1 Connecting User Input and Viewing

Interactive data viewing requires us to connect user input to changes in the view system parameters. User actions generate either specific changes in the view parameters or incremental modifications. The forms of interactive view modification include panning (translating), scaling, and rotating.

- Panning/Translating: should move the VRP around the view plane, keeping the orientation of VUP and U fixed.
- Scaling: should increase or decrease the extent of the view volume in the view plane, keeping other parameters fixed.
- Rotation:
  - In 2D, should rotate the VUP vector about the z-axis, modifying the VUP vector and the derived U vector, but leaving their Z values at 0.
  - In 3D, the VRP may move in a circle around a fixed point (e.g. center of the current view volume). The same rotation can be applied to the VPN, VUP, and U vectors. An alternative interface method is to rotate the view direction according to mouse motions, keeping the VRP constant during the rotation.

The mouse gives the user the ability to move within 2 dimensions. Therefore, whether we are translating, rotating, or scaling, we have to have a method for translating user actions into modifications of the view reference coordinates.

- Translation of the mouse: translation of the VRP in the view plane
- Translation of the mouse: rotation of the VUP vector
- Translation of the mouse: scaling of the extent of the view window

A typical mouse has three buttons (left, right, center) which translate into three separate button clicks. Many mice also have a wheel or trackball on them, which provides an additional input device. The keyboard and menu items provide additional inputs that we can use as modifiers to the mouse action.

The key is how we connect the user interface elements to the parameters of the view transformation. Some key design questions include the following.
• Which mouse actions, buttons, or menu states connect to which view parameters?
• How does each action or button press relate to changes in one or more view parameters?
• What is the control law for the relationship?
• What are the control parameters of the relationship?
• Why is this a useful relationship?

Note that the first four questions are up to the programmer. You can make the relationships be whatever you want. The last question is what should guide the design process, and is often the motivation for user studies that look at the effects of different design decisions.
1.1 Interfaces as Control Laws

Any time we are connecting an input to an action, we have to write a control law. Consider, for example, connecting the motion of a mouse to panning of the data. The mouse moves in screen coordinates. Therefore, the input to the system is in pixels. Pixels, however, are not a meaningful unit in the data space. We can write the relationship between mouse motion and motion in the data space as follows, where $\Delta U$ is the motion in the data space, $\Delta x$ is motion in screen space, and $k_p$ is a proportional constant relating the two motions.

$$\Delta U = k_p \Delta x$$

(1)

We can control the relationship between the two spaces by varying the value of $k_p$. If we set $k_p = \frac{E_x}{S_x}$, which is the ratio of the data space extent to the screen space, then the data will appear to track the mouse as it moves. However, we could make $k_p$ smaller or larger than that ratio and get different effects. If $k_p$ is smaller, the data will lag the mouse motion. If $k_p$ is bigger, the data will precede the mouse motion. The former is useful for fine tuning of a visualization, the latter is useful for moving quickly between different parts of the data.

We are not limited to a simple proportional relationship, however. Consider, for example, the use of an inertia term.

$$\Delta U_t = k_p \Delta x + k_n \Delta U_{t-1}$$

(2)

The inertia term means that the motion of the data is dependent not only on the current user input, but also the prior user input. A more complex form of the motion results if we make $k_n$ dependent upon whether the user is actively controlling the device. Under active control, we can make $k_n = 0$, changing it to something like $k_n = 0.9$ when the user stops actively controlling the mouse. The iPod, and other multi-touch devices make use of this type of relationship to enable quick flipping and scrolling through lists.

Control theory also gives us an intuitive way of understanding the results of different design choices.

