Computer Vision - 01 (Spring 2025)

Professor Hadi Akbarpour

Team 10

VGGSfM: Visual Geometry Grounded Deep Structure From Motion

Submitted by:

Hassan Sheraz,

Vishal Reddy Putta,

Ravali Maddela

Comprehensive Project Analysis & Implementation Report

1. Introduction & Background

VGGSfM represents a significant advancement in 3D reconstruction from 2D images by introducing a fully differentiable end-to-end deep learning approach to Structure-from-Motion (SfM). Developed by researchers at Facebook/Meta (Wang et al., 2023-2024), this system addresses a long-standing challenge in computer vision by integrating deep learning with traditional geometric algorithms.

Structure-from-Motion aims to simultaneously reconstruct both 3D scene structure and camera motion from unconstrained 2D images. While traditional methods like COLMAP use a non-differentiable pipeline with separate stages (keypoint detection, matching, registration, triangulation, and bundle adjustment), VGGSfM's innovation is making the entire process differentiable and trainable together.

VGGSfM has been publicly released (with v1.1 available in April 2024) along with pretrained models, and has demonstrated state-of-the-art performance on multiple benchmark datasets. The system won first place in the CVPR 2024 IMC Challenge for camera pose estimation, showing its superiority over traditional methods.

2. Core Architecture & Key Components

The VGGSfM architecture introduces several innovative components that create a fully differentiable SfM pipeline:

2.1 Deep 2D Point Tracking

- **CoTracker Integration**: Leverages CoTracker, a transformer-based model for tracking any pixel in a video with high accuracy
- **Coarse-to-Fine Tracking**: Implements a two-stage tracking mechanism that first establishes coarse correspondences, then refines them for pixel-accurate results
- **Eliminates Matching Chains**: Traditional SfM requires chaining pairwise matches across images, whereas VGGSfM directly tracks points across multiple frames
- **Feature Extraction**: Combines with LightGlue, a lightweight feature matcher that uses adaptive pruning techniques for efficient matching

2.2 Simultaneous Camera Recovery

- **Global Camera Registration**: Recovers all camera parameters simultaneously, rather than incrementally registering cameras as in traditional SfM

- **Image and Track Features**: Uses both image content and track information to estimate camera poses
- **Transformer-Based Approach**: Employs attention mechanisms to process relationships between images and points
- **Scaling Consistency**: Addresses the scale ambiguity inherent in monocular SfM for more consistent reconstructions

2.3 Differentiable Bundle Adjustment

- **End-to-End Optimization**: Implements a differentiable bundle adjustment layer that jointly optimizes camera parameters and 3D points
- **Trainable Refinement**: Unlike traditional bundle adjustment, this component can learn optimal refinement strategies through training
- **Geometric Constraints**: Maintains strict geometric constraints while allowing backpropagation

2.4 Technical Implementation Details

- **Sliding Window Processing**: For long video sequences, employs a sliding window approach for reconstructing frames
- **PyTorch Framework**: Built on PyTorch (2.1) with CUDA acceleration for efficient processing
- **Memory Optimization**: Implements adaptive point pruning and early stopping to reduce memory requirements
- **Flash Attention**: Leverages accelerated attention mechanisms for faster processing when available

2.5 Additional Capabilities

- **Dense Depth Maps**: Integration with Depth-Anything-V2 to create aligned dense depth maps from sparse reconstructions
- **Dynamic Object Handling**: Ability to filter out dynamic pixels using binary masks (1 for dynamic, 0 for static)
- **Visualization Tools**: Support for multiple visualization options including Visdom, Gradio, and COLMAP-compatible outputs
- **Video Processing**: Special VideoRunner class for processing video sequences in a sliding window manner

3. Implementation Workflow & Key Learnings

Based on your implementation experience, the workflow involves:

1. **Environment Setup**:
    a. Installing dependencies (PyTorch 2.1, CUDA 12.1, LightGlue, CoTracker, pycolmap, poselib, etc.)
    b. Managing CUDA compatibility issues (adaptation to CPU-only environments)
    c. Resolving memory limitations through parameter adjustments
    d. Setting up visualization tools (Visdom, Gradio optional)

2. **Data Processing**:
    a. Using a custom kitchen/statue scene as input
    b. Processing 25 images through the VGGSfM pipeline
    c. Parameter tuning (reducing max query points, adjusting precision settings)
    d. Optional mask creation for filtering dynamic objects (binary masks with matching filenames)

