Reconstruction of Machine-Made Shapes from Bitmap Sketches

IVAN PUHACHOV, Université de Montréal, Canada and Huawei Technologies, Canada
CEDRIC MARTENS, Université de Montréal, Canada
PAUL G. KRY, McGill University, Canada and Huawei Technologies, Canada
MIKHAIL BESSMELTSEV, Université de Montréal, Canada

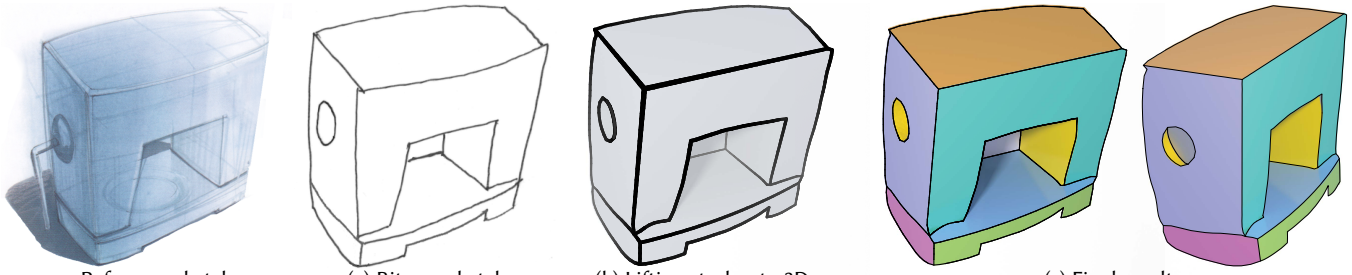


Fig. 1. Given a bitmap sketch (a), we lift the sketch strokes into 3D by fitting 3D geometric primitives (b, stroke thickness indicates depth) and then create a 3D mesh interpolating these strokes and lying on the primitives while preserving the sketch shape (c). Reference sketch ©Eissen and Steur, 2011.

We propose a method of reconstructing 3D machine-made shapes from bitmap sketches by separating an input image into individual patches and jointly optimizing their geometry. We rely on two main observations: (1) human observers interpret sketches of man-made shapes as a collection of simple geometric primitives, and (2) sketch strokes often indicate occlusion contours or sharp ridges between those primitives. Using these main observations we design a system that takes a single bitmap image of a shape, estimates image depth and segmentation into primitives with neural networks, then fits primitives to the predicted depth while determining occlusion contours and aligning intersections with the input drawing via optimization. Unlike previous work, our approach does not require additional input, annotation, or templates, and does not require retraining for a new category of man-made shapes. Our method produces triangular meshes that display sharp geometric features and are suitable for downstream applications, such as editing, rendering, and shading.

CCS Concepts: • **Computing methodologies** → **Shape modeling**.

Additional Key Words and Phrases: industrial design, sketches, line drawing, sketch-based modeling, 3D reconstruction

ACM Reference Format:
Ivan Puhachov, Cedric Martens, Paul G. Kry, and Mikhail Bessmeltsev. 2023. Reconstruction of Machine-Made Shapes from Bitmap Sketches. *ACM Trans. Graph.* 42, 6, Article 1 (December 2023), 16 pages. https://doi.org/10.1145/3618361

Authors' addresses: Ivan Puhachov, Université de Montréal, Canada and Huawei Technologies, Canada, ivan.puhachov@umontreal.ca; Cedric Martens, Université de Montréal, Canada, cedric.martens@umontreal.ca; Paul G. Kry, McGill University, Canada and Huawei Technologies, Canada, kry@cs.mcgill.ca; Mikhail Bessmeltsev, Université de Montréal, Canada, bmpix@iro.umontreal.ca.

0730-0301/2023/12-ART1 $15.00
https://doi.org/10.1145/3618361

1 INTRODUCTION

Sketching, in bitmap software or on paper, is often one of the first steps in designing machine-made shapes, typically followed by modeling the drawn 3D objects in a software. For professionals, the latter modeling step is a tedious routine with little creativity, while for inexperienced hobbyists it might be an impediment to creating the 3D model. A system accepting a natural bitmap sketch and producing a 3D model, thus sidestepping modeling altogether, would alleviate the need for specialized training and simplify prototyping.

Despite recent progress, sketch-based modeling systems for machine-made shapes are still rarely used in production. The reasons for artists' reluctance to use those tools are diverse. For instance, some system require annotated vector input [Gryaditskaya et al. 2020; Xu et al. 2014], even though sketches are often drawn in bitmap format. Other systems require sketches from multiple viewpoints [Delanoy et al. 2018, 2019] or interfere with the design process by combining sketching and modeling into an iterative framework [Li et al. 2020]. Most importantly, however, modern sketch-based modeling systems tend to produce overly smooth shapes that ostensibly ignore some of the drawn curves in the sketch (Fig. 2).

These limitations of the existing systems are not surprising, as designing an algorithm to reconstruct a 3D machine-made shape from a single imprecise bitmap sketch is a daunting task: not only 3D reconstruction from 2D input is ambiguous by itself, but sketches are also a very sparse and distorted depiction of a 3D object. In particular, bitmap sketches often have no shading, or very little, and depict only the front side of an object, impeding inference of 3D information; shapes are often heavily distorted due to the incorrect depiction of perspective [Schmidt et al. 2009], proportions [Gryaditskaya et al. 2019], or simply because of imprecise drawing; finally, bitmap sketches have no explicit connectivity between points, making 2D proximity an unreliable cue for 3D proximity.

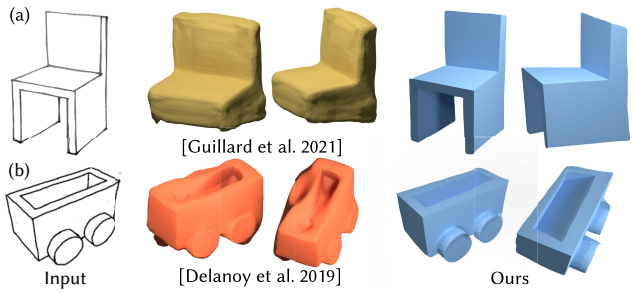


Fig. 2. In most learning-based approaches, the input strokes are often smoothed out or outright ignored (center). In contrast, our results preserve the drawn strokes, introducing sharp ridges and occlusions into the final shape (right). Input images: chair [Eissen and Steur 2011], wagon [Delanoy et al. 2019].

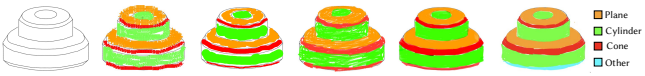


Fig. 3. Given an artist sketch of a machine-made shape, human observers (five participants in our study) can consistently segment the image into primitives. We hypothesize that this segmentation allows humans to mentally reconstruct a 3D shape despite the inaccuracies. Input image [Manda et al. 2021].

We propose a novel system that targets these issues and reconstructs a 3D machine-made shape from a single natural bitmap sketch (Fig. 1). Our main observation is that despite the inaccuracies in a sketch, human observers can consistently segment an input drawing into the typical geometric primitives forming man-made shapes: cylinders, planes, cones, spheres, as well as smooth freeform surfaces (Fig. 3, Sec. 7). We conjecture that it is precisely this segmentation into primitives that allows human observers to alleviate numerous distortions and errors in the input drawings. We replicate that behavior in our algorithmic pipeline, where we segment the input sketch and classify each region, fit geometric primitives to each, and use these primitives as a reliable proxy for 3D shape reconstruction.

Machine-made shapes are piecewise smooth, so their component geometric primitives can either connect smoothly or form sharp ridges (Fig. 1). These sharp ridges are consistently drawn in artist sketches [Cole et al. 2008], which often also contain occlusion contours and other strokes. We therefore observe that the surface between the drawn strokes is smooth, while each stroke can signify either a depth discontinuity depicted by an occlusion contour, or a sharp ridge. Preserving those features is crucial for a quality 3D reconstruction. We leverage this insight directly by using line drawing vectorization as a component in our system, thus preserving the drawn curves. We enforce smoothness of the 3D shape away from the extracted curves, as well as smoothness of those curves in 3D. Furthermore, we algorithmically distinguish sharp ridges from occlusion contours, allowing us to reconstruct sharp manifold 3D shapes.

We combine these observations in a novel system using deep networks to infer the approximate depth and to segment the sketch into separate geometric primitives (Fig. 4). We then leverage these predictions, as well as an initial rough vectorization of the input sketch, in a novel optimization that first lifts the drawing into 3D by fitting the depicted geometric primitives while deciding which stroke is an occlusion contour and aligning intersections of those primitives with the drawn strokes. After using the primitives to find the surfaces interpolating the 3D strokes, this produces a manifold piecewise smooth reconstruction of the visible surface of the artist-intended shape, with contours close to the sketch strokes. Our system inputs natural raster images and does not require manual annotations to reconstruct piecewise smooth 3D shapes.

In summary, our main contributions are:

- a novel system for 3D man-made shape reconstruction from a single natural bitmap sketch, combining deep learning, vectorization, and numerical optimization, and
- a user study demonstrating human observers' consistency in segmenting 2D bitmap sketches into geometric primitives.

We validate our system on a gallery of natural sketches, comparisons with prior work, and user studies.

2 RELATED WORK

We divide the related work into three main categories: sketch-based modeling of machine-made shapes, 3D reconstruction from RGB images, and the creation of CAD shapes from point clouds.

Sketch-based modeling of machine-made shapes. One of the typical dominant characteristics of machine-made shapes is that they are piecewise smooth with sharp ridges. These objects require a largely different modeling toolkit from natural smooth objects. A complete survey of sketch-based modeling of machine-made shapes is outside our scope; please see [Bonnici et al. 2019; Li et al. 2022b; Yue et al. 2020; Zhong et al. 2022].

Our work is unlike methods that process input strokes in an incremental fashion [Chen et al. 2013; Cherlin et al. 2005; Gingold et al. 2009; Igarashi et al. 1999], which is arguably an easier problem given the extra information available in contrast to a bitmap image. Nevertheless, a recent exciting line of work focuses on transforming a sequence of input strokes into CAD commands [Li et al. 2020, 2022a]. Instead, we target a complete single drawing as an input, thus preserving the existing artistic workflows and enabling 3D reconstruction from legacy sketches.

A related line of work processes a vector drawing as an input, either lifting vector curves into 3D [Gryaditskaya et al. 2020; Xu et al. 2014] or reconstructing the complete 3D shape [Deng et al. 2022]. However, artists typically prefer to draw sketches on paper or in bitmap editors; automatically vectorizing bitmap sketches with the precision required by these methods is an open problem [Puhachov et al. 2021]. Our method uses a bitmap sketch as its only input. We use automatic vectorization as a part of our pipeline, treating it only as a conservative estimate of which points in the sketch are connected.

