# Gigapixel Image Rendering

Daniel Garcia-Briseno
Matthew Ostovarpour
Douglas Peterson
Thomas O'Brien

## Project Sponsor

- Trent Hare

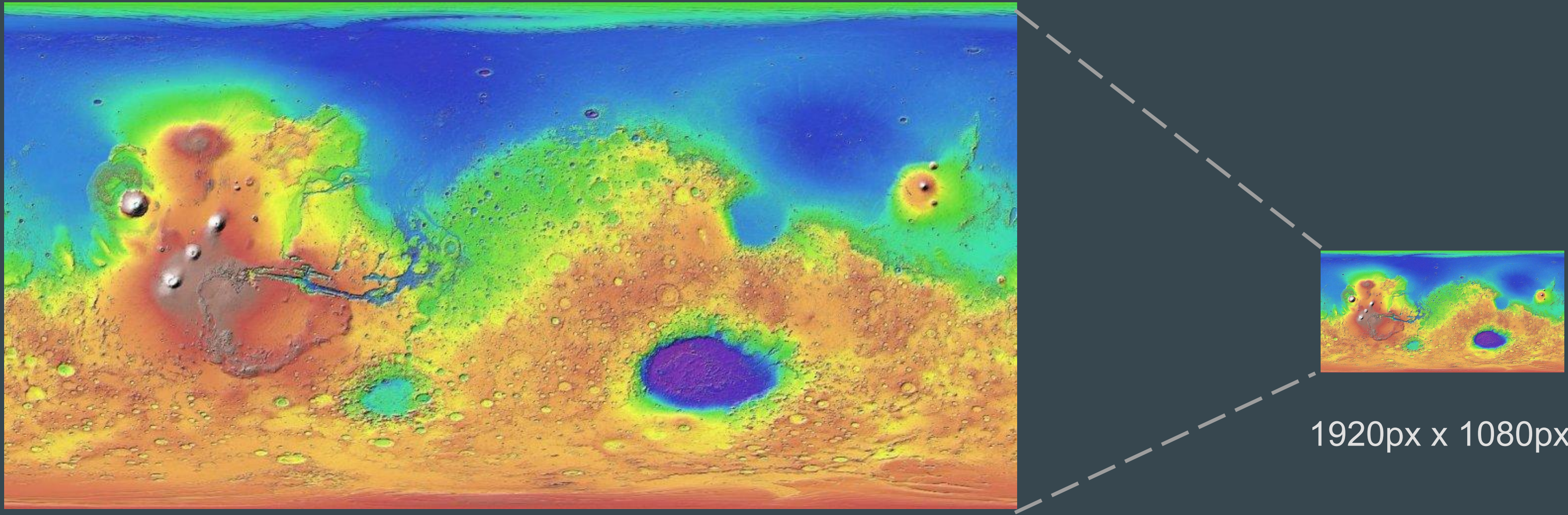
  Cartographer specializing in the creation of geospatial tools.

- United States Geological Survey

## Faculty Mentor

- Dr. James Palmer
  - NAU faculty since 2006
  - Chair of CS and EE
  - Research interests:
    - Domain specific computer languages
    - Visualization
    - Web-based architectures

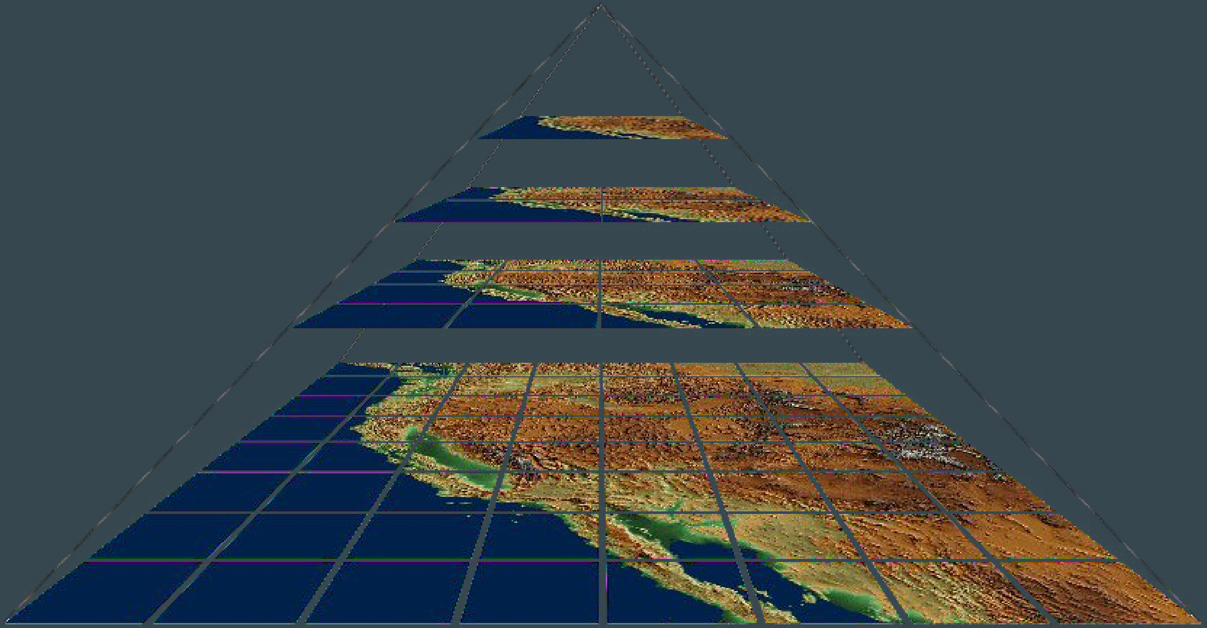
# Image Downsampling



46080px x 23040px and up

1920px x 1080px

## Pyramids

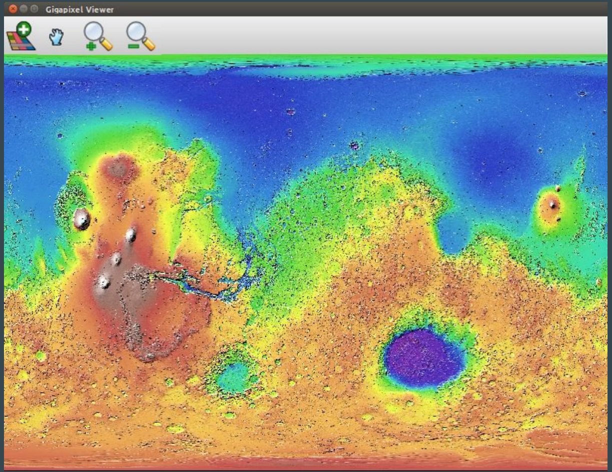


## What is GDAL?

- Widely Used
- Open Source
- Support for 100+ file formats
- Primary library used by USGS