3. **Reconstruction Steps:**
    a. **2D Tracking**: Using CoTracker to establish reliable point correspondences across images
    b. **Camera Parameter Estimation**: Recovering extrinsic and intrinsic camera parameters
    c. **3D Triangulation**: Calculating 3D point positions from 2D tracks and camera parameters
    d. **Bundle Adjustment**: Refining the reconstruction through optimization

4. **Performance Optimization**:
    a. Reducing workload by lowering max_query_points (recommended for CPU-only environments)
    b. Setting mixed_precision=None for better CPU compatibility
    c. Memory management through batch processing for larger datasets

5. **Output Generation**:
    a. Creating COLMAP-compatible binary files (cameras.bin, images.bin, points3D.bin)
    b. Converting to PLY format for visualization
    c. Generating ~1800 3D points from the reconstruction process
    d. Optional dense depth map creation with Depth-Anything-V2 integration

6. **Visualization**:
    a. Using MeshLab for point cloud inspection
    b. Using COLMAP GUI for viewing cameras (red pyramids) and 3D points together
    c. Optional Visdom visualization for real-time monitoring

4. Evaluation Metrics & Performance Analysis

VGGSfM evaluates performance using several standard metrics:

4.1 Camera Pose Accuracy

- **AUC (Area Under Curve)**: Evaluates camera pose accuracy at different error thresholds
- **Angular Error Thresholds**: Typically measured at 5°, 10°, 15°, and 20° thresholds
- **Relative Translation Error**: Measures accuracy of camera translations

4.2 Reconstruction Quality

- **Point Cloud Density**: Number of successfully triangulated 3D points (~1800 in your implementation)
- **Reprojection Error**: Distance between original 2D points and reprojections of 3D reconstructions
- **Visual Consistency**: Qualitative assessment of the reconstruction completeness

4.3 Benchmark Datasets

VGGSfM demonstrates state-of-the-art performance on multiple standard datasets:

- **CO3D**: Common Objects in 3D dataset
- **IMC Phototourism**: Internet-based reconstruction challenge
- **ETH3D**: Multi-view stereo benchmark

4.4 Ablation Studies

The research includes analysis of component contributions:

- Removal of fine tracker reduces performance (AUC@10 drops from 73.92% to 62.30%)
- Camera initializer outperforms alternatives like PoseDiffusion
- Triangulation methods significantly impact final results

5. Implementation Challenges & Solutions

Your implementation encountered several challenges:

1. **CUDA Compatibility**:
    a. **Challenge**: CUDA checkpoint loading on CPU-only system
    b. **Solution**: Modified torch.load call to force CPU loading (slower but functional)
    c. **Code Adaptation**: Added `map_location='cpu'` parameter to torch.load calls

2. **System Resource Management**:
    a. **Challenge**: High CPU load and memory constraints
    b. **Solution**: Reduced max query points, adjusted precision settings
    c. **Parameter Tuning**: Found optimal settings with lower max_query_points (e.g., 1600 instead of default)

3. **Parameter Configuration**:
    a. **Challenge**: Mixed precision setting causing assertion errors
    b. **Solution**: Used mixed_precision=None instead of fp32
    c. **Error Message**: Assertion error when using fp32 since code only accepts None, bf16, or fp16

4. **Dependency Integration**:
    a. **Challenge**: Managing complex dependencies (pycolmap, LightGlue, etc.)
    b. **Solution**: Proper repository cloning and compatibility verification
    c. **Repository Structure**: Needed to clone LightGlue and pycolmap repositories separately

5. **Missing Attributes:**
    a. **Challenge**: Missing registered and pose_refinement attributes in pycolmap
    b. **Solution**: Implementation workarounds to bypass unavailable modules
    c. **Fallback Strategy**: Focusing on reconstruction even without full pose estimation

6. **Library Version Conflicts:**
    a. **Challenge**: Incompatibilities between PyTorch, torchvision, and CUDA versions
    b. **Solution**: Ensuring compatible combinations (PyTorch 2.1 with CUDA 12.1)
    c. **Dependency Resolution**: Careful management of NumPy version compatibility

7. **Memory Management**:
    a. **Challenge**: Memory errors during processing of large scenes
    b. **Solution**: Batch processing and parameter optimization
    c. **Performance Trade-off**: Balancing between reconstruction quality and memory usage

