Skolkovo Institute of Science and Technology

as a manuscript

Vage Egiazarian

IMAGE VECTORIZATION USING DEEP LEARNING

PhD Dissertation Summary

for the purpose of obtaining academic degree
Doctor of Philosophy in Computer Science

Academic Supervisor:
Doctor of Science
Evgeny V. Burnaev

Moscow — 2024

Contents

1 Dissertation Topic

The theme of this research is the development of efficient methods for raster image and depth map for three-dimensional object vectorization using deep learning. Vectorization of objects refers to finding object representation using mathematical primitives and relationships between them.

To achieve this goal, the following tasks were addressed: data collection, construction of mathematical models, and development of vectorization algorithms. Data collection was performed through synthetic data generation and processing of real scanned images of two-dimensional and three-dimensional objects. Automatic algorithms using computer vision methods were developed for data cleaning and processing, along with procedures for manual data processing. The proposed algorithms enable obtaining annotated data in a semi-automatic mode, which opens up the possibility of using deep learning methods to train neural networks on those data.

Various neural network architectures, such as convolutional and recurrent networks, as well as transformers, were investigated to build architectures that effectively solve the set tasks. The aim of constructing an optimal architecture is to create models capable of accurately and efficiently vectorizing various types of images and three-dimensional objects.

The proposed algorithms demonstrate high accuracy and efficiency in solving object vectorization tasks. These algorithms can be applied in various fields such as computer vision, robotics, and data visualization to represent objects as vector graphics with high accuracy and detail.

The relevance of research

There are many software products for image and document recognition, but most of them focus on text recognition and either do not work with images or perform poorly with them. Typically, these products do not allow representing the recognition results as vector graphics, one of the most convenient formats for subsequent work. In the case of two-dimensional and three-dimensional data, existing vectorization approaches based solely on optimization algorithms have a large number of customizable hyperparameters and perform poorly with noise in the input data. Using neural networks can reduce the number of hyperparameters, and because neural networks learn to recognize patterns, it is possible to efficiently process even significantly noisy data.

The relevance of the study is justified by the fact that image and form vectorization is an important and complex task in the field of computer vision, which has not been fully solved yet. In this dissertation, the problem of precise vectorization for two-dimensional technical drawings and three-dimensional component models is considered, and a method for recognizing drawings and objects using neural networks is proposed for tasks requiring precise and flexible representation of two-dimensional and three-dimensional objects. The research results can be

applied in various applications such as automatic recognition and analysis of technical drawings, automated modeling and editing of 3D objects, as well as virtual and augmented reality.

The **objective** of this dissertation is to study and develop efficient methods for vectorization of two-dimensional technical drawings and three-dimensional models in the field of computer vision. The main tasks include studying existing methods, collecting and processing data, developing deep learning algorithms, evaluating and comparing vectorization methods, and studying their practical application.

The objective is achieved through the solution of the following **tasks**:

- obtaining high-quality geometric data for two-dimensional and three-dimensional objects for subsequent training of neural network models;
- developing a new system for vectorization of raster images of technical drawings;
- developing an algorithm for reconstructing parametric models of three-dimensional objects.

The work is based on the use of **methodology and methods** of machine learning, differential geometry, and mathematical modeling.

2 Key Results and Contributions

The work is based on the use of **methodology and methods** of computer vision, computer graphics, machine learning, and deep learning.

The **scientific novelty** of the work can be described by the following points:

1. A new method was proposed for generating high-resolution three-dimensional objects at different scales. A new deep multi-scale model for point cloud generation, Latent-Space Laplacian Pyramid GAN, was developed based on advanced technologies in the field of generative adversarial networks for point cloud data [1] and approaches for modeling data at different scales [8, 26].
2. For the first time, a system was proposed that allows for a more precise solution to the vectorization problem of scanned drawings. The system includes several components: specially prepared high-quality data for training neural network models, a new architecture of the trainable neural network model for vectorization and a new approach for optimizing primitives to build the final representation of the object in vector format. Trainable neural network models are used at multiple steps of this approach: for preprocessing the image to reduce noise levels; for obtaining an initial vector approximation of the image. Then, using the solution to an optimization task, the final result is constructed from the obtained initial approximation, and its post-processing is performed to minimize the number of primitives.

3. A new method for extracting parametric curves from three-dimensional point clouds was presented, allowing for accurate transformation of point clouds into analytical models of special curves in three-dimensional space, which describe the structural features of three-dimensional component models and are necessary for further building 3D descriptions of these objects.

   These results were published in the proceedings of international conferences at the Core A* and Core B level, with the papers undergoing double-blind peer review, and the presentations being given at international conferences.

Theoretical and Practical Significance

The theoretical significance of the work lies in the new methods and algorithms in the field of image processing, vectorization, and generation of three-dimensional data. The current state of approaches to vectorization and generation of images and three-dimensional data was analyzed in the study. Advanced approaches from machine learning, modern neural network architectures and optimization procedures were applied to build models and methods. Following the obtained practical results the understanding of the applicability boundaries of modern machine learning methods for vectorization tasks has been expanded.

The developed methods have practical applicability in various fields, such as automatic recognition and analysis of technical drawings, automated modeling and editing of three-dimensional objects.

Results Presented for Defense:

1. *Methodology* and algorithms for data generation,
2. *Algorithm* for transforming raster technical drawings into vector graphics while preserving information,
3. *Algorithm* for transforming three-dimensional scanned objects into vector graphics consisting of three-dimensional curves,
4. *Methodology* for evaluating the accuracy and quality of vectorized data.

The **reliability** of the obtained results is ensured by the correct use of a tested set of tools for research and analysis. The proposed algorithms were experimentally tested on various tasks and on real datasets of both two-dimensional and three-dimensional objects. Detailed descriptions of the experiments conducted and the source code are publicly available, enabling replication of the obtained results. The research results were published in leading scientific journals and at conferences dedicated to computer vision and pattern recognition.

Personal Contribution to the Presented Results: All stated results were obtained by the author of this dissertation. In all mentioned cases, both the text of the papers and the experimental results presented therein are the results of collaboration with other authors.

In Paper 1: "Deep vectorization of technical drawings," Egiazaryan V.G., as the primary author, developed and implemented a system for processing input images with subsequent acquisition of vector representations. Additionally, the author was responsible for developing the neural network architecture and its training algorithm, which transforms raster images into vector primitives, and jointly with co-authors developed methods and experimental design for evaluating the effectiveness of the proposed method.

In Paper 2: "Def: Deep estimation of sharp geometric features in 3d shapes," Egiazaryan V.G. was responsible for developing and improving the second component of the entire pipeline: the algorithm for recovering parametric curves from point clouds.

In Paper 3: "Latent-Space Laplacian Pyramid GAN," the author of the dissertation developed and implemented the algorithm for multi-scale generation of three-dimensional point clouds, as well as a methodology for evaluating the accuracy and quality of the generated objects.

3 Publications and Validation of the Work

First-tier publications

1. *Vage Egiazarian**, Oleg Voynov*, Alexey Artemov, Denis Volkhonskiy, Aleksandr Safin, Maria Taktasheva, Denis Zorin, Evgeny Burnaev. Deep vectorization of technical drawings. In Proceedings of the European Conference on Computer Vision, pp. 582-598, Springer, Cham. ECCV 2020, CORE A*, indexed by SCOPUS.

2. Albert Matveev*, Ruslan Rakhimov*, Alexey Artemov, Gleb Bobrovskikh, *Vage Egiazarian*, Emil Bogomolov, Daniele Panozzo, Denis Zorin, Evgeny Burnaev. DEF: Deep estimation of sharp geometric features in 3d shapes. In Proceedings of the SIGGRAPH 2022 conf., ACM Transactions on Graphics (TOG), 41(4):122, 2022. CORE A*, indexed by SCOPUS.

Second-tier publications

3. *Vage Egiazarian**, Savva Ignatyev*, Alexey Artemov, Oleg Voynov, Andrey Kravchenko, Youyi Zheng, Luiz Velho, Evgeny Burnaev. Latent-space laplacian pyramids for adversarial representation learning with 3d point clouds. In Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, pp. 421-428, VISIGRAPP 2020, CORE B, indexed by SCOPUS.

The author has also contributed to the following publication

4. Oleg Voynov, Alexey Artemov, *Vage Egiazarian*, Alexander Notchenko, Gleb Bobrovskikh, Denis Zorin, Evgeny Burnaev. *Perceptual deep depth super-resolution.* In Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 5653-5663, 2019. CORE A*, indexed by SCOPUS.

5. Denis Mazur*, *Vage Egiazarian**, Stanislav Morozov*, Artem Babenko. *Beyond vector spaces: Compact data representation as differentiable weighted graphs*. In Proceedings of the Advances in Neural Information Processing Systems (NeurIPS), Volume. 32, 2019. CORE A*, indexed by SCOPUS.

6. Alexander Korotin, *Vage Egiazarian*, Arip Asadulaev, Alexander Safin, Evgeny Burnaev. *Wasserstein-2 generative networks*. In Proceedings of the International Conference on Learning Representations (ICLR), 2021. CORE A*, indexed by SCOPUS.

7. Alexander Korotin, *Vage Egiazarian*, Lingxiao Li, Evgeny Burnaev. *Wasserstein iterative networks for barycenter estimation*. In Proceedings of the Neural Information Processing Systems (NeurIPS), 2022. Volume 35. CORE A*, indexed by SCOPUS.

8. Arip Asadulaev*, Alexander Korotin*, *Vage Egiazarian*, Petr Mokrov, Evgeny Burnaev. *Neural optimal transport with general cost functionals*. In Proceedings of the International Conference on Learning Representations (ICLR), 2024. CORE A*, indexed by SCOPUS.

9. Tim Dettmers*, Ruslan Svirschevski*, *Vage Egiazarian**, Denis Kuznedelev*, Elias Frantar, Saleh Ashkboos, Alexander Borzunov, Torsten Hoefler, Dan Alistarh. *SpQR: A Sparse-Quantized Representation for Near-Lossless LLM Weight Compression*. In Proceedings of the International Conference on Learning Representations (ICLR), 2024. CORE A*, indexed by SCOPUS.

10. *Vage Egiazarian**, Andrei Panferov*, Denis Kuznedelev, Elias Frantar, Artem Babenko, Dan Alistarh. *Extreme Compression of Large Language Models via Additive Quantization*. In Proceedings of The Forty-first International Conference on Machine Learnings (ICML), 2024. CORE A*, indexed by SCOPUS.

Reports at conferences and seminars

- Deep vectorization of technical drawings, poster presentation at 16th European Conference on Computer Vision (ECCV), Online, 2020;
- "Vectorization using deep learning," presentation at the seminar of the Association "Artificial Intelligence in Industry," Online;
- "DEF: Deep estimation of sharp geometric features in 3D shapes," poster presentation at the SIGGRAPH conference, Canada, 2022;
- "Latent-space Laplacian pyramids for adversarial representation learning with 3D point clouds," poster presentation at the VISAPP conference, Malta, 2020.

