

Coordinate systems

Oliver W. Layton

CS251: Data analysis and visualization

Lecture 4, Fall 2018

Wednesday February 13

Plan

- Recap: Aesthetics, scale, GUI.
- Visualization coordinate systems
- 2D example
- Homogeneous coordinates
- Dot product and matrix-vector right multiplication
- Translation and scaling matrices

Data coordinates

- The native coordinates of the data space.
- The max, min, and average values of each variable are defined in this space.
- For any data set, there is a bounding box in data coordinates that contains all of the data.
- Can be any number of dimensions, depending on how many features you have in the data. 100 features: 100 dimensional data volume.

Viewing volume coordinates (1/2)

- **Viewing volume:** Volume (e.g. cube) that will contain the data that we want to visualize.
- **Extent:** SIZE of the data space that we want to view/visualize. Can be entire (or subset of) data space.
 - May be defined by max-min of some of the data variables, or a smaller portion than that.
- **Viewing volume origin:** One of the corners of the viewing volume is at 0 — where all feature values have 0 components.
- At most 3D.

Viewing volume coordinates (2/2)

Remember: Viewing volume coordinates origin at 0.

Two options to set up the view volume coordinate axes:

- Overlap / match the data coordinate axis orientations (i.e. data "x direction" points in same direction as view "x direction")
 - Then **viewing volume coordinates** have extent measured along the data variable directions.
- Have arbitrary oriented axes (not aligned with data axes).
 - Then **viewing volume coordinates** have set of orthonormal axes.
 - Extent of viewing volume measured along the orthonormal axes.

Normalized viewing volume coordinates

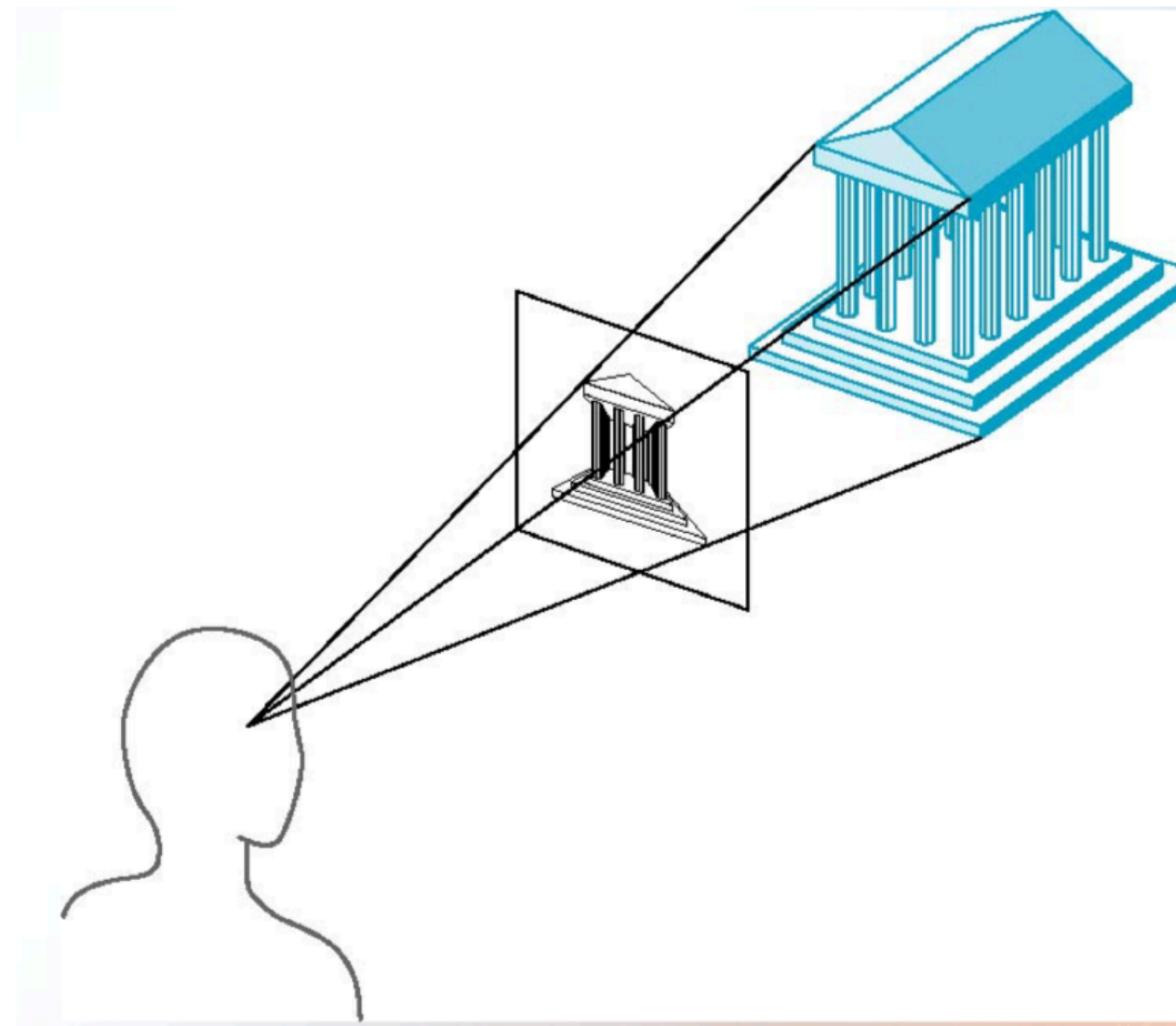
- Scale the data so that all visible points within the view volume fit within the range of $[0, 1]$ in all dimensions.

