3 Transformations

We have seen how to define simple curves, however, these canonical curves or *primitives* are not by themselves sufficiently rich to describe the complex shapes of everyday objects. We could attempt to find the particular parametric equations that define the shape we are modeling, but this can become cumbersome very quickly, and if the shape changes then we would require different equations.
Fortunately, there is a powerful and simple way to increase the expressive power of our primitive shapes so that they can describe objects of arbitrary complexity: **Transformations**. Simply stated, we will define a family of operations we can apply to our geometric primitives, such as changing their size, rotating them, or moving them around. We will then use combinations of transformations in order to give our primitive curves the shape and location we want in our scene. In what follows, we look at how these transformations are handled mathematically. Initially we will look at the 2D case, but in the next Section we will see that generalizing this process to 3D objects and surfaces is straightforward.
Given a small number of graphical primitives, we can use transformations to build shapes of arbitrary complexity easily.

3.1 2D Transformations

Given a point cloud, polygon, or sampled parametric curve, we can use transformations for several purposes:

1. Change coordinate frames (world, window, viewport, device, etc).
2. Compose objects of simple parts with local scale/position/orientation of one part defined with regard to other parts. For example, for articulated objects.
3. Use deformation to create new shapes.
4. Useful for animation.

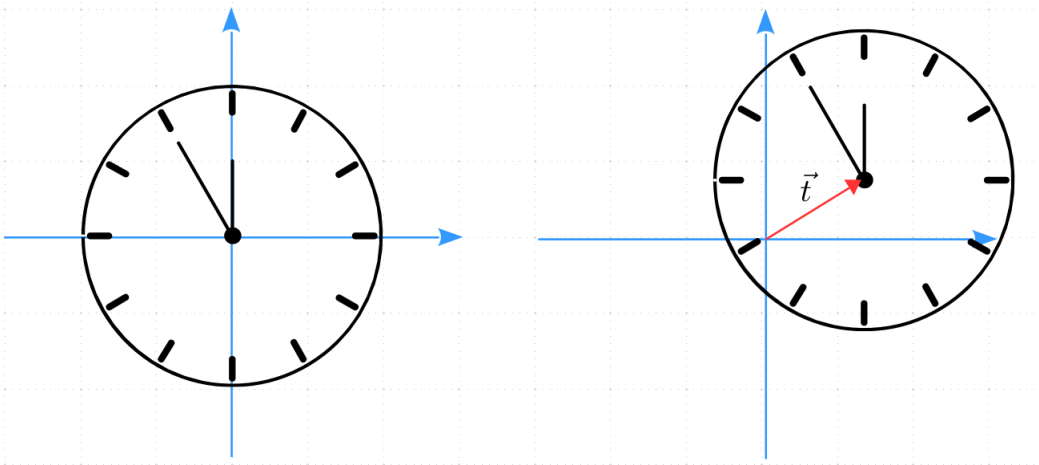
There are three basic classes of transformations:

1. **Rigid body** - Preserves distance, angles, and orientation.
    - Examples: translation and rotation.
2. **Conformal** - Preserves angles and orientation.
    - Examples: translation, rotation, and uniform scaling.
3. **Affine** - Preserves parallelism. Lines remain lines.

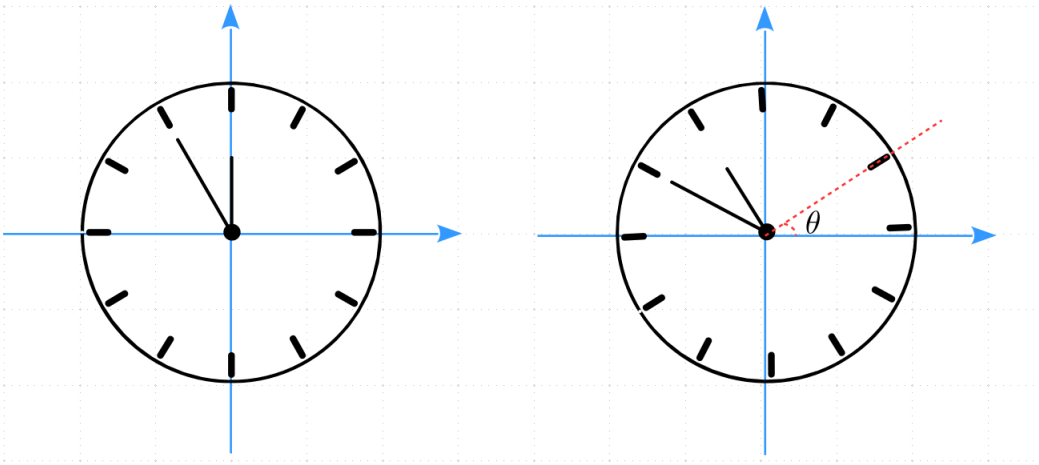
- Examples: translation, rotation, scaling, shear, and reflection.