# Stochastic Image Sampling

1. Sampling Algorithm in GDAL
2. Image Viewer



## Key Requirements

- **Algorithm**
  - Functional
    - Randomly sample pixel data
    - Implemented inside GDAL
  - Non-functional
    - Speed - Must be faster than pyramids
- **Viewer**
  - Functional
    - Allow pan and zoom features
    - Maintain image aspect ratio
  - Non-functional
    - Highly responsive

## Development

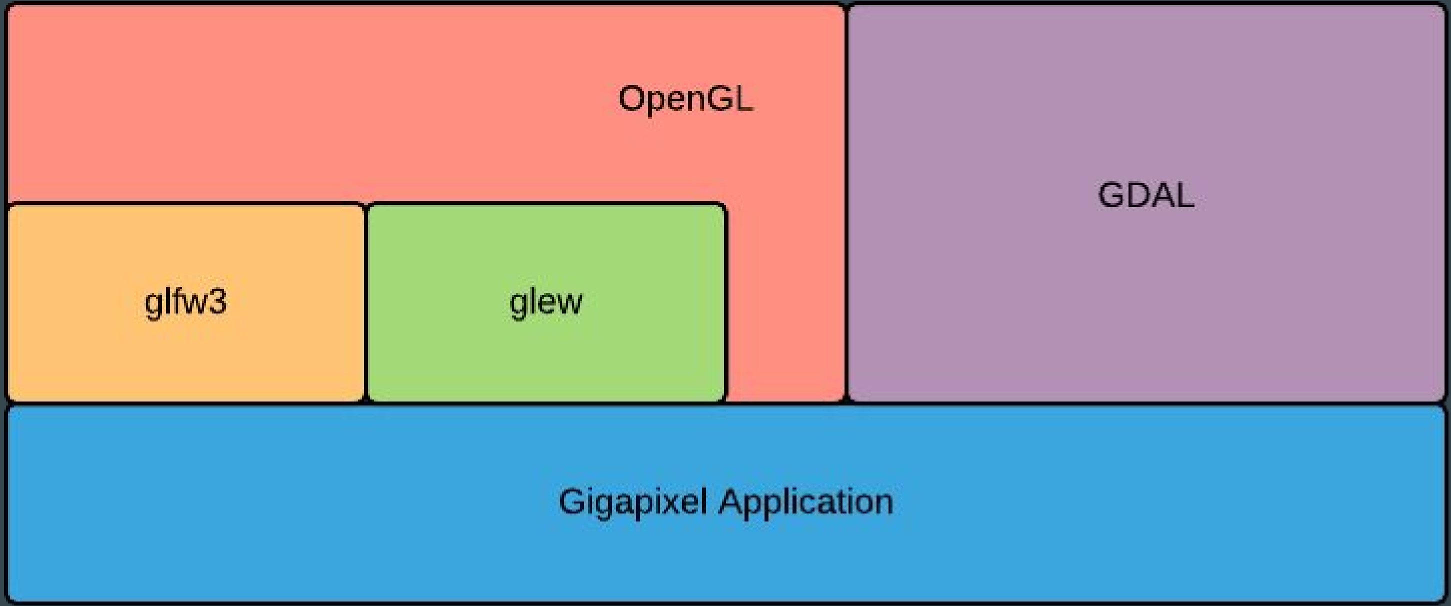
Waterfall

Trello

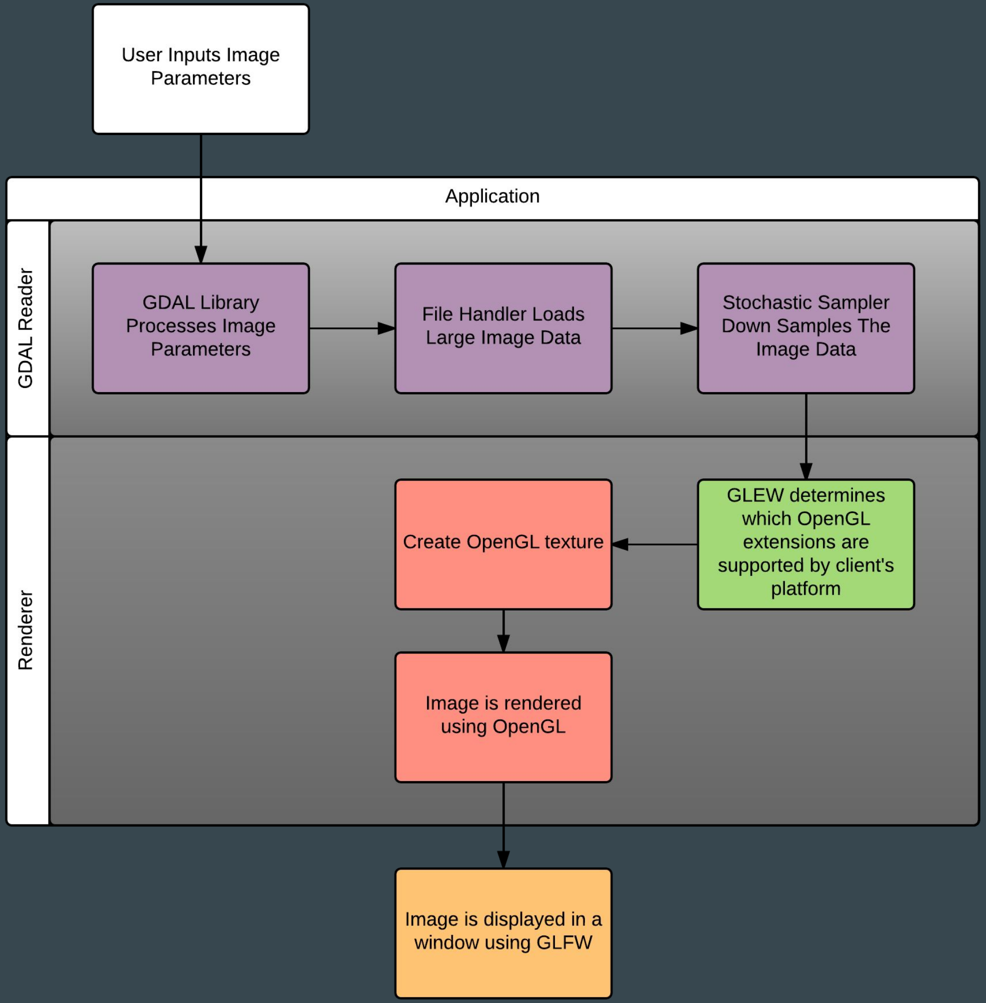
