

Interactive scaling

Oliver W. Layton

CS251: Data analysis and visualization

Lecture 11, Spring 2019

Friday March 1

Plan

Plan

- Finish rotation

Plan

- Finish rotation
- Interactive scaling

Plan

- Finish rotation
- Interactive scaling
- Review panning/translation

Plan

- Finish rotation
- Interactive scaling
- Review panning/translation
- Keyboard control

Circling rotation implementation

Compute joint x-y rotation $X(\theta_h, \theta_v)$:

$$T \left(V R P + \frac{E_z}{2} V P N \right) R_{XYZ}^T R_x(\theta_v) R_y(\theta_h) R_{XYZ} T \left(-V R P - \frac{E_z}{2} V P N \right)$$

where

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Input to circling rotation matrix pipeline

Input to circling rotation matrix pipeline

- Input to matrix multiplication chain on last slide: observer coordinate system $\vec{U}_0, V\vec{U}P_0, V\vec{P}N_0$ (pre-rotation versions), **NOT** the data D .

Input to circling rotation matrix pipeline

- Input to matrix multiplication chain on last slide: observer coordinate system $\vec{U}_0, V\vec{U}P_0, V\vec{P}N_0$ (pre-rotation versions), **NOT** the data D .
- Output: "New" $\vec{U}, V\vec{U}P, V\vec{P}N$ that reflect the observer's rotated view.

Input to circling rotation matrix pipeline

- Input to matrix multiplication chain on last slide: observer coordinate system $\vec{U}_0, V\vec{U}P_0, V\vec{P}N_0$ (pre-rotation versions), **NOT** the data D .
- Output: "New" $\vec{U}, V\vec{U}P, V\vec{P}N$ that reflect the observer's rotated view.
- Update $\vec{U}, V\vec{U}P, V\vec{P}N$ by horizontally stacking them (column vectors) in a matrix with 0 homogeneous coordinates (M). Add VRP to M as a column vector with homogeneous coordinate 1.

Input to circling rotation matrix pipeline

- Input to matrix multiplication chain on last slide: observer coordinate system $\vec{U}_0, V\vec{U}P_0, V\vec{P}N_0$ (pre-rotation versions), **NOT** the data D .
- Output: "New" $\vec{U}, V\vec{U}P, V\vec{P}N$ that reflect the observer's rotated view.
- Update $\vec{U}, V\vec{U}P, V\vec{P}N$ by horizontally stacking them (column vectors) in a matrix with 0 homogeneous coordinates (M). Add VRP to M as a column vector with homogeneous coordinate 1.
 - Let's draw out why we want homogenous coordinate of 0 (not 1) for VRP.

Input to circling rotation matrix pipeline

- Input to matrix multiplication chain on last slide: observer coordinate system $\vec{U}_0, V\vec{U}P_0, V\vec{P}N_0$ (pre-rotation versions), **NOT** the data D .
- Output: "New" $\vec{U}, V\vec{U}P, V\vec{P}N$ that reflect the observer's rotated view.
- Update $\vec{U}, V\vec{U}P, V\vec{P}N$ by horizontally stacking them (column vectors) in a matrix with 0 homogeneous coordinates (M). Add VRP to M as a column vector with homogeneous coordinate 1.
 - Let's draw out why we want homogenous coordinate of 0 (not 1) for VRP.
- Multiply by the joint rotation matrix: $X(\theta_h, \theta_v)M$

Input to circling rotation matrix pipeline

- Input to matrix multiplication chain on last slide: observer coordinate system $\vec{U}_0, V\vec{U}P_0, V\vec{P}N_0$ (pre-rotation versions), **NOT** the data D .
- Output: "New" $\vec{U}, V\vec{U}P, V\vec{P}N$ that reflect the observer's rotated view.
- Update $\vec{U}, V\vec{U}P, V\vec{P}N$ by horizontally stacking them (column vectors) in a matrix with 0 homogeneous coordinates (M). Add VRP to M as a column vector with homogeneous coordinate 1.
 - Let's draw out why we want homogenous coordinate of 0 (not 1) for VRP.
- Multiply by the joint rotation matrix: $X(\theta_h, \theta_v)M$
- Re-build the VTM matrix (THIS affects the data matrix D).

Input to circling rotation matrix pipeline

- Input to matrix multiplication chain on last slide: observer coordinate system $\vec{U}_0, V\vec{U}P_0, V\vec{P}N_0$ (pre-rotation versions), **NOT** the data D .
- Output: "New" $\vec{U}, V\vec{U}P, V\vec{P}N$ that reflect the observer's rotated view.
- Update $\vec{U}, V\vec{U}P, V\vec{P}N$ by horizontally stacking them (column vectors) in a matrix with 0 homogeneous coordinates (M). Add VRP to M as a column vector with homogeneous coordinate 1.
 - Let's draw out why we want homogenous coordinate of 0 (not 1) for VRP.
- Multiply by the joint rotation matrix: $X(\theta_h, \theta_v)M$
- Re-build the VTM matrix (THIS affects the data matrix D).
- Update the view locations for the data and visual objects.

"Head" rotation

"Head" rotation

- Simulates the turning of the virtual observer's head, the data in the world remains fixed.

"Head" rotation

- Simulates the turning of the virtual observer's head, the data in the world remains fixed.
- What needs to change about our rotation workflow?

"Head" rotation

- Simulates the turning of the virtual observer's head, the data in the world remains fixed.
- What needs to change about our rotation workflow?
 - Rotation anchor point is observer's eye (VRP).