Observer (Screen) coordinates

- Coordinate system that coincides with the center of the virtual observer's "eye".
- Requires a **3D-to-2D projection** of the 3D world onto a 2D plane that spans the x-y axes of the virtual camera that models the observer's eye.
 - Imagine a projector screen that moves with the observer
- Think first-person shooter game
 - Data coordinates: game world, not all seen at once.
 - Observer coordinates: Virtual observer's view on your computer screen, often accompanied by a gun overlay.

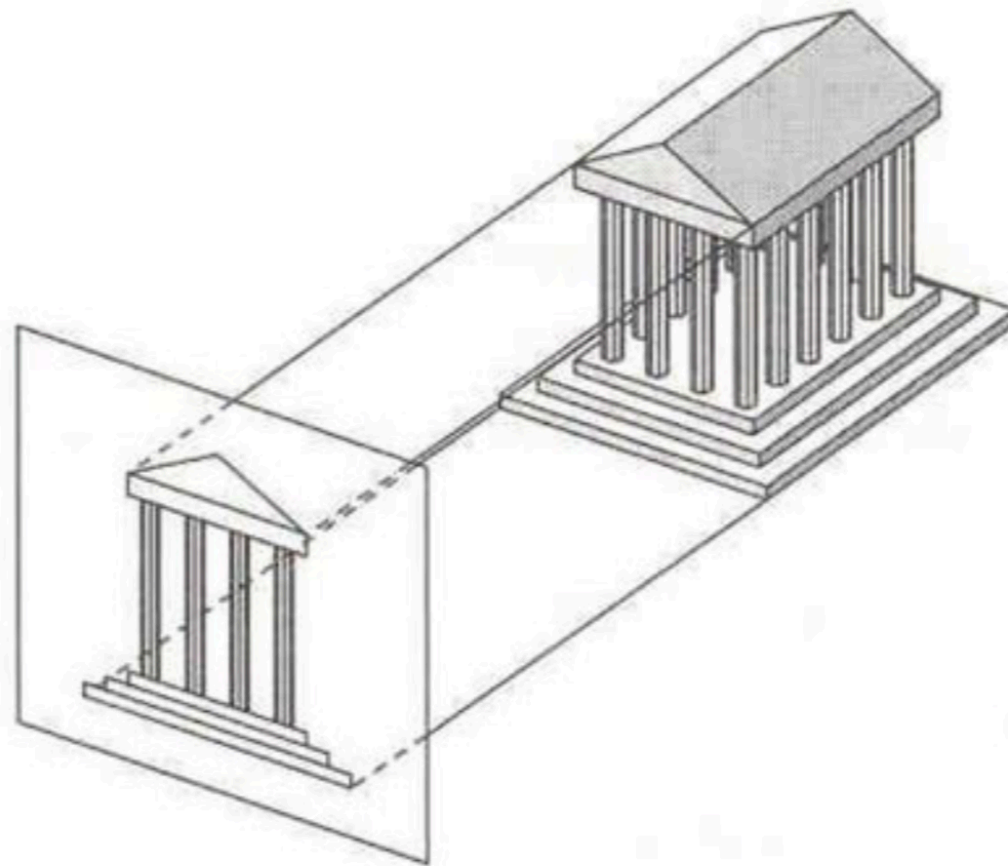
Perspective projection

Two main kinds of observer view projections. First is **Perspective**:



Perspective and Orthographic projections

- **Parallel** (or **Orthographic**): Drop Z world coordinate of objects in sight, like if looking at the world from infinity far away (can't discern depth).
- We will only use an orthographic in this class.