GitHub

Slack

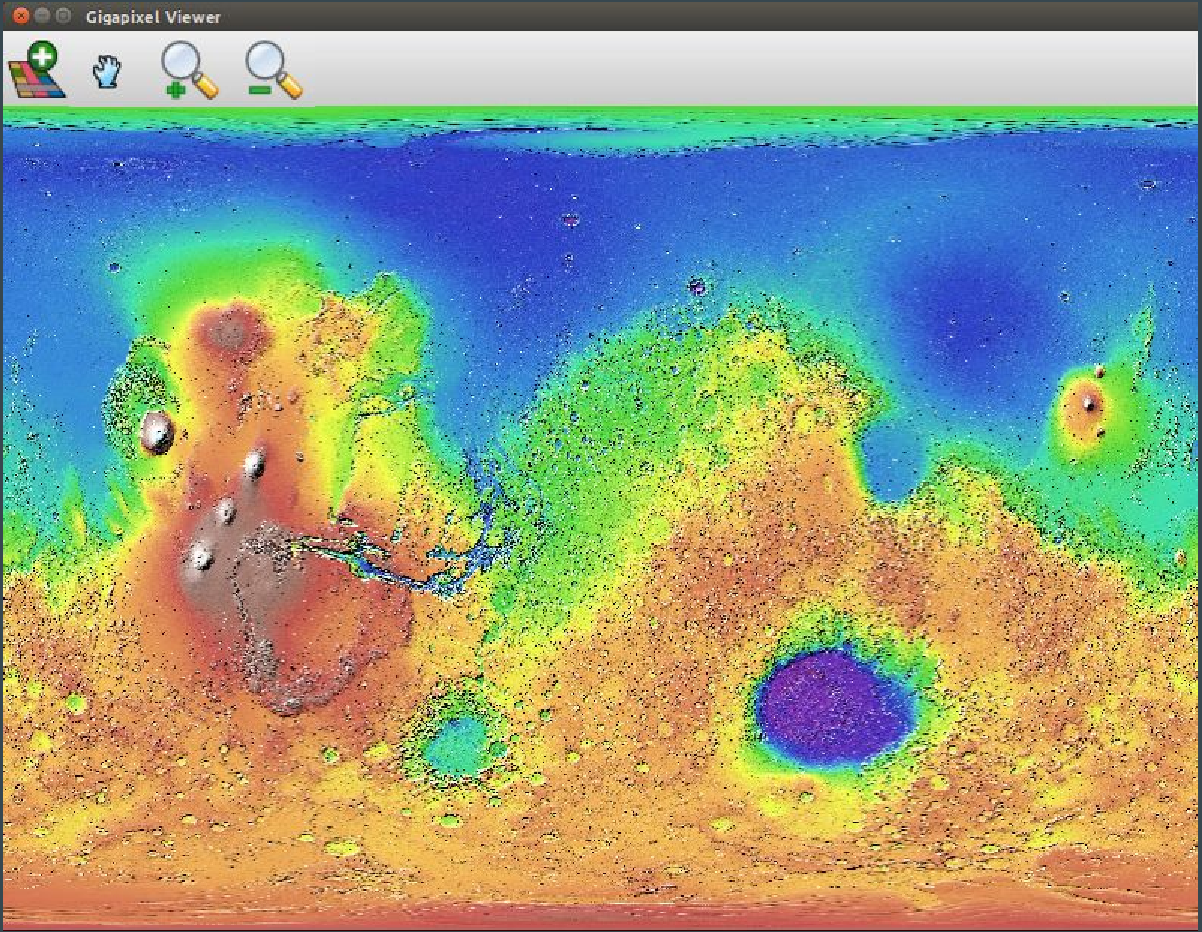
## Technology Stack



# Architecture

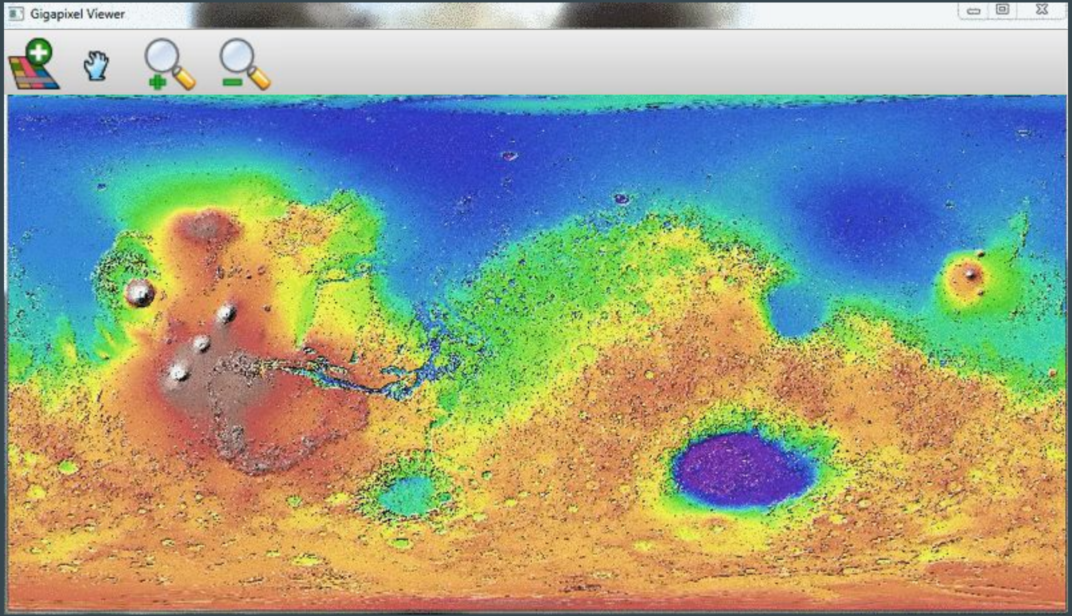

User Inputs Image Parameters
Application
GDAL Reader
GDAL Library Processes Image Parameters
File Handler Loads Large Image Data
Stochastic Sampler Down Samples The Image Data
Renderer
Create OpenGL texture
GLEW determines which OpenGL extensions are supported by client's platform
Image is rendered using OpenGL
Image is displayed in a window using GLFW

