CMPS 6360 Data Visualization Final Project

Background:

The martian south pole hosts the third largest reservoir of water ice in the solar system, after Antarctica and Greenland. Variations in ice albedo seen along the exposed interior of the ice are due to variable amounts of intermixed dust deposited during past climate configurations (Cutts and Lewis, 1982; Cutts, 1973; Fishbaugh et al., 2010; Milkovich and Plaut, 2008; Plaut et al., 2007; Phillips et al., 2008; Smith et al., 2016; Laskar et al., 2002, 2004; Murray et al., 1972). Therefore, the south polar layered deposits (SPLD) are a useful tool for reconstructing past climate conditions on Mars.

A picture containing dark

Description automatically generatedOrbital sounding radar allows observers to peer through the interior of the SPLD to understand its internal structure and distribution of material. This method is different from observing geological structure using optical sensors, as stratigraphic horizons in radar are defined by dielectric contrasts within the PLD materials. (Campbell and Morgan, 2018; Foss et al., 2017; Milkovich et al., 2009; Phillips et al., 2011; Plaut et al., 2007; Putzig et al., 2018; Whitten and Campbell, 2018). Reflections at these horizons are measured in two-way time relative to the instrument in orbit, which can be processed into radargrams to visualize the return signals (Figure 1).

Figure . Time-delay radargram of SHARAD (Shallow Radar) track 37633. SPLD radar stratigraphy is clearer on the right (Promethei Lingula), while volume scattering (a phenomenon I am also studying) dominates the rest of the SPLD.

In order to know the actual depth of the observed horizons, the composition of the material must be known. Previous work has established the SPLD to be predominantly water ice, with some CO2 ice, and ~10% dust by volume. Due to its dominant presence, typical depth corrections assume pure water ice for simplicity. To visualize the importance in knowing the material composition in radar data, one can take a vertical slice of a time-delay radargram and convert the time-depth into actual depth using different dielectric constants (dielectric constant is the ratio of electric permittivity in a material and a vacuum). The vertical extent of each slice will change depending on the material that is assumed, highlighting the importance in accurately judging its composition in order to locate features of interest within the ice.

Processing:

The script I wrote (datavis\_project.py) takes a specific dataset I created to generate two visualizations. The radar data comes from SHARAD (Shallow Radar) track 37633, which is shown as a radargram in Figure 1. I extracted ten adjacent traces (columns) of values from the raw radar data within a region called Promethei Lingula, where reflector stratigraphy is relatively clear. The columns record the radar power at surface of the SPLD and extends to the assumed basal interface. To smooth noise found at the sample resolution, I averaged the 10 traces laterally (horizontally). The relevant information is stored in three .csv files and imported into python, along with the radargram for visual reference. In Python, I clean the data and apply three different depth corrections to compare their differences. The three materials (dielectric constants) used were water ice (3.15), CO2 ice (2.1), and seawater ice (6.0).

Interactive Visualizations:

The first visualization I created displays a cropped radargram image in a subplot (left) with the starting trace position (red line) for the 10 averaged traces for reference. On the right subplot, three possible visualizations of ice depth are generated depending on the user’s mouse input. The x and y axes are radar power converted to dB and depth in meters, respectively. The default plot shows the depth of the ice column assuming a CO2 ice composition, or 2.1 dielectric constant. Horizontal peaks represent high dielectric contrasts whose measured depths are dependent on the material that is assumed. To visualize this, the user can click, right-click, or double-click the plot to show the same visualization for different materials. One click generates a water ice profile, while still maintaining the CO2 profile as a background item. Right-clicking sends water ice to the background and generates a profile for seawater ice, allowing all three to be compared. Double-clicking returns the plot to CO2 ice by itself. This visualization highlights the depth of a given horizon being sensitive to assumptions made by the observer.

To create this visualization, I generated an initial figure and then defined an onclick event using the matplotlib library in Python. Each interaction is controlled by conditional if/elif statements. Depending on the user’s input, the image redraws itself. The goal was to be able to use the cursor to trace each foreground profile with a cursor and follow the corresponding location on the red line, however this proved to be more difficult than anticipated, so I generated a separate visualization that partially achieves this.

The second visualization I created displays the same cropped radargram image on left, but instead of a single power-depth profile, I generated all three at once in their own separate bounds. Each subplot draws a crosshair that sticks to the profile within the plot and follows the mouse cursor as it moves up and down the plot. Each plot is sensitive to the cursor movement in other plots, meaning the user can follow a single profile and see the corresponding location in the other plots in real time. The coordinates corresponding to the location of the crosshairs are also displayed near the x-axes for each subplot. To do this, I created the FollowDataWithCursor class that takes an axes object, x, and y. Within are two functions: the first initializes the inputs and creates the crosshairs, and the second (follow\_mouse) updates the input function depending on the location of the mouse cursor. One unforeseen result of having all the plots update when the cursor moves is response time. Overall, the cursor is pretty sluggish, but is notably quicker if only one plot has interaction functionality.

Obstacles:

The original goal was to combine the second visualization’s functionality with the first visualization. The main obstacle I ran into was preserving the mouse event handling (cursor crosshairs) after initializing a click event. The cursor will follow the initial profile generated in the first visualization, but after initializing a click event, the cursor was unaware of the new profile and continued to act as if the original profile was still the focus. My suspicion is that this is due to a limitation in the way I wrote the follow\_mouse function in the FollowDataWithCursor class, since the inputs require an initial x and y dataset that I don’t think I was able to have update following subsequent events. If I had this working, I imagine it would also speed up the plot updates for cursor movements since it would only have to redraw one plot, rather than three.

**Figure 2. Static images of both visualizations.**

