CSPP Geo Direct Broadcast Software for GOES-16 and Himawari-8: Project Overview and Lessons Learned

Graeme Martin, Liam Gumley, Nick Bearson, Jessica Braun, Geoff Cureton, Alan De Smet,
Ray Garcia, Tommy Jasmin, Scott Mindock, Eva Schiffer, Kathy Strabala
Space Science and Engineering Center, University of Wisconsin - Madison

ITSC-21
Darmstadt, Germany
29 November - 5 December 2017

CSPP Geo Overview: Geostationary Products Software for Direct Broadcast

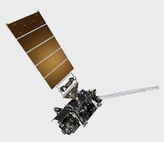
The CSPP Geo project is funded by the NOAA GOES-R program to create software allowing users to process data received directly from geostationary satellites.

All software is:

- ☑ Publicly available and free of charge
- ☑ Distributed as binary packages for 64-bit CentOS6-compatible Linux
- ☑ Distributed with all required 3rd party software bundled
- ☑ Easy to install and run
- ☑ Released with an optional test data package

Software downloads: http://cimss.ssec.wisc.edu/csppgeo/
User support: csppgeo.issues@ssec.wisc.edu

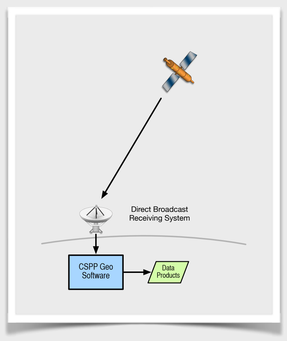
GOES-16 and GOES Rebroadcast (GRB)



The GOES-R satellite was launched in November 2016, and was renamed to GOES-16 when it reached geostationary orbit. It is the first of a new generation of U.S. weather satellites with greatly improved observational capabilities, carrying the Advanced Baseline Imager (ABI), the Geostationary Lightning Mapper (GLM) and multiple space weather instruments. ABI has greatly improved spatial, spectral and temporal resolution compared to the previous generation of imagers, and GLM is the first ever operational lightning mapper to be flown in geostationary orbit.

Direct Broadcast users who have a line of sight to the satellite and the appropriate hardware can receive data from all GOES-R instruments via the GRB stream, and can generate products using CSPP Geo software. GRB has the lowest latency of the GOES-16 distribution methods.

Parallel processing is required to keep up with the higher data rate from the new generation of instruments, and to reduce overall product latency. Hardware requirements for each software package are available on the CSPP Geo website.



*GOES 13/15 vs GOES-16 direct broadcast data volumes**

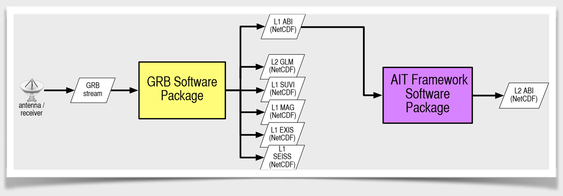
	GOES 13/15 (GVAR)	GOES-16 (GRB)
Data rate	2.1 Mbps	31 Mbps
Raw data volume	21.5 GB/day	310 GB/day
Raw data volume (idle / null data removed)	14.24 GB/ day	99 GB/day (Mode 3) 150 GB/day (Mode 4)

*ABI Level 1B product volume (NetCDF4)**

uncompressed	550GB/day (Mode 3)
compressed	135-140 GB/day (Mode 3) 215 GB/day (Mode 4)

**based on volumes observed at the SSEC GRB receiving station during PLT*

GOES-16 Processing Software



The GRB Software Package allows users to process the raw GRB stream with the CSPP Geo GRB software, reconstructing the products from all instruments as they were created on the ground system. Output is in NetCDF4 format.

During the Post-Launch Test (PLT) period, data from different instruments was added successively to the GRB stream, with the last instrument (GLM) added in June 2016. During this time, interim "dev snapshot" versions of the GRB package were released, incorporating the latest software improvements. Users are currently generating products in real-time with the latest version, 0.4.6.

Version 1.0 of the GRB package is planned for end of 2017, and will include support for UDP multicast, improved logging, improved data validation and error handling, additional configuration options, support for product compression, updated documentation and various other improvements and bug fixes.

