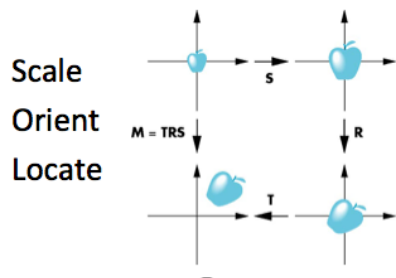
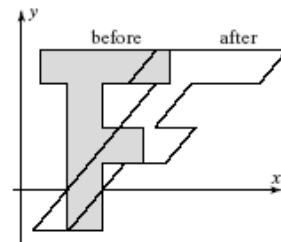
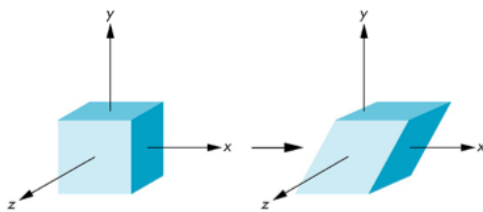


## Transformation – Part 2

- How to perform the inverse transformation? i.e., undo transformation?
  - Although we could compute inverse matrices by general formulas, we can use simple geometric observations
    - Translation:  $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
    - Rotation:  $\mathbf{R}^{-1}(\theta) = \mathbf{R}(-\theta)$
    - Scaling:  $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$
- Forming composite Matrix (concatenate matrices) -- We can form arbitrary affine transformation matrices by multiplying together rotation, translation, and scaling matrices
  - Does the order of the matrices matter?
  - How to form the order of the matrices?
    - right to left, rightmost the first operation applied
    - $Q = MP = \mathbf{T}(p_f) \mathbf{R}(q) \mathbf{T}(-p_f)P$  // rotation about a fixed point
    - `gl_Position = translation * rz * ry * rx * scale * vPosition;` // in vertex shader
  - In modeling, we often start with a simple object centered at the origin, oriented with the axis, and at a standard size. We apply an **instance transformation** to its vertices:

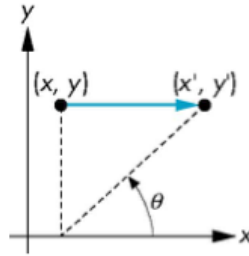


- Practice Exercise: Given a unit square centered at the origin, what does the square look like after the following sequence of transformations have been applied?
  - Translate along X-axis by 3 units, along Y-axis by 2 units
  - Rotate 45 degrees along the Z-axis
  - Scale it along X-axis by 3, along Y-axis by 2
- Change the order of the operations in: `gl_Position = translation * rz * ry * rx * scale * vPosition;` // in vertex shader ?
- Shear** -- Equivalent to pulling faces in opposite directions
  - Shear is the translation along an axis (say, X axis) by an amount that increases linearly with another axis (Y).



- Matrix for this Shear transformation is :  $\mathbf{T} = \begin{bmatrix} 1 & \cot \theta & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

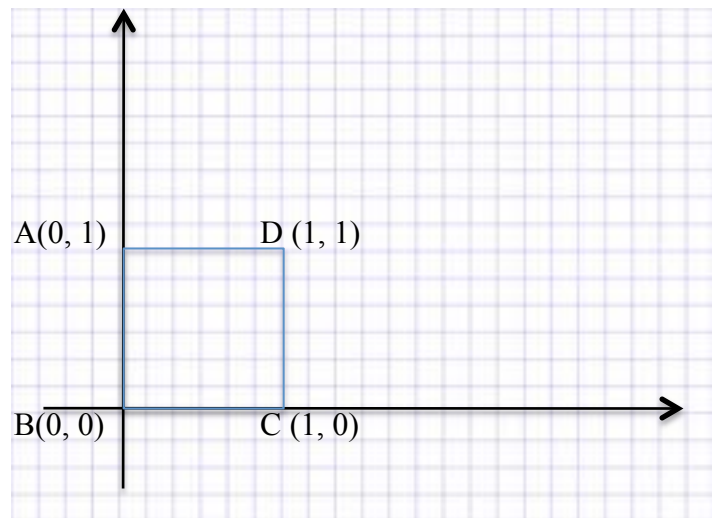
$$\begin{aligned}x' &= x + y \cot \theta \\y' &= y \\z' &= z\end{aligned}$$



