Project Summary: Using A Mesh Renderer for Architectural Sketch-based Modeling

Nathan Shapiro
University of Michigan
nashapir@umich.edu

Abstract

Advances in the photorealistic quality and geometric detail of 3D renderings have made CAD software an indispensable tool for many practicing architects. However, sketching with pen and paper also remains a critical part of their design process. Architects are often left with the arduous task of "converting" an analog illustration into a digital model, manually creating the model from scratch and taking educated guesses at details like proportion, size, color, etc. Recent progress in and around deep learning has sparked new ideas for using computers to automate this conversion, but many of these attempts have trouble reconciling the imprecision of a human-drawn sketch with the hyper-precision of a computer model. This work proposes a reinforcement learning system which uses differentiable rendering to iteratively deform and refine an existing rectilinear mesh until its silhouette resembles that of an input sketch, thus creating a 3D "block version." We use an entirely unsupervised pipeline which, unlike other recent works, does not rely on 3D priors or a normal image GAN to compensate for ambiguities in the input sketch. By avoiding supervised methods of form generation, this work represents a small step in ignoring visual information that isn't on the page to better understand what is.

1. Introduction

Computer aided design (CAD) employs tactics such as synthetic lighting, shading and perspective to flatten a 3D model onto a 2D screen through a process called *rendering*. Architecture, in particular, utilizes CAD renderings to generate photorealistic

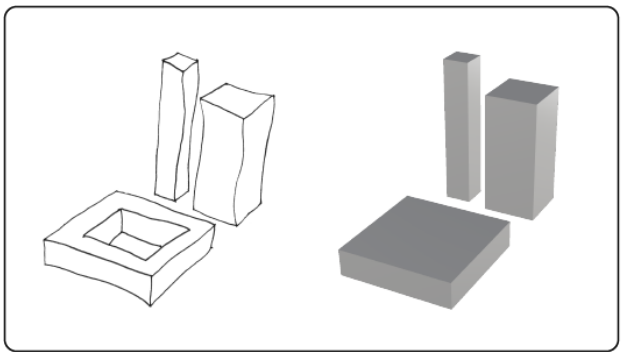

Figure 1: Example of multi-object conversion from architectural sketch to block CAD (input left, output right).

representations of designed space. Indeed, rendering has become so essential to the architect's process, and benefited from so much technical research, that the results are sometimes indiscernible from actual photographs. With remarkable fidelity, the architect can refine details and present informative depictions across a range of scales, from the smallest doorknob to the largest facade. However, despite increasing reliance on these technologies, critical steps in the design process remain manual: particularly during early exploration and refinement, hand-drawn sketches prove invaluable. They offer an economy of time and material expense, and frequently express an idea's essential character. Architects rely on illustrations to quickly and effectively convey the built (or soon-to-be-built) environment. Yet nothing gets built without *digital* design documents, and historically the act of reconciling the analog with the digital has proven challenging, requiring a laborious

