Ryu-Scape (龍景): A Deep Learning Pipeline for 2D-to-3D Satellite Image Reconstruction

Author: Silragon ryu

Date: 06/10/2025

Contents

1. Introduction

1.1. Motivation and Problem Statement

Satellite imagery offers a vast and accessible source of global data, forming the backbone of modern geospatial analysis. However, its inherent 2D nature presents a significant limitation for applications requiring topographical and structural understanding. Fields such as urban planning, telecommunications (signal propagation), disaster simulation (e.g., flood risk analysis), and environmental monitoring demand accurate 3D data. Traditional methods for acquiring this data, primarily LIDAR and aerial photogrammetry, are highly accurate but suffer from prohibitive costs, slow acquisition times, and incomplete global coverage.

This project addresses this gap by exploring a modern, data-driven alternative. We frame the task of 3D reconstruction as an **image-to-image translation problem**, where the goal is to infer 3D structure directly from 2D satellite data using deep learning. The primary objective is to automate the generation of urban landscapes, creating a fast, scalable, and cost-effective pipeline.

1.2. Project Objectives

The project was structured around three core objectives:

1. **Build an End-to-End Pipeline:** To design and implement a complete system that processes raw satellite data, utilizes a trained AI model for inference, and generates an interactive 3D visualization.
2. **Compare and Refine Model Approaches:** To iteratively experiment with different deep learning architectures and problem formulations to achieve the highest possible accuracy in building identification.
3. **Develop a User-Facing Application:** To encapsulate the entire pipeline within a polished, intuitive desktop application for seamless user interaction.

1.3. Contribution Summary

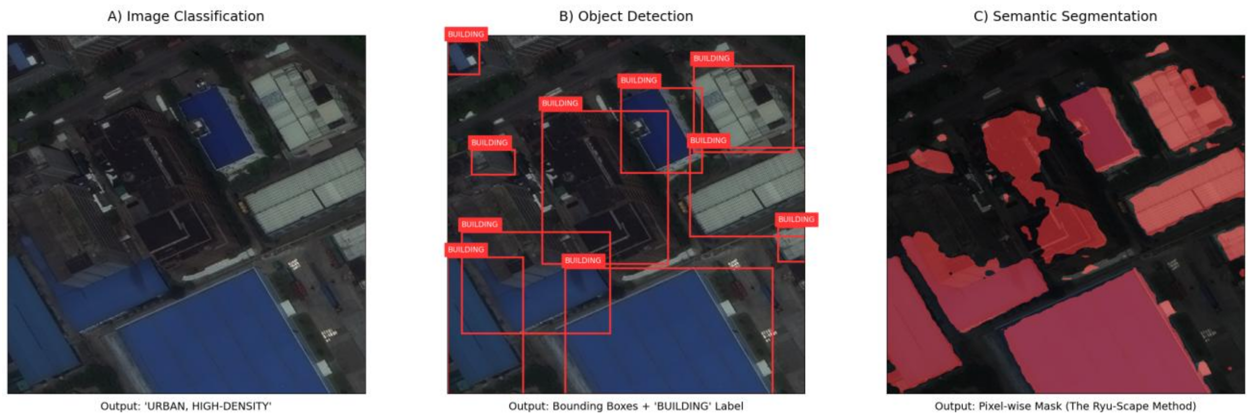
This report presents Ryu-Scape, a novel application that successfully integrates a custom-trained DeepLabV3 model with a procedural 3D generation engine. The system automates the conversion of 2D satellite images into detailed 3D scenes, complete with textured ground, extruded buildings, and architectural details. The final deliverable is a standalone Windows executable that showcases the fusion of deep learning, 3D graphics, and user interface design.

2. Background and Literature Review

2.1. Semantic Segmentation

The core AI task in this project is **semantic segmentation**. Unlike image classification (which assigns one label to an entire image) or object detection (which draws bounding boxes), semantic segmentation assigns a class label to *every single pixel* in an image. In our case, the classes are simple: "Building" or "Background."

A Comparison of Core Computer Vision Tasks on Satellite Imagery


A) Image Classification
B) Object Detection
C) Semantic Segmentation
BUILDING
Output: 'URBAN, HIGH-DENSITY'
Output: Bounding Boxes + 'BUILDING' Label
Output: Pixel-wise Mask (The Ryu-Scape Method)

2.2. Key Architectures in Image Segmentation

Our research and experimentation focused on architectures designed for dense, pixel-wise prediction.

