Manufacturing-Aware Reconstruction

James Noeckel

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2024

Reading Commitee:

Brian Curless, Chair

Adriana Schulz, Chair

Ranjay Krishna

Program Authorized to Offer Degree:
Computer Science & Engineering

University of Washington

Abstract

Manufacturing-Aware Reconstruction

James Noeckel

Chairs of the Supervisory Committee:
Brian Curless
Adriana Schulz
Computer Science & Engineering

Most of the objects that surround us begin their life as designs. The machines and people involved in the manufacturing process must communicate with a high degree of precision, demanding information-rich designs capable of specifying geometry down to individual mathematically-defined surfaces, as well as other relevant details such as functionality, materials, and assembly instructions. Consequently, designs are a concise representation for the built objects in our environment. In this work, we explore various techniques to reconstruct man-made objects from measured data using geometric priors inherent in these design representations to enhance the accuracy and robustness of our methods. Some key questions we address are: Can we reconstruct designs when our observations of the world are partial? And can we achieve this without significantly sacrificing generality? We will explore methods for using prior knowledge about the manufacturing domain to extract complete carpentry designs from partial visual information. We will see that knowledge of the manufacturing process, even in the absence of the full procedure, allows us to more effectively leverage visual data for reconstruction. We will also present work on reconstructing partially-observed objects in terms of a broader class of geometries associated with

computer-aided design (CAD) representations, achieving high precision reconstructions of objects in the wild across many different fabrication domains. Looking beyond geometry, we also present work on augmenting purely geometric designs with functionality by inferring the motions of parts using deep learning. In summation, we demonstrate new techniques to reconstruct various types of designs from partial observations, exploring geometry- and functionality-focused methods, by leveraging manufacturing priors.

Contents

Acknowledgments

I am grateful to my great friend, mentor, and collaborator Benjamin Jones, without whose encouragement and guidance I would have been lost.

Thanks to my advisors Adriana Schulz and Brian Curless for supporting me through thick and thin.

Special thanks to Yuxuan Mei, who took time out of her busy schedule to 3D print baubles for testing my reconstruction algorithms, without which the primary work of this thesis, View2CAD, would not have been possible.

Chapter 1

Introduction

In navigating the physical world, we necessarily do so with partial information. Our senses afford us a limited, albeit varied view of our surroundings—the appearance of illuminated surfaces, the sounds of materials in contact, the feedback from forces applied to objects, and more—but it is through the collection of these varied stimuli throughout our lives that we build an understanding of the world that facilitates successful action. In many settings, it is necessary to formalize this understanding. In architectural floor plans, doors are represented by not just their dimensions, but the area they sweep when they open and close, conveying aspects of their dynamics and functionality. When the properties of an object are codified in this way, the object becomes imbued with exact geometric descriptions, physical quantities, and concrete relations between parts, elevating our vague, practical understanding of the object to a **design**. Working with designs lets us reason about objects at a much more precise level—an architect can be sure that every door has an unobstructed opening path simply by looking at the swept areas in a blueprint.

Most of the objects that surround us began their life as designs. Even the most innocuous object likely has associated with it a mathematical description of all of its surfaces, fasteners, or other applicable properties. This is because man-made objects are manufactured using a variety of

processes demanding high precision, whether that means specifying the shape of a mold, the tool path of a CNC machine, or the alignment of fasteners and screw holes when assembling the parts together. Due to the complexity of these processes, it is essential for the people and machines involved to communicate using a precise design language. This language may vary depending on the domain, but designed objects nevertheless follow certain conventions as a result of this shared representation, such as a tendency to exhibit right angles or symmetries. So when we encounter a man-made object and analyze its shape, motion, and other properties, we are really **reconstructing** a design.

A design is useful not only for specifying how to manufacture an object, but as a concise semantic description of the object pertaining to its function. Once we recognize when an object is man-made, we may reason with the assumption that such a design exists, thereby imposing certain constraints on the characteristics of the object based on the conventions of the corresponding design language, which are in turn informed by some underlying physical manufacturing process. For instance, the existence of hinges as a design element common to virtually all doors gives us a way to reason about the axis about which a door will open, where we can stand to stay out of its way, and where we should push to maximize torque. Many downstream applications involving an object therefore benefit from recovering this design.

A natural question to ask, then, is: Given an object, how can we reconstruct its design? A human could, in many cases, take precise measurements, determine which fabrication process underlies the construction, and transcribe the object as a design corresponding to that process. However, this is quite laborious, and perhaps not something that could reasonably be done on the fly, or on a large scale. In which case, we might instead ask: Could computer algorithms aid us in this task? The term *reconstruction* certainly calls forth a plethora of algorithms for procuring 3D geometry from observations. However, the 3D geometry produced by most of these reconstruction methods is divorced from any semantically meaningful understanding of

the target object: clouds of colorful points, grids of tiny cubes, or floating, semi-transparent blobs. Though these representations mimic the shape and appearance of an object, the constituent elements lend little insight into the design beyond that afforded by the original observations. Furthermore, the geometry itself, despite typically comprising thousands or millions of discrete elements, is incomplete and lacking in precision. To see why, consider the class of methods referred to as *photogrammetry* techniques. Generally speaking, these methods use photometric cues to triangulate 3D points from matched features in pairs of photographs; the final geometry is the collection of all 3D points gleaned from all sufficiently well-matched pairs of images in the input, or sometimes a surface extracted from that point set using standard techniques. The geometry is **incomplete** because the points comprising it are only well-defined in regions that have been directly photographed, leaving holes in the model elsewhere. What's more, it is **imprecise** because none of the points adhere to any underlying physical constraints, so that every point of the surface is free to vary, limited only by the accuracy of measurements and the numerical fidelity of the geometric algorithms. Consequently, extracting a semantically valid design from such a reconstruction algorithm would be just as laborious a transcription process as before. So what good are these representations? Their main advantage is that they are quite *general*—because these methods make no assumptions about the target domain, they are free to model the geometry of objects captured in the wild without any constraints, using general-purpose algorithms. The price we pay is that they do not help us reason about objects beyond our partial observations, which we have determined is a crucial feature of design representations.

Some key questions that will be addressed in this thesis are: Can we reconstruct designs when our observations of the world are partial? And can we achieve this without significantly sacrificing *generality*? In Chapter 2, we will introduce several expressive geometric representations that serve as a foundation for the reconstruction approaches discussed in this dissertation. In Chapter 3, we will explore the first of these methods which uses prior knowledge about a particular

manufacturing domain to extract complete carpentry designs from partial visual information. We will see that knowledge of the manufacturing process, even in the absence of the full procedure, allows us to more effectively leverage visual data for reconstruction. In Chapter 4, We will also present work on reconstructing partially-observed objects in terms of a broader class of geometries commonly used in man-made designs, achieving high precision in reconstructions across many different fabrication domains.

Recalling that designs convey information beyond the geometry of objects, we might also ask: Can we reconstruct designs which capture other aspects of our physical world, such as motion and functionality? In Chapter 5, we address this question in our preliminary work on augmenting purely geometric (and often approximate) designs with functionality by inferring the motions of parts. We discuss further future directions exploring other design spaces and modalities in Chapter 6. In summation, the works comprising this thesis demonstrate new techniques to reconstruct designs from partial observations, exploring **geometry**- and **functionality**-focused methods, each leveraging manufacturing priors in different ways to make their respective problems tractable.

Chapter 2

Background

2.1 Design Representations and CAD

Most man-made objects possess some form of regular geometric structure: They may be symmetric, or composed of repeating elements, or be described by parametric curves and surfaces with a clean mathematical representation. The unstructured point clouds discussed in Chapter 1 can only go so far in representing these forms—they may give an impression of the general shape or dimensions, but they fall short in any applications involving designs. If we wanted to 3D print our geometry, for example, we would need a closed surface to differentiate between the inside and outside of the object. While there exist algorithms to attempt this conversion from point clouds, it would be useful to familiarize ourselves with the representations which *can* specify closed surfaces, continuous geometries, and other structures pertinent to design tasks. In software, there are various geometric representations more suited to design applications, referred to as **computer-aided design (CAD)** representations. These enable the concise description of objects through the use of mathematically-defined primitive structures along with topological relations which together define exact surfaces and volumes. There are a variety of CAD representations

which range from high-level to low-level. High-level representations typically use programmatic abstractions to define geometry. An example is **constructive solid geometry (CSG)**, which uses geometric primitives (spheres, cones, cylinders, cuboids, etc.) to carve out space in order to form solid shapes, where the final geometry is the result of *evaluating* a tree of boolean operations on successive combinations of the geometric primitives. Conversely, **meshes** would be considered a low-level representation, as they directly represent the final geometry by approximating it using faces comprised of triangles or N-gons. Meshes may utilize various data structures to encode the topological relationships of the faces, allowing meshes to define watertight surfaces by ensuring that all faces are joined at the edges.

Meshes have the downside of being unable to represent curved geometries exactly. Meanwhile, the abstraction of CSG trees makes it difficult to intuitively edit the underlying geometry. Some representations—namely, vector graphics in 2D and B-Reps in 3D—possess some of the benefits of both low-level and high-level representations, in that they allow for direct editing as well as representing curved structures exactly using geometric primitives. In the works I will be presenting in this thesis, these representations feature prominently, and are detailed below.

2.1.1 Vector Graphics

Planar paths or sketches comprised of curves and lines are the foundation of most design representations. The path of an engraving, the profile shape of a chair leg, and color boundaries on an advertising billboard can all be defined in terms of these paths. In the engineering world, even the most complex three-dimensional shapes can be reduced to sequences of operations guided by planar sketches. Defining designs in terms of 2D sketches carries major advantages. The visual media we use to communicate designs (particularly before computers) occupy flat planes, such as sheets of paper; physical rulers and guides used in drawing only work on flat surfaces. In

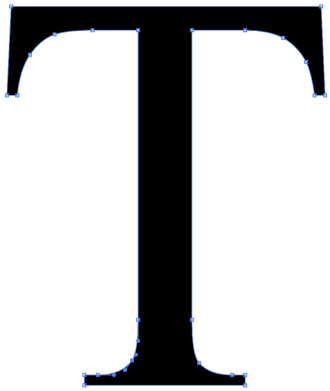

Figure 2.1: An example of a vector graphics representation of a font character, with spline endpoints shown as blue dots.

addition, our visual system only allows us to interpret 2D views of an object at any one time, so 2D technical drawings may distill the important aspects of a design into a form that humans can digest at a glance. Consequently, many tools have been developed to enable designers to specify the 2D paths underlying these designs with high precision. Early drafters used deformable rulers to draw precise curves: These so-called "spline-rulers" came equipped with weights and clamps along their length so that a continuous curve could be defined completely in terms of a few discrete control handles.

In digital software, collections of paths consisting of lines and curves defined by control handles have the name **vector graphics** (for an example, see Figure 2.1). Digital vector graphics have their origin in early efforts to create device-independent abstractions for graphics imaging [WW82], where the target display devices might include both raster or vector displays, alongside other devices like pen plotters. The basic building blocks of these vector graphics representations are **splines**, which mathematically model curves (or lines) using functions of the form

$$\mathbf{P} : [0, 1] \to \mathbb{R}^2 \tag{2.1}$$

where all points along the curve can be sampled by evaluating $\mathbf{P}(t)$, with $\mathbf{P}(0)$ and $\mathbf{P}(1)$ being the endpoints of the curve. Certain classes of spline are especially popular due to their useful properties; one such example is the Bézier spline, whose spline function $\mathbf{P}(t)$ is a *cubic polynomial* which can be written as the following matrix expression:

$$\mathbf{P}(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}$$

where $\mathbf{P}_0$, $\mathbf{P}_1$, $\mathbf{P}_2$, $\mathbf{P}_3$ are four 2D **control points** which uniquely determine the shape of the curve. This curve has the property that all points $\mathbf{P}(t)$ lie wihin the *convex hull* of the four control points, and that the line through the two pairs of points $\mathbf{P}_0$, $\mathbf{P}_1$ and $\mathbf{P}_2$, $\mathbf{P}_3$ define the *tangent direction* of the curve at the two endpoints, allowing for precise artist control of the slope and sharpness of their vector graphics at the junctions between spline segments.

Various other common 2D primitives may comprise a vector graphic design, such as rectangles, circles, arcs, and regular polygons. What unites these as **vector** representations is that the shapes are defined by a relatively small collection of parameters, such as the control point positions, radii, or sidelengths of constituent elements, as opposed to **raster** representations like images, which must specify the color of millions of pixels individually. In engineering contexts, it is also useful to define **constraints** between elements of a design. For example, a set of screw holes should have the same radius, and each hole should be the same distance from the edge of a board. Since the 1980s, CAD systems come equipped with constraint solvers capable of satisfying sets of user-defined constraints, so that the exact values of the above parameters may be chosen automatically to obey the semantic requirements of the design set forth by the user. Together, these vector representations and constraint solver systems serve to reduce the number

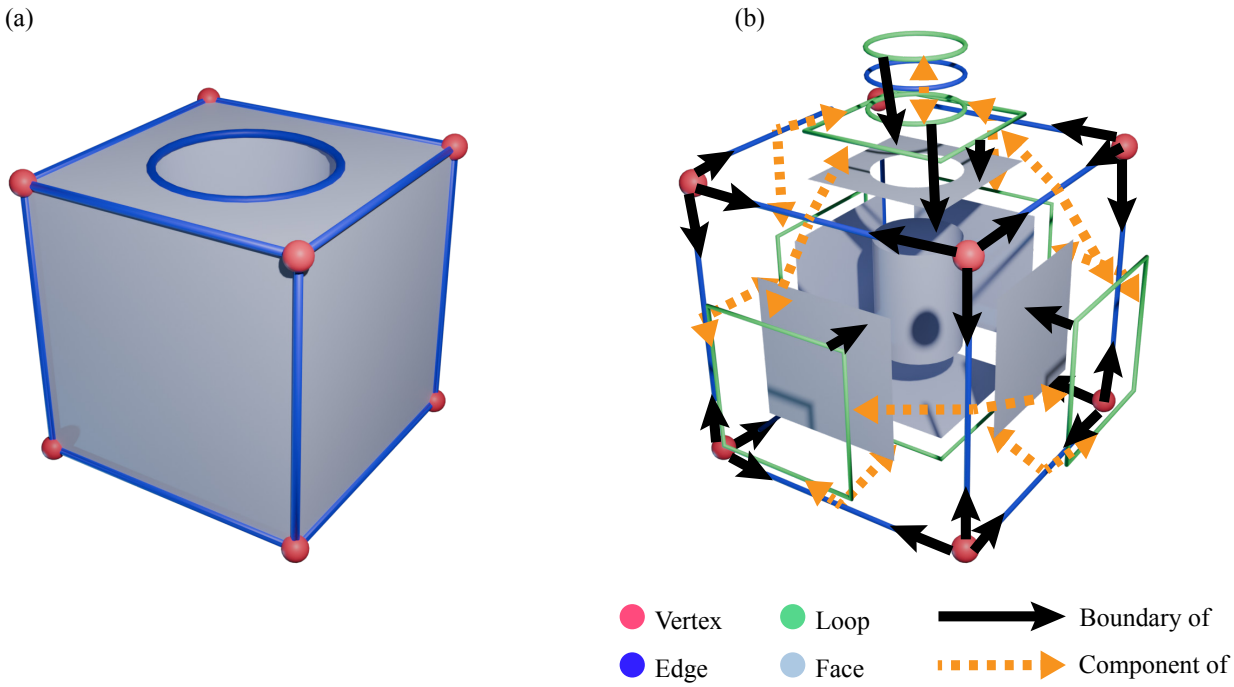

Figure 2.2: B-Rep geometry (a) and the corresponding graph structure (b). There are two different types of edges in a B-Rep graph: **boundary of** and **component of** edges. Vertices are **boundaries of** edges, edges are **components of** (multiple) loops, loops are **boundaries of** faces. Boundary relations are necessary to evaluate the B-Rep geometry, since underlying curves and surfaces may be defined using parametric equations without any intrinsic bounds on their domain.

of parameters, needed to characterize a design. This can also be seen as reducing the number of possible *choices* one might make during the design process. In the context of reconstruction, this can be a major boon, as it means fewer possible designs to search through. In Chapter 3, we will present an approach for reconstructing the individual parts of a carpentry design, each with a vector graphics curve defining its shape.

2.1.2 Boundary Representations

Analogous to vector graphics in 2D which use collections of parametric curves to represent planar designs, Boundary Representations (B-Reps) use collections of parametric curves and surfaces

to represent 3D geometry. Their name comes from the additional structural information they encode, in the form of *boundary relations* between the various elements. Specifically, B-Reps are composed of **vertices** (0D), **edges** (1D), and **faces** (2D), which all form nodes in a graph structure with boundary relations as the connections: vertices bound edges, and edges comprise loops, which bound faces. This structure is illustrated in Figure 2.2. The geometries of the various elements are represented using points for vertices, and parametric curves and surfaces for edges and faces, respectively. The reason we need this graph structure is that the parametric equations defining curve and surface geometry may not have any intrinsic bounds, as shown in Figure 2.3. While the B-Rep face shown in the figure is bounded, its geometry is defined by an unbounded cylinder—the only way to determine which parts of the cylinder are actually part of the model is to use the neighboring loop in the B-Rep graph as a boundary. This corresponds with finding a subset $U \subset \mathbb{R}^2$ in the domain of the parametric surface $f : \mathbb{R}^2 \to \mathbb{R}^3$ for which $f(U)$ is within those bounds. Using 3D space curves to bound parametric surfaces leads to some challenges: it is necessary for the parameters of the curve and surface geometries to agree, such that the curves actually lie within the surfaces they bound. Many CAD systems provide numerical tolerances for considering curves and surfaces coincident in space for the purpose of evaluating boundaries. In Chapter 4, we will explore the problem of reconstructing B-Reps from measurements, where we will need to handle these geometric constraints in detail.

2.2 3D Reconstruction

Since we are interested in reconstructing designs largely in order to extract accurate 3D geometry, we provide here an overview of the existing work in 3D reconstruction. We will begin with a discussion of dense photogrammetry and multi-view stereo approaches, followed by a review of related approaches for reconstructing CAD-specific representations, retrieval-based approaches,

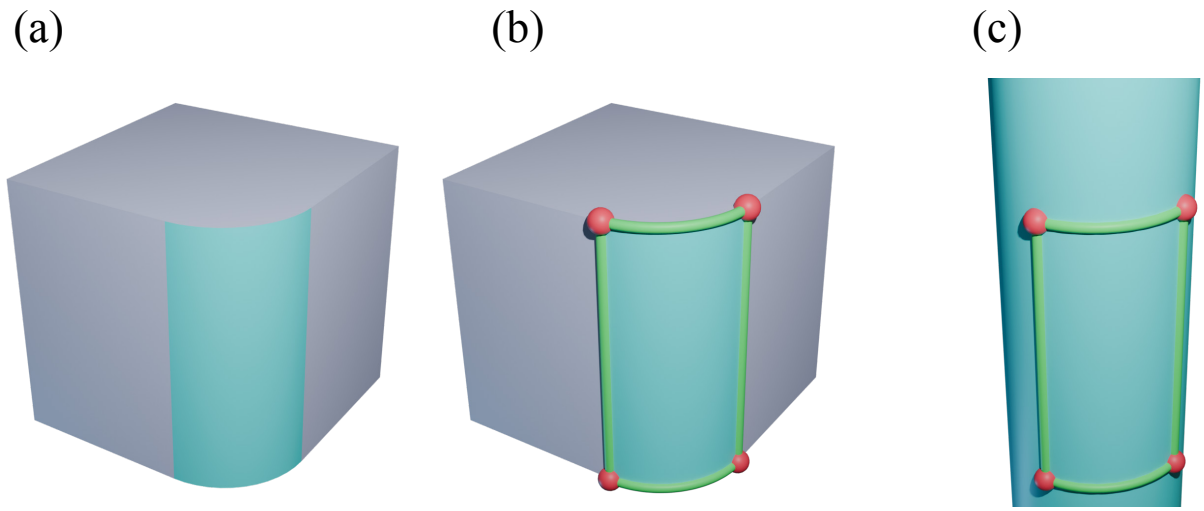

Figure 2.3: Representing B-Rep faces using parametric surface primitives. The selected B-Rep face (a) is bounded by a loop (b), which defines the domain on which to evaluate the underlying parametric surface geometry (c).

and parametric primitive fitting.

2.2.1 Multi-view Stereo

Multi-view stereo (MVS) systems have been an active area of research for several decades, and are widely used today to obtain 3D models from calibrated images. The basic goal of multi-view stereo is to compute a dense correspondence between multiple calibrated images, resulting in a dense 3D reconstruction. Some examples of popular systems used for this purpose are ColMap [SF16, SZPF16], PMVS [FP10], Meshroom, and RealityCapture (many of these software packages include other common functionality useful for photogrammetry, such as algorithms for camera calibration and geometry post-processing such as mesh reconstruction). MVS algorithms can be roughly divided into two categories: Depth map based and volume based. Volume-based methods such as space-carving [KS99, SD99] have the advantage of handling large camera baselines, but make extra assumptions such as that objects are separated from the background via masking, and require

the geometry to be stored in a discrete voxel grid, limiting their resolution; consequently, depth-map based algorithms tend to perform better. These methods function by computing pixel-wise matching scores with neighboring views to produce disparity/depth maps which may be fused into a final 3D shape; scores are typically based on patch-based similarity metrics such as normalized cross correlation (NCC) or sum of squared differences (SSD). There are many algorithms for obtaining these matches. One popular approach is PatchMatch [BSFG09], which matches blocks of an image and propagates matches to neighboring pixels. Another approach is plane-sweep stereo, which constructs a plane-sweep volume centered around the source image, with each cell representing a pixel position and disparity and containing the corresponding warped pixel color of the neighboring views. From this, a so-called cost volume can be constructed, where the lowest cost cells determine the final dense disparity map. Multi-view stereo systems are highly diverse; they differ in the scene representation they use, how they measure photo-consistency and handle visibility/occlusion from different viewpoints, and what shape priors, if any, they may employ [SCD+06]. Conventional MVS systems have been made to scale to large-scale scenes using over 100K images [AFS+11].

Nevertheless, shadows, specular reflections, transparent objects, scattering media, clutter, occlusion, poor texturing, and other factors will still cause these systems to fail much of the time. More recent deep learning approaches have sought to alleviate some of these factors by using learned priors about scenes to fill in the gaps—for example, assigning the correct disparity to textureless regions where patch-based matching would fail. Among the first deep-learning approaches to MVS is GC-Net [KMD+17]. Rather than using patch-based similarity scores, deep convolutional neural networks are used to produce per-pixel features to use in matching. After constructing cost volumes for each view, the volume is fed into an additional 3D convolutional network, producing the final disparity predictons for each view. The model is trained using LIDAR data from the KITTI dataset to provide the ground truth disparity. Further deep learning

approaches have added the ability to process arbitrary numbers of images [HMK+18, YLL+18] through the use of inter-volume feature aggregation via max-pooling, or by aggregating the N feature volumes into a single cost volume. They demonstrate better performance than classical methods such as ColMap for tasks such as distinguishing the disparity in flat, textureless regions and sky—the benefit of learned scene understanding that data-driven approaches enable.

The general approach of dense reconstruction from images, especially with the help of learned priors, is a promising direction for obtaining high-quality geometry from images. However, as a price for their generality, these methods suffer from two main limitations:

1. They are hampered by a lack of precision. In order to represent arbitrary shapes, many adopt dense geometric representations such as point clouds and voxel grids. None of these representations explicitly capture the details that characterize fabricated objects, such as the smooth parametric surfaces that comprise a shape.

2. They rely on strong image evidence of all reconstructed geometry—the presence of occlusions or clutter, improper lighting, insufficient surface texture, and other less predictable deficiencies in the input lead to errors in reconstruction. In the absence of strong priors about the geometry of objects, such errors are simply the result of the under-specified and inherently ill-posed nature of the reconstruction problem.

2.2.2 Reconstruction of CAD representations from images

Aside from general-purpose multi-view reconstruction algorithms, there also exist various specialized reconstruction techniques where strong assumptions can be made about the fabrication process or the type of geometry involved. [XLX+16] infers a parts-based mechanical 3D model comprised of cuboids and generalized cylinders from multi-view images with the help of user-specified sketch strokes indicating the cylinder profile and extrusion distance, and hints for how

part geometries should snap to their surroundings. [GCLZ16] provides a guided 3D modeling tool for producing a moving furniture model from a single image. [CZS+13] recovers interactive manipulable 3D shapes from a single photograph, guided by user-supplied sketches of generalized cylinders and other primitives. Some drawbacks of these methods are that they rely heavily on user input, and are thus not fully automatic, and they make strong assumptions about the geometry and do not generalize beyond the target domain.

2.2.3 Single-view Reconstruction and 3D shape generation

In contrast to multi-view stereo, which uses correspondences within a collection of images to reconstruct geometry, single-view reconstruction seeks to infer plausible 3D geometry from a single image. This is a far more ill-posed problem, necessitating the use of constraints or priors to make the problem tractable. These come at the potential cost of making the results generalize poorly—early works in the field sought to use strong assumptions about geometry, lighting and reflectance models, seeking exact, analytic solutions for the observed surfaces given observations and requiring separate measurement of the material relevant parameters to use as input [Hor89]. Learning-based approaches to single-view reconstruction seek to leverage priors learned from data rather than rigorous modeling of the light transport problem. The earliest learning-based approaches trained probabilistic models to predict coarse geometric structure of an image by assigning geometric parameters to image superpixels [HEH05, SSN08]. These methods were trained on relatively few images (82 to 534 images), and the quality is about good enough to produce plausible novel-view renderings using the inferred geometry. More modern deep-learning-based approaches are trained on the many large-scale RGBD datasets (such as KITTI, NYU, BlendedMVS, and many others) to make accurate, dense depth predictions from a single RGB image. For a time, neural network architectures for performing dense predictions on

images consisted exclusively of convolutional neural networks arranged in an encoder-decoder architecture [CPK+17, RFB15]. The small receptive field of convolutional layers necessitated large layer counts with many downsampling operations to keep memory costs reasonable, coming at the cost of feature granularity; much research has been devoted to preventing loss of detail in the output as a result. Recently, the transformer architecture [VSP+17], originally finding success in natural language tasks for its ability to reason about long-range dependencies in sequential data, has been adapted for prediction tasks on images by reducing an image into discrete tokens comprised of image patches [DBK+21]. This has shown to produce state-of-the-art results for dense prediction tasks including 3D depth from RGB images [RBK21], leveraging the transformer's attention mechanism to make use of global context at every stage, compared to the small receptive field of the image convolution.

In addition to these predominantly *image-space* techniques, another avenue of research that has seen greater traction recently is *full 3D generation*. Image-space methods do not operate on 3D data directly, but rather 2.5D representations such as depth maps. Conversely, methods that perform full 3D generation are capable of generating 3D representations such as volumes and point clouds, which necessarily involves hallucinating portions of the shape. As neural architectures for processing additional geometry representations have evolved (such as PointNet architectures for processing point clouds, and coordinate MLPs for generating neural implicit shapes), these works have become more prominent. The majority of these methods are single-object focused; that is, rather than reconstructing a scene, the task is to reconstruct a central object captured in the photograph. This is driven by the emergence of large CAD data repositories like ShapeNet with many examples of objects in a wide range of categories, which are conducive to learning generative models for 3D objects. Among the first methods to generate fully-3D output from one (or more) images was 3D-R2N2 [CXG+16], which used an encoder-decoder architecture to encode the input images into a latent space from which a 32^3 voxel grid shape could be decoded. The encoder and

decoder used convolutional neural networks, and LSTMs were used to aggregate observations from arbitrary numbers of views into the latent space in a principled manner. Subsequent work, Point Set Gen [FSG16], demonstrated more accurate results while outputting 3D point clouds rather than voxels; these have the advantage of eliminating the discretization artifacts inherent to voxel representations. AtlasNet [GFK+18] developed a novel neural surface patch representation using coordinate MLPs to encode parameterizations from $\mathbb{R}^2 \rightarrow \mathbb{R}^3$. By additionally taking a shape latent code as input, they trained a single-view reconstruction network producing surface patches as output. They demonstrate superior performance to both 3D-R2N2 and PSG based on chamfer distance. Notably, AtlasNet surface patches can be readily converted into a mesh, albeit non-watertight, by sampling grids in parameter space.

