

# CAD2Sketch: Generating Concept Sketches from CAD Sequences

FELIX HÄHNLEIN, Inria, Université Côte d'Azur, France

CHANGJIAN LI, Inria, Université Côte d'Azur, France and University College London, United Kingdom

NILOY J. MITRA, University College London, United Kingdom and Adobe Research, United Kingdom

ADRIEN BOUSSEAU, Inria, Université Côte d'Azur, France

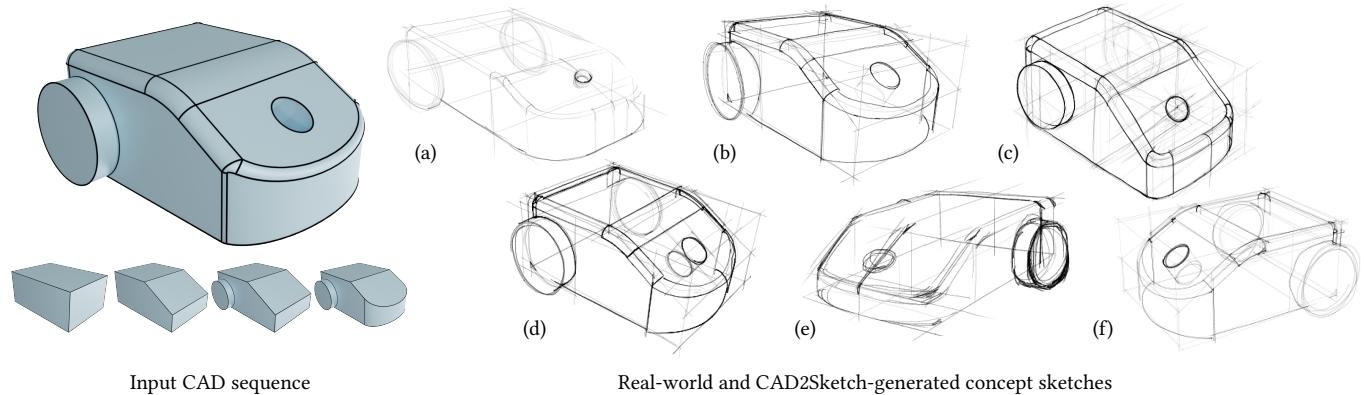


Fig. 1. We introduce CAD2Sketch to synthesize concept sketches directly from CAD sequences. Our work focuses on reducing the domain gap between freehand concept sketches and synthetically generated ones. We achieve this by focusing on generating intermediate lines (i.e., scaffold and auxiliary construction lines used by artists to reduce errors in ‘sighting’ perspective effects) and by recreating drawing style to reflect importance of lines. Here we show three real-world sketches and three synthetic sketches generated with our approach. Can you tell them apart? (See Section 5 for answers.)

Concept sketches are ubiquitous in industrial design, as they allow designers to quickly depict imaginary 3D objects. To construct their sketches with accurate perspective, designers rely on longstanding drawing techniques, including the use of auxiliary construction lines to identify midpoints of perspective planes, to align points vertically and horizontally, and to project planar curves from one perspective plane to another. We present a method to synthesize such construction lines from CAD sequences. Importantly, our method balances the presence of construction lines with overall clutter, such that the resulting sketch is both well-constructed and readable, as professional designers are trained to do. In addition to generating sketches that are visually similar to real ones, we apply our method to synthesize a large quantity of paired sketches and normal maps, and show that the resulting dataset can be used to train a neural network to infer normals from concept sketches.<sup>1</sup>

CCS Concepts: • Computing methodologies → Non-photorealistic rendering; Parametric curve and surface models.

<sup>1</sup>Code and data available at <https://ns.inria.fr/d3/cad2sketch/>

Authors' addresses: Felix Hähnlein, felix.hahnlein@inria.fr, Inria, Université Côte d'Azur, France; Changjian Li, Inria, Université Côte d'Azur, France, University College London, United Kingdom, chjil2011@gmail.com; Niloy J. Mitra, University College London, United Kingdom, Adobe Research, United Kingdom, n.mitra@cs.ucl.ac.uk; Adrien Bousseau, adrien.bousseau@inria.fr, Inria, Université Côte d'Azur, France.

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.

© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM. 0730-0301/2022/12-ART279 \$15.00  
<https://doi.org/10.1145/3550454.3555488>

Additional Key Words and Phrases: computer-aided design, industrial design, non-photorealistic rendering, sketching, line drawing, sketch-based modeling

## ACM Reference Format:

Felix Hähnlein, Changjian Li, Niloy J. Mitra, and Adrien Bousseau. 2022. CAD2Sketch: Generating Concept Sketches from CAD Sequences. *ACM Trans. Graph.* 41, 6, Article 279 (December 2022), 18 pages. <https://doi.org/10.1145/3550454.3555488>

## 1 INTRODUCTION

Industrial designers routinely imagine 3D shapes, and use freehand *concept sketching* to crystallize their mental visions and communicate them to collaborators and clients. Concept sketches often contain a large number of *construction lines* that designers draw to achieve accurate perspective. Early lines represent abstract shape primitives (cuboids, cylinders) that are easy to draw, and that provide anchors to trace finer shape details. Intermediate construction lines also help designers achieve proper alignments and proportions, for instance by locating the midpoint of perspective planes. Unfortunately, existing work in non-photorealistic rendering (NPR) focused on reproducing feature lines of 3D models [Bénard and Hertzmann 2019] and largely ignored the challenge of reproducing the *construction sequences* that designers follow to draw such lines, and that greatly contribute to the unique visual style of concept sketches.

In addition to their aesthetic appeal, concept sketches are drawn to communicate 3D information, and as such form a valuable input for sketch-based modeling systems that hold the promise of

accelerating the transition from freehand sketching to computer-aided design (CAD). But recovering 3D information from concept sketches is very challenging due to the inherent ambiguity of sparse line drawings. As with many ill-posed visual computing tasks, deep neural networks offer a promising solution to map 2D drawings to 3D information after being trained on large datasets of paired drawings and 3D shapes [Zhong et al. 2020a]. But due to the scarcity of real-world sketches, existing methods resort to synthetic drawings generated from 3D models for training. Unfortunately, the *domain gap* between synthetic and real data prevents these neural networks to generalize well to real concept sketches [Gryaditskaya et al. 2019].

Motivated by applications in NPR and in sketch-based modeling, we propose an algorithm to synthesize human-like concept sketches from CAD models. Figure 1 showcases several sketches generated with our method, which are difficult to distinguish from real sketches drawn by professional designers.

A key insight behind our approach is that concept sketching often serves as a preliminary step to CAD modeling, to the point where many similarities exist between the two workflows [Henry 2012]. We distill and leverage these similarities to convert CAD models into synthetic concept sketches. Working with CAD models brings multiple advantages compared to 3D meshes: not only the boundary representation of a CAD model captures precise feature lines of the shape, the underlying *sequence of CAD operations* provides the entire creation history of the shape, which follows similar steps as the ones followed by designers when sketching. CAD sequences also encode additional information about design intent in the form of constraints on alignment and proportion between parts of the shape, which we exploit to generate the corresponding construction lines. Furthermore, several large datasets of CAD sequences recently became available [Koch et al. 2019; Willis et al. 2021], along with libraries to parse such data [Wu et al. 2021], making the generation of synthetic sketches from such data timely.

However, while auxiliary construction lines are critical to position feature lines accurately in freehand drawings, they also introduce significant clutter if employed systematically. This is why designers introduce construction lines with parsimony, deciding as they construct their sketch whether a line is necessary to achieve an accurate and legible depiction of the shape. We formulate this decision problem as a binary optimization. Starting with an overcomplete set of lines generated from CAD operations, our algorithm selects which lines to keep such that the resulting drawing sequence trades construction quality with sketch readability. We also observe that designers reduce visual clutter by varying the opacity of their strokes, using faint strokes for construction lines and darker strokes for the final feature lines. We reproduce this style in the last step of our method by copying the shape and opacity of strokes taken from real-world sketches.

In a study with design teachers, our synthetic sketches were judged to be of similar quality as real sketches. As an application, we show that including stylized construction lines in synthetic training data improves the performance of a normal predictor when tested on synthetic as well as on real sketches. We will release our source code and data to ease reproduction of our method.

## 2 RELATED WORK

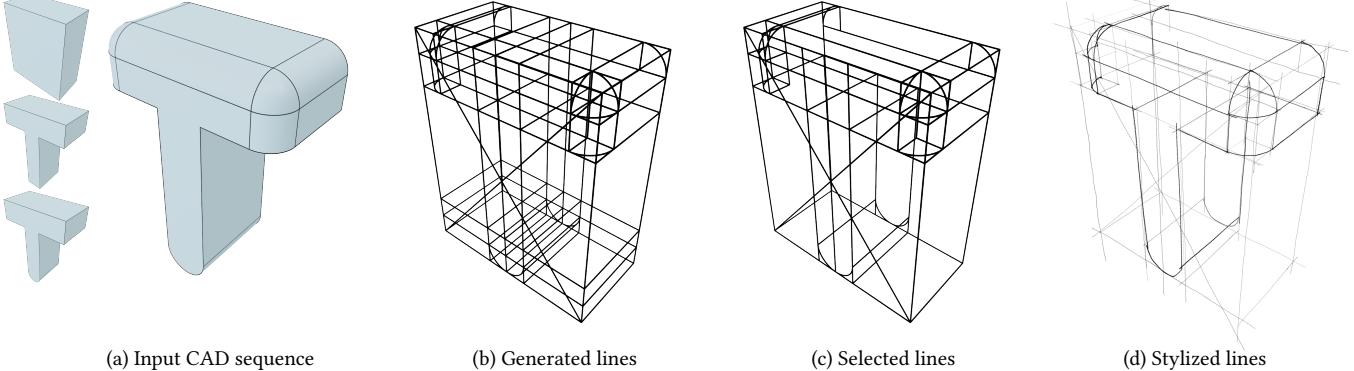
Our work is motivated by data-driven methods for sketch-based modeling, and is inspired by NPR algorithms that seek to render 3D shapes as line drawings. We discuss relevant work in these two domains, and refer readers to recent surveys for more general discussions [Bénard and Hertzmann 2019; Bonnici et al. 2019].

*Sketch-based modeling.* Recovering 3D information from line drawings is an ill-posed task for which deep learning allowed significant progress over the past few years. By training deep learning models on large datasets of paired drawings and 3D shapes, several methods succeeded in predicting depth, normal maps, voxel grids, 3D meshes, parametric shapes [Delanoy et al. 2018; Guillard et al. 2021; Huang et al. 2016; Li et al. 2020, 2018; Lun et al. 2017; Nishida et al. 2016; Su et al. 2018; Zhong et al. 2020a,b]. Because real-world drawings are costly to collect in large quantities, these methods rely on NPR algorithms to generate synthetic line drawings from 3D models. Zhang et al. [2021] alleviate the need for paired data by using differentiable rendering to compare their 3D reconstruction to the input sketch, but their rendering algorithm only extracts the silhouette of the object. While the synthetic drawings produced by existing NPR systems form a reasonable proxy for drawings created by novices [Wang et al. 2021], the recent study by Gryaditskaya et al. [2019] reveals that they lack many of the lines that professional designers draw to construct accurate perspective drawings from imagination, and that a normal predictor trained with synthetic lines fails to interpret professional concept sketches.

On the other hand, construction lines have been exploited with success by algorithms based on geometric optimization, as they provide strong visual cues of parallelism and orthogonality that help lift 2D strokes to 3D [Gryaditskaya et al. 2020; Schmidt et al. 2009b]. Our work is most related to the algorithm by Hähnlein et al. [2022], who leverage properties of concept sketches to reconstruct drawings of symmetric shapes, which they formulate as an assignment problem. While we target the different task of *generating* sketches, we designed our method to reproduce several of the properties they identified, and we take inspiration from their formulation to define some of the terms in our score function.

Gryaditskaya et al. [2019] provide a detailed taxonomy of construction lines, along with an analysis of their usage in a dataset of over 400 industrial design sketches. We augment their study by highlighting similarities between concept sketching principles and CAD modeling operations. The similarities between concept sketching and CAD also inspired the *Sketch2CAD* modeling system by Li et al. [2020]. However, *Sketch2CAD* targets novice users and was trained on synthetic drawings free of construction lines.

*Non-photorealistic rendering (NPR).* We developed our approach by following a popular methodology in NPR, where domain-specific principles are first derived from textbooks and artworks, and are then implemented as algorithms [Agrawala et al. 2011]. Representative instances of this methodology include early work on pen-and-ink [Winkenbach and Salesin 1994], cutaways [Li et al. 2007], and other technical illustrations [Mitra et al. 2010]. We contribute to this family of work by focusing on concept sketches (Section 3), which



**Fig. 2. Overview.** Our method takes as input a sequence of CAD operations, as common in history-based CAD modeling (a). We first generate a set of design-inspired construction lines for each operation (b). We then reduce clutter by selecting a subset of the lines, such that the resulting drawing balances construction quality with readability (c). Finally, we stylize the lines by copying stroke shapes and opacity from a dataset of real design drawings (d).

despite being ubiquitous in industrial design, cannot be faithfully reproduced by existing NPR methods.

A number of algorithms have been proposed to extract and stylize lines from 3D surfaces, such as contours and creases [Bénard and Hertzmann 2019; DeCarlo et al. 2003; Grabli et al. 2004, 2010; Hertzmann and Zorin 2000; Judd et al. 2007; Ohtake et al. 2004]. Yet, many of the construction lines mentioned above do not lie on 3D surfaces, but rather on imaginary geometric primitives that surround the surface of interest. Recent methods based on deep learning using paired [Liu et al. 2021, 2020] or unpaired [Chan et al. 2022] training data share the same limitation: they reproduce well surface contours and creases that convey local surface discontinuities, but they do not model construction lines that convey more global shape abstractions, alignments and proportions (Figure 18).

