CSS-587 Project Design:
LP-SIFT Implementation and Optimization

David Li, Ben Schipunov, Kris Yu

November 24, 2025

1 Project Objectives

This is a list of objectives to accomplish. It is derived from our initial Project Proposal. Our intent is to verify the paper and result is legitimate, convert the scripts to C++, rerun the experiment, optimize, and output a working demo. Along the way we should observe and discuss results.

- ☐ Validate paper results using provided MATLAB implementation
- ☐ Implement LP-SIFT in C++ with OpenCV (library + CLI tool)
- ☐ Benchmark against SIFT, ORB, BRISK, SURF on small/medium/large images
- ☐ Apply CPU optimizations for local peak algorithm
- ☐ Implement multi-image mosaic stitching for random fragments (demo application)
- ☐ **Stretch**: GPU acceleration with CUDA

2 Problems to Solve

2.1 Overview of Image Stitching Challenge

Image stitching combines multiple images with overlapping fields of view to create larger, seamless panoramic images. Traditional SIFT-based methods are robust but computationally expensive, especially for large images (>3 megapixels).

Key Bottleneck: SIFT's Gaussian pyramid and Difference-of-Gaussian (DoG) pyramid construction consumes the majority of processing time, making it impractical for real-time or large-scale applications.

LP-SIFT Solution: Replaces the Gaussian pyramid with **multiscale local peak detection**, achieving 10–100× speedup while maintaining SIFT's robust descriptor computation and matching quality.

2.2 Specific Tasks to Accomplish

2.2.1 Reproduce LP-SIFT Experiment in C++

1. **Implement a test framework** to run full image stitching pipeline on SIFT, SURF, ORB, BRISK and finally our LP-SIFT
2. **Tabulate our results** in identical table to paper for review and comparison

2.2.2 Implement LP-SIFT subclassing cv::Feature2D

1. **Override parent methods** (see Table 1 and Section 3.2.1)
   - Override detection methods for multiscale local peak extraction
   - Reuse SIFT descriptor computation (possibly by subclassing `cv::SIFT`)
2. **Build complete stitching pipeline** (see Section 4)
3. **Create benchmark framework** to compare SIFT, SURF, ORB, BRISK, and LP-SIFT

3 Classes and Architecture

3.1 OpenCV Integration via cv::Feature2D

Our implementation leverages OpenCV's `cv::Feature2D` interface to ensure LP-SIFT is a drop-in replacement for existing detectors (SIFT, ORB, BRISK, SURF). This enables the `ImageStitcher` to work with any detector via dependency inversion.

Table 1: cv::Feature2D Interface - Methods to Override

Return Type	Method Signature & Description
virtual void	`detect(InputArray image, std::vector<KeyPoint> &keypoints, InputArray mask=noArray())` Detects keypoints in an image.
virtual void	`compute(InputArray image, std::vector<KeyPoint> &keypoints, OutputArray descriptors)` Computes descriptors for detected keypoints.
virtual void	`detectAndCompute(InputArray image, InputArray mask, std::vector<KeyPoint> &keypoints, OutputArray descriptors, bool useProvidedKeypoints=false)` Combined detection and descriptor computation (most used).
virtual int	`descriptorSize() const` Returns the descriptor size (128 for SIFT/LP-SIFT).
virtual int	`descriptorType() const` Returns descriptor type (CV_32F for SIFT/LP-SIFT).
virtual int	`defaultNorm() const` Returns norm type for matching (NORM_L2 for SIFT/LP-SIFT).

3.2 Core Classes

3.2.1 LPSIFTDetector

Primary class implementing the LP-SIFT algorithm. Inherits from `cv::Feature2D` and overrides methods from Table 1.

Key Parameters:

- `windowSizes`: Vector of interrogation window sizes L (e.g., {32, 40, 64, 128})
- `linearNoiseCoeff`: Small noise coefficient α (typically 10^{-6})
- `beta0`: Square region scaling factor for descriptor computation

Implementation Strategy:

- `detect()`: Implements multiscale local peak extraction (replaces DoG pyramid)
- `compute()`: Reuses standard SIFT descriptor computation
- `detectAndCompute()`: Combined operation for efficiency
- May subclass `cv::SIFT` directly to inherit descriptor computation

3.2.2 ImageStitcher

Pipeline orchestrator implementing the complete stitching workflow (detailed in Section 4). Uses dependency inversion by accepting any `cv::Feature2D` detector.

3.2.3 BenchmarkRunner

Automated testing framework for comparing LP-SIFT against SIFT, ORB, BRISK, and SURF across multiple datasets.

