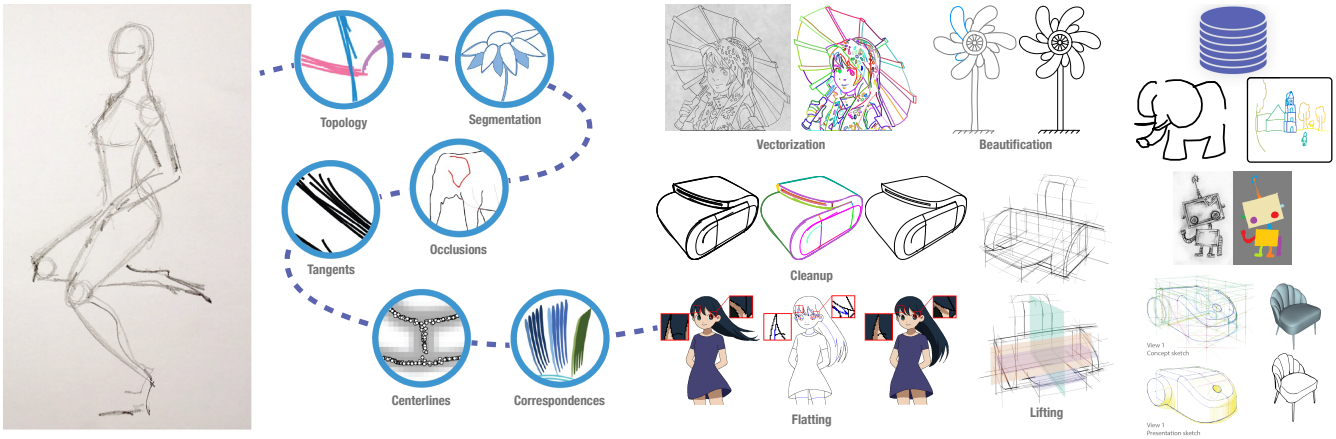
EUROGRAPHICS '0x
N.N. and N.N.
(Editors)

COMPUTER GRAPHICS *forum*
Volume 0 (200x), Number 0
STAR – State of The Art Report

State-of-the-art Report in Sketch Processing

Chenxi Liu[1,*], Mikhail Bessmeltsev[2,*]

[1]University of Toronto, Toronto, Canada [2]Université de Montréal, Montréal, Canada
*Joint first authors



(a) Sketch fundamentals (b) Core challenges (c) State-of-the-art methods (d) Sketch datasets

Figure 1: *In this survey of sketch processing methods, we first summarize key properties and fundamentals in sketch structure and perception (a, Sec. 2). We then identify core geometrical and topological challenges shared by many processing methods, as well as downstream applications (b, Sec. 3). Building upon that analysis, we then survey sketch processing methods for most popular sketch processing tasks (c, Sec. 4) and outline the commonly used sketch datasets (d, Sec. 5). Images in (b) are adapted from [LRS18, MNB23, DSC*20, NHS*13, WNS*10]. See later figures for other credits.*

Abstract

Sketches are a powerful and natural form of communication and are used in numerous systems for modelling, animation, shape retrieval, and editing. Despite their popularity, rough *sketches — whether raster or vector, 2D or 3D — are often too complex and imprecise to be used directly and thus need special processing. For instance, many downstream applications, such as shape reconstruction, have strict requirements for cleanliness and accuracy of the input sketch. Alternatively, if a drawing is the final result, users might want to further process the sketch through tasks such as vectorization, beautification, cleanup, flat colorization, and more. In this state-of-the-art report, we identify core geometrical and topological challenges shared by many processing methods, such as identifying endpoints, strokes, and junctions. Building upon that analysis, we then survey sketch processing methods in each task category. Furthermore, we outline the commonly used sketch datasets and promising avenues for future research in sketch processing.*

CCS Concepts

• ***Computing methodologies*** → ***Parametric curve and surface models; Shape analysis; Perception;***

1. Introduction

Sketches are a powerful form of communication, effectively used for millennia. They are intuitive and expressive, often capturing

complex ideas and emotions with simple strokes and shapes. Despite their apparent simplicity, sketches can serve as a surprisingly effective means to convey a shape. Whether used for brainstorming, storytelling, or art, sketches provide a unique and effective way to communicate ideas that transcend language and cultural barriers. Sketches are used as an input in many downstream tasks, such as 3D modelling [DSC*20], character posing and animation [BB22, GCR13], 3D shape retrieval [ERB*12], editing [GRYF21], as well as an art form on their own.

Many of those systems have restrictions on the input sketches: some systems only accept clean vector input [DSC*20], many require the input to be bounded by a closed curve [JBPS11], some rely on junctions being precise [PMKB23]. Similarly, systems that treat the sketch as an art form by itself and perform beautification [FASS16], colorization [YCY*22], or inbetweening [WNS*10], often rely on sketch processing, such as finding closed regions or determining junctions. While in general deep learning–based methods have weaker restrictions on the input sketches, many still process sketches to prepare their datasets [PNCB21, YLA*24].

Therefore, like other geometry representations, such as meshes, signed distance functions, or point clouds, sketches often need processing for these downstream tasks. Similarly to those other representations, sketches often have noise and distortions of a particular structure, which may require denoising (*cleanup*, *smoothing*) or distortion correction (*beautification*). As a sparse representation with little connectivity, sketches are reminiscent of point clouds and thus can require reconstruction that fits curves (*vectorization* for raster sketches, *consolidation* for vector sketches) or determining topology (*junction detection*, *closed region detection*).

Compared to standard geometry representations, such as images and meshes, however, processing sketches is rather difficult and less explored; the challenges in sketch processing are often unique. For instance, computer vision methods often rely on the precise and consistent nature of photographs, which follow standard assumptions about perspective, lighting, and texture. In contrast, sketches are often sparse, imprecise, distort perspective, and exhibit significant variations in style and level of abstraction. These inherent imprecisions and stylistic diversities mean that the techniques designed for photographs may fail when applied to sketches. All in all, sketch processing needs a distinct set of approaches that can accommodate unique characteristics of sketches, such as their imprecision, the variability in line thickness, incompleteness, and the absence of reliable colour information.

1.1. Scope

In this survey, by *sketch processing* we mean tasks involving sketches as input **and** as output, possibly including extra information in either input or output, such as timestamps for input or junctions and regions detected in the input sketches, or depth order. We do not include methods that process or create a completely different object, like a text, a 3D surface or an animation. This criterion excludes methods that, for instance, generate sketches from text prompts, model 3D shapes from sketches, or perform inbetweening. While some sketches contain handwriting, processing handwriting is out of our scope. Sketches that include shading or hatching are within our scope. We cover both 2D and 3D sketches, with an emphasis on 2D sketches. For more extensive discussions on 3D sketching, we refer readers to a recent book [AMW*23] and a doctoral thesis [Yu23].

Among the related surveys, the work by [OPP*21] focuses on the methods of 2D curve reconstruction from point clouds, including closed, open, and non-manifold 2D curve structures. Since their input is a point cloud, their survey is complementary to ours. The survey on sketch-based content creation [BC20] focuses on modelling shapes via sketches. Similarly, those methods are outside our scope, as their output is a 3D model and not a sketch. The most recent related survey is about deep learning methods on sketch data by Xu et al. [XHY*22]. Our scopes have very little overlap: they focus on deep learning–based methods with sketches as input or output; we focus on sketch processing methods specifically, regardless of the technique they use. Consequently, most of the methods they discuss are outside our scope, such as sketch-based retrieval, recognition, generation, and others.

1.2. Overview

We focus on three main topics: fundamentals, core challenges, and state-of-the-art methods for sketch processing. We begin by outlining the key characteristics of sketches that distinguish them from more common geometry representations; we then summarize the research into the structure of sketches, introducing their main components and their perceptual significance (Sec. 2). Next, we discuss core challenges in sketch processing that are typical for downstream tasks; progress in any of those core challenges can simplify or make downstream methods more robust (Sec. 3). We then examine various methods for sketch processing, including vectorization, beautification, cleanup, and other methods (Sec. 4). Additionally, we provide a brief overview of datasets useful for sketch processing (Sec. 5). Through this comprehensive review, we aim to provide a thorough understanding of the state-of-the-art in sketch processing and identify promising directions for future research (Sec. 6).

2. Fundamentals

In this section, we first describe key characteristics that distinguish sketches from more common geometry representations, such as meshes, point clouds, or photographs. We then discuss the role of human perception of sketches in both creation and interpretation of these sketches. These observations form a common ground for many sketch processing algorithms, as well as many downstream applications of sketches.

2.1. Sketch Structure

Digital sketches are created in different media and with a plethora of techniques (Fig. 2). Some are created digitally using a tablet (Fig. 2bc), which generally results in cleaner 2D or 3D images, often easier to process. Others are hand-drawn with pencil or pens (Fig. 2a), incorporating both lines and fills, and then scanned or photographed, leading to noise, glares, uneven lighting, resulting in additional processing challenges. The sketching techniques used can also vary significantly; some artists prefer clean curves, while

Figure 2: *Traditional pen-on-paper sketches can be digitized via scanning (a). Digital sketches can be created directly on a tablet, either on a 2D canvas (b) or within a 3D environment displayed on a 2D screen (c). Recently, 3D sketches have been created with handheld 6-DoF tracked controllers in VR/AR environments, where they are also displayed immersively (d). Photo (a) © Spencer Nugent; (b) © David Revoy under CC BY 4.0; (c) © Evi Meyer under CC BY-NC-ND 4.0; (d) © Elinor Palomares.*

others use overdrawn strokes to better capture shapes or indicate thickness. Sketches can either be filled or composed solely of strokes — those are known as *line drawings*. In 3D, sketches, often created in VR/AR (Virtual and Augmented Reality) environments with handheld 6-DoF tracked controllers (Fig. 2d), are typically imprecise 3D line drawings. The distinction between clean and rough sketches is approximate but crucial for certain methods, as some algorithms may assume clean continuous strokes drawn on an empty background and fail when this assumption is not met.

The two most popular representations for 2D sketches are raster images, composed of pixels, and vector images, composed of geometric entities like strokes and fills. Artists often prefer drawing in raster applications or on paper due to the freedom it provides, along with the richness and ease of use of editing tools. Vector graphics, frequently utilized in animation and various design forms, offer the advantages of infinite resolution and relative ease of deformation and animation. Some topological information, such as face-edge adjacency, is absent in standard vector graphics, but can be added by using alternative data structures [DRvdP14,DRvdP15]. Unlike raster images, vector graphics contain direct geometric information, making them highly suitable for geometric processing. The conversion from vector to raster, known as rasterization, is a well-defined process that can be made differentiable [LLMRK20]. As different sketches can be rasterized into the same image, the inverse problem, vectorization, is not well-defined and presents a significant challenge (Sec. 4.1).

Sketches, whether in raster or vector format, in 2D or 3D within VR/AR, typically have *strokes* as their main primitives; each stroke is a single trajectory of a pen. Depending on the medium and the tool used, strokes can have thickness and some texture, both of which can vary along the stroke. A stroke's key properties are its *centerline*, *endpoints*, and *junctions*. In raster format, strokes and their endpoints are depicted but not explicitly stored; determining them is one of the goals of line drawing vectorization. In most representations, raster or vector, junctions are not explicitly stored, but are perceived by humans and are critical for interpretation. In VR/AR, a stroke is often rendered as a tube or a ribbon. To support tube rendering, in addition to sampled 3D positions, a stroke is assigned a volume, typically created through extrusion. [AMW*23, Chapter 7]. Sketches may contain extra elements, such as fills, time stamps, pen pressure information, or orientation in 3D, among others.

Sketch strokes can be closed or open; strokes may intersect each other and have self-intersections. Open strokes have endpoints that can form different junctions, including T-junctions, where one stroke terminates at another; Y-junctions, where two curves merge smoothly, visually forming a single curve. The points where two strokes intersect and do not merge, or when a stroke has a self-intersection, are called interchangeably intersection points, or X-junctions. The vast majority of junctions typically have valences 3 or 4 and are one of those types. High-valence junctions are also possible, but are especially prevalent in technical illustrations. For both vector and raster images, junction information is typically not stored explicitly and thus needs to be extracted if required (Sec. 3.1). T-junctions in particular are used in 3D reconstruction, as they are often created due to an occlusion: top T's stroke is typically occluding the stem stroke, and thus a junction indicates a local depth order [NM90]. Note that T-junctions can also form when projecting 3D shape's sharp corners [VMS05] or may even be textural, meaning they do not necessarily indicate occlusions. Junctions have numerous other uses, including for detecting closed regions (Sec. 3.4).

2.2. Key characteristics of sketches

Sketches hold a unique position among geometric representations due to their distinct characteristics that set them apart from more traditional representations such as photographs, meshes, points clouds, or implicits.

One of the key sketch characteristics is the presence of broken and overdrawn strokes (Fig. 3, top). Unlike the continuous and consistent lines found in photographs, sketches often have strokes that are interrupted or repeatedly drawn. Those overdrawn strokes might be simple imprecisions, might be stylistic choices, or might be a means to express stroke width (Fig. 3, middle) [LRS18,LABS23]. This property poses challenges both for vectorization of sketches (Sec. 4.1), as well as for downstream processing task that could benefit from cleaner curves [PMKB23]. These chal-

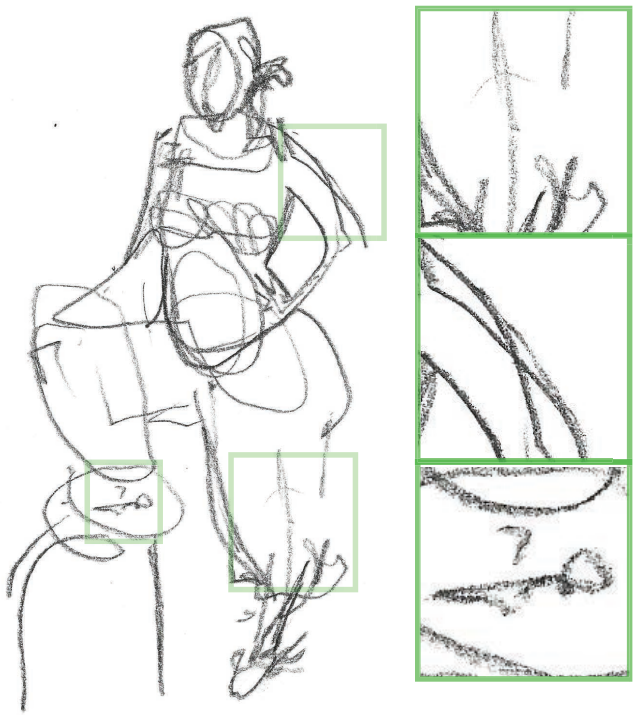


Figure 3: *Sketches have a few unique characteristics, affecting downstream processing. Sometimes a stroke, perceived as one, is broken into a few 'interrupted' parts (zoom, top). Often artists use overdrawing, repeating similar strokes a few times, whether to better capture the imagined stroke shape, to express the stroke width, or as a stylistic choice (zoom, middle). Some strokes have unclear interpretation, or semantics (zoom, bottom). Typically, many proportions are distorted (compare hip and forearm widths, for example). Sketch © Olga Posukh.*

lenges motivate sketch cleanup methods specifically designed to tackle overdrawn strokes either in raster or vector (Sec. 4.3).

This trait can complicate the interpretation and analysis of sketches, as the viewer or algorithm must decipher the intended form from these fragmented or overdrawn strokes. Moreover, strokes in sketches may have unclear interpretation (Fig. 3, bottom), where even human annotators would disagree on the precise function of the stroke, further challenging their processing [YVG20]. Unlike the detailed textures and shading of photographs, sketches rely on minimalistic and sometimes ambiguous lines to convey complex forms and ideas, demanding a higher level of abstraction and interpretation.

Another notable characteristic of sketches is their distorted shapes and proportions (Fig. 3, compare hip and forearm widths). Unlike the accurate proportions captured in photographs, sketches routinely exaggerate or simplify certain elements. In general, sketches often distort even simple shapes such as straight lines or ellipses [SKKS09]. Most corners and junctions in sketches are drawn with a level of imprecision, frequently leaving small gaps or producing curves that fail to intersect at the intended points. These gaps and imprecise intersections may pose significant challenges for sketch processing or reconstruction methods [CSSaJ05]. Techniques that depend on accurate depictions of shapes and precise junctions become brittle when applied to sketches, as they struggle to interpret and reconstruct the intended forms from these imprecise and variable line drawings [BVS16]. This inherent variability necessitates the development of robust algorithms capable of handling the nuances of sketches, ensuring accurate analysis and reconstruction despite the lack of precision.

Even with no intentional distortion of shapes, the overall projection of a 2D sketch often does not adhere to the precise projection models, such as perspective or orthographic projections. Instead, sketches might present a nonlinear or even mathematically implausible projection [Sin02], where spatial relationships are altered to emphasize certain features or to fit the artist's stylistic choices. Humans, including professional artists, often struggle to accurately depict projections, particularly perspective projection [SKKS09]. This inconsistency is further compounded by the fact that many artists deliberately avoid perfect single-viewpoint projections, opting instead for multiperspective or nonlinear perspective to convey more information or achieve a specific artistic effect. Consequently, methods that rely on the assumption that a sketch is a precise projection of a 3D object can result in incorrect 3D reconstructions [BB22]. The variability and intentional deviations in artistic sketches mean that these methods must be adapted to handle the complexities and nuances of human-drawn projections, ensuring they can still produce accurate and meaningful 3D interpretations despite these challenges. These deviations from photographic norms pose significant challenges for standard computer vision techniques, which rely on projection models and overall precision of the depiction.

2.3. Sketches and Perception

Despite their inherent inaccuracies, sketches in general, and even line drawings with just strokes and no shading, remain easily and consistently understandable for humans [Her21]. Some fMRI studies suggest that the same areas in our brain activate when viewing both natural photographs and line drawings [WCC*11]. This may explain why we perceive and interpret sketches as readily as we do more detailed images, perhaps assuming that a sketch is a realistic scene under specific lighting with specific materials [Her20]. Furthermore, Hertzmann's hypothesis [Her24] posits that humans may not form a consistent 3D model while exploring the world. If this is the case, the inconsistencies in sketches do not significantly hinder our ability to understand the depicted shapes. Our brains are adept at filling in gaps and interpreting abstract representations, making sketches a powerful and efficient means of communication and visualization despite their lack of rich geometric information and precision.

The strokes that people draw often relate to real 3D objects that people see or imagine. Experiments have shown that when individuals are asked to draw a 3D model rendered from a specific viewpoint, the strokes can be mapped back, or registered, to the model [CGL*08]. The larger challenge, however, is deciphering how exactly sketch strokes relate to the 3D geometry.

These studies indicate that people primarily draw *occluding con-*

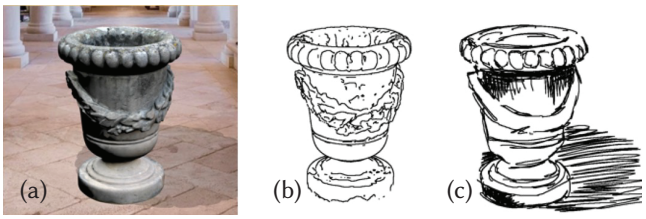


Figure 4: *Despite decades of progress in non-photorealistic rendering (NPR) and analysis of sketches, state-of-the-art NPR renderings (b) exhibit significant domain gap from real sketches (c). This limits uses of synthetic sketches as training data. (a) Target 3D model. Figure from [WQF*21].*

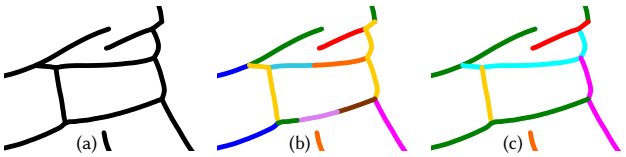


Figure 5: *Given a stroke union (a), the problem of segmenting it into separate strokes is formally ill-posed and inherently perceptual. Often algorithms would unnecessarily split strokes or incorrectly connect them across junctions, making the final vectorization hard to edit (b). Ground truth (c).*

tours, which delineate depth discontinuities when parts of the object occlude itself or other objects (Fig. 11). For smooth 3D shapes, occluding contours are often defined as points where the normal is perpendicular to the gaze direction. Occluding contours, however, can only convey limited geometric information, so an extension of this concept is *suggestive contours*, which can be loosely defined as points that are occluding contours for nearby views [DFRS03]. Another set of curves that can explain what artists depict is ridges and valleys [OBS04] and apparent ridges [JDA07]. Ridges and valleys include sharp edges and in general local maxima and minima of curvature. Apparent ridges and valleys are an extension of the formulation that first projects the curvature onto the view plane first and then finds the maxima. In certain drawings, particularly in industrial design, artists often depict cross-sections, which correspond to the principal curvature lines of an object [XCS*14]. These lines are orthogonal in 3D space, but may not be orthogonal on the 2D projection, providing us with a strong cue of the 3D shape. This cue aids humans in comprehending the 3D shape, and it can be utilized in reconstructing the 3D form [GHL*20]. In design sketches, another kind of strokes are *construction lines* that play a dual role: They help the designers more accurately depict proportions, symmetry, and perspective, and help viewers to interpret the shapes. In particular, designers often depict perspective axes with a horizon line and vanishing points, and sometimes draw scaffolds like bounding boxes or other primitive shapes [GSH*19, ES08]. Note that some of these studies show reference images to participants [CGL*08, BSM*13], which may bias the choice of strokes to draw, as even designers tend to copy the strokes instead of drawing from their imagination [GSH*19].

Unfortunately, many strokes cannot be modelled using these geometric frameworks. Recent studies indicate that only approximately 60% of drawn strokes can be explained by these models [WQF*21]. The remaining strokes, which include 'textural' and 'shading' strokes, lack robust geometric models for accurate representation. Shading elements, in particular, give an illusion of a 3D shape, but using them to infer a shape, in the style of shape from shading works in computer vision [ZTCS99], seems to be challenging [KHW*22].

As a result, there is a significant domain gap between synthetic sketches, non-photorealistic renderings (NPR) using these stroke models, and real sketches (Fig. 4). This domain gap is one of the central challenges preventing efficient generalization of models trained on such synthetic data to real sketches [ZGZS20].

2.4. Sketch Structure and Perception

Depending on the application, there are two ways of treating a sketch: as is, where each stroke is significant and needs to be preserved, or as an approximate depiction of a perceived shape. For instance, a shading stroke or a group of strokes might be interpreted as strokes on their own or a depiction of a solid fill (Fig. 4c, shadow). This subtle viewpoint difference is critical to the choices of a processing method. For instance, the goals of sketch beautification (Sec. 4.2) and cleanup (Sec. 4.3) can be stated as inferring the perceived shape. In contrast, such tasks as sketch deformation, animation, and parameterization often treat the sketch as is, where each stroke is preserved. In vectorization, this difference of viewpoint leads to two general families of methods, for *clean vectorization* that attempts to vectorize each stroke separately, true to the image (Sec. 4.1.1), and *rough vectorization* that simplifies the final drawing (Sec. 4.1.2).

In both those interpretations, from a geometrical standpoint, the union of all the strokes in a line drawing is a nonmanifold 1D structure: While each stroke is often a 1-manifold, junctions and intersections make the overall shape non-manifold. Note that even a single self-intersecting stroke is already a non-manifold. If there are open strokes, the full object becomes a non-manifold structure with boundary. This implies that in general, processing must assume a non-manifold structure. This is one of the differences between line drawing vectorization and surface reconstruction — despite superficial similarities, the latter typically focuses on reconstructing manifold surfaces.

If the overall non-manifold structure, i.e., the union of all the strokes, is known, decomposing it back into individual strokes is an ill-posed and, again, perceptual problem, as it requires preserving continuous strokes away from junctions and identifying continuations around every junction. This is one of the key steps in some vectorization pipelines [NHS*13], as its success defines whether the final vectorization will be intuitive to edit or not. In an extreme case, the final result may look correct, but contain strokes that are not useful for editing (Fig. 5).