We are inspired by several systems [Li et al. 2018; Lun et al. 2018] that predict depth and normals, and then convert them into a surface. For instance, Lun et al. [2018] fuse those two outputs into a point cloud, then use standard surface reconstruction techniques for the final surface. Li et al. [2018] additionally predict confidence

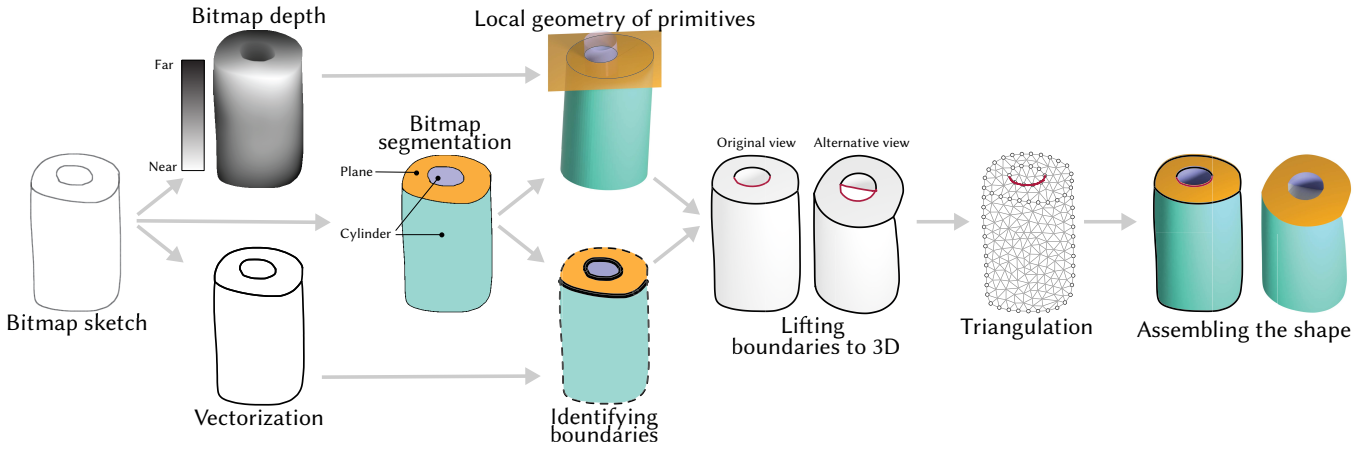


Fig. 4. Starting with a natural bitmap sketch of a machine-made shape, we predict its depth and segmentation into primitives, and locally fit primitives to each region's depth. We leverage those along with the vectorization of the input sketch to lift the drawn strokes into 3D while refining the primitives and determining occlusion contours via an optimization. We then triangulate the input sketch, cutting the triangulation along the identified occlusion contours, and assemble the final shape by projecting this triangulation onto the primitives while maintaining its piecewise smoothness. Input image [Manda et al. 2021].

and curvature directions from annotated sketches from multiple views, reconstructing the visible side of a natural 3D shape. These systems primarily target natural shapes, assuming smoothness of the reconstructed surface. In contrast, with our focus on man-made shapes we reconstruct piecewise smooth surfaces along with sharp ridges and occlusions.

A few other systems exploit alternative representations of geometry: predicting, for instance, point clouds [Wang et al. 2022a], voxelizations [Delanoy et al. 2018; Han et al. 2020], or collections of Coons patches [Smirnov et al. 2021]. A point cloud or a voxelization can be then converted into a piecewise smooth mesh with sharp ridges [Chen and Zhang 2021]. Delanoy et al. [2019] explore a different approach by reconstructing piecewise smooth normals and leveraging them in the surface reconstruction. Unfortunately, all of these methods have no guarantees that the drawn strokes will have direct control over the reconstructed shape, as sometimes strokes get smoothed out or outright ignored (Fig. 2). In our system, in contrast, the drawn strokes explicitly affect the reconstruction, e.g., by introducing discontinuities or sharp ridges. We compare with Delanoy et al. [2019] in Sec. 7.

Finally, some works only shade sketches without fully reconstructing the 3D geometry, either directly via fully convolutional architectures [He et al. 2021; Su et al. 2018; Zheng et al. 2020] or by predicting normals and then inflating the surface [Hudon et al. 2018, 2019]. We compare with Zheng et al. [2020] and Hudon et al. [2018] in Sec. 7.

3D reconstruction from a RGB image. Single-view 3D reconstruction from a photo is a classic problem in computer vision related to our problem. We refer to the survey by Bhattacharjee and Chaudhuri [2020] for a detailed overview, and below we focus on only the most relevant works.

In general, these computer vision methods infer shape information from RGB images: depth maps [Hickson et al. 2019; Tatarchenko et al. 2016; Yao et al. 2020], point clouds [Achlioptas et al. 2018; Fan et al. 2017; Gadelha et al. 2018; Yang et al. 2018], voxelizations [Choy et al. 2016; Girdhar et al. 2016; Liao et al. 2018; Wu et al. 2016, 2018], implicit representations [Chen et al. 2020; Chen and Zhang 2019; Guillard et al. 2021; Mescheder et al. 2019; Remelli et al. 2020], or surface parameterizations [Groueix et al. 2018]. Another output representation is deformation of a fixed template [Wang et al. 2018]. This significantly regularizes the ambiguous task of reconstruction, but limits the output shapes to be topologically equivalent to the template. Pan et al. [2019] address this limitation by predicting per-triangle error and pruning triangles with large error, thus modifying the mesh topology. We compare with Guillard et al. [2021] in Sec. 7.

Despite being successful for computer vision applications, these methods typically fail to reconstruct plausible 3D objects from a sketch. One reason for this is that there does not exist a large-scale dataset of sketches and corresponding ground-truth 3D machine-made shapes, making retraining these systems impossible for our task. Most available datasets of that kind are either small-scale or synthetic, often featuring sketch-like renders instead of natural drawings [Gryaditskaya et al. 2019; Manda et al. 2021; Zhong et al. 2022].

A way around this dataset issue is to use differential rendering, to allow prediction of a 3D object in a non-supervised manner [Kato et al. 2020]. But differential rendering is poorly suited to sketches because sketches, unlike photos, are only approximate depictions of 3D geometry with significant distortions and incorrect perspective, so they cannot be interpreted as precise renderings of 3D geometry [Wang et al. 2021].

CAD shapes from point clouds and voxelizations. Another mature area of computer vision is reconstruction of CAD or machine-made shapes from a potentially noisy or incomplete 3D point clouds. Berger et al. [2017] provide a recent survey.

Starting from the classic methods like RANSAC [Schnabel et al. 2007] or Hough Transform [Duda and Hart 1972], there exist mature methods to find and fit geometric primitives to point clouds. We are inspired by this area of work, and our optimization uses some of the previously proposed objective functions directly [Eberly 2018] (Sec. 4.2, App. C).

Modern methods can regress parameters controlling the geometric primitives in a differentiable manner [Li et al. 2019; Sharma et al. 2020; Yan et al. 2021]. These works, however, have no control over the final surface topology and often reconstruct non-manifold shapes. Other recent methods [Du et al. 2018; Guo et al. 2022; Lambourne et al. 2022; Yu et al. 2022] infer CAD shapes with sharp features from a point cloud, voxelization, or mesh. Unfortunately, these methods are unsuitable for our task as they often require the inputs to be complete and noise-free (Fig.16).

Another strategy is to reduce the search space to make the reconstruction more well-defined: Point2Cyl focuses on reconstruction of generalized cylinders only [Uy et al. 2022], and [Hu et al. 2022] requires a fixed user-defined template. For our problem of sketch-based reconstruction, both of these assumptions are too limiting.

A key stage in our reconstruction pipeline is segmentation. Wang et al. [2022b] propose a method for segmentating point clouds, after which, each segment can then be fitted with a geometric primitive using standard machinery. In contrast to this method, we perform segmentation in the image space, allowing us to reconstruct the drawn boundaries between geometric primitives in a manner consistent with the artist's intent.

Finally, an exciting avenue of work is the use of generative models, such as SolidGen [Jayaraman et al. 2022] or DeepCAD [Wu et al. 2021]. Adapting these methods to reconstruct CAD shapes from sketches is an interesting alternative that has not been investigated.

3 OBSERVATIONS AND OVERVIEW

We focus on *presentation* sketches [Gryaditskaya et al. 2019], which contain few construction or auxiliary lines and mostly consist of ridges and occlusion contours. In order to reconstruct the artist-intended machine-made 3D shape despite the ambiguities and inaccuracies in the sketch, we analyze perception and modelling research, and formulate the following observations that guide our algorithmic choices.

- Segmentation: Human observers can consistently segment and classify sketches of machine-made shapes into the primitives. Even if the lines are approximate, one can use that segmentation to compensate for inaccuracies. Machine-made models typically consist of *patches*, which can be roughly categorized into the following types: planes, cylinders, tori, cones, spheres, surfaces of revolution, and others [Eissen and Steur 2011; Koch et al. 2019]. Our hypothesis is that artists draw in a way that permits the shape of each patch to be easily recognizable, while exploiting the natural ability that humans have for classifying patches.
- Fidelity to the drawing: In general, we expect the 3D reconstruction to stay close to the sketch. We note however, that this requirement directly contradicts the segmentation requirement: due to inaccuracies typical for a natural sketch, the curves where primitives intersect are often drawn imprecisely. For example, two planes may be drawn intersecting along a curved (i.e., not straight) line, which is geometrically impossible. We therefore aim to preserve the shape of the primitive boundaries, but only subject to the segmentation requirement. Moreover, as known in the modeling and perception literature, human observers prioritize preserving curve parallelism to preserving absolute positions [Xu et al. 2014]. Thus, we allow translation of the drawn strokes more easily than deformation of their shape.
- Piecewise smoothness: Consistent with perception research and line drawing analysis [Cole et al. 2008; Wang et al. 2021], we expect all sharp features of the surface to be explicitly drawn. A stroke may also be an occlusion contour, in which case it signifies depth discontinuity, an external silhouette, a sharp ridges, or it may simply denote a boundary of a geometric primitive/patch. We therefore interpret the space between the drawn strokes, i.e., the white space or shaded space, to be smooth. Moreover, this also implies that the drawn strokes themselves, taken as curves in 3D, are smooth unless otherwise explicitly indicated in the drawing. We therefore enforce smoothness of both the surface between the drawn strokes and the reconstructed depth of the drawn strokes (see Sec. 5.2). To correctly reconstruct the shape, we detect which of the drawn strokes are occlusion contours and enforce depth discontinuities accordingly.
- Minimal occlusion: Perception research [Blanz et al. 1999] indicates that human observers prefer views containing few or no occlusions to the ones where important features of an object are occluded. Within our framework, we interpret this recommendation by minimizing the length of strokes classified as occlusion contours, strongly preferring to interpret strokes as ridges instead.
- Regularity: As suggested by perception and modeling work [Xu et al. 2014], we speculate that human observers rely on regularity cues when interpreting sketches of machine-made shapes. In particular, when then the normals or axes of primitives are nearly parallel or orthogonal, we expect the viewers to interpret them as exactly parallel or orthogonal respectively. Therefore, in our pipeline, after detecting pairs of primitives with nearly parallel or orthogonal normals or axes, we enforce this as a constraint in a subsequent optimization. Other shape regularity constraints, such as cylinder concentricity or straightness of boundaries, can be implemented in a similar fashion. This, however, drives the results further from the input, so we do not enforce them.

Leveraging these observations, we use deep networks to predict per-pixel depths as well as a segmentation and classification into geometric primitives (Fig. 4, Sec. 4.1). Using these classes, we fit primitives to the depth prediction within each region (Sec. 4.2). We then vectorize the input image into strokes and identify the strokes with the primitives (Sec. 5.1). We use the preliminary primitive fits in our main optimization, where we lift the strokes into 3D, while deciding which of them are occlusion contours and refining the primitives so as to intersect at those lifted 3D strokes (Sec. 5.2).

Finally, we triangulate the refined 2D sketch, transfer the segmentation onto this new triangulation (Sec. 6.1), and lift it into 3D to lie on the optimized primitives so as to interpolate the lifted 3D strokes (Sec. 6.2).

4 EXTRACTING GEOMETRIC PRIMITIVES

The input to our pipeline is a greyscale bitmap sketch, containing a single machine-made shape. We start by predicting depth of each pixel in the sketch and segmenting the input sketch into separate geometric primitives, leveraging the structure of sketches of man-made shapes. For simplicity of exposition here we assume that sketches are drawn with an orthographic projection; our pipeline can ultimately handle some gentle perspective (more discussion in Sec. 7).

4.1 Image Segmentation and Depth

For both the depth and the segmentation, we train two encoder-decoder convolutional neural networks. For depth estimation, the network outputs one scalar per pixel; for segmentation, it outputs a probability of a pixel belonging to each shape class or background. We use non-photorealistic rendering of ABC dataset [Koch et al. 2019] as training data. For architecture, training procedure, and dataset details please refer to App. A and B.

Segmenting into primitives. For non-background pixels, the segmentation network provides probabilities of belonging to each class of geometric primitives. To actually segment the image, i.e., separate it into connected regions of a particular class, one normally selects for each pixel the class with the maximum probability. We find, however, this strategy to be unreliable for line drawings (Fig. 5). First, such strategy is unable to discern two different primitives of the same class next to each other. Second, the boundaries of the segmentation regions extracted in such manner often do not align with the drawing (Fig. 5b).

