McIDAS-X & Python

The Powers that Drive the COD-NEXLAB Satellite Imagery

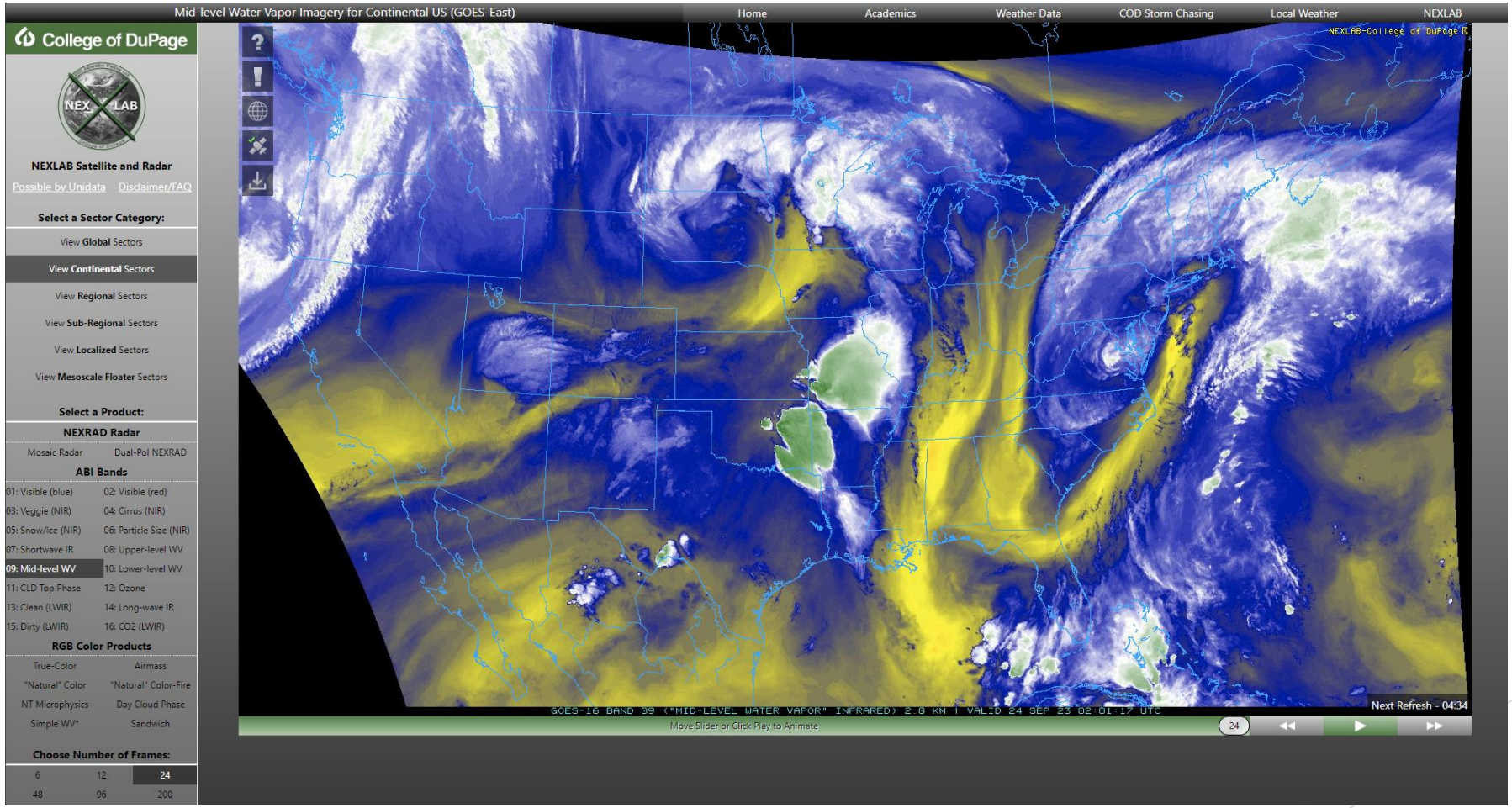
McIDAS Users' Group 2023

College of DuPage

Mike Zuranski – Unidata

09-26-2023

College of DuPage - NEXLAB



ABI Bands, RGB Products, GLM, Derived Products, Data & Mapping Overlays

College of DuPage – NEXLAB
Meteorology Faculty and Staff

Paul Sirvatka

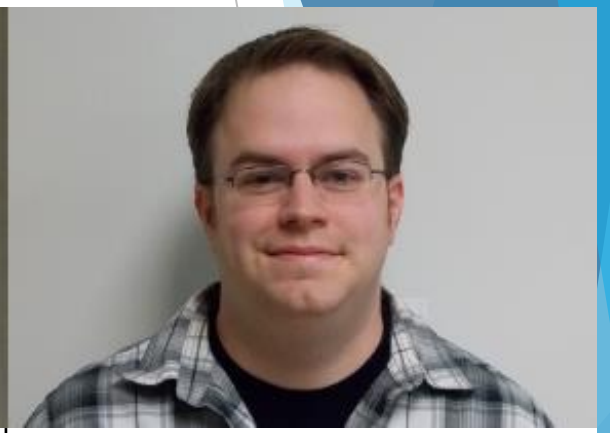
Professor of Meteorology

Ron Stenz

Associate Professor of Meteorology

Gilbert Sebenste

Support Analyst & Product Developer

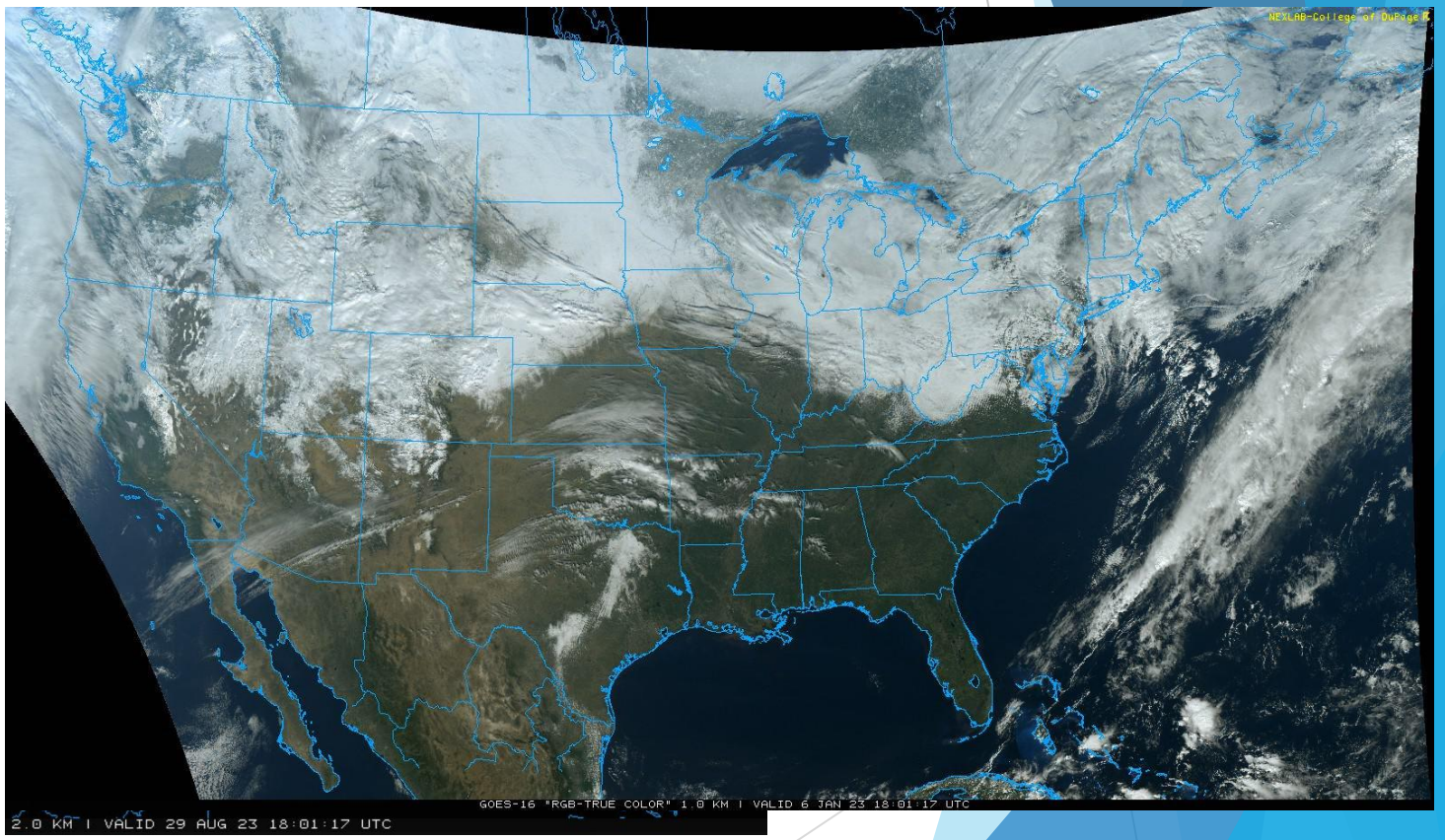


Evan Anderson

Web Developer & Lab Manager

College of DuPage - NEXLAB

- Normal Day:
  - 12-15 Million Hits
  - 20-25 Thousand Unique Visitors