## Image Viewer



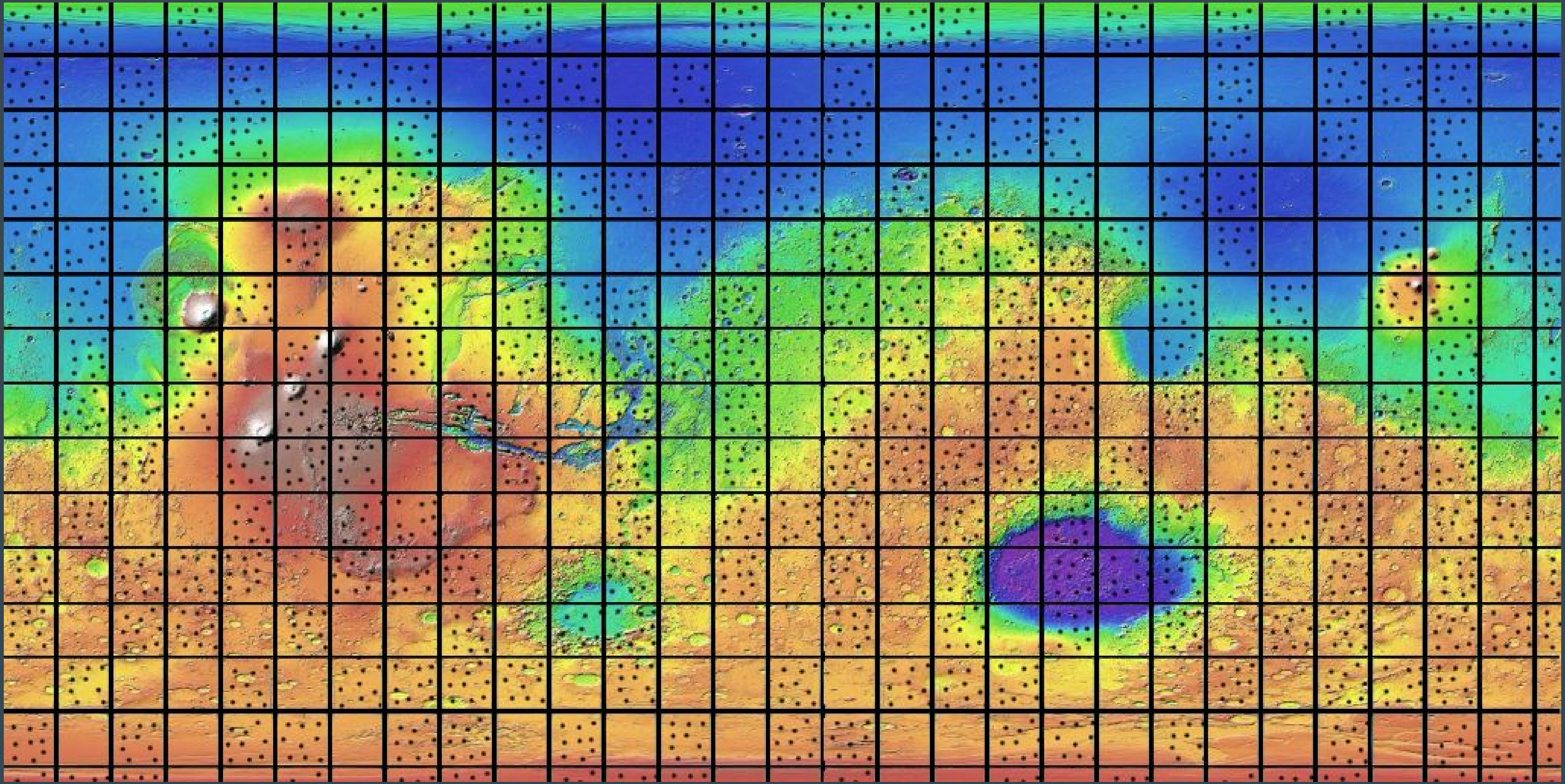
## Pan & Zoom Features

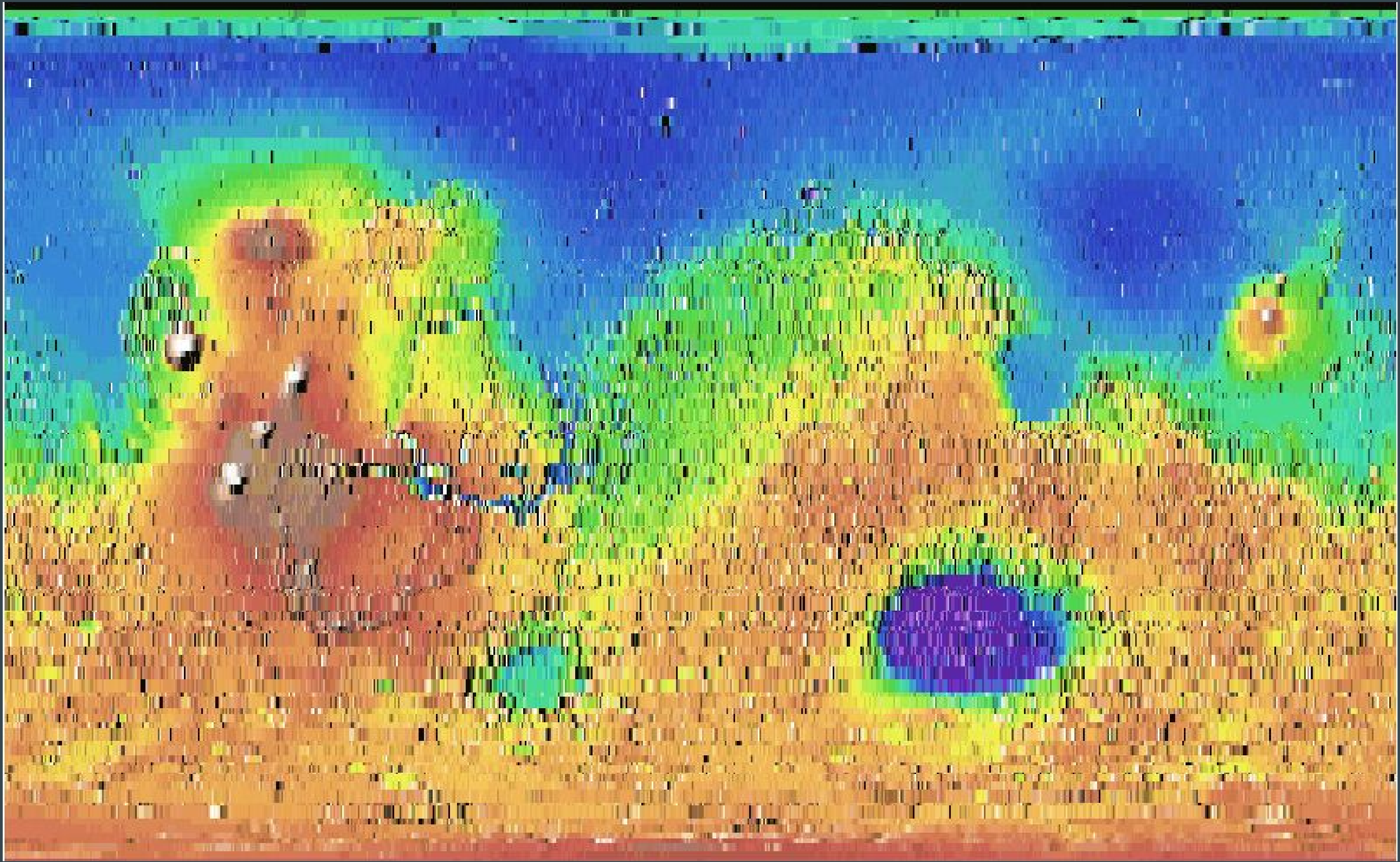
- Manipulating the image viewport
  - Changing viewport size allows for zoom
  - Changing viewport position allows for panning