Examples of transformations:

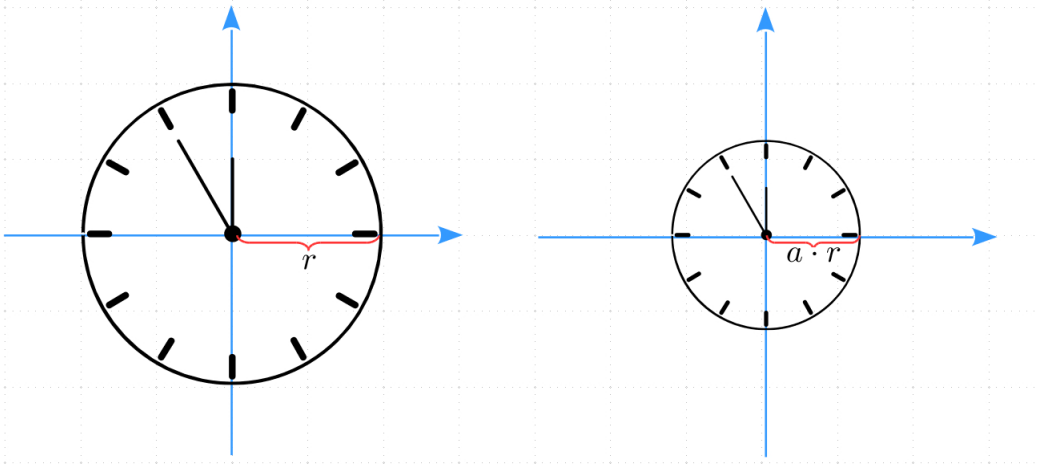
- **Translation** by vector $\vec{t}$: $\bar{p}_1 = \bar{p}_0 + \vec{t}$.



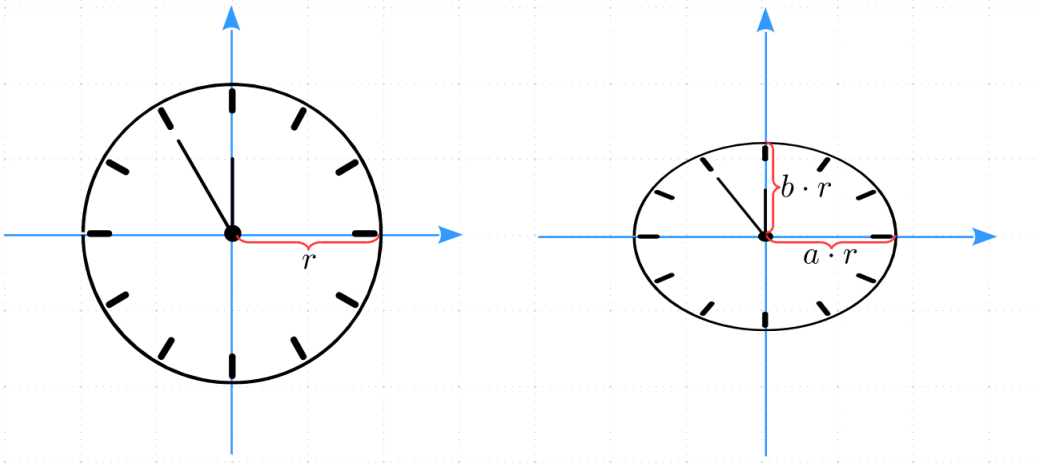
- **Rotation** counterclockwise by θ: $\bar{p}_1 = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \bar{p}_0$.



