ABSTRACTED WORKFLOW FRAMEWORK WITH A STRUCTURE FROM MOTION APPLICATION

[1,2]*Adam J. Rossi,* [1]*Harvey Rhody,* [1]*Carl Salvaggio*

[1]Rochester Institute of Technology
Chester F. Carlson Center for Imaging Science
54 Lomb Memorial Drive
Rochester, NY 14623
{ajr3217, rhody, salvaggio}@cis.rit.edu

[2]*Derek J. Walvoord*

[2]ITT Exelis
Geospatial Systems
400 Initiative Drive
Rochester, NY 14624
derek.walvoord@exelisinc.com

ABSTRACT

It is common for many disparate software tools to be developed in both open source and proprietary environments, which are incompatible and leave the community with disposable software. As such, a processing workflow has been abstracted and implemented in a generic fashion, where the core engine can be easily extended into domains for applications that require functionality from many independent components. Specifically, there have been a host of tools developed to address the structure from motion (SfM) domain that require interfacing in order to create a functional system. Many application-specific solutions have been constructed that are inflexible and repeatedly duplicated. The theory of operation behind the core framework and components for the SfM application are presented to demonstrate how the abstracted workflow can be easily applied to generate 3D models from 2D imagery of a scene captured from multiple cameras and varying perspectives.

Index Terms— Software workflow, Structure from Motion (SfM), 3D reconstruction, 3D modeling

1. INTRODUCTION

The existence of a host of disparate applications developed for the SfM domain necessitates the need for an abstracted workflow. The ability to seamlessly replace one component for another during development and experimentation allows the user to quickly optimize the workflow and understand how an individual component affects the overall system. Not only does the concept of a generic workflow extend easily into multiple domains, it also provides a solid foundation for the simplification of future development and re-use. Ultimately, a flexible and scalable workflow framework was developed with a self-documenting feature that encourages re-use of the components.

2. WORKFLOW CONCEPT

In many domains, the concept of a workflow, or chain, is applied to carry out a sequence of tasks that can be represented by individual stages. This concept is often utilized in the area of image processing. The output from a given stage is fed into subsequent stages, and becomes the input required to carry out a specific process. This process continues sequentially until the final stage is complete. In the current study, the generic workflow framework was implemented in the Python programming language due to its object-oriented, platform-agnostic, and ubiquitous nature. In addition, Python has gained a large following throughout the scientific community and most of the anticipated target users are familiar with this language. Chains can be built programmatically or through the usage of a graphical user interface (GUI). Both methods support persistence for interoperability between environments and future usage of the constructed chain. The stages included as part of the core framework, and custom stages implemented by users, are dynamically discovered and loaded for ease of deployment, integration, operation, and extensibility.

2.1. Stages

The fundamental component of the workflow is the stage. The stage encapsulates a function and should be developed in such a manner that provides general capability across chain instances (i.e., it should be designed generically so that it can be leveraged by other applications).

A stage defines three classes of information within it, including the properties and self-documentation, input interface, and output interface (Fig. 1). The properties required in the constructor are used to control behavior and modes of the stage. Additionally, information contained in the constructor facilitates self-documentation of the stage, whereby it can be leveraged to provide online help for users. Next, the input interface is declared with type information. This is required

prior to rendering the chain to validate interface consistency among connected stages. Finally, the output interface is defined in the same manner as the input. The advantage of defining stage interfaces is that it allows the user to conveniently exchange one stage for another.

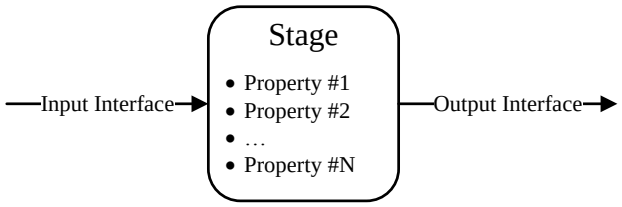

Fig. 1. Representative stage structure showing properties and input/output interfaces. The input data comes from previous stages, while the output data is fed into subsequent stage(s). The individual stage interfaces are well-defined such that equivalent stages can be swapped for other implementations in a chain.

At the core of the stage lies the execution method, where the main function of the stage is invoked. Due to the *demand-pull* nature of the chain, which is explained further in 2.3, stages request required input parameters from the associated up-stream stage(s) outputs. This affects a recursive request up-stream. The output parameters from the up-stream stage(s) and the properties provided upon construction by the instantiator comprise all of the required information needed to carry out the function of a given stage. The stage executes and sets the output parameters as defined on the output interface. Finally, the framework ensures that all of the outputs are set and of the correct type upon the completion of stage execution. This guarantees consistency with the interface contracts specified between connected stages.

2.2. Chain

A chain is composed of an arbitrarily complex sequence of stages representing a desired workflow that is required to carry out a task. The user has the ability to employ a stage as input into one or many down-stream stages, or even include it in different chain segments. In addition, the user may construct the chain in such a manner that results in more than one output stage. In this case, the output stages and the respective chains can be rendered in parallel or sequentially, whichever is desired.