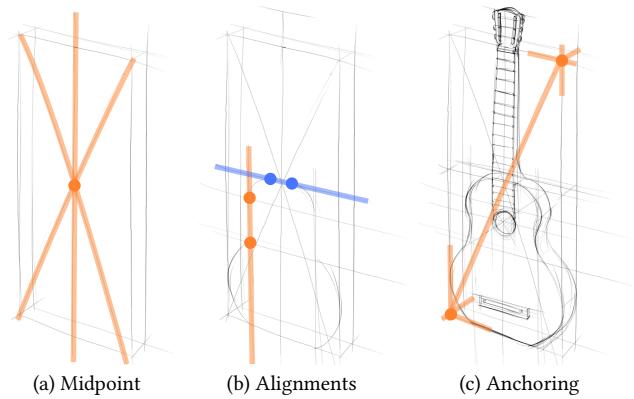
Closer to our goal is the work by Hennessey et al. [2016], who generate sketching tutorials by approximating a segmented 3D mesh with geometric primitives, and by adjusting these primitives such that they exhibit easy-to-construct proportions. Segmented meshes are also used by Liu et al. [2014] to simulate observational drawing techniques of organic shapes. In contrast, we take as input CAD sequences that already contain intermediate geometric primitives, and we leverage the common visual vocabulary of CAD modeling and concept sketching to convert CAD operations into sketching steps not covered by Hennessey et al. [2016]. Our work also relates to the algorithms proposed by Schmidt et al. [2007] and Gori et al. [2017], who take inspiration from sketching techniques to depict smooth surfaces using planar sections and other curvature-aligned curves. But these methods do not consider the intermediate constructions lines that characterize concept sketches in industrial design.

### 3 PRINCIPLES OF CONCEPT SKETCHING

In industrial design, artists typically use freehand drawings during early exploration phases and switch to analytical CAD models in a later prototyping phase. In what follows, we use the term *concept sketches* to refer to freehand perspective drawings, while we reserve the term *profiles* to refer to the parametric 2D shapes that designers create in CAD software to form 3D shapes via extrusion, revolution, or lofting. We adapt this terminology to avoid confusion with the

term *CAD sketches* that is sometimes used instead of *profiles* in the CAD community [Willis et al. 2021].

To bridge the domain gap between synthetically-rendered CAD sequences and artist-drawn concept sketches, we studied the design workflow as documented in sketching guidebooks [Eissen and Steur 2008, 2011; Henry 2012; Hlavács 2014; Robertson and Bertling 2013] and as recorded in datasets of concept sketches [Gryaditskaya et al. 2019] and CAD models [Koch et al. 2019]. The similarities and differences between the two mediums result in unique characteristics of concept sketches and CAD models, both in terms of content and style. On the one hand, important principles of concept sketching relate to common CAD operations [Henry 2012]. On the other hand, while artists are trained to use auxiliary *construction lines* to create perspectively-correct sketches in freehand, such lines are redundant in current CAD workflow, where perspective projection is exact by construction due to analytical handling of the camera projection. We thus need to synthesize these lines to mimic freehand sketching.



**Fig. 3. Freehand drawing in perspective.** Tracing diagonals of a perspective plane allows to locate its midpoint (a). The rectangle can then be divided into halves by tracing a vertical or horizontal line through the midpoint. Vertical and horizontal lines also help align points in perspective (b). When multiple construction lines intersect, they result in salient intersections of high valence through which new lines can be anchored (c). Drawing sequence from [Gryaditskaya et al. 2020].

(i) *Anchoring*. When sketching freehand, it is easier to accurately draw a line if it runs through salient points that are *already* present in the sketch, such as intersections and (imaginary) vanishing points. We refer to such points as *anchors*. As noted by Hähnlein et al. [2022], this anchoring principle results in tightly-connected sketches with many high-valence intersections, and with most lines going through at least two intersections (Figure 3c). Further, these intersections commonly lie at the extremities of the lines to give designers precise targets when tracing strokes.

(ii) *Alignment and proportions*. ‘Sighting’ (i.e., eyeballing) alignment and proportions, without guidance, results in errors even by experienced sketchers (cf., Chapter 8 by Edwards [1979]). Designers often construct auxiliary lines to avoid ‘sighting’ and its associated errors. For accurate alignment, they trace axis-aligned lines to position points that should be aligned horizontally or vertically over perspective planes (Figure 3b). For accurate proportion, they divide perspective planes into equal parts by drawing diagonal lines that intersect at the midpoint of perspective rectangles, and trace vertical or horizontal lines through these midpoints to divide the rectangles in quadrants (Figure 3b, [Eissen and Steur 2011; Hennessey et al. 2016]). In contrast, these alignments and proportions are directly enforced in CAD by specifying constraints on profiles.

(iii) *Coarse-to-fine scaffolding*. CAD users often construct shapes by first assembling basic geometry primitives (e.g., cuboids, cylinders) and then progressively add or subtract parts to form holes, buttons, or rounded edges. A similar coarse-to-fine construction strategy is also common in concept sketching [Eissen and Steur 2011], as illustrated in Figure 4. We follow the terminology of Schmidt et al. [2009b] and Gryaditskaya et al. [2020; 2019] and call such intermediate primitives *scaffolds*. The scaffold planes and intersections provide support to anchor subsequent lines and curves.

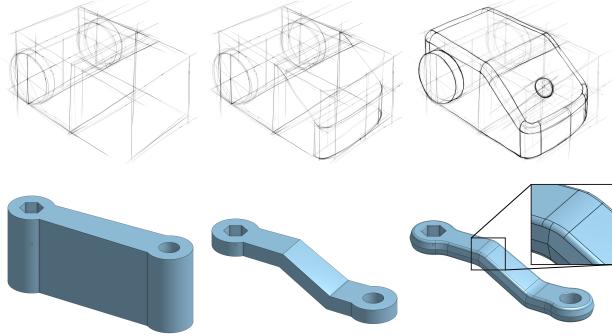


Fig. 4. **Scaffold lines**. Designers draw and model complex shapes in a coarse-to-fine manner, where successive operations refine basic shapes by adding or subtracting parts, and beveling or rounding edges. The lines employed during intermediate steps form a *scaffold* to anchor subsequent lines. Drawing sequence from *OpenSketch* [Gryaditskaya et al. 2019] and CAD modeling sequence after a tutorial in *OnShape* [PTC 2019].

(iv) *Fillets*. Many industrial products exhibit rounded edges and corners, which are created in CAD using the *fillet* operation. We measured that fillet is the third most used CAD operation in the

ABC dataset [Koch et al. 2019], after the creation of 2D profiles and their extrusion. In concept sketching, fillets are constructed by placing local planes perpendicularly to the corner edges to be refined, and by drawing circular arcs within these planes (Figure 5).

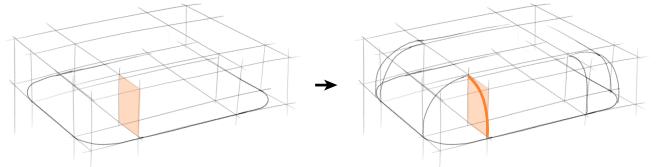


Fig. 5. **Fillets**. Designers draw rounded edges by tracing circular arcs over planes that are perpendicular to these corner edges. Construction sequence after a sketching handbook [Eissen and Steur 2011].

(v) *Projection*. Complex objects are often created by extruding 2D profiles to form holes and protrusions over existing surfaces, to the point where recent work on data-driven CAD modeling relies solely on successive extrusions of profiles to create 3D shapes [Willis et al. 2021; Wu et al. 2021]. In concept sketching, artists sketch accurate extrusions by drawing the profile curves over a scaffold plane and then projecting these curves onto another plane by tracing *projection lines* through each salient point of the curves. Further, to anchor these projection lines in both planes, designers first trace axis-aligned *planar sections* going through each point to be extruded, as illustrated in Figure 6.

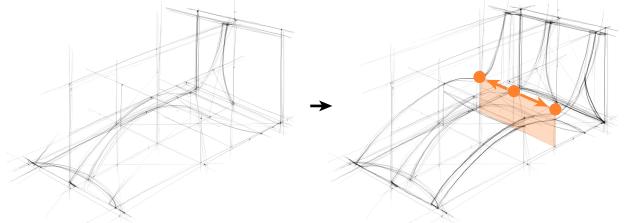


Fig. 6. **Projection**. Starting with the central profile of the chair (left), a designer traces vertical planar sections to project salient points outward (right), which act as anchors to trace the curved boundary of the shape.

(vi) *Readability*. Complete anchoring of all lines and curves may require drawing many construction lines. While such auxiliary construction lines limit sketching error, they often occlude other lines, and, in the process, introduce spurious intersections in the drawing. The resultant clutter hinders shape perception as observers need to distinguish true intersections between 3D lines from occlusions [Gryaditskaya et al. 2020]. Hence, to avoid unnecessary clutter, designers use only essential construction lines and predominantly draw the visible parts of the shape (inset below). As an indication, we measured that in the 107 first-view concept sketches from OpenSketch [Gryaditskaya et al. 2019], 43% of the lines were labeled as visible features, versus 3% of the lines were labeled as hidden features. Further, designers tend to only draw construction lines that help anchor multiple subsequent lines, thus restricting error accumulation.

(vii) *Line precision and opacity.* To engage into a creative flow, design educators recommend practitioners to draw *fast and fearless* [Eissen and Steur 2011; Hlavács 2014]. The resulting swiftly-drawn pen strokes often exhibit subtle imprecision, deviating from their intended trajectory and overshooting beyond intersections. Designers also adjust stroke opacity depending on the uncertainty of the lines they draw. Early in the drawing process, they use faint over-shot lines to lay down the main structure of the shape (inset); subsequently switch to more precise darker lines to draw details; and finish with dark lines to draw definite contours [Eissen and Steur 2011; Henry 2012; Hlavács 2014].



#### 4 RENDERING CAD SEQUENCES AS CONCEPT SKETCHES

The principles we distilled in the previous section inform us on what lines should be drawn for different CAD operations (principle *iii*, *iv*, and *v*) and for different CAD constraints (principle *ii*). Other principles are unique to sketching and tell us how to judge whether a line is well constructed given preceding lines (principle *i*), and how to select and stylize the lines to reduce visual clutter and achieve the look of hand-drawn sketches (principle *vi* and *vii*).

We built on these principles to develop an algorithm composed of three main steps, illustrated in Figure 2. First, we generate candidate lines for every operation of a CAD sequence using procedures proper to each type of operation (Section 4.1). Next, we select a subset of these lines to achieve a good balance between the overall clutter of the drawing and the anchoring of each line (Section 4.2). Finally, we adjust the opacity and shape of each line to achieve the same visual look as real-world concept sketches (Section 4.3).

##### 4.1 Generating lines

This section describes how we generate lines from the corresponding CAD operations and constraints (principles *ii*, *iii*, *iv*, and *v*). Note that successive CAD operations sometimes generate multiple copies of the same line. To reduce computation load, we detect such copies and we only keep the first instance of the line.

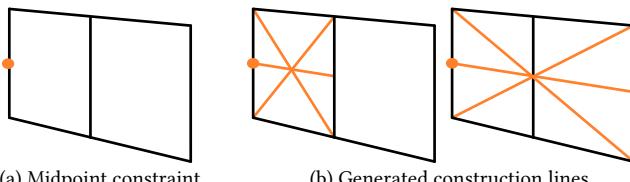


Fig. 7. **Auxiliary lines.** In the presence of a midpoint constraint in a CAD profile (a), we generate an over-complete set of construction lines by considering all rectangles adjacent to the constrained segment (b).

*Alignment and proportions.* We generate lines that depict alignment and proportions whenever such constraints are present in a CAD profile. We depict horizontal and vertical alignments by tracing axis-aligned lines over the plane in which the profile is embedded.

We depict midpoints on profile segments by generating pairs of diagonal lines whose crossing points connect to the target midpoint via an axis-aligned line. We generate these pairs of diagonals from preceding lines by considering all rectangles adjacent to the segment to be divided (Figure 7).

*Coarse-to-fine scaffolding.* We generate scaffolds by extracting all the feature curves produced by intermediate operations along the CAD sequence. To do so, we process one operation at a time and we identify the curves that the operation introduces in the boundary representation (BREP) of the CAD model, as illustrated in Figure 8. We order the lines in the same order as the operations, and we follow the order of the BREP curves within each operation.

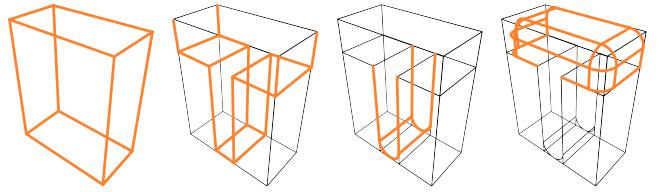


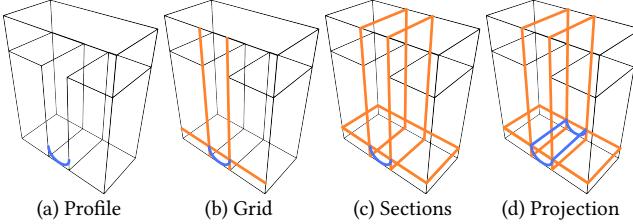
Fig. 8. **Scaffold lines generation.** We generate scaffold lines by detecting the feature curves introduced by successive operations of the CAD sequence (highlighted in orange).

*Fillets.* We generate local planes for each circular arc produced by a fillet operation, which appear at the extremities of the feature curves impacted by the fillet. We first draw the planes and then the circular arcs.