Output: CSV files with timing, feature counts, and quality metrics for analysis.

4 Image Stitching Pipeline

Figure 1 shows the complete stitching workflow. The `ImageStitcher` class implements this pipeline with the following stages:

1. **Image Preprocessing**: Add linear noise filter ($\alpha \ll 1$) to both reference and registered images
2. **Feature Detection**: Extract keypoints using any `cv::Feature2D` detector (SIFT, SURF, ORB, BRISK, or LP-SIFT)
3. **Descriptor Computation**: Compute 128-element SIFT descriptors for all keypoints
4. **Feature Matching**: Match descriptors using BFMatcher (per paper) or FLANN
5. **Homography Estimation**: RANSAC to find transformation matrix H from inlier correspondences
6. **Image Warping and Blending**: Transform registered image and merge with reference

Key Design: LP-SIFT only replaces Stage 2 (detection), while Stages 3–6 remain identical to traditional SIFT-based stitching, ensuring fair comparison.

5 Data Structures

5.1 StitchingMetrics

Comprehensive metrics structure for performance evaluation and comparison.

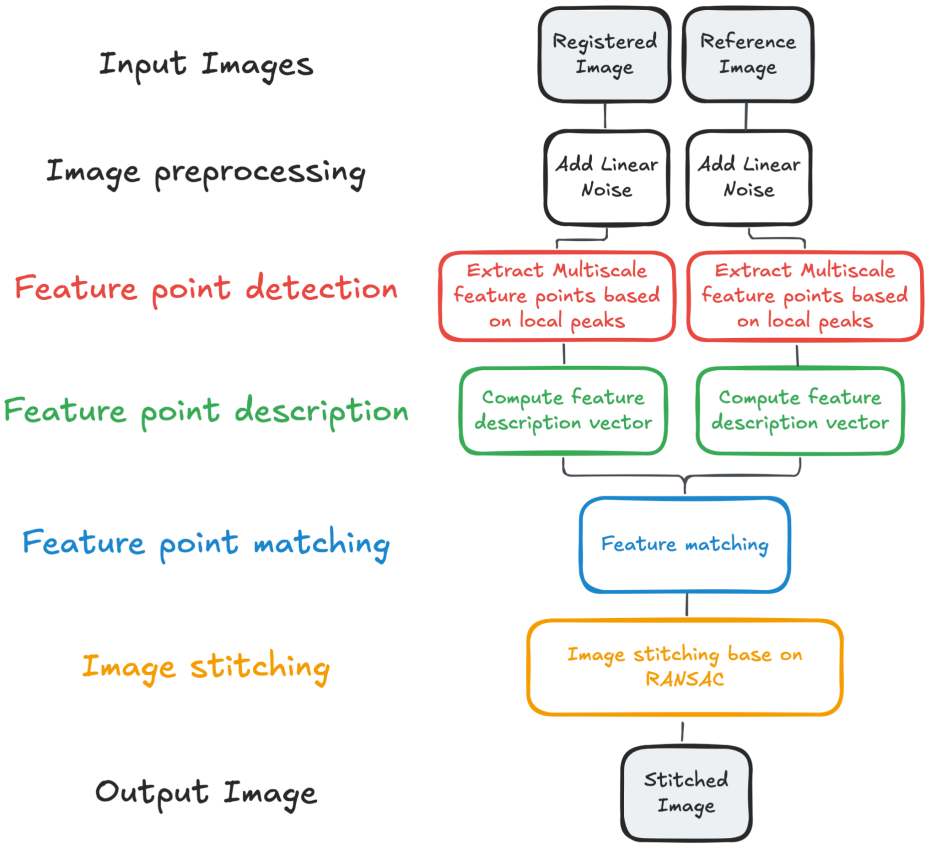

Figure 1: Image Stitching Pipeline - LP-SIFT replaces traditional SIFT only in the feature detection stage (highlighted in red), while reusing the same descriptor computation and matching pipeline.

