the principle of flow-line segmentation and matching. A formal perceptual study of the precise difference between ambiguous and design curves (Figure 3.2) is thus an ambitious but worthy goal. While our segmentation refinement strategy works well in general, approaches with theoretical guarantees of match quality are also worth exploring. Our method focuses on quad-only meshing, however in some cases designer intent is better served by allowing a small number of triangular elements (e.g. Figure 3.14 (b)), motivating a technique for mixed but predominantly-quad meshes. We would also like to apply our technique as-is to the finite-element meshing of closed planar domains, balancing flow-line alignment against mesh quality.

Chapter 4

Modeling Character Canvases from Cartoon Drawings

4.1 Introduction

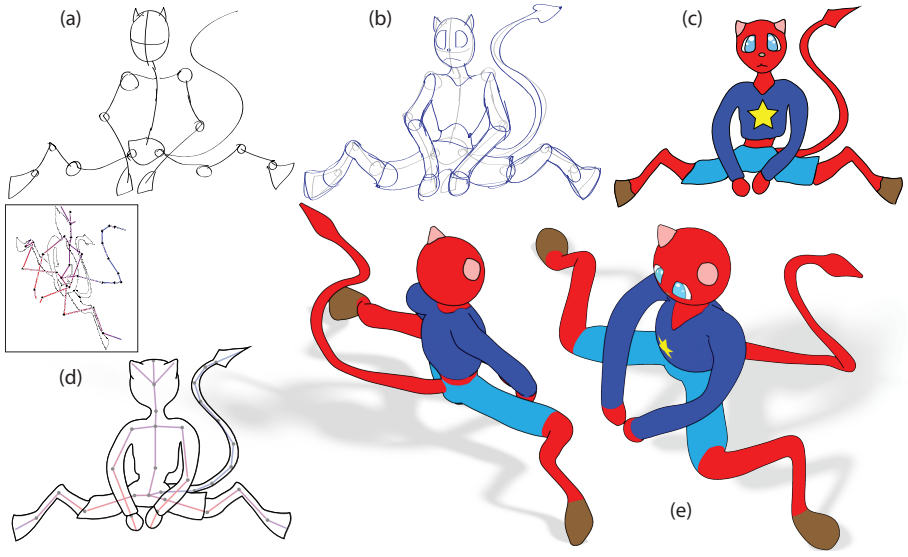


Figure 4.1: Character drawings (c) are traditionally anchored around a skeleton (a), surrounded by generalized surfaces of revolution (b). We use the drawn character contours (d) and a corresponding 3D skeleton (red-to-blue coloring reflects near-to-far skeleton depth variation), to automatically compute a 3D canvas, employed to freely manipulate the character in 3D (e).

In this chapter we describe our second main contribution, a novel approach for automatically constructing a rigged 3D character proxy, or *canvas*, directly from a single 2D cartoon drawing and a correspondingly posed,

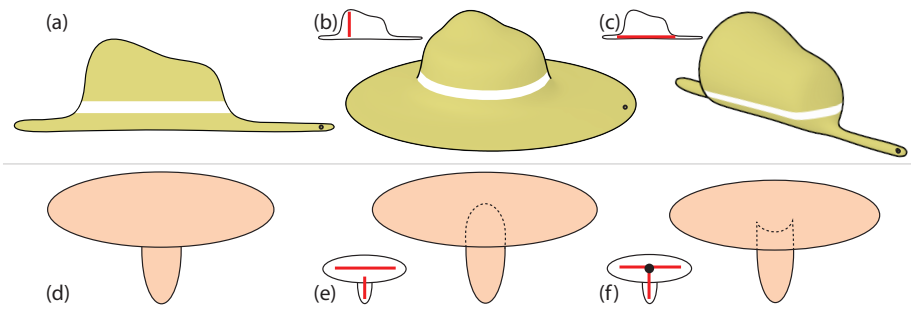


Figure 4.2: Character contours alone (left) frequently do not provide sufficient information to conclusively determine 3D shape both on occlusion free (top) and partially occluded (bottom) inputs. A 3D skeleton, shown in the insets, resolves much of the ambiguity present in contours alone facilitating plausible shape interpretation.

user-supplied, 3D skeleton. Such approach allows users to sidestep the time-consuming manual modeling and rigging steps (Fig. 4.1(d,e)). The project has since been published in ACM Transactions on Graphics [10].

Our 3D canvases allow artists to directly articulate the drawn characters, generate convincing cartoon style character renders from alternate views (Fig. 4.1(e)), and provide support for various 3D effects created by drawing on and around the canvas (Fig. 4.23). Using a skeleton as an aid, our framework infers complex, complete character shapes from individual 2D drawings with significant contour depth variation, foreshortening, and multiple inter-part occlusions (Fig. 4.3 (left)) - a significant deviation from prior art, which assumes drawn contours that are largely occlusion free, flat, and nearly perpendicular to the view direction (Section 2.2).

Our choice of input and subsequent construction methods are motivated by the observation [57, 136] that cartoon character anatomy is well described by a union of body parts supported by a skeletal system, where each part is approximately a generalized surface of revolution (Fig. 4.1(a,b)). Artist-drawn character contours are inherently ambiguous (Fig. 4.2) and human observers frequently rely on either explicit familiarity with the drawn objects, or on semantic information encoded by additional drawing elements, such as facial features, to consistently interpret the 3D character shape. Such extra information is hard to enumerate or formalize algorithmically; our input skeleton, posed to reflect the character's structure, helps resolve these shape ambiguities.

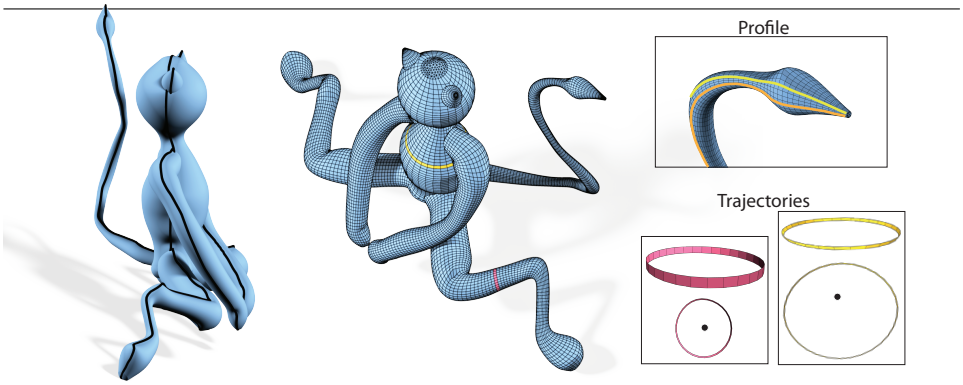


Figure 4.3: The canvas (center) of the catwoman in Fig. 4.1: (left) thick black line shows reconstructed 3D contour curves, (right) insets visualize representative trajectories and profiles.

Overview Our guiding premise is that when artists create descriptive character drawings, they inherently rely upon and exploit the same perceptual principles that viewers use to lift drawings off paper and into 3D [116]. Following previous work, we rely on viewer preferences for *conformity* and *simplicity* (Section 4.2, Fig. 4.5) in reconstructing individual part geometry. Conformity is the unstated belief that the drawing is an accurate representation of the 3D character, and that the projected contours of the 3D characters will conform to the drawn contours in the input view and pose. Simplicity (or the law of Pragnanz [69]) states that viewers rely on symmetry assumptions as strong cues for image understanding [56, 105]. Given viewer familiarity with character anatomy expected to resemble partwise surfaces of revolution, this principle suggests a strong viewer preference for envisioning body-parts with maximal rotational symmetry around the bone axis (Fig. 4.5 (b)).

We augment these two principles with observations about *Gestalt continuation* and shape *persistence* which help us parse complete, complex drawings and reconstruct coherent overall character shapes. To handle inter-part occlusions in the drawings, we exploit *Gestalt continuation* by noting that viewers resolve occlusions in line drawings by grouping together disjoint curves whose end-points can be smoothly connected [69] (e.g. the outlines of the tights of the catwoman in Figure 4.1). In reconstructing the complete character geometry from a single view drawing, we rely on the notion of shape, or contour, *persistence*. Contour persistence or the non-accidental view assumption [93, 138] indicates that viewers perceive the artist-selected view and pose as non-accidental and expect the drawn contours to be in-

dicative of contour shape in alternate, and especially nearby, views.

We begin the modeling process by segmenting the input 2D contours into sections outlining individual body parts corresponding to the bones of the input skeleton. We resolve inter-part occlusions and group disjoint outline segments by leveraging skeletal depth and Gestalt continuation. We use the computed contour segmentation to generate the 3D canvas geometry, modeling body parts using generalized surfaces of revolution. While a canonical surface of revolution is defined by rotating a fixed planar *profile* curve along a circular *trajectory* around an axis, we account for a range of body shapes by supporting both more complex closed planar trajectory curves, and by allowing the profile shape to vary smoothly as the profile rotates around the part's bone or axis (Fig. 4.3). Supporting profile variation is critical for processing asymmetric part contours, such as those on the catwoman's hoofs. To balance conformity against simplicity we first refine the artist given straight-line skeleton to a geometric curve-skeleton [29], and symmetrically locate it with respect to the artist-drawn contours. The surfaces of the different body parts are then optimized to form a unified 3D canvas centered around this curve-skeleton by enforcing conformity while balancing individual part simplicity against contour persistence across the canvas. Our final canvases are represented as quad-dominant meshes (Fig. 4.3 (center)) with explicit angular and axial parameterization which supports a range of texturing effects (Fig. 4.23).

Contribution Our overall contribution is a framework for computing a believable 3D character canvas from two pieces of user input: a vectorized, single-view, descriptive, 2D contour drawing and a correspondingly created and posed 3D skeleton. Our key technical contributions are two algorithms derived from perception principles. First, we present a novel algorithm for correctly segmenting artist-drawn contours into body part outlines associated with individual skeletal bones, which can robustly handle multiple inter-part occlusions (Section 4.3). Second, we show how to use this segmentation to generate believable 3D character canvases which balance simplicity and persistence, allowing for variable contour depth and overcoming inaccuracies in skeleton posing (Section 4.4). Our resulting 3D character canvases are, as the name suggests, an ideal support structure for painterly strokes and cartoon rendering; however, they are not designed to capture the complex detail of realistic 3D character models.

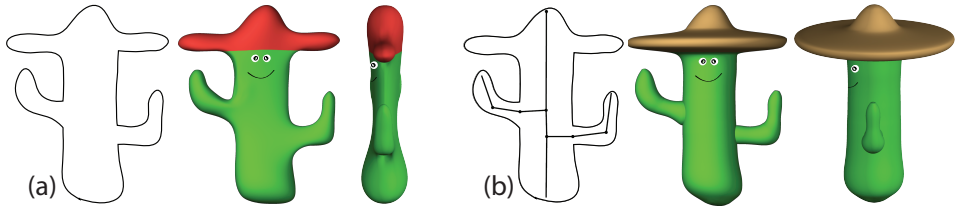


Figure 4.4: (a) Lacking part information, character shape reconstruction can at best exploit overall shape smoothness, e.g [68, 95]; (b) by using a skeleton to facilitate contour partition and part modeling, we generate a more believable character shape.

Evaluation We evaluate our approach in a number of ways (Sections 4.5, 4.6). We show that the task of positioning a 3D skeleton to match a 2D cartoon drawing is well-defined and intuitive, taking most artists less than ten minutes for typical cartoon drawings. We validate our segmentation algorithm via an informal evaluation, verifying that viewers consistently segment and associate character contours to skeletal bones and that this segmentation matches our algorithmic output. We reproduce ground truth 3D character shapes from a contour rendering and 3D skeleton, and compare our results to both ground truth and artist drawings created from the same input and in the same views, validating that our results are visually similar to both. We show a variety of character canvases created from diversely sourced contour drawings and 3D skeletons, demonstrating our approach to be resilient to complex views, and poses with multiple occlusions and significant foreshortening. These canvases are illustrated using cartoon shading and other forms of non-photorealistic rendering, and are confirmed by artists to show plausible alternate-view renders of the drawn inputs. Finally, we compare our method to prior work, producing similar output quality with significantly less user-input.

4.2 Framework Overview

We now describe the three key components of our canvas computation framework, and the observations that motivate them (Fig. 4.6).

Algorithm Input The input to our system is a 2D vectorized cartoon drawing and a correspondingly posed 3D skeleton with no extra annotation. Like other research in articulated figure modeling [8, 17, 123] our approach is based on the proposition from cartoon drawing literature [57, 136] that

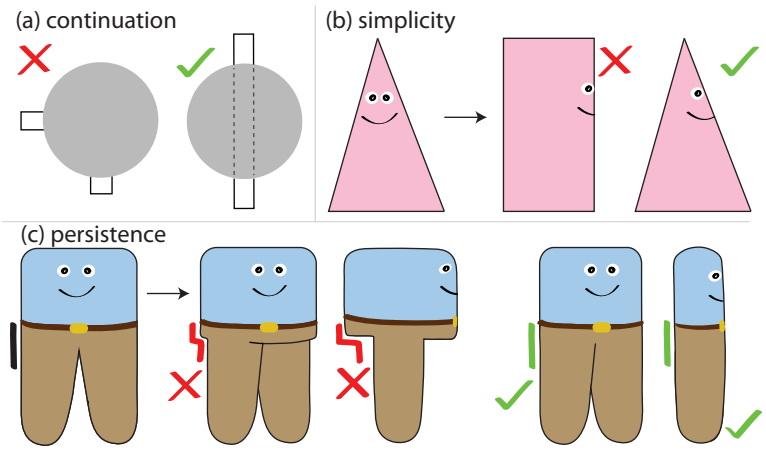


Figure 4.5: (a) Perceptual studies indicate that viewers group curves that can be smoothly joined together ((a), right), seeing those as a continuation of one another; while treating those with non-smooth transition ((a), left) as distinct; viewers tend to prefer interpretations that balance part simplicity (b) against contour persistence, preferring interpretations that preserve contour shape under small view changes (c).

character shape is well approximated by a union of body parts represented by generalized surfaces of revolution around a skeletal structure. As the shape of a surface of revolution is driven by the choice of an axis, leveraging this observation for modeling requires a skeletal structure (Fig. 4.4). While curvature extrema and discontinuities in character contours hint at the underlying skeletal structure, automatic skeleton extraction [17, 20] may not reflect the artist-intended shape as it always aligns the skeleton with the dominant axis in elliptical regions. This is illustrated by Fig. 4.2 (top), where using a geometric skeleton would lead to the "snake swallowing an elephant" reconstruction on the right. This bias is confounded by ambiguous skeleton topology in the presence of occlusions (Fig. 4.2, bottom). Fortunately, artists can consistently and efficiently pose a 3D skeleton to match a 2D contour drawing (Section 4.5.1), motivating our choice of input.

Skeleton-Driven Contour Segmentation To successfully capture body parts with surfaces of revolution, we must first identify which portions of the input contour belong to the same body part (Fig. 4.6 (a)). Our algorithm therefore begins by segmenting the input contours into sections associated

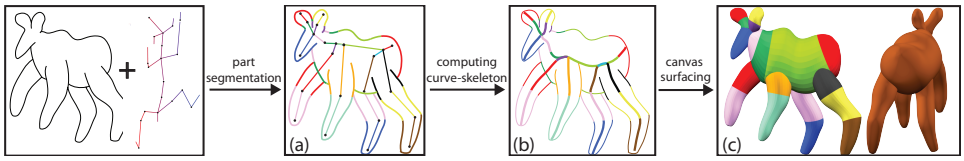


Figure 4.6: Canvas construction: Given a sketch and a skeleton (shown in side view) we first segment the input contours into sections associated with skeletal bones (a), correspondences shown by matching color), correctly resolving occlusions; we use the segmentation to replace the straight-line skeleton by a curved-skeleton optimized for symmetry (b); and finally generate maximally simple body part geometries around this new skeleton while maintaining contour persistence with respect to the input drawing (c).

with each bone. This segmentation is guided by the principles employed in 3D skeleton-driven surface segmentation algorithms, e.g [6, 29]. These methods construct surface charts whose connectivity reflects skeletal adjacencies, associating charts with proximal bones, and aligning chart boundaries with curvature extrema. We apply these principles of surface segmentation to 2D contour drawings. Since, in the presence of occlusions 2D proximity is not a reliable proxy for 3D proximity (Fig. 4.8), we leverage skeletal depth information to facilitate correct proximal bone-to-contour association and use Gestalt continuity [69] to correctly group disjoint contour segments (see Section 4.3, Fig. 4.5 (a)).

Canvas Modeling We construct a 3D canvas from our segmentation by exploiting the perceptual cues of sketch conformity, simplicity, and contour persistence (Fig. 4.6 (c)). In our context, conformity requires that the contours of the created 3D canvas project onto the 2D character contours in the input drawing with reasonable accuracy, and simplicity implies a preference for maximally symmetric surface-of-revolution part geometries (Fig. 4.5 (b)). Maximizing symmetry when recovering 3D part geometry requires an optimal local axis of revolution. However, while the artist-posed straight-line skeletons adequately describe the character structure, they are not detailed or accurate enough to capture a geometrically centered *curve skeleton* [29] of the target character surface. We therefore generate the desired curve skeleton by leveraging a correspondence between the straight skeleton and the segmented 2D contours (Fig. 4.6 (b)), before computing the final canvas surface. We position the curve skeleton to maximize the symmetry of body

parts. (Section 4.4.1). Using only conformity and simplicity to compute the canvas geometry around this curve skeleton leads to plausible individual part geometries, but ignores the shape correlation between adjacent body parts outlined with a single contour. Contour persistence (Fig. 4.5 (c)) argues for these joint contours to retain their shape when the viewpoint changes, and especially to avoid introducing sharp discontinuities [138]. Accounting for simplicity alone can introduce such undesirable artifacts (see Fig. 4.5 (c)) and the accompanying video). We therefore enforce persistence across the character model by restricting the change in local profile slope with respect to its corresponding axis, allowing trajectory shape to deviate from a perfect circle to accommodate this constraint (Section 4.4, Fig. 4.3).

4.3 Part Segmentation

Existing research on skeleton-assisted part segmentation of 3D shapes [6, 29] employs a number of perception-driven segmentation criteria, variants of which apply to the segmentation of 2D contours (Fig. 4.7). The primary criterion is topological - in 3D each bone corresponds to a single segment, and segments are adjacent only if the corresponding bones are. The secondary criterion is bone proximity - segments are computed so as to be closest to their associated bones. Lastly, while the placement of segment boundaries is dominated by proximity to the corresponding bones, boundary locations are aligned with local curvature extrema on the surface to better match bends at skeletal joints. In describing how to apply these criteria for 2D contour segmentation we first address the simpler, occlusion-free setup, and then describe the extension to the general case.

Bisector-Based, Occlusion-Free Contour Segmentation Absent occlusions, the contour of a drawn character is a single closed curve. In this scenario (Fig. 4.7) each terminal bone corresponds to a single segment and each interior bone (purple in the Figure) corresponds to two segments, one on each side. A circular "half-edge" traversal of the contour uniquely defines the connectivity between the segments (Fig. 4.7(b)). We can therefore generate a segmentation by appropriately positioning the boundary points between these topological segments. While we can optimize for proximity by segmenting the contours using the Voronoi diagram of the bones (Fig. 4.7(c)), as-is this segmentation results in a different, undesirable, segment connectivity; note in particular the green and blue segments at the bottom. However, using a subset of the diagram intersections - specifically,

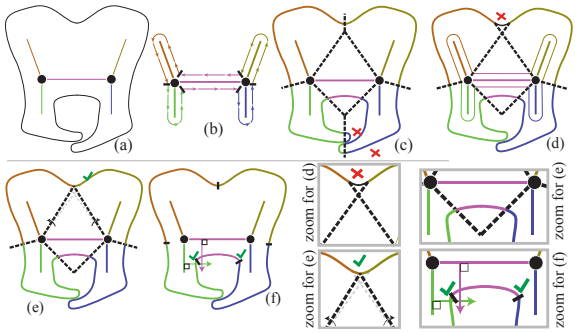


Figure 4.7: Skeleton-driven segmentation of a simple contour (a) must match skeletal topology (b) and reflect bone proximity. Proximity alone does not guarantee skeleton matching segment topology (c). A more topologically consistent segmentation (d) may need to be refined by bisector rotation to avoid segment overlap (e). Boundaries are then adjusted to best align with negative curvature extrema (f).

the first intersection between the contour and a ray emanating from each skeletal joint along its angular bisector - to define boundaries of contour segments associated with the participating bones (Fig. 4.7(d)) - results in a solution largely consistent with the circular ordering. Inconsistencies show up only at locations where the contours veer far from the skeleton; at these locations bisector rays starting at adjacent joints can cross prior to intersecting the contour, resulting in ill-defined, overlapping, segments. Such interior intersections can be trivially detected and fixed by rotating the offending bisectors in opposite directions to move the intersection onto or outside the contour (Fig. 4.7(e)). The resulting segmentation has the desired connectivity and each segment is adjacent to its associated bone. As a last step, we adjust boundary locations to align them with bends at skeletal joints by moving them to nearby curvature extrema (Fig. 4.7(f)).

