**Computer Graphics**

**Homogeneous Coordinates**

If we want to maintain points and vectors as two different geometric objects, we should avoid the problem of representing a point and a vector by a $2\times 1$ matrix. This will lead to confusion and difficulties when we want to implement change of frames (or change of coordinates with an origin), because this transformation is not obtained by multiplication of matrices.

We use homogeneous coordinates, i.e. a $3\times 1$ matrix. In the frame specified by $\mathbf{u}_1$, $\mathbf{u}_2$ and by the point $P_0$ we can specify any point by

$$P = P_0 + \beta_1 \mathbf{u}_1 + \beta_2 \mathbf{u}_2$$

If we define multiplication of a scalar by a point in the form: $0.P = 0$ and $1.P = P$, we can write

$$P = (\beta_1, \beta_2, 1) (\mathbf{u}_1, \mathbf{u}_2, P_0)^T.$$  This means that we can associate $P$ with a column matrix

$$\mathbf{p} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ 1 \end{bmatrix}$$

In the same frame a vector $\mathbf{w} = \eta_1 \mathbf{u}_1 + \eta_2 \mathbf{u}_2$ can be expressed as

$$\mathbf{w} = (\eta_1, \eta_2, 0) (\mathbf{u}_1, \mathbf{u}_2, P_0)^T.$$  Then we can associate the vector $\mathbf{w}$ with a column matrix

$$\mathbf{a} = \begin{bmatrix} \eta_1 \\ \eta_2 \\ 0 \end{bmatrix}$$

We can implement a change of frames as a multiplication of matrices, in fact we have that if $\{\mathbf{u}_1, \mathbf{u}_2, P_0\}$ is a frame and $\{\mathbf{v}_1, \mathbf{v}_2, Q_0\}$ is another one, then

$$\mathbf{v}_1 = \gamma_{11} \mathbf{u}_1 + \gamma_{12} \mathbf{u}_2$$

$$\mathbf{v}_2 = \gamma_{21} \mathbf{u}_1 + \gamma_{22} \mathbf{u}_2$$

$$Q_0 = \gamma_{31} \mathbf{u}_1 + \gamma_{32} \mathbf{u}_2 + P_0$$
Equations that can be written in the form

\[
\begin{bmatrix}
    v_1 \\
    v_2 \\
    Q_0
\end{bmatrix}
= R
\begin{bmatrix}
    u_1 \\
    u_2 \\
    P_0
\end{bmatrix},
\]

where \( R \) is the matrix representation of the change of frames.

As before, we can compute the changes in the representations using the same ideas that we used in the case of change of coordinate systems:

\[
b^T v_2 = b^T R u_2 = a^T u_2, \quad \text{hence } a = R^T b, \quad \text{or } b = (R^T)^{-1} a
\]

Remark: All affine transformations, i.e., transformations preserving straight lines, can be represented by multiplication of matrices when using homogeneous coordinates.

Example of change of frames:

We use the same transformation of coordinates that we did previously, i.e.,

\[
v_1 = u_1 + u_2; \quad v_2 = u_1;
\]

But the point \( Q_0 = P_0 + 3u_1 + 2u_2 \). So the matrix \( R \) is given by

\[
R = \begin{bmatrix}
1 & 1 & 0 \\
1 & 0 & 0 \\
3 & 2 & 1
\end{bmatrix}
\]

Knowing the matrix \( R \), we want to find the representation \( p' \) of the point \( P \) in the new frame, if his representation was \( p = (3, 0, 1)^T \) in the original frame, We also want to find the representation \( b \) of the vector \( w \) in the new frame, if his representation in the old one was \( a = (3, 2, 0)^T \).

Since \( b = (R^T)^{-1} a \), we need to find the inverse of \( R^T \). In fact

\[
(R^T)^{-1} = \begin{bmatrix}
0 & 1 & -2 \\
1 & -1 & -1 \\
0 & 0 & 1
\end{bmatrix}
\]

and therefore we can write, just by multiplying the matrices, that

\[
p' = (-2, 2, 1)^T \quad \text{and that } b = (2, 1, 0)^T.
\]
Remark: Using homogeneous coordinates we lose uniqueness, i.e., a point $P$ could have different representations with respect to the same frame. The nice thing is that all differ by a multiplying constant. Therefore in homogeneous coordinates, the following representations of the point $P$, although they look different, they are the same:

$$ p = \lambda (\alpha_1, \alpha_2, 1)^T $$

for any value of $\lambda \neq 0$. In order to recover the real point, we need to find a lambda such that the third coordinate is a 1. If we are dealing with a point, not a vector, there is always such a lambda.

Another disadvantage of homogeneous coordinates is that we cannot define the cross product of vectors in 3D using their homogeneous coordinates. We have to step down into 3D coordinates.

**Affine Transformations**

A transformation is a function that takes a point (or vector) and maps that point (or vector) into another point (or vector).

We use homogeneous coordinates for the purpose of representing both points and vectors as three-dimensional column matrices (if we are in 2D) or as four-dimensional column matrices, in 3D. These matrices will be called vertices.