Upon the development of implicit neural representations (INRS) for shape generation [PFS+19], which represent shapes using the level sets of implicit functions from $\mathbb{R}^3 \rightarrow \mathbb{R}$ parameterized by neural networks, such representations were also adapted for the single-view reconstruction task. IM-Net [CZ19] and the concurrent work OccNet [MON+19] utilize an implicit shape decoder that takes a latent shape code plus a 3D coordinate to a real number representing whether one is inside or outside the shape, and similar to AtlasNet, also demonstrates the ability to perform single-view reconstruction by conditioning the generator on the output of an image encoder. The benefit of INRs is that they are not limited by a grid resolution, and can represent watertight surfaces with arbitrary topology. Other work has sought to increase the practical output resolution in voxel space by generating sparse octrees using an Octree Generating Network (OGN) [TDB17]. Matryoshka Networks [RR18] predict a layered set of depth maps from the input image, which are fused into a single volumetric object. All of these methods are trained on the ShapeNet shape collection using synthetic renders as input RGB images. State-of-the-art methods have also employed transformers to improve results [YWTT22, PIQD22].

However, it has been observed that many state-of-the-art single-view reconstruction systems

do not so much reconstruct as classify images into known shape categories [TRR+19]. A common procedure for 3D generation-based approaches is to first train a generative model of the 3D domain in question, using an auto-encoder architecture, and then to train an image encoder that embed images of objects into the corresponding point in the latent space, such that the end-to-end system can generate a 3D model from an image [GFRG16, GFK+18, CZ19]. This setup is already somewhat reminiscent of retrieval. This was demonstrated in [TRR+19] by comparing AtlasNet, OGN, and Matryoshka Networks against a pair of simple baselines including simple retrieval. They computed a unique shape descriptor for each model in the training set based on the pairwise similarity matrix of all shapes in the training set, and trained a ResNet image encoder to produce a shape descriptor closest to the target shape. This retrieval baseline outperformed the other single-view reconstruction methods.

By and large, the 3D-generation based body of work in single-view reconstruction appears ill-suited to the task of reconstructing CAD representations from images. Generated results tend to lack detail that would be necessary to make distinctions about where individual surface primitives lie. A notable exception is PolyGen [NGEB20], which is a generative auto-regressive model for CAD models represented as polygonal meshes that can be conditioned on images. By using transformer decoders to generate sequences of vertices, edges, and faces, with each order of topology referencing elements of lower orders through the use of pointer networks, an indexed N-gon mesh is generated. The model can also be conditioned on images to perform single-view reconstruction. Though the output is approaching the fabricable representation we are after, the results of single-view reconstruction do not necessarily match the input image. Being a conditional generative model trained on ShapeNet, it is likely the case that it would also fail to reconstruct more general fabricable objects not belonging to a known shape category. A related method, SolidGen [JLD+23a], uses an architecture similar to PolyGen to predict indexed B-reps instead of N-gon meshes, making use of post-processing to infer the exact parametric primitive types of

faces and edges based on the number and type of elements they reference. They, too, demonstrate conditional generation on images, though there is also high variance in the results compared to the input images. Notably, the ratio of valid results goes down as they condition on images, which they hypothesize as being due to the fact that the CAD dataset they train on has no clear categories, meaning images could be far from the training distribution. This reinforces the notion that conditional generative models do not generalize beyond shapes they have already seen, which makes them unsuitable for true reconstruction tasks in the wild.

Aside from regressing or optimizing 3D geometry directly to fit images, there is also a body of work that reconstructs scenes by retrieving matching objects from a database. Mask2CAD [KALD20] learns to predict object masks and object centers in an image of a scene, and retrieves and poses objects from a database into each masked region. They achieve the retrieval through the use of a shared embedding space between images and CAD models, which they learn by representing CAD models as k multi-view images. Since we are mainly focused on reconstructing individual objects, these methods are not directly applicable to our problem.

Recent Image-based techniques are of interest due to their fairly good performance at single-view depth reconstruction. The dense predictions produced by [RBK21], for example, can provide enough geometric detail to potentially extract information about surface primitives and their relations. This also indicates that a similar architecture for predicting per-pixel surface primitive parameters might prove a promising direction.

2.2.4 Inferring parametric primitives from dense geometry

There is a large body of work addressing the problem of **reverse engineering** CAD representations from dense geometry such as point clouds or volumes to obtain parametric primitives [WFAR02, BTS+17]. The classical methods in this area focus largely on constrained numerical optimization.

These early works often solved the problem using variational approaches that iterate between segmenting the input point cloud and fitting primitives to regions. A prominent and widely-used probabilistic approach is Efficient RANSAC [SWK07]. This method adapts the common RANSAC algorithm to the problem of fitting an arbitrary number of primitives (planes, cylinders, cones, spheres, and tori) to a point cloud using an efficient candidate sampling strategy to discover primitives with large point support. The downside is a reliance on a hand-picked distance distance threshold for points to be considered part of a primitive. GlobFit [LWC+11] attempts to improve the robustness of the RANSAC approach by incorporating geometric priors, or *alignments*, such as orientation alignment, orthogonality, regular angles, coplanarity, coaxiality and equality between primitives, by means of constrained optimization. This involves iteratively running RANSAC, discovering candidate relations in the fitted primitives in the form of a *relation graph*, and alignment by re-optimizing the primitive fits subject to the discovered constraints, after which any points no longer near any primitives are run through RANSAC again, and the process repeats until convergence. One issue with GlobFit is the highly combinatorial nature of the graph search problem when finding alignments, which quickly gets out of control for large number of primitives. The aforementioned methods are all prone to local optima due to the combinatorial complexity of CAD representations.

More recent methods for inferring primitive structures from point clouds consists of deep-learning-aided methods such as SPFN [LSD+19a], ParSeNet [SLM+20], PrimitiveNet [HZS21] and HPNet [YYM+21a]. In SPFN, an input point cloud is fed into a PointNet++ encoder to produce per-point labels indicating membership probabilities for each of K primitives, geometric normals, and primitive types. The core component is a *differentiable model estimator* which, given these per-point predictions, can differentiably fit the optimal primitive parameters for each of the K primitives, enabling end-to-end training using geometric supervision in the form of squared distance to the fitted primitives. ParSeNet adds the ability to fit arbitrary spline patches by adding a differentiable

spline-fitting module, improving expressivity of the method, as well as adopting a metric learning approach to segmentation that gives better results. PrimitiveNet [HZS21] employ both global and local features to enable primitive instance segmentation of large scenes, and additionally use adversarial training by means of a primitive discriminator network to achieve state-of-the-art results. HPNet [YYM+21a] uses a DGCNN [WSL+19] to learn dense, per-point descriptors (normals, primitive types, and primitive parameters, including spline parameters obtained using a the pre-trained SplineNet from [SLM+20]). They introduce two spectral embedding modules, which take adjacency matrices constructed based on distances to neighboring points' primitives and agreement between neighboring points' normals, in order to create feature vectors that incorporate information about local smoothness of normals and consistency of primitive assignments. The combined feature vectors are then supervised using metric learning in order to cluster points belonging to the same primitive instances, after which mean-shift clustering is used to compute the final primitive segmentation. More recently, Point2CAD [LOWS24] collects various existing point cloud segmentation approaches (such as HPNet, ParSeNet, SPFN and others) into a holistic framework for extracting B-Rep geometry by a unified geometry optimization strategy.

Rather than treat primitive segmentation and fitting as separate steps, the recent work ComplexGen [GLP+22] has taken a holistic approach by directly predicting the parametric primitives of different orders (surfaces, curves, vertices) and their combinatorial structure, forming a full B-rep chain complex, in a unified neural framework. They utilize transformer decoders to decode the encoded point cloud into neural parametric curves and surface patches, along with adjacency matrices establishing the boundary relations between them. The post-process the result by removing redundant predicted elements, then perform combinatorial optimization, to compute the most probable connectivity given the predicted adjacencies, and finally perform geometric refinement to ensure that the shapes obey the inferred topological structure. They show that compared to methods that perform segmentation followed by fitting, like ParSeNet, their method

is more robust to inputs perturbed by noise, indicating that their approach can find more globally optimal solutions.

Another recent approach, Split-And-Fit [LCP+24], takes a novel approach of inferring volumetric Voronoi diagrams representing distance to the nearest parametric surface primitive at each point in space, from which the final primitive structures can be inferred. This has the advantage of decoupling inference from the input point cloud compared to the aforementioned point cloud segmentation approaches.

A somewhat related related body of work deals with "shape parsing," or inferring abstract primitive structure representations of 3D shapes [ZYY+17, SDR+22, PUG19]. Rather than exactly fitting surfaces or volumetric primitives to the geometry, these works are more concerned with capturing the high-level parts-based structure of shapes through the hierarchical relations of the primitives, which are usually simple cuboids or ellipsoids.

It is not obvious how most of these approaches could be adapted to the problem of reconstruction from partial observations, short of using them in conjunction with generative models for completing point clouds—in particular, it is uncertain how robust these methods would be to incomplete geometry. Certainly, if one of our goals is to utilize domain-specific constraints to infer unobserved geometry, methods relying on segmentation followed by fitting [LSD+19a, SLM+20, YYM+21a] would have no way to extend the geometry to include unobserved faces. While ComplexGen doesn't necessarily possess this limitation, the involved combinatorial optimization and refinement steps depend on all candidate primitives being found by the network in the first stage, which may be difficult in the absence of complete observations.

2.2.5 Inferring edge structures

Rather than infer surface primitives, another direction is to generate wireframe structures. This is in some ways more challenging, since it places more emphasis on the precise topological structures. DEF [MRA+22] takes a set of depth images as input, and predicts a distance field over the shape, a "distance-to-feature" field to the nearest sharp features in the input shape. [LDSW21] convert a 3D point cloud to a wireframe model using deep neural nets in several stages by first predicting a set of candidate vertices, then pruning the set of all possible edges between vertices to obtain the final wireframe. To achieve this, they use FGCF [CPK19] to predict deep point-wise features for the point cloud, followed by a block that predicts whether "patches" or neighborhoods of points contain a corner. After regressing the locations of the predicted corners, they similarly predict whether candidate edges should remain in the final wireframe using non-maximal suppression. They develop various metrics for evaluating wireframe accuracy which incorporate potential topological differences between compared wireframes, including **Mean Average precision for vertices**, **Point-wise precision and recall for edges**, **Structural average precision for wireframes**, **Wireframe edit distance (WED)**. Neural Edge Fields [YYG+23] extract wireframes from multi-view images. Unlike [LDSW21], a dense point cloud is not required, and these wireframes can include curves as well as straight lines, making the method more suitable for practical CAD settings. First, they compute edge maps from all of the input images using PiDiNet [SLY+21]. Then, they optimize a neural radiance field (NeRF) subject to the edge maps in order to obtain a volumetric density field representing the wireframe. This optimization is difficult due to the sparsity of the edges and view inconsistency due to occlusions, so they employ various loss terms to mitigate this: a consistency loss that down-weights false negative edges caused by occlusions, and a sparsity loss which is a regularizer term to penalize unnecessary edge densities within the volume. Given the optimized NEF, the next step is to extract 3D parametric curves by fitting in a

coarse-to-fine manner. A seeming drawback of this method is that they have only demonstrated results on extremely synthetic floating shapes with flat shading, of the sort would not cause any false positive edges in the edge detection stage; it is unclear whether this entire process would still work on real images.

Nevertheless, wireframes provide rich insights into the combinatorial structure of a shape; a method to produce accurate wireframe structures from images would be instrumental in eventually inferring a fabricable representation.

2.2.6 Inferring procedural CAD representations

Along with B-reps and primitive structures, there are also procedural CAD representations which treat a 3D shape as a sequence of operations, such as sketches and extrudes, or constructive solid geometry (CSG) trees. Such representations are used internally in many CAD systems, and are used for defining so-called parametric CAD models, which expose meaningful parameters that change aspects of a design, such as the length or spacing of chair legs. Inferring such representations can truly be said to be reverse engineering in the sense that one is recovering the underlying design intention behind a 3D shape. These are, however, difficult to infer, as programs are among the most information-dense representations for a 3D shape. Recently, however, large, annotated CAD data collections such as the ABC [KMJ+19a] and Fusion360 [WPL+21a] datasets have been released, making spurring research into data-driven approaches to generating and reconstructing CAD representations.

Given sufficiently clean input geometry, analytical optimization-driven approaches exist for reverse engineering procedural representations. Given a mesh, InverseCSG [DIP+18] infers a CSG tree by first finding all surface primitives using a modified version of Efficient RANSAC [SWK07]. This naturally divides the entire space into regions carved out by these primitives, called “canonical

intersection terms." Then, by intelligently sampling points in the space inside and outside the mesh, they constrain each canonical intersection term to be considered solid or empty space, which is then followed by a program synthesis step, which is a search for an optimal binary CSG tree that satisfies the above constraints (employing a SAT solver-based approach to make the combinatorial problem tractable).

More recently, [GLPG22] introduce neural half-space representations for converting a B-rep into a boolean CSG tree whose leaves are *neural halfspaces* represented by coordinate MLPs taking $\mathbb{R}^3 \rightarrow \mathbb{R}$. They first perform discrete optimization to generate a tree structure, then use a learning-based method to infer the neural halfspace at each node. CapriNet [YCL+21] also converts input CAD models into boolean CSG representations by first predicting P primitives and constructing a CSG tree. Their method is self-supervised using reconstruction loss, requiring no ground truth CSG assemblies.

Generative approaches Other methods have been developed to generate procedural CAD representations according to a learned latent distribution, enabling reconstruction of a sort in the form of conditional generation. DeepCAD [WXZ21] pioneers a generative model of CAD shapes represented procedurally using sketch-and-extrude operations, though they leave conditioning on input geometry for future work. SkexGen [XWL+22] builds on DeepCAD, but employs disentangled codebooks to allow control over generated results using a separate *geometry* and *topology* codes. Point2Cyl [UyCS+22] builds sketch-and-extrude CAD models from point clouds by fitting generalized cylinders to an input point cloud in a supervised way, since any sketch-extrude model can be formed by the CSG combination of generalized cylinders. They first predict per-point geometric proxies including instance labels for cylinders, followed by a differentiable fitting step, allowing them to supervise the consistency of the output model with the ground truth CAD profile sketches, among other geometric losses. [LWJ+22] learns to reconstruct prismatic CAD

models (defined by sketch-and-extrude programs) from rounded voxel shapes. The rationale behind the use of rounded voxels is that many reconstruction processes smooth their output as a regularization step, resulting in rounded shapes. To overcome the combinatorial complexity of CAD program search, they extract a small set of template sketch-extrude construction sequences and use a separate prediction branch for each one (picking the one with the highest reconstruction accuracy at the end). The input is fed into a voxel encoder, which is then decoded in each template branch by a series of linear layers to produce discrete cross-section and extrusion profiles. Additionally, they extract 1690 sketch shapes beforehand which are then retrieved at inference time to form the final profile shapes, thus avoiding the problem of inferring parametric sketch curves directly.

Methods for predicting procedural CAD programs generally only deal with very simple shapes, or make stringent assumptions about the geometry, such as [LWJ+22] with its template construction sequences. Several recent approaches have been proposed to infer CAD geometry from images and other inputs, but they are limited to the simple geometric domain of sketch-extrude CAD [YUH+24, CYH+24, KDA+24], or otherwise restricted to a learned latent space [XLJ+24, JLD+23b].

2.3 Predicting Functionality from Geometry

While we have looked at approaches to infer geometry from static observations, design representations may specify other important properties, such as functionality and motion of parts. Since the functionality of objects is something we can often intuit at a glance, it seems reasonable to expect that we could learn to automatically reconstruct such information from visual cues, such as the geometric structure of a CAD model. Many learning representations and datasets have been developed in recent years that could potentially aid in this task.

Deep Learning on CAD Data Large repositories of CAD formatted geometry and assemblies have long been available [gra], and there has been an explosion of large curated [KMJ+19b] and annotated [SOZA20, WPL+21b, JHC+21a] datasets. B-Reps have a natural graph representation, which has inspired several graph neural networks for B-Rep modeling and classification tasks [CRH+20, JSL+21, MSK+21, LWJ+21]. Two works address B-Rep assemblies; [JHC+21a] predict part alignment and joint type given rough locations on pairs of parts and provides the dateset used in this work, while [WJC+21] predict alignment axes and offsets for part pairs, but not joint type. These works are mainly geared towards helping to auto-complete part placement during the design process, rather than predicting the motion of an entire object given its geometry.

Learning-Based Motion Inference from Static Geometry Looking beyond CAD, many works have addressed the problem of learning motion representations directly from dense geometry. [HLVK+17] use metric learning to query similar articulated pairs from a database of such constructs. [WZS+19] train separate motion proposal and optimization networks on point clouds to segment parts and infer motion as a motion *type* and *axis* for each part. This motion representation captures the rigid motions exhibited in mechanical objects, and is what we use in this work. [YHY+20] instead infer per-point displacements, allowing more general part motions to be represented. [XRSR22] avoid needing joint annotations by learning similarity transformations within a semantic object category. All of these works rely on semantic object categories to make learning tractable, due to the unstructured geometry they use as input.

Chapter 3

Fabrication-Aware Reverse Engineering for Carpentry

The work discussed in this chapter was presented at SGP 2021 and published in the Eurographics proceedings of that year [NZCS21].

Carpentered, wooden furniture and objects abound in our everyday lives, from stools to bookshelves to trays for carrying food. These objects are composed of parts that have been cut from wood, which limits the types of operations involved, and therefore geometries produced, to

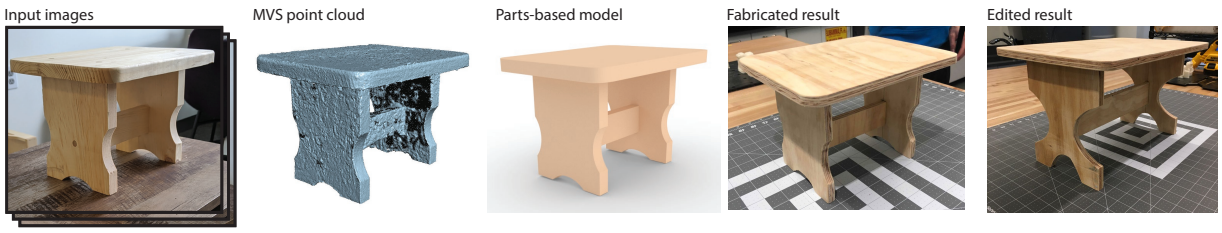

Figure 3.1: Given a set of images of a carpentered object and the resulting multi-view stereo (MVS) reconstruction, we produce a concise, part-based assembly that can be fabricated, producing a physical replica nearly identical to the original. Our model is easily edited in CAD software, allowing for the fabrication of modified versions of captured designs (right).

those permissible within the constraints of the carpentry domain. We propose a novel direction for recovering representations of carpented objects: describing the *fabrication process itself*. By framing this as a reverse engineering problem in a specific fabrication domain, we introduce constraints that significantly reduce the search space of viable 3D models. In particular, we operate on carpentered objects consisting of parts that are cut from sheets of wood and then connected together. The space of fabrication instructions in this domain is still highly expressive, covering a variety of everyday objects, as we will show, while also adhering to the real-world constraints governing the construction process, so that the output is ready to be built.

This reverse engineering problem introduces its own set of challenges: arriving at a fabricable solution requires identifying the parts, and optimizing for their precise shapes and the part-to-part connections constraining those shapes. This mixture of discrete and continuous degrees of freedom makes for a challenging optimization problem; to make this more tractable, we propose a multi-stage algorithm in which we first select the initial geometry and positions of parts in the assembled object, progressively detect assembly constraints (i.e. connections between parts), and then refine the geometry subject to these new constraints. The input images, captured by simply walking around an object and taking photos with a smartphone, guide this process at multiple stages. First, we use the images to recover a multi-view stereo point cloud that, though incomplete, drives the initial CAD part recovery. Second, the images provide evidence of seams – discontinuities in appearance – that indicate how different pieces of wood fit together when the connections are otherwise ambiguous based on geometry alone. Finally, by rectifying the images to each part plane, we can co-segment the part faces to obtain more accurate contours, i.e., cut paths for fabrication. Each of these co-segmented contours, extracted at the pixel level, is not concise and may not respect assembly constraints; we additionally propose an algorithm to find a simple parametric boundary that accurately represents the cut path of each part while respecting contact constraints between parts.

Our key contributions are:

- A fabrication-aware pipeline for selecting a plausible part structure representing the input object
- An algorithm for recovering contacts between connected parts using geometric and image evidence
- A method for extracting cut paths representing part shapes using multiple image views
- A method for incorporating assembly constraints into fitting of regularized, parametric contours to imperfect data

We note that, since our approach is based on features observed in images, we assume that the carpentered objects are textured, which is typical for wood that is unfinished or varnished, but not painted a uniform color. Further, though we don't require complete reconstruction of the surface, we do require that the wood sheet surface of each part be at least partially visible. Finally, we restrict the class of objects reconstructed to those that can be assembled with parts cut from sheets of wood. We demonstrate results on a variety of objects of different size and complexity, and show the efficacy of our method by fabricating two of our results, along with edited versions.

3.1 Overview

The input to our algorithm is a set of images of a carpentered object taken from different viewpoints. The output is a fabricable model describing the set of parts along with how they should be connected. Parts are assumed to be cut from wood *sheets*, so that their boundary contains two *sheet planes* (from the front and back of the sheet). Parts are represented as a triplet of the wood sheet position T (a rigid transformation), the sheet thickness d, and cut path Θ represented in the

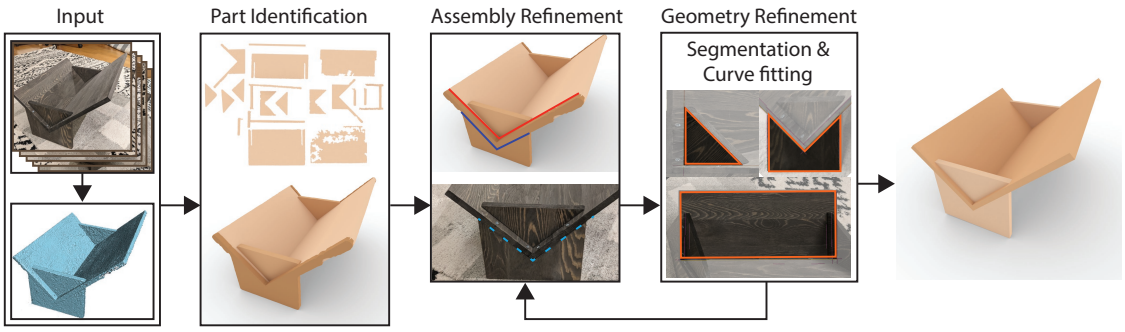

Figure 3.2: Overview of our method: Given input images, we recover an oriented point cloud with 3D reconstruction software. The images, camera poses, and point cloud are inputs to our system. We then identify an overcomplete set of parts based on plane fits to the point cloud, along with the part's cut path and thickness. These are pruned to minimize volume overlap while still adhering to the point cloud. The resulting part set is then refined in the assembly stage that determines how parts connect, resolves remaining volume conflicts (based on visible evidence of seams), and re-aligns the parts. The geometry refinement stage then uses multi-view image segmentation followed by concise curve fitting, both constrained by part connections, to obtain more accurate and more CAD-like cut paths. The geometry refinement can lead to cut paths that imply new part contacts, and thus we alternate between assembly and geometry refinement until the final model is complete.

2D plane of the sheet. We allow individual straight cuts to be made at arbitrary angles against the wood plane to allow for slanted contacts between parts; Θ also contains these bevel angles. Note that the cut path may include holes in the interior of the shape. The assembly information specifies a list of part pairs that are joined, and the surfaces along which these connections occur, which we represent using *interfaces* that we assume to be planar, as planar contacts are common in carpentry assembly. We say the model is *fabricable* if, in their assembled configuration, the parts do not overlap and meet snugly at the joints. We refer to the pairwise contact constraints implied by this requirement as *assembly constraints*.

While assembly constraints assist in recovering partially unobserved parts, the need to infer both the set of parts and the assembly constraints from incomplete data presents a challenge: Inferring the shapes and positions of parts depends on how they are connected, and likewise finding probable connections depends on the part geometry. This interdependence implies that these properties should be considered jointly in order to arrive at a feasible solution. To address the complex search space of possible part assemblies, we adopt a multi-stage approach in which we first detect an approximate set of parts absent any connections, then iteratively refine the model by alternately optimizing for the connection contacts and the part geometry subject to the new contact constraints. Finally, we approximate each cut path with a concise, piecewise-smooth curve that balances simplicity and accuracy. Our approach is illustrated in Figure 3.2.

Preparing the Input. Given our images, we use 3D reconstruction software [Cap] to obtain camera poses and a semi-dense, oriented, point cloud $\mathcal{S}$ (positions and normals) for the observed surfaces. The point cloud is not expected to capture every surface; entire sides of the input model may be missed. The images are acquired by walking around the model and taking photos, with enough coverage that at least one sheet plane per part is observed well enough to be partially reconstructed. The point cloud is expected to have the model separated from background points

as well as being oriented so that the object is approximately vertical, which in practice means the user indicates a ground plane and rough bounding volume for the object.

Part Identification. The goal of the first stage is to recover a set of parts, each with a rough approximation of T, d, Θ. These parts should closely match $\mathcal{S}$ and, although they may not strictly satisfy assembly constraints, they should maximise assembly feasibility in order to serve as a plausible basis for subsequent refinement. Our approach is to initially detect an over-complete set of candidate parts by searching for planes that could be wooden sheets in the point cloud and extracting initial thickness and shape from point cloud features. We merge candidate parts when they are better represented as single sheets, e.g. when they originate from disparate points observed from opposite sides of the sheet. From among these candidates, we extract a subset $\mathcal{P}$ of the parts with the best coverage of $\mathcal{S}$ that is also plausible from a fabrication standpoint; parts should not represent cuts through implausibly thick wood planes, and every part should subtend a minimum volume free of overlaps with other parts.

Model Assembly Having decided on an initial set of parts $\mathcal{P}$ from the part identification stage, we can proceed to infer the *assembly*—defined by the connections between parts—and refine the part orientations to regularize the angles in the design. Detecting the correct joinery between parts is challenging since it is "hidden" beneath the surface geometry of the model; we address this problem with two key insights. First, we can identify a small set of types of connections possible with our fabrication assumptions, which significantly reduces the search space. Second, we can use image cues, such as the presence of seams or material changes visible in the wood, to identify regions where a connection interface is likely to exist (if geometric cues are insufficient). We use these ideas to disambiguate connections, followed by a global optimization step that aligns near-orthogonal connected parts while staying close to the point cloud.

Geometry Refinement. The final step is to refine the geometry of the parts to obtain a fabricable model consisting of concise parametric curves, ensuring that the final model is consistent with the assembly constraints and represented using only the necessary number of primitives. We strive for simplicity to facilitate editing and because it is also usually consistent with how objects are designed. We utilize the shapes visible in the input images to obtain more accurate cut path contours: Since the sheet plane position T has already been identified for every part, we can use it as a reference to drive a multi-view image segmentation after projecting each image into this plane. We then apply a curve fitting algorithm over the resulting segmentation mask boundary which preserves the assembly constraints while globally minimizing an energy function that balances complexity and accuracy. Our final result is regularized by aligning curves and lines in the resulting shapes. In practice, the refined shapes may reveal new connections, so we iterate model assembly and refinement until no new connections are found.