## Stochastic Algorithm

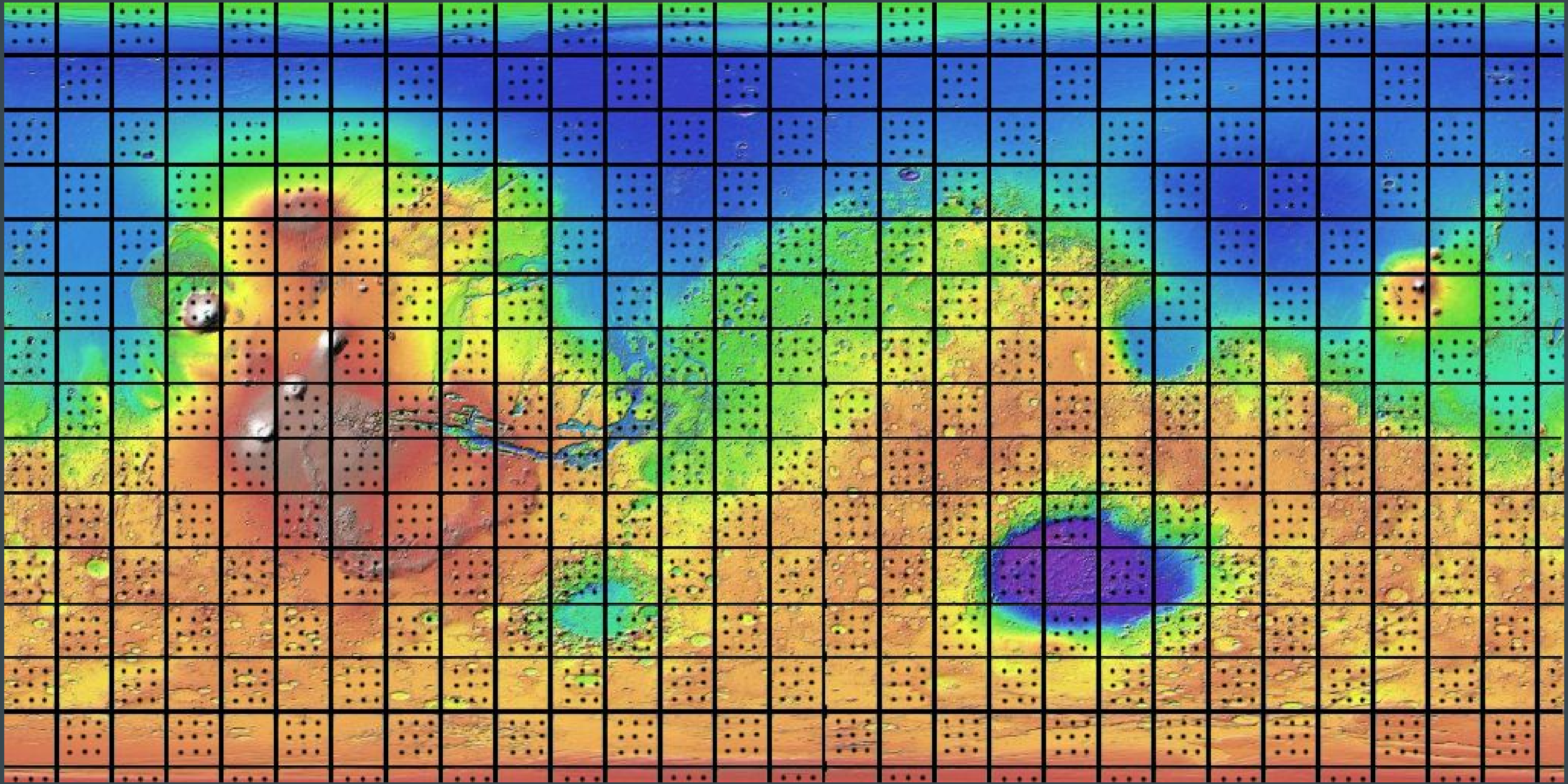
0. GDAL Calculates “most efficient way to read sections”
1. Determine how many sections  to look at, many are skipped.
2. Randomly choose sections
3. Get random pixels from the section and calculate output location