* - Equal contribution

4 Dissertation Structure

The main part of the dissertation consists of four sections:

Section 5 describes the methodology for obtaining vectorization data. Methods for obtaining both synthetic and real data necessary for training and evaluating vectorization models are considered.

Section 6 presents a data processing system for 2D technical drawings aimed at obtaining vector graphics. Methods and algorithms used to transform raster images into vector representations are described.

Section 7 describes a new method for generating high-quality three-dimensional objects. Existing methods for generating three-dimensional data are discussed, and the general architecture and structure of the proposed new method are described.

Section 8 proposes a method for obtaining three-dimensional parametric curves from three-dimensional objects. The efficiency of the developed method is compared to existing approaches.

5 Description of Data and Problem Statement

Vector representations are often used for technical drawings and three-dimensional models, such as architectural plans, engineering drawings, 3D models of buildings and parts, etc. The demand for vector representations is justified by several advantages. First, vector graphics are scale-independent, much more compact, and, most importantly, support easy editing at the primitive level. Another important advantage of vector representation is that it provides the basis for a higher-level semantic structure (e.g., with sets of primitives hierarchically grouped into semantic objects).

However, in many cases, vector representation is unavailable, and only the "raw" object is accessible. In the two-dimensional case, this is a scanned image, and in the three-dimensional case, it is often RGB-D images or point clouds. Examples of such scanned images include old hand-drawn sketches, for which only a printed copy of the original source is available, or images found in online collections. When the original vector representation of the document is unavailable, it is usually manually reconstructed based on scans or photographs. The conversion to a vector representation is commonly referred to as *vectorization*.

There is a wide range of methods that often solve various variations of the vectorization problem and are tailored to different types of input and output data. Most of them are not based on machine learning approaches and apply algorithmic and optimization methods to solve the vectorization task. However, such approaches typically have a number of drawbacks, including a large number of hyperparameters, low processing speed, and little consideration of the processing specifics of different types of images.

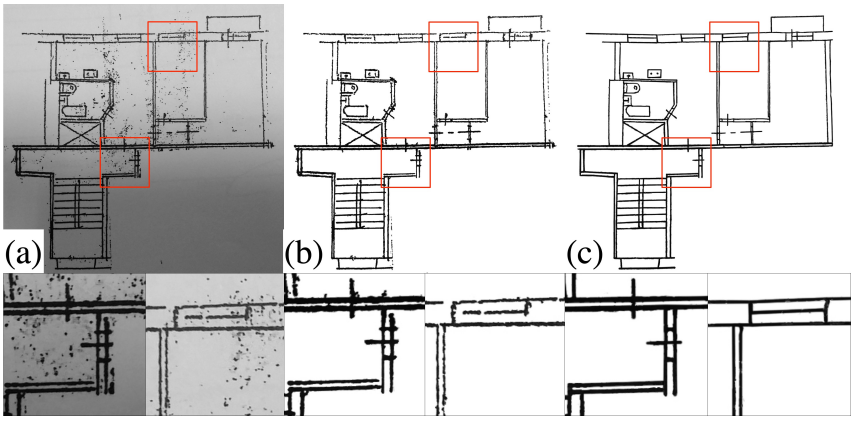


Figure 1: Example from the DLD dataset: (a) original input image, (b) image cleaned from background and noise, (c) final target mask with filled lines.

With the development of deep learning, the idea of using approaches from this field to solve the vectorization task naturally arises. However, such methods also have their own limitations, including the need for a large amount of training data. In the case of vectorization, a significant amount of input and output data pairs are required for training neural networks.

Collecting real data is a laborious task that requires significant resources [24, 38]. Generating synthetic data is less resource-intensive, but the obtained data may significantly differ from real data and contain domain shifts. This means that neural networks trained only on synthetic data may perform poorly when applied to real data. Therefore, a good strategy is to use a combination of synthetic and real data.

5.1 Description of Data for Two-Dimensional Drawings

There are several ways to obtain pairs of raster and vector images. One approach is to first collect raster images and then trace the lines manually, thereby creating a vector representation. This method is accurate in terms of matching the output vector representation to the input raster image but is resource-intensive. In this case, the location and style of the vector image may not correspond to the original drawing. Another approach is to start with a vector image and render it into a raster image. This approach is cheaper and requires significantly fewer human resources but does not necessarily create realistic raster images, as it is difficult to model the degradation of real-world documents [21]. Due to these limitations, it was decided to use both approaches.

The task of image vectorization can be divided into two parts: improving the quality of the raster image to remove noise and artifacts, and obtaining pairs of clean raster and vector images.

To solve the problem of improving the quality of the raster image, a dataset of real technical drawings (DLD) and a set of synthetic data were collected.

The real data consists of photos and scans of 81 floor plans with a resolution of $\sim 1300 \times 1000$. To prepare the raster and corresponding ground truth data, each image was manually cleaned

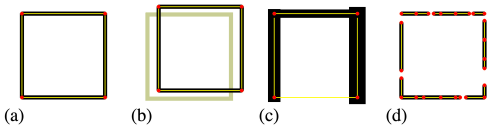


Figure 2: (a) Vector image serving as the ground truth, and artifacts used for vectorization performance evaluation. (b) Skeleton structure deviation. (c) Shape distortion. (d) Excessive parameterization.

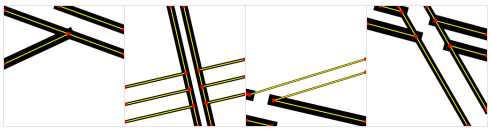


Figure 3: The examples of synthetic training data for our primitive extraction network.

by removing text, background, and noise, as well as enhancing the line structure by filling gaps and sharpening edges (Figure 1).

The synthetic data includes 20,000 pairs of images with a resolution of 512×512. The ground truth data were obtained by generating random mathematical primitives. The input images were obtained by rendering the ground truth data onto one of 40 realistic scanned paper backgrounds and degrading the rendering with random blurring, distortion, noise, etc.(Figure 2)

To obtain pairs of clean raster and vector images, the **PFP vector floor plan dataset** was created, which includes 1554 architectural floor plans from the real world [35], and the **ABC vector mechanical parts dataset** containing 10,000 vector images of mechanical parts obtained from the dataset [23] by projecting them onto a two-dimensional canvas using Opencascade [34] (see Figure 3).

5.2 Description of Data for Three-Dimensional Objects

Three-dimensional objects have different representations, among which the most common ones are Point Clouds, RGB-D, Mesh, and CAD models. RGB-D and point clouds can be obtained by measuring distance using 3D scanners (e.g., LiDARs, RGB-D cameras, and structured light scanners) or computed using stereo matching algorithms. Mesh and CAD models are typically created by engineers or designers in professional software such as Blender, AutoCAD, etc. Similar to two-dimensional data, there are several ways to obtain training samples for three-dimensional models.

One approach, complex in implementation but allowing relatively inexpensive creation of a large amount of data, is the use of generative modeling. Data created in this way are synthetic and often less realistic, but they provide many opportunities for manipulation. Alternatively, data can be extracted from CAD models or other representations (e.g., from the ABC dataset [23]). These data are usually of higher quality compared to the results of generative modeling, and they can provide not only input data but also ground truth. However, their volume is limited by the number of original models, and they also cannot model real distributions

of real scanned data. Models trained on the above data may not always perform well on real data due to domain shift. The most accurate, albeit most costly, way to obtain training data is to scan real objects using LiDARs, RGB-D cameras, and structured light scanners [40].

Obtaining Data from CAD Models

The data presented in [28] is derived from CAD models [23] and constitutes a collection of local patches and complete three-dimensional models at various resolutions: low, medium, and high, as well as at different levels of noise.

To obtain the data, n_v virtual cameras were set up with uniformly distributed positions on a sphere around the object (using Fibonacci sampling [16]), with the z-axis pointing to its center of mass. For each camera, a regular grid (image) of size 64×64 pixels (with r denoting the pixel size) was created, and rays perpendicular to the grid were cast from each pixel corner. Thus, patches of up to 4096 points were obtained (points outside the object were set to a background value).

Given the camera parameters K, T, where $K \in \mathbb{R}^{2\times3}$ is the intrinsic matrix transforming point coordinates from camera coordinates to the image plane, and $T \in \mathbb{R}^{4\times4}$ is the external camera transformation matrix from camera coordinates to the global coordinate system [17], selected points $p_{ij} = (x_{ij}, y_{ij}, z_{ij})$ (in homogeneous coordinates) could be associated with the depth image $I = (z_{ij}^{\text{cam}})$, where $z_{ij}^{\text{cam}} = (KT^{-1}p_{ij})_3$ denotes the z-coordinate of point p_{ij} in camera coordinates. Annotation images of distances to the object were created by computing $d = (d^{\varepsilon}(p_{ij}))$ and recording the pair I, d as a training example. A value of 18 was chosen for n_v, and the dataset was augmented by rotating and shifting the grid of images during data generation, while preserving the same orientation of the z-axis [28]. Thus, a training set was obtained for 68 complete three-dimensional models at various resolutions.

6 Vectorization of Two-Dimensional Images

There are numerous methods for vectorizing images and drawings [30, 37, 12, 9, 3, 5, 2, 13, 15]. One of the most widely used methods for image vectorization is Potrace, which finds contour of curves in the provided images[37]. Potrace can handle both colored and black and white images. Some works use Potrace as a stage in their algorithms [31, 22]. Another set of works is based on extracting a network of curves and cleaning up the topology [12, 3, 32, 5, 4, 33, 19] [42, 20]. The work by Ellis et al. [11] uses machine learning to extract high-level representations from raster sketches without aiming to precisely reproduce the geometry of primitives. Guo et al. [14] focus on improving the accuracy of topology reconstruction. The algorithm by Gao et al. [13] considers only simple data and uses a neural network for more accurate matching of the geometry of curve segments.

Each of these methods addresses different versions of the vectorization problem and works with different types of input and output data. Some algorithms can handle noisy data, while

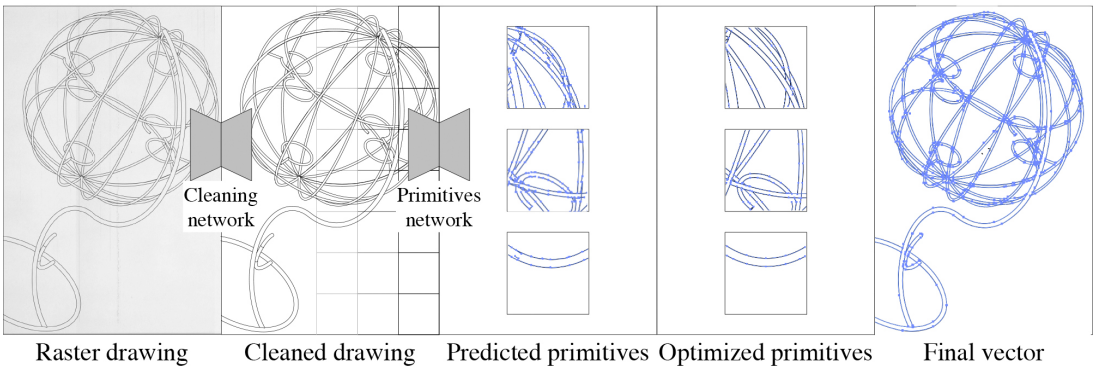


Figure 4: Overview of the presented vectorization method. In the initial stage, the input image is cleaned using a deep convolutional neural network. Then the clean result is divided into patches, and for each patch, the placement of primitives is estimated using a deep neural network. Next, the primitives in each patch are refined through iterative optimization. At the final stage, patches are merged into a single vector image.

others require clean input; some methods can process colored images, while others cannot. Different methods yield different sets of mathematical primitives as output.