3.2 Technique

3.2.1 Parameters

Given oriented point cloud $\mathcal{S}$, we define the global diameter D as the points' maximum bounding box dimension. Our method has many parameters which depend on the scale of the model, which is arbitrary; we therefore define these parameters in terms of D. For detecting orthogonality and parallel features throughout our pipeline, we have a global angle threshold α.

3.2.2 Part Identification

In this stage, we both identify parts that comprise the model and estimate each part's rigid pose and rough shapes, as a basis for subsequent refinement. Our strategy for identifying the set of

parts builds on the assumption that they are cut-outs from flat sheets and the fact that planes can be easily detected from 3D point clouds. Based on this insight, we can begin with primitive detection on the point cloud, followed by generating 3D parts from extrusions of paths in those planes, which amounts to detecting a cut path Θ and the sheet thickness d. Note that using all detected planes as potential part planes results in many more parts than are actually in the model, as any given part has at least two planar surfaces, and more for straight line cuts; ultimately, only a subset of planes should be used. Our approach is to first generate part geometry for *all* detected planes to form an over-complete set of candidate parts, then optimize for the subset that best approximates the model while representing a feasible construction.

Primitive Detection and Adjacency

We employ Efficient RANSAC [SWK07] to segment the oriented point cloud into clusters of points that fit planes and cylinders; the points are roughly contiguous sets on those primitives. The planes correspond to wood sheets, as well as any straight line cuts. The points that fit better to cylinders tend to lie only on curved cut paths. We used an inlier threshold of $\tau = D/300$ for the RANSAC algorithm.

We say that two primitives are adjacent if the minimum distance between their respective point sets is less than τ. We track adjacency between plane primitives and other primitives; let Adj_i be the set of primitives adjacent to plane primitive i. We later use adjacency to guide part depth estimation and to help bound the cut path for each part.

At this stage, every plane primitive now corresponds to a part P_i with transformation T_i that maps the x-y plane to that primitive plane. This set of parts is highly redundant; e.g., a fully observed cuboid part would have six planes corresponding to the sides of the part, and thus this one part would initially be over-represented by six parts. We address part redundancy in the part selection phase at the end of this section. First, we estimate the cut path Θ and thickness d for

every candidate part.

Cut Region Approximation

Each detected plane is a candidate to be a part's wood sheet plane. We approximate its cut path as follows: collect the plane's associated points, transform them by T^{-1} and project them onto the $x-y$ plane, convolve (in 2D) with a Gaussian kernel ($\sigma = D/400$), sample the resulting field over a regular grid, and extract a set of isocontours (isovalue of $1/\sqrt{e}$ where e is Euler's number). The continuous field from which we extract these isocontours can be written as:

$$f(\mathbf{x}) = \sum_{\mathbf{q}} \exp(-\|\mathbf{q} - \mathbf{x}\|^2/(2\sigma)) \tag{3.1}$$

where $\mathbf{q}$ are the 2D points, and σ is $(D/400)$. We extract a level set $\{x|f(x) = 1/e\}$ using the Marching Squares algorithm with a grid length of 4σ.

We assign the contour with the largest enclosed region, along with any countours inside that region, to be the approximate cut path. Note that interior contours enable parts to have holes cut into them.

Initial Sheet Thickness Estimation

The thickness of a part can be determined in two ways: by measuring the cut surface or by the distance between its front and back sheet planes. As we may not always see the back face (e.g., a part facing downward near the floor that happens to be missed during capture), we initially estimate the cut surface width to determine thickness.

For each candidate part P_i, we estimate the cut surface width using a discrete plane sweep approach. Specifically, we collect the points associated with the part plane's adjacent primitives Adj_i. We then move the plane of P_i in the direction opposite its normal in discrete jumps Δd

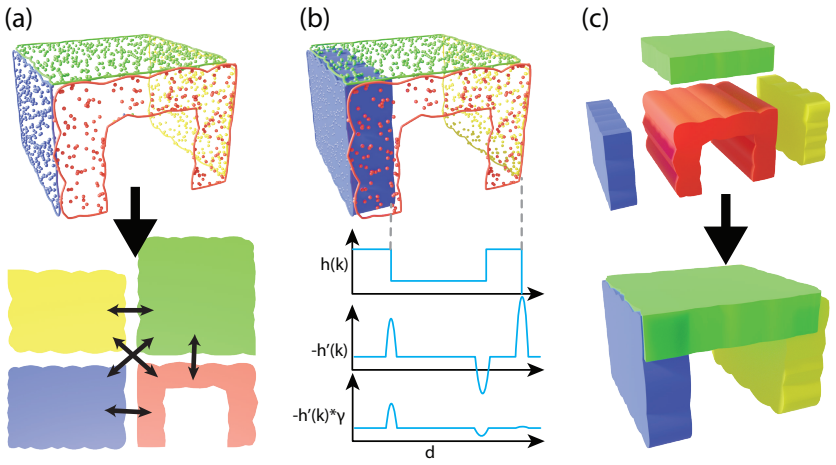

Figure 3.3: Overview of part identification. (a) We begin by detecting primitives and adjacencies between them, along with initial shapes for each plane. (In this example, the far side was not observed, thus the red curve does not have a symmetric counterpart). (b) We illustrate finding the thickness of the blue part by sweeping its plane and counting points on adjacent primitives near the sweep plane, forming histogram $h(k)$. We identify the plane-sweep thickness as the largest robustly identify peak, weighted by a spatial discount factor $\gamma(k)$. (c) shows the selected subset of generated candidate parts for the final model.

and form a histogram $h(k)$ of adjacent points within $\Delta d/2$ of the plane offset by $k\Delta d$. We set $\Delta d = D/100$.

When the plane sweeps past its cut surface, we expect a sharp discontinuity in the histogram, i.e., a peak in $-h'(k)$. We compute $-h'(k)$ using finite differences and identify robust peaks with non-maximum suppression. The largest peak may not correspond to the correct thickness; peaks closer to the part plane should take precedence (see Figure 3.3 (b)). To address this, we weight the peak magnitudes with a spatial discounting factor, $\gamma(k) = \exp(-c_{\text{falloff}} \cdot k\Delta d)$. We set $c_{\text{falloff}} = 5/D$. The part thickness after this stage is set to $d = k_{\text{peak}}\Delta d$, where k_{peak} is the bin of the largest robust, spatially-weighted peak.

We additionally estimate thickness by considering planes that could be the opposite side of a part's wood sheet. Given part P_i, for each part P_j with opposite sheet normal, we transform its cut path Θ_j by T_j^{-1} and project it onto the plane of P_i. If Θ_i and Θ_j overlap, we consider the distance

between the planes of P_i and P_j to be a candidate thickness; we take the min over all of these thicknesses, call it d_{opposite}. If d_{opposite} is within Δd of d_i computed above, we then set $d_i = d_{\text{opposite}}$. During this step, we also record all other opposing parts j with cut path overlap that are within Δd of the final thickness (not just the closest part); call this set O_i, to be used later for merging parts.

Part Selection

The final step in the part identification stage is to select a subset of parts to be assembled and refined in the next stages. We first reduce the number of parts through pruning and merging steps, and then perform a global optimization to give a set of parts that covers $\mathcal{S}$ well without too much overlap between parts.

To prune the part set, we first adopt a heuristic: parts are unlikely to be much deeper (thicker) than they are wide. For example, if we have a cuboid part that is 30cm x 30cm x 1cm, we will prefer a 30cm x 30cm face cut into a sheet 1cm thick over a 1cm x 30cm face cut into sheet 30cm thick. We prune as follows: if the estimated thickness of a P_i yields a cut surface with total surface area greater than five times the area of Θ_i, we discard it. In Figure 3.3, the red part is one such candidate and is thus discarded.

Next, we merge part candidates if we have evidence they correspond to a single cut of the same wood sheet. In particular, for part P_i, the set O_i contains parts with opposite faces, cut paths overlapping P_i's, and with planes roughly d_i away from the plane of P_i. These opposing parts are likely part of the same cut from the same sheet, and thus we transform and project the cut path Θ_j for each part $P_j \in O_i$ into the plane of P_i and take its union with Θ_i, after which we discard part P_j. This step is useful to recover parts only reconstructed partially from different sides due to occlusions. We perform this process recursively until no such opposite-and-overlapping candidates remain. Note that, when merging P_j into P_i, we also merge the adjacency sets, i.e.,

$\text{Adj}_i \leftarrow \text{Adj}_i \cup \text{Adj}_j$, useful later for geometry refinement.

Among the remaining parts, we generally still have over-representation, i.e., parts that overlap each other heavily. Some amount of overlap is tolerable. E.g., two cuboid parts that meet at a corner may overlap because it is unclear which part goes all the way to the corner and which has a cut face that abuts that other part; we will allow this small overlap and disambiguate it in the next section. We now pose a (non-trivial) discrete optimization problem: select the set of parts $\mathcal{P}$ that minimizes distance between $\mathcal{S}$ and $\mathcal{P}$ without conflict. We say that a part is in *conflict* if more than half of its volume overlaps with other parts in $\mathcal{P}$. We use simulated annealing to optimize for the final subset, where our energy function is the total squared distance between points in $\mathcal{S}$ and $\mathcal{P}$. We scale down all dimensions by $1/D$ to ensure consistent behavior regardless of scale. To solve this problem, we employ the Metropolis-Hastings (MH) algorithm with transitions consisting of either adding or removing parts from the solution set, while prohibiting changes that lead to conflicts. Simply adding or removing individual parts at each step, however, results in poor convergence due to the large number of potential conflicts, so we additionally permit "replacement" transitions in which a part in the set may be swapped for another outside the set if adding the outside part would have otherwise caused it to be in conflict. To be precise, our proposal distribution is the result of the following decision process: with equal probability, either choose a part uniformly at random to add or remove from the set, or perform a replacement move between two parts as discussed above. We run MH for 1000 iterations with start and end temperatures of 10 and 0.1, respectively, and find that results typically converge within 10 seconds.

3.2.3 Assembly Refinement

Model assembly involves determining which pairs of parts are connected and the surfaces at which parts make contact, known as interfaces. Our approach is to first identify the connections

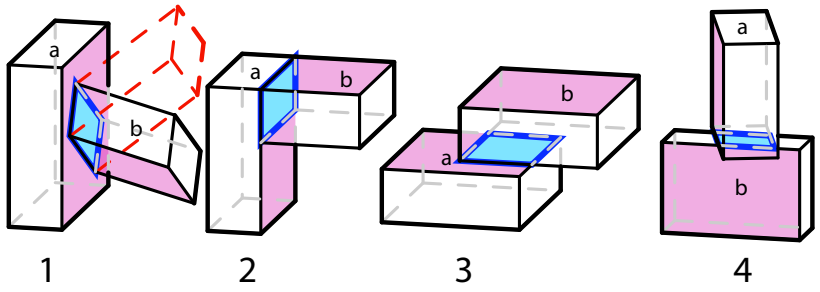

Figure 3.4: Types of unique connections based on which surfaces make contact. Connection interfaces are shown in blue. Each part's sheet plane is highlighted in pink. (1) cut to center face ; (2) cut to corner face ; (3) face-only; (4) cut-only. In (1), we illustrate that the sheet planes need not be orthogonal, made possible with a bevel cut to b.

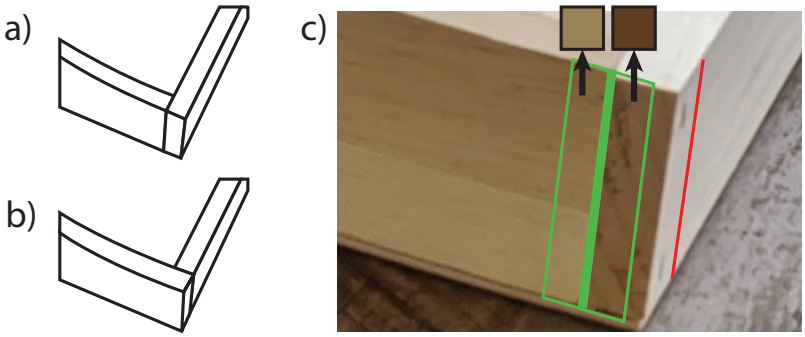

Figure 3.5: Two competing corner configurations (left), and the seams in the image used to disambiguate them. The color difference suggests that the green line is the boundary between parts, rather than the red line.

for individual pairs of parts followed by a global alignment step that ensures manufacturability constraints over the graph of connections. The result of finding and classifying the connections between parts is a set of planar *interface surfaces* along which the pairs of parts join, shown in blue in Figure 3.4.

Connections

First, we identify pairs of parts to connect. Specifically, if the minimum distance between the surfaces of two parts is less than $\tau_c = D/30$, then they are connected.

Next, we determine the type of connection between the parts. Based on our fabrication assumptions we identify 4 possible types of unique connections according to the types of surfaces that make contact, illustrated in Figure 3.4. We exclusively deal with connection types 1 and 2, as we did not observe the other types in any of our example models. If the sheet planes of parts a and b are not orthogonal, as shown in Figure 3.4 (1), we use bevel cuts to satisfy the planar contact. To detect type 1 connections between two parts a and b, we determine whether a and b form a T-junction by checking if b terminates at a's sheet plane, and in the reverse case, if a terminates at b's sheet plane. If both a and b terminate at the other part's sheet plane, we say they meet at a corner and classify the connection as type 2.

To detect whether part b approximately terminates at part a's sheet plane, thus forming a T-junction necessary for considering type 1 connections, we use part a's sheet normal $\mathbf{n}_a$, and a's sheet plane offsets $o_{\min}$ and $o_{\max}$ to compute the projected offsets of all points in part b: $z_{\min} = \min_{\mathbf{p} \in P_b}(\mathbf{p} \cdot \mathbf{n}_a)$ and $z_{\max} = \max_{\mathbf{p} \in P_b}(\mathbf{p} \cdot \mathbf{n}_a)$. If

$$z_{\min} < o_{\min} - \tau_c \tag{3.2}$$

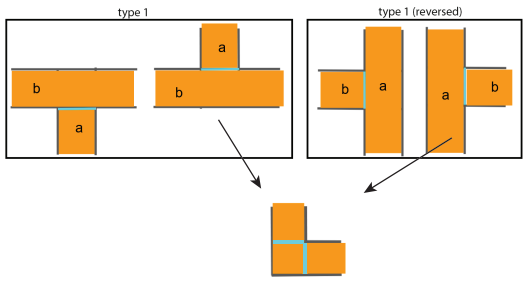

Figure 3.6: Top: Four cases of connection type 1 for a pair of parts, viewed from the side so that both parts' sheet normals are parallel to the page. Bottom: one corner configuration determined from the orientations of both candidate type 1 connections. Interface surfaces are shown in cyan.

and

$$z_{\max} > o_{\max} + \tau_c \tag{3.3}$$

are both true, part b is not confined to either side of part a's sheet, preventing connection type 1. As long as only one is true, the connection is allowed; the interface plane offset is $o_{\min}$ if (3.2) holds, and $o_{\max}$ if (3.3) holds. Type 2 connections occur when a type 1 connection is valid for both orderings of parts a and b.

Consequently, there are actually 4 discrete configurations for a type 1 connection between two parts, as shown in Figure 3.6: For the connection and its reverse (where a and b are swapped), the connection may involve contact with one of two sides of the sheet plane (whether the interface plane offset is $o_{\min}$ or $o_{\max}$). For type 2 connections, the positions of these contacts for both the the type 1 connection and its reverse are used to determine the corner configuration, as shown in the bottom of Figure 3.6; knowing where the potential contact surfaces lie is crucial to knowing where in the images to look for seams.

We assume corners (type 2 connections) are right angles. Though it would not be difficult to allow them to vary, our choice of two "natural" corner configurations requiring only orthogonal cuts no longer makes sense; bevel cuts will be needed no matter what, so potentially more complex joints would need to be detected. Furthermore, non-orthogonal corners are uncommon.

Disambiguating corners

A type 2 connection has two equally viable solutions for how the two parts meet. To resolve this ambiguity, we look for evidence of a seam in the images that may indicate which part extends to the corner (see Figure 3.5 (a) and (b)). Given the approximate part geometry, we can determine which images have an unoccluded view of the (possible) seam. For each such image I_i, we compute two measures of visual discontinuity. First, we compute the gradient in the direction orthogonal

to the seam at each pixel along the seam and average their magnitudes; call this value g_i. We additionally compute a very coarse gradient across the seam by computing the average color within a rectangle of width τ_c on either side of the seam (see Figure 3.5 (c)) and compute the magnitude of the difference of the colors, call it $\bar{g}_i$. The seam score for this view is just $g_i + \bar{g}_i$. We then average this score across all views of the seam to compute the final seam score and choose the type 2 configuration with the higher seam score. If there are no views of a seam (e.g., if it is against the floor and thus not viewable), then we assign it a seam score of 0.01 (where pixel intensities range form 0.0 to 1.0) so that it can still be chosen if the other seam score is low (not a seam).

Interface surfaces & constraints

After determining the connection types, we compute the finite interface surfaces within each interface plane indicating where the parts make contact. These give an estimate of where the final parts *will* make contact for purposes of constraining their shapes. In both cases 1 and 2, we find this by intersecting the solid shape of part b with the abutting plane of part a, offset by τ_c toward part b to correct for any gaps between parts caused by Θ^0. This interface can only take the form of one or more rectangles, each with width equal to the thickness of part b.

The constraints imposed by an interface surface on part b's cut shape are line segments that the shape cannot cross (without butting into another part), formed by the projection of the above surfaces onto the plane of part b. There are also additional constraints imposed by type 2 connections: In our final shape, the parts should meet perfectly at the corner determined by the line of intersection of the sheet planes from a and b on the outer surface of the corner. Finally, aside from connections, we have the constraints implied by adjacent plane primitives in Adj_i that meet the part plane with a convex interior angle, since neighboring cut surfaces are evidence of the shape boundary. We exclude adjacent planes associated with connected parts, as satisfying the

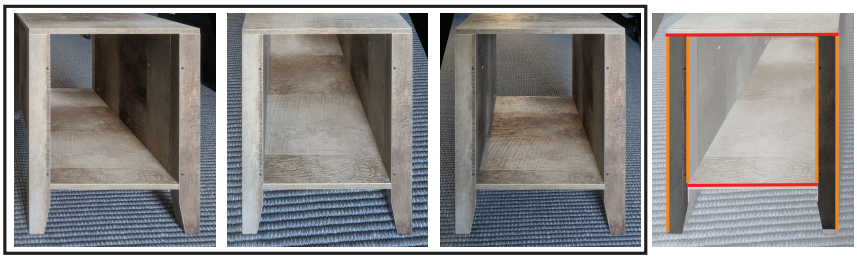

Figure 3.7: Several rectified images (left) used to obtain a segmentation mask of the cut region of a part (right). The regions where $\mathbf{M} = 1$ are shaded dark. Interface constraints are shown as the red regions; other surface constraints, such as those arising from neighboring detected planes, are shown as orange lines.

interface constraints should take precedence. These planes only correspond to planar cut faces; we do not include adjacent cylinder primitives, as we found their fits to be less faithful to the curves that they tend to fit. These *constraint segments* in each part's sheet plane are used during geometry refinement so that the result conforms to detected surfaces and the precise connections inferred above. They define a half-space in the plane that belongs outside the cut region, for points that project to the line within the segment boundaries. For each constraint segment, we also make note of the angle of the interface surface relative to the part so that we can accurately define the bevel cut angle to enable this contact later.

We also expect the final model to lie flat against the ground. To incorporate this constraint, we add a ground plane, positioned at the lowest point of the input geometry, and form a fake "part" which we include in the above assembly analysis above to obtain additional contact constraints for any parts in contact with the floor.

3.2.4 Geometry Refinement

In the final stage of our pipeline, we refine the cut path for each part to be consistent with assembly constraints and image evidence and to be represented with concise, piecewise smooth curves. We do this in two stages: Image co-segmentation and constrained curve fitting.

Joint Image-Based Segmentation

We leverage multiple views to optimize for a binary segmentation mask $\mathbf{M}$ in the sheet plane representing the cut region for each part. Taking inspiration from [KSS12], aimed at segmentation and plane reconstruction, we pose our multi-view segmentation problem as an MRF optimization. Unlike [KSS12], we use the known part plane as reference, leverage visibility cues in the rest of the reconstruction, and incorporate assembly constraints in the segmentation.

In particular, for part P_i, we project each image I_j from its camera viewpoint onto the nearest part plane of P_i. We resample the projection in the plane to form rectified image $\tilde{I}_j$. We then optimize for $\mathbf{M}$ by defining a binary MRF on the set of pixels $\mathcal{V}$ in $\mathbf{M}$ with 4-connected grid edges denoted by $\mathcal{E}$, with the energy

$$E_{MRF} = \sum_{\mathbf{x}\in\mathcal{V}} E_d(\mathbf{x}) + \lambda_s \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{E}} E_s(\mathbf{x},\mathbf{y}) \tag{3.4}$$

where E_d is an appearance-based cost, and E_s is a pairwise edge-sensitive smoothness term. These energy terms depend on the per pixel mask labels $\mathbf{M}(\mathbf{x})$ (1 for inside the cut path, 0 for outside), and we seek the lowest energy labeling with respect to all views together. We set $\lambda_s = 50$.

Data term We model appearance using Gaussian Mixture Models (GMMs) with 5 components for the colors (in LAB space) inside and outside the part's current cut region for each view, which gives us probabilities P_j^1 and P_j^0 that a pixel belongs inside or outside, respectively, for view j. For the interior, we erode the cut region by τ_c (multiplied by the world-to-pixel scale factor) since the pixels near the boundary of the initial cut region are uncertain, and then consider only the subset of those pixels potentially visible to view j; for this purpose we construct a visibility mask $\mathbf{V}_j(\mathbf{x})$ which is 0 if a ray from view j to pixel x intersects another part before reaching the part plane and 1 otherwise. For the exterior, we similarly dilate the cut region and consider all pixels outside

of it with $\mathbf{V}_j(\mathbf{x}) = 1$.

We now define the data term as:

$$E_d(\mathbf{x}) = -\frac{1}{N_V(\mathbf{x})} \cdot \sum_{j=1}^{N} \mathbf{V}_j(\mathbf{x}) \cdot \log(P_j^{\mathbf{M}(\mathbf{x})}(\mathbf{x})) \tag{3.5}$$

where $N_V(\mathbf{x}) = \sum_{j=1}^{N} \mathbf{V}_j(\mathbf{x})$ is the number of views that can see pixel x on the part plane. If N_V is 0, we set E_d to 0 regardless of $\mathbf{M}(\mathbf{x})$. In general, some "outside" pixels may belong to other parts with similar appearance, so we modify E_d to incorporate the constraint segments computed earlier: $E_d(\mathbf{x})$ is set to ∞ for $\mathbf{M}(\mathbf{x}) = 1$ if $\mathbf{x}$ is in the excluded region of any of the constraint segments. The result of using these constraint segments for segmentation is shown in Figure 3.7 (b). The red interface constraints prevent the mask region from including surfaces from adjacent parts, which purely appearance-based segmentation would do.

Smoothness term We define E_s using a contrast-sensitive Potts model to regularize the result while aligning the mask with high-contrast regions which we expect at shape boundaries.

$$E_s(\mathbf{x}, \mathbf{y}) = |\mathbf{M}(\mathbf{x}) - \mathbf{M}(\mathbf{y})| \exp\left(-\frac{1}{N_V(\mathbf{x})\sigma_s^2} \sum_{j=1}^{N} \mathbf{V}_j(\mathbf{x})(\tilde{\mathbf{I}}_j(\mathbf{x}) - \tilde{\mathbf{I}}_j(\mathbf{y}))^2\right) \tag{3.6}$$

where σ_s (set to 50) controls the strength of the smoothing penalty falloff as contrast increases. Analogous to how we modify E_d, we also set $E_s(\mathbf{x}, \mathbf{y})$ to zero in cases where the 3D locations corresponding to $\mathbf{x}$ and $\mathbf{y}$ straddle a constraint segment, to encourage the boundary of $\mathbf{M}$ to adhere to these known edges; i.e., there is no penalty for label change at these boundaries where label changes are likely.

In practice, for efficiency, we only consider views I_j for which $V_j(\mathbf{x}) = 1$ for at least half the pixels inside Θ, and then use the top seven views sorted by how close their central viewing rays are aligned with the plane normal. Note that this set of views may come from one or both sides of

the part. We solve the MRF using graph cuts [BJ01] to obtain the final mask. We also re-use the resulting **M** (cut path) to learn more accurate GMM parameters, and re-run the above algorithm once more to slightly improve results.

Updating model topology It is possible for **M** to have more than one connected component after optimizing **M** with the assembly constraints, if multiple parts were detected as one in previous steps (as is the case in Figure 3.7; the legs are forced into separate pieces by the assembly constraints). We restructure the model in these cases by adding each connected component in **M** as a separate part. Finally, we rerun the assembly stage to find new connections and constraints due to the updated shapes and potentially separated parts. We repeat the segmentation and assembly steps until no new connections are found (we observe at most 1 or 2 iterations in our experiments).

Global Alignment

Before extracting a final CAD model, we align connected parts that are close to orthogonal, as right angles are a feature of many manmade designs. This is important because it simplifies the fabrication process considerably as well; parts connected at right angles only require orthogonal cuts, which can be made with a wider variety of tools. Since this optimization only concerns small perturbations to the orientation, we represent each part P_i by its sheet plane π_i and optimize over plane parameters (normal $\mathbf{n}_i$ and offset o_i) such that detected orthogonal connected parts are aligned. We minimize the total squared distance of the planes to their detected point sets to regularize the result. We take an approach similar to [LWC+11] for plane alignment; we find

$$\min_{\forall \mathbf{n},o} \sum_i E_p(\mathcal{S}_i, \pi_i) \tag{3.7}$$

subject to $\mathbf{n}_i \cdot \mathbf{n}_j = 0$ for all i, j for which P_i and P_j are connected and the angle between $\mathbf{n}_i$ $\mathbf{n}_j$ is within α of 90°, where $\mathcal{S}_i$ is plane i's point set, and E_p is the total squared distance. To ensure unit normals, we represent each $\mathbf{n}_i$ using 2 angle parameters. We solve this global optimization problem using a sequential least-squares quadratic programming (SLSQP) solver, and then update T_i to align the parts with these new plane parameters.

Thickness Regularization

Typically, an object is constructed by cutting from a small number of wood sheets, with a small number of thicknesses. However, since our initial thicknesses are based on analysis of noisy point clouds (3.2.2), we typically estimate a different thickness for every part. Minimizing the number of distinct thicknesses in a model makes it more practical to build and therefore more plausible. Thus, we cluster thicknesses by averaging any part thicknesses that differ by less than a threshold τ_c.

Constrained Curve Fitting

Our curve fitting approach draws ideas from from prior work that trades-off global fit accuracy with curve complexity [PS83, FF02, FCOS05] as well as prior work that favors straight edges and sharp corners [ADSG$^+$20], and applies them to the context of handling imperfect binary masks, with certain known edges that the solution must adhere to.

For each part, the output of the segmentation step is a cut path defined by the raster boundary of a segmentation mask; it is neither exact nor concise. Our final step is to extract a CAD representation of this path by fitting a low-dimensional 2D shape representation that approximates the segmentation boundary while adhering to any contact constraints. In related works on vectorization, the perceptual criteria of accuracy and simplicity, along with continuity and regularity, are prominent objectives ([HDS$^+$18, KL11, ADSG$^+$20]). We find these objectives to be well-suited to

our problem: We desire a shape that is close to the input boundary while adhering exactly to the contact constraints, and which also provides a simple, continuous explanation for the input mask boundary, while capturing regularity in the man-made objects that are our focus.