  - 750 GB – 1 TB of Data Out
- Busiest Day (so far) 8-29-2023:
  - 40 Million Hits
  - 57 Thousand Unique Visitors
  - 3 TB of Data Out



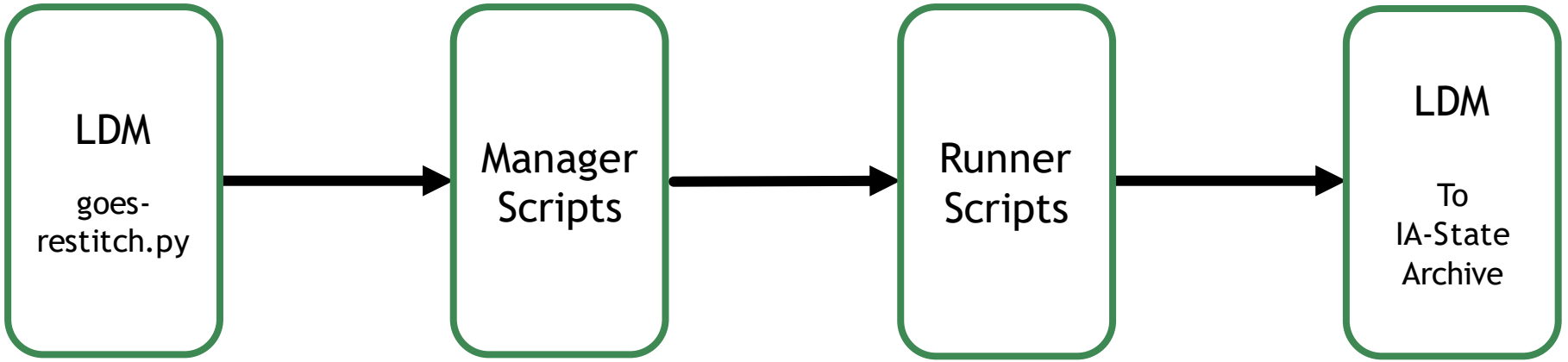
The College of DuPage – NEXLAB Satellite Imagery Processing

A Behind-the-Scenes Look at How COD's Satellite Imagery is Made

Overview

- 16 ABI Bands, 8 RGB products, 10 L2 Derived products, 4 GLM products
- Nearly all this data comes from NOAAPort/SBN
- Exceptions:
  - Gridded GLM (Unidata LDM feed)
  - Select GRB imagery (Unidata ADDE)
- Almost everything is initiated by LDM

Overview
General Processing Workflow



goes-info.py

Centralization
Of information
And functions

map_gen.py

Map overlay
generation

Automation
Local Data Manager (LDM)