Focusing on the perceived shape, multiple strokes can represent a single curve overdrawn several times, forming a *stroke cluster* [ES08, LRS18]. Clusters can indicate that the artist attempted to

capture the shape through multiple attempts or to depict the width/-texture of a stroke. Even in vector drawings, clusters require additional processing to determine the exact locations of endpoints, junctions, and the centerline. When interpreted as depictions of stroke width, such clusters taken together form a *foreground* of a 2D or 3D sketch [MNB23], akin to fills.

Such decision of which strokes are perceived as filled regions is a key to distinguishing the perceived shape, or foreground, from the empty space, or the background — a non-trivial perception-based task. For instance, in certain areas, it is unclear whether an object has a hole (indicating background) or if it is simply incomplete and assumed to be part of the shape. This distinction is crucial for applications that rely on understanding the genus of the shape or its boundary. In 3D sketches, such shading strokes may indicate a surface and are often used to improve 3D sketch readability, as the shaded surfaces occlude the objects in the background.

Similarly to how gaps may be perceived as part of the foreground, some gaps between separate strokes can be perceived as parts of a single continuous stroke (Fig. 6, right). For example, what formally appear as two separate strokes may be intended to represent a single stroke, with the artist having chosen to split it into two segments [KH06]. For purposes such as editing, reconstruction, and correspondence, these should be treated as a single stroke as viewers tend to interpret those as connected [HF99]. These perception effects are studied in psychological research, including Gestalt theory, which addresses such phenomena [Kof55, Wer38]. In Gestalt theory, these two strokes are said to exhibit 'good continuation'. Similarly, junctions that are not precisely drawn but are perceived by the viewer also fall under this perceptual framework (Fig. 6, left).

Considering the 3D shape that a 2D drawing represents, T-junctions play a significant role. For example, a T-junction indicates that one curve is "in front" and another is "behind", providing crucial depth information and establishing depth order at that point [Wil94]. Specifically, a T-junction can suggest that one part of the drawing occludes another. However, not all T-junctions convey this meaning. For instance, in areas with hatching, T-junctions often do not indicate depth relationships but are instead part of the texture or shading.

In 2D, one of the key elements of a sketch are perceived regions [YLL*22]: delineated by a few strokes, with perhaps sparsely placed strokes inside, those are routinely coloured in *flatting* (Sec. 4.4), a part of comic strips production, automatic colorization, as well as search for correspondences [ZCZ*09]. A direct equivalent of this in 3D are loops that can denote boundaries of a depicted surface patch; for clean 3D sketches containing no open curves these can be directly identified, for rough 3D sketches the problem is significantly more challenging [YAB*22].

In both 2D and 3D sketches, the empty space in line drawings or other sketches with little shading is also a key element in determining the perceived shape. Generally, this empty area may be outside the intended shape, i.e., depicting a hole or background, or inside. Within the empty spaces inside the shape, the shape is either smooth or contains details that the artist chose to omit [Gup22]. In many cases, the absence of strokes indicates a smooth, uninterrupted surface — yet relying on this assumption, especially important in the context of reconstruction, often leads to overly smooth shapes (Fig. 7). Alternatively, the empty space could imply that certain intricate details were intentionally left out for simplicity or clarity, leaving the viewer to infer these aspects.

Interpreting sketches involves not only spatial information but also temporal data, such as time stamps and drawing order. Temporal data contains a wealth of information, ranging from insights into the artistic drawing process [WQF*21] to the perception of sketches [LABS23]. A group of methods focuses on reconstructing drawing order from a complete vector sketch. The recovered drawing order can be used to generate drawing animations [FZLM11, LFT14] or even easy-to-follow tutorials for industrial design sketches [HLW*16]. Qiu et al. [QWM*23] explore the question, "Is drawing order important?" Their findings include: stroke order significantly impacts the perceived naturalness of a drawing process; multiple orderings of the same set of strokes can be perceived as human-drawn; relative local ordering matters more than global ordering; and descending length ordering alone is insufficient for naturalness. They also evaluate an existing method [FZLM11] and find that it performs well for object drawings but similarly to random ordering for scene drawings.

3. Core Challenges of Sketch Processing

In this section, we focus on the core challenges in sketch processing that are common across various sketch-based systems. These challenges include determining sketch topology, detecting and disambiguating intersections, junctions, and occlusions, segmenting background from the foreground, robustly defining tangents, finding centerlines and correspondences. Depending on a system, these challenges might form a part of preprocessing, or be at the core of the system. For instance, typical sketch-based modelling systems often need to robustly find junctions as a preprocessing [PMKB23], while vectorization systems have centerline computation at their very core [NHS*13]. These challenges are typical for both bitmap and vector sketches, and, except for the occlusions that are a 2D-specific phenomenon, are applicable to both 2D and 3D sketches. These seemingly harmless preprocessing steps can be a major source of system robustness issues, as well as inherent causes of artifacts in the final results. Progress in these core challenges has direct implications on the quality and robustness of downstream systems and applications.

We will start with the most basic challenges, sketch topology, junctions, and endpoints (Sec. 3.1), tangents (Sec. 3.2), and centerlines (Sec. 3.3), then move on to more complex issues like segmentation (Sec. 3.4), occlusions (Sec. 3.5), and finally finding correspondences between sketches (Sec. 3.6).

3.1. Sketch Topology, Junctions, and Endpoints

One of the most basic challenges in sketch analysis is determining the topology of the sketch. One of the first steps to defining sketch topology is identifying the locations of *stroke key points*: endpoints, junctions, and sharp corners. Once junctions are found, topology analysis requires determining junction types, discerning whether intersections or junctions are intentional or coincidental, and whether certain lines form closed regions (Fig. 6).

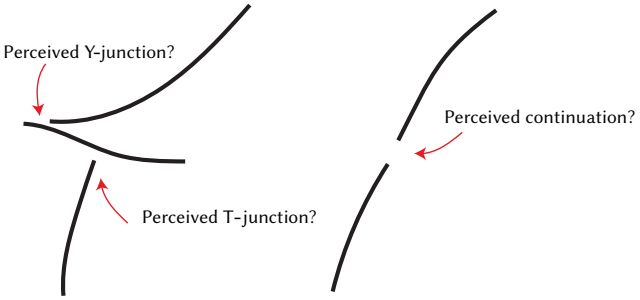


Figure 6: *Depending on the local scale and context, these strokes might be perceived as disconnected strokes or junctions (left). Similarly, two separate strokes might be interpreted as one continuous stroke (right).*

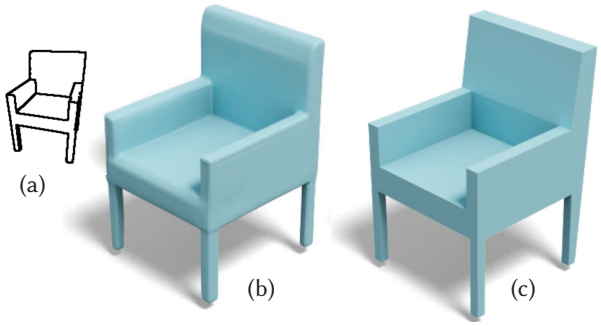


Figure 7: *While white space often suggests a smooth surface, often artists decide to omit details or simplify the shape. Relying on smoothness in the white space (b) might result in missing some geometric details, compared to the ground truth (c). Image adapted from [ZPW*23].*

Modern vectorization systems often begin by detecting these key points and then connecting them with strokes [PNCB21, YLA*24]; it is similarly a critical decision in vector sketch consolidation and cleanup [LABS23], 3D reconstruction [GHL*20], and other tasks.

In computer vision, corner detection has been a part of a standard pipeline long before the deep learning era [Sze10], including the industry standard Harris detector [HS88]. Early sketch-specific approaches assume a clean drawing and either assume all junctions are precise [LB90] or rely on simple thresholds on distances and/or angles between strokes [CKX*08] or simple priors like smoothness and low valence, as noted by [NHS*13]. For noisy sketches, those methods have been largely superseded by deep-learning-based approaches. While it is possible to train a modern keypoint detector for this task, even with relatively high accuracy, these detectors can still frequently fail [YLA*24]. Relying entirely on their predictions is risky and can lead to significant errors in the downstream tasks.

One of the most significant challenges in algorithmically understanding sketches is determining whether an element is a bug or a feature. For instance, the decision of whether a gap is intentional or it is a T-junction drawn sloppily can greatly impact the interpretation of the sketch, including local depth order. What complicates matters further is that, while the artist can easily discern whether an intersection or a corner is intentional, other observers may be inconsistent in their interpretations. This inconsistency makes collecting reliable datasets for training particularly challenging, making inferring *perceived* junctions or intersections (see discussion in Sec. 2.4) particularly challenging. One of the earlier approaches completes perceptually closed curves using observations from Gestalt psychology and formulating it as a graph search [Sau03]. A recent approach by Yin et al. is leveraging classical machine learning approaches like random forests, which may be preferred in scenarios with limited data, as deep learning models typically require large datasets [YLL*22].

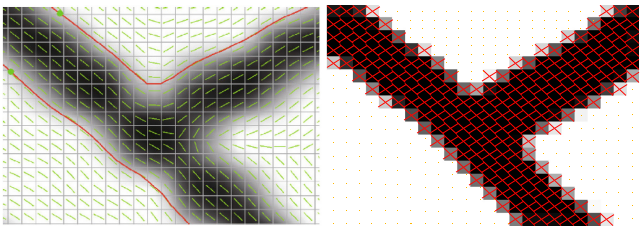
For sketches, either raster or vector, that are composed of clusters, a related challenge is determining which strokes are perceived as grouped and thus form clusters. From a topological standpoint, the strokes within a cluster form a neighbourhood, so many operations, such as editing, animation, or search for correspondences is often done for the cluster as a whole [LABS23]. The main perceptual observations, starting from early works [Ros94] are that strokes within a cluster are roughly parallel, close to each other relative to their length, and form a Gestalt good continuation. Most vector-based approaches implicitly or explicitly use these observations [BTS05, LABS23, OK11, LRS18]. We discuss those methods in detail in Sec. 4.3.

3.2. Tangents

Beyond determining the topology, the next basic question is geometrical — determining stroke tangents. Determining the tangent at a point in a sketch can be quite challenging, and yet is needed for many applications including vectorization (Sec. 4.1), 3D reconstruction, and editing. Traditional computer vision methods compute the image gradient for a raster image, yielding 2D normals that can be rotated by 90 degrees to obtain the tangent. Even for clean sketches, however, gradient is meaningful only at the edges of strokes: inside a stroke and away from the strokes gradient only highlights noise, so it is often filtered using simple magnitude thresholds.

In general, this traditional approach fails around endpoints and junctions and performs poorly in noisy images [NHS*13]. In vector drawings with texture strokes or clusters, the difficulty persists despite the ability to compute the tangent of a single stroke due to overdrawn strokes [BCF*07]. The distinction between shape and stroke texture is particularly tricky, as humans can easily interpret color variations as texture rather than shape, even when color intensity is noisy.

To suppress noise in the gradient estimation, the typical approach is smoothing. Smoothing gradients, however, requires some care, as naïve approaches like Gaussian blur normally fail: gradient estimation produces a vector field that points towards the stroke. Therefore, the orientations of vectors on different sides of the same stroke will be opposite and thus cannot be smoothed out directly. Instead, smoothing is typically done in an orientation-independent representation, such as structure tensor or similar matrices [CGBG13, CLMP15] (Fig. 8a). An alternative is to use a natural association between vectors and complex numbers, and perform smoothing of squares of those complex numbers [DVPSH14].



(a) Noisy raw image gradients (b) Smooth frame field

Figure 8: *Compared to the noisy raw image gradients computed by Chen et al. [CGBG13] (a), the smooth frame field constructed using Bessmeltsev and Solomon's approach [BS19] enables the formation of more accurate junctions (b). Figure adapted from [BS19].*

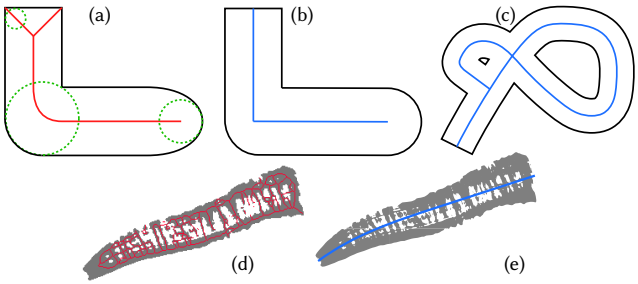


Figure 9: *Stroke centerline (b, blue) is a single curve that can be close to the medial axis (a, red), but does not have Y- or T-junctions except, perhaps at endpoints (c, blue). Centerlines, however, can contain self-intersections (c). Note that in (b) the brush is varying in shape and width, from a disk to a square; caps for two endpoints are also different. Note also that while a medial axis is contained within the stroke (d), the centerline does not have to be (e). Brush taken from Graphicsfuel* `https://www.graphicsfuel.com/30-hand-drawn-brushes-for-illustrator/`

For line drawings, tangents can be captured more robustly using frame fields [BS19], which represent a non-orthogonal cross per pixel instead of a single vector like gradients (Fig. 8b). The only remaining challenge is determining which direction represents the tangent, which, away from field singularities, can be done via combing. While it is possible to eliminate these singularities, doing so makes finding such a frame field significantly more computationally expensive [GHB*23].

Some sketches, especially in industrial design, typically contain junctions of higher valence that cannot be fully captured by frame fields with only two directions. Even though the frame field approach can be generalized to this case, to our knowledge, it is so far not explored in the literature.

3.3. Stroke Centerlines

A related and equally fundamental question as finding stroke tangents is determining stroke centerlines. Finding stroke centerlines is a common operation in line drawing vectorization (Sec. 4.1), editing vector sketches, reconstruction, and animation. Overall, it is an ill-posed inverse problem, where the task is to reconstruct a path of an unknown, possible varying brush (possibly in both texture and width) that forms the target image. For a vector sketch that has been pre-segmented into clusters, the main problem is typically finding a centerline of a stroke cluster. Many methods directly apply off-the-shelf 2D point fitting [PS83, WPL06]. Early methods first order point samples on a stroke by projecting to the dominant axis [KS07], or by computing Laplacian spectral embedding [OK11], then fit a curve to fully ordered points. This strategy, however, often has issues with self-intersecting centerlines. StrokeAggregator [LRS18] utilizes a moving least squares–based fitting that integrates a tangent fitting term in addition to the position fitting term used by the earlier methods. StrokeStrip [PvMLV*21] orients strokes, explicitly compute a 1D parameterization using isolines as the basic element, and fit the final curve using a positional, a tangential and a curvature smoothing term.

In general, for bitmap images it is related to a classical computer vision problem of finding a medial axis, or skeletonization, often done via morphological thinning or distance transform [Sze10]. For some brushes, where stroke width is constant and the stroke has a solid color, stroke centerline can be close to a subset of a medial axis. Unlike medial axis, however, due to uneven brush texture and width along the stroke, centerline is not necessarily contained fully within a stroke (Fig. 9de). Furthermore, since stroke is drawn in a single continuous motion of a pen, stroke centerline does not have Y- or T-junctions except, possibly, at its ends, but might have self-intersections (X-junctions) (Fig. 9).

Earlier approaches, such as [HT06], followed these observations and adapted the computer vision skeletonization algorithms [dB94] to extract simple centerlines, like straight lines and circles. Later and more general approaches sometimes still use classical skeletonization [FLB16], but typically rely on some estimation of normals [NHS*13, BS19] or use deep learning [MSSG*21, YLA*24]. Noris et al.'s gradient descent-like approach [NHS*13] only targets clean, mostly digital sketches, as the gradient information is typically noisy for paper sketches. Favreau et al. [FLB16] use a classical morphological thinning approach for open curves and find centerlines of detected regions boundaries using image dilation. Using an estimation of normal, [BF12, BF23, BS19, PNCB21] compute centroids of normal cross-sections of a stroke. Stanko et al. [SBBB20] compute a parameterization of a sketch so that a subset of the parameterization isolines form the centerlines. A few works [MSSG*21, DYH*21] introduce a recurrent neural network (RNN) that produces centerlines, either trained in an unsupervised fashion with a differentiable rasterization loss or similar techniques [MSSG*21, LLLW22] or in a supervised manner [DYH*21]. Yan et al. [YLA*24] represent centerlines as zeros of an unsigned distance field that they learn.

3.4. Sketch Segmentation

Another core challenge for both raster and vector sketches is segmenting them. There are a few variants of segmentation problems for sketches: **object segmentation** that separates different objects, such as characters, and a more general **semantic segmentation**, **positive/negative space segmentation** segmenting regions perceived as foreground (e.g., filled) from the background, and **region segmentation** that finds closed regions.

For some downstream tasks, such as animation, one of the first steps of processing a complex sketch is segmenting it into single objects, i.e., **object segmentation**. For sketches of humans, current state of the art is applying an R-CNN, fine-tuned on character sketches, which outputs a bounding box of each character [SZL*23]. **Semantic segmentation** pursues a more general goal of segmenting a sketch into meaningful parts, necessary for fine-grained sketch analysis. Often focusing on 2D vector sketches, some works leverage domain knowledge to develop hand-crafted features, often leading to limited generalization [DL15, GKSS05]. Other early approaches rely on heavy user annotation [PBG*15, NSS*12]. Some of the more recent approaches combine domain knowledge with a data-driven methodology [HFL14, ST16]. In [HFL14], domain-specific observations, including continuity and parallelism of strokes, as well as connectedness of the overall sketch structure, are used to formulate a mixed-integer optimization problem yielding the segmentation. [ST16] use distance-based heuristics to create a graph capturing the sketch topology (see Sec. 3.1), which they use in a Conditional Random Field (CRF) model for segmentation. For a unary classifier, necessary for CRF, they use Fisher vectors. The more recent deep learning methods use a variety of architectures to tackle the task. For instance, [LFT18, LPS*19, QT19, SDBM17, WQLY18, ZXZ20, ZXS*23] use CNNs, typical approach in raster image segmentation, that ignore all the stroke connectivity and are thus unable to take advantage of strong within-stroke connectivity information. Other approaches [YZF*21, QGXS22, ZPD*24, WL24, ZXS*22] use more advanced architectures, including a GNN (Graph Neural Network) [YZF*21]. Qi et al. [QGXS22], similarly, use a GNN, segmenting a sketch by deforming a given template sketch. Zheng et al. [ZXS*22] additionally incorporates drawing order into a GNN. Wang et al. [WL24] use a combination of a CNN predicting a stroke-based distance field, and a segmentation Transformer. Most of these methods, however, are trained on simplistic doodle-type sketches (see discussion in Sec. 5) and do not generalize well to complex sketches typical for design and animation.

For **positive/negative space segmentation**, similarly to the computer vision problem of segmenting background from the objects in the foreground in a photograph or a video, 2D sketches need non-trivial processing to separate sketched objects from the background. In its trivial form, this requires distinguishing stroke pixels (usually dark) from background (often white), also called *binarization*, which is often done by edge detection filtering [NHS*13, CLMP15, DCP17, DCP19] or image intensity thresholding [BF12, BS19, PNCB21]; both strategies are prone to errors. In a more general setup, the key challenge here is that while filled objects are often perceived as foreground, line drawings with no fill also contain foreground objects. For raster sketches, this segmentation task does not seem to be compatible with the standard computer vision approaches, as their segmentation pipelines rely heavily on texture and color, which for sketches are unreliable cues: Line drawings contain little to no texture, and even shaded sketches often contain gaps [SZL*23]. Instead, [SZL*23] resort to traditional yet robust computer vision techniques, performing segmentation via a combination of dilation and flood fill. A recent work [BB24] instead trains a U-Net predicting heatmaps of the foreground, reporting an IoU (Intersection over Union) of 0.65, highlighting that this problem is still largely open.

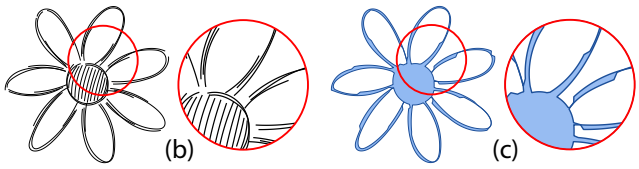


Figure 10: *For a sketch with overdrawn strokes, gaps, and uneven fills (left), outlining foreground ('positive space') from background ('negative space') requires non-trivial decisions on which strokes are grouped and which ones are separate (right). Adapted from [MNB23].*

For vector sketches, even though, unlike raster, they do not contain background textures like paper, and formally all the strokes form the foreground, determining the closed region tightly containing the strokes is non-trivial. This is a requirement for many applications, including deformation and 3D modelling [JBPS11, SKC*14], which then often triangulate the interior of the domain. In some settings, a naive approach of simply closing each curve is enough [IMT99, DSC*20], although this assumes that a single object (e.g., a body part) is drawn via a single stroke — an assumption often broken for freehand drawings. In earlier works, this is done via simple thresholding, but this strategy can cause various artifacts when multiple stroke endpoints are close together. This problem has been explored from computational topology perspective by Kurlin [Kur14]. A recent work by Myronova et al. [MNB23] continues this computational topology tradition, inspired by Alpha Shapes, and introduces an algorithm driven by a parameter controlling the largest distance between endpoints that can be considered connected (Fig. 10). These approaches, however, does not attempt to reconstruct *perceived* foreground, so this remains an open problem.

For **region segmentation**, or separating an input sketch into closed regions despite potentially small gaps, is a typical step in beautification (Sec. 4.2), colorization (Sec. 4.4), and some vectorization algorithms (Sec. 4.1). One of the classical approaches is a trapped ball algorithm [ZCZ*09]. Intuitively, the algorithm defines a closed region as a space that can be covered by continuously moving a fixed-radius ball that cannot pass through gaps smaller than its size. Trapped Ball algorithm often leaves unlabelled narrow regions that are inaccessible to the moving ball. To address this, [ZCZ*09] grow the trapped ball regions, roughly following the classical region growing segmentation and Lloyd's algorithm [CSAD04, Llo82]. Liu et al. [LWH15] jointly determine stroke clustering and region segmentation by alternating between merging small regions and raw strokes, with dynamically adjusted thresholds. Parakkat et al. [PPM18] use a variant of the region growing segmentation algorithm on a Delaunay triangulation of the sketch samples, which is followed by a semi-automatic approach merging initial regions based on user input [PCS21]. A recent approach by Scrivener et al. [SCC24] uses generalized winding numbers, quantifying how closed a region is, combined with a classical k-means algorithm to segment a vector sketch. It is unclear whether

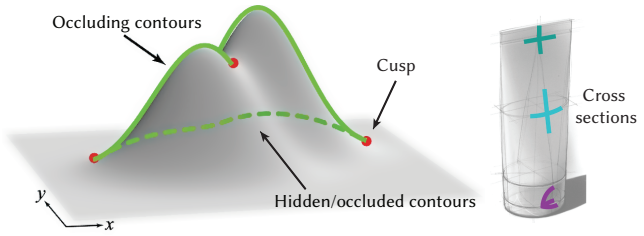


Figure 11: *Left: For a given smooth 3D shape and a camera, points on the surface that have normal perpendicular to the view direction, are categorized into visible (*occluding contours*) and invisible (*occluded, or hidden, contours*). Adapted from [CGL*08]. Right: in addition to occluding contours, one of the most important types of strokes routinely drawn in sketches of machine-made shapes are cross-sections, related to the lines of curvature. Adapted from [XCS*14].*

the regions identified by these methods are perceptually valid. Yin et al. [YLL*22] partially address that concern by detecting perceived junctions in a vector sketch.

A related problem is detecting closed loops in a 3D sketch, a necessary step in a few surface reconstruction pipelines from 3D sketches [PLS*15] (Sec. 4.4.2.1). For clean 3D sketches with precise junctions, this can be solved via a graph *routing system* [ZZCJ13a]. For rough 3D sketches, such closed loops might be detected as a by-product of deforming and segmenting a template surface [YAB*22], but in general it remains an open problem.

3.5. Occlusions

Another significant challenge is determining the location and geometry of hidden, or occluded, contours in 2D sketches [VMS05, CLT08, KS09]. This is a key challenge in both 3D reconstruction from sketches [DSC*20] as well as templates-based animation and deformation of sketches [BB24, DLKS18], as the geometry of the template is defined using occluded contours.