We represent cut paths as closed G_0 continuous polycurves consisting of connected cubic Bézier curves and straight line segments, where we call the endpoints between neighboring segments *nodes*. For each part, our algorithm takes as input the raster boundary of the segmentation mask, a (clockwise) ordered set of 2D points X. We restrict nodes to lie on points in X and therefore have a discrete set of possible nodes, where each segment is the least-squares best fit for the range of data points between nodes.

We solve for the curves by building on the dynamic programming approach outlined in [PS83]. Let e_{ij} be the cost of fitting a curve to the subrange between X_i and X_j, which is the sum of squared point distance error plus a constant curve cost c_1. We define a sub-total energy E_{ij} as the least total error over all possible choices of nodes between X_i and X_j, giving rise to the recurrence relation $E_{ij} = \min_k(E_{ik} + e_{kj}), i < k < j$, allowing us to solve for the optimal node locations using dynamic programming. Support for G^1 continuity at curve transitions is added by pre-computing tangents at each point in X using curves fit to a local point neighborhoods, and constraining the end tangent directions of curves during fitting.

We add support for different curve types by extending E_{ij} to E_{ijk}, where k is the type of the curve ending at X_j; sub-total energies are now computed by summing over all previous curve types, as well as all previous nodes. We now have separate curve costs c_k for each type. We let $k = 0$ indicate line segments, and set the cost $c_0 = c_1/2$ to encode a preference for straight lines.

Dynamic Programming Fitting Algorithm The dynamic programming curve fitting algorithm can infer the optimal set of nodes from the input point set, with the caveat that it requires a starting point from which the optimal sub-ranges belonging to separate curves are determined.

This starting point is necessarily a node, since it is the start of the first such range returned by the algorithm. A first instinct might be to choose a starting point that looks like it should be a corner; however, if the input shape has no true sharp corners, the exact tangent behavior at the start of the loop will depend on the behavior at the end of the loop, which violates the sequential order in which we find the curves.

Instead, we do the opposite: We look for a starting node in a region that is as flat as possible (which we determine using the curvature of a Bézier curve fit to a neighborhood of points centered at the query point). Such a region can be assumed to always exist for well-behaved inputs approximating continuous shapes, and allows the tangents at the loop boundaries to be assumed to be smooth. The downside is that the starting node often bisects a region that could be better described with a single curve or line segment. We therefore filter out the extra node whenever possible by merging collinear line segments (the most likely case).

In practice, it is very inefficient to consider every possible sub-range of points as a candidate curve. The space and time complexity of the dynamic programming algorithm is quadratic in the number of candidate nodes, and this number is potentially very large when the input is a dense bitmap mask boundary. So given a maximum number of candidate nodes K, we find the K input points with the highest curvature and mark them as candidates, since such corners are likely to mark the boundaries between separate curves.

The full recurrence relation for our energy function E_{ijk} is

$$E_{ijk} = \min_{j',k'}(E_{ij'k'} + e_{j'jk'k}), i < j' < j \tag{3.8}$$

where i and j are the start and end points of the considered range of points, and k is the type of curve fit to the range ending at j (so E_{ijk} is the energy of a sequence of curves whose last curve has type k). Note that the sub-range energy $e_{j'jk'k}$ depends not only on the starting and end node

indices, but also the *type* of the previous and current curve. Because k also defines whether a Bézier curve should use the fixed (precomputed) tangent at its last endpoint, this allows us to define the rules (as defined in earlier in the section) governing the behavior of neighboring curves, including the angles between their tangents where they meet.

We also wish to capture both sharp corners and smooth transitions in our solution. Because the input may contain artifacts, and furthermore is not representative of the final boundary that adheres to all the desired constraints, we do not detect sharp corners in the input as is usually done in vectorization, but rather incorporate the choice into our curve fitting algorithm to encourage sharp corners that lead to a better fit to the data. To allow sharp and smooth tangent behavior at nodes, neighboring curves must be able to agree on either G_1 continuous or unconstrained tangents. To make this possible, we parameterize right end tangent behavior using two types $k = 1$ and $k = 2$, where each type is a cubic Bézier curve with a constrained and free right end tangent, respectively. This way, we can ensure that each curve's left tangent behavior matches the previous curve's right tangent behavior when computing E_{ijk}. Finally, we can filter out sharp corners by requiring that a curve with type $k = 2$ must meet the next curve with an angle greater than $\alpha_{\min}$. Because an unconstrained curve will always fit with smaller MSE error, cases where the tangent angle is $> \alpha_{\min}$ represent unconstrained curves that differ significantly, and therefore should be preferred in the interest of accuracy. We set the curve cost $c_1 = (\frac{D}{1000W})^2$ where W is the width of the input boundary's mask, and $\alpha_{\min} = 10^\circ$ in our experiments.

The constraint segments used in the segmentation stage are also used in curve fitting; we wish for the curve to "snap" to these segments wherever they are near enough, or if it would result in a simpler solution. We incorporate these constraints into our curve fitting algorithm by first identifying points in X within τ_c of a constraint segment, and forcing any segment fit to a range containing these points to be a straight line segment. The result is shown in Figure 3.8 (a) and (b); the dynamic programming fit is guaranteed to produce line segments where they are needed, and

does so while still fulfilling its other objectives of continuity and simplicity.

Post-processing Having guaranteed straight edge segments in the *vicinity* of constraint segments, we project the nodes bordering lines near these constraint segments to the exact lines of these constraints to obtain a fabricable solution (Figure 3.8 (c)). Neighboring curves are modified so as to preserve their tangent angle with the displaced lines. It is not always possible to ensure consistent tangent behavior in the above framework; transitions between curves and line segments are troublesome since the latter lack the degrees of freedom to adhere to the pre-computed tangents used for smooth transitions. The inherent order of curves considered by the dynamic programming algorithm prevents curves from correcting for the behavior of subsequent neighbors. We therefore apply an additional smoothing step in which corners below angle $\alpha_{\min}$ are made smooth by altering curve tangents. This step is only done if the resulting change to the shape is not too drastic; we approximate this change by the total displacement of Bézier control points, which we limit to τ_c.

We additionally find lines that are parallel/orthogonal (within angle α) and further align them. Constraint line orientations are left unchanged, and any edges nearly parallel to them copy their orientation, and likewise for nearly orthogonal edges. In many cases, this produces 90 degree angles in shapes where parts connect.

3.3 Experimental Results

We tested our algorithm on seven carpentered objects of varying complexity, i.e., varying numbers of parts with some objects exhibiting more difficult features, such as non-axis aligned parts (the diagonal bookshelf and bookholder), shapes with holes and curves (the stool and tray), non-orthogonal connections, and one which deviates slightly from our model assumptions in the form

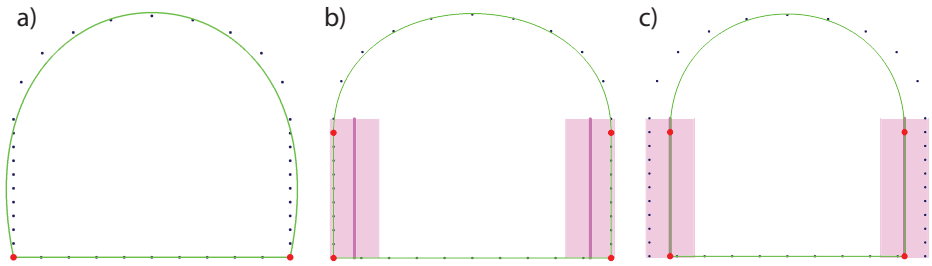

Figure 3.8: Left to right: Polycurve fitting without and with constraints. Input points are shown in black, output curves in green. Constraints are shown as purple line segments; the pink region represents the tolerance of the constraint. In (a), the unconstrained solution fits a single curve to the upper points, while in (b) the constrained solution adds line segments which preserve smooth transitions, allowing them to be trivially displaced to adhere to the constraints while preserving smoothness (c).

of smoothed corners and grooves in the sheet plane, as well as having some highly occluded part sheet planes (complex stool). We obtained fabricable reconstructions for six of the objects, and discuss the seventh as a limitation in the final section.

Experimental Setup We photographed our objects with a hand-held Google Pixel 3 camera in two distinct, well-lit indoor locations with a variety of backgrounds (different rugs, etc.). For each model, we took between 30 and 70 photos from viewpoints facing the object and situated approximately on a hemisphere around it. We used RealityCapture [Cap] to recover camera poses and semi-dense point cloud reconstructions which requires some minor user input to select the reconstruction region to isolate the object one desires to capture, in particular to omit the ground plane.

Qualitative assessment Figure 3.9 shows results for six of the models. In all cases, there were faces of the model that were either missing or incompletely represented in the point cloud (second row), such as the unseen undersides of the top of the stool and nightstand, as well as faces that are less well-textured or in shadow, as with some parts of the bookshelf. For all six models shown, we

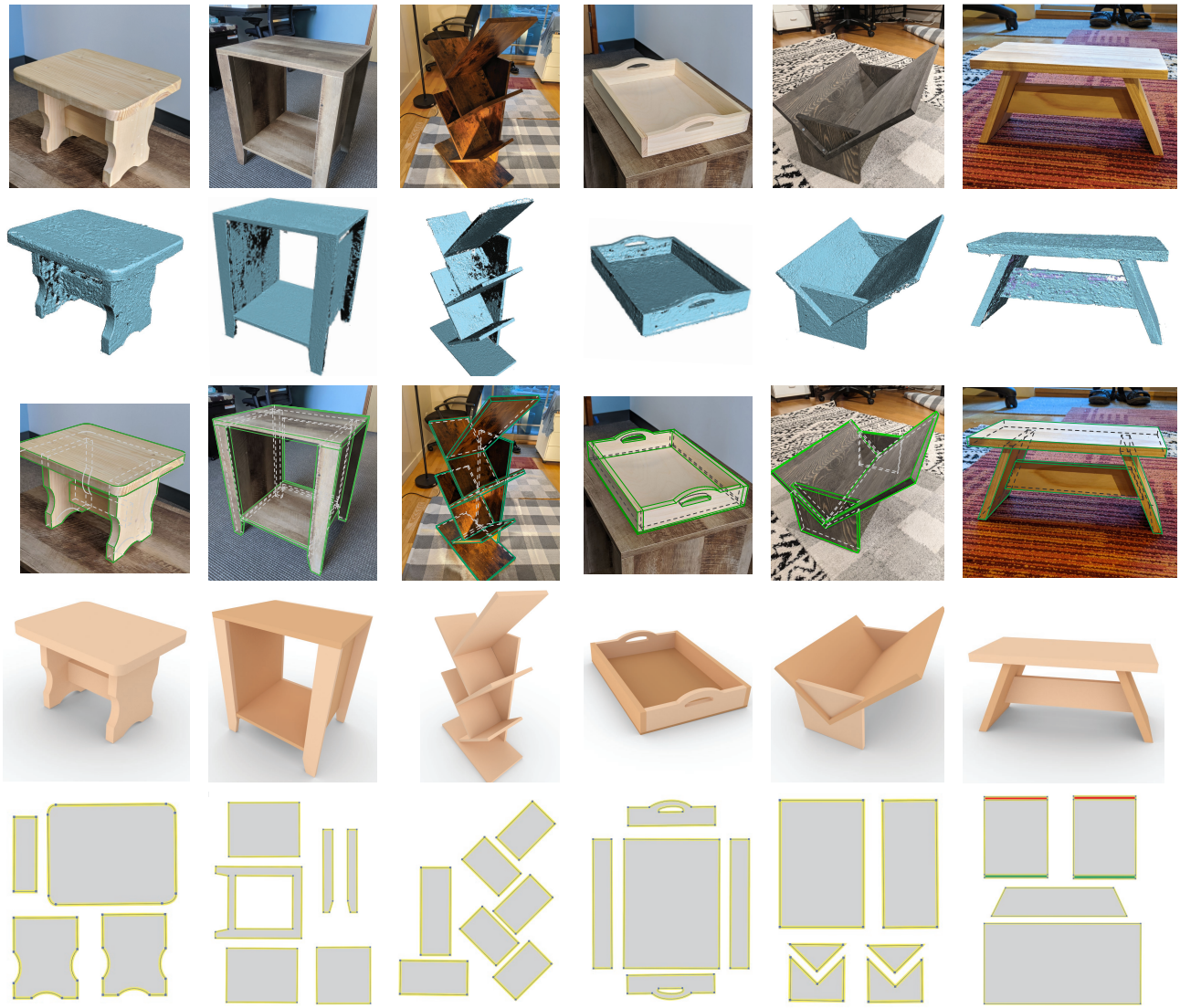

Figure 3.9: Results from six example models. Left to right: stool, nightstand, bookshelf, tray, bookholder, tilted stool. The results shown, from top to bottom: representative input image; reconstructed point cloud; our result superimposed on the input image; a clean render of the result; and the 2D cut paths, with curve endpoints shown as blue dots. Bevel cut surfaces are displayed in red and green, indicating that the surface normal faces into or out of the page, respectively.

were able to generate fabricable results, shown in the fourth row. In the third row, we superimpose the reconstructed CAD model on the input image showing how faithfully our reconstructions fit to the observed object. Some minor failure regions include the horizontal pieces of the nightstand, which are slightly thicker and, in the bookshelf, two of the parts are slightly translated from their original position. In the final row, the cut shapes for all the parts in the output are shown as combinations of line segments and Bézier curves. The nodes, shown in blue, indicate the transitions between curves and may either be smooth or sharp corners. For the most part, these simplified curves are consistent with the input, but in the case of the tray, the smooth transitions on either side of the rounded handles, as well as on the bottom of the handholds, are sharpened. The cut shapes also reveal that the back side of the nightstand was detected as a single piece. In fact, that piece is made up of 4 smaller pieces, as shown in Figure 3.13 (a). Though we do not decompose detected parts based on detected seams, this result can be "fixed" by adding cuts after the fact. To evaluate the importance of the methods in our technique, we also show some results with various parts of our pipeline simplified in Section 4 of the supplemental material.

Quantitative evaluation To measure the accuracy of the final fabricable, simple CAD model to the original object, we use the RMSE distance of the point cloud to the model surface, multiplied by $1/D$ for scale independence. As shown in Table 3.1 for the same five models in Fig 3.9, this error is on the order of 0.4% of the model width, indicating that in terms of geometric displacement, our reconstructed shapes remain true to the original objects.

We also measure the number of incorrect connections, i.e. extra, missing, or misclassified connections. Among the five models, only the nightstand misses some connections; the four parts making up the back surface are detected as one part (see Figure 3.13 (a)).

Another important measure of accuracy is in the representation of the cut paths themselves; smooth curves and sharp corners should be reflected in the final result. Two of our models contain

model	#images	#parts (actual)	#parts (found)	RMSE (% of D)	identification	assembly	segmentation	curve fitting
stool	68	4	4	0.433	3s	0.3s	267s	129s
nightstand	37	10	7	0.477	25s	25s	498s	70s
bookshelf	37	7	7	0.204	8.7s	4.7s	507s	31s
tray	30	5	5	0.259	2.5s	26s	54s	23s
bookholder	42	6	6	0.406	5.2s	27s	131s	23s
tilted stool	48	4	4	0.432	15s	.038s	199s	20s

Table 3.1: Statistics for five input models, showing the complexity of each model, the geometric accuracy of the reconstruction, and the time taken in each stage of the algorithm

smooth curves: In the case of the stool, these features were handled without problems; for the tray, some additional corners are present in the handle holes, as well as on either side of the arches.

3.3.1 Evaluating Design Choices

To gauge the necessity of some of the stages of our pipeline, we discuss how disabling or simplifying them impacts the quality of results.

Joint Image-Based Segmentation Figure 3.10 shows the result of skipping image segmentation, and directly applying curve fitting to the point set boundaries from Section 4.2.2. Because regions of the model with less visibility, such as cavities and undersides, are often missing from the point cloud, the final result contains gaps. Compared to the full pipeline, we lack the ability to expand part shapes into regions of similar texture, which is a method to close gaps such as this.

Using Multiple Views in Segmentation We also show results from using only a single image per part in the segmentation phase in Figure 3.11. Without multiple views to disambiguate foreground and background, similarly-colored surfaces become erroneously associated with the part shape, leading to an artifact-laden result. In general, material and lighting conditions can

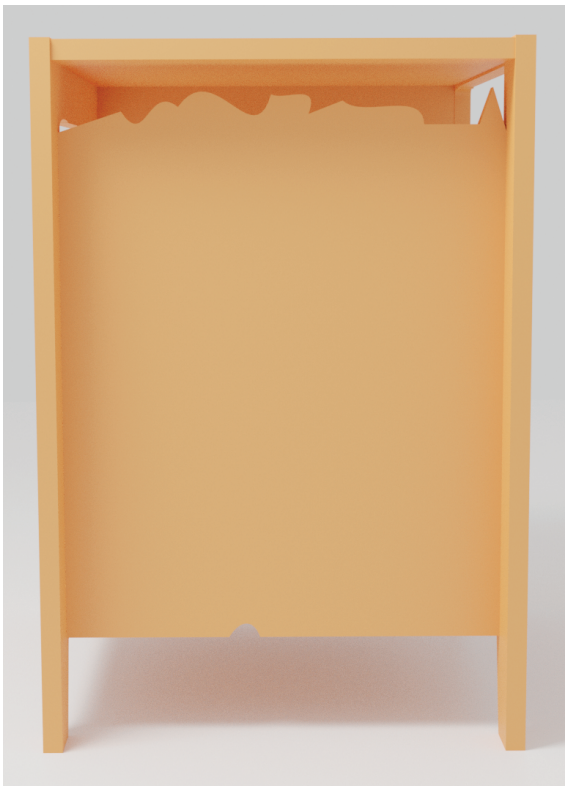

Figure 3.10: Without the image segmentation step, deficiencies in the input point cloud persist in the final result.

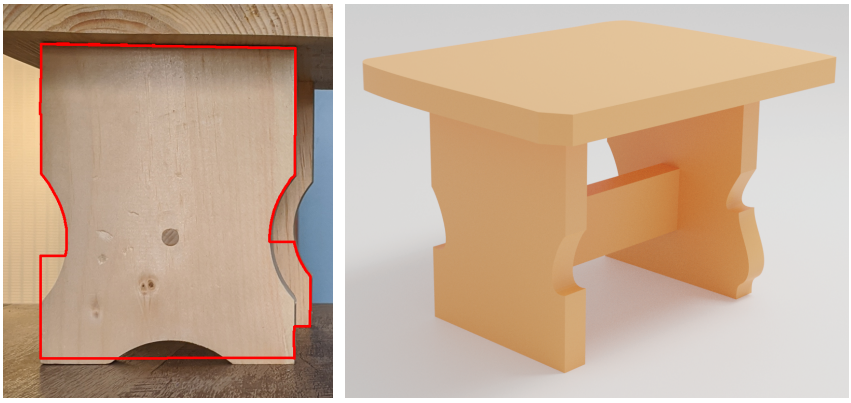

Figure 3.11: Results from using only one image per part in the segmentation phase. Left: segmentation result for one part, with the segmented shape superimposed in red on the corresponding image. Right: The full result.

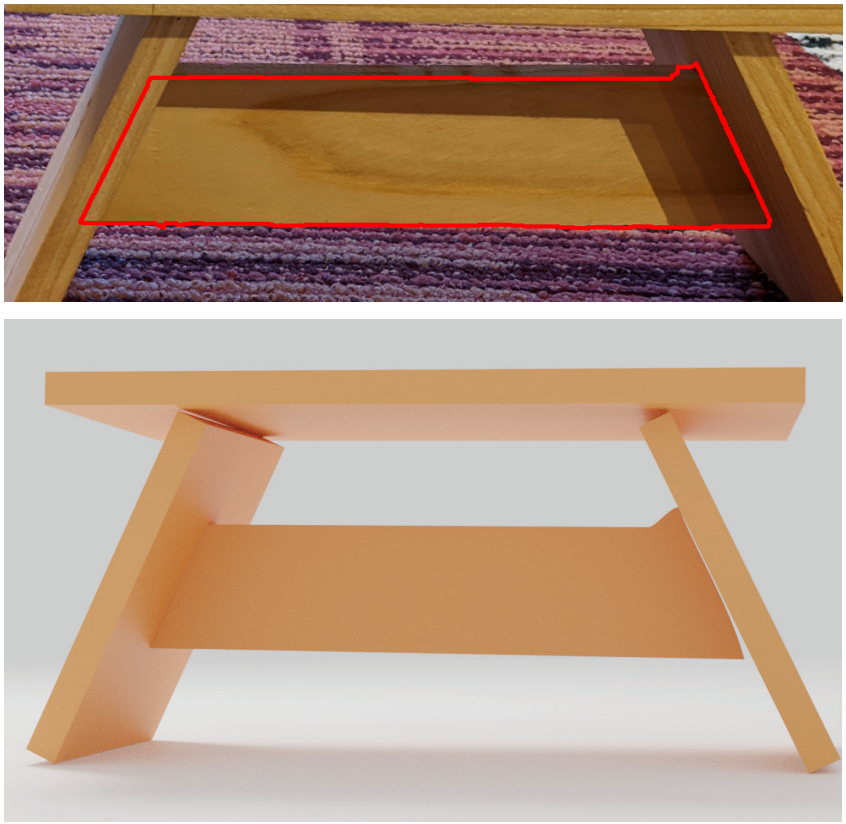

Figure 3.12: In the presence of some segmentation artifacts (top), unconstrained curve fitting produces an incorrect result (bottom). Note that the supposed contact between the central piece and the right leg is curved.

make discerning part shapes from certain views difficult; averaging the segmentation energy over multiple reprojected views exploits the view-dependence of pixels not belonging to the part, since similarly-colored background surfaces are less likely to interfere with the same pixels in every view.

Constrained Curve Fitting The constraints in the curve fitting stage straighten curves in the vicinity of connection contacts, effectively flattening nearby artifacts arising from the segmentation stage (which occur due to the poor visibility at some junctions). In our tilted stool example, one

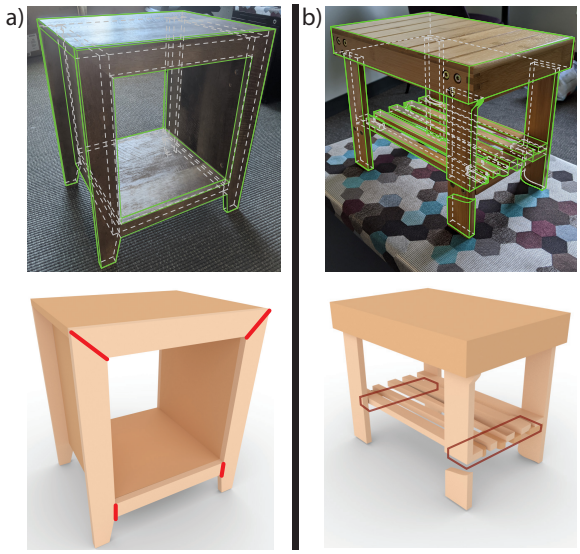

Figure 3.13: Limitations of our ability to capture part some part structures. In (a), the nightstand contains four co-planar parts that are merged into one in our result; the seams where they should be cut apart are highlighted in red. For the complex stool in (b), failure to detect a complete set of part candidates causes the result to be invalid.

such artifact occurs due to heavy shadowing (see the top of Figure 3.12), but our constrained curve fitting gives a clean result (see Figure 8 in the main paper). The bottom of Figure 3.12 shows the result without considering constraints in the curve fitting stage. The artifact persists; furthermore, the contacts between all parts are imperfect. Amending the contacts purely in a post-process gives rise to new challenges, for without enforcing straight segments in the vicinity of contacts, there may not be a single part of the boundary curve that can be "snapped" to the surface. For example, the cut edge on the right of the central part has been chosen as part of a longer, continuous curve.

3.3.2 Limitations and Future Work

A key feature and a notable limitation of our method is its reliance on image evidence. The images enable the MVS reconstruction, corner assembly disambiguation, and recovery of nice cut paths. However, we can only reconstruct what we observe sufficiently. If a model has structure that hides

some parts from view, it can be difficult or impossible for our method to accurately reconstruct the model. In the model in Figure 3.13b ("complex stool"), the parts highlighted in red are occluded by the boards directly above them, causing them to be detected as disjoint, floating pieces; the result is not fabricable or even connected. This might have been addressed by observing the underside of the red parts (and straightforwardly handling type 3 connections), but the underside was out of view. In future work, it would be interesting to explore the use of additional priors or learned semantics to reconstruct objects with incomplete observations. For instance, if a model could not be assembled only with detected parts (perhaps due to floating, unconnected pieces), we could potentially hallucinate new structures. Another way to address the method's reliance on image evidence would be to extend the system to enable real-time capture by incorporating new observations incrementally, and guiding the user to capture new images that resolve ambiguities or missing geometry in the model.

Our method has a number of thresholds, cost terms, and other parameters that affect the final quality of the result. Many of these parameters were related to object dimensions, to limit dependency on object size. The method, particularly the segmentation, and curve fitting stages, are sensitive to the choice of some of these parameters. We determined good values by experimentation up front and then used the same parameter values (or proportionality constants for parameters related to object scale) for every model. In the future, it would be desirable to to detect some of these parameters adaptively to improve the robustness of our method. For instance, some specific stock thicknesses are more common as building materials, so part thicknesses could be used to detect the true object scale. This can further be used to ensure that parts can be made from readily available materials.

It might also be worth exploiting alternative or complementary features derived from the input data. For example, we could use all seams in the wood as evidence of cuts to find connected co-planar parts as in Figure 3.13a. To incorporate arbitrary seams robustly, we would need to

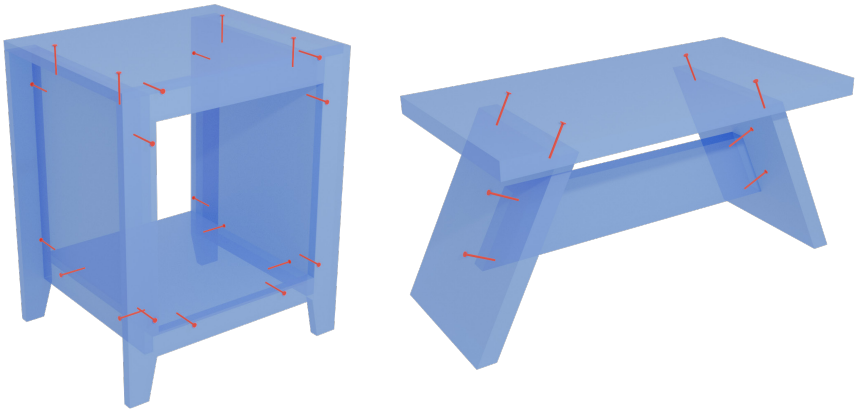

Figure 3.14: Examples of procedurally generated connectors used for assembly.

account for other texture features, such as ever-present wood grain, that could be mistaken for cuts. In addition, man-made objects often have symmetries and repetitions, which could be exploited both to aid in detection and to further regularize the model. Lastly, in addition to planar surfaces detected from point clouds, we could also use the detected curved (cut) surfaces to guide segmentation. This would also help minimize the bias towards straight lines observed in the tray example.

3.4 Beyond Carpentry

In this chapter, we presented a method for recovering accurate representations of built, carpentered objects from a set of photographs by working within the space of the fabrication process itself. This representation is both highly expressive and subject to real world constraints, as the process describes models that can be physically realized, making it a good candidate for solving inverse problems in 3D reconstruction. Given enough images covering a subset of the surfaces, our solution in the carpentry domain can recover the parts and connections that comprise captured real world objects, complete with concise contours describing the cut paths of each part, making

it easy to edit with CAD software to create design variations.