Although different applications have different requirements, overall "good" vectorization adheres to the following requirements:

- Well approximate the input image semantically or perceptually;
- Avoid vectorizing noise and handle artifacts;
- Minimize the number of primitives used, creating a compact and easily editable representation.

The method proposed by the author of this dissertation [10] focuses on vectorizing technical images. The developed system (Figure 4) takes a raster sketch as input and produces a set of graphic primitives, such as line segments and quadratic Bezier curves, with width.

The method consists of the following steps:

1. Preprocessing of raster images. At this stage, a pre-trained neural network removes noise, corrects contrast, and fills in missing parts of the image.
2. Initial primitives estimation. The cleaned image from the previous step is divided into fragments, and initial parameters of primitives are estimated for each fragment using a neural network.
3. Optimization of primitive parameters. Refinement of primitive parameters occurs by aligning them with the cleaned raster.

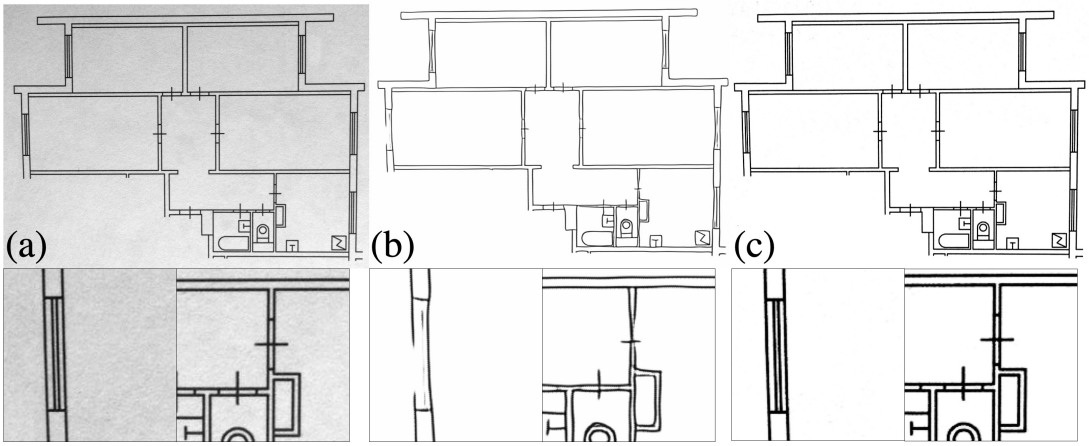


Figure 5: Example of image preprocessing: (a) original raw image, (b) result from [38], (c) result from our model.

	IoU,%	PSNR
[38]	49	15.7
Ours	**92**	**25.5**

Table 1: Comparison of data cleaning with [38].

4. Post-processing. Predictions from all fragments are combined, and redundant primitives are removed.

Let's now consider each step in more detail.

6.1 Preprocessing of images

The goal of preprocessing is to clean raw input data by removing noise, filling in missing parts of lines, and setting the background to white. This task can be considered as an image segmentation problem into two classes, where one class represents the background and the other represents everything else. Following the ideas of [24, 38], the work utilizes the U-net architecture [36], which is widely used in segmentation tasks. The model is trained on synthetically generated data and fine-tuned on real data described in Section 5. Binary cross-entropy loss function is used. Comparisons with the method of [38] are presented in Table 1, and an example of the work is shown in Figure 5.

6.2 Initial primitives estimation

First, the cleaned raster image obtained from the previous stage is divided into patches. Then, the parameters of primitives for each patch are independently initialized using a neural network. The neural network consists of an encoder based on ResNet [18] and a sequence of n_{dec} transformer blocks [39].

Each patch $I_p \in [0,1]^{64\times64}$ is encoded using the encoder network $X^{\text{im}} = \text{ResNet}\,(I_p)$, and then the feature embeddings of primitives X_i^{pr} are decoded using the transformer blocks (1).

$$X_i^{\text{pr}} = \text{Transformer}\left(X_{i-1}^{\text{pr}}, X^{\text{im}}\right) \in \mathbb{R}^{n_{\text{prim}}\times d_{\text{emb}}}, \qquad i = 1,\ldots,n_{\text{dec}}. \tag{1}$$

At the output, we obtain a set of parameters n_{prim} of size d_{emb}. The maximum number of primitives is determined by the size of the 0-th embedding $X_0^{\text{pr}} \in \mathbb{R}^{n_{\text{prim}}\times d_{\text{emb}}}$, initialized with positional encoding as described in [39].

More than 97% of patches in the training dataset contain no more than 10 primitives. Based on this, it was decided to fix the maximum number of predicted primitives at ten and filter out any additional ones. To obtain the coordinates of control points and the width of primitives $\Theta = \{\boldsymbol{\theta}_k = (x_{k,1}, y_{k,1}, \ldots, w_k)\}_{k=1}^{n_{\text{prim}}}$ and the confidence values that the primitive exists $\boldsymbol{p} \in [0,1]^{n_{\text{prim}}}$, the last obtained feature embedding is passed to a fully connected block. If the width or confidence value is less than a threshold (typically set to 0.5), the primitive is discarded.

To predict primitive parameters, a neural network was trained using a loss function that evaluates the accuracy of simultaneously solving multiple tasks: binary entropy for confidence values and a weighted sum of parameter deviations with (L1) and (L2) losses. Since this loss function is not permutation-invariant regarding the primitives and their control points, the endpoints in primitives and the primitives themselves are sorted lexicographically.

$$L\left(\boldsymbol{p}, \hat{\boldsymbol{p}}, \Theta, \hat{\Theta}\right) = \frac{1}{n_{\text{prim}}} \sum_{k=1}^{n_{\text{prim}}} \left(L_{\text{cls}}\left(p_k, \hat{p}_k\right) + L_{\text{loc}}\left(\boldsymbol{\theta}_k, \hat{\boldsymbol{\theta}}_k\right)\right), \tag{2}$$

$$L_{\text{cls}}\left(p_k, \hat{p}_k\right) = -\hat{p}_k \log p_k - (1-\hat{p}_k)\log(1-p_k), \tag{3}$$

$$L_{\text{loc}}\left(\boldsymbol{\theta}_k, \hat{\boldsymbol{\theta}}_k\right) = (1-\lambda)\,\|\boldsymbol{\theta}_k - \hat{\boldsymbol{\theta}}_k\|_1 + \lambda\|\boldsymbol{\theta}_k - \hat{\boldsymbol{\theta}}_k\|_2^2. \tag{4}$$

6.3 Primitive parameters optimization

The neural network for predicting primitives minimizes the mean squared error, but even with a small average deviation, individual estimates can be inaccurate. To refine the primitive parameters, a functional (5) has been developed, which aligns primitives to the raster image.

$$\Theta^{\text{ref}} = \underset{\Theta}{\text{argmin}}\, E\left(\Theta, I_p\right). \tag{5}$$

For each primitive, an optimization functional was defined that consist of the sum of three terms:

$$E\left(\Theta^{\text{pos}}, \Theta^{\text{size}}, I_p\right) = \sum_{k=1}^{n_{\text{prim}}} E_k^{\text{size}} + E_k^{\text{pos}} + E_k^{\text{rdn}}, \tag{6}$$

where $\Theta^{\text{pos}} = \left\{\boldsymbol{\theta}_k^{\text{pos}}\right\}_{k=1}^{n_{\text{prim}}}$ — is a primitive parameter, $\Theta^{\text{size}} = \left\{\boldsymbol{\theta}_k^{\text{size}}\right\}_{k=1}^{n_{\text{prim}}}$ —size parameter, and $\boldsymbol{\theta}_k = \left(\boldsymbol{\theta}_k^{\text{pos}}, \boldsymbol{\theta}_k^{\text{size}}\right)$.

The positions of the line are defined by the coordinates of its midpoint and its inclination angle, while the size is determined by its length and width. In the case of an arc, the midpoint is defined as the intersection of the curve and the bisector of the angle between the segments connecting the midpoint and the endpoints. The lengths of these segments and the inclination angles connecting the "midpoint" with the endpoints are also utilized.

Interaction of Charges. Various parts of the functional are based on the energy of interaction between unit point charges $\boldsymbol{r}_1$, $\boldsymbol{r}_2$, defined as the sum of near and far-field potentials.

$$\varphi(\boldsymbol{r}_1, \boldsymbol{r}_2) = e^{-\frac{\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|^2}{R_\text{c}^2}} + \lambda_\text{f} e^{-\frac{\|\boldsymbol{r}_1 - \boldsymbol{r}_2\|^2}{R_\text{f}^2}}, \tag{7}$$

The parameters R_c, R_f, λ_f are chosen experimentally.

The interaction energy between a uniformly positively charged region of the i-th primitive Ω_k and a grid of point charges $\boldsymbol{q} = \{q_i\}_{i=1}^{n_\text{pix}}$ at the centers of pixels $\boldsymbol{r}_i$ is determined by the following equation:

$$E_k(\boldsymbol{q}) = \sum_{i=1}^{n_\text{pix}} q_i \iint\limits_{\Omega_k} \varphi(\boldsymbol{r}, \boldsymbol{r}_i)\, dr^2. \tag{8}$$

In the case of curves, it is approximated by the sum of integrals over segments of a polyline that straightens the curve.

The proposed functional utilizes three different charge grids, encoded as vectors of length n_pix: $\hat{\boldsymbol{q}}$ represents the raster image with charges set to pixel intensities, $\boldsymbol{q}_k$ represents the rendering of the i-th primitive with its current parameter values, and $\boldsymbol{q}$ represents the rendering of all primitives in the patch. The sets of charges $\boldsymbol{q}_k$ and $\boldsymbol{q}$ are updated at each iteration.

Let's examine in detail formula (6), which consists of three components.

The first component is responsible for expanding and shrinking the primitive to cover filled pixels (with the primitive's position fixed):

$$E_k^\text{size} = E_k\left([\boldsymbol{q} - \hat{\boldsymbol{q}}] \odot \boldsymbol{c}_k + \boldsymbol{q}_k \odot [\boldsymbol{1} - \boldsymbol{c}_k]\right). \tag{9}$$

The weight $c_{k,i} \in \{0, 1\}$ ensures coverage of the continuous raster region, following the shape and orientation of the primitive.

