`PICASSO`: A Feed-Forward Framework for Parametric Inference of CAD Sketches via Rendering Self-Supervision

Ahmet Serdar Karadeniz[1] Dimitrios Mallis[1] Nesryne Mejri[1] Kseniya Cherenkova[1,2]
Anis Kacem[1] Djamila Aouada[1]
[1] SnT, University of Luxembourg [2] Artec 3D

Abstract

*This work introduces `PICASSO`, a framework for the parameterization of 2D CAD sketches from hand-drawn and precise sketch images. `PICASSO` converts a given CAD sketch image into parametric primitives that can be seamlessly integrated into CAD software. Our framework leverages rendering self-supervision to enable the pre-training of a CAD sketch parameterization network using sketch renderings only, thereby eliminating the need for corresponding CAD parameterization. Thus, we significantly reduce reliance on parameter-level annotations, which are often unavailable, particularly for hand-drawn sketches. The two primary components of `PICASSO` are **(1)** a Sketch Parameterization Network (SPN) that predicts a series of parametric primitives from CAD sketch images, and **(2)** a Sketch Rendering Network (SRN) that renders parametric CAD sketches in a differentiable manner and facilitates the computation of a rendering (image-level) loss for self-supervision. We demonstrate that the proposed `PICASSO` can achieve reasonable performance even when finetuned with only a small number of parametric CAD sketches. Extensive evaluation on the widely used SketchGraphs [37] and CAD as Language [14] datasets validates the effectiveness of the proposed approach on zero- and few-shot learning scenarios.*

1. Introduction

Computer-Aided Design (CAD) has become the industry norm for mechanical design of any product prior to manufacturing. CAD software [2,3] enhances the productivity of engineers and enables efficient extension or alteration of existing designs. The modern CAD workflow widely adopts the paradigm of *feature-based* modeling [44], where initially a series of two-dimensional parametric CAD sketches is specified, followed by CAD operations (*e.g*. extrusion, revolution, etc.) to form a 3D solid. CAD sketches comprise a collection of geometric primitives (*e.g*. lines, arcs) as well as constraints enforced between those primitives (*e.g*. coincident, parallel, etc.). Feature-based CAD constitutes an efficient way of constructing complex 3D models [11,12,20,26]. Commonly, the modelling process starts with conceptualization of a design by roughly drawing it by hand [38]. Designers are tasked with meticulously translating these drawings, often in the form of raster images, into parametric CAD sketches. Depending on the design complexity, the task can be time-consuming even for highly skilled designers. Thus, the automation of the CAD parameterization process (Figure 1-*top*) has gained attention in the research community [38,46] and CAD industry [1,3].

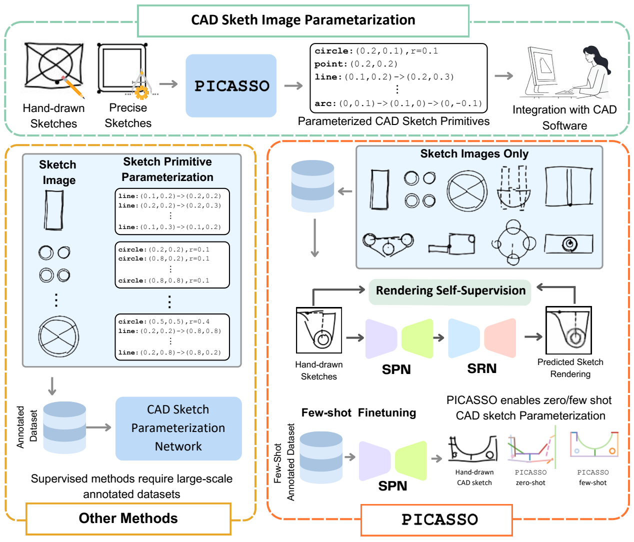
CAD Sketh Image Parametarization
Hand-drawn Sketches
Precise Sketches
PICASSO
circle: (0.2,0.1),r=0.1
point: (0.2,0.2)
line: (0.1,0.2)->(0.2,0.3)
arc: (0,0.1)->(0.1,0)->(0,-0.1)
Parameterized CAD Sketch Primitives
Integration with CAD Software
Sketch Image
Sketch Primitive Parameterization
Annotated Dataset
CAD Sketch Parameterization Network
Supervised methods require large-scale annotated datasets
Other Methods
Sketch Images Only
Rendering Self-Supervision
Hand-drawn Sketches
SPN
SRN
Predicted Sketch Rendering
Few-shot Finetuning
PICASSO enables zero/few shot CAD sketch Parameterization
Few-Shot Annotated Dataset
SPN
Hand-drawn CAD sketch
PICASSO zero-shot
PICASSO few-shot
PICASSO

Figure 1. `PICASSO` is a novel self-supervised framework for CAD sketch parameterization. Unlike methods relying on parametric annotations, `PICASSO` is pretrained via rendering self-supervision on CAD sketch images only, thus drastically reducing the need for parametrically annotated sketches. We demonstrate `PICASSO`'s effectiveness in both few-shot and zero-shot settings.

Parameterization of CAD sketches from raster images constitutes a complex problem due to the large solution space required to model parametric entities, the nuanced intricacies of sketch designs, and the inherently inaccurate

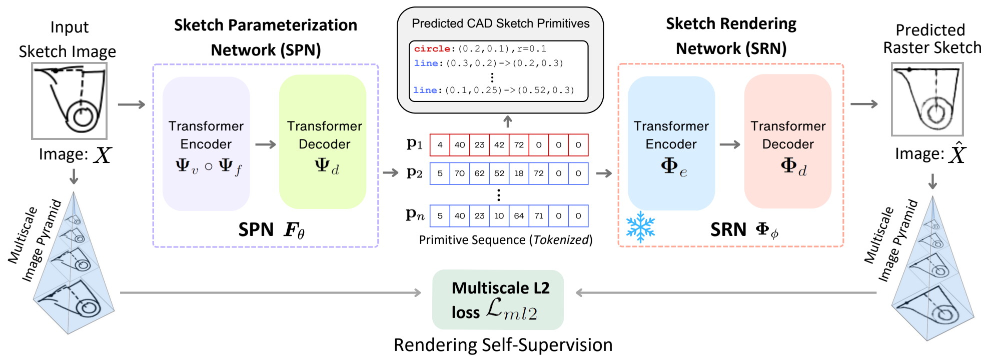
Input Sketch Image
Image: X
Multiscale Image Pyramid
Sketch Parameterization Network (SPN)
Transformer Encoder $\Psi_v \circ \Psi_f$
Transformer Decoder Ψ_d
SPN F_θ
Predicted CAD Sketch Primitives
circle: (0.2,0.1), r=0.1
line: (0.3,0.2)->(0.2,0.3)
line: (0.1,0.25)->(0.52,0.3)
$\mathbf{p}_1$
$\mathbf{p}_2$
$\mathbf{p}_n$
Primitive Sequence (Tokenized)
Sketch Rendering Network (SRN)
Transformer Encoder Φ_e
Transformer Decoder Φ_d
SRN Φ_ϕ
Predicted Raster Sketch
Image: $\hat{X}$
Multiscale L2 loss $\mathcal{L}_{ml2}$
Rendering Self-Supervision

Figure 2. Framework overview. PICASSO is composed of two networks, namely the *Sketch Parametrization Network* (SPN) and *Sketch Rendering Network* (SRN). Once trained, SRN is kept frozen and used for rendering self-supervision using a multiscale $l2$ loss. This allows for image-level pre-training of the CAD sketch parameterization network SPN.

nature of hand-drawings. Compared to vectorized stroke inputs [4], raster images pose additional challenges due to indistinguishable, overlapping or closely positioned primitives on the image domain. The availability of large-scale CAD sketch datasets [14, 37] has enabled tackling this problem with learning-based approaches. Recent works [14, 29, 38] propose *autoregressive* generative models to learn CAD sketch parameterization through general-purpose language modelling. Similarly to next word prediction for natural language processing (NLP), these models predict the next primitive in the sketch and can be conditioned on images for sketch parameterization. Transformer-based networks are typically trained to predict tokens representing the type and parameters of each primitive via parameter-level supervision. These networks are adept at exploiting large-scale annotated datasets with millions of sketches (*e.g.* SketchGraphs [37]) (Figure 1-*bottom left*). Nevertheless, there are real-world situations where this quantity of labeled sketches is unavailable. Examples include rough hand-drawn CAD sketches and 2D cross-sections [6] from 3D scans, where collecting parameter annotations is challenging. These examples exhibit a distribution shift w.r.t. to CAD sketches in large-scale datasets, underscoring the necessity of learning CAD sketch parametrization from raster images when parameter-level annotations are scarce or nonexistent.

In this work, we introduce a framework for **P**arametric **I**nference of **CA**D **S**ketches via Rendering **S**elf-SupervisiO**n**, refereed to as PICASSO. PICASSO enables learning parametric CAD sketches directly from precise or hand-drawn images, even when parameter-level annotations are limited or unavailable (Figure 1-*bottom right*). This is achieved by utilizing the geometric appearance of sketches as a learning signal to pretrain a CAD parameterization network. Specifically, PICASSO is composed of two main components: **(1)** a *Sketch Parameterization Network* (SPN) that predicts a set of parametric primitives from CAD sketch images and **(2)** a *Sketch Rendering Network* (SRN) to render parametric CAD sketches in a differentiable manner. SRN enables image-to-image loss computation that can be used to pretrain SPN, leading to *zero-* and *few-shot* learning scenarios for hand-drawn sketch parametrization (see Figure 1). To the best of our knowledge, we are the first to address CAD sketch parameterization with limited or without parametric annotations. PICASSO can achieve strong parameterization performance with only a small number of annotated samples. Moreover, departing from recent work based on autoregressive modelling [14, 29, 38], the proposed SPN is a *feed-forward* network. The sketch is treated as an unordered set of primitives and a CAD sequence is predicted in a non-autoregressive manner. Experiments show that SPN outperforms recent autoregressive state-of-the-art on parameterizing precise and hand-drawn CAD sketch images. An overview of PICASSO is presented in Figure 2.

Contributions: The main contributions of this work can be summarized as follows:

1. PICASSO is a novel framework that enables image-level pretraining for CAD sketch parameterization, thus allowing, for the first time, few and zero-shot learning scenarios of CAD sketch parameterization from hand-drawn or precise CAD sketch raster images.
2. SRN, a neural differentiable renderer to rasterize CAD parametric primitives is proposed. By leveraging SRN, we are the first to explore image-level pretraining for CAD sketch parameterization.
3. The proposed feed-forward SPN results in state-of-the-art CAD sketch parameterization.
4. PICASSO is thoroughly evaluated both qualitatively and quantitatively on the widely-used *SketchGraphs* dataset [37] under few or zero-shot evaluation settings.

This paper is organized as follows; Section 2 reviews the related works. Section 3 formulates the problem of CAD sketch parameterization. The proposed PICASSO framework is described in Section 4. The experiments are provided in Section 5. Finally, Section 6 concludes this work.

2. Related Works

Most sketch-related literature focuses on understanding human-made sketches, typically represented as set of free-hand strokes. Relevant applications include sketch synthesis [5, 15], recognition [24, 47], segmentation [32, 45], grouping of individual strokes [43], sketch classification [43] and abstraction [4]. Free-hand sketches are distinct from CAD sketches for feature-based CAD modelling. The free-hand representation is non-parametric, limited in terms of editability and generally is the result of a spontaneous drawing process. The focus of this work is the separate paradigm of CAD sketch parameterization [29,37,38] from precise or hand-drawn CAD sketch images. The remainder of this section will introduce related methods for CAD parameterization or generation and discuss recent advancements in differentiable rendering of parametric entities.