Workflow to display 2D dataset

For 2D, we convert native data units -> pixels (how ever large our visualization app window extent is).

1. Translate (move) the data so the origin becomes 0.

(Data -> View Volume Coordinates)

2. Normalize (scale) data so that each coordinate lies between 0 and 1.

(View Volume Coordinates -> Normalized View Volume Coordinates)

3. Scale coordinates to extent of the app window (pixels). Y-invert because window origin is top-left, not bottom-left. Translate by screen y extent to make things positive in y.

(Normalized View Volume Coordinates -> Screen Coordinates)

Let's work out an example with the following 2D GDP dataset

Country	GDP	Life Expectancy (LE)
Czech Republic	22860	78
United States	41728	79
China	8848	73
Russia	14738	69

Automating the coordinate transform process

- 3 step workflow not too bad, but would be nice if we didn't have to use lots of loops, temporary variables, etc. Especially when things get more cumbersome with 3D viewing.
- We can collapse workflow to a single line of code with linear algebra!
- The idea
 - Represent each data point as a **column vector**.
 - Offload all the coordinate transform work to a single **transformation matrix**.
 - Matrix multiplication does all the work for us!
- In fact, we can transform the data "in-place" with the concept of **homogeneous coordinates**!

Homogenous coordinates (2D)

Given the data point (X, Y) , its homogeneous coordinates are given by (x, y, h) .
Convert between data and homogenous coordinates via the following:

$$x = \frac{X}{h}$$

$$y = \frac{Y}{h}$$

For data points, we always set $h = 1$.

Let's look at an example for the GDP data.

Quick review: Dot product

Given vectors

$$\vec{\alpha} = [\alpha_0, \alpha_1, \alpha_2, \dots, \alpha_{N-1}] \text{ and}$$

$$\vec{\beta} = [\beta_0, \beta_1, \beta_2, \dots, \beta_{N-1}],$$

the dot product yields a single scalar value after multiplying the vectors element-wise then summing:

$$\vec{\alpha} \cdot \vec{\beta} = \sum_{i=0}^{N-1} \alpha_i \beta_i$$

Quick review: Matrix-vector right multiplication

Right-multiplying¹ a matrix by a vector is just N dot products, one with each matrix *row* and the entire column vector:

$$A\vec{b} = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} \vec{a}_0 \cdot \vec{b} \\ \vec{a}_1 \cdot \vec{b} \\ \vec{a}_2 \cdot \vec{b} \end{bmatrix} = \begin{bmatrix} a_{0,0}b_0 + a_{0,1}b_1 + a_{0,2}b_2 \\ a_{1,0}b_0 + a_{1,1}b_1 + a_{1,2}b_2 \\ a_{2,0}b_0 + a_{2,1}b_1 + a_{2,2}b_2 \end{bmatrix}$$

It yields another column vector.

¹ Remember, matrix multiplication does not commute. $A\vec{b} \neq \vec{b}A$

Let's work out how apply the matrix multiplication approach to the GDP example to do the translation and scaling operations.

Transformation matrices

2D Translation matrix with translation vector $\vec{t} = [t_x, t_y]$:

$$\begin{bmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{bmatrix}$$

2D Scaling matrix with scaling vector $\vec{s} = [s_x, s_y]$:

$$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Let's walk through the entire GDP example
with the translation and scaling matrices.

General case: Matrix-matrix multiplication

To transform more than 1 data point, we would use a matrix rather than column vector — you would line up the other data points as columns in the right-hand-side matrix (B below).

Here's a 3x3 example:

$$AB = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{bmatrix} \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{bmatrix} = \begin{bmatrix} a_{0,*} \cdot b_{*,0} + a_{0,*} \cdot b_{*,1} + a_{0,*} \cdot b_{*,2} \\ a_{1,*} \cdot b_{*,0} + a_{1,*} \cdot b_{*,1} + a_{1,*} \cdot b_{*,2} \\ a_{2,*} \cdot b_{*,0} + a_{2,*} \cdot b_{*,1} + a_{2,*} \cdot b_{*,2} \end{bmatrix}$$

- Recall for non-square matrices: If $\dim(A) = R \times N$ and $\dim(B) = N \times C$ then the output matrix has dimensions $R \times C$.
- Recall that matrix multiplication does not commute. $AB \neq BA$.
- **In this class:** The transformation matrix will always be on the left (A) and the data matrix will be on the right (B).