The second component is responsible for aligning primitives of fixed size.

$$E_k^\text{pos} = E_k\left([\boldsymbol{q} - \boldsymbol{q}_k - \hat{\boldsymbol{q}}] \odot [\boldsymbol{1} + 3\boldsymbol{c}_k]\right). \tag{10}$$

The last component is responsible for collapsing overlapping *collinear* primitives; for this term, $\lambda_\text{f} = 0$ is used:

$$E_k^\text{rdn} = E_k\left(\boldsymbol{q}_k^\text{rdn}\right),\ q_{k,i}^\text{rdn} = \exp\left(-\left[|\boldsymbol{l}_{k,i} \cdot \boldsymbol{m}_{k,i}| - 1\right]^2 \beta\right) \|\boldsymbol{m}_{k,i}\|, \tag{11}$$

where $\boldsymbol{l}_{k,i}$ —the direction of the primitive at its nearest point to the i-th , $\boldsymbol{m}_{k,i} = \sum_{j \neq k} \boldsymbol{l}_{j,i} q_{j,i}$ — the sum of directions of all other primitives, weighted relative to their "presence", and $\beta = (\cos 15^\circ - 1)^{-2}$ is chosen experimentally.

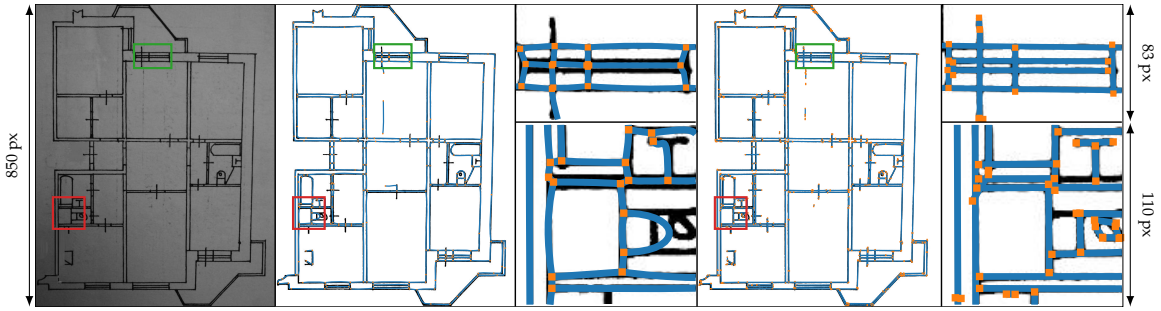


Figure 6: Qualitative comparison on a scan of a real floor plan. IoU / #P with the best result are highlighted in bold. Primitives are shown in blue color with orange endpoints overlaying the cleaned raster image.

An approximate solution for formula (5) can be obtained by considering each energy term E_k^{pos}, E_k^{size}, E_k^{rdn} as dependent only on the parameters of the i-th primitive. This allows for efficiently computing the gradient for the proposed functional, as it is necessary to differentiate each term only with respect to a small number of parameters.

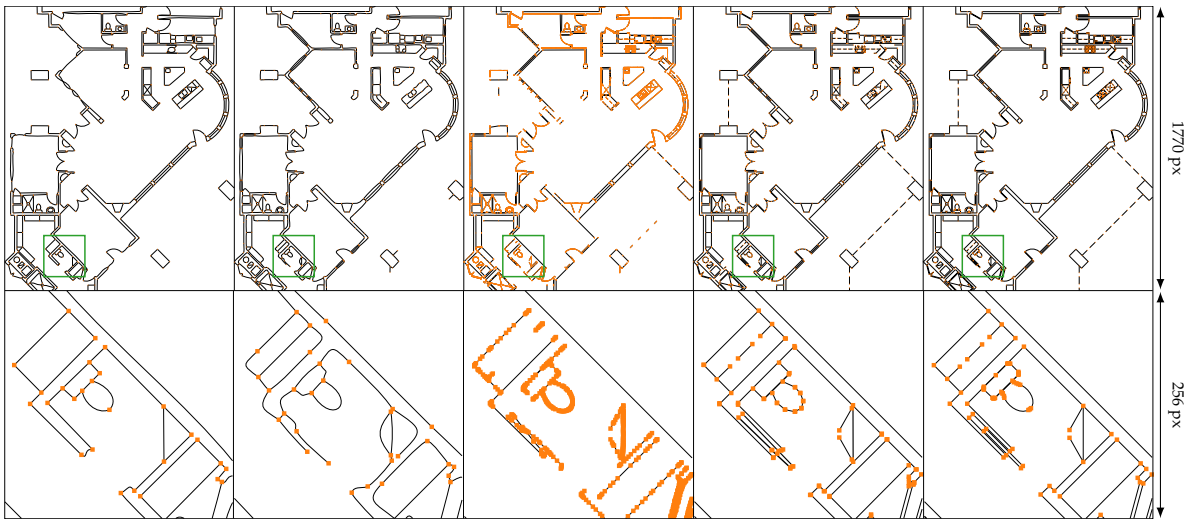
6.4 Experimental evaluation

To evaluate the algorithm's performance, 15 test images from the DLD dataset were used. The quantitative results of this evaluation are presented in Table 2, and the qualitative results are shown in Figure 6. Additionally, a comparison was made on 40 images from the PFP dataset and 50 images from ABC with resolutions of $\sim 2000 \times 3000$. The quantitative results on these data are also presented in Table 2, and the qualitative results are shown in Figures 7 and 8.

As was previously mentioned, good vectorization should contain a small number of primitives while still effectively approximating the input image. Based on the approximation quality metrics on the PFP dataset, our vectorization system outperforms other methods, only being surpassed in the number of primitives by the FvS model. In the case of the ABC dataset, despite the PVF model outperforming our vectorization system in terms of IoU metric, its result contains significantly more primitives.

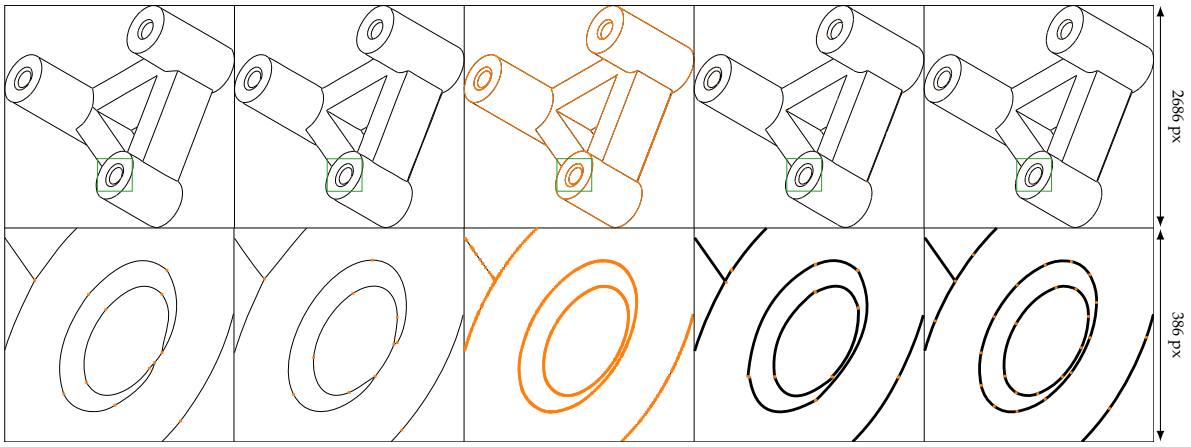
	PFP				ABC				DLD	
	IoU,%	HD, px	d_M, px	#P	IoU,%	HD, px	d_M, px	#P	IoU,%	#P
FvS	31	381	2.8	696	65	38	1.7	63		
CHD	22	214	2.1	1214	60	9	1	109	47	329
PVF	60	204	1.5	38k	89	17	0.7	7818		
Ours	86/88	25	0.2	1331	77/77	19	0.6	97	79/82	452

Table 2: Quantitative vectorization results on PFP, ABC, and DLD datasets using various methods.



FvS	CHD	PVF	Ours	GT,
29% / 415px	21% / 215px	64% / 140px	**89%** / **28px**	#P 1634
4.2px / **615**	1.9px / 1192	0.9px / 35k	**0.2px** / 1286	

Figure 7: Qualitative comparison on an image from the PFP dataset. The values of IoU / HD / d_M / #P with the best result are highlighted in bold. The endpoints of primitives are shown in orange color.



FvS	CHD	PVF	Ours	GT,
67%/ 32px	67%/ 7px	**95%**/ **4px**	86%/ 5px	#P 139
1.1px/ 79	1.0px/ 108	**0.2px**/ 9.5k	0.4px/ 139	

Figure 8: Qualitative comparison on an image from the ABC dataset. The values of IoU / HD / d_M / #P with the best result are highlighted in bold. The endpoints of primitives are shown in orange color.

Conclusion

This section discusses a new method for vectorizing technical images using neural networks. It involves cleaning the input image with a neural network based on the U-Net architecture, initial estimation of primitives using a combination of ResNet and Transformer neural networks, optimization of primitive parameters, and post-processing.

The proposed work significantly improves the quality of the resulting vector images of technical scans and reduces the number of hyperparameters thanks to the use of deep learning methods and a new approach to optimization.

7 Generation of High-Quality Synthetic Data with Descriptions of Three-Dimensional Objects

As was mentioned earlier, obtaining real data is a labor-intensive task, and training a neural network requires a large amount of data (see section 5). A neural network trained on synthetic data does not always perform well on real data due to domain shift. The problem of domain shift can be addressed by fine-tuning on real data, making synthetic data more realistic, or transforming real data to resemble synthetic data during inference.

The problem of collecting real data for vectorization is even more complex in three-dimensional cases. Fine-tuning on real data still requires a large amount of data relevant to the task. An alternative approach of training on synthetic data that approximates real data allows for the use of an unpaired dataset of synthetic and real data. Such a dataset can be collected using open sources from the internet. As a result, it is possible to train a model that makes synthetic data more realistic or vice versa. Therefore, it was decided to study the issue of translating one distribution into another in three-dimensional space using the standard task of generating three-dimensional point clouds from noise.

Generating three-dimensional data is a complex task. A recent trend is the use of data-driven generation methods such as deep generative models [1, 25, 6]. Most methods operate only on coarse three-dimensional geometry (low resolution) because high-resolution three-dimensional shapes require significant computational resources for processing and are challenging to train. To generate high-resolution data, a new deep cascaded model called Laplacian Pyramid in Latent Space GAN (LSLP-GAN) was developed [10].