CAD Sketch Parameterization: CAD sketches relate to the formal profiling of mechanical components and are represented by a set of parametric entities (*e.g.*, lines, circles, arcs, and splines) which are often constrained by defined relationships that maintain design intent. Typically the sketch is defined as a 2D drawing on a 3D plane, subsequently transformed into a 3D solid via CAD operations like extrusion, cutting, or revolution. Statistics from the Onshape CAD platform [2] report that sketches constitute approximately 35% of daily feature creation [8, 14]. Literature on parametric CAD sketches mostly focuses on the task of CAD sketch generation or synthesis. Recent methods [14, 29, 38] adopt a unified approach centered around autoregressive transformers. While the primary focus is on generation, transformer decoders can be further conditioned on sketch images for CAD parameterization. One of the first attempts to address CAD generation through generic language modelling was [14]. The authors employ the protocol buffer specification for 2D sketches and generate constrained CAD sketches using a transformer decoder. In [41], authors explore two distinct sketch representations based on either hypergraphs or turtle graphics (sequence of pen-up, pen-move actions) and introduce corresponding generative models, CurveGen and TurtleGen, respectively.

Concurrently, SketchGen [29] and Vitruvion [38] propose two-stage architectures for both primitive and constraint generation. Seff *et al.* [38] also investigate their model's efficacy for the parameterization of hand-drawn sketches. These methods follow the autoregressive learning strategy as it constitutes the natural choice for generative modelling. Autoregressive inference is also well-suited for the related application of sketch auto-completion but might be suboptimal for CAD parameterization from images (see detailed discussion in Section 4.2). In contrast, the proposed SPN is a feed-forward network. Related to ours is also the non-autoregressive method of [46]. Authors introduce the task of modular concept discovery that is addressed through a program library induction perspective. Recently, the authors in [40] presented a concurrent work to ours. They highlighted the issue of error accumulation in autoregressive models and introduced a feed-forward strategy for CAD sketch parameterization, which is similar to our Sketch Parameterization Network (SPN). Also concurrently to ours, the work of [19] proposes a single-stage architecture for joint sketch parameterization and constraint prediction. All aforementioned methods solely rely on parameter-level supervision and overlook the rendered geometry of a CAD sketch. In this work, rendering self-supervision is proposed to provide an alternative signal for sketch parameterization. Such signal allows for pre-training CAD sketch parameterization network, hence enabling it to parameterize hand-drawn or precise CAD sketches under limited or without parametric supervision.

Parametric Rendering: Rendering refers to the process of converting vector parameters into a raster image. Given that direct vector supervision is not always available, rendering modules can bridge the vector and raster domains and enable gradient-based optimization. Applications include visually supervised parameterization [9, 13, 23, 25, 27, 33], as well as painterly rendering or seam carving [23]. Generally, parametric rendering cannot be directly integrated within end-to-end training pipelines, since vectorized shapes (represented as indicator functions) are not differentiable. A line of work explores differentiable renderers [18, 23] that can automatically compute gradients with respect to input vector parameters. Even though these methods are generalizable, gradients are only given for continuous parameters and cannot affect discrete decisions such as adding, rearranging or removing primitives [23]. More relevant to us, a line of work investigates learnable rendering approaches [18,27,28,48] to convert parameters into raster images that allow optimization with image-base losses. Rendering modules proposed in these works operate on vector graphics, commonly in the form of Bezier paths. Compared to vector graphics, rendering used for self-supervision of parametric CAD sketches should allow the capturing of multiple types of parametric primitives (*e.g.*, lines, circles). To our knowledge, we are the first to explore neural rendering of parametric CAD sketches.

3. Problem Statement

Given a binary sketch image $\mathbf{X} \in \{0,1\}^{h\times w}$, where h and w denote the height and the width, respectively, our goal is to infer a set of n parametric primitives $\{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\} \in \mathcal{P}^n$ reconstructing the input image $\mathbf{X}$. Here, $\mathcal{P}$ denotes the space of possible primitives. Similarly to [38], each $\mathbf{p}_i \in \mathcal{P}$ can be one of the following types:

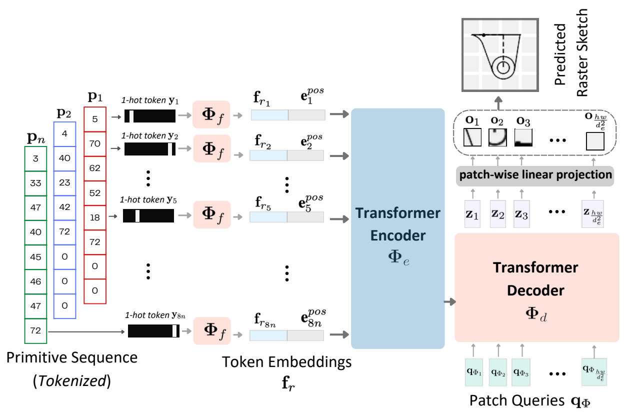
Predicted Raster Sketch
$\mathbf{o}_1$ $\mathbf{o}_2$ $\mathbf{o}_3$ ... $\mathbf{o}_{\frac{hw}{d_e^2}}$
patch-wise linear projection
$\mathbf{z}_1$ $\mathbf{z}_2$ $\mathbf{z}_3$ ... $\mathbf{z}_{\frac{hw}{d_e^2}}$
$\mathbf{p}_n$
$\mathbf{p}_2$
$\mathbf{p}_1$
1-hot token $\mathbf{y}_1$
1-hot token $\mathbf{y}_2$
1-hot token $\mathbf{y}_5$
1-hot token $\mathbf{y}_{8n}$
Φ_f
$\mathbf{f}_{r_1}$ $\mathbf{e}_1^{pos}$
$\mathbf{f}_{r_2}$ $\mathbf{e}_2^{pos}$
$\mathbf{f}_{r_5}$ $\mathbf{e}_5^{pos}$
$\mathbf{f}_{r_{8n}}$ $\mathbf{e}_{8n}^{pos}$
Transformer Encoder Φ_e
Transformer Decoder Φ_d
$\mathbf{q}_{\Phi_1}$ $\mathbf{q}_{\Phi_2}$ $\mathbf{q}_{\Phi_3}$ ... $\mathbf{q}_{\Phi_{\frac{hw}{d_e^2}}}$
Primitive Sequence (*Tokenized*)
Token Embeddings $\mathbf{f}_r$
Patch Queries $\mathbf{q}_\Phi$

Figure 3. Overview of the *Sketch Rendering Network* (SRN). SRN is modeled by a transformer encoder-decoder that learns a mapping from parametric primitive tokens to the sketch image domain. SRN enables neural differentiable rendering that we leverage for rendering self-supervision of SPN.

Line. A line $\mathbf{l}_i$ is defined by its start and end points $(x_s, y_s) \in \mathbb{R}^2, (x_e, y_e) \in \mathbb{R}^2$.
Circle. A circle $\mathbf{c}_i$ is represented by its center point $(x_c, y_c) \in \mathbb{R}^2$, and its radius $r \in \mathbb{R}$.
Arc. An arc $\mathbf{a}_i$ is defined by its start, middle, and end points $(x_s, y_s) \in \mathbb{R}^2, (x_m, y_m) \in \mathbb{R}^2, (x_e, y_e) \in \mathbb{R}^2$.
Point. A point $\mathbf{d}_i$, parameterized through its coordinates $(x_p, y_p) \in \mathbb{R}^2$.

Our goal becomes to learn the mapping $\boldsymbol{F}_\theta : \{0,1\}^{h \times w} \rightarrow \mathcal{P}^n$ with model parameters θ, from the image domain to the space of parametric primitives. As in [38], each primitive is represented by a set of *tokens* defining primitive types and quantized primitive parameters. A 6-bit uniform quantization is adopted. More specifically, every primitive is expressed as a set of 8 tokens $\mathbf{p}_n = \{t_i^j\}_{j \in [\![1..8]\!]}$ with $t_i^j \in [\![0..72]\!]$. Tokens in $[\![0..6]\!]$ represent primitive types *i.e.*, *padding*, *start*, *end*, *arc*, *circle*, *line*, and *point*, respectively. Tokens in the range $[\![7..70]\!]$ correspond to quantized primitive parameters and values $\{71, 72\}$ refer to whether the primitive is a construction primitive (used by designers for referencing). The complete set of quantized primitives can be formally defined as $\mathbf{y} \in \{0,1\}^{8n \times 73}$ where 8 represents the tokens for each primitive and 73 is the tokenization interval.

4. PICASSO: Proposed Framework

PICASSO comprises two transformer-based networks, namely the Sketch Rendering Network (SRN) and the Sketch Parameterization Network (SPN). We leverage the differentiable neural renderer SRN for rendering self-supervision, enabling large-scale pre-training of CAD parameterization models without requiring any parameter-level annotations. Details on both SRN and SPN along with the CAD sketch parameterization scenarios enabled by the proposed image-level pre-training are described next.

4.1. Sketch Rendering Network

This work proposes a Sketch Rendering Network (SRN) designed for neural rendering of parametric CAD sketches. SRN learns $\boldsymbol{\Phi}_\phi : \mathcal{P}^n \rightarrow \{0,1\}^{h \times w}$, *i.e.*, the mapping from a set of parametric primitive tokens to the sketch image domain. The primary utility of SRN is to allow the computation of an image-to-image loss between predicted primitives and their corresponding precise or hand-drawn input sketch image, thus enabling image-level pre-training for CAD sketch parametrization. Note that *explicit* rendering of CAD sketches implies the direct rasterization of parametric primitives, and it is inherently a non-differentiable process. While differentiable rendering solutions exist in literature [23], their use is limited to parameter refinement.

Network Architecture: The proposed architecture for SRN is depicted in Figure 3. SRN operates on one-hot input tokens $\mathbf{y} = [\mathbf{y}_1\ \mathbf{y}_2\ ...\ \mathbf{y}_{8n}] \in \{0,1\}^{8n \times 73}$. Firstly, input tokens are individually projected through a linear layer $\boldsymbol{\Phi}_f$ to produce token embeddings $\mathbf{f}_r = [\mathbf{f}_{r_1}, \mathbf{f}_{r_2}, ..., \mathbf{f}_{r_{8n}}]$, where $\mathbf{f}_{r_i} = \boldsymbol{\Phi}_f(\mathbf{y}_i) \in \mathbb{R}^{d_q}$ for the i-th token and d_q denotes the embedding dimension. The $\mathbf{f}_r$ embeddings are concatenated with fixed positional embeddings $\mathbf{e}_i^{pos}$ [30] and processed by a transformer encoder network $\boldsymbol{\Phi}_e$ without changing their dimensions. In the decoding phase, the decoder $\boldsymbol{\Phi}_d$, inspired by [16], employs self and cross-attention to map a set of randomly initialized *learnable* patch queries $\mathbf{q}_\Phi \in \mathbb{R}^{\frac{hw}{d_e^2} \times d_e \times d_e}$ to a set of decoded patch embeddings $\mathbf{z} = [\mathbf{z}_1\ \mathbf{z}_2\ ...\ \mathbf{z}_{\frac{hw}{d_e^2}}] \in \mathbb{R}^{\frac{hw}{d_e^2} \times d_e \times d_e}$. Patch queries $\mathbf{q}_\Phi$ serve as learnable tokens that query the encoded representations to extract spatial and structural information that is required for the rendering. Finally, each decoded patch embedding $\mathbf{z}_i$ pass through a patch-wise linear layer with a sigmoid activation function, to produce the corresponding output patch of pixel-intensities $\mathbf{o}_{\frac{hw}{d_e^2}} \in [0,1]^{d_e \times d_e}$. The final rendered image is reconstructed by assembling the output patches $\mathbf{o}_i$.