Contour Segmentation with Occlusions While the algorithm above works extremely well for occlusion-free closed contours, real-world character contours contain inter-part occlusions which pose two further challenges (Fig. 4.8). First, in the presence of occlusions, 2D distances are not a reliable proxy for 3D distance; in Figure 4.8(b), for example, the contour between the pinkie and ring finger bones is closer, in 3D, to the ring finger bone despite being closer in 2D to the pinkie bone. Second, occlusions fragment the

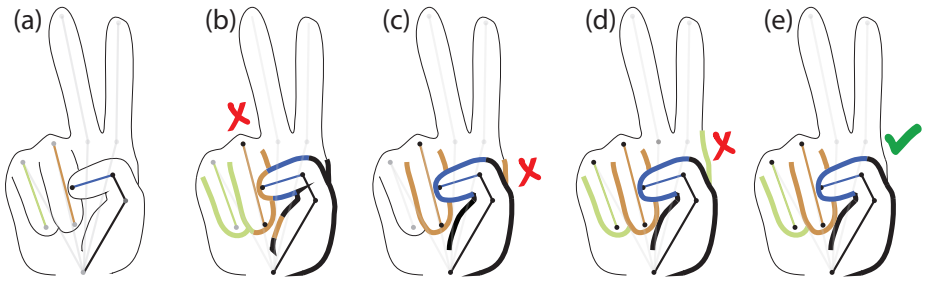


Figure 4.8: A character drawing with inter-part occlusions contains multiple contour curves and the left and right outlines of a body part may now contain multiple Gestalt continuous segments (a); thus 2D proximity based segmentation is no longer adequate (b). Taking into account skeletal depth as well as 2D proximity but neglecting Gestalt continuity leads to better, but still perceptually wrong results (c,d). Our framework accounts for both considerations resulting in the desired segmentation (e).

single closed contour into multiple disjoint contours, complicating the use of topological criteria for segmentation. When contours are fragmented a bone can be associated with any number of disjoint segments; e.g. in Figure 4.8, the terminal bone of the partially occluded ring finger should be associated with two disjoint contour segments. Furthermore, adjacent skeletal bones may correspond to segments on different contour strokes. Nevertheless, as we discuss below, the overall bisector-based segmentation strategy remains applicable, but requires modifications that leverage the depth information provided by the input 3D skeleton to better estimate proximity, and use Gestalt continuation to analyze disjoint contours.

2D to 3D Proximity We first note that 2D proximity is still a good proxy for 3D proximity; a bone can be associated with a farther away contour using the bisector based approach **only** if the body part associated with this bone is partially occluded and the contour in question belongs to the occluder. For typical 3D character geometry, the depth ordering between bones reflects depth ordering between their corresponding body parts, as well as their contours. Thus in general, a contour closest to a bone in 2D, should be associated with a different bone only if that bone is nearer to the viewer than the original one. While it is conceivable to create geometry and poses that violate this assumption, drawings of such shapes are inherently ambiguous even to human observers and are thus beyond the scope of this

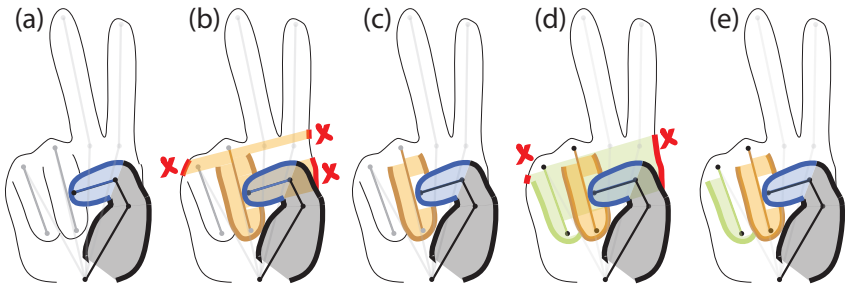


Figure 4.9: Segmentation algorithm: iterating between a z-ordering based pass and consistency validation.

chapter. Consequently for closest to the viewer bones we can still use 2D proximity as a reliable proxy for its 3D counterpart. Similarly, for bones farther away we can still continue to rely on 2D proximity as long as we ignore, or skip, contours associated with nearer to the viewer bones.

While a total depth ordering of bones may not exist, a total ordering of *mini-bones* is readily created by precisely subdividing bones that overlap in depth (*a la* the painter's algorithm) or approximately by simply subdividing all bones into mini-bones of some small maximum length (one tenth of the shortest bone in our implementation). As discussed below, the latter approach helps address the one-to-many bone to segment matching problem, as we can plausibly assume that each mini-bone has at most one visible contour segment on each side. Mini-bones resulting from subdividing a skeletal bone are seen as meeting at unarticulated valence two joints.

Topological Consistency The bisector-based segmentation algorithm for occlusion-free inputs ensures topological consistency along the closed input contour - that is, adjacent bones are mapped to adjacent, continuous, contour segments. When occlusions are present, adjacent mini-bones can be associated with different, disjoint, contour segments (Fig. 4.8(c,d)) or alternatively with hidden, or imaginary, segments. Unlike the occlusion-free case, a traversal of a single input contour curve in a circular fashion does not induce a traversal of the skeleton and vice versa; at most, we can hope that, as we traverse along mini-bone half-edges on the skeleton using the same counter-clockwise traversal, the associated contour portions should either be continuous, or plausibly connected by an obscured contour portion. We argue that humans employ the Gestalt continuity principle to evaluate association probability in such cases, and ignore associations inconsistent

with this principle (Fig. 4.8(c,d)).

Rather than directly assigning contour segments to bones so that every assignment is Gestalt continuous, we employ a restart mechanism with a taboo list. After assigning mini-bones to contours, we evaluate all assignments of adjacent mini-bones for Gestalt continuation. When assignments are inconsistent, as is the case in Figures 4.8(c,d), the proximity criterion argues for keeping the correspondence for the segment closer to the bone in 2D, while disassociating the segment further away from the bone. If and when an assignment is deemed inconsistent, we restart the near-to-far processing algorithm as the disassociated segment needs to be associated with a different bone.

Final Algorithm Our final segmentation algorithm that accounts for both proximity and topological consistency proceeds as follows (Fig. 4.9):

- We traverse all mini-bones in near-to-far depth order (Fig. 4.9(a)). The rationale for the ordering is that shallower bones, closer to the viewer, have priority over deeper bones in associating with visible contours as a consequence of the 2D to 3D proximity linkage.

- For each joint of a mini-bone we compute two joint bisector rays in 2D, one on each side of the joint as described in Section 4.3, and associate each ray with the first intersecting contour segment that has not yet been mapped to a shallower bone. The ray intersections (from a single joint for a terminal mini-bone, or from two joints on the same side of internal mini-bones) demarcate contour segments that are mapped to the mini-bone. Joint bisector ray intersections for deeper bones segment and associate with the closest intersecting contour segment that has not already been mapped to a shallower bone. The orange bone for example, does not associate with the tip of thumb since this tip is already mapped to the shallower blue bone in Figure 4.9(b).

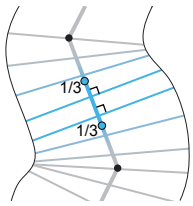
- Once all the mini-bones for a sequence of bones connected via valence two joints have been traversed (or an individual bone if it has no valence two joints), we evaluate the contour segments associated with these mini-bones for Gestalt continuity (Section 4.3).

- If erroneously mapped contour segments are detected, we disassociate them from their current bones, forbid them from being associated to these bones in the future, and restart the algorithm. In Figure 4.9(b),

once the incorrectly associated segment on the right side of the hand is found, we restart the algorithm and prohibit the ring finger from associating with that segment. In the next iteration we generate the configuration in Figure 4.9(c). Similarly, a new incorrect segment for the pinkie bone is found and the algorithm is restarted (Fig. 4.9(d)). Finally, we finish with the correct assignment in Figure 4.9(e).

Mini-joint bisector rays

Strictly speaking the joint bisectors for internal mini-bones are simply the two opposing directions orthogonal to the bone in 2D (blue in the inset). For mini-bones close to the end-joints of an original bone, such orthogonal internal bisectors are likely to intersect the joint bisectors emanating from these end-joints before reaching the contours resulting in overlapping segments which would need to be fixed later on (Section 4.3).



To reduce the number of subsequent fixes we preemptively rotate the internal bisectors. Specifically, we split the bone into thirds; the joint bisectors of the mini-bones in the middle third are left orthogonal to the bone, while at both ends of the bone we set the internal bisector angle to smoothly change from orthogonal to aligned with the end-joint bisector (see inset).

Evaluating Segment Continuity

For each pair of rays bounding a mini-bone, or sequence of mini-bones, we evaluate whether the mini-bone joint assignments are consistent with the Gestalt continuity principle by testing if their associated contours are perceived as a continuation of one another. We consider all the possible scenarios enumerated in Fig. 4.10:

A. In the most common scenario where both rays intersect the same contour segment (Fig. 4.10(a)) this contour is clearly continuous.

B. If neither ray is associated with a contour intersection (Fig. 4.10(b)) we similarly deem the assignments as consistent; this case suggests that the contour segment associated with the mini-bone chain between them is occluded.

C. In more rare cases, the two rays intersect different contour segments immediately next to a shared T-junction (Fig. 4.10(c)).

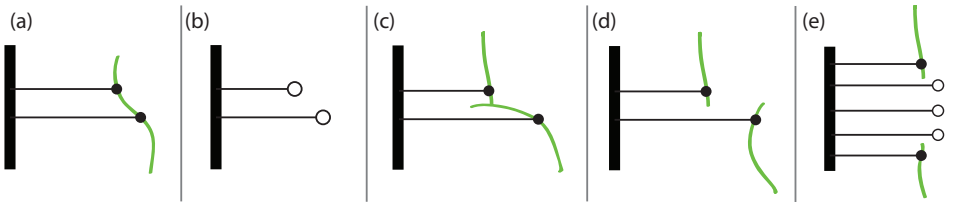
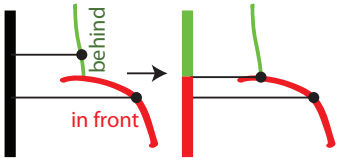
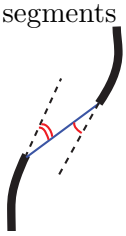


Figure 4.10: Possible scenarios of contour intersections (filled circles) for rays bounding a mini-bone. Empty circle means the ray has no associated contour intersection.



This scenario is consistent with a local occlusion (see inset). To associate each mini-bone with a single contour, we move the intersection point on the occluded contour (see inset) to the T-junction. The current mini-bone is now associated only with the occluding contour.

D. In the fourth scenario, the two rays may intersect different contour segments while not next to a common T-junction (Fig. 4.10(d)).



This is the first scenario where Gestalt continuity needs to be taken into account to decide if the assignment is topologically consistent. According to perception studies [55], more than 90% of viewers visually connect disconnected curve segments into a single contour if the angles between the segments and a straight line connecting their end-points (see inset) are less than 18°. We employ this test as-is to evaluate *Gestalt continuity* for pairs of ray-contour intersections along different contours. If the two contours are deemed discontinuous, we assume that the ray intersection, or contour assignment, that is closest to the bone in 2D is more likely to be correct, and disassociate the farther away contour segment.

E. One ray intersects a contour segment and the other ray has no associated intersection (Fig. 4.10(e)). Here we test whether Gestalt continuity is satisfied across a sequence of mini-bones that have no associated intersections due to occlusion using the same test as above.

Rotating intersecting rays

Similar to the occlusion free scenario, if two rays intersect prior to intersecting the same contour curve, they conceptually create overlapping segments. Thus, to preserve consistency we rotate them to flip intersection order. We apply the same rule to rays intersecting disjoint but Gestalt continuous curves, using the criterion above to determine continuity.

Once the distance and continuity driven segmentation is complete, we locally slide the boundary points on their associated curves towards local curvature extrema. Whenever a section of a contour remains unmapped, we split it between the closest adjacent mapped segments. In our experiments the resulting contour segmentations agreed with viewer intent (see Section 4.5.3), and we never observed an entire curve left unassigned.

4.4 Canvas Modeling

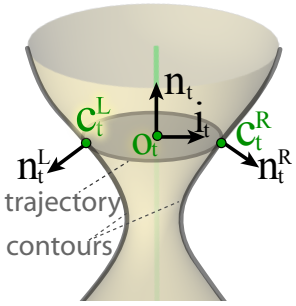
A canonical surface of revolution can be computed analytically from its 3D axis of revolution and its 2D contours (Fig. 2.2 (top)) by first positioning the contours in 3D by leveraging rotational symmetry at all contour points, and then defining the surface by setting the radius of revolution at each point on the axis to the orthogonal distance from the axis to the 3D contours [137]. In this scenario, the part segmentation computed in Section 4.3 would be sufficient to precisely define a 3D canvas for contour drawings that depict canonical surfaces of revolution around corresponding bones of the input 3D skeleton. Unfortunately, character body parts are rarely perfectly symmetric. Furthermore, our input, artist-provided 3D skeletons are typically only a coarse piece-wise linear approximation of a geometrically centered exact *curve-skeleton* [29] of the target character surface (Fig. 4.11).

To recover a plausible canvas surface despite inexact skeleton posing and imperfect part symmetry we use a three-step process. We first compute a 3D curve-skeleton which is close to the artist defined straight-line one, but well-centered with respect to the drawn contours (Section 5.1). We then use continuity along contour curves to determine the canvas connectivity across input skeleton joints, and construct a quad dominant mesh to represent the canvas (Section 5.2). Finally, we compute the optimal 3D vertex positions across the canvas (Section 5.3), balancing rotational part symmetry with respect to the curve skeleton against contour conformity and persistence.

4.4.1 Computing a 3D Curve-Skeleton and 3D Contours

We define the curve skeleton to have the same topology as its straight-line counterpart, and aim to position each branch so that it is maximally centered with respect to the contours of its corresponding body part. We initialize the curve skeleton by evenly sampling the straight-line skeleton, adding samples along the continuation of terminal bones until the point where that continuation's projection into 2D space intersects with a drawn contour, to support surface formation in these areas. Each curve skeleton vertex o_t is associated with a planar trajectory t with the vertex serving as its origin. We simultaneously compute the positions of both the curve skeleton vertices and the right and left contour points on their trajectories, balancing contour symmetry with respect to the 3D curve-skeleton, similarity between the curve- and straight-line skeletons, and 3D contour smoothness subject to input conformity (Fig. 4.11).

Symmetry In our computation we seek three-fold symmetry. First, we aim for left and right contour curves to be maximally mirror symmetric around the curve skeleton. Given a planar trajectory with center o_t and normal n_t that intersects the 3D contours at points c_t^L and c_t^R, mirror symmetry is satisfied if the contour points are symmetric around the plane (with plane normal i_t) containing the axis of revolution ((o_t, n_t) and the view direction (z-axis) as shown in the inset. We also seek local front-back symmetry at each contour point expressed as an expectation for the surface normal along the contour to be inside the plane spanned by the local axis of revolution and the contour point. Finally, to optimize the rotational symmetry of the surface profiles connecting adjacent trajectories we expect the lines connecting adjacent trajectory origins to be aligned with their respective normals. The combined symmetric energy is formulated as,



$$E_s = \sum_t \|(c_t^L - o_t) \cdot i_t + (c_t^R - o_t) \cdot i_t\|^2 + \left(n_t^L \cdot (n_t \times (c_t^L - o_t))\right)^2 + \left(n_t^R \cdot (n_t \times (c_t^R - o_t))\right)^2 + \sum_{(t',t)} \|(o_{t'} - o_t) \times (n_t + n_{t'})/2\|^2 \quad (4.1)$$

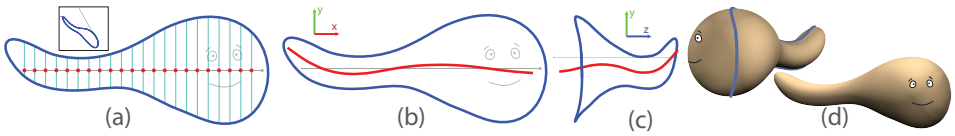


Figure 4.11: Curve skeleton computation: (a) user posed straight-line skeleton with the initial trajectory centers and their corresponding trajectory contour points marked; (b,c) front and side views of curve skeleton and 3D contours; (d) final surface with contours highlighted.

where $i_t = n_t \times (0, 0, 1)$, and the last term's summation index (t', t) represents all adjacent pairs of trajectories. The first term expresses the mirror symmetry between contours; the next two express the local front/back symmetry at each contour; and the last term encodes origin alignment. Since this term is direction invariant, we explicitly constrain the lines connecting pairs of adjacent trajectory origins to have the same orientation as the normals $(o_{t'} - o_t) \cdot (n_t + n_{t'}) > 0$ with consistently oriented n_t and $n_{t'}$. Lastly to ensure trajectory planarity we enforce

$$(c_t^L - o_t) \cdot n_t = (c_t^R - o_t) \cdot n_t = 0. \tag{4.2}$$

Skeleton similarity Since we expect the artist skeleton to approximate the target curve skeleton shape, we minimize the distance between the joints j_c and j_l on the two skeletons,

$$E_c = \sum_j \|j_c - j_l\|^2. \tag{4.3}$$

Contour depth Finally, we minimize depth change along contours,

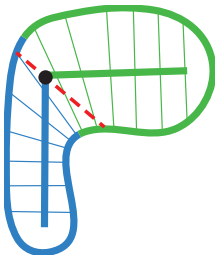
$$E_z = \sum_{(c_t^s, c_{t'}^s)} (c_t^s.z - c_{t'}^s.z)^2 \tag{4.4}$$

where c_t^s and $c_{t'}^s, s \in \{L, R\}$ are consecutive points on the same contour curve. This term is most important at joints, where it communicates depth information between adjacent body parts.

In the combined energy functional, symmetry and skeleton similarity are assigned unit weights, while contour depth is assigned a smaller weight of 0.1:

$$E = E_s + E_c + 0.1E_z. \tag{4.5}$$

Trajectory Normal Computation Simply including the trajectory normals n_t as unknowns in Equation 4.5 results in a highly nonlinear formula that is challenging to optimize efficiently. We therefore reduce the energy complexity to a simple quadratic formulation by independently precomputing these normals. In general we expect trajectory normals to be close to the directions of the straight-skeleton bones that the trajectories are originally associated with. On a curved skeleton, however, we expect these directions to change smoothly at valence 2 joints. We use the segmentation to determine the best transition angle by considering whether the joint has visible segment boundaries associated with it (see inset). If so, we rotate the axis at the curve-skeleton vertices closest to the joint so that the plane will intersect the contour just next to the boundary point. If both boundaries are visible we use an average rotation to best fit both, while if no boundary is visible we rotate the axis to the relevant joint bisector. We then smoothly propagate the rotation along the bones. Note that those rotations may differ from the joint bisector, shown as a red dashed line in the inset.



Contour-Skeleton Matching To account for input contour shape, we need to match the curve-skeleton vertices *of each* bone with densely sampled points on the input contours that we previously associated with this bone during our segmentation process. Incorporating the search for best skeleton/contour correspondences into the curve-skeleton computation is both challenging and time consuming. We therefore pre-compute the matches by leveraging the expectation that contour points on each trajectory are mirror symmetric around the local trajectory axis. This expectation implies that the line connecting such pairs of points should be orthogonal to the 2D projection of the local axis. To compute the correspondences for each initial curve-skeleton vertex, we shoot rays left and right orthogonally to local trajectory axis n_t to locate pairs of intersections on contours belonging to opposite sides of the body part. Note that in the presence of occlusions we may locate only one such intersection, or no intersections at all. These intersections are used as the image space locations of the corresponding contour points and are fixed throughout the optimization process.

We consequently solve for the 3D positions of the curve-skeleton vertices and the depth of their associated contour points using a quadratic solver that minimizes the combined energy function subject to the equality and inequality constraints above. We then compute the radii r_t of each trajectory

as the average distance from its origin to its two contour points and use those in the subsequent canvas mesh computation step.

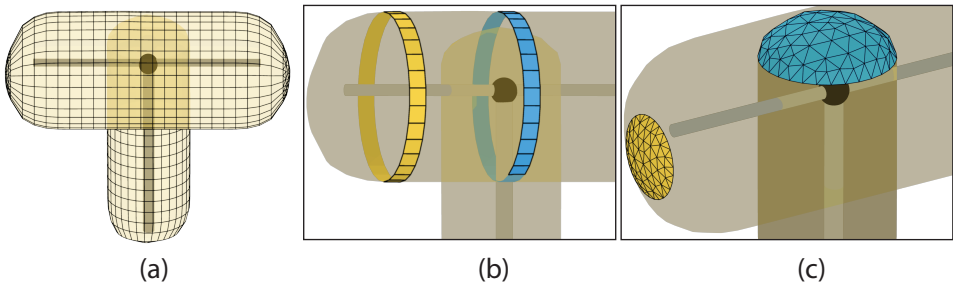


Figure 4.12: Canvas connectivity (a) with close-ups of quad strips between trajectories (b) and triangulated terminal trajectories (c).

