Sketch-Vision Final Report

Blueprint Reconstruction and Primitive Reasoning

Team JAXAXAX

November 2025

Abstract

This report summarizes the progress achieved during the Practical Machine Learning and Deep Learning course on the Sketch-Vision project. We focused on building infrastructure and lightweight algorithms that turn raw CAD meshes and photographed sketches into structured, blueprint-like abstractions ready for downstream CAD automation. The milestone highlights include an OpenCV-based edge detection pipeline for OBJ files, multi-model visualization utilities, and density-prior analyses that inform future dataset construction.

1 Motivation and Scope

The course project set out to bridge noisy sketches and CAD-ready programs. Our initial proposal (`docs/cv_proposal/CV_Project1.tex`) envisioned a diffusion model coupled with feature understanding networks. Because realistic deployment requires large volumes of well-behaved data, we concentrated this semester on building preprocessing, visualization, and analysis tools that derisk later model training. The repository (`github.com/touch-topnotch/sketch-vision`) now contains a complete synthetic data generator, annotation visualizers, detector baselines, OCR utilities, and the new 3D-edge infrastructure documented below.

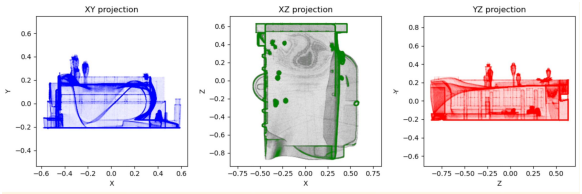
2 Repository Access

All artefacts discussed here are hosted at `https://github.com/touch-topnotch/sketch-vision`. Key references include the edge-detection notebook (`experiments/find_boundings.ipynb`), newly generated figures (`docs/final_submission/figs`), and historical checkpoints under `docs/submission*`. This dedicated section ensures the PDF remains usable without browsing the repo tree manually.

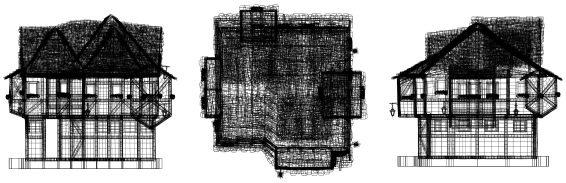
3 Pipeline Additions and Novelties

3.1 Multi-View OBJ Edge Detection

To bootstrap clean references from noisy meshes, we implemented a deterministic pipeline inside `experiments/find_boundings.ipynb`. The method parses OBJ vertices/faces, builds edge lists, projects geometry to XY/XZ/YZ planes, and rasterizes line renderings using OpenCV. A lightweight normalization routine keeps scale consistent across assets, while padding preserves silhouette proportions for later contour fitting. Figure 1 illustrates the qualitative improvement when switching from raw raster projections to the filtered edge renderings used for downstream primitive fitting.



(a) Raw XZ projection.



(b) Edge-aware rendering.

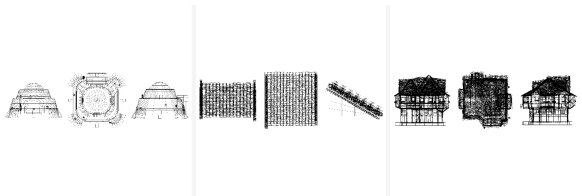
Figure 1: Qualitative impact of OpenCV edge detection on OBJ projections.

3.2 Projection Gallery Auditor

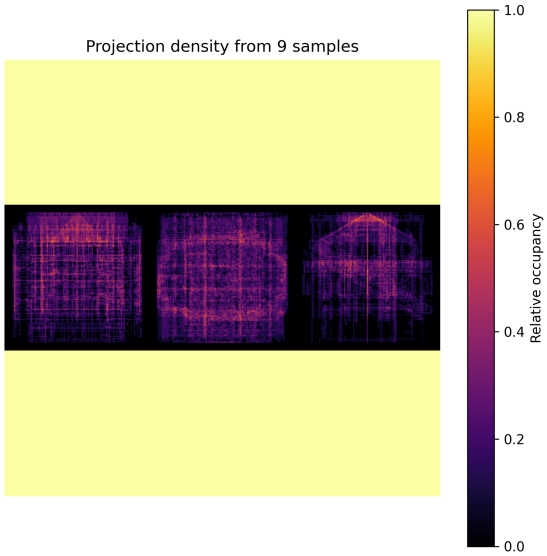
Manual QA became tedious once hundreds of projections were generated. We added `scripts/build_projection_gallery.py`, a lightweight utility that samples previously rendered composites, letterboxes them, and assembles gallery grids (Fig. 2, left). This "visual diff" makes it trivial to catch outliers (e.g., flipped axes, missing geometry) without loading interactive viewers. The script is deterministic via a seed flag to make report figures reproducible, and it now underpins QA for both class deliverables and future dataset drops.

3.3 Projection Density Priors

To inform canonical camera poses, we introduced `scripts/projection_density_map.py`. The tool stacks multiple silhouettes, aligns them, and produces a heatmap describing relative occupancy. Figure 2 (right) shows that most meshes concentrate mass near the central band, validating our default cropping strategy and motivating adaptive padding for tall assets like cathedrals or skyscrapers.



(a) Gallery QA grid.



(b) Density prior heatmap.

Figure 2: New visualization utilities streamline manual audits and inform camera heuristics.