6. Results Interpretation & Visualization

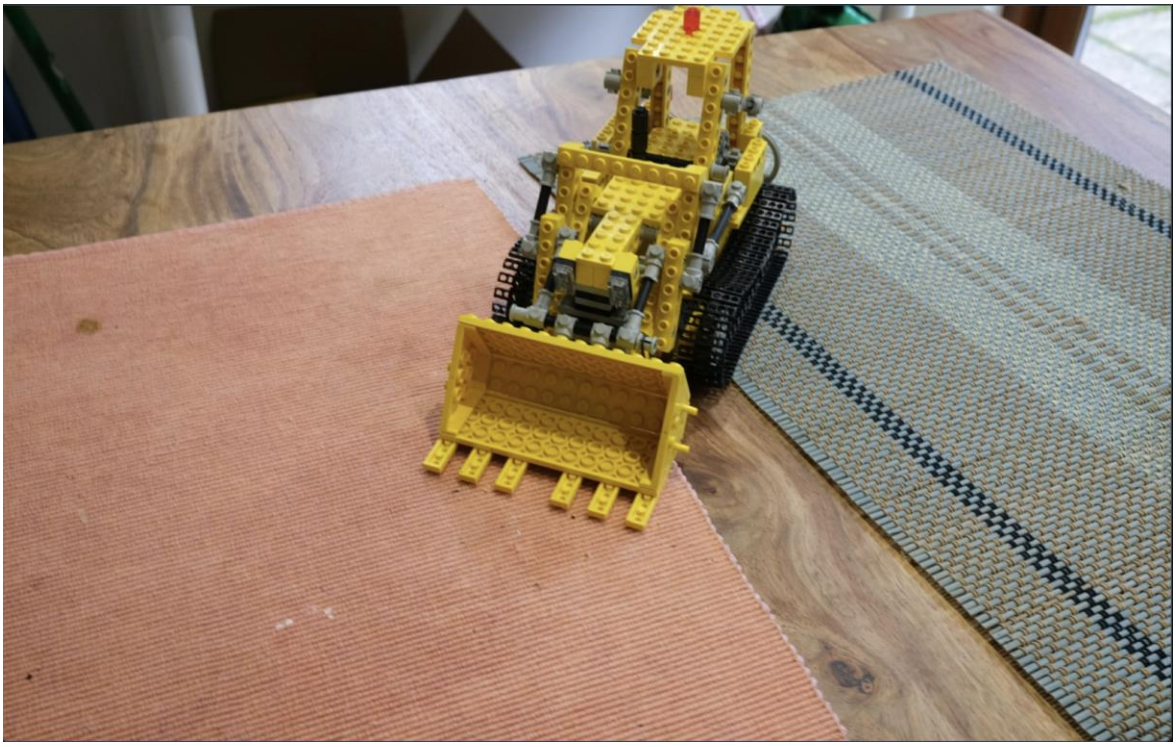

Figure 1: Input example - A LEGO bulldozer on a wooden table, representing a typical subject for 3D reconstruction

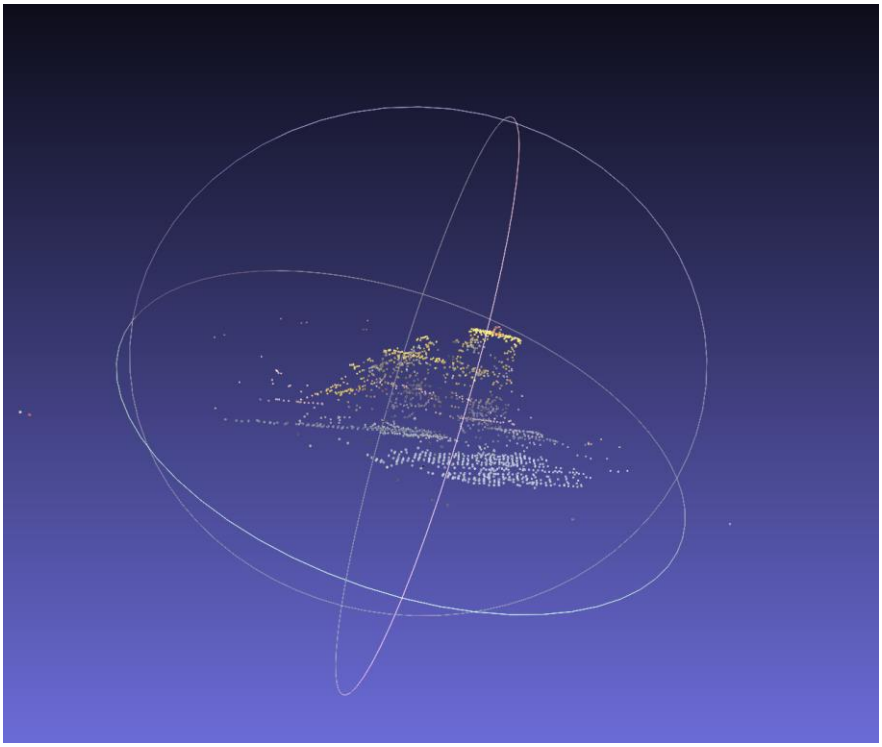

Figure 2: Camera pose visualization - Red pyramids show camera positions in a circular arrangement around the reconstructed scene