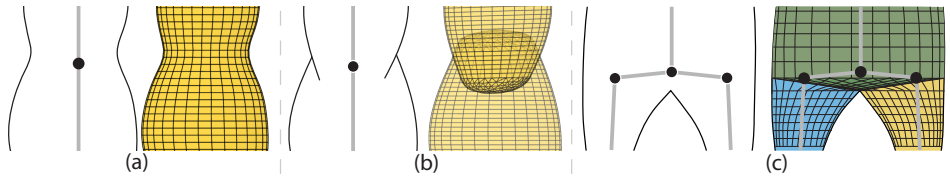


Figure 4.13: Connectivity across joints: (a) visually continuous parts; (b) Discontinuous parts; (c) the top part is deemed continuous with both lower ones, while the two bottom parts are deemed discontinuous since their shared contour curve has a cusp between them.

4.4.2 Canvas Connectivity

We represent the canvas using a set of planar, closed vertex cycles, or *trajectories* circling the skeleton, connected by a quad-dominant mesh. (Fig. 4.12). We place cycles around each trajectory center computed in the previous curve-skeleton computation stage; all cycles have the same number of vertices and a consistent circular indexing facilitating explicit angular and axial parameterization of the parts. We then form quad strips between pairs of adjacent cycles along each skeleton bone placing edges between vertices with same angular index on both (yellow strip in Fig. 4.12(b)) and triangulate the last, terminal, cycles at each terminal joint (yellow, Fig. 4.12(c)). The connectivity choices at interior joints are determined based on the interaction between the drawn outlines of the participating parts. Specifically, for

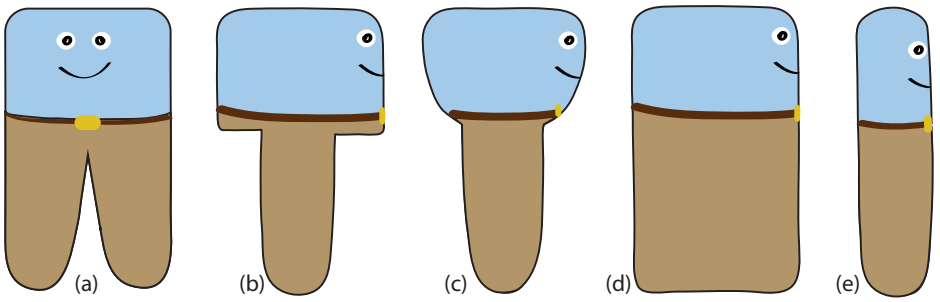


Figure 4.14: Given the input sketch (a), contour persistence indicates that side view contours (b,c) significantly differing from front-view ones are undesirable. Viewers similarly do not anticipate extreme foreshortening (d). Our result (e) is persistent with the front view contours.

each pair of parts adjacent to a joint we determine if the parts are a continuation of one another or not. If two body parts are deemed continuous we fuse their canvas surfaces, placing a quad strip between the part trajectories immediately adjacent to one another across the shared joint (blue strip in Fig. 4.12(b)). If a part has no continuous neighbors across an interior joint, its last cycle at the joint is simply triangulated (blue, Fig. 4.12 (c)).

Two parts are deemed continuous if their outlines are either adjacent to one another along a single smooth contour curve or are Gestalt continuous (Fig. 4.13). We deem a contour curve smooth if it has no cusp at the boundary between the two outlines. This smoothness requirement is motivated by the observation that artists frequently omit drawing small T-junctions, connecting what in 3D should be separate contours into a single, albeit non-smooth one (Fig. 4.13(c)). Our joint processing can, by design, lead to non-manifold, as well as self intersecting canvases. If desired, the surfacing step (Section 4.4.3), which leverages our current canvas connectivity, can be followed by a more complex fusion process similar to [8, 17] resulting in a smooth manifold mesh. However, we found this step unnecessary for the canvas applications shown in this chapter.

4.4.3 Canvas Surfacing

The key step in computing the canvas shape is to position trajectory vertices balancing the goals of maximally symmetric body parts, contour conformity, and persistence. The remaining vertices, those in the triangulated regions next to terminal trajectories, are computed in a post-process which seeks

for smooth canvas geometry overall.

We constrain each trajectory t with vertices $v_0^t, \ldots, v_n^t = v_0^t$ to be orthogonal to the previously computed normal n_t,

$$(v_i - v_{i-1}) \cdot n_t = 0 \quad i = 1, \ldots, n.$$

Part Symmetry To maximize part symmetry we seek canvas trajectories which are as circular as possible and aim for profiles connecting consecutive trajectories along each bone to have as constant as possible angle of revolution, or slope, with respect to each trajectory's axis. We cast circularity as a quadratic energy term,

$$E_c(t) = \sum_i (v_i^t - (v_{i-1}^t + v_{i+1}^t)/2 - \delta_i^t)^2 \tag{4.6}$$

where the vectors δ_i^t are the Laplacian coordinates of the i'th vertex in a planar circle whose normal and radius are the pre-computed n_t and r_t. To account for different axes of revolution assigned to different trajectories, we express profile symmetry for each trajectory t and a neighboring trajectory t' as

$$E_p(t, t') = \sum_i (v_i^t - M_{t',t} v_i^{t'} - R^n (v_{i+1}^t - M_{t',t} v_{i+1}^{t'}))^2, \tag{4.7}$$

where R^n is a rotation matrix of π/n around the axis (o^t, n^t), and $M_{t',t}$ is the shortest path coordinate transformation aligning the axis $(o^{t'}, n^{t'})$ with (o^t, n^t).

Conformity We want the visible contours of the canvas to match the artist drawn ones. To achieve this, the contour vertices on the final trajectories, i.e. those whose normals are in the view plane, must coincide in 2D with the previously computed trajectory contour points c_t. While we do not know the final trajectory shape, we assume that this shape will remain close to the ideal circular one; we therefore select the left and right vertices whose normals on these ideal trajectories are most orthogonal to the view direction as the potential contour vertices. For each such vertex v_t, if a matching (left or right) trajectory contour point c_t exists we force their 2D locations to coincide,

$$v^t.x = c^t.x \text{ and } v^t.y = c^t.y.$$

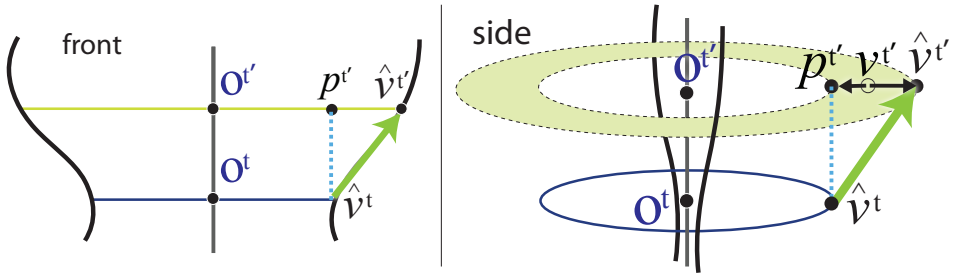


Figure 4.15: We constrain the profile angle to the range between the ideal profile slope given by the two ring radii and the axis direction.

Persistence Previous work has relied upon part symmetry and contour conformity alone when attempting to recover 3D models from character drawings. This produces intuitive individual part geometries, and plausible transitions between both discontinuous parts and those deemed continuous along both side contours; however, it also generates sharp depth discontinuities, contradicting viewer perception, between parts classified as continuous along only one side contour, such as a leg and a torso (Fig. 4.14(b)). The reason for such discontinuities is that in these situations the trajectories adjacent across the joints typically have vastly different radii and far apart centers. Since the artist contours provide no hint of discontinuity, we believe that viewers mentally eliminate them by deforming the parts to bring them closer together. Moreover, we speculate that viewers expect the character contours to maintain their overall drawn shape in alternate views up to inevitable foreshortening, avoiding the behavior visualized in Figure 4.14(c). This observation is supported by the minimal-variation principle observed by [138]. Following these observations we incorporate persistence into our setup as follows. When two parts are continuous along only one side contour we explicitly minimize the depth variation along quad-strips connecting these parts,

$$E_d(t,t') = \sum_i (v_i^t.z - v_{j(i)}^{t'}.z)^2 \tag{4.8}$$

where t is the trajectory with the smaller radius and $v_{j(i)}^{t'}$ is the closest vertex to v_i^t in image space, on the larger trajectory. We use vertex positions on perfect circular trajectories with centers o_t and $o_{t'}$ and radii r_t and $r_{t'}$ to compute these distances. Note that both values $v_i^t.z$ and $v_{j(i)}^{t'}.z$ are free, but the correspondences between their vertices $j(i)$ are fixed throughout the optimization.

To avoid creating discontinuities elsewhere, when two parts are continuous along both side contours, we minimize profile variation along the quad-strip joining them using Equation 4.7. This formulation leverages the slope along the two contours to optimize for depth variation consistent with viewer perception.

Lastly, to avoid undesirable derivative discontinuities (Fig. 4.14(c)) anywhere across the canvas surface we explicitly constrain the profile angle with respect to each axis of revolution to the range between the ideal profile slope given by the two ring radii and the axis direction (Fig. 4.15),

$$(v_i^{t'} - \hat{v}_i^{t'}) \cdot (\hat{v}_i^{t'} - \hat{v}_i^{t}) \leq 0$$
$$(v_i^{t'} - p_i^{t'}) \cdot (\hat{v}_i^{t'} - \hat{v}_i^{t}) \geq 0$$

Here $\hat{v}_i^t$ are the positions of the corresponding cycle vertices v_i^t on an ideal circular trajectory, and $p_i^{t'} = \hat{v}_i^t + o^{t'} - o^t$. In Figure 4.15, for the trajectory t' with an adjacent trajectory t, those two inequalities constrain vertex positions along t' to lie within the green ring whose boundaries are derived from the contour slopes between the pair of trajectories t and t'.

Given the terms above we proceed to optimize symmetry and persistence at joints subject to the trajectory planarity, conformity and profile slope constraints listed above:

$$E = \sum_t w(r_t) E_c(t) + \sum_{(t,t') \in B \setminus J} E_p(t,t') + \sum_{(t,t') \in J} E_d(t,t'), \qquad (4.9)$$

Here B is the set of pairs of canvas trajectories connected by a quad strip and J is the subset of such pairs with only one-sided contour continuity across joints. To promote the preservation of smaller trajectories, where even a small absolute error introduces large deviation from the ideal circular shape (Fig. 4.14(d)) we introduce per-trajectory weights $w(r_t) = 25e^{-(r_t/2\sigma)^2}$ with σ set to one third of the average trajectory radius. All other terms in the functional are assigned unit weights. To avoid depth ambiguity, we fix the z coordinate of one vertex. We use a quadratic programming package [50] to obtain the desired minimizer.

The resulting canvas is smoothed using standard Laplacian smoothing, while weakly holding the positions of contour vertices to eliminate local artifacts that can emerge due to imperfections in the input contours and small surface discontinuities due to the use of range constraints. To position the vertices in the triangulated regions next to terminal bone tips we use a simple Laplacian formulation that enforces tangent continuity with the rest of the surface.

4.5 Perceptual and Design Validation

We perform three-fold validation of the key aspects of our algorithm: we evaluate artist ability to provide the desired inputs, compare our results to ground truth and artist drawings, and validate our segmentation algorithm via an informal evaluation.

4.5.1 Creating Overlaid 3D Skeletons

Current animation practice uses 2D character drawings, such as those used as inputs to our system (e.g. Fig. 4.1), as a visual reference to manually author a 3D character model in a symmetric canonical pose [83]. A 3D skeletal structure is then interactively created and positioned within this 3D model. Our workflow expects animators to effectively create a 3D skeleton without an explicit 3D model, and pose it directly over a 2D character drawing.

To ensure the viability of our workflow, we asked three Maya animators to create 3D skeletons over two ground truth drawings (Fig. 4.16). Two animators (purple and maroon in Fig. 4.16) first created a 2D skeleton overlaid on the drawing and then re-positioned joints in an alternate view to get a desired 3D pose. One (purple) further used a measurement tool to compare symmetric parts and then further moved joints in 3D in an attempt to equalize the lengths of symmetric parts. These skeletons show a discrepancy in the average 3D length of symmetric parts (8% avg., 14% max. for purple and 19% avg., 33% max. for maroon) in Fig. 4.16, c.

The third animator (blue) first used the drawing simply as a visual reference, to create a symmetric, canonical skeleton and roughly pose it in 3D. This 3D skeleton was then moved onto the drawing and the pose refined by rotations and symmetric scaling of parts, to satisfactorily overlay the skeleton in 2D on the drawing. We described this workflow to animator #2 (maroon), who concurred that despite the natural tendency to first oversketch a 2D skeleton on the drawing, a canonical 3D skeleton allowed animators better control over symmetry and part proportion. The brown skeleton in Figure 4.16(c) was easily created by animator #2 using this workflow.

All animators took between 5-10 minutes to create and pose these qualitatively similar skeletons in 3D. The above exploration gives us confidence that animators imagine the 3D pose of 2D character drawings consistently and with practice can capture this pose with a 3D skeleton, overlaid directly on a 2D drawing.

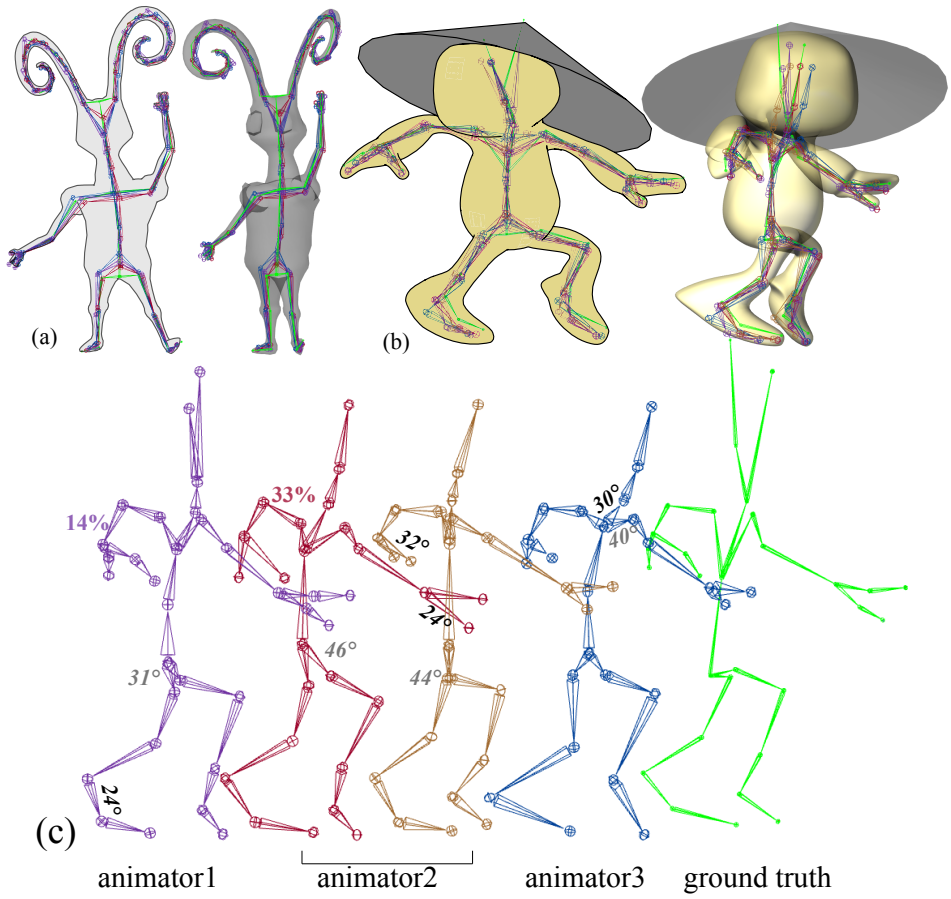


Figure 4.16: Ground truth (green) and 3D skeletons created by 3 animators overlaid on two ground truth 2D character drawings (a), (b), also shown from an alternate view overlaid on the ground truth 3D canvas. The skeletons in (b) shown individually (c). The purple and maroon skeletons, created by manipulating an overlaid 2D skeleton have differences in 3D limb length between symmetric limbs. The maximum difference for each skeleton, 14% and 33%, is marked on the longer limb. The brown skeleton was created by animator #2 mimicking the workflow of animator #3. The angular deviation between the corresponding bones on the ground truth and artist skeletons is dominated by control bones (hips and shoulders) which have no impact on the result geometry. The maximal deviations without (and with) control bones are: 24° (31°) for the purple skeleton, 24° (46°) maroon, 32° (44°) brown , and 30° (40°) blue. Average angle differences are 13°, 15°, 15°, and 18° respectively.

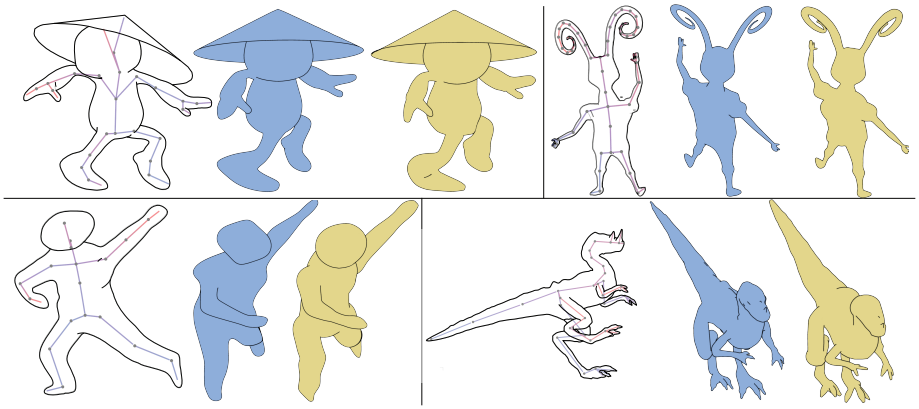


Figure 4.17: Comparing our results to ground truth data: Left to right: contours and skeletons of ground truth (GT) models; GT (blue) and our (yellow) models rendered from alternate views.

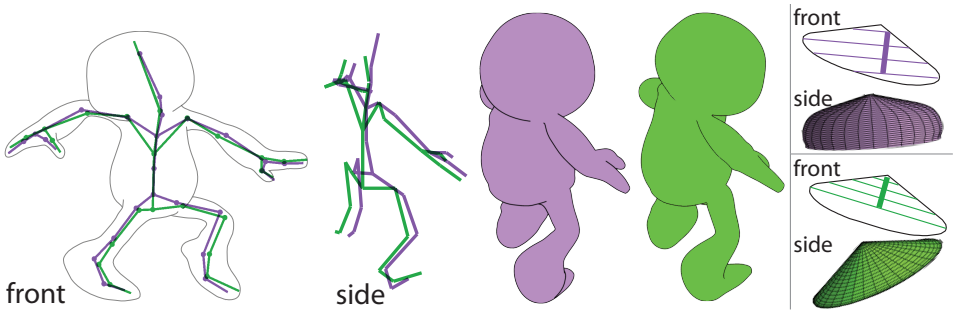


Figure 4.18: Left: Given the same input sketch, small variations in skeleton posing (green and purple Figure in 4.16) lead to minor changes in character shape. Right: significant change in bone slope and location for a symmetric contour leads to larger shape difference.

Robustness to Input Variation We also examined the impact of using different artist skeletons on the canvases created by our system (Fig. 4.18). As demonstrated, while the character pose predictably changes with changes in bone posing, the body part shape remains largely fixed, thanks to our robust curved-skeleton computation stage. The invariance to minor posing changes is important, since artists are unlikely to pose a skeleton perfectly. The shape change is most pronounced (Fig. 4.18 (right)) when a bone for a perfectly symmetric surface of revolution is significantly misaligned compared to the expected axis. Such misplacement is easy to spot and fix. Overall, as long as the depth ordering of the bones is correct, the intrinsic geometry of our results changes only marginally with changes in 3D skeleton posing. In particular angle and depth changes (in this example we have bones orientations vary by up to 30°) cause only small difference in the results. The output is more dependent on the image space skeleton positioning, and in particular on how off-center the skeleton is with respect to the drawing. Artists can easily center skeletons in 2D.

4.5.2 Comparison to Ground Truth and Artist Drawings

To validate our method, we compare the canvases created by our algorithm to ground-truth canvases for given contours and to alternate view drawings created by an artist given the same input as our method. To perform the ground truth comparison, we had an artist create and rig four 3D models (Fig. 4.17). We then used contour only renders of these models and the artist skeletons as input to our method. The resulting canvases are extremely similar to the original models, validating our design choices.

We further validated the perceptual correctness of our framework by comparing these results to artist generated drawings of the input sketched characters in alternate views (Fig. 4.19). We provided another artist with our input drawings with the skeleton overlaid, a 2D view of the skeleton in the desired output pose, and a 3D posed skeleton which allowed the artist to better relate the two poses. We then asked him to redraw the input character matching the skeleton pose. The results were qualitatively very similar, though the artist's characters tended to be leaner than our interpretation.