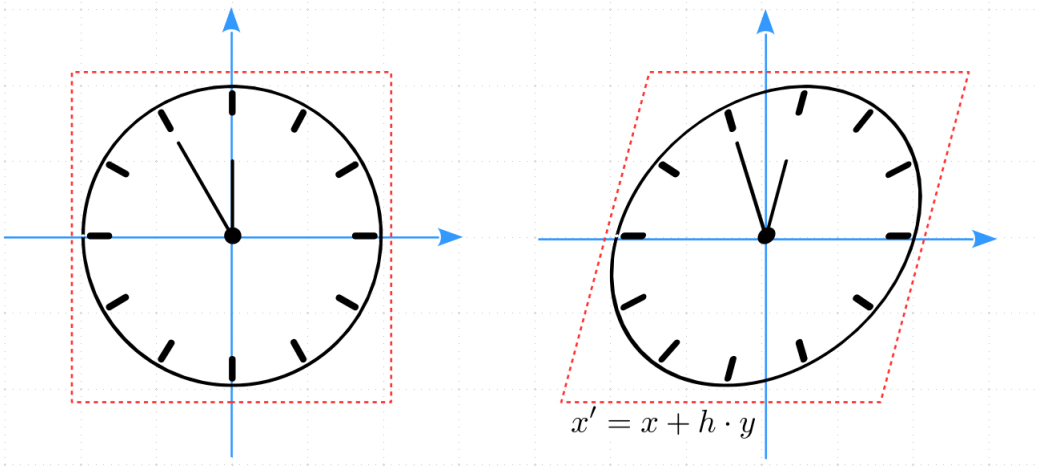
- **Uniform scaling** by scalar a: $\bar{p}_1 = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix} \bar{p}_0$.



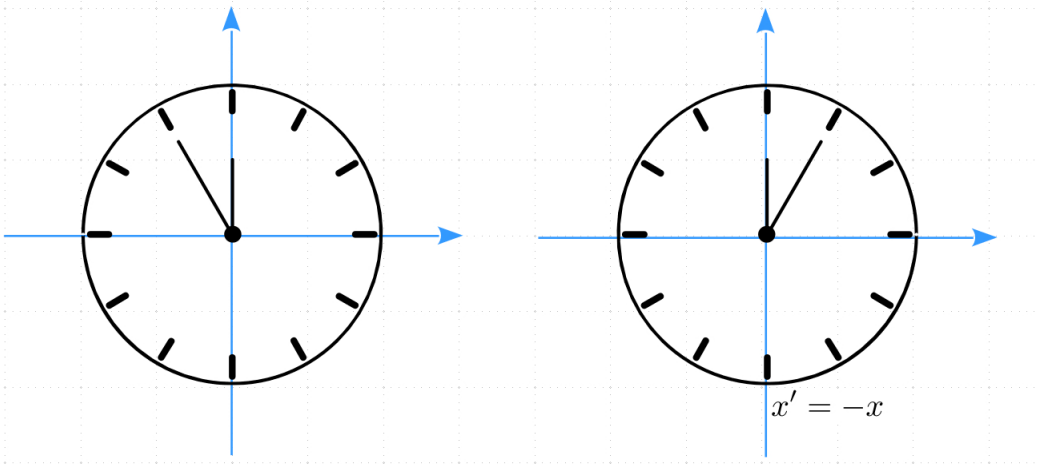
- **Nonuniform scaling** by a and b: $\bar{p}_1 = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \bar{p}_0$.



- **Shear** by scalar h: $\bar{p}_1 = \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix} \bar{p}_0$.



- **Reflection** about the y-axis: $\bar{p}_1 = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \bar{p}_0$.



3.2 Affine Transformations

An **affine transformation** takes a point $\bar{p}$ to $\bar{q}$ according to $\bar{q} = F(\bar{p}) = A\bar{p} + \vec{t}$, a linear transformation followed by a translation. You should understand the following proofs.

