1 Coordinate systems

Coordinate systems are at the heart of visualization techniques. Most of the work involved in generating visualizations is simply transforming data from one coordinate system to another. The starting coordinate system is the data space itself, the native coordinate system of the data. The final coordinate system is the visualization device, for example, a window on a computer screen. It is useful to create a number of intermediate coordinate systems between the initial and final coordinates to make the process of visualization tractable. The following is a fairly standard sequence of coordinate systems used in visualization.

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 within which all of the data resides.

View Volume Coordinates: the volume of the data space the user wants to view. The volume may be defined by the max and min variable values, or it may include only a subset of the data. If the view volume is constrained to be axis-aligned, then it is defined by an origin and an extent in each data variable direction. If the view volume can have arbitrary orientations, then it is defined by an origin and a set of orthonormal axes that define its orientation. The size of the volume is determined by an extent measured along the orthonormal axes.

Normalized Viewing Coordinates: A scaling of the data so that the
visible data points—those within the view volume—fit within the range \([0, 1]\)
in all dimensions.

**Screen Coordinates:** A scaling, possibly a translation, and a projection to
convert the normalized coordinates into screen coordinates where they are
drawn.

### 1.1 Displaying in 2D

To display the data, we need to transform each point from its native coordi-
nates to screen coordinates. This is a 3 step process:

1. Convert to view volume coordinates. Translate the data so the new
   origin becomes zero.

2. Convert to normalized view volume coordinates. Scale the data so that
   each coordinate is between zero and 1.

3. Convert to screen coordinates. The screen coordinates have their origin
   in the upper left corner.

### 1.2 2D Example

Suppose we have this data set:

<table>
<thead>
<tr>
<th>Country</th>
<th>GPD/capita</th>
<th>Life Expectancy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Czech Republic</td>
<td>22860</td>
<td>78</td>
</tr>
<tr>
<td>USA</td>
<td>41728</td>
<td>79</td>
</tr>
<tr>
<td>China</td>
<td>8848</td>
<td>73</td>
</tr>
<tr>
<td>Russia</td>
<td>14738</td>
<td>69</td>
</tr>
</tbody>
</table>

We will display two dimensions of the information in the x-y-plane. Suppose
the area of the screen is 500 pixels high and 300 pixels wide. Let’s make
the x-axis represent GDB and the y-axis represent life expectancy. Since we
want to see all the data, the range in data coordinates is \([8848,41728]\) for x
and \([69,79]\) for y. The extent in x is 41728-8848=32880 and the extent in y
is 79-69=10.
To display the data, we convert each data point from data coordinates to screen coordinates.

1. Convert to the view volume coordinates. Our first data point is (22860, 78) in data coordinates. To convert to view coordinate, we need to move (translate) the data so that the minimum values of each range map to the origin. In other words, we need to subtract the min values. (22860-8848, 78-69) = (14012, 9) is the point in view coordinates.

2. Convert to normalized view volume coordinates. We need to make sure all values are between 0 and 1. The min got mapped to 0 and the max gets mapped to 1. To do that, we need to scale the coordinates by the extent. I.e. we need to scale the x-coordinate by 32880 and the y-coordinate by 10. Our first data point is then (14012/32880, 9/10) = (0.42616, 0.9) in normalized view coordinates.

3. Convert to screen coordinates. First we need to scale the data - we stretch it and flip its y-direction. So the point becomes (0.42616*300, 0.9*(-500)) = (127.8467, -450). Then, we need to translate it so that the origin is in the upper left corner: (213.08+0, -450+500) = (127.8467, 50).

Here are all of the coordinates in all of the coordinate systems for the example (with GDP/capita abbreviated to GDP and Life Expectancy abbreviated to LE). vc indicates view coordinates, nvc indicates normalized view coordinates, and sc indicates screen coordinates.

