Synthesizing 3D Abstractions by Inverting Procedural Buildings with Transformers

Maximilian Dax,[1,*] Jordi Berbel,[2] Jan Stria,[3] Leonidas Guibas,[2] Urs Bergmann[3]
[1]Max Planck Institute for Intelligent Systems, Tübingen, Germany, [2]Google DeepMind, [3]Google
[*]Work done at Google. Correspondence: maximilian.dax@tuebingen.mpg.de, ursbergmann@google.com

Abstract

We generate abstractions of buildings, reflecting the essential aspects of their geometry and structure, by learning to invert procedural models. We first build a dataset of abstract procedural building models paired with simulated point clouds and then learn the inverse mapping through a transformer. Given a point cloud, the trained transformer then infers the corresponding abstracted building in terms of a programmatic language description. This approach leverages expressive procedural models developed for gaming and animation, and thereby retains desirable properties such as efficient rendering of the inferred abstractions and strong priors for regularity and symmetry. Our approach achieves good reconstruction accuracy in terms of geometry and structure, as well as structurally consistent inpainting.

1. Introduction

Abstract visual representations aim to capture the key geometric and structural properties of an object. Abstractions are useful in many applications as they facilitate perceptual comparisons and understanding. For buildings, use cases for abstract representations range from 3D mapping for navigation to the generation of synthetic environments for training of deep learning agents. However, inferring abstractions based on sensor data at a desired level of detail is a challenging problem. Traditional approaches are largely based on geometric simplification and optimization, while learning approaches face a lack of training data.

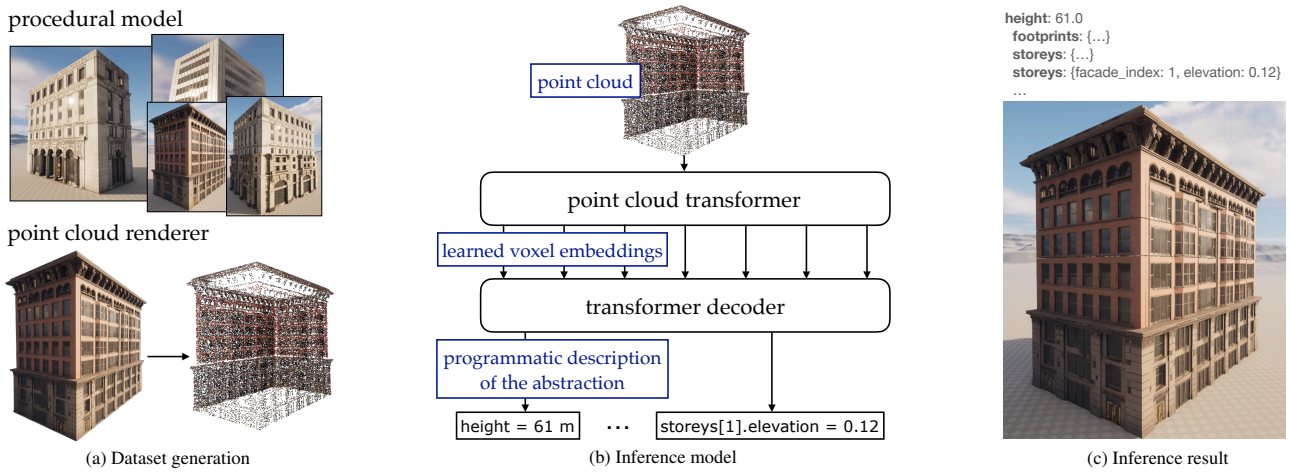
We here propose a model to infer abstractions from point cloud data. While manually labeled large-scale datasets for this task are costly and thus unavailable, there is an abundance of high-quality 3D building simulators. Specifically, the gaming and animation industries have developed models to sample synthetic environments and render corresponding 3D representations at various levels of complexity—the inverse direction of what we aim to achieve (Fig. 1). Our framework inverts such 3D models with a transformer model [47] which takes point clouds as input and predicts corresponding abstractions (Sec. 2). Training is fully supervised, based on a dataset of procedural buildings paired with corresponding point cloud simulations. We develop various technical components tailored to the generation of abstractions. This includes the design of a programmatic language to efficiently represent abstractions, its combination with a technique to guarantee transformer outputs consistent with the structure imposed by this language, and an encoder-decoder architecture for the inference model.

Our approach achieves accurate in-distribution geometric and structural reconstruction, and structurally consistent inference for incomplete inputs (Sec. 3). We find that the main limitations are attributed to constraints of the procedural model, not the inference framework itself. While this can be partially mitigated by data augmentation, our results suggest that procedural model advancements will be crucial for real-world applicability. Based on our analysis, we suggest to make procedural models more flexible to better suit inversion (Sec. 4).

Related Work. There is vast literature on 3D reconstruction or abstraction from sensor data. A traditional approach to abstraction is geometry simplification—reducing the number of geometric elements used to represent the model [13, 20]. Other works perform geometry abstraction by fitting geometric primitives (e.g., boxes or cylinders) to raw data (e.g., point clouds) [9, 22, 23, 27, 51]. Several works aim to detect symmetries and regularities [31, 37, 43] or generate structure preserving abstractions [12, 21, 28, 48]. Finally, generative AI has been explored to create geometry from various inputs [2, 32, 38, 41]. In particular, several works represent CAD models and procedural descriptions with a structured, domain specific language—and aim to use language tools for CAD generation [10, 36, 40].