- The inverse of an affine transformation is also affine, assuming it exists.

Proof:
Let $\bar{q} = A\bar{p} + \vec{t}$ and assume A^{-1} exists, i.e. $\det(A) \neq 0$.
Then $A\bar{p} = \bar{q} - \vec{t}$, so $\bar{p} = A^{-1}\bar{q} - A^{-1}\vec{t}$. This can be rewritten as $\bar{p} = B\bar{q} + \vec{d}$, where $B = A^{-1}$ and $\vec{d} = -A^{-1}\vec{t}$.

Note:
The inverse of a 2D linear transformation is

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}.$$

- Lines and parallelism are preserved under affine transformations.

Proof:
To prove lines are preserved, we must show that $\bar{q}(\lambda) = F(\bar{l}(\lambda))$ is a line, where $F(\bar{p}) = A\bar{p} + \vec{t}$ and $\bar{l}(\lambda) = \bar{p}_0 + \lambda\vec{d}$.

$$\begin{aligned} \bar{q}(\lambda) &= A\bar{l}(\lambda) + \vec{t} \\ &= A(\bar{p}_0 + \lambda\vec{d}) + \vec{t} \\ &= (A\bar{p}_0 + \vec{t}) + \lambda A\vec{d} \end{aligned}$$

This is a parametric form of a line through $A\bar{p}_0 + \vec{t}$ with direction $A\vec{d}$.

- Given a closed region, the area under an affine transformation $A\bar{p} + \vec{t}$ is scaled by $\det(A)$.

Note:

- Rotations and translations have $\det(A) = 1$.
- Scaling $A = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$ has $\det(A) = ab$.
- Singularities have $\det(A) = 0$.

Example:
The matrix $A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$ maps all points to the x-axis, so the area of any closed region will become zero. We have $\det(A) = 0$, which verifies that any closed region's area will be scaled by zero.

- A composition of affine transformations is still affine.

Proof:
Let $F_1(\bar{p}) = A_1\bar{p} + \vec{t}_1$ and $F_2(\bar{p}) = A_2\bar{p} + \vec{t}_2$.
Then,

$$F(\bar{p}) = F_2(F_1(\bar{p}))$$

$$\begin{aligned} &= A_2(A_1\bar{p} + \vec{t}_1) + \vec{t}_2 \\ &= A_2A_1\bar{p} + (A_2\vec{t}_1 + \vec{t}_2). \end{aligned}$$

Letting $A = A_2A_1$ and $\vec{t} = A_2\vec{t}_1 + \vec{t}_2$, we have $F(\bar{p}) = A\bar{p} + \vec{t}$, and this is an affine transformation.

3.3 Homogeneous Coordinates

Homogeneous coordinates are another way to represent points to simplify the way in which we express affine transformations. As shown juat above, compositing transformations becomes cumbersome because of the need to carry two terms with each transformation.
With homogeneous coordinates, affine transformations become matrices, and composition of transformations is as simple as matrix multiplication. In future sections of the course we exploit this in much more powerful ways.

With homogeneous coordinates, a point $\bar{p} = \begin{bmatrix} p_x \\ p_y \end{bmatrix}$ is augmented with a 1, to form

$\hat{p} = \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ p_w \end{bmatrix}$,
Notice that, in certain conditions, p_w may not be 1.

Given $\hat{p}$ in homogeneous coordinates, to get $\bar{p}$, we divide $\hat{p}$ by its last component and discard the last component, $\bar{p} = \begin{bmatrix} p'_x/p_w \\ p'_y/p_w \end{bmatrix}$. Because of this definition, all points $\begin{bmatrix} \alpha\bar{p} \\ \alpha \end{bmatrix}$ represent the same point $\bar{p}$ for real $\alpha \neq 0$.

Example:
The homogeneous points $(2, 4, 2)$ and $(1, 2, 1)$ both represent the Cartesian point $(1, 2)$. It's the orientation of $\hat{p}$ that matters, not its length.