1. overdamped - system doesn’t respond rapidly enough (sluggish)
2. critically damped - system responds just right
3. underdamped - system responds too strongly and in an effort to correct the overshoot, you end up oscillating
4. unstable - system just doesn’t do the right thing
1.2 Panning within the view plane

- Each change in the mouse position corresponds to a change in the position in the data space
- Horizontal motion in the view plane should move along the U axis
- Vertical motion in the view plane should move along the VUP axis
- The view volume extent and the screen size tell us how to scale pixel motion to data space motion

Process

1. Calculate how much the mouse moved on the screen $\Delta x, \Delta y$.

2. Scale the motion into data space by dividing by the screen size and multiplying by the extent.

$$ (\Delta u, \Delta v) = (\Delta x \frac{Ex}{S_x}, \Delta y \frac{Ey}{S_y}) $$ (3)

3. Multiply the horizontal screen motion by the U axis and the vertical screen motion by the V axis to get the motion of the VRP in data space.

$$ \Delta VRP_x = \Delta u U_x + \Delta v VUP_x $$
$$ \Delta VRP_y = \Delta u U_y + \Delta v VUP_y $$
$$ \Delta VRP_z = \Delta u U_z + \Delta v VUP_z $$ (4)

4. Add the motions onto the VRP.

$$ VRP = (VRP_x + \Delta VRP_x, VRP_y + \Delta VRP_y, VRP_z + \Delta VRP_z) $$ (5)

5. Recalculate the view transformation matrix

6. Calculate the new view locations of the data

7. Adjust the coordinates of the visual objects
1.3 Scaling the extent

- The extent should scale uniformly in all directions
- The user should be able to scale up and down using the same motion
- An easy solution is to translate vertical motion into scaling

Process

1. Store the initial mouse click as a reference point $P_0$
2. Store the initial view extent $E_0$
3. For each mouse motion
   (a) Calculate the vertical distance between the initial click and the current mouse location.
      \[ \Delta v = P_{iy} - P_{0y} \]  
      (6)
   (b) Generate a multiplication factor from the difference. Set the value of $k$ to control the speed of the scaling.
      \[ f = 1.0 + k\Delta v \]  
      (7)
   (c) Bound the factor on the zoom side to a small number larger than zero (e.g. 0.05)
   (d) Multiply the initial view extent by the factor and update the view extent
      \[ E_i = fE_0 \]  
      (8)
   (e) Recalculate the view matrix
   (f) Calculate the new view locations of the data
   (g) Adjust the coordinates of the visual objects
1.4 Rotating the view

1.4.1 Rotation that circles around the data

Rotation (Circle around data): rotate the view plane around the center of the view volume.

- Left-right motion: translate the center of view volume extent to the origin, align the VRC axes, rotate around VUP, unalign, untranslate

\[
x_h(\theta_h) = T(VRP + \frac{E_z}{2}VPN)R_{xyz}^tR_y(\theta_h)R_{xyz}T(-VRP - \frac{E_z}{2}VPN)
\]  

- up-down motion: translate the center of view volume extent to the origin, align the VRC axes, rotate around U, unalign, untranslate

\[
x_v(\theta_v) = T(VRP + \frac{E_z}{2}VPN)R_{xyz}^tR_x(\theta_v)R_{xyz}T(-VRP - \frac{E_z}{2}VPN)
\]

We need to use these matrices to transform the VRP point (homogeneous coordinate of 1) and the U, VUP, and VPN vectors (homogeneous coordinate of 0). Then rebuild the view matrix to update the data. The effect is that of the data rotating in space while the user stays fixed.

We can combine the two matrices thus:

\[
X(\theta_h, \theta_v) = X_v(\theta_v)X_h(\theta_h)
\]

Why? That is an exercise left to the reader. It is a simple matter of the matrices being inverses of each other. They cancel out.

Note: \( T \) indicates a translation matrix, \( R_{xyz}^t \) indicates an alignment matrix, \( R_y(\theta) \) indicates a rotates around the Y axis, and \( R_x \) indicates a rotation around the X axis. We have seen the first two matrices in the 2- and 3-D viewing pipeline.