- **U-Net:** A foundational architecture for biomedical image segmentation, characterized by its symmetrical encoder-decoder structure. The "skip connections" that bridge the contracting (encoder) and expanding (decoder) paths are its key innovation, allowing the network to combine deep, semantic features with shallow, high-resolution features to produce precise segmentation masks.
- **DeepLabV3:** A state-of-the-art architecture that excels at multi-scale feature extraction. Its signature component is the **Atrous Spatial Pyramid Pooling (ASPP)** module. ASPP uses atrous (or dilated) convolutions with different "dilation rates" to probe an incoming feature map at multiple scales without increasing the number of parameters. This allows the model to effectively recognize objects of various sizes, which is critical for identifying both small houses and large skyscrapers in satellite imagery.

3. Data Acquisition and Preprocessing

3.1. Data Sources and Characteristics

A multi-modal dataset from the Shanghai region was used for the final model.

- **RGB Satellite Imagery:** Sourced from the **SpaceNet-2 dataset**, providing high-resolution visual context.
- **Digital Surface Model (DSM):** Sourced from the **ALOS World 3D (AW3D30) dataset**. This data provides absolute elevation values.
- **GeoJSON Annotations:** Vector polygons from SpaceNet-2 provided the ground truth for building footprints.

3.2. Data Pipeline and Engineering

A robust preprocessing pipeline was critical for the project's success.

1. **Mask Generation:** The vector-based GeoJSON data was rasterized into binary (black and white) building masks, creating a pixel-aligned ground truth for our convolutional model.
2. **Data Alignment and Cleaning:** A script was developed to align the disparate data sources. This involved cropping and resampling the DSM tiles to match the exact dimensions and geospatial projection of their corresponding RGB images. A critical cleaning step identified and discarded image pairs containing invalid data (such as nodata values of -9999 in the DSM tiles), resulting in a clean final dataset of ~1,100 image pairs.
3. **Input Tensor Formulation:** The model's input was engineered into a single 4-channel tensor, stacking the DSM as a fourth channel alongside the RGB data. This allows the network to learn the direct correlation between visual patterns and elevation data.
4. **Normalization:** All pixel values across all four channels were normalized to a range of [0,1] using the min-max scaling equation to ensure stable training:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

4. Methodology and Model Development

The project followed an iterative development process, beginning with simpler hypotheses and progressively moving to more complex and successful approaches

4.1. Approach 1 (Initial Exploration): 3D Convolutional Networks

- **Hypothesis:** A 3D CNN could learn spatial and temporal features from a stack of co-registered satellite images.
- **Implementation:** An adapted 3D U-Net was implemented. The input was a volume of 3×1024×1024.
- **Result: Failure.** This approach, while theoretically sound, proved to be computationally infeasible. The memory footprint of the 3D convolutional layers, especially during backpropagation, consistently caused kernel crashes and memory exhaustion on the available hardware.

4.2. Approach 2 (Iterative Refinement): Direct DSM Regression

- **Hypothesis:** A standard 2D U-Net could be trained to directly regress the DSM height values from an RGB input.
- **Implementation:** A U-Net was trained using Mean Squared Error (MSE) loss.
- **Result: Failure.** The model struggled to converge. The validation loss was extremely high (on the order of 7×106), indicating that predicting precise, continuous elevation values is a significantly more difficult task than classification.

4.3. Approach 3 (Final Architecture): Segmentation with DeepLabV3

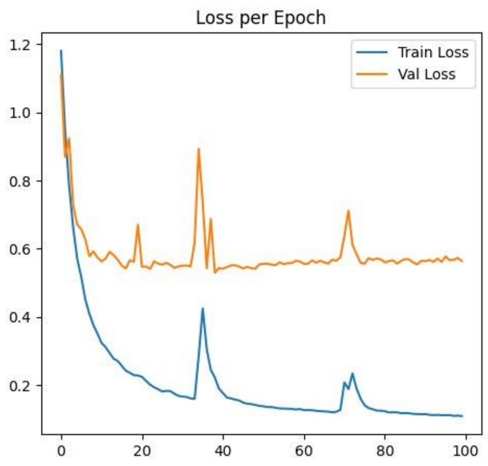
The key insight that led to the final, successful model was to **reframe the problem from regression to segmentation.** Instead of asking "How high is this pixel?", we asked "Is this pixel part of a building?".