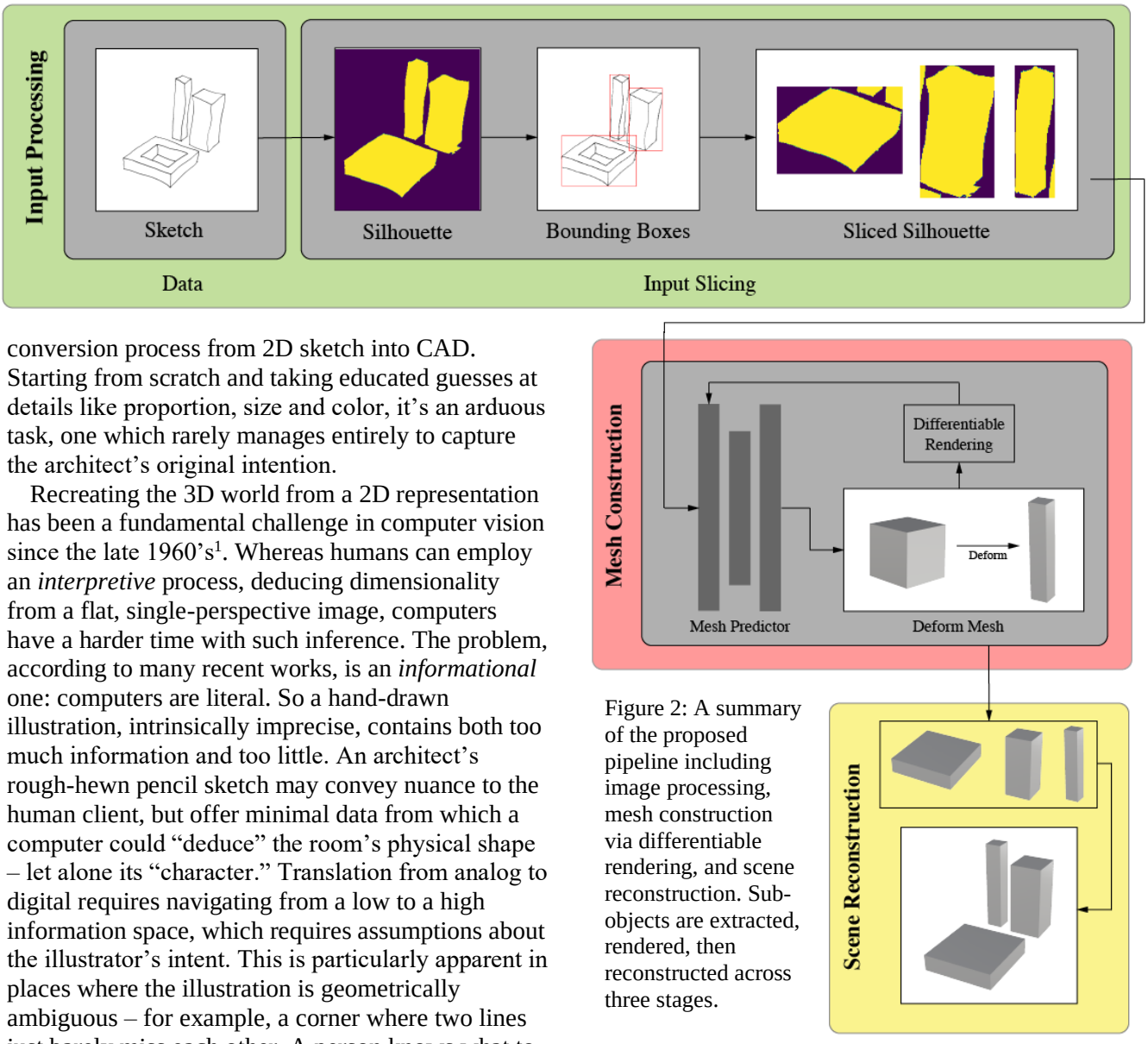

Figure 2: A summary of the proposed pipeline including image processing, mesh construction via differentiable rendering, and scene reconstruction. Sub-objects are extracted, rendered, then reconstructed across three stages.

conversion process from 2D sketch into CAD. Starting from scratch and taking educated guesses at details like proportion, size and color, it's an arduous task, one which rarely manages entirely to capture the architect's original intention.

Recreating the 3D world from a 2D representation has been a fundamental challenge in computer vision since the late 1960's[1]. Whereas humans can employ an *interpretive* process, deducing dimensionality from a flat, single-perspective image, computers have a harder time with such inference. The problem, according to many recent works, is an *informational* one: computers are literal. So a hand-drawn illustration, intrinsically imprecise, contains both too much information and too little. An architect's rough-hewn pencil sketch may convey nuance to the human client, but offer minimal data from which a computer could "deduce" the room's physical shape – let alone its "character." Translation from analog to digital requires navigating from a low to a high information space, which requires assumptions about the illustrator's intent. This is particularly apparent in places where the illustration is geometrically ambiguous – for example, a corner where two lines just barely miss each other. A person knows what to make of incongruities, recognizing in this case that the two lines belong to the same shape, even if technically they do not. For a computer, shapes which appear discrete *are* discrete.

We summarize our contributions as follows:

- We present an end-to-end learning framework that uses the silhouette of a simple, architectural sketch to deform an existing rectilinear mesh until its dimensions resemble those of the sketch, creating a simple 3D "block version" of the sketch.
- We take a small step towards evaluating the effectiveness of neural rendering in a completely unsupervised pipeline, so as to put the translational impetus on existing features in the sketch and more purposefully employ all of the visual information available.