*Projection.* In the presence of an extrusion operation applied on a CAD profile, we generate construction lines that help anchor the profile curves within the sketch (Figure 9). We first create a grid-like structure within the profile plane by tracing the bounding box of the profile, and by tracing vertical and horizontal lines that connect each vertex of the profile to opposite sides of the bounding box. We then connect this grid to other parts of the sketch by tracing planar axis-aligned sections across the shape. We locate the section lines by intersecting the shape with vertical and horizontal planes going through the aforementioned grid lines. Finally, for each vertex of the profile, we trace a projection line connecting the start and endpoint of the extrusion. We deduce the endpoint from the length of the extrusion when it is specified, or we cast a ray from the start point along the extrusion direction until we hit a face of the BREP. We order the lines such that the bounding box is drawn first, followed by the profile curves, the grid lines and the corresponding section lines, and finally the projection lines.

##### 4.2 Selecting lines

The procedures introduced in the previous section complement the feature lines of the CAD sequence with a large number of auxiliary construction lines. Our goal is now to select among all generated lines a subset that yields a well-constructed yet readable depiction of the shape. We formulate this goal as a binary optimization problem, where we associate each generated line  $s_p \in \mathcal{S}$  with a binary selection variable  $s_p$  that we set to 1 to indicate that the line should be



**Fig. 9. Extruding a profile curve.** To extrude a profile curve within a shape (a, blue), we first trace an axis-aligned grid through the curve vertices (b) and connect it to other lines by tracing horizontal and vertical cross-sections (c). The intersections of these sections provide anchoring points to project the profile onto the opposite face (d, blue).

kept in the solution, 0 otherwise (we use bold typeface to represent binary variables throughout the paper). We optimize these selection variables such that the following objective function is maximized:

$$F(\{s_p\}) := F_{\text{visibility}} - \lambda_1 F_{\text{construction}} - \lambda_2 F_{\text{clutter}}, \quad (1)$$

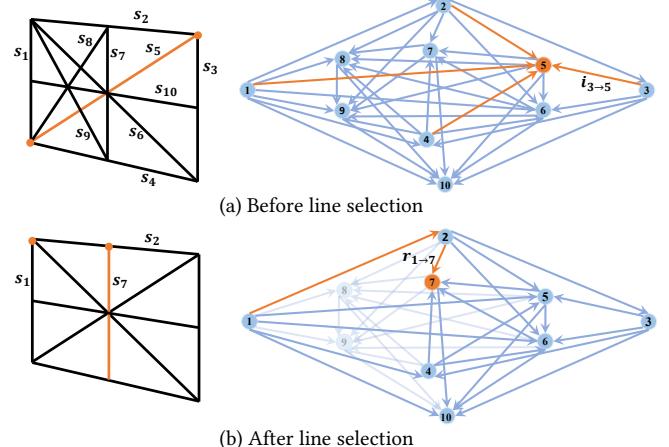
where  $F_{\text{visibility}}$  favors lines that depict visible parts of the shape,  $F_{\text{construction}}$  penalizes poorly-constructed lines, and  $F_{\text{clutter}}$  penalizes visual clutter.

The computational challenge in solving this optimization resides in the fact that many of the principles listed in Section 3 impose inter-dependencies between the lines, such that lines cannot be selected one-by-one, but rather need to be selected in accordance with other lines. Several approaches exist to make such a challenging optimization tractable, such as search algorithms that consider simultaneously different solutions and employ pruning and backtracking strategies to manage the exploration/exploitation trade-off. However, such approaches would require implementing a custom algorithm with dedicated heuristics. Furthermore, inter-dependencies between distant lines would be computationally challenging for such sequential algorithms. Our solution is to formulate the optimization as an integer program where we express dependency between lines via mathematical constraints on their respective binary variables. Importantly, we formulate the objective terms and constraints in a linear form such that the resulting discrete optimization can be efficiently performed using a commercial solver [Gurobi Optimization, LLC 2021].

Before detailing each term of Equation 1, we first introduce the graph data-structure we rely on to reason about the presence of various construction techniques and the corresponding line inter-dependencies in a given selection.

**4.2.1 Intersection graph.** The quality of a construction sequence greatly depends on the intersections that any line forms with preceding lines. We keep track of this information by computing the directed acyclic graph of intersections between the ordered lines generated by the above procedures. In this (dual) graph, each node  $s_p$  represents a line, and each directed edge  $i_{p \rightarrow q}$  represents an intersection between a line  $s_p$  and a subsequent line  $s_q$ . Figure 10a illustrates the intersection graph computed for all generated lines for a simple shape.

For a given assignment of binary variables  $\{s_p\}$ , only the subgraph composed of selected lines matters, as shown in Figure 10b.



**Fig. 10. Intersection graph.** We use intersection graphs to encode ordered construction sequences of scaffold and auxiliary lines. The graph is computed in pre-process over all generated lines (a). In this example, since line  $s_5$  intersects lines  $s_1, s_2, s_3$  and  $s_4$  at its extremities, it has 4 incoming edges in the intersection graph. During optimization, a subset of lines is selected by the integer program (b). The intersection graph restricted to selected lines allows us to test whether a line depends on a preceding line in the solution, as is the case for  $s_7$  that is reachable from  $s_1$  via  $s_2$ .

We can identify whether a line  $s_p$  is used to construct a subsequent line  $s_q$  by detecting whether there exists a path that connects  $s_p$  to  $s_q$  within this sub-graph. If this is the case, we consider that  $s_q$  can be *reached* from  $s_p$  in the solution, which we indicate by setting an auxiliary binary variable  $r_{p \rightarrow q}$  to 1. These auxiliary variables will then serve to evaluate  $F_{\text{construction}}$  (Section 4.2.3).

We compute  $r_{p \rightarrow q}$  by using a recursive formulation. We first consider all lines  $s_q$  that follow and intersect line  $s_p$ , i.e., for which there is an edge  $i_{p \rightarrow q}$  in the intersection graph. We impose that  $r_{p \rightarrow q}$  is selected if  $s_p$  and  $s_q$  are selected with the constraint

$$s_p + s_q \leq 1 + r_{p \rightarrow q}, \forall s_q \in \mathcal{V}_p^1, \quad (2)$$

where  $\mathcal{V}_p^1$  denotes the outgoing 1-ring neighborhood of node  $s_p$ . To satisfy the inequality,  $r_{p \rightarrow q}$  needs to be set to 1 if  $s_p$  and  $s_q$  are set to 1. Note that this constraint alone does not prevent  $r_{p \rightarrow q}$  to equal 1 when  $s_p$  or  $s_q$  are set to 0. But such a configuration is prevented implicitly because it would increase the  $F_{\text{construction}}$  penalty unnecessarily, as detailed in Section 4.2.3. We then recursively identify lines  $s_q$  that depend indirectly on  $s_p$  via intermediate lines  $s_t$  by formulating the constraint

$$s_p + r_{t \rightarrow q} \leq 1 + r_{p \rightarrow q}, \forall s_t \in \mathcal{V}_p^1, \quad (3)$$

which expresses the fact that  $s_q$  can be reached from  $s_p$  if  $s_p$  intersects  $s_t$  and  $s_q$  can be reached from  $s_t$ .

**4.2.2 Visibility.** We favor drawings that depict visible parts of the shape by measuring how much of each feature line is occluded from the camera. We compute the partial visibility  $v(s_p) \in [0, 1]$  for each feature line in preprocessing by shooting rays from the camera toward samples regularly distributed over the line and by counting the percentage of rays that reach the line without intersecting any face of the BREP. We then compute the visibility term of a solution

by summing  $v(s_p)$  over all selected lines:

$$F_{\text{visibility}} = \sum_p s_p v(s_p). \quad (4)$$

**4.2.3 Construction quality.** As detailed in Section 3, designers employ several techniques to construct perspective sketches, although they use these techniques with parsimony to avoid clutter. Instead of enforcing each technique strictly, we encapsulate all techniques into a single function that penalizes lines that are only supported by few techniques. Formally, we introduce auxiliary binary variables which we set to 1 to indicate the presence of specific construction techniques in support of a given line  $s_p$ , and we count how many of these techniques are present. Three such variables indicate the presence of anchoring techniques ( $w_p$ ,  $f_p$ , and  $h_p$ ), two variables indicate the presence of alignments and midpoints ( $a_p$  and  $c_p$ ), and one variable indicates the presence of projections ( $p_p$ ). We test the presence of a particular technique by implementing linear constraints on the binary variables associated with the lines involved in the technique.

Furthermore, lines that appear early in the construction sequence often serve to anchor subsequent lines. Hence, a poorly-constructed line negatively impacts all lines that are anchored on it. Based on this observation, we weigh the construction penalty of a line  $s_p$  proportionally to the number of lines that can be reached from it, which we express as  $w_{\text{dependency}}(s_p) = \sum_q r_{p \rightarrow q}$ .

Given these terms, we formulate the construction penalty as

$$F_{\text{construction}} = \sum_p \left( w_{\text{dependency}}(s_p) (6 - w_p - f_p - h_p - a_p - c_p - p_p) \right). \quad (5)$$

**Anchoring.** We express the degree of anchoring of a line by testing three complementary criteria, which follow principle (i) from Section 3. First, we consider that a line is *weakly anchored* if it intersects at least one preceding line, in which case we set a binary variable  $w_p$  to 1. Second, we consider that a line is *fully anchored* if it intersects preceding lines at its start and end points, in which case we set a binary variable  $f_p$  to 1. Finally, we favor lines that go through at least one intersection of valence 3 or higher, for which we set a binary variable  $h_p$  to 1.

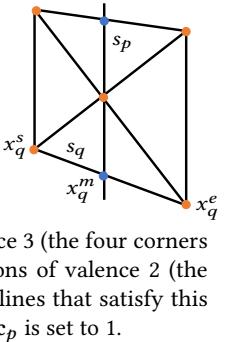
To implement the latter criteria, we consider the set of intersections  $\{i_p^k\}$  that a line  $s_p$  forms with any preceding lines, and we associate each intersection with a binary variable  $h_p^k$  that indicates whether at least 2 selected lines that precede  $s_p$  go through that intersection. Denoting  $S_p^k$  the set of all lines that precede  $s_p$  and go through  $i_p^k$ , we follow the *Big M method* [Griva et al. 2008] and impose the constraint

$$\sum_{q \in S_p^k} s_q \geq 2 - M(1 - h_p^k), \quad (6)$$

with  $M$  a large constant, set to 100 in our implementation. Thanks to this inequality, at least 2 lines of  $S_p^k$  must be selected for  $h_p^k$  to be selected and contribute to the anchoring score. We then test if at least one intersection of high valence exists along  $s_p$  by imposing the constraint  $\sum_k h_p^k \geq h_p$ .

**Alignment.** A point on a line  $s_p$  is well-aligned with a point on a preceding line  $s_q$  if at least one third axis-aligned line  $s_t$  passes through the two points to be aligned (see inset). Let us denote as  $x_p^i$  and  $x_q^j$  the two points that should be aligned. In preprocessing, we detect all axis-aligned lines that precede  $s_p$  and that pass through the two points, denoted by the set  $\mathcal{A}_{pq}^{ij}$ . We repeat that process for all points along  $s_p$  that are subject to alignments, which we denote as the set  $\mathcal{X}_p$ . We then set a binary variable  $a_p$  to 1 if at least one line from  $\mathcal{A}_{pq}^{ij}$  is selected for each  $x_p^i$ , which we ensure with the constraint

$$\sum_{t \in \mathcal{A}_{pq}^{ij}} s_t \geq a_p, \forall x_p^i \in \mathcal{X}_p. \quad (7)$$



**Midpoints.** Tracing a line  $s_p$  through the midpoint of a segment  $s_q$  requires a pattern of seven lines, where four lines form a rectangle adjacent to  $s_q$ , two lines form the diagonals of that rectangle, and one line goes over  $s_p$  to join the crossing point of the diagonals with the midpoints of  $s_q$  and its opposite side (see inset). A characteristic feature of this pattern is that it contains five intersections of valence 3 (the four corners and the crossing point) and two intersections of valence 2 (the two midpoints). Our goal is to select seven lines that satisfy this condition, in which case the binary variable  $c_p$  is set to 1.

Our line generation procedure synthesizes multiple lines that can form the necessary construction sequence (Figure 7). Out of all generated lines, we first identify the ones that precede  $s_p$  and that are eligible to form one of the seven lines of the pattern. Denoting  $x_q^s$  and  $x_q^e$  the start and end points of the segment to be divided, and  $x_q^m$  its midpoint, we look for all lines that are coincident to  $s_q$  and that extend beyond  $x_q^s$  and  $x_q^e$ , all lines parallel to  $s_q$ , all lines going through  $x_q^s$ ,  $x_q^e$  or  $x_q^m$  and that are perpendicular to  $s_q$ , and all diagonal (i.e., non-axis-aligned) lines going through  $x_q^s$  or  $x_q^e$ . We denote the resulting seven sets of lines as  $\mathcal{S}_p^{1..7}$ .