We observe that artists often (although not always) delineate boundaries between separate geometric primitives when drawing machine-made shapes (Fig. 5). We therefore can use the drawn strokes to segment the input sketch into regions, integrate the predicted per-pixel probabilities over these regions, and pick the maximum for each. Such a segmentation along the drawn strokes, however, is not trivial, as artists often leave gaps between strokes. To address that, we use a traditional the trapped ball algorithm [Li 2018; Zhang et al. 2009], with a minimum radius of 1px and a maximum of 4px.

Note, however, that in some cases artists omit the separating lines between different primitives (Fig. 22). We discuss this in Sec. 8.1.

4.2 Local Geometry of Each Primitive

Leveraging the per-pixel depth and the segmentation of the input sketch, we now find the parameters of each geometric primitive; at this stage, independently of other primitives. This provides the initial 3D shape that roughly corresponds to the drawing and the estimated depth, but does not have any connectivity between the primitives (Fig. 4, 9). In the subsequent stages (Sec. 5), we assemble these disconnected primitives into a connected 3D shape.

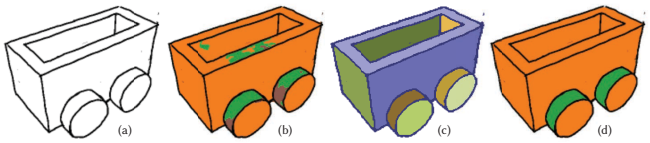


Fig. 5. Given the predicted segmentation, naïvely picking the most probable class per pixel leads to regions that are not aligned with the drawn strokes (b). Instead, we use the Trapped Ball algorithm [Zhang et al. 2009] robustly separating the input sketch (a) into regions (c). We then integrate the per-pixel primitive probabilities over these regions and choose the maximum, improving the segmentation results (d). Here planes are orange, cylinders are green, and tori are brown. Input image [Delanoy et al. 2019].

We use standard techniques to fit a geometric primitive to an extracted depth of each region by minimizing l_2 distance (Appendix C). Note that we use the same fitting energies in this stage as we do in subsequent stages (Sec. 5). Here, because we are fitting each primitive independently, we use closed-form solutions when possible.

Some of these optimization problems, such as the ones for cylinder and circular cone, are non-convex and depend heavily on the initial guess. We use two strategies to compute a set of initial guesses and choose the fit with the minimal error among the different options. First, following the regularity observation (Sec. 3), we try the axes of previously identified primitives as initial guesses to fit the new cylinders and cones. For robustness, we start by fitting planes, then continue by fitting the cylinders in decreasing order of area of their 2D region, as larger regions have more data and thus are more reliable. Second, we use the classic property of cylinders and cones: Their normals, interpreted as points on a unit sphere (also known as a Gauss map), form a circle on a plane with the axis of the primitive as the normal. Therefore, in order to compute the initial guess, we estimate per-pixel normals via finite differences, compute the Gauss map, and fit a plane and a circle to the points on the unit sphere using standard methods. We then use the normal of that circle as the initial guess for the axis of the cylinder or cone, and find the rest of parameters via a standard least squares fitting.

5 BOUNDARY AND PRIMITIVE OPTIMIZATION

Given the collection of initial primitives, our goal is to assemble them into a manifold 3D shape, refining their parameters such that their boundaries match the sketch strokes (Fig. 4). To do so, we first vectorize the input sketch and identify the vectorized strokes with the primitives, thus determining primitive boundaries (Sec. 5.1). We then lift those boundaries into 3D, while simultaneously detecting occlusions and refining the 3D primitives (Sec. 5.2).

5.1 Identifying primitive boundaries

The identification of primitive boundaries invovles two steps. First we vectorize the bitmap image, and then we compute stroke-primitive correspondences.

Vectorization. We vectorize the input sketch using the method of Puhachov et al. [2021] with their default parameters. Their algorithm may produce minor artifacts, so we perform a simple cleanup.

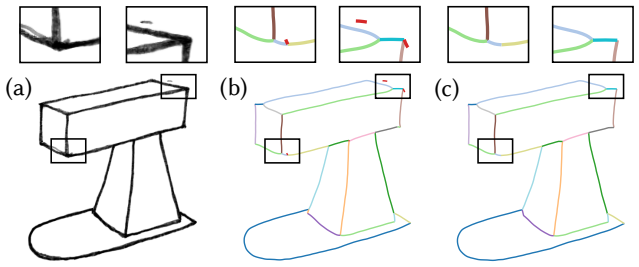


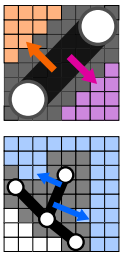
Fig. 6. We vectorize the input sketch (a) using the method of Puhachov et al. [2021] (b), and then postprocess their results using simple heuristics, e.g., removing the short curves in red (b,c). Input images [Eissen and Steur 2011].

We remove all strokes shorter than a pixel and split curves at intersections. We also connect curves if their endpoints are within a threshold distance (10px). We then remove short curves (with arclength < 20px in our implementation) if they are not connected to any other curve. We also remove curves if at least half of their length belongs to the background (Fig. 6).

For rough drawings with overdrawn strokes, the vectorization often produces many disconnected strokes. Hence, we treat it as a conservative estimate of the sketch connectivity: We consider the points connected by the vectorization to be connected; we do not draw any conclusions regarding the points disconnected by the vectorization.

We sample each vectorized stroke uniformly with a fixed step size, yielding a polyline $\{(\hat{x}_1, \hat{y}_1), \ldots, (\hat{x}_n, \hat{y}_n)\}$, where $(\hat{x}_i, \hat{y}_i) \in \mathbb{R}^2$. In our implementation, for the input images of 512^2 resolution, we use a step size of 10px for performance reasons.

Stroke-primitive correspondence. For each edge of a vectorized stroke we determine its adjacent primitives (Fig. 7).



For each side of the edge, we find the closest non-boundary pixel such that the straight line between the pixel and the edge lies completely inside the sketch, i.e., does not cross any background pixels (inset, top). We then associate the side of the edge with that pixel's region. Whenever only one side of an edge finds a primitive, or when the two sides find the same primitive, we mark this edge as *free* (inset, bottom). Otherwise, the edge is called *shared* (inset, top). All the shared boundaries directly influence the primitive fitting in later steps: we interpret them as an intersection of two primitives or an occlusion contour. For an edge e, we denote the set of adjacent primitives as $P(e)$.

5.2 Lifting boundaries into 3D

Equipped with these identifications, we now lift the vectorized strokes into 3D by computing depth coordinates $z_i \in \mathbb{R}$ for each vertex i (Fig. 8) and slightly adjusted image plane (x_i, y_i) coordinates to compensate for inaccuracies in the sketch.

The core principle of this optimization is to find the shape of each geometric primitive such that their intersections align with

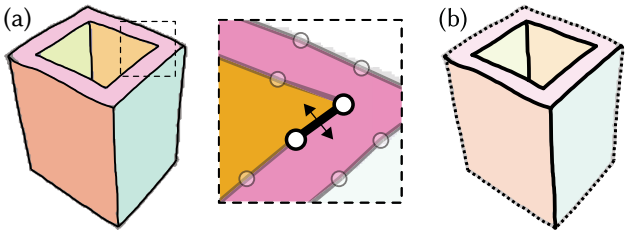


Fig. 7. We determine adjacent segmentation regions for each edge of vectorized strokes. We define edges adjacent to only one region as *free* boundaries (dashed lines), and other edges adjacent to two regions as *shared* boundaries (black solid lines). Input image [Delanoy et al. 2019].

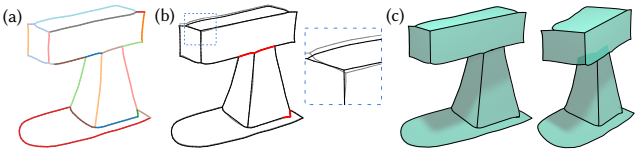


Fig. 8. Our main optimization leverages the depth map, segmentation, and the vectorization to lift the vectorized 2D primitive boundaries (a) into 3D (c). This optimization refines the 2D drawn strokes, decides which strokes are occlusion contours, and refines the 3D primitives so that they connect at the drawn strokes (b). Input image [Eissen and Steur 2011].

their shared boundary strokes. Note, however, that some shared boundary strokes may only belong to one visible primitive in 3D, i.e., when the shared boundary is an occlusion contour (Fig. 8). In other words, primitives adjacent to a shared boundary in 2D, need not be adjacent in 3D and forcing them to have an intersection would be incorrect. Thus, the key component of our optimization is to decide which strokes depict occlusion contours. This allows us to preserve the intended discontinuities in the sketch.

A further complication is that a single artist stroke may be only partially an occluding contour (Fig. 4, the boundary of a hole on top of the cylinder is a single stroke that is partially an occlusion contour, partially a ridge). We therefore encode this decision as a binary variable $b_e \in \{0, 1\}$ per edge e of vectorized shared boundary strokes, where 1 denotes that the shared edge is part of a non-occluding contour. For free boundary edges e we let $b_e = 1$.

Finally, due to the inherent ambiguity of 3D reconstruction from 2D, as well as distortions typical for natural sketches, the predicted depth (Sec. 4.1) is often severely distorted. Naïvely using this depth to fit primitives within each patch directly contradicts the drawing or produces gaps around shared edges (Fig. 9). Instead, we prioritize fidelity to the sketch while allowing the primitives to deviate from the predicted depth within a certain bound.

Setting this maximum deviation bound manually, however, is not trivial; instead, we minimize it. We note that for a vector-valued function $f : \mathbb{R} \to \mathbb{R}^n$, an optimization problem (in the epigraph form) minimizing such bound $t \in \mathbb{R}$,

$$\min_x t \quad \text{s.t.} -t \le f_j(x) \le t \; \forall j,$$

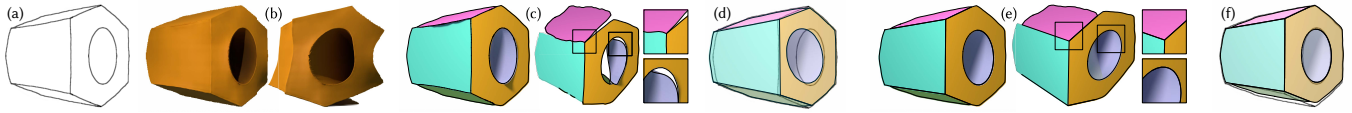


Fig. 9. Due to distortions typical for natural sketches, the depth prediction networks often output heavily distorted surfaces (b), which cannot be directly used to fit primitives, either causing gaps if primitives are cut at the input strokes (c) or misalignment with the sketch if cut at the primitives' intersections (d). Instead, our optimization simultaneously resolves primitive shapes and 3D stroke shapes while identifying occlusion contours, thus reconstructing the intended shape (e) and staying true to the original drawing (f). Input image [Manda et al. 2021].

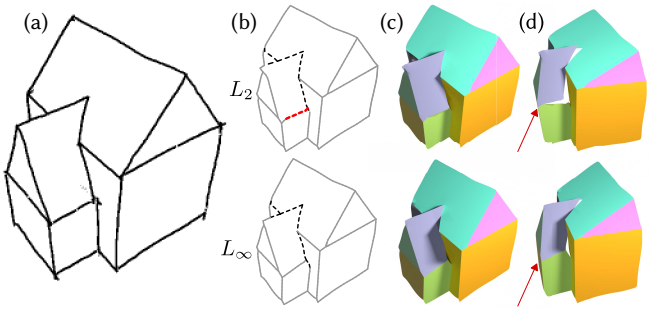


Fig. 10. Using L_∞ distance allows patches to deviate from the predicted depth within a bound, resulting in fewer occlusion contours in the final result, compared with L_2. Input image [Delanoy et al. 2019]

is equivalent to solving

$$\min_x \|f(x)\|_\infty.$$

Hence we propose to minimize L_∞ distance to the predicted depth map, thus intuitively allowing the primitives to freely deviate from the predictions within a certain minimized upper bound (Fig. 10).