While it is possible to ask the user to draw these contours [BCHS20a, DSC*20], it can be difficult for humans to accurately imagine and depict them for many objects (Fig. 11) [KH06, KVDKT97]. Formally known to always have a topologically plausible solution [BBP09], finding the geometry of the hidden contours is still a challenging problem. Ullman [Ull76] outlines the properties of hidden contours, such as isotropy, smoothness, and curvature minimization, and suggests a network reconstruction them, very reminiscent of modern CNNs. Starting with the pioneering book by Kanizsa [Kan79], many works in the area leverage various ideas from Gestalt psychology, including Gestalt good continuation, as well as modern perception research to reconstruct the hidden contours. Earlier works [WJ97, Mum94] connect T-junctions, which usually span the hidden contours, via C_1-continuous *elastica* curves by considering random walks tangential to the stem of T from each side, and choose the maximum likelihood walk as the hidden contour. Karpenko et al. [KH06] extends their approach to contours containing cusps via discrete matching of T-junctions and endpoints, creating a hidden curve

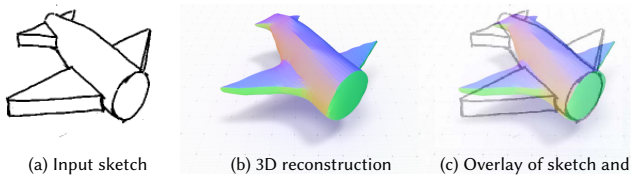


Figure 12: *3D reconstruction (b) based an input sketch (a) often struggle to stay close to the input (c). Adapted from [DCLB19].*

with a cusp between them (Fig. 11). Entem et al. [EPB*19] extend this approach to more complex scenarios by identifying and layering object parts and completing hidden contours by minimizing the total variation of curvature. For some applications, such as sketches of characters, one can reconstruct hidden contours using symmetry cues, including the symmetry of a surface around a skeleton [CS07] and the symmetries between different body parts [RTB*18, EBC*15].

An alternative to reconstructing occluded contours is performing a full 3D reconstruction directly, using some shape priors such as symmetry, smoothness, or data-driven priors [YSR*20]. Many reconstruction approaches, particularly learning-based, however, often have no control over which parts of the contour are visible and thus struggle to stay close to the input sketch (Fig. 12).

3.6. Correspondences

When dealing with multiple images, such as for automatic colorization, inbetweening, or sometimes 3D reconstruction from multiple views, one key problem is finding pointwise correspondences between similar sketches. This can be viewed as an instance of a shape correspondence problem in geometry processing [VKZHCO11, Sah20], with a satellite problem of non-rigid registration [DYDZ22], with a few caveats. First, a typical correspondence problem for sketches looks for partial correspondence due to the occlusions potentially changing between sketches. Second, measuring pointwise similarity is challenging, since the way the same area is depicted can vary significantly between sketches.

Borrowing techniques from video processing, a few methods attempt posing the problem of correspondences using the idea of optical flow [HS81], often focusing on densely sampled, very similar animation frames rather than separate sketches [SGX*23, HZH*22]. Being trained on high-FPS (frames per second) videos, these methods are only applicable to very similar drawings.

Generally, the problem is hard to solve directly, so many methods rely on some form of annotations, the vector structure, or identified regions. For instance, annotations may include sparse point correspondences [Ree81, ADN*17], guidelines [CMV17], or user corrections [MFXM21, YSC*18]. For instance, [ADN*17] use an iterative match-warp algorithm, along with additional user guidance, to form dense correspondences and interpolate between concept sketches. The vector structure of sketches can be also useful: if matching between strokes is known, obtaining pointwise correspondence becomes much easier [LCY*11, YBS*12]. For instance, if one assumes similar vector sketch topologies, the prob-

Table 1: *Specific sketch processing tasks have various input and output formats.*

Task	Input	Output	Section
Vectorization	Raster	Vector	4.1
Beautification	Vector Vector in 3D	Vector Vector in 3D	4.2
Cleanup	Raster or Vector	Raster or Vector	4.3
Flat Colorization	Raster or Vector Vector in 3D	2D region partition 3D cycles	4.4
Lifting	Vector	Vector in 3D	4.5

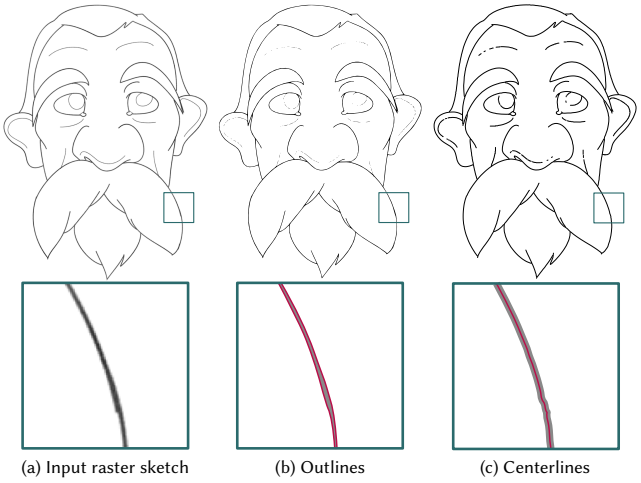


Figure 13: *Common vectorization software converts raster sketches (a) into stroke outline polygons (b) or stroke centerlines (c). In this example, Potrace [Sel03] was used for outline vectorization, and Adobe Illustrator's Image Trace [Ado] for centerline vectorization, both with default settings. Even simple, clean raster inputs pose challenges for accurate vectorization using common software with off-the-shelf configurations. Input image is from [NHS*13].*

lem of correspondences can be formulated via graph isomorphism [WNS*10]. In practice, this is a very restrictive requirement, and sketch connectivity can be very unreliable. Of course, identifying this stroke matching itself is a difficult discrete problem; a typical approach requires many-to-many matching with some form of shape descriptors [YBS*12]. Similarly, whenever regions are identified, finding correspondences between them is a difficult discrete matching problem. However, once it is known, pointwise correspondence becomes an easier task [ZLWH16]. Realistically, sketches often contain open curves in addition to closed regions, making this approach also limited in practice. In a spirit of non-rigid registration, Mo et al. [MGW24] predict a deformation of a given vector sketch onto an input bitmap, thus forming pointwise correspondences.

4. State-of-the-Art Methods

We organize sketch processing methods based on the specific tasks: vectorization, beautification, cleanup, flat colorization, and lifting

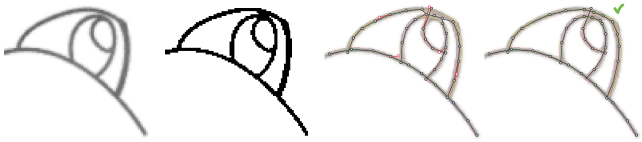


Figure 14: *Typical clean sketch vectorization methods begin by identifying content pixels (b) through binarizing the input raster sketch (a). These methods then construct an initial graph (c), which is refined into a final graph with a more accurate topology (d). The final graph is further up-sampled and optimized to capture finer stroke geometries. Figure adapted from [BS19].*

to 3D. The input and output formats of these tasks are summarized in Table 1. We focus on tasks involving 2D sketches and only discuss 3D sketch processing literature when it extends or adapts 2D approaches (see the later parts of Sec. 4.2, 4.4.2, 4.5). For further readings on 3D sketching, see Sec. 1.1.

4.1. Vectorization

Many sketches are stored in raster format when they are initially drawn on paper and later scanned or directly created using raster drawing software. To print the image in high resolution or use in downstream applications, such as colorization or animation, the sketch might need to be converted into a vector format, or *vectorized*. Compared to image vectorization targeting photographs, which converts similarly coloured areas of a photo into polygons (Fig. 13b), sketch vectorization outputs strokes that include endpoints, centerlines, and width (Fig. 13c). Sketch vectorization methods can be further divided into two categories: *clean* and *rough* sketch vectorization. The former assumes the input is clean and tries to preserve details as much as possible; the latter vectorizes and cleans up the input at the same time and is related to clean up methods (Sec. 4.3). The majority of sketch vectorization methods fall into the first category. The second category, i.e., rough sketch vectorization can be thought of as and compared with a combination of clean vectorization (Sec. 4.1.1) with subsequent cleanup (Sec. 4.3.2).

4.1.1. Clean sketch vectorization

One of the key stages in many clean vectorization methods is extracting a union of stroke centerlines, represented as an undirected graph, known as a 1-skeleton. Often this extraction is guided by aligning stroke centerline directions with raw image gradient. Most of these methods follow a common workflow in Fig. 14: (1) identify the content pixels, a process called binarization (Sec. 3.4); (2) construct an initial graph by skeletonization [NHS*13, DCP17, DCP19, CDQM18], creating curves aligned with local tangents [BF12, CLMP15, BS19, PNCB21, SBBB20, GHB*23] (Sec. 3.2, 3.3); (3) finalize by random sampling [HT06] or optimizing graph topology with several methods focusing on junction topology as well as optimizing the graph geometry [NHS*13, CLMP15, BS19, PNCB21, CDQM18, FLB16]. An early work [HT06] presents a system that vectorizes graphical parts of paper-based technical line

drawings into primitives of line segments and arcs with an analysis of algorithm robustness and complexity. The system eliminates detected text and separates input pixels into layers of even stroke thickness before vectorization. Noris et al. [NHS*13] additionally decide junction connectivity by using thresholds determined on a set of annotated sketches (Fig. 15). The methods by Donati et al. [DCP17, DCP19] extract curvilinear pixel structures using Pearson's correlation coefficient, skeletonize, and fit Bézier curves. Chen et al. [CDQM18] vectorize rasterized vector images where curves are relatively clean but can come in various thickness. They adapted a similar pipeline of thinning and refining the topology taking into account that the clean curve outlines can provide additional hints for topology reconstruction. Bessmeltsev and Solomon [BS19] improve the noisy raw image gradients (Fig. 8a) by solving for a smooth frame field (Fig. 8b) where one direction is aligned with the gradient. To tackle the challenge surrounding sketch tangents (Sec. 3.2), this formulation of two local directions allows forming of more accurate junctions, especially around T- and X-junctions. However, the precise reconstruction of sketch topology remains a challenge to this method — it may produce redundant parallel strokes; junction positions and connectivities are determined by error-prone heuristics. Puhachov et al. [PNCB21] use the same frame field and improve the topology reconstruction with the help of a keypoint extraction neural network that determines keypoint types (junction, sharp corners, and endpoints) and positions, as discussed in Sec. 3.1. After connecting keypoints with strokes, they optimize the stroke geometry to align to the frame field. In practice, relying on the keypoint predictor may cause this method to occasionally miss some parts of the line drawing when predictor fails. Gutan et al. [GHB*23] improve the frame field used in these methods by making it singularity-free, albeit at a considerably larger computational cost. The vectorization method by Bao et al. [BF23] follows the common workflow with coarse-to-fine curve network optimization to improve resulting junction connectivity and geometry fidelity.

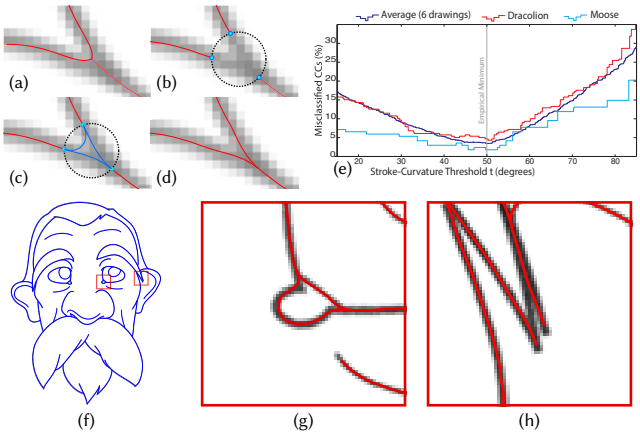


Figure 15: *Noris et al. [NHS*13] additionally focuses on the junction connectivity: their method identifies ambiguous junction regions (a), removes the corresponding curves (b), constructs candidate centerlines (c) and determines the final junction connectivity based on curvature thresholds determined by a set of annotated sketches (e). Given an input (f), the method may be inaccurate in fine details (g) and may extract incorrect topologies (h). Figure adapted from [NHS*13].*

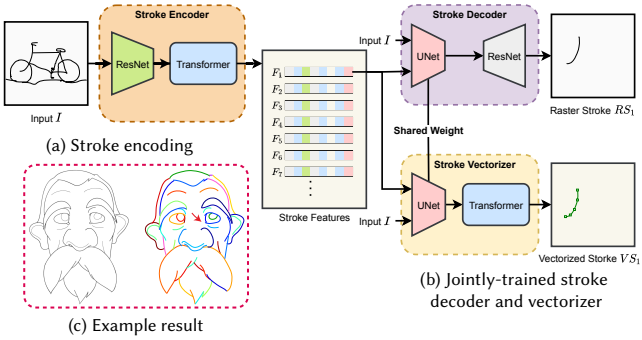


Figure 16: *Liu et al. [LLLW22] applies a global Transformer-based stroke encoder to the raster input (a). Each identified stroke is then passed to a raster stroke decoder and a Transformer-based stroke vectorizer with shared weights (b). This method produces relatively clean topologies but may miss fine details, as indicated by the red arrow (c). Figure adapted from [LLLW22].*

Some recent approaches are based on supervised or self-supervised learning [GZH*19, BCY*21, KWÖG18, EVA*20, YLA*24]. Guo et al. [GZH*19] extract sketch centerlines, using convolutional neural networks (CNN) similar to Smart Inker [SII18b], and a junction image. Then they subdivide the centerlines at junctions and reconstruct junction connectivity via a topology construction network. Their networks are trained on a fully synthetic dataset of rasterized vector curves, possibly leading to issues with generalization. Bhunia et al. [BCY*21] adapt a self-supervised training framework by posing the problem as cross-modal translation between vector and raster image spaces. Trained on vector and raster image pairs, their approach learns latent representations and corresponding raster and vector encoders and decoders. These encoders and decoders can be assembled with the shared latent representation to complete various tasks such as sketch recognition and vectorization. Kim et al. [KWÖG18] formulate the problem as a stroke segmentation in pixel space, which can then be converted into vector strokes. They adapt the classic graph cut segmentation setup by dropping the data term, duplicating the overlapping pixels and learning the binary similarity term with a CNN. Egiazarian et al. [EVA*20] present a deep learning–based method to vectorize technical line drawings, such as floor plans, architectural drawings, and 2D CAD images. The input to their method is first preprocessed by a raster-space cleaning network that removes the noise, adjusts the contrast, and fills in missing parts. The cleaned image is then broken into patches, then each patch is fed into a network predicting primitives; these primitives are then aligned with the input via optimization. Cloud2Curve [DYH*21] targets 2D point clouds, which can be derived from sketch pixels. It employs a generative model that iteratively outputs Bézier curves to approximate the input point cloud. Liu et al. [LLLW22] adapt a stroke tracing workflow, similar to Mo et al. [MSSG*21] in Sec. 4.1.2, with Transformer-based networks (Fig. 16). Unlike the sliding window view of Mo et al.'s, this method applies a global Transformer-based stroke encoder to the raster input to identify

strokes and encode them into stroke features, each of which is then passed to a raster stroke decoder and a Transformer-based stroke vectorizer. The stroke decoder and vectorizer are trained with shared weights with multi-modal supervision. Their results tend to contain longer, more consistent vector strokes than the ones of [MSSG*21]. Yan et al. [YLA*24] propose an efficient vectorization method based on distance fields and junction detection, using the keypoint detectors similar to [PNCB21] (Fig. 17). Their key idea is to predict unsigned distance field and extract the sketch centerlines via Neural Dual Contouring [CTFZ22]. The reconstruction from Neural Dual Contouring depends on the grid size and suffers from under-sampling. To fix this issue, the authors explicitly detect undersampling and refine the corresponding parts in a post-processing step. Additionally, to improve the accuracy of junction recovery, the system predicts keypoint maps, which help in resolving sharp corners and complex multi-way junctions during post-processing.

4.1.2. Rough sketch vectorization

Rough sketch vectorization methods simultaneously clean up and vectorize an input raster sketch. In theory, the same can be done via a combination of a clean vectorization method and a cleanup method, yet there are multiple issues with that approach. Using raster cleanup methods (Sec. 4.3.1) as preprocessing for a clean vectorization can significantly change the topology of the sketch, e.g., by introducing gaps or deforming junctions; semi-manual cleanup is similarly challenging and error-prone (Fig. 18c,d). Applying clean sketch vectorization methods first and then using vector-based cleanup methods (Sec. 4.3.2) is also far from ideal: Clean sketch vectorization methods produce many redundant strokes that have different structure or shape from natural overdrawn sketches (Fig. 18b), contradicting the assumptions of vector cleanup methods that are designed for natural sketches. Rough sketch vectorization methods are specifically designed to resolve this issue.

An early method [BCF*07] follows a similar workflow as the clean sketch vectorization and uses a bank of 32 Gabor filters to account for various local orientations and stroke cluster internal gaps in rough sketches. The estimated orientations and centerline locations are then traced using Kalman filter.

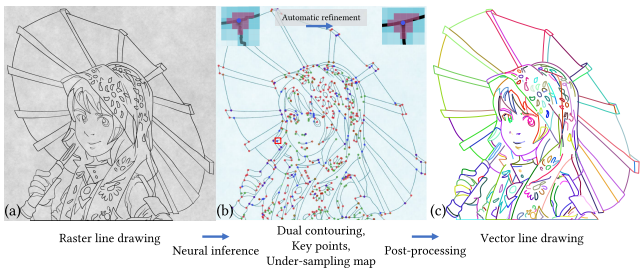


Figure 17: *Yan et al. [YLA*24] propose an efficient vectorization method based on distance fields and junction detection. The reconstructed vector strokes from Neural Dual Contouring may suffer from under-sampling with large grid sizes, which the authors address by detecting and refining these areas in a post-processing step. Figure adapted from [YLA*24].*

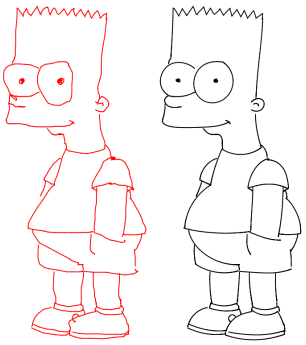
Many methods [FLB16, PPM18, PCS21] are based on region information (Sec. 3.4). They extract the intermediate vector line drawing as boundaries defined by these regions, simplify the graph topology and optimize the geometry into the final vector result. This formulation relate these vectorization methods to the flat colorization problem (Sec. 4.4.1) as they can produce region partitions as a by-product and vice versa. Fidelity vs. simplicity [FLB16] extracts the initial skeleton of the overdrawn input using the trapped ball algorithm [ZCZ*09] (Sec. 3.3), then constructs a hypergraph where nearby potential junctions are grouped, and finally optimizes for the junction connectivities balancing a fidelity term for fitting accuracy and simplicity term for graph simplicity (Fig. 19). In practice, their implementation has problems handling open curves due to the dependence on the trapped ball algorithm. Parakkat et al. introduce two Delaunay-triangulation-based methods [PPM18, PCS21] that start by finding regions in the sketch (Sec. 3.4). They then mark all other triangles as strokes and perform morphological thinning on that, which after some smoothing, creating a 1-skeleton; they expand it into an interactive system in [PCS21].

Stanko et al. [SBBB20] define the strokes as integer isolines of a parameterization aligned to the frame field, thus applying quad meshing methods to the vectorization problem. However, this method's ability to discern small details heavily depends on a global kernel size, while level of detail can be different in different areas of the same sketch, so they resort to user-painted level of detail masks.

In Mo et al. [MSSG*21] at each iteration, a combination of CNN and recurrent neural network (RNN) outputs a pen trajectory starting from a center of a small window, and a pen state. Pen state refers to drawing (pen down) or shifting the pen (pen up). Even though to avoid redundant overlapping strokes, they use a loss that balance fidelity and simplicity similar to the prior art [FLB16], their outputs often contain redundant overlapping curves. A followup work [MGW24] adapts this tracing framework to the new scenario of joint tracing (Fig. 20). In 2D animation production, animators typically trace consecutive raster frames manually, despite the close correspondences between the traced vector sketches. Given a traced vector sketch from the first raster frame and a sequence of subsequent frames, the method aligns local tracing windows — centered at stroke starting points — between the reference frame and the target using a MLP. It then traces in the target window given the reference vector stroke using a RNN. To reduce the redundant strokes in results, they introduce a new window transformation module which enables tracing windows to have higher degrees of freedom.

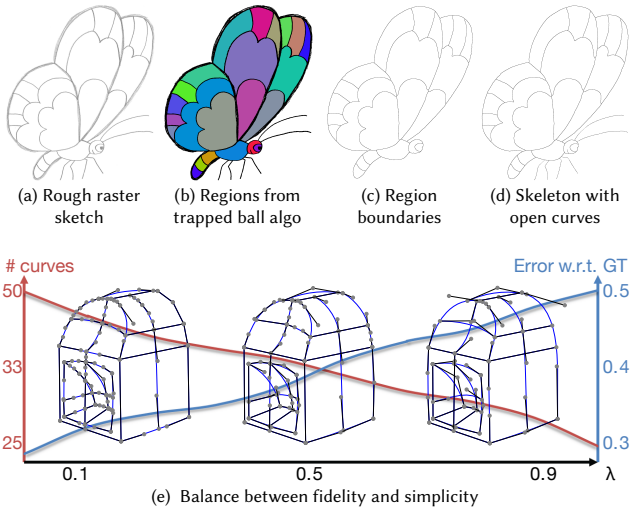
4.2. Beautification

Sketches people draw may not always look exactly as intended, especially when the artist is less experienced or drawing with imprecise digital devices (red/left, inset; adapted from [TSB11]).



(a) Rough raster sketch (b) Result by clean sketch vectorization (c) Semi-automatic raster cleanup (d) Vectorized manual cleanup

Figure 18: *(a) A typical professional raster sketch in the wild often contains a significant number of overdrawn strokes. (b) Applying state-of-the-art clean sketch vectorization [YLA*24] results in redundant strokes due to overdrawing. (c) A human user can produce a high-quality cleaned-up raster sketch using semi-automatic tools, such as Smart Inker in this example [SII18b]. (d) Applying clean sketch vectorization to this cleaned raster sketch may introduce additional errors. Images are from [YLA*24] and [SII18b].*



(a) Rough raster sketch (b) Regions from trapped ball algo (c) Region boundaries (d) Skeleton with open curves

(e) Balance between fidelity and simplicity

Figure 19: *Fidelity vs. simplicity [FLB16] extracts the initial skeleton of the overdrawn input using the trapped ball algorithm [ZCZ*09] (a-d), and optimizes for the junction connectivities balancing a fidelity term for fitting accuracy and simplicity term for graph simplicity (e). Figure adapted from [FLB16].*

Since the early days of digital sketching systems, beautification has been a necessary component. For instance, the pioneering SketchPad [Sut98] supports snapping close endpoints and constraining strokes to lines or circles. The process to correct these errors is referred as *beautification*, *neatening* or *auto-correction*. More formally, a beautification method refines strokes in a vector sketch to make the overall sketch more visually appealing (black-/right, inset). While most methods achieve this by beautifying each vector stroke individually, recent approaches have begun optimizing stroke layouts [YLGF23].