We have seen that by working within a specific fabrication domain, we can leverage the constraints of the associated processes to reconstruct complete designs from measured data; furthermore, these constraints allow us to more fully utilize that data—for example, interpreting our images as depictions of part shapes within a plane—allowing us to achieve this with only a partial set of observations. It might be noted, however, that many of the priors we leverage in this work are not unique to carpentry at all; right angles and parts with uniform thickness are geometric features favored across many domains. Should it not therefore be possible to design a system to reconstruct fabricable objects in general?

Chapter 4

View2CAD: Reconstructing View-Centric B-Reps from Single RGB-D Scans

In the previous chapter, we detailed a method for reconstructing carpentry designs from incomplete photographs. These designs, in a broad sense, can be classified as computer-aided design (CAD) representations—the piecewise curve representation we use for the parts translates directly to standard formats such as the Scalable Vector Graphics (SVG) format (which is how we exported and visualized the part layouts in the bottom row of Figure 3.9). CAD representations are highly versatile, and the geometries they represent are not limited to any particular fabrication domain. Consequently, most man-made objects are designed in CAD software, so that CAD geometry can be found everywhere. While CAD models are not constrained by any particular fabrication process, they capture a more fundamental set of constraints. One of these is the need to be able to interpret the collection of all parametrically-defined curves and surfaces as a coherent object. There are also various design practices common to many human-made designs, such as a preference for right angles. It would therefore be of interest to incorporate these fundamental constraints when reconstructing the geometry of man-made objects, as they encompass a broader

class of objects while still providing useful priors, potentially giving way to techniques that are robust to incomplete data in the same way as the domain-specific approach examined in the previous chapter. In this chapter, we will identify a key set of universal constraints that allows us to reconstruct precise 3D CAD models of arbitrary man-made objects in the wild using imprecise measurements from only a single view of an object.

3D CAD geometry is frequently represented using Boundary Representations (B-rep), where shapes are expressed as parametric primitives along with their topological relationships (see Chapter 2). Existing methods for reconstructing detailed B-rep geometry typically take complete point clouds as input [BCF+18], while image-to-CAD approaches often rely on retrieval methods, which lack accuracy [ISS17, BI17, KLL17, KALD20]. Our objective is to bridge this gap by developing a method to derive CAD models directly from images. In this work, we take as input a single view with color and depth (RGB-D) information, motivated by the ease with which such data can be captured in industrial settings or robotics applications.

Given that we seek to reconstruct accurate B-Rep geometry from only a single view of an object, it is necessary to limit the scope of what we should reconstruct; we formulate this as the problem of reconstructing only the *observed* geometry. This requirement does not map well to standard B-rep formats, however, which require complete topological information to define the boundaries of all parametric geometry. To address the challenge of representing what we can see of CAD surface geometry, we define a *view-centric B-Rep*, or **VB-Rep**, as a modification of the standard CAD B-Rep, which incorporates additional structures to account for the limits of visibility while maintaining the precision of B-Reps. Furthermore, the visibility boundaries also encode the uncertainty of geometry reconstructed from a single-view, allowing us to reason about which elements belong to the underlying CAD model and which arise from visibility artifacts, such as occlusions or silhouettes.

Reconstructing a VB-rep from an RGB-D image of a CAD model presents significant challenges.

The first is the lack of full topological information. Topological information defines validity constraints that can support B-rep reconstruction, but this information is not fully available in a single view. The second challenge is that the topological information we do have is ambiguous. The uncertainty about which of the features we can discern in the 2D image correspond to features of the original B-rep and which are introduced by partial visibility adds layers of complexity to the reconstruction process.

Our solution is built on two key insights. The first is that there is local information that is independent of the topology, which can be directly extracted from images. The second is that by treating the visibility information encoded in the VB-Rep edges as a measure of geometric uncertainty during the reconstruction, we can systematically use the elements that we have reconstructed with high confidence to refine and improve our reconstruction in an iterative optimization process.

From the first insight, we propose to recover information pertaining to visible faces, including parametric surface types and orientations. We can accomplish this while simultaneously detecting the regions of the image belonging to each of the visible faces by treating the task as a *panoptic segmentation* problem. Building on the second insight, we leverage this information to iteratively reconstruct the edges that bound the parametric surfaces and align the surfaces to these inferred bounds. By identifying edges that belong to the CAD B-Rep rather than being artifacts of silhouettes or occlusions, we can then refine the fits of surface parameters by constraining these edges to align with neighboring surfaces in the view space. We iterate on these two steps, using the optimized parameters to progressively update the set of B-Rep edges we consider. This edge-based guidance, together with the orientation priors we uncover through segmentation, serve as a strong CAD geometry prior for reconstructing well-aligned, topologically valid surface and edge elements in the face of incomplete or uncertain range data.

We demonstrate the effectiveness of our method on a collection of real objects captured with a

commodity depth sensor, as well as a synthetic dataset of photorealistic RGB-D images. Through ablation studies, we highlight the importance of the various types geometric guidance we extract through vision and optimization.

4.1 Background

Reconstructing B-Reps from partial observations Existing work in robotics has explored reconstruction of boundary representations from single depth images. Most relevant to our work, [SH16] reconstruct planar B-Reps, and further present a method of fusing together reconstructions from multiple views, enabling incremental reconstruction from a stream of inputs. To this end, the authors also present a SLAM system built on pose estimation of partial, planar B-Reps [SH17]. While their method is highly performant and suitable for on-line reconstruction, at each frame they segment the depth into planar elements by a region-growing technique that is sensitive to the choice of threshold parameter, which must be chosen to avoid either over-segmenting or oversimplifying, as well as requiring the quality of the depth map to be sufficient for discerning all the primitives.

By contrast, our proposed work strongly leverages RGB images and deep learning to more confidently segment the input into primitive surfaces, as well as predicting geometric constraints to aid in reconstruction. Our method can therefore detect small rounded corners, thin ridges, and other details too minute to locate or classify from depth alone, unaffected by common faults in depth estimates such as holes. Furthermore, in addition to planes, we reconstruct B-Reps consisting of the full range of surface primitives found in typical CAD models.

Sketch to 3D Somewhat relevant to our problem, [PMKB23] addressed the problem of reconstructing CAD-like representations from hand-drawn bitmap sketches. Their method uses deep

learning to predict per-pixel depth and primitive segmentation maps, followed by primitive fitting and geometry optimization steps to produce a final triangulated mesh that adheres to the sketch as closely as possible while retaining the sharp features of the CAD model. Because they take a clean line drawing as input, the boundaries in the drawing can be used directly (with some processing) as the final instance boundaries between primitives. In our setting of RGB-D input, these boundaries are not provided and must be inferred. Furthermore, since the sketch is an imperfect rendition of a 3D shape, the final mesh only approximately adheres to parametric surface primitives while balancing faithfulness to the original sketch. However, the authors of [PMKB23] define a geometric optimization objective with some similarities to our own—most notably, they define a terms which encourage strokes to lie at the intersections of geometric primitives, and detect the difference between intersection and occlusion contours.

Our work, in contrast, is the first to adapt the CAD reverse engineering problem to the setting of single-view RGB-D captures with low-precision depth sensors.

4.2 View-centric B-Reps

Our goal of reconstructing a CAD model from a single view can be framed as creating a partial twin of some unknown "true" CAD model, in which only the visible features are present, but represented with the same fidelity as in the original.

In solid modeling, **boundary representations** (B-Reps) are the standard representation for CAD models. A B-Rep is a heterogeneous graph representation consisting of faces, edges, and vertices, and the boundary relations between these elements. The faces, edges, and vertices are associated with parametric geometry functions for surfaces (2D), curves (1D), and points (0D), respectively, which can represent smooth geometry with infinite precision. Crucially, parametric geometry has no boundary and the domain on which each element is evaluated must be inferred

from neighboring boundary elements (points bounding curves, curves bounding surfaces). One implication of this is that the edges bounding a face must form a set of closed loops, so taking a naive subset of the elements in a B-Rep can break this structural constraint that defines the bounds of all the primitives.

However, we observe that in a single view of an object, the regions of the view comprised of different visible surface primitives naturally give rise to a graph of edges, in the form of the boundaries between the different observable primitives. By construction, these edges form a *planar* graph partitioning 2D space into bounded regions. A subset of these edges map to real edges of the B-Rep, while others are visibility artifacts, but all of them define, in one way or another, the visible limits of the surfaces they bound (see Figure 4.1).

Motivated by this strong structural prior, we define a modified geometry representation, which we call a *View-centric B-Rep or **VB-Rep***, that augments the standard B-Rep by incorporating the concept of visibility into the boundary graph. Given a B-Rep and a view, we can create a VB-Rep where some of the entities originate in an underlying **true B-Rep** $\mathcal{B}$ and others from visibility considerations. In the rest of this section, we define a VB-Rep with respect to a known B-Rep, and in the rest of this paper we discuss an algorithm for reconstructing it from an RGB-D image.

Similar to a B-rep, a VB-Rep is a graph structure consisting of faces bounded by edges bounded by vertices, but while all faces are geometric entities from $\mathcal{B}$, the edges and vertices of a VB-rep are classified according to the various sources of visual boundaries. Edges can be classified as **intersection** edges, **visibility** edges, **silhouette** edges, and **occluded** edges. Intersection edges are edges known to be present in $\mathcal{B}$, because they coincide with the intersection of two visible B-rep faces. This also means that the edge geometry can be defined in terms of the surface intersection. Visibility edges are those that can be explained by the visible limits of the underlying primitive. Occluded edges are caused by a surface nearer to the camera occluding the underlying surface; these always coincide with an occluding silhouette or visibility edge (e.g. edge (b) and (c) in Figure

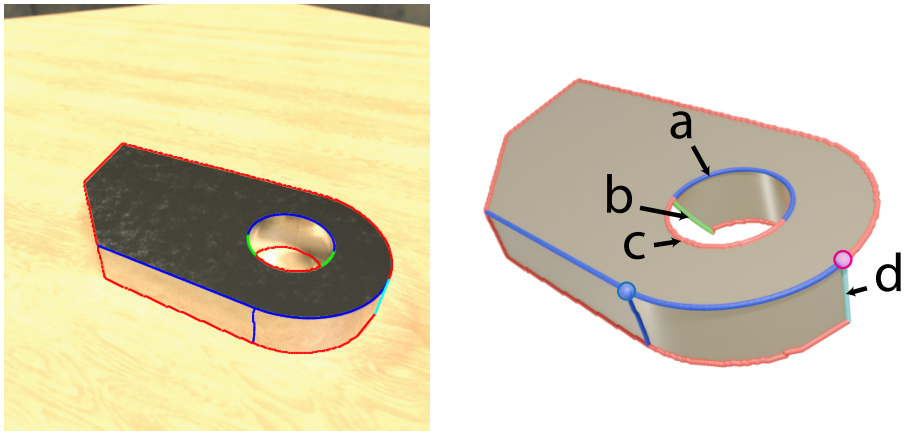

Figure 4.1: (Left): The visible surfaces subtend regions in the 2D image whose boundaries form a planar graph. (Right): The 2D edges of this graph map to edges in the VB-Rep. These VB-Rep edges can have different types based on the visibility information they encode; (a) blue denotes intersection, (b) green denotes occluded, (c) red denotes silhouette, and (d) cyan denotes visibility. The blue circled vertex is a triple intersection, and the magenta circled vertex is an intersection-visibility vertex.

4.1). Silhouette edges are edges are all other edges not explained by the above visibility phenomena, for which the only plausible explanation is the existence of an unseen surface intersecting along the observed edge. In our method, we approximate all edges using densely-sampled piecewise line segments (due to the difficulty of representing geometries with only one constraining surface), and classify our boundaries at the level of these segments.

The vertices in our VB-Rep possess the same classifications, with the caveat that a vertex may be both an intersection vertex and a visibility vertex (see the magenta point in Figure 4.1), and vertices with three intersecting surfaces (blue circle in Figure 4.1) are **corners** which correspond with vertices in $\mathcal{B}$.

To summarize, the four categories of visibility phenomena give us varying degrees of certainty that the edges and vertices originate in the underlying B-Rep, while intersection edges and corner vertices in particular are true edges and vertices in $\mathcal{B}$.

As with regular B-reps, each element in the VB-rep graph is associated with geometry. For

faces, we use parametric surfaces as in classic B-reps as these can be fitted by a reconstruction algorithm. In this work, we use five of the standard primitives: **planes**, **cylinders**, **spheres**, **cones**, and **tori**. Intersection edges are also associated with parametric curves defined by the intersection of two surfaces. The notion of an intersection curve is a representation supported by common B-Rep packages, such as Parasolid [Sie]. For the other types of curves, there is no analytic geometric representation like the intersection of two surfaces, so we fall back on our polyline approximation. Finally, vertices are represented as standard points.

This representation is useful for two key reasons. First, it enables visualization of partially reconstructed CAD models, as it preserves the validness of the bounds needed to evaluate the domains of primitives that comprise them. Second, this representation allows reasoning about the limits of the observable geometry that can be extracted from a single view of a B-Rep; in particular, it gives a measure of the certainty with which boundary elements (edges and vertices) can be considered part of the true B-Rep. This helps with CAD reconstruction, as knowing which edges arise from *intersecting primitives* also provides guidance for when and where surfaces should be constrained to intersect.

4.3 Overview

Our method takes as input RGB and depth images ($\mathcal{I}_{\text{rgb}}, \mathcal{I}_{\text{D}}$) of an object created via solid modeling, such that it possesses a **true B-Rep** representation $\mathcal{B}$. The depth capture parameters are considered known, so we obtain a frontal point cloud $\mathbf{P}_{\text{depth}}$ approximating the view-space geometry of $\mathcal{B}$. From the given viewpoint, we wish to reconstruct a VB-Rep $\mathcal{V}$ containing the visible parts of the geometry of $\mathcal{B}$, as defined in Section 4.2. Since our method should enable partial CAD reconstruction from RGB-D captures in the wild, it must also be robust to significant noise in $\mathcal{I}_{\text{D}}$.

Figure 4.2 illustrates our proposed pipeline. Our method has three steps: **image segmentation**,

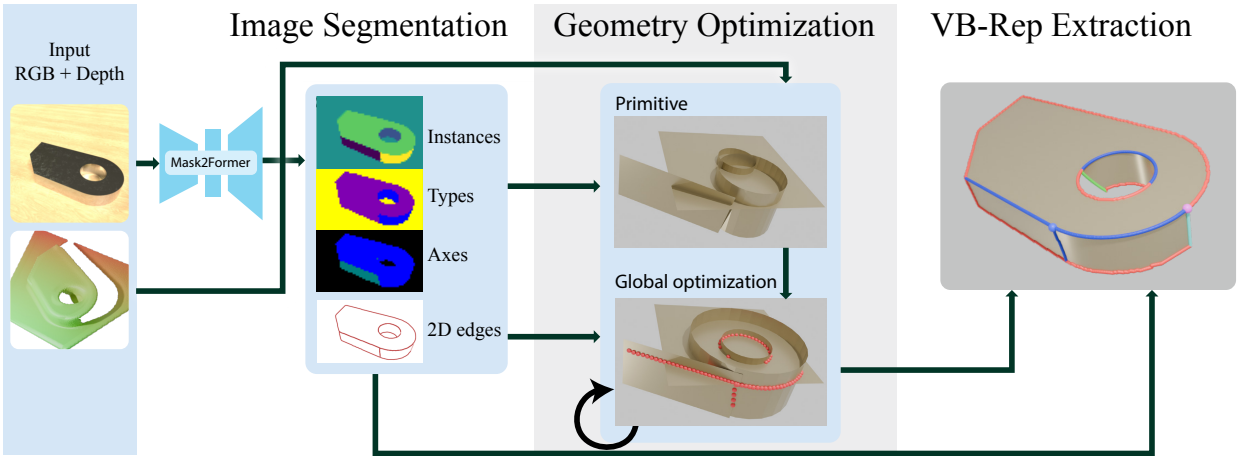

Figure 4.2: An overview of our pipeline. In the **segmentation stage**, we train a Mask2Former model [CMS+22] to perform panoptic segmentation of the RGB image into primitive instances along with classifying them according to primitive type and axis alignment. In the **geometry optimization** stage, we use the predicted primitive instance masks and types to fit primitives to the input depth point cloud, and globally optimize them to enforce consistency with predicted orientation and intersection constraints. Furthermore, we *iterate* this step to update our knowledge of which image features to use as guidance in the optimization. Finally, in the **VB-Rep extraction stage**, we use the refined surface primitives and edges to build a coherent CAD representation consisting of bounded surfaces, curves, and points, labeled according to visibility information.

geometry optimization, and **VB-Rep extraction**.

Image segmentation We aim to reconstruct VB-Reps with details potentially below the scale of the structure discernible in the noisy depth map, so we leverage the comparatively dense, high-quality RGB data to detect the visible surface primitives $\tilde{\mathcal{S}}$. Our key insight is that we can formulate primitive detection as a **panoptic segmentation** task over $\mathcal{I}_{\text{rgb}}$, and use a deep image segmentation model to simultaneously predict and classify the various surfaces visible in the image.

We could take the approach of [LSD+19b] and others, and predict the types of each surface primitive so that we can eventually fit those surfaces to $\mathbf{P}_{\text{depth}}$ according to standard least-squares fitting techniques. However, we observe that because such a method relies solely on $\mathbf{P}_{\text{depth}}$ to infer the geometry, such fits are highly sensitive to noise and errors in the depth, particularly for surfaces corresponding to small image regions of the image, like fillets, extrusions, or faces viewed from extreme angles (see Figure 4.3)

We address this with the observation that a significant number of features in CAD B-Reps (and man-made objects in general) are aligned with the coordinate axes, and that this can serve as a strong geometric prior in the presence of uncertain depth estimates. To make use of this prior and to extract more geometric guidance from the deep network, we further train our segmentation model to predict axis alignment constraints alongside surface type, since axis alignment can be treated as a classification task. This allows us to leverage robust network architectures [CMS+22, CSK21] to perform both the task of inferring the structure of surface primitives comprising $\mathcal{V}$ and the geometric constraints governing these surfaces. To leverage these methods for our surface-level segmentation task, we synthesize an image dataset with ground truth labels for view-space reconstruction of CAD objects (see Section 4.5).

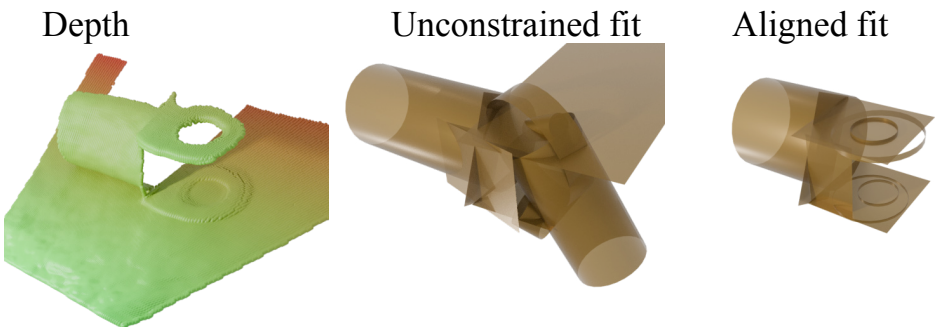

Figure 4.3: Fitting primitives to a noisy point cloud. Fitting primitives individually (middle) fails to recover the correct orientations especially in the thin cylindrical borders where point supervision is sparse. Meanwhile, globally aligned fitting (right) allows the thin structures to be correctly oriented with the same level of noise in the points.

Geometry Optimization Given the labeled segmentation and points $\mathbf{P}_{\text{depth}}$, we can now optimize for the parameters of visible surface primitives given the predicted primitive types and axis alignment. We can formulate the aligned primitive fitting problem as a nonlinear least-squares optimization problem of minimizing point-primitive residuals. However, the parameters of surfaces in a true CAD model are not independent; all geometry must be be consistent with respect to the model topology, which specifies the boundary relationships between surfaces, curves, and vertices. Since our primitives must be consistent with the topology of $\mathcal{V}$, it is necessary to consider not just surfaces, but their boundary curves during this optimization. In particular, for our view-space reconstruction problem, this means that visible surfaces in $\tilde{\mathcal{S}}$ must **intersect** along intersection edges in $\mathcal{V}$. This motivates us to incorporate this intersection constraint into our optimization objective.

The challenge is that we do not know which edges in the segmentation map correspond to the edges of the B-Rep, and which are visibility artifacts. Therefore, we propose an iterative solution that initializes the geometry using only local surface information, and using these refined geometric surfaces, improves our estimate of which edges belong to the set of constrained intersections in the underlying B-Rep. We can then incorporate these new intersection constraints into our

optimization to further refine the geometry, and repeat this process until convergence.

VB-Rep extraction The final stage in our method is reconstructing a VB-Rep $\mathcal{V}$ from the set of optimized surfaces and discrete edges from the previous step. Due to the inherent uncertainty about how to interpret our 2D image observations as 3D geometry, observed surfaces can still have multiple different 3D interpretations—even when the surface primitive parameters are known (as shown in 4.7 (c) by the differently colored surfaces). To resolve this ambiguity, we employ a **surface-aware wireframe extraction** algorithm to determine the optimal configuration of 3D boundaries that comprise $\mathcal{V}$ subject to the segmented 2D boundaries and optimized surface primitive parameters. This process provides the *3D geometry* and *types* of all of our edges; based on where the 3D boundaries lie with respect to their adjacent surfaces, we can decide whether they are occluded, visibility, silhouette, or intersection edges.

4.4 Method

4.4.1 Primitive Segmentation

The low-precision depth maps obtained from near-field captures of small objects lack the detail necessary to discern precise geometry. We therefore train a deep image segmentation model to infer the discrete structure and geometric information about the VB-Rep $\mathcal{V}$ from the input RGB image. Our goal with segmentation is to directly detect and classify surface primitives, as well as infer orientation constraints. The shapes and connectivity of the segmentation boundaries also indicate where visible surfaces intersect. The surface types, orientations, and boundaries detected in this segmentation step are used in the subsequent geometry optimization steps to align the surfaces; furthermore, the image-space adjacency of the segments serves as the basis for the final structure of our reconstructed VB-Rep.

Our goal can be stated as a panoptic segmentation problem, where we must identify all the distinct visible parametric surfaces $\tilde{\mathcal{S}}$ present in the view, as well as classify them. As previously discussed, we want to classify not just the primitive type, but the orientation constraints for each primitive in order to incorporate the prior knowledge that man-made objects tend to exhibit axis-aligned features. This can be treated as a classification into the four categories of **(X, Y, Z, or unaligned)** consistently across all the primitives. Note that the exact orientation of the axis-aligned coordinate system will be optimized jointly with the geometry in the optimization step.

We extract this rich information from our RGB images by training a state-of-the-art deep panoptic segmentation model Mask2Former [CMS+22] on our synthetic dataset of rendered CAD models. We use the `maskformer2_R50_bs16_50ep` variant of the model.

We observe that deducing the orientation of a surface's underlying primitive axis is interdependent with knowing the primitive type. Therefore, we find that the predicting axis alignment constraints and surface primitive types should be treated as a *joint* problem, which we achieve by redefining the predicted type labels **T** to denote a primitive type with one of the four orientation classes, for a total of 17 possible types. For more details on the dataset used for training the model, see Section 4.5.

We take extra steps to ensure that these edges are at a quality that they can be used as geometric guidance; we find it important to train the model with an explicit background class in order to obtain high-quality mask borders at silhouette boundaries, and dilate the resulting K masks in order to fill any gaps between them, so that they can be compiled into a single segmentation map with every pixel labeled according to a specific surface, or background.

Wireframe As defined in Section 4.2, all surfaces in the VB-Rep are bounded by the limits of their visibility from a given viewpoint. The final set of VB-Rep edges, which we refer to as the

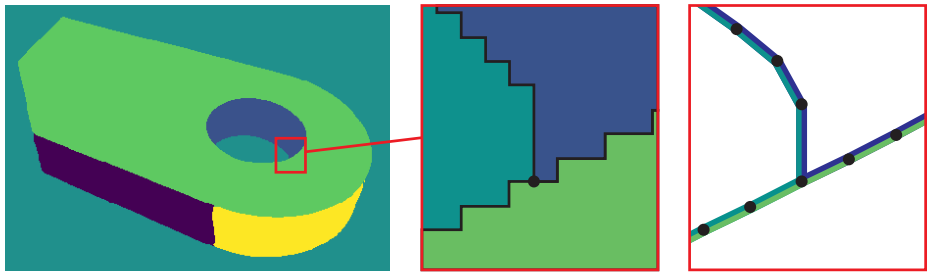

Figure 4.4: Given the initial instance map, we extract pixel-wise contours, then infer a simplified edge graph with adjacency information.

wireframe, should therefore recreate the 2D planar edge graph formed by the segmentation map when viewed from the reconstruction viewpoint. To this end, we extract this planar edge graph as the basis for our final wireframe, as illustrated in Figure 4.4. Given our dense segmentation map, we extract the set of boundary contours on the pixel grid between different instance labels, and compile these contours into a **2D edge graph** $\mathcal{E}$ represented by the graph $\mathcal{E} = (\mathbf{V}_w, \mathbf{E}_w)$, where the edges $\mathbf{E}_w$ are line segments between the 2D vertices $\mathbf{V}_w$. We further simplify this graph by keeping every N^{th} vertex along each contour. We associate each edge $E_i \in \mathbf{E}_w$ with its two neighboring instances N_i^1 and N_i^2 in the segmentation map, denoted by the dual edge colors in Figure 4.4.

4.4.2 Geometric Optimization

Constrained Primitive Fitting

Local fitting initialization Given the primitive instances and their predicted types and axis alignment, we can now use geometric optimization to reconstruct the 3D parameters of the detected surfaces that best fit both the predicted constraints and the input depth points. For each primitive $\mathbf{S}_i \in \tilde{\mathcal{S}}$, we have an associated segmentation mask $\mathbf{M}_i$, which associates with that primitive a subset $P_i \in \mathbf{P}_{\text{depth}}$. We use the RANSAC paradigm outlined in [SWK07] to compute

the parameters for planes, cylinders, spheres, cones, and tori from random minimal sets of points sampled from P_i. We keep the parameters with the greatest number of inliers among P_i within d_{ransac} of $\mathbf{S}_i$ and discard the outliers, obtaining a cleaned point set $\tilde{P}_i$. Some of the parameter estimates rely on point normals, which we compute from the gradient image of the depth map, estimated using a Sobel filter.

Global refinement We use the initial primitive fits to globally optimize all primitives with an alignment axis to fit the point cloud subject to the resulting constraints. Since the 3D axes themselves are unknown, we additionally optimize the orientation of an orthogonal coordinate frame in this global optimization step. Because the constraints require aligning groups of primitive axes with one of the three orthogonal frame axes, we represent the orientation of the alignment coordinate frame as a single axis-angle rotation $\mathcal{R}$ with three degrees of freedom. We re-parameterize all aligned axes as functions of $\mathcal{R}$.

The alignment optimization objective function is therefore

$$E_{\text{recon}}(\boldsymbol{\Theta}, \mathcal{R}) = \sum_i \sum_{p \in \tilde{P}_i} \text{dist}\left(\mathbf{S}_i(\boldsymbol{\Theta}, \mathcal{R}), p\right) \tag{4.1}$$

where $\boldsymbol{\Theta}$ are the non-constrained primitive parameters.

Iterative Intersection Refinement

Having minimized 4.1, we have arranged the visible surface primitives with respect to their probable orientations in the $\mathcal{V}$. However, these *orientation* constraints alone cannot ensure that the desired VB-Rep topology is feasible—the surfaces must also agree with the observed intersection boundaries, represented by the 2D edge graph $\mathcal{E}$.

