PRINCETON PLASMA PHYSICS LABORATORY

High Performance Visualization of *f(x,y,t)* Data

Zachary Kaplan, Michael Knyszek, Matthew Lotocki, Eliot Feibush

This work is supported by the US DOE Contract No.DE-AC02-09CH11466

Motivation

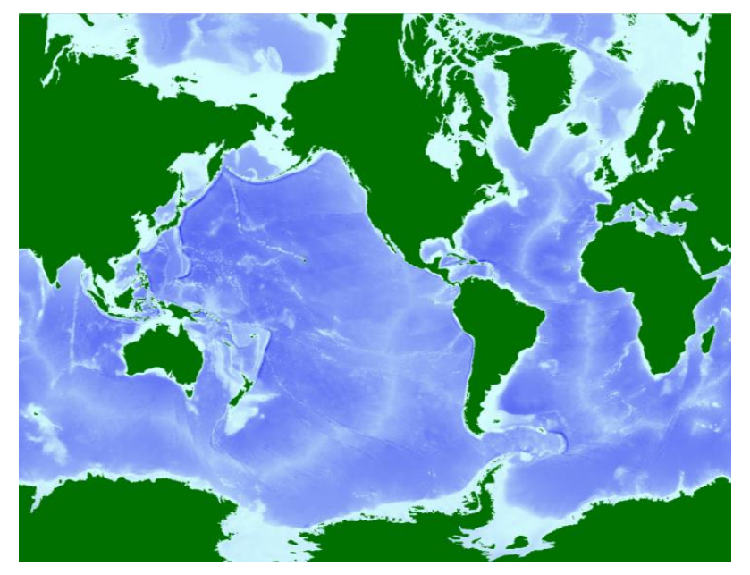
- GFDL runs large climate simulations and stores time-based and grid-based data in large **NetCDF files (>70 GB)**
- Current solutions:
  - **ncview, VisIt**
  - Maps file into virtual memory; **swapping is slow**!
- Need streamlined visualization workflow
  - Quicken analysis, increase productivity
- Write it in **Python** for maximum portability

Challenges

- Need to maximize **portability**
  - Which also means minimize dependencies and sysadmin installation
- The standard visualization tools that support NetCDF files in Python are just as bad as ncview and VisIt
  - scipy is available but has the same issues
- Want to create both images and movies
- Want to support layering of datasets with transparency
- Want the generation process to be **fast**

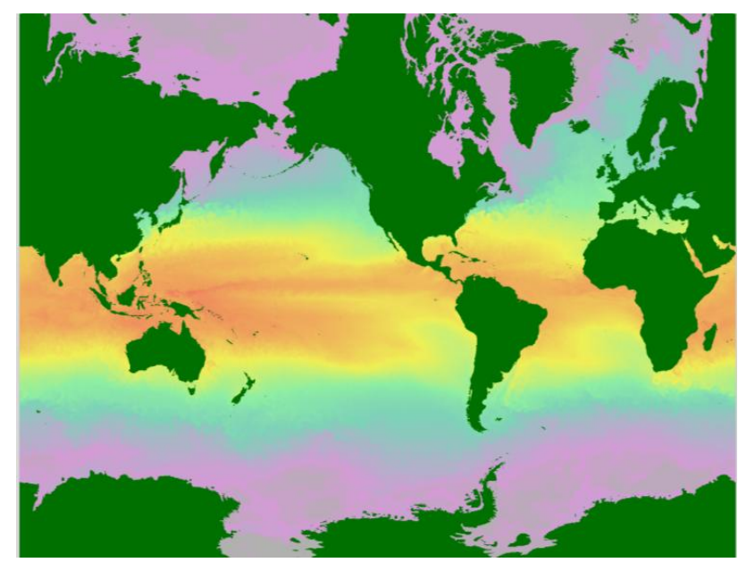
Solutions

- Use the **Anaconda** environment
  - Everyone builds a personal, local environment
  - Manages Python packages for you and supports most
- For NetCDF reading, use **netcdf4-python**
  - Needs on libnetcdf, but Anaconda handles it
  - Does not map whole file
    - Small memory footprint, fast reads of data
- Generation process
  1. Apply colormap to slice of dataset (creates layer)
  2. Run alpha compositing algorithm to merge layers
  3. Use **Python Imaging Library** to create images
  4. Use **ffmpeg** to create a movie from the images!