We first select one and only one line in each set to form a pattern of seven lines. To achieve this goal, we associate each line  $s_q$  of each set with an auxiliary binary variable  $\hat{s}_q$ , and we constrain:

$$\sum_{q \in \mathcal{S}_p^l} \hat{s}_q \leq 1, \forall l \in \{1..7\}. \quad (8)$$

We then make sure that the lines forming the pattern are only selected if they also appear in the final solution:

$$\hat{s}_q \leq s_q, \forall s_q. \quad (9)$$

We constrain that seven lines are selected to form a complete pattern:

$$\sum_{q \in \mathcal{S}_p^{1..7}} \hat{s}_q \geq 7 - M(1 - c_p). \quad (10)$$

We then test the valence of the intersections formed by the seven lines. Denoting  $\mathcal{I}_p^{1..7}$  the set of all possible intersections between all lines in  $\mathcal{S}_p^{1..7}$ , we associate each intersection  $i^k \in \mathcal{I}_p^{1..7}$  with a

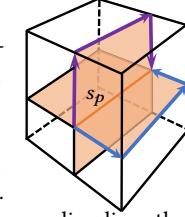
binary variable  $i^k$  set to 1 if at least two selected lines go through this intersection, and with a binary variable  $h^k$  set to 1 if at least three selected lines go through this intersection. These variables allow us to implement the condition that seven intersections are selected, including five of valence 3 or higher:

$$\sum_{i^k \in I_p^{1..7}} i^k \geq 7 - M(1 - c_p), \quad (11)$$

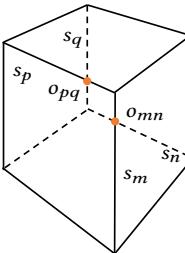
$$\sum_{i^k \in I_p^{1..7}} h^k \geq 5 - M(1 - c_p). \quad (12)$$

*Fillets.* Our line generation procedure ensures that each fillet arc comes with four lines forming a local scaffold plane. Yet, other lines might coincide with these four lines, and our goal is that at least one line is selected for each side of the plane. We express this goal by defining an alignment constraint for each pair of vertices that form the four edges of the supporting plane, and by treating these constraints as described in the previous paragraph. This strategy can yield more economical line usage by selecting a single line passing through multiple local scaffold planes.

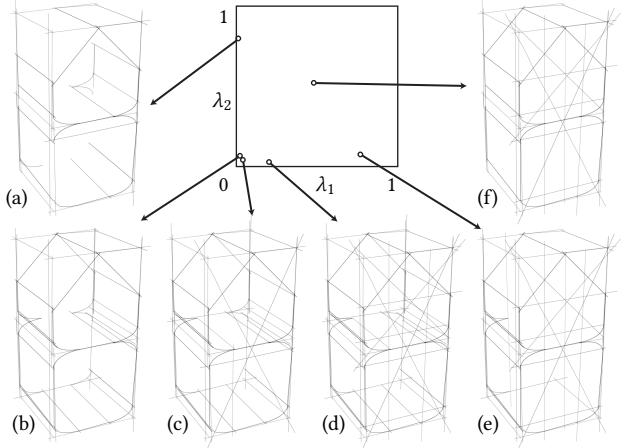
*Projection lines.* A projection line is well constructed if there exists a path of lines that form a planar, axis-aligned section connecting the start and end points of the projection line (see inset, where the blue or the purple path can serve to anchor the orange projection line  $s_p$ ). For a given projection line, we first identify all preceding lines that lie in any of the two axis-aligned planes that go through the projection line. For each plane, we then compute the directed graph of intersections between all lines it contains. In each of the two intersection graphs, we test the existence of a path of selected lines that connects the two lines intersected by the projection line at its extremities. We use the formulation from Problem IP3 in Magnanti and Mirchandani [1993] to implement this path search as an integer program. If such a path exists within one of the two intersection graphs, we set a binary variable  $p_p$  to 1.



**4.2.4 Visual clutter.** We penalize clutter by counting the number of occlusions between lines once projected in the image plane. We detect occlusions  $o_{pq}$  between all generated lines in preprocessing, and we set a binary variable  $o_{pq}$  to 1 when the corresponding lines  $s_p$  and  $s_q$  are selected. To avoid a quadratic constraint of the form  $o_{pq} = s_p s_q$ , we identify occlusion by imposing the linear constraint  $s_p + s_q < 2 + M o_{pq}$ . If both  $s_p$  and  $s_q$  get selected,  $o_{pq}$  must also get selected to ensure the inequality. The number of occlusions is then given by  $\sum_{p,q} o_{pq}$ .



In complement to this measure of occlusion, we also reduce clutter by penalizing the use of unnecessary construction lines. We achieve this goal by counting the number of construction lines in the solution,  $\sum_{p \in S^c} s_p$ , where  $S^c$  denotes the set of all generated auxiliary construction lines.



**Fig. 11. Balancing between better anchoring versus reduced clutter.** Varying  $\lambda_1$  and  $\lambda_2$  allows to generate sketches with different amount of construction lines and clutter. For low values of  $\lambda_1$ , few construction lines are present (a,b,c). Increasing  $\lambda_1$  trades hidden feature lines for construction lines (c vs. e). Increasing  $\lambda_2$  reduces clutter, either by removing lines that yield occlusions (a vs. b) or by removing hidden lines (f vs. e). For this particular model, the automatic procedure described in supplemental materials considers that the best parameter setting is (e).

Combining these two terms gives the score function:

$$F_{\text{clutter}} = \sum_{p,q} o_{pq} + \epsilon \sum_{p \in S^c} s_p, \quad (13)$$

where we give a small weight  $\epsilon = 1e-3$  to the second term to act as a weak regularization.

**4.2.5 Optimization.** We maximize Equation 1 subject to the listed constraints (Equations 2, 3, 6, 7, 8, 9, 10, 11, and 12) using the commercial solver *Gurobi* [Gurobi Optimization, LLC 2021]. We provide the complete set of equations as supplemental materials. Figure 11 illustrates the visual impact of varying the  $\lambda_1$  and  $\lambda_2$  parameters, which allow users to adjust the amount of construction lines and clutter. We describe in supplemental materials a procedure to tune these parameters automatically for an input CAD sequence.

### 4.3 Stylizing lines

Designers depict lines by tracing pen strokes on canvas. The shape of these strokes, their approximate trajectory, as well as their opacity all contribute to the visual style of concept sketches. We adopt a data-driven approach to reproduce this style, using the 107 first-view concept sketches of OpenSketch [Gryaditskaya et al. 2019] as a stroke library on which we compute statistics about various visual effects. Depending on the target application, we can restrict the library to specific participants of OpenSketch to reproduce their individual style, as shown in Figure 12.

*Stroke aim.* When sketched quickly, pen strokes often deviate from their intended trajectory. We compute an estimate of the overall imprecision of a real sketch by considering all straight strokes

that converge towards one of the vanishing points, and by measuring the angular deviation between each stroke and the line joining the starting point of the stroke to its vanishing point. We fit a normal distribution on these measurements, which we sample to apply similar deviation to our synthetic strokes. We use the calibrated perspective cameras provided in OpenSketch to locate the three dominant vanishing points for each sketch.

**Stroke overshooting.** We define overshooting as the ratio between the length of a line and the length of the stroke used to depict that line. We measure this ratio on strokes of OpenSketch by assuming that the line that a given stroke depicts joins the first and last intersections along the stroke. Figure 13 plots the value of this ratio as a function of the relative position of the stroke in the sketching sequence, averaged over all sketches of a participant of OpenSketch and discretized over 100 bins. To reproduce an artist’s overshooting style, we extend each stroke by the average ratio measured at the same relative position in the sequence.

**Stroke shape.** Pen strokes in real drawings are never perfectly straight, or elliptic, but rather exhibit subtle imprecision in their execution. Inspired by the sketch stylization approach by Berger et al. [2013], we reproduce the shape of real pen strokes by copying and transforming strokes from OpenSketch. For each synthetic line, we select the  $k = 10$  best-fitting strokes by performing an ICP registration against all strokes in the library, and we randomly select one of these strokes for rendering. Compared to selecting the best stroke overall, taking one of the  $k$  best strokes yields more plausible sketches as it introduces a small amount of imperfection.

**Stroke opacity.** Designers draw strokes using different levels of pressure depending on the line type [Eissen and Steur 2011; Henry 2012], and pen pressure also varies along an individual stroke as the pen is pressed and lifted. We reproduce these effects in two steps. We first model the distribution of opacity per line type in OpenSketch as a normal distribution, and we sample from these distributions to obtain an opacity value for each synthetic stroke depending on its type (Figure 14). We then generate variations of opacity along each stroke by copying opacity profiles of real

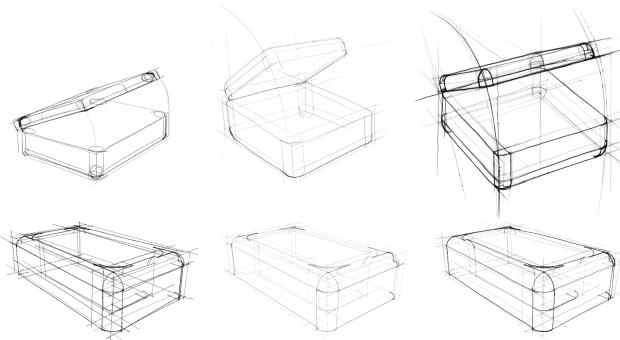


Fig. 12. **Stroke statistics.** Given labeled sketches of professional designers from OpenSketch [Gryaditskaya et al. 2019] (top), we measure stroke statistics (opacity, overshooting) proper to individual designers to reproduce their style using CAD2Sketch (bottom).

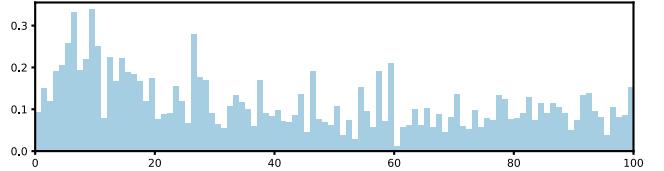


Fig. 13. **Overshooting statistics.** Ratio of overshooting as a function of the relative position of a stroke in the sketching sequence, measured for a specific participant of the OpenSketch dataset [Gryaditskaya et al. 2019]. The plot reveals that this participant overshoots more the early strokes.

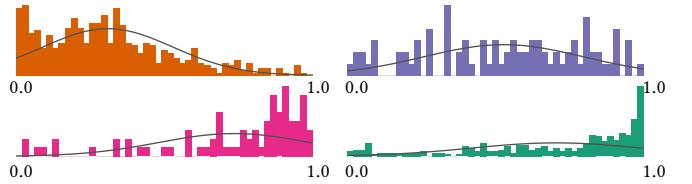


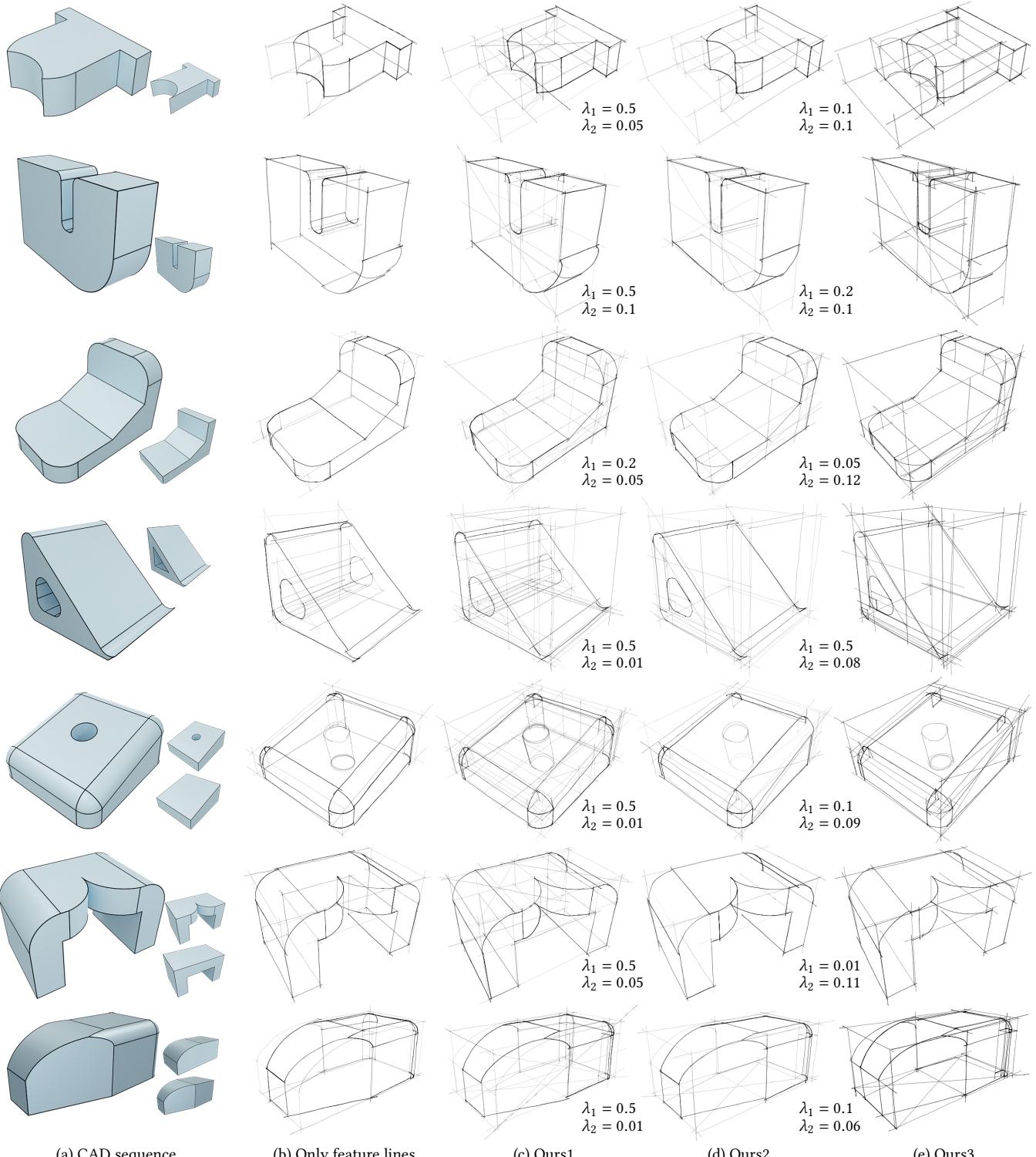
Fig. 14. **Stroke opacity statistics.** Histogram of stroke opacity for different line types for a participant of OpenSketch (**Scaffold lines**, **Hidden feature lines**, **Visible feature lines**, **Silhouette lines**). We fit a normal distribution over each histogram and sample from these distributions to assign opacity to synthetic strokes according to their type.

strokes from OpenSketch. For a given synthetic stroke, we select the opacity profile whose median value is closest to the value sampled in the first step. Care must be taken when measuring opacity in real-world sketches, because lines that appear dark might actually been drawn with multiple faint, overlapping strokes. In contrast, when generating our synthetic sketches, we use a single stroke to trace a given line. We account for this characteristic of real sketches by measuring their opacity in raster form, where the value of a pixel corresponds to the accumulated opacity of all strokes running over that pixel. We assign the opacity value of a pixel to the last stroke drawn over that pixel, as it corresponds to the amount of darkness that the designer intended to obtain when tracing that stroke.