2.3. Chain Rendering

Once a chain is constructed from a collection of stages, it is ready for rendering. The verb *render* is borrowed from the image processing concept, describing the method by which the chain is executed to produce the desired output. The chain is rendered in a *demand-pull* fashion. This means that the chain is rendered from the perspective of the output stage. The request is made from stage to stage up-stream, and each stage executes its function in order to produce an output for the stage(s) in front of it.

Caching is employed as the chain is rendered, such that a stage and its associated outputs are only calculated once. This is essential in minimizing the computation time of the overall chain. The caching scheme described is beneficial when a chain is constructed with one stage connected to multiple up- or down-stream stages. It is important to note that the developer is isolated from the rendering and caching capability, as the development methodology described in 2.1 ensures that both are employed as part of the framework.

2.4. Abstraction Benefits

Using the generic components of the workflow framework described in Section 2, the user defines customized stages by inheriting from a base stage class. This provides the self-documentation feature, interface validation, caching, and rendering capability. A benefit to this abstract approach is that an enhanced chain capability can be easily developed to operate in multi-threaded, distributed, or high performance computing environments without modification to the concrete stage classes. The powerful benefit of abstraction easily allows for scalability. Additionally, this architecture provides the flexibility to develop an entirely different rendering or caching scheme without modification to the stage implementations.

3. STRUCTURE FROM MOTION APPLICATION

Utilizing this generic workflow framework, a set of stages for the SfM application have been implemented. The high level SfM workflow is depicted in Fig. 2.

3.1. Sources

The first stage of the SfM chain is to identify the desired input images. A basic image source is provided, which allows the user to specify a directory and extension of the image files of interest. This stage has no input interface, but generates a collection of image objects that contain path and metadata required for stages down-stream. The properties of this stage include a path and file extension string. Additional source stages were developed that include recursively searching, filtering, sorting, randomizing, symbolically linking, and conversion of image file formats, however their detail is omitted here for brevity.

3.2. Feature Extraction

Once the input images are declared, a feature extraction algorithm must be run on each of the images. This stage generates a set of feature descriptors or vectors for each image. The

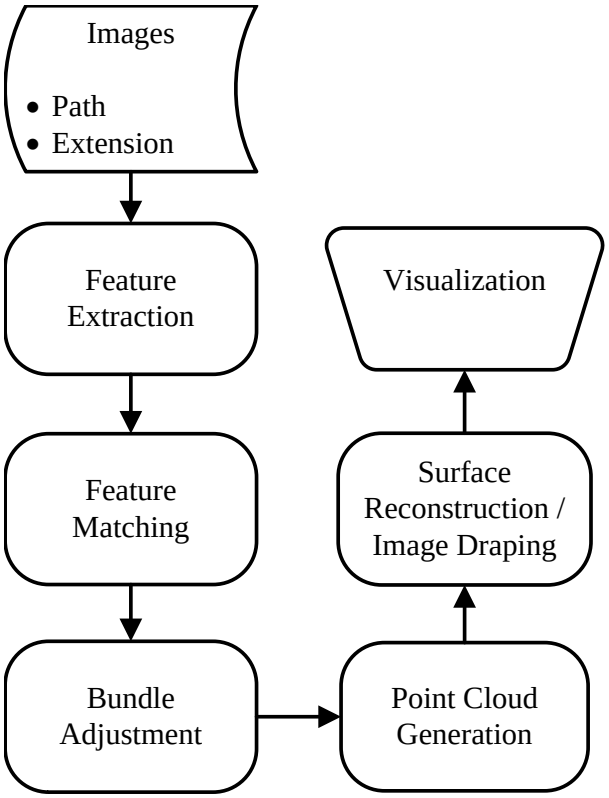

Fig. 2. High-level depiction of a chain necessary to carry out the procedure for the SfM task. Due to the demand-pull method, the chain render is invoked from the output stage (Visualization).

process is inherently parallelizable, thus motivating the development of high performance computing techniques within the underlying chain implementation. There are a host of feature extractors that have been developed, the most common being the scale-invariant feature transform (SIFT) algorithm [1], due to its ability to provide a robust descriptor across varying imaging conditions. Other common feature extractors include SURF [2], DAISY [3], RIFT [4], and GLOH [5]. It is imperative that every feature extraction stage output compatible descriptors such that the stages down-stream are indifferent to the method employed. This statement holds true for all stages to promote re-use, flexibility, and interoperability.

3.3. Feature Matching

The next step in the SfM chain is to compute a table of feature matches relating all combinations of descriptors between images. This process effectively computes tie points or correspondences between images. Typically, this is a computationally expensive operation due to the sheer number of comparisons required. However, similar to the feature extraction step, the process lends itself well to parallelization, and can be cleanly broken up into individual tasks with no inter-dependencies. A multi-core or clustered computing environment can be easily employed to shorten the computation time required.

There are a variety of techniques available, from brute-force feature matching to model-fitting algorithms, that employ random sample consensus (RANSAC) [6]. All implementations should be transparent to stages connected within the chain. As a key benefit to the modular architecture, new stages for feature matching can easily be added to the system in the future.