2. Method

We train an inference model that takes a building point cloud x as input and infers a parametric description θ, an abstraction, of the corresponding building. When training



(a) Dataset generation (b) Inference model (c) Inference result

Figure 1. (a) We generate a synthetic training dataset by first unconditionally sampling building abstractions with a procedural model and then composing corresponding point clouds for each abstraction. (b) The inference model encodes the input with a point cloud transformer operating on point cloud voxels (Sec. 2.3). It then employs a language transformer to predict the corresponding abstraction in terms of our custom programmatic language (Sec. 2.1). (c) The transformer output is parsed as a Protocol Buffer and can be rendered in Unreal Engine 5.

the model, we aim to combine *structural knowledge* (e.g. a building is made of storeys/floors, style and position of windows, etc.) and *geometric fit*.

We formalize these notions in a Bayesian framework, and represent *structural knowledge* in the *prior* distribution $p(\theta)$. We further represent the *geometric fit* in terms of the distribution over point clouds x conditional on abstractions θ—the *likelihood* $p(x|\theta)$. With these definitions, the Bayesian posterior $p(\theta|x) \propto p(\theta)p(x|\theta)$ represents the distribution over abstractions θ for a given point cloud x. We solve this Bayesian inference problem by training an inference model $q(\theta|x)$ to approximate the posterior $p(\theta|x)$. An abstraction for a point cloud x can then be inferred by sampling from the trained model $\theta \sim q(\theta|x)$.

Below, we define $p(\theta)$ in terms of a procedural building model and $p(x|\theta)$ in terms of a renderer (Sec. 2.1), introduce the training objective for optimization of $q(\theta|x)$ (Sec. 2.2) and describe the architecture of $q(\theta|x)$ (Sec. 2.3).

2.1. Synthetic training data

Prior. We employ a procedural building model adapted from the Unreal Engine 5 *City Samples* [1] demo. This model samples abstract buildings θ by placing a collection of building assets (e.g., facade segments) according to a set of rules (appendix Fig. 4), and hence serves as our implicit prior distribution over buildings $p(\theta)$. Both, assets and rules, have been carefully designed to capture styles of buildings of San Francisco, Chicago and New York. In total there are 911 unique assets, and a typical building is composed of around $10 - 40$ different asset types. We *represent abstractions* in a custom data format based on Protocol Buffers [46] (Fig. 5), which we designed to facilitate representation of regularities and significantly minimize redundancy, thereby simplifying inference. It is chosen as a hierarchical format, starting from macroscopic structure (height, footprints), and then goes into more detail (storeys, facade description etc). Importantly, identical structures can refer to the same description – e.g. different storeys can refer to the same facade. For more details, see appendix A.2.

Likelihood. The likelihood $p(x|\theta)$ models the distribution over point clouds consistent with a given abstraction θ. We represent the likelihood implicitly: for a given abstraction θ we sample $x \sim p(x|\theta)$ by composing surface points from all assets of a building into a joint point cloud, filtering interior points, and adding Gaussian noise with mean 0 and variance σ^2 to each point. As log-likelihood maximization with Gaussian kernels corresponds to mean squared error minimization, this likelihood approximates a geometric distance between point cloud and inferred geometry.

Dataset. We generate a dataset with 341721 pairs of abstractions and point clouds, hence samples from the joint distribution $(\theta, x) \sim p(\theta, x)$ (see Fig. 1a for examples). We augment the asset colors in the dataset by adding uncorrelated Gaussian noise with a standard deviation of 0.15 in the HSV color system to half of the dataset.

2.2. Training objective

We train the network $q(\theta|x)$ using simulation-based inference [6] ideas (see appendix A.1 for details) with the loss

$$L = \mathbb{E}_{x \sim p(x|\theta),\, \theta \sim p(\theta)} \left[-\log q(\theta|x) \right], \tag{1}$$

across the dataset of simulated buildings from Sec. 2.1. This objective formally corresponds to minimization of the Kullback–Leibler divergence $D_{\mathrm{KL}}\left(p(\theta|x) \parallel q(\theta|x)\right)$. With suf-

ficient training data and network capacity, the inference network $q(\theta|x)$ will therefore become an accurate estimator of $p(\theta|x)$ [33]. After training, the inference network thus inverts the forward model (i.e., the procedural model and point cloud renderer), and an abstraction for a point cloud x can be generated by sampling $\theta \sim q(\theta|x)$.

2.3. Model Architecture

We now define the density estimator $q(\theta|x)$ which maps a point cloud x to a conditional distribution over abstractions θ. We use a transformer-based encoder-decoder model.

Tokenization of Protocol Buffers. To obtain a transformer-compatible data format for the abstraction programs θ we convert these to sequences of tokens. We follow [11], which offers a method to bijectively map any Protocal Buffer to a token sequence. Importantly, at inference time, this allows to mask invalid tokens, ensuring syntactically valid Protocol Buffers. For details see appendix A.2.