This observation guides the first term in our optimization problem, an l_∞ distance,

$$E_{\text{depth}} = w_{\text{depth}} \max_{(x',y')} \left(z' - \text{Proj}_P(x',y')\right)^2, \tag{1}$$

where z' is the estimated depth (Sec.4.1), w_{depth} is the weight for this term, and $\text{Proj}_P(x',y')$ provides the depth of the point $(x',y',0)$ after $z-$projection onto the primitive P corresponding to the non-background pixel (x',y') in the sketch.

The remaining terms in our optimization problem focus on the strokes. Thus, the following optimization problem lifts the boundaries into 3D, detects occlusion contours, and adjusts primitive parameters:

$$\min_{x,y,z,b,p} E_{\text{depth}} + E_{\text{fit}} + E_{\text{pos}} + E_{\text{shape}} + E_{\text{smooth}} + E_{\text{occlusion}}. \tag{2}$$

Here, the free parameters are the lifted stroke geometry points (x,y,z), the binary variables indicating occlusion vs. ridge b, and primitive parameters p. Details about the individual terms in Eq. 2 are described below, in the order they appear:

- Stroke fitting. This term minimizes the distance from each edge e of a 3D stroke to its corresponding primitives. We minimize this distance only when it is not part of an occluding contour, i.e., $b_e = 1$:

$$E_{\text{fit}} = w_{\text{fit}} \sum_{e\in E,\ p\in P(e)} b_e \ \text{dist}(e, \text{Proj}_p(e))^2. \tag{3}$$

  Note that occlusion boundary edges are not lifted in this optimization, but we do handle them later (Sec. 6.2). In practice, we discretize the L_2 distance from the edge to the primitive by taking the edge midpoint, i.e.,

$$\text{Proj}_p(e_i) \equiv \text{Proj}_p\Big(\frac{x_i + x_{i+1}}{2}, \frac{y_i + y_{i+1}}{2}\Big).$$

- Stroke Position. Following our principles, we allow for only small refinement of the drawn strokes, so this term minimizes the distance to the original vertex positions of each stroke s. We normalize the term by $|s|$, the number of vertices in the stroke:

$$E_{\text{pos}} = w_{\text{pos}} \sum_s \frac{1}{|s|} \sum_{(x,y)\in s} \|(x,y) - (\hat{x},\hat{y})\|_2^2. \tag{4}$$

- Boundary Shape. Following our principle of fidelity, we are aiming to preserve the shape of each vectorized stroke and prefer a smooth reconstruction of its z coordinate. Precisely, for a stroke $s = \{(x_1,y_1),\ldots,(x_n,y_n)\}$, we formulate shape preservation using Laplacian coordinates [Sorkine et al. 2004]. That is, we preserve vectors Δx_s and Δy_s, where Δ is a 1D Laplacian operator [Desbrun et al. 1999] and $x_s, y_s \in \mathbb{R}^n$ are coordinate vectors of stroke s. We additionally aim to preserve stroke tangents, which we compute with a finite difference operator D. For brevity, we define $\bar{x}_s = x_s - \hat{x}_s$ and $\bar{y}_s = y_s - \hat{y}_s$, which lets us write

$$E_{\text{shape}} = w_{\text{shape}} \sum_s \|\Delta\bar{x}_s\|_2^2 + \|\Delta\bar{y}_s\|_2^2 + \|D\bar{x}_s\|_2^2 + \|D\bar{y}_s\|_2^2. \tag{5}$$

- Depth Smoothness. Our next term enforces smoothness of our reconstructed 3D strokes using a Laplacian energy:

$$E_{\text{smooth}} = w_{\text{smooth}} \sum_s \|\Delta z_s\|_2^2. \tag{6}$$

- Occlusion. Finally, following our minimal occlusions principle, the last term aims to minimize the total length of occlusion contours, which is, due to uniform sampling, equivalent to the number of edges with b_e equal to zero:

$$E_{\text{occlusion}} = w_{\text{occlusion}} \sum_{e\in\mathcal{J}} (1 - b_e). \tag{7}$$

  Here, $\mathcal{J}$ is the set of shared boundary edges.

Following the regularity observation (Sec. 3), we formulate constraints for axis orthogonality and alignment between some patches. Using the initial primitive fits (Sec.4.2), we mark primitive pairs as nearly aligned or orthogonal when the angle between their axes is within an $\varepsilon_{\text{angle}}$ threshold from 0°or 90°, respectively. We then maintain those angles within the threshold via non-linear inequalities, i.e., for an orthogonal pair i, j,

$$\left(\frac{a_i \cdot a_j}{\|a_i\|\|a_j\|}\right)^2 \leq \cos^2 \varepsilon_{\text{angle}}, \tag{8}$$

and a similar inequality for the aligned pairs.

We use the following parameters in our optimization: $w_{\text{depth}} = 1$, $w_{\text{fit}} = 1000$, $w_{\text{pos}} = 10$, $w_{\text{shape}} = 100$, $w_{\text{smooth}} = 1$, $w_{\text{occlusion}} = 0.005$, $\varepsilon_{\text{angle}} = 14°$.

Solver. The integer constraints on the b_e variables (easily hundreds per sketch) make this optimization problem impractical. Instead, we empirically observe that a relaxation of this problem, i.e., replacing the integrality constraints by inequalities $0 \leq b_e \leq 1$, always converges to integer values. While we have no formal proof of this phenomenon, we use this relaxation in our implementation.

Therefore, we solve the nonlinear optimization problem with equality and inequality constraints in Eq. 2 via interior point method (IPOPT 3.11.1 [Wächter and Biegler 2006]) until convergence with their default settings. This provides us with the final primitive boundaries and refined primitive shapes (Fig. 8).

6 COMPUTING THE FINAL 3D SHAPE

We now find the final piecewise smooth mesh on the inferred primitives interpolating the calculated 3D boundary strokes, allowing for discontinuities only at the occlusion contours. We start by triangulating the sketch interior, then perform an optimization balancing the piecewise smoothness of the final surface with conformity to the 3D strokes and the primitives.

6.1 2D Triangulation

Our goal is to create a piecewise smooth mesh, enforcing smoothness everywhere except for the calculated boundary curves. To do so, we need a *segmented* triangulation of the sketch containing the 2D strokes refined in Sec. 5.2. To get the 2D triangulation, we first compute the conforming Delaunay triangulation with the *original* 2D sketch strokes as constrained edges. In our implementation, we use the Triangle library [Shewchuk 1996]. This outputs a triangulated set of vertices with coordinates $(\hat{x}_j, \hat{y}_j)$, $j = 1, \ldots, N$, which include the original polyline vertices and new vertices in the interior. Using the As-Rigid-As-Possible framework [Sorkine and Alexa 2007], we then compute deformed vertices (X_j, Y_j) of the triangulation such that the initial sketch strokes align with the refined sketch strokes. Finally, we cut this triangulation (duplicating vertices) along the predicted occlusion contours.

6.2 Optimizing the complete shape

We now lift the triangulation vertices (Fig. 11) into 3D, i.e., find $Z_j \in \mathbb{R}$ for each triangulation vertex $(X_j, Y_j) \in \mathbb{R}^2$, with the non-occlusion stroke vertex depths fixed to the values calculated previously. Note that the optimization in Sec. 5.2 does not optimize stroke depth along occlusion contours. We therefore do not constrain the Z values of the vertices along the occluded contours.

For regions classified as planes, cylinders, cones or spheres, our goal is to project the 2D triangulation vertices onto the corresponding primitives. Naïvely performing z-projection, i.e., finding an intersection of the ray in the view direction with the primitive, however, will often fail. This is because some 2D vertices may lie outside the fitted primitive, so the ray does not have an intersection point (e.g., if the optimization in Sec. 5.2 underestimates the radius of a cylinder or a cone). In other cases, the ray might have two intersections, so one needs to choose the appropriate one, which is not necessarily the closest one (e.g., circular hole in the middle of the 'nut' example, Fig. 20).

Instead, we optimize the following energy, balancing projection distance against fairness:

$$\begin{aligned} &\min_{Z \in \mathbb{R}^N} E_{\text{primitives}}(Z) + w_{\text{fairness}} E_{\text{fairness}}(Z), \\ &\text{s.t. } Z_j = z_j, \ \ j \in \text{non-occlusion boundary}. \end{aligned} \tag{9}$$

The $E_{\text{primitives}}$ term is a sum of squared $z-$distances between the mesh vertices and the corresponding primitive. We first project each triangulation vertex $x_j = (X_j, Y_j, Z_j)$ along the ray parallel to the z-axis onto the primitive, yielding point x_j^*, and then compute $\|x_j - x_j^*\|_2^2$. When this ray has multiple intersections with the primitive, we take the closest one; when there is no intersection, we take the closest distance between the ray and the primitive instead. This term is zero in regions classified as 'others'.

The E_{fairness} term is a squared Laplacian energy on each region:

$$E_{\text{fairness}} = \sum_p \int_p \|\Delta Z\|^2 dA, \tag{10}$$

where p iterates over all primitives. Here, the Laplacian operator is computed once on the flat triangulation with the edges cut along the occlusion contours, which allows to introduce discontinuities there.

For all our experiments, we set $w_{\text{fairness}} = 0.1$. For parametric patches, this optimization performs a smooth projection of the triangulation onto the primitive. For other patches, $E_{\text{primitives}} = 0$, so this optimization is Laplacian smoothing.

The energy in Eq. 9 is non-convex, and, in general, non-trivial to optimize due to the $E_{\text{primitives}}$ term. We propose an efficient ADMM algorithm [Boyd et al. 2011], which alternates between projecting the points onto the primitives and smoothing the result with the fairness term. More specifically, we first eliminate the fixed variables Z_j, $j \in$ non-occlusion boundary, making the original problem unconstrained, and rewrite an equivalent optimization problem in the standard ADMM form:

$$\begin{aligned} &\min_{Z, \tilde{Z} \in \mathbb{R}^N} E_{\text{primitives}}(Z) + E_{\text{fairness}}(\tilde{Z}), \\ &\text{s.t. } Z = \tilde{Z}. \end{aligned}$$

We then form the augmented Lagrangian for this problem,

$$\mathcal{L} = E_{\text{primitives}}(Z) + E_{\text{fairness}}(\tilde{Z}) + \lambda^T (Z - \tilde{Z}) + (\rho/2)\|Z - \tilde{Z}\|_2^2,$$

where λ is a vector of dual variables. We use $\rho = 0.2$ in our results. The actual optimization consists of alternating between minimizing

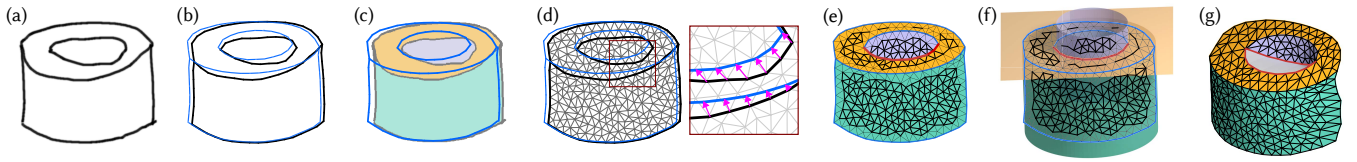


Fig. 11. After lifting the boundaries to 3D and refining their 2D shape (Sec. 5.2), the original strokes (a) are deformed (b, new strokes in blue) and no longer align with the segmentation of the input (c). We therefore triangulate the original vectorized drawing (d), deform it to match the new strokes (d,e), resulting in segmented triangulation, cut at occlusion contours (e,red), and leverage the fitted primitives (f) to compute the final 3D surface (g).

the augmented Lagrangian $\mathcal{L}$ over Z keeping $\tilde{Z}$ fixed, minimizing over $\tilde{Z}$ keeping Z fixed, and a dual update (see [Boyd et al. 2011] for more details). The solution to the first optimization problem has a closed form solution computed via $z-$projection of points onto the primitives; the second one is a linear solve. For the latter, we use SciPy sparse linear system solver, and perform ten ADMM iterations in our implementation.