Initial set of AIT Framework products

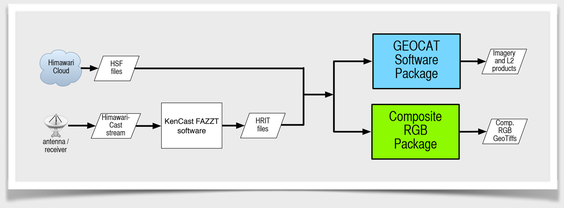
Aerosol Detection: Smoke and Dust
Aerosol Optical Depth
Clear Sky Masks
Cloud and Moisture Imagery
Cloud Optical Depth (day/night)
Cloud Particle Size Distribution (day/ night)
Cloud Top Height
Cloud Top Phase
Cloud Top Pressure
Cloud Top Temperature
Land Surface Temperature (Skin)

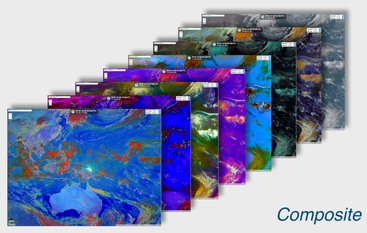
The AIT Framework Level 2 Software Package for ABI allows users to further process ABI Level 1B data to generate Level 2 geophysical products. The core software was developed by the Algorithm Integration Team at NOAA as an integration point for algorithms that were developed by the GOES-R science teams.

The initial version of the AIT Framework Package will include research implementations of the GOES-R baseline algorithms, and will generate a subset of the baseline products. An alpha version of the software was released in June 2017, and a beta version will be released in late 2017. Users who are interested in testing early software versions should contact csppgeo.issues@ssec.wisc.edu.

Later releases will include algorithm updates and additional products.

Himawari-8 Processing Software





Composite RGB images

The CSPP Geo project has developed software to process data from the Advanced Himawari Imager (AHI), which is on the Japanese Himawari-8 satellite. The software is currently at beta status. To obtain early versions of Himawari-8 software, contact csppgeo.issues@ssec.wisc.edu.

The GEOCAT Level 2 Software Package for AHI generates Level 2 products using algorithms that were developed for GOES-R. Both the HimawariCloud and HimawariCast input data formats are supported.

The Composite RGB Package for AHI generates RGB images emphasizing different aspects of the atmosphere and surface. The software uses formulas that were developed by Eumetsat and adapted by JMA for AHI.

Initial set of GEOCAT products

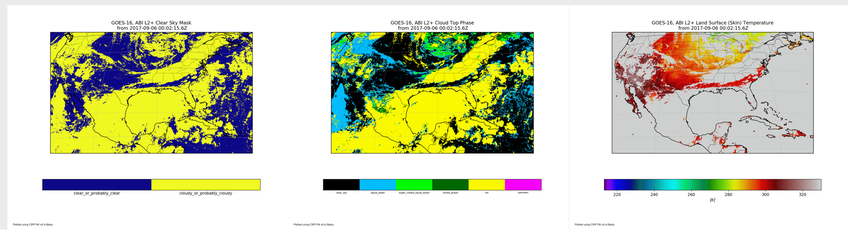
Clear Sky Masks
Cloud and Moisture Imagery
Cloud Optical Depth
Cloud Particle Size Distribution
Cloud Top Height
Cloud Top Phase
Cloud Top Pressure
Cloud Top Temperature
Low Cloud and Fog (*current GOES only)

