The term $a$ in the shear matrix is the proportionality constant. Notice that the product $S H_x [x \ y \ 1]^T$ is $[x + ay \ y \ 1]^T$, clearly demonstrating the proportional change in $x$ as a function of $y$.

Similarly, the matrix
\[
S H_y = \begin{bmatrix}
1 & 0 & 0 \\
b & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\] (5.29)

shears along the $y$ axis.

## 5.4 COMPOSITION OF 2D TRANSFORMATIONS

The idea of composition was introduced in Section 5.3. Here, we use composition to combine the fundamental $R$, $S$, and $T$ matrices to produce desired general results. The basic purpose of composing transformations is to gain efficiency by applying a single composed transformation to a point, rather than applying a series of transformations, one after the other.

Consider the rotation of an object about some arbitrary point $P_1$. Because we know how to rotate only about the origin, we convert our original (difficult) problem into three separate (easy) problems. Thus, to rotate about $P_1$, we need a sequence of three fundamental transformations:

1. Translate such that $P_1$ is at the origin.
2. Rotate.
3. Translate such that the point at the origin returns to $P_1$.

This sequence is illustrated in Fig. 5.10, in which our house is rotated about $P_1(x_1, y_1)$. The first translation is by $(-x_1, -y_1)$, whereas the later translation is by the inverse $(x_1, y_1)$. The result is rather different from that of applying just the rotation.

![Figure 5.10](image_url)  
**Figure 5.10** Rotation of a house about the point $P_1$ by an angle $\theta$.  

The net transformation is

\[
T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1) = \begin{bmatrix}
1 & 0 & x_1 \\
0 & 1 & y_1 \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
1 & 0 & -x_1 \\
0 & 1 & -y_1 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
\cos \theta & -\sin \theta & x_1(1 - \cos \theta) + y_1\sin \theta \\
\sin \theta & \cos \theta & y_1(1 - \cos \theta) - x_1\sin \theta \\
0 & 0 & 1
\end{bmatrix}.
\]  

(5.30)

A similar approach is used to scale an object about an arbitrary point \( P_1 \). First, translate such that \( P_1 \) goes to the origin, then scale, then translate back to \( P_1 \). In this case, the net transformation is

\[
T(x_1, y_1) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1) = \begin{bmatrix}
1 & 0 & x_1 \\
0 & 1 & y_1 \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
s_x & 0 & 0 \\
0 & s_y & 0 \\
0 & 0 & 1
\end{bmatrix} \cdot \begin{bmatrix}
1 & 0 & -x_1 \\
0 & 1 & -y_1 \\
0 & 0 & 1
\end{bmatrix}
\]

\[
= \begin{bmatrix}
s_x & 0 & x_1(1 - s_x) \\
0 & s_y & y_1(1 - s_y) \\
0 & 0 & 1
\end{bmatrix}.
\]  

(5.31)

Suppose that we wish to scale, rotate, and position the house shown in Fig. 5.11 with \( P_1 \) as the center for the rotation and scaling. The sequence is to translate \( P_1 \) to the origin, to perform the scaling and rotation, and then to translate from the origin to the new position \( P_2 \) where the house is to be placed. A data structure that records this transformation might contain the scale factor(s), rotation angle, and translation amounts, and the order in which the transformations were applied, or it might simply record the composite transformation matrix:

\[
T(x_2, y_2) \cdot R(\theta) \cdot S(s_x, s_y) \cdot T(-x_1, -y_1)
\]  

(5.32)

If \( M_1 \) and \( M_2 \) each represent a fundamental translation, scaling, or rotation, when is \( M_1 \cdot M_2 = M_2 \cdot M_1 \)? That is, when do \( M_1 \) and \( M_2 \) commute? In general, of

---

**Figure 5.11** Rotation of a house about the point \( P_1 \), and placement such that what was at \( P_1 \) is at \( P_2 \).
course, matrix multiplication is not commutative. However, it is easy to show that, in the following special cases, commutativity holds:

\[
M_1 \\
\text{Translate} \\
\text{Scale} \\
\text{Rotate} \\
\text{Scale (with } s_x = s_y) \\
M_2 \\
\text{Translate} \\
\text{Scale} \\
\text{Rotate} \\
\text{Rotate}
\]

In these cases, we do not have to be concerned about the order of matrix composition.

5.5 THE WINDOW-TO-VIEWPORT TRANSFORMATION

Some graphics packages allow the programmer to specify output primitive coordinates in a floating-point world-coordinate system, using whatever units are meaningful to the application program: angstroms, microns, meters, miles, light-years, and so on. The term world is used because the application program is representing a world that is being interactively created or displayed to the user.

Given that output primitives are specified in world coordinates, the graphics subroutine package must be told how to map world coordinates onto screen coordinates (we use the specific term screen coordinates to relate this discussion specifically to SRGP, but hardcopy output devices might be used, in which case the term device coordinates would be more appropriate). We could do this mapping by having the application programmer provide the graphics package with a transformation matrix to effect the mapping. Another way is to have the application programmer specify a rectangular region in world coordinates, called the world-coordinate window, and a corresponding rectangular region in screen coordinates, called the viewport, into which the world-coordinate window is to be mapped. The transformation that maps the window into the viewport is applied to all of the output primitives in world coordinates, thus mapping them into screen coordinates. Figure 5.12 shows this concept. As you can see in this figure, if the window and viewport do not have the same height-to-width ratio, a nonuniform scaling occurs. If the application program changes the window or viewport, then new output primitives drawn onto the screen will be affected by the change. Existing output primitives are not affected by such a change.

The modifier world-coordinate is used with window to emphasize that we are not discussing a window-manager window, which is a different and more recent concept, and which unfortunately has the same name. Whenever there is no ambiguity as to which type of window is meant, we shall drop the modifier.

If SRGP were to provide world-coordinate output primitives, the viewport would be on the current canvas, which defaults to canvas 0, the screen. The application program would be able to change the window or the viewport at any time, in which case subsequently specified output primitives would be subjected to a new