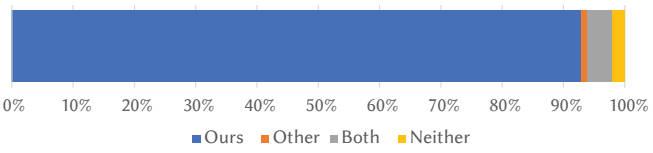


Fig. 12. Summary of comparative preferences in our perceptual user study. Participants strongly preferred our results over the alternatives [Delanoy et al. 2019; Guillard et al. 2021].

7 RESULTS AND VALIDATION

Throughout the paper, we demonstrate the performance of our method on natural sketches (e.g., Fig. 1, 4, 2,9). We demonstrate additional results with primitives highlighted in Fig. 20. These include both shapes from standard categories such as chairs and cars, as well as miscellaneous machine-made shapes. Some of the input drawings are in orthographic projection or weak perspective (e.g., Fig. 4,14, bottom), some are with more pronounced perspective (e.g., Fig. 20, chair, building). Our inputs contain five sketches from [Delanoy et al. 2019], one sketch from OpenSketch dataset [Gryaditskaya et al. 2019] (Fig. 20, house), two sketches from [Manda et al. 2021], and four new sketches, including three derived from [Eissen and Steur 2011] (Fig. 4, Fig. 6, Fig. 20 'bin').

Qualitative Evaluation. We asked 7 non-professionals and 6 artists to comment on the results of our algorithm. We showed each one of them an input sketch and two views of our algorithmic result and asked to comment on the following statement, separately for each sketch, "The shapes (bottom) capture the intent of the drawing (top)". The questionnaire presented five Likert-type reply options: "Strongly disagree" (-2), "Disagree" (-1), "Neither disagree nor agree" (0), "Agree" (1), "Strongly Agree" (2). On average, the participants agreed with the statement ($avg = 1.4, std = 0.74$). After the questionnaire, whenever a user disagreed for at least one example, we performed a short follow-up semi-structured interview with a single question: "Do you have any comments on some of your choices?". Two of the participants commented on the our 'Wagon' and 'Train' having only two wheels instead of four. We note that we only reconstruct the visible surface, while the two extra wheels are fully hidden in the input sketches.

7.1 Comparison with Prior Art

We compare our method to prior art, including methods that produce full 3D models from bitmap sketches [Delanoy et al. 2019; Guillard et al. 2021][1], a method that only produces normals [Hudon et al. 2018], and a recent method that produces shading [Zheng et al. 2020]. We additionally compare our algorithmic results to the state-of-the-art system that converts a 3D point cloud into a manifold machine-made shape [Guo et al. 2022].

Qualitative Comparisons. We qualitatively assess our results by comparing them to algorithmic alternatives via a comparative perceptual study. Study participants were shown input sketches, together with rendered images of 3D reconstructions from our method and an alternative method. The input marked as 'A' is shown at the top, and the two reconstruction outputs are placed at the bottom and marked as 'B' and 'C'. Participants were asked to select the output that is a more accurate reconstruction of the sketch: "Which of the blue shapes on the bottom, B or C, more accurately represents drawing A (top)?". The answer options were 'B', 'C', 'Both', and 'Neither'. We collected answers for each query from 17 different participants (12 males, 5 females, 18-35 years old). The protocol, the study layout, and all study data is provided in the supplementary material.

Unlike our system, which targets arbitrary machine-made shapes, the work of Zhong et al. [2022] and Guillard et al. [2021] are limited to the standard ShapeNet shape categories, such as chairs or cars. Because of this, we only compare with these methods on those objects (Fig. 15, 14). The method of Delanoy et al. [2019] can accept one or more views of an object as input. Since we reconstruct the object from a single view, for fairness, we run their method with a single image as input.

Participants preferred our results over those of Delanoy et al. [2019] 93% of the time, and preferred the alternative only 1% of the time. Precisely, on the 'camera' input (Fig. 14 top), one person preferred the result of Delanoy et al. [2019], and four participants

[1]We do not compare to [Zhong et al. 2022], since despite having their code available, their pretrained models are not available, and training together with precomputation takes 7 weeks according to their measurement.

judged our results both accurate. For the 'bearing' input (Fig. 14, bottom), two participants answered 'neither'. Nobody preferred any of the results of Guillard et al. [2021] to ours. This outcome is consistent with the expectation that human observers treat the artist strokes as meaningful in the specification of a concrete 3D object.

For completeness, we also qualitatively compare results of methods that produce normals [Hudon et al. 2018] or shading [Zheng et al. 2020] with the normals or shading of our 3D reconstructions (Fig. 17). Both of these methods were trained on natural shapes, so for machine-made shapes we notice that they tend to predict smoothly varying normals, even on planar faces. Our system, however, produces precise normals and shading characteristic of piecewise-smooth machine-made shapes with sharp ridges and occlusions.

Finally, we qualitatively compare our algorithmic results with those of Guo et al. [2022], which takes a point cloud as an input. While we do not have the complete point cloud, we run their method with our predicted depth map, converted into a point cloud, as an input. The comparison is presented in Fig. 16. Their method targets complete point clouds with some noise, and is unable to process partial point clouds with heavily distorted shapes, which is typical for natural sketches. Our method successfully alleviates these imprecisions despite starting from the same predicted depth. We ran their code with the default parameters; note that their official implementation does not fully implement their method, skipping primitive trimming [Guo 2023].

Quantitative Comparisons. Our algorithm aims to reconstruct the 3D machine-made object perceived by the viewer, as the artist-intended it from a sketch. To validate this, we asked each of three 3D modeling experts to manually model seven of our input sketches in a CAD software of their choice. The experts took roughly from 5 to 45 minutes to model each object, while our algorithm inferred each shape in less than two minutes on average (Sec. 7).

The manual modeling results are visually similar to ours (Fig. 13 and Supplementary). We have furthermore measured chamfer distance from our 3D models to the 3D objects modelled by experts, as well as chamfer distance between 3D models of different experts, to highlight the natural variation in interpretation. Since our method only reconstructs the visible surface of the object, we compute the one-sided chamfer distance, i.e., from points on the reconstruction to the manual model. We normalize all the distances by the shape diameter.

Over the seven models, the average chamfer distance from our algorithmic results to the models created by artists is 0.0476%, which is very close to the average chamfer distance between different artists' interpretations, 0.0428%.

7.2 Ablation study

Parameter Sensitivity. We show (Fig. 21) that our method produces plausible results for a range of parameters. Varying $w_{\text{occlusion}}$ controls how many edges are identified as occlusion contours; an extremely large number may make the final shape more flat. For a large range of parameter values, however, the changes in the result are insignificant. Varying w_{fit} controls how much we trust the primitives, so a higher weight would prioritize refining the shared boundaries. Finally, varying w_{depth} controls how much we trust the predicted depth.

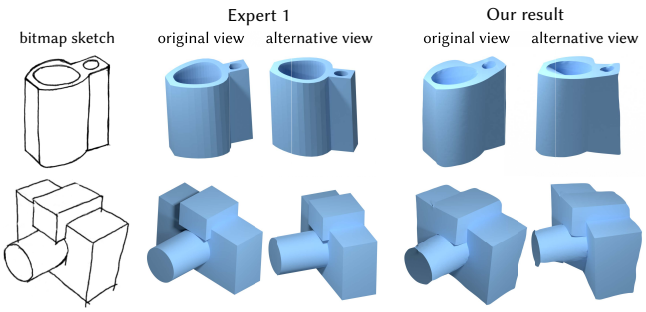


Fig. 13. Compared to manually created models from the same sketches, our algorithmic results are similar both visually and quantitatively. Input images: bin [Eissen and Steur 2011], camera [Delanoy et al. 2019].

Table 1. Performance statistics for our pipeline in seconds. We separately include the timings for the vectorization algorithm of Puhachov et al. [2021].

Name	Fig.	Vect.	Our time	Total
armchair	22	52	85	137
bin	20	32	83	115
building	20	24	54	78
camera	13	50	105	155
car	20	42	76	118
chair30	20	32	75	107
chair37	15	36	76	112
cylinder	16	17	69	86
goblet	21	24	61	85
hexnut	9	22	81	103
house	10	35	72	107
machine	1	100	97	197
nut	20	29	69	98
pulley	14	43	147	190
train	20	40	72	112
wagon	2	47	86	133

Performance. We implemented our system in Python using PyTorch for neural networks in Sec. 4.1, and Pyomo 6.5.0 [Bynum et al. 2021; Hart et al. 2011] for optimization in Sec. 5.2. All the results presented in the paper were computed with the default parameters presented in the text. On our desktop machine (Intel® Core™ i7-9700K CPU @ 3.60GHz with NVIDIA® GeForce® RTX 2080Ti), the results take 82 seconds on average, not including vectorization. Full statistics are in Table 1. Most of the time is spent in the boundary shape optimization (Sec. 5.2). Training the depth network took 9 hours; training the segmentation network took 30 hours.

Ablation Study. We perform an ablation study of our method (Fig. 19). We demonstrate results on a few examples, each time skipping one stage or a constraint in our pipeline. Skipping the boundary optimization stage (Sec. 5.2) by simply projecting the 2D strokes onto the primitives fitted independently, yields the final shape with

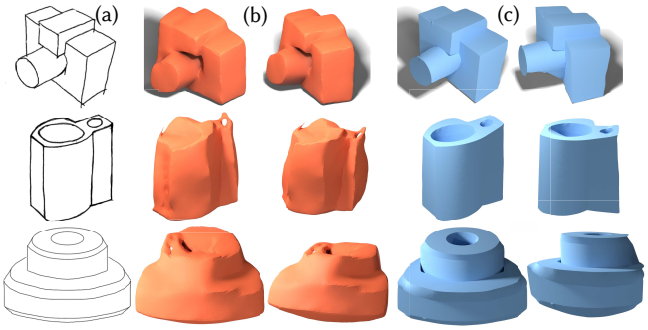


Fig. 14. Compared to the smooth results of [Delanoy et al. 2019] (b, orange), our results (c, blue) are piecewise smooth surfaces that are explicitly controlled by the drawn strokes (a). Input images: camera [Delanoy et al. 2019], bin [Eissen and Steur 2011], pulley [Manda et al. 2021].

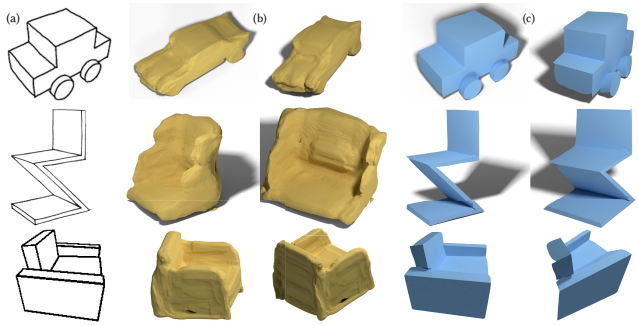


Fig. 15. Even on a fixed shape category, typical end-to-end learning pipelines, such as Sketch2Mesh [Guillard et al. 2021] often produce shapes (b) that only roughly resemble the input sketch (a). Our result (c). Input images: car [Delanoy et al. 2019], chair [Eissen and Steur 2011], armchair [Guillard et al. 2021]. Note that the armchair image is a non-photorealistic rendering, not a natural sketch.