<table>
<thead>
<tr>
<th>Country</th>
<th>GDP</th>
<th>LE</th>
<th>GDP(vc)</th>
<th>LE(vc)</th>
<th>GDP(nvc)</th>
<th>LE(nvc)</th>
<th>GDP(sc)</th>
<th>LE(sc)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Czech Republic</td>
<td>22860</td>
<td>78</td>
<td>14012</td>
<td>9</td>
<td>0.4262</td>
<td>0.9</td>
<td>127.8467</td>
<td>50</td>
</tr>
<tr>
<td>USA</td>
<td>41728</td>
<td>79</td>
<td>32880</td>
<td>10</td>
<td>1</td>
<td>1</td>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>China</td>
<td>8848</td>
<td>73</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0.4</td>
<td>0</td>
<td>300</td>
</tr>
<tr>
<td>Russia</td>
<td>14738</td>
<td>69</td>
<td>5890</td>
<td>0</td>
<td>0.1791</td>
<td>0</td>
<td>53.7409</td>
<td>50</td>
</tr>
</tbody>
</table>

1.3 Formalizing the coordinate transformation process

In the 2D case, it is relatively straight-forward to transform the coordinates - it requires only scaling and translation. But once we get to 3D, there will be rotation and projection as well. To perform these coordinate transformations, we can use some tools from linear algebra. In particular each coordinate can
be represented as a vector and each transformation accomplished by a matrix multiplication.

1.3.1 Homogeneous Coordinates

First, let’s formalize the representation of the coordinates of a data point. We will use homogeneous coordinates.

Homogeneous coordinates are the basis for building a generic manipulation system for rigid objects. The homogeneous coordinates for a 2-D Cartesian point \((x_c, y_c)\) are \((x, y, h)\). The relationship between the homogeneous and standard coordinates is given in (1.3.1).

\[
x_c = \frac{x}{h} \\
y_c = \frac{y}{h}
\]

Example

Let’s represent the USA’s data in homogeneous coordinates.

\[
\begin{pmatrix}
41728 \\
79 \\
1
\end{pmatrix}
\]

3-D

A 3-D Cartesian point \((x_c, y_c, z_c)\) has the homogeneous representation \((x, y, z, h)\), and the normalized form is the same as the 2-D case, just extended to the z-axis.

Note that homogeneous coordinates permit many representations of a single 2-D Cartesian point. In the standard, or normalized form of homogeneous coordinates \(h = 1\). In this course, we will always maintain homogeneous coordinates in their normalized form. Using homogeneous coordinates lets us implement transformations such as translation, rotation, and scaling using a single matrix representation.
1.3.2 Matrix-Vector Multiplication

Now we turn to matrix multiplication. Each transformation will happen when we multiple the coordinate vector by a particular matrix.

**Dot Product** Matrix multiplication is built upon the dot product, also called the scalar product or inner product, of two vectors. Given two N-element vectors $\vec{\alpha} = [\alpha_0 \alpha_1 \ldots \alpha_{N-1}]$ and $\vec{\beta} = [\beta_0 \beta_1 \ldots \beta_{N-1}]$ their scalar product is defined as the multiplication of corresponding elements in the two vectors, followed by the summation of the products. The dot product of two vectors is always a single scalar value.

$$d(\vec{\alpha}, \vec{\beta}) = \sum_{i=0}^{N-1} \alpha_i \beta_i$$

**Matrix-Vector multiplication is a bunch of dot products** To multiply a matrix by a vector, we perform a dot product between each row of the matrix and the vector like this:

$$A\vec{b} = \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} d(a_{0,*}, \vec{b}) \\ d(a_{1,*}, \vec{b}) \\ d(a_{2,*}, \vec{b}) \end{pmatrix} = \begin{pmatrix} a_{0,0}b_0 + a_{0,1}b_1 + a_{0,2}b_2 \\ a_{1,0}b_0 + a_{1,1}b_1 + a_{1,2}b_2 \\ a_{2,0}b_0 + a_{2,1}b_1 + a_{2,2}b_2 \end{pmatrix}$$