4.2. Sketch Parametrization Network

Our Sketch Parametrization Network (SPN) learns the inverse mapping $\boldsymbol{F}_\theta$ from the sketch image domain to the set of parametric token primitives (as defined in Section 3). $\boldsymbol{F}_\theta$ follows a feed-forward transformer encoder-decoder architecture inspired by [7].

Network Architecture: Figure 4 shows the architecture of SPN. First, a convolutional [35] backbone $\boldsymbol{\Psi}_f$ extracts a set of image features $\mathbf{f} \in \mathbb{R}^{c \times h \times w}$, for input sketch image $\mathbf{X}$. The input feature map is divided into non-overlapping patches that are embedded along with fixed positional embeddings [30] and passed through a standard vi-

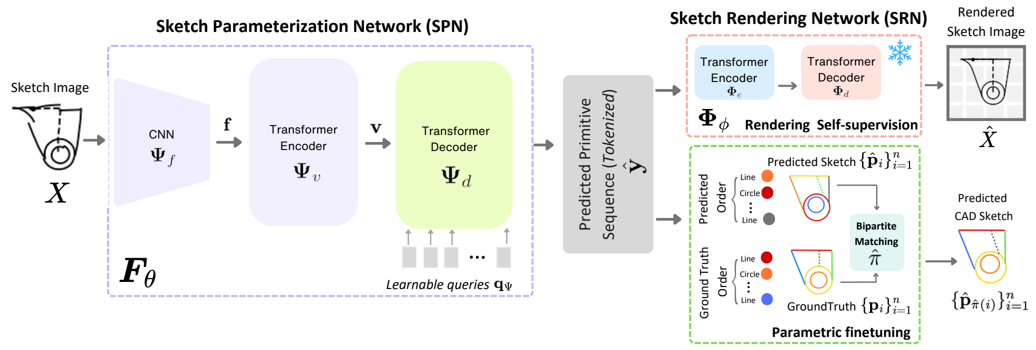
Sketch Parameterization Network (SPN)
Sketch Image
X
CNN Ψ_f
$\mathbf{f}$
Transformer Encoder Ψ_v
$\mathbf{v}$
Transformer Decoder Ψ_d
$\boldsymbol{F}_\theta$
Learnable queries $\mathbf{q}_\Psi$
Predicted Primitive Sequence (Tokenized) $\hat{\mathbf{y}}$
Sketch Rendering Network (SRN)
Transformer Encoder Φ_e
Transformer Decoder Φ_d
Φ_ϕ Rendering Self-supervision
Rendered Sketch Image
$\hat{X}$
Predicted Sketch $\{\hat{\mathbf{p}}_i\}_{i=1}^n$
Predicted Order
Line
Circle
Line
Bipartite Matching $\hat{\pi}$
Ground Truth Order
Line
Circle
Line
GroundTruth $\{\mathbf{p}_i\}_{i=1}^n$
Parametric finetuning
Predicted CAD Sketch
$\{\hat{\mathbf{p}}_{\hat{\pi}(i)}\}_{i=1}^n$

Figure 4. Overview of the *Sketch Parameterization Network* (SPN). An input raster sketch image is processed by a convolutional backbone and the produced feature map is fed to a transformer encoder-decoder for sketch parameterization. SPN is pre-trained using rendering self-supervision provided by SRN, allowing zero-shot CAD sketch parameterization, and finetuned with parameter-level annotations for few-shot scenario.

sion transformer encoder [10] $\mathbf{\Psi}_v$ to produce a set patches $\mathbf{v} \in \mathbb{R}^{\frac{hw}{d_e^2} \times d_e \times d_e}$. The decoder $\mathbf{\Psi}_d$ is a vanilla transformer [39] trained to decode a set of d randomly initialized learnable query vectors $\mathbf{q}_\Psi \in \mathbb{R}^{d \times d_q}$ into primitive tokens via self and cross-attention. Note that d_e and d_q maintain the same value as for SRN. The decoded tokens are classified independently via a linear layer with softmax activation resulting in token probabilities $\hat{\mathbf{y}} \in [0,1]^{8n \times 73}$.

Feed-Forward *vs.* Autoregressive Network Design: The suggested learning function for CAD sketch parameterization $\boldsymbol{F}_\theta$, constitutes an unordered set learning strategy via feed-forward network design that departs from the sequence modelling achieved by autoregressive networks in recent works [14, 29, 38]. Autoregressive methods are inherently order-aware as learning $\boldsymbol{F}_\theta$ results in next token prediction. Their main limitation is that the succession of primitives is clearly non-injective, and typically many primitive sequences might result in the same final geometry. Another strategy to mitigate order ambiguity is to sort primitives by parameter coordinates. Both sorting and ordered sketch parameterization strategies greatly expand the possible solution space. Additionally, autoregressive inference may introduce inconsistencies such as exposure bias [36] that impair the model's learning ability. Thus, we propose a feed-forward strategy for CAD sketch parameterization.

4.3. Rendering Self-Supervision for CAD Sketch Parameterization

While parametric supervision can train well-performing CAD sketch parameterization models from raster images [14, 38], required annotations are not always available. Particularly for the parameterization of hand-drawn sketches, recent methods [38] are trained on synthesized hand-drawn samples to ensure the availability of parameter-level annotations. Nonetheless, this approach entails acquiring knowledge of the distribution specific to artificial hand-drawn sketches, potentially hindering the model's ability to generalize effectively to sketches created by humans.

PICASSO pre-trains a sketch parametrization network SPN via a neural renderer SRN. An input hand-drawn or precise sketch image $\mathbf{X}$ is fed to SPN, which in turn encodes it into a set of primitive tokens in $\hat{\mathbf{y}} \in [0,1]^{8n \times 73}$ that are rendered by SRN. The resulting raster sketch image $\hat{\mathbf{X}}$ along with the input image $\mathbf{X}$ are used within a *multiscale* $l2$ loss similar to [34]. Specifically, the rendered sketch image by SRN and the input raster image are successively downsampled to obtain multiscale image pyramids. The image loss is computed at each pyramid level as follows,

$$\mathcal{L}_{ml2} = \sum_{s \in S} \left\| d_s \left(\mathbf{\Phi}_\phi (\boldsymbol{F}_\theta(\hat{\mathbf{X}})) \right) - d_s \left(\mathbf{X} \right) \right\|_2^2 , \qquad (1)$$

where $d_s(.)$ represents the downsampling operation for a scale s and the pyramid level is denoted by $S \in [\![1..5]\!]$. This mechanism ensures that if the rendered sketch only partially overlaps with the input raster image at a higher resolution, coarser resolutions can produce informative gradients for SPN. Note that the SRN is first learned synthetically and subsequently kept frozen during the training of the CAD parameterization model, preventing it from shifting towards a non-interpretable latent space.

Synthetic Training of SRN: Training of SRN requires a set of parametric primitives and their corresponding sketch images. In practice, we train it synthetically by randomly generating primitives and their explicit renderings. Given a a collection of sketch images $\mathcal{X} = \{\mathbf{X}_z\}_{z=1}^{N_z}$ along with token one-hot encoding $\{\mathbf{y}_z\}_{z=1}^{N_z} \in [0,1]^{8n \times 73}$, SRN is similarly learned through the multiscale $l2$ loss, formally $\mathcal{L}_{ml2} = \sum_{s \in S} \| d_s(\mathbf{\Phi}_\phi(\mathbf{y}_z)) - d_s(\mathbf{X}_z) \|_2^2$.

4.4. Zero- and Few-Shot Learning of CAD Sketch Parameterization

The pre-training described in Section 4.3 allows for CAD sketch parameterization from precise and hand-drawn sketch images when parametric annotations are limited or unavailable. For the zero-shot setting, the SPN model, pre-trained with rendering self-supervision is directly used to infer a set of parametric primitives $\{\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_n\}$ from the sketch image $\mathbf{X}$. In the few-shot scenario, the pre-trained SPN is finetuned with few annotated samples using parameter-level supervision. Note that due to the unordered set modeling strategy via the feed-forward nature of SPN, the order of predicted primitives does not necessarily match that of ground truth ones. Hence, the finetuning of SPN with parameter-level supervision is enabled through optimal bipartite matching. For predicted primitives $\{\hat{\mathbf{p}}_i\}_{i=1}^n$, we recover the permutation $\hat{\pi} \in \Pi_n$ such as

$$\hat{\pi} = \underset{\pi \in \Pi_n}{\arg\min} \sum_{i=1}^{n} \mathcal{L}_{param}(\mathbf{p}_i, \hat{\mathbf{p}}_{\pi(i)}) \ , \tag{2}$$

where Π_n is the space of all bijections from the set $[\![1..n]\!]$ to itself. The assignment $\hat{\pi}$ can be efficiently computed through Hungarian matching [22]. $\mathcal{L}_{parm}$ is a cross-entropy loss between predicted and ground truth tokens.

5. Experiments

Dataset: We evaluate PICASSO on the SketchGraphs dataset [37] of parametric CAD sketches. We adopt the pre-processing of [29, 38] where duplicates and sketches containing less than 6 primitives are removed. The final size of the dataset after filtering is around 1.53 million sketches. We use the train/val split as in [38]. Our test partition includes 5000 samples. We follow the hand-drawn synthesis strategy of [38] where primitives are subject to random translations/rotations and precise renderings are augmented with a Gaussian process model. A cross-dataset evaluation is also performed on the *CAD as a Language* dataset [14]. We filter out samples that include splines and report performance on the first 5000 sketches of the test set.

Implementation Details: Images are of size 128×128. SPN uses a U-Net [35] convolutional backbone with a ResNet34 [17] that produces a $16 \times 128 \times 128$ feature map. For SPN, the transformer encoder-decoder is formed by 4 layers each, with 8 heads and $d_q = 256$ latent dimensions. The SRN transformer has 12 layers with 8 heads and the same latent dimension as SPN. Patch size d_e is set to 16. As in [38], the maximum number of primitives is fixed to $n = 16$. We train SRN and SPN for 40 and 20 epochs respectively, with a learning rate of 1×10^{-4} and a batch size of 128. Note that SRN is trained with one-hot encoded primitive tokens as input but operates on token probabilities output by SPN for image-level pre-training. Primitives with incorrect syntax are removed. We implemented modules in Pytorch [31] and use the Adam optimizer [21].

Evaluation: For quantitative evaluation, we report both *parameter-based* and *image-based* metrics. Parameter-based metrics are computed between predicted and ground truth primitive tokens. Note that PICASSO predicts primitives as an unordered set, so correspondence w.r.t. the ground truth is recovered through optimal bipartite matching as described in Eq.(2). To ensure a fair comparison, the same correspondence recovery step is performed for the autoregressive method of [38]. We report *accuracy* and parametric *mean-squared error (ParamMSE)*. Accuracy considers the whole predicted sequence and ParamMSE is computed on the parameters of predicted tokens. Image-based metrics are reported on the explicit rendering of predicted primitive sequences. We measure a normalized pixel-wise *Mean Squared Error* (*MSE*) and bidirectional *Chamfer Distance* (*CD*) [42]. To compute CD, a set of points is uniformly sampled on foreground pixel coordinates of the explicitly rendered sketches.