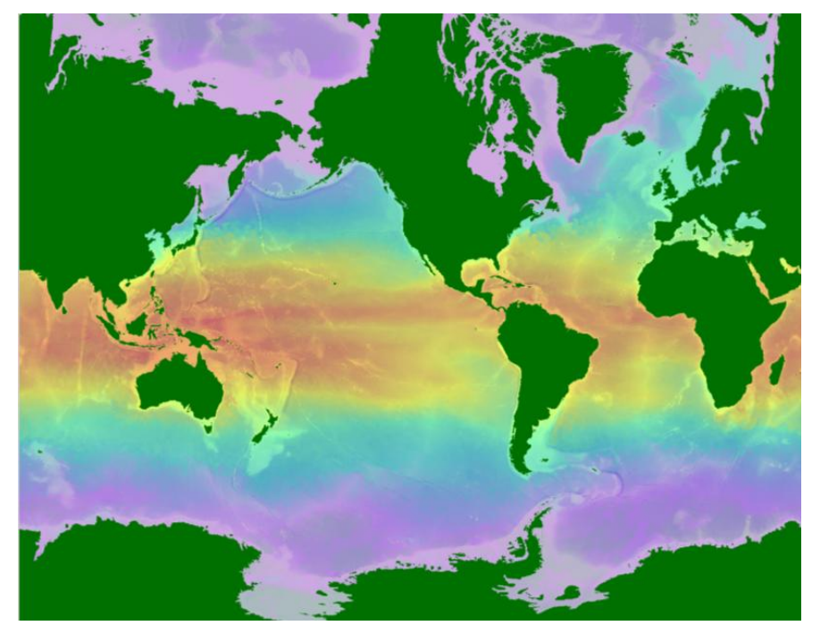
Bathymetry

+



Sea Surface Temp.

=



Composited Image

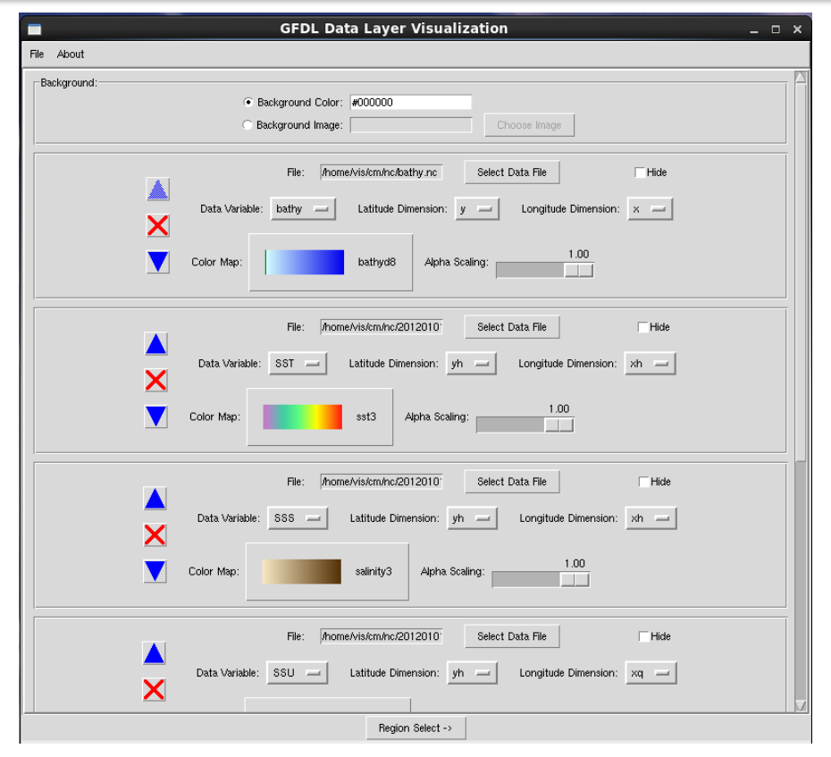
But how did we make it 15000x faster?