4.2.1. 2D Beautification

For general purpose sketches, beautification systems fit strokes closely to the input points (Fig. 21a). In contrast, beautification systems for technical diagrams often perform template matching, as the range of admissible stroke shapes is typically more limited (Fig. 21b). PaleoSketch [PH08] is a stroke recognition system which also beautifies the input stroke as a by-product. Their system supports a range of geometric primitives such as ellipses or straight lines, as well as Bézier curves, each time providing a list of suggestions of beautified strokes. Frisken [Fri08] propose a beautification method that incrementally fits cubic Bézier curves to incoming input points, leveraging a vector distance field for gradient-based optimization. Baran et al. [BLP10] promote the use of clothoid splines, a curve representation with a piecewise linear curvature profile. A key to their problem is a segmentation of a stroke into curve primitives (lines, arcs, and clothoids), which they cast as a shortest path problem on a weighted graph. McCrae and Singh [MS11] get inspiration from traditional design where French curves or sweeps have been extensively used in the creation and editing of 2D design curves. Their system loads a set of pre-authored French curves and automatically finds an optimal piecewise combination of French curve segments with user controlled continuity up to G^2. Elasticurves [TSB11] focuses on real-time beautification and is based on the observation that slower sketching speed indicates more accurate strokes, while fast sketching often induces more noise and thus requires more smoothing. Elasticurves are represented by arc splines where the individual arcs have curvatures corresponding to the local sketching speed. HelpingHand [LYFD12] is a data-driven beautification method that not only neatens strokes but also assigns calligraphic widths. Built on artist stroke datasets, HelpingHand segments a stroke into smaller segments, retrieves corresponding artist segments, and blends these segments via optimization. EZ-Sketching [SLWF14] is an image-tracing system that automatically refines sketches to be more faithful to the underlying images and visually pleasing. The system employs an optimization framework at three levels: local, where several candidate image edges are proposed for a stroke to snap onto; semi-global, where points on strokes are triangulated, and spatio-temporal neighboring stroke points are brought closer through en-

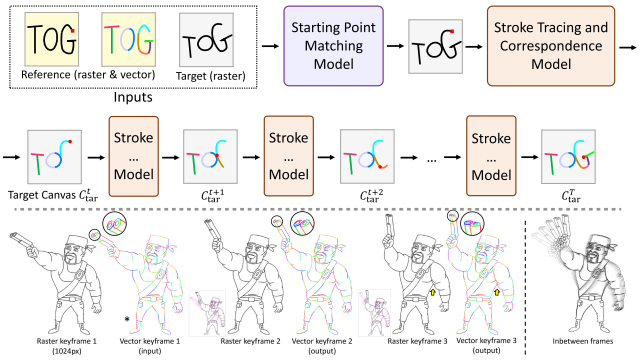
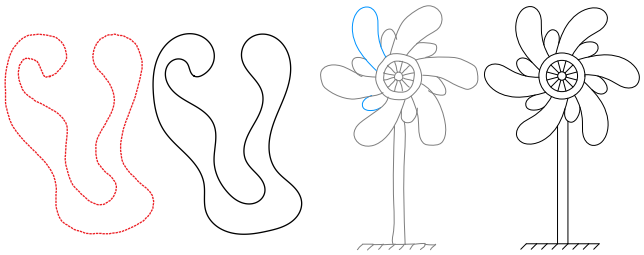


Figure 20: *Mo et al. [MGW24] transform a given traced vector sketch from the first raster frame to a later raster frame in the sequence. At each iteration, their method first identifies corresponding tracing windows on the reference and target frames, then, given a reference vector stroke, generates a pen trajectory starting from the center of the target window (top). Given a traced vector sketch on the first keyframe, this method generates vector sketches for all subsequent keyframes (bottom). Figure adapted from [MGW24].*



(a) Beautification for general purpose sketches

(b) Beautification for technical diagrams

Figure 21: *Beautification systems for general-purpose sketches fit more closely to the input points (a) compared to those designed for technical diagrams (b). For example, Baran et al.'s [BLP10] system utilizes three types of curve primitives to construct flexible splines (a), while ShipShape [FASS16] replaces entire strokes with lines, arcs, or template strokes drawn earlier in the same sketch (blue, b). Images are from [BLP10] and [FASS16].*

ergy minimization; and global, where undetermined strokes are deformed based on the underlying mesh.

Beautification methods for technical drawings often match and replace input raw stroke samples with a template shape from a library. Some methods in this category also neaten junctions [IMKT07, MSR09, CGL12, FASS16] or account for high-level global properties such as path identity or symmetry [IMKT07, FASS16]. Pavlidis and Van Wyk's method [PVW85] first clusters the input line segments based on three properties (angle, line segment length, and endpoint alignment), then enforces the properties of member segments to match the cluster means. Igarashi et al. [IMKT07] propose an incremental and semi-automatic method for interactive geometric design. Every time when a new stroke is created, their system generates the set of all possible constraints satisfying endpoint snapping, endpoint alignment, line axis alignment, symmetry, and parallelism. Then, based on the user selected constraints, the system solves constrained line fitting problems and presents the user with a set of all possible line segments to select from. Murugappan et al. [MSR09] introduce a pipeline and an interactive workflow similar to the previous work [IMKT07]. They consider multiple strokes, more primitives than just the line segment, and more constraint types. To prevent potential constraint combinations and solutions from explosion, they have a constraint selection step and a more complex suggestion evaluation step. QuickDraw [CGL12] is a drawing system that follows the similar pipeline of recognizing primitives (lines and circles), inferring geometric constraints, and solving for constrained fitting. ShipShape [FASS16] extends the work on incremental and semi-automatic beautification methods with three improvements: (1) the primitive set includes Bézier curve chains drawn early in the input; (2) activated constraints are selected by exploring a constraints tree; (3) the stroke similarity detection is based on Fréchet distance and supports similarity measure between Bézier chains. See Fig. 21 for an example. Yu et al. [YLGF23] propose a learning-based beautification system for human-made objects. The system first semantically parses the input into parts, then retrieves clean parts from the training data and beautifies through interpolation between the two. Finally, the layout of the parts is refined through structure beautification.

4.2.2. 3D Beautification

3D sketching systems that process mid-air 3D strokes face challenges similar to those in 2D sketching, such as imprecision from digital devices and less experienced users. These challenges are even more pronounced as VR/AR technologies and drawing practices are still evolving. Basic filtering or smoothing strategies are commonly used due to their simplicity and low computational cost [AMW*23, Chapter 7]. Despite their differences, many 3D sketching systems are inspired by 2D beautification, extending them as components of more complex systems that incorporate novel interactions, visual guidance, and editing tools. Currently, few papers explore match-and-replace strategies, similar to 2D common technical drawing beautification, while most focus on generating high-fidelity strokes that closely fit the input sample points, akin to 2D general-purpose beautification.

In the first category, Mockup Builder [DACJH13] uses a customized stereoscopic multi-touch system to support a workflow involving silhouette creation, volume creation (through pushing and pulling), and shape manipulation (e.g., scaling, rotation, and translation). Silhouette creation is performed by sketching directly on the multi-touch surface. In addition to the incremental fitting of lines and cubic Bézier curves, the system detects circles and ellipses and converts them into precise Bézier curve representations. Multiplanes [MAS*18] displays visual guidance planes for snapping and points to trigger beautification. Beyond snapping to vertices, strokes, and planes, 3D strokes are detected and replaced with neat lines, general curves, arcs, and circles.

The majority of 3D sketching systems focus on producing high-fidelity 3D strokes, though these systems can be specialized for certain types of drawings, such as curve networks, industrial design sketches, and curves on 3D surfaces. An early work, Drawing on Air [KZL07], proposes two complementary input techniques

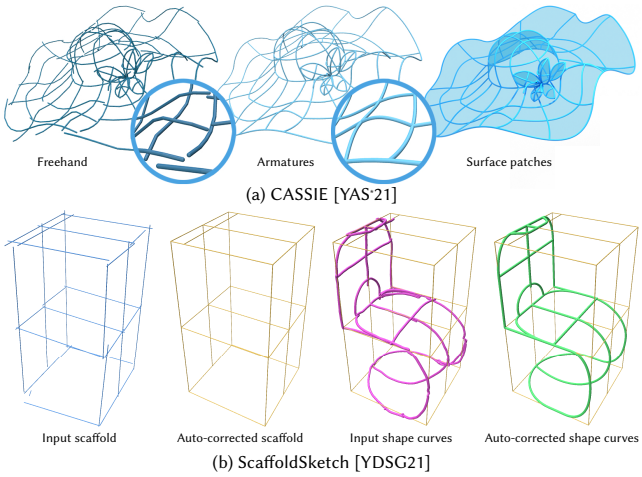


Figure 22: *CASSIE [YAS*21] (a) and ScaffoldSketch [YDSG21] (b) are 3D sketching systems designed for curve networks and 3D industrial design sketches, respectively. They can be considered complementary: CASSIE focuses on a more freeform and seamless sketching experience; ScaffoldSketch closely follows traditional precise industrial design sketching practices. Images are from [YAS*21] (a) and [YDSG21] (b).*

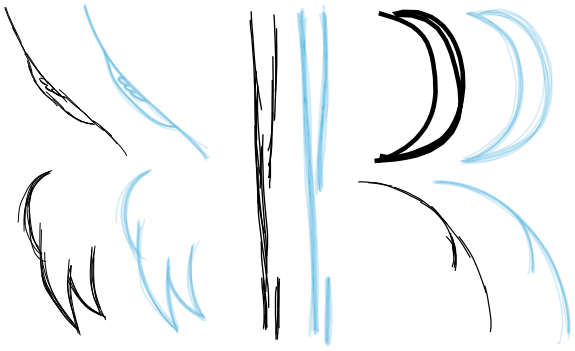


Figure 23: *Liu et al. [LRS18] conducted a perceptual study demonstrating that human viewers effortlessly perceive clusters or strips of rough strokes as jointly depicting a single, artist-intended curve, and their observations generally align with one another. The overdrawn strokes in question are shown in black, while manually created clean lines by ten study participants are overlaid in blue. Figure adapted from [LRS18].*

that implicitly beautify 3D strokes. The one-handed drag drawing technique employs a "tow rope" metaphor, similar to the one used by Elasticurves [TSB11], to dynamically smooth the drawing direction and reduce jitter. CASSIE [YAS*21] is a 3D curve network creation system for VR that offers three modes: freehand (with minimal smoothing), armatures (curve network), and surface patches (Fig. 22a). The armature mode is inspired by beautification and supports the creation of precise curve networks through continuous and discrete stroke optimization. Another crucial component is curve network surfacing of the patch mode, which is discussed in Sec. 4.4.2.1. Their user studies show that, compared to the freehand mode, the additional mental load imposed by the armature and patch modes does not noticeably hinder users' creative potential. ScaffoldSketch [YDSG21] is a 3D industrial design sketching system for VR that utilizes two types of lines: scaffold and shape, each beautified with distinct auto-correction algorithms (Fig. 22b). Scaffold lines are straight lines that can be easily drawn accurately, commonly serving as visual guides for precise and aesthetic projections. ScaffoldSketch implements scaffolds as elastic lines with tick marks. The auto-correction is formulated as determining a set of mutually satisfiable constraints, discarding conflicts via iteratively re-weighted least squares. Shape lines describe the intended form, which are beautified by constraining to be smooth while passing through and tangent to the scaffold lines. CASSIE and ScaffoldSketch can be considered as two complementary systems since curve networks bear connections to shape lines. While CASSIE targets a more freeform and seamless sketching experience, ScaffoldSketch follows closely to traditional precise industrial design sketching practices. Moreover, both systems report incorrect system responses due to thresholds determined empirically by system developers, highlighting the need for advances in determining more customized interpretations of user inputs. Arora et al. [AS21] study a more specific question: how to intuitively and aesthetically project a *mid-air* 3D stroke onto a 3D surface. They demonstrate the limitations of the most commonly used projection method in commercial VR painting software, which adopts a "spray can" metaphor where a raycast projection creates the stroke. Instead, Arora and Singh introduce "mimicry", a curve beautification method that transposes the user's 3D painting motion to a stroke that is smoothly projected onto a nearby 3D surface. This method achieves a smooth projection that corresponds to the user's input motion by anchoring the current stroke offset (the vector between two consecutive stroke points) to the last projected point on the surface and applying a smooth closest-point projection (a modified Phong projection [PBDSH13]).

4.3. Cleanup

When creating an initial sketch, artists often draw roughly parallel strokes, a technique called *overdrawing*, to correct or refine earlier strokes, add emphasis, depict thickness, or break down hard to draw long and complex curves into shorter, easier to sketch strokes (Fig. 3). While human viewers effortlessly perceive a cluster or a strip of rough strokes as jointly depicting a single artist intended curve (Fig. 23), downstream sketch-based applications have difficulties process such sketches. Sketch cleanup, also referred to as *consolidation* or *simplification*, takes a rough sketch with overdrawing and produces a corresponding clean line drawing.

4.3.1. Raster-to-raster cleanup

Raster-to-raster cleanup methods transform a raster sketch with overdrawing into a clean raster sketch. These methods are closely related to the problem of vectorization, where the majority of systems (Sec. 4.1) only support clean raster sketches as inputs. For such vectorization systems, raster-to-raster cleanup methods can serve as preprocessing (Fig. 18cd). In addition to overdrawing, these methods also handle raster sketches scanned in poor condi-

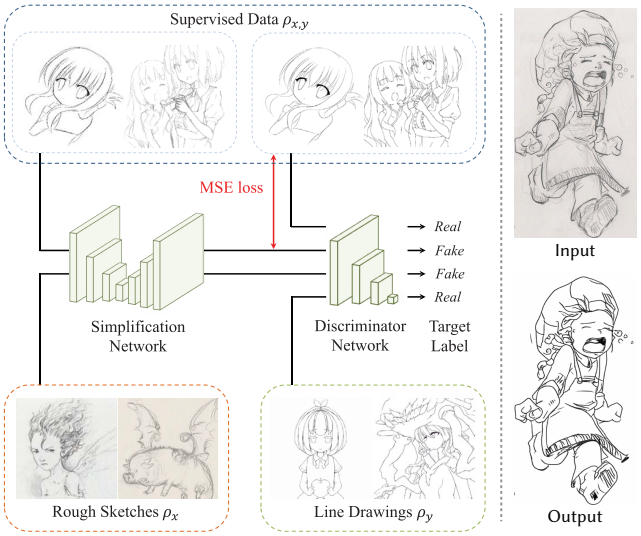


Figure 24: *Simo-Serra et al. [SII18a] base their raster-to-raster cleanup network on the network proposed in their first approach [SISI16]. With the newly introduced discriminator network, this model is jointly trained with supervised and unsupervised data. An example is shown on the right. Figure from [SII18a].*

tions, such as non-uniform canvas textures, uneven lighting, and low contrast.

A classical method of Chen et al. [CGBG13], after computing a line field parallel to local tangents (Sec. 3.2), uses the directions of the line field for an oriented filter with a fixed kernel size, merging parallel strokes. Due to the fixed kernel size, their method may not work on sketches containing details of different sizes. Recent raster-to-raster cleanup methods are often deep learning based and often struggle because of scarcity of training data. Simo-Serra et al. [SISI16] start this line of work by training a CNN with annotated rough-clean sketch pairs. They improve the performance in a followup paper [SII18a] by jointly training with supervised and unsupervised data leveraging a GAN (Generative Adversarial Network) loss (Fig. 24). Smart Inker [SII18b] is an interactive cleanup system that introduces smart tools designed specifically for rough sketches to connect strokes, erase shading, and fine-tune the line drawing output (Fig. 18c). This system has a module that improves the anti-aliasing and normalizes input line widths for a more consistent training dataset. Xu et al. [XXM*19] show that compared to the pixel loss or discriminator loss, the perceptual loss is more effective at global semantic understanding and cleanup of extremely sketchy inputs. They propose a new CNN model with the integration of VGG (Very Deep Convolutional Networks) layers trained on a new rough sketch dataset. SketchCleanNet [MKDM22] targets rough query sketches for 3D CAD model retrieval systems. The method uses a Fully Convolutional Network trained on synthetic line drawings traced based on 3D CAD objects.

A generic reconstruction method by Chen et al. [CGBG13] can also be applied to the raster-to-raster cleanup task. They introduce a moving least squares–based (MLS) method to robustly construct continuous gradient line fields (Sec. 3.2) from a variety of noisy

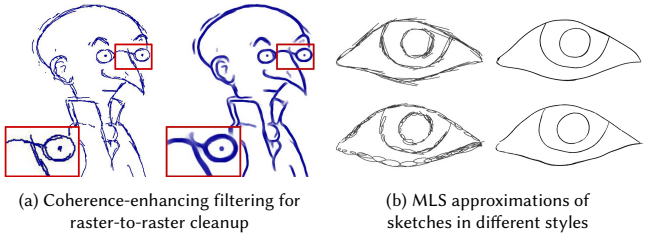


Figure 25: *Chen et al. [CGBG13] demonstrate that their MLS method can be applied to clean up raster sketches with overdrawn strokes as an image filter (a), and to process sketches across various drawing styles (b). The result in (a) is generated using their isotropic linear approximation with a uniform shock size (i.e., line width). Inputs are shown on the left, and results on the right. Figure adapted from [CGBG13].*

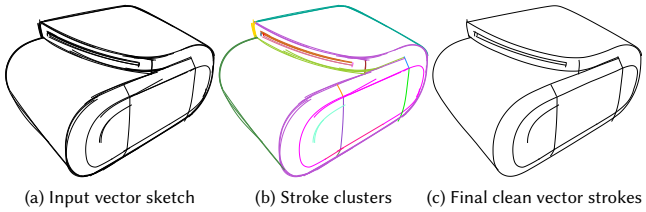


Figure 26: *Given an input vector sketch (a), most vector-to-vector cleanup methods identify groups of strokes perceived as depicting single curves (b), and fitting the aggregate curve to each group (c). Images are results of [LABS23].*

discrete samples, such as general raster images and point clouds. When applied to raster rough sketches, their method performs line drawing stylization, producing a raster image with improved line density and directional coherence, i.e., cleaner lines (Fig. 25a). They also demonstrate the application of their method to sketches in different styles (Fig. 25b).

4.3.2. Vector-to-vector cleanup

Vector-to-vector cleanup methods mainly follow a framework of identifying clusters, groups of strokes perceived as depicting single curves (Fig. 26b), and fitting the aggregate curve to each group using methods like Bézier curve fitting or StrokeStrip [PvMLV*21] (Fig. 26c) (Sec. 2.4, 3.3). Early vector-to-vector cleanup methods

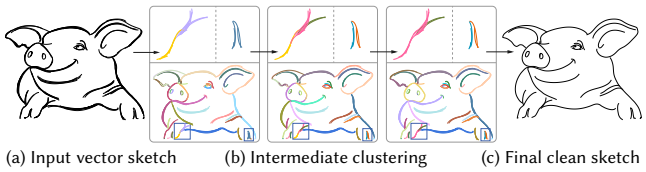
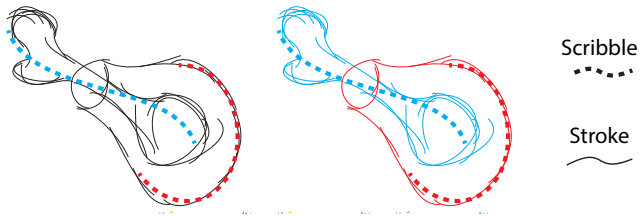


Figure 27: *StrokeAggregator [LRS18] utilizes global and local perceptual cues determined by perceptual studies and This method performs clustering in three stages: angle and density-based coarse clustering, branch separation, and final merging. Figure adapted from [LRS18].*

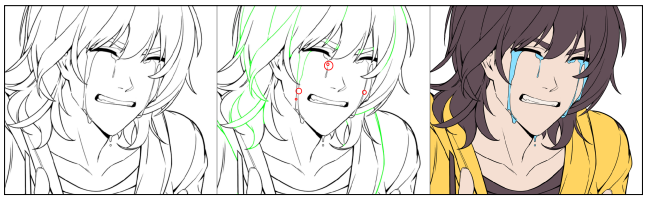


(a) Input vector sketch and scribbles (b) Output stroke clustering

Figure 28: *Noris et al. [NSS*12] leverage user-drawn guiding strokes, called* scribbles *(dashed lines), and cluster raw input strokes based on their perceptual similarities to those scribbles, taking drawing order into account. Figure adapted from [NSS*12].*

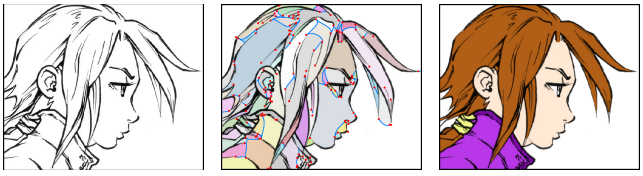
[Ros94, BTS05, SC08] compute stroke-wise similarities based on perceptual cues such as absolute proximity, degree of parallelism, and Gestalt good continuation. These methods determine grouping via hard thresholding, which often requires manual tweaking of threshold per input. Some methods [Ros94, BTS05] quantify cues based on correspondences, e.g., the closest points, endpoints and midpoints, which are sensitive to noise; Shesh et al. [SC08] rely on all combinations of sufficiently close points, which is computationally expensive. Orbay and Kara [OK11] propose a neural network–based method and identify *branching*, when adjacent strokes diverge forming a Y-junction, as a challenge for vector-to-vector cleanup. Their model learns a pairwise probability function based on angles and distances, then builds initial clusters by thresholding the probabilities and subsequently refines these clusters using branch separation. The method works well when trained and tested on drawings produced by the same artist, but fails to generalize to drawings from multiple sources; it typically over-merges. Liu et al. [LWH15] introduce contextual angle and proximity metrics defined relative to the size of empty spaces, or regions, enclosed by the input strokes (Sec. 3.4). Their method still requires users to manually adjust two thresholds per input. StrokeAggregator [LRS18] performs stroke grouping based on a cluster parameterization, following global and local perceptual cues determined by perceptual studies. This method performs clustering in three stages: angle and density-based coarse clustering, branch separation, and final merging (Fig. 27).

Some vector-to-vector cleanup methods are interactive, i.e. they produce clean strokes on the fly, enabling sketching in vector drawing systems. Several interactive drawing systems [Bau94, BBS08, GJ12] are designed for 3D sketching, and due to the usage of simple cues similar to those of early consolidation methods, heavily depend on additional user input, such as gestures or clicks. Noris et al. [NSS*12] leverage user-drawn guiding strokes, called *scribbles*, and cluster raw input strokes based on their perceptual similarities to those scribbles, taking drawing order into account. Unlike the cleanup setting where raw strokes are grouped into strips, this interactive stroke grouping system aims to divide strokes into semantic components (Fig. 28b). StripMaker [LABS23] (Fig. 26) is an interactive learning-based method that formulates the grouping problem into smaller binary decisions of whether to merge a stroke and a group or a group and another group. To address training data scarcity, their local approach allows them to break down each annotated sketch into many training examples. One drawback of this method is its runtime, which increases significantly with the number of existing groups and strokes. Typically, once the clusters are known, those methods find centerlines, as explained in Sec. 3.3.



(a) Input sketch (b) Inaccurate junctions (c) Flat colorization

Figure 29: *Manually drawn sketches (a) often contain imprecise junctions (b), including falsely closed lines (green, b) and falsely open areas (red, b). In this example, a professional comic artist needs to carefully correct these issues to achieve the desired flat colorization. Figure adapted from [YCY*22].*



(a) Input raster sketch (b) Flatted regions (c) Regions with hand-picked colors

Figure 30: *Fourey et al.'s [FTR18] system extends dangling stroke endpoints (red dots, b) with line segments or splines (blue curves, b) to close regions (colorized regions, c). The system is semi-automatic and relies on users to select over-partitioned regions for flat colorization. Figure adapted from [FTR18].*

4.4. Flat Colorization

Flat colorization, or *flatting*, a common stage of colorizing a sketch, involves partitioning a drawing into regions and filling each with a constant color. The exact colors do not matter — those are later often manually picked by an artist, which distinguishes this problem from the broader topic of image colorization. As described in Sec. 3.1, the key issue is that the standard tools like flood fill ('bucket tool') typically fail: junctions in sketches are imprecise, burdening artists with tedious low-level manual corrections (Fig. 29).

4.4.1. Raster flat colorization

In raster drawing software, flatting is often done manually by selecting the pixels in a perceived region. As an automation of this tedious process, Zhang et al. [ZCZ*09] propose a trapped ball algorithm (Sec. 3.4) as the segmentation step of their cartoon vectorization pipeline. A useful UI tool to speed up manual selection of regions is "color scribbles", a short curve with an intended color to loosely indicate the region [QWH06, SDC09]. Qu et al. [QWH06] develop a scribble-style method based on level-set segmentation for black and white mangas that contain hatching, halftoning, and screening. Lazybrush [SDC09] follows a similar interaction and

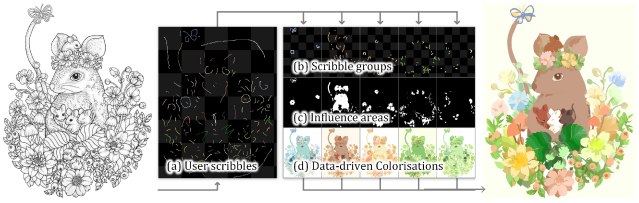


Figure 31: *Zhang et al. [ZLSS*21] use deep neural networks to estimate influence areas (c) and resulting colors (d) in a classic semi-automatic workflow based on user scribbles (a). Figure adapted from [ZLSS*21].*

Figure 32: *Dai et al. [DZL*24] use deep neural networks to transfer flat colorization from a reference frame (a) to a target frame (b). Their method accurately handles tiny segments (c), closely resembling the results of manual flat colorization (d). Figure adapted from [DZL*24].*

detects regions with the classic multiway graph cut segmentation. This method is integrated into several commercial drawing software, such as TVPaint [Laz24] and Krita [Kri24]. Fourey et al. [FTR18] develop a semi-automatic method for digital drawing that extends dangling strokes with line segments or splines to close regions (Fig. 30).