3.4. Bundle Adjustment

At this point, correspondences for all of the images (views) establish a full image-based geometry for a coarse estimate of the 3D sparse point cloud. After triangulation, the full set of equations that relate points in image coordinates to world coordinates is typically subjected to a nonlinear optimization technique. The popular Levenberg-Marquardt algorithm [7] is used to perform a bundle adjustment. These procedures employ multi-view geometry (MVG) techniques to simultaneously establish consistent camera models / matrices for each of the views by using 2D image correspondences and effectively recovering the 3D structure of the scene. This describes the origin of the term "SfM"- the structure is recovered from camera motion, where the motion is nothing more than different perspective views of the scene captured from single or multiple cameras. This process relies heavily on the theory described by Hartley and Zisserman [8] and the implementation of the Bundler software [9], which utilizes the sparse bundle adjustment (sba) library [10] developed by Lourakis and Argyros.

3.5. Point Cloud Generation

Once a set of camera models are derived for a scene, a 3D point cloud must be generated. This process employs the camera models, 2D image correspondences, features around the 2D correspondences, and a triangulation algorithm to derive 3D points for the scene. One such implementation is patch-based multi-view stereo software (PMVS) [11].

3.6. Surface Reconstruction / Image Draping

The 3D model generated from the point cloud process might be sufficient for automated exploitation algorithms, however, the model produced at this stage is typically sub-optimal for human consumption. Therefore, an image draping or surface reconstruction algorithm, such as Poisson Surface Reconstruction, [12] must be performed on the 3D point cloud, typically requiring the estimated camera models and original imagery to carry out the task.

3.7. Visualization / Exploitation

The final stage in the SfM workflow is to either visualize the 3D model or exploit it using automated methods. There are viewers available, including Meshlab [13] and Blender [14], that allow for interactive manipulation of the model and allow for the camera perspective to be defined. The 3D nature of the data enables realization of functions and capabilities typically not available from 2D imagery. These include, but are not limited to, mensuration, generating fly-through animations, dynamic perspective changes, and elevation model extraction. Fig. 3 exemplifies a 3D model of downtown Rochester, New York, which was rendered using the SfM workflow from multiple 2D images captured using RIT's WASP sensor. In this case, the 3D model was visualized using the Blender viewer.

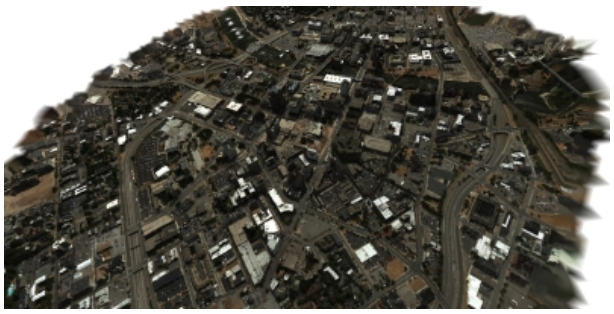

Fig. 3. 3D model generated of the city of Rochester using imagery from RIT's WASP sensor and the SfM workflow implementation, rendered by the Blender viewer.

4. CONCLUSION

A generic workflow framework was architected and implemented in Python. This framework addresses aspects related to efficient execution, ease of implementation, scalability, and documentation. Specifically, a collection of stages were developed to facilitate the SfM task. The workflow and stages developed provide a functional SfM system and establishes a platform for extending this particular application while providing a generic capability for other domains. This open system facilitates collaborative, incremental development to form the basis for both a test framework and an operational heterogeneous workflow system.

5. REFERENCES

[1] David G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004.

[2] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, June 2008.

[3] Engin Tola, Vincent Lepetit, and Pascal Fua, "A fast local descriptor for dense matching," in *Conference on Computer Vision and Pattern Recognition*, Alaska, USA, 2008.

[4] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce, "A sparse texture representation using local affine regions," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, pp. 1265–1278, 2005.

[5] Krystian Mikolajczyk and Cordelia Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[6] Martin A. Fischler and Robert C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[7] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[8] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[9] Noah Snavely, Steven M. Seitz, and Richard Szeliski, "Photo tourism: exploring photo collections in 3d," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 835–846, July 2006.

[10] M.I. A. Lourakis and A.A. Argyros, "SBA: A Software Package for Generic Sparse Bundle Adjustment," *ACM Trans. Math. Software*, vol. 36, no. 1, pp. 1–30, 2009.

[11] Yasutaka Furukawa and Jean Ponce, "Accurate, dense, and robust multi-view stereopsis," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 32, no. 8, pp. 1362–1376, 2010.

[12] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe, "Poisson surface reconstruction," in *Proceedings of the fourth Eurographics symposium on Geometry processing*, Aire-la-Ville, Switzerland, Switzerland, 2006, SGP '06, pp. 61–70, Eurographics Association.

[13] Paolo Cignoni, Massimiliano Corsini, and Guido Ranzuglia, "Meshlab: an open-source 3d mesh processing system," April 2008.

[14] Ton Roosendaal, "Blender 3d computer graphics software," Oct. 2012.