- Nearly all data arrives via the LDM
- ABI tiles are stitched together using goes-restitch.py
- ABI and L2 data are saved to disk
  - Their paths are passed to the manager scripts
- GLM data is saved to disk for use later, no actions taken

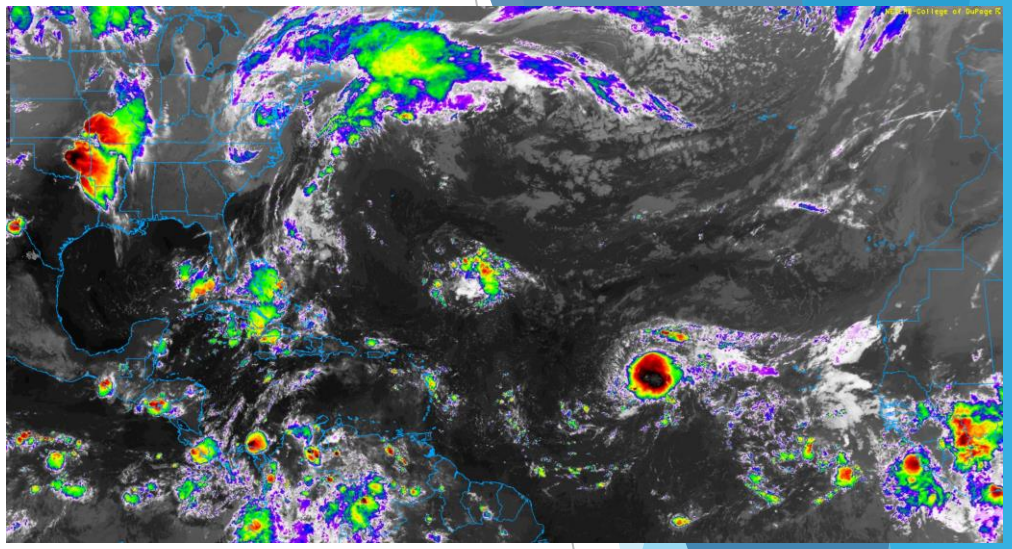
Automation
Manager Scripts

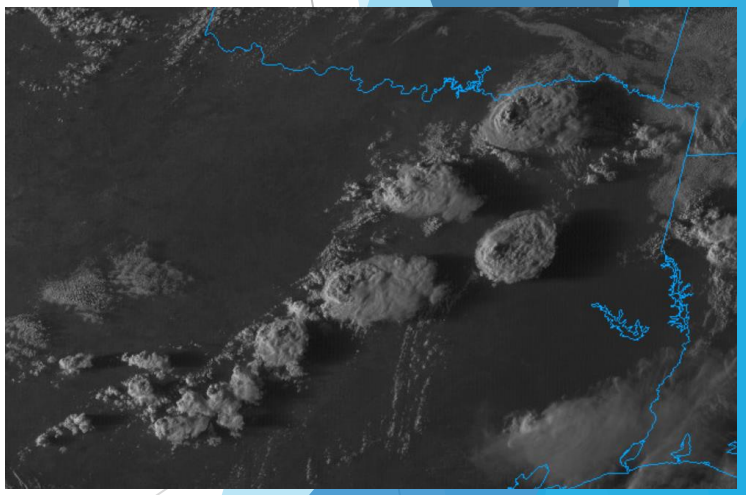
- manager.py, manager_derived.py and manager_remote.py
- These take the file name/path from goes-restitch.py
- Invokes the “runners” and enforces timeouts
- Invocation details and timeouts determined by product and scene
- Any runner script still running at the timeout
  is culled to prevent runaway processing

Automation
Runner Scripts

- sat_runner.py, sat_runner_meso.py, rgb_runner.py, sandwhich_runner.py, derived_runner.py
- These are the product generation scripts
  - This is where McIDAS is invoked
  - Includes handling for scenes, bands, products, projections, etc.
- Utilizes multiprocessing where possible
  - Not generally possible for Full-Disk – many unique projections

Automation
ABI Imagery





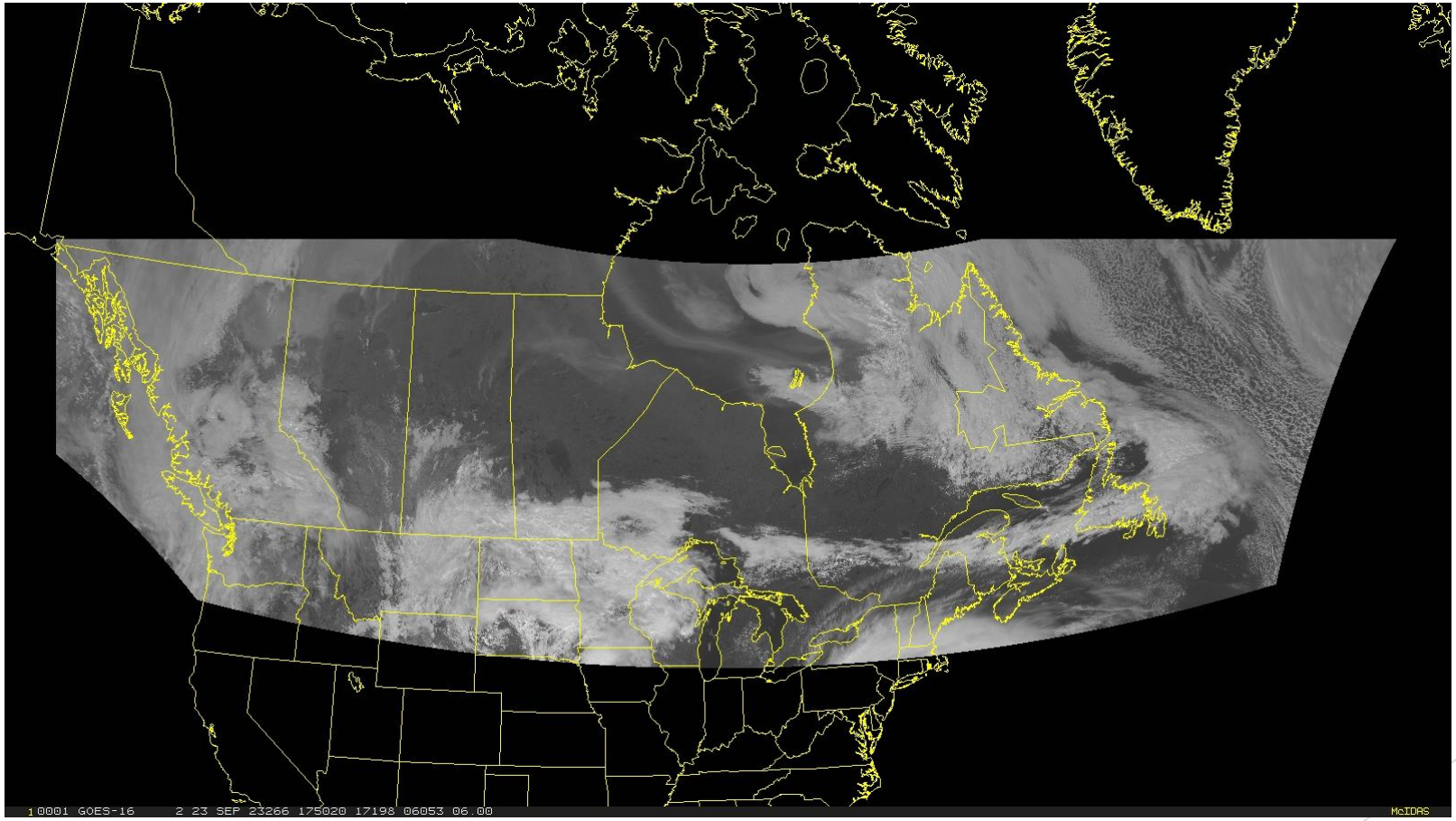
1. Reproject if necessary
2. For Each sector and scale, make the base image
   a) If CONUS or Mesoscale, make GLM imagery
   b) If Mesoscale, handle 30-second imagery and new location mapping
   c) If Full-Disk, make background images
3. Send images to IA-State for archival