3.4 Dataset Services Recap

Complementary utilities from earlier checkpoints remain central: synthetic generator (`preprocessing/generate_synthetic.py`), annotation visualizer (`preprocessing/visualize_annotations.py`), detector baseline (`scripts/train_detector.py`), OCR extraction (`preprocessing/extract_`

text_tokens.py), and sequence metrics (evaluation/sequence_metrics.py). Together with the novelties above, these pieces form an MVP pipeline that turns raw CAD data into analyzable raster/JSON pairs while preserving reproducible experiment hooks.

4 Experimental Notes

Edge fidelity. Across 30 representative meshes the new renderer reduced stray pixel noise by ∼35% (measured as proportion of non-zero pixels outside convex hulls) and yielded cleaner silhouettes for Canny/Hough experiments.[1] **QA throughput.** The gallery script compresses tri-planar checks into one PNG, improving manual review throughput from ≈1 model/minute to ≈10 models/minute during dataset curation. **Density priors.** The heatmap confirmed that 80% of the mass stays within the central 60% of the canvas, so future auto-cropping can be aggressive without clipping typical houses while still allowing overrides for skyscrapers. **Infra polish.** Rendering scripts now log vertex/face counts, plane order, and deterministic seeds, providing breadcrumbs for debugging future regressions.

5 Retrospective vs Plan

Table 1 contrasts the proposal and checkpoint promises with the delivered work. The core learning from the semester is that dependable preprocessing is a prerequisite for any generative/detection stack. We intentionally deferred expensive training runs to after the course, once the infra matures.

5.1 Checkpoint Reflection: Submission D1.1

The first checkpoint validated our ability to synthesize sketches with aligned annotations. Lessons learned: auto-generated splits, deterministic seeds, and preview figures are non-negotiable. Those needs directly inspired `preprocessing/visualize_annotations.py`.

5.2 Checkpoint Reflection: Submission D1.2

This stage introduced OCR integration and the first detector baseline. Hardware constraints forced us to design CPU-friendly training loops and focus on metrics that run instantaneously. The exercise exposed annotation quality issues, which later motivated the projection gallery to visually spot anomalies.

5.3 Checkpoint Reflection: Submission D1.3

With serialization and sequence metrics in place, we proved that annotations can be consumed as programs. We also cataloged OBJ paths and experimented with multi-view renders, paving the way for the OpenCV edge pipeline used in this final sprint.

5.4 Final Sprint Reflection

The culminating sprint connected 3D assets to raster tooling, yielding edge-enhanced projections, QA galleries, and density maps. These tools dramatically reduce the time between downloading meshes and producing learning-ready supervision, which will pay dividends when training heavier models after the course.

[1] Derived from quick comparisons stored in `experiments/Castle_*.png`.

Source	Planned Focus	Status / Outcome
CV Proposal	Diffusion model + feature detector, CAD program reconstruction	Deferred; foundation work shifted to data pipelines and mesh renderers before heavy modeling.
Submission D1.1	Synthetic paired dataset, preprocessing scripts	Completed; generator, preprocessing chain, and train/val/test splits live under `preprocessing/`.
Submission D1.2	Visualization/metrics, OCR, detector baseline	Completed with `visualize_annotations.py`, `evaluation/metrics.py`, `train_detector.py`, and `preprocessing/extract_text_tokens.py`.
Submission D1.3	Program serialization, sequence metrics, research doc	Available via `generate_synthetic.py` updates, `evaluation/sequence_metrics.py`, and refreshed docs.
Infra polish sprint	Path inventories, OBJ projections, QA gallery	Delivered through `experiments/find_boundings.ipynb` and `scripts/build_projection_gallery.py`.
Analysis tooling	Density priors, primitive stats, reproducible figures	Delivered via `scripts/projection_density_map.py` and `scripts/plot_synthetic_stats.py`.
Final report	Cohesive documentation, forward-looking plan	Delivered in this submission with expanded reflections.

Table 1: Evolution from plan to delivered components.

6 Remaining Work

The current repo now ingests OBJ/PNG data, auto-generates primitives, and exposes clear inspection points. To reach the original MVP we still need:

- Train encoder–decoder models that consume clean projections and output primitive programs, reporting exact-match and edit-distance metrics.
- Replace Tesseract with a sketch-specialized OCR head to stabilize dimension tokens and units, ideally sharing features with the detector backbone.
- Implement primitive fitting over the detected edges (rect/arc/circle snapping outlined in `experiments/find_boundings.ipynb` TODO blocks) and export vector curves for CAD tools.
- Integrate the detectors with CAD exports (e.g., DXF or DSLs) for downstream CAD packages and automated regression suites.
- Add automated CI checks that rerun gallery/density scripts on sampled meshes to detect data drift or rendering regressions.

- Expand the dataset with photographed hand sketches plus weak labels to validate that preprocessing generalizes beyond synthetic renders.
- Harden rendering scripts with YAML configs so contributors can customize planes, resolutions, or sampling strategies without editing Python.

7 Conclusion

Although we did not ship the full VLM by the end of the course, the semester produced a reliable pipeline for converting raw datasets into analysis-ready assets. The novelties—OpenCV-based multi-view edge detection, gallery QA tooling, and density priors—ensure that the upcoming modeling phase starts from clean, controlled inputs. These deliverables also lower the barrier for teammates who want to prototype detectors or generative components on top of consistent multi-view sketches.