Many transformations become linear in homogeneous coordinates, including affine transformations:

$$\begin{aligned} \begin{bmatrix} q_x \\ q_y \end{bmatrix} &= \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} p_x \\ p_y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \\ &= \begin{bmatrix} a & b & t_x \\ c & d & t_y \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} A & \vec{t} \end{bmatrix} \hat{p} \end{aligned}$$

To produce $\hat{q}$ rather than $\bar{q}$, we can add a row to the matrix:

$$\hat{q} = \begin{bmatrix} A & \vec{t} \\ \vec{0}^T & 1 \end{bmatrix} \hat{p} = \begin{bmatrix} a & b & t_x \\ c & d & t_y \\ 0 & 0 & 1 \end{bmatrix} \hat{p}.$$

This is linear! Bookkeeping becomes simple under composition.

Example:
$F_3(F_2(F_1(\bar{p})))$, where $F_i(\bar{p}) = A_i(\bar{p}) + \vec{t}_i$ becomes $M_3 M_2 M_1 \bar{p}$, where $M_i = \begin{bmatrix} A_i & \vec{t}_i \\ \vec{0}^T & 1 \end{bmatrix}$.

With homogeneous coordinates, the following properties of affine transformations become apparent:

- Affine transformations are associative.
  For affine transformations F_1, F_2, and F_3,

$$(F_3 \circ F_2) \circ F_1 = F_3 \circ (F_2 \circ F_1).$$

- Affine transformations are *not* commutative.
  For affine transformations F_1 and F_2,

$$F_2 \circ F_1 \neq F_1 \circ F_2.$$

3.4 Uses and Abuses of Homogeneous Coordinates

Homogeneous coordinates provide a different representation for Cartesian coordinates, and cannot be treated in quite the same way. For example, consider the midpoint between two points $\bar{p}_1 = (1, 1)$ and $\bar{p}_2 = (5, 5)$. The midpoint is $(\bar{p}_1 + \bar{p}_2)/2 = (3, 3)$. We can represent these points in homogeneous coordinates as $\hat{p}_1 = (1, 1, 1)$ and $\hat{p}_2 = (5, 5, 1)$. Directly applying the same computation as above gives the same resulting point: $(3, 3, 1)$. However, we can *also* represent these points as $\hat{p}'_1 = (2, 2, 2)$ and $\hat{p}'_2 = (5, 5, 1)$. We then have $(\hat{p}'_1 + \hat{p}'_2)/2 = (7/2, 7/2, 3/2)$, which corresponds to the Cartesian point $(7/3, 7/3)$. This is a different point, and illustrates that we cannot blindly apply geometric operations to homogeneous coordinates. The simplest solution is to **always convert homogeneous coordinates to Cartesian coordinates.** That said, there are several important operations that can be performed correctly in terms of homogeneous coordinates, as follows.

Affine transformations. An important case in the previous section is applying an affine transformation to a point in homogeneous coordinates:

$$\bar{q} = F(\bar{p}) = A\bar{p} + \vec{t} \quad (1)$$
$$\hat{q} = \hat{A}\hat{p} = (x', y', 1)^T \quad (2)$$

It is easy to see that this operation is correct, since rescaling $\hat{p}$ does not change the result:

$$\hat{A}(\alpha\hat{p}) = \alpha(\hat{A}\hat{p}) = \alpha\hat{q} = (\alpha x', \alpha y', \alpha)^T \quad (3)$$

which is the same geometric point as $\hat{q} = (x', y', 1)^T$

Vectors. We can represent a vector $\vec{v} = (x, y)$ in homogeneous coordinates by setting the last element of the vector to be zero: $\hat{v} = (x, y, 0)$. However, when adding a vector to a point, the point must have the third component be 1.

$$\hat{q} = \hat{p} + \hat{v} \quad (4)$$
$$(x', y', 1)^T = (x_p, y_p, 1) + (x, y, 0) \quad (5)$$

The result is clearly incorrect if the third component of the vector is not 0.