Note that only a subset of edges in $\mathcal{E}$ should be interpreted as intersections, while others are

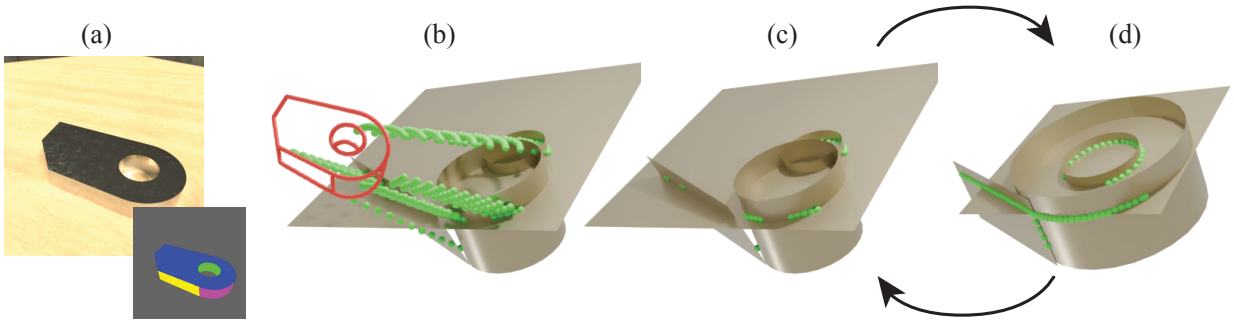

Figure 4.5: Overview of our intersection guidance optimization. Given the edges extracted from the segmentation map (a), we can minimize the distance of the 3D vertices $s_i\hat{\mathbf{q}}_i$ to their neighboring surface primitives along the projected view direction (b), with $\hat{\mathbf{q}}_i$ shown in red. Edge points whose minimal distance is below d_{int} are added to the intersection objective (c), shown in green. Finally, the surfaces and edge points are jointly optimized to minimize intersection distance while preserving the view projections of the edges (d); we also detect more intersection edge candidates (green) at this stage, so that we can repeat steps (c) and (d).

visibility artifacts. To mitigate this uncertainty, we adopt an iterative approach that alternately optimizes the subset of $\mathcal{E}$ corresponding to surface intersections and the parameters of all surface primitives that satisfy these intersections. For an overview of this approach, see Figure 4.5.

Because $\mathcal{E}$ exists in image space, it only specifies where the surface intersections should project onto the 2D image plane, so we supervise the primitive optimization in this step using a 3D wireframe that can vary freely in depth with respect to $\mathcal{E}$. Specifically, given the vertices $\mathbf{V}_w \in \mathcal{E}$, we can define a corresponding 3D wireframe as $\mathbf{V}_{3\text{D}}(\mathbf{s}) = \{s_i\hat{\mathbf{v}}_i,\ i = 1 \dots |\mathbf{V}_w|\}$ with scalar variables s_i controlling the depth, and initial vertices $\hat{\mathbf{v}}_i$ projected to the 3D image plane in camera space based on the known RGB camera intrinsics. We can then minimize the total distance between each vertex $\mathbf{q}_i \in \mathbf{V}_{3\text{D}}$ and all its neighboring surfaces $\hat{N}_i$, which we define as all surfaces neighboring the adjacent edges:

$$\hat{N}_i := \{N_j^k \mid i \in \mathbf{E}_w^{(j)},\ k = 1, 2\}$$

This leads us to our final intersection energy:

$$E_{\text{int}}(\boldsymbol{\Theta}, \mathbf{s}, \mathcal{R}) = \sum_i I_i \sum_k^{|\hat{N}_i|} \text{dist}(s_i \hat{\mathbf{q}}_i,\ \mathrm{S}_{\hat{N}_i^k}(\boldsymbol{\Theta}, \mathcal{R}))^2 \tag{4.2}$$

where $I_i \in \{0,\ 1\}$ is a label for whether each vertex lies at an intersection, telling us the subset of $\mathcal{E}$ to consider for this optimization, and $\mathcal{R}$, as before, controls the axis-aligned frame. In practice, we minimize $E_{\text{int}} + w_{\text{r}} E_{\text{recon}}$ with $w_{\text{r}} = 0.1$ in this stage to avoid overfitting to edge artifacts.

Since the intersection edges are initially unknown, we use an iterative approach to alternately select I_i and optimize primitive parameters. Specifically, we first attempt to minimize 4.2 with respect to *only* the s_i, assuming I is $\mathbf{1}$. For vertices whose final distance to both neighboring primitives is below d_{int}, we set I_i to 1. Finally, we minimize 4.2 with respect to both the wireframe depths $\mathbf{s}$ and primitive parameters $\boldsymbol{\Theta}$. This process is performed twice to capture intersection points initially missed (compare the points in Figure 4.5 (c) and (d)); we initially use threshold d_{int} to discover as many intersections as possible with the unrefined surfaces, followed by $d_{\text{int}}/5$ in the second iteration to enhance accuracy.

Due to the use of a fixed distance threshold d_{int} during intersection discovery, it can happen that surfaces that are merely close, as well as adjacent to a shared edge in the 2D edge graph, are erroneously classified as intersections, leading the refinement to produce an incorrect result (see Figure 4.6 (b)). We find that a common case where this happens is in models with slightly extruded surfaces, where a plane borders an offset plane in the 2D image, and the offset between them is smaller than d_{int}. To avoid this common failure case, we explicitly use the knowledge that parallel planes do not intersect, and since we predict whether any two planes should share an axis in the previous steps, we can exclude any vertices neighboring two planes with the same predicted axis (provided that the predicted axis is not unconstrained) from consideration in the intersection refinement step. The resulting improvement is shown in Figure 4.6 (c).

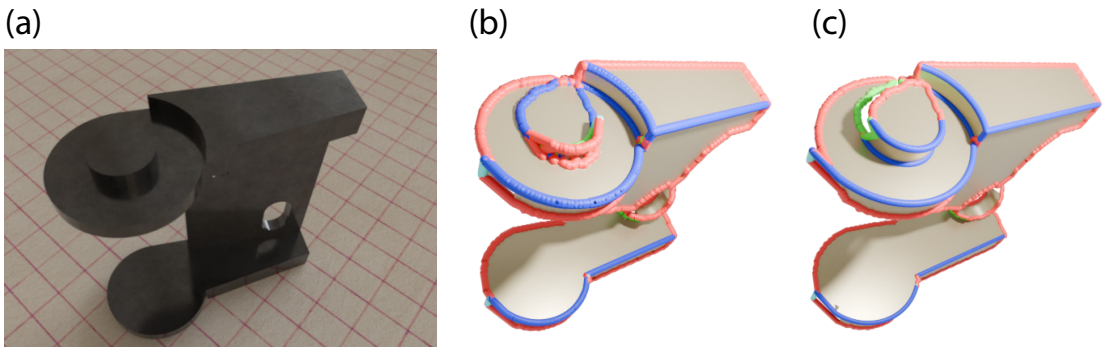

Figure 4.6: With an input object (a) containing slightly offset plane features, intersection refinement can lead to an incorrect result (b); note how the top stud gets "glued" to the plane below. We can explicitly use the assumption that parallel planes cannot intersect (c), fixing this common failure case.

We retain the steps mentioned in Section 4.4.2 to constrain the primitives according to predicted axis alignment during optimization. In all optimization stages of this pipeline, we employ the Levenberg-Marquardt algorithm with a dynamic damping factor as outlined in [Sha98]. We use parameter values $d_{\text{int}} = 0.05$ and $d_{\text{ransac}} = 0.03$, assuming our models are scaled to have a maximum sidelength of 1.

4.4.3 VB-Rep extraction

After the previous steps, we have optimized surface primitives $\tilde{\mathcal{S}}$ which will comprise the visible faces of our VB-Rep $\mathcal{V}$. To complete the structure, we also require edges to serve as boundaries for these faces, comprising a 3D wireframe. By the definition of a VB-Rep, these edges should all correspond to the observed 2D boundaries $\mathcal{E}$. The main challenge of VB-Rep extraction is therefore correctly *lifting* $\mathcal{E}$ to 3D. We outline our procedure in Figure 4.7.

Surface-aware wireframe extraction

Intersection Optimization As input to this step, we take the optimized surface primitives, the 2D edge graph $\mathcal{E}$, and the 3D vertices $\mathbf{V}_{3D}^{int} = \{\mathbf{V}_{3D}^{(i)} | I_i = 1\}$ from the intersection refinement step. Due to imprecision in the pixel-wise segmentation boundaries, the $\mathbf{V}_{3D}^{int}$ do not exactly adhere to the surface intersections, so we locally optimize these points to minimize the distance to each neighboring surface, but this time *allow the full 3D position to vary freely*, thus adhering to the boundary of the adjacent surfaces (Figure 4.7 (b)). Corner points adjacent to three surfaces are handled as well, ensuring the correctness of the VB-Rep vertices. Due to errors in the predicted segmentation boundaries, some edges may be missed from the set I entirely. To mitigate this, we perform this optimization for all points $\mathbf{V}_{3D}^{(i)}$ for which the visible neighbor set $\hat{N}_i$ contains at least two surfaces, and add it to the intersection subset if its new projected position is within 5 pixels of its starting coordinates.

Ambiguity resolution For primitive surfaces that can have more than one depth from a given viewpoint, the interpretation of our boundaries so far may be ambiguous. This problem is illustrated in Figure 4.7 (c), where the cylindrical surface may be interpreted as a protrusion (purple) or a hole (green). In general, there are up to four different bounded subsets of a primitive surface with the same boundary when projected onto view space, which amounts to making a discrete choice. We construct view-centric meshes $\mathcal{M}$ representing each of these choices and select the configuration whose mesh agrees most closely with $\mathbf{P}_{depth}$. The procedure we use for extracting $\mathcal{M}$ is detailed in section 4.4.3.

A more subtle problem arises from the fact that only the visible subsets of surfaces can be present in $\mathcal{V}$: Our $\mathbf{V}_{3D}^{(int)}$ may include vertices that are not close to this visible subset—note the differing blue intersection curves in Figure 4.7 (c), each a different subset of $\mathbf{V}_{3D}^{(int)}$ that borders the chosen visible surface subset. Therefore, in cases where more than one visible surface subset

exists, we retain only vertices in $\mathbf{V}_{3D}^{(int)}$ that are closer to the constructed mesh for the chosen visible subset than any others.

Wireframe splitting With these adjustments, $\mathbf{V}_{3D}^{(int)}$ now contains exactly the vertices in our 3D wireframe that belong to the boundaries of multiple surfaces in $\mathcal{V}$. All other vertices either border only one surface, or otherwise represent occluding surfaces. To handle these occlusions, we take all $\mathbf{v}_i \notin \mathbf{V}_{3D}^{(int)}$ with a neighbor set of size $|\hat{N}_i| > 1$ and split them into as many points as they have neighboring surfaces, such that each $\mathbf{v}_i$ becomes $\{\tilde{\mathbf{v}}_i^k \; | k = 1 \dots |\hat{N}_i|\}$. Then each new split point is only adjacent to one surface $\hat{N}_i^k$. We split our edges $\mathbf{E}_w$ according to this new structure, so that for each edge $(i, j) \in \mathbf{E}_w$ where either $\mathbf{v}_i$ or $\mathbf{v}_j$ are not intersection vertices and $|N_{ij}| > 1$, we replace the edge with new edges (c_i^k, c_j^k) for $k = 1 \dots |N_{ij}|$ where c_i^k is the index of the split vertex $\tilde{\mathbf{v}}_i^k$—or $\mathbf{v}_i$ if the endpoint is not a split vertex—in the list of vertices of the split wireframe. With all remaining vertices neighboring only one surface, we simply project them to that surface to complete our 3D boundaries. In practice, we optimize the vertex depths to minimize the distance to each surface; to ensure we converge on the correct local minimum, we first minimize the distance to $\mathcal{M}$ as an initialization step for surfaces with overlapping depths. Finally, we mark all newly-split vertices $\tilde{\mathbf{v}}_i^k$ as **occluded** vertices if $\text{depth}(\tilde{\mathbf{v}}_i^k) > \text{depth}(\tilde{\mathbf{v}}_i^{k'})$ for any k'. The result of this process can be seen in Figure 4.7 (d).

Visibility optimization One last type of boundary remains: Silhouettes which are solely caused by the intrinsic geometry of the surface, such as where a sphere or cylinder curves away from the view (Figure 4.1 (d)). Since the viewing direction is tangent to the surface along this boundary, the projected noise in the predicted silhouette tends to be amplified after the previous projection to 3D, causing a jagged boundary (Figure 4.7 (d)). We make the observation, however, that there exists an exact mathematical visibility boundary corresponding to this silhouette—the visibility

boundary is the set of points at which the surface normal is orthogonal to the view ray to that point. As in the intersection optimization stage, we attempt to optimizate all points with respect to this objective, and label those which converge to within 5 pixels of their starting position as visibility boundaries, updating their positions according to the optimization result. Note that we retain any and all intersection objectives in this step, as a vertex may lie at both the intersection of surfaces and the visibility boundary of a surface (see the two purple points in Figure 4.7 (e)).

Edge labeling It should be noted that in all of the above steps, we have classified parts of our wireframe as intersection, visibility, silhouette, and occluded at the level of vertices, rather the segments joining them. To obtain a VB-Rep with edge structures labeled according to the various visibility phenomena, we must precisely label the segments in our wireframe. We adopt the following rules: An edge segment is an **intersection** edge if both neighboring vertices are intersection vertices; an edge segment is a **visibility** edge if both neighboring vertices are visibility vertices; an edge segment is **occluded** if at least one of its vertices is not an intersection vertex, and at least one of its vertices is an occluded vertex. If our piecewise linear approximation of the VB-Rep edges is taken to the limit where segments are infinitesimal and all surfaces locally planar, these rules hold exactly.

Post-processing The above optimization steps, especially the visibility optimization, may displace the vertices of the wireframe considerably. We take steps to minimize this 3D displacement through post-processing, but these measures, as well as the optimization itself, cause overlapping edges and other problems. We post-process the wireframe by deleting vertices with degenerate edge angles (in practice, less than 36°) iteratively until none are left; we find that this fixes most geometric artifacts arising from the above steps.

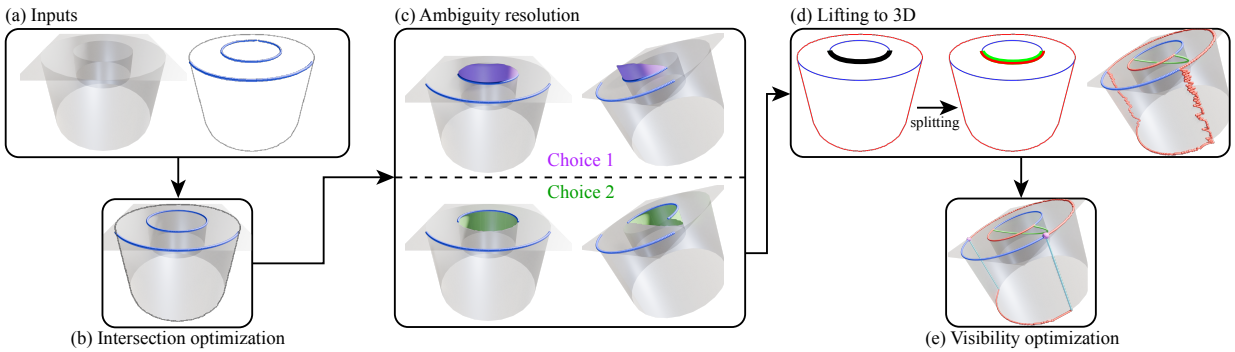

Figure 4.7: Surface-aware wireframe extraction pipeline. Input surfaces, 2D edge graph, and lifted intersection vertices (a) are optimized to the exact surface intersections (b), recovering potential missed intersections. We resolve ambiguities in how to lift the 2D boundaries into 3D (c) by comparing the resulting bounded surfaces, and keep the subset of intersection edges that agrees with that configuration. We can now split the wireframe along all non-intersection edges between surfaces (highlighed black edge in (d)), creating silhouette edges (red) and occluded edges (green), which we project to each surface to form a complete wireframe. Finally, we optimize the remaining silhouette vertices to discover any visibility boundaries (e), shown in cyan.

Surface mesh extraction

For extracting the visible surfaces $\mathcal{M}$ in Section 4.4.3 as well as our final VB-Rep surfaces, we use a discrete mesh processing algorithm which leverages the assumption that all surfaces in the VB-Rep are bounded by their visual extent from a particular viewpoint. Our procedure is illustrated in Figure 4.8: First, we generate a mesh $\hat{\mathcal{M}}_i$ for $\mathrm{S}_i \in \tilde{\mathcal{S}}$ by mapping a high-resolution square grid mesh to 3D using the surface parameterization from UV space to 3D $F_i : \mathbb{R}^2 \rightarrow \mathbb{R}^3$. We extract the vertices of $\hat{\mathcal{M}}_i$ that are contained in the 2D boundary of the region corresponding with S_i, illustrated by the blue curve in 4.8 (a), resulting in a submesh $\bar{\mathcal{M}}_i$ shown in Figure4.8 (b). To ensure that the distinct depth layers of S_i are correctly separated, we also label the faces by their normal dot product with the view ray (Figure 4.8 (c) and use a connected components algorithm on the face adjacency graph of $\bar{\mathcal{M}}_i$ with this face labeling to extract our final visibility meshes $\mathcal{M}_i^k$ for $k = 1 \ldots N_{\text{depths}}$.

Note that the surfaces $\tilde{\mathcal{S}}$ may be unbounded. When generating $\hat{\mathcal{M}}_i$ using a finite square grid

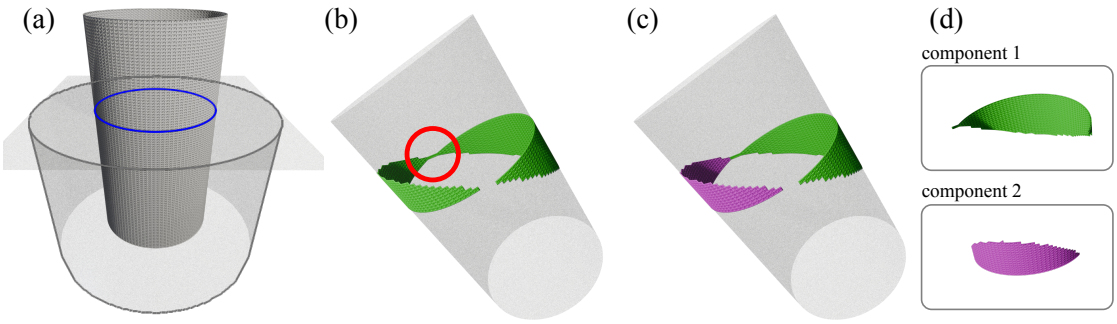

Figure 4.8: Visibility mesh extraction. Given the 2D boundary of a particular surface S (the blue curve in (a)), we extract the submesh of S with vertices whose projection is contained in the 2D boundary (b). Mutually exclusive regions may be incorrectly joined in this mesh (see the red circle), so we label the submesh faces according to their view normal (c), and use connected components on the face adjacency graph and the orientation labeling to extract all bounded submeshes (d).

in UV space, we adaptively scale the relevant dimensions (height for cylinders and cones, width and height for planes) until the contained mesh $\bar{\mathcal{M}}_i$ contains no edges from the UV grid boundary (indicating where our mesh cuts off prematurely).

For visualization purposes in the final surface meshes, we close the gap between the mesh edges and the boundary wireframe by projecting the non-manifold edges of this mesh to the nearest point on the wireframe.

4.5 Dataset

Our CAD primitive segmentation dataset consists of synthetic renders of 50,000 single-part CAD models from the AutoMate Part Dataset [JHC+21b], consisting of plane, cylinder, sphere, cone, and torus primitives. In order to close the sim-to-real training gap, we render our images with randomized lighting, material, and camera viewpoint using the physically-based path tracer Mitsuba [JSR+22], and along with each image, we render ground truth labels associating pixels to primitive instances. The dataset includes the labeling of each of the instances into one of the five

primitive types; in addition, we include labels for each primitive's primary orientation axis.

Scene setup Each of our scenes is comprised of a single ground plane with random rectangular bounds with the CAD part placed in the center, and the camera looking inwards from a random point sampled on a sphere of radius r_{view}. For lighting, we use a set of 10 randomly chosen environment maps downloaded from Poly Haven, and for materials, we use a collection of 19 PBR materials for different kinds of wood and metals downloaded from `3DTextures.me` complete with color, normal, roughness, and metallic maps. We also randomize the colors for a subset of these materials to provide some additional variation to the training data. We choose a natural "resting" orientation for the object based on maximal contact with the ground plane, in order for our scenes to resemble a plausible real capture scenario. We render the scenes with Mitsuba using the standard **path** integrator for simulating realistic multiple-bounce lighting. See Figure 4.9 for some representative images of the rendered RGB images in the dataset.

Axis labels Each primitive can be labeled as aligned with the X axis, Y axis, Z axis, or no axis if its orientation is unconstrained. Spheres have no orientation, so multiplying each of the remaining primitive types by these orientation choices gives us a total of 17 types to classify. Since all of the models are scraped directly from CAD modeling software, axis aligned features are almost always aligned solely with the model's local coordinate axes, allowing us to easily identify the alignment of each primitive using a small threshold (we use 1.5° angle difference between the primitive axis and the coordinate axis). However, these labels pertain to the *model* coordinate frame, which is arbitrary since for single-view reconstruction, rotating the model is no different from changing one's viewpoint. We therefore redefine the X axis to be whichever of the model coordinate axes is closest to parallel with the view (so that features facing the view head-on are aligned with X), and leave Z pointing up to obtain the final, view-corrected orientation labels.

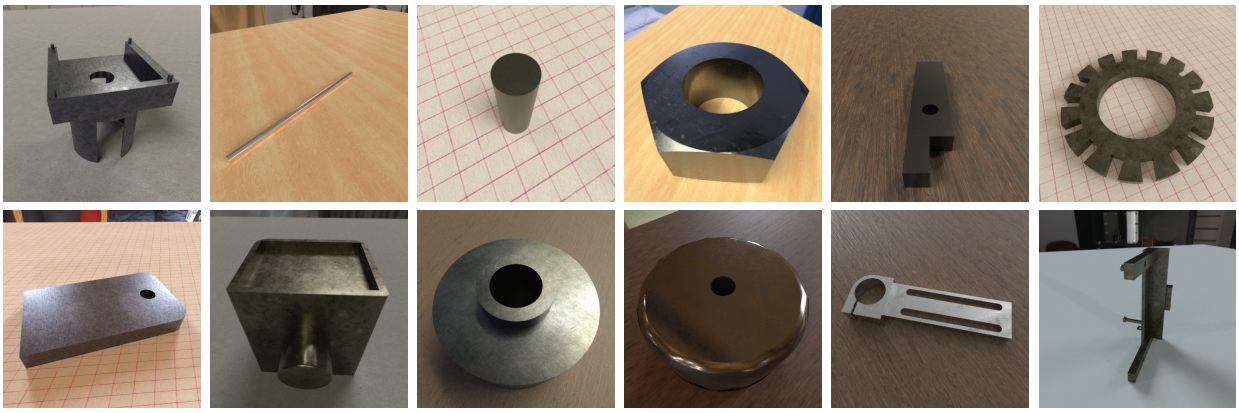

Figure 4.9: Representative images from synthetic training dataset.

4.6 Experimental Results

We evaluate our method on synthetic and real RGB-D images of man-made objects of varying complexity. We show experimental evaluations on the synthetic dataset described in Section 4.5 consisting of synthetic RGB-D renders of CAD parts from the AutoMate part dataset [JHC+21b], as well as a few results using real-world RGB-D measurements using an Intel® RealSense™ depth camera.

4.6.1 Qualitative assessment

Experiments with real data Results of our method on real-world RGB-D captures are shown in 4.11. We use a RealSense™ camera to capture paired RGB and depth maps for this purpose. Due to the fact that the RGB and depth cameras do not share the same viewpoint, we must take additional steps to reproject the depth point clouds to the viewpoint of the RGB camera so that we can associate the correct subsets of points in $\mathcal{S}$ with the predicted segmentation regions; this introduces some artifacts, such as holes in the depth points at occlusion boundaries. It can also be seen that considerable artifacts and holes exist in the depth point clouds themselves. Some care was

taken in selecting objects with materials that cooperated with the depth sensor—the first object, which we 3D printed, needed to be spray-painted with a matte coat for this purpose—though we emphasize that these measures were taken primarily due to limitations of the particular sensor we used, rather than the method itself.

The combination of intersection and alignment constraints makes the method quite robust; despite extreme errors and lack of detail in the depth point cloud, the reconstructed models exhibit clean topology (up to the limits of visibility). Note that some errors exist in the segmentation due to the sim-to-real gap between our training data and our captured objects. However, our algorithm is robust enough to correct many of these errors (such as the split plane in the third row), while minimizing others (the incomplete boundary jutting out the top in the second row). In other cases, the method fails "gracefully" (the missing floor of the box in the fourth row), allowing good quality reconstruction of all detected geometry.

synthetic validation In the interest of obtaining quantitative metrics with respect to ground truth geometry for a large collection of models, we also use synthetic data generated in accordance with Section 4.5 to validate our method. To simulate sensor noise, we use a combination of fractal brownian motion (FBM) noise followed by a bilateral gaussian blur; the former simulates large- and small-scale systematic errors, and the latter blurs out some model details, similar to the effects we observed with the RealSense™ camera. A visualization of the FBM noise used is shown in Figure 4.10, along with the parameters used to generate it.

Figure 4.12 shows the results of our method on several synthetic inputs. We observe that the segmentation model predicts detailed boundaries of the primitive instances, along with types and orientations, allowing the geometry optimization stage to produce high-quality VB-Reps for each example. Furthermore, we demonstrate that even when the segmentation model misses certain regions (as in the first and second rows), plausible geometry is inferred outside of these localized

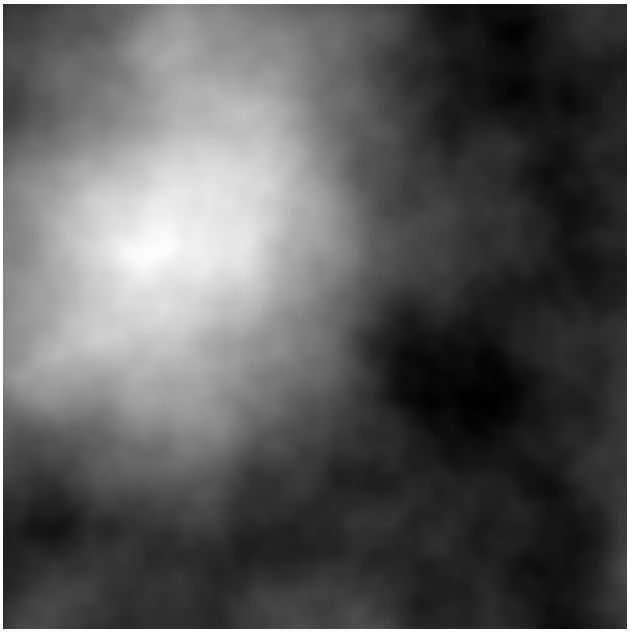

Figure 4.10: Fractal Brownian Motion noise used to perturb the synthetic depth maps, with the parameters lacunarity = 2.0, persistence = 0.5, octaves = 7, scale = 1000.

errors.

The blue intersection edges and vertices in Figures 4.11 and 4.12 show which VB-Rep elements can be identified as part of the original CAD B-Rep geometry. Since they are constrained by neighboring optimized surface geometry, they form clean, precise curves. Other types of edges show the various ways parts of a model can be hidden from view.