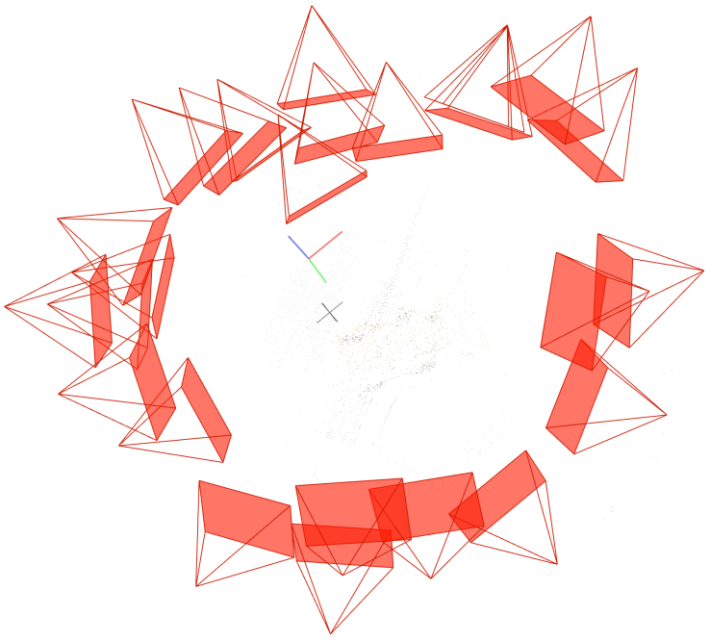

Figure 3: 3D point cloud - The sparse reconstruction showing feature points in 3D space with coordinate axes

The reconstruction results provide valuable insights:

1. **Camera Positioning**:
    a. The red pyramids in COLMAP GUI represent camera positions and orientations
    b. Their arrangement reveals the camera path during image capture

2. **Point Cloud Analysis**:
    a. ~1800 reconstructed 3D points from 25 images indicates good tracking quality
    b. Point density varies based on texture richness in the scene

3. **Scene Understanding**:
    a. The sparse reconstruction captures the fundamental structure of the kitchen/statue scene
    b. This provides a foundation for potential dense reconstruction

4. **Visual Assessment:**
    a. MeshLab visualization shows the raw point cloud quality
    b. COLMAP GUI provides context by showing both points and cameras

7. Conclusions & Future Work

7.1 Key Findings

1. **End-to-End Learning Benefits**: VGGSfM demonstrates that even well-established pipelines like SfM can benefit from end-to-end differentiable learning

2. **Simplified Architecture**: The approach is simpler than traditional SfM frameworks while achieving better performance

3. **Real-World Applicability**: Successfully works with custom image sets, despite implementation challenges

4. **Computational Considerations**: CPU-only implementations are possible but significantly slower than GPU-accelerated processing

7.2 Future Directions

1. **Dense Reconstruction**: Integrating with Depth-Anything-V2 for complete dense reconstruction

2. **Dynamic Scene Handling**: Further exploration of mask-based filtering for dynamic objects

3. **Performance Optimization**: Investigating parameter tuning for optimal speed-accuracy tradeoffs

4. **Integration with Downstream Applications**: Exploring use in neural rendering, AR/VR, and scene understanding

7.3 Practical Applications

1. **3D Asset Creation**: Generating 3D models from image collections

2. **Scene Documentation**: Architectural and archaeological documentation

3. **Visual Localization**: Camera pose estimation for augmented reality

4. **Neural Field Initialization**: Providing camera poses and sparse points for NeRF and similar techniques

8. Related Applications & Integration Possibilities

VGGSfM has broad applications across computer vision and extends beyond just 3D reconstruction:

8.1 Neural Rendering Applications

- **NeRF Initialization**: Providing camera poses and sparse points for Neural Radiance Fields
- **Novel View Synthesis**: Creating new viewpoints of scenes from limited inputs
- **Multi-view Consistency**: Enforcing geometric consistency in neural rendering pipelines