The proposed model (illustrated schematically in Figure 9) for training using three-dimensional point clouds is based on works on latent GANs [1] and Laplacian GANs [8]. In [1], it was proposed to train GANs not in the space of point clouds but on latent codes obtained using an autoencoder. Laplacian GANs [8] address the challenge of training high-resolution objects by cascading image synthesis using a series of generative networks $G_0, \ldots, G_n$, where each network G_k learns to generate high-frequency residual images $r_k = G_k(U(I_{k+1}), z_k)$, with I_k being the reconstructed image obtained from G_{k-1}. Thus, the image at stage k is represented

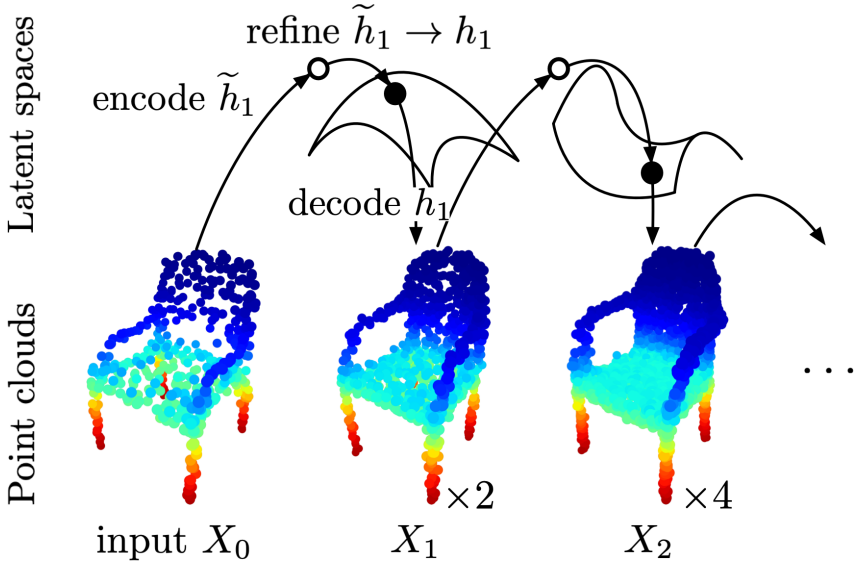


Figure 9: The proposed multi-stage Generative Adversarial Network (GAN) module over a latent space.

as follows:

$$I_k = U(I_{k+1}) + G_k(U(I_{k+1}), z_k), \tag{12}$$

where $U(\cdot)$ is the up-sampling operator, and z_k is the noise vector. Since modeling high-detail three-dimensional point clouds is challenging due to the high dimensionality, the idea of [8] was adopted to start with a low-detail (but easily constructible) model and break down the generation task into a sequence of simpler stages, each aimed at gradually increasing the level of detail.

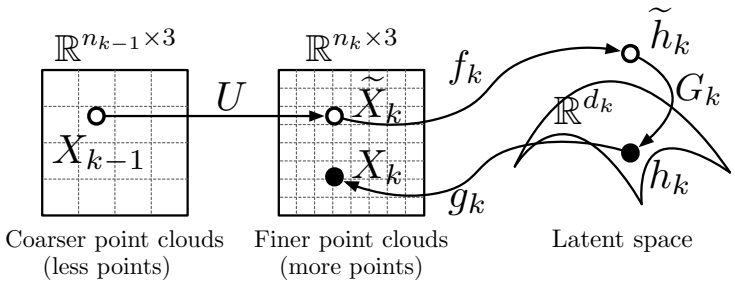


Figure 10: Detailed schematic of our Laplacian pyramid in the latent space.

From the space of three-dimensional shapes $\{\mathbb{R}^{n_k\times 3}\}_{k=1}^K$, corresponding latent code spaces $\{\mathbb{R}^{d_k}\}_{k=1}^K$ were constructed using trained point cloud autoencoders $\{(f_k, g_k)\}_{k=1}^K$. The autoencoder (f_k, g_k) is trained using a resolution of n_k point clouds, which increases as k grows. After training the autoencoders, their parameters were fixed, and hidden codes were extracted for point clouds in each of the three-dimensional shape spaces.

Laplacian Pyramid in Latent Code Space

The system takes as input a point cloud $X_{k-1} \in \mathbb{R}^{n_{k-1}\times 3}$. The goal is to transition from X_{k-1} to

X_k, i.e., to increase the resolution from n_{k-1} to $n_k = 2n_{k-1}$ by generating additional points on the shape's surface (see Figure 10).

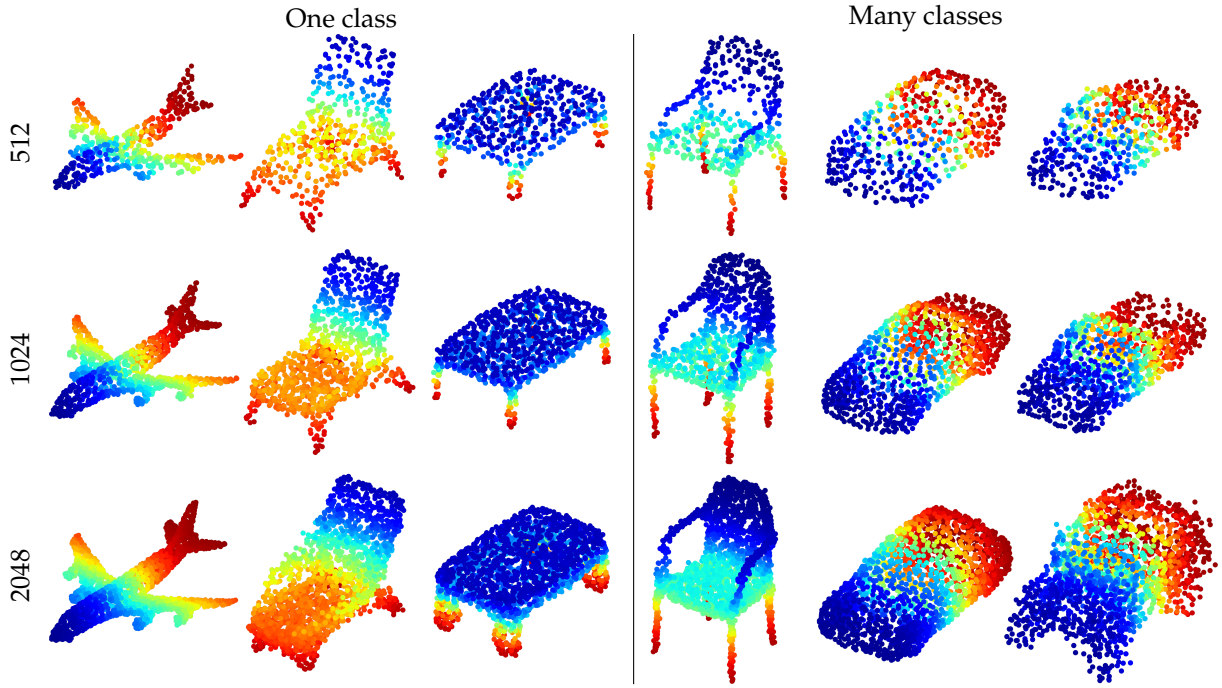


Figure 11: Examples of shapes synthesized using our LSLP-GAN model. *Left:* airplanes, chairs, and tables synthesized using our single-class models. *Right:* samples of three-dimensional shapes synthesized using our multi-class model.

The processing of the input point cloud begins with a simple interpolation operator $U(\cdot)$. The process of obtaining the coarse point cloud $\widetilde{X}_k = U(X_{k-1})$ occurred as follows: for each point $x \in X_{k-1}$, a new instance $\widetilde{x} = \frac{1}{m}\sum_{i\in \mathrm{NN}(x)} x_i$ was created, where $\mathrm{NN}(x)$ is a set of m nearest neighbors of x in the Euclidean space X_{k-1} (in the work, $m = 7$ neighbors, including x, were used), which was added to the point cloud. This procedure constitutes a simple linear interpolation and generates n_k points close to the real surface.

However, the computed point cloud $\widetilde{X}_k$ usually contains distorted points, and it was considered only as an approximation to our desired shape X_k. The coarse point cloud $\widetilde{X}_k$ was mapped using f_k to a hidden code $\widetilde{h}_k = f_k(\widetilde{X}_k)$, which is assumed to deviate slightly from the manifold of hidden representations due to interpolation error in $\widetilde{X}_k$. To compensate for this error in the latent space, an additional correction r_k for $\widetilde{h}_k$ was computed using the generative network G_k, resulting in the corrected code $h_k = \widetilde{h}_k + r_k = \widetilde{h}_k + G_k(\widetilde{h}_k, z_k)$. Decoding h_k is done using g_k, thereby obtaining the refined point cloud $X_k = g_k(h_k)$ with resolution n_k.

The complete procedure in the latent space represents a series of connections:

$$h_k = f_k(U(X_{k-1})) + G_k(f_k(U(X_{k-1})), z_k), \tag{13}$$

which is analogous to the Laplacian pyramid in the latent space (12). This procedure was named Latent-Space Laplacian Pyramid GAN (LSLP-GAN).

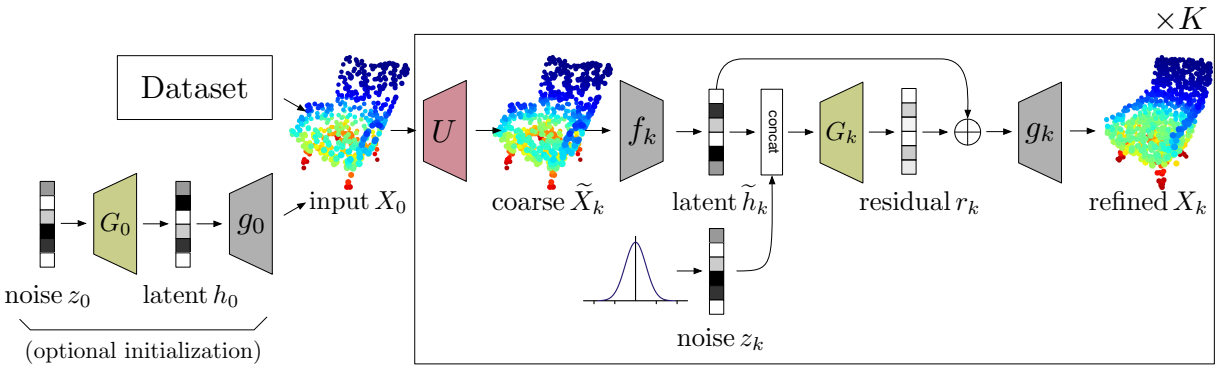


Figure 12: Complete architecture of the LSLP-GAN model. The network takes or generates an initial point cloud X_0 and processes it through a series of K trainable steps. Each step (1) increases the input resolution using the non-trainable operator U, (2) encodes the up-scaled version in the latent space using f_k, (3) performs correction of the hidden code using conditional GAN G_k, and (4) decodes the corrected hidden code using g_k.

Training GANs on Latent Codes

A series of generative adversarial networks $\{(G_k, D_k)\}_{k=1}^{K}$ were trained on the latent codes. The generator G_k synthesizes "residuals" r_k in the latent space based on the input $\widetilde{h}_k$, while the discriminator D_k distinguishes between real hidden codes $h_k \in \mathbb{R}^{d_k}$ and synthetic ones $\widetilde{h}_k + G_k(\widetilde{h}_k, z_k)$. It is noteworthy that each (except the first) generative adversarial network takes the coarse latent code $\widetilde{h}_k$ and can be considered as a conditional generative adversarial network (CGAN) [29].

Examples of generated objects can be seen in Figure 11.

Conclusion

This section presented a new method for multi-level generation of three-dimensional point clouds using a generative adversarial neural networks. The generative networks operate in the latent space of an autoencoder, generating increments that gradually increase the number and correct arrangement of points in the output model.