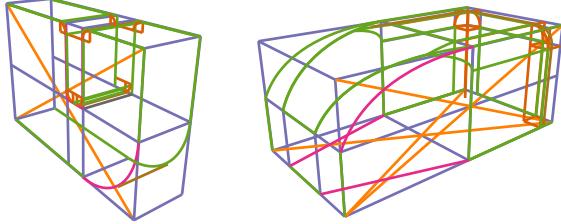
## 5 EVALUATION AND RESULTS

Figure 1 combines real sketches from OpenSketch [Gryaditskaya et al. 2019] with synthetic sketches generated by our method. Sketches (b), (d) and (f) are the synthetic ones. Figure 15 presents a gallery of sketches generated from CAD sequences in the ABC dataset [Koch et al. 2019]. For each model, we show three sketches produced by our method with different parameter settings and stroke styles, along with one sketch generated by a naive method that only renders all the intermediate feature lines of the CAD sequence. We also provide animated drawing sequences of some of our results as a supplemental video, which highlights the construction steps produced by our method. Finally, Figure 16 illustrates the various types of lines generated by our system for several CAD models.

Table 1 provides runtime statistics for the main steps of our method on three representative examples. The total time remains in the order of minutes for all models we tested, varying with the number of generated lines, the number of intersections, and the viewpoint. Our implementation of stylization is slow because it



**Fig. 15. Results gallery.** ABC models rendered as concept sketches using CAD2Sketch. Compared to a naive method that only draws feature lines of each CAD operation (b), our method generates and selects scaffold and auxiliary lines to support the construction of accurate alignment, proportions, and projections (c,d,e). The balance between construction lines and clutter can be adjusted via the parameters  $\lambda_1$  and  $\lambda_2$  (c vs. d), and lines can be rendered according to the style of different artists (d vs. e).



**Fig. 16. Different line types.** We show the different line types generated for the results from Fig. 15 2nd row (c) and Fig. 15 last row (c). For illustration purposes, we show the synthetic lines before stylization. Also, since a line can have been generated by multiple techniques, here we only show the *first technique* which created a specific line. The color code is: **Profile lines**, **Midpoint lines**, **Projection lines**, **Fillet lines**, **Grid lines**, **Feature lines**.

**Table 1. Performance statistics.** Runtime of the different steps of our method, in seconds. Note that the complexity varies with the number of lines, the chosen viewpoint, as well as with the number of intersections.

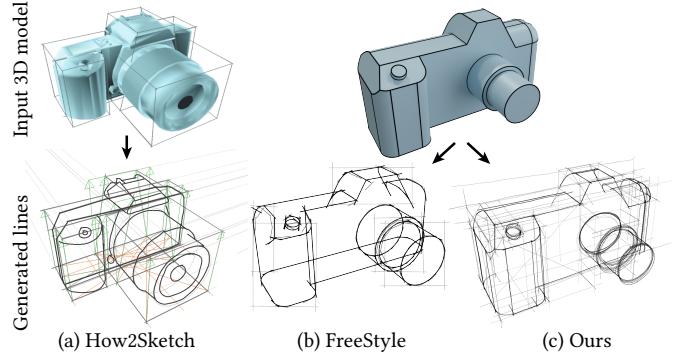
Shape	#Lines	Line gen.	Inter. graph calculation	Integer program	Stylization
Fig. 11	143	13s	2s	2s	26s
Fig. 12	263	26s	11s	15s	48s
Fig. 1	374	52s	6s	4s	72s

performs an ICP registration between each line and all strokes in the library, which could be accelerated by using shape descriptors [Berger et al. 2013].

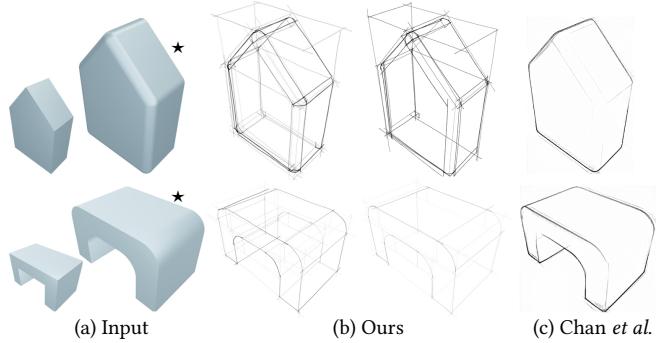
### 5.1 Comparison with non-photorealistic rendering and with image stylization

Our work follows the tradition of NPR research as it attempts to reproduce principles of construction lines – an artistic practice that is ubiquitous in design sketching and yet has been largely overlooked by prior work on line drawing generation. Figure 17 provides a visual comparison to the two NPR systems we felt closest to our work in their attempt to synthesize construction lines. *FreeStyle* [Grabli et al. 2004, 2010] emulates 2D construction techniques by approximating curved strokes with straight lines and geometric primitives (squares, circles)<sup>2</sup>. The resulting lines do not resemble the 3D construction techniques we observed in real-world concept sketches. *How2Sketch* [Hennessey et al. 2016] generates step-by-step tutorials that are easy to follow by novices, but these tutorials only include a subset of the lines we generate (scaffolds, alignments and proportions). Furthermore, *How2Sketch* does not consider the clutter produced when drawing the construction lines generated for all steps of the tutorial. In summary, our system is the first to target the generation of well-constructed yet readable 3D concept sketches that look like real drawings.

<sup>2</sup>We used the implementation of FreeStyle available in Blender, <https://docs.blender.org/manual/en/latest/render/freestyle/introduction.html>

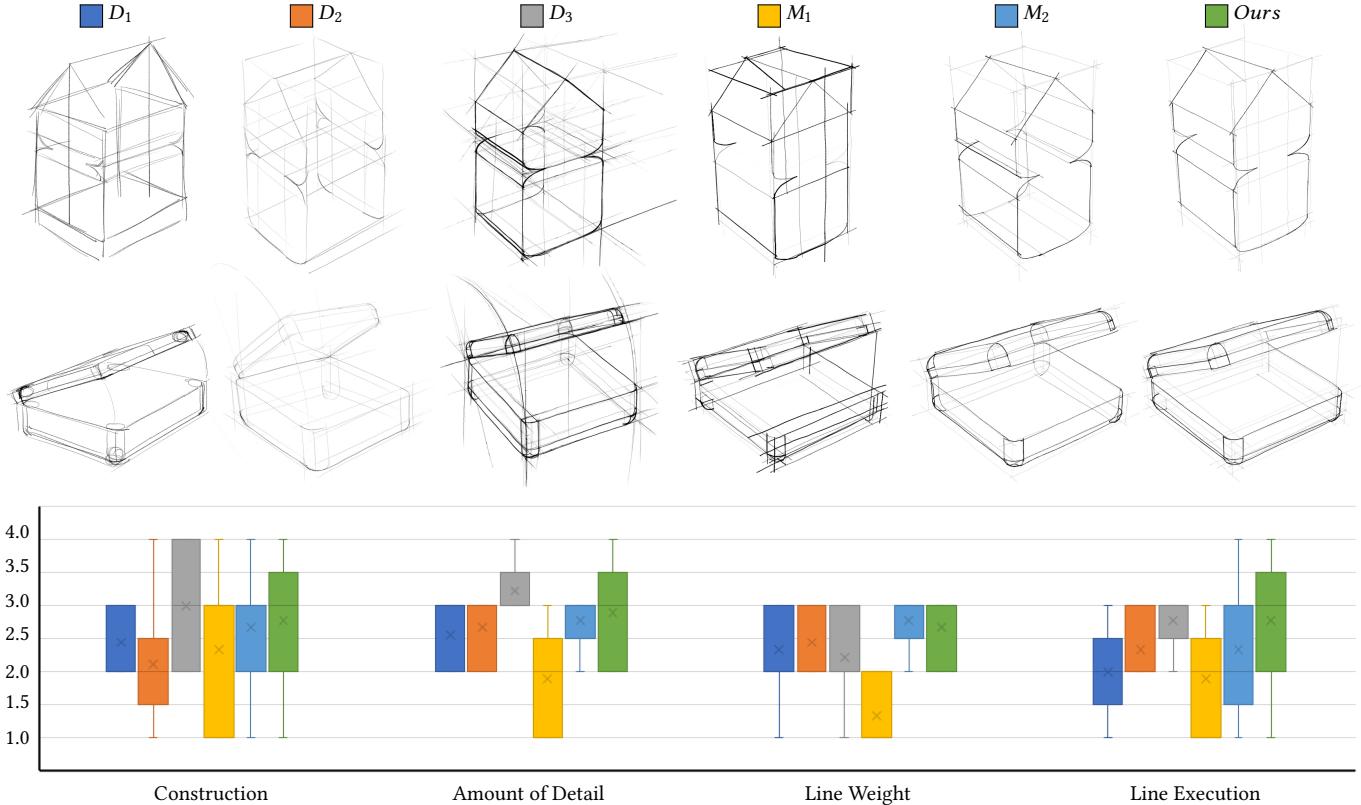


**Fig. 17. Comparison with NPR methods.** Existing NPR systems focus on generating easy-to-follow step-by-step drawing tutorials rather than a readable final sketch (a, input 3D model and generated lines from [Hennessey et al. 2016]), or only generate 2D construction lines (b, [Grabli et al. 2010]). Ours is the first to balance constructability of 3D lines with readability (c).



**Fig. 18. Comparison with translation network.** Comparison with image-to-image translation [Chan et al. 2022]. Given a CAD sequence (a), our method can generate construction lines in different styles (b). In contrast, a convolutional neural network trained to translate images into sketches only traces salient feature lines of the shape (c). Note that we feed the neural network with a shaded image of the CAD model (marked with a ★).

Our approach implements longstanding principles of concept sketching in the form of a line generation and optimization procedure. This methodology contrasts with recent work in image stylization that relies on machine learning to translate images to line drawings without an explicit formalization of the underlying principles. We compare to the state-of-the-art image-to-image translation method by Chan et al. [2022] in Figure 18. Similarly to ours, this method has been developed to generate drawings that reproduce the visual style of OpenSketch while conveying well the geometry of the depicted objects, which the authors achieved by combining an adversarial loss for style and a depth estimation loss for geometry. However, their method takes photographs as input rather than CAD sequences. We thus perform the comparison by feeding their pre-trained method with shaded renderings of the CAD model (Fig. 18a★). While their method captures salient feature lines of the shape and reproduces the stroke texture, it does not generate construction lines. We hypothesize that this limitation of their method is due to the fact that construction lines often lie away from the input image edges, and as such are difficult to model by the restricted receptive



**Fig. 19. Evaluation study by design teachers.** Real and synthetic sketches used in our evaluation study, along with a boxplot summarizing the ratings aggregated over shapes and evaluators. The real-world sketches were drawn by designers selected from OpenSketch ( $D_1$ ,  $D_2$  and  $D_3$ ), while the synthetic sketches were generated by variants of our method ( $M_1$  with random opacity,  $M_2$  without auxiliary construction lines nor line selection, and *Ours*). In the plot, the top and bottom of the box correspond to the first and third quartile, the whiskers correspond to the min and max scores, and the cross denotes the mean score. Our method (green) is on par with real designers, and even outperforms some of them in terms of construction, amount of detail and line execution.

field of generative convolutional neural networks. The temporal dependency inherent to construction lines is also challenging to infer from a single image.

## 5.2 Comparison with real-world sketches

Our primary goal is to generate concept sketches that look like real ones. To evaluate whether we achieved this goal, we asked three design teachers to rate a set of 18 sketches, without telling them that half of the sketches were synthetically generated. All three evaluators were experienced in this task, one being a sketching instructor for more than 7 years ( $R_2$ ), the two others being teaching assistant for 4 and 5 years respectively ( $R_3$  and  $R_1$ ). They rated each sketch according to four criteria that they commonly use when evaluating students, and that reflect the key features of our method: 1. Construction, 2. Amount of detail, 3. Line weight, 4. Line execution. Each criteria was evaluated on a 4-point scale that reflects the expected level of a design student at the end of a 5-years study curriculum, where 1 corresponds to “clearly below graduation level” and 4 corresponds to “better than graduation level”. The evaluators spent between 30 minutes and one hour to perform this task.

