

# The InSAR Scientific Computing Environment

Paul A. Rosen, Jet Propulsion Laboratory, California Institute of Technology, USA

Eric Gurrola, Jet Propulsion Laboratory, California Institute of Technology, USA

Gian Franco Sacco, Jet Propulsion Laboratory, California Institute of Technology, USA

Howard Zebker, Stanford University, USA

## Abstract

We have developed a flexible and extensible Interferometric SAR (InSAR) Scientific Computing Environment (ISCE) for geodetic image processing. ISCE was designed from the ground up as a geophysics community tool for generating stacks of interferograms that lend themselves to various forms of time-series analysis, with attention paid to accuracy, extensibility, and modularity. The framework is python-based, with code elements rigorously componentized by separating input/output operations from the processing engines. This allows greater flexibility and extensibility in the data models, and creates algorithmic code that is less susceptible to unnecessary modification when new data types and sensors are available. In addition, the components support provenance and checkpointing to facilitate reprocessing and algorithm exploration. The algorithms, based on legacy processing codes, have been adapted to assume a common reference track approach for all images acquired from nearby orbits, simplifying and systematizing the geometry for time-series analysis. The framework is designed to easily allow user contributions, and is distributed for free use by researchers. ISCE can process data from the ALOS, ERS, EnviSAT, Cosmo-SkyMed, RadarSAT-1, RadarSAT-2, and TerraSAR-X platforms, starting from Level-0 or Level 1 as provided from the data source, and going as far as Level 3 geocoded deformation products. With its flexible design, it can be extended with raw/meta data parsers to enable it to work with radar data from other platforms.

## 1 Introduction

In 2008 at community request, NASA sponsored an Interferometric Synthetic Aperture Radar (InSAR) processing workshop to assess the strengths and weaknesses of the existing InSAR processing packages at the time, define the capabilities of the next-generation processors required by the user community, and set the standards and structure for new InSAR processor development. The workshop participants were geophysicists and developers, and the targeted user community was those who wish to manipulate InSAR data toward a specific scientific objective, often in an exploratory fashion and in combination with other data sets and those interested in the details of the processing workflow, with an interest in access to a processing package that can be modified and extended to meet their needs. These users generally require well-defined interfaces and documentation to be able to contribute to the code development or extension.

The top requirements for the next generation radar processing package coming out of the 2008 workshop were: (1) precise and well-characterized products; (2) flexible and extensible modular code to encourage

modification and improvement by the user community; (3) and a comprehensive set of user documentation.

The NASA Earth Science Technology Office funded the InSAR Scientific Computing Environment (ISCE) project through the Advanced Information Systems Technology (AIST) program to implement these recommendations [1,2]. The software package was specifically designed to support that portion of the user community that requires access to code so they can modify and extend it for their research purposes, and who interact with large volumes of data in an exploratory fashion. Ideally, the package would be easy for novices to use, but also deep and sufficiently documented that computer-savvy geophysicists and geologists would be able to understand it, and adapt it to their needs.

We collected community-based requirements for InSAR processing methods and generalized data models, through the 2008 workshop final report and by interacting with our key science collaborators on the project. We then used these requirements to define an object-oriented framework. With the framework in place, we populated it with processing modules. Along the way, we are creating documentation of the

framework, modules, and use cases. The project duration was three years, ending in mid 2012, and the ISCE package is complete, in the sense that the capabilities proposed to NASA have been implemented and tested. We anticipate that extensions and improvements will occur in much the same way that the JPL ROI\_PAC InSAR package improvements have over the past 10 years [3].

## 2 Architectural Elements

At the core of the ISCE architecture are two legacy InSAR processing packages: ROI\_PAC [3] and STD\_PROC [4]. These software packages are written in Fortran and C and with scripts written in Perl (ROI\_PAC) and Python (STD\_PROC). Both are written in a similar style that is very effective at accomplishing the processing steps but neither are particularly flexible or extensible for users to experiment with new modules or workflows. The ISCE architecture seeks to inject some modern software principles that allow for easier use and greater flexibility and extensibility.

### 2.1 Components

We restructured the executable code elements in ROI\_PAC and STD\_PROC by extracting the core compute elements from the overall program flow. These programs followed the typical input-process-output programming style. We surround the core compute elements with structures that deliver services to the legacy programs, users and developer. The services replace legacy code interactions with the external world that were previously handled using mostly primitive language features.