4.5.3 Perceived Contour Segmentation

To evaluate consistency across viewers and compare our algorithm with viewer perception, we asked 12 viewers (from various backgrounds, 10 with

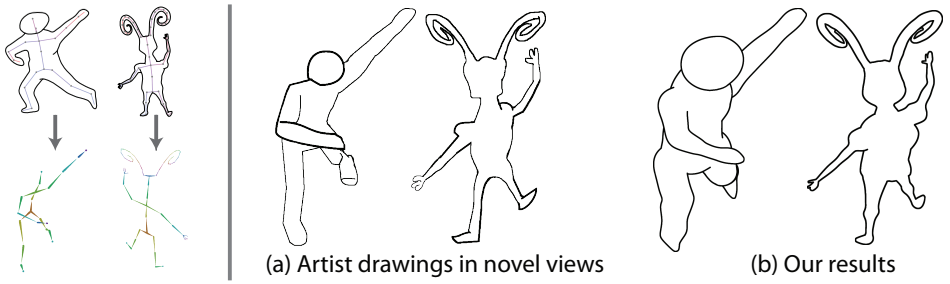


Figure 4.19: Comparison of our results (b) to sketches produced by artists (a) for the same view and pose.

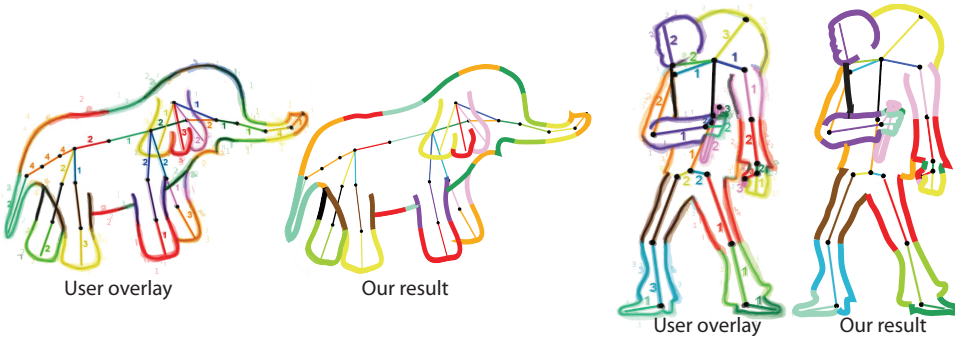


Figure 4.20: Overlaid user segmentations (left) for both the elephant and the scientist are qualitatively similar to the algorithmic results (right).

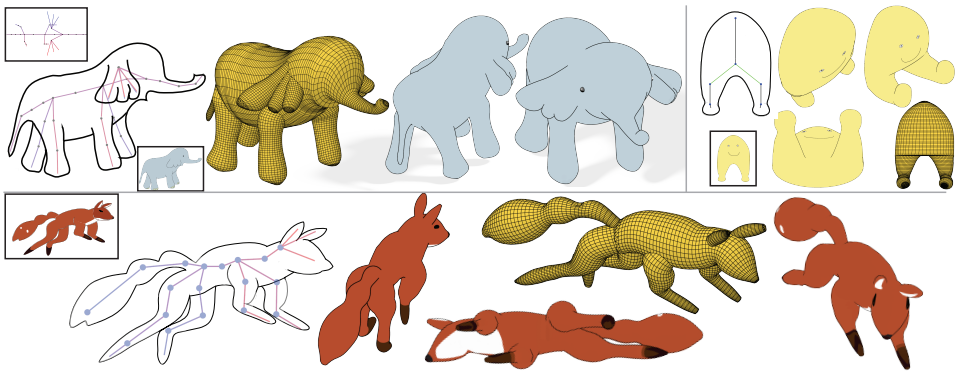


Figure 4.21: Canvases and alternate view renders generated using our system from the inputs on the right.

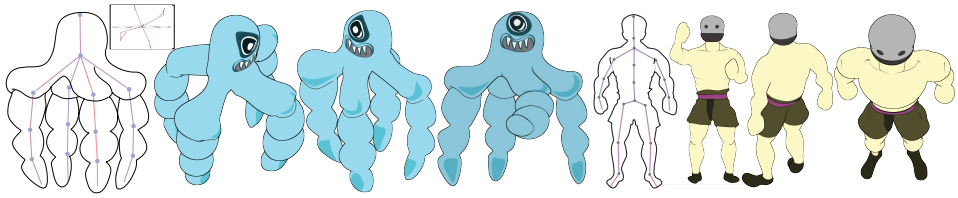


Figure 4.22: A variety of re-posed renders generated automatically from the inputs on the right.

no experience of 3D modeling, 8 females and 4 males) to hand segment the contours on four simple and five complex drawings and associate each segment with bones of an overlaid 2D skeleton. The full text and the results of the evaluation can be found at `http://cs.ubc.ca/~bmpix`. We chose to distinguish joints by color as numbered labels for skeletons with dozens of bones were visually confusing. None of the users remarked that color based segment association was problematic as a task. Fig. 4.20 summarizes the resulting segmentations on two complex inputs, with various user segmentations overlaid to visualize correlations across viewers. While viewers had less information than our algorithm (a 2D rather than 3D skeleton), their segmentations are largely consistent and match well our algorithmic segmentation. We thus believe that our 3D character canvas is built on a robust and perceptually meaningful contour segmentation algorithm.

Figure 4.23: The explicit cylindrical parameterization of our canvases allows for a range of advanced texturing effects, hard to achieve without a an underlying 3D structure.

4.6 Results

We demonstrate the results of our character canvas modeling framework throughout this chapter. We show both the actual canvases created by the method (Fig. 4.3, 4.21), as well as a range of NPR renders created using these canvases from different view directions in both the input and alternate poses (see Fig. 4.1, 4.22). The rendering examples include significant changes in contour topology compared to the input view, which cannot be achieved purely by 2D control (e.g. see back view of the catwoman Fig. 4.1). Using our canvases, with their built-in cylindrical parameterization, one can easily apply advanced rendering techniques such as fur or feathers simulation (Fig. 4.23), enabling artists to generate 3D effects without resorting to complex 3D modeling tools.

One of the main technical challenges, addressed by our method, and showcased by these examples is correct resolution of inter-part occlusions. Not only does it enable artists to draw characters in natural rather than artificial canonical poses, but it enables them to draw characters whose anatomy does not allow for such occlusion-free pose, e.g. one simply cannot draw a quadropus (Fig. 4.22) with both the head and all four legs fully visible. Other such examples include the fox and anteater (Fig. 4.21, 4.23).

Workflow The inputs we evaluated our framework on were created using two workflows motivated by different target applications. In the first one, an artist created a set of sketches, e.g. catwoman or elephant and then fitted a

skeleton to those using Maya or other animation software (see Section 4.5.1). This framework is best suited for creating new cartoon art and bringing to life legacy characters, where a drawing of the character already exists.

In the second workflow, artists created and pose a 3D skeleton first, and use it as an armature over which to draw character contours from an interesting viewpoint (fox, anteater, quadropus). This approach is particularly useful in animation setups where artists already have a skeletal animation sequence they want to adapt to a new character. The accompanying video shows several animation sequences, each generated from a single frame, created using this workflow. The amount of work required to generate these animations was drastically lower than using the traditional 2D animation workflow, where key-frames describing out-of-plane motion are typically drawn by hand.

Global Symmetry Besides local symmetry which is used throughout the algorithm, characters frequently exhibit left-right global symmetries, which viewers use to complete occluded body part geometry. We employ this principle in two examples to recover fully occluded geometry (elephant) or correct for inaccurate artist drawing (fox) by enforcing similar trajectory shape for matching trajectories on symmetric joints.

Impact of Design Choices Figure 4.14 demonstrates the importance of our design choices when surfacing the canvas. Not accounting for persistence at joints (Fig. 4.14(b)) results in unexpected surface discontinuities. Locally minimizing depth variation (Fig. 4.14(c)) is similarly insufficient. Our framework (Fig. 4.14(e)) which constrains profile slope and minimizes foreshortening produces more natural results.

Parameters and Runtimes Our method has no tunable parameters. For canvas modeling we use thirty vertices per trajectory and have uniform trajectory density across all bones; the density is determined so as to have at least ten trajectories along the shortest bone, and to have the distance between consecutive trajectories be no more than one percent of the character's bounding box diagonal. Our software takes between ten to sixty seconds to generate a canvas on an Intel Core i7 machine with 32GB of RAM. Roughly 25% of this time is spent in the segmentation stage and the rest is spent by the QP solver computing the canvas surface. This fast turnaround allows artists to quickly repose the skeleton or update the drawing were they to find the results inadequate.

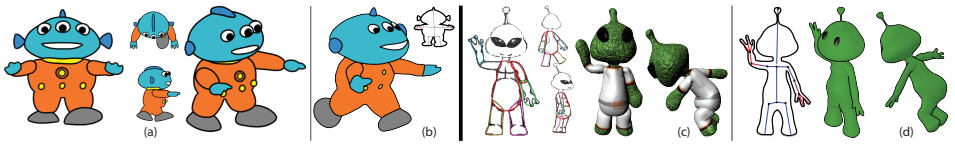


Figure 4.24: Given a single drawing and a posed skeleton we generate qualitatively similar results (b,d) to those created by multi-drawing systems which employ manually specified curve correspondences between drawn curves: [106] (a) and [74] (c).

Comparison to Prior Art Figure 4.24 highlights our ability to generate models of equal complexity to those generated by multi-view approaches such as [74, 106], without the need for multiple corresponding drawings. We performed this comparison by using one of the input views utilized by these prior systems, tracing 2D curves over it as our sketch input and posing corresponding skeletons. Our method employs significantly less user input than Levy et al, who require at least three corresponding drawings each with an appropriately posed skeleton. While Rivers et al. do not require a skeleton, they still expect at least three drawings with correspondences and cannot articulate the results.

We successfully handle a much wider range of sketches than previous methods, most of which, e.g. [20] can handle only occlusion free inputs. While Cordier et al. [28] support partial occlusions, they assume perfect rigid mirror symmetry, and expect every part silhouette to be drawn as a separate curve. Karpenko and Hughes [68] make a similar curve planarity assumption. Our framework successfully handles complex occlusions, including scenarios deemed ambiguous by previous methods (e.g. elephant in Fig. 4.21, see [68]); does not require posing symmetry (e.g. see the mad scientist) nor separate part outlines (e.g. see hind side of the fox), and plausibly recovers non-planar contours (see Fig. 2.2). As demonstrated by Figure 4.4, our shape computation which aims to maximize simplicity, generates results more consistent with user expectation than inflation based frameworks such as [68, 95]. By accounting for persistence (Fig. 4.14) our method avoids depth discontinuities at complex joints bound to show up when parts are assumed to have perfect rotational symmetry [20, 28].

Qualitative Evaluation. We asked six computer artists to provide visual critique of our outputs (catwoman, elephant, quadropus) by showing them the input drawings and the output renders (see Figure 4.25), and asking them if our results represent the same character as the input drawing. All

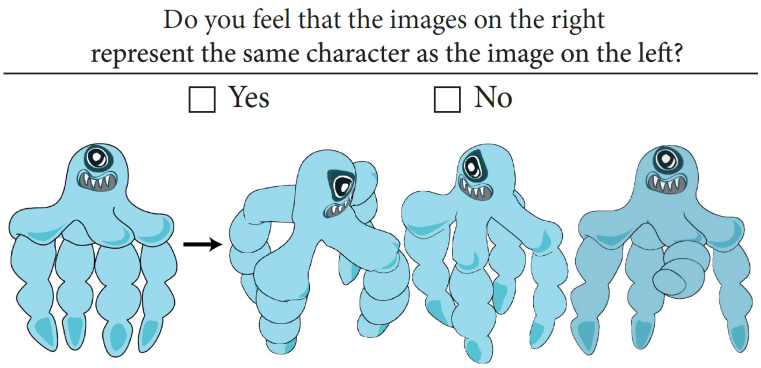


Figure 4.25: An example of the qualitative evaluation questionnaire.

six agreed that our results faithfully capture the original input in new poses and views and expressed strong interest in using our system in their work.

Limitations and Improvements. Like human observers, our method's ability to predict the shape of a character is inherently limited by the descriptive power of the input drawings, and our algorithm can be misled by badly posed or obfuscated drawings. For example, faced with an oblique view of a bird's wings, neither viewers nor our method can guess their depth without resorting to prior knowledge of bird anatomy (Fig. 4.26(a)). Since selecting a single view where all character body parts are well described can sometimes be challenging, we provide users with an incremental, overdraw interface. In this interface, users can first generate a character model from a single view, and then update the canvas from another view using contour overdrawing framework that follows [96] (Fig. 4.26(a)).

While our method is robust against minor inaccuracies in the input skeleton, major errors in skeleton depth placement may clearly cause undesirable artifacts such as intersections between body parts. We did not encounter such situations on the tested inputs. We believe that the simpler solution would be for the artist to adjust the skeleton, if and when they find the result unsatisfactory, and rerun the algorithm. However if desired, one can incorporate additional non-intersection constraints into the optimization in Equation 4.9, or fix the self-intersections as a post-process step once the canvas is generated. Regardless, we are still dependent on the ability of the artist to pose a skeleton with respect to a cartoon drawing in a manner that avoids intersection between body parts.

A fundamental premise of our work is that the 3D canvas is a collec-

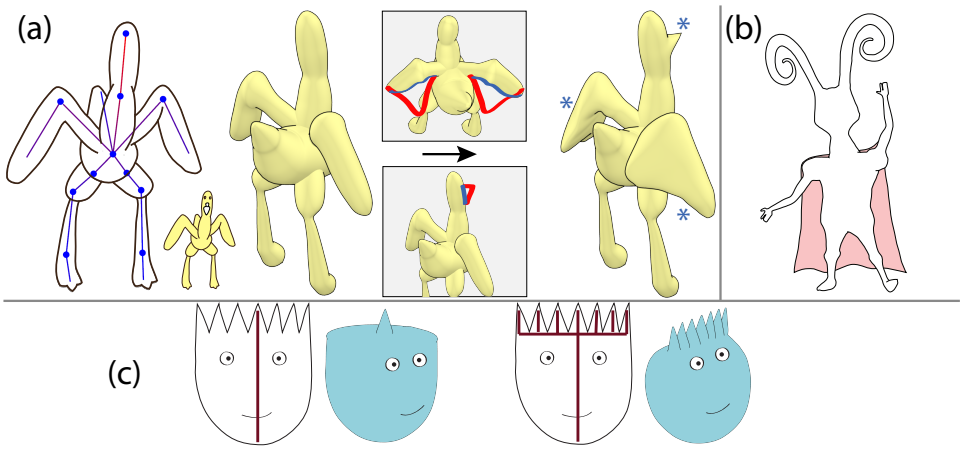


Figure 4.26: Our ability to plausibly recover character shape is limited by the descriptive power of the inputs. Without cues to the contrary we generate round bird wings, instead of anatomically correct ones (a). Since we use a standard mesh representation, the canvas can be easily edited to correct the wings or add extra features (beak) using standard tools (a, right). Geometries not-well represented by generalized surfaces of revolution, such as loose clothing (b, pink cape) must be modeled by other means. While some fine details can be captured by using skeleton refinement (c), alternate editing tools are likely to achieve this goal faster.

tion of generalized surfaces of revolution parts, each part being defined by a bone of the input 3D skeleton. Surface detail for a 3D canvas that strongly deviates from this premise, like cloth folds with internal occluding contours (Fig. 4.26(b)) are thus not captured by our approach. While the hair spikes of (Fig. 4.26(c)) can be constructed using surface of revolution parts, it is unlikely that artists would provide the necessary definition for each hair spike with a bone on the input 3D skeleton. Thus while our system is well suited for canvas creation, artists should combine it with other mesh-editing tools to generate detailed, dressed, characters. Some cartoon characters may have elements which are designed to consistently face towards the camera regardless of the viewer position (cartoon eyes, or the ears of a cartoon mouse); we do not attempt to recover these features from the input sketch. In a production environment such features are best implemented using billboard vector elements. In general, realistic cartoon drawings combine a mix of strokes that define a 3D canvas, view-dependent 3D geometry, and 3D detail drawn on and around the surface of the 3D canvas [113]. We have

focused on simplified cartoon drawings where the strokes strictly comprise a character canvas. The classification of strokes of arbitrary cartoon drawings as described, and their 3D reconstruction, is subject to future work.

4.7 Conclusions

We presented the first, to our knowledge, system for 3D character canvas modeling from a single naturally-posed character drawing and overlaid 3D skeleton. We can process input with complex inter-part occlusions and large variations in contour depth. As demonstrated, our output 3D geometry is appropriate for use as an animation canvas: facilitating non-trivial reposing and large viewpoint changes of complex characters, that remain consistent with the input drawing, and enabling non-photorealistic animation using painterly strokes on and around the canvas.

Our work is aligned with a recent trend to simultaneously model 3D character geometry and its corresponding skeleton [8, 17]. While we have focused on 3D proxy geometry creation from minimal input in the form of drawn contours, our coupling of 3D skeleton and input drawings using a perceptual framework is extensible. In the future we expect that our algorithmic approach, adapted to richer input drawings, embellished with internal contours, construction lines and shading, will result in fully detailed and complex 3D character models.

Chapter 5

Gesture3D: Posing 3D Character via a Gesture Drawing

5.1 Introduction

In this chapter we introduce a novel system to pose rigged 3D characters via a gesture drawing. A variant of this chapter has been submitted to SIGGRAPH ASIA 2016.

Gesture drawings - rough, yet expressive, contour drawings of posed characters (Figure 5.1b,e) - are routinely used by artists to quickly convey the action, form, and pose of a character figure [12, 53, 97]. Artists are trained to create *descriptive* gesture drawings which unambiguously convey a character's pose in just a few minutes [73], and use them ubiquitously when conceiving character poses and motion key-frames for storyboarding. In digital media production, artists subsequently apply these envisioned poses to 3D character models. In current practice, posing is performed separately, using the drawings as a reference only, and requires additional, often significant, user interaction (Section 5.2). We seamlessly connect the ideation and modeling steps by introducing the first method for 3D character posing which poses the characters algorithmically using gesture drawings as

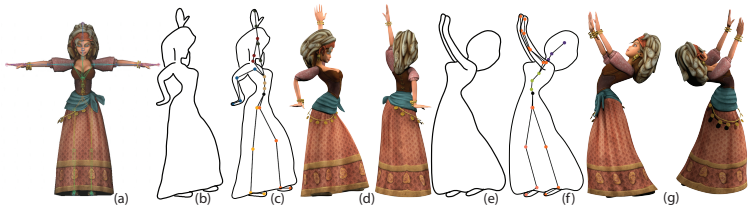


Figure 5.1: Gesture3D: gesture drawings (b,e) of an input character model (a); estimated 2D skeleton projections (c,f) and new poses automatically computed from the drawings (d,g).

input, allowing artists to directly communicate their ideas using drawings and sidestepping the mental overhead of interacting with a complex software interface. As demonstrated, our method plausibly poses 3D characters using quickly generated, rough, vectorized gesture drawings and rigged character models, provided in a neutral bind pose, as the only inputs. It successfully handles complex poses with varying and significant part foreshortening, occlusions, and drawing inaccuracies (Figure 5.1).

The advantage of gesture drawings over other types of 2D inputs explored by previous posing approaches (Section 5.2) is the lack of perceptual ambiguity. Unlike stick-figures, lines of action, and outer silhouettes (Figure 5.2), gesture drawings allow artists to *unambiguously* convey poses to human observers. By identifying and leveraging the perceptual pose cues used by artists when creating these drawings, we are able to automatically recover character poses that are consistent with artist intent.

Our framework centers around analysis of the stroke curves forming the gesture drawings (Section 5.4). Like many other line drawings, gesture drawings are dominated by contour curves, conveying the occlusion contours of the depicted characters. However, since gesture drawings focus on conveying pose rather than shape, they typically only depict approximate, abstracted, character anatomy. In particular, artists typically use simple low-curvature stroke segments to outline body parts and use higher-curvature sections to depict their connecting joints [53]. These high-curvature *anatomical landmarks* assist observers in parsing the drawings. The abstracted contour strokes of a gesture drawing are designed to convey largely smooth 3D character geometry. As observed in the previous chapter, in such scenarios the contours of both individual body parts and part chains are usually *continuous*; thus adjacent contour stroke segments always outline adjacent body parts, and adjacent body part outlines are typically depicted using one shared stroke, or multiple Gestalt continuous [69] strokes. We also observe that body part contours are consistently *oriented* with respect to the parts' skeletal bone and rarely *cross* the bone's 2D projection. Combined together, these three *contour consistency* cues allows observers to identify poses with globally consistent joint and bone locations.

Generic projected contours allow multiple depth interpretations, thus artists are trained to use drawing cues to reduce ambiguity. When estimating depth from 2D drawings, viewers prefer *less foreshortened* interpretations of the observed shapes, thus to best convey the intended poses artists seek to select viewpoints with smaller foreshortening [57]. We also observe that in gesture drawings artists prominently use local, suggestive, occlusions to convey changes in depth and to specify depth order between adjacent joints.