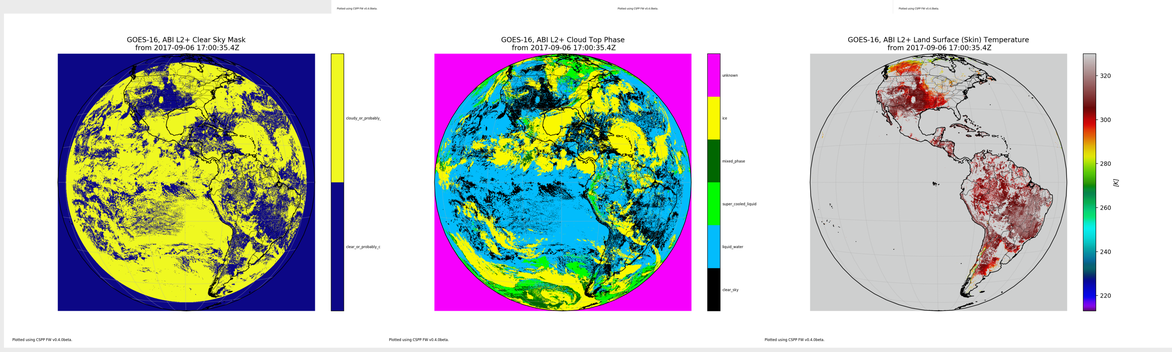
Quicklook Images

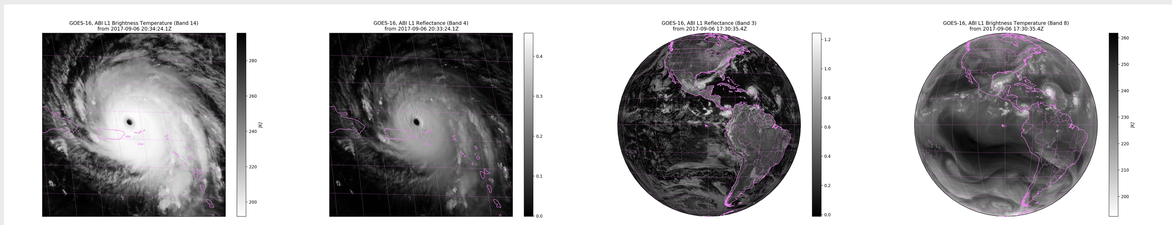
CSPP Geo software packages include the optional capability to create quicklook images.

GOES-16 Preliminary, Non-Operational Data

Cloud and LST product quicklooks from the AIT Framework package (top and middle), and reflectance and brightness temperature quicklooks from the GRB package (bottom).



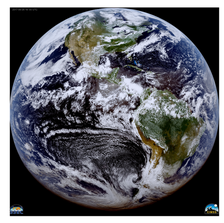




Collaborators

NOAA / AIT: Steve Goodman, Satya Kalluri, Walter Wolf, Shanna Sampson and AIT team, Mike Pavolonis, Andy Heidinger, Tim Schmit

UW-Madison SSEC: Kaba Bah, Denis Botambekov, Corey Calvert, Dan Forrest, Jordan Gerth, Mat Gunshor, Pat Heck, David Hoese, Jarno Mielikainen, Jerry Robaidek, William Straka, Andi Walther, Steve Wanzong



Lessons Learned

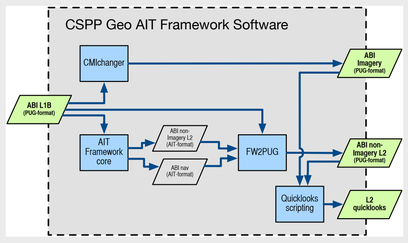
1. Python is fine for real-time processing

Python has long been used in a "glue" role in direct broadcast software developed at SSEC. More recently we have started developing core components in Python. We are well aware of the advantages of Python (rapid software development, availability of a broad spectrum of community-supported software modules for scientific programming), but initially there were questions about the suitability of Python for core processing in a real-time or operational-type environment. In practice, we have found robustness to be equivalent to that of a compiled language, and performance to be adequate as long as best practices are followed. Once core functionality has been demonstrated, optimizations can be made if needed by selectively re-writing computationally intensive routines in a compiled language like C.

2. Re-use existing software where possible

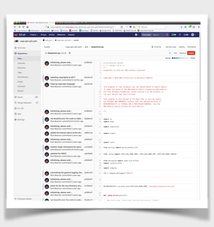
The CSPP team has had good success in building packages encapsulating pre-existing software, often with few to no modifications for direct broadcast use. This strategy is particularly effective for Level 2 products, because it keeps science domain experts in charge of algorithms, and allows the CSPP team to focus on glue code development and computer science considerations. It also reduces costs compared to waterfall-style development, by avoiding software re-writes and simplifying verification and debugging. For this to be an option, the third-party software must be sufficiently mature (i.e. fully validated), reliable and robust. Any necessary improvements and bug fixes to the core algorithm code can be passed upstream to the original developers.

The AIT Framework package is composed of modules written in Python and compiled languages, tied together with Python "glue"



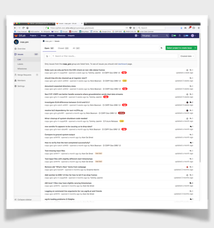
3. Put some thought into software versioning

Though it may seem minor, having an explicit software versioning scheme that is well thought-out can help to avoid a lot of confusion among a development team and among users. The versioning scheme should specify the meanings of each component of a version number and the rules for when they are incremented, for example based on interface changes, added functionality and bug fixes. Software version numbers should appear consistently throughout software, documentation and products. We have found it to be useful to have the freedom to increment the patch number (lowest order component) freely as new packages are created for internal testing purposes, so there is no ambiguity about what version of the software is being run at any time.



