Interactive control

Oliver W. Layton

CS251: Data analysis and visualization

Lecture 9, Spring 2019

Monday February 25
Plan

• Interactivity: Connecting viewing and user input
• Panning the view
• Rotating the view (start today)
Types of interactive view manipulation

We can use the keyboard and mouse to make viewing interactive! How could this be beneficial?

- What kinds of useful movements through the view volume could we control via mouse and/or keyboard?

- **Panning/translation**: Keep U and VUP fixed, move VRP around in the viewing plane.
  - Instead of moving the data, we move the virtual observer. Mathematically equivalent.

- **Rotation**: Rotate the observer's position about a sphere, centered on the middle of the view volume.
  - Another option: Keep observer fixed, change only U, VUP, VPN vectors to manipulate the observer's direction of gaze.

- **Scaling**: Change the view volume extent \((E_x, E_y, E_z)\) so that more/less of the data fit inside the screen window.
Control degrees of freedom (1/2)

- Key idea: input devices (mouse, keyboard, etc.) control the view transformation matrix (VTM) parameters. User interactively updates the view of data.

- How many axes of movement can a typical mouse or trackpad control?
  - 2

- How about a keyboard button?
  - 1
Control degrees of freedom (2/2)

- Mouses usually have 2-3 buttons (left, middle wheel, right), which act like more buttons.
  - Trackpads\(^1\) have one- and two-finger tap.

- "Overload" mouse movement behavior by requiring user to hold down a modifier button (e.g. Cmd, Opt, Cntl, Shift) to enter a new "mode".
  - Example: holding down a mouse click while moving the mouse pans the view, while holding Cmd and doing the same thing scales the viewing extent.

\(^1\) I will post code with an example of how you can map B3 onto a trackpad with tkinter.
Control Law for Panning/Translation

A control law describes how inputs map onto actions. Let's develop one for panning.

- **Panning** refers to translation parallel to the observer's viewing screen (e.g. virtual observer side-steps).
  - Move observer rather than data
- Horizontal mouse movement should move the VRP along which observer axis?
- Vertical mouse movement should move the VRP along which observer axis?
- Mathematically: $\Delta U \propto \Delta x$, where $\Delta U$ is movement thru data space, and $\Delta x$ is mouse motion on the screen.
- Imagine clicking and dragging the mouse and seeing the data zoom by. What problems should we anticipate as we design our control law?
Panning problem 1

• Our mouse motion moves us way too fast or too slow through the data space. How can we fix our control law?

• Add a constant $k_p$: $\Delta U = k_p \Delta x$.

• If $k_p$ is too small, data will lag behind mouse.

• If $k_p$ too large, data will precede the mouse screen motion.

• How can we calibrate $k_p$?

  • Ratio of view volume extent to screen extent (i.e. $k_{p,x} = \frac{E_x}{s_x}$). Units meaningful.

  • $\neq 1:1$ ratio is fine for tuning the interface (i.e. $k_{p,x} = c_x \frac{E_x}{s_x}$).
Panning problem 2

• We move mouse in one direction, but data moves on-screen in the opposite direction! Suggested fix?

• Negate $k_p$ in the reversed directions (x and/or y).
Panning implementation (1/2)

1. Calculate how much the mouse moved (pixels) since the last known position: \((\Delta x, \Delta y)\)

2. Calibrate the panning speed by defining \(k_p = (k_{p,x}, k_{p,y})\): \(k_p = \left( \frac{E_x}{s_x}, \frac{E_y}{s_y} \right)\)

3. Calculate "raw" motion thru data space, using calibrated scaling factor \(k_p\):

\[
(\Delta u, \Delta v) = k_p \cdot (\Delta x, \Delta y) = \left( \Delta x \frac{E_x}{s_x}, \Delta y \frac{E_y}{s_y} \right)
\]

4. Assign x-y movement contributions to observer’s U and VUP axes\(^2\):

\[
\begin{align*}
\Delta VRP_x &= \Delta u U_x + \Delta v VUP_x \\
\Delta VRP_y &= \Delta u U_y + \Delta v VUP_y \\
\Delta VRP_z &= \Delta u U_z + \Delta v VUP_z
\end{align*}
\]

\(^2\) We involve U and VUP because of rotations. U and VUP may NOT be aligned with X and Y.
Panning implementation (2/2)

- 5) Update observer's position using our just-computed VRP displacement: \( \vec{VRP} = \vec{VRP} + \Delta \vec{VRP} \)

- 6) Re-build of view transformation matrix to reflect the change in position (7 step process).

- 7) Given the new view, compute updated locations of data points and other visual objects.
Rotation (Observer circling around data)

• We want the virtual observer to circle around the data in a sphere, centered on the middle of the viewing volume.

• Let's draw this scenario and the workflow out on the board.