Encoder. We subdivide the point cloud into cubes of length 7 meters, resulting in n_{v} smaller point clouds $\{\hat{x}_1, \ldots, \hat{x}_{n_{\text{v}}}\}$. Each voxel point cloud is embedded into a feature space $z_i = g(\hat{x}_i) \in \mathbb{R}^{512}$, where g is a PointCloudTransformer [17] (see Tab. 3 for hyperparameters) and $\hat{x}_i$ includes point coordinates and color information. We provide the voxel positions by adding sinusoidal positional embeddings [29] to the PointCloudTransformer's output.

Decoder. We employ a standard transformer decoder [47] (see Tab. 3 for hyperparameters). The decoder is conditioned on the embeddings z_i via cross-attention, which enables spatially global information integration.

Training. We train the encoder-decoder model end-to-end for 10^5 steps, with batch size 16, employing AdamW [24] with $\beta_1 = 0.9, \beta_2 = 0.98$ and a learning rate of $5 \cdot 10^{-4}$ after a linear warm up for 10^3 steps. We apply dropout on the voxel embeddings with rates varying between 0 and 0.8, to force the model to infer missing information based on regularity in the data. We regularize with additive Gaussian noise on the point cloud with standard deviations σ between 0 m and 0.5 m.

3. Results

We first evaluate our trained model on a hold-out subset of 17086 point clouds from the simulated dataset. For each point cloud, we compare the inferred abstraction to the corresponding ground truth, both in terms of structural properties and geometric fit (Fig. 2). Structural high-level variables, such has the predicted number of storeys of the buildings, the number of facades, and asset precision and recall, are inferred correctly with high accuracies of above 95%. As color/materials are augmented for some assets of a building, the intersection over union (IoU) of the predicted and the ground truth modifications is calculated (94.2%), and the Euclidean distance in HSV color space is evaluated on the intersecting assets. The deviation between inferred building geometries and the input point cloud is around 10 cm (in the absence of point cloud noise). We conclude that our model achieves excellent in-distribution reconstruction.

We next test properties of the model under point cloud modifications. Specifically, we drop large blocks of the point clouds (a) at random or (b) systematically; and we (c) split the point clouds and move both parts away from each other (Fig. 3). These manipulations lead to missing information, and (c) additionally could lead to out-of-distribution buildings. We find that the inference model successfully reconstructs missing information by leveraging regularity and symmetry priors from the procedural model. We further observe that inferred buildings for (c) are not simply stretched versions of the originals, but contain additional asset instances (e.g., more windows). While all buildings are visually neat, we observe occasional visual artefacts and slight inaccuracies of asset placements.

4. Discussion

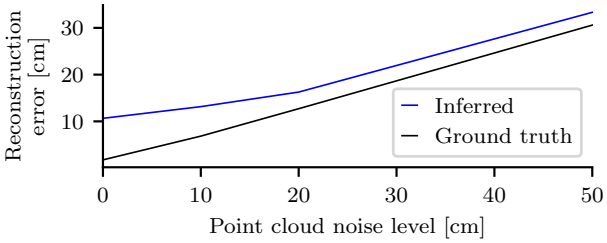
Our framework casts abstraction into an inverse problem, with the forward direction defined by a procedural model and a point cloud renderer. The inference network is trained to invert this forward direction, thereby converting point cloud data to the procedural model space. In this space, buildings are composed of assets, which can directly serve as abstract representations: they allow for high-level editing and rendering at different levels of detail.

Limitations. As a central limitation, our approach is constrained to the scope of the forward direction, hence is limited to buildings that are captured by the procedural model. While procedural models generate *diverse* buildings, they usually are not *comprehensive*, because for simulation of realistic environments it is not necessary to capture *all* buildings within the targeted distribution. Therefore, the procedural model used here is limited to only a subset of buildings. Application to real buildings further requires the point clouds to be consistent with real point cloud measurements (e.g., with photogrammetry, lidar). We designed our renderer to capture some of the main features of realistic point clouds (e.g., only rendering points from the surface, added noise), but more realistic settings require further extension (e.g., local variation of noise levels). Additionally, it may be necessary to explicitly account for domain shifts between simulated and real data within our model [5, 49].

Strengths. Our simulation-based approach has various advantages compared to other methods. First, it enables training without human annotations with large amounts of synthetic data. Second, regularity priors implemented in the procedural model are captured by the trained inference model. Buildings are highly regular objects, and such priors enable inference of missing information. Third, our approach provides an inductive bias to prioritize structure over

Accuracy number of storeys	97.9%
Accuracy number of facades	97.9%
Accuracy storeys structure	95.8%
Assets: precision	99.4%
Assets: recall	98.7%
IoU Material Variations	94.2%
L2 HSV Color Distance	0.083

(a) Structural evaluation



(b) Geometric evaluation

Figure 2. Reconstruction performance of the transformer model. (a) Performance on various structural variables (see appendix Tab. 4 for definitions). (b) Reconstruction error as a function of the point cloud noise level, measured in terms of the mean deviation between noisy input point clouds and building geometries. The reconstruction error of the inferred buildings (blue) is only slightly larger than the reconstruction error of the ground truth buildings (black). Naturally, both grow with increasing point cloud noise level.