**Translation (2-D Example)**

Let’s start with a small example.

$$\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 41728 \\ 79 \\ 1 \end{pmatrix}$$

The result is

$$\begin{pmatrix} 1 * 41728 + 0 * 79 + 3 * 1 \\ 0 * 41728 + 1 * 79 + -1 * 1 \\ 0 * 41728 + 0 * 79 + 1 * 1 \end{pmatrix} = \begin{pmatrix} 41731 \\ 78 \\ 1 \end{pmatrix}$$
The effect of the above matrix is to translate the data - it translates it in x by 3 and in y by -1. How would you get it to translate by different amounts? Note that we can’t translate in the final dimension - the homogeneous coordinate is playing a special role - it is only because we have the extra coordinate that we can do the translation in the other two).

In class, we update this matrix to do the translation we needed for our GDP example. It is

$$\begin{pmatrix}
1 & 0 & -8848 \\
0 & 1 & -69 \\
0 & 0 & 1
\end{pmatrix}$$

Translation (3-D Example)

Here is a similar example in 3 dimensions.

$$\begin{pmatrix}
1 & 0 & 0 & 3 \\
0 & 1 & 0 & -1 \\
0 & 0 & 1 & 2 \\
0 & 0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
41728 \\
79 \\
3 \\
1
\end{pmatrix}$$

The result is

$$\begin{pmatrix}
1 \times 41728 + 0 \times 79 + 0 \times 3 + 3 \times 1 \\
0 \times 41728 + 1 \times 79 + 0 \times 3 + -1 \times 1 \\
0 \times 41728 + 0 \times 79 + 1 \times 3 + 2 \times 1 \\
0 \times 41728 + 0 \times 79 + 0 \times 3 + 1 \times 1
\end{pmatrix} = \begin{pmatrix}
41731 \\
78 \\
5 \\
1
\end{pmatrix}$$

Scaling (2-D Example)

What would happen if we multiplied by

$$\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}$$

Nothing. This is the identity matrix (because it is the multiplicative identity in matrix multiplication).
Now let’s construct a new matrix. How would we scale the data in x by 2 and y by 3?

Multiply it by

\[
\begin{pmatrix}
2 & 0 & 0 \\
0 & 3 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

In class, we develop the matrix that performs the scaling we need to translate the GDP example from view coordinates to normalized view coordinates. It is

\[
\begin{pmatrix}
\frac{1}{32880} & 0 & 0 \\
0 & \frac{1}{10} & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

**Scaling (3-D Example)**

What would happen if we multiplied by

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

? Nothing. This is the identity matrix (because it is the multiplicative identity in matrix multiplication).

Now let’s construct a new matrix. How would we scale the data in x by 2 and y by 3?

Multiply it by

\[
\begin{pmatrix}
2 & 0 & 0 & 0 \\
0 & 3 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

In class, we developed the matrix that performed the scaling we need to translate the GDP example from view coordinates to normalized view coordinates.
A general form for scaling and translation matrices

I am going to write the formal system for 3-D coordinates. To have the 2-D version, simply remove the 3rd row and 3rd column.

We can perform multiple coordinate transformations by multiplying by multiple matrices. These operations are applied from right to left. If I wanted to translate and then scale a 3-D system, I would perform this set of multiplications:

\[
\begin{pmatrix}
s_x & 0 & 0 & 0 \\
0 & s_y & 0 & 0 \\
0 & 0 & s_z & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 & t_x \\
0 & 1 & 0 & t_y \\
0 & 0 & 1 & t_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
1
\end{pmatrix}
\]

where \(s_x\), \(s_y\), and \(s_z\) are the scale factors and \(t_x\), \(t_y\), and \(t_z\) are the translation amounts. Although the math can be performed from left to right (it is possible to multiply the matrices together and matrix multiplication is transitive), I prefer to do the math from right to left (first translate, then scale).