More modern raster flat colorization methods use neural networks. An early raster-to-raster model by Sasaki et al. [SISI17] directly fills in the gaps for easy partitioned by other methods. Zhang et al. [ZLSS*21] apply deep neural network to the classic semi-automatic approach based on the DanbooRegion dataset of annotated colorized artwork [ZJL20] (Fig. 31). A recent raster based deep learning method [DZL*24] targets a similar animation production scenario as Mo et al. [MGW24] (Sec. 4.1.2) and allows users to simultaneously colorize characters across a sequence of raster frames (Fig. 32). The system first generates a coarse colorization using optical flows between two frames, and then refines it by deforming reference regions with a multiplex transformer. The model is trained on a new dataset consisting of professional annotations and synthetic data generated through non-photorealistic rendering.

A recent system, FlatMagic [YCY*22], introduces an improved bucket tool and fine-grained control over region boundaries. Their design choices are informed by their user studies with professional comic artists (Fig. 33). Their bucket tool combines the classic trapped ball algorithm and a neural fill method. The fine-grained boundary editing allows users to either select from a set of boundaries proposed by a neural network or draw the boundary manually.

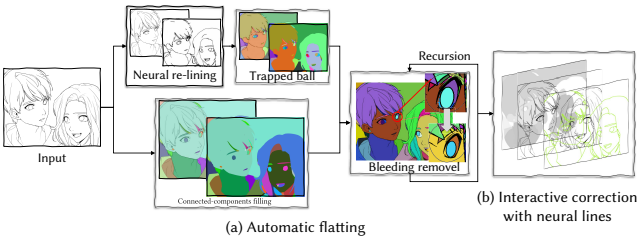
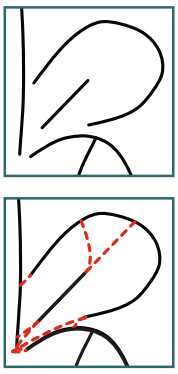


Figure 33: *FlatMagic [YCY*22] provides users with automatically generated flatted regions (a) and allows further user corrections using neural line options (green, b). The automatic flatting is achieved by detecting neural fills with the trapped ball algorithm (top, a) and classic connected-components filling (bottom, a). These regions then undergo iterative bleeding removal (right, a). Figure adapted from [YCY*22].*

Another recent system, FlatGAN [KLL*23], generates a region map and a distance field of the input lines, referred to as a contour map. This GAN is trained on augmented DanbooRegion data [ZJL20], with segments of lines randomly erased and random noise added. The region map and contour map are then post-processed to produce accurate regions with anti-aliased boundaries.

4.4.2. Vector flat colorization



Even when input sketches are provided in vector format, flat colorization remains a non-trivial problem. Many methods in this category follow a common pipeline: (1) construct potential junctions using stroke geometry (inset, dashed lines; adapted from [JSL21]); (2) for each junction, determine its connectivity or remove it from consideration. A group of methods focus on *wireframe* drawings of polyhedra [WY09, CPVC19, WZW*20], i.e. with all the lines visible, and leverage stronger assumptions that these drawings contain only straight lines and do not contain intentionally dangling endpoints. These assumptions, as well as domain-specific observations allows for a rather straightforward detection of potential junctions. In those methods, the final connectivity is determined by hand-crafted rules based on angle and distance thresholds [WY09, WZW*20] that are sometimes chosen via perceptual studies [CPVC19]. Jiang et al. [JSL21] propose a semi-automatic method that first constructs potential junctions by clustering dangling endpoints, then determines final connectivity based on a series of cues and thresholds. Since their method assumes that the vast majority of dangling endpoints are unintended, their results often contain incorrect junctions requiring user corrections. Yin et al. [YLL*22] follow a similar workflow and further categorize junctions into binary junctions and high-valence junctions, which are composed of binary ones (Fig. 34). This junction formulation allows them to adapt a learning based approach, using binary junction connectivity as a basic decision (Sec. 3.1). Scrivener et al. [SCC24], without explicitly constructing junctions, utilize generalized winding number to find regions of clean vector sketches

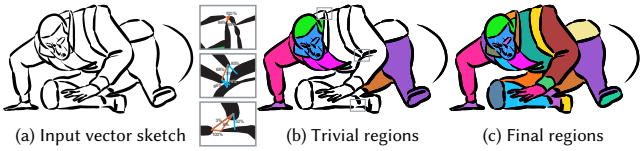
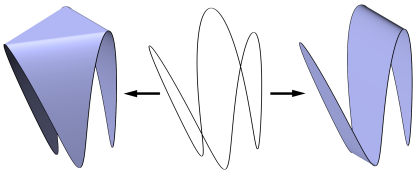


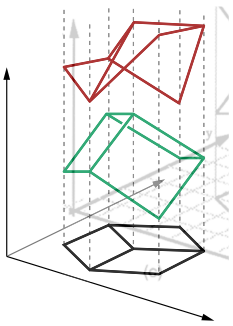
Figure 34: *Yin et al. [YLL*22] categorize junctions into binary junctions and high-valence junctions, which consist of multiple binary junctions (zooms, b). Zoom in to see the predicted connectivity scores in this example. Figure adapted from [YLL*22].*

(Sec. 3.4, Fig. 35). Parakkat et al. [PMC22] further extend their Delaunay-based cleanup method [PPM18, PCS21] for flat colorization. This improved method supports better gap completion lines, colorization of hatched regions and automatic colorization via flooding.



4.4.2.1. Cycle detection in 3D curve networks Cycle detection is a necessary step in surfacing a 3D curve network and can be thought of as a higher-dimensional form of colorization. While there are analogies between cycle detection and flat colorization in 2D vector sketches, the challenges arise from different causes: 3D curve networks have precise junctions, but the topology of the intended surfaces can remain ambiguous (inset; adapted from [BWSS12]). Since cycle detection is at the boundary of our scope, we provide only a brief summary of existing work, hoping that future 3D sketching systems can incorporate these components as building blocks. Cycle detection methods generally assume that the input 3D curve network represents human-ideated shapes rather than arbitrary ones. As a result, most approaches rely on sets of perception-inspired heuristics and regularization conditions within a global optimization framework [AJA12, ZZCJ13b]. Rose et al. [RSW*07] construct developable surfaces from input boundary curves using a branch-and-bound algorithm that explores locally convex triangulations. Sadri and Singh [SS14] observe that human perception of 3D curve networks tends to be consistent and aligns with persistent homology, which they leverage in their flow complex-based method. Several approaches make stronger assumptions about the network to simplify the cycle detection. For instance, Orbay and Kara [OK12] assume the curves form a connected graph, enabling graph loop detection; CASSIE [YAS*21] assumes that the network is incrementally created, allowing for local updates and curve sorting.

4.5. Lifting to 3D



Sketch strokes often can be viewed as a 2D depiction of 3D curves on or close to a 3D surface (Sec. 2.3). The goal of 3D lifting is reconstructing these 3D curves, which can then be used for visualization purposes, edited, or used in 3D surface reconstruction. Starting with the pioneering work of Roberts [Rob63], these methods algorithmically undo this projection and *lift* a sketch into 3D by determining depth values of points along the strokes. The focus of these methods is typically sketches of machine-made shapes, commonly used in industrial and architectural design. Formally, this is an ill-posed problem, as even if we assume 2D strokes are a perfect projection of 3D curves (Sec. 2.2), a single 2D sketch can correspond to an infinite number of 3D objects (inset; adapted from [SA93]). The main challenge of this line of work is exactly resolving this ambiguity.

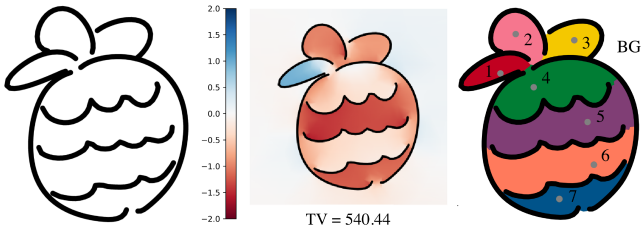


Figure 35: *Scrivener et al. [SCC24] determine the optimal stroke orientations based on the total variance of generalized winding numbers (b) and conduct flat colorization using k-means clustering (c). Figure adapted from [SCC24].*

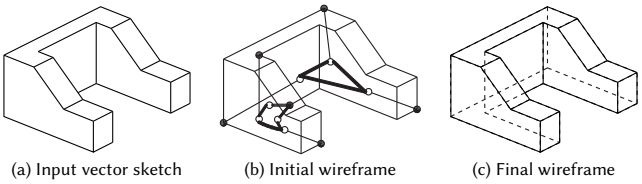


Figure 36: *Kyratzi et al. [KS09] design their method to infer the hidden lines (c) such that the wireframe sketch remains valid while minimizing its complexity (b). Figure adapted from [KS09].*

4.5.1. Offline lifting

Early works are based on labelling of strokes, referred to as 'edges' in this literature, and object corners in clean vector line drawings consisting of line segments [Clo71, Huf71], shadows [Wal75], or curves [Coo08]. The edge labelling process distinguishes convex and concave edges, bounding areas of the surface that are convex or concave for the viewer. Similarly, edge labelling distinguishes curved from planar surface patches [Coo08]. These methods often assume that at all junctions edges are orthogonal to each other in 3D, making the lifting problem less ambiguous. A later method lifts free-hand architectural sketches [CKX*08] by analyzing edges and corners and applying maximum likelihood interpretation on geometric primitives, detailed features and textures.

A common challenge is to determine the hidden lines occluded from the viewpoint (Sec. 3.5). Cao et al. [CLT08] first infer the topology of invisible straight line edges and vertices based on geometric and topological constraints, and then recover the depth information using perceptual clues such as symmetry. Kyratzi et

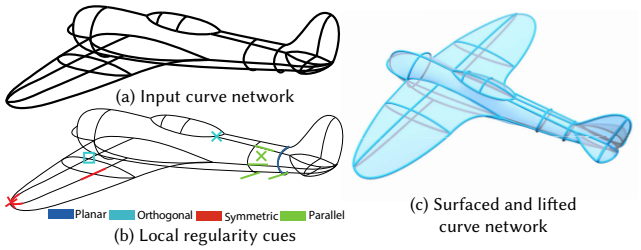


Figure 37: *True2Form [XCS*14] reconstructs 3D curve networks (c) from 2D curve network sketches consisting of piecewise-smooth curves (a) by selectively applying geometric regularity properties, such as parallelism, symmetry, orthogonality, and planarity (b). Figure adapted from [XCS*14].*

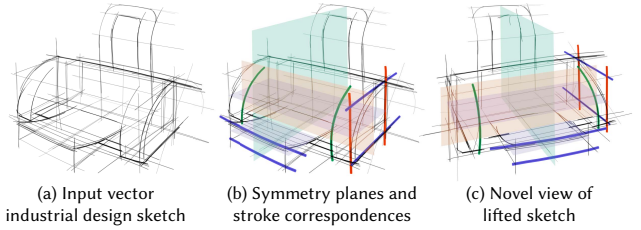


Figure 38: *Hähnlein et al. [HGSB22] adapt symmetry cues (b) to lift freehand industrial design sketches (a). The resulting lifted sketch can be used for sketch novel view synthesis (c). Figure adapted from [HGSB22].*

al. [KS09] minimize the number of hidden lines and regions added to the sketch (Fig. 36). In contrast to the methods above, Varley et al. [VMS05] question the necessity of line labeling for lifting sketches of CAD objects, instead relying solely on visible T-junctions.

A curve network is a set of interconnected curves that define the shape and structure of a surface or object in the context of 3D modeling and CAD (Fig. 37a). Methods in this category often adopt an optimization-based framework with domain-specific regularizers or constraints. Although designed for 2D freehand sketches of wireframe CAD objects rather than curve networks, Lipson and Shpitalni [LS96] propose an early example of these optimization-based lifting approaches. Domain-specific properties of curve networks are often formulated into regularizers or constraints to resolve projection ambiguities. For instance, Cordier et al. [CSMS13] propose a system to lift a 2D sketch of a mirror-symmetric 3D shape. The algorithm infers the 3D geometry by identifying pairs of symmetric curves and computing their 3D positions while ensuring that their orthogonal projections match the input sketch. True2Form [XCS*14] reconstructs 3D curve networks from 2D curve network sketches consisting of piecewise-smooth curves by selectively applying geometric regularity properties, such as parallelism, symmetry, orthogonality, and planarity (Fig. 37). Their key observation is that cross-sections, often drawn in those sketches (Sec. 2.4), are depiction of principal curvature lines and therefore are orthogonal in 3D. Their inputs must be clean, without overdrawing, and have precise junctions labelled by users.

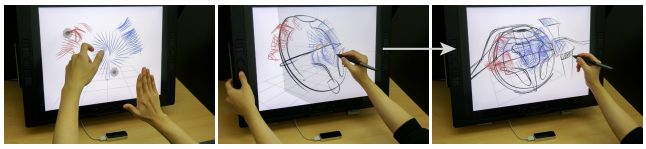


Figure 39: *Kim et al. [KALB18] propose a 3D sketching system that complements regular pen sketching with quick hand-motion scaffold creation (a). With a rough 3D scaffold, users can sketch and refine 3D strokes relative to this scaffold (b). Figure adapted from [KALB18].*

Free-hand industrial design sketches share similarities with curve network sketches but are often more challenging due to the presence of overdrawn strokes and inaccurate topology (Sec. 3.1). Mitani et al. [MSK02] design a sketching system, used for novel view synthesis, that inflates input vector sketches potentially containing overdrawn strokes by matching a predefined cube template. In a later paper, Gryaditskaya et al. [GHL*20] focus on automatically lifting raw, real-world freehand industrial design sketches, which include both scaffold lines and surface curves. The authors formulate depth estimation as the problem of assigning binary activation values to intersections, namely distinguishing accidental (not intersecting in 3D, but intersecting in 2D due to the projection) and true intersections (intersecting in 3D and 2D), and propose an efficient search algorithm that exploits stroke drawing order. In a following paper, Hähnlein et al. [HGSB22] adapt symmetry as a powerful cue into this intersection formulation (Fig. 38): This method identifies local and global symmetries within the sketches, leveraging them in an integer programming framework. The authors show improved robustness for imperfect or incomplete sketches.

Another closely related topic is single-view modelling of freeform shapes. Many methods begin by lifting input descriptive construction lines [AS11, BCHS20b] or silhouette lines [BCV*15], often using T-junctions as cues for Z-ordering, followed by a later surfacing step. Additionally, these methods often require users to provide topologically accurate junctions to correctly determine the stacking order of different shape regions. We refer readers to a survey [BC20] for a full discussion of sketch-based single-view modelling.

4.5.2. Interactive lifting

Interactive lifting of 2D strokes enables 3D sketching with a 2D input interface. The resulting 3D sketch can be displayed as a 3D scene on a 2D screen or in a VR/AR environment. Lifting methods vary by target display — VR/AR approaches assume a larger physical workspace and a more immersive viewing experience.

When sketching on a 3D scene displayed in 2D, users draw strokes directly on the scene, sometimes supplemented by additional input such as curve shadows [CMZ*99] or alternative techniques such as hand gestures [KB16, KALB18] (Fig. 39). Many works in this category develop complete drawing systems targeting seamless interactions, where lifting serves as a submodule. Lifting is mainly done by projecting to 3D proxies: base template models

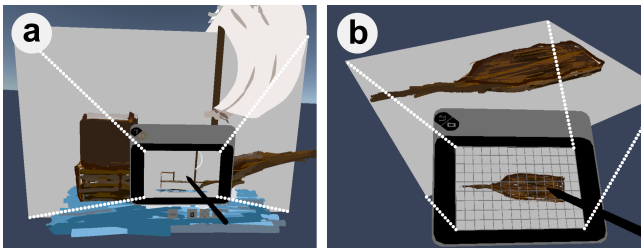


Figure 40: *VRSketchIn [DGK*20] is a VR drawing system that allows users to freely place 3D drawing surfaces (a) and lifts 2D strokes on these surfaces to 3D via direct mapping. Figure from [DGK*20].*

[KS07]; surface patches selected and created by the user [BBS09]; planes that are created to be axis-aligned or freely via pen ticking [BBS08, KB16] or with respect to a scaffold [SKSK09, KALB18]; plane canvases that can be positioned in 3D [DXS*07]; freeform canvases created via extrusion [OCR*19]. Some papers approach lifting through more advanced optimization techniques, such as geodesic deformation when only the stroke endpoints lie on the base template model [KS07], or combinatorial optimization based on a fixed configuration set of the input stroke and salient geometric features [XFZ*18]. Schmidt et al. [SKSK09] introduce a pioneering interactive design tool for industrial design, based on priors about sketching techniques — as done in other later work [GHL*20, HGSB22, YDSG21]. Using this tool, designers first create 3D scaffold lines lifted based on rules of parallelism, perpendicularity, and connections to known 3D points. Freehand shape lines are lifted with respect to scaffold lines via optimization with a total absolute curvature minimization regularizer and position and tangency constraints. SecondSkin [DPS15] is a sketch-based modeling system designed for in-context layered shape creation, such as garments that conform closely to character shapes. To lift a 2D stroke, it is first classified into one of four predefined curve types. Based on the curve type, stroke points are projected or aligned with the underlying layer geometry, including the input model and previously created layers. The system then generates surfaces or volumes defined by stroke loops. SweepCanvas [LLZ*17] focuses on in-context modelling within an RGB-D photo of a real-world scene. In this system, a pair of 2D strokes representing a profile is used to generate a sweep surface along a trajectory stroke. Lifting is formulated as a stroke-plane assignment problem given estimated planes in the scene. Skippy [KYC*17] is an interactive single-view modelling tool designed for intuitive curve creation around input shapes. A curve is generated by finding a path in a segment graph and then smoothed by minimizing internal and external curvature-related energies.

An early work about lifting in a VR/AR environment explores fully manual lifting, where a user draws over a 2D sketch displayed in VR and manually adjusts the depth of the endpoints [JK16]. However, physical constraints make it challenging to lift arms and draw on a floating canvas over a 3D scene, unlike on a 2D display. Recent VR/AR drawing systems often incorporate an additional handheld input surface, such as a touch-sensitive tablet [AHKG*18, DGK*20] or the user's non-dominant hand [JZF*21] (Fig. 40). In this scenario, projection is no longer applicable; instead, direct mapping is used to lift 2D strokes drawn on a texture-mapped 3D surface into the VR/AR scene.

5. Datasets

Sketch datasets can be roughly divided into two categories based on their function: used as an input and created as an art form or for visual design. The former are utilized for computer vision tasks, while the latter are studied in research on visual creation understanding and assistance. As demonstrated in Fig. 41, sketches drawn as an input for recognition, retrieval, and other classical computer vision tasks, tend to be abstract and symbolic. Similarly, most sketches drawn for 3D shape generation are restricted to single objects. For instance, QuickDraw [HE18], a popular dataset for retrieval, consists of abstract sketches drawn by users under 20 seconds, typically with a mouse. In comparison, sketches drawn as an art form or for visual design tend to be less abstract (right, Fig. 41). Such sketches are often created by professional artists and designers on digital tablets and often are more complex and refined. Drawing, as well as annotating many of these sketches requires domain-specific knowledge. As a result, datasets in this category tend to be smaller (hundreds to thousands) compared to sketches as input (thousands to millions). In this survey, we focus on datasets with sketches as an art form or for visual design and only briefly mention well-known machine learning sketch datasets as they can serve as out-of-distribution test sets (see the survey by Xu et al. [XHY*22] for a comprehensive list). The detailed dataset statistics are summarized in Table 2.

Compared to more widespread representations, such as videos and photos, sketches are harder to create and collect. As identified by Xu et al. [XHY*22], challenges in collecting or recording sketches are due to their (1) time-sequence nature: while raster sketches are abundant online, they often lack crucial temporal information; (2) missing demographic information: meta-data about the artists may not always be available, which is crucial to the fairness and bias-mitigation of the sketch-based applications. Furthermore, vector sketches have an additional format challenge: vector sketches are frequently stored in SVG format that does not support timestamps and varying stroke thickness, posing challenges to sketch data collection and sharing. Finally, sketches as an art form require an advanced expertise: as visual creation applications target professional artists, it is vital to use data created and annotated by professionals, preventing the usage of crowdsourcing platforms.