Gestalt psychology [5, 69] points to a persistent viewer preference for *simple* drawing interpretations. In the context of gesture drawings, we believe that viewers use two types of simplicity cues: in cases where drawings are ambiguous, viewers prefer more *natural* poses, or ones with angles closer to those in the input bind pose; as studied in the previous chapter, viewers also visually complete hidden body parts and correct drawing inaccuracies by using *regularity* cues, such as pose symmetry. Finally, we note that while human observers can clearly parse professional gesture drawings, reliance on the drawing to accurately depict character proportions and projected joint locations must be qualified: drawings are typically *inexact*, as even experts depict foreshortened objects inaccurately [114] and fail to correctly account for perspective [138].

Overview. We use these observations to pose the input 3D rigged character model into the artist intended pose conveyed by the input gesture drawing. We first match skeletal elements against corresponding contour stroke segments, placing joints next to their matching contours. We formulate joint placement as a discrete 2D embedding that matches joints to corresponding contour samples and is dominated by contour consistency and anatomical landmark matching considerations. We then compute the desired embedding by casting it as a variation of the tree-structured Markov Random Field (MRF) problem (Figure 5.9b, Section 5.5). We extend our solution to 3D by leveraging the depth order implied by occlusion contours, and the observations about viewer preference for simple and less foreshortened poses. To overcome drawing inaccuracy, we formulate 3D embedding as an energy minimization problem which balances landmark-implied 2D joint placement against the simplicity and foreshortening cues (Figure 5.9d, Section 5.7).

Contribution. Our contribution in this chapter is two-fold: we formulate the properties of effective gesture drawings, bringing together insights from multiple sources in the areas of psychology, art, and computer graphics, highlighting key perceptual cues which enable viewers to perceive the artist intended character poses; we then use these observations to introduce the first gesture drawing based algorithm for posing 3D characters. Our method enables artists to directly convert their ideated posed character drawings into 3D character poses, and supports complex drawings with occlusions, variable body part foreshortening, and drawing inaccuracies.

Validation. We exhibit a gallery of character poses obtained automatically from gesture drawings of a range of 3D characters (Section 5.9) and validate our algorithm in a number of ways (Section 5.8). We evaluate our results against ground truth data, by first rendering projected contours of

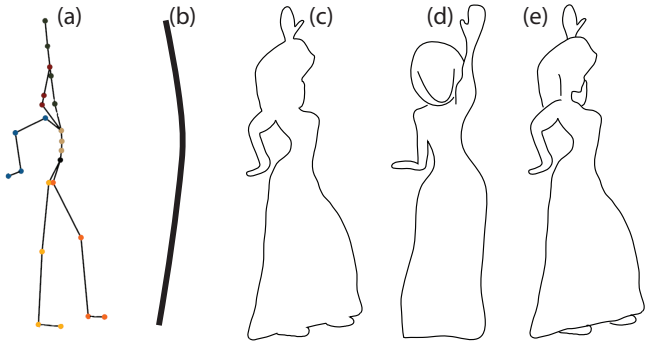


Figure 5.2: Stick figure drawings (a), lines of action (b), and outer silhouettes (c) allow for multiple perceptually valid pose interpretations. (d) Poor view selection results in highly foreshortened contours leading to loss of pose information (e.g bends on the left arm or the curved spine). Gesture drawings, consciously drawn from descriptive views (e) effectively convey the intended pose.

posed character models, then using these contours as input to our method and comparing our results against original poses; we compare our algorithm's results with characters posed by artists given the same drawings as input; we compare the character-contour correspondences computed by our method against manual annotation by human observers; and we collect qualitative result evaluations by experts and non-experts alike. Finally, we compare our method against prior work and algorithmic alternatives. These validations confirm that the poses we compute are consistent with viewer perception and artist intent.

5.2 Related Work

5.3 Parsing Gesture Drawings

Gesture drawings are ubiquitously used by artists to clearly convey complex 3D poses.To understand and formulate the properties that make them effective, we combine observations from drawing tutorials, modeling research, and perception studies.

Anatomical Landmarks In a typical character drawing, most strokes depict projected contours, i.e. curves along which the normal to the posed character's body lies in the image plane. Unlike detailed drawings of geo-

metric shapes, gesture drawings focus on depicting pose and motion; hence their contour strokes are often highly abstracted and only approximate the shape of the actual 3D contours. We note that gesture drawings employ idealized character anatomy, well described by a union of approximately cylindrical body parts connected by spherical joints [12, 53, 57] (see inset).

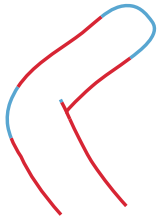


Figure 5.3: Portion of a gesture drawing with annotated joint (blue) and part (red) contours.

Consequently, contours of body-parts surrounding skeletal bones are typically dominated by low-curvature lines. In contrast, joint contours in all views are well approximated by circular arcs whose radii are roughly equal to the body radius around the joints. These higher curvature joint contour arcs are most prominent next to bent or terminal (single bone) joints. As a consequence of this curvature difference, we speculate that humans can easily discern the likely locations of such prominent joints, or anatomical landmarks, in a gesture drawing, and use those to anchor the overall character pose. Since artists seek to communicate their target pose, they typically select views where multiple anatomical landmarks are visible and clearly depicted [52]. Clearly not all high-curvature contour segments correspond to joints (see the skirt "corners" in Figure 5.1); many drawn joints are not bent and therefore not easy to pinpoint; and multiple joints may have the same radii, making them hard to distinguish. Our algorithmic challenge is to discern the relevant markers on the drawing and to associate them with their corresponding joints.

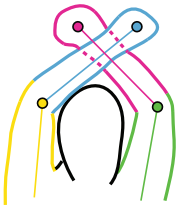


Figure 5.4: Contour-skeleton correspondences, with Gestalt continuous contours connected by dashed lines.

Contour Consistency. As noted by Bessmeltsev et al. [10], absent occlusions a typical character's contour is a single closed curve; each body part around a terminal bone (bone with a terminal joint) is outlined by a single contour segment, while parts around interior, or non-terminal, bones define two outline segments, one on each side of the bone; and adjacent segments along the contour correspond to adjacent skeletal bones (see inset). In the presence of occlusions, the *Gestalt continuation* principle [69] indicates that viewers complete the drawing by mentally connecting pairs of end-points of partially occluded curves (T-junction stems) by invisible contour sec-

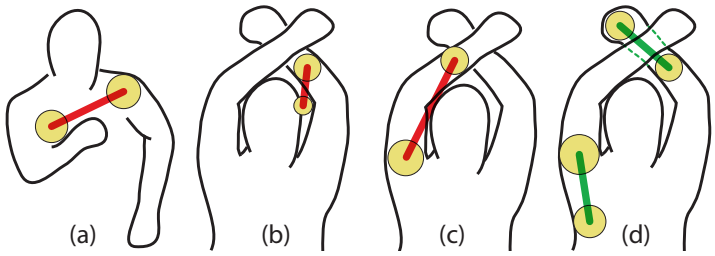


Figure 5.5: Implausible bone locations that violate (a) adjacency, (b) orientation, or (c) crossing cues; consistent placements (d).

tions if they can be smoothly connected (Figure 5.4). In this scenario, the properties above continue to hold once these invisible contour sections are taken into account. In this general case, terminal bones correspond to a single sequence of (one or more) Gestalt continuous curves, and interior bones correspond to two such sequences - one on the left and one on the right. Adjacent segments along the same contour stroke still correspond to adjacent bones, while bones joined by a valence two joint correspond to either immediately adjacent, or Gestalt-continuous, left and right contour segments. In addition to reflecting skeletal *adjacencies*, body part contours are consistently *oriented* with respect to their corresponding skeletal bones - a body's surface and consequently its contours clearly separate inside from outside (Figure 5.5b). Since body mass typically surrounds the bones, contours rarely *cross* 2D bone projections (Figure 5.5c). Viewers are known to rely on domain priors when deciphering drawings, and therefore we expect them to indirectly leverage this set of contour-bone *consistency* expectations when parsing gesture drawing and matching joints to landmarks.

Simplicity Previous graphics research (e.g. [138]) had heavily relied on insights from Gestalt psychology [69] which points to a viewer preference for *simple* or regular drawing interpretations. While some of these works (e.g. [138]) focus on generic *regularities* such as symmetry or parallelism, others (e.g. [10]) highlight domain-specific simplicity priors. We speculate that viewers leverage both regularity and naturality when interpreting gesture drawings: they choose more likely or *natural* character poses among those consistent with the drawn contours (Figure 5.8), and use *regularity* cues, particularly symmetry, when presented with different ambiguous inputs (for instance when mentally completing partially occluded poses, such as the hands of the character in Figure 5.8, or the fetal pose in Figure 5.14,

top row).

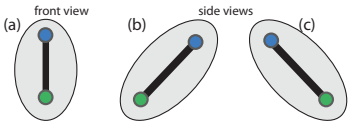


Figure 5.6: Depth ambiguity

Depth In general, an infinite number of 3D geometries have the same 2D projection. However, for each individual bone of a known length, if the 2D positions of its end-joints are known, the z-difference between the bone end-points is fully determined; what needs to be determined is their depth order (see inset).

While the simplicity priors discussed above often help viewers to resolve order ambiguities, contours of posed characters taken from a poor view-point (Figure 5.2d) remain ambiguous. Consequently, artists are consistently advised to strategically select descriptive views [36], and specifically to avoid views with large uneven foreshortening. Our observation of artist-generated gesture drawings suggests that in selecting views they also strategically use occlusions to clarify depth ordering, and add suggestive local, intra-part, occlusion contours (see inset) to further clarify local depth order.

Figure 5.7: Occlusion types.

Inaccuracy Experiments [114] show that even trained artists fail to correctly draw foreshortened shapes and frequently exaggerate perspective scaling effects. As indicated by prior work on interpreting design sketches [138], viewers are adept at mentally correcting such errors by biasing the envisioned solutions toward more simple and less *foreshortened* interpretations. In the context of gesture drawings, we observe that while viewers use landmarks to anchor the envisioned pose, they mentally tweak the locations of these landmarks in favor of such simpler pose interpretations.

5.4 Framework Overview

The input to our method is a rigged and skinned character model, in a bind pose, and a roughly same scale vectorized gesture drawing. As artists typically create the gesture drawings using the character as a reference, scale similarity is easy for them to satisfy; alternately, manually scaling drawings generated independently from the character model takes seconds for both experts and amateurs. As is typical of skeletal posing systems, the pose of a rigged character is fully determined by the positions of its joints. We

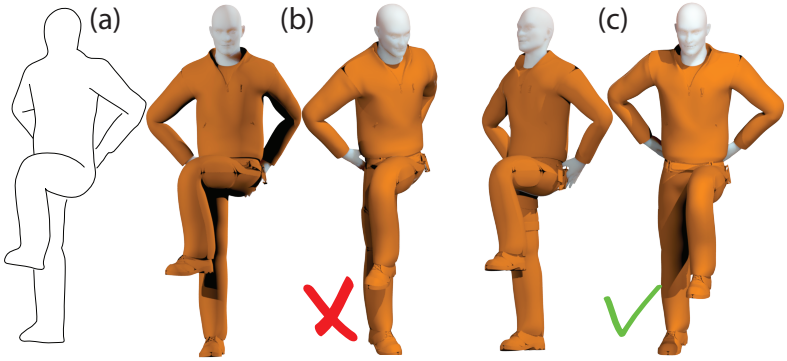


Figure 5.8: Less natural (b) and more natural (c) interpretations of a drawn pose (a) (leg bent sideways vs forward).

compute the joint positions that best reflect the depicted pose using the following steps (Figure 5.9).

Joint-Contour Matching We first match drawn contours against the body parts they describe, and place a projected character skeleton in the image plane so that its surrounding body contours roughly align with their matched drawn ones. We formulate the matching as a computation of optimal joint locations along the contours. As the continuous solution space of all possible joint locations is too large to operate on efficiently, we discretize the problem by considering only a finite set of *potential joint locations* on the 2D drawing. We associate each possible joint location with an unary assignment probability derived from our anatomical landmark prior (Section 5.5.1), and associate binary and ternary probabilities for assignments of adjacent pairs and triplets of joints based on consistency, simplicity, and low foreshortening priors (Section 5.5.3). The resulting discrete optimization problem can be cast as a High-Order Tree-Structured Markov Random Field (MRF) problem [70]. We then minimize this combined cost function subject to additional global constraints imposed by the drawing (Section 5.5.4). Adding these constraints makes the general assignment problem NP-hard; however, as we demonstrate, our greedy solution framework works well in practice (Section 5.5.5).

2D Pose Optimization Our discrete solution considers only a finite set of possible joint locations; accordingly while it provides a good estimate of the joint locations and joint contour correspondences, the final joint place-

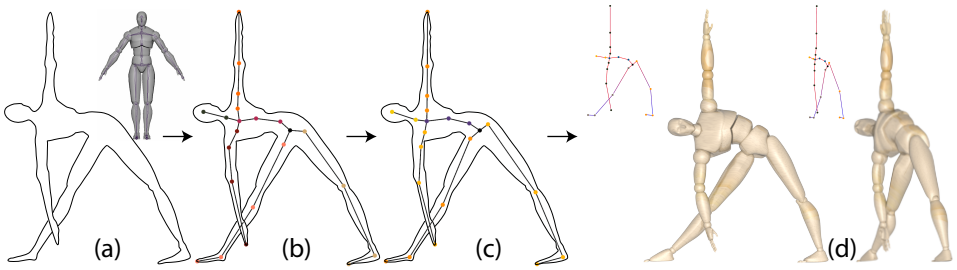


Figure 5.9: Overview: (a) algorithm input; (b) discrete 2D joint embedding; (c) optimized 2D embedding; (d) 3D skeleton (color visualizes depth) and posed model.

ment may be locally sub-optimal. We consequently use continuous location optimization to further improve this solution and compute joint locations that best capture the artist intent (Figure 5.9b-c).

Full Pose Estimation. We proceed to fully pose the character by assigning 3D positions to its joints, further adjusting the joint 2D positions when necessary. We note that exact 2D joint locations are more sensitive to artist errors than bone directions and lengths, and consequently rely on the latter when recovering the full pose. We seek poses that satisfy the ordering cues provided by occluding contours in the gesture drawing, and which balance preservation of the bone directions and 2D lengths, estimated from the drawing, against our expectations of simplicity and foreshortening minimization.

We formulate joint positioning as a constrained energy minimization problem, then obtain the minimum by recasting the energy in term of twist variables [18] and using a Newton-type solution method that follows the approach of [43].

5.5 Character-Contour Correspondence

Initialization To evaluate anatomical landmark correspondences, we need to associate a likely contour arc radius for each character joint. To compute the radius we use a variation on the method of Thierry et al. [126] to fit a sphere to the region on the character mesh surrounding the joint. While many joints are well approximated by spheres, some parts of a character, such as the palm of the hand, are more elliptical and consequently have a

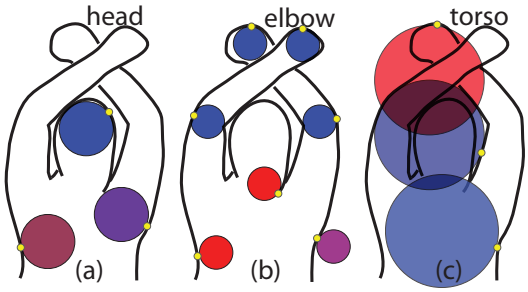


Figure 5.10: Joint cost visualization. Here the color shows the matching cost on a scale from red (poor match) to blue (good).

range of plausible contour arc radii. Given the extracted mesh region around each joint we therefore use PCA to obtain the maximum and minimum radii, and use a discrete set of joint radii with a step of ε within this range in subsequent computations. We set ε to 2% of the drawing bounding box and use it as the default discretization density throughout the discrete solution.

To facilitate the computation and evaluation of contour consistency in the presence of occlusions, we preprocess the contours to detect Gestalt continuous strokes. We use the continuity test described in [10]: given each pair of strokes, we connect their end-points with a straight line and measure the angles between this line and the stroke tangents. A pair of strokes is classified as Gestalt Continuous if both angles are below the 18° threshold identified in perception literature [55]. For each pair of drawing strokes we test all four end-point configurations. When strokes are deemed continuous we retain the connecting line as a *Gestalt bridge* between them. We consider each pair of strokes connected by a bridge as a single *bridged* contour.

5.5.1 Solution Space

As previously noted, artists approximate the contours surrounding joints as circular arcs centered at the joints whose radius reflects the distance from the character joint to the surrounding surface. We therefore expect joints with visible contours to be located approximately a radius distance away from these contours along the contour normal (Figure 5.10). We use this observation to generate potential locations for joints with visible contours. We uniformly sample the input drawing contours at ε-intervals, and treat the samples as potential tangential contact points for joint circle placement. For each sample point we consider the options of placing the circle on either side of its contour, conceptually duplicating all samples into left and right

instances. We compute *potential* joint *locations* by placing each joint along the normal to the contour at the sample at an offset equivalent to it's circle radius (Figure 5.10).

Character joints may be entirely occluded (e.g the man's palms in Figure 5.8). To be able to plausibly place such joints, we sample the bounding box of the drawing using a regular grid with density equal to ε and add these samples to the discrete solution space.

5.5.2 Unary Assignment Cost

We compute, for each joint, the likelihood that it is placed at each potential location. The grid-based locations are assigned the maximal assignment cost of 1 since, absent information to the contrary, we expect contours associated with joints to be visible. For tangential locations, we aim to match appropriate joints to corresponding anatomical landmarks, and hence prioritize placements where sections of the contours are well aligned (in terms of both location and normal) with the joint's circle. Since non-terminal joints are often adjacent to multiple contour segments on different sides of the circle (Figure 5.10), our evaluation looks at all contour samples close to the circle and not just those immediately next to the *originating* tangent sample. Since humans rarely draw perfect circular arcs, we do not expect perfect alignment; to evaluate fit between a joint i and a potential location P_a^i we therefore measure the portion of a circle with radius r_i centered around P_a^i that approximately aligns with the contours using simple distance and normal thresholds. Specifically, we uniformly sample the circle and count the percentage of circle sample points s_c that have nearby contour samples s with contour normals n_s close to the circle sample point normals:

$$T(P_a^i) = \{s_c : |\|s_c - s\| < \min(\varepsilon, \frac{r_i}{2}) \text{ and } \angle(s_c - P_a^i, n_s) < \alpha\}$$
$$C(i, P_a^i) = 1 - \|T(P_a^i)\|/N \qquad (5.1)$$

Here N is the number of samples on the circle. The angle threshold α is set empirically to 15°. When a contour matching a terminal joint is visible in the drawing, we expect a non-negligible portion of the contour to closely align with the joint's circle. We found this threshold based solution to work better than using a falloff function that depends on how close the contours are to the circle. We consider terminal joint locations to be reliable if at least 15% of their osculating cycle is matched by the contours, and assign the maximal cost of 1 to locations that do not pass this threshold. For each

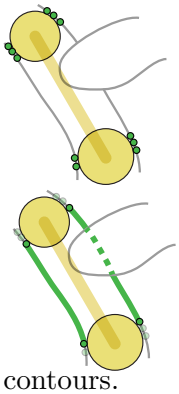
joint i and a potential assigned location P_a^i , in addition to the cost we store the originating contour sample s_a^i and the set of all contour samples S_a^i that satisfy the alignment threshold.

Position Consolidation. Near high-curvature regions on the contours, we typically encounter several potential low cost joint locations for a given joint which have nearly identical sets of well-aligned contour points. To reduce the solution space during computation we consolidate these potential joint locations into one, selecting the location whose originating sample lies closest to the stroke's curvature extremum.

5.5.3 Assignment Compatibility

Our compatibility term is designed to promote contour consistency, and to weakly encourage less foreshortened and more natural solutions.

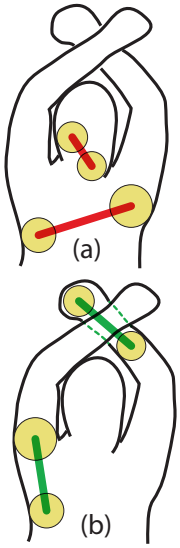
Bone Contours. Each pair of position assignments for the end-joints of a bone indirectly defines the contour segments corresponding to this bone (Figure 5.4). Given a pair of such assignments, we compute the potential bone contour segments defined by these assignments as follows (see inset).



We consider all pairs of well aligned samples, where each sample is associated with a different end-joint. If the two samples lie on the same contour, or on contours connected via bridges, we associate the contour segment or segment chain between them with the bone. We trim the segments by selecting the two samples, one in each joint's set, that are closest to one another along this shared contour as segment end points. We use the computed bone segments to assess the compatibility of the bone's end-joint assignments. Note that occlusions or poor assignments may lead to bones with no corresponding contours.

Consistency. We explicitly prohibit inconsistent assignments where a bone's end-joints lie on opposite sides of the bone's contour, violating our orientation prior. Since a bone is expected to be inside the body part it anchors, it typically should not cross its associated contours. We use a consistency penalty cost C_c, which is set to 1 if a bone's 2D projection intersects any of its associated contour segments, and is 0 otherwise. We use a penalty instead of a hard constraint to account for drawing inaccuracies and sampling artifacts.