2. Related Work

Previous work on reconciling the differences in precision (between illustration and rendering) has employed a range of strategies. Parametric Solutions use a set of rules and procedures to extract additional

information from the input[2]. These methods tend to generalize poorly, performing with limited accuracy on unlisted shapes and sketches with a high number of irregular lines. Other works stitch together multiple viewpoints[3], merging information across sketches to expose hidden or under-described elements. This strategy, while at times more visually convincing, has limited application within architecture where a single, quick sketch is often all that's available.

Recent progress in and around deep learning has sparked new ideas for addressing the issues of shape reconstruction. Deloney et al.[3] rely on a set of 3D priors for training a CNN to predict a voxel output. This category of approach is computationally expensive, even while producing low-fidelity (often grainy) results.

Research beginning as late as 2018 looks to a process called differentiable rendering to incorporate rendering into the neural pipeline – allowing 2D gradients to flow back across a 3D input. Rasterization, one of the sub-processes within rendering (wherein a flattened 2D shape is imposed onto a grid of pixels), determines which pixels receive color and what color they receive. The step is discrete, making it impossible to calculate true gradients on its output. Rather, the core discovery of differentiable rendering involves computing a set of artificial gradients and employing them during forward and back propagation, so that losses calculated on the 2D output can flow back and provide iterative refinements to the 3D input.

OpenDR[4], which is credited as one of the first differentiable rendering-based models, uses a first-order Taylor expansion to estimate the gradients. Kato et al.[5] coined the term "Neural Mesh Renderer" in implementing a custom linear estimation scheme. They put the model to use on single-image reconstruction and a variety of other common vision tasks that use 2D losses (like style transfer and DeepDream). Xiang et al.[6] employed differentiable rendering to single sketch-based retrieval, utilizing a normal image GAN with a trained discriminator to create a standardized representation of the input sketch.

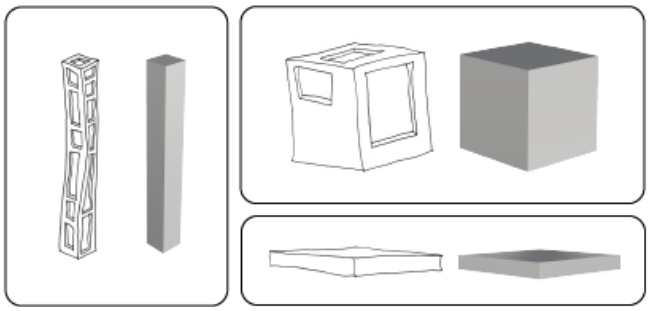

Figure 3: Three additional examples demonstrate the pipeline's ability to address illustrations of varying dimensions.

3. Methods

All of these approaches rely on a pipeline that involves some degree of shape supervision to address ambiguities in the sketch. Conversely, this work proposes an unsupervised reinforcement learning system, using differentiable rendering to repeatedly deform an existing rectilinear mesh until its silhouette closely resembles that of an input sketch, thereby providing a 3D "block version" of the original 2D illustration. Figure 2 offers a graphical overview of the end-to-end pipeline.

We begin by creating a binary mask to capture the silhouette of each sub-object, wherein all "outside" pixels are 0 and all "inside" pixels remain 1. Next, we use a breadth-first search to create bounding boxes for each of the previously mapped silhouettes. The search traverses the inside of each sub-object, roaming the inside of the silhouette until it has exhausted all pixels in the shape. We keep track of the highest, lowest, left-most, and right-most pixel of the sub-object, using the four coordinates to estimate the dimensions of a bounding box. Finally, we crop the silhouettes out of the sketch and perform some minor scaling until each silhouette is approximately equisized.