Raster sketch data can be used to understand the creation process on the pixel level and to train raster-to-raster learning based methods [SII18a, SII18b, SISSI18, ZLSS*21]. Cole et al. [CGL*08] prompt professional artists with 3D object renderings and then ask the artists to manually register their freehand raster sketches to the 3D renderings. They focus on measuring pixel overlap to understand where people draw lines when depicting a 3D object. In later papers inspired by this work, people turn to collect vector sketches and study them on the stroke level. As mentioned in Sec. 2.4, the majority of production drawing software is raster based, which motivates raster-to-raster methods. Danbooru [AcB21] is an anime image dataset with sketches assigned with tags that can directly be used for image recognition tasks. Based on Danbooru, DanbooRe-

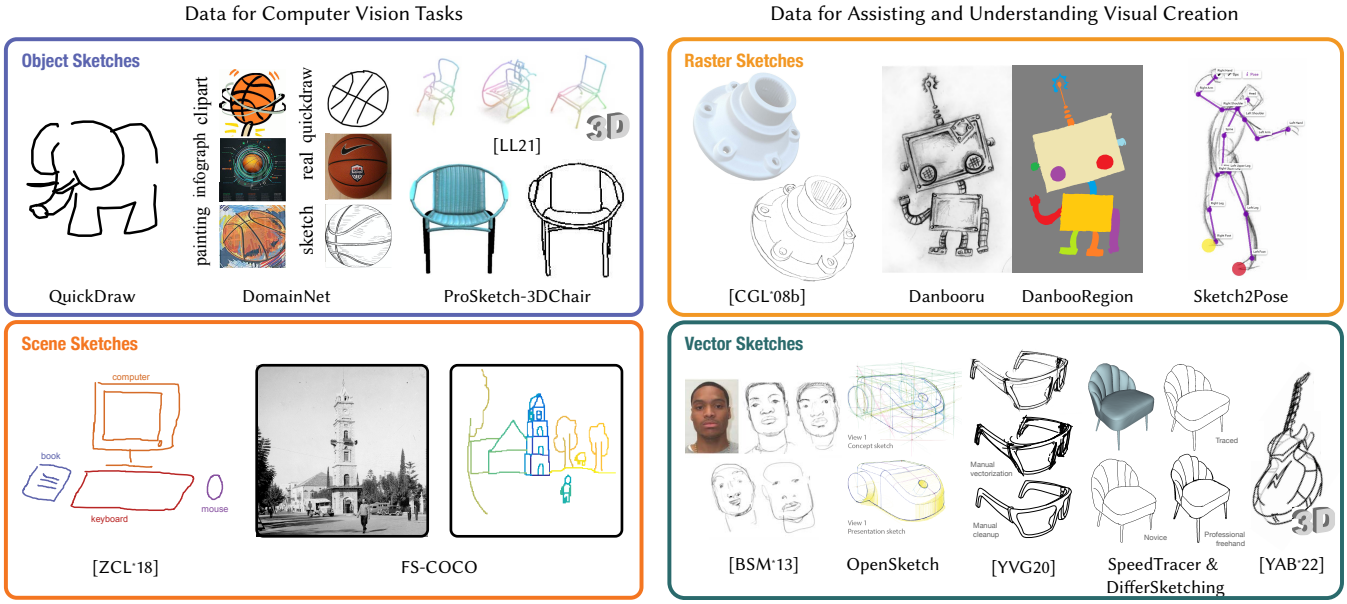


Figure 41: *Sketch data can be roughly categorized based on their function: sketches used as input for computer vision tasks (left) and sketches created as an art form or for visual design (right). The former tend to be abstract and symbolic, such as QuickDraw sketches drawn by users in under 20 seconds [HE18]. The latter are typically more complex and application-specific, such as sketches accompanied by 2D character skeletons [BB22] or scaffold lines [GSH*19].*

Table 2: *Summary of sketch datasets. "Gr." stands for "Granularity," where "o," "c," and "s" represent "object," "character," and "scene," respectively. "Vect" indicates whether sketches are stored in vector format, and "Time" indicates whether temporal information is available. "K" and "M" denote "thousand" and "million." Under "Creator," "P" and "N" stand for "Professional" and "Novice." "Repetition" shows how many different responses are collected per prompt; depending on the dataset, "prompt" could refer to a category (listed as "/cat") or a single reference photo or rendering. "✓" indicates "Yes", while a blank space in the corresponding column indicates "No."*

Dataset	Gr.	Subject	Vect	Time	# Sketches	Creator	Repetition	Public	Remark
Cole et al. [CGL*08]	o	3D models			208	P	5-10	✓	3D registration of 12 objects
Danbooru [AcB21]	c	Anime characters			4.23M	P+N	1	✓	130M tags
DanbooRegion [ZJL20]	c	Anime characters			5,377	P	≥3	✓	Region annotations
Sketch2Pose [BB22]	c	Cartoon characters			4,631	P+ N	18	✓	14,462 2D character skeletons
Berger et al. [BSM*13]	c	Protraits	✓	✓	672	P	7		24 faces at 4 abstraction levels
OpenSketch [GSH*19]	o	3D models	✓	✓	417	P	7-15	✓	3D registration of 12 objects, stroke annotations
Yan et al. [YVG20]	o, c, s	In-the-wild subjects	✓		281	P	3-5	✓	Raster-vector pairs, clean vector sketches
SpeedTracer [WQF*21]	o, c, s	In-the-wild subjects	✓	✓	1,498	P+N	11-21	✓	100 photorealistic image prompts, pressure data
DifferSketching [XSL*22]	o	Multi-view 3D renderings	✓	✓	3,620	P+N	10	✓	9 object categories
QuickDraw [HE18]	o	Common objects	✓		25.9M	N	~75k	✓	345 object categories
TU-Berlin [EHA12]	o	Common objects	✓		20K	N	~80	✓	250 object categories
DomainNet [PBX*19]	o	Common objects	✓		0.6M	P+N	220	✓	6 categories, multi-modal images including sketches
ProSketch-3DChair [ZQG*20]	o	Chairs	✓		1.5K	P	1	✓	500 3D shapes drawn by 10 artists
Zhang et al. [ZCL*18]	s	Real-world scenes	✓		332	N	1	✓	70 object categories, 7 types of scenes
FS-COCO [CSB*22]	s	Real-world scenes	✓	✓	10K	N	1-866/cat	✓	Over 92 object categories, photo-sketch pairs
SFSD [ZDL*23]	s	Real-world scenes	✓	✓	≥12K	N	141-6429/cat		40 object category, photo-sketch pairs
FrISS [KS24]	s	Real-world scenes	✓	✓	1K	N	N/A	✓	403 object category, textual descriptions
Bainbridge et al. [BHB19]	s	Real-world scenes			2,682	N	12	✓	30 scenes
Bainbridge et al. [BPEB21]	s	Indoor scenes			655	N	N/A	✓	3 scene categories
Manga109 [AFO*20]	o, c, s	Manga pages			21,142	P	1	✓	109 books by 94 authors
Luo et al. [LL21]	o	3D chairs	✓		1,497	N	1	✓	3D models from ShapeNetCore chairs [CFG*15]
Yu et al. [YAB*22]	o	Common objects	✓		62	P+N	1	✓	Created by previous papers or with commercial tools

gion [ZJL20] is a dataset with regions annotated by artists in a semi-automatic manner for flat colorization task. Sketch2Pose [BB22] collected bitmap character sketches, annotated with 2D skeletons.

In vector sketch datasets, Berger et al. [BSM*13] collect vector portrait sketches from professional artists at four abstraction levels. OpenSketch [GSH*19] is a dataset of freehand industrial design sketches in vector format created by product designers with varying expertise. Yan et al. [YVG20] collect a benchmark dataset for sketch vectorization, simplification, and cleanup tasks. SpeedTracer [WQF*21] is a dataset designed to understand the similarity

between traced sketches and freehand sketches. It contains tracings and freehand drawings by 110 participants (85 are professional). DifferSketching [XSL*22] studies the difference between novice and professional freehand sketches. It consists of sketches for 362 prompts (multi-view 3D renderings of 136 objects) created by 70 novice and 38 professional artists.

Datasets for machine learning tasks can be used for sketch processing methods. QuickDraw [HE18] and TU-Berlin [EHA12] are two popular large sketch datasets (roughly 25.9 million and 20 thousands respectively). The sketches are created in vector format given text prompts and typically in a matter of seconds. These datasets are mostly used for recognition and retrieval tasks but can be utilized as stress test inputs [YLL*22]. DomainNet [PBX*19] is a multi-modal dataset with categorized images from different domains, including raster sketches. ProSketch-3DChair [ZQG*20] is created by professional artists for sketch-based 3D shape generation. Apart from object-centric datasets, there are datasets of scene sketches. Zhang et al. [ZCL*18] collect vector sketches of simple real-world scenes consisting of annotated common objects. FS-COCO [CSB*22] is a dataset of freehand scene vector sketches created by 100 non-expert individuals with photos as reference. SFSD [ZDL*23] is a scene sketch dataset organized as photo-sketch pairs, similar to FS-COCO. FrISS [KS24] is a smaller dataset of scene vector sketches covering a larger object category set. Psychology researchers also collect smaller sketch datasets to study memory [BHB19] and the mental ability to visualize [BPEB21]. Manga109 [AFO*20] is a raster dataset of annotated manga pages. Although it is designed and annotated for manga recognition and understanding tasks, it provides example sketches with manga-specific textures, *screentone* — an important entity to manga creation applications [QWH06, TIA19].

Compared to 2D sketch data, 3D sketch data is even rarer. Luo et al. [LL21] collect the first fine-grained dataset of 1,497 3D VR sketches paired with 3D shapes, covering 1,005 chair models from ShapeNetCore with high shape diversity, contributed by 50 participants. Yu et al. [YAB*22], along with their surfacing method, curate and release a dataset of 62 3D sketches from several previous works [GSV*17, YAS*21, LCC*21, SKSK09, BBS08, KB16, XCS*14] as well as new sketches created with commercial tools.

Apart from academic datasets, some professional artists share their artwork under creative commons (CC) licenses online, providing valuable sources for professional in-the-wild sketches. Blender Art Gallery [Ble24] is a collection of free Blender projects with animation frames and drawings. These drawings are in a non-conventional vector format created with Blender Grease Pencil. Unlike the universal Scalable Vector Graphics (SVG) format, this vector format supports varying thickness along a stroke — a stroke property that is rarely considered by algorithms due to its uncommonness [YLL*22]. David Revoy [Rev24] is an artist promoting open source. He has published a large number of sketches online under CC licenses, and his work is frequently used by researchers [FTR18, YVG20]. For 3D sketches, many Tilt Brush [Goo16] and Quill [Smo16] users share their work on platforms such as Sketchfab [Ske12].

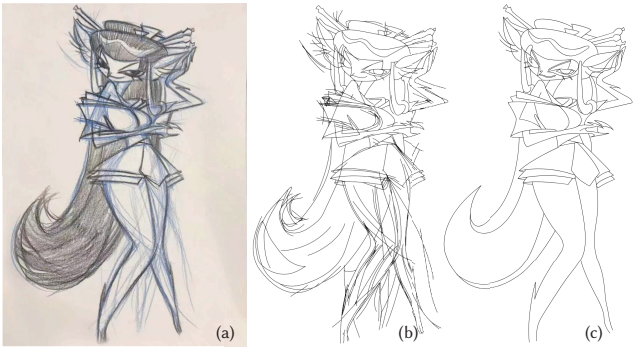


Figure 42: *Even though hatching and shading strokes are ubiquitous in real sketches (a), many sketch processing methods do not account for them, leading to suboptimal results (b, vectorized by [NHS*13]). Manual vectorization (c). Input image by Rui Hao (CC-BY-NC-4.0). Results from [YVG20].*

6. Conclusions and Directions for Future Research

Sketch processing remains an active research area for advancing our understanding of and assisting visual creation, enabling various applications ranging from shape retrieval to animation and modelling. By outlining and discussing its core challenges and providing a summary of the existing literature, we aim to inspire future research into both practical applications and the fundamental principles of sketch processing.

One necessary direction of future work is classification of strokes into textural, shading strokes, including hatching, occluding contours, and other types. Shading and textural strokes are omnipresent in many if not most natural sketches, and yet numerous sketch processing methods assume the input drawings are free of those kinds of strokes (Fig. 42). While collecting datasets for such classification task may be costly, it will benefit numerous downstream frameworks, including 3D reconstruction.

Another vast direction of potential future work is expanding the set of benchmarks for sketch processing. Currently, the main benchmark [YVG20] spans only a subset of tasks, namely vectorization, simplification, and cleanup. They also use purely geometrical error metrics, such as Chamfer distance, while perceptual error metrics might be more relevant for some applications.

One interesting direction is developing a sketch representation that can support a variable number of strokes, both open and closed, in a differentiable manner. Typical explicit representations, such as a set of parametric curves, necessarily require a fixed number of strokes or complex sequence architectures to train, such as RNN or Transformers. Both signed and unsigned distance fields combined with extraction algorithms, such as marching cubes or dual contouring, differentiably represent topological changes and thus neither require nor control the number of strokes, but are best-suited for closed curves without intersections. For a set of closed curves, this has been explored [MCR23]; for a variable number of open as well as closed curves this remains an active area of research. Open strokes or strokes with intersections can be represented as level set of non-differentiable functions, but extracting those level sets might

be challenging [YLA*24]. An alternative is to represent stroke outline, but then extracting centerline is non-trivial, as discussed in Sec. 3.3 (Fig. 13).

Reconstructing perceived foreground and perceptually correct deformations in 3D sketches in general is a new scarcely explored area. For instance, one of the typical challenges when sketching in VR is the absence of surfaces occluding some strokes. Without occluding surfaces, all strokes are visible, making the whole sketching experience difficult and messy. While one way of occluding strokes might be a full surface reconstruction, the main challenge is that this has to be done and updated in real time as an artist adds new strokes. One could imagine drawing inspiration from point cloud–based visibility methods [KT15], yet unlike point clouds, 3D sketches typically do not densely sample the intended surface. There are many other open problems in 3D sketches, for instance, finding loops in rough 3D sketches, similar to [ZZCJ13a], would significantly simplify modern 3D surface reconstruction from 3D sketches.

One general underexplored area is modeling human perception of sketches, for instance modeling how artists omit or simplify some strokes, leading to abstraction, or quantifying and undoing how artists distort proportions and layouts. There are some approaches for modelling how artists select which lines to draw given a 3D model [LNHK20] or how they stylize [LFHK21], but these works do not explore the artistic choices of overdrawing, distortions, and layout. Progress in this area is crucial for improving the quality of non-photorealistic renders of 3D models in a style of a sketch and thus narrowing the domain gap between renders and real sketches. Sketch abstraction in general has also been explored in some limited scenarios. For instance, there are early approaches to sketch abstraction that predict which strokes can be omitted [MYS*18], but they do not shed light on the perceptual significance of strokes, and do not generalize to simplifying the stroke shapes. Alternatively, some works like [VACOS23] convert an image into a sketch with a given level of abstraction, but do not connect the strokes with a particular 3D model. Understanding and undoing distortions in object proportions, as well as scene layouts, would be crucial for beautification and 3D reconstruction systems.

Many sketch processing tasks face the challenge of limited high-quality data, as discussed in Sec. 5. Researchers typically approach this issue by incorporating unlabelled data [SII18a] and utilizing data synthesis [SII18b, GZH*19, MKDM22, DZL*24]. However, as shown by Zhong et al. [ZQG*20], systems built entirely on synthetically rendered or novice-created data often fall short of meeting the practical needs of professional artists. Recently, pretrained large text-based models like CLIP [RKH*21] and text-to-image models have emerged as high-quality priors for visual tasks. This trend motivates new research in sketch generation and editing using neural priors, such as CLIPasso [VPB*22] and DiffSketcher [XWZ*23]. Leveraging these neural priors for sketch processing tasks, particularly within professional visual creation contexts, presents an exciting direction for future research.

Acknowledgment

We thank Emilie Yu for her valuable suggestions on the sections covering 3D sketching, Thor Vestergaard Christiansen and Abhishek Madan for their careful proofreading, and other DGP members at the University of Toronto for their support throughout the project. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) under Grant No.: RGPIN-2019-05097 (“Creating Virtual Shapes via Intuitive Input”) and RGPIN-2024-04968 (“Modelling and animation via intuitive input”), the NSERC - Fonds de recherche du Québec - Nature et technologies (FRQNT) NOVA Grant No. 314090., and Arts & Science Postdoctoral Fellowship at the University of Toronto.

References

[AcB21] ANONYMOUS, COMMUNITY D., BRANWEN G.: Danbooru2020: A large-scale crowdsourced & tagged anime illustration dataset. https://gwern.net/Danbooru2020, January 2021. 2024-07-31. URL: https://gwern.net/Danbooru2020. 22, 23

[ADN*17] ARORA R., DAROLIA I., NAMBOODIRI V. P., SINGH K., BOUSSEAU A.: Sketchsoup: Exploratory ideation using design sketches. *Computer Graphics Forum* (2017). 10

[Ado] ADOBE INC.: Adobe illustrator. URL: https://adobe.com/products/illustrator. 11

[AFO*20] AIZAWA K., FUJIMOTO A., OTSUBO A., OGAWA T., MATSUI Y., TSUBOTA K., IKUTA H.: Building a manga dataset “manga109” with annotations for multimedia applications. *IEEE multimedia 27*, 2 (2020), 8–18. 23, 24

[AHKG*18] ARORA R., HABIB KAZI R., GROSSMAN T., FITZMAURICE G., SINGH K.: Symbiosissketch: Combining 2d & 3d sketching for designing detailed 3d objects in situ. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018), pp. 1–15. 22

[AJA12] ABBASINEJAD F., JOSHI P., AMENTA N.: Surface patches from unorganized space curves. In *Proceedings of the twenty-eighth annual symposium on Computational geometry* (2012), pp. 417–418. 20

[AMW*23] ARORA R., MACHUCA M. D. B., WACKER P., KEEFE D., ISRAEL J. H.: Introduction to 3d sketching. In *Interactive Sketch-Based Interfaces and Modelling for Design*. River Publishers, 2023, pp. 151–177. 2, 3, 15

[AS11] ANDRE A., SAITO S.: Single-view sketch based modeling. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2011), pp. 133–140. 21

[AS21] ARORA R., SINGH K.: Mid-Air Drawing of Curves on 3D Surfaces in Virtual Reality, Mar. 2021. arXiv:2009.09029 [cs]. 16

[Bau94] BAUDEL T.: A mark-based interaction paradigm for free-hand drawing. In *Proc. UIST* (1994), pp. 185–192. 18

[BB22] BRODT K., BESSMELTSEV M.: Sketch2pose: Estimating a 3d character pose from a bitmap sketch. *ACM Transactions on Graphics 41*, 4 (7 2022). 2, 4, 23

[BB24] BRODT K., BESSMELTSEV M.: Skeleton-driven inbetweening of bitmap character drawings. *ACM Transactions on Graphics (TOG) 43*, 6 (2024). 9, 10

[BBP09] BELLETTINI G., BEORCHIA V., PAOLINI M.: Completion of visible contours. *SIAM Journal on Imaging Sciences 2*, 3 (2009), 777–799. 10

[BBS08] BAE S.-H., BALAKRISHNAN R., SINGH K.: Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *Proc. UIST* (2008), pp. 151–160. 18, 22, 24

[BBS09] BAE S.-H., BALAKRISHNAN R., SINGH K.: Everybodylovessketch: 3d sketching for a broader audience. In *Proceedings of the 22nd annual ACM symposium on User interface software and technology* (2009), pp. 59–68. 22

[BC20] BHATTACHARJEE S., CHAUDHURI P.: A survey on sketch based content creation: from the desktop to virtual and augmented reality. In *Computer Graphics Forum* (2020), vol. 39, Wiley Online Library, pp. 757–780. 2, 21

[BCF*07] BARTOLO A., CAMILLERI K. P., FABRI S. G., BORG J. C., FARRUGIA P. J.: Scribbles to vectors: preparation of scribble drawings for cad interpretation. In *Proceedings of the 4th Eurographics workshop on Sketch-based interfaces and modeling* (2007), pp. 123–130. 7, 13

[BCHS20a] BOBENRIETH C., CORDIER F., HABIBI A., SEO H.: Descriptive: Interactive 3d shape modeling from a single descriptive sketch. *Computer-Aided Design 128* (2020), 102904. 10

[BCHS20b] BOBENRIETH C., CORDIER F., HABIBI A., SEO H.: Descriptive: Interactive 3d shape modeling from a single descriptive sketch. *Computer-Aided Design 128* (2020), 102904. 21

[BCV*15] BESSMELTSEV M., CHANG W., VINING N., SHEFFER A., SINGH K.: Modeling character canvases from cartoon drawings. *ACM Transactions on Graphics (TOG) 34*, 5 (2015), 1–16. 21

[BCY*21] BHUNIA A. K., CHOWDHURY P. N., YANG Y., HOSPEDALES T. M., XIANG T., SONG Y.-Z.: Vectorization and rasterization: Self-supervised learning for sketch and handwriting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 5672–5681. 12

[BF12] BAO B., FU H.: Vectorizing line drawings with near-constant line width. In *2012 19th IEEE International Conference on Image Processing* (Sept. 2012), pp. 805–808. 8, 9, 11

[BF23] BAO B., FU H.: Line drawing vectorization via coarse-to-fine curve network optimization. *Computer Graphics Forum 42*, 6 (2023), e14787. 8, 12

[BHB19] BAINBRIDGE W. A., HALL E. H., BAKER C. I.: Drawings of real-world scenes during free recall reveal detailed object and spatial information in memory. *Nature communications 10*, 1 (2019), 5. 23, 24

[Ble24] BLENDER: Blender Cloud, 2024. URL: https://cloud.blender.org/p/gallery/5b642e25bf419c1042056fc6. 24

[BLP10] BARAN I., LEHTINEN J., POPOVIĆ J.: Sketching Clothoid Splines Using Shortest Paths. *Comput. Graph. Forum 29*, 2 (2010), 655–664. 14, 15

[BPEB21] BAINBRIDGE W. A., POUNDER Z., EARDLEY A. F., BAKER C. I.: Quantifying aphantasia through drawing: Those without visual imagery show deficits in object but not spatial memory. *Cortex 135* (2021), 159–172. 23, 24

[BS19] BESSMELTSEV M., SOLOMON J.: Vectorization of Line Drawings via Polyvector Fields. *ACM Trans. Graph. 38*, 1 (Jan. 2019), 9:1–9:12. 8, 9, 11, 12

[BSM*13] BERGER I., SHAMIR A., MAHLER M., CARTER E., HODGINS J.: Style and abstraction in portrait sketching. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 1–12. 5, 23

[BTS05] BARLA P., THOLLOT J., SILLION F.: Geometric clustering for line drawing simplification. In *Proceedings of the Eurographics Symposium on Rendering* (2005). 7, 18

[BVS16] BESSMELTSEV M., VINING N., SHEFFER A.: Gesture3d: posing 3d characters via gesture drawings. *ACM Trans. Graph. 35*, 6 (Dec. 2016). 4

[BWSS12] BESSMELTSEV M., WANG C., SHEFFER A., SINGH K.: Design-driven quadrangulation of closed 3d curves. *ACM Trans. Graph. 31*, 6 (2012), 178–1. 20

[CDQM18] CHEN J., DU M., QIN X., MIAO Y.: An improved topology extraction approach for vectorization of sketchy line drawings. *Vis Comput 34*, 12 (Dec. 2018), 1633–1644. 11, 12

[CFG*15] CHANG A. X., FUNKHOUSER T., GUIBAS L., HANRAHAN P., HUANG Q., LI Z., SAVARESE S., SAVVA M., SONG S., SU H., XIAO J., YI L., YU F.: *ShapeNet: An Information-Rich 3D Model Repository*. Tech. Rep. arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015. 23

[CGBG13] CHEN J., GUENNEBAUD G., BARLA P., GRANIER X.: Non-oriented mls gradient fields. *Computer Graphics Forum 32*, 8 (2013), 98–109. 7, 8, 17

[CGL*08] COLE F., GOLOVINSKIY A., LIMPAECHER A., BARROS H. S., FINKELSTEIN A., FUNKHOUSER T., RUSINKIEWICZ S.: Where do people draw lines? *ACM Trans. Graph. 27*, 3 (Aug. 2008). URL: https://doi.org/10.1145/1360612.1360687, doi:10.1145/1360612.1360687. 4, 5, 10, 22, 23

[CGL12] CHEEMA S., GULWANI S., LAVIOLA J.: QuickDraw: Improving drawing experience for geometric diagrams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (May 2012), CHI '12, Association for Computing Machinery, pp. 1037–1064. 15

[CKX*08] CHEN X., KANG S. B., XU Y.-Q., DORSEY J., SHUM H.-Y.: Sketching reality: Realistic interpretation of architectural designs. *ACM Transactions on Graphics (TOG) 27*, 2 (2008), 1–15. 7, 20

[CLMP15] CHEN J., LEI Q., MIAO Y., PENG Q.: Vectorization of line drawing image based on junction analysis. *Sci. China Inf. Sci. 58*, 7 (July 2015), 1–14. 7, 9, 11

[Clo71] CLOWES M. B.: On seeing things. *Artificial intelligence 2*, 1 (1971), 79–116. 20

[CLT08] CAO L., LIU J., TANG X.: What the back of the object looks like: 3d reconstruction from line drawings without hidden lines. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 3 (2008), 507–517. 10, 20

[CMV17] CARVALHO L., MARROQUIM R., VITAL BRAZIL E.: Dilight: Digital light table – inbetweening for 2d animations using guidelines. *Computers & Graphics 65* (2017), 31–44. 10

[CMZ*99] COHEN J. M., MARKOSIAN L., ZELEZNIK R. C., HUGHES J. F., BARZEL R.: An interface for sketching 3d curves. In *Proceedings of the 1999 symposium on Interactive 3D graphics* (1999), pp. 17–21. 21

[Coo08] COOPER M.: A rich discrete labeling scheme for line drawings of curved objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence 30*, 4 (2008), 741–745. 20

[CPVC19] COMPANY P., PLUMED R., VARLEY P. A. C., CAMBA J. D.: Algorithmic Perception of Vertices in Sketched Drawings of Polyhedral Shapes. *ACM Trans. Appl. Percept. 16*, 3 (Aug. 2019), 18:1–18:19. 19

[CS07] CORDIER F., SEO H.: Free-form sketching of self-occluding objects. *IEEE Computer Graphics and Applications 27*, 1 (2007), 50–59. doi:10.1109/MCG.2007.8. 10

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. 905–914. 9

[CSB*22] CHOWDHURY P. N., SAIN A., BHUNIA A. K., XIANG T., GRYADITSKAYA Y., SONG Y.-Z.: Fs-coco: Towards understanding of freehand sketches of common objects in context. In *European conference on computer vision* (2022), Springer, pp. 253–270. 23, 24

[CSMS13] CORDIER F., SEO H., MELKEMI M., SAPIDIS N. S.: Inferring mirror symmetric 3D shapes from sketches. *CAD Computer Aided Design 45*, 2 (2013), 301–311. 21

[CSSaJ05] CHERLIN J. J., SAMAVATI F., SOUSA M. C., A. JORGE J.: Sketch-based modeling with few strokes. *Proceedings of the 21st spring conference on Computer graphics - SCCG '05 1*, 212 (2005), 137. 4