We prefer assignments where bones are associated with at least one, either simple or bridged, contour segment. Moreover, we aim for adjacent bones to be associated with the same continuous contour. We encode both preferences by focusing on the contour associated with the originating samples of the end-joint assigned locations: we leave the consistency cost C_c unchanged if a pair of end-joints of a bone are assigned locations with the same originating single or bridged contour, and set it to 1 otherwise.



Bone Contour Conformity We expect the contour segments associated with bones to have relatively low-curvature (see inset). To evaluate contour conformity, we measure the ratio between the length of each bone segment and the Euclidean distance between its endpoints:

$$C_{cf}(i,j) = 1 - e^{-(1-L_c/L)/2\sigma^2},$$

where L_c is the length of the contour segment and L is the Euclidean distance between its end-points. We empirically set $\sigma = 4\%$ of the bounding box diagonal. If a bone has multiple associated contour segments, we repeat the cost computation and, to be conservative, use the lower of the two costs as the conformity cost. If the joints have no shared bone contours, we set the cost to 1.

Pose Preferences. We assign a per-bone cost term for each assignment of its end-joints to a pair of potential positions, based on the difference between the bone length and the image-space distance between the two positions. We expect the artist to select views where the drawn body parts, and consequently bone projections, undergo relatively small foreshortening; we therefore weakly penalize foreshortening when it occurs. While real character bones do not stretch, artist drawings can contain errors in character proportion description. We therefore tolerate assignments where the image-space distance is larger than the respective bone length, but penalize such assignments with a large penalty cost. The combined cost is:

$$C_l(i,j) = \begin{cases} 1 - e^{-(l'_{ij}-l_{ij})^2/2\sigma^2}, & \text{if } l'_{ij} > l_{ij} \\ 1 - e^{-(l'_{ij}-l_{ij})^2/2(\sigma/3)^2}, & \text{otherwise .} \end{cases} \tag{5.2}$$

where $l'_{ji} = \|P_a^i - P_a^j\|$ and l_{ij} is the bone length. We use the same σ as for bone conformity. We evaluate the difference between the two lengths

rather than their ratio, since ratio-based costs are extremely sensitive to artist errors on short bones.

We encode our expectation for simpler, more natural character poses, depicted from a descriptive view, as a preference for 2D joint angles in the output pose that are close to their bind pose counterparts:

$$C_n(i,j,k) = 1 - e^{-(\gamma-\gamma')^2/2\sigma_a^2}$$

Here γ and γ' are the current and bind pose angles respectively between pairs of emanating bones (i,j) and (j,k) at a joint j. We set σ_a to $\pi/3$ if the 3 involved joints share an originating contour, and $\pi/6$ otherwise, enforcing a stronger preference for the bind pose angle when there is no clear contour suggesting 2D bone directions, and a weaker preference for bind pose angles when the adjacent bones follow one continuous contour and the 2D bone direction is well-suggested. These costs are measured for each triplet of adjacent joints. This term can be replaced by more advanced anatomical machinery used in prior work for predicting plausible angles: for instance, if multiple reference poses are provided, one can look at the smallest angle difference across these poses.

Combined Local Cost Function. Combining the different terms above, the cost for assigning a pair of bone end-points i and j to a pair of locations is measured as

$$E(i,j) = 1 - (1 - C_l(i,j))(1 - C_{cf}(i,j))(1 - W_c C_c(i,j)). \tag{5.3}$$

We empirically set the consistency penalty weight to $W_c = 0.9$. The combined energy function encoding all local preferences for a given assignment of joints to point locations is

$$E_{match} = \sum_i C(i, P_a^i) + \sum_{i,j} E(i,j) + \sum_{ijk} C_n(i,j,k) \tag{5.4}$$

where the first term sums the per-joint assignment costs, the second sums the per-bone costs and the third considers the joint triplet costs. All terms have equal weight.

5.5.4 Global Consistency

In addition to the local criteria above, when evaluating the plausibility of a skeleton embedding we need to evaluate the likelihood of the overall contour-to-skeleton assignments it imposes (Figure 5.11). In addition to the bone-segment correspondence computed earlier, this task requires a joint-contour

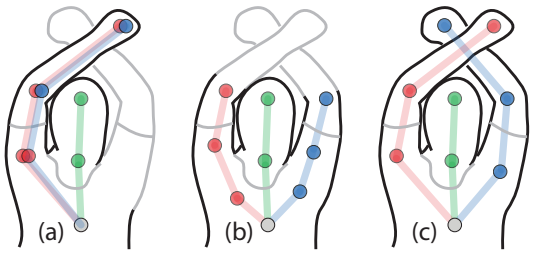


Figure 5.11: Full solutions: (a) contains overlaps; (b) poor coverage; (c) preferred.

correspondence. We compute segments associated with joints as follows. For terminal joints we consider the longest segment delineated by its aligned samples which does not overlap the segments associated with its bone. For interior joints we consider each pair of bones emanating from the joint. If the bones are associated with segments on the same contour, we associate the contour segment in-between them with the joint (Figure 5.4).

In real life, projected visible contours of different character body parts can *overlap* only if the two parts are in contact (i.e. on opposite sides of the contour), or if one is both perfectly parallel to and occluding the other (Figure 5.14, top row). We therefore test whether any pair of same-side contour segments associated with disjoint bones or joints overlap and, if they do, this configuration is assigned a high penalty score, empirically set to 10 (Figure 5.11a).

In a drawing that contains only contours of body parts surrounding skeletal bones, a valid solution must associate all contours with some bone or joint. In practice our drawings can and do occasionally contain extra curves, e.g the cat and horse ears in Figure 5.14. Thus instead of full coverage, we seek for a sufficient one, requiring coverage of over 85% percent of the contours (Figure 5.11b-c). We note that when the soft non-overlap constraint is satisfied, our local energy terms implicitly encourage coverage maximization, since we penalize joints not being matched to contours and discourage undesirable foreshortening. We incorporate coverage constraints into our framework as discussed in Section 5.5.5.

Our local energy does not clearly distinguish between fully or partially symmetric solutions. While hard to penalize locally, partial symmetries (e.g. left arm and right leg mapped to the same side of the spine) are easily detected on a complete solution by evaluating the degree of *twist* the spine must undergo to accommodate them. While twist can be intentional, we expect it to be clearly indicated by the contours, with the undesired

"untwisted" solution in these cases violating consistency constraints. We differentiate between fully symmetric solutions by observing that, all things being equal, artists strongly prefer views where the face of the character is clearly visible. We similarly use this *frontal* preference in our global pose evaluation.

5.5.5 Solver Mechanism

Optimizing E_{match} alone without addressing global preferences can be cast as a classical tree-structured high-order Markov Random Field (MRF) problem by translating our cost terms into probabilities, and optimized efficiently using standard techniques [70]. Unfortunately, we are not aware of any standard mechanism that allows us to incorporate the coverage constraints into such frameworks; the general problem of maximal *a posteriori* estimation over a Markov Random field is a classical NP-hard problem [117]. Instead we develop a simple domain-specific method that works well on all our inputs. We note that, on typical gesture drawings, for terminal joints our unary cost computation produces only about a dozen possible assignments with less than maximal cost; furthermore, our desired assignment is expected to match most terminals, with the exception of occluded ones, to one of these below maximum cost placements. Because of our stringent contour consistency constraints, given the correct assignment of terminals, using the basic E_{match} optimization for assigning other joints results in the desired global solution. Clearly we do not *a priori* know what this correct terminal assignment is; however, given the small number of terminals (typically six or less) and the small number of placement choices for them, an exhaustive search of all possible alternatives is a practical option.

This search can be further sped up by traversing the different alternatives in a strategic order. Specifically, we order all possible terminal assignments based on the sum of their unary costs, and then process them in increasing cost order, penalizing assignment combinations where terminal assignments violate the non-overlap constraints and placing them at the end of the queue. For each terminal assignment we then optimize E_{match} on a reduced set of joints and with a reduced solution space. Specifically, when a terminal joint has a below maximum cost assignment, we remove this node from the solved-for joint set and update the unary and binary costs of its neighboring vertex to reflect the selected assignment. We let the optimization determine the best assignment for terminal joints associated with the maximal cost, but remove all assignments with below maximum cost from their solution space. If the located solution satisfies all our constraints, and in particular if it

produces over 85% coverage, we stop the iterations.

The same coverage can sometimes be produced by a permutation of the desired terminal placements; however different permutations lead to different minima of matching energy E_{match} which may better satisfy our preference for more front facing and less twisted solutions. We thus process all partially and fully symmetric permutations of the obtained solution, and select the least twisted and most front facing one from among those solutions that satisfy all our constraints.

5.6 2D Pose Optimization

While our discrete solver correctly captures the overall contour-joint correspondences, it operates on a finite set of potential positions and thus may end up generating imperfect joint placements (Figure 5.9b). Moreover, to enable an efficient solutions, our discrete formulation assumes all joints are fully flexible. In real models, many joints have a reduced set of degrees of freedom (DOFs), with pelvic and shoulder joints typically supporting only rigid transformations. To address both issues we iterate over the joints to further optimize their positions and enforce the allowable degrees of freedom. For each joint we use a local random walk to find a new location that improves the overall matching energy (Equation 5.4) while constraining the joint to remain on the same side with respect to all nearby contours, and disallowing moves that violate consistency or introduce overlaps.

For joints with a reduced DOF set, we then recompute the positions of the joint and its immediate neighbors which satisfy the DOF constraints and are maximally close to the current ones, using an ICP variant. Specifically, given the current 2D locations of a joint and its neighbors, we search for a 3D transformation of these joints in the bind pose that satisfies the DOF constraints while maximally aligning the 2D coordinates of each joint and its current location. We repeat the two steps until convergence.

5.7 Full Pose Optimization

Once we have generated a 2D skeletal embedding, we associate a depth value with each joint by leveraging viewer expectations of simplicity and weak foreshortening. In this process we also refine image plane joint positions to correct drawing and 2D estimation inaccuracies. In our computations we assume an orthographic projection since, as noted by [138], estimates of artist intended perspective are highly unreliable. Our solution is based on

three key observations. First, we note that even small inaccuracies in depicting body proportions, due to inexact foreshortening, inaccurate perspective, and other artifacts, accumulate to form large errors in 2D joint placement. Therefore, rather than minimizing absolute 2D solution displacement compared to the 2D embedding, we encode conformity with this embedding in terms of slopes and lengths of projected bones. Second we note that human observers are know to underestimate foreshortening in drawings [114], a fact that often causes artists to exaggerate it [57]. Consequently, foreshortening predictions based directly on drawn body-part lengths may be inaccurate. In our observations, viewers rely on relative rather than absolute foreshortening when predicting a character's pose from a drawing - even when presented with a reference model. Consequently, when predicting the degree of foreshortening per bone, we similarly take into account relative foreshortening as compared to other bones. Our last observation is that while we seek for natural poses, i.e. those closer to the input bind pose, minimizing this difference directly is problematic as many drawn poses are quite far from the input one by design. For this reason, we do not explicitly consider the distance to the bind pose in our optimization. Instead we use the bind pose as an initial guess for the solution and limit the step size in each iteration so that our final pose gradually evolves from the bind pose. In doing so, we indirectly guide our final solution towards a more natural pose by searching for a smooth motion path from the bind pose to the final one.

Conformity We encode conformity to the estimated 2D skeletal pose as preservation of 2D bone slopes and lengths:

$$E_c = \sum_{(i,j)\in S} w_c(i,j)((P_i^y - P_j^y) - d_{ij}^y)^2 + ((P_i^x - P_j^x) - d_{ij}^x)^2 \tag{5.5}$$

where P_k are joint positions, S is the set of all skeletal bones, and d_{ij}^x, d_{ij}^y are the x and y differences between joint positions in the 2D embedding. To focus on relative rather than absolute bone projection preservation we set $w_c(i,j) = 1/l_{ij}^2$ where l_{ij} is the length of the bone (i,j).

Foreshortening When the 2D projected bone lengths l'_{ij} are fixed, the depth along each bone is fully determined by the difference between the 3D and 2D projected bone lengths: $(d_{ij}^z = \sqrt{l_{ij}^2 - l'^2_{ij}})$. However image space lengths are sensitive to artist errors, as well as scale mismatches between

the character model and the drawing. Leveraging our previous observations about human preference for foreshortened interpretations, we consequently combine conformity with a foreshortening minimization term which, together with the regularity constraints below, aims to mitigate drawing inaccuracies:

$$E_v = \sum_{(i,j)\in S} w_v(i,j) \cdot (P_i^z - P_j^z)^2. \tag{5.6}$$

The weights $w_v(i,j)$ are determined by the anticipated foreshortening of the bone (i,j):

$$w_v(i,j) = \begin{cases} e^{-\frac{(f_{ij}-f_{avg})^2}{2\sigma_f^2}}, & \text{if } f_{i,j} < f_{avg} \\ 1.0, & \text{otherwise} \end{cases} \tag{5.7}$$

Here $f_{ij} = l'_{ij}/l_{ij}$ is bone foreshortening and f_{avg} is the average bone foreshortening for the entire character in the 2D solution. This weight is a monotonically decreasing function of the 2D-3D length ratio and is maximized when this ratio is equal to or larger than the average across the drawing. We view a ratio below 0.6 of the average as intentional foreshortening and consequently force the weight of the foreshortening minimization term drop to zero for such ratios by setting $\sigma_f = 0.2$ using the three-sigma rule.

Regularity Previous work on the interpretation of drawings (e.g. [10, 138]) has discussed numerous domain-specific regularity criteria. In our work we found four key regularity cues which viewers expect to hold when envisioning drawn poses: parallelism, symmetry, contact, and smoothness. We use the 2D embedding to detect near-regular relationships and then strictly enforce them in 3D. For each pair of bones (i,j) and (m,n) with roughly parallel 2D projections (within 10°), we enforce their 3D bone directions to be the same: $P_i - P_j = l_{ij}/l_{mn}(P_m - P_n)$. Similarly, if two symmetrical limb bones are nearly symmetric around the spine plane, we force exact symmetry - since symmetry is detected in 3D, we enforce this constraint in a post-process step. We also note that human observers expect close 2D adjacencies, specifically contacts observed in 2D, to be preserved in 3D. We therefore detect pairs of adjacent 2D joint contour segments and constrain the distance between their corresponding 3D joints. Lastly, we note that gesture drawings typically aim to convey aesthetic poses [48]. Motivated by Guay et al., we fit a quadratic polynomial spline to each skeletal limb in the 2D embedding; if all joints along the limb are deemed to be close

enough to this spline, i.e. within half each joint's radius from it, we add soft constraints attracting them toward corresponding spline locations.

Joint Ordering. The drawing contours define two types of occlusions, inter- and intra- part (Figure 5.7). Inter-part occlusions, such as an arm in front of a body, indicate that at a particular point along one bone, the body part surrounding this bone is in front of a particular location on the body part around another bone. We encode these using relative locations on the participating bones:

$$P_i^z t_{ij} + P_j^z (1 - t_{ij}) + R_{ij}(t_{ij}) < P_k^z t_{kl} + P_l^z (1 - t_{kl}) - R_{kl}(t_{kl}) \tag{5.8}$$

Here the two participating bones are (i, j) and (k, l), t_{ij} and t_{kl} are the linear parameters of the occluded and occluder points and R_{ij} and R_{kl} are the corresponding body part radii at these points.

Intra-part occlusions, depicted via local contour T-junctions, encode pairwise joint ordering between end-joints i and j of individual bones. The joint associated with the stem of the "T" is expected to be farther away than the one associated with its top. To enforce these relationships we add the inequality constraint:

$$P_i^z < P_j^z.$$

Solver We minimize $E_c + E_f$ subject to the simplicity and order constraints detailed above. While our posing criteria are for convenience expressed via positions, using positions as optimization variables is problematic, since preserving fixed bone lengths using a position based formulation requires quadratic constraints, which are known to be hard to operate on [43] . Instead we follow the standard approach used in kinematics and robotics and represent our 3D pose in terms of *twist coordinates* θ_{ij} [18]. We then use a solution method advocated by Gall et al. [43], who represent vertex positions via twists, and use a Taylor expansion to linearize the resulting expressions. Using such linearizion we formulate the optimization of E as a sequence of constrained quadratic optimizations. We augment the quadratic function being minimized at each iteration with a stabilization term aimed at keeping the new solution close to the previous one:

$$\alpha \sum_{ij} (\theta_{ij}^n - \theta_{ij}^{n-1})^2 \tag{5.9}$$

Here the sum is evaluated over all twist variables θ_{ij} in the current n and previous $n - 1$ iterations. We use a large $\alpha = 200$ to avoid introducing

unnecessary and unnatural deviations from the bind pose. Note that since the stabilizer is computed with respect to the previous solution, this process allows for slow, but arbitrarily far, deviation from this pose. The resulting quadratic optimization with ordering constraints is solved at each iteration using the Gurobi optimizer (www.gurobi.com). Since we have just a few dozen variables the entire process takes on average 30 seconds.

5.8 Validation

We validate the key aspects of our method in a number of ways.

Ground Truth and Perception Comparison. We validate our method on Ground Truth (GT) data, by posing two models into complex poses (Figure 5.12) and using retraced projected occlusion contours as inputs to our method together with the same models in neutral bind pose. Our results closely resemble the original.

Our method aims to recover the viewer-perceived pose from the drawings; therefore a more interesting test is to compare our poses to viewer perceived ones. We performed this test using the same data, by providing our inputs to two 3D modeling experts and asking them to pose the models into poses depicted by the drawings. The result (Figure 5.12) are visually even more similar to ours than ground truth. We showed each artists the ground truth models, our results and the result produced by the other artist, without identifying which output was produced by which method, and asked "How well do these poses capture the artist intended pose?". Both assessed all the shown 3D poses as reflective of the drawn one, and one commented that our result was "the most natural". The full text and the results of the evaluation can be found at `http://cs.ubc.ca/~bmpix`. The artists required roughly 15 minutes to pose each model, 5 to 10 time more than our automatic posing times of 1.5 and 3 minutes.

Perceived 2D Skeletal Embedding. To evaluate consistency across viewers and to compare our algorithm with viewer perception, we asked 10 viewers (2 artists and 8 with no art background, 6 females and 4 males) to manually embed skeletons to match 4 gesture drawings. We provide viewers with 2D images of the models and skeletons in the bind pose, with joints clearly marked, and with bone chains numbered and colored with different colors to facilitate distinction between symmetric limbs. The full text and the results of the evaluation can be found at `http://cs.ubc.ca/~bmpix`.

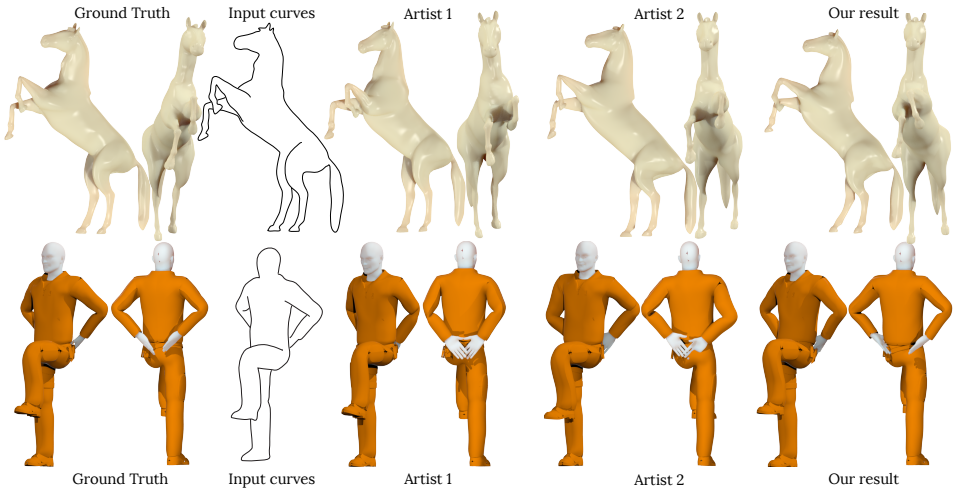


Figure 5.12: Comparing our results to GT data and artist modeled poses. We use as input the projected contours of the posed GT models combined with their bind posed originals (Figure 5.14) to automatically create poses qualitatively similar to both GT and artist results.

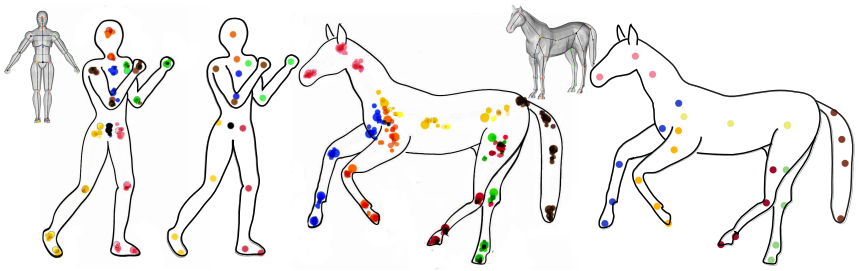


Figure 5.13: Overlays of viewer created skeleton embeddings (lines removed for clarity) and our results on same inputs.