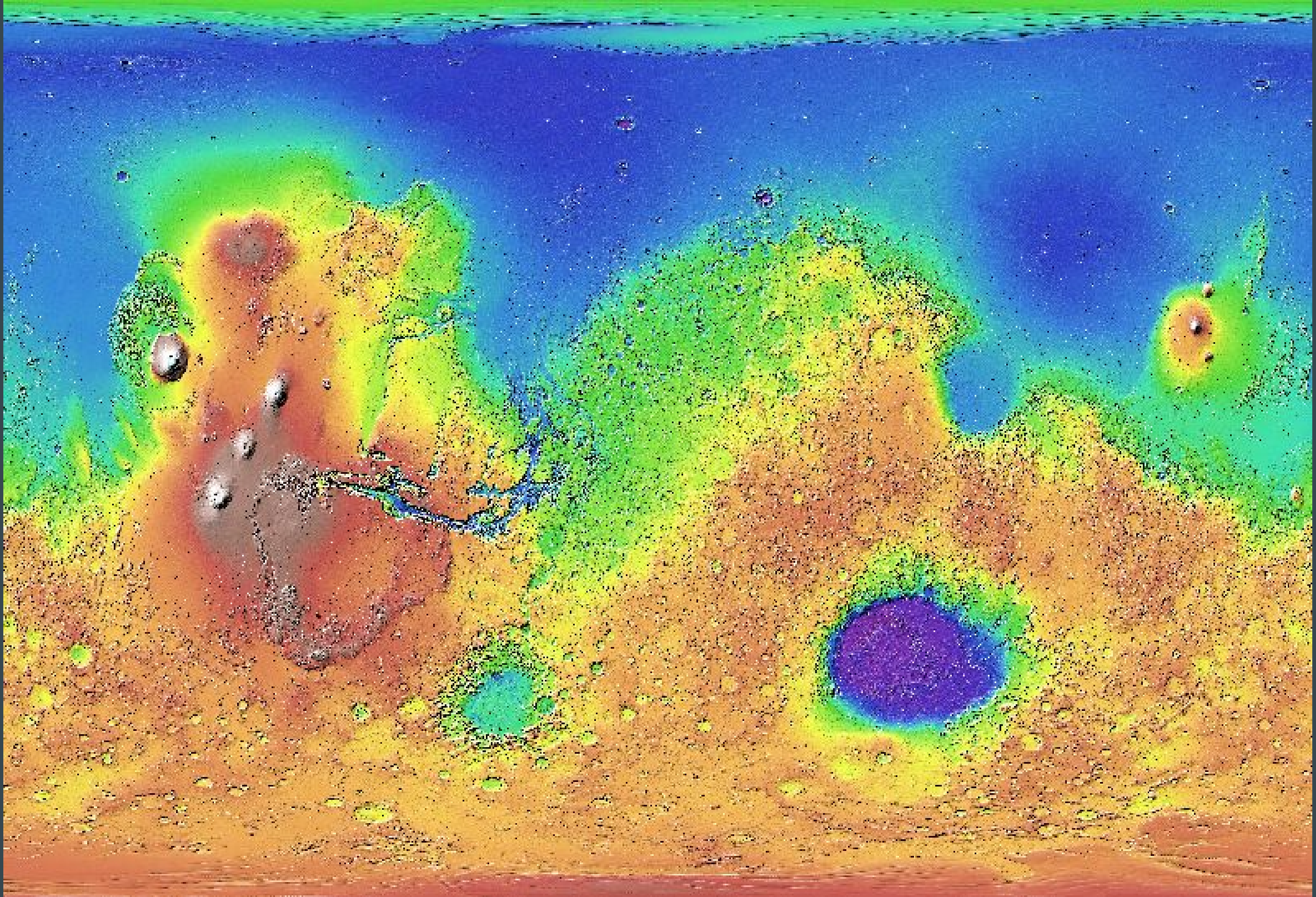




## Nearest Neighbor

1. GDAL determines sections to read
2. Determine how many sections need to be read
3. Determine how many pixels to get from each section
4. Evenly read sections and pixels within the section





## Algorithm Summary

### Stochastic

- Randomly loads image sections
- Randomly reads pixels
- Provides extra parameter for less reads