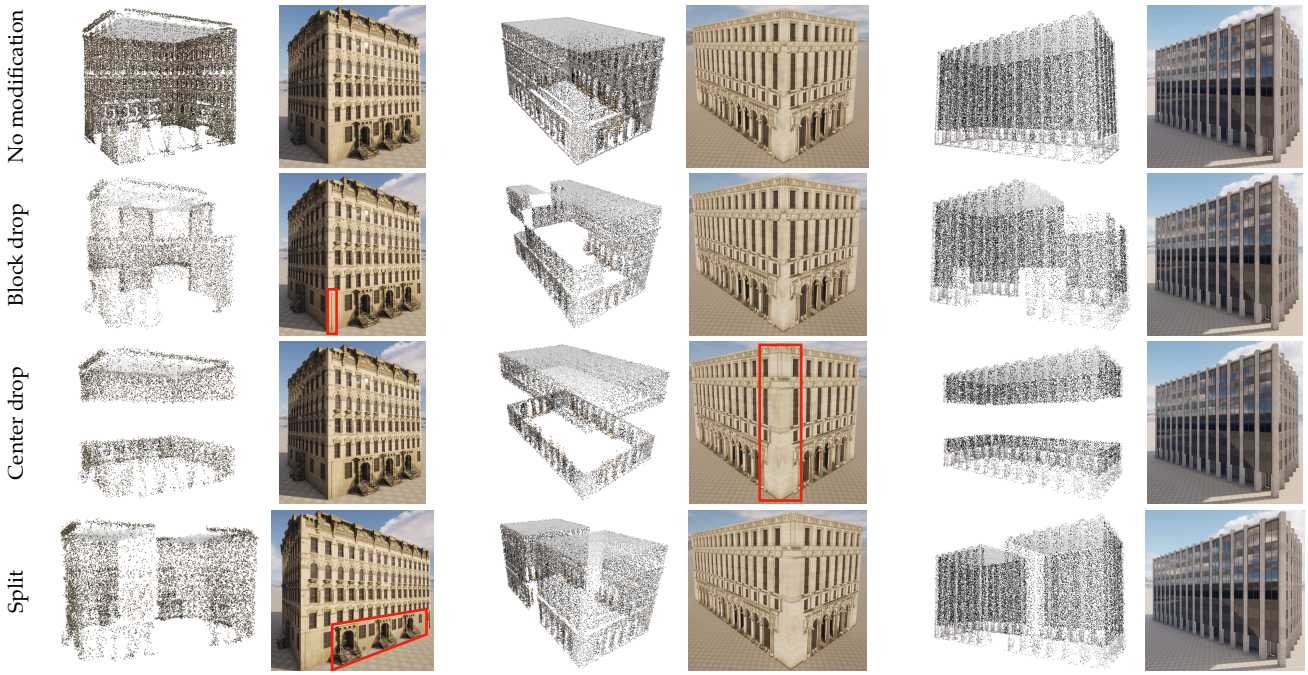


Figure 3. Inference results with modified point clouds. We drop random blocks (second row), a single large block in the center (third row), or split the point cloud in the center and move both halves away from each other (fourth row). The inference model successfully reconstructs the associated building. With the split modification, the missing information is filled with additional asset instances, resulting in additional columns of windows. We occasionally observe artefacts (red), like gaps between assets or slight offsets in window positions.

geometric accuracy, as it is defined on the abstract descriptions (as opposed to a geometric reconstruction loss). Indeed, we observed that inferred buildings sometimes did not optimally capture the input point cloud, but always looked structurally self-consistent. This is desirable, as structure is typically more important than rigorous geometric accuracy for abstractions. Fourth, the native representation of procedural models is optimized for extremely efficient rendering, which is inherited by our framework. Fifth, our approach enables human-in-the-loop interaction between procedural modeling and its inversion. This could help to quantitatively assess procedural models at scale, validating how well they capture real buildings and identifying improvements.

Optimizing procedural models for inversion. Designing procedural models with inversion in mind likely improves the scope and performance of our framework. E.g., our experiments suggest a need for more flexible asset selection and placement. This could alleviate inconsistencies between asset placement and augmented point clouds (e.g., Fig. 3, split manipulation). Parametric assets, e.g. variable window sizes, would further enhance flexibility, which however also weakens the regularity prior. Such modifications to procedural models will be crucial for future applicability of our framework to real data.

References

[1] Epic Games, Inc.: City sample buildings, 2021. 2

[2] Armen Avetisyan, Christopher Xie, Henry Howard-Jenkins, Tsun-Yi Yang, Samir Aroudj, Suvam Patra, Fuyang Zhang, Duncan Frost, Luke Holland, Campbell Orme, et al. Scenescript: Reconstructing scenes with an autoregressive structured language model. *arXiv preprint arXiv:2403.13064*, 2024. 1

[3] Mark A Beaumont, Wenyang Zhang, and David J Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–2035, 2002. A1

[4] Johann Brehmer, Kyle Cranmer, Gilles Louppe, and Juan Pavez. Constraining effective field theories with machine learning. *Physical review letters*, 121(11):111801, 2018. A1

[5] Patrick Cannon, Daniel Ward, and Sebastian M Schmon. Investigating the impact of model misspecification in neural simulation-based inference. *arXiv preprint arXiv:2209.01845*, 2022. 3