- Use the GPU -> **PyCUDA**
- Take advantage of cores -> Python Processes
- Use **numpy** vector operations -> Major speedup!
- For portability, provide all three options for users

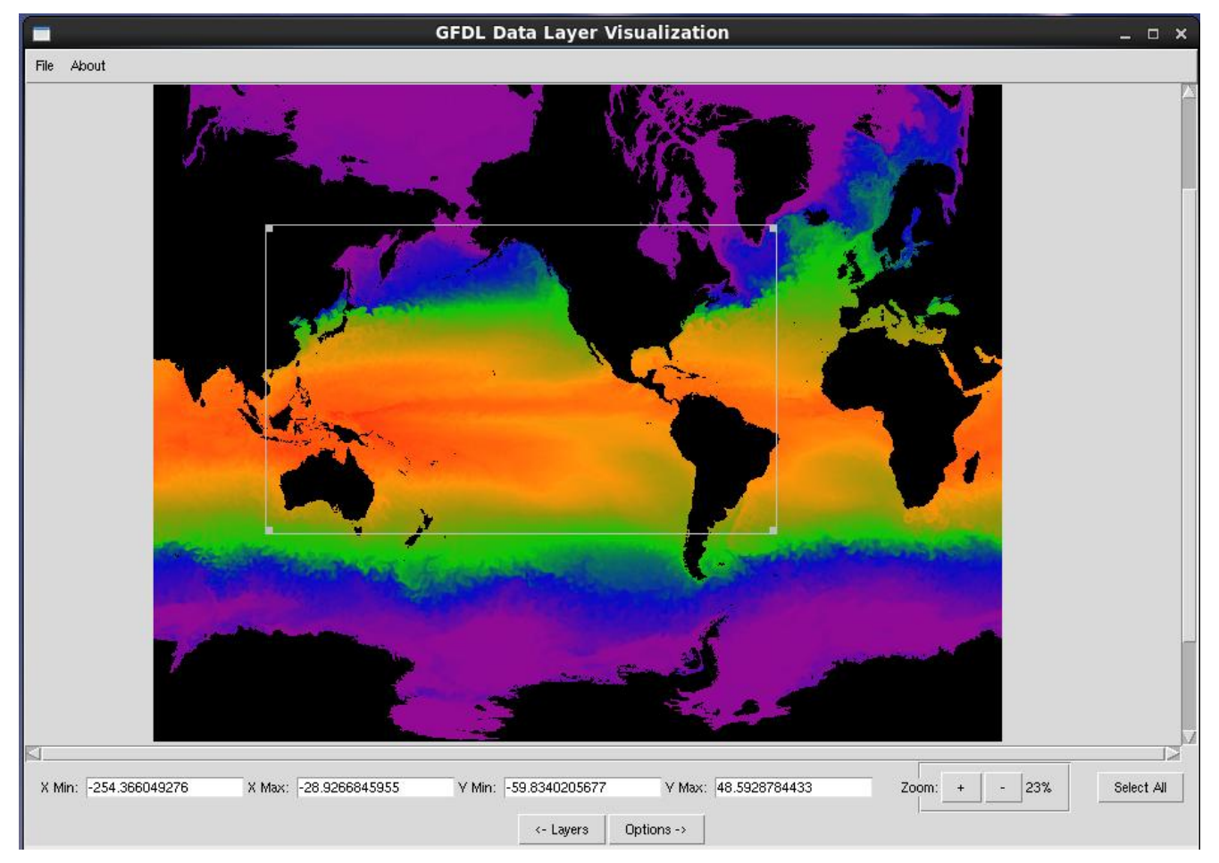
Processing speedup from 2¼ **minutes per image to 10 ms!**