### Nearest Neighbor

- Evenly loads image sections
- Evenly reads pixels

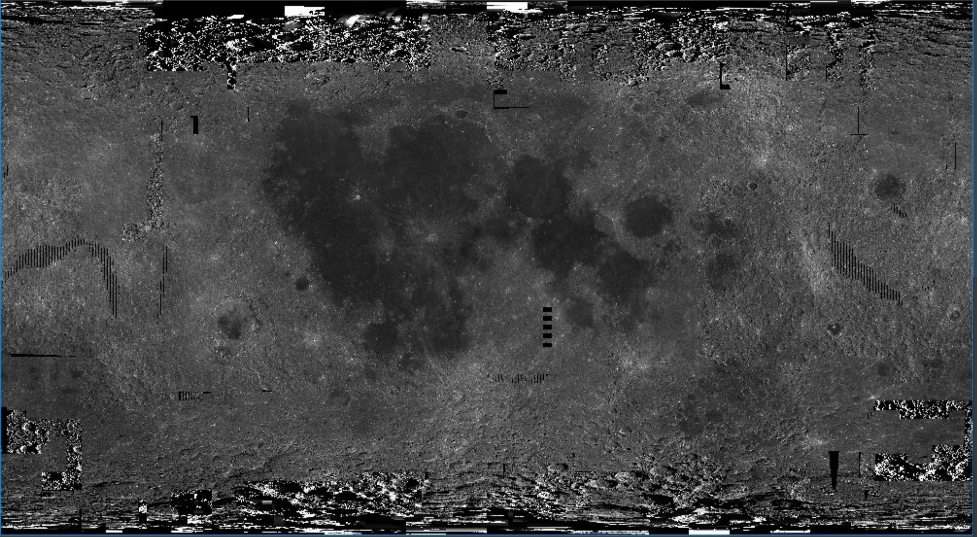
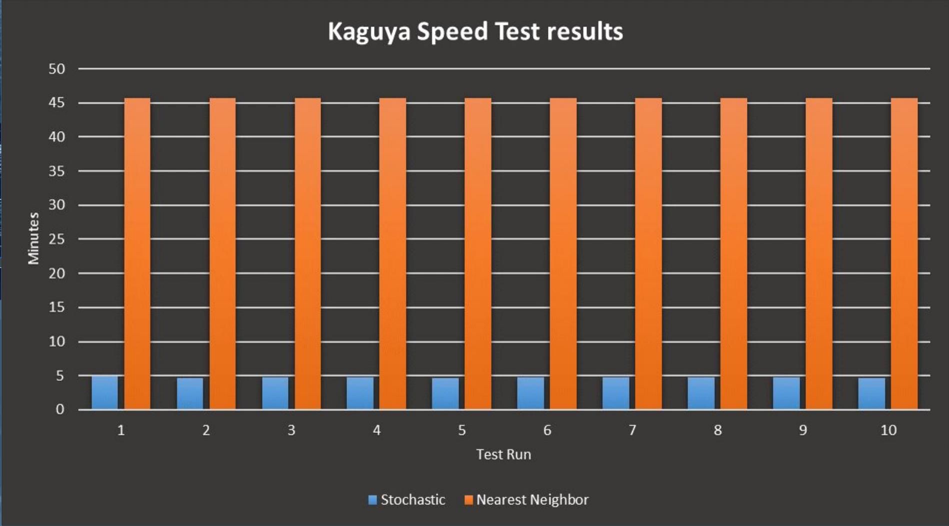


Image rendered using Nearest Neighbor



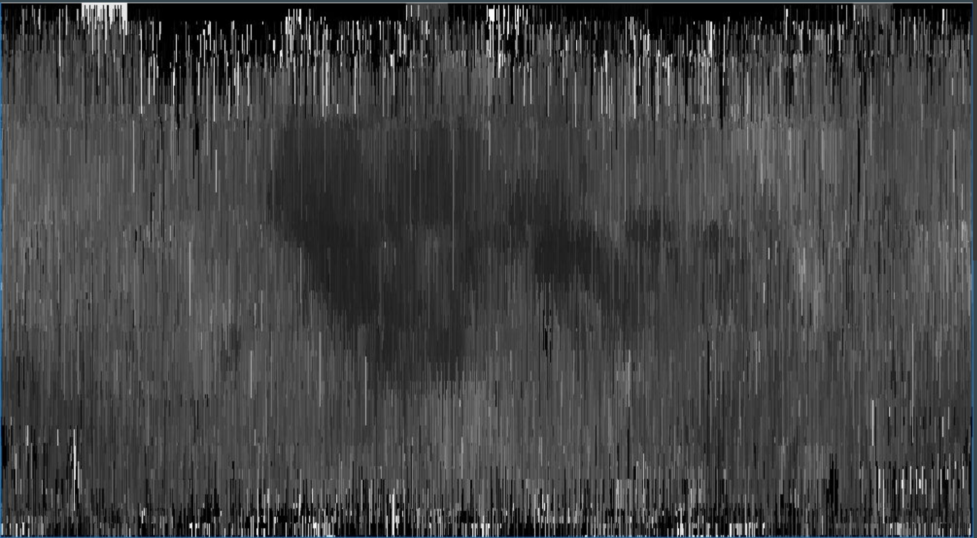


Image rendered using Stochastic Sampling

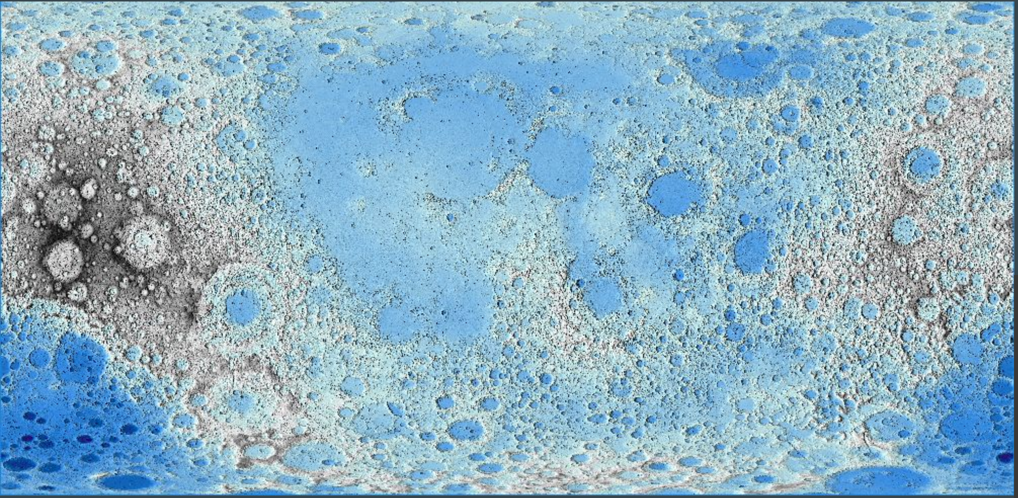
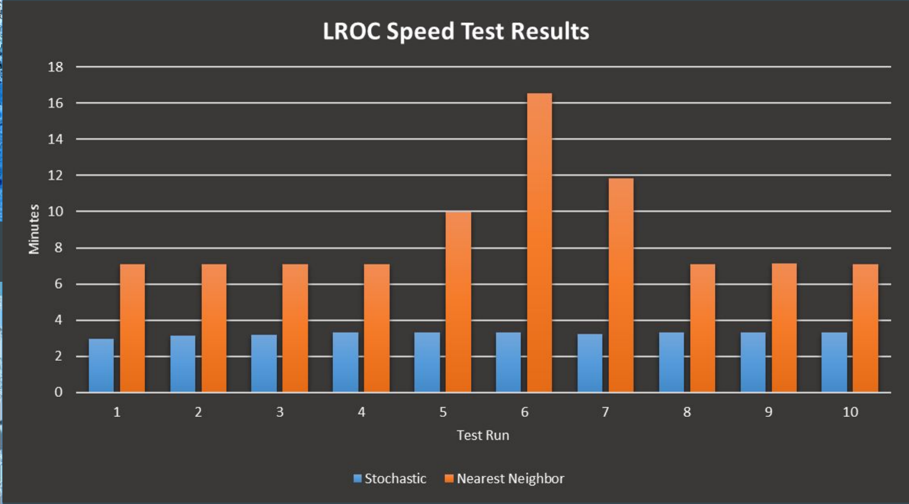