Note that matrix-multiplication is not commutative – order matters!

Completing our 2D GDP example

Above, we spoke about the process in 4 steps (OK, it was 3 steps, but the third step is really two):

1. Convert to the view volume coordinates. This is a translation. We developed this above. Here it is again.

\[
\begin{pmatrix}
1 & 0 & -8848 \\
0 & 1 & -69 \\
0 & 0 & 1
\end{pmatrix}
\]

2. Convert to normalized view volume coordinates. This is a scaling we developed above. Here it is again.

\[
\begin{pmatrix}
1/32880 & 0 & 0 \\
0 & 1/10 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
3. Convert to screen coordinates.

(a) Scale and reverse the direction for the y-coordinate. (A flipping will be just a negative scaling).

\[
\begin{pmatrix}
300 & 0 & 0 \\
0 & -500 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

(b) Translate so that the origin is in the upper left corner

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 500 \\
0 & 0 & 1
\end{pmatrix}
\]

Line them all up to convert the data for the US to screen coordinates like this:

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 500 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
300 & 0 & 0 \\
0 & -500 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1/32880 & 0 & 0 \\
0 & 1/10 & 0 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & -8848 \\
0 & 1 & -69 \\
0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
41728 \\
79 \\
1
\end{pmatrix}
\]

1.3.3 Matrix-Matrix Multiplication

You can represent a set of coordinates as a matrix - instead of 1 column you can have C columns for C data points. Then you can multiple the transformation matrix by the coordinate matrix and the result will be a matrix of transformed coordinates, with each column containing the coordinates for a point.

In matrix-matrix multiplication, the dot product uses the corresponding row in the first matrix and the corresponding column in the second. For example, consider two 3x3 matrices given below and their product written as a set of dot products.

\[
\begin{bmatrix}
\begin{pmatrix} d(a_{0,*},b_{*,0}) & d(a_{0,*},b_{*,1}) & d(a_{0,*},b_{*,2}) \\ d(a_{1,*},b_{*,0}) & d(a_{1,*},b_{*,1}) & d(a_{1,*},b_{*,2}) \\ d(a_{2,*},b_{*,0}) & d(a_{2,*},b_{*,1}) & d(a_{2,*},b_{*,2}) \end{pmatrix} \\
\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} \\ a_{1,0} & a_{1,1} & a_{1,2} \\ a_{2,0} & a_{2,1} & a_{2,2} \end{pmatrix} \\
\begin{pmatrix} b_{0,0} & b_{0,1} & b_{0,2} \\ b_{1,0} & b_{1,1} & b_{1,2} \\ b_{2,0} & b_{2,1} & b_{2,2} \end{pmatrix}
\end{bmatrix}
\]
The same rule applies for matrices that are not square. The only requirement for matrix multiplication to be valid is that the number of columns in the first matrix must match the number of rows in the second matrix. If the input matrices are RxN and NxC, then the output matrix will be RxC. Matrix multiplication is not commutative, in general.

For our purposes, the most important case of matrix multiplication with non-square matrices is the case of multiplying a matrix and a vector.

\[
\begin{bmatrix}
  d(a_{0,*}, b_{*,0}) \\
  d(a_{1,*}, b_{*,0}) \\
  d(a_{2,*}, b_{*,0})
\end{bmatrix}
= \begin{bmatrix}
  a_{0,0} & a_{0,1} & a_{0,2} \\
  a_{1,0} & a_{1,1} & a_{1,2} \\
  a_{2,0} & a_{2,1} & a_{2,2}
\end{bmatrix}
\begin{bmatrix}
  b_{0,0} \\
  b_{1,0} \\
  b_{2,0}
\end{bmatrix}
\]  

(2)

If this is new to you, you might want to check out the tutorial at

http://www.zweigmedia.com/ThirdEdSite/tutorialsf1/frames3_2.html

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