"Head" rotation

- Simulates the turning of the virtual observer's head, the data in the world remains fixed.
- What needs to change about our rotation workflow?
 - Rotation anchor point is observer's eye (VRP).
 - We translate by VRP rather than $\text{VRP} + \text{Ez}/2 \text{ VPN}$. Everything else stays the same.

"Head" rotation

- Simulates the turning of the virtual observer's head, the data in the world remains fixed.
- What needs to change about our rotation workflow?
 - Rotation anchor point is observer's eye (VRP).
 - We translate by VRP rather than $\text{VRP} + E_z/2 \text{ VPN}$. Everything else stays the same.
- Here's our joint rotation transform:

"Head" rotation

- Simulates the turning of the virtual observer's head, the data in the world remains fixed.
- What needs to change about our rotation workflow?
 - Rotation anchor point is observer's eye (VRP).
 - We translate by VRP rather than VRP + Ez/2 VPN. Everything else stays the same.
- Here's our joint rotation transform:
- $T(VRP) R_{XYZ}^T R_x(\theta_v) R_y(\theta_h) R_{XYZ} T(-VRP)$

Scaling

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.
- Scaling up (zooming in) simulates bringing view closer to the data. Effect on extent?

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.
- Scaling up (zooming in) simulates bringing view closer to the data. Effect on extent?
 - DECREASE our extent!

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.
- Scaling up (zooming in) simulates bringing view closer to the data. Effect on extent?
 - DECREASE our extent!
- Scaling down (zooming out) simulates bringing view farther from the data. Effect on extent?

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.
- Scaling up (zooming in) simulates bringing view closer to the data. Effect on extent?
 - DECREASE our extent!
- Scaling down (zooming out) simulates bringing view farther from the data. Effect on extent?
 - INCREASE our extent!

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.
- Scaling up (zooming in) simulates bringing view closer to the data. Effect on extent?
 - DECREASE our extent!
- Scaling down (zooming out) simulates bringing view farther from the data. Effect on extent?
 - INCREASE our extent!
- Control (*B3motion* in tkinter)

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.
- Scaling up (zooming in) simulates bringing view closer to the data. Effect on extent?
 - DECREASE our extent!
- Scaling down (zooming out) simulates bringing view farther from the data. Effect on extent?
 - INCREASE our extent!
- Control (*B3motion* in tkinter)
 - Trackpad: 1) Hold modifier key 2) click (and hold it) trackpad 3) slide finger on trackpad vertically

Scaling

- Think of pinching-to-zoom on iPhone. We want *uniform* scaling in all directions.
- Can be implemented by changing the extent (bounding box) of our view of the data.
- Scaling up (zooming in) simulates bringing view closer to the data. Effect on extent?
 - DECREASE our extent!
- Scaling down (zooming out) simulates bringing view farther from the data. Effect on extent?
 - INCREASE our extent!
- Control (*B3motion* in tkinter)
 - Trackpad: 1) Hold modifier key 2) click (and hold it) trackpad 3) slide finger on trackpad vertically
 - Mouse: 1) Click mouse (and hold it), 2) use scroll wheel.

Scaling implementation (1/2)

Scaling implementation (1/2)

1. Save the screen position of the initial mouse click (P_0).

Scaling implementation (1/2)

1. Save the screen position of the initial mouse click (P_0).
2. Save the initial view volume extent (E_0).

Scaling implementation (1/2)

1. Save the screen position of the initial mouse click (P_0).
2. Save the initial view volume extent (E_0).
3. Define a scaling factor k_s that controls the speed of scaling/*zooming*.

Scaling implementation (1/2)

1. Save the screen position of the initial mouse click (P_0).
2. Save the initial view volume extent (E_0).
3. Define a scaling factor k_s that controls the speed of scaling/*zooming*.
4. For each mouse movement i :

Scaling implementation (1/2)

1. Save the screen position of the initial mouse click (P_0).
2. Save the initial view volume extent (E_0).
3. Define a scaling factor k_s that controls the speed of scaling/*zooming*.
4. For each mouse movement i :
 - Calculate the current vertical screen displacement relative to the initial click (pixels): $\Delta v = P_i y - P_0 y$

Scaling implementation (1/2)

1. Save the screen position of the initial mouse click (P_0).
2. Save the initial view volume extent (E_0).
3. Define a scaling factor k_s that controls the speed of scaling/*zooming*.
4. For each mouse movement i :
 - Calculate the current vertical screen displacement relative to the initial click (pixels): $\Delta v = P_i y - P_0 y$
 - Convert displacement to scale factor f (1.0 means...?):
 $f = 1.0 + k_s \Delta v$

Scaling implementation (2/2)

Scaling implementation (2/2)

- Threshold scale factor so that it doesn't grow too large (e.g. 3.0) or too small (e.g. 0.1). Why?

Scaling implementation (2/2)

- Threshold scale factor so that it doesn't grow too large (e.g. 3.0) or too small (e.g. 0.1). Why?
- Actually scale the extent: $E_i = fE_0$

Scaling implementation (2/2)

- Threshold scale factor so that it doesn't grow too large (e.g. 3.0) or too small (e.g. 0.1). Why?
- Actually scale the extent: $E_i = fE_0$
- Re-build VTM

Scaling implementation (2/2)

- Threshold scale factor so that it doesn't grow too large (e.g. 3.0) or too small (e.g. 0.1). Why?
- Actually scale the extent: $E_i = fE_0$
- Re-build VTM
- Update the positions of data points and visual objects in the view volume.

Scaling and the orthographic pipeline

Which stage(s) of the orthographic viewing pipeline does scaling influence?

$$T_2 S_2 S_1 R_{XYZ} T_1 D$$