User Interface & Workflow

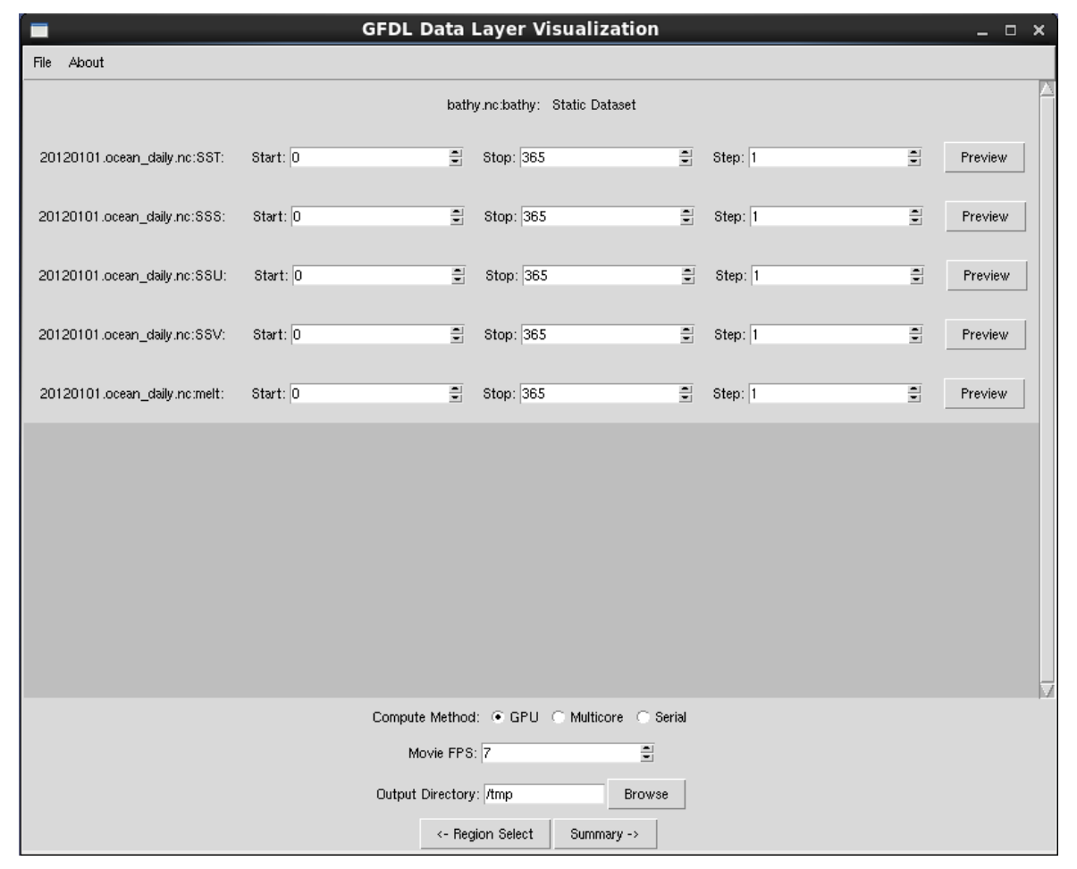
1. Set up layers
   - Load data files, select variables
   - Choose/Create Colormaps
     - Custom colormap backend
     - Supports VisIt and Paraview formats