4. Use version control software, heavily

Version control systems have become a critical tool for any software development project. The provide an audit trail of what changes were made to code, when they were made and by whom. They allow recovery of old versions of code, encourage experimentation and are particularly useful in collaborative development situations. Modern version control systems like git are practically idiot-proof. Practically. The CSPP Geo team uses gitlab, which provides a browser-based interface on top of git that allows branching and merging, viewing of source files from different branches and commits, file comparisons, issue tracking, continuous integration automation hooks, and wiki functionality.



5. Issue tracking + ticket reviews + code freezes

Modern issue tracking systems allow developers and support staff to track issues reported by customers, as well as coding and testing tasks associated with individual software releases. Tasks can be assigned to team members, given priorities and associated with milestones such as future software releases. Team member can see what tasks remain and who they are assigned to, improving the overall level of coordination. The CSPP Geo team typically reviews and prioritizes tickets and sets a "code freeze" date at the beginning of a development cycle.

6. Allocate time and resources for testing and documentation

A significant amount of time and effort are needed to ensure that software is working properly before release, and to create user documentation. The testing should include all command line and configuration options, new functionality and bug fixes, and automated regression testing to ensure old problems have not reappeared. Thorough pre-release testing often flushes bugs that are not apparent to the developer, avoiding the need for later bug-fix software releases and ultimately benefitting users.

7. Make sure you own at least one machine that matches each hardware specification

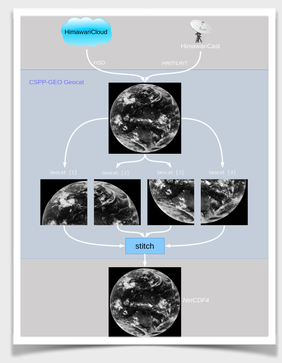
CSPP Geo software is tested on a variety of machines before release, including a reference machine that exactly matches the published hardware specification for that software package. Testing on a reference machine gives confidence that a software build performs as expected, and will also do so on a user's machine. Having a dedicated "model" test machine under our control allows us to test new builds as they are created, and to run uninterrupted and exclusively for long periods of time to evaluate stability. We also make use of a number of "burn-in" testers and experimental/build facility machines.

8. Use collaboration tools

The CSPP Geo team relies heavily on Slack, which is basically a freemium group chat application with some enhancements tailored to software developers. Slack users can define channels dedicated to specific topics, and can also send direct messages. Team members can passively monitor channels of interest, and will receive notifications when their handles or topics of interest are mentioned. Information of general interest to the team is often posted to Slack. Logs are searchable, so it works as an information repository.

9. Parallel processing is required to process direct broadcast streams from geostationary satellites

The data rate from the new generation of geostationary weather satellites (Himawari-8, GOES-16) is high enough that parallel processing is required to generate products with the current generation of CPUs. Furthermore, overall product latency can be reduced by processing a single image on multiple cores. Additional considerations include dependencies on products from previous timesteps and inter-algorithm dependencies. Some care and effort must be put into designing and implementing a parallel processing scheme.



10. Wherever possible, distribute pre-built binaries with bundled third party software

CSPP Leo and Geo software is distributed as binary tarballs, with all required third-party software pre-built and bundled. In practice we have found this method to be very effective. The advantages of this method are easy installation (untar and go), and elimination of a class of user problems including build issues and run-time issues due to different library and compiler versions. The disadvantages include limits to the number of platforms that can be supported, and that some effort must be put into pre-building portable libraries while ensuring proper internal linkage and external dependency minimization. However, since software dependencies tend to be common across packages (e.g. Python runtime, NetCDF4, HDF5), we can build libraries and language runtimes once and share across packages, with occasional updates for new library versions.

11. Be prepared to release rapid updates

The GOES-16 direct broadcast stream (GRB) was turned on in December 2016 and data from each instrument was added over a period of about 6 months. During that period, data was considered to be provisional quality and the priority was to get software updates to users as soon as possible to allow them to generate products. To do this, we bypassed our usual release process in order to get rapid "interim" software versions to users, essentially developer snapshots of the current codebase which had been minimally tested. We found this to be an effective strategy, aided by automated testing infrastructure and effective use of issue tracking and version control.

Graeme Martin: graemem@ssec.wisc.edu