Next, we construct a rectilinear mesh for each of the cropped silhouettes. This process begins with a unit

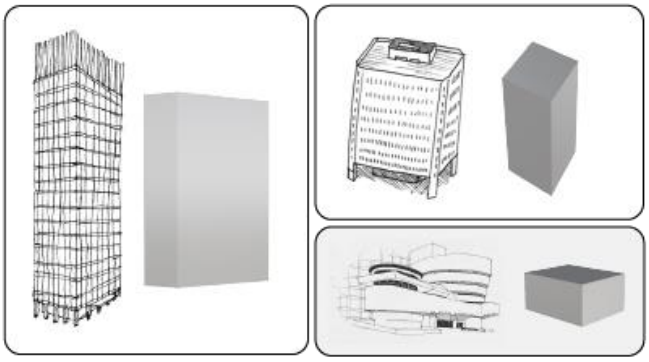

Figure 4: Examples of three sketches where the complexity and disorientation of the sketch leads to an inaccurate (or unhelpful) result.

mesh cube. The cube's height, width, and depth are stored in a live vector and updated as the cube changes shape. During each pass, these dimensions are used to transform the base mesh into a new, updated mesh. A silhouette loss is calculated between the rendering of the resulting mesh and the cropped input silhouette. The gradients on this loss are then passed back through the renderer and used to update the dimension vector. With each pass, the dimensions of the constructed mesh more closely approximate those of the illustrated shape. After the calculated loss falls below a set level (or the process times out), the model exits the loop and saves the resulting dimensions and mesh. This step is repeated for each sub-object in the input sketch.

As a final, optional step in the pipeline, we reassemble the original sketch using renderings of the result meshes. Using the bounding boxes calculated during silhouette extraction as rough placement coordinates, we resize and position the cropped renderings onto a blank canvas.

4. Experiments

While Figure 3 shows that we were able to create convincing results under fragile conditions, Figure 4 exposes places where the pipeline lacks generalization.

The viewpoint of in the illustration (which has to match the renderer's) offers the largest bottleneck for any practical application of the pipeline. Our most successful reconstructions used a viewpoint with a 45-degree azimuth rotation and 20 degrees of elevation. This maintains a complete horizontal symmetry, which helps minimize the distortion our renderer applies to simulate vanishing angles on the object. Other (non-symmetrical) viewpoints don't offer enough information about one or more of the three visible sides to create a convincing reconstruction.

5. Conclusion

Addressing the viewpoint bottleneck would make for a strong future contribution to the pipeline (and a potentially simple one, as Pytorch3D already has viewpoint calibration as a documented use case).

Future iterations of the pipeline might also replace the manual silhouette cropping process with a simple, single-pass object recognition model like YOLO[7].

Finally, we would like to see a version of the pipeline that preserves features of the original illustration beyond its general dimensions. For example, it might be useful to contort the outline of the mesh's edges to match those in the illustration.

This work uses a pipeline which does *not* rely on 3D priors or a normal image GAN to compensate for ambiguities in the input sketch. Almost no existing research has used differentiable rendering by itself, absent any auxiliary supervision. (We acknowledge that our approach relies on a fixed-form starter mesh, which constitutes a form of prior.) While it is true that hand-drawn illustrations are often geometrically imprecise, that imprecision can carry critical information worth exploring and capturing. There is a reason hand drawings are understood to have more "character" than their digital counterparts. In avoiding supervised methods of form generation, our technique offers an initial strategy to make fuller use of pictorial nuance and detail available without annotated data.

---

[1] Sutherland I. Sketchpad: a man-machine graphical communication system. In: AFIPS conference proceedings, vol. 23, 1963.

[2] Malik J. Interpreting line drawings of curved objects. Int J Comput Vis. 1987;1(1):73–103.

[3] Delanoy J, Aubry M, Isola P, Efros AA, Bousseau A. 3D sketching using multi-view deep volumetric prediction. Proc ACM Comput Graph Interact Techn. 2018.

[4] Loper MM, Black MJ. OpenDR: An approximate differentiable renderer. Proceedings of the European Conference on Computer Vision. New York, NY: Springer; 2014. p. 154–169.

[5] Kato H, Ushiku Y, Harada T. Neural 3d mesh renderer. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, Utah, USA; 2018. p. 3907–3916.

[6] Xiang N, Wang R, Jiang T, et al. Sketch-based modeling with a differentiable renderer. *Comput Anim Virtual Worlds.* 2020;31:e1939.https://doi.org/10.1002/cav.1939

[7] Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR. (2016)