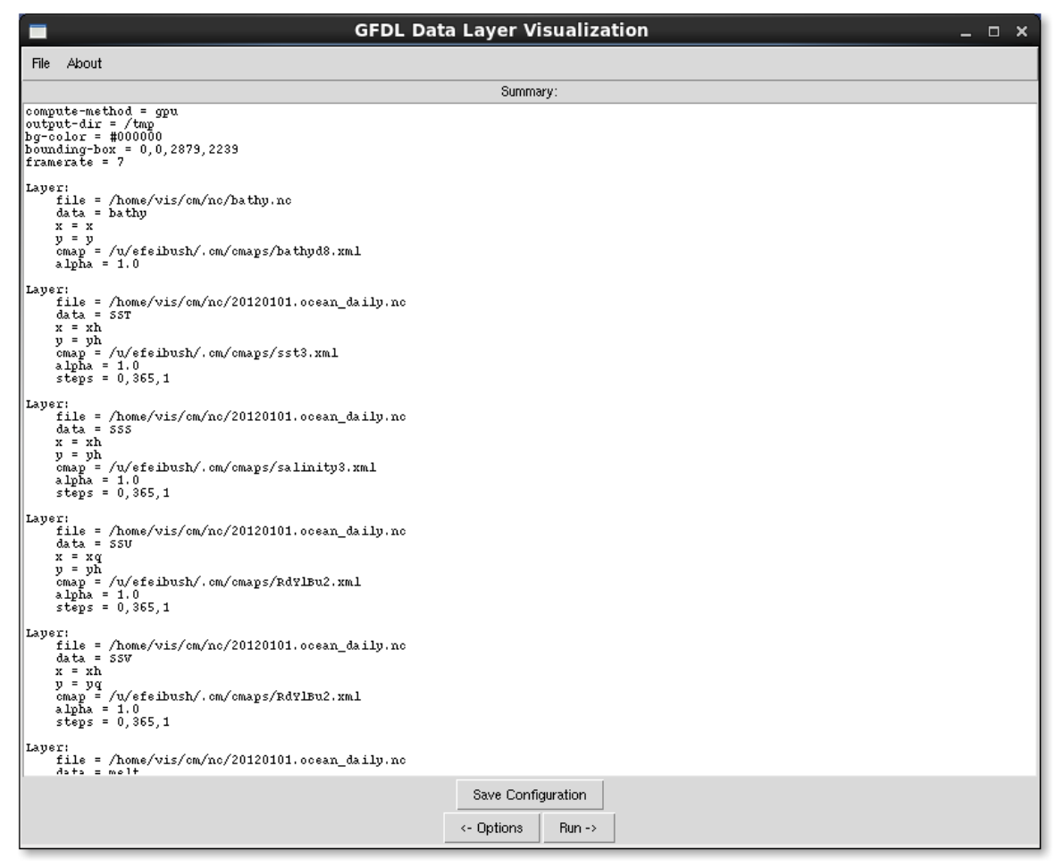
2. Choose your region
   - Interactive crop selection of region to visualize
   - Can index in original units (latitude, etc.)