	Precise Sketch Images				Hand-drawn Sketch Images			
Method	*Acc*	*ParamMSE*	*ImgMSE*	*CD*	*Acc*	*ParamMSE*	*ImgMSE*	*CD*
Resnet34	0.465	908	0.199	5.883	0.396	1048	0.240	6.908
PpaCAD [40]	0.524	589	0.195	5.097	0.464	744	0.244	6.904
Vitruvion [38]	0.537	624	0.186	4.901	0.461	685	0.237	5.258
PICASSO (w/o pt.)	0.681	326	0.134	2.344	0.595	451	0.156	2.789
PICASSO (w/ pt.)	**0.751**	**281**	**0.075**	**0.729**	**0.658**	**365**	**0.117**	**1.090**

Table 1. Comparison with Vitruvion [38] and Resnet34, trained on $16k$ samples from SketchGraphs dataset [37]. Performance for PICASSO is reported w/o and w/ self-supervised pre-training.

	Precise Sketch Images				Hand-drawn Sketch Images			
Method	*Acc*	*ParamMSE*	*ImgMSE*	*CD*	*Acc*	*ParamMSE*	*ImgMSE*	*CD*
Resnet34	0.524	829	0.189	5.698	0.448	946	0.230	6.692
PpaCAD [40]	0.562	601	0.272	6.600	0.508	754	0.289	8.193
Vitruvion	0.560	608	0.190	5.568	0.483	664	0.239	5.818
PICASSO	**0.809**	**199**	**0.067**	**0.739**	**0.669**	**360**	**0.120**	**1.715**

Table 2. Cross-dataset few-shot evaluation on *CAD as a Language* [14] dataset. Methods are trained on $16k$ samples.

5.1. Few-shot and Zero-shot CAD Sketch Parameterization

The proposed PICASSO enables the pre-training of CAD sketch parameterization directly from raster sketch images via rendering self-supervision. In this section, we evaluate the effectiveness of self-supervised pre-training for few and zero-shot settings. For results on hand-drawn sketch images, we follow [38] and experiment on synthetic sketches to ensure data availability.

Few-shot Evaluation: For the few-shot setting, we first pre-train PICASSO with rendering self-supervision (defined in Eq.(1)). The learned CAD sketch parameterization model is subsequently fine-tuned on smaller, curated sets of parameterized sketches (as discussed in Section. 4.4). Note that two separate models are pre-trained and fine-tuned for hand-drawn and precise sketch images. In Figure 14, the pre-trained PICASSO (w/ pt.) is compared to its from-scratch counterpart (w/o pt.). Overall, the pre-training outperforms learning from scratch across different sizes of fine-tuning datasets both with precise and hand-drawn images. A comparison with Vitruvion [38], PpaCAD [40], and ResNet34 baseline on a 16k-shot setting is reported in Table 1. We observe that both Vitruvion and PpaCAD completely underperform when trained with only a small num-

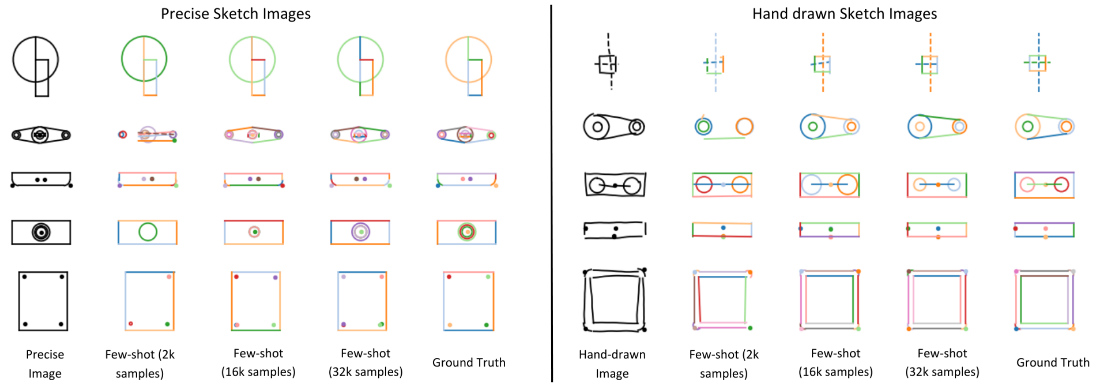

Figure 5. Few-shot setting. Qualitative results of PICASSO learned CAD sketch parameterization from precise and hand-drawn sketches.

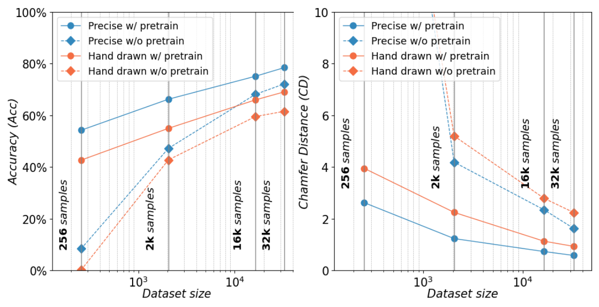

Figure 6. Effectiveness of rendering self-supervision on a few-shot evaluation. PICASSO is pre-trained on a dataset of both precise and hand-drawn sketches. We report (**left**) parametric *Accuracy (Acc)* and (**right**) image-level *Chamfer Distance (CD)* . Pre-trained SPN (w/ pre-train) is compared to a from-scratch counterpart (w/o pre-train). Best viewed in colors.

ber of annotated samples. Furthermore, we demonstrate in Figure 5 (**right**) that PICASSO is able to to parameterize challenging hand-drawn sketches even when fine-tuned with a set of only 2k annotated samples. To assess the generalization capabilities of all methods, a cross-dataset evaluation is considered, where few-shot models trained on Sketchgraphs are tested on the CAD as a Language dataset [14]. Table 2 shows that PICASSO consistently outperforms Vitruvion [38], PpaCAD [40], and the Resnet34 baseline.

Zero-shot Evaluation: By leveraging rendering self-supervision, PICASSO can estimate the parameters of sketches directly without requiring parametric supervision. We evaluate the performance of PICASSO on the challenging zero-shot CAD sketch parameterization scenario in Figure 7 (**left**). Comparison is performed w.r.t. to PMN [4], a recent self-supervised method for vectorized sketch abstraction. PMN operates on strokes (not images) that we form by sampling points on primitives. To adapt PMN for CAD sketch parameterization, we parameterize the predicted drawing primitive (stroke and type) via least-squares fitting. The network is trained on the SketchGraphs dataset [37]. We observe that PICASSO surpasses PMN

Method	*ImgMSE*	*CD*
PMN [4]	0.233	3.243
PICASSO	**0.184**	**1.880**

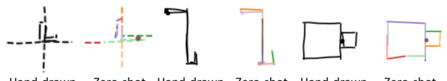

Figure 7. Zero-shot evaluation on hand-drawn images. (**left**) Comparison w.r.t. the method of [4] adapted for CAD sketch parameterization and trained on the SketchGraphs dataset. (**right**) Qualitative results for PICASSO learned in a zero-shot setting.

Method	*ImgMSE*	*CD*
Initial	0.047	0.24
SPN + Diffvg	0.087	0.48
PICASSO	**0.045**	**0.21**

Precise	**Zero Shot**	**Few Shot (16k)**	
Method	*CD* ↓	*Acc* ↑	*CD* ↓
SPN + DiffVG	5.84	0.47	2.49
PICASSO	**1.28**	**0.73**	**1.64**

Table 3. SRN-PICASSO is compared to the differentiable renderer DiffVG [23]. **(left)** Test-time optimization setting. **(right)** End-to-end training comparison.

performance in terms of image-based metrics. Note that we do not report *Acc* and *ParamMSE* on the zero-shot setting as any given CAD sketch can be constructed by arbitrarily many parameterizations and a self-supervised method cannot exactly match the parameters employed by the designer (by merging co-linear lines, flipping start-end points, etc.). Figure 7 (**right**) shows some qualitative results of PICASSO on the zero-shot setting. Note that despite the large complexity of the task, PICASSO still recovers plausible sketch parameterization.

5.2. Ablation Study

SRN vs DiffVG: We start by comparing the proposed SRN to the differentiable rendered DiffVG [23] on a test-time optimization setting. In particular, rendering self-supervision by both SRN and DiffVG is used to enhance CAD parameterization produced by a parameterically supervised SPN at test-time. This is similar to the widely used refinement of Scalable Vector Graphics (SVG), where differentiable renderers, such as DiffVG [23], are commonly used to optimize predicted SVG parameters for an input sketch image. Table 3 **(left)** reports the results after optimization. DiffVG fails to optimize the sketch parameters and can even

Loss Type	*ImgMSE*	*CD*
Binary Cross Entropy	0.021	0.116
$l2$	0.022	0.119
Multiscale $l2$	**0.020**	**0.109**

Table 4. Effect of different loss functions on SRN training.

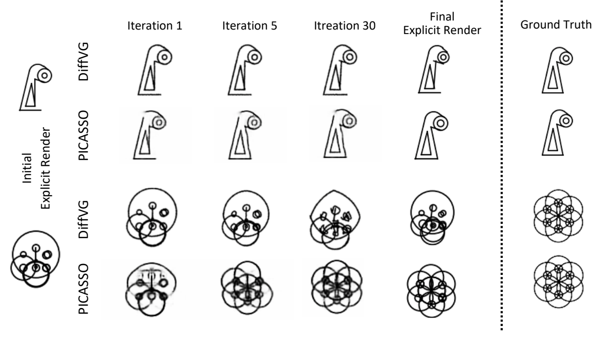

Figure 8. Test-time optimization with SRN-PICASSO. Comparison with DiffVG [23]. SRN-PICASSO progressively improves the parameterization and results in smooth transitions.

make the initial prediction less favourable. On the contrary, the proposed SRN improves the initial prediction in terms of image-based metrics. In Figure 8, qualitative results of the test-time optimization are illustrated. We also compare SRN to DiffVG in the pretraining setting. While SPN can also be pretrained with DiffVG, we observed the following issues: (1) DiffVG is sensitive to SPN network size and primitive thickness due to sparsity of gradients [23]. It remains stable only with a small SPN (1-layer transformer) with $4\times$ thicker primitives, whereas the original setting leads to degenerate geometries. (2) Convergence is slow, further hindered by DiffVG's batching inability[1]. As shown in Table 3 **(right)**, a small SPN pretrained with SRN surpasses DiffVG-based pretaining in zero/few-shot settings.

SRN Rendering Performance: Table 4 ablates different losses used to train SRN for parametric rendering. The quality of predicted raster images is evaluated w.r.t. ground truth explicit renderings. The multiscale $l2$ loss achieves the best performance on reported image-based metrics.

SPN with Parametric Supervision: We evaluate SPN of PICASSO for CAD sketch parameterization of precise and hand-drawn sketch images. For this evaluation, SPN is trained with parametric supervision on the complete SketchGraphs dataset. Results are reported in Table 5. SPN surpasses both the autoregressive Vitruvion [38] and the non-autoregressive Resnet34 baseline by a large margin. It is also important to mention that SPN is approximately $10\times$ faster than Vitruvion thanks to its feed-forward nature.

SPN Architecture: Table 6 **(left)** depicts the ablation study of the architectural components of SPN. A transformer trained on sorted primitives is contrasted to a transformer learned with bipartite matching. Bipartite matching introduces a significant performance gain as the network does not allocate learning capacity on capturing primitive order. Finally, it is shown that the UNet-like backbone Ψ_f enhances the performance via multiscale feature extraction.

[1]Issues #17 and #67 on https://github.com/BachiLi/diffvg

	Precise Sketch Images				Hand-drawn Sketch Images			
Method	*Acc*	*ParamMSE*	*ImgMSE*	*CD*	*Acc*	*ParamMSE*	*ImgMSE*	*CD*
Resnet34	0.702	433	0.146	2.842	0.628	565	0.167	3.531
Vitruvion	0.813	226	0.052	0.267	0.658	391	0.112	0.875
SPN	**0.878**	**107**	**0.047**	**0.237**	**0.827**	**146**	**0.075**	**0.499**

Table 5. Evaluation of the proposed feed-forward architecture on the SketchGraphs dataset [37]. For all methods, separate models are trained to parameterize precise and hand-drawn sketch images.