Figure 1 shows the architecture of a component that has an embedded legacy core. The processing components are built from framework components and properties through either class inheritance or composition. Configuration and control parameters flow from a controlling or driving application at the top into the component initialization method. The con-

figuration and control parameters are derived from user inputs, either from the command line or from input files, and defaults defined in preferences files or within the application itself. The component itself may also define defaults for parameters. Defaults can always be overridden by user inputs.

The components, applications, and other support software involve a mixture of different programming languages and styles. This multilingual structure requires proper use of application program interfaces (APIs) for effective cooperation among languages. We use C/C++ as the binding intermediary between Python and Fortran because there is a standard Python API that allows Python and C programs to interact.

A component must be instantiated in another type of component called an *application*, which has the responsibility of collecting the user inputs and of managing its components from their initialization to the flow of data through them to their finalization.

### 2.2 Software

The main ISCE applications and components are contained under the Applications and Packages directory areas of the distribution. Packages are collections of logically related Components, Legacy Cores, and other support software. Current Packages include: *icesys*, which contains the ISCE system or framework components and properties as well as several APIs; *isceobj*, which contains class definitions for several objects used by the components; *mroipec*, which contains the recasting of ROI\_PAC into components; and *stdproc*, which contains the recasting of STD\_PROC into components. Contributed software can be located in self-contained areas of the ISCE distribution without intermingling with the main ISCE distribution. For example, a new package for polarimetric data processing and calibration, and ionospheric estimation is included in the ISCE distribution as a contributed package [5].

### 2.3 Polymorphism

ISCE supports dynamic alteration of the software for a processing run using object-oriented *polymorphism* design patterns. ISCE allows two types of polymorphism: (1) *facility* polymorphism where major components may be morphed at run-time; and (2) a *plugin* type of polymorphism where lower level, common functions such as fast Fourier transforms (FFTs) may be selected at run-time. The mechanism is implemented through Facilities, which define a task and an interface that are implemented by a component. Registering a Component as a Facility alerts the Application to allow the user to specify an alternate Component at runtime to implement the Facility.

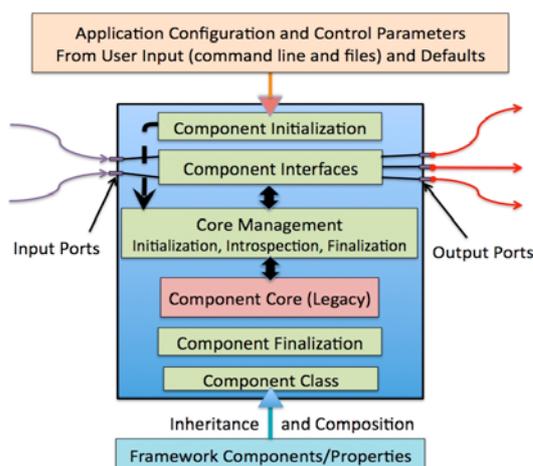


Figure 1 ISCE Component

## 2.4 Provenance

Provenance allows users to keep track of: the versions of applications, components, and other software that were used to produce a data product; the configuration parameters used to initialize those applications and components; the input data and other output data products at the time of creation of the data product of interest. Provenance enables an investigator to explore data by using different versions of the software or iteratively tweaking parameters, while keeping a record of what was done at every step. This record allows the user or colleagues to reproduce results exactly, by sharing scripts with the community, facilitating reproducible collaborations and publications.

ISCE supports provenance through database management and logging of processing steps and metadata at each step of the processing chain. Given the python-based object-oriented methods in ISCE, the code lends itself to being used within software packages with higher levels of sophistication that provide provenance capability as well.

## 2.5 Framework APIs

The ISCE framework contains various elements that support the ISCE component architecture. Key framework Application Program Interfaces (APIs) control processing flow among ISCE modules. These APIs are the:

- Image API – provides a set of versatile library functions for input and output operations on images
- Control API – is a set of classes, features and methodologies providing an easy, reproducible, extensible and reconfigurable mechanisms to pass data, and to set and examine attributes through *set* and *get* methods
- StdOE API – a C++ static class used for reconfigurable standard output and error.