Figure 19(top) shows 12 of the sketches used in the study, and the 3 other real sketches appear in Figure 1. To meaningfully interpret

the evaluation results, we need to be able to assess how the teachers’ grades vary among (a) sketches from the same designer/algorithim and (b) sketches from different designers/algorithms depicting the same object. Given these constraints and the available sketches from OpenSketch, we could only choose among 6 designers who all drew the same 12 objects. We chose 3 designers with different drawing styles (choice of lines and line stylization), denoted as  $D_1$ ,  $D_2$  and  $D_3$ . We then chose 3 objects that we could re-model using CAD operations that are supported by our system. In addition to the three real sketches, we generated three other sketches for each object using variants of our method. The first variant – denoted  $M_1$  – contains our selected feature and construction lines but rendered with random opacity and overshooting. The second variant –  $M_2$  – renders the lines using our stylization procedure, but only displays intermediate feature lines (i.e., scaffolds) without any auxiliary construction lines and without selecting a subset of lines according to Equation 1.  $M_1$  and  $M_2$  serve as naive baselines compared to our full approach – denoted *Ours* – that includes the generation, selection and stylization of various auxiliary construction lines to achieve a balance between construction quality and visual clutter. We used the automatic procedure described in supplemental materials to

select the values of  $\lambda_1$  and  $\lambda_2$  in our results, and we use sketches from  $D3$  to build the stroke library used for stylization.

Note that we do not model the error that designers do when tracing lines that are not well anchored [Schmidt et al. 2009a]. As a result, the sketches produced by  $M2$  exhibit as perfect a perspective as ours, even though it is likely that they would suffer from more distortion if drawn by hand, as was reported by Gryaditskaya et al. [2019] who measured a positive correlation between sketch accuracy and presence of construction lines in OpenSketch.

Figure 19(bottom) summarizes the ratings obtained by each real-world designer and by each variant of our method, aggregated over all 3 shapes and 3 raters. Overall, the method with random stylization ( $M1$ ) is judged inferior compared to other methods and to real-world sketches. In contrast, both  $M2$  and *Ours* are on par with real designers, with *Ours* being slightly superior even over some of the designers in terms of construction and amount of detail. Our method was also judged stronger than real designers in terms of line execution, which might be due to the lack of a realistic distortion model as mentioned above.

The three evaluators sometimes disagreed on specific cases, such as for our vacuum cleaner that  $R3$  judged “*well constructed with the right amount of lines*”, while  $R1$  criticized that vertical lines are not vertical, which might be due to our choice of viewpoint. The comments by  $R1$  and  $R3$  also highlight the strengths and weaknesses of our method. For instance,  $R1$  noted that our sketch of the house is missing midpoint lines, which is actually due to the fact that the corresponding constraint was not present in the version of the CAD model we used for the study. For the box,  $R3$  appreciated that “*rounded edges are constructed pretty well*” in our result, but both  $R1$  and  $R3$  would have liked to see an elliptic arc to clarify the movement of the lid, as can be seen in the sketches by  $D1$ ,  $D2$  and  $D3$ . When commenting on line weight for our results,  $R3$  appreciated that “*the drawing uses an appropriate distinction between construction and finalized shape*” and  $R1$  stressed the “*clear division of construction lines and product outline*”, although  $R1$  suggests the need for more variation as “*some line weight difference can be made at certain edges*”. We provide the full evaluation grids by the three evaluators as supplemental materials, along with a visualization of their ratings for each sketch.

### 5.3 Application to normal map prediction

Our research is partly motivated by the need to synthesize large quantities of synthetic data to train deep learning models to interpret real-world sketches. We demonstrate this application of our work on the task of predicting normal maps from concept sketches, by which we also evaluate the impact of our method’s different ingredients.

*Network structure.* We built our neural network implementation upon Sketch2Normal [Su et al. 2018]. The only changes we made was to encode the sketch as a 1-channel gray-level image rather than a 3-channel color image, and to remove the input channel they used to provide normal hints.

*Dataset generation.* We implemented a simple shape grammar to generate synthetic CAD sequences, which we detail in supplemental materials. We used this grammar to create 2000 models for training,

and 300 for testing, some of which can be seen in Figure 20. For each model, we generated 3 sketches from different 3/4 viewpoints to obtain a total of 6000 paired sketches and normal maps. To avoid the cost of testing many parameter values, we fixed  $\lambda_1 = 0.5$  and  $\lambda_2 = 0.01$ , which we found to produce more construction lines.

In complement to this synthetic data, we also generated a dataset of 108 sketches of ABC models [Koch et al. 2019] to test the generalization of the trained neural networks over real-world shapes. We selected these models to have a similar aspect ratio as ours, and to not involve imported geometry, nor the OnShape *shell* feature that converts solids into surfaces of constant thickness.

Finally, we also created two test sets from OpenSketch [Gryaditskaya et al. 2019], each composed of 6 sketches drawn by the same designers  $D1$  and  $D3$  as in Section 5.2. We selected these sketches to represent shapes similar to the ones produced by our shape grammar, rather than freeform surfaces we do not support. These test sets allow us to evaluate the generalization of the trained networks to real-world sketches.

*Ablation study.* We generated training datasets with several variants of our method to evaluate the impact of including construction lines and their selection, as well as the impact of stroke stylization:

- $B_0$ . Only the feature lines of the final shape, rendered with stylized strokes. The resulting line drawings are similar to the ones produced by standard NPR algorithms
- $B_1$ . All intermediate feature lines (i.e., scaffolds), no auxiliary lines, rendered as black lines.

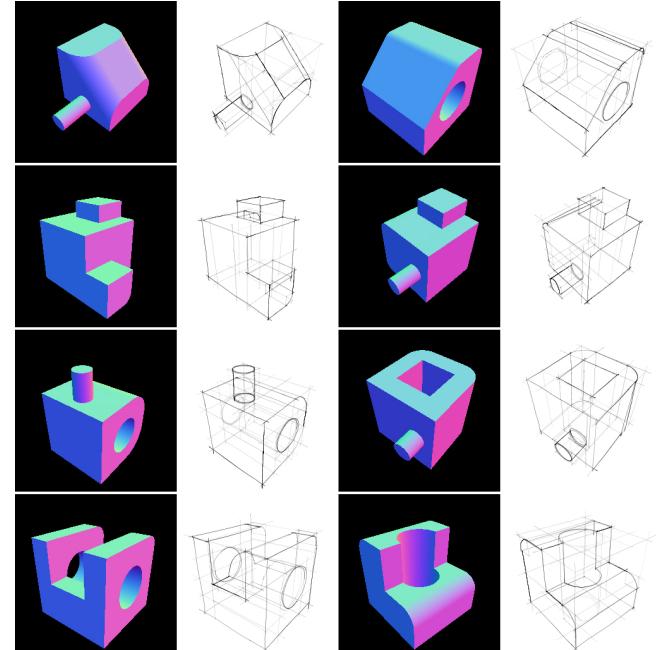


Fig. 20. **Synthetic data generation.** Example synthetic shapes (normal maps shown here) generated with our shape grammar along with the resulting synthetic sketches produced by CAD2Sketch.

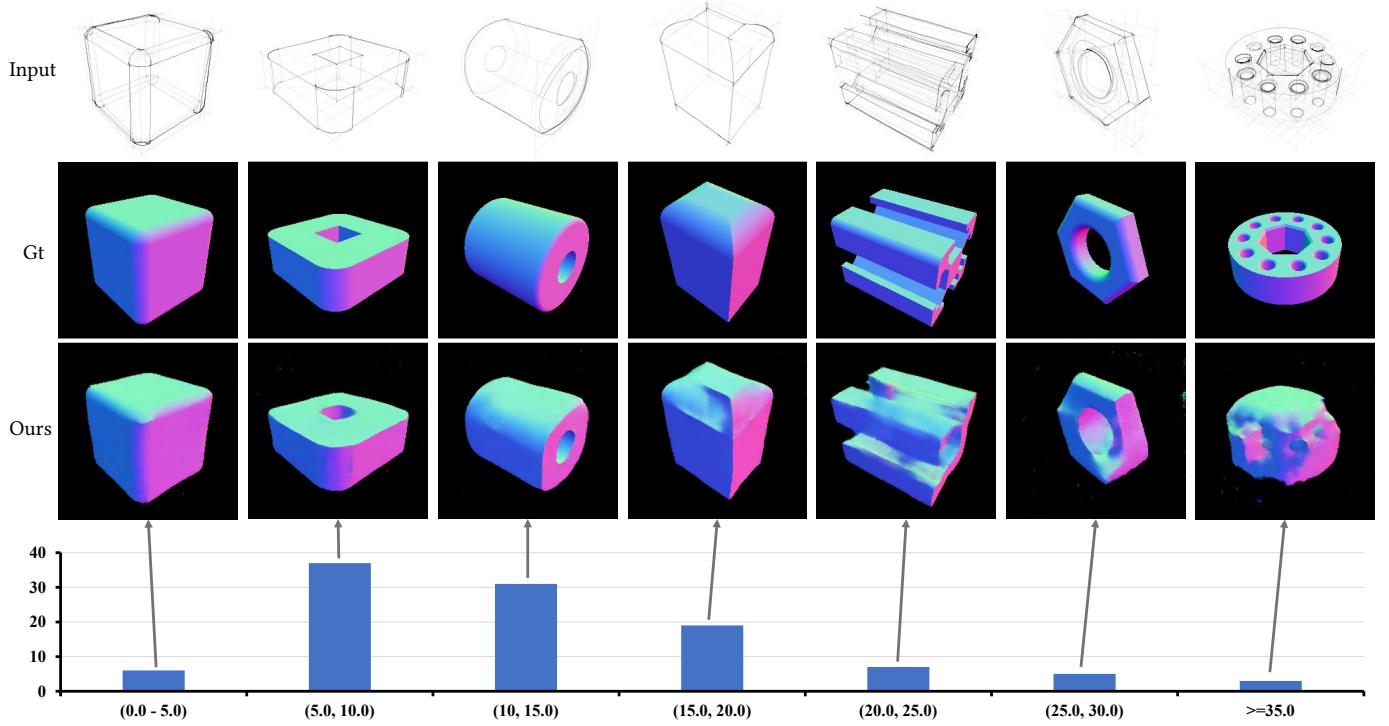


Fig. 21. **Error distribution on the ABC dataset.** In the context of the normal prediction task, we tested a prediction network trained on concept sketches generated using CAD sequences from a shape grammar on sketches produced using the ABC dataset [Koch et al. 2019]. We picked 200 random models from the ABC dataset, limited to those involving CAD commands supported in our implementation. Here we present the error histogram, x-axis shows error range in degrees, along with representative examples from each error group. Ground truth normal maps are provided for reference.

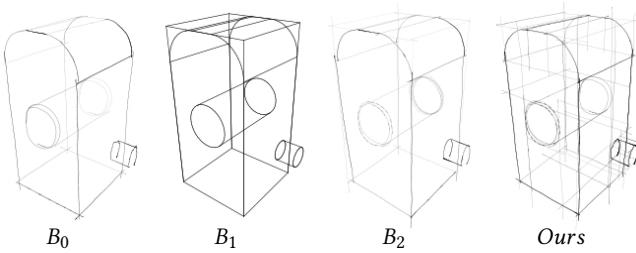


Fig. 22. **Ablation setting.** Example sketches produced by different training settings used in our ablation study. Refer to the text for details.

- $B_2$ . All intermediate feature lines (i.e., scaffolds), no auxiliary lines, rendered with stylized strokes.
- $Ours$ . Intermediate feature lines and auxiliary construction lines selected by our optimization and rendered with stylized strokes. We created three versions of this dataset of increasing size to evaluate the benefit of synthesizing more sketches.

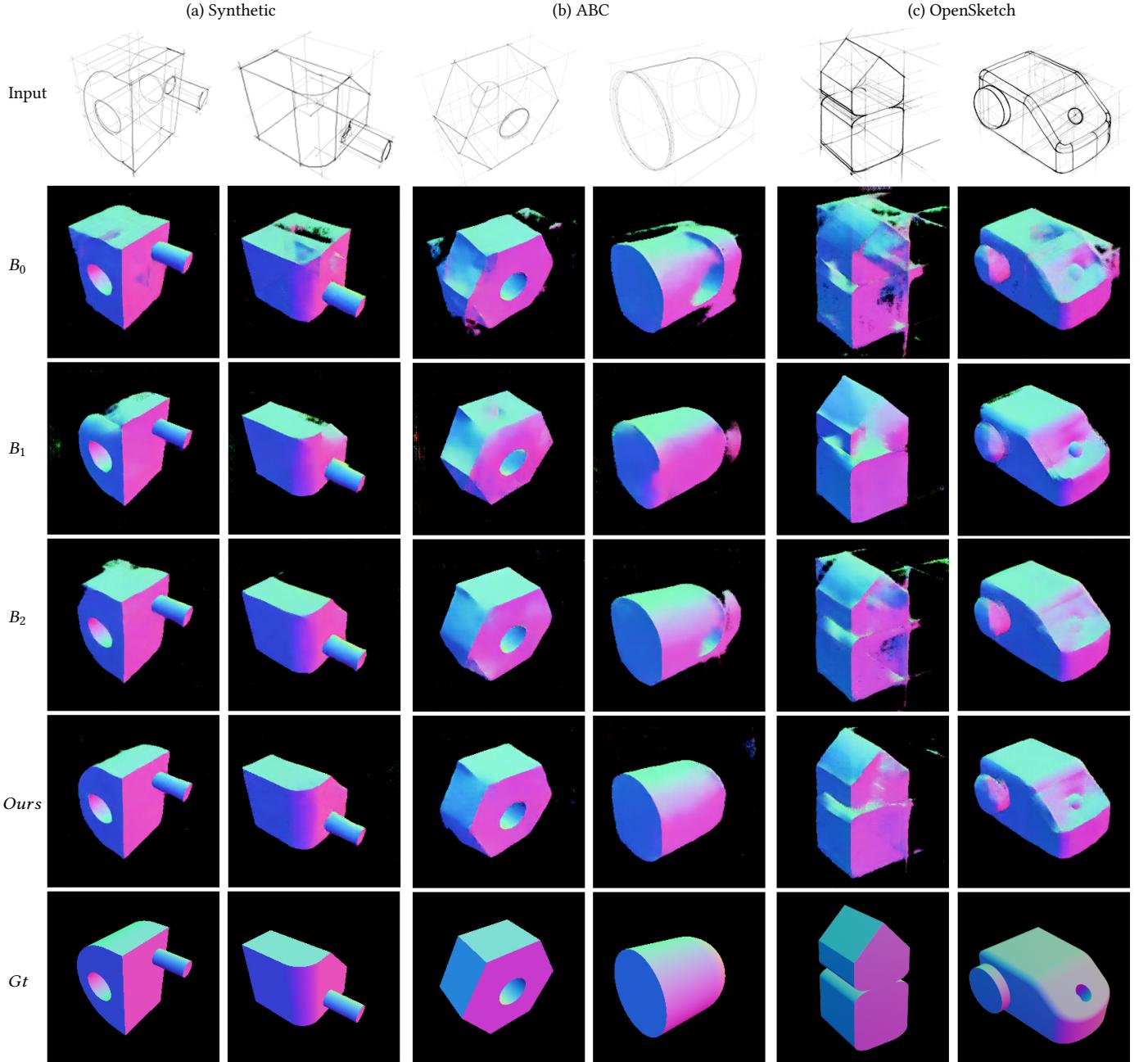
We used the style statistics from designer  $D3$  to render strokes for  $B_0$ ,  $B_2$  and  $Ours$ , as well as to render the synthetic and ABC test sets using our full method. Figure 22 shows the visual difference between all four training datasets.