Table 2: Metrics to Record

Metric	**Units**	**Description**
Stitching times	Seconds (1/100 precision)	Before keypoint detection to After stitching image returned
Keypoint detection time (Reference)	Seconds (1/100 precision)	Before algorithm start to After reference keypoints returned
Keypoint detection time (Registered)	Seconds (1/100 precision)	Before algorithm start to After registered keypoints returned
Number of features detected (Reference)	Keypoint Features	Number of reference keypoints returned from detectors
Number of features detected (Registered)	Keypoint Features	Number of registered keypoints returned from detectors
Resolution of image (Reference)	[Width, Height] in pixels	Image dimensions used to correlate with performance findings
Resolution of image (Registered)	[Width, Height] in pixels	Image dimensions used to correlate with performance findings
Number of matches	Integer	Feature correspondences after matching
Number of inliers	Integer	RANSAC inliers used for homography

5.2 Additional Data Structures

- **HomographyMatrix**: Stores 3×3 transformation matrix, inlier count, reprojection error
- **ImageGraph**: Connectivity graph for multi-image stitching (Section 4 of paper)
- **LocalPeak**: Position, intensity value, scale (L), type (max/min)

6 Test Data and Testing Methodology

6.1 Dataset Source

We will select datasets from the comprehensive **Image Stitching Dataset Repository**[1], which provides over 25 benchmark datasets spanning traditional and deep learning approaches.

6.2 Dataset Selection Strategy

To thoroughly evaluate LP-SIFT across different scenarios, we will select datasets based on:

1. **Image Size Diversity**: Small (<1MP), Medium (1–3MP), Large (>3MP)
2. **Transformation Types**: Translation, rotation, scaling, perspective, parallax
3. **Scene Complexity**: Textured vs. low-texture, indoor vs. outdoor
4. **Number of Images**: Two-view pairs and multi-image sets (3–10 images)

6.3 Candidate Datasets Under Consideration

Table 3: Candidate datasets from the repository (final selection pending)

Dataset	Year	Size	Characteristics
APAP	2013	8 sets	Diverse scenes; projective warping
Parallax-tolerant	2014	36 pairs	Two-view with parallax challenges
NISwGSP	2016	42 sets	Multi-image sets; natural scenes
SEAGULL	2016	24 pairs	Mobile phone captures; challenging parallax
GES-50	2022	50 groups	Most comprehensive; 2–35 images per set
OpenPano	2016	8 sets	4–38 images per set; panoramas

6.4 Proposed Minimal Test Suite

To be finalized (10–12 test cases):

1. **Small Images (2–3 pairs)**: From APAP or Parallax-tolerant
   - Target: 600–800×400–600 pixels
   - Purpose: Baseline comparison, verify correctness
2. **Medium Images (3–4 pairs)**: From SEAGULL or NISwGSP

[1] `https://github.com/visionxiang/Image-Stitching-Dataset`

- Target: 1000–2000×1500–3000 pixels
- Purpose: Moderate speedup demonstration

3. **Large Images (2–3 pairs)**: Custom captures or OpenPano
   - Target: 3000×4000+ pixels
   - Purpose: Showcase LP-SIFT's primary advantage
4. **Multi-Image Mosaic (1–2 sets)**: From GES-50 or NISwGSP
   - Target: 5–10 images per set
   - Purpose: Validate random stitching strategy

Note: Final dataset selection will be documented in our benchmark results and GitHub repository.

6.5 Testing Methodology

6.5.1 Phase 1: MATLAB Validation

- Run paper's provided MATLAB implementation
- Verify 10–100× speedup claims on test datasets
- Document baseline performance metrics

6.5.2 Phase 2: C++ Implementation

- Implement `LPSIFTDetector` class
- Unit test components: linear noise, peak detection, descriptors
- Integration test with small image pairs
- Verify correctness against MATLAB output

6.5.3 Phase 3: Comprehensive Benchmarking

Protocol: For each dataset and each algorithm (SIFT, ORB, BRISK, SURF, LP-SIFT):

1. Load reference and registered images
2. Start timer
3. Detect features in both images (record time and counts)
4. Compute descriptors
5. Match descriptors using BFMatcher
6. Apply RANSAC to compute homography
7. Warp and blend images
8. Stop timer

9. Measure output quality (visual inspection, PSNR, SSIM)
10. Record all metrics to CSV

Success Criteria:

- C++ LP-SIFT matches MATLAB LP-SIFT within 10% timing
- LP-SIFT achieves $> 10\times$ speedup vs. SIFT on large images
- Stitching quality (PSNR) within 5% of SIFT
- Successfully stitches all datasets without failure

7 Conclusion

This design demonstrates a thorough plan for implementing LP-SIFT in C++ with OpenCV integration. By leveraging the `cv::Feature2D` interface, we ensure compatibility with existing vision pipelines while achieving dramatic performance improvements for large-scale image stitching.

Planned Contributions:

1. First open-source C++ implementation of LP-SIFT
2. Comprehensive benchmark against state-of-the-art detectors
3. Production-ready library and CLI tool
4. Multi-image mosaic stitching without prior knowledge
5. (Stretch) GPU-accelerated version