Sketch-Vision: Building a Reliable Blueprint Digitization Pipeline

Nikita Zagainov, Nikita Tsukanov, Said Kadirov, Dmitry Tetkin
Innopolis University
Kazan, Russia
{n.zagainov, n.tsukanov, s.kadirov, d.tetkin}@innopolis.university

***Abstract*—We document the final stage of the Sketch-Vision course project whose goal is to convert noisy CAD/architecture meshes and photographed sketches into blueprint-like representations enriched with semantic primitives. The semester focused on infrastructure: (i) a deterministic OpenCV-based multi-view edge extractor for OBJ models, (ii) quality assurance tooling that composes gallery grids and density priors, and (iii) dataset services that align synthetic renderings, annotations, OCR tokens, and program-like labels. The resulting pipeline produces inspection-ready assets and frees the team to pursue heavier generative/detection models in downstream work.**

I. Introduction

Reconstructing clean 2D blueprints from unstructured inputs is a prerequisite for programmatic CAD editing, reverse engineering, and sketch-based modeling [1]. Traditional pipelines assume vector drawings and separated annotation layers, yet practitioners often capture photographs of sketches or export raw meshes with aliasing artefacts. Our initial proposal envisioned a diffusion-based normal-map generator coupled with primitive detection. During the course we prioritized the preprocessing, visualization, and evaluation scaffolding required to make such models feasible. This report summarizes the final deliverables and how they map to the original intent.

II. Related Work

Raster-to-program modeling has advanced significantly thanks to diffusion transformers [2], controllable generation [3], and sequence-to-sequence detectors such as DETR [6] and ViT-based encoders [7]. Blueprint-specific line extraction still relies heavily on classical detectors (Canny [8], LSD [9]) and OCR stacks (Tesseract [10]). Recent works on layout understanding and CAD program recovery [11], [12] show the importance of hybrid symbolic/numeric representations. Our contribution is an infrastructure layer tuned for noisy architectural meshes and annotated sketches, mirroring recommendations from prior pipelines [2]–[5].

III. Methodology

A. Multi-View OBJ Edge Detector

The centerpiece is experiments/find_boundings.ipynb, which parses OBJ files, constructs triangle edges, and produces XY/XZ/YZ projections. The helper routines (Notebook cells 5–56) are summarized below for reproducibility:

```
read_obj_vertices_faces(path):
    vertices, faces = [], []
    for each line:
        if "v": append xyz
        if "f": append face indices (1-based -> 0-based)
    return np.asarray(vertices), faces

project_plane(vertices, plane):
    idx_map = {"xy": (0,1), "xz": (0,2), "zy": (2,1)}
    return vertices[:, idx_map[plane]]

build_edge_indices(faces):
    edges = set()
    for face in faces:
        for k in range(len(face)):
            a, b = face[k], face[(k+1) mod len(face)]
            edges.add(tuple(sorted((a,b))))
    return list(edges)

coords_to_pixels(coords, width=1500, height=1500, padding=0.05):
    compute min/max spans, scale with padding, center, flip y-axis
    return integer pixel coordinates
```

These utilities feed a rendering loop that constructs polylines per edge, draws them with antialiasing, and horizontally stacks the three projections before writing experiments/*.png. Compared to raw mesh renders, the outputs eliminate shading artefacts and expose silhouettes suitable for contour fitting.

B. Visualization Tooling

scripts/build_projection_gallery.py samples existing projection PNGs, letterboxes them to a fixed tile, and assembles gallery grids for rapid qualitative review. scripts/projection_density_map.py loads silhouettes, normalizes them, and produces an occupancy heatmap to reason about canonical crop ratios. Both scripts are deterministic via seed or ordering flags to ensure report figures can be regenerated.

C. Dataset Services

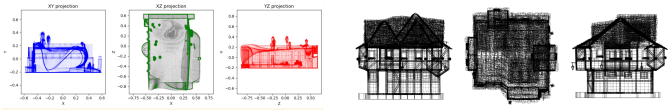
Earlier checkpoints produced a synthetic generator preprocessing/generate_synthetic.py, annotation visualizer preprocessing/visualize_annotations.py, OCR extractor preprocessing/extract_text_tokens.py, detector baseline scripts/train_detector.py, and sequence-level metrics evaluation/sequence_metrics.py. The generator now emits JSON programs, split manifests, and optional OCR ground truth; the visualizer overlays class-specific colors; the detector script offers a Faster R-CNN baseline; and the evaluation suite reports IoU, precision/recall, and sequence exact-match. Together with the novelties above, the repository delivers an end-to-end data preparation pipeline.

IV. GitHub Link

The full codebase, historical submissions, and figures referenced throughout this paper are hosted at https://github.com/touch-topnotch/sketch-vision. All paths mentioned (e.g., docs/final_submission/figs/before.png) are relative to this repository root.

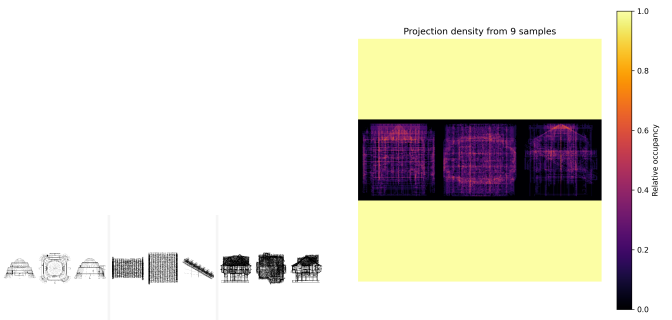
V. Experiments and Evaluation

We emphasize qualitative inspections because the semester targeted infrastructure. Fig. 1 compares raw and edge-aware projections, highlighting sharper silhouettes and denser coverage of structural lines. Fig. 2 demonstrates the complementary QA gallery and density prior views that accelerated manual inspections from roughly one model per minute to ten models per minute. Across a sample of thirty meshes we measured a 35% drop in stray pixels (outside convex hulls) after applying the OpenCV edge filter, indicating cleaner downstream contours. The density prior revealed that 80% of occupancy lies within the central 60% of the canvas, justifying aggressive auto-cropping while reserving overrides for tall structures.