CONUS	Mesoscale	Full-Disk	Remote
One projection	No reprojection	Many projections	One projection
GLM data	GLM data	No GLM data*	GLM Data
5-minute Standard	Possible 30-sec Mode	Possible 5-min Mode	Check each min for new data

Automation
ABI Imagery – Remote Cropping

- We needed imagery over Canada, but SBN Full-Disk is too coarse
- Full-Disk GRB data is large!
- Solution: Take a cropping of GRB imagery remotely with IMGREMAP
  - I download only what I need
  - Faster to work with too

Automation
ABI Imagery – Remote Cropping

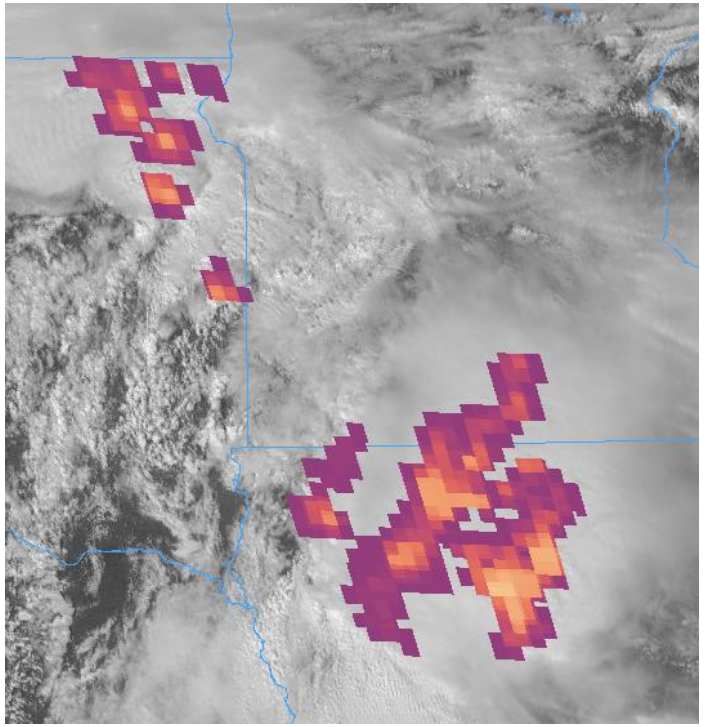


IMGREMAP RTGOESR/FDC02 GOESDATA.9902 PRO=LAMB 2 26 95 SSIZE=2000 10500 SIZE=3600 9400 LAT=50.2 82.5

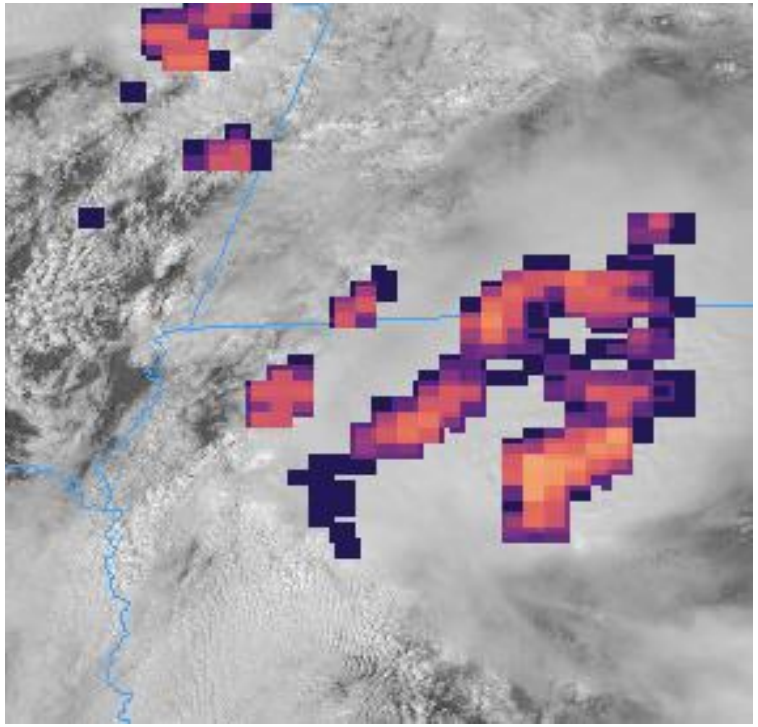
Automation
GLM Products

- All GLM products are made as part of the ABI processing
  - Each GLM product is initiated by an ABI band
- GLM products are overlays – Must make them transparent
- EU Tables:
  - FED: L2-COD
  - TOE and MFA: Derived from AWIPS color tables
    - Different EU tables for reprojected imagery
    - Accuracy not guaranteed
    - Logarithmic scaling required to do make these products the right way