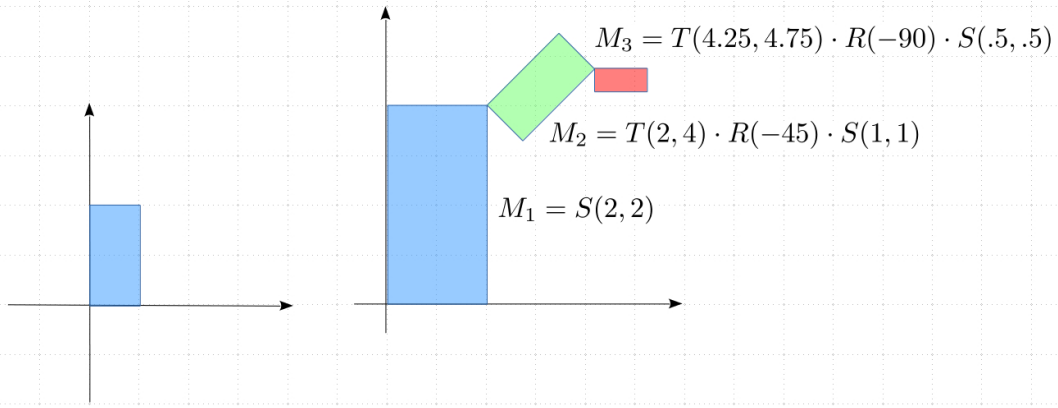
Aside:
Homogeneous coordinates are a representation of points in **projective geometry.**

3.5 Hierarchical Transformations

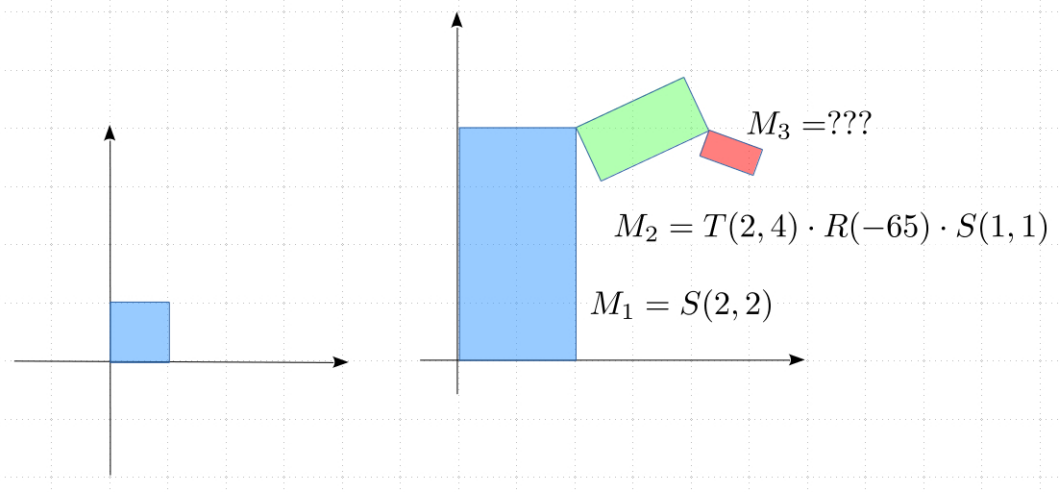
It is often convenient to model objects as hierarchically connected parts. For example, a robot arm might be made up of an upper arm, forearm, palm, and fingers. Rotating at the shoulder on the upper arm would affect all of the rest of the arm, but rotating the forearm at the elbow would affect the palm and fingers, but not the upper arm. A reasonable hierarchy, then, would have the upper arm at the root, with the forearm as its only child, which in turn connects only to the palm, and the palm would be the parent to all of the fingers.

Each part in the hierarchy can be modeled in its own local coordinates, independent of the other parts. For a robot, a simple square might be used to model each of the upper arm, forearm, and so on. Rigid body transformations are then applied to each part relative to its parent to achieve the proper alignment and pose of the object. For example, the fingers are positioned to be in the appropriate places in the palm coordinates, the fingers and palm together are positioned in forearm coordinates, and the process continues up the hierarchy. Then a transformation applied to upper arm coordinates is also applied to all parts down the hierarchy. A simple example is shown below.