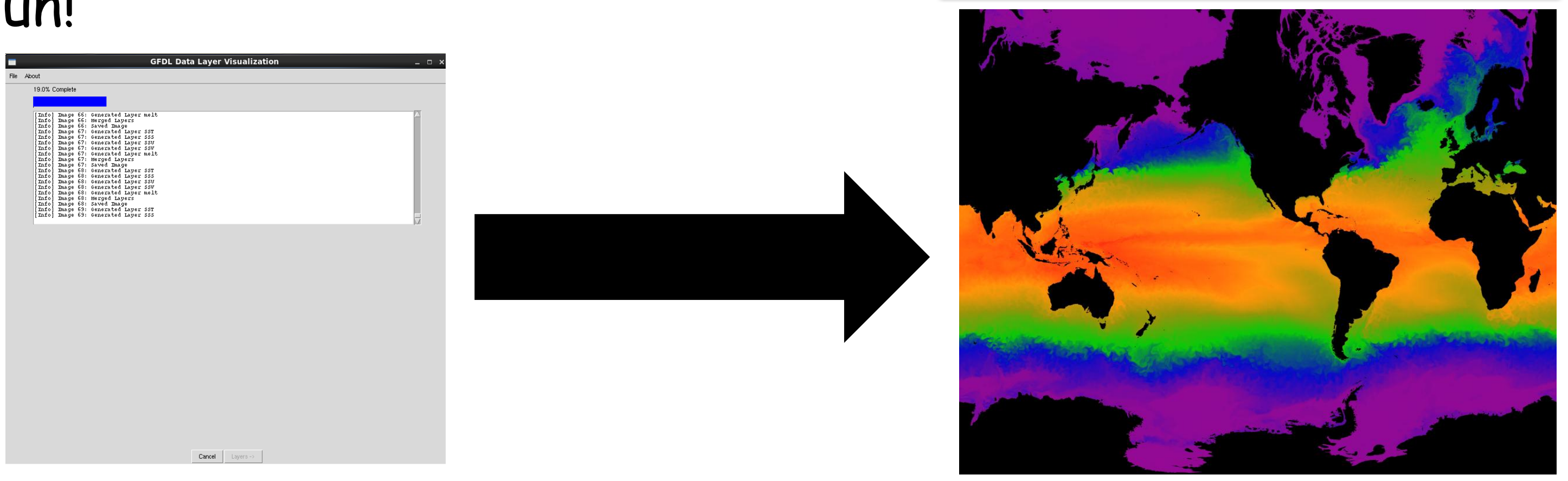
3. Select additional options
   - Choose time steps to render
   - Can set framerate, output location, etc.
   - Can select one of three methods:
     - GPU, Multicore, Serial
   - Live preview of a single layer (see next section)



4. View summary
   - See a summary of your selections in a custom configuration file format
   - Can save configuration file and run from terminal