Automation
GLM Products



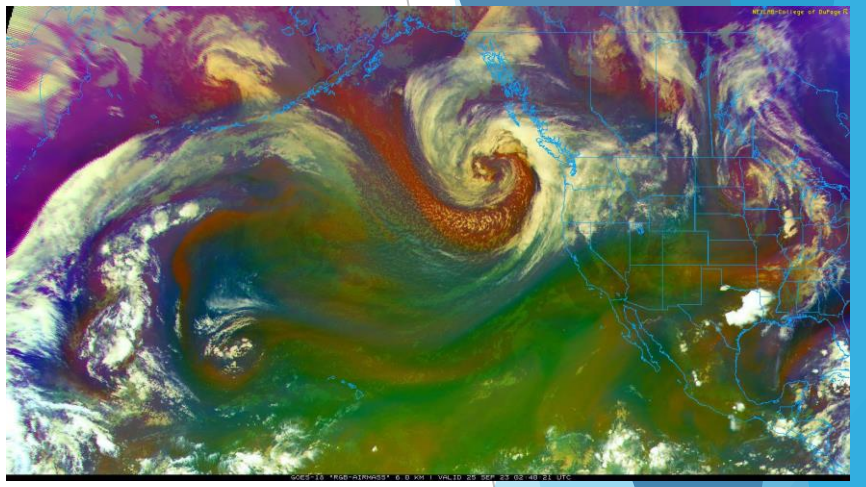
Reprojected CONUS

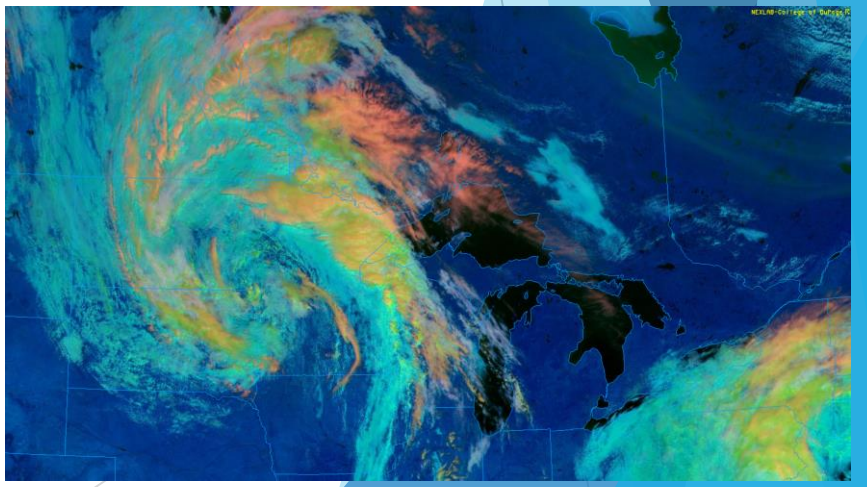


Mesoscale

Automation
RGB Imagery

- Very similar structure and workflow as ABI imagery
- A single band will initiate a product
  - E.g. Band-8 will start making Airmass imagery
  - The rgb_runner will wait until all required bands are available
  - Separate timeout for waiting on the other bands
- Mesoscale handling included in rgb_runner
- RGB recipes are stored in the rgb_runner as part of its code
  - Different recipes for reprojected imagery



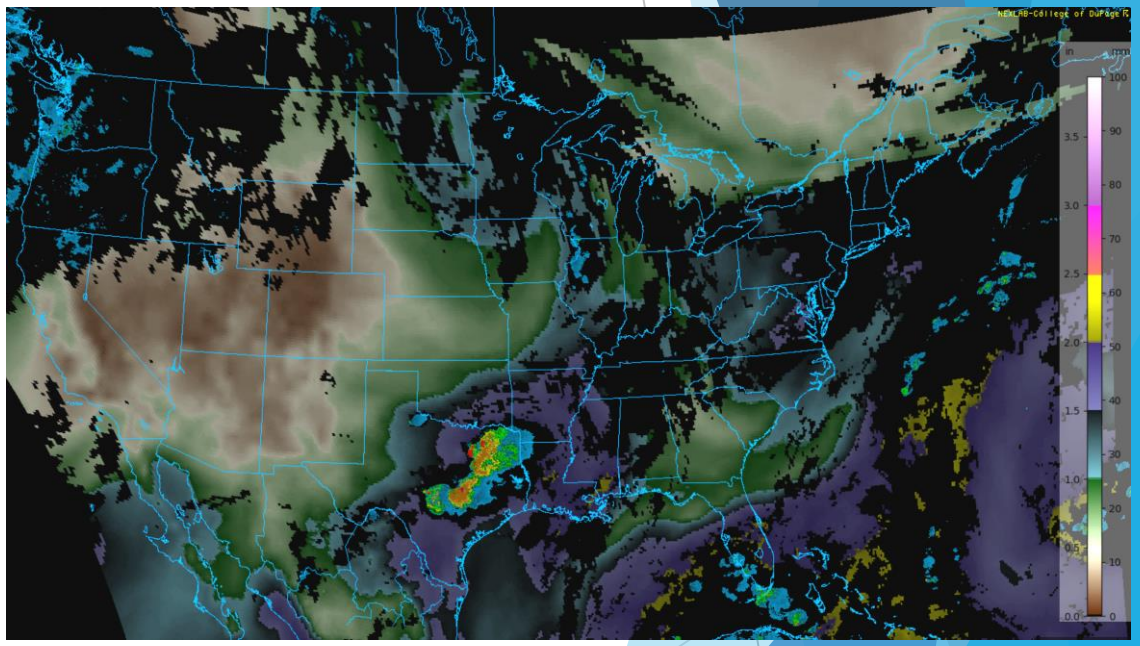


Iowa State Archive
Daryl Herzmann

- Began archiving in 2018
- All ABI and RGB products (except local sectors)
- Images are inserted into LDM after they are made
- Example URL:
  https://mtarchive.geol.iastate.edu/2023/09/26/cod/sat/
- There will be a front-end… Someday™