[6] Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020. 2, A1

[7] Maximilian Dax, Stephen R. Green, Jonathan Gair, Jakob H. Macke, Alessandra Buonanno, and Bernhard Schölkopf. Real-Time Gravitational Wave Science with Neural Posterior Estimation. *Phys. Rev. Lett.*, 127(24):241103, 2021. A1

[8] Maximilian Dax, Stephen R. Green, Jonathan Gair, Michael Pürrer, Jonas Wildberger, Jakob H. Macke, Alessandra Buonanno, and Bernhard Schölkopf. Neural Importance Sampling for Rapid and Reliable Gravitational-Wave Inference. *Phys. Rev. Lett.*, 130(17):171403, 2023. A1

[9] Tao Du, Jeevana Priya Inala, Yewen Pu, Andrew Spielberg, Adriana Schulz, Daniela Rus, Armando Solar-Lezama, and Wojciech Matusik. Inversecsg: Automatic conversion of 3d models to csg trees. *ACM Trans. Graph.*, 37(6), 2018. 1

[10] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. *Advances in Neural Information Processing Systems*, 34:5885–5897, 2021. 1

[11] Yaroslav Ganin, Sergey Bartunov, Yujia Li, Ethan Keller, and Stefano Saliceti. Computer-aided design as language. *Advances in Neural Information Processing Systems*, 34:5885–5897, 2021. 3, A2

[12] Xifeng Gao, Kui Wu, and Zherong Pan. Low-poly mesh generation for building models. In *ACM SIGGRAPH 2022 Conference Proceedings*, New York, NY, USA, 2022. Association for Computing Machinery. 1

[13] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, page 209–216, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 1

[14] Tomas Geffner, George Papamakarios, and Andriy Mnih. Compositional score modeling for simulation-based inference. In *Proceedings of the 40th International Conference on Machine Learning*, pages 11098–11116. PMLR, 2023. A1

[15] Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. Training deep neural density estimators to identify mechanistic models of neural dynamics. *Elife*, 9:e56261, 2020. A1

[16] David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*, pages 2404–2414. PMLR, 2019. A1

[17] Meng-Hao Guo, Jun-Xiong Cai, Zheng-Ning Liu, Tai-Jiang Mu, Ralph R Martin, and Shi-Min Hu. Pct: Point cloud transformer. *Computational Visual Media*, 7:187–199, 2021. 3, A2

[18] Joeri Hermans, Volodimir Begy, and Gilles Louppe. Likelihood-free mcmc with approximate likelihood ratios. *arXiv preprint arXiv:1903.04057*, 10, 2019. A1

[19] Joeri Hermans, Nilanjan Banik, Christoph Weniger, Gianfranco Bertone, and Gilles Louppe. Towards constraining warm dark matter with stellar streams through neural simulation-based inference. *Monthly Notices of the Royal Astronomical Society*, 507(2):1999–2011, 2021. A1

[20] Hugues Hoppe. Progressive meshes. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, page 99–108, New York, NY, USA, 1996. Association for Computing Machinery. 1

[21] Jingwei Huang, Shanshan Zhang, Bo Duan, Yanfeng Zhang, Xiaoyang Guo, Mingwei Sun, and Li Yi. Arrangementnet: Learning scene arrangements for vectorized indoor scene modeling. *ACM Trans. Graph.*, 42(4), 2023. 1

[22] R. Kenny Jones, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. Shapecoder: Discovering abstractions for visual programs from unstructured primitives. *ACM Trans. Graph.*, 42(4), 2023. 1

[23] Lingxiao Li, Minhyuk Sung, Anastasia Dubrovina, Li Yi, and Leonidas J. Guibas. Supervised fitting of geometric primitives to 3d point clouds. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2647–2655, 2019. 1

[24] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 3

[25] Jan-Matthis Lueckmann, Pedro J Gonçalves, Giacomo Bassetto, Kaan Öcal, Marcel Nonnenmacher, and Jakob H Macke. Flexible statistical inference for mechanistic models of neural dynamics. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pages 1289–1299, 2017. A1

[26] Jan-Matthis Lueckmann, Giacomo Bassetto, Theofanis Karaletsos, and Jakob H Macke. Likelihood-free inference with emulator networks. In *Symposium on Advances in Approximate Bayesian Inference*, pages 32–53. PMLR, 2019. A1

[27] Eric-Tuan Lê, Minhyuk Sung, Duygu Ceylan, Radomir Mech, Tamy Boubekeur, and Niloy J. Mitra. Cpfn: Cascaded primitive fitting networks for high-resolution point clouds. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 7438–7446, 2021. 1

[28] Ravish Mehra, Qingnan Zhou, Jeremy Long, Alla Sheffer, Amy Gooch, and Niloy J. Mitra. Abstraction of man-made shapes. *ACM Trans. Graph.*, 28(5):1–10, 2009. 1

[29] Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021. 3

[30] Benjamin K Miller, Christoph Weniger, and Patrick Forré. Contrastive neural ratio estimation. *Advances in Neural Information Processing Systems*, 35:3262–3278, 2022. A1

[31] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Partial and approximate symmetry detection for 3d geometry. In *ACM SIGGRAPH 2006 Papers*, page 560–568, New York, NY, USA, 2006. Association for Computing Machinery. 1