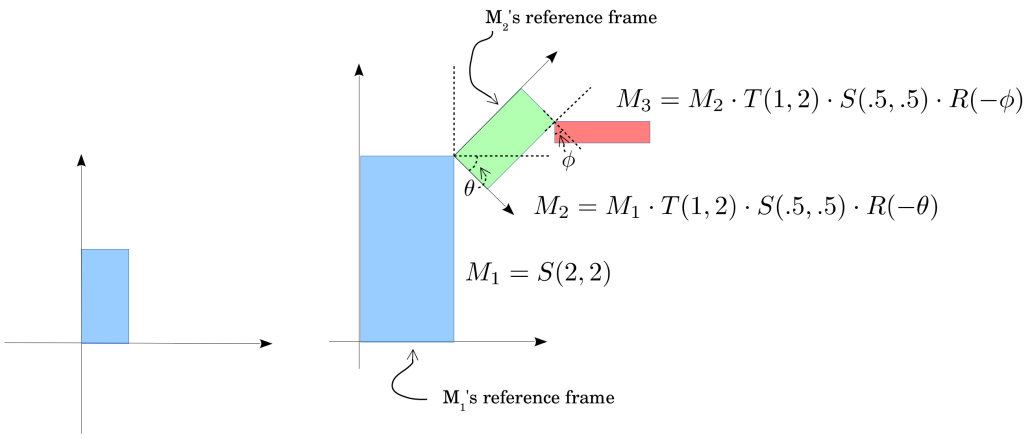
Example:
Example of hierarchical transformations for articulated objects. Here we have a very simple robot arm consisting of 3 suitably transformed rectangles as shown below. Note the **global** transformations required to set up each shape in the right place.



Because the transformations are global (i.e. each shape's transform matrix is independent of all other shapes), changing the configuration of one of the parts (for example, rotating the middle segment as depicted below) forces us to figure out the global matrices required for anything else attached to the part that moved. This may not be a straightforward task, and in any event, is cumbersome to do for complex objects



Consider now the situation in the following diagram.



Here we define the robot arm as a hierarchical shape consisting of three parts, where each part is defined *in the local reference frame* of the part that precedes it.

The first part establishes a local coordinate frame aligned with the global X and Y axes, but scaled by 2, i.e. a change of one unit in X_{M_1} is equal to two units along the global X axis, and the same for Y.

We can now define the transformation that puts the middle segment of the robot's arm in the right place, but we will do so *within M_1's local coordinate frame*. First note that the middle segment is located at the top-right corner of the bottom segment, so a translation is needed. The right translation in M_1's coordinate frame is $T(1,2)$ - the size of the original rectangle shape! We then need to scale the shape by $.5$ in each direction because the middle segment is half the size of the first, and we need to rotate by $-\theta$.

The resulting transform M_2 defines a local coordinate frame aligned with the mid-segment of the robot's arm. Drawing the end segment within this coordinate frame is simple: Translate $T(1,2)$, scale $S(.5,.5)$, and rotate by $-\phi$.

The great advantage of this approach is that all we need to do to specify any configuration of the robot's arm is to set the values for θ and ϕ. Changing these parameters no longer forces us to figure out a new global matrix for related parts. Any objects composed of multiple, articulated parts, are best modeled using hierarchical transformations of this kind.

Note: When composing the transformation matrices, it's important to get the order of the transformations right. As noted above, when using homogeneous coordinates the composition of affine transforms is simply a matrix multiplication. For two affine transformations that are performed in sequence, T_i followed by T_j, the final transform

is obtained by *left multyplying* T_i by T_j, that is $T = T_j \cdot T_i$.

With hierarchical transforms, however, the order is inverted. That is, in order to draw part j whose parent part i has a hierarchical transform matrix M_i, we *right multiply* M_i by the local affine transform T_{jlocal} required to obtain part j in part i's local reference frame. The final hierarchical transform matrix in this case is given by $M_j = M_i \cdot T_{\text{jlocal}}$.

Caveat: Hierarchical transforms do not work in general with non-uniform scaling. Because of the order in which the transformation matrices are composited, introducing non-uniform scaling anywhere along the hierarchy changes the way rotations behave. Avoid non-uniform scaling if you plan to use hierarchically defined objects. This is the reason we started with a rectangle in the example above, and not from a square.