Image rendered using Nearest Neighbor



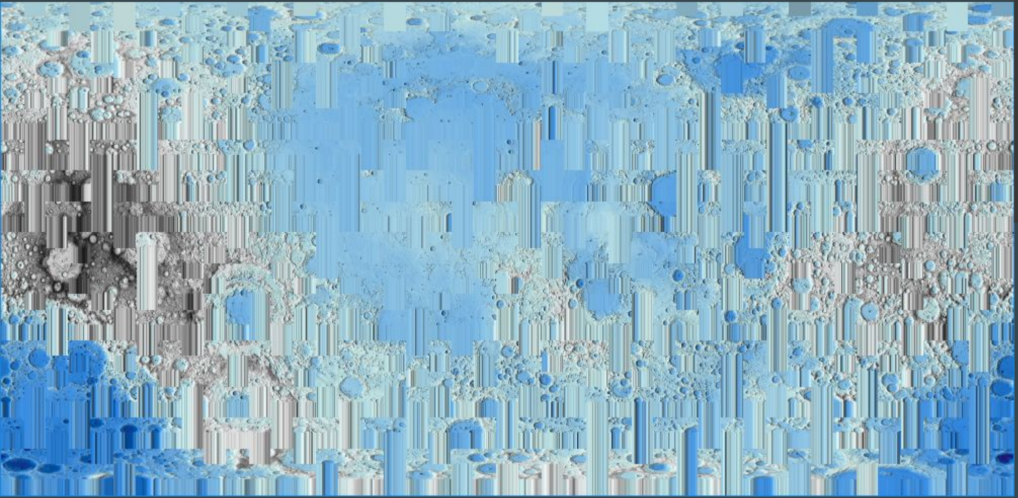


Image rendered using Stochastic Sampling

## Challenges & Risks

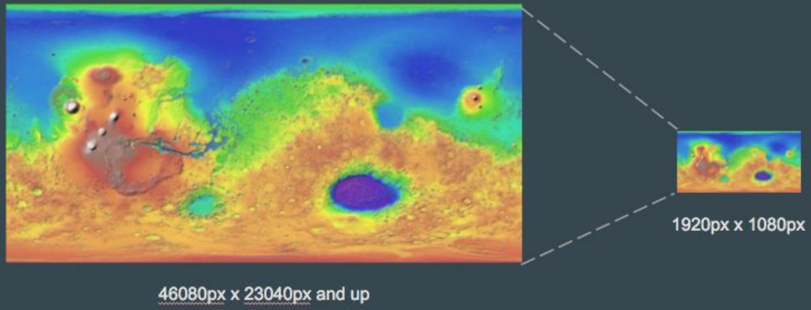
- Already Overcome
  - Image quality is lost during downsampling - Client wants a good representation, not excessive quality.
  - Hardware incompatibility
    - For windows: OpenGL depends on the user’s graphics card and drivers
    - This is not an issue for Linux or Mac
- Remaining
  - Integrate our software into USGS’s existing system
    - Issues running on virtual machine

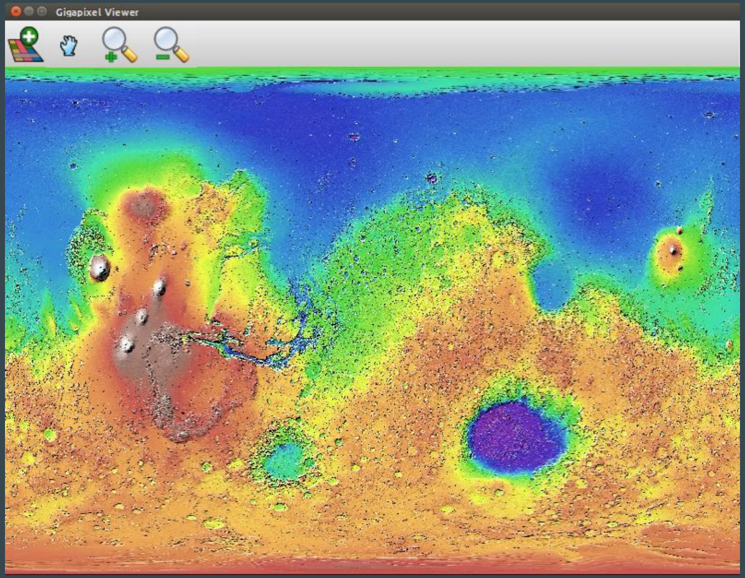
## Future Work

- Refactor code (if needed)
- Get feedback from GDAL users
- Integrate into the GDAL main repository

## Conclusion

- USGS needs a way to render large images faster than current methods
- Stochastic sampling techniques drastically reduce time needed to render image
- Two main components of our application
  - GDAL Reader
  - Renderer
- Finalize the project by working with USGS to fully integrate our code into their existing software structure.





## Gigapixel Image Rendering