Automation
Level-2 Derived Products

- Very similar structure and workflow as ABI imagery
- Some additional logic for specific products/domains
  - E.g. SSTs, do we need that in the Midwest?
- These are all overlays – Must make them transparent
- These are not sent to IA-State for archival



Automation Centralization – goes_info.py

- This is home for things referenced in multiple locations
  - Sector information, projections, frame labels
  - Common functions
- Examples of common functions
  - insert_into_ldm()
  - sueu()
  - AREA file mapping logic
  - Non-operational message overlays

```
def sueu(band, meso=False):
    sueu = ''
    # IR Bands:
    if band in ['13', '14', '15']:
        sueu = 'EU=RBTOP'
    # WV Bands:
    if band in ['08', '09', '10']:
        if meso:
            sueu = 'SU=WVCIMSS EU=WVCIMSS'
        else:
            sueu = 'SU=WVNEW EU=WVCIMSS'

    return sueu
```

Automation
Map Generation – map_gen.py

- This is the only script where maps are made
  - This is also where map sets are defined
  - Maps used vary depending on scale and location
- Most maps are pre-made once, mesoscale when moved to a new location
- Maps include roads, lakes & rivers, counties, CWAs, Lat/Lon, ARTCC boundaries
  - CWA and ARTCC were made from GEMPAK maps

Optimizations, Tips and Tricks

I wanna go fast! - Ricky Bobby

Setting the McIDAS Environment in Python

- Set McIDAS environmental variables in the top level of the script…
- They will be used across the board from that point on.

```
import os
mchome = os.path.join("/home", "mcidas")
os.environ["PATH"] = os.pathsep.join([os.path.join(mchome, "bin"), os.environ["PATH"]])
os.environ["MCDATA"] = os.path.join(mchome, "workdata")
os.environ["MCTABLE_READ"] = os.pathsep.join(os.path.join(mchome, "workdata", "MCTABLE.TXT"),
                                             os.path.join(mchome + "data", "ADDESITE.TXT"))
os.environ["MCTABLE_WRITE"] = os.path.join(mchome + "data", "ADDESITE.TXT")
os.environ["MCPATH"] = os.pathsep.join([os.path.join(mchome, "batches"),
                                        os.path.join(mchome, "bin"),
                                        os.path.join(mchome, "data"),
                                        os.path.join(mchome, "help"),
                                        os.path.join(mchome, "workdata")])
```

String Formatting
It's used everywhere!

- Parsing data file path/name so I don't have to crack open the data

```
# This function takes a data filename and parses it for initial product info:
def parse_filename(filepath):
    """
    band = '02'
    sat_id = 'GOES16'
    scene = 'CONUS'
    dateString = '20220301'
    timeString = 225117'
    Expected format is as follows (data subdir + OSPO convention):
    GOES16/CONUS/Channel02/20220301/OR_ABI-L2-CMIPC-M6C02_G16_s20220602251170_e20220602251170_c20220602251170.nc
    """
    prodinfo = {}
    prodinfo["args"] = filepath.replace(DATA_BASEDIR, "")
    prodinfo["filename"] = prodinfo["args"].split("/")[-1]
    prodinfo["filenamefull"] = DATA_BASEDIR + prodinfo["args"]
    prodinfo["band"] = prodinfo["filename"].split("_")[1].split("C")[-1]
    prodinfo["sat_id"] = prodinfo["args"].split("/")[0]
    prodinfo["scene"] = prodinfo["args"].split("/")[1]
    prodinfo["dateString"] = prodinfo["args"].split("/")[3]
    prodinfo["timeString"] = prodinfo["filename"].split("_")[3][8:14]
```

String Formatting
It's used everywhere!

- Poll the remote ADDE server to check for new data

```
mcidas@mun: ~
mcidas@mun:~$ imglist.k RTGOESR/FDC02
Image file directory listing for:RTGOESR/FDC02
 Pos Satellite/          Date       Time      Center    Band(s)
     sensor                                  Lat  Lon
 --- ------------  ------------  --------  ---- ---- ------------
 192  GOES-16       24 SEP 23267  00:00:20     0   75 2
imglist.k: done
mcidas@mun:~$
```

```python
# Parse IMGLIST output to determine the valid time of remote data:
result = check_output(['imglist.k', 'RTGOESR/FDC{}'.format(band)], stderr=STDOUT, timeout=30).decode()
dateTimeStr = ' '.join(result.split()[23:27])
# After some minor contorting, date string looks like this: '19 FEB 18050 19:45:39'
# Note that is the Julian date!
pattern = '%d %b %y%j %H:%M:%S'
valit_timestamp = int(time.mktime(time.strptime(dateTimeStr, pattern)))
```

String Formatting
It’s used everywhere!