[32] Gen Nishida, Ignacio Garcia-Dorado, Daniel G. Aliaga, Bedrich Benes, and Adrien Bousseau. Interactive sketching of urban procedural models. *ACM Trans. Graph.*, 35(4), 2016. 1

[33] George Papamakarios and Iain Murray. Fast ε-free inference of simulation models with bayesian conditional density estimation. *Advances in neural information processing systems*, 29, 2016. 3, A1

[34] George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 837–848. PMLR, 2019. A1

[35] George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *Journal of Machine Learning Research*, 22(57):1–64, 2021. A1

[36] Wamiq Para, Shariq Bhat, Paul Guerrero, Tom Kelly, Niloy Mitra, Leonidas J Guibas, and Peter Wonka. Sketchgen: Generating constrained cad sketches. In *Advances in Neural Information Processing Systems*, pages 5077–5088. Curran Associates, Inc., 2021. 1

[37] Mark Pauly, Niloy J. Mitra, Johannes Wallner, Helmut Pottmann, and Leonidas J. Guibas. Discovering structural regularity in 3d geometry. In *ACM SIGGRAPH 2008 Papers*, New York, NY, USA, 2008. Association for Computing Machinery. 1

[38] Aleksander Płocharski, Jan Swidzinski, Joanna Porter-Sobieraj, and Przemyslaw Musialski. Neuro-symbolic transformation of architectural facades into their procedural representations. In *ACM SIGGRAPH 2024 Posters*, pages 1–2. 2024. 1

[39] Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning*, pages 1530–1538, 2015. A1

[40] Ari Seff, Wenda Zhou, Nick Richardson, and Ryan P Adams. Vitruvion: A generative model of parametric cad sketches. *arXiv preprint arXiv:2109.14124*, 2021. 1

[41] M. Shabani, S. Hosseini, and Y. Furukawa. Housediffusion: Vector floorplan generation via a diffusion model with discrete and continuous denoising. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5466–5475, Los Alamitos, CA, USA, 2023. IEEE Computer Society. 1

[42] Louis Sharrock, Jack Simons, Song Liu, and Mark Beaumont. Sequential neural score estimation: Likelihood-free inference with conditional score based diffusion models. *arXiv preprint arXiv:2210.04872*, 2022. A1

[43] Zeyun Shi, Pierre Alliez, Mathieu Desbrun, Hujun Bao, and Jin Huang. Symmetry and orbit detection via lie-algebra voting. *Computer Graphics Forum*, 35(5):217–227, 2016. 1

[44] Scott A Sisson, Yanan Fan, and Mark A Beaumont. Overview of abc. In *Handbook of approximate Bayesian computation*, pages 3–54. Chapman and Hall/CRC, 2018. A1

[45] Owen Thomas, Ritabrata Dutta, Jukka Corander, Samuel Kaski, and Michael U. Gutmann. Likelihood-free inference by ratio estimation, 2020. A1

[46] Kenton Varda. Google protocol buffers: Google's data interchange format. *Technical report*, 2008. 2

[47] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 3, A2

[48] Yael Vinker, Ehsan Pajouheshgar, Jessica Y. Bo, Roman Christian Bachmann, Amit Haim Bermano, Daniel Cohen-Or, Amir Zamir, and Ariel Shamir. Clipasso: Semantically-aware object sketching. *ACM Trans. Graph.*, 41(4), 2022. 1

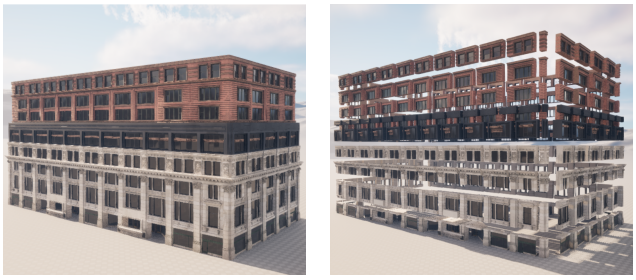
[49] Antoine Wehenkel, Juan L Gamella, Ozan Sener, Jens Behrmann, Guillermo Sapiro, Marco Cuturi, and Jörn-Henrik Jacobsen. Addressing misspecification in simulation-based inference through data-driven calibration. *arXiv preprint arXiv:2405.08719*, 2024. 3

[50] Jonas Bernhard Wildberger, Maximilian Dax, Simon Buchholz, Stephen R Green, Jakob H. Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. A1

[51] Q. Wu, K. Xu, and J. Wang. Constructing 3d csg models from 3d raw point clouds. *Computer Graphics Forum*, 37 (5):221–232, 2018. 1

Synthesizing 3D Abstractions by Inverting Procedural Buildings with Transformers

Supplementary Material



(a) Procedural building (b) Assets in procedural building

Figure 4. A procedural building is generated by placing a set of assets according to a set of handcrafted rules.

Description	Protocol Buffer fields
Noise level*	`noise_level`
Absolute coordinates*	`height`,`x`, `y`,
Relative coordinates*	`offset`, `elevation`
Asset scales* & rotations*	`scale_x`, `scale_y`, `quaternion_3`, `quaternion_4`
Asset index	`cell_type`
Pointer index	`facade_index`, `footprint_index`

Table 1. Token groups for the fields of the Protocol Buffer representing the programmatic language for abstractions θ (omitting prefixes, see Fig. 5 and F for complete definitions). We use continuous groups (marked with asterisk) for absolute coordinates of the building, for relative coordinates within a storey and for scale and rotation parameters of assets. We use discrete integer groups for determining asset types from an external asset index and for pointer indices within the Protocol Buffer.