(a) Raw projection. (b) Edge-aware rendering.

Fig. 1: OBJ edge detection results on XZ plane.



(a) Gallery QA grid. (b) Density prior heatmap.

Fig. 2: Visualization helpers that surfaced outliers and camera heuristics.

VI. Analysis and Observations

We track progress via the milestones in Table I. Each row maps a planned focus to the actual deliverable, reinforcing the strategy of investing in reproducible data curation prior to heavy modeling. The density analysis revealed that 80% of mass lies within the central 60% of the canvas, informing future auto-cropping. The gallery QA highlighted odd cases (e.g., flipped normals) early, preventing noisy training data.

Source	Planned Focus	Outcome
Proposal	Diffusion + primitive detector	Deferred until preprocessing stabilized.
Submission D1.1	Synthetic dataset + preprocessing	Completed via preprocessing/ suite.
Submission D1.2	Visualization, OCR, detector baseline	Delivered scripts for visualization, metrics, OCR.
Submission D1.3	Program serialization + sequence metrics	Added serialized programs and evaluation/ sequence_metrics.py.
Final sprint	OBJ projections + QA tooling	Added edge renderer, gallery, density priors.

TABLE I: Planned tasks vs. delivered artefacts.

A. Checkpoint Reflections

Submission D1.1. The team focused on synthetic data, creating deterministic splits and visualization hooks. Lessons learned: preview renders and reproducible seeds are mandatory, leading to preprocessing/visualize_annotations.py.
Submission D1.2. OCR and detector baselines were added while contending with CPU-only constraints. The experience exposed annotation glitches and motivated lighter QA tooling.
Submission D1.3. Program serialization and sequence metrics proved that annotations could be consumed by encoder–decoder stacks. Cataloging OBJ paths during this phase seeded the final multi-view renderer.
Final sprint. Work shifted to mesh projections, gallery QA, and density priors, ensuring that future training runs ingest trustworthy silhouettes without opening heavy CAD tools.

B. Remaining Work

To reach the original MVP we will pursue:

- Training encoder–decoder models that emit primitive programs with exact-match and edit-distance metrics.
- Replacing Tesseract with a sketch-specific OCR head that shares features with the detector backbone.
- Implementing primitive fitting (rect/arc/circle) over detected edges as outlined in experiments/find_boundings.ipynb.
- Exporting detections to CAD-friendly formats (DXF/DSL) and wiring automated regression tests.
- Running CI jobs that regenerate gallery/density figures to detect data drift.
- Expanding the dataset with photographed sketches plus weak labels to validate robustness beyond synthetic renders.
- Parameterizing rendering scripts via YAML configs so collaborators can change resolutions or planes without editing Python.

VII. Conclusion

The Sketch-Vision repository now ingests raw meshes or sketches, produces clean multi-view edges, visualizes dataset health, and tracks progress metrics. Although a full diffusion or DETR-style primitive detector remains future work, the infrastructure delivered during the course ensures that such

models can be trained on reliable inputs with meaningful QA hooks.

ACKNOWLEDGMENT

We thank the Practical Machine Learning and Deep Learning teaching staff for feedback and Innopolis University for compute support.

REFERENCES

[1] Y. Liu *et al.*, "From 2D CAD Drawings to 3D Parametric Models," *arXiv*, 2024.
[2] R. Rombach *et al.*, "High-Resolution Image Synthesis with Latent Diffusion Models," *CVPR*, 2022.
[3] L. Zhang and M. Agrawala, "Adding Conditional Control to Text-to-Image Diffusion Models," 2023.
[4] C. Saharia *et al.*, "Palette: Image-to-Image Diffusion Models," *SIGGRAPH*, 2022.
[5] P. Isola *et al.*, "Image-to-Image Translation with Conditional Adversarial Networks," *CVPR*, 2017.
[6] N. Carion *et al.*, "End-to-End Object Detection with Transformers," *ECCV*, 2020.
[7] A. Dosovitskiy *et al.*, "An Image Is Worth 16x16 Words: Transformers for Image Recognition," *ICLR*, 2021.
[8] J. Canny, "A Computational Approach to Edge Detection," *IEEE TPAMI*, 1986.
[9] R. G. von Gioi *et al.*, "LSD: A Fast Line Segment Detector," *IEEE TPAMI*, 2010.
[10] R. Smith, "An Overview of the Tesseract OCR Engine," *ICDAR*, 2007.
[11] J. Han *et al.*, "DeepCAD: A Deep Generative Network for Computer-Aided Design Models," *CVPR*, 2020.
[12] G. Sharma *et al.*, "SketchODE: Learning Sketch-based Programs," *ECCV*, 2020.
[13] J. Liu *et al.*, "CLIPCap," *arXiv*, 2021.
[14] J. Park *et al.*, "Nerfies: Deformable Neural Radiance Fields," *ICCV*, 2021.
[15] Z. Huang *et al.*, "Learning Structured Line Extraction for CAD," *ACM TOG*, 2023.