**Metrics.** We measure the quality of normal prediction according to two metrics. First, we test whether the shape predicted by the

Table 2. **Normal map prediction accuracy.** Performance of the normal prediction neural network when trained with different datasets and tested on synthetic shapes and models from ABC [Koch et al. 2019]. As expected, performance increases with access to larger dataset. More interesting is that a network trained using synthetic sketches, obtained using our shape grammar and CAD2Sketch, continues to produce good results on sketches obtained using CAD sequences from the ABC dataset.

	Metric	$B_0$	$B_1$	$B_2$	Ours-15k	Ours-30k	Ours-60k
Syn.	IoU $\uparrow$	0.9	0.95	0.93	0.94	0.95	<b>0.96</b>
	Angerr $\downarrow$	9.67	11.42	8.79	12.75	9.45	<b>8.31</b>
ABC	IoU $\uparrow$	0.87	0.88	0.9	0.86	0.9	<b>0.91</b>
	Angerr $\downarrow$	14.63	18.82	14.01	18.4	15.71	<b>13.24</b>

neural network overlaps well with the ground truth, as measured by the Intersection over Union (IoU) over the binary masks obtained by thresholding background pixels. This metric penalizes both holes inside the shape and spurious surfaces hallucinated over the background. Second, we measure the angular error between the predicted normals and the ground truth. Since stroke stylization can introduce a slight misalignment between the sketch and the ground truth normal map, we search for each pixel of the ground truth the best predicted normal in a  $5 \times 5$  window.

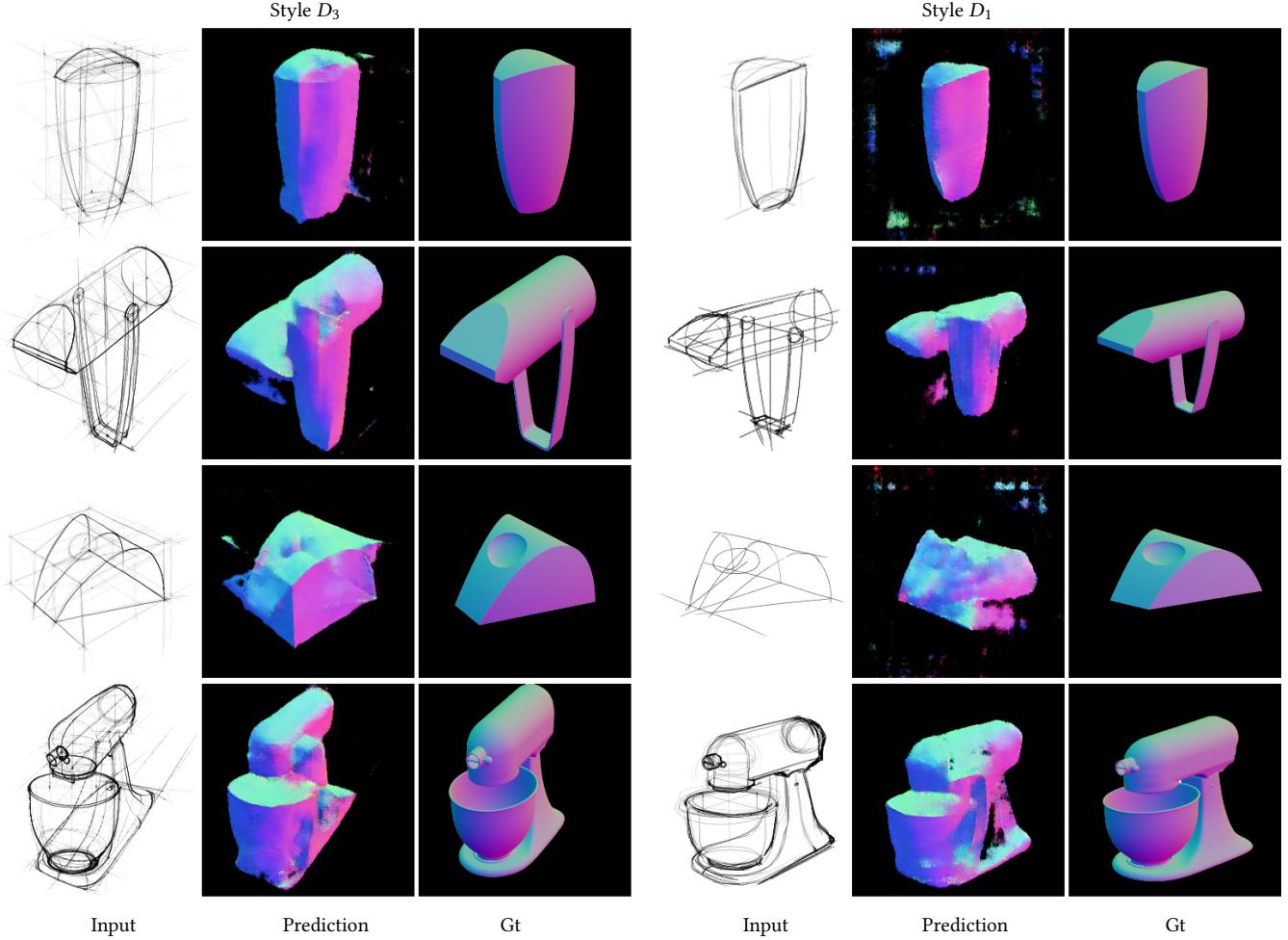


**Fig. 23. Normal prediction from sketches.** Qualitative results of the normal prediction network trained with different datasets, always produced from our synthetic grammar only, and tested on synthetic sketches generated from our shape grammar and from ABC shapes, and on real-world sketches. The network trained with our sketches produces normal maps closer to ground truth, while networks trained without auxiliary construction lines or stylized strokes misinterpret construction lines as feature lines of the surface. Refer to the text for details on  $B_0$ ,  $B_1$ , and  $B_2$ .

**Results.** Table 2 summarizes the performance of the different networks according to the two metrics, when tested on our synthetic dataset and ABC dataset. The network trained with sketches generated by our method outperforms all baselines on both metrics,

even though performance degrades when tested on the ABC models, which are more diverse than the shapes in our training set.

Figure 21 visualizes this trend by plotting the number of ABC models for different error ranges, along with representative models in each category. The neural network performs best on simple



**Fig. 24. Handling real world sketches.** Qualitative results of the normal prediction network for two designers from OpenSketch. In each case, we trained the network using synthetic sketches proper to the designer. The network succeeds in recovering the overall surface orientations, even though it struggles with configurations that are rare in our synthetic sketches, such as the isolated horizontal line at the front of the shape in the 3<sup>rd</sup> sketch of  $D_1$ , and the cubic corner of the scaffold that is accidentally aligned with the dark feature curve of the shape in the 3<sup>rd</sup> sketch of  $D_3$ . In both cases, the network interprets these construction lines as feature lines of the final shape and creates spurious surfaces.

models similar to our training set, capturing well the overall face orientations, rounded edges and corners, and holes. However, error increases for complex models, which are rare. We provide visual results on all 108 models of this test set as supplemental materials.

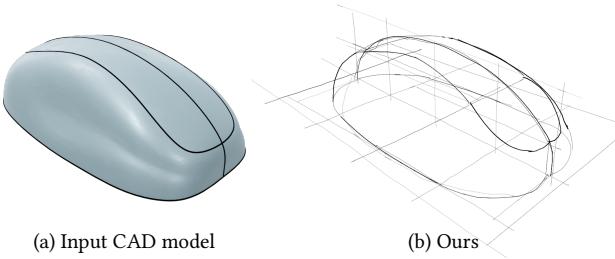
Figure 23 provides a visual comparison between the predictions of the different networks when tested on synthetic sketches produced by our method as well as on real sketches by designer  $D_3$ . The network trained with our full method predicts normals closer to the ground truth, with fewer spurious surfaces around the shape compared to other networks that tend to confuse construction lines with feature lines of the shape. The improvement is especially noticeable when comparing our results to  $B_0$ , which only contains feature lines as produced by existing NPR systems.

Finally, Figure 24 shows results obtained by training two neural networks, one with sketches rendered in the style of designer  $D_1$  and one with sketches rendered in the style of designer  $D_3$ , and tested

on real sketches of these two designers. The network specialized for  $D_3$  performs better on the respective sketches, which exhibit higher contrast between faint construction lines and dark feature lines.

## 6 CONCLUSION, LIMITATIONS AND FUTURE WORK

Despite the prevalence of freehand concept sketches in industrial design, no generative model exists to synthesize human-like concept sketches from 3D shapes. After studying how designers draw in perspective, we identified two major sources of the domain gap between synthetic drawings and real-world concept sketches. First, in addition to drawing the feature curves of the final shape, designers sketch a variety of auxiliary lines that they carefully order to achieve proper perspective. Second, designers strategically introduce and stylize these lines to construct the sketch without obscuring the final depiction of the shape. Building on these observations, and leveraging the design intent encoded in history-based CAD sequences,



**Fig. 25. Non-supported operations.** The surface of this computer mouse has been constructed using sweep and loft operations (a). Since our system does not add any construction lines for these operators, the result is solely made out of profile lines, grid lines and feature lines (b). Professional designers typically add curvature-aligned flow-lines to depict such smooth surfaces [Gori et al. 2017].

our method combines discrete optimization and data-driven stylization to generate legible concept sketches. We have shown through qualitative and quantitative evaluation that our synthetic sketches are useful both for convincing non-photorealistic rendering and for challenging downstream sketch interpretation tasks. We see this contribution as an important step towards the generation of large datasets for sketch-based design applications, just as photorealistic rendering has been key to the generation of large datasets for computer vision applications [Greff et al. 2022; Wood et al. 2021].

We envision several directions to extend and leverage our method:

**CAD and sketching operations.** While our system supports common CAD operations and the corresponding construction lines, it could be extended to cover more diverse shapes. In particular, free-form shapes are often created by lofting CAD profiles, which designers typically sketch using planar cross-sections and other *flow-lines* [Gori et al. 2017]. Currently, we only render grid and feature lines for such operations, see Fig. 25. CAD profiles also contain constraints we do not support yet, such as concentricity. Nevertheless, these constraints are typically sketched using the same construction techniques as the ones we implemented for alignments and proportions. Other construction techniques we do not support yet include elliptic arcs to depict equal length on hinge mechanisms, as present in the real sketches of the box (Figure 19). Finally, we assume that all necessary constraints are specified in the input CAD model. Progress in automatic detection of constraints in CAD profiles would benefit our approach [Para et al. 2021; Seff et al. 2022].

**CAD and sketching datasets.** While our approach benefits from the availability of CAD datasets, current datasets come with limitations. Fusion360 Gallery [Willis et al. 2021] focuses solely on sequences composed of 2D profiles and extrusion operations, while ABC [Koch et al. 2019] does not provide ready access to the CAD sequence for each model. While we could access ABC sequences via OnShape’s API [PTC 2019], the induced computational load limited us to a few hundred sequences out of the 1 million CAD models present in the original dataset. Similarly, the limited size of OpenSketch [Gryaditskaya et al. 2019] prevented us from conducting a large-scale evaluation on real sketches of diverse shapes.

**Modeling sketching errors.** In our three-stage pipeline, we only mimic the limited motor skills of humans by perturbing stroke trajectory during the final stylization stage. In reality, an error made on a pen stroke propagates to subsequent strokes anchored on it. For example, a distorted scaffold typically results in distorted feature curves. However, modeling such error propagation raises numerous challenges, including developing a motor model of how designers trace strokes [Cao and Zhai 2007], and keeping track of tracing errors for different selections of lines during optimization.

**Reproducing over-sketching.** Our stylization procedure does not reproduce over-sketching, where a single line is drawn with intermittent or overlapping strokes. To mimic this effect, one major challenge is to detect and quantify over-sketching in real-world sketches. One approach would consist in aggregating nearby strokes that are likely to represent the same line [Liu et al. 2018], although existing aggregation algorithms tend to merge construction and feature lines in concept sketches [Gryaditskaya et al. 2020].