- Dynamically create McIDAS commands

```
elif product == 'natcolor':
    areas = areas_from_scale(['1004', '1005', '1006'], dataSource)
    areas = areas_from_satellite(areas, satellite, 'RGB')
    mcRemap.imgoper("{prefix}01 GOESDATA.{a3} SIZE=ALL SCALE=30 230 0 255".format(prefix=dsetPrefix, a3=areas[2]))

    if reprojection:
        mcRemap.imgremap("{prefix}05 GOESDATA.{a1} {proj}".format(prefix=dsetPrefix, a1=areas[0], proj=proj))
        mcRemap.imgremap("{prefix}03 GOESDATA.{a2} {proj}".format(prefix=dsetPrefix, a2=areas[1], proj=proj))
        mcRemap.imgremap("GOESDATA.{a3} GOESDATA.{a3} {proj}".format(a3=areas[2], proj=proj))
        dsets = ["GOESDATA.{a1}".format(a1=areas[0]),
                 "GOESDATA.{a2}".format(a2=areas[1]),
                 "GOESDATA.{a3}".format(a3=areas[2])]
    else:
        dsets = ["{prefix}05".format(prefix=dsetPrefix),
                 "{prefix}03".format(prefix=dsetPrefix),
                 "GOESDATA.{a3}".format(a3=areas[2])]
```

Put key directories in a RAMDisk

- Python's tempfile module uses /tmp
  - This default can be changed
- Goes-restitch.py uses tempfile
- McIDAS reads/writes AREA files in ~/workdata
  - This too can be changed
- Putting both of these into RAMDisk *significantly* reduces I/O overhead
- All these things can be lost, and that's okay

Transparent Images with Python Using Pillow (PIL)

```python
def make_transparent(imageFileTemp, imageFileFinal):
    from PIL import Image
    img = Image.open(imageFileTemp)
    img = img.convert("RGBA")
    datas = img.getdata()
    newData = []
    for item in datas:
        if item[0] == 0 and item[1] == 0 and item[2] == 0:
            newData.append((0,0,0,0))
        else:
            newData.append(item)
    img.putdata(newData)
    img.save(imageFileFinal, "PNG")
```

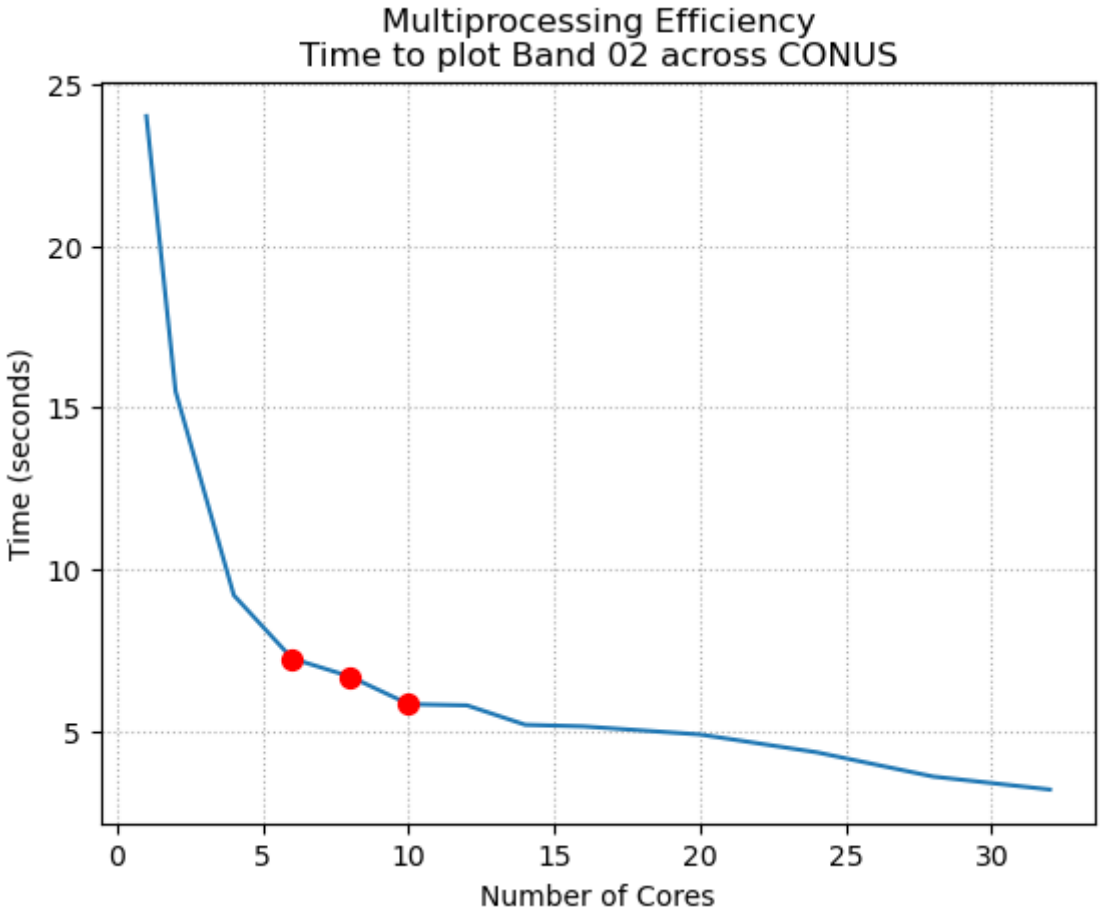
Image File Sizes

- Base imagery saved with QUA = 90
- Background imagery with QUA = 70
- Archive imagery with QUA = 85 (Pillow)
- Thumbnails for mesoscale selection
- These gave about a 40% reduction in file sizes
- Web cache settings were also critical

Multiprocessing

```
import multiprocessing
numProcs = 10  # 6, 8 or 10 appear to be the most efficient.  See proctimes.txt for metrics.
for scaleTuple in sector_scales():
    scale, mag = scaleTuple
    secTuples = list(conus_sectors(scale).items())
    pool = multiprocessing.Pool(processes=numProcs)
    maps = pool.map_async(process_conus_sectors, secTuples)
    try:
        results = maps.get(timeout=120)
    except TimeoutError:
        logger.error('Timeout reached processing conus sectors, closing pool...')
        pool.close()
    except Exception as e:
        logger.error('Error in one of the CONUS children procs:')
        logger.error(e)
```

Multiprocessing



Number of Cores	Time (seconds)
1	24
2	15.5
4	9.2
6	7.2
8	6.7
10	5.83
12	5.8
14	5.2
16	5.15
20	4.9
24	4.35
28	3.6
32	3.2