Thanks to the multi-level approach, this method can be applied not only for generating objects from noise, but also for enhancing the input point cloud. Specifically, this method can generate synthetic data close to real data to reduce domain shift during training. The proposed model can also be used as a standalone module to improve the point cloud obtained from the depth image of scanned objects for more accurate depth image vectorization. The effectiveness of these approaches for improving vectorization is a subject for future research.

Additionally, another application of the method can be the generation of vector graphics from noise. This generation occurs in two stages: first, a point cloud is generated, and then the vectorization method is applied.

8 Vectorization of Three-Dimensional Objects

Vectorizing three-dimensional objects, analogous to two-dimensional vectorization, involves representing objects and their connections using mathematical primitives. The most common examples are Computer Aided Design (CAD), object skeletons, and parametric curves.

This section will discuss a method for extracting parametric curves from scanned three-dimensional objects. The input consists of depth images of the object from various viewpoints, and the output is three-dimensional splines describing the input object. The method consists of two parts: obtaining an estimated distance field of geometric features and extracting parametric curves based on this information.

To obtain the distance field, a U-Net neural network based on ResNet-152 was used to solve the regression task of predicting the truncated distance field to special curves on the depth map. The proposed new approach then combined these predictions to produce estimated distance fields on full three-dimensional models [28].

The extraction of parametric curves utilizes the information from the estimated distance fields obtained earlier with Deep Estimators of Features (DEFs) [28] and involves a local classifier for detecting corner vertices, constructing a graph structure, and spline approximation.

The algorithm is based on the method described in the paper [27]. The overall structure of the approach is preserved, but significant improvements have been made to automate its operation, increase stability, and improve the quality of the obtained results. These improvements were achieved through the following changes:

- using a different criterion for detecting corners and endpoints of curves in (14), (17);
- a more reliable stage of polyline construction based on k nearest neighbors and a different optimization functional in (19);
- adding a post-processing method in (21).

Figure 13 provides a visual illustration of the differences between the two algorithms. As seen in this figure, the proposed algorithm handles complex three-dimensional objects better and is less prone to generating outliers.

8.1 Description of the Developed Method for Obtaining Three-Dimensional Parametric Curves

The proposed method includes several stages:

- Initialization,
- Corner Detection,
- Curve and Corner Segmentation,

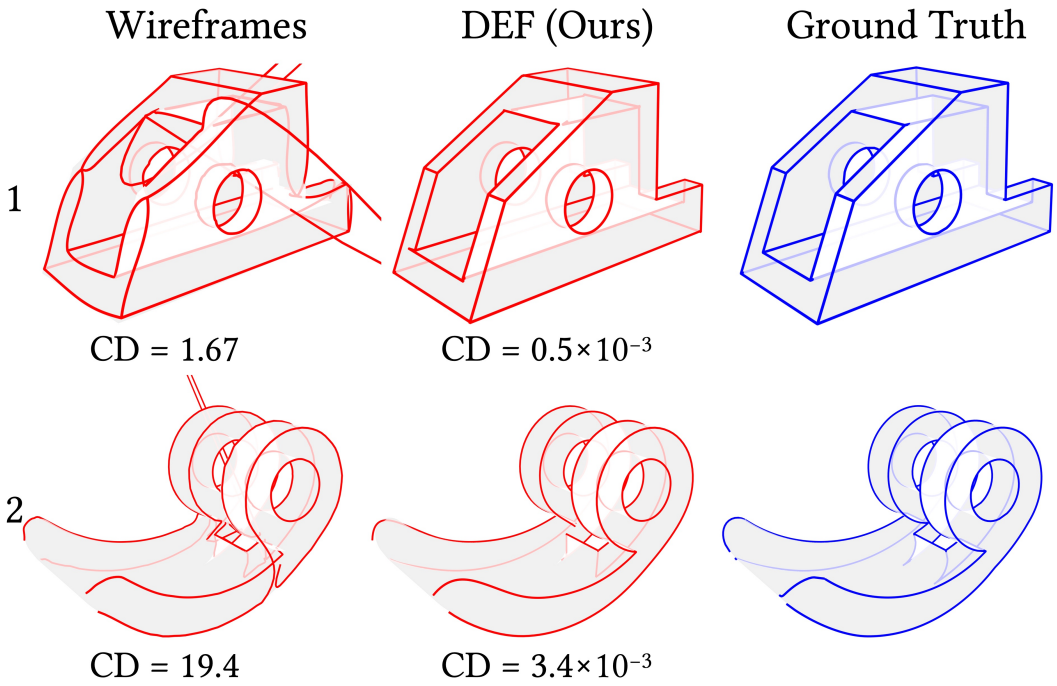


Figure 13: The proposed method improves corner detection (line 1) and can resolve complex curves (line 2), while *Wireframes* occasionally produce curves with significantly large deviations from the ground truth.

- Curve Graph Extraction,
- Spline Approximation and Optimization,
- Post-processing of representations obtained based on splines.

Figure 14 presents a visual scheme of our algorithm's operation. Below, each of these stages will be examined in more detail.

8.1.1 Initialization

At this stage, a subset of points is selected from the initial point cloud P, that consists of points with estimated distance to nearest sharp features $\widehat{d}$ less than the threshold d_{sharp}. To further reduce the number of points, the Poisson disk sampling [7] method was used, leaving only 10% of points. As a result, a subset of points P_{sharp} was obtained.

8.1.2 Corner Detection

At this stage, anchor points were selected from P_{sharp} using the farthest point method. 20% of points from P_{sharp} were used as anchor points. Then sets B_i were constructed, containing points located in overlapping Euclidean balls with a radius of R_{corner}, centered at anchor points and covering P_{sharp}.

Each of these local sets B_i was approximated by its ellipsoidal shape by computing the principal components (PCA) on the points of the set and obtaining the vector of variances $(\lambda_1, \lambda_2, \lambda_3)$, such that $\lambda_1 \leqslant \lambda_2 \leqslant \lambda_3$ and $\sum_{k=1}^{3} \lambda_k = 1$, describing the lengths of the ellipsoid axes. For each specific set B_i, the obtained vectors were used to compute the normalized aggregation of the quadratic distance using the following expression:

$$\Lambda_i = \sum_{k=1}^{3} \sum_{j \in \mathcal{N}_i} \left(\frac{\lambda_k^i - \lambda_k^j}{\delta_{ij}} \right)^2, \tag{14}$$

where $\mathcal{N}_i$ is the set of indices of sets B_j nearest to set B_i, and δ_{ij} is the Euclidean distance between the anchor points of sets B_i and B_j.

Next, the problem of assigning the local set B_i to clusters of points, belonging to corners, is solved. This is done by comparing Λ_i with the threshold T_{variance}, and labeling B_i as either a corner set or a curve type set:

$$\begin{aligned} \mathcal{B}_{\text{corner}} &= \{B_i \,|\, \Lambda_i > T_{\text{variance}}\}, \\ \mathcal{B}_{\text{curve}} &= \{B_i \,|\, \Lambda_i \leqslant T_{\text{variance}}\}. \end{aligned} \tag{15}$$

By varying $\mathcal{N}_i$, T_{variance}, and R_{corner} within small ranges, 60 combinations of classifications were obtained. Then, based on the proportion of corner classifications in a specific set B_i, the probability that this set is a corner was calculated.

To extend the classification to all points, the k nearest neighbors method with $k = 50$ was applied. Thus, for each point, a value $0 \leqslant w(p) \leqslant 1$ was obtained.

To determine the corner points a threshold value T_{corner} is applied:

$$P_{\text{corner}} = \{p \in P_{\text{sharp}} : w(p) > T_{\text{corner}}\}.$$

8.1.3 Segmentation of curves and angles

For curve segmentation, two sets of points were used: P_{corner}, containing corner points, and $P_{\text{curve}} = P_{\text{sharp}} \setminus P_{\text{corner}}$, containing points that are not corners. Both of these sets were processed to extract clusters that define individual corners and curves respectively.

For distinguish points belonging to separate curves, a dense graph was constructed using the k-nearest neighbors (kNN) method. All points in the set P_{curve} that were within a distance r (refer to Equation (16)) were connected to each other. The resulting graph was partitioned into connected components to obtain separate clusters of points corresponding to different curves.

$$\underbrace{r}_{\substack{\text{sampling} \\ \text{distance}}} \times \underbrace{n}_{\substack{\text{num. samples} \\ \text{per feature}}} = \underbrace{l}_{\substack{\text{characteristic} \\ \text{spatial size}}} \times \underbrace{s}_{\substack{\text{scaling} \\ \text{factor}}}, \tag{16}$$

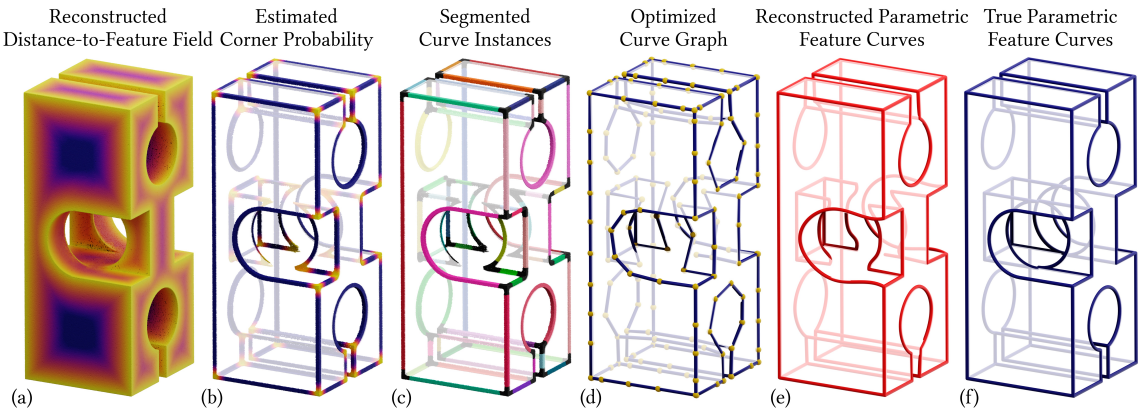


Figure 14: At the initial stage (a), thresholding is applied to the distances to obtain a subset P_{sharp}, which is then used for angle probability estimation (b) and curve segmentation (black clusters correspond to detected corner neighborhoods) (c). Next, the detected corners and curves allow the construction and optimization of the curve graph (d). In the final stage, the curve graph is transformed into a set of parametric curves (e), reflecting the geometry of the original shape (f).

The connected component defines one of the n_{curve} curves, and all of them constitute a set of point clusters corresponding to each curve:

$$\mathcal{P}_{\text{curve}} = \left\{ P_c \subseteq P_{\text{curve}} \,|\, \forall p \in P_c \, \exists q \in P_c, p \neq q : \|p - q\| \leqslant r \right\}_{c=1}^{n_{\text{curve}}}.$$

For corner points P_{corner}, the procedure is similar. Clusters of corners $\mathcal{P}_{\text{corner}}$ are extracted by partitioning the connected components of detected corner sets.

8.1.4 Curve Graph Extraction

From the segmentation obtained in the previous step, a curve graph corresponding to P_{sharp} is constructed. This procedure consists of the following steps:

- detection of endpoints for each curve, based on which curves are defined as open or closed;
- approximation of each curve by the shortest polyline;
- connecting the fitted polylines, corners, and endpoints into a complete curve graph;
- refinement of the positions of endpoints and corners.