8.2 Computer Vision Pipelines

- **AR/VR Content Creation**: Generating 3D assets from casual photography
- **Scene Understanding**: Complementing semantic segmentation with geometric structure
- **SLAM Systems**: Improving robustness of Simultaneous Localization and Mapping

8.3 Specialized Domains

- **Cultural Heritage Preservation**: Documenting archaeological sites and artifacts in 3D
- **E-commerce**: Creating 3D product visualizations from product images
- **Architectural Visualization**: Converting 2D building photos into 3D models

8.4 Integration with Other Technologies

- **Integration with SAM/SAM2**: Combining with segment anything models for object-aware reconstruction
- **Track-Anything Pipeline**: Joining with video segmentation for dynamic scene understanding
- **Depth-Anything-V2**: Creating dense depth maps aligned with sparse SfM reconstructions

9. References

1. Wang, J., Karaev, N., Rupprecht, C., & Novotny, D. (2024). VGGSfM: Visual Geometry Grounded Deep Structure From Motion. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 21686-21697.

2. Project Website: https://vggsfm.github.io/

3. GitHub Repository: https://github.com/facebookresearch/vggsfm

4. CoTracker GitHub: https://github.com/facebookresearch/co-tracker

5. LightGlue: https://github.com/cvg/LightGlue

6. Depth-Anything-V2: https://github.com/DepthAnything/Depth-Anything-V2

10. Colab Implementation Process

The specific implementation followed in the Colab notebook includes:

10.1 Repository and Dependency Setup

The implementation begins with cloning the VGGSfM repository and installing necessary dependencies:

Clone the VGGSfM repository

```
!git clone https://github.com/facebookresearch/vggsfm.git
%cd vggsfm


# Install system dependencies
!apt-get update
!apt-get install -y colmap ffmpeg


# Install Python dependencies
!pip install -U pip
!pip install opencv-python hydra-core omegaconf visdom pycolmap kornia einops
!pip install git+https://github.com/cvg/LightGlue.git
!pip install git+https://github.com/facebookresearch/dinov2.git
```

10.2 PyTorch Version Management

A critical aspect of the implementation involved managing PyTorch versions to ensure compatibility:

```
# Install specific PyTorch version compatible with the model
!pip install torch==2.0.1 torchvision==0.15.2 torchaudio==2.0.2 --index-url
https://download.pytorch.org/whl/cu118
!pip install xformers==0.0.29
```

10.3 Patching Dependencies

The implementation required patching some dependencies to resolve version conflicts:

```
# Clone and patch dinov2 to accept newer xformers version
!rm -rf dinov2
!git clone https://github.com/facebookresearch/dinov2.git
%cd dinov2
!sed -i 's/xformers==0.0.18/xformers>=0.0.18/' setup.py
!pip install .
```

10.4 Building Ceres Solver

A significant portion of the implementation involved building Ceres Solver for optimization:

```
# Clone and build Ceres Solver
```

```
!git clone -b 2.1.0 https://github.com/ceres-solver/ceres-solver.git
%cd ceres-solver
!mkdir build && cd build
!cmake .. -DBUILD_TESTING=OFF
!make -j8
!make install
```

10.5 Running the VGGSfM Pipeline

The main pipeline was executed using the following command:

```
# Run the demo on the kitchen dataset
!python demo.py --config-path cfgs --config-name demo
```

The configuration used for the run included:

- Model: vggsfm_v2_0_0
- Image size: 1024
- Mixed precision: fp16
- Query frame number: 1
- Max query points: 512
- Scene directory: examples/kitchen

10.6 Implementation Challenges

Several challenges were encountered during the implementation:

1. **Dependency Conflicts**: The implementation struggled with compatibility between xformers and dinov2, requiring manual patching of the setup files.

2. **Ceres Solver Building**: Building Ceres Solver presented challenges, especially with finding the right versions of dependencies like abseil.

3. **CUDA Compatibility**: Ensuring CUDA compatibility across PyTorch, torchvision, and other packages required specific version combinations.

4. **Missing pyceres Module**: The implementation encountered issues with the pyceres module, which is critical for triangulation in the VGGSfM pipeline.

10.7 Hardware Considerations

The implementation was adapted to run on Google Colab's hardware:

- Used CUDA-enabled runtime when available
- Managed memory usage by limiting max_query_pts to 512
- Implemented mixed_precision=fp16 for more efficient computation
- Disabled certain visualization features that would require more resources