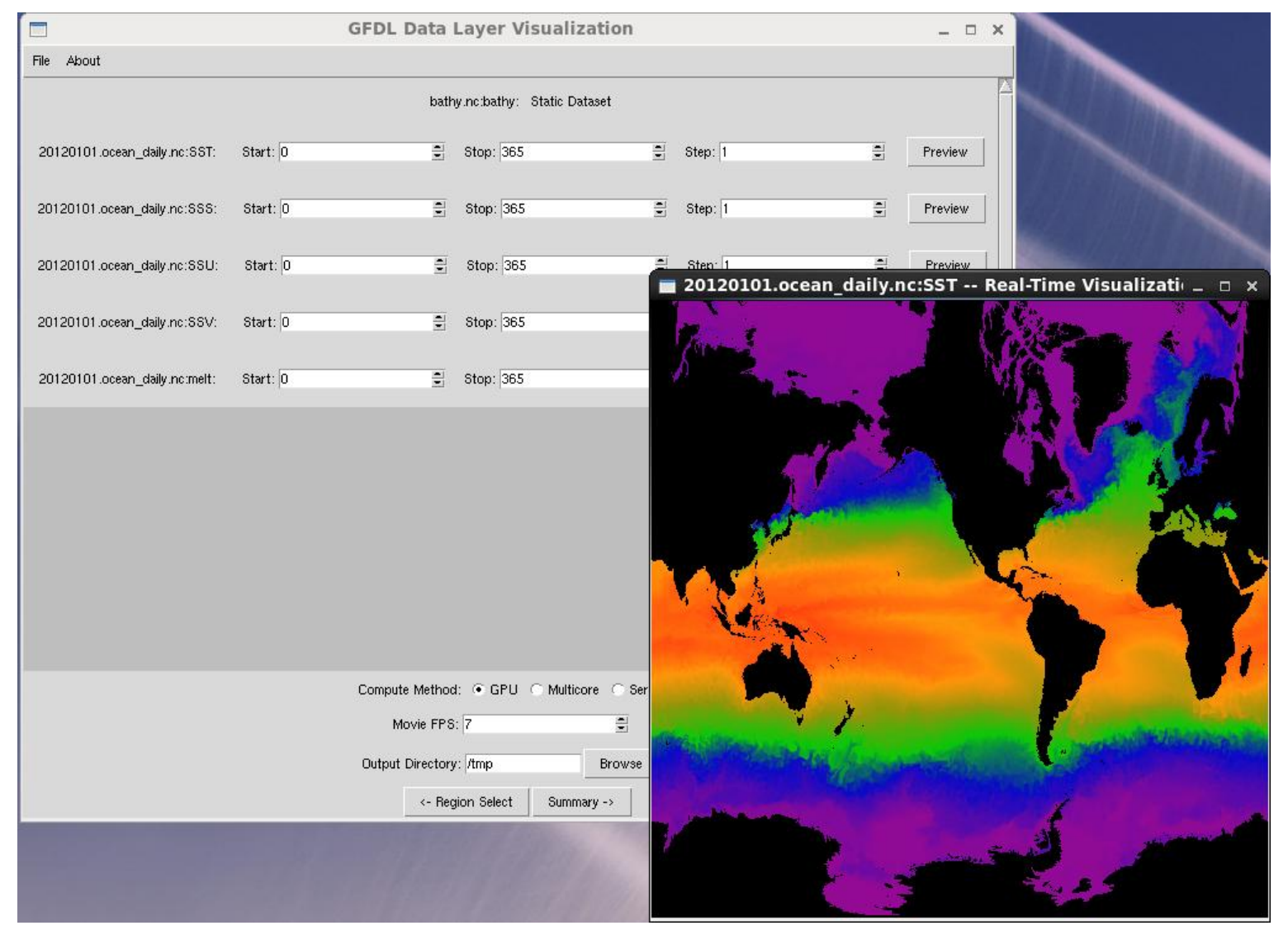


5. Run!



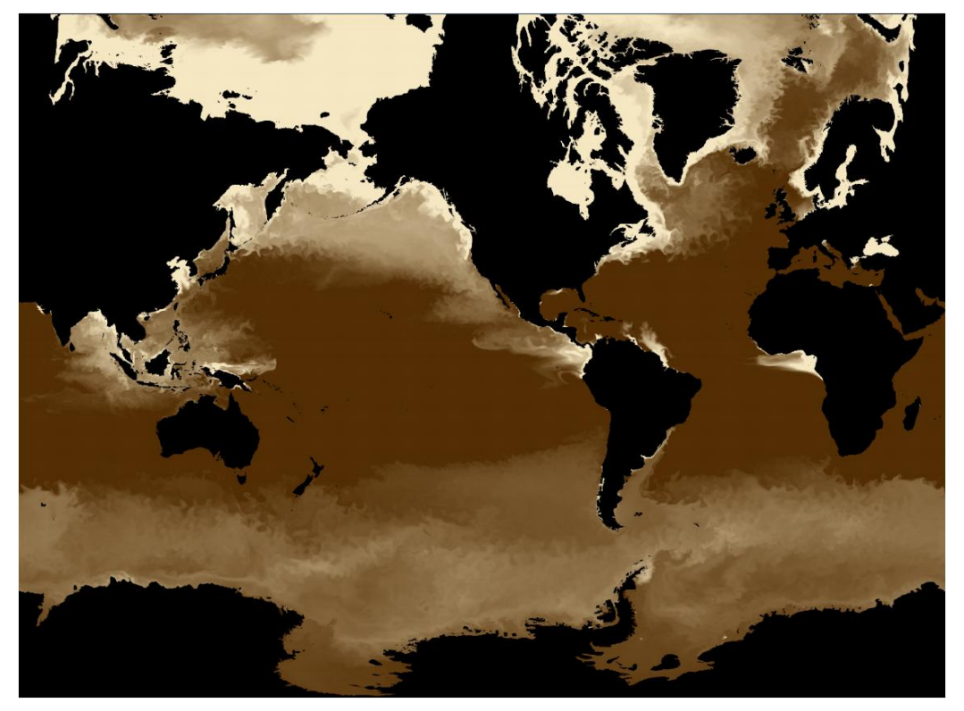
Real-Time Visualization

- Real-time preview of a dataset's timesteps
- Works using **PyOpenGL, PyCUDA**, and **CUDA/GL Interoperability**
- Supports key bindings for pause and reset in a pure OpenGL window!