A.1. Relation to simulation-based inference

Our framework for procedural inversion is closely related to simulation-based inference [6] (SBI). SBI is a paradigm for solving inverse problems entirely based on *simulations* from the forward model, without requiring access to underlying densities. SBI has become a standard tool for scientific inference [4, 7, 15, 19], and a variety of methods have been developed in recent years [3, 8, 14, 16, 18, 25, 26, 30, 33, 34, 42, 44, 45, 50]. Our approach can be seen as a variant of neural posterior estimation [16, 25, 33] (NPE), an SBI method which directly targets the Bayesian posterior.

However, compared to common scientific use cases, our application of SBI to infer abstractions of buildings has various distinct properties. First, our inference domain (i.e., abstractions θ) is represented in terms of a programmatic language while in scientific applications it is typically a low-dimensional vector space. Our density estimator $q(\theta|x)$ is thus parameterized with a transformer model as opposed to a continuous density estimator such as a normalizing flow [35, 39]. Second, our prior is a complicated procedural model whereas in scientific applications it is typically a simple distribution and the complexity of the problem is dominated by the likelihood. Third, we are ultimately interested in generating individual high-likelihood abstractions, not necessarily in the distributional properties of the posterior such as correlations and uncertainties.

A.2. Technical details

High-level Description of the Protocol Buffers. A building is chosen to be represented in a hierarchical description with a Protocol Buffer. This hierarchical format starts with the macroscopic structure in terms of the building `height` and (possibly multiple) `footprints` layouts. It then specifies a series of `storeys`. A storey is defined in terms of its `elevation` and by linking to a `facade`. A facade is specified by a sequence of `cells_patterns`, which in turn contain collections of asset instances. The nested structure of this data format provides a natural way of representing recurring patterns. For example, multiple storeys can link to the same facade, and multiple facades can link to the same footprint layout. Where appropriate, we define coordinates relative to previously specified data (Tab. 1). For example, asset positions are defined in relative terms to the associated footprint line segment, and their horizontal scales are computed as the difference to respective previously placed assets, enforcing consistency between asset placements and the building footprints. To account for (rare) cases, where asset properties differ from the derived choice, we add a custom `cell_modifier`. Finally, each asset uses one or more materials, and the building Protocol Buffer contains a set of color modifiers for material and asset pairs. If unset for an asset's material, the standard color of the material is used.

Complete definition of the Protocol Buffers. Fig. 5 defines the Protocol Buffer we used, and Fig. 6 displays the additional omitted definitions. A Footprint is parameterized as a planar polygon. The line segments of the footprint determine the layout of the corresponding facades, along

Description	Range	Resolution
Noise level [m]	$[0, 1]$	0.01
Absolute coordinates [m]	$[-100, 100]$	0.1
Relative coordinates	$[0, 1]$	0.005
Asset scales & rotations	$[-5, 5]$	0.01

Table 2. Discretization of continuous variables for tokenization. Noise level and absolute coordinates are specified in meters.

which the asset cells are arranged.

A CellModifier can be used to change the scale and rotation of a Cell instance (which wraps a single asset instance). In most cases, scales and rotations can be uniquely determined based on heuristic rules (e.g., using the cell position within the footprint and coordinates of neighbouring cells). In our definition of the Protocol Buffer, we thus do not specify cell scales and rotation by default, and instead derive them based on such heuristics. This removes redundancy from the Protocol Buffer representation and also decreases the length of the corresponding tokenized sequences. However, in some edge cases, these rules don't provide a unique result. In such cases, the optional CellModifier can be used to overwrite scale and rotation parameters.

Finally, MaterialVariation optionally modifies the material type and color parameters of a specific asset. When applied, this modifies all asset instances of the specified type in the same way.

Tokenization of Protocol Buffers. To obtain a transformer-compatible data format for the abstraction programs θ we need to convert these to sequences of tokens. Following [11], we leverage the known structure of the Protocol Buffers to obtain an efficient conversion scheme. We first define token groups for the different data types in the Protocol Buffer (Tab. 1) and assign to each group a set of tokens that covers all potential values. We further use a set of special tokens in the sequence to navigate within the Protocol Buffer whenever the next step is not unique. This includes end tokens for `repeated` fields and selector tokens for `optional` and `oneof` fields (see [11] for details).

This scheme can convert any Protocol Buffer representation to a sequence of tokens. Conversely, at inference time we can mask out invalid options for the next token at each step of the autoregressive transformer prediction, such that any inferred sequence can be converted back into the Protocol Buffer representation. This bijective mapping between Protocol Buffers and token sequences enables straightforward application of standard language modeling techniques for the estimation of the programs θ representing building abstractions.

Conversion of the Protocol Buffers into the tokenized representation requires assignment of discrete tokens to each possible value for each Protocol Buffer field. Following [11], we group fields with similar functions together (Tab. 1; e.g., {`facade_index`,`footprint_index`}, which are both used to cross-reference objects within the Protocol Buffer, or {`height`, `x`, `y`}, which all refer to spatial coordinates). Continuous values further need to be discretized (Tab. 2).