Method	*Acc*	*ParamMSE*	*ImgMSE*	*CD*
Sorted	0.722	432	0.114	2.20
BM	0.831	165	0.053	0.33
BM+UNet	**0.878**	**107**	**0.047**	**0.23**

Method	Precise	Hand-drawn
GTParams$_{\text{(upper bound)}}$	0.90	
Vitruvion	0.62	0.59
SPN	**0.64**	**0.62**

Table 6. **(left)** Ablation on the various architectural components of SPN-PICASSO. We assess the impact of Bipartite Matching (BM) and the UNet-based backbone on SPN. **(right)** Performance for the constraint prediction of [38] with input parameterization by SPN and [38] in terms of next-token accuracy. GTParams represents the upper bound achieved with the ground truth primitives.

Impact of CAD Sketch Parameterization on CAD Constraint Inference: Parameterized primitives inferred by PICASSO can be directly constrained by existing constraint prediction models like [38]. In Table 6 **(right)**, we report performance for the constraint prediction model of [38] for input CAD parameterization predicted by SPN and [38]. Our model improves constraint prediction performance by providing better CAD sketch parameterization.

6. Conclusions and Future Works

In this paper, PICASSO is introduced, a novel framework for CAD sketch parameterization. It includes a feed-forward Sketch Parameterization Network (SPN) and a Sketch Rendering Network (SRN). SRN is a neural differentiable renderer that enables the use of rendering self-supervision for large-scale CAD sketch parameterization without requiring parametric annotations. This in turn accounts for real-world scenarios where large annotated hand-drawn CAD sketch datasets are not available. Experiments on few- and zero-shot settings validate the effectiveness of the proposed framework. Due to data unavailability, PICASSO was evaluated on four types of primitives (lines, points, arcs, and circles). In the supplementary, preliminary experiments are conducted with synthetic b-splines suggesting the applicability of PICASSO to free-form curves. This investigation in the context of CAD is left for the future.

7. Acknowledgements

This work is supported by the National Research Fund (FNR), Luxembourg, BRIDGES2021/IS/16849599/FREE-3D project and by Artec3D.

References

[1] CADSketch - CAD International — cad.com.au. https://cad.com.au/software/cadsketch/. [Accessed 15-07-2024]. 1

[2] Onshape. https://www.onshape.com. 1, 3, 19

[3] Solidworks. https://www.solidworks.com. 1

[4] Stephan Alaniz, Massimiliano Mancini, Anjan Dutta, Diego Marcos, and Zeynep Akata. Abstracting sketches through simple primitives. In *ECCV*, 2022. 2, 3, 7, 13, 16

[5] Ayan Kumar Bhunia, Ayan Das, Umar Riaz Muhammad, Yongxin Yang, Timothy M. Hospedales, Tao Xiang, Yulia Gryaditskaya, and Yi-Zhe Song. Pixelor: A competitive sketching ai agent. so you think you can sketch? *ACM TOG*, 2020. 3

[6] Francesco Buonamici, Monica Carfagni, Rocco Furferi, Lapo Governi, Alessandro Lapini, and Yary Volpe. Reverse engineering modeling methods and tools: a survey. *Computer-Aided Design and Applications*, 2018. 2

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020. 4

[8] Paul Chastell. Under the hood: Onshape sketches, 2015. Accessed: 2024-15-07. 3

[9] Ye Chen, Bingbing Ni, Xuanhong Chen, and Zhangli Hu. Editable image geometric abstraction via neural primitive assembly. In *ICCV*, 2023. 3

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 5

[11] Elona Dupont, Kseniya Cherenkova, Anis Kacem, Sk Aziz Ali, Ilya Arzhannikov, Gleb Gusev, and Djamila Aouada. Cadops-net: Jointly learning cad operation types and steps from boundary-representations. In *3DV*, 2022. 1

[12] Elona Dupont, Kseniya Cherenkova, Dimitrios Mallis, Gleb Gusev, Anis Kacem, and Djamila Aouada. Transcad: A hierarchical transformer for cad sequence inference from point clouds. *ECCV*, 2024. 1

[13] Vage Egiazarian, Oleg Voynov, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, and Evgeny Burnaev. Deep vectorization of technical drawings. In *ECCV*, 2020. 3

[14] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. In *NeurIPS*, 2021. 1, 2, 3, 5, 6, 7

[15] David Ha and Douglas Eck. A neural representation of sketch drawings. In *ICLR*, 2018. 3

[16] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022. 4

[17] Kaiming He, X. Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2015. 6

[18] Zhewei Huang, Wen Heng, and Shuchang Zhou. Learning to paint with model-based deep reinforcement learning. In *ICCV*, 2019. 3

[19] Ahmet Serdar Karadeniz, Dimitrios Mallis, Nesryne Mejri, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Davinci: A single-stage architecture for constrained cad sketch inference. In *BMVC*, 2024. 3

[20] Mohammad Sadil Khan, Elona Dupont, Sk Aziz Ali, Kseniya Cherenkova, Anis Kacem, and Djamila Aouada. Cad-signet: Cad language inference from point clouds using layer-wise sketch instance guided attention. In *CVPR*, 2024. 1

[21] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. 6

[22] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 1955. 6

[23] Tzu-Mao Li, Michal Lukáč, Michaël Gharbi, and Jonathan Ragan-Kelley. Differentiable vector graphics rasterization for editing and learning. *ACM TOG*, 2020. 3, 4, 7, 8, 13

[24] Yi Li, Timothy M Hospedales, Yi-Zhe Song, and Shaogang Gong. Free-hand sketch recognition by multi-kernel feature learning. *CVIU*, 2015. 3

[25] Xu Ma, Yuqian Zhou, Xingqian Xu, Bin Sun, Valerii Filev, Nikita Orlov, Yun Fu, and Humphrey Shi. Towards layer-wise image vectorization. In *CVPR*, 2022. 3

[26] Dimitrios Mallis, Ali Sk Aziz, Elona Dupont, Kseniya Cherenkova, Ahmet Serdar Karadeniz, Mohammad Sadil Khan, Anis Kacem, Gleb Gusev, and Djamila Aouada. Sharp challenge 2023: Solving cad history and parameters recovery from point clouds and 3d scans. overview, datasets, metrics, and baselines. In *ICCVW*, 2023. 1

[27] Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. General virtual sketching framework for vector line art. *ACM TOG*, 2021. 3

[28] Reiichiro Nakano. Neural painters: A learned differentiable constraint for generating brushstroke paintings. *ArXiv*, 2019. 3

[29] Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas J Guibas, and Peter Wonka. Sketchgen: Generating constrained cad sketches. In *NeurIPS*, 2021. 2, 3, 5, 6

[30] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam M. Shazeer, Alexander Ku, and Dustin Tran. Image transformer. In *ICML*, 2018. 4

[31] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zach DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 6

[32] Anran Qi, Yulia Gryaditskaya, Tao Xiang, and Yi-Zhe Song. One sketch for all: One-shot personalized sketch segmentation. *IEEE TIP*, 2022. 3

[33] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *CVPR*, 2021. 3

[34] Pradyumna Reddy, Michaël Gharbi, Michal Lukác, and Niloy Jyoti Mitra. Im2vec: Synthesizing vector graphics without vector supervision. In *CVPR*, pages 7338–7347, 2021. 5

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*. Springer, 2015. 4, 6

[36] Florian Schmidt. Generalization in generation: A closer look at exposure bias. *arXiv preprint arXiv:1910.00292*, 2019. 5

[37] Ari Seff, Yaniv Ovadia, Wenda Zhou, and Ryan P. Adams. SketchGraphs: A large-scale dataset for modeling relational geometry in computer-aided design. In *ICMLW*, 2020. 1, 2, 3, 6, 7, 8, 13, 14, 15

[38] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. Vitruvion: A generative model of parametric cad sketches. In *ICLR*, 2022. 1, 2, 3, 4, 5, 6, 7, 8, 11, 12, 13, 15

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017. 5

[40] Xiaogang Wang, Liang Wang, Hongyu Wu, Guoqiang Xiao, and Kai Xu. Parametric primitive analysis of cad sketches with vision transformer. *IEEE Transactions on Industrial Informatics*, 2024. 3, 6, 7, 13

[41] Karl DD Willis, Pradeep Kumar Jayaraman, Joseph G Lambourne, Hang Chu, and Yewen Pu. Engineering sketch generation for computer-aided design. In *CVPR*, 2021. 3

[42] Tong Wu, Liang Pan, Junzhe Zhang, Tai Wang, Ziwei Liu, and Dahua Lin. Density-aware chamfer distance as a comprehensive metric for point cloud completion. *arXiv preprint arXiv:2111.12702*, 2021. 6

[43] Peng Xu, Timothy M Hospedales, Qiyue Yin, Yi-Zhe Song, Tao Xiang, and Liang Wang. Deep learning for free-hand sketch: A survey. *IEEE TPAMI*, 2022. 3

[44] Xianghao Xu, Wenzhe Peng, Chin-Yi Cheng, Karl DD Willis, and Daniel Ritchie. Inferring cad modeling sequences using zone graphs. In *CVPR*, 2021. 1

[45] Lumin Yang, Jiajie Zhuang, Hongbo Fu, Xiangzhi Wei, Kun Zhou, and Youyi Zheng. Sketchgnn: Semantic sketch segmentation with graph neural networks. *ACM TOG*, 2021. 3

[46] Yuezhi Yang and Hao Pan. Discovering design concepts for cad sketches. In *NeurIPS*, 2022. 1, 3

[47] Qian Yu, Yongxin Yang, Feng Liu, Yi-Zhe Song, Tao Xiang, and Timothy M Hospedales. Sketch-a-net: A deep neural network that beats humans. *IJCV*, 2017. 3

[48] N. Zheng, Yifan Jiang, and Ding Huang. Strokenet: A neural painting environment. In *ICLR*, 2018. 3

PICASSO: A Feed-Forward Framework for Parametric Inference of CAD Sketches via Rendering Self-Supervision

Supplementary Material

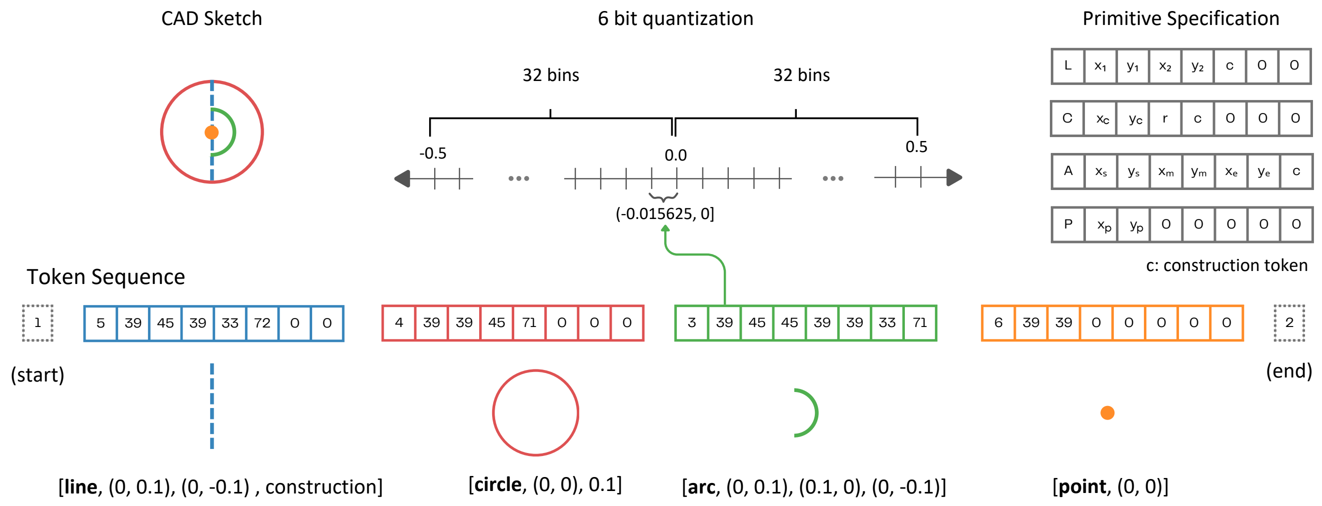

Figure 9. Overview of the tokenization process. A parameterized CAD sketch is represented by a set of primitives, each comprising a sequence of 8 tokens. The first primitive token specifies the primitive type, followed by quantized primitive parameters. As depicted in the example, the parametric value 0.0 is mapped on the bin $(-0.015625, 0]$ due to the 6-bit quantization. Additional tokens are padded with the 0 token value. The primitive sequence is initiated with the *start* token and completed by the *end* token.