4.6.2 Quantitative evaluation

Evaluation Metrics To measure the geometric reconstruction accuracy on a synthetic validation set of 41 examples from the validation set of our dataset (see Section 4.5), we use the **Chamfer distance (CD)** between the reconstructed VB-Rep geometry and the ground truth VB-Rep. To obtain the ground truth VB-Rep, we use the wireframe extraction pipeline outlined in 4.4.3 with input consisting of the ground truth surfaces from $\mathcal{B}$ and 2D edge graph extracted from the

ground truth segmentation map rendered from $\mathcal{B}$; the CD distance is computed between the surface meshes of each VB-Rep generated as described in Section 4.4.3. Additionally, to assess the overall alignment of the reconstructed elements with those in the ground truth B-Rep, we define a **Primitive alignment** metric for individual primitives, measured as average Euclidean distances between primitive axis (we consider axes with opposite signs as equivalent for this metric, and so use the smallest distance among $\text{dist}(\hat{\mathbf{a}}_1,\ \hat{\mathbf{a}}_2)$ and $\text{dist}(\hat{\mathbf{a}}_1,\ -\hat{\mathbf{a}}_2)$). Primitive alignment is between corresponding pairs of primitives, so we compute a greedy matching across both sets of primitive instances according to their overlapping IoU in image space, stopping at 75% IoU. Averaged results are shown in Table 4.1.

We also compute a **face matching precision/face matching recall** based on this matching, which measure the proportion of ground truth faces that are matched by a predicted face, and the proportion of predicted faces that are matched by a ground truth face, respectively. Given the ground truth geometry of our synthetic examples, we evaluate the performance of primitive instance detection using standard classification metrics over the set of primitives: **mean primitive type accuracy** and **axis type accuracy**. We find the average values of these **segmentation-focused** metrics are as follows: face matching precision: 88.1%; face matching recall: 91.3%; axis accuracy: 94.5%; type accuracy: 99.2%.

The runtime of our method is about 50 seconds per example on average.

Ablation study We examine the effect of holding out various parts of our algorithm on the quality of results in Figure 4.13. With **No-int**, we leave out E_{int} from the optimization; with **No-axis**, we do not constrain axes based on predicted alignment during primitive fitting; and **Fit-only** uses neither of these refinements. For purposes of comparing the performance of each method, we use identically seeded synthetic noise maps (see Figure 4.10) for each ablation. Disabling intersection guidance leads to many tears in the model, showing that the intersection term is

Table 4.1: Ablation study: We evaluate our method with various features removed as described in Section 4.6.2 and average the results over 41 examples. All experiments use synthetic sensor errors as described in Section 4.6.1.

Method	CD(↓)	Prim. align.(↓)
Ours-all	**0.0628**	**0.0693**
No-axis	0.101	0.251
No-int	0.0768	0.0940
Fit-only	0.0819	0.244

crucial for ensuring continuity of the VB-Rep. However, the intersection term alone cannot ensure a consistent CAD model, as without axis guidance, features features become misaligned, sometimes causing broken topology as in the top example.

Quantitative effects of the above modifications (using the synthetic error model described in 4.6.2) are shown in Table 4.1. We observe that *ours-all* outperforms all other ablations, and that axis guidance, in particular, is essential for predicting the correct primitive orientations.

4.6.3 Comparisons with Point Cloud Segmentation Works

Since there is no directly applicable method for the problem of view-centric CAD reconstruction from a single RGB-D image, we examine prior works for reconstructing B-Reps from point clouds. We compare with Point2CAD [LOWS24], which generates a B-Rep by first segmenting a point cloud using a selection of backbone segmentation models, then fitting and intersecting geometric surface primitives to the resulting point clusters. We start by running the primitive segmentation backbone model HPNet [YYM+21b] on our depth points, with results shown in Figure 4.14. Even though we removed background points and normalized the remaining points to the centered unit cube in an attempt to match the expected input, the result is heavily over-segmented and cannot be used to extract a B-Rep.

We also attempted to compare with a recent work, Split-and-Fit [LCP+24], but were unable to

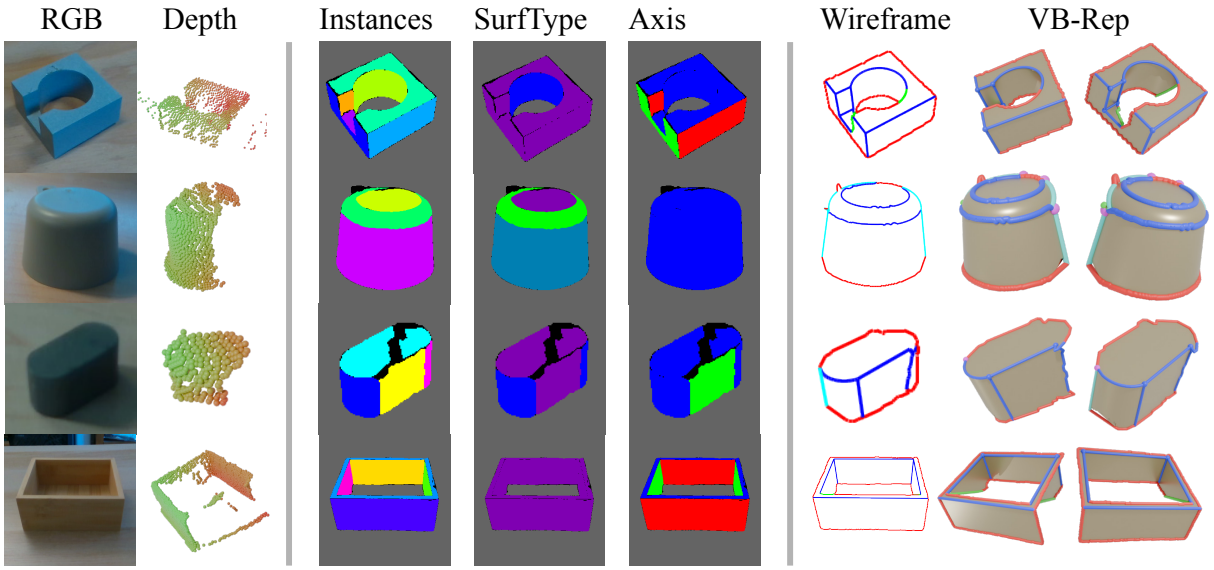

Figure 4.11: Results on real RGB-D images taken with a RealSense™ camera. Although the depth maps lack detail and contain noise and holes, the segmentation operates independently on the RGB image and provides much of the geometry supervision. The final VB-Reps recover clean geometry, correcting for errors in both the depth and segmentation. Note how the third example's split segmentation is corrected through segmentation post-processing, resuling in correct a wireframe. All edges in the resulting VB-Reps are colored according to their type: intersection edges are blue, silhouette edges are red, and visibility edges are cyan, and occluded edges are green, and same for vertices. In addition, triple intersection vertices and intersection-visibility vertices are enlarged, with the latter shown in magenta.

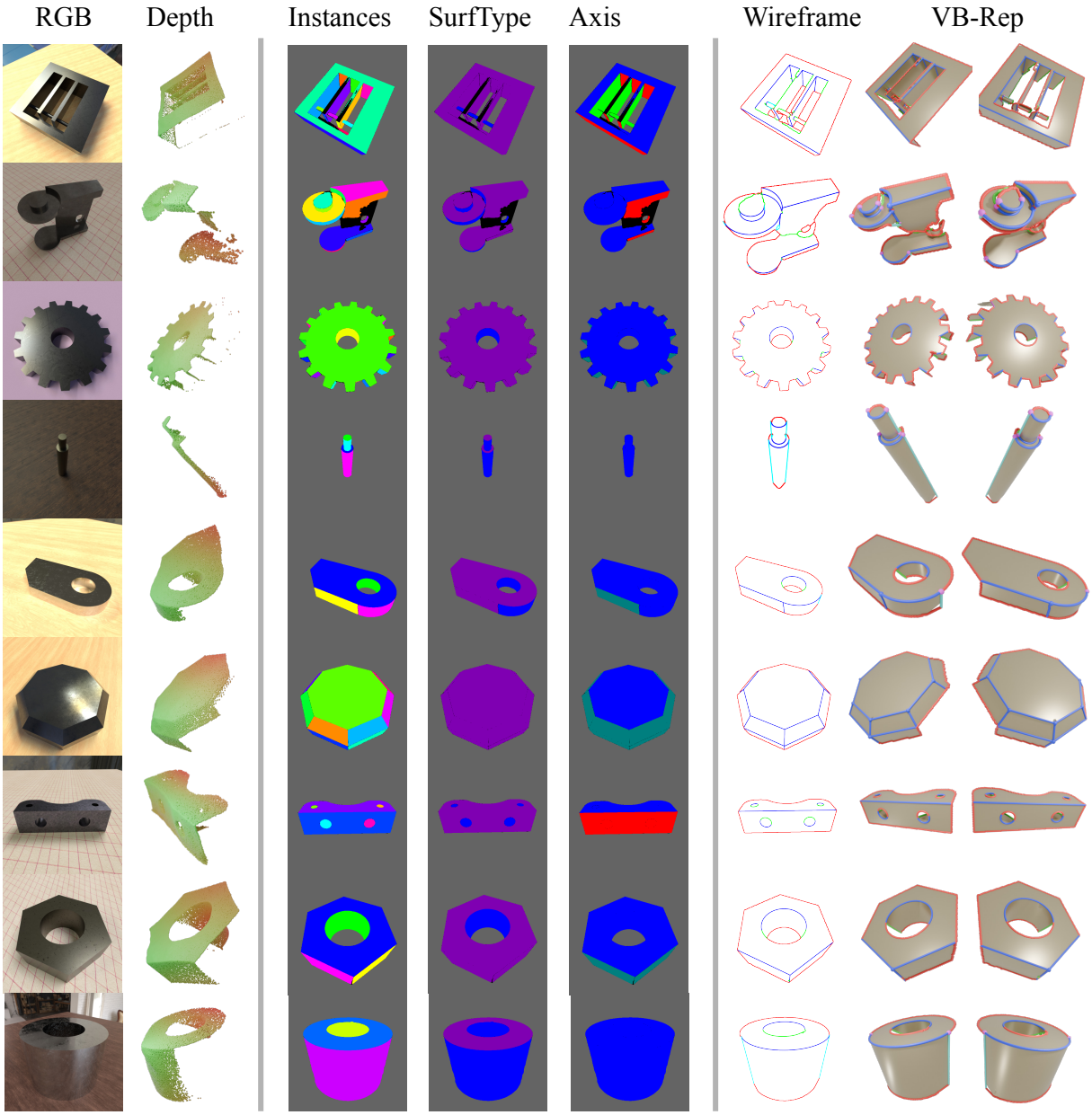

Figure 4.12: Results of our full pipeline on a collection of synthetic models with depth errors simulated using FBM noise and bilateral Gaussian blur. The VB-Rep edges are colored using the same conventions as Figure 4.11.

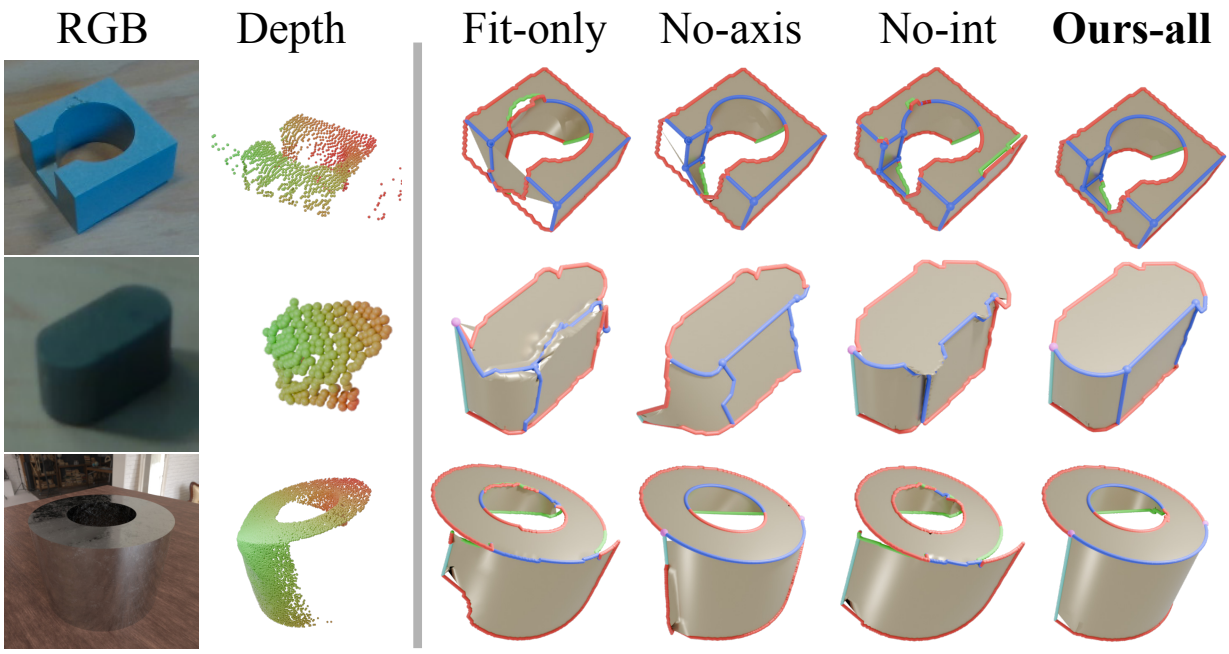

Figure 4.13: Comparison of results with our full pipeline with individual features left out on real data (first two rows) and a synthetic example (third row) with depth error simulated by a bilateral blur filter. With **No int**, we leave out E_{int} from the optimization, and with **No axis**, we do not constrain axes based on predicted alignment during primitive fitting. With **Fit-only**, we use neither constraint and simply use the RANSAC surface fits to the point cloud as input to the VB-Rep extraction stage. It can be seen that with no axis guidance, features features become slanted, causing a break in the topology in the top example. No intersection guidance leads many tears to appear in the model, as the observed edges cannot be explained by intersections of the fit primitives.

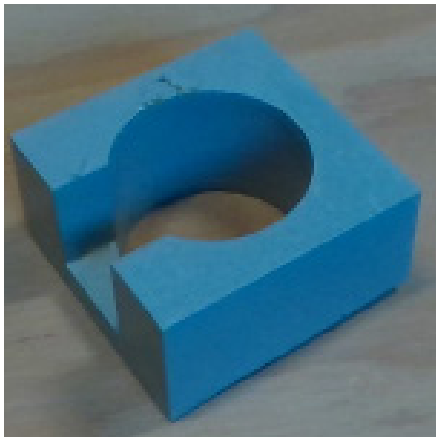

Figure 4.14: Result of using HPNet [YYM+21b] to segment a depth point cloud. The representative RGB image is shown on the left for reference—it is not part of the input to HPNet. Note that we have removed the background from the pointcloud and normalized it to lie within the bounds $[-1, 1]^3 \in \mathbb{R}^3$, the result is heavily over-segmented and impossible to use for extracting a valid B-Rep using downstream methods like Point2CAD [LOWS24].

produce any output, as the model is trained only on clean, synthetic point clouds.

4.7 Beyond Geometry

We presented a novel method for reconstructing CAD shapes from a single RGB-D image using a view-centric B-rep (VB-Rep) representation. Our approach addresses the challenges of incomplete and ambiguous topological information by incorporating visibility limits and encoding geometric uncertainty. Where Chapter 3 presents the possibility of using fabrication domain-specific constraints to reconstruct accurate CAD models from incomplete observations, we have here demonstrated the possibility of using the constraints inherent in CAD representations themselves to obtain similarly clean and precise models from just a single-view RGB-D image with significant noise. Additionally, our abstraction of partially-observed CAD models is a representation which uses CAD as a means of *understanding* an object's semantics—such as the surfaces and boundaries that comprise it, along with the precise limits of our visibility—extending CAD beyond its

traditional role as a tool in the process of manufacture and design. While we have looked at CAD geometry representations, CAD designs are often annotated with rich information beyond just the geometry. With this in mind, can we extend reconstruction techniques to cover other aspects of designs?

Chapter 5

Learning How Mechanical CAD Assemblies Work

The work discussed in this chapter was presented at the ICML 2022 Workshop on Machine Learning in Computational Design [NJW+23].

Understanding the functionality of objects is, in some ways, as fundamental as understanding their shape. We parse the functionality of objects around us automatically in order to decide how to grasp, push, and pull them to enact desired outcomes. Furthermore, our modern-day surroundings are crowded with mechanized gadgets demanding precise operation. Computer-Aided Design representations for these objects are annotated with critical information for understanding how they function—in addition to geometry, CAD systems allow designers to specify degrees of freedom between assembled parts.

Typically, CAD systems represent motion in an assembly using *mates*, which specify the degrees of freedom between pairs of parts. However, in most large repositories of 3D models, this motion information is absent. The primary reason is that each CAD system has its own internal and proprietary method for keeping track of mate information. CAD models are generally

exported and exchanged as purely geometric B-Reps, and while B-Reps describe the geometry of parts in an assembly, they do not include information about their mechanical degrees of freedom. This work therefore proposes to use B-Reps as the input format to our inference problem.

While prior work has addressed the problem of inferring motion from static assemblies [HLVK+17, WZS+19, YHY+20], they do not work directly with CAD representations, using instead geometric datasets (point clouds or meshes) that have been hand-annotated. The drawback of these approaches is that they are restricted to specific classes of common, recognizable objects, rather than working on arbitrary mechanisms using underlying geometric or structural properties. Recently, large repositories of CAD assemblies have been made public that include detailed information about how parts are mated together and move [JHC+21a, WJC+21]. These collections include a large variety of mechanical parts using rich CAD representations that have the potential to enable inference beyond specific object classes. This work aims to address the fundamental question: Can we learn to infer motion in general from collections of CAD assemblies?

While CAD assembly repositories are a valuable source of real-world assemblies, they also present several challenges to learning. The user's intent when creating an assembly affects which type of motion to use, or whether to annotate motion at all, leading to ambiguous or missing motion labels. Even a deterministic set of motions for an assembly has several equivalent ways to represent it using mates, leading to conflicting signal for learning-based methods seeking to understand just the motion through the mates (Figure 5.1.)

In this work we take the first step at addressing these challenges to learn how mechanical assemblies work from collections of CAD mates. Our key contributions are:

- A Dataset of moving assemblies for learning, with filters and a modified representation to mitigate the errors and ambiguity found in raw assemblies
- A user-annotated validation set

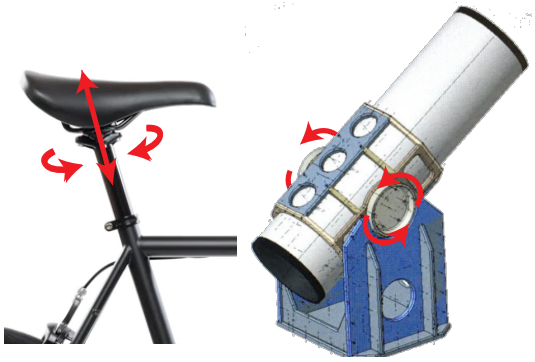

Figure 5.1: Sources of ambiguity. Left: A bicycle seat could be classified as sliding/rotating/fixed depending on which aspects of the bicycle's operation the *user intends* to model. Right: A single mate on either side of the telescope is sufficient to represent the motion creating *motion to mate* ambiguity.

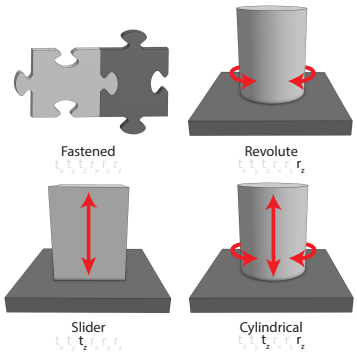

Figure 5.2: The four mate types and their DoFs.

- Baselines for the mate prediction task.

5.1 Dataset

We build upon the dataset of [JHC+21a], containing 125,133 CAD assemblies created on the Onshape platform. There are several key issues with learning directly from this dataset:

- *Noise*: Many assemblies are incomplete, or learning/practice material.
- *Motion-to-mate ambiguity*: the same motion can be represented by many different choices

of mates.

- *User intent ambiguity*: The correct mate/motion may depend on what the user considered important.

Making use of the vast amount of CAD assemblies at our disposal necessitates correcting some of the ambiguity inherent in the dataset. We address this problem in three ways. First, we propose a set of automatic methods for filtering the assemblies to remove noise. While these heuristic-driven methods cannot ensure that all non-plausible mates are resolved, these methods are able to remove a large number of models that are not physically plausible. Second, we propose a novel method to automatically remove motion-to-mate ambiguity by redefining mates, addressing problem (2). Finally, we create a validation set that is free of noise (1) and user intent ambiguity (2) by running a user study over a hand-selected portion of the dataset.

Heuristic Filters To mitigate problem (5.1), we apply several heuristic filters based on our understanding of the domain. We first filter for assemblies that have at least one moving part to fit our task. To account for designs that are incomplete, we filter out all assemblies with disconnected pieces, i.e. separately connected sub-assemblies rather than a single connected assembly. We further remove what we call *compound mates*, in which multiple, possibly conflicting, mates are defined between the same pair of parts, rendering the motion invalid. Finally, we remove spherical and pin-slot mates, which are seldom used, and parallel and planar mates, which are seldom used correctly. Our assemblies consist of four common mechanical joints: Fastens, Revolutes, Sliders, and Cylindricals (see Figure 5.2). These filters leave us with 13,957 assemblies, which we call the *cleaned* set.

Removing Motion-to-Mate Ambiguity To address (5.1), we augment our dataset so that the mates underlying part motions are defined in a consistent way. Specifically, our goal is to

ensure maximal connectivity between the parts in an assembly, which is a unique description of its motion. Returning to the telescope example in Figure 5.1, this means adding an additional mate on the other side of the telescope, since the hinge makes contact on both sides. We can create such maximal connectivity if we ensure that all pairs of parts that can be mated are mated. If we have consistent criteria for which parts can be mated, we can create the additional mates by using the degrees of freedom inferred from the existing mates. Specifically, we examine the chain of existing mates connecting the two parts (which must exist after the previous filter), and define a new mate using the degrees of freedom allowed by the existing mates' combined constraints.

Our key insight is that we can use geometric cues to identify which pairs of parts can be mated. Namely, we note that 1) in physically realizable assembles, mated parts are in contact and 2) mates are predominantly created from a discrete set of axes derived from the geometry of each part (cylinder axes, face center normals, etc.); we use the set of unique mate coordinate frame z-axes from [JHC+21a], keeping only the ray direction and offset rather than the full relative coordinate transform, since this is only useful if one needs to derive the part transforms, which we already have.

We use these two geometric assumptions that determine which pairs of parts can be mated to filter out assemblies in our dataset. First, we discard assemblies with parts that are mated together but do not satisfy the geometric assumption. This discards 36% of the cleaned assemblies. By visual inspection, we noticed that these models tend to be physically implausible, i.e. they have floating parts. We further discard assemblies for which we could not add all the missing mates—pairs of parts that should be mated under our maximal mate assumption, but whose derived relative motion is not a simple motion type. This discards 14% of the remaining assemblies. At the end of this process, we are left with a total of 7,328 assemblies in our final set.

Validation Set Finally, we recruited 4 CAD experts to construct a consistent set of type labels for a sub-collection of assemblies, hand selected by us for visual clarity. We presented the participants with mate-less CAD assemblies with pre-positioned parts, and asked them to add mates using a commercial CAD software. A total of 100 assemblies were annotated 3 times each to generate validated labels by majority consensus.

5.2 Motion Prediction

Our system takes as input an assembly represented as a B-rep describing the various parts. We infer the connectivity structure using the criteria discussed in the previous section, so the remaining task is to find the correct motion type and the correct motion axis, when applicable (see below).

Motion Type We used the SBGCN architecture [JHC+21a] to encode and pool the topological features from each part to form axis and part-level features. For each pair of mated parts, we combine the resulting features as input to an MLP, which outputs four class probabilities, corresponding to mate types. We experimented with many variations and additions, but found that none made a noticeable improvement. We discuss these further in Section 5.3.

Motion Axis For predicting the correct axis of motion, we select among the possible shared axes between all (touching) pairs of parts in the assembly (see Section 5.1). For certain mate types, multiple axes may be equally valid: For sliders, any axis with the same direction is equally valid; for fastens, any axis is valid as there is no motion. Taking this into account, in 88.2% of mates, there are no incorrect choices for the axis location. For the remaining cases, we train a predictor to infer a probability score for each axis to be used in a mate, and then take the maximum probability axis among each group of axes belonging to the same pair of parts as the location of the mate axis.

Similar to the type predictor, we use SBGCN to obtain pooled features for each part, to which we concatenate the features of topological entities used in each axis, which are input to the final MLP layer.

5.3 Experimental Results

We split the assemblies into train, validation, and test sets with a split of 80% - 10% - 10%. We created two additional test sets based on the original test set:

- A hand-selected subset of the test set consisting of assemblies that look like a human could infer what they are
- A subset of the above assemblies, with the mate labels recreated by consensus of human experts (see the description of the user study in Section 5.1.

Type Prediction The accuracy of our mate type predictor is 65% on the full dataset, 62.8% on the handpicked data, and 49.4% on a manually created test set (see below). We tried various modifications to our network architecture and features used during learning. As a baseline, we attempted to train a predictor using PointNet [QSMG16] rather than SBGCN, and found that the accuracy on the full dataset dropped from 65% to 58.5%. Sampling surface points and incorporating the UV-Net encoder of [JSL+21], then concatenating the resulting features to those of SBGCN makes no difference to the accuracy. We also attempted to incorporate assembly-wide context in various ways, such as adding graph message passing layers between mated parts, and passing a surface point cloud of the entire assembly. These methods fail to make a difference. We also attempted to incorporate axis information in the form of per-topology SBGCN features of the mate connectors along which parts are mated, or point clouds depicting snapshots of rotating and sliding motions between the parts, but it did not help. Finally, we created a heuristic set of labels

for each mate, indicating whether it should be able to rotate or slide based on geometric analysis of the part overlaps in motion, and used these to filter our dataset. The accuracy was unaffected by training on this subset, but we note that the test accuracy when restricted to this subset was 72%, up from 65%. Using these heuristic labels as additional input did not help, however.

Location Prediction For the mate axis location problem, we get 71% accuracy among the mates where there is more than one choice of axis (only 12.8% of the data), resulting in 96.3% accuracy overall.

Comparison to Automate Direct comparison with Automate [JHC+21a] is not possible since Automate predicts among 8 classes, and has a far more skewed mate type distribution. Instead we compare the accuracy improvement over a model that predicts the most common mate type in each dataset: fastened for Automate, and revolute for ours. Automate achieves roughly a 10% lift by this metric, whereas our work produces a 25% improvement.

5.3.1 User Study

Not all mates in the dataset had a corresponding mate chosen by our experts (even after densifying the mates as discussed in Section 5.1). Ultimately, we obtained a set of 341 mates for which at least two expert-defined mates could be compared, out of the full 349. Out of these 341 mates, 301 had an agreed upon type according to those two or more experts, so 88% of mates which could be compared between multiple experts were agreed to be of one particular type. The type chosen by consensus among these mates agreed with the original mate type 66.8% of the time. On average, in 68.8% of the mates, the original type in the dataset agreed with the mate types the experts chose chose.

5.4 Discussion

Our performance on the hand-picked data is slightly worse than on the full data, indicating that the subjective criteria by which we deemed those assemblies reasonable do not make them easier to predict for our machine learning model. Furthermore, the performance is much worse on the user study-based test set, which might suggest that the task of inserting missing mates into an assembly results in different biases in assigning mate types than assembling from scratch, or that some CAD assemblies are not annotated with functional motion in mind. The agreement rate of the expert labels, both with each other and with the original mates, suggests that human performance on a dataset of this quality is bounded at about 70%.