## 3 Functional Capabilities

Repeat pass InSAR forms topographic and displacement maps from radar data collected at two different times by a platform such as an orbiting spacecraft. InSAR processing depends on precise knowledge of the position and velocity of the platform at the time of observation of a pixel at a given range and Doppler relative to the platform.

The STD\_PROC processor at the core of ISCE preserves the accuracy of its data products by taking advantage of the improved accuracy of orbit determination now available and implementing all of the code in a uniform geometric framework [4]. This approach, based on well-known motion compensation techniques, also facilitates analysis of a time series of

many observations of a particular location on Earth by its use of a motion-compensated geodetic coordinate system, referred to as SCH (in which S is the local along track direction, C is the cross track direction, and H is the height above the approximating sphere), is based on a local spherical approximation of the ellipsoidal Earth. The equations implemented in the processor are simplified by use of a spherical Earth and a corresponding circular approximation of the platform orbit.

ISCE can process data from the ALOS, ERS, Envisat, Cosmo-SkyMed, RadarSAT-1, RadarSAT-2, and TerraSAR-X platforms, starting from Level-0 or Level 1 as provided from the data source, and going as far as Level 3 geocoded deformation products. An example of ISCE output is shown in Fig. 2, a geocod-

Figure 2. “First-light” processed ALOS PALSAR interferogram from the ISCE framework.



ed interferogram with topography removed, from the ALOS PALSAR L-band SAR system. These data

were used to verify consistency of processing with the initial legacy code in STD\_PROC.

The STD\_PROC, and therefore ISCE, code is parallelized using MPI directives, and is very efficient. Phase unwrapping is implemented as an optional module that can be added before the geocoding modules. (Results in Fig. 2 are not phase-unwrapped.) Application scripts are also provided to generate consistently processed scenes over time for time series analysis.

## 4 Future Developments

The three-year development under the NASA AIST program is now complete [6], and the ISCE package is currently being distributed to researchers at US institutions that are part of the WInSAR consortium (<http://winsar.unavco.org>). Others can contact JPL and Stanford directly to license the software (<http://software.jpl.nasa.gov>).

It is hoped that the community will embrace the software and choose to improve, extend, and document it. There are a number of extensions that we anticipate will be a priority to users, and will be incorporated shortly. These include extensions and generalizations of time series analysis, as well as integration of the package with other commonly used packages, such as GUI drivers, plotting tools, and even possibly MATLAB-like analysis packages.

The current development team is planning to extend the current package for cloud applications. We have already created virtual machine instantiations of ISCE on single nodes of the Amazon cloud. We expect the software to port to multiple nodes in an embarrassingly parallel fashion, but plan to restructure some of the modules to granularize operations across local and distributed nodes to take advantage of where on the cloud the data reside.

## Acknowledgments

Albert Chen and Cody Wortham, both PhD candidates at Stanford University, and Piyush Shanker, PostDoc at Caltech, formerly PhD student at Stanford, contributed to the Stanford legacy code at the core of ISCE. The authors would like to thank the Earth Science Technology Office at NASA for support under the Advanced Information Systems Technology Program. This work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with NASA, and at Stanford University under a contract with JPL.

## References

- [1] Rosen, P. *et al.* (2009). “InSAR Scientific Computing Environment”. 2009 American Geophysical Union Meeting.
- [2] Gurrola, E., P. Rosen, G. Sacco, W. Szeliga, H. Zebker, M. Simons, D. Sandwell, P. Shanker, C. Wortham, and A. Chen (2010). “InSAR Scientific Computing Environment”. 2010 American Geophysical Union Meeting.
- [3] Rosen, P. A., S. Hensley, and G. Peltzer (2004), Updated Repeat Orbit Interferometry Package released, *Eos Trans. AGU*, 85(5).
- [4] Zebker, H., S. Hensley, P. Shanker, C. Wortham (2010). Geodetically Accurate InSAR Data Processor. *IEEE Trans. On Geoscience and Remote Sensing*, 48(12).
- [5] Rosen, P. , M. Lavalley, X. Pi, S. Buckley, W. Szeliga, H. Zebker, E. Gurrola (2011) Techniques and tools for estimating ionospheric effects in interferometric and polarimetric SAR data, *Proc. Intl. Geosci. Rem. Sens. Society*, DOI: 10.1109/IGARSS.2011.6049352
- [6] <http://www.techbriefs.com/component/content/article/10935>