[CTFZ22] CHEN Z., TAGLIASACCHI A., FUNKHOUSER T., ZHANG H.: Neural dual contouring. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–13. 13

[DACJH13] DE ARAÚJO B. R., CASIEZ G., JORGE J. A., HACHET M.: Mockup Builder: 3D modeling on and above the surface. *Computers & Graphics 37*, 3 (May 2013), 165–178. 15

[dB94] DI BAJA G. S.: Well-shaped, stable, and reversible skeletons from the (3, 4)-distance transform. *Journal of visual communication and image representation 5*, 1 (1994), 107–115. 8

[DCLB19] DELANOY J., COEURJOLLY D., LACHAUD J.-O., BOUSSEAU A.: Combining voxel and normal predictions for multi-view 3d sketching. *Computers & Graphics 82* (2019), 65–72. 10

[DCP17] DONATI L., CESANO S., PRATI A.: An Accurate System for Fashion Hand-Drawn Sketches Vectorization. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)* (Oct. 2017), pp. 2280–2286. 9, 11, 12

[DCP19] DONATI L., CESANO S., PRATI A.: A complete hand-drawn sketch vectorization framework. *Multimed Tools Appl 78*, 14 (July 2019), 19083–19113. 9, 11, 12

[DFRS03] DECARLO D., FINKELSTEIN A., RUSINKIEWICZ S., SANTELLA A.: Suggestive contours for conveying shape. *ACM Trans. Graph. 22*, 3 (July 2003), 848–855. URL: https://doi.org/10.1145/882262.882354, doi:10.1145/882262.882354. 5

[DGK*20] DREY T., GUGENHEIMER J., KARLBAUER J., MILO M., RUKZIO E.: Vrsketchin: Exploring the design space of pen and tablet interaction for 3d sketching in virtual reality. In *Proceedings of the 2020 CHI conference on human factors in computing systems* (2020), pp. 1–14. 22

[DL15] DELAYE A., LEE K.: A flexible framework for online document segmentation by pairwise stroke distance learning. *Pattern Recognition 48*, 4 (2015), 1197–1210. 9

[DLKS18] DVOROZNAK M., LI W., KIM V. G., SYKORA D.: ToonSynth: Example-based synthesis of hand-colored cartoon animations. *ACM Transactions on Graphics 37*, 4 (2018). 10

[DPS15] DE PAOLI C., SINGH K.: Secondskin: sketch-based construction of layered 3d models. *ACM Transactions on Graphics (TOG) 34*, 4 (2015), 1–10. 22

[DRvdP14] DALSTEIN B., RONFARD R., VAN DE PANNE M.: Vector graphics complexes. *ACM Trans. Graph. 33*, 4 (July 2014). 3

[DRvdP15] DALSTEIN B., RONFARD R., VAN DE PANNE M.: Vector graphics animation with time-varying topology. *ACM Trans. Graph. 34*, 4 (July 2015). 3

[DSC*20] DVOROZNÁK M., SÝKORA D., CURTIS C., CURLESS B., SORKINE-HORNUNG O., SALESIN D.: Monster mash: a single-view approach to casual 3d modeling and animation. *ACM Transactions on Graphics (ToG) 39*, 6 (2020), 1–12. 1, 2, 9, 10

[DVPSH14] DIAMANTI O., VAXMAN A., PANOZZO D., SORKINE-HORNUNG O.: Designing n-polyvector fields with complex polynomials. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 1–11. 7

[DXS*07] DORSEY J., XU S., SMEDRESMAN G., RUSHMEIER H., MCMILLAN L.: The mental canvas: A tool for conceptual architectural design and analysis. In *15th Pacific Conference on Computer Graphics and Applications (PG'07)* (2007), IEEE, pp. 201–210. 22

[DYDZ22] DENG B., YAO Y., DYKE R. M., ZHANG J.: A survey of non-rigid 3d registration. *Computer Graphics Forum 41*, 2 (2022), 559–589. 10

[DYH*21] DAS A., YANG Y., HOSPEDALES T., XIANG T., SONG Y.-Z.: Cloud2curve: Generation and vectorization of parametric sketches, 2021. URL: https://arxiv.org/abs/2103.15536, arXiv:2103.15536. 8, 12

[DZL*24] DAI Y., ZHOU S., LI Q., LI C., LOY C. C.: Learning inclusion matching for animation paint bucket colorization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2024), pp. 25544–25553. 19, 25

[EBC*15] ENTEM E., BARTHE L., CANI M.-P., CORDIER F., VAN DE PANNE M.: Modeling 3d animals from a side-view sketch. *Computers & Graphics 46* (2015), 221–230. Shape Modeling International 2014. 10

[EHA12] EITZ M., HAYS J., ALEXA M.: How do humans sketch objects? *ACM Transactions on graphics (TOG) 31*, 4 (2012), 1–10. 23, 24

[EPB*19] ENTEM E., PARAKKAT A. D., BARTHE L., MUTHUGANAPATHY R., CANI M.-P.: Automatic structuring of organic shapes from a single drawing. *Comput. Graph. 81*, C (June 2019), 125–139. URL: https://doi.org/10.1016/j.cag.2019.04.006, doi:10.1016/j.cag.2019.04.006. 10

[ERB*12] EITZ M., RICHTER R., BOUBEKEUR T., HILDEBRAND K., ALEXA M.: Sketch-based shape retrieval. *ACM Transactions on Graphics 31*, 4 (July 2012), 1–10. 2

[ES08] EISSEN K., STEUR R.: *Sketching: Drawing Techniques for Product Designers*. Bis Publishers, 2008. 5

[EVA*20] EGIAZARIAN V., VOYNOV O., ARTEMOV A., VOLKHONSKIY D., SAFIN A., TAKTASHEVA M., ZORIN D., BURNAEV E.: Deep vectorization of technical drawings. In *European Conference on Computer Vision* (2020), Springer, pp. 582–598. 12

[FASS16] FISER J., ASENTE P., SCHILLER S., SÝKORA D.: Advanced drawing beautification with ShipShape. *Computers & Graphics 56* (2016), 46–58. 2, 15

[FLB16] FAVREAU J.-D., LAFARGE F., BOUSSEAU A.: Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 1–10. 8, 11, 13, 14

[Fri08] FRISKEN S. F.: Efficient curve fitting. *Journal of Graphics Tools 13*, 2 (2008), 37–54. 14

[FTR18] FOUREY S., TSCHUMPERLÉ D., REVOY D.: A fast and efficient semi-guided algorithm for flat coloring line-arts. In *International Symposium on Vision, Modeling and Visualization* (2018). 18, 19, 24

[FZLM11] FU H., ZHOU S., LIU L., MITRA N. J.: Animated construction of line drawings. In *Proceedings of the 2011 SIGGRAPH Asia Conference* (2011), pp. 1–10. 6

[GCR13] GUAY M., CANI M.-P., RONFARD R.: The Line of Action: An Intuitive Interface for Expressive Character Posing. *ACM SIGGRAPH Asia 2013 6*, 6 (2013), 8. 2

[GHB*23] GUȚAN O., HEGDE S., BERUMEN E. J., BESSMELTSEV M., CHIEN E.: Singularity-free frame fields for line drawing vectorization. *Computer Graphics Forum 42*, 5 (2023). 8, 11, 12

[GHL*20] GRYADITSKAYA Y., HAHNLEIN F., LIU C., SHEFFER A., BOUSSEAU A.: Lifting freehand concept sketches into 3d. *ACM Transactions on Graphics (Proc. SIGGRAPH Asia) 39* (12 2020). 5, 7, 21, 22

[GJ12] GRIMM C., JOSHI P.: Just drawit: A 3d sketching system. In *Proc. SBIM* (2012), pp. 121–130. 18

[GKSS05] GENNARI L., KARA L. B., STAHOVICH T. F., SHIMADA K.: Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics 29*, 4 (2005), 547–562. 9

[Goo16] GOOGLE LLC: Tilt brush, 2016. 24

[GRYF21] GUILLARD B., REMELLI E., YVERNAY P., FUA P.: Sketch2mesh: Reconstructing and editing 3d shapes from sketches. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 13023–13032. 2

[GSH*19] GRYADITSKAYA Y., SYPESTEYN M., HOFTIJZER J. W., PONT S. C., DURAND F., BOUSSEAU A.: Opensketch: a richly-annotated dataset of product design sketches. *ACM Trans. Graph. 38*, 6 (2019), 232–1. 5, 23

[GSV*17] GORI G., SHEFFER A., VINING N., ROSALES E., CARR N., JU T.: Flowrep: Descriptive curve networks for free-form design shapes. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 1–14. 24

[Gup22] GUPTILL A. L.: *Sketching and Rendering in Pencil*. New York, The Pencil Points Press, Inc., 1922. 6

[GZH*19] GUO Y., ZHANG Z., HAN C., HU W., LI C., WONG T.-T.: Deep Line Drawing Vectorization via Line Subdivision and Topology Reconstruction. *Computer Graphics Forum 38*, 7 (Oct. 2019), 81–90. 12, 25

[HE18] HA D., ECK D.: A neural representation of sketch drawings. In *ICLR 2018* (2018). 2018. 22, 23, 24

[Her20] HERTZMANN A.: Why do line drawings work? a realism hypothesis. *Perception 49*, 4 (2020), 439–451. PMID: 32126897. 4

[Her21] HERTZMANN A.: The role of edges in line drawing perception. *Perception 50*, 3 (2021), 266–275. PMID: 33706622. 4

[Her24] HERTZMANN A.: Toward a theory of perspective perception in pictures. *Journal of Vision 24*, 4 (2024), 23–23. 4

[HF99] HESS R., FIELD D.: Integration of contours: New insights. *Trends in Cognitive Sciences 3*, 12 (Dec. 1999), 480–486. 6

[HFL14] HUANG Z., FU H., LAU R. W. H.: Data-driven segmentation and labeling of freehand sketches. *ACM Trans. Graph. 33*, 6 (Nov. 2014). 9

[HGSB22] HÄHNLEIN F., GRYADITSKAYA Y., SHEFFER A., BOUSSEAU A.: Symmetry-driven 3d reconstruction from concept sketches. In *ACM SIGGRAPH 2022 Conference Proceedings* (2022), pp. 1–8. 21, 22

[HLW*16] HENNESSEY J. W., LIU H., WINNEMÖLLER H., DONTCHEVA M., MITRA N. J.: How2sketch: generating easy-to-follow tutorials for sketching 3d objects. *arXiv preprint arXiv:1607.07980* (2016). 6

[HS81] HORN B. K., SCHUNCK B. G.: Determining optical flow. *Artificial Intelligence 17*, 1 (1981), 185–203. 10

[HS88] HARRIS C., STEPHENS M.: A combined corner and edge detector. In *Proc. of Fourth Alvey Vision Conference* (1988), pp. 147–151. 7

[HT06] HILAIRE X., TOMBRE K.: Robust and accurate vectorization of line drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 6 (2006), 890–904. 8, 11

[Huf71] HUFFMAN D. A.: Impossible objects as nonsense sentences. *Machine intelligence 6* (1971), 295–323. 20

[HZH*22] HUANG Z., ZHANG T., HENG W., SHI B., ZHOU S.: Real-time intermediate flow estimation for video frame interpolation. In *Proceedings of the European Conference on Computer Vision (ECCV)* (2022). 10

[IMKT07] IGARASHI T., MATSUOKA S., KAWACHIYA S., TANAKA H.: Interactive beautification: A technique for rapid geometric design. SIGGRAPH '07, Association for Computing Machinery, p. 18–es. 15

[IMT99] IGARASHI T., MATSUOKA S., TANAKA H.: Teddy: A sketching interface for 3D freeform design. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99* (1999), 409–416. 9

[JBPS11] JACOBSON A., BARAN I., POPOVIC J., SORKINE O.: Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph* (2011). 2, 9

[JDA07] JUDD T., DURAND F., ADELSON E. H.: Apparent ridges for line drawing. *ACM Trans. Graph. 26*, 3 (2007), 19. 5

[JK16] JACKSON B., KEEFE D. F.: Lift-off: Using reference imagery and freehand sketching to create 3d models in vr. *IEEE transactions on visualization and computer graphics 22*, 4 (2016), 1442–1451. 22

[JSL21] JIANG J., SEAH H. S., LIEW H. Z.: Handling gaps for vector graphics coloring. *Vis Comput 37*, 9 (Sept. 2021), 2473–2484. 19

[JZF*21] JIANG Y., ZHANG C., FU H., CANNAVÒ A., LAMBERTI F., LAU H. Y., WANG W.: Handpainter-3d sketching in vr with hand-based physical proxy. In *Proceedings of the 2021 CHI conference on human factors in computing systems* (2021), pp. 1–13. 22

[KALB18] KIM Y., AN S.-G., LEE J. H., BAE S.-H.: Agile 3d sketching with air scaffolding. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems* (2018), pp. 1–12. 21, 22

[Kan79] KANIZSA G.: *Organization in Vision: Essays on Gestalt Perception*. Praeger Publishers, New York, 1979. 10

[KB16] KIM Y., BAE S.-H.: Sketchingwithhands: 3d sketching handheld products with first-person hand posture. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (2016), pp. 797–808. 21, 22, 24

[KH06] KARPENKO O. A., HUGHES J. F.: Smoothsketch: 3d free-form shapes from complex sketches. *ACM Trans. Graph. 25*, 3 (July 2006), 589–598. URL: https://doi.org/10.1145/1141911.1141928, doi:10.1145/1141911.1141928. 6, 10

[KHW*22] KIM B., HUANG X., WUELFROTH L., TANG J., CORDONNIER G., GROSS M., SOLENTHALER B.: Deep reconstruction of 3d smoke densities from artist sketches. In *Computer Graphics Forum* (2022), vol. 41, Wiley Online Library, pp. 97–110. 5

[KLL*23] KIM H., LEE C., LEE J., KIM D., LEE K., OH M., KIM D.: Flatgan: A holistic approach for robust flat-coloring in high-definition with understanding line discontinuity. In *Proceedings of the 31st ACM International Conference on Multimedia* (2023), pp. 8242–8250. 19

[Kof55] KOFFKA K.: *Principles of Gestalt Psychology*. Routledge \& K. Paul, 1955. 6

[Kri24] KRITA FOUNDATION: Krita colorize mask, 2024. URL: https://docs.krita.org/en/reference_manual/tools/colorize_mask.html. 19

[KS07] KARA L. B., SHIMADA K.: Sketch-based 3d-shape creation for industrial styling design. *IEEE Computer Graphics and Applications 27*, 1 (2007), 60–71. 8, 22

[KS09] KYRATZI S., SAPIDIS N.: Extracting a polyhedron from a single-view sketch: Topological construction of a wireframe sketch with minimal hidden elements. *Computers & Graphics 33*, 3 (2009), 270–279. 10, 20, 21

[KS24] KÜTÜK A., SEZGIN T. M.: Class-agnostic visio-temporal scene sketch semantic segmentation, 2024. URL: https://arxiv.org/abs/2410.00266, arXiv:2410.00266. 23, 24

[KT15] KATZ S., TAL A.: On the visibility of point clouds. In *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 1350–1358. doi:10.1109/ICCV.2015.159. 25

[Kur14] KURLIN V.: Auto-completion of contours in sketches, maps, and sparse 2d images based on topological persistence. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing* (2014), pp. 594–601. 9

[KVDKT97] KOENDERINK J. J., VAN DOORN A. J., KAPPERS A. M., TODD J. T.: The visual contour in depth. *Perception & Psychophysics 59*, 6 (1997), 828–838. 10

[KWÖG18] KIM B., WANG O., ÖZTIRELI A. C., GROSS M.: Semantic segmentation for line drawing vectorization using neural networks. In *Computer Graphics Forum* (2018), vol. 37, Wiley Online Library, pp. 329–338. 12

[KYC*17] KRS V., YUMER E., CARR N., BENES B., MECH R.: Skippy: Single view 3d curve interactive modeling. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 1–12. 22

[KZL07] KEEFE D., ZELEZNIK R., LAIDLAW D.: Drawing on Air: Input Techniques for Controlled 3D Line Illustration. *IEEE Transactions on Visualization and Computer Graphics 13*, 5 (Sept. 2007), 1067–1081. Conference Name: IEEE Transactions on Visualization and Computer Graphics. 15

[LABS23] LIU C., AOKI T., BESSMELTSEV M., SHEFFER A.: Stripmaker: Perception-driven learned vector sketch consolidation. *ACM Trans. Graph. 42*, 4 (jul 2023). 3, 6, 7, 17, 18

[Laz24] LAZYBRUSH TEAM: Lazybrush tvpaint plugin, 2024. URL: http://lazy-brush.com/. 19

[LB90] LAMB D., BANDOPADHAY A.: Interpreting a 3d object from a rough 2d line drawing. In *Proceedings of the First IEEE Conference on Visualization: Visualization90* (1990), IEEE, pp. 59–66. 7

[LCC*21] LIU L., CHEN N., CEYLAN D., THEOBALT C., WANG W., MITRA N. J.: Curvefusion: Reconstructing thin structures from rgbd sequences. *arXiv preprint arXiv:2107.05284* (2021). 24

[LCY*11] LIU D., CHEN Q., YU J., GU H., TAO D., SEAH H. S.: Stroke correspondence construction using manifold learning. In *Computer Graphics Forum* (2011), vol. 30, Wiley Online Library, pp. 2194–2207. 10

[LFHK21] LIU D., FISHER M., HERTZMANN A., KALOGERAKIS E.: Neural strokes: Stylized line drawing of 3d shapes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2021), pp. 14204–14213. 25

[LFT14] LIU J., FU H., TAI C.-L.: Dynamic sketching: Simulating the process of observational drawing. In *Proceedings of the Workshop on Computational Aesthetics* (2014), pp. 15–22. 6

[LFT18] LI L., FU H., TAI C.-L.: Fast sketch segmentation and labeling with deep learning. *IEEE computer graphics and applications 39*, 2 (2018), 38–51. 9

[LL21] LING LUO YULIA GRYADITSKAYA Y. Y. T. X. Y.-Z. S.: Fine-Grained VR Sketching: Dataset and Insights. In *Proceedings of International Conference on 3D Vision (3DV)* (2021). 23, 24

[LLLW22] LIU H., LI C., LIU X., WONG T.-T.: End-to-end line drawing vectorization. In *Proceedings of Thirty-Sixth AAAI Conference on Artificial Intelligence (AAAI2022)* (Feburary 2022), vol. 36, pp. 4559–4566. 8, 12

[LLMRK20] LI T.-M., LUKAC M., MICHAEL G., RAGAN-KELLEY J.: Differentiable vector graphics rasterization for editing and learning. *ACM Trans. Graph. (Proc. SIGGRAPH Asia) 39*, 6 (2020), 193:1–193:15. 3

[Llo82] LLOYD S.: Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137. 9

[LLZ*17] LI Y., LUO X., ZHENG Y., XU P., FU H.: Sweepcanvas: Sketch-based 3d prototyping on an rgb-d image. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology* (2017), pp. 387–399. 22

[LNHK20] LIU D., NABAIL M., HERTZMANN A., KALOGERAKIS E.: Neural contours: Learning to draw lines from 3d shapes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2020), pp. 5428–5436. 25

[LPS*19] LI K., PANG K., SONG Y.-Z., XIANG T., HOSPEDALES T. M., ZHANG H.: Toward deep universal sketch perceptual grouper. *IEEE Transactions on Image Processing 28*, 7 (2019), 3219–3231. 9

[LRS18] LIU C., ROSALES E., SHEFFER A.: Strokeaggregator: Consolidating raw sketches into artist-intended curve drawings. *ACM Transactions on Graphics (TOG) 37*, 4 (2018), 1–15. 1, 3, 5, 7, 8, 16, 17, 18

[LS96] LIPSON H., SHPITALNI M.: Optimization-based reconstruction of a 3d object from a single freehand line drawing. *Computer-Aided Design 8*, 28 (1996), 651–663. 21

[LWH15] LIU X., WONG T.-T., HENG P.-A.: Closure-aware sketch simplification. *ACM Trans. Graph. 34*, 6 (2015), 168. 9, 18

[LYFD12] LU J., YU F., FINKELSTEIN A., DIVERDI S.: HelpingHand: Example-based stroke stylization. *ACM Transactions on Graphics (TOG) 31*, 4 (2012), 1–10. 14

[MAS*18] MACHUCA M. D. B., ASENTE P., STUERZLINGER W., LU J., KIM B.: Multiplanes: Assisted Freehand VR Sketching. In *Proceedings of the 2018 ACM Symposium on Spatial User Interaction* (New York, NY, USA, Oct. 2018), SUI '18, Association for Computing Machinery, pp. 36–47. 15

[MCR23] MEHTA I., CHANDRAKER M., RAMAMOORTHI R.: A theory of topological derivatives for inverse rendering of geometry. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2023), pp. 419–429. 24

[MFXM21] MIYAUCHI R., FUKUSATO T., XIE H., MIYATA K.: Stroke correspondence by labeling closed areas. In *2021 Nicograph International (NicoInt)* (2021), pp. 34–41. 10

[MGW24] MO H., GAO C., WANG R.: Joint stroke tracing and correspondence for 2d animation. *ACM Transactions on Graphics 43*, 3 (2024), 1–17. 11, 13, 15, 19

[MKDM22] MANDA B., KENDRE P. P., DEY S., MUTHUGANAPATHY R.: Sketchcleannet—a deep learning approach to the enhancement and correction of query sketches for a 3d cad model retrieval system. *Computers & Graphics 107* (2022), 73–83. 17, 25

[MNB23] MYRONOVA M., NEVEU W., BESSMELTSEV M.: Differential operators on sketches via alpha contours. *ACM Trans. Graph. 42*, 4 (jul 2023). 1, 6, 9

[MS11] MCCRAE J., SINGH K.: Neatening sketched strokes using piecewise french curves. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling* (2011), pp. 141–148. 14

[MSK02] MITANI J., SUZUKI H., KIMURA F.: 3d sketch: sketch-based model reconstruction and rendering. In *From Geometric Modeling to Shape Modeling: IFIP TC5 WG5. 2 Seventh Workshop on Geometric Modeling: Fundamentals and Applications October 2–4, 2000, Parma, Italy* (2002), Springer, pp. 85–98. 21

[MSR09] MURUGAPPAN S., SELLAMANI S., RAMANI K.: Towards beautification of freehand sketches using suggestions. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (Aug. 2009), SBIM '09, Association for Computing Machinery, pp. 69–76. 15

[MSSG*21] MO H., SIMO-SERRA E., GAO C., ZOU C., WANG R.: General virtual sketching framework for vector line art. *ACM Trans. Graph. 40*, 4 (jul 2021). 8, 12, 13

[Mum94] MUMFORD D.: Elastica and computer vision. In *Algebraic Geometry and its Applications: Collections of Papers from Shreeram S. Abhyankar's 60th Birthday Conference* (1994), Springer, pp. 491–506. 10

[MYS*18] MUHAMMAD U. R., YANG Y., SONG Y.-Z., XIANG T., HOSPEDALES T. M.: Learning deep sketch abstraction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2018), pp. 8014–8023. 25

[NHS*13] NORIS G., HORNUNG A., SUMNER R. W., SIMMONS M., GROSS M.: Topology-driven vectorization of clean line drawings. *ACM Trans. Graph. 32*, 1 (Feb. 2013), 4:1–4:11. 1, 5, 6, 7, 8, 9, 11, 12, 24

[NM90] NITZBERG M., MUMFORD D. B.: The 2.1-d sketch. IEEE Computer Society Press. 3

[NSS*12] NORIS G., SỲKORA D., SHAMIR A., COROS S., WHITED B., SIMMONS M., HORNUNG A., GROSS M., SUMNER R.: Smart scribbles for sketch segmentation. In *Computer Graphics Forum* (2012), vol. 31, pp. 2516–2527. 9, 18

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P.: Ridge-valley lines on meshes via implicit surface fitting. In *ACM SIGGRAPH 2004 Papers*. 2004, pp. 609–612. 5

[OCR*19] OLIVIER P., CHABRIER R., ROHMER D., DE THOISY E., CANI M.-P.: Nested explorative maps: A new 3d canvas for conceptual design in architecture. *Computers & Graphics 82* (2019), 203–213. 22