While viewers found the task conceptually easy, marking locations for all joints and connecting them took participants 5 to 10 minutes per drawing. Figure 5.13 summarizes the resulting embeddings on two complex inputs, with various user embeddings overlaid to visualize correlations across viewers. Viewer embeddings are largely consistent and agree very well with our algorithmic results, confirming that our method is built on solid perceptual foundations.

Qualitative Evaluation. We asked 3 artists and 6 non-experts to comment on our results. We showed them each pair of input and result separately and asked "How well does this 3D character pose capture the artist intended drawn pose?". The full text of the evaluation can be found at `http://cs.ubc.ca/~bmpix`. All respondents agreed that our results successfully capture the drawn poses. Minor differences noted by two participants included: variation in geometric details beyond the control of a skeletal rig, such as extended vs contracted character belly in the yoga pose, Figure 5.14, top; and insufficient tightness of the cross-armed pose in Figure 5.14,bottom. The latter example is particularly challenging since the artist did not draw the actual character palms.

5.9 Results

Throughout the chapter we have shown numerous examples of gesture posing using our method. These examples range from relatively simple occlusion-free and relatively flat ones, e.g. Figure 5.15, to the karate, cat, and dance poses which exhibit large foreshortening and complex occlusions (Figures 5.1, 5.14, 5.17). Our results extend beyond typical humanoid models attempted by previous 2D posing methods (e.g. [33]), to whimsical characters and animals (Figure 5.14). Across all examples our method believably reproduces the drawn poses. It seamlessly overcomes drawing inaccuracies, clearly visible in inputs such as the gymnastics poses in Figures 5.9, 5.15, 5.16 where the drawn limbs are consistently longer and skinnier in proportion to its torso than those of the character model.

Workflow. Most of our inputs were created using a traditional keyframing workflow, where the artists had a model in front of them and drew the poses with this character in mind (Figures 5.1, 5.14). We also evaluated an inverse workflow inspired by legacy drawings - tracing the strokes on existing gesture drawings and adjusting the character dimensions to roughly fit those (e.g. the karate sequence in Figure 5.17). This workflow can enable

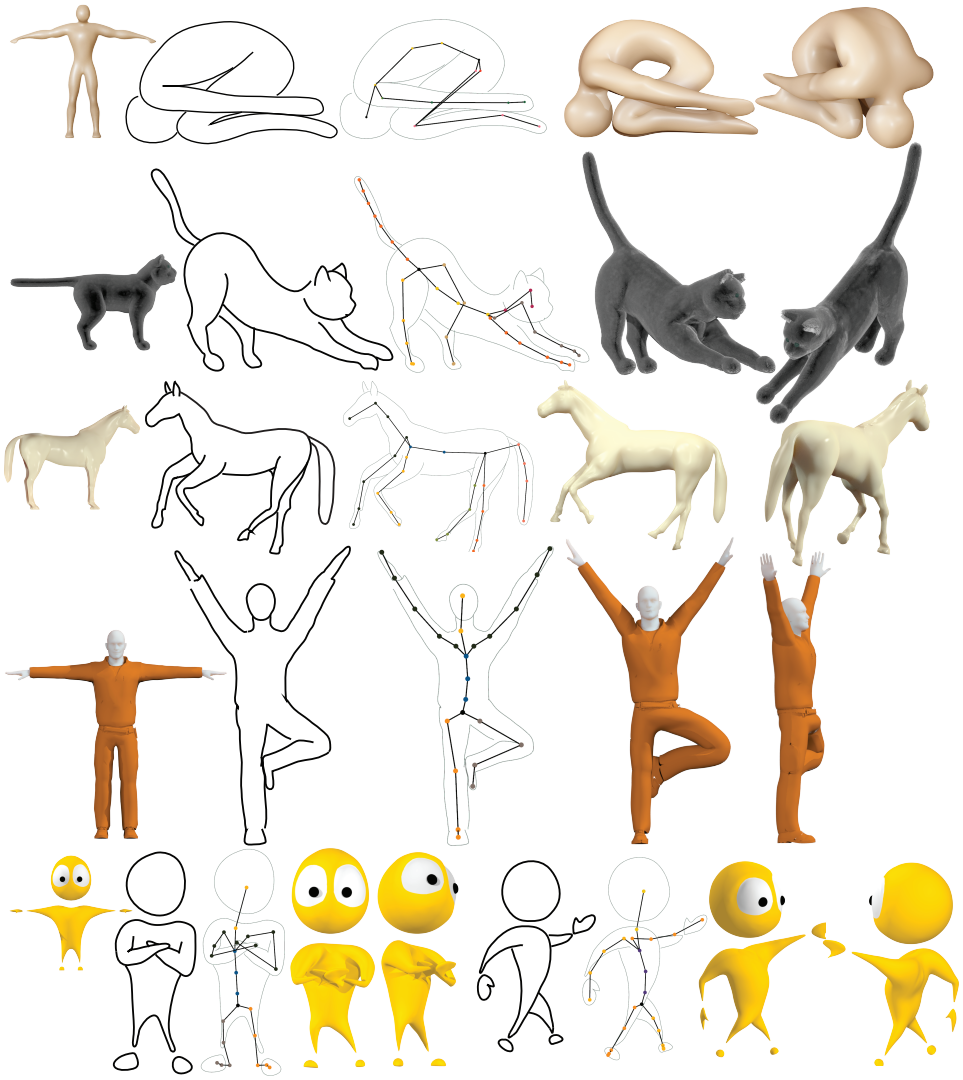


Figure 5.14: Typical two-stage processing results. Left to right: input model, drawing, 2D skeleton fitting, output model.

non-artists to create compelling poses and animations by re-using existing material and assets, but is likely to be more challenging as the character proportions are more likely to differ.

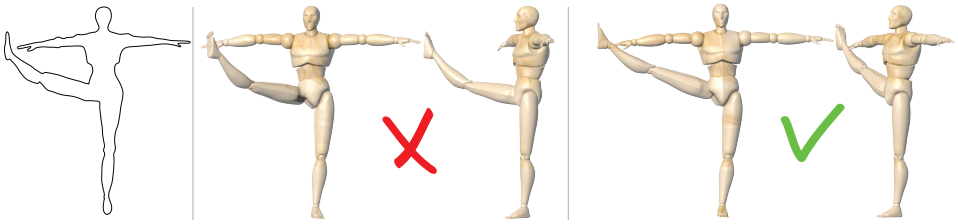


Figure 5.15: (center) 3D posing using only drawing conformity, (right) full 3D solution.

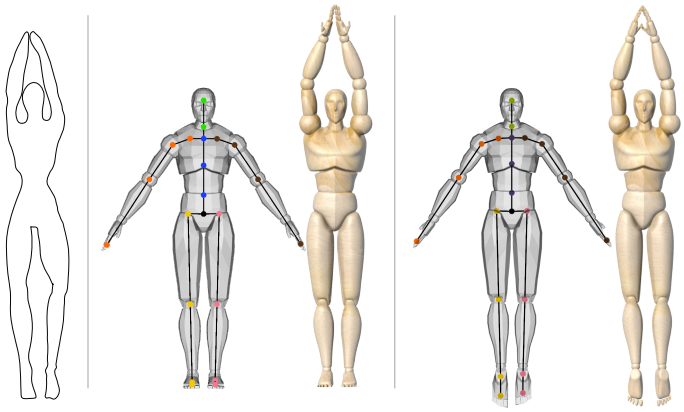


Figure 5.16: Impact of different bind poses.

Impact of Design Choices. Figures 5.5, 5.8, 5.11, and 5.15 demonstrate the importance of our algorithmic choices, highlighting what can happen if we omit one or more of the perceptual cues we employ. Figure 5.15 demonstrates the effect of our foreshortening and regularity terms on 3D pose reconstruction. Absent these terms, the posed character better conforms to the input contours, but the 3D pose becomes less predictable or natural. Figure 5.8 further highlights the distinction between more and less natural interpretations.

Figure 5.16 shows the impact on our results of using different bind poses. As demonstrated the bind pose impacts part orientation for cases where the drawing does not provide clear pose information, e.g. the feet of the character, or when the skeletal resolution is not sufficient to capture orientation details, e.g. the character's palm orientation.

Comparison to Prior Art. Figure 5.17 compares our results against [33], the closest prior work in terms of 2D posing ability. While both methods

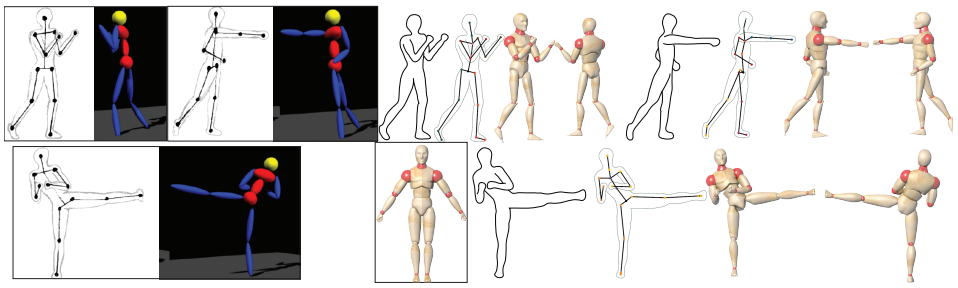


Figure 5.17: (right) Davis et al.[33] trace stick figures over gesture drawings and then pose characters semi-automatically. (left), We use the original drawings to automatically pose characters.

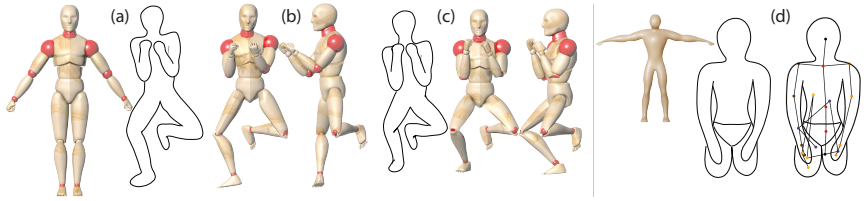


Figure 5.18: Extreme mismatch in proportions between model and drawing (a) can lead to poor depth reconstruction (b); correcting the proportions in the drawing (c) corrects the reconstruction. (d) Ambiguous drawings using highly oblique views can cause our 2D pose estimation to fail.

recover qualitatively similar poses, we compute the pose fully automatically, and use only the drawings and the model in a bind pose as inputs. In contrast Davis et al. use a much more elaborate and time consuming process - users first draw a stick figure on top of the drawing, marking all 2D joint locations, then add extra annotations and select between multiple solutions to resolve input ambiguities. As our evaluation shows, while drawing a stick figure is not difficult it is nevertheless time consuming.

Parameters and Runtimes. All our results were computed with the default parameters listed in the text. For the multi-component model 'wynky' (Figure 5.14, bottom row) we disabled the crossing cost as on this model bones must intersect contours. Our method takes between 1 to 3 minutes to compute the output pose; roughly 60% of this time is spent on the 2D discrete embedding computation.

Limitations. Our method is inherently limited by the descriptiveness of the drawing (Figure 5.18). We rely on a combination of drawing and

model's proportions to predict foreshortening. When the proportions of the drawn and posed characters are drastically different (in Figure 5.18a-b the drawn arms are much shorter and the drawn legs much longer than their model counterparts), our framework will by necessity misestimate the degree of output foreshortening. Once the drawing proportions are adjusted we correctly recover the intended pose (Figure 5.18c). Our pose estimation can fail when a gesture is not evident from the drawing itself, due to e.g. oblique views (Figures 5.2c, 5.18d), but can typically correctly recover the pose given a more descriptive view (Figure 5.1).

5.10 Conclusions

We have presented and validated the first method for character posing using gesture drawings. Our method leverages a set of observations about the key properties of gesture drawings that make them effective at conveying character pose. Using these observations we are able to first recover a 2D projection of the character's pose that matches the drawing, and then imbue it with depth. We are able to reconstruct convincing 3D poses, confirmed to agree with viewer expectations, from a single gesture drawing while robustly correcting for drawing inaccuracy.

Our work raises many directions for future research. It is empirically known that in artist drawings "errors of intent are inherent and unavoidable, and furthermore can be of significant magnitude" [114]. An interesting perceptual question would therefore be to explore when and where artist intent and viewer perception diverge, and at which point human observers are no longer able to correct for artist inaccuracies. The algorithmic impact of this exploration would provide more strict definitions of when and how pose recovery should deviate from conformity constrains. Our framework focuses on drawing cues, and it would also be interesting to explore how we can combine those cues with stronger anatomical priors on plausible character poses and other domain cues.

Chapter 6

Discussion and Conclusion

In this thesis, we have discussed several systems to recover 3D shape from concept and pose drawings, along with their underlying ideas and insights into interpretation of line drawings. We have presented the first system to quadrangulate closed 3D curve networks, capable of creating a surface consistent with artist intent. We have analyzed and formalized the defining principle to construct the artist-intended surfaces by interpreting input curves as flow-lines. We have then introduced and discussed a novel system to construct 3D character canvas from a single complete drawing and a 3D skeleton. We have shown that 3D skeleton is sufficient to resolve ambiguities in drawings without imposing unrealistic simplifying requirements on 3D shape. Finally, we have presented the first system to pose rigged 3D characters via a single descriptive gesture drawing. Thus we show that when the 3D shape is known, it is possible to interpret a gesture drawing with no extra user input.

6.1 Discussion

Here we briefly re-iterate over the contributions of all the proposed methods. We also include a short discussion of each method within the scope of sketch interpretation. In-depth discussion and additional details of each particular method can be found in the corresponding chapters of this thesis.

3D curve networks for CAD objects can be created via modern interfaces [7, 138], can effectively communicate shape [35, 88, 89], and, as we show in Chapter 3, can be automatically surfaced. These results imply that 3D curve networks can become an efficient tool to accelerate CAD modeling, combining the expressivity of pen-and-paper sketches with full power of 3D models. Our contribution is the first method that infers the artist-intended surface automatically by interpreting the input lines as representative flow-lines, iteratively segmenting and pairing them via stable matching, and then using that matching to represent the final surface as a set of gradually changing flow-lines.

Cartoon drawings are known to unambiguously convey shape to human observers, yet are notoriously hard to parse and interpret automatically. Such difficulty stems in part from various ambiguities in the drawing, which, as we show in Chapter 4, can be successfully resolved by specifying the corresponding pose via an overlaid 3D skeleton. Conversely, as we show in Chapter 5, when the 3D model of the rigged character is known, the gesture drawing alone can determine the character's pose. Interestingly, human observers seem to be able to infer both shape and pose from character drawings alone, perhaps relying on stronger anatomical priors. More perception research is needed to characterize such anatomical knowledge and outline the limits of human interpretation of drawings.

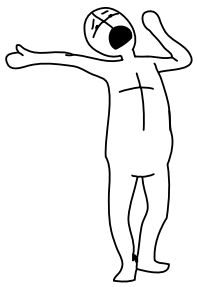
It has to be also noted that concept drawings of a character (Chapter 4) and gesture drawings (Chapter 5) are drawn for different purposes, and thus may exhibit different features. Gesture drawings allow for more simplified shape, distorted proportions, incorrect foreshortening, and are aimed at conveying the pose only; cartoon drawings, however, typically demonstrate more attention to details in order to faithfully capture the character's shape. Nevertheless, as our analysis of the art literature and perceptual studies shows in Chapters 4 and 5, the general concepts we can use to interpret them automatically are similar. Thus, we observe that a valid 3D interpretation of a drawing should conform to the drawn contours (*conformity*), should be simple, regular, and natural (*simplicity*), and should consider the interplay between skeletal and contour adjacencies (*line consistency*).

6.2 Future Work

Since its publication, the method in Chapter 3 has inspired some follow-up work [103] and work on related issues [142]. Those papers solve some of the originally proposed future work, such as automatic loop extraction from the curve network and automatic classification of the input curves into trimming curves and flow-lines. However, both the proposed in the current thesis and the current state of the art [103] methods assume the whole curve network to be complete, and are not directly suitable for handling incremental updates to the curve network. Instead, it would be more interesting to see a sketch-based interactive system that allows to start with a very few curves and iteratively refine the suggested surface.

The method proposed in Chapter 4 relies on all the lines in the input drawing being occlusion contours. More complicated drawings, however, contain multiple feature curves, such as eyes, nose, or auxiliary curves (see

inset below) [26], which, unless annotated, may cause artifacts in the final 3D shape of our method. While determining feature curves based on purely geometrical information seems a hard task, we envision this can be solved using machine learning techniques by casting it as a classification problem.



Furthermore, output canvases in Chapter 4 are unions of manifold meshes, not a single manifold mesh. A naive approach to improve that might be to apply a mesh boolean operation [72], though that may introduce triangles of poor quality near mesh intersections. Instead, a more direct way would be to use the trajectories (Section 4.1) as an input for a surface reconstruction method from cross-sections [143]. A combined system of our method, a machine-learning classifier of the input curves, and a surface reconstruction system may become a powerful modeling framework.

In all our projects we don't require any user annotation or extra information about the input curves. This is very typical for vectorized drawings, but in some sketching systems [7, 99] more information is available. Some of that information, particularly timestamps of each curve, might prove to be a useful cue. The most direct application would be a more robust resolution of Gestalt-continuous contours: while we currently use a simple angular threshold method, one would expect Gestalt-continuous contours to be drawn one exactly after another. In a more subtle way, there might be some correlation which part of the character users draw first, which could aid the full-pose optimization process (Section 5.5.5). These questions call for more in-depth study.

Additionally, as we noted in Section 4.1, when interpreting character drawings, viewers often rely on semantic information of some extra elements, such as facial features. The full-pose optimization process (Section 5.5.5) could benefit from a machine learning element that classifies such features, thus reducing our search space.

6.3 Conclusions

Progress in touch screen manufacturing process has turned much of the commodities, such as cellphones, tablets, or laptops, into convenient sketching and drawing devices. Nevertheless, software, capable of correctly interpreting drawings, still has a long way to go. We hope that the approaches presented in this thesis will contribute to forming a solid foundation of the future drawing interpretation methods.

Bibliography

[1] Fatemeh Abbasinejad, Pushkar Joshi, and Nina Amenta. Surface patches from unorganized space curves. *Comput. Graph. Forum*, 30(5):1379–1387, 2011.

[2] Adobe Flash. Professional. Adobe, Inc., 2013.

[3] J. K. Aggarwal and Q. Cai. Human motion analysis: a review. In *Nonrigid and Articulated Motion Workshop, 1997. Proceedings., IEEE*, pages 90–102, Jun 1997.

[4] Anime Studio. Smith Micro Software, 2013.

[5] Fred Attneave and Robert Frost. The determination of perceived tridimensional orientation by minimum criteria. *Perception & Psychophysics*, 6(6):391–396, 1969.

[6] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. *ACM Trans. Graph.*, 27(3):44:1–44:10, 2008.

[7] S.H. Bae, Ravin Balakrishnan, and Karan Singh. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *Proc. Symposium on User interface software and technology*, pages 151–160. ACM, 2008.

[8] J. Andreas Bærentzen, Rinat Abdrashitov, and Karan Singh. Interactive shape modeling using a skeleton-mesh co-representation. *ACM Trans. Graph.*, 33(4):132:1–132:10, July 2014.

[9] Katie Bassett, Ilya Baran, Johannes Schmid, Markus Gross, and Robert W. Sumner. Authoring and animating painterly characters. *ACM Trans. Graph.*, 32(5):156:1–156:12, October 2013.

[10] Mikhail Bessmeltsev, Will Chang, Nicholas Vining, Alla Sheffer, and Karan Singh. Modeling character canvases from cartoon drawings. *Transactions on Graphics*, 34(5), 2015.

[11] Mikhail Bessmeltsev, Caoyu Wang, Alla Sheffer, and Karan Singh. Design-driven quadrangulation of closed 3d curves. *Transactions on Graphics (Proc. SIGGRAPH ASIA 2012)*, 31(5), 2012.

[12] Preston Blair. *Cartoon Animation*. Walter Foster Publishing, 1994.

[13] Jules Bloomenthal and Brian Wyvill, editors. *Introduction to Implicit Surfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1997.

[14] David Bommes, T. Lempfer, and Leif Kobbelt. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum*, 30(2):375–384, 2011.

[15] David Bommes, Tobias Vossemer, and Leif Kobbelt. Quadrangular parameterization for reverse engineering. In *Proceedings of the 7th international conference on Mathematical Methods for Curves and Surfaces*, MMCS'08, pages 55–69, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] M. Bordegoni and C. Rizzi. *Innovation in Product Design: From CAD to Virtual Prototyping*. Springer, 2011.

[17] Péter Borosán, Ming Jin, Doug DeCarlo, Yotam Gingold, and Andrew Nealen. Rigmesh: Automatic rigging for part-based shape modeling and deformation. *ACM Trans. Graph.*, 31(6):198:1–198:9, 2012.

[18] Christoph Bregler, Jitendra Malik, and Katherine Pullen. Twist based acquisition and tracking of animal and human kinematics. *Int. J. Comput. Vision*, 56(3):179–194, 2004.

[19] Michael Brewer, Lori Freitag Diachin, Patrick Knupp, Thomas Leurent, and Darryl Melander. The mesquite mesh quality improvement toolkit. In *Proceedings, 12th International Meshing Roundtable*, pages 239–250, 2003.