To rotate about the Y-axis:

\[
R_y(\theta) = \begin{pmatrix}
cos \theta & 0 & \sin \theta & 0 \\
0 & 1 & 0 & 0 \\
-\sin \theta & 0 & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

To rotate about the X-axis:

\[
R_x(\theta) = \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & \cos \theta & -\sin \theta & 0 \\
0 & \sin \theta & \cos \theta & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]
Process:

1. Store the initial mouse click as a reference point \( P_0 \)
2. Store the initial view reference axes (VRP, UP, U, and VPN) as VRP\(_0\), UP\(_0\), U\(_0\), and VPN\(_0\).
3. For each mouse motion
   (a) Calculate the horizontal distance between the initial click and the current mouse location.
   \[
   \Delta x = P_{ix} - P_{0x} \tag{13}
   \]
   (b) Convert it to the rotation angle (for rotating about UP). Here, I am assuming that dragging the mouse across the entire window leads to a complete \(2\pi\) rotation:
   \[
   \theta_h = -\frac{\Delta x}{0.5S_x}\pi \tag{14}
   \]
   (c) Calculate the vertical distance between the initial click and the current mouse location.
   \[
   \Delta y = P_{iy} - P_{0y} \tag{15}
   \]
   (d) Convert it to the rotation angle (for rotating about U)
   \[
   \theta_v = \frac{\Delta y}{0.5S_y}\pi \tag{16}
   \]
   (e) Compute \(X(\theta_h, \theta_v)\)
   (f) Recompute the VPN, UP, U, and VRP by putting VRP\(_0\), UP\(_0\), U\(_0\), and VPN\(_0\) into homogeneous coordinates and multiplying them by \(X\) (\(X\) is on the left)
   (g) Recalculate the view matrix
   (h) Calculate the new view locations of the data
   (i) Adjust the coordinates of the visual objects
1.4.2 Fly-through rotation

Rotation (fly-through): rotate around VUP, anchored at the VRP, for right-left motion of the mouse and rotate around the U-vector, anchored at the VRP, for up-down motion. The rotations apply only to the orientation of the view coordinate system, which is defined as a set of vectors and is translation invariant.

- Left-right motion: translate VRP to the origin, align the axes, rotate by $\theta_h$ about the Y axis, invert the alignment, then translate back. To undo a rotation, we can multiply by the inverse matrix. The inverse of a rotation matrix happens to be its transpose, which makes inverting rotations an easy thing to do.

$$X_h(\theta_h) = T(VRP)R_{xyz}(U, VUP, VPN)^tR_y(\theta_h)R_{xyz}(U, VUP, VPN)T(-VRP) \quad (17)$$

- Up-down motion: align axes, rotate by $\theta_v$ about the X axis, invert the alignment

$$X_v(\theta_v) = T(VRP)R_{xyz}(U, VUP, VPN)^tR_x(\theta_v)R_{xyz}(U, VUP, VPN)T(-VRP) \quad (18)$$

Use $X_u(\theta_u)$ and $X_v(\theta_v)$ to transform the VRP point (homogeneous coordinate of 1) and the U, VUP, and VPN vectors (homogeneous coordinate of 0). Then rebuild the view matrix to update the data. The effect is that of turning your head around while the world stays fixed. Note that if both $\theta_u$ and $\theta_v$ are non-zero, you can do a single transformation process with the two rotation matrices in the center of the expression.

1.5 Controlling with keys instead of the mouse

Translation: using typical keyboard controls (e.g. wasd for forward/left/back/right) works well. Number pad controls also work well.

- Forward/backward moves the VRP along the VPN by some step size $\gamma$.

$$VRP_{t+1} = VRP_t + \gamma VPN \quad (19)$$

- Left/right moves the VRP along the U axis.

$$VRP_{t+1} = VRP_t + \gamma U \quad (20)$$

- Up/down moves the VRP along the VUP axis.

$$VRP_{t+1} = VRP_t + \gamma VUP \quad (21)$$

Acknowledgement: Most of the material in this lecture is taken from Bruce Maxwell’s spring 2012 CS251 notes.