[OK11] ORBAY G., KARA L. B.: Beautification of design sketches using trainable stroke clustering and curve fitting. *IEEE Transactions on Visualization and Computer Graphics 17* (2011), 694–708. 7, 8, 18

[OK12] ORBAY G., KARA L. B.: Sketch-based surface design using malleable curve networks. *Computers & Graphics 36*, 8 (2012), 916–929. 20

[OPP*21] OHRHALLINGER S., PEETHAMBARAN J., PARAKKAT A. D., DEY T. K., MUTHUGANAPATHY R.: 2d points curve reconstruction survey and benchmark. In *Computer Graphics Forum* (2021), vol. 40, Wiley Online Library, pp. 611–632. 2

[PBDSH13] PANOZZO D., BARAN I., DIAMANTI O., SORKINE-HORNUNG O.: Weighted averages on surfaces. *ACM Transactions on Graphics (TOG) 32*, 4 (2013), 1–12. 16

[PBG*15] PERTENEDER F., BRESLER M., GROSSAUER E.-M., LEONG J., HALLER M.: cluster: Smart clustering of free-hand sketches on large interactive surfaces. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology* (New York, NY, USA, 2015), UIST '15, Association for Computing Machinery, p. 37–46. 9

[PBX*19] PENG X., BAI Q., XIA X., HUANG Z., SAENKO K., WANG B.: Moment matching for multi-source domain adaptation. In *Proceedings of the IEEE/CVF international conference on computer vision* (2019), pp. 1406–1415. 23, 24

[PCS21] PARAKKAT A. D., CANI M.-P. R., SINGH K.: Color by numbers: Interactive structuring and vectorization of sketch imagery. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (New York, NY, USA, 2021), CHI '21, Association for Computing Machinery. 9, 13, 20

[PH08] PAULSON B., HAMMOND T.: PaleoSketch: Accurate primitive sketch recognition and beautification. In *Proceedings of the 13th International Conference on Intelligent User Interfaces* (Jan. 2008), IUI '08, Association for Computing Machinery, pp. 1–10. 14

[PLS*15] PAN H., LIU Y., SHEFFER A., VINING N., LI C.-J., WANG W.: Flow aligned surfacing of curve networks. *ACM Transactions on Graphics 34*, 4 (2015), 127:1–127:10. 10

[PMC22] PARAKKAT A. D., MEMARI P., CANI M.-P.: Delaunay painting: Perceptual image colouring from raster contours with gaps. In *Computer Graphics Forum* (2022), vol. 41, Wiley Online Library, pp. 166–181. 20

[PMKB23] PUHACHOV I., MARTENS C., KRY P., BESSMELTSEV M.: Reconstruction of machine-made shapes from bitmap sketches. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia) 42*, 6 (Dec. 2023). 2, 3, 6

[PNCB21] PUHACHOV I., NEVEU W., CHIEN E., BESSMELTSEV M.: Keypoint-driven line drawing vectorization via polyvector flow. *ACM Trans. on Graph. (Proc. of SIGGRAPH Asia) 40*, 6 (12 2021). 2, 7, 8, 9, 11, 12, 13

[PPM18] PARAKKAT A. D., PUNDARIKAKSHA U. B., MUTHUGANAPATHY R.: A delaunay triangulation based approach for cleaning rough sketches. *Computers & Graphics 74* (2018), 171–181. 9, 13, 20

[PS83] PLASS M., STONE M.: Curve-fitting with piecewise parametric cubics. *SIGGRAPH Comput. Graph. 17*, 3 (July 1983), 229–239. 8

[PvMLV*21] PAGUREK VAN MOSSEL D., LIU C., VINING N., BESSMELTSEV M., SHEFFER A.: Strokestrip: Joint parameterization and fitting of stroke clusters. *ACM Transactions on Graphics 40*, 4 (2021). 8, 17

[PVW85] PAVLIDIS T., VAN WYK C. J.: An automatic beautifier for drawings and illustrations. *SIGGRAPH Comput. Graph. 19*, 3 (July 1985), 225–234. 15

[QGXS22] QI A., GRYADITSKAYA Y., XIANG T., SONG Y.-Z.: One sketch for all: One-shot personalized sketch segmentation. *IEEE Transactions on Image Processing 31* (2022), 2673–2682. 9

[QT19] QI Y., TAN Z.-H.: Sketchsegnet+: An end-to-end learning of rnn for multi-class sketch semantic segmentation. *Ieee Access 7* (2019), 102717–102726. 9

[QWH06] QU Y., WONG T.-T., HENG P.-A.: Manga colorization. *ACM Trans. Graph. 25*, 3 (July 2006), 1214–1220. 18, 24

[QWM*23] QIU S., WANG Z., MCMILLAN L., RUSHMEIER H., DORSEY J.: Is drawing order important? *Proceedings of EUROGRAPHICS-Short Papers* (2023). 6

[Ree81] REEVES W. T.: Inbetweening for computer animation utilizing moving point constraints. *SIGGRAPH Comput. Graph. 15*, 3 (Aug. 1981), 263–269. 10

[Rev24] REVOY D.: David revoy, 2024. URL: https://www.davidrevoy.com/. 24

[RKH*21] RADFORD A., KIM J. W., HALLACY C., RAMESH A., GOH G., AGARWAL S., SASTRY G., ASKELL A., MISHKIN P., CLARK J., ET AL.: Learning transferable visual models from natural language supervision. In *International conference on machine learning* (2021), PMLR, pp. 8748–8763. 25

[Rob63] ROBERTS L. G.: *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963. 20

[Ros94] ROSIN P. L.: Grouping curved lines. In *BMVC* (1994), Citeseer, pp. 1–10. 7, 18

[RSW*07] ROSE K., SHEFFER A., WITHER J., CANI M.-P., THIBERT B.: Developable surfaces from arbitrary sketched boundaries. In *SGP'07-5th Eurographics Symposium on Geometry Processing* (2007), Eurographics Association, pp. 163–172. 20

[RTB*18] RAMOS S., TREVISAN D. F., BATAGELO H. C., SOUSA M. C., GOIS J. P.: Contour-aware 3d reconstruction of side-view sketches. *Computers & Graphics 77* (2018), 97–107. 10

[SA93] SINHA P., ADELSON E.: Recovering 3d shapes from 2d line-drawings. In *Intelligent Robotics; Proceedings of the International Symposium on Intelligent Robotics* (1993), pp. 7–9. 20

[Sah20] SAHILLIOĞLU Y.: Recent advances in shape correspondence. *The Visual Computer 36*, 8 (2020), 1705–1721. 10

[Sau03] SAUND E.: Finding perceptually closed paths in sketches and drawings. *IEEE transactions on pattern analysis and machine intelligence 25*, 4 (2003), 475–491. 7

[SBBB20] STANKO T., BESSMELTSEV M., BOMMES D., BOUSSEAU A.: Integer-grid sketch simplification and vectorization. In *Computer Graphics Forum* (2020), vol. 39. 8, 11, 13

[SC08] SHESH A., CHEN B.: Efficient and dynamic simplification of line drawings. In *Computer Graphics Forum* (2008), vol. 27, pp. 537–545. 18

[SCC24] SCRIVENER D., COLDREN E., CHIEN E.: Winding Number Features for Vector Sketch Colorization. *Computer Graphics Forum* (2024). 9, 19, 20

[SDBM17] SARVADEVABHATLA R. K., DWIVEDI I., BISWAS A., MANOCHA S.: Sketchparse: Towards rich descriptions for poorly drawn sketches using multi-task hierarchical deep networks. In *Proceedings of the 25th ACM international conference on Multimedia* (2017), pp. 10–18. 9

[SDC09] SỲKORA D., DINGLIANA J., COLLINS S.: Lazybrush: Flexible painting tool for hand-drawn cartoons. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 599–608. 18

[Sel03] SELINGER P.: Potrace: a polygon-based tracing algorithm, 2003. 11

[SGX*23] SIYAO L., GU T., XIAO W., DING H., LIU Z., LOY C. C.: Deep geometrized cartoon line inbetweening, 2023. URL: https://arxiv.org/abs/2309.16643, arXiv:2309.16643. 10

[SII18a] SIMO-SERRA E., IIZUKA S., ISHIKAWA H.: Mastering Sketching: Adversarial Augmentation for Structured Prediction. *ACM Trans. Graph. 37*, 1 (2018), 11:1–11:13. 17, 22, 25

[SII18b] SIMO-SERRA E., IIZUKA S., ISHIKAWA H.: Real-time data-driven interactive rough sketch inking. *ACM Trans. Graph. 37*, 4 (2018), 98:1–98:14. 12, 14, 17, 22, 25

[Sin02] SINGH K.: A fresh perspective. In *Proceedings of the Graphics Interface 2002 Conference* (May 2002), pp. 17–24. 4

[SISI16] SIMO-SERRA E., IIZUKA S., SASAKI K., ISHIKAWA H.: Learning to simplify: Fully convolutional networks for rough sketch cleanup. *ACM Trans. Graph. 35*, 4 (2016), 121:1–121:11. 17

[SISI17] SASAKI K., IIZUKA S., SIMO-SERRA E., ISHIKAWA H.: Joint Gap Detection and Inpainting of Line Drawings. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017), pp. 5768–5776. 19

[SISSI18] SASAKI K., IIZUKA S., SIMO-SERRA E., ISHIKAWA H.: Learning to restore deteriorated line drawing. *The visual computer 34* (2018), 1077–1085. 22

[SKC*14] SYKORA D., KAVAN L., CADIK M., JAMRISKA O., JACOBSON A., WHITED B., SIMMONS M., SORKINE-HORNUNG O.: Ink-and-Ray: Bas-Relief Meshes for Adding Global Illumination Effects to Hand-Drawn Characters. *ACM Transactions on Graphics 33*, 2 (2014), 1–15. 9

[Ske12] SKETCHFAB INC.: Sketchfab, 2012. URL: https://sketchfab.com/. 24

[SKKS09] SCHMIDT R., KHAN A., KURTENBACH G., SINGH K.: On expert performance in 3D curve-drawing tasks. *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling - SBIM '09 1* (2009), 133. 4

[SKSK09] SCHMIDT R., KHAN A., SINGH K., KURTENBACH G.: Analytic drawing of 3D scaffolds. *ACM Transactions on Graphics 28*, 5 (2009), 1. 22, 24

[SLWF14] SU Q., LI W. H. A., WANG J., FU H.: Ez-sketching: Three-level optimization for error-tolerant image tracing. *ACM Transactions on Graphics (TOG) 33*, 4 (2014), 1–9. 14

[Smo16] SMOOTHSTEP LLC: Quill, 2016. 24

[SS14] SADRI B., SINGH K.: Flow-complex-based shape reconstruction from 3d curves. *ACM Transactions on Graphics (TOG) 33*, 2 (2014), 1–15. 20

[ST16] SCHNEIDER R. G., TUYTELAARS T.: Example-based sketch segmentation and labeling using crfs. *ACM Trans. Graph. 35*, 5 (July 2016). 9

[Sut98] SUTHERLAND I. E.: *Sketchpad—a man-machine graphical communication system*. Association for Computing Machinery, New York, NY, USA, 1998, p. 391–408. 14

[Sze10] SZELISKI R.: *Computer Vision: Algorithms and Applications*, 1st ed. Springer-Verlag New York, Inc., New York, NY, USA, 2010. 7, 8

[SZL*23] SMITH H. J., ZHENG Q., LI Y., JAIN S., HODGINS J. K.: A method for animating children's drawings of the human figure. *ACM Trans. Graph. 42*, 3 (June 2023). 9

[TIA19] TSUBOTA K., IKAMI D., AIZAWA K.: Synthesis of screentone patterns of manga characters. In *2019 IEEE international symposium on multimedia (ISM)* (2019), IEEE, pp. 212–2123. 24

[TSB11] THIEL Y., SINGH K., BALAKRISHNAN R.: Elasticurves: Exploiting stroke dynamics and inertia for the real-time neatening of sketched 2D curves. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology* (Oct. 2011), UIST '11, Association for Computing Machinery, pp. 383–392. 13, 14, 16

[Ull76] ULLMAN S.: Filling-in the gaps: The shape of subjective contours and a model for their generation. *Biological Cybernetics 25*, 1 (1976), 1–6. 10

[VACOS23] VINKER Y., ALALUF Y., COHEN-OR D., SHAMIR A.: Clipascene: Scene sketching with different types and levels of abstraction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (2023), pp. 4146–4156. 25

[VKZHCO11] VAN KAICK O., ZHANG H., HAMARNEH G., COHEN-OR D.: A survey on shape correspondence. In *Computer graphics forum* (2011), vol. 30, Wiley Online Library, pp. 1681–1707. 10

[VMS05] VARLEY P., MARTIN R., SUZUKI H.: Frontal geometry from sketches of engineering objects: is line labelling necessary? *Computer-Aided Design 37*, 12 (2005), 1285–1307. 3, 10, 21

[VPB*22] VINKER Y., PAJOUHESHGAR E., BO J. Y., BACHMANN R. C., BERMANO A. H., COHEN-OR D., ZAMIR A., SHAMIR A.: Clipasso: Semantically-aware object sketching. *ACM Transactions on Graphics (TOG) 41*, 4 (2022), 1–11. 25

[Wal75] WALTZ D. L.: Understanding line drawings of scenes with shadows. *The psychology of computer vision* (1975), 19–91. 20

[WCC*11] WALTHER D. B., CHAI B., CADDIGAN E., BECK D. M., FEI-FEI L.: Simple line drawings suffice for functional mri decoding of natural scene categories. *Proceedings of the National Academy of Sciences 108*, 23 (2011), 9661–9666. 4

[Wer38] WERTHEIMER M.: Laws of organization in perceptual forms. 6

[Wil94] WILLIAMS L. R.: *Topological Reconstruction of a Smooth Manifold-Solid From Its Occluding Contour*. Tech. rep., USA, 1994. 6

[WJ97] WILLIAMS L. R., JACOBS D. W.: Stochastic completion fields: A neural model of illusory contour shape and salience. *Neural computation 9*, 4 (1997), 837–858. 10

[WL24] WANG J., LI C.: Contextseg: Sketch semantic segmentation by querying the context with attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2024), pp. 3679–3688. 9

[WNS*10] WHITED B., NORIS G., SIMMONS M., SUMNER R. W., GROSS M., ROSSIGNAC J.: Betweenit: An interactive tool for tight inbetweening. *Computer Graphics Forum 29*, 2 (2010), 605–614. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2009.01630.x, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2009.01630.x, doi:https://doi.org/10.1111/j.1467-8659.2009.01630.x. 1, 2, 11

[WPL06] WANG W., POTTMANN H., LIU Y.: Fitting b-spline curves to point clouds by curvature-based squared distance minimization. *ACM Trans. Graph. 25*, 2 (Apr. 2006), 214–238. 8

[WQF*21] WANG Z., QIU S., FENG N., RUSHMEIER H., MCMILLAN L., DORSEY J.: Tracing versus freehand for evaluating computer-generated drawings. *ACM Transactions on Graphics (TOG) 40*, 4 (2021), 1–12. 5, 6, 23

[WQLY18] WU X., QI Y., LIU J., YANG J.: Sketchsegnet: A rnn model for labeling sketch strokes. In *2018 IEEE 28th International Workshop on Machine Learning for Signal Processing (MLSP)* (2018), IEEE, pp. 1–6. 9

[WY09] WANG S., YU S.-H.: Endpoint fusing of freehand 3d object sketch with hidden-part-draw. *2009 IEEE 10th International Conference on Computer-Aided Industrial Design & Conceptual Design* (2009), 586–590. 19

[WZW*20] WANG S., ZHANG Q., WANG S., JING X., GAO M.: Endpoint fusing method of online freehand-sketched polyhedrons. *Vis Comput 36*, 2 (Feb. 2020), 291–303. 19

[XCS*14] XU B., CHANG W., SHEFFER A., BOUSSEAU A., MCCRAE J., SINGH K.: True2form: 3d curve networks from 2d sketches via selective regularization. *Transactions on Graphics (Proc. SIGGRAPH 2014) 33*, 4 (2014). 5, 10, 21, 24

[XFZ*18] XU P., FU H., ZHENG Y., SINGH K., HUANG H., TAI C.-L.: Model-guided 3d sketching. *IEEE Transactions on Visualization and Computer Graphics 25*, 10 (2018), 2927–2939. 22

[XHY*22] XU P., HOSPEDALES T. M., YIN Q., SONG Y.-Z., XIANG T., WANG L.: Deep learning for free-hand sketch: A survey. *IEEE transactions on pattern analysis and machine intelligence 45*, 1 (2022), 285–312. 2, 22

[XSL*22] XIAO C., SU W., LIAO J., LIAN Z., SONG Y.-Z., FU H.: Differsketching: How differently do people sketch 3d objects? *ACM Transactions on Graphics (TOG) 41*, 6 (2022), 1–16. 23, 24

[XWZ*23] XING X., WANG C., ZHOU H., ZHANG J., YU Q., XU D.: Diffsketcher: Text guided vector sketch synthesis through latent diffusion models. *Advances in Neural Information Processing Systems 36* (2023), 15869–15889. 25

[XXM*19] XU X., XIE M., MIAO P., QU W., XIAO W., ZHANG H., LIU X., WONG T.-T.: Perceptual-aware sketch simplification based on integrated vgg layers. *IEEE Transactions on Visualization and Computer Graphics* (2019). 17

[YAB*22] YU E., ARORA R., BÆRENTZEN J. A., SINGH K., BOUSSEAU A.: Piecewise-smooth surface fitting onto unstructured 3D sketches. *ACM Transactions on Graphics 41*, 4 (July 2022), 1–16. 6, 10, 23, 24

[YAS*21] YU E., ARORA R., STANKO T., BAERENTZEN J. A., SINGH K., BOUSSEAU A.: Cassie: Curve and surface sketching in immersive environments. In *Conference on Human Factors in Computing Systems - Proceedings* (2021). 16, 20, 24

[YBS*12] YU J., BIAN W., SONG M., CHENG J., TAO D.: Graph based transductive learning for cartoon correspondence construction. *Neurocomputing 79* (2012), 105–114. 10, 11

[YCY*22] YAN C., CHUNG J. J. Y., YOON K., GINGOLD Y., ADAR E., HONG S. R.: FlatMagic: Improving flat colorization through ai-driven design for digital comic professionals. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (2022), CHI. 2, 18, 19

[YDSG21] YU X., DIVERDI S., SHARMA A., GINGOLD Y.: ScaffoldSketch: Accurate Industrial Design Drawing in VR. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (New York, NY, USA, Oct. 2021), UIST '21, Association for Computing Machinery, pp. 372–384. 16, 22

[YLA*24] YAN C., LI Y., ANEJA D., FISHER M., SIMO-SERRA E., GINGOLD Y.: Deep sketch vectorization via implicit surface extraction. *ACM Transactions on Graphics (TOG) 43*, 4 (2024), 1–13. 2, 7, 8, 12, 13, 14, 25

[YLGF23] YU D., LAU M., GAO L., FU H.: Sketch beautification: Learning part beautification and structure refinement for sketches of man-made objects. *IEEE Transactions on Visualization and Computer Graphics* (2023). 14, 15

[YLL*22] YIN J., LIU C., LIN R., VINING N., RHODIN H., SHEFFER A.: Detecting viewer-perceived intended vector sketch connectivity. *ACM Transactions on Graphics 41* (2022). 6, 7, 10, 19, 20, 24

[YSC*18] YANG W., SEAH H.-S., CHEN Q., LIEW H.-Z., SÝKORA D.: Ftp-sc: Fuzzy topology preserving stroke correspondence. *Computer Graphics Forum 37*, 8 (2018), 125–135. 10

[YSR*20] YAO Y., SCHERTLER N., ROSALES E., RHODIN H., SIGAL L., SHEFFER A.: Front2back: Single view 3d shape reconstruction via front to back prediction. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2020). 10

[Yu23] YU E.: *Designing tools for 3D content authoring based on 3D sketching*. PhD thesis, Université Côte d'Azur, 2023. 2

[YVG20] YAN C., VANDERHAEGHE D., GINGOLD Y.: A benchmark for rough sketch cleanup. *ACM Transactions on Graphics (TOG) 39*, 6 (2020), 1–14. 4, 23, 24

[YZF*21] YANG L., ZHUANG J., FU H., WEI X., ZHOU K., ZHENG Y.: Sketchgnn: Semantic sketch segmentation with graph neural networks. *ACM Trans. Graph. 40*, 3 (Aug. 2021). 9

[ZCL*18] ZHANG J., CHEN Y., LI L., FU H., TAI C.-L.: Context-based sketch classification. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch-Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering* (2018), pp. 1–10. 23, 24

[ZCZ*09] ZHANG S.-H., CHEN T., ZHANG Y.-F., HU S.-M., MARTIN R. R.: Vectorizing Cartoon Animations. *IEEE Trans. Vis. Comput. Graph. 15*, 4 (July 2009), 618–629. 6, 9, 13, 14, 18

[ZDL*23] ZHANG Z., DENG X., LI J., LAI Y., MA C., LIU Y., WANG H.: Stroke-based semantic segmentation for scene-level free-hand sketches. *The Visual Computer 39*, 12 (2023), 6309–6321. 23, 24

[ZGZS20] ZHONG Y., GRYADITSKAYA Y., ZHANG H., SONG Y.-Z.: Deep sketch-based modeling: Tips and tricks. In *2020 International Conference on 3D Vision (3DV)* (2020), IEEE, pp. 543–552. 5

[ZJL20] ZHANG L., JI Y., LIU C.: Danbooregion: An illustration region dataset. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16* (2020), Springer, pp. 137–154. 19, 23

[ZLSS*21] ZHANG L., LI C., SIMO-SERRA E., JI Y., WONG T.-T., LIU C.: User-guided line art flat filling with split filling mechanism. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2021), pp. 9889–9898. 19, 22

[ZLWH16] ZHU H., LIU X., WONG T.-T., HENG P.-A.: Globally optimal toon tracking. *ACM Transactions on Graphics (TOG) 35*, 4 (2016), 1–10. 11

[ZPD*24] ZHENG Y., PANG K., DAS A., CHANG D., SONG Y.-Z., MA Z.: Creativeseg: Semantic segmentation of creative sketches. *IEEE Transactions on Image Processing 33* (2024), 2266–2278. 9

[ZPW*23] ZHENG X.-Y., PAN H., WANG P.-S., TONG X., LIU Y., SHUM H.-Y.: Locally attentional sdf diffusion for controllable 3d shape generation. *ACM Transactions on Graphics 42*, 4 (July 2023), 1–13. 7

[ZQG*20] ZHONG Y., QI Y., GRYADITSKAYA Y., ZHANG H., SONG Y.-Z.: 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. 23, 24, 25

[ZTCS99] ZHANG R., TSAI P.-S., CRYER J. E., SHAH M.: Shape-from-shading: a survey. *IEEE transactions on pattern analysis and machine intelligence 21*, 8 (1999), 690–706. 5

[ZXS*22] ZHENG Y., XIE J., SAIN A., MA Z., SONG Y.-Z., GUO J.: Ende-gnn: An encoder-decoder gnn framework for sketch semantic segmentation. In *2022 IEEE International Conference on Visual Communications and Image Processing (VCIP)* (2022), pp. 1–5. 9

[ZXS*23] ZHENG Y., XIE J., SAIN A., SONG Y.-Z., MA Z.: Sketchsegformer: Transformer-based segmentation for figurative and creative sketches. *IEEE Transactions on Image Processing 32* (2023), 4595–4609. 9

[ZXZ20] ZHU X., XIAO Y., ZHENG Y.: 2d freehand sketch labeling using cnn and crf. *Multimedia Tools and Applications 79*, 1 (2020), 1585–1602. 9

[ZZCJ13a] ZHUANG Y., ZOU M., CARR N., JU T.: A general and efficient method for finding cycles in 3D curve networks. *ACM Transactions on Graphics 32*, 6 (Nov. 2013), 1–10. 10, 25

[ZZCJ13b] ZHUANG Y., ZOU M., CARR N., JU T.: A general and efficient method for finding cycles in 3d curve networks. *ACM Transactions on Graphics (TOG) 32*, 6 (2013), 1–10. 20