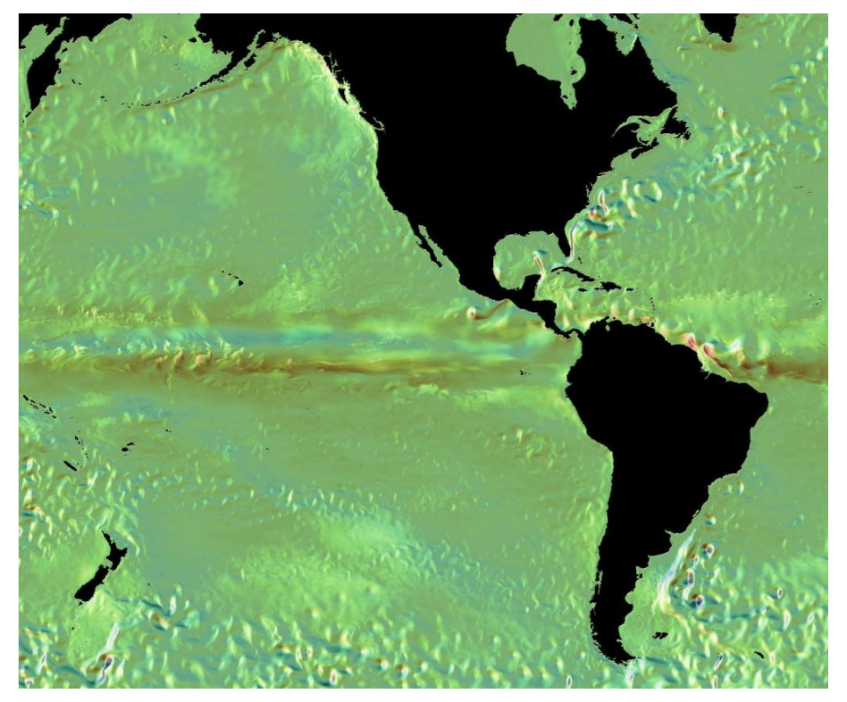


Conclusion

- **High-performance computing** approach to visualizing f(x,y,t) data
  - Uses GPU and multicore systems to maximize computing power
- Fast speeds allow for real-time interactive data preview
- Generality of software allows the visualization of **any f(x,y,t) dataset**, not just ocean data
- Using Python with **Tkinter** for the GUI makes it portable
- Written in pure Python, with a few strings of CUDA C
- **Already deployed** and working on GFDL computers
  - Using Anaconda made installation seamless



Sea Surface Salinity



Zonal/Meridional Velocities

Future Work

- Support arbitrary projections for display
  - Involves polygon mapping onto a grid
  - Would allow for viewing Earth from any direction
- View and slice N-dimensional time-dependent data
- Map images onto sphere for display as a globe
  - OpenGL texture mapping
- Add satellite terrain image to land background

References

- Full movies available at http://w3.pppl.gov/~efeibush/cm/
- **Anaconda** https://store.continuum.io/cshop/anaconda/
- **netcdf4-python** https://github.com/Unidata/netcdf4-python
- **PyCUDA** http://mathema.tician.de/software/pycuda/
- **PyOpenGL** http://pyopengl.sourceforge.net/

Acknowledgements

- *This work was made possible by funding from the Department of Energy for the Summer Undergraduate Laboratory Internship (SULI) program. This work is supported by the US DOE Contract No.DE-AC02-09CH11466*
- We would like to thank our mentor Eliot Feibush (PPPL). We would additionally like to thank Whit Anderson, Matthew Harrison, and Youngrak Cho of GFDL for the opportunity to work on this project this summer