$$T = \begin{bmatrix} 1 & h & 0 & 0 \\ g & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Matrix for a shear transformation along both x and y:  $T =$
- Practice Exercise:
  - Given a unit square shown, what is the square after the following shear transformation?

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



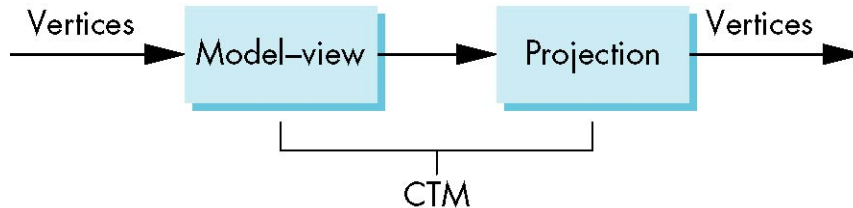
• **Perform transformations in WebGL using the provided library functions**

- Library functions:
  - var r = rotate(theta, vx, vy, vz);
  - var t = translate(tx, ty, tz);
  - var s = scale(sx, sy, sz);
  - var m = mat4() → identity matrix
  - var m = mult(m1, m2) → multiple 2 4x4 matrices
- Example 1: Rotation about z axis by 30 degrees with a fixed point of (1.0, 2.0, 3.0)
  - var t1=translate(1.0, 2.0, 3.0);
  - var t2= translate(-1.0, -2.0, -3.0));
  - var r=rotate(30.0, 0.0, 0.0, 1.0));
  - var m = mult(t1, r);
  - m = mult(m, t2);
  - // now send the matrix m to the vertex shader
  - (note: the last matrix specified in the program is the first applied)

- Practice Exercise: Given a unit square centered at the origin, what is the WebGL .js code for the following transformations?

- Translate along X-axis by 3 units, along Y-axis by 2 units, and along Z-axis by 5 units
- Rotate 45 degrees along the Z-axis
- Scale it to be 3 times as wide along X-axis, two times as tall along Y-axis, no change in Z-axis.

- Current Transformation Matrix (CTM) in WebGL
  - The *current transformation matrix* (CTM) is a 4 x 4 homogeneous coordinate matrix
  - It is part of the state and is applied to all vertices that pass down the pipeline



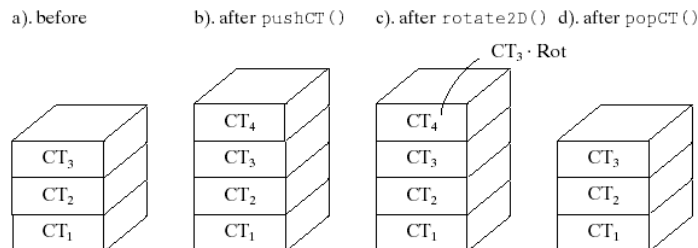
- In WebGL, the model-view matrix is used to
  - Position the camera
    - Can be done by rotations and translations but is often easier to use the `lookAt` function in `MV.js`
  - Build models of objects
- The projection matrix is used to define the view volume and to select a camera lens
- $p' = P \cdot MV \cdot p$

### Matrix Stacks

- In many situations we want to save transformation matrices for use later. In JS, it can be achieved by:

```
var stack = [ ]
stack.push(modelViewMatrix);
modelViewMatrix = stack.pop();
```

- Transformation matrix stacks:



- Practice Exercise: how to draw the following figures? using a single branch for the snowflake and using the vertices of a single dinosaur?