This supplementary material includes various details that were not reported in the main paper due to space constraints. To demonstrate the benefit of the proposed PICASSO, we also expand our experimental evaluation.

8. System Details

We start by reiterating details of the tokenization strategy followed by PICASSO. This section also discusses the effect of the multiscale loss $\mathcal{L}_{ml2}$ and reports inference times for our proposed method and that of [38].

8.1. Tokenization

For our problem formulation, each primitive $\mathbf{p}_i$ is expressed as a collection of 8 tokens $\{t_i^j\}_{j\in[1,8]}$ with $t_i^j \in [0, 72]$ that capture both types and quantized primitive parameters. A detailed description of token types and corresponding token values is shown in Table 7. In Fig. 9, we present an overview of the tokenization process.

8.2. Multiscale $l2$ loss.

Rendering self-supervision via the proposed Sketch Rendering Network (SRN) is facilitated by a multiscale $l2$ loss denoted by $\mathcal{L}_{ml2}$. The multiscale $l2$ loss enables effective rendering self-supervision for precise as well as hand-drawn sketch images. Even though discrepancies are inevitably introduced due to the imprecise nature of hand-drawn lines, the loss at a lower resolution can still provide an informative learning signal. A visualisation of image pyramids constructed for $\mathcal{L}_{ml2}$ computation is presented in Fig. 10. Qualitative results of renderings produced by SRN when trained with different image-level losses are given in Fig. 11. We observe that utilizing the multiscale $l2$ loss during training, results in sharper images and accurate rendering of

Token Value	*Token Description*
0	Padding
1	Start
2	End
3	Arc
4	Circle
5	Line
6	Point
$[7, 70]$	Quantized primitive parameters
71	Construction Primitive
72	Non-Construction Primitive

Table 7. Description of tokens used in our problem formulation.

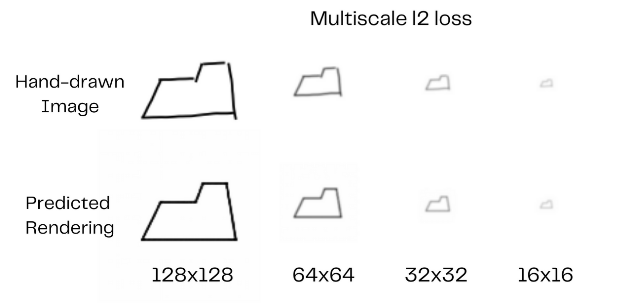

Figure 10. Illustration of the $\mathcal{L}_{ml2}$ for visually supervised CAD parameterization. The immediate support between the predicted rendering and the imprecise hand-drawn sketch image increases at lower resolutions. This mechanism compensates for noisy gradient estimates due to partial overlap at higher resolutions.

finer details. Improved SRN renderings in turn lead to the computation of informative gradients that enable rendering self-supervision and zero / few-shot learning scenarios for PICASSO, as demonstrated in Sections 5.2 of the main paper.

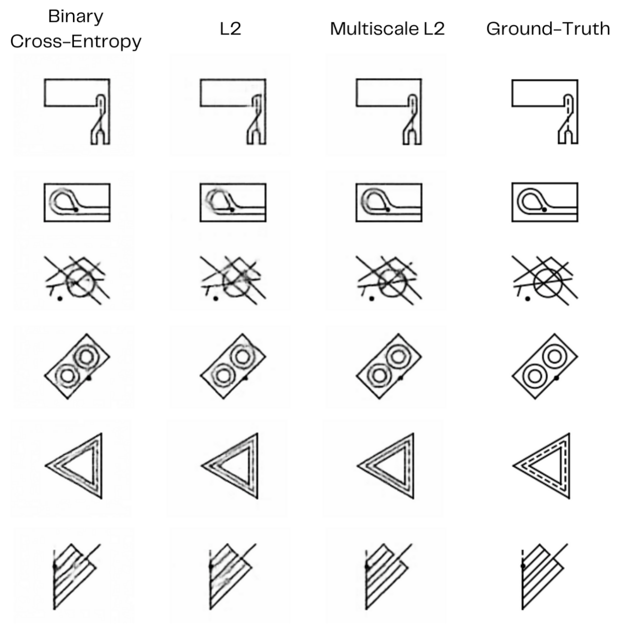

Figure 11. Sketch renderings by the SRN-PICASSO, trained using different image-level losses. Training SNR via a multiscale $l2$ loss results in sharp CAD sketch renderings. In contrast, employing binary cross-entropy or a standard $l2$ loss during the learning process may lead to renderings that are blurred, lack fine details, or have disconnected primitives.

8.3. Inference Time Comparison

In Table 8, we report inference time in seconds for our method and that of [38]. We observe that our Sketch Parametrization Network (SPN) enables faster inference. In contrast, Vitruvion [38] is an autoregressive method that requires multiple forward passes per sample, leading to increased total inference time.

Method	Inference (*sec*)
Vitruvion [38]	1.1005
SPN-PICSASSO	0.1101

Table 8. Inference times per-sample for our proposed method and that of [38]. Results are computed for a batch size of 1.

9. Metrics

The metrics utilized for quantitative evaluation are detailed as follows.

Acc: To enable the computation of parameter-based metrics, the permutation $\hat{\pi} \in \Pi_n$ of the predicted w.r.t the ground truth is recovered with bipartite matching. Accuracy is computed as:

$$Acc = \frac{1}{T_{Acc}} \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 0] \cdot \mathbb{1}[t_i^j = \hat{t}^j_{\hat{\pi}(i)}] \quad (3)$$

where $\hat{t}^j_{\hat{\pi}(i)}$ and t_i^j and the predicted and ground truth token sequences respectively, $n_z = |\{\boldsymbol{p}^z\}|$ is the number of primitives for the z'th test sample, T_{Acc} is the number of non-padding tokens in the ground truth sequence or formally $T_{Acc} = \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 0]$, and $\mathbb{1}[.]$ is the indicator function.

ParamMSE: Parametric *mean-squared error (ParamMSE)* considers solely the parameter tokens of predicted primitives, thus type, padding, and construction tokens are excluded. Formally,

$$ParamMSE = \frac{1}{T_{MSE}} \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 6] \cdot \mathbb{1}[t_i^j < 71] \cdot (t_i^j - \hat{t}^j_{\hat{\pi}(i)})^2 \quad (4)$$

where $T_{MSE} = \sum_{i=1}^{n_z} \sum_{j=1}^{8} \mathbb{1}[t_i^j > 6] \cdot \mathbb{1}[t_i^j < 71]$ is the number of parameter tokens in the ground truth sequence.

ImgMSE: The pixel-wise *Mean Squared Error ImgMSE* comprise two MSE terms. Formally:

$$\begin{aligned} ImgMSE = & \frac{1}{2N_F} \sum_{k=1}^{w \cdot h} \mathbb{1}[\mathbf{X}_k = 1] \cdot (\hat{\mathbf{X}}_k - \mathbf{X}_k)^2 \\ & + \frac{1}{2w \cdot h} \sum_{k=1}^{w \cdot h} (\hat{\mathbf{X}}_k - \mathbf{X}_k)^2 \end{aligned} \quad (5)$$

where $N_F = \sum_{k=1}^{w \cdot h} \mathbb{1}[\mathbf{X}_k = 1]$ is the number of foreground pixels.

CD: To compute bidirectional Chamfer Distance (CD), we form a set of foreground pixel coordinates $\zeta \in \{(i,j) \mid 1 \leq i \leq h,\ 1 \leq j \leq w\}$ for both the ground truth and predicted explicit renderings. The result is the sets $\mathrm{Z} = \{\zeta_n\}_{n=1}^{N_f}$ and $\hat{\mathrm{Z}} = \{\hat{\zeta}_n\}_{n=1}^{\hat{N}_f}$ where N_f and $\hat{N}_f$ is the number of foreground pixels for ground truth and prediction explicit renderings respectively. Bi-directional chamfer distance is given by:

$$CD = \frac{1}{2\hat{N}_f} \sum_{n=1}^{\hat{N}_f} \min_{\zeta_k \in \mathrm{Z}} \|\hat{\zeta}_n - \zeta_k\|_2^2 + \frac{1}{2N_f} \sum_{n=1}^{N_f} \min_{\hat{\zeta}_k \in \hat{\mathrm{Z}}} \|\zeta_n - \hat{\zeta}_k\|_2^2, \tag{6}$$

10. Implementation Details for Comparative Analysis

To evaluate the effectiveness of the proposed PICASSO, comparisons are performed in two settings; *(1)* For few-shot w.r.t. the state-of-the-art autoregressive method of Vitruvion [38] and a non-autoregressive baseline based on a convolutional ResNet34 backbone and *(2)* for zero-shot w.r.t the Primitive Matching Network of [4]. The proposed SRN is also contrasted to the differentiable renderer DiffVG [23]. This section will expand on implementation details related to these methods.

Vitruvion: We train Vitruvion [38] on the SketchGraphs [37] dataset using the publicly available implementation[2]. For autoregressive CAD parameterization, we select the next token via $argmax$ instead of the nucleus sampling used for sketch generation. This modification enhances parameterization performance, while also ensuring consistent reproducibility of results. All hyperparameters are set as in the original paper [38].

ResNet34: To form a non-autoregressive baseline we trained a ResNet34 followed by global pooling. The output of the convolutional backbone is fed into a Multi-Layer Perceptron (*MLP*) with 2 linear layers and a ReLU activation. The final token predictions are produced by a softmax on the output logits of the MLP.

PpaCAD: We implemented the concurrent method PpaCAD [40] as no public code is available. Image encoding uses a 2-layer patch MLP followed by a transformer. Separate losses are computed for each primitive type, parameter, and construction flag. Further details are in [40].

Primitive Matching Network (PMN): For comparison with PMN, we re-train on the Sketchgraphs dataset using the publicly available code[3]. We form strokes by sampling points on parametric primitives that are provided as input to PMN. The input for PMN comprises multiple sets of coordinates, with each set uniquely representing a single distinct primitive. Note that this scenario presents a less complex challenge than that encountered in PICASSO, where parameterization is derived directly from raster images. In such case, primitives may overlap or be positioned in close proximity, significantly increasing the complexity of the parameterization task. Strokes are processed by PMN to obtain *drawing primitives* that consist of the abstracted output stroke and the stroke type (line, circle, half-circle and point). Finally, the output strokes are parameterized via least-square fitting based on their predicted types.