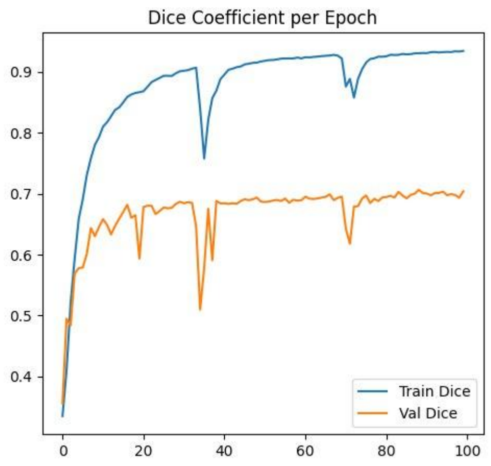
- **Architecture:** A **DeepLabV3** model with a **ResNet-50** backbone was chosen.
- **Modifications:**
    1. The first convolutional layer was replaced to accept our **4-channel (RGB+DSM) input**.
    2. The final classifier was modified to output a **single-channel probability mask**.
- **Training Protocol:**

- **Loss Function:** A composite Dice+BCE loss function was used.

$$L_{total} = -(y\log(p) + (1-y)\log(1-p)) + (1 - \frac{2|A \cap B|}{|A|+|B|})$$

- **Optimizer:** Adam with a learning rate of 1×10−4.
- **Performance:** The model was trained for 100 epochs, achieving a validation Dice score of **0.7066** at its peak (Epoch 89). The close alignment of training and validation loss curves demonstrated that the model generalized well without overfitting.





5. 3D Scene Reconstruction and Visualization Engine

5.1. The 3D Generation Pipeline

The 2D mask produced by the AI serves as a blueprint for the 3D scene.

1. **Mask Binarization:** The output probability mask is binarized using a confidence threshold of τ=0.5.
2. **Contour Detection:** cv2.findContours is applied to the binary mask to extract the outline of each building.
3. **Component Generation:** For each contour, the system generates distinct 3D objects: a ground plane, building roofs, and building walls.

5.2. Procedural Detailing Algorithms

To enhance realism, several procedural algorithms were implemented:

- **Textured Ground:** The ground is rendered as a point cloud, with each point colored according to the corresponding pixel in the original RGB image.
- **Extruded Walls with Paneling:** Walls are generated as meshes. To break up flat surfaces, they are procedurally divided into vertical panels with slightly alternating shades based on the average color of the building's roof.
- **Rooftop Parapets:** A small lip is procedurally added around the roofline of each building to provide a more realistic silhouette.
- **Window Generation:** Based on the length of a wall segment, a series of simple rectangular window meshes are algorithmically placed to add architectural detail.

5.3. Application Architecture and GUI

- **Framework:** The entire system is encapsulated in a desktop application built with **PyQt5**.
- **Asynchronous Processing:** To ensure a responsive UI, the entire AI inference and 3D generation pipeline is executed on a background QThread. The main UI thread remains free, displaying a "Processing..." message.
- **High-Performance Viewer:** The 3D viewer is a custom QGLWidget that leverages **Vertex Buffer Objects (VBOs)**. This modern OpenGL technique uploads all geometry data to the GPU's memory a single time, enabling smooth, lag-free rendering and interaction.