We restrict ourselves to affine transformation, i.e., a combination of a linear transformations and translations.

A linear transformation $f$ is such that verify that for any scalars $\gamma$ and $\delta$, and any vertices $r$ and $s$

$$ f(\gamma r + \delta s) = \gamma f(r) + \delta f(s) $$

They are affine transformations, and they map straight lines into straight lines. This means, that when we transform a straight line, we need to find the transformation of two vertices, and then find the line that goes through them in the transformed space.

We set up an affine transformation in 2D, when we define a $3 \times 3$ matrix $T$, and we applied it to the representation $d$ of a vertex to obtain the new representation $g$, i.e.

$$ g = Td $$

The matrix $T$ is of a particular type, due to the way we have represented vectors and points as vertices. In fact $T$ has 9 free parameters, since it needs to be of the form
to transform points into points, and vectors into vectors.

Note that if we have the line $P(\delta) = P_0 + \delta \mathbf{b}$, that can be expressed with respect to a frame as

$p(\delta) = p_0 + \delta \mathbf{b}$, we have the value of the transformation under the affine transformation defined by $\mathbf{T}$ is

$\mathbf{T} p(\delta) = \mathbf{T} p_0 + \delta \mathbf{T} \mathbf{b}$ that is also a straight line.

**Basic affine transformations:**

**Translation:** A transformation that displaces points by a fixed vector. It applies only to points

$P' = P + \mathbf{b}; \quad x' = x + \beta_1; \quad y' = y + \beta_2$

Therefore using homogeneous coordinates we can write

$$
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & \beta_1 \\
0 & 1 & \beta_2 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
$$

**Rotation about the origin with an angle $\theta$:** This transformation maps a point into another one, whose distance to the origin is the same, but the angle of the line going through the origin and the point is incremented by $\theta$ in a counterclockwise way. Using polar coordinates we can write, using $(x, y)^T$ for the original point and $(x', y')^T$ for the transformed one.

$$
\begin{align*}
x &= \rho \cos \varphi; \quad y = \rho \sin \varphi \\
x' &= \rho \cos (\varphi + \theta); \quad y' = \rho \sin(\varphi + \theta)
\end{align*}
$$

See the following figure:
Expanding the sine and cosine of the sum of two angles, we have

\[ x' = \rho \cos \phi \cos \theta - \rho \sin \phi \sin \theta = x \cos \theta - y \sin \theta \]

\[ y' = \rho \sin \phi \cos \theta + \rho \cos \phi \sin \theta = y \cos \theta + x \sin \theta \]

and this can be written in matrix form as

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  \cos \theta & -\sin \theta \\
  \sin \theta & \cos \theta
\end{bmatrix}
\begin{bmatrix}
  x \\
  y
\end{bmatrix}
\]

Using homogeneous coordinates we can write

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  \cos \theta & -\sin \theta & 0 \\
  \sin \theta & \cos \theta & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Translation and rotations are called rigid-body transformations because they don’t alter the shape of an object. They simply change the location and orientation of an object.

Scaling: It alters the size of an object. This alteration can have different measures in the x and y directions.

\[ x' = \lambda_x x \]

\[ y' = \lambda_y y \]

In other words

\[
\begin{bmatrix}
  x' \\
  y'
\end{bmatrix}
= \begin{bmatrix}
  \lambda_x & 0 & 0 \\
  0 & \lambda_y & 0 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]
If the values of $\lambda_i$, $i = x, y$, are negatives, we obtain a reflection in the corresponding scaling direction.

**Concatenation of Transformations:**

Rotation of angle $\theta$ about a pivot point of coordinates $(x_F, y_F)^T$.

We don’t know how to rotate around an arbitrary point, but we know how to rotate about the origin. So we follow these steps:

1) Bring the pivot point to the origin (a translation)
2) Rotate the angle $\theta$ about the origin (a rotation)
3) Translate the origin back to the pivot (a translation)

Therefore the transformation will be given by the matrix obtained by multiplying the matrices corresponding to these transformations, i.e.

\[
\begin{bmatrix}
1 & 0 & x_F \\
0 & 1 & y_F \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
1 & 0 & -x_F \\
0 & 1 & -y_F \\
0 & 0 & 1
\end{bmatrix}
\]

**Reflection along a line:**

a) First we consider the reflection with respect to the x-axis. The y-coordinate changes sign. Therefore the matrix is

\[
R_x = \begin{pmatrix}
1 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

b) Now we consider the reflection with respect to a line through the origin, with slope $m$.

We need to bring back this reflection to the previous one. So we
1) Rotate the line until it coincides with the x-axis
2) Make the reflection with respect to the x-axis
3) Bring back the line to its original position, using the inverse rotation

Therefore we need the following composition of transformations (given by the product of matrices):
Now we need to compute the angle $\theta$. But we know the slope $m = \tan \theta$.

Therefore, using simple trigonometric equalities like $\tan \theta = \frac{\sin \theta}{\cos \theta}$ and $\cos^2 \theta + \sin^2 \theta = 1$, we obtain that $\sin \theta = \frac{m}{\sqrt{m^2 + 1}}$ and $\cos \theta = \frac{1}{\sqrt{m^2 + 1}}$.

c) Now we consider the reflection with respect to a line that does not go through the origin, but with slope $m$.