Encoder (PointCloudTransformer [17])	
Voxel size	7 m × 7 m × 7 m
Max. # points per voxel	300
# attention layers	4
# attention heads	4
Dimension of QKV	256
Output dimension	512

Decoder (Transformer [47])	
Size of context window	2048
# attention layers	12
# attention heads	8
Dimension of QKV	512

Table 3. Hyperparameters of the inference model.

A.3. Additional results

Tab. 4 defines the structural evaluation quantities from Tab. 2. Fig. 7 expands over Fig. 2, also showing the reconstruction accuracy for input point clouds with voxel dropout. The reconstruction error is a bit higher in case of dropout, which indicates that the inference model uses global point cloud information, which also improves its local estimates.

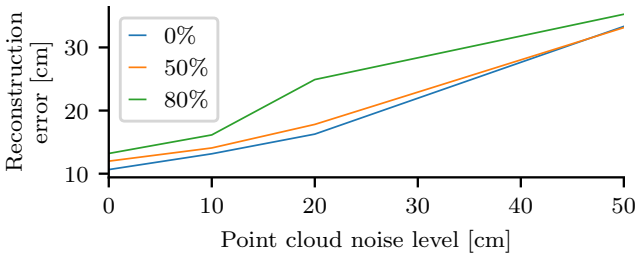


Figure 7. Reconstruction error as a function of the point cloud noise level. Compared to Fig. 2, this also shows results for input point clouds with 50% and 80% dropout augmentation of point cloud voxels.

```
// A cell wraps a single asset instance, adding a positional parameter and optional modifiers.
message Cell {
  int32 cell_type;                                    // Enumeration into an external index
  float offset;                                       // Offset along the corresponding linesegment
  optional CellModifier cell_modifier;                // Modifier that overwrites scales and roations
}

// A cell pattern combines a sequence of asset cells along one line segment of the footprint
message CellsPattern {
  repeated Cell cells;                                // Sequence of cells
}

// A facade is made by defining the cells along all line-segments of a footprint
message Facade {
  repeated CellsPattern cells_pattern;                // A pattern of asset instances
  int32 footprint_index;                              // Index to the footprint array
}

// A storey is made by combining a footprint layout with a facade pattern
message Storey {
  int32 facade_index;                                 // Sets which facade to use for this storey
  float elevation;                                    // Relative in [0, 1] * Building.height
}

message Building {
  float noise_level;
  float height;                                       // Total height of the building
  repeated Footprint footprints;                      // Footprint polygons
  repeated Storey storeys;                            // Building storeys
  repeated Facade facades;                            // Unique facade patterns
  repeated MaterialVariation material_variations;     // Material variation (colors, material type)
}
```

Figure 5. Custom format for the representation of abstract buildings. This hierarchically combines asset instances ("Cells") into recurring patterns ("CellsPattern"). These patterns are combined into facade instances, which can in turn be linked by the building storeys. Finally, a building is composed by combining such storeys along with variables characterizing the high-level geometry (height, footprint polygons) and the point cloud noise level. Definitions of CellModifier, Footprint and MaterialVariation are provided in Fig. 6.

Name	Description
Accuracy number of storeys	Fraction of inferred buildings with correct number of storeys.
Accuracy number of facades	Fraction of inferred buildings with correct number of distinct facades.
Accuracy storeys structure	Fraction of storeys which link to the correct facade.
Assets: precision	Fraction of distinct assets in inferred building, which are also present in ground truth building.
Assets: recall	Fraction of distinct assets in ground truth building, which are also present in inferred building.
IoU Material Variations	Intersection over union of modified materials in inferred and ground truth building.
L2 HSV Color Distance	L2 distance between ground truth and inferred material colors in HSV basis.

Table 4. Description of metrics used in Fig. 2.

```
// 2D vector
message Vector2d {
  double x;
  double y;
}

// 2D polygon for building footprints (different building storeys can have different footprints)
message Footprint {
  repeated Vector2d points;
}

// Cell modifier to overwrite default scales and rotations of assets
message CellModifier {
  float scale_x_overwrite;                          // Used only if can't predict.
  float scale_y_overwrite;                          // Used only if can't predict.
  float quaternion_3_overwrite;                     // In case orientation is wrong.
  float quaternion_4_overwrite;                     // In case orientation is wrong.
}

// List of material types for assets
enum MaterialBaseType {
  UNSPECIFIED;
  GLASS;
  BRICK;
  PAINTEDMETAL;
  LIMESTONE;
  PAINTEDSTONE;
  GRANITE;
  WOOD;
  METAL;
  COPPER;
  BRASS0;
  BRUSHED1;
  ASPHALT2;
  DIRTY3;
  ROOFTOP4;
  SIDEWALK5;
  BLOCK6;
}

// Material variation, defining modifications to specific asset types
message MaterialVariation {
  int32 cell_type;                                  // Which asset type to modify
  MaterialBaseType material_base_type;              // Optional change of asset material
  float hue;                                        // Color hue in HSV
  float saturation;                                 // Color saturation in HSV
  float brightness;                                 // Color brightness in HSV
}
```

Figure 6. Protocol Buffer definition of base objects referred to in Fig. 5.