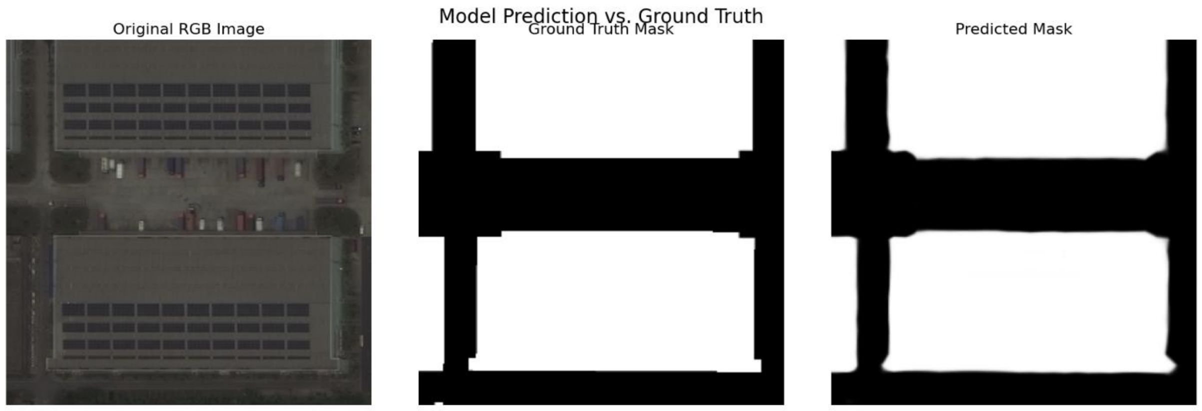
6. Results and Discussion

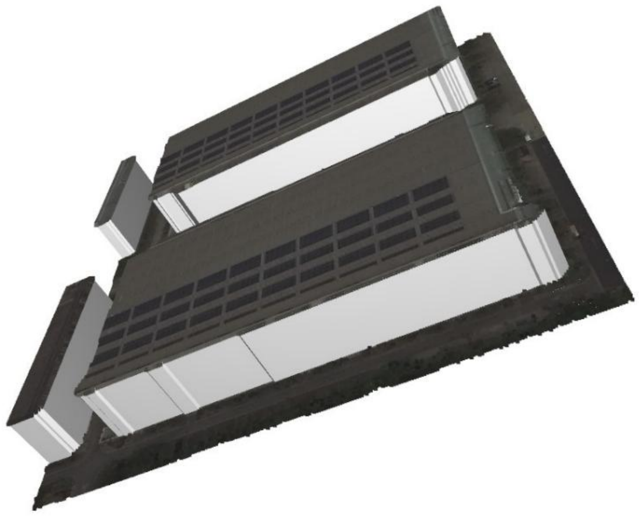
6.1. Quantitative Results

The final model achieved a peak validation **Dice Score of 0.7066** and a final validation **loss of ~0.5**. This indicates a high degree of overlap and accuracy between the predicted building footprints and the ground truth data.

6.2. Qualitative Results

A visual comparison of the model's output against the ground truth reveals a high level of accuracy. The model is capable of segmenting complex building shapes and distinguishing between closely packed structures.





6.3. Limitations

- **Data Dependency:** The model's accuracy is highly dependent on the availability of a corresponding DSM file. It has not been trained to infer height from monocular (RGB-only) cues.
- **Segmentation Artifacts:** In some cases, the model may produce minor artifacts or merge adjacent buildings. This can lead to inaccuracies in the final 3D model.
- **Simple Proceduralism:** The procedural detailing (windows, paneling) is based on simple heuristics and does not represent true architectural styles.

7. Conclusion and Future Work

7.1. Project Summary

The Ryu-Scape project successfully met all its objectives, delivering a complete end-to-end pipeline that transforms 2D satellite data into interactive 3D scenes. The iterative development process led to a robust solution centered on a modified DeepLabV3 architecture, and the final application demonstrates a successful fusion of deep learning, procedural generation, and high-performance computer graphics.

7.2. Future Work

- **RGB-Only Model:** The highest priority for future work would be to train an RGB-only model. This would involve removing the DSM channel and retraining the network to infer height from monocular cues like shadows and perspective, dramatically increasing the application's versatility.
- **Advanced Procedural Generation:** The current procedural engine could be expanded to generate more diverse and realistic architectural details.
- **Model Export:** A key feature would be to add the ability to export the generated 3D scene to standard file formats like .obj or .ply, allowing the models to be used in other 3D software.

8. Appendices

8.1. Environment Setup

A requirements.txt file is provided in the root of the repository. The environment can be recreated using:

```
pip install -r requirements.txt
```

8.2. Packaging Command

The application was packaged into a standalone executable using the following PyInstaller command:

```
pyinstaller --windowed --name Ryu-Scape --icon="logo.png" --add-data "Dataset;Dataset"
--add-data "logo.png;." ryu_scape_app.py
```