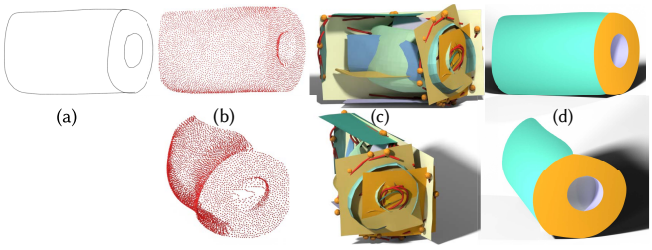


Fig. 16. State-of-the-art approaches reconstructing machine-made shapes from point clouds, such as [Guo et al. 2022], often target complete or almost complete point clouds with, perhaps, some noise. Our depth, inferred from a sketch (a) and converted into a point cloud (b), only contains depths of the visible part of the shape and thus is incomplete. Moreover, the small distortions in the sketch get propagated into the 3D shape, impeding the use of these approaches (c). Our framework is designed to alleviate these distortions (d). Here we visualize only 4% of the points in the point cloud. Input image [Manda et al. 2021].

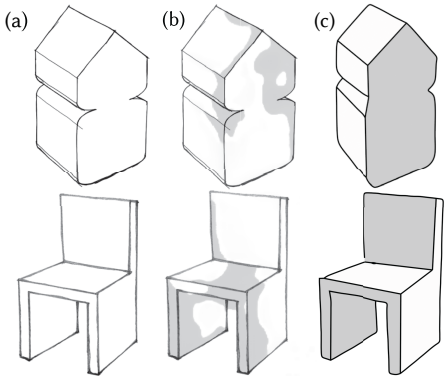


Fig. 17. Shading prediction methods, such as [Zheng et al. 2020], are typically trained on images of natural shapes with smoothly varying normals. For sketches of machine-made shapes (a), these methods often produce unrealistic, smoothly varying shading (b). Our reconstruction result rendered in a similar style (c). Input images: building [Gryaditskaya et al. 2019], chair [Eissen and Steur 2011].

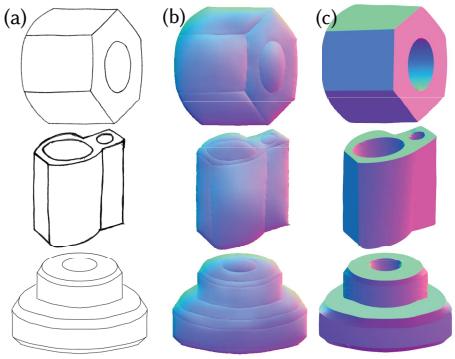


Fig. 18. Normal prediction methods, when trained on natural objects, typically fail to predict correct normals on machine-made shapes, e.g., (b) shows results when trained on characters [Hudon et al. 2018]. Our 3D results, rendered in the same style (c). Input images: nut (top), pulley (bottom) [Manda et al. 2021], and bin (middle) [Eissen and Steur 2011].

gaps at the strokes (Fig. 9c). Removing the regularity constraint (Eq.8) leads to misaligned primitives (Fig. 19a). Performing a naïve projection instead of Sec. 6.2 leads to artifacts whenever the drawn shape disagrees with the fitted primitives (Fig. 19b) and produces suboptimal shapes of non-parametric patches (Fig. 19b, pink).

Segmentation Study. To verify our hypothesis that human observers can consistently segment and classify sketches of man-made shape into different geometric primitives, we performed a smaller user study, where we asked five participants with normal color vision to color the sketch into different colors corresponding to different geometric shapes. The exact task was: "In each drawing, using the colors below, draw the regions depicting the following geometrical shapes: cylinder, sphere, cone, plane, and torus. If none of those fit a region, please mark it as 'other'. Keep the regions outside the shape white.". The results are presented in Fig. 3 and Supplementary. In general, the participants consistently segment

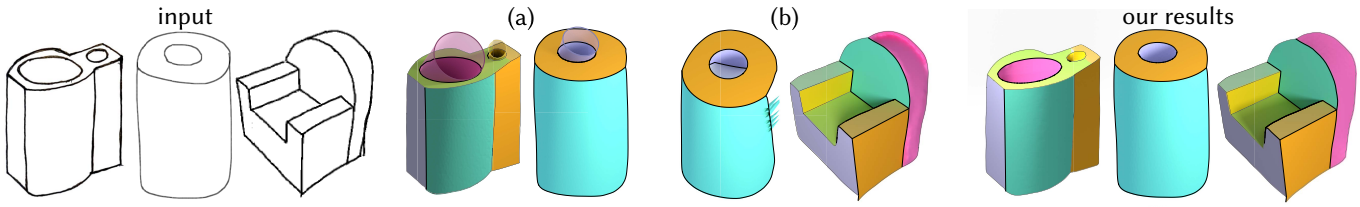


Fig. 19. Ablation study: (a) Omitting regularity constraints (Eq. 8) produces misaligned primitives. (b) Skipping Sec. 6.2 leads to artifacts whenever the drawn shape disagrees with the fitted primitives, or produces inaccurate non-parametric patches (pink). Input images: bin [Eissen and Steur 2011], cylinder [Manda et al. 2021], armchair [Delanoy et al. 2019].

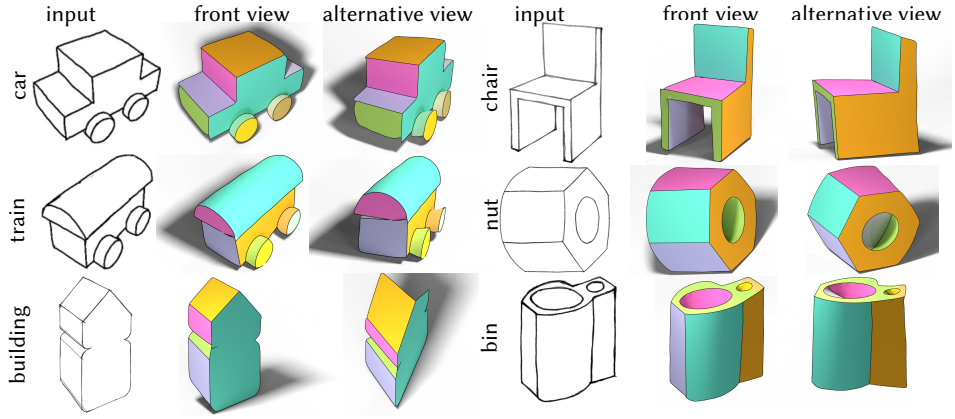


Fig. 20. A gallery of additional results with coloured patches. Input images: car, train [Delanoy et al. 2019], bin, chair [Eissen and Steur 2011], building [Gryaditskaya et al. 2019].

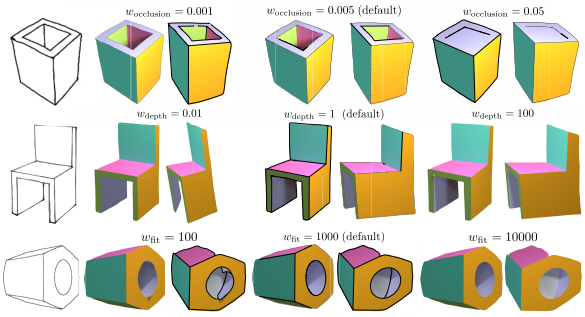


Fig. 21. Our pipeline is robust to a wide range of parameter values. An extremely high value of $w_{\text{occlusions}}$ may prohibit occlusions altogether, leading to a flatter reconstruction (top). Input images: goblet [Delanoy et al. 2019], chair [Eissen and Steur 2011], hexnut [Manda et al. 2021].

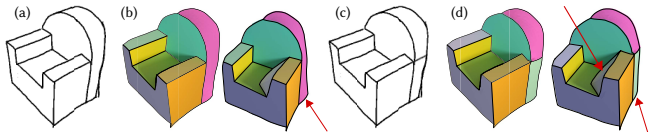


Fig. 22. When artists do not fully delineate primitives (a, rightmost region on the chair's back), our pipeline may produce a suboptimal surface (b, arrow). If desired, artists can complete the delineation (c, the right region now separated in two), which improves the reconstruction (d), in this case making the light-green region planar, as perhaps intended. Furthermore, we only reconstruct the visible part of the surface, so the reconstruction of the invisible parts may disagree with the artist's intent (d, upper arrow). Input image [Delanoy et al. 2019].

and classify the regions. The only disagreement was between a small torus or a sphere (Supplementary). We also note that almost always the drawn boundaries of the regions align with the drawn strokes. For the region where the correct segmentation would create a new boundary, one participant drew that boundary, others marked the region as 'other' (Supplementary, chair).

8 CONCLUSIONS

We have presented and validated a novel system to infer a 3D machine-made shape from a single natural bitmap sketch. Our system is based on the segmentation and a vectorization of the input sketch, allowing us to reconstruct piecewise smooth shapes with ridges and occlusion contours that are precisely aligned with the drawn strokes. Our system can process sketches of different level of roughness, distorted shapes, and extra strokes.

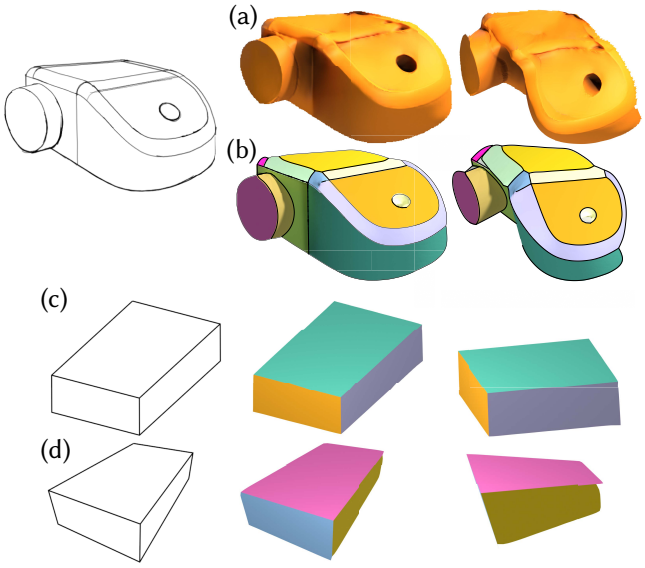


Fig. 23. Limitations: On complex drawings, the typical imprecision in neural networks' outputs (a) can lead to inaccuracies in the final results (b). We do not estimate the perspective in the sketch, always assuming orthographic projection (c.f. c and d). Input image [Gryaditskaya et al. 2019]

8.1 Limitations

Some of our limitations stem from our use of a line drawing algorithm that produces rather clean drawings: Our system does not handle input sketches with significant shading, as it leads to artifacts in the vectorization. Furthermore, our system only targets *presentation* sketches that contain very few extra strokes, such as scaffolds or other auxiliary construction lines. When artists do not draw some of the boundaries of primitives, such as on the back of the chair in (Fig. 22a), our pipeline may produce a suboptimal surface (Fig. 22b). If desired, artists can complete the boundaries (Fig. 22c, a new stroke on the back), improving the final result (Fig. 22d, lower arrow).

Our pipeline is driven by depth prediction and segmentation networks; both can produce noisy outputs (Fig. 3b, 9b). Small patches, abundant in complex shapes, have fewer pixels and thus contain less depth and segmentation information, leading to potential errors down the pipeline and suboptimal results (Fig. 23 (top)). We account for possible segmentation mistakes by fitting all the other primitive types and picking the primitive with the smallest error. For our inputs, it only happened with 2 patches on Fig. 20 (train, car). These standard drawbacks can be addressed by using advanced neural architectures and training procedures or a target-specific dataset.

Finally, our system targets drawings with orthographic projection or weak perspective. To support stronger perspective one may need to retrain the neural networks and modify the projection transformations. We show the effect of perspective in Fig. 23 (bottom).

8.2 Future work

One clear avenue for future work is the prediction of a piecewise smooth hidden surface of the object that connects with the visible surface we output (Fig. 22f, upper arrow). One might use ideas similar to work of Yao et al. [2020] or use multiple viewpoints. Another interesting direction is to combine cues from segmentation and drawn strokes with rudimentary shading because some artists use this to capture fine variation of the depicted surfaces.

Our system can also be extended to incorporate additional user annotations (e.g., correcting primitive types and enforcing orthogonality or alignment), or enable patch-based shape manipulation.

ACKNOWLEDGMENTS

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds de recherche du Québec – Nature et technologies (FRQNT) NOVA Grant No.: 314090.