Strategic Use of AREA Files

```
.0xxx   ABI Bands - GOES-16
.1xxx   RGB Products - GOES-16
.2xxx   L2 Derived Products - GOES-16
.3xxx   ABI Bands - GOES-17
.4xxx   RGB Products - GOES-17
.5xxx   L2 Derived Products - GOES-17
.7xxx   RESERVED - Files made/used for other projects
.8xxx   Legacy products (old 800x600 format and such)
.9xxx   Dev products (test/volatile data, not for operational usage)

.x0xx   CONUS
.x1xx   MESO1
.x2xx   MESO2
.x3xx   FullDisk
.x4xx   PRREGI
.x5xx   RTGOESR_Canada
.x6xx   MESO3
.x7xx   MESO4
.x8xx   HIREGI
.x9xx   AKREGI
```

Now you know why I have those helper functions!

```
ABI Bands:
.xx01 - .xx16
.xx20   Reserved for map overlay generation

RGB Products:
.xx01 - .xx03   Airmass
.xx04 - .xx06   Natural Color
.xx07 - .xx12   True Color
.xx13 - .xx15   Nighttime Micro Physics
.xx16 - .xx18   Simple Water Vapor
.xx19 - .xx23   Day Cloud Phase
.xx24 - .xx26   Natural Color - Fire
.xx27 - .xx29   Sandwich
.xx31 - .xx47   True Color (Advanced)

L2 Derived Products:
.xx01           TPW  - Total Precipitable Water
.xx02           DSI  - CAPE
.xx03           ACHA - Cloud Top Height
.xx04           SST
.xx05           LST
.xx06           ADP  - Smoke
.xx07           ADP  - Dust
.xx08           ACTP
.xx09           RRQPE
.xx10           ACHT
.xx11           GLM_FED
.xx12           GLM_TOE
.xx12           GLM_MFA
```

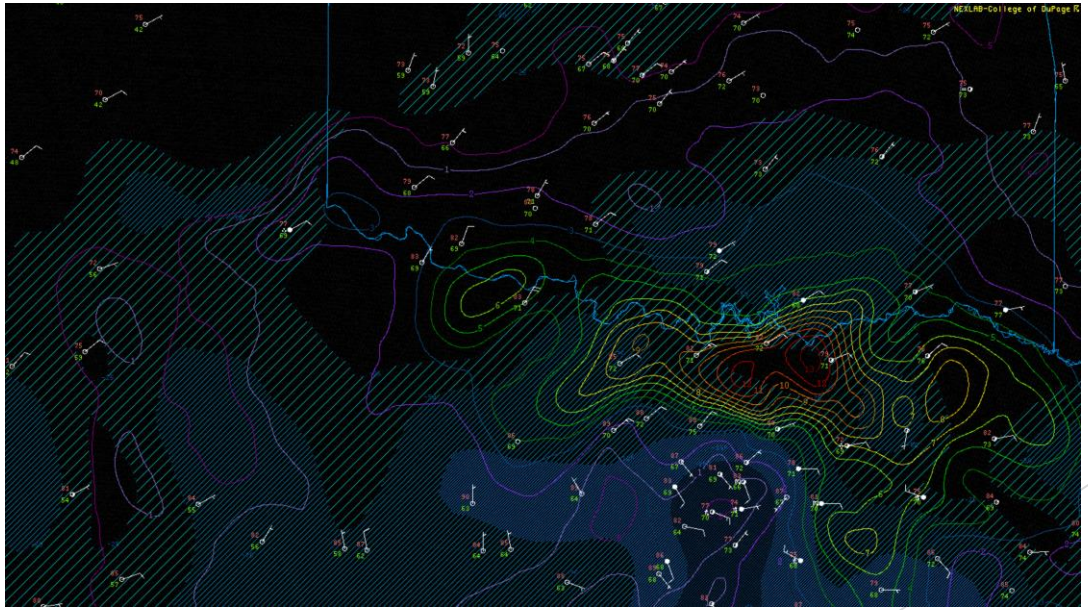
Matching Domains between McIDAS and GEMPAK

- Convert center point & mag into lower-left and upper-right

```
REM %1  Dataset to plot
REM %2  Lat
REM %3  Lon
REM %4  Mag
REM %5  Name of Sector

IMGDISP %1 1 LAT=%2 %3 MAG=%4

REM ECHO %5
ECHO %SECNAME%
PC T 900 1
E
PC T 1 1600
E
```



Some Observations, Hindsights And Final Thoughts

McIDAS Python Wrapper

- Works great, near-zero issues!
- It made all that automation possible
- Greatly simplifies the automation
- Easily capture STDOUT from McIDAS invocations
  - Which can then be acted upon

McIDAS Python Wrapper
Some Ideas

- Return the imagery made as an object
- Return data values, coordinates and other information
- Return metadata (e.g. IMGLIST output)
- Returns as Xarray or Pandas dataframes
- Jupyter Notebook integration
- Tools for working with color bars, EU and SU tables

RGB Recipes Are a Pain
And I did it to myself

- Adding or modifying these recipes is cumbersome
- Extracting the actual McIDAS commands is *very* cumbersome
- What I would consider doing differently:
  - Use .BAT files to store recipes
  - Script option to print recipes in McIDAS format

McIDAS Map Files

- Somewhat common feedback: *"Your maps are out of date."*
- Plenty of GIS data that could be useful as maps
  - Forecast, Fire, Marine zones
  - Canadian Forecast Regions
  - International boundaries
- I'm making a shapefile -> map file utility
  - Automatically updates to latest AWIPS Basemap and TIGER road shapefiles
  - Convert your own
  - Reduction with Mapshaper

Final Thoughts

- McIDAS is the only way to batch process as much imagery as COD does
- Python made the automation approachable
- While I could have used .BAT files, the wrapper made things much easier
- Additional Python integrations would make McIDAS much more accessible
  - And more in-line with current teachings

Thank You!

- SSEC, for McIDAS and the ongoing partnership with Unidata
- Unidata, for letting COD be a guinea pig & beta tester
- COD, for letting me brag about the operation I'm no longer a part of
- Daryl Herzmann, for archiving an insane amount of imagery (and many other things)

Questions?