**Analysis by synthesis.** An exciting direction for future work would be to solve for the  $\lambda_1$  and  $\lambda_2$  parameters that yield the best reproduction of a target sketch given a CAD sequence of the shape depicted in the sketch. Our synthesis method could then be used end-to-end with sketch analysis methods, for instance to predict a CAD sequence from a concept sketch [Li et al. 2022].

**Applications to sketch-based design.** Our synthetic sketches could serve many applications beyond normal estimation. By following the same construction principles as real designers, our method also synthesizes an *ordering* of the strokes, opening the door for training sequential models that exploit this temporal information. In addition to sketch-based modeling, potential applications include suggestive auto-completion from partial sketches [Lee et al. 2011].

## ACKNOWLEDGMENTS

The authors would like to thank the reviewers for their valuable suggestions, Yulia Gryaditskaya for early discussions on synthesizing concept sketches, Mark Sypesteyn and Jan Willem Hoftijzer for help on conducting the evaluation with design teachers. The authors are also grateful to the OPAL infrastructure from Université Côte d’Azur for providing resources and support. This work was supported by ERC Starting Grant D3 (ERC-2016-STG 714221), an Adobe Research internship, Marie Skłodowska-Curie grant 956585, and software and research donations from Adobe.

## REFERENCES

- Maneesh Agrawala, Wilmot Li, and Floraine Berthouzoz. 2011. Design Principles for Visual Communication. *Commun. ACM* 54, 4 (2011).
- Pierre Benard and Aaron Hertzmann. 2019. Line drawings from 3D models: a tutorial. *Foundations and Trends in Computer Graphics and Vision* 11, 1–2 (2019).
- Itamar Berger, Ariel Shamir, Moshe Mähler, Elizabeth Carter, and Jessica Hodgins. 2013. Style and abstraction in portrait sketching. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.
- Alexandra Bonnici, Alican Akman, Gabriel Calleja, Kenneth P Camilleri, Patrick Fehling, Alfredo Ferreira, Florian Hermuth, Johann Habakuk Israel, Tom Landwehr, Juncheng Liu, et al. 2019. Sketch-based interaction and modeling: where do we stand? *Artificial intelligence for engineering design analysis and manufacturing* (2019), 1–19.
- Xiang Cao and Shumin Zhai. 2007. Modeling Human Performance of Pen Stroke Gestures (*ACM SIGCHI Conference on Human Factors in Computing Systems (CHI)*).

- Caroline Chan, Frédéric Durand, and Phillip Isola. 2022. Learning to generate line drawings that convey geometry and semantics. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, and Anthony Santella. 2003. Suggestive Contours for Conveying Shape. *ACM Transactions on Graphics (Proc. SIGGRAPH) 22*, 3 (July 2003), 848–855.
- Johanna Delanoy, Mathieu Aubry, Phillip Isola, Alexei Efros, and Adrien Bousseau. 2018. 3D Sketching using Multi-View Deep Volumetric Prediction. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 21 (2018).
- Betty Edwards. 1979. *Drawing on the right side of the brain : a course in enhancing creativity and artistic confidence*.
- Koos Eissen and Roselien Steur. 2008. *Sketching: Drawing Techniques for Product Designers*. Bis Publishers.
- Koos Eissen and Roselien Steur. 2011. *Sketching: The Basics*. Bis Publishers.
- Giorgio Gori, Alla Sheffer, Nicholas Vining, Enrique Rosales, Nathan Carr, and Tao Ju. 2017. FlowRep: Descriptive Curve Networks for Free-Form Design Shapes. *ACM Transaction on Graphics (proc. SIGGRAPH) 36*, 4 (2017).
- Stéphane Grabli, Emmanuel Turquin, Frédéric Durand, and François X. Sillion. 2004. Programmable Style for NPR Line Drawing. In *Proc. Eurographics Conference on Rendering Techniques (EGSR)*.
- Stéphane Grabli, Emmanuel Turquin, Frédéric Durand, and François X. Sillion. 2010. Programmable rendering of line drawing from 3D scenes. *ACM Transactions on Graphics 29*, 2 (2010).
- Klaus Greff, Francois Belletti, Lucas Beyer, Carl Doersch, Yilun Du, Daniel Duckworth, David J Fleet, Dan Gnanapragasam, Florian Golemo, Charles Herrmann, Thomas Kipf, Abhijit Kundu, Dmitry Lagun, Issam Laradji, Hsueh-Ti (Derek) Liu, Henning Meyer, Yishu Miao, Derek Nowrouzezahrai, Cengiz Oztireli, Etienne Pot, Noha Radwan, Daniel Rebain, Sara Sabour, Mehdi S. M. Sajjadi, Matan Sela, Vincent Sitzmann, Austin Stone, Deqing Sun, Suhani Vora, Ziyu Wang, Tianhao Wu, Kwang Moo Yi, Fangcheng Zhong, and Andrea Tagliasacchi. 2022. Kubric: a scalable dataset generator. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Igor Griva, Stephen G. Nash, and Ariela Sofer. 2008. *Linear and Nonlinear Optimization (2. ed.)*. SIAM.
- Yulia Gryaditskaya, Felix Hähnlein, Chenxi Liu, Alla Sheffer, and Adrien Bousseau. 2020. Lifting Freehand Concept Sketches into 3D. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) (2020)*.
- Yulia Gryaditskaya, Mark Sypesteyn, Jan Willem Hoftijzer, Sylvia Pont, Frédéric Durand, and Adrien Bousseau. 2019. OpenSketch: A Richly-Annotated Dataset of Product Design Sketches. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) (2019)*.
- Benoit Guillard, Edoardo Remelli, Pierre Yvernat, and Pascal Fua. 2021. Sketch2Mesh: Reconstructing and Editing 3D Shapes From Sketches. In *Proc. IEEE International Conference on Computer Vision (ICCV)*.
- Gurobi Optimization, LLC. 2021. Gurobi Optimizer Reference Manual. <https://www.gurobi.com>
- Felix Hähnlein, Yulia Gryaditskaya, Alla Sheffer, and Adrien Bousseau. 2022. Symmetry-driven 3D Reconstruction from Concept Sketches. In *SIGGRAPH Conference Proceedings*, 1–8.
- James W Hennessey, Han Liu, Holger Winnemöller, Mira Dontcheva, and Niloy J Mitra. 2016. How2Sketch: generating easy-to-follow tutorials for sketching 3D objects. *arXiv preprint arXiv:1607.07980* (2016).
- Kevin Henry. 2012. *Drawing for Product Designers*. Laurence King Publishing.
- Aaron Hertzmann and Denis Zorin. 2000. Illustrating Smooth Surfaces. In *SIGGRAPH*.
- George Hlavács. 2014. *The Exceptionally Simple Theory of Sketching*. Laurence King Publishing.
- Haibin Huang, Evangelos Kalogerakis, Ersin Yumer, and Radomir Mech. 2016. Shape Synthesis from Sketches via Procedural Models and Convolutional Networks. *IEEE Transactions on Visualization and Computer Graphics (TVCG) 22*, 10 (2016), 1.
- Tilke Judd, Frédéric Durand, and Edward H. Adelson. 2007. Apparent ridges for line drawing. *ACM Transactions on Graphics 26*, 3 (2007).
- Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. 2019. Abc: A big cad model dataset for geometric deep learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9601–9611.
- Yong Jae Lee, C. Lawrence Zitnick, and Michael F. Cohen. 2011. ShadowDraw: Real-Time User Guidance for Freehand Drawing. *ACM Transaction on Graphics (proc. SIGGRAPH) (2011)*.
- Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J Mitra. 2020. Sketch2cad: Sequential cad modeling by sketching in context. *ACM Transactions on Graphics (TOG) 39*, 6 (2020), 1–14.
- Changjian Li, Hao Pan, Adrien Bousseau, and Niloy J Mitra. 2022. Free2CAD: Parsing Freehand Drawings into CAD Commands. *ACM Transaction on Graphics (proc. SIGGRAPH) (2022)*.
- Changjian Li, Hao Pan, Yang Liu, Xin Tong, Alla Sheffer, and Wenping Wang. 2018. Robust flow-guided neural prediction for sketch-based freeform surface modeling. In *ACM Trans. Graph. (Proc. SIGGRAPH Asia)*. ACM, 238.
- Wilmot Li, Lincoln Ritter, Maneesh Agrawala, Brian Curless, and David Salesin. 2007. Interactive Cutaway Illustrations of Complex 3D Models. *ACM Transactions on Graphics (proc. SIGGRAPH) 26*, 3 (July 2007).
- Chenxi Liu, Enrique Rosales, and Alla Sheffer. 2018. StrokeAggregator: consolidating raw sketches into artist-intended curve drawings. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 97.
- Difan Liu, Matthew Fisher, Aaron Hertzmann, and Evangelos Kalogerakis. 2021. Neural Strokes: Stylized Line Drawing of 3D Shapes. In *Proc. IEEE International Conference on Computer Vision (ICCV)*. 14204–14213.
- Difan Liu, Mohamed Nabail, Aaron Hertzmann, and Evangelos Kalogerakis. 2020. Neural contours: Learning to draw lines from 3d shapes. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 5428–5436.
- Jingbo Liu, Hongbo Fu, and Chiew-Lan Tai. 2014. Dynamic Sketching: Simulating the Process of Observational Drawing. In *Proceedings of the Workshop on Computational Aesthetics*.
- Zhaoliang Lun, Matheus Gadelha, Evangelos Kalogerakis, Subhransu Maji, and Rui Wang. 2017. 3D shape reconstruction from sketches via multi-view convolutional networks. In *IEEE International Conference on 3D Vision (3DV)*. 67–77.
- Thomas L Magnanti and Prakash Mirchandani. 1993. Shortest paths, single origin-destination network design, and associated polyhedra. *Networks 23*, 2 (1993), 103–121.
- Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. 2010. Illustrating How Mechanical Assemblies Work. *ACM Transactions on Graphics (Proc. SIGGRAPH) 29*, 3 (2010).
- Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. 2016. Interactive Sketching of Urban Procedural Models. *ACM Trans. Graph. (SIGGRAPH) 35*, 4, Article 130 (July 2016), 11 pages.
- Yutaka Ohtake, Alexander Belyaev, and Hans-Peter Seidel. 2004. Ridge-Valley Lines on Meshes via Implicit Surface Fitting. *ACM Transactions on Graphics (Proc. SIGGRAPH) 23*, 3 (2004).
- Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas J Guibas, and Peter Wonka. 2021. SketchGen: Generating Constrained CAD Sketches. In *Advances in Neural Information Processing Systems (NeurIPS)*, Vol. 34.
- PTC. 2019. *OnShape*. [www.onshape.com](http://www.onshape.com)
- Scott Robertson and Thomas Bertling. 2013. *How to draw*. Design Studio Press.
- Ryan Schmidt, Tobias Isenberg, Pauline Jepp, Karan Singh, and Brian Wyvill. 2007. Sketching, Scaffolding, and Inking: A Visual History for Interactive 3D Modeling. In *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*.
- Ryan Schmidt, Azam Khan, Gord Kurtenbach, and Karan Singh. 2009a. On Expert Performance in 3D Curve-drawing Tasks. In *Sketch-Based Interfaces and Modeling (SBIM) 8*.
- Ryan Schmidt, Azam Khan, Karan Singh, and Gord Kurtenbach. 2009b. Analytic drawing of 3D scaffolds. In *ACM transactions on graphics (TOG)*, Vol. 28. ACM, 149.
- Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. 2022. Vitruvion: A Generative Model of Parametric CAD Sketches. In *International Conference on Learning Representations (ICLR)*.
- Wanchao Su, Dong Du, Xin Yang, Shizhe Zhou, and Hongbo Fu. 2018. Interactive sketch-based normal map generation with deep neural networks. *Proceedings of the ACM on Computer Graphics and Interactive Techniques 1*, 1 (2018), 22.
- Zeyu Wang, Sherry Qiu, Nicole Feng, Holly Rushmeier, Leonard McMillan, and Julie Dorsey. 2021. Tracing versus Freehand for Evaluating Computer-Generated Drawings. *ACM Transactions on Graphics (Proc. SIGGRAPH) 40*, 4 (2021).
- Karl D. D. Willis, Yewen Pu, Jieliang Luo, Hang Chu, Tao Du, Joseph G. Lambourne, Armando Solar-Lezama, and Wojciech Matusik. 2021. Fusion 360 Gallery: A Dataset and Environment for Programmatic CAD Construction from Human Design Sequences. *ACM Transactions on Graphics (TOG) 40*, 4 (2021).
- Georges Winkenbach and David H. Salesin. 1994. Computer-Generated Pen-and-Ink Illustration. In *SIGGRAPH*.
- Erroll Wood, Tadas Baltrušaitis, Charlie Hewitt, Sebastian Dziadzio, Thomas J. Cashman, and Jamie Shotton. 2021. Fake It Till You Make It: Face Analysis in the Wild Using Synthetic Data Alone. In *IEEE International Conference on Computer Vision (ICCV)*.
- Rundi Wu, Chang Xiao, and Changxi Zheng. 2021. DeepCAD: A Deep Generative Network for Computer-Aided Design Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 6772–6782.
- Song-Hai Zhang, Yuan-Chen Guo, and Qing-Wen Gu. 2021. Sketch2Model: View-Aware 3D Modeling From Single Free-Hand Sketches. In *Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Yue Zhong, Yulia Gryaditskaya, Honggang Zhang, and Yi-Zhe Song. 2020a. Deep Sketch-Based Modeling: Tips and Tricks. In *Proceedings of International Conference on 3D Vision (3DV)*.
- Yue Zhong, Yonggang Qi, Yulia Gryaditskaya, Honggang Zhang, and Yi-Zhe Song. 2020b. Towards practical sketch-based 3d shape generation: The role of professional sketches. *IEEE Transactions on Circuits and Systems for Video Technology 31*, 9 (2020), 3518–3528.