REFERENCES

P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. 2018. Learning representations and generative models for 3d point clouds. In *International conference on machine learning*. PMLR, 40–49.

M. Berger, A. Tagliasacchi, L. M. Seversky, P. Alliez, G. Guennebaud, J. A. Levine, A. Sharf, and C. T. Silva. 2017. A Survey of Surface Reconstruction from Point Clouds. *Comput. Graph. Forum* 36, 1 (jan 2017), 301–329. https://doi.org/10.1111/cgf.12802

S. Bhattacharjee and P. Chaudhuri. 2020. A survey on sketch based content creation: from the desktop to virtual and augmented reality. 39, 2 (2020), 757–780.

V. Blanz, M. J. Tarr, and H. H. Bülthoff. 1999. What Object Attributes Determine Canonical Views? *Perception* 28, 5 (1999), 575–599. https://doi.org/10.1068/p2897 arXiv:https://doi.org/10.1068/p2897 PMID: 10664755.

A. Bonnici, A. Akman, G. Calleja, K. Camilleri, P. Fehling, A. Ferreira, F. Hermuth, J. Israel, T. Landwehr, J. Liu, N. Padfield, T. Sezgin, and P. Rosin. 2019. Sketch-based interaction and modeling: where do we stand? *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 33 (11 2019), 1–19.

S. Boyd, N. Parikh, E. Chu, B. Peleato, J. Eckstein, et al. 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine learning* 3, 1 (2011), 1–122.

M. L. Bynum, G. A. Hackebeil, W. E. Hart, C. D. Laird, B. L. Nicholson, J. D. Siirola, J.-P. Watson, and D. L. Woodruff. 2021. *Pyomo–optimization modeling in python* (third ed.). Vol. 67. Springer Science & Business Media.

T. Chen, Z. Zhu, A. Shamir, S.-M. Hu, and D. Cohen-Or. 2013. 3-Sweep: Extracting Editable Objects from a Single Photo. *ACM Trans. Graph.* 32, 6, Article 195 (nov 2013), 10 pages. https://doi.org/10.1145/2508363.2508378

Z. Chen, A. Tagliasacchi, and H. Zhang. 2020. BSP-Net: Generating Compact Meshes via Binary Space Partitioning. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).

Z. Chen and H. Zhang. 2019. Learning Implicit Fields for Generative Shape Modeling. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

Z. Chen and H. Zhang. 2021. Neural Marching Cubes. *ACM Trans. Graph.* 40, 6, Article 251 (dec 2021), 15 pages. https://doi.org/10.1145/3478513.3480518

J. J. Cherlin, F. Samavati, M. C. Sousa, and J. A. Jorge. 2005. Sketch-based modeling with few strokes. *Proceedings of the 21st spring conference on Computer graphics - SCCG '05* 1, 212 (2005), 137. https://doi.org/10.1145/1090122.1090145

C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese. 2016. 3D-R2N2: A Unified Approach for Single and Multi-view 3D Object Reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

F. Cole, A. Golovinskiy, A. Limpaecher, H. S. Barros, A. Finkelstein, T. Funkhouser, and S. Rusinkiewicz. 2008. Where Do People Draw Lines?. In *ACM SIGGRAPH 2008 Papers* (Los Angeles, California) *(SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, Article 88, 11 pages. https://doi.org/10.1145/1399504.1360687

J. Delanoy, M. Aubry, P. Isola, A. A. Efros, and A. Bousseau. 2018. 3D Sketching using Multi-View Deep Volumetric Prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 1, 1 (jul 2018), 1–22. https://doi.org/10.1145/3203197

J. Delanoy, D. Coeurjolly, J.-O. Lachaud, and A. Bousseau. 2019. Combining voxel and normal predictions for multi-view 3D sketching. *Computers & Graphics* 82 (2019), 65–72. https://doi.org/10.1016/j.cag.2019.05.024

Z. Deng, Y. Liu, H. Pan, W. Jabi, J. Zhang, and B. Deng. 2022. Sketch2PQ: Freeform Planar Quadrilateral Mesh Design via a Single Sketch. *IEEE Transactions on Visualization and Computer Graphics* PP (2022), 1–1.

M. Desbrun, M. Meyer, P. Schröder, and A. H. Barr. 1999. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. 317–324.

T. Du, J. P. Inala, Y. Pu, A. Spielberg, A. Schulz, D. Rus, A. Solar-Lezama, and W. Matusik. 2018. InverseCSG: Automatic conversion of 3d models to csg trees. *ACM Transactions on Graphics (TOG)* 37, 6 (2018), 1–16.

R. O. Duda and P. E. Hart. 1972. Use of the Hough Transformation to Detect Lines and Curves in Pictures. *Commun. ACM* 15, 1 (jan 1972), 11–15.

D. H. Eberly. 2018. Least Squares Fitting of Data by Linear or Quadratic Structures.

K. Eissen and R. Steur. 2011. *Sketching: The Basics.* Bis Publishers.

H. Fan, H. Su, and L. J. Guibas. 2017. A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 605–613.

M. Gadelha, R. Wang, and S. Maji. 2018. Multiresolution tree networks for 3d point cloud processing. In *Proceedings of the European Conference on Computer Vision (ECCV).* 103–118.

Y. Gingold, T. Igarashi, and D. Zorin. 2009. Structured annotations for 2D-to-3D modeling. *ACM Transactions on Graphics* 28, 5 (2009), 1.

R. Girdhar, D. Fouhey, M. Rodriguez, and A. Gupta. 2016. Learning a Predictable and Generative Vector Representation for Objects. In *ECCV*.

T. Groueix, M. Fisher, V. G. Kim, B. Russell, and M. Aubry. 2018. AtlasNet: A Papier-Mâché Approach to Learning 3D Surface Generation. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).*

Y. Gryaditskaya, F. Hähnlein, C. Liu, A. Sheffer, and A. Bousseau. 2020. Lifting Freehand Concept Sketches into 3D. *ACM Trans. Graph.* 39, 6, Article 167 (nov 2020), 16 pages. https://doi.org/10.1145/3414685.3417851

Y. Gryaditskaya, M. Sypesteyn, J. W. Hoftijzer, S. Pont, F. Durand, and A. Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia)* 38 (11 2019).

B. Guillard, E. Remelli, P. Yvernay, and P. Fua. 2021. Sketch2mesh: Reconstructing and editing 3d shapes from sketches. In *Proceedings of the IEEE/CVF International Conference on Computer Vision.* 13023–13032.

H. Guo. 2023. About how to make patches trimmed in visualization. https://github.com/guohaoxiang/ComplexGen/issues/9#issuecomment-1458402505. [Online; accessed 22-May-2023.

H. Guo, S. Liu, H. Pan, Y. Liu, X. Tong, and B. Guo. 2022. ComplexGen: CAD Reconstruction by B-Rep Chain Complex Generation. *ACM Trans. Graph. (SIGGRAPH)* 41, 4, Article 129 (July 2022), 18 pages. https://doi.org/10.1145/3528223.3530078

Z. Han, B. Ma, Y.-S. Liu, and M. Zwicker. 2020. Reconstructing 3D shapes from multiple sketches using direct shape optimization. *IEEE Transactions on Image Processing* 29 (2020), 8721–8734.

W. E. Hart, J.-P. Watson, and D. L. Woodruff. 2011. Pyomo: modeling and solving mathematical programs in Python. *Mathematical Programming Computation* 3, 3 (2011), 219–260.

Y. He, H. Xie, C. Zhang, X. Yang, and K. Miyata. 2021. Sketch-based normal map generation with geometric sampling. In *International Workshop on Advanced Imaging Technology (IWAIT) 2021*, Wen-Nung Lie, Qian Kemao, Jae-Gon Kim, and Masayuki Nakajima (Eds.). SPIE. https://doi.org/10.1117/12.2590760

S. Hickson, K. Raveendran, A. Fathi, K. Murphy, and I. Essa. 2019. Floors are Flat: Leveraging Semantics for Real-Time Surface Normal Prediction. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW).* IEEE Computer Society, Los Alamitos, CA, USA, 4065–4074. https://doi.ieeecomputersociety.org/10.1109/ICCVW.2019.00501

S. Hu, A. Polette, and J.-P. Pernot. 2022. SMA-Net: Deep learning-based identification and fitting of CAD models from point clouds. *Engineering with Computers* 38 (2022), 1–22. Issue 6.

M. Hudon, M. Grogan, R. Pagés, and A. Smolić. 2018. Deep Normal Estimation for Automatic Shading of Hand-Drawn Characters. In *European Conference on Computer Vision.* Springer, 246–262.

M. Hudon, S. Lutz, R. Pagés, and A. Smolic. 2019. Augmenting Hand-Drawn Art with Global Illumination Effects through Surface Inflation. In *Proceedings of the 16th ACM SIGGRAPH European Conference on Visual Media Production* (London, United Kingdom) *(CVMP '19).* Association for Computing Machinery, New York, NY, USA, Article 4, 9 pages. https://doi.org/10.1145/3359998.3369400

P. Iakubovskii. 2019. Segmentation Models Pytorch. https://github.com/qubvel/segmentation_models.pytorch.

T. Igarashi, S. Matsuoka, and H. Tanaka. 1999. Teddy: A Sketching Interface for 3D Freeform Design. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '99).* ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 409–416. https://doi.org/10.1145/311535.311602

P. K. Jayaraman, J. G. Lambourne, N. Desai, K. Willis, A. Sanghi, and N. J. Morris. 2022. SolidGen: An Autoregressive Model for Direct B-rep Synthesis. *Transactions on Machine Learning Research* (2022).

H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon. 2020. Differentiable Rendering: A Survey. *CoRR* abs/2006.12057 (2020). arXiv:2006.12057 https://arxiv.org/abs/2006.12057

S. Koch, A. Matveev, Z. Jiang, F. Williams, A. Artemov, E. Burnaev, M. Alexa, D. Zorin, and D. Panozzo. 2019. ABC: A Big CAD Model Dataset For Geometric Deep Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

J. G. Lambourne, K. Willis, P. K. Jayaraman, L. Zhang, A. Sanghi, and K. R. Malekshan. 2022. Reconstructing editable prismatic CAD from rounded voxel models. In *SIGGRAPH Asia 2022 Conference Papers.* 1–9.

C. Li, H. Pan, A. Bousseau, and N. J. Mitra. 2020. Sketch2CAD: Sequential CAD Modeling by Sketching in Context. *ACM Trans. Graph. (Proceedings of SIGGRAPH Asia 2020)* 39, 6 (2020), 164:1–164:14. https://doi.org/10.1145/3414685.3417807

C. Li, H. Pan, A. Bousseau, and N. J. Mitra. 2022a. Free2CAD: Parsing Freehand Drawings into CAD Commands. *ACM Trans. Graph.* 41, 4, Article 93 (jul 2022), 16 pages. https://doi.org/10.1145/3528223.3530133

C. Li, H. Pan, Y. Liu, A. Sheffer, and W. Wang. 2018. Robust Flow-Guided Neural Prediction for Sketch-Based Freeform Surface Modeling. *ACM Trans. Graph. (SIGGRAPH ASIA)* 37, 6 (2018), 238:1–238:12. https://doi.org/10.1145/3272127.3275051

L. Li, M. Sung, A. Dubrovina, L. Yi, and L. J. Guibas. 2019. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.* 2652–2660.

X. Li, Y. Wang, and Z. Sha. 2022b. Deep Learning of Cross-Modal Tasks for Conceptual Design of Engineered Products: A Review. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Vol. 86267. American Society of Mechanical Engineers, V006T06A016.

Z. Li. 2018. LineFiller. https://github.com/hepesu/LineFiller. [Online; accessed 20-May-2023.

Y. Liao, S. Donné, and A. Geiger. 2018. Deep Marching Cubes: Learning Explicit Surface Representations. In *Conference on Computer Vision and Pattern Recognition (CVPR).*

T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. 2017a. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 2117–2125.

T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. 2017b. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision.* 2980–2988.