While recent annotated CAD datasets present a wealth of metadata pertaining to object functionality, the use of these annotations is optional, and far from standardized. As such, the ambiguities we hoped to solve in this work also plague our training data, and, as our user study results indicate, are inherent in the interpretation of CAD assemblies, particularly without sufficient context. Looking at the broader scope of the problem, it is unreasonable to expect to be able to predict motion for an arbitrary collection of shapes using geometry alone. Therefore, further work in this direction will likely require significant human effort in annotating assemblies for which this task is feasible.

Alternatively, to side-step the quality of motion data in CAD datasets, one might consider self-supervised approaches that use only shape priors. A possible approach might be to "simulate" proposed part motions while evaluating the validity of the resulting geometry according to the learned shape priors. The challenge with this approach is performing such motions in the presence of multiple, possibly coupled degrees of freedom, which presents challenges related to combinatorial search as well as numerical optimization.

Chapter 6

Conclusions and Future Work

In this thesis, we have seen that computer-aided design representations can serve as geometric priors that facilitate precise reconstruction, even in the face of uncertain or incomplete data. This is in large part because they tell us more about what we are looking at, enabling the use of specialized techniques to analyze our data; in Chapter 3, knowledge that our model was composed of planar parts allowed us to extract cut paths using multi-view part segmentation, and in Chapter 4, we find that constraints relating to common CAD practices can be learned directly from images. Looking beyond geometry, we have seen evidence that large CAD datasets can make it possible to infer functionality in the form of part motions from shape and structural cues in CAD assemblies.

B-Reps are a ubiquitous and versatile abstraction for representing fabricable geometry, and we have demonstrated that the constraints that underlying them can be leveraged to produce clean reconstructions from partial observations. We note, however, that the VB-Reps we introduced in Chapter 4 only include a subset of geometries found in many real-world CAD designs, such as spline surfaces with arbitrary numbers of control points. The constraint set we considered is also not exhaustive; our current work uses a minimal set of constraints to establish a foundation for solving this problem effectively. Beyond these, many other implicit constraints are found in CAD

designs, some of which draw from the underlying operations that produced them, such as tangent constraints between a bevel and its neighboring surfaces. We speculate that implementing all such constraints would be a task rivaling the development of a CAD geometry kernel itself, and leave it to future work. Furthermore, incorporating these additional constraints places more burden on the vision algorithms and associated datasets to procure this information in image space. We therefore also leave it to future work to collect a larger and higher-quality dataset. This could potentially entail real-world data collection by tracking the pose of real objects with known CAD models for training in real capture setups.

In Chapter 2, we briefly discussed the blossoming field of procedural CAD generative modeling. While we set these aside in our discussion thus far due to their limited scope, the potential for procedural representations is great; after all, they are closer to "true" designs than B-Reps, which can be seen as a lower-level geometric by-product of procedural CAD operations. Such representations are sequential in nature, since they record an ordered list of operations. Rather than simply placing geometry piece by piece, operations depend on each other in complex ways, referencing geometry produced in previous steps, and potentially breaking if certain criteria are not met. Therefore, interpreting CAD operations can be seen as interpreting a *program*; this is the case in the popular CAD tool Onshape, where every part has an associated program in the domain-specific language FeatureScript. With this view, reconstruction can be framed as a *program synthesis* problem. Program synthesis approaches have certain desirable features: Using control flow constructs, it is possible to naturally and concisely define repetitions and symmetries common in many human designs. A major challenge with program synthesis is the ambiguity in representing shapes; many different programs may produce the same geometry with drastically different complexity, making the search for the "best" program a large combinatorial search, limiting the scalability of this problem. This problem plagues procedural representations—we don't know which root to pick for a CSG tree, or what order to sketch and extrude features.

These are problems endemic to the field of programming languages in general, so research in this direction could benefit from further fundamental developments in program synthesis. Hybrid approaches might also be considered, which use high-level geometric heuristics to decompose an object into parts that can feasibly be represented using simple programs.

We have explored reconstructing the visual world in the form of geometry, and presented preliminary work on reconstructing the non-visual world in the form of object functionality. Many other aspects of designed objects are intangible at first glance—it is not immediately obvious how a chair was put together, or how much friction it generates with the floor. In Chapter 3, we identified the detailed joints and connectors between parts (Figures 3.4 and 3.14), which can be followed to assemble a complete object (Figure 3.1). However, true assembly instruction must reason about the parts in greater detail: One must ensure that connectors do not obstruct each other, or that any tools used in assembly have enough room to work, such as by specifying a particular assembly order. One possible direction to incorporate detailed assembly instructions into reconstructed designs is to adapt prior work in automatic generation of assembly instructions [APH+03]. Aside from assembly order, the internal geometry of joints is another hidden aspect of designs—even in carpentry, complex joinery can improve structural integrity by using gravity and interlocking structures in ingenious ways. While vision approaches alone might be insufficient to identify all but the simplest types of joints (as we did), a possible approach to explore is constraining the search for internal geometric structures by optimizing with respect to structural stability, similar to [YKGA17], or with respect to additional objectives such as fabrication cost and packing efficiency [WSP21]. Concerns such as the physical properties and structural stability of an object, among other things, must also take into account the materials used in construction. Some recent work has been proposed to predict materials in CAD assemblies [BGL+24], though the problem of representing materials in general (beyond simple categorical material libraries present in CAD systems) is an open problem. Some insights might be gained from the field of physically-based

rendering, which uses a principled material model with many tuneable parameters to represent any desired appearance; or the field of material discovery in the materials sciences, where material can be defined as a microstructure with the desired physical properties. Ultimately, as we desire to reconstruct more of the intangible properties of objects, it will likely become necessary to incorporate multi-sensory information beyond vision. Material, in particular, is largely experienced through tactile sense. Recent work on tactile-augmented radiance fields [DYL+24] opens exciting avenues of research in the area of multi-sensory reconstruction.

Bibliography

[ADSG+20] Edoardo Alberto Dominici, Nico Schertler, Jonathan Griffin, Shayan Hoshyari, Leonid Sigal, and Alla Sheffer. Polyfit: Perception-aligned vectorization of raster clip-art via intermediate polygonal fitting. *ACM Transaction on Graphics*, 39(4), 2020.

[AFS+11] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.

[APH+03] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics (TOG)*, 22(3):828–837, 2003.

[BCF+18] Francesco Buonamici, Monica Carfagni, Rocco Furferi, Lapo Governi, Alessandro Lapini, and Yary Volpe. Reverse engineering modeling methods and tools: a survey. *Computer-Aided Design and Applications*, 15(3):443–464, May 2018.

[BGL+24] Shijie Bian, Daniele Grandi, Tianyang Liu, Pradeep Kumar Jayaraman, Karl Willis, Elliot Sadler, Bodia Borijin, Thomas Lu, Richard Otis, Nhut Ho, et al. Hg-cad: hierarchical graph learning for material prediction and recommendation in computer-aided design. *Journal of Computing and Information Science in Engineering*, 24(1):011007, 2024.

[BI17] Tolga Birdal and Slobodan Ilic. CAD Priors for Accurate and Flexible Instance Reconstruction. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 133–142, Venice, October 2017. IEEE.

[BJ01] Yuri Y Boykov and M-P Jolly. Interactive graph cuts for optimal boundary & region segmentation of objects in nd images. In *Proceedings eighth IEEE international conference on computer vision. ICCV 2001*, volume 1, pages 105–112. IEEE, 2001.

[BSFG09] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. Patchmatch: A randomized correspondence algorithm for structural image editing. *ACM Trans. Graph.*, 28(3):24, 2009.

[BTS+17] Matthew Berger, Andrea Tagliasacchi, Lee M Seversky, Pierre Alliez, Gael Guennebaud, Joshua A Levine, Andrei Sharf, and Claudio T Silva. A survey of surface reconstruction from point clouds. In *Computer Graphics Forum*, volume 36, pages 301–329. Wiley Online Library, 2017.

[Cap] Capturing Reality. Realitycapture.

[CMS+22] Bowen Cheng, Ishan Misra, Alexander G. Schwing, Alexander Kirillov, and Rohit Girdhar. Masked-attention mask transformer for universal image segmentation. 2022.

[CPK+17] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017.

[CPK19] Christopher Choy, Jaesik Park, and Vladlen Koltun. Fully convolutional geometric features. In *ICCV*, 2019.

[CRH+20] Weijuan Cao, Trevor Robinson, Yang Hua, Flavien Boussuge, Andrew R. Colligan, and Wanbin Pan. Graph representation of 3d cad models for machining feature recognition with deep learning. volume Volume 11A: 46th Design Automation Conference (DAC) of *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, 08 2020.

[CSK21] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation. 2021.

[CXG+16] Christopher B Choy, Danfei Xu, JunYoung Gwak, Kevin Chen, and Silvio Savarese. 3d-r2n2: A unified approach for single and multi-view 3d object reconstruction. In *European conference on computer vision*, pages 628–644. Springer, 2016.

[CYH+24] Tianrun Chen, Chunan Yu, Yuanqi Hu, Jing Li, Tao Xu, Runlong Cao, Lanyun Zhu, Ying Zang, Yong Zhang, Zejian Li, et al. Img2cad: Conditioned 3d cad model generation from single image with structured visual geometry. *arXiv preprint arXiv:2410.03417*, 2024.

[CZ19] Zhiqin Chen and Hao Zhang. Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948, 2019.

[CZS+13] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep: Extracting editable objects from a single photo. *ACM Trans. Graph.*, 32(6), nov 2013.

[DBK+21] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold,

Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[DIP+18] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. InverseCSG: automatic conversion of 3D models to CSG trees. *ACM Transactions on Graphics*, 37(6):1–16, December 2018.

[DYL+24] Yiming Dou, Fengyu Yang, Yi Liu, Antonio Loquercio, and Andrew Owens. Tactile-augmented radiance fields. *arXiv preprint arXiv:2405.04534*, 2024.

[FCOS05] Shachar Fleishman, Daniel Cohen-Or, and Cláudio T Silva. Robust moving least-squares fitting with sharp features. *ACM transactions on graphics (TOG)*, 24(3):544–552, 2005.

[FF02] Gerald E Farin and Gerald Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002.

[FP10] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376, 2010.

[FSG16] Haoqiang Fan, Hao Su, and Leonidas Guibas. A point set generation network for 3d object reconstruction from a single image, 2016.

[GCLZ16] Bo Gao, Xiaowu Chen, Jianwei Li, and Dongqing Zou. Modeling interactive furniture from a single image. *Comput. Graph.*, 58(C):102–108, aug 2016.

[GFK+18] Thibault Groueix, Matthew Fisher, Vladimir G. Kim, Bryan C. Russell, and Mathieu Aubry. AtlasNet: A Papier-M\^ach\'e Approach to Learning 3D Surface Generation, July 2018. arXiv:1802.05384 [cs].

[GFRG16] Rohit Girdhar, David F. Fouhey, Mikel Rodriguez, and Abhinav Gupta. Learning a predictable and generative vector representation for objects, 2016.

[GLP+22] Haoxiang Guo, Shilin Liu, Hao Pan, Yang Liu, Xin Tong, and Baining Guo. ComplexGen: CAD reconstruction by B-rep chain complex generation. *ACM Transactions on Graphics*, 41(4):1–18, July 2022.

[GLPG22] Hao-Xiang Guo, Yang Liu, Hao Pan, and Baining Guo. Implicit conversion of manifold b-rep solids by neural halfspace representation. *ACM Transactions on Graphics*, 41(6):1–15, nov 2022.

[gra] Grabcad. `https://grabcad.com/`. Accessed: 2022-05-19.

[HDS+18] Shayan Hoshyari, Edoardo Alberto Dominici, Alla Sheffer, Nathan Carr, Zhaowen Wang, Duygu Ceylan, and I-Chao Shen. Perception-driven semi-structured boundary vectorization. *ACM Trans. Graph.*, 37(4), July 2018.

[HEH05] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Automatic photo pop-up. *ACM Trans. Graph.*, 24(3):577–584, jul 2005.

[HLVK+17] Ruizhen Hu, Wenchao Li, Oliver Van Kaick, Ariel Shamir, Hao Zhang, and Hui Huang. Learning to predict part mobility from a single static snapshot. *ACM Transactions on Graphics (TOG)*, 36(6):1–13, 2017.

[HMK+18] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. Deepmvs: Learning multi-view stereopsis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2821–2830, 2018.

[Hor89] Berthold K P Horn. Obtaining Shape from Shading Information. 1989.

[HZS21] Jingwei Huang, Yanfeng Zhang, and Mingwei Sun. Primitivenet: Primitive instance segmentation with local primitive embedding under adversarial metric. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15343–15353, October 2021.

[ISS17] Hamid Izadinia, Qi Shan, and Steven M. Seitz. Im2cad. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2422–2431, 2017.

[JHC+21a] Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G. Kim, and Adriana Schulz. Automate: A dataset and learning approach for automatic mating of cad assemblies. *ACM Trans. Graph.*, 40(6), dec 2021.

[JHC+21b] Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G. Kim, and Adriana Schulz. Automate: a dataset and learning approach for automatic mating of cad assemblies. *ACM Trans. Graph.*, 40(6), dec 2021.

[JLD+23a] Pradeep Kumar Jayaraman, Joseph G. Lambourne, Nishkrit Desai, Karl D. D. Willis, Aditya Sanghi, and Nigel J. W. Morris. SolidGen: An Autoregressive Model for Direct B-rep Synthesis, February 2023. arXiv:2203.13944 [cs].

[JLD+23b] Pradeep Kumar Jayaraman, Joseph G. Lambourne, Nishkrit Desai, Karl D. D. Willis, Aditya Sanghi, and Nigel J. W. Morris. Solidgen: An autoregressive model for direct b-rep synthesis, 2023.

[JSL+21] Pradeep Kumar Jayaraman, Aditya Sanghi, Joseph G. Lambourne, Karl D.D. Willis, Thomas Davies, Hooman Shayani, and Nigel Morris. Uv-net: Learning from boundary representations. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021.

[JSR+22] Wenzel Jakob, Sébastien Speierer, Nicolas Roussel, Merlin Nimier-David, Delio Vicini, Tizian Zeltner, Baptiste Nicolet, Miguel Crespo, Vincent Leroy, and Ziyi Zhang. Mitsuba 3 renderer, 2022. https://mitsuba-renderer.org.

[KALD20] Weicheng Kuo, Anelia Angelova, Tsung-Yi Lin, and Angela Dai. Mask2CAD: 3D Shape Prediction by Learning to Segment and Retrieve, July 2020. arXiv:2007.13034 [cs, eess].

[KDA+24] Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4713–4722, 2024.

[KL11] Johannes Kopf and Dani Lischinski. Depixelizing pixel art. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2011)*, 30(4):99:1 – 99:8, 2011.

[KLL17] Chen Kong, Chen-Hsuan Lin, and Simon Lucey. Using locally corresponding cad models for dense 3d reconstructions from a single image. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5603–5611, 2017.

[KMD+17] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE international conference on computer vision*, pages 66–75, 2017.

[KMJ+19a] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[KMJ+19b] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9601–9611, 2019.

[KS99] K.N. Kutulakos and S.M. Seitz. A theory of shape by space carving. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, pages 307–314 vol.1, Kerkyra, Greece, 1999. IEEE.

[KSS12] Adarsh Kowdle, Sudipta Sinha, and Rick Szeliski. Multiple view object cosegmentation using appearance and stereo cues. In *Proceedings of the 12th European Conference on Computer Vision (ECCV)*. Springer Verlag, October 2012.

[LCP+24] Yilin Liu, Jiale Chen, Shanshan Pan, Daniel Cohen-Or, Hao Zhang, and Hui Huang. Split-and-Fit: Learning B-Reps via structure-aware Voronoi partitioning. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 43(4):108:1–108:13, 2024.

[LDSW21] Yujia Liu, Stefano D'Aronco, Konrad Schindler, and Jan Dirk Wegner. Pc2wf: 3d wireframe reconstruction from raw point clouds, 2021.

[LOWS24] Yujia Liu, Anton Obukhov, Jan Dirk Wegner, and Konrad Schindler. Point2cad: Reverse engineering cad models from 3d point clouds. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3763–3772, 2024.

[LSD+19a] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas Guibas. Supervised Fitting of Geometric Primitives to 3D Point Clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2647–2655, June 2019. arXiv:1811.08988 [cs].

[LSD+19b] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2652–2660, 2019.

[LWC+11] Yangyan Li, Xiaokun Wu, Yiorgos Chrysathou, Andrei Sharf, Daniel Cohen-Or, and Niloy J Mitra. Globfit: Consistently fitting primitives by discovering global relations. In *ACM SIGGRAPH 2011 papers*, pages 1–12. 2011.

[LWJ+21] Joseph G Lambourne, Karl DD Willis, Pradeep Kumar Jayaraman, Aditya Sanghi, Peter Meltzer, and Hooman Shayani. Brepnet: A topological message passing system for solid models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12773–12782, 2021.

[LWJ+22] Joseph G. Lambourne, Karl D. D. Willis, Pradeep Kumar Jayaraman, Longfei Zhang, Aditya Sanghi, and Kamal Rahimi Malekshan. Reconstructing editable prismatic CAD from rounded voxel models. In *SIGGRAPH Asia 2022 Conference Papers*, pages 1–9, November 2022. arXiv:2209.01161 [cs].

[MON+19] Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space, 2019.

[MRA+22] Albert Matveev, Ruslan Rakhimov, Alexey Artemov, Gleb Bobrovskikh, Vage Egiazarian, Emil Bogomolov, Daniele Panozzo, Denis Zorin, and Evgeny Burnaev. Def: Deep estimation of sharp geometric features in 3d shapes, 2022.

[MSK+21] Peter Meltzer, Hooman Shayani, Amir Khasahmadi, Pradeep Kumar Jayaraman, Aditya Sanghi, and Joseph Lambourne. UVStyle-Net: Unsupervised Few-shot Learning of 3D Style Similarity Measure for B-Reps. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021. arXiv: 2105.02961.

[NGEB20] Charlie Nash, Yaroslav Ganin, S. M. Ali Eslami, and Peter W. Battaglia. PolyGen: An Autoregressive Generative Model of 3D Meshes, February 2020. arXiv:2002.10880 [cs, stat].

[NJW+23] James Noeckel, Benjamin T. Jones, Karl Willis, Brian Curless, and Adriana Schulz. Mates2motion: Learning how mechanical cad assemblies work, 2023.

[NZCS21] James Noeckel, Haisen Zhao, Brian Curless, and Adriana Schulz. Fabrication-aware reverse engineering for carpentry. *Computer Graphics Forum*, 40:301–314, 08 2021.

[PFS+19] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174, 2019.

[PIQD22] Kebin Peng, Rifatul Islam, John Quarles, and Kevin Desai. Tmvnet: Using transformers for multi-view voxel-based 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 222–230, June 2022.

[PMKB23] Ivan Puhachov, Cedric Martens, Paul Kry, and Mikhail Bessmeltsev. Reconstruction of machine-made shapes from bitmap sketches. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 42(6), December 2023.

[PS83] Michael Plass and Maureen Stone. Curve-fitting with piecewise parametric cubics. *SIGGRAPH Comput. Graph.*, 17(3):229–239, July 1983.

[PUG19] Despoina Paschalidou, Ali Osman Ulusoy, and Andreas Geiger. Superquadrics revisited: Learning 3d shape parsing beyond cuboids, 2019.

[QSMG16] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[RBK21] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision Transformers for Dense Prediction, March 2021. arXiv:2103.13413 [cs].

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[RR18] Stephan R Richter and Stefan Roth. Matryoshka networks: Predicting 3d geometry via nested shape layers. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1936–1944, 2018.

[SCD+06] S.M. Seitz, B. Curless, J. Diebel, D. Scharstein, and R. Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 519–528, 2006.

[SD99] Steven M Seitz and Charles R Dyer. Photorealistic scene reconstruction by voxel coloring. *International journal of computer vision*, 35:151–173, 1999.

[SDR+22] G. Sharma, B. Dash, A. RoyChowdhury, M. Gadelha, M. Loizou, L. Cao, R. Wang, E. G. Learned-Miller, S. Maji, and E. Kalogerakis. Prifit: Learning to fit primitives improves few shot point cloud segmentation, 2022.

[SF16] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[SH16] Maximilian Sand and Dominik Henrich. Incremental reconstruction of planar b-rep models from multiple point clouds. *Vis. Comput.*, 32(6–8):945–954, jun 2016.

[SH17] Maximilian Sand and Dominik Henrich. Matching and pose estimation of noisy, partial and planar b-rep models. In *Proceedings of the Computer Graphics International Conference*, CGI '17, New York, NY, USA, 2017. Association for Computing Machinery.

[Sha98] Craig M Shakarji. Least-squares fitting algorithms of the nist algorithm testing system. *Journal of research of the National Institute of Standards and Technology*, 103(6):633, 1998.

[Sie] Siemens Digital Industries Software. Parasolid.

[SLM+20] Gopal Sharma, Difan Liu, Subhransu Maji, Evangelos Kalogerakis, Siddhartha Chaudhuri, and Radomír Měch. ParSeNet: A Parametric Surface Fitting Network for 3D Point Clouds, September 2020. arXiv:2003.12181 [cs].

[SLY+21] Zhuo Su, Wenzhe Liu, Zitong Yu, Dewen Hu, Qing Liao, Qi Tian, Matti Pietikäinen, and Li Liu. Pixel difference networks for efficient edge detection, 2021.

[SOZA20] Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P Adams. Sketchgraphs: A large-scale dataset for modeling relational geometry in computer-aided design. *arXiv preprint arXiv:2007.08506*, 2020.

[SSN08] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3d: Learning 3d scene structure from a single still image. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):824–840, 2008.

[SWK07] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[SZPF16] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision (ECCV)*, 2016.

[TDB17] Maxim Tatarchenko, Alexey Dosovitskiy, and Thomas Brox. Octree Generating Networks: Efficient Convolutional Architectures for High-resolution 3D Outputs. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2107–2115, Venice, October 2017. IEEE.

[TRR+19] Maxim Tatarchenko, Stephan R. Richter, Rene Ranftl, Zhuwen Li, Vladlen Koltun, and Thomas Brox. What Do Single-View 3D Reconstruction Networks Learn? In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3400–3409, Long Beach, CA, USA, June 2019. IEEE.

[UyCS+22] Mikaela Angelina Uy, Yen yu Chang, Minhyuk Sung, Purvi Goel, Joseph Lambourne, Tolga Birdal, and Leonidas Guibas. Point2cyl: Reverse engineering 3d objects from point clouds to extrusion cylinders. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[WFAR02] Naoufel Werghi, Robert Fisher, Anthony Ashbrook, and Craig Robertson. Shape reconstruction incorporating multiple nonlinear geometric constraints. *Constraints*, 7(2):117–149, 2002.

[WJC+21] Karl DD Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G Lambourne, Armando Solar-Lezama, et al. Joinable: Learning bottom-up assembly of parametric cad joints. *arXiv preprint arXiv:2111.12772*, 2021.

[WPL+21a] Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4), 2021.

[WPL+21b] Karl DD Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G Lambourne, Armando Solar-Lezama, and Wojciech Matusik. Fusion 360 gallery: A dataset and environment for programmatic cad construction from human design sequences. *ACM Transactions on Graphics (TOG)*, 40(4):1–24, 2021.

[WSL+19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic graph cnn for learning on point clouds, 2019.

[WSP21] Ziqi Wang, Peng Song, and Mark Pauly. State of the art on computational design of assemblies with rigid parts. *Computer Graphics Forum*, 40(2):633–657, 2021.

[WW82] John Warnock and Douglas K. Wyatt. A device independent graphics imaging model for use with raster devices. In *Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '82, page 313–319, New York, NY, USA, 1982. Association for Computing Machinery.

[WXZ21] Rundi Wu, Chang Xiao, and Changxi Zheng. Deepcad: A deep generative network for computer-aided design models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6772–6782, 2021.

[WZS+19] Xiaogang Wang, Bin Zhou, Yahao Shi, Xiaowu Chen, Qinping Zhao, and Kai Xu. Shape2motion: Joint analysis of motion parts and attributes from 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8876–8884, 2019.

[XLJ+24] Xiang Xu, Joseph Lambourne, Pradeep Jayaraman, Zhengqing Wang, Karl Willis, and Yasutaka Furukawa. Brepgen: A b-rep generative diffusion model with structured latent geometry. *ACM Trans. Graph.*, 43(4), July 2024.

[XLX+16] Mingliang Xu, Mingyuan Li, Weiwei Xu, Zhigang Deng, Yin Yang, and Kun Zhou. Interactive mechanism modeling from multi-view images. *ACM Transactions on Graphics (TOG)*, 35(6):1–13, 2016.

[XRSR22] Xu Xianghao, Yifan Ruan, Srinath Sridhar, and Daniel Ritchie. Unsupervised kinematic motion detection for part-segmented 3d shape collections. *ACM Trans. Graph.*, aug 2022.

[XWL+22] Xiang Xu, Karl D. D. Willis, Joseph G. Lambourne, Chin-Yi Cheng, Pradeep Kumar Jayaraman, and Yasutaka Furukawa. SkexGen: Autoregressive Generation of CAD Construction Sequences with Disentangled Codebooks, July 2022. arXiv:2207.04632 [cs].

[YCL+21] Fenggen Yu, Zhiqin Chen, Manyi Li, Aditya Sanghi, Hooman Shayani, Ali Mahdavi-Amiri, and Hao Zhang. Capri-net: Learning compact cad shapes with adaptive primitive assembly, 2021.

[YHY+20] Zihao Yan, Ruizhen Hu, Xingguang Yan, Luanmin Chen, Oliver Van Kaick, Hao Zhang, and Hui Huang. Rpm-net: recurrent prediction of motion and parts from point cloud. *arXiv preprint arXiv:2006.14865*, 2020.

[YKGA17] Jiaxian Yao, Danny M. Kaufman, Yotam Gingold, and Maneesh Agrawala. Interactive design and stability analysis of decorative joinery for furniture. *ACM Trans. Graph.*, 36(2), March 2017.

[YLL+18] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)*, pages 767–783, 2018.

[YUH+24] Yang You, Mikaela Angelina Uy, Jiaqi Han, Rahul Thomas, Haotong Zhang, Suya You, and Leonidas Guibas. Img2cad: Reverse engineering 3d cad models from images through vlm-assisted conditional factorization. *arXiv preprint arXiv:2408.01437*, 2024.

[YWTT22] Farid Yagubbayli, Yida Wang, Alessio Tonioni, and Federico Tombari. Legoformer: Transformers for block-by-block multi-view 3d reconstruction, 2022.

[YYG+23] Yunfan Ye, Renjiao Yi, Zhirui Gao, Chenyang Zhu, Zhiping Cai, and Kai Xu. NEF: Neural Edge Fields for 3D Parametric Curve Reconstruction from Multi-view Images, March 2023. arXiv:2303.07653 [cs].

[YYM+21a] Siming Yan, Zhenpei Yang, Chongyang Ma, Haibin Huang, Etienne Vouga, and Qixing Huang. HPNet: Deep Primitive Segmentation Using Hybrid Representations, October 2021. arXiv:2105.10620 [cs].

[YYM+21b] Siming Yan, Zhenpei Yang, Chongyang Ma, Haibin Huang, Etienne Vouga, and Qixing Huang. Hpnet: Deep primitive segmentation using hybrid representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2753–2762, 2021.

[ZYY+17] Chuhang Zou, Ersin Yumer, Jimei Yang, Duygu Ceylan, and Derek Hoiem. 3d-prnn: Generating shape primitives with recurrent neural networks, 2017.