Below, we will look at each of these steps in more detail.

Detection of Endpoints for Each Curve. To detect endpoints of the segmented curve cluster P_c, an endpoint detector based on the neighborhood, similar to the corner detector, is constructed. Then, Euclidean neighborhoods E_i with a radius R_{endpoint}, centered at anchor points p_{ai} chosen from P_c, are constructed, and their approximation by a line is computed by performing PCA on the points in E_i and reducing its dimensionality to one principal component. Then

each point $p \in E_i$ is parameterized by one coordinate $t(p)$ obtained from PCA. To determine the endpoints of the curve, the fraction of points $p \in E_i$ with parametric coordinates $t(p)$ greater or less than the parametric coordinate t_{ai} of the anchor point p_{ai} is computed:

$$V_i = \Big| \frac{1}{|E_i|} \sum_{p \in E_i} \operatorname{sign}(t(p) - t_{ai}) \Big|, \tag{17}$$

where p_{ai} is considered an endpoint if V_i is greater than the threshold value T_{endpoint}. $V_i = 0$ corresponds to a fully symmetric case, while $V_i = 1$ indicates a strong predominance of points on one side of the anchor point. If there is only one such anchor point for the curve cluster P_c, the anchor p_{ai} with the second largest value of V_i is selected as the second endpoint; if more than two endpoints are detected, the two farthest points are chosen; if endpoints are not detected, the curve is considered closed.

Approximation of Each Curve by the Shortest Polyline. For an open curve, a graph is created using the k-nearest neighbors method by connecting all anchor points of the curve p_{ai} from P_c that are within a distance not exceeding twice the average sampling distance from each other. The polyline is initialized by finding the shortest path in such a graph between the detected endpoints using Dijkstra's algorithm.

For a closed curve, three points with the greatest spread are chosen from the cluster and connected. Then the remaining polyline points are found using a splitting strategy. Candidates for splitting are determined by computing p_{split}:

$$p_{\text{split}} = \arg\max_{p_i \in P_c} \big| \widehat{d}_i - \|p_i - \min_l \pi^l(p_i)\| \big| \tag{18}$$

for points p_i from the current curve cluster $P_c \in \mathcal{P}$curve, where $\min_l \pi^l(p_i)$ is the projection of p_i onto the nearest polyline segment l. To continue the splitting, the absolute difference between the estimates distance to the nearest sharp features $\widehat{d}_i$ and the actual distances $|p_i - \pi^l(p_i)|$ is compared with the threshold value T_{split}; for candidates p_{split} exceeding this value, we split the polyline, assigning p_{split} as new vertices of the polylines and splitting the corresponding segment into two.

Connecting Fitted Polylines, Corners, and Endpoints into a Complete Graph. The detected endpoints of open curves are replaced with the corresponding nearest centers of corner clusters. Thus, a final curve graph $G(q, e)$ is obtained, defined by the positions of nodes q and the connections e between them.

Refinement of the Positions of Endpoints and Corners. Formula (19) is used for optimizing the positions of nodes:

$$\min_q \Big(\frac{1}{|P_{\text{sharp}}|} \sum_{p \in P_{\text{sharp}}} |\widehat{d}(p) - \big|p - \pi^{G(q,e)}(p)\big|| - \sum_{\overline{q} \in l[G(q,e)]} \cos \overline{q} \Big), \tag{19}$$

where $\pi^G(p)$ is the projection of point p onto the nearest edge in the curve graph G, and $\sum_{\overline{q} \in l[G(q,e)]} \cos \overline{q}$ is the sum of cosines of angles between consecutive edges incident to node

$\overline{q}$, computed only for the set of nodes $l[G(q, e)]$ having exactly two incident edges. The second term represents polyline stiffness, which prevents sharp angles between edges. Optimization helps to determine the positions of graph nodes more accurately, especially at intersections of multiple curves, while the stiffness term makes polyline segments straighter. Upon completion of this step, the final positions of corners are determined as the coordinates of graph nodes with more than two incident segments.

8.1.5 Spline Approximation and Optimization

To approximate splines, it is necessary to obtain a consistent parametrization for each characteristic curve. This is achieved by partitioning the curve graph into shortest paths between nodes of degree not equal to 2, where each path serves as a proxy for a curve, determining the parameter coordinates of points along the characteristic curve. For path g, represented as a sequence of graph nodes $q_g = \{q_i\}_{i=1}^{|g|}$, a set of nearest points $P_g \in P_{\text{sharp}}$ is obtained, then projections $\pi^g(p_i)$, $p_i \in P_g$ are computed, and parameter values $u_g = \{u_i\}_{i=1}^{|P_g|}$ are obtained as the cumulative sum of norms $\pi^g(p_i)$ along path g. At the same time, nodes t_g are computed as uniformly distributed parameters; the number of nodes is determined as $\max\left(5, \frac{|g|}{2}\right)$.

Approximating path g with a spline s_g results in a set of control points c_s, defining the precise shape of the spline curve. After spline approximation, the point $P_s(c_s) = \gamma(u_g, P_g, t_g, c_s)$ on spline curve s_g is evaluated. These points ideally should be as distant from point cloud P_g as implied by distance field $\widehat{d}$. To ensure this property, control points are optimized to fit the spline to distance values:

$$\min_c \sum_{i=1}^{|P_g|} \left(\widehat{d}_i - \|p_i - \gamma(u_i, p_i, t_i, c)\|\right)^2, \tag{20}$$

where $p_i \in P_g$, $\widehat{d}_i$ is the corresponding distance value, and $\gamma(u_i, p_i, t_i, c)$ is the point corresponding to p_i evaluated on spline s_g. Additionally, constraints are imposed on spline endpoints to match the polyline endpoints.

The described steps are analogous for closed curves: spline endpoints must meet at a single point, and tangents at endpoint positions must be equal.

8.1.6 Post-processing of Spline-Based Representations

To improve the final result, a post-processing procedure is applied. After this procedure, only curves that fit well the original point cloud remain. The quality of describing a curve from the original point cloud is determined using a quality metric, which is calculated as the F_1 score of Chamfer distances between selected curves and P_{sharp}, and vice versa (21). The lower this score, the better.

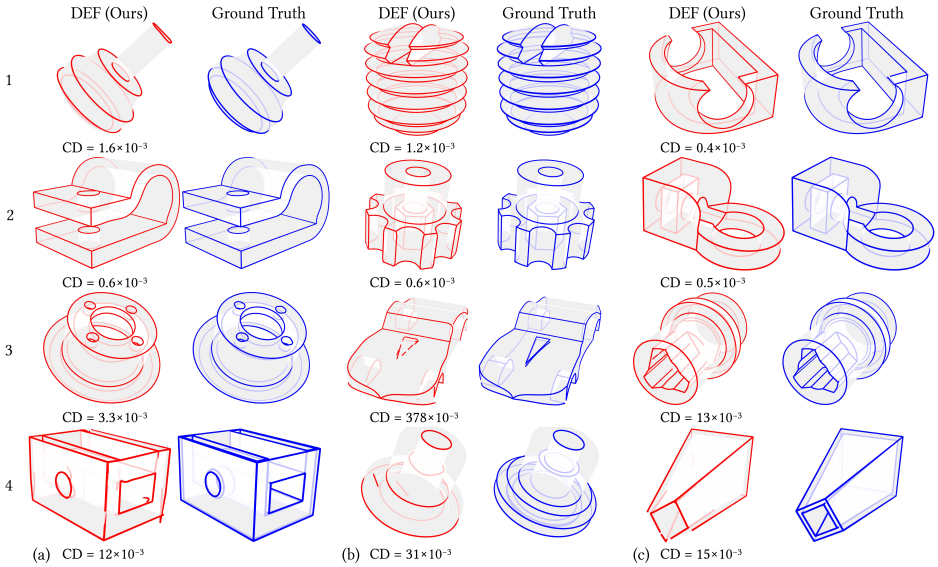


Figure 15: The result of the proposed method for full 3D vectorization with DEF

$$\begin{aligned} \mathrm{CD}_{X\to Y} &= \frac{1}{N_X}\sum_{x\in X}\inf_{y\in Y}\|x-y\|^2, \\ F_1(T_{\text{metric}}) &= \frac{2\cdot\mathbb{1}(\mathrm{CD}_{X\to Y}\leqslant T_{\text{metric}})\cdot\mathbb{1}(\mathrm{CD}_{Y\to X}\leqslant T_{\text{metric}})}{\mathbb{1}(\mathrm{CD}_{X\to Y}\leqslant T_{\text{metric}})+\mathbb{1}(\mathrm{CD}_{Y\to X}\leqslant T_{\text{metric}})}, \end{aligned} \tag{21}$$

where $\mathrm{CD}_{X\to Y}$ is the Chamfer distance from the set of points X to the set of points Y, $\mathbb{1}$ is the indicator function, and T_{metric} is the threshold used to convert real distances into hard 0-1 labels. When using this metric for post-processing, P_{sharp} is defined as one of the sets of points, while the other set represents a discretized set of curves.

Initially, this metric is computed using all curves. Then, each curve is sequentially excluded from the calculation, and the metric is computed again. If, after this, the distance increases or remains the same, the previously excluded curve remains in the final set of curves. Otherwise, the curve is removed.

To conclude the post-processing procedure, curve filtering based on their length is applied. This process involves detecting connected sets of curves, for each of which the number of curves forming it is counted, and the total length of all curves in it is calculated. Then, all curves with a computed length less than a threshold are removed.

The illustration of the complete vectorization system is presented in Figure 15.

8.2 Evaluation of the Proposed Approach

To test the described approach, the vectorization method is run on complete 3D models sampled using $n_v = 128$ views. After setting the parameters, the method was run without manual intervention. The outputs of the method are (1) spline curve parameters and (2) coordinates of endpoints of straight lines.

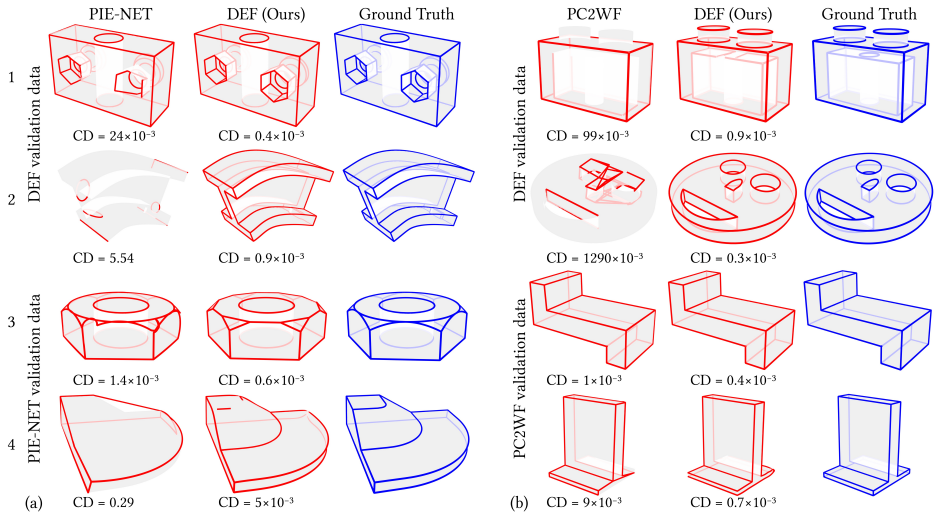


Figure 16: A qualitative comparison of the performance of the proposed method DEF with the results of PIE-NET (a) and PC2WF (b).