[2] https://github.com/PrincetonLIPS/vitruvion

Method	*Acc*	*ParamMSE*	*ImgMSE*	*CD*
PICASSO (w/o pt.)	0.595	451	0.156	2.789
PICASSO (w/o pt. + semi-supervised)	0.608	432	0.145	1.833

Table 9. Semi-supervised learning results for PICASSO.

DiffVG: Comparison to DiffVG [23] is performed on 1) pre-training and 2) test-time optimization settings as discussed in Section 5.3 of the main paper. Predicted sequences are transformed into Bezier paths to enable an image-level loss. For the pre-training setting, SPN is trained with respect to the image loss. For test-time optimization, the paths are iteratively updated through differentiable rendering. As already noted, DiffVG can only update path parameters, but it is unable to change discrete decisions like path types. Lines are converted to paths with two endpoints and no control points. Points are also composed of 2 endpoints formed by shifting the point coordinate by 1 quantization unit. Arcs and circles are formed by Bezier paths, computed via the Python package svgpathtools[4]. After optimization, paths are converted back to the considered primitives (lines, arcs, circles, and points) and evaluated directly w.r.t ground truth sequences.

11. Semi-Supervised CAD Sketch Parameterization via Rendering Supervision

Rendering supervision enabled by the proposed SRN can also be applied to other learning schemes. We investigate a semi-supervised learning scenario where SRN is trained through rendering supervision on unlabelled sketch images and parametric supervision on a smaller set of parameterized sketches (16k samples). Table 9 shows quantitative results of the semi-supervised PICASSO compared to its parametrically supervised counterpart on 16k samples. By leveraging unlabelled sketches through rendering supervi-

[3] https://github.com/ExplainableML/sketch-primitives

[4] https://github.com/mathandy/svgpathtools

sion, the semi-supervised model can achieve better performance. The difference is more noticeable in terms of image-based metrics, as rendering supervision can result in a model that discovers plausible geometric reconstructions, that might depart from the ground truth CAD sketch parameterization.

12. Sensitivity to Rendering Quality

For PICASSO, rendering self-supervision is facilitated by neural differentiable rendering via SRN. We conduct an ablation study to explore how variations in SRN's rendering performance influence the effectiveness of self-supervised pretraining. To that end, we vary SRN rendering quality by training SRN for different number of epochs. Specifically, PICASSO is pretrained via rendering self-supervision using different SRN renderers trained for 5, 10, 30 and 40 epochs. The results are presented in Figure 12. We find that SRN rendering performance (measured in terms of chamfer distance) stabilizes after a few training epochs *(orange line)*. However, the zero-shot performance of SPN is notably stronger when using a fully trained SRN *(blue line)*. As the neural rendering improves, fewer artifacts are introduced and SRN can more accurately replicates the input sketch parameters.

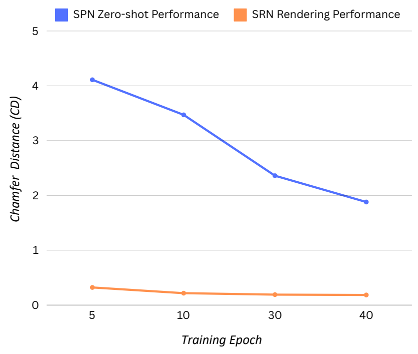

Figure 12. *(orange)* Rendering performance (in terms of chamfer distance) of SRN trained for different number of epochs. *(blue)* Zero-shot performance of PICASSO-SPN when pretrained by the aforementioned SRN renderers.

13. Extension to Other Primitives

The main experiments of PICASSO are conducted for parameterizing lines, circles, arcs, and points. Free-form curves such as B-splines, hyperbolas, and NURBS are excluded similarly to recent works as they are underrepresented in existing datasets (*e.g.* b-splines are 2.57% of SketchGraphs primitives [37]). As a preliminary experiment for future work, we trained and tested SPN and SRN on a *synthetic* dataset including randomly generated b-splines. Training is performed for 20 epochs and a different random sketch is sampled at each iteration. Table 10 reports a comparison to DiffVG in terms of test time optimization on 100 synthetically generated images. SRN self-supervision improves b-spline predictions of SPN and significantly surpasses DiffVG in terms of Chamfer Distance (CD). Figure 13 shows an example where SPN prediction on a synthetic sketch with B-spline is being improved with SRN supervision.

Method	*CD*
SPN	1.07
SPN+DiffVG	3.60
SPN+SRN	0.48

Table 10. Test time optimization of sketches that include B-splines. Optimization via SRN performs better in terms of Chamfer Distance (CD).

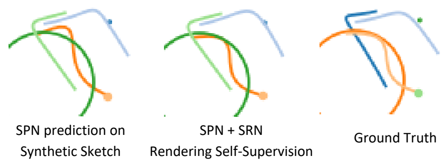
SPN prediction on Synthetic Sketch
SPN + SRN Rendering Self-Supervision
Ground Truth

Figure 13. PICASSO on synthetic CAD sketch including B-spline.

14. Additional Qualitative Results

This section expands on the qualitative evaluation reported in the main paper.

14.1. Few-shot CAD Sketch Parameterization

This subsection expands the qualitative evaluation shown in subsection 5.2 *(Few-shot Evaluation)* of the main paper. Visual results for finetuning with 2k, 16k and 32k samples are shown in Fig. 14. We observe that the 32k-shot setting results in robust CAD parameterization from challenging precise and hand-drawn sketch images, even though the network is trained only with a fraction of the original dataset ($\approx 2\%$ of the SketchGraph dataset [37]).

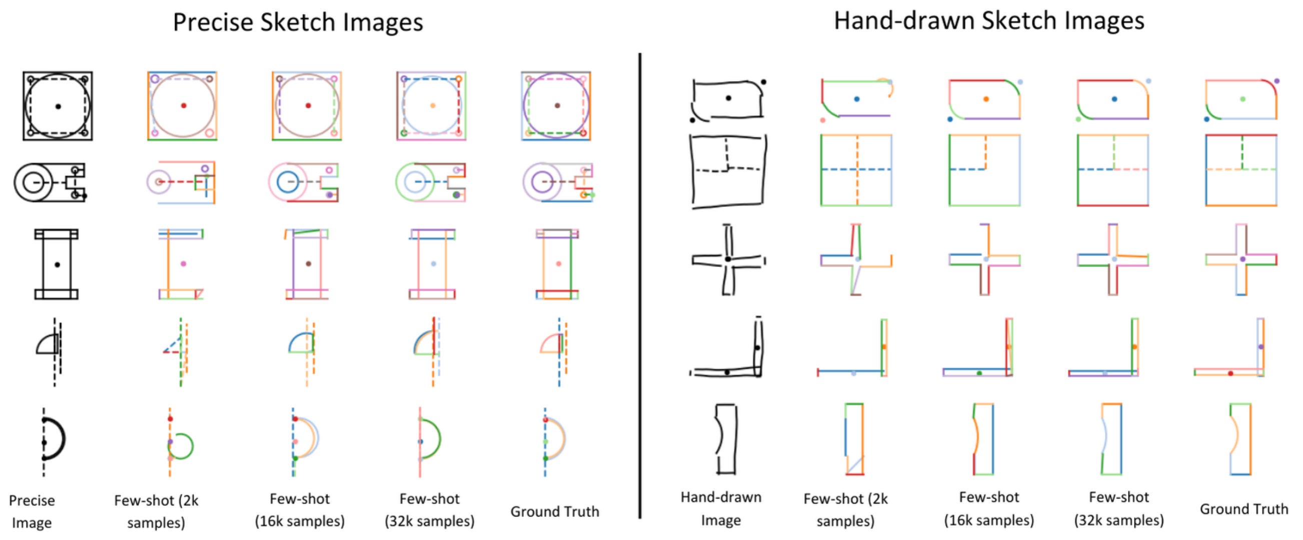

Figure 14. Few-shot setting. Qualitative results of PICASSO learned CAD sketch parameterization from precise and hand-drawn sketches. Best viewed in colors.

Fig. 15 depicts the qualitative comparison of our model with that of Vitruvion [38] for the parameterization of precise and hand-drawn sketches. It can be observed that the proposed method produces plausible parameterizations closer to the ground truth.

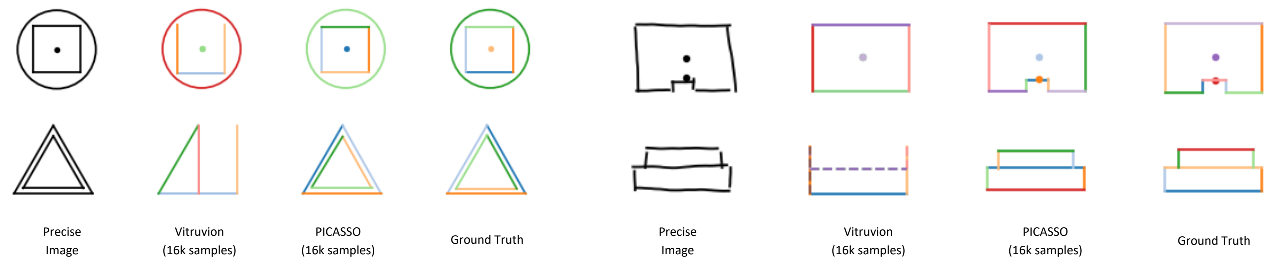

Figure 15. Visual examples of CAD sketch parameterization from hand-drawn and precise sketches by Vitruvion [38] and PICASSO on a 16k-shot setting.

14.2. Zero-shot CAD Sketch Parameterization

This subsection expands the qualitative evaluation shown in subsection 5.2 *(Zero-shot Evaluation)* of the main paper. In Fig. 16, we present visual examples of CAD sketch parameterization from hand-drawn sketches, learned via an image-level loss only. Under a complete lack of parametric supervision, PICASSO is able to roughly parameterize hand-drawn sketches. Note that compared to few-shot setting, SPN is further constrained to output a fixed number of primitives per type for the zero-shot evaluation. While PICASSO achieves plausible zero-shot parameterizations, we find that rendering self-supervision can be hindered by the discrepancy between hand-drawn sketches and the precise ones rendered by SRN. The development of hand-drawn invariant losses that can enhance zero-shot performance is identified as interesting future work.

Fig. 17 illustrates a qualitative comparison with PMN [4] for the zero-shot setting. PICASSO predicts more consistent sketches with primitives that are not geometrically far from the input image. It is important to highlight that our zero-shot model works on a more challenging setup of direct parameterization from images without having access to individual groupings of strokes contrary to PMN [4]. Also, note that we do not conduct comparison to PMN on precise images. Since PMN is aware of the grouping of distinct strokes, parameterization of precise inputs becomes a trivial task, reduced to merely identifying the types of primitive strokes.

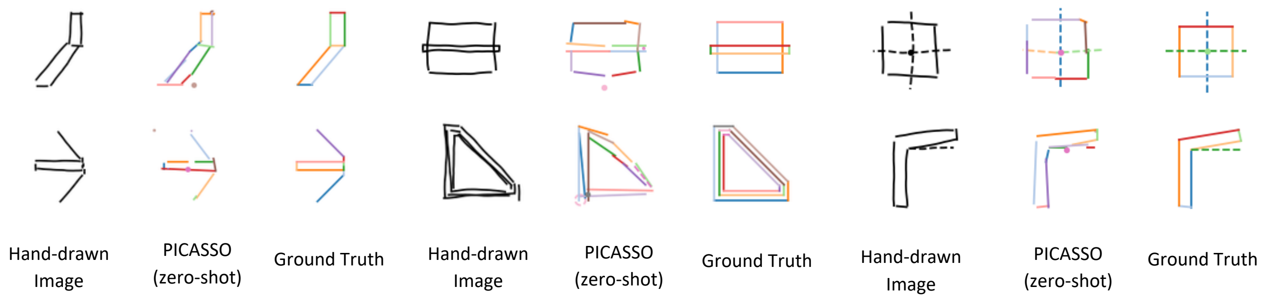

Figure 16. Qualitative results for CAD sketch parameterization of hand-drawn sketches, learned solely through rendering self-supervision with SRN-PICASSO. Best visualised in colors.