Z. Lun, M. Gadelha, E. Kalogerakis, S. Maji, and R. Wang. 2018. 3D Shape Reconstruction from Sketches via Multiview Convolutional Networks.

B. Manda, S. Dhayarkar, S. Mitheran, V. Viekash, and R. Muthuganapathy. 2021. 'CADSketchNet' - An Annotated Sketch dataset for 3D CAD Model Retrieval with Deep Neural Networks. *Computers & Graphics* 99 (2021), 100–113. https://doi.org/10.1016/j.cag.2021.07.001

L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. 2019. Occupancy Networks: Learning 3D Reconstruction in Function Space. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition (CVPR).*

J. Pan, X. Han, W. Chen, J. Tang, and K. Jia. 2019. Deep Mesh Reconstruction from Single RGB Images via Topology Modification Networks. In *Proceedings of the IEEE International Conference on Computer Vision.* 9964–9973.

I. Puhachov, W. Neveu, E. Chien, and M. Bessmeltsev. 2021. Keypoint-Driven Line Drawing Vectorization via PolyVector Flow. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 40, 6 (Dec. 2021). https://doi.org/10.1145/3478513.3480529

E. Remelli, A. Lukoianov, S. Richter, B. Guillard, T. Bagautdinov, P. Baque, and P. Fua. 2020. MeshSDF: Differentiable Iso-Surface Extraction. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin (Eds.), Vol. 33. Curran Associates, Inc., 22468–22478. https://proceedings.neurips.cc/paper/2020/file/fe40fb944ee700392ed51bfe84dd4e3d-Paper.pdf

C. Romanengo, A. Raffo, S. Biasotti, B. Falcidieno, V. Fotis, I. Romanelis, E. Psatha, K. Moustakas, I. Sipiran, Q.-T. Nguyen, C.-B. Chu, K.-N. Nguyen-Ngoc, D.-K. Vo, T.-A. To, N.-T. Nguyen, N.-Q. Le-Pham, H.-D. Nguyen, M.-T. Tran, Y. Qie, and N. Anwer. 2022. SHREC 2022: Fitting and recognition of simple geometric primitives on point clouds. *Computers & Graphics* 107 (2022), 32–49. https://www.sciencedirect.com/science/article/pii/S0097849322001224

O. Ronneberger, P. Fischer, and T. Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18.* Springer, 234–241.

M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. 2018. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition.* 4510–4520.

R. Schmidt, A. Khan, G. Kurtenbach, and K. Singh. 2009. On expert performance in 3D curve-drawing tasks. *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling - SBIM '09* 1 (2009), 133.

R. Schnabel, R. Wahl, and R. Klein. 2007. Efficient RANSAC for point-cloud shape detection. In *Computer graphics forum*, Vol. 26. Wiley Online Library, 214–226.

C. Shakarji. 1998. Least-Squares Fitting Algorithms of the NIST Algorithm Testing System. 103 (1998-12-01 1998). https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=821955

G. Sharma, D. Liu, S. Maji, E. Kalogerakis, S. Chaudhuri, and R. Měch. 2020. Parsenet: A parametric surface fitting network for 3d point clouds. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16.* Springer, 261–276.

J. R. Shewchuk. 1996. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, Ming C. Lin and Dinesh Manocha (Eds.). Lecture Notes in Computer Science, Vol. 1148. Springer-Verlag, 203–222. From the First ACM Workshop on Applied Computational Geometry.

D. Smirnov, M. Bessmeltsev, and J. Solomon. 2021. Learning Manifold Patch-Based Representations of Man-Made Shapes. In *International Conference on Learning Representations (ICLR)*.

O. Sorkine and M. Alexa. 2007. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, Vol. 4. 109–116.

O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. 2004. Laplacian Surface Editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (Nice, France) *(SGP '04)*. Association for Computing Machinery, New York, NY, USA, 175–184. https://doi.org/10.1145/1057432.1057456

W. Su, D. Du, X. Yang, S. Zhou, and H. Fu. 2018. Interactive Sketch-Based Normal Map Generation with Deep Neural Networks. *Proc. ACM Comput. Graph. Interact. Tech.* 1, 1, Article 22 (jul 2018), 17 pages. https://doi.org/10.1145/3203186

M. Tan and Q. Le. 2019. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*. PMLR, 6105–6114.

M. Tatarchenko, A. Dosovitskiy, and T. Brox. 2016. Multi-view 3d models from single images with a convolutional network. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VII 14*. Springer, 322–337.

M. A. Uy, Y.-Y. Chang, M. Sung, P. Goel, J. G. Lambourne, T. Birdal, and L. J. Guibas. 2022. Point2cyl: Reverse engineering 3d objects from point clouds to extrusion cylinders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 11850–11860.

A. Wächter and L. T. Biegler. 2006. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming* 106 (2006), 25–57.

J. Wang, J. Lin, Q. Yu, R. Liu, Y. Chen, and S. X. Yu. 2022a. 3d shape reconstruction from free-hand sketches. In *European Conference on Computer Vision*. Springer, 184–202.

J. Wang, H. Zhu, H. Guo, A. Al Mamun, P. Vadakkepat, and T. H. Lee. 2022b. CAM/CAD Point Cloud Part Segmentation via Few-Shot Learning. In *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*. IEEE, 359–365.

N. Wang, Y. Zhang, Z. Li, Y. Fu, W. Liu, and Y.-G. Jiang. 2018. Pixel2Mesh: Generating 3D Mesh Models from Single RGB Images. In *ECCV*.

Z. Wang, S. Qiu, N. Feng, H. Rushmeier, L. McMillan, and J. Dorsey. 2021. Tracing versus freehand for evaluating computer-generated drawings. *ACM Transactions on Graphics (TOG)* 40, 4 (2021), 1–12.

J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum. 2016. Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling. In *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (Eds.), Vol. 29. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2016/file/44f683a84163b3523afe57c2e008bc8c-Paper.pdf

J. Wu, C. Zhang, X. Zhang, Z. Zhang, W. T. Freeman, and J. B. Tenenbaum. 2018. Learning shape priors for single-view 3d completion and reconstruction. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 646–662.

R. Wu, C. Xiao, and C. Zheng. 2021. DeepCAD: A Deep Generative Network for Computer-Aided Design Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 6772–6782.

B. Xu, W. Chang, A. Sheffer, A. Bousseau, J. McCrae, and K. Singh. 2014. True2Form: 3D Curve Networks from 2D Sketches via Selective Regularization. *Transactions on Graphics (Proc. SIGGRAPH 2014)* 33, 4 (2014). https://doi.org/2601097.2601128

S. Yan, Z. Yang, C. Ma, H. Huang, E. Vouga, and Q. Huang. 2021. Hpnet: Deep primitive segmentation using hybrid representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2753–2762.

Y. Yang, C. Feng, Y. Shen, and D. Tian. 2018. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 206–215.

Y. Yao, N. Schertler, E. Rosales, H. Rhodin, L. Sigal, and A. Sheffer. 2020. Front2Back: Single View 3D Shape Reconstruction via Front to Back Prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

F. Yu, Z. Chen, M. Li, A. Sanghi, H. Shayani, A. Mahdavi-Amiri, and H. Zhang. 2022. CAPRI-Net: Learning Compact CAD Shapes With Adaptive Primitive Assembly. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 11768–11778.

Z. Yue, G. Yulia, Z. Honggang, and S. Yi-Zhe. 2020. Deep Sketch-Based Modeling: Tips and Tricks. In *Proceedings of International Conference on 3D Vision (3DV)*.

S.-H. Zhang, T. Chen, Y.-F. Zhang, S.-M. Hu, and R. R. Martin. 2009. Vectorizing Cartoon Animations. *IEEE Transactions on Visualization and Computer Graphics* 15, 4 (2009), 618–629. https://doi.org/10.1109/TVCG.2009.9

Q. Zheng, Z. Li, and A. Bargteil. 2020. Learning to Shadow Hand-Drawn Sketches. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Y. Zhong, Y. Gryaditskaya, H. Zhang, and Y.-Z. Song. 2022. A study of deep single sketch-based modeling: View/style invariance, sparsity and latent space disentanglement. *Computers & Graphics* 106 (2022), 237–247. https://doi.org/10.1016/j.cag.2022.06.005

A DATASET GENERATION

We render a subset of shapes from ABC dataset [Koch et al. 2019]. The dataset contains 3D shapes along with shape decomposition into primitives and labeled curves connecting those primitives.

We discard the shapes with multiple connected components. To help balance the primitive distribution in the dataset, we also remove the shapes consisting solely of planes.

Rendering. Given a shape and corresponding geometric information from ABC dataset, we render the training image via Blender Freestyle, and generate the corresponding depth and segmentation images. We normalize each object preserving its ratio to fit into $(-1, 1)^3$ box and translate the object's barycenter to the origin. Following [Zhong et al. 2022] we sample camera locations uniformly on a sphere of radius 5; each shape is rendered from four different viewpoints. We rescale the depthmap values from the range of $[3.5, 6.5]$ to $[0, 1]$. We perform the same augmentation procedure for the rendered images as in Puhachov et al. [2021] to make our networks robust to the different stroke styles and widths.

We used 3,900 shapes for training, 150 shapes for validation, and 50 for testing. In total, our dataset contains 16,400 labeled image triplets.

B ARCHITECTURE AND TRAINING

We use Segmentation Models PyTorch framework for image segmentation models [Iakubovskii 2019] and PyTorch Lightning to build and train the neural networks. The depth estimation network uses MobileNetV2 encoder [Sandler et al. 2018] with U-Net architecture [Ronneberger et al. 2015]. We use masked RMSE loss, calculated only for foreground pixels. Segmentation network uses EfficientNet-B0 encoder [Tan and Le 2019] with feature pyramid architecture [Lin et al. 2017a]. We use focal loss [Lin et al. 2017b] for balancing. Both networks were trained using Adam optimizer. Depth estimation network was trained for 60 epochs, the best checkpoint scoring 0.002 masked RMSE on our test set. Segmentation network was trained 200 epochs. It terminated with 99.60% IoU score on our test set.

C FITTING GEOMETRIC PRIMITIVES

The energies we minimize are similar to the ones used by Eberly [2018] and Romanengo et al. [2022]. Below, we define $X_i \in \mathbb{R}^3, i = 1, \ldots, N$ as the set of points that we are fitting to each primitive.

Plane. We minimize the sum of squared distances to the plane, parameterized by its normal $n \in \mathbb{R}^3$, $\|n\| = 1$ and a scalar $d \in \mathbb{R}$:

$$E_{\text{plane}}(n, d) = \sum_{i=0}^{N} \|n \cdot X_i + d\|^2. \tag{11}$$

To fit a single plane, we use a singular value decomposition.

Sphere. We minimize the sum of squared distances between the points and the sphere, which is parameterized by its center $C \in \mathbb{R}^3$ and radius $r > 0$:

$$E_{\text{sphere}}(C, r) = \sum_{i=1}^{N} (\|X_i - C\|^2 - r^2)^2.$$

To fit a single sphere, we solve an over-determined linear system via QR decomposition.

Cylinder. A cylinder is defined by its axis, parameterized by a point $C \in \mathbb{R}^3$ and a direction $W \in \mathbb{R}^3$, $\|W\| = 1$, and its radius $r > 0$. We then minimize

$$E_{\text{cylinder}}(C, W, r) = \sum_{i=1}^{N} \left(d_{\text{axis}}(X_i) - r\right)^2, \tag{12}$$

where the unsigned distance to the axis is computed as

$$d^2_{\text{axis}}(X_i) = \left|(X_i - C)^\top (I - WW^\top)(X_i - C)\right|. \tag{13}$$

We use an interior point solver Ipopt 3.11.1 using their default parameters.

Cone. Cone is defined by its apex $V \in \mathbb{R}^3$, axis $U \in \mathbb{R}^3$ and angle $\theta \in (0, \frac{\pi}{2})$. We use the standard least-squares fitting, as in [Shakarji 1998]. Please refer to their paper for details.

Other primitives. For all the other classes of primitives, we directly use the predicted 3D pixel depths as an estimate of their local geometry.