[20] Philip Buchanan, R. Mukundan, and Michael Doggett. Automatic single-view character model reconstruction. In *Proc. Symp. Sketch-Based Interfaces and Modeling*, pages 5–14, 2013.

[21] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep: Extracting editable objects from a single photo. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)*, 32(6), 2013.

[22] Yu Chen, Tae-Kyun Kim, and Roberto Cipolla. Silhouette-based object phenotype recognition using 3d shape priors. In *IEEE International Conference on Computer Vision, ICCV*, pages 25–32, 2011.

[23] Joseph Jacob Cherlin, Faramarz Samavati, Mario Costa Sousa, and Joaquim a. Jorge. Sketch-based modeling with few strokes. *Proceedings of the 21st spring conference on Computer graphics - SCCG '05*, 1(212):137, 2005.

[24] M. G. Choi, K. Yang, T. Igarashi, J. Mitani, and J. Lee. Retrieval and visualization of human motion data via stick figures. *Computer Graphics Forum*, 31:2057–2065, 2012.

[25] M.B. Clowes. On seeing things. *Artificial Intelligence*, 2(1):79 – 116, 1971.

[26] Forrester Cole, Aleksey Golovinskiy, Alex Limpaecher, Heather Stoddart Barros, Adam Finkelstein, Thomas Funkhouser, and Szymon Rusinkiewicz. Where do people draw lines? *ACM Transactions on Graphics*, 27(3):1, August 2008.

[27] S. Coons. *Surfaces for computer aided design.* Technical Report, MIT., 1964.

[28] Frederic Cordier, Hyewon Seo, Jinho Park, and Jun Yong Noh. Sketching of mirror-symmetric shapes. *IEEE Trans. Visualization and Computer Graphics*, 17(11), 2011.

[29] Nicu D. Cornea, Deborah Silver, Xiaosong Yuan, and Raman Balasubramanian. Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer*, 21(11):945–955, 2005.

[30] Joel Daniels, Claudio T. Silva, and Elaine Cohen. Semi-regular quadrilateral-only remeshing from simplified base domains. In *Proc. Symposium on Geometry Processing*, pages 1427–1435, 2009.

[31] Joel Daniels, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. Quadrilateral mesh simplification. *ACM Transactions on Graphics*, 27(5):1, 2008.

[32] K Das, P Diaz-Gutierrez, and M Gopi. Sketching free-form surfaces using network of curves. *Sketch-based interfaces and modeling SBIM*, 2005.

[33] James Davis, Maneesh Agrawala, Erika Chuang, Zoran Popović, and David Salesin. A Sketching Interface for Articulated Figure Animation. *Proc. Symposium on Computer Animation*, pages 320–328, 2003.

[34] Edilson de Aguiar, Carsten Stoll, Christian Theobalt, Naveed Ahmed, Hans-Peter Seidel, and Sebastian Thrun. Performance capture from sparse multi-view video. *ACM Transactions on Graphics*, 27:1, 2008.

[35] F. de Goes, S. Goldenstein, M. Desbrun, and L. Velho. Exoskeleton: Curve network abstraction for 3d shapes. *Computer and Graphics*, 35(1):112–121, 2011.

[36] Koos Eissen and Roselien Steur. *Sketching: The Basics*. Bis Publishers, 2011.

[37] Even Entem, Loïc Barthe, Marie-Paule Cani, Frederic Cordier, and Michiel Van De Panne. Modeling 3D animals from a side-view sketch. *Computer and Graphics*, (38), 2014.

[38] Gerald Farin. *Curves and surfaces for computer aided geometric design: a practical guide*. Academic Press, 1992.

[39] Gerald Farin and Dianne Hansford. Discrete Coons patches. *Computer Aided Geometric Design*, 16:691–700, 1999.

[40] Mark Finch and Hugues Hoppe. Freeform Vector Graphics with Controlled Thin-Plate Splines. *ACM Trans. on Graphics (SIGGRAPH Asia)*, 30(6), 2011.

[41] Fabian Di Fiore, Philip Schaeken, and Koen Elens. Automatic inbetweening in computer assisted animation by exploiting 2.5 D modelling techniques. *Proc. Conference on Computer Animation*, pages 192–200, 2001.

[42] A. Gahan. *3D Automotive Modeling: An Insider's Guide to 3D Car Modeling and Design for Games and Film*. Elsevier Science, 2010.

[43] Juergen Gall, Bodo Rosenhahn, Thomas Brox, and Hans Peter Seidel. Optimization and filtering for human motion capture : AAA multilayer framework. *International Journal of Computer Vision*, 87(1-2):75–92, 2010.

[44] Juergen Gall, Carsten Stoll, Edilson De Aguiar, Christian Theobalt, Bodo Rosenhahn, and Hans Peter Seidel. Motion capture using joint

skeleton tracking and surface estimation. *IEEE Computer Vision and Pattern Recognition Workshops*, pages 1746–1753, 2009.

[45] Kun Gao and Alyn Rockwood. Multi-sided attribute based modeling. *Mathematics of Surfaces XI*, pages 219–232, 2005.

[46] Yotam Gingold, Takeo Igarashi, and Denis Zorin. Structured annotations for 2D-to-3D modeling. *ACM Trans. Graph.*, 28(5), 2009.

[47] Oliver Glauser, Wan-Chun Ma, Daniele Panozzo, Alec Jacobson, Otmar Hilliges, and Olga Sorkine-Hornung. Rig Animation with a Tangible and Modular Input Device. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH)*, 2016.

[48] Martin Guay, Marie-paule Cani, and Remi Ronfard. The Line of Action : an Intuitive Interface for Expressive Character Posing. *ACM Trans. on Graphics*, (6):8, 2013.

[49] Arthur Leighton Guptill. *Sketching and rendering in pencil*. New York, The Pencil Points Press, Inc., 1922.

[50] Gurobi Optimization. http://www.gurobi.com/, 2013.

[51] Fabian Hahn, Frederik Mutzel, Stelian Coros, Bernhard Thomaszewski, Maurizio Nitti, Markus Gross, and Robert W. Sumner. Sketch abstractions for character posing. In *Proc. Symp. Computer Animation*, pages 185–191, 2015.

[52] R.B. Hale and T. Coyle. *Master Class in Figure Drawing*. Watson-Guptill, 1991.

[53] Michael Hampton. *Figure Drawing: Design and Invention*. Figure-drawing.info, 2009.

[54] Ronie Hecker and Kenneth Perlin. Controlling 3d objects by sketching 2d views. *Proc. SPIE*, 1828:46–48, 1992.

[55] Robert Hess and David Field. Integration of contours: new insights. *Trends in Cognitive Sciences*, 3(12):480–486, December 1999.

[56] Donald D Hoffman. *Visual intelligence: how to create what we see*. Norton, New York, NY, 2000.

[57] Burne Hogarth. *Dynamic Figure Drawing*. Watson-Guptill, 1996.

[58] Alexander Hornung, Ellen Dekkers, and Leif Kobbelt. Character animation from 2D pictures and 3D motion data. *ACM Transactions on Graphics*, 26(1):1–9, 2007.

[59] D. A. Huffman. Impossible Objects as Nonsense Sentences. *Machine Intelligence*, 6:295–323, 1971.

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

[61] Clara Ionescu, Dragos Papava, Vlad Olaru, and Cristian Sminchisescu. Human3.6m: Large scale datasets and predictive methods for 3d human sensing in natural environments. *IEEE Trans. Pattern Analysis & Machine Intelligence*, 36(7):1325–1339, 2014.

[62] Robert W. Irving. An efficient algorithm for the "stable roommates" problem. *J. Algorithms*, 6(4):577–595, 1985.

[63] Alec Jacobson, Ilya Baran, J Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78:1—-78:8, 2011.

[64] Alec Jacobson and Olga Sorkine. Stretchable and Twistable Bones for Skeletal Shape Deformation. *ACM Trans. Graph.*, 30(6):165:1–8, 2011.

[65] Eakta Jain, Yaser Sheikh, Moshe Mahler, and Jessica Hodgins. Three-dimensional proxies for hand-drawn characters. *ACM Transactions on Graphics*, 31(1):1–16, 2012.

[66] Tao Ju, Qian-Yi Zhou, Michiel van de Panne, Daniel Cohen-Or, and Ulrich Neumann. Reusable skinning templates using cage-based deformations. *ACM Trans. Graph.*, 27(5):122:1–122:10, December 2008.

[67] Felix Kälberer, Matthias Nieser, and Konrad Polthier. Quadcover - surface parameterization using branched coverings. *Computer Graphics Forum*, 26(3):375–384, 2007.

[68] Olga A. Karpenko and John F. Hughes. SmoothSketch: 3D free-form shapes from complex sketches. *ACM Transactions on Graphics (TOG)*, 1(212):589–598, 2006.

[69] K. Koffka. *Principles of Gestalt Psychology*. International library of psychology, philosophy, and scientific method. Routledge & K. Paul, 1955.

[70] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[71] Vladislav Kraevoy, Alla Sheffer, and Michiel van de Panne. Modeling from contour drawings. In *Proc. Symposium on Sketch-Based Interfaces and Modeling*, pages 37–44, 2009.

[72] Shankar Krishnan and Dinesh Manocha. An efficient surface intersection algorithm based on lower-dimensional formulation. *ACM Trans. Graph.*, 16(1):74–106, January 1997.

[73] N. Leland. *The New Creative Artist*. F+W Media, 2006.

[74] Zohar Levi and Craig Gotsman. ArtiSketch: A System for Articulated Sketch Modeling. *Computer Graphics Forum*, 32(2):235–244, 2013.

[75] Bruno Levy. Dual domain extrapolation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 22(3):364–369, 2003.

[76] Bruno Levy and Yang Liu. Lp centroidal voronoi tesselation and its applications. *ACM Trans. Graph.*, 2010.

[77] Juncong Lin, Takeo Igarashi, Jun Mitani, and Greg Saul. A sketching interface for sitting-pose design. In *Proc. Sketch-Based Interfaces and Modeling Symposium*, pages 111–118, 2010.

[78] H Lipson and M Shpitalni. Optimization-based reconstruction of a 3d object from a single freehand line drawing. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[79] Alan K. Mackworth. On reading sketch maps. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'77, pages 598–606, San Francisco, CA, USA, 1977. Morgan Kaufmann Publishers Inc.

[80] Jitendra Malik. Interpreting line drawings of curved objects. *International Journal of Computer Vision*, 1(1):73–103, 1987.

[81] P. Malraison. *N-SIDED Surfaces: a Survey*. Defense Technical Information Center, 2000.

[82] C Mao, S F Qin, and D K Wright. A sketch-based gesture interface for rough 3D stick figure animation. *Proc. Sketch Based Interfaces and Modeling*, 2005.

[83] C. Maraffi. *Maya Character Creation: Modeling and Animation Controls*. Voices that matter. Pearson Education, 2003.

[84] Martin Marinov and Leif Kobbelt. A Robust Two-Step Procedure for Quad-Dominant Remeshing. *Computer Graphics Forum*, 25(3):537–546, September 2006.

[85] George Markowsky and Michael A. Wesley. Fleshing out wire frames. *IBM J. Res. Dev.*, 24(5):582–597, September 1980.

[86] D. Marr. Analysis of occluding contour. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 197(1129):441–475, 1977.

[87] James McCrae and Karan Singh. Sketching piecewise clothoid curves. *Comput. Graph.*, 33:452–461, August 2009.

[88] James McCrae, Karan Singh, and Niloy J. Mitra. Slices: a shape-proxy based on planar sections. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 168:1–168:12, 2011.

[89] Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. Abstraction of man-made shapes. *TOG (Proc. SIGGRAPH Asia)*, 28(5):1–10, 2009.

[90] Paul Michalik, Dae Hyun Kim, and Beat D. Bruderlin. Sketch- and constraint-based design of b-spline surfaces. In *Proceedings of the Seventh ACM Symposium on Solid Modeling and Applications*, SMA '02, pages 297–304, New York, NY, USA, 2002. ACM.

[91] Scott A. Mitchell. High fidelity interval assignment. In *Proceedings, 6th International Meshing Roundtable*, pages 33–44, 1997.

[92] Thomas B. Moeslund, Adrian Hilton, and Volker Krger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104(23):90 – 126, 2006.

[93] K. Nakayama and S. Shimojo. Experiencing and Perceiving Visual Surfaces. *Science*, 257:1357–1363, 1992.

[94] A. Nasri, M. Sabin, and Z. Yasseen. Filling N -Sided Regions by Quad Meshes for Subdivision Surfaces. *Computer Graphics Forum*, 28(6):1644–1658, September 2009.

[95] Andrew Nealen, Takeo Igarashi, Olga Sorkine, and Marc Alexa. Fibermesh: designing freeform surfaces with 3d curves. *ACM Trans. Graph.*, 26, July 2007.

[96] Andrew Nealen, Olga Sorkine, Marc Alexa, and Daniel Cohen-Or. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph.*, 24(3):1142–1147, 2005.

[97] Kimon Nicolades. *The Natural Way to Draw*. Houghton Mifflin, 1975.

[98] L. Olsen, F.F. Samavati, M.C. Sousa, and J. Jorge. Sketch-based modeling: A survey. *Computers & Graphics*, 33, 2009.

[99] Luke Olsen, Faramarz Samavati, and Joaquim A. Jorge. Naturasketch: Modeling from images and natural sketches. *IEEE Computer Graphics and Applications*, 31(6):24–34, 2011.

[100] Günay Orbay and Levent Burak Kara. Sketch-based modeling of smooth surfaces using adaptive curve networks. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 71–78, 2011.

[101] S.J Owen. A survey of unstructured mesh generation technology. In *Proc. International Meshing Roundtable*, 1998.

[102] Stephen E. Palmer. The effects of contextual scenes on the identification of objects. *Memory and Cognition*, 3(5):519–526, 1975.

[103] Hao Pan, Yang Liu, Alla Sheffer, Nicholas Vining, Chang-Jian Li, and Wenping Wang. Flow aligned surfacing of curve networks. *ACM Trans. Graph.*, 34(4):127:1–127:10, July 2015.

[104] Paperman. Walt Disney Animation Studios, 2012.

[105] Zygmunt Pizlo and AdamK. Stevenson. Shape constancy from novel views. *Perception & Psychophysics*, 61(7):1299–1307, 1999.

[106] Alec Rivers, Fredo Durand, and Takeo Igarashi. 2.5D cartoon models. *ACM Transactions on Graphics*, 2010.

[107] K.L.P. Rose. Modeling developable surfaces from arbitrary boundary curves. *Processing*, (August), 2007.

[108] E. Ruiz-Gironés and J. Sarrate. Generation of structured meshes in multiply connected surfaces using submapping. *Adv. Eng. Softw.*, 41:379–387, February 2010.

[109] David Salomon. *Curves and Surfaces for Computer Graphics*. Springer-Verlag, 2006.

[110] Peter Sand, L McMillan, and J Popovic. Continuous capture of skin deformation. *ACM Transactions on Graphics (TOG)*, pages 578–586, 2003.

[111] Benjamin Sapp, Alexander Toshev, and Ben Taskar. Cascaded models for articulated pose estimation. *Lecture Notes in Computer Science*, 6312:406–420, 2010.

[112] S. Schaefer, J. Warren, and D. Zorin. Lofting curve networks using subdivision surfaces. *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing - SGP '04*, page 103, 2004.

[113] Johannes Schmid, Martin Sebastian Senn, Markus Gross, and Robert W. Sumner. Overcoat: an implicit canvas for 3d painting. *ACM Trans. Graph.*, 30:28:1–28:10, August 2011.

[114] Ryan Schmidt, Azam Khan, Gord Kurtenbach, and Karan Singh. On expert performance in 3D curve-drawing tasks. *Proc. Symposium on Sketch-Based Interfaces and Modeling*, 1:133, 2009.

[115] Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. Analytic drawing of 3d scaffolds. *ACM Trans. on Graph. (Proc. SIGGRAPH Asia)*, 28(5), 2009.

[116] Cloud Shao, Adrien Bousseau, Alla Sheffer, and Karan Singh. Crossshade: Shading concept sketches using cross-section curves. *ACM Trans. Graphics*, 31(4), 2012.

[117] Solomon Eyal Shimony. Finding maps for belief networks is np-hard. *Artificial Intelligence*, 68(2):399 – 410, 1994.

[118] Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. Geosemantic snapping for sketch-based modeling. *Computer Graphics Forum (Proc. Eurographics)*, 32(2):245–253, 2013.

[119] Karan Singh, Hans Pedersen, and Venkat Krishnamurthy. Feature based retargeting of parameterized geometry. In *Proceedings of the Geometric Modeling and Processing 2004*, GMP '04, pages 163–, Washington, DC, USA, 2004. IEEE Computer Society.

[120] Kent A. Stevens. The visual interpretation of surface contours. *Artificial Intelligence*, 17, 1981.

[121] Christopher Summerfield and Etienne Koechlin. A neural representation of prior information during perceptual inference. *Neuron*, 59(2):336–47, July 2008.

[122] Daniel Sýkora, Ladislav Kavan, Martin Čadík, Ondřej Jamriška, Alec Jacobson, Brian Whited, Maryann Simmons, and Olga Sorkine-Hornung. Ink-and-ray: Bas-relief meshes for adding global illumination effects to hand-drawn characters. *ACM Transaction on Graphics*, 33(2):16, 2014.

[123] Chiew-lan Tai, Hongxin Zhang, and Jacky Chun-kin Fong. Prototype Modeling from Sketched Silhouettes based on Convolution Surfaces. *Computer Graphics Forum*, 23(1):71–83, 2004.

[124] Jimmy J. M. Tan. A necessary and sufficient condition for the existence of a complete stable matching. *J. Algorithms*, 12(1):154–178, January 1991.

[125] B Tekin, X Sun, Wang, V. Lepetit, and P. Fua. Predicting people's 3d poses from short sequences. In *arXiv preprint arXiv:1504.08200*, 2015.

[126] Jean-Marc Thiery, Emilie Guy, and Tamy Boubekeur. Sphere-meshes: Shape approximation using spherical quadric error metrics. *ACM Transaction on Graphics*, 32(6), 2013.

[127] Y. Tong, P. Alliez, D. Cohen-Steiner, and M. Desbrun. Designing quadrangulations with discrete harmonic forms. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, SGP '06, pages 201–210, Aire-la-Ville, Switzerland, Switzerland, 2006. Eurographics Association.

[128] T. Várady, Alyn Rockwood, and P. Salvi. Transfinite surface interpolation over irregular n-sided domains. *Computer-Aided Design*, (iv), 2011.

[129] David L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical report, Cambridge, MA, USA, 1972.

[130] Yu Wang, Alec Jacobson, Jernej Barbič, and Ladislav Kavan. Linear subspace design for real-time shape deformation. *ACM Trans. Graph.*, 34(4):57:1–57:11, July 2015.

[131] Xiaolin Wei and Jinxiang Chai. Intuitive interactive human-character posing with millions of example poses. *IEEE Comput. Graph. Appl.*, 31(4):78–88, 2011.

[132] Robert W. Weiner, Irving B. Healy, Alice F. Proctor. *Handbook of Psychology, Experimental Psychology (2nd Edition)*. 2012.

[133] M. A. Wesley and G. Markowsky. Fleshing out projections. *IBM Journal of Research and Development*, 25(6):934–954, Nov 1981.

[134] B. Whited, G. Noris, M. Simmons, R. Sumner, M. Gross, and J. Rossignac. Betweenit: An interactive tool for tight inbetweening. *Comput. Graphics Forum (Proc. Eurographics)*, 29(2):605–614, 2010.

[135] Lance Williams. 3d paint. In *Proceedings of the 1990 Symposium on Interactive 3D Graphics*, I3D '90, pages 225–233, New York, NY, USA, 1990. ACM.

[136] Richard Williams. *The Animator's Survival Kit*. Faber and Faber, 2001.

[137] Kwan-Yee K. Wong, Paulo R.S. Mendona, and Roberto Cipolla. Reconstruction of surfaces of revolution from single uncalibrated views. *Image and Vision Computing*, 22(10):829 – 836, 2004.

[138] Baoxuan Xu, William Chang, Alla Sheffer, Adrien Bousseau, James McCrae, and Karan Singh. True2form: 3d curve networks from 2d sketches via selective regularization. *ACM Transactions on Graphics*, 33(4), 2014.

[139] Genzhi Ye, Yebin Liu, Nils Hasler, Xiangyang Ji, Qionghai Dai, and Christian Theobalt. Performance capture of interacting characters with handheld kinects. In *Proc. European Conference on Computer Vision*, pages 828–841, 2012.

[140] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 163–170, New York, NY, USA, 1996. ACM.

[141] Jianmin Zhao and Norman I. Badler. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics*, 13(4):313–336, 1994.

[142] Yixin Zhuang, Ming Zou, Nathan Carr, and Tao Ju. A general and efficient method for finding cycles in 3d curve networks. *ACM Trans. Graph.*, 32(6):180:1–180:10, November 2013.

[143] Ming Zou, Michelle Holloway, Nathan Carr, and Tao Ju. Topology-constrained surface reconstruction from cross-sections. *ACM Trans. Graph.*, 34(4):128:1–128:10, July 2015.