For evaluating the model quality, the system was tested on the test set of 68 complete 3D models (DEF-Sim). For comparison, the methods *PIE-NET* and *PC2WF* were also run on the same dataset, and the obtained results were compared with true parametric curves. To compute the metrics, point clouds were obtained by sampling the predicted curves and lines, as well as the set of true curves. Chamfer, Hausdorff, and Sinkhorn distances were computed between predictions and ground truth for the obtained point clouds. The aggregated statistical evaluation of the metrics for our method and *PIE-NET* is provided in Table 3. The qualitative comparison is presented in Figure 16.

Table 3: Comparison of parametric curves obtained using DEF and *PIE-NET*.

Method	CD ↓	HD ↓	SD ↓
PIE-NET [41]	0.97	2.19	0.84
DEF (Ours)	**0.04**	**0.55**	**0.05**

Compared to *PIE-NET*, DEF detects more instances of curves, and thanks to the predicted distance field, the fitting procedure depends not only on point positions but also has no issues with sampling. DEF can fit curves of various types, while *PC2WF* is designed only for straight lines.

Additionally, it was demonstrated that the proposed vectorization system outperforms the method it is based on (*Wireframes*). The qualitative comparison is presented in Figure 13. Improved corner detection and polyline construction based on k nearest neighbors allow the developed method to handle cases of close corners and complex curves. The topology of the curve graph guides the curve fitting stage, and if this topology is inaccurate, it can lead to incorrect curves, as seen in the output of *Wireframes*.

Conclusion

This section presented a method for vectorizing depth images to obtain a parametric representation of a three-dimensional object. In this context, the vector representation of a three-dimensional object is a wireframe, represented by parametric curves.

To obtain the parametric curves, a U-Net-based neural network was applied to each input depth image to predict the distance to sharp features lines. The resulting predictions were then combined into a three-dimensional distance field. Based on this field, parametric curves were extracted, thereby producing a vector representation of the input object.

9 Conclusion

This research is dedicated to the development of methods for solving vectorization tasks of two-dimensional images and depth images for three-dimensional objects using deep learning. To address this task, deep learning methods and optimization algorithms specifically adapted for vectorization tasks of two-dimensional technical drawings and three-dimensional models were developed and applied. Relevant data, including synthetically generated and real images and 3D shapes, were collected and preprocessed, enabling the creation of datasets necessary for training and evaluating the developed models. Neural networks and optimization methods were then developed to accurately and efficiently vectorize input objects.

Within this research, the following results were obtained:

- A method was proposed for obtaining high-quality geometric data for two-dimensional and three-dimensional objects;
- A new system was developed for vectorizing raster images of technical drawings;
- An algorithm was developed for reconstructing parametric models that describe special curves of three-dimensional shapes.

The results of our research represent a significant contribution to the field of deep learning and object vectorization. The developed algorithms achieved high accuracy and efficiency in solving object vectorization tasks. The presented models demonstrate the ability to extract mathematical primitives and relationships between them, representing objects as accurate vector representations. The developed algorithms have great potential for application in various fields, including recognition and analysis of technical drawings, automated modeling, and editing of three-dimensional objects.

Future research could delve into the study of alternative neural network architectures, the development of more sophisticated optimization methods, and the consideration of other types of data for vectorization. It is also important to continue working on improving the quality and generalization ability of models.

This research represents an important step towards the development of efficient methods for object vectorization using deep learning. The developed methods open up new perspectives for various applications and further research, stimulating the advancement of the field as a whole and making a significant contribution to the scientific community.

References

[1] Panos Achlioptas, Olga Diamanti, Ioannis Mitliagkas, and Leonidas Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, pages 40–49, 2018.

[2] Bin Bao and Hongbo Fu. Vectorizing line drawings with near-constant line width. In *2012 19th IEEE International Conference on Image Processing*, pages 805–808. IEEE, 2012.

[3] Mikhail Bessmeltsev and Justin Solomon. Vectorization of line drawings via polyvector fields. *ACM Transactions on Graphics (TOG)*, 38(1):9, 2019.

[4] Jiazhou Chen, Mengqi Du, Xujia Qin, and Yongwei Miao. An improved topology extraction approach for vectorization of sketchy line drawings. *The Visual Computer*, 34(12):1633–1644, 2018.

[5] JiaZhou Chen, Qi Lei, YongWei Miao, and QunSheng Peng. Vectorization of line drawing image based on junction analysis. *Science China Information Sciences*, 58(7):1–14, 2015.

[6] Xuelin Chen, Baoquan Chen, and Niloy J Mitra. Unpaired point cloud completion on real scans using adversarial training. *arXiv preprint arXiv:1904.00069*, 2019.

[7] Robert L. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graph.*, 5(1):5172, jan 1986.

[8] Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, pages 1486–1494, 2015.

[9] Luca Donati, Simone Cesano, and Andrea Prati. A complete hand-drawn sketch vectorization framework. *Multimedia Tools and Applications*, 78(14):19083–19113, 2019.

[10] Vage Egiazarian, Savva Ignatyev, Alexey Artemov, Oleg Voynov, Andrey Kravchenko, Youyi Zheng, Luiz Velho, and Evgeny Burnaev. Latent-space laplacian pyramids for adversarial representation learning with 3d point clouds. In *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*. SCITEPRESS - Science and Technology Publications, 2020.

[11] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in neural information processing systems*, pages 6059–6068, 2018.

[12] Jean-Dominique Favreau, Florent Lafarge, and Adrien Bousseau. Fidelity vs. simplicity: a global approach to line drawing vectorization. *ACM Transactions on Graphics (TOG)*, 35(4):120, 2016.

[13] Jun Gao, Chengcheng Tang, Vignesh Ganapathi-Subramanian, Jiahui Huang, Hao Su, and Leonidas J Guibas. Deepspline: Data-driven reconstruction of parametric curves and surfaces. *arXiv preprint arXiv:1901.03781*, 2019.

[14] Yi Guo, Zhuming Zhang, Chu Han, Wen-Bo Hu, Chengze Li, and Tien-Tsin Wong. Deep line drawing vectorization via line subdivision and topology reconstruction. *Comput. Graph. Forum*, 38:81–90, 2019.

[15] David Ha and Douglas Eck. A neural representation of sketch drawings. In *International Conference on Learning Representations*, 2018.

[16] JH Hannay and JF Nye. Fibonacci numerical integration on a sphere. *Journal of Physics A: Mathematical and General*, 37(48):11591, 2004.

[17] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2 edition, 2004.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Xavier Hilaire and Karl Tombre. Robust and accurate vectorization of line drawings. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):890–904, 2006.

[20] Ruchin Kansal and Subodh Kumar. A vectorization framework for constant and linear gradient filled regions. *The Visual Computer*, 31(5):717–732, 2015.

[21] Tapas Kanungo, Robert M. Haralick, Henry S. Baird, Werner Stuezle, and David Madigan. A statistical, nonparametric methodology for document degradation model validation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1209–1223, 2000.

[22] Byungsoo Kim, Oliver Wang, A Cengiz Öztireli, and Markus Gross. Semantic segmentation for line drawing vectorization using neural networks. In *Computer Graphics Forum*, volume 37, pages 329–338. Wiley Online Library, 2018.

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

[24] Chengze Li, Xueting Liu, and Tien-Tsin Wong. Deep extraction of manga structural lines. *ACM Transactions on Graphics (TOG)*, 36(4):117, 2017.

[25] Chun-Liang Li, Manzil Zaheer, Yang Zhang, Barnabas Poczos, and Ruslan Salakhutdinov. Point cloud gan. *arXiv preprint arXiv:1810.05795*, 2018.

[26] Priyanka Mandikal and Venkatesh Babu Radhakrishnan. Dense 3d point cloud reconstruction using a deep pyramid network. In *WACV*, pages 1052–1060. IEEE, 2019.

[27] Albert Matveev, Alexey Artemov, Denis Zorin, and Evgeny Burnaev. 3d parametric wireframe extraction based on distance fields. In *2021 4th International Conference on Artificial Intelligence and Pattern Recognition*, AIPR 2021, page 316322, New York, NY, USA, 2021. Association for Computing Machinery.

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

[29] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

[30] Haoran Mo, Edgar Simo-Serra, Chengying Gao, Changqing Zou, and Ruomei Wang. General virtual sketching framework for vector line art. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2021)*, 40(4):51:1–51:14, 2021.

[31] Vishaal Munusamy Kabilan, Brandon Morris, and Anh Nguyen. Vectordefense: Vectorization as a defense to adversarial examples. *arXiv preprint arXiv:1804.08529*, 2018.

[32] Patryk Najgebauer and Rafal Scherer. Inertia-based fast vectorization of line drawings. *Comput. Graph. Forum*, 38:203–213, 2019.

[33] Gioacchino Noris, Alexander Hornung, Robert W Sumner, Maryann Simmons, and Markus Gross. Topology-driven vectorization of clean line drawings. *ACM Transactions on Graphics (TOG)*, 32(1):4, 2013.

[34] Open CASCADE Technology OCCT. https://www.opencascade.com/, 2021. Accessed: 2021-06-01.

[35] PrecisionFloorplan. PrecisionFloorplan. http://precisionfloorplan.com. Accessed: 2020-03-05.

[36] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[37] Peter Selinger. Potrace: a polygon-based tracing algorithm. *Potrace (online), http://potrace.sourceforge.net/potrace.pdf (2009-07-01)*, 2003.

[38] Edgar Simo-Serra, Satoshi Iizuka, and Hiroshi Ishikawa. Mastering sketching: adversarial augmentation for structured prediction. *ACM Transactions on Graphics (TOG)*, 37(1):11, 2018.

[39] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[40] Oleg Voynov, Gleb Bobrovskikh, Pavel Karpyshev, Saveliy Galochkin, Andrei-Timotei Ardelean, Arseniy Bozhenko, Ekaterina Karmanova, Pavel Kopanev, Yaroslav Labutin-Rymsho, Ruslan Rakhimov, et al. Multi-sensor large-scale dataset for multi-view 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21392–21403, 2023.

[41] Xiaogang Wang, Yuelang Xu, Kai Xu, Andrea Tagliasacchi, Bin Zhou, Ali Mahdavi-Amiri, and Hao Zhang. Pie-net: Parametric inference of point cloud edges. *Advances in Neural Information Processing Systems*, 33, 2020.

[42] Jiaojiao Zhao, Jie Feng, and Bingfeng Zhou. Image vectorization using blue-noise sampling. In *Imaging and Printing in a Web 2.0 World IV*, volume 8664, page 86640H. International Society for Optics and Photonics, 2013.