Here the steps are the following:
1) Bring any point on the line to the origin.
2) Rotate the line to coincide with the x-axis
3) Make the reflection with respect to the x-axis
4) Rotate the line back
5) Translate the point back

In other words, our matrix for reflection is now

$$R_m = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
1 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
\cos \theta & \sin \theta & 0 \\
-\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix}.$$

Examples:

1) Find the reflection matrix with respect to the straight line $y = 0.8x + 2$

Answer: $R_{tm} = \begin{bmatrix}
9 & 40 & -80 \\
41 & 41 & 41 \\
41 & -9 & 100 \\
41 & 41 & 41 \\
0 & 0 & 1
\end{bmatrix}$

2) Find the matrix of the transformation corresponding to a change of coordinates, whose new center will be the point $(3, -2)$ and the view-up vector (the new y-axis) will be in the direction given by the vector $(2, 1)$. Find the coordinates of the rectangle $(-2, -1), (-1, -1), (-2, -3), (-1, -3)$ under this new frame system. (Solution later on).

Use homogeneous coordinates for both problems as a tool. Then go back to Cartesian coordinates.
Viewing pipeline

We have a window in the real world, that is the area selected for our display \((W_L, W_R, W_B, W_T)\). We also have a viewport, which is a display device area onto which the real world window is mapped \((V_L, V_R, V_B, V_T)\). Going from the window to the viewport is called a viewing transformation.

We will use an intermediate step, to make the transformation device independent. We introduce the Normalized Device Coordinates space. It consists of a region of the xy-plane defined by \(0 \leq x \leq 1\), and \(0 \leq y \leq 0.75\). Then we define the viewport on the NDC space, instead of on the screen.

The window and the viewport should be homothetic, to avoid distorting the image. First we transform the window into the NDC viewport. In order to perform this transformation, we break it into several steps:

1) Translate \((W_L, W_B)\) to the origin
2) Scale the window to become the square of side 1
3) Scale square to become the viewport
4) Translate the origin to \((V_L, V_B)\)

These are transformations that we have already seen:

\[
1) \quad T = \begin{bmatrix} 1 & 0 & -W_L \\ 0 & 1 & -W_B \\ 0 & 0 & 1 \end{bmatrix} \\
2) \quad S = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{W_R - W_L} & 0 & 0 \\ 0 & \frac{1}{W_T - W_B} & 0 \end{bmatrix} \quad ; \quad 3) \quad S' = \begin{bmatrix} 0 & 0 & 0 \\ V_R - V_L & 0 & 0 \\ 0 & V_T - V_B & 0 \end{bmatrix} \\
4) \quad T' = \begin{bmatrix} 1 & 0 & V_L \\ 0 & 1 & V_B \\ 0 & 0 & 1 \end{bmatrix}
\]

The concatenation of all these matrices gives us

\[
C = \begin{bmatrix} \frac{V_R - V_L}{W_R - W_L} & 0 & -W_L(V_R - V_L) + V_L \\ \frac{V_T - V_B}{W_T - W_B} & 0 & -W_B(V_T - V_B) + V_B \\ 0 & \frac{1}{W_T - W_B} & 1 \end{bmatrix}
\]
and therefore \( x_{\text{NDC}} = x \frac{V_R - V_L}{W_R - W_L} + \frac{W_L V_L - W_R V_B}{W_R - W_L} \)

\[ y_{\text{NDC}} = y \frac{V_T - V_B}{W_T - W_B} + \frac{W_B V_B - W_T V_T}{W_T - W_B} \]

Similarly for the transformation from NDC to the display device, we can write the following simple transformations:

1) Scale the NDC rectangle into a square
2) Scale the square into the screen, with resolution \( \text{MaxX} \times \text{MaxY} \).

1) \( S_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 4/3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \); 2) \( S_1' = \begin{pmatrix} \text{MaxX} & 0 & 0 \\ 0 & \text{MaxY} & 0 \\ 0 & 0 & 1 \end{pmatrix} \)

But the product of these two transformations will show the pictures upside down. So we change the coordinate system on the screen:

3) Origin go down to \((0, \text{MaxY})\)
4) Make a reflection with respect to the \(x\)-axis

3) \( T_1' = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & -\text{MaxY} \\ 0 & 0 & 1 \end{pmatrix} \); 4) \( R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \)

The concatenation of all these transformations is given by the matrix

\[ C' = \begin{pmatrix} \text{MaxX} & 0 & 0 \\ 0 & -\frac{4\text{MaxY}}{3} & \text{MaxY} \\ 0 & 0 & 1 \end{pmatrix} \]

and therefore we can write:

\( x = \text{round} \left( \text{MaxX} \ x_{\text{NDC}} \right) \)

\( y = \text{round} \left( -\frac{4}{3} \text{MaxY} \ y_{\text{NDC}} + \text{MaxY} \right) \)

where \((x,y)\) are the pixel coordinates on the screen.