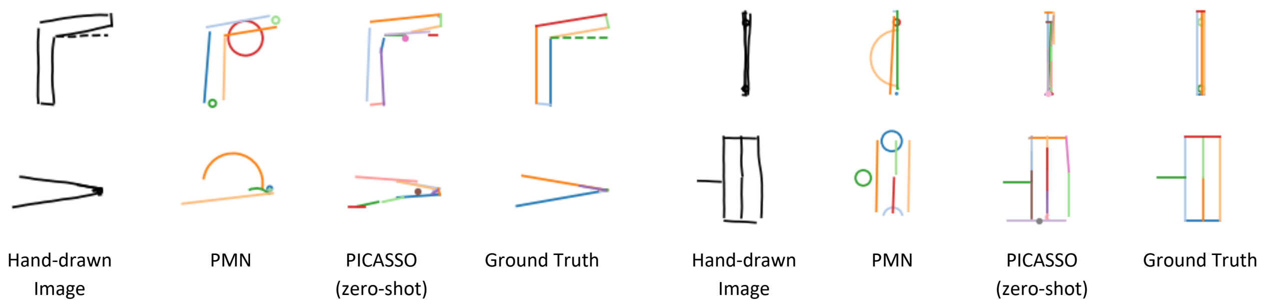

Figure 17. Zero-shot CAD sketch parameterization from hand-drawn sketches by PMN [4] and PICASSO. Note that PMN has prior information on grouping individual strokes with their coordinate points, whereas PICASSO infers directly from the image space without any grouping of primitives.

14.3. Test-time Optimization with SRN

As shown in subsection 5.3 *(SRN vs DiffVG)* of the main paper, rendering self-supervision can be used to enhance CAD sketch parameterization produced by a parameterically supervised SPN at test-time. In Fig. 18, we show qualitative results for test-time optimization of precise sketches. We observe that SRN can improve geometric reconstruction of CAD sketches at inference time.

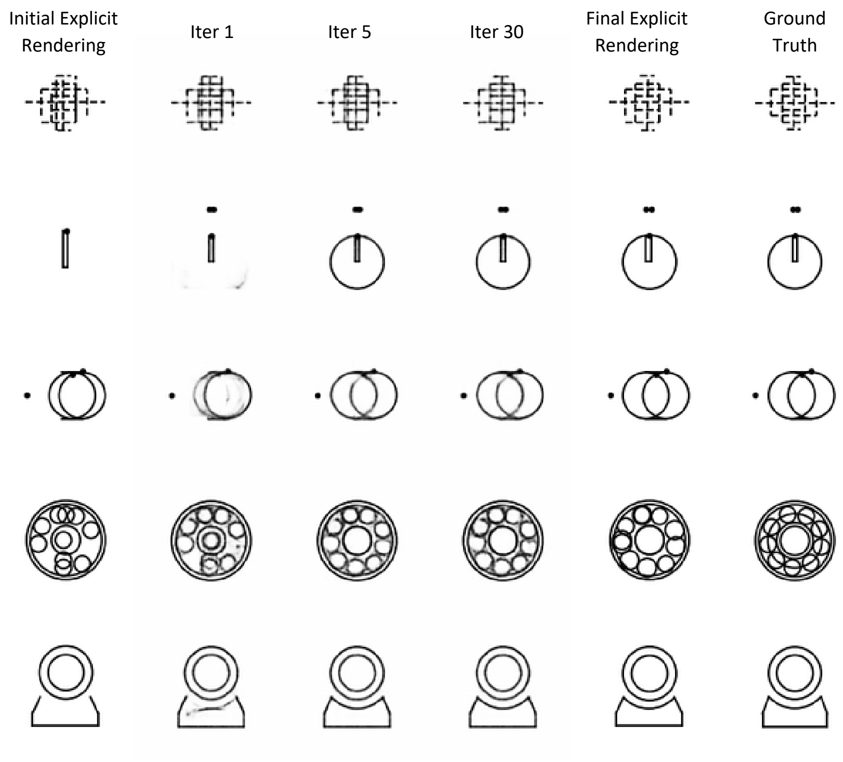

Figure 18. Test-time optimization with SRN-`PICASSO`. SRN enables the computation of an image-level loss between predicted rendering and the input precise sketch image. CAD parameterization improves over multiple backpropagation steps on a specific test sample.

15. Permutation Invariance of SRN w.r.t. Primitive Order

Since SRN processes input primitive tokens as a set, it should demonstrate permutation invariance with respect to their order. Although this invariance is not explicitly designed into the architecture, it naturally emerges from the synthetic training process, where primitives can appear in any sequence position. Figure 19 qualitatively illustrates the model's robustness to input order, with renderings of CAD sketches showing only subtle differences despite permuted primitive sequences.

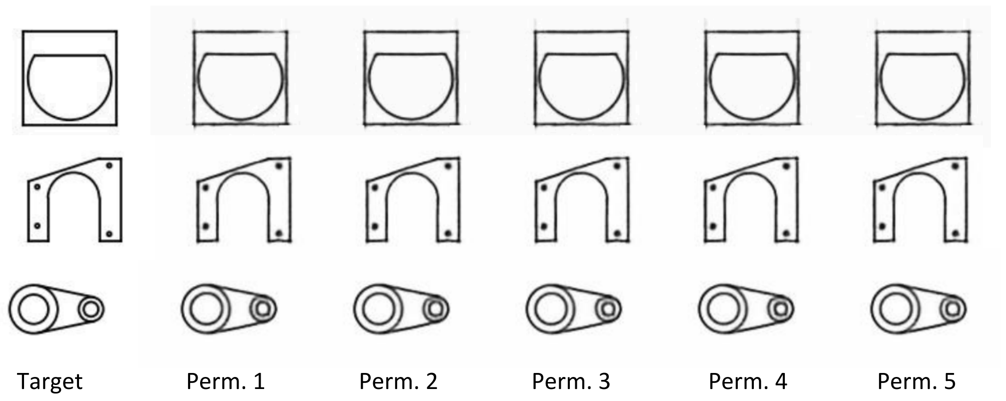

Figure 19. SRN renderings of the parametric CAD sketches where the primitives are randomly permuted.

16. Failure Case Analysis

We conduct a qualitative analysis of common failure cases identified for PICASSO finetuned with 16k samples. Figure 20 highlights several of these cases, where the model produces suboptimal parameterizations. Notable issues include difficulty handling closely positioned primitives, inaccurate coordinate predictions, missing primitives, failure to capture fine details, overly large arc predictions or combinations of the aforementioned cases.

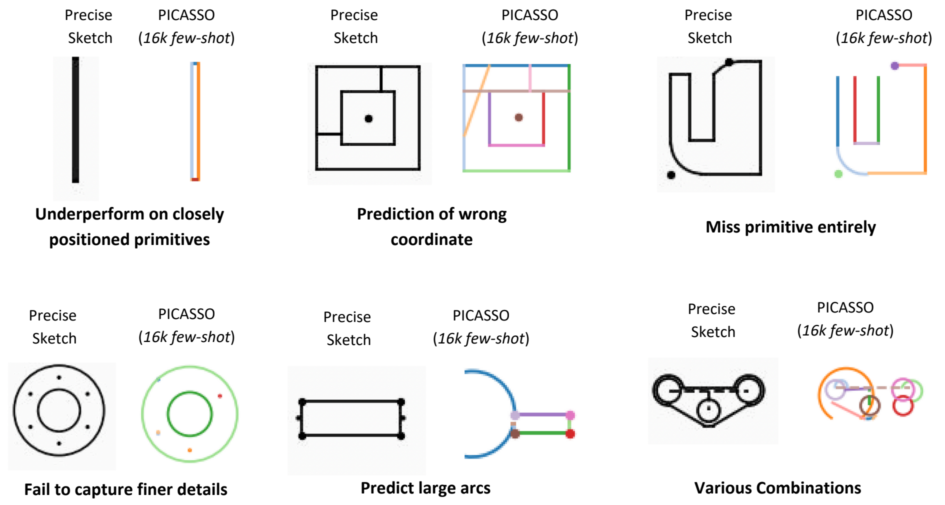

Figure 20. Common failure cases of PICASSO finetuned with parametric supervision on 16k samples.

17. PICASSO for Feature-based 3D CAD Modeling

2D CAD sketches are an essential component of feature-based CAD modeling. In Figure 21, we demonstrate how PICASSO enables the design of 3D CAD models by parameterizing hand-drawn sketches. These parameterized sketches are then uploaded to Onshape [2], where simple extrusions and revolutions are applied. This results in 3D solids that are combined to form the 3D models depicted in the figure.

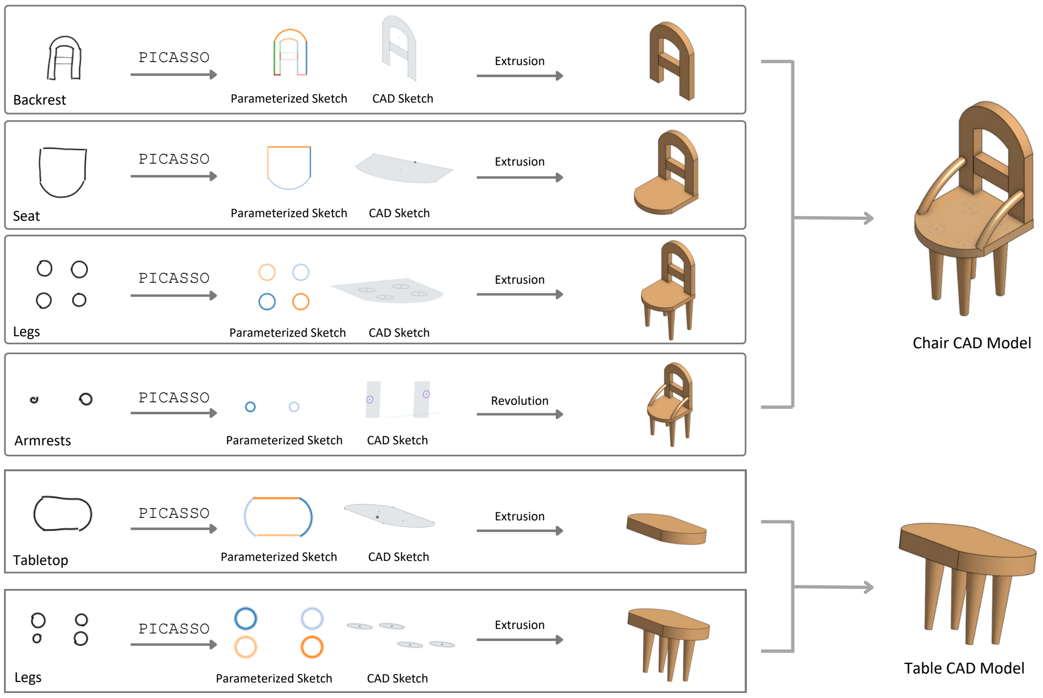

Figure 21. 3D CAD modeling from hand-drawn sketches with PICASSO.