# CS3621 Introduction to Computing with Geometry
# Solutions Drill Exercises – Geometric Transformations and Simple Models

1. Suppose the point $(1,1,1)$ is rotated about the $x$-, then $y$-, and then $z$- axis 45°. What are the results? Write down the rotation matrices and the results.

   **Solution:** Simple. Do it yourself.

2. Suppose the point $(1,1,1)$ is first rotated about the $x$-axis 45° and then about the $z$-axis 30°. What is the rotation matrix and the result?

   **Solution:** The matrices of rotating about the $x$- and $z$-axis 45°, $R_x(45°)$ and $R_z(45°)$, are

$$R_x(45°) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{and} \quad R_z(45°) = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   Therefore, the desired rotation has a transformation matrix as follows (note the order of multiplication):

$$R_z(45°) \cdot R_x(45°) = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 & 0 \\ \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{1}{2} & \frac{1}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   The desired result is computed as follows:

$$R_z(45°) \cdot R_x(45°) \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & -\frac{1}{2} & \frac{1}{2} & 0 \\ \frac{\sqrt{2}}{2} & \frac{1}{2} & -\frac{1}{2} & 0 \\ 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} \\ \sqrt{2} \\ 1 \end{bmatrix}$$

   Thus, $(1,1,1)$ is transformed to $(\frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2}, \sqrt{2})$.

3. Suppose the point $(1,1,1)$ is first rotated about the $y$-axis 45° and then translated in the direction of $\langle 2,2,2 \rangle$. What is the resulting transformation matrix and the result?

   **Solution:** Translation in the direction of $\langle 2,2,2 \rangle$ has transformation matrix:

$$T(\langle 2,2,2 \rangle) = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   Rotating about the $y$-axis 45° has transformation matrix

$$R_y(45°) = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   Thus, a rotation followed by a translation yields the following transformation matrix:

$$T(\langle 2,2,2 \rangle) \cdot R_y(45°) = \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2 \\ 0 & 1 & 0 & 2 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Thus, $(1,1,1)$ is transformed to

$$\begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2 \\ 0 & 1 & 0 & 2 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+\sqrt{2} \\ 3 \\ 2 \\ 1 \end{bmatrix}$$

The new point is $(2+\sqrt{2}, 3, 2)$.

4. Suppose the point $(1,1,1)$ is first translated in the direction of $\langle 2,2,2\rangle$ and then rotated about the $y$-axis 45°. What is the resulting transformation matrix and the result? Is this result identical to the one you obtained in the previous problem?

   **Solution:** The transformation matrix is

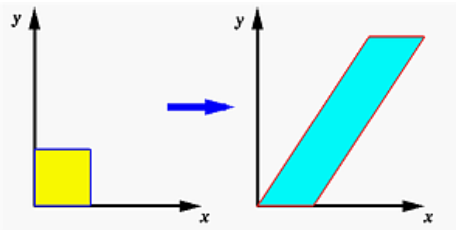
$$R_y(45°) \cdot T(\langle 2,2,2\rangle) = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2\sqrt{2} \\ 0 & 1 & 0 & 2 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

   Therefore, $(1,1,1)$ is transformed to

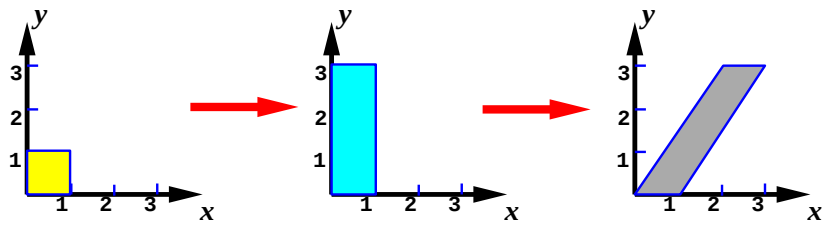
$$\begin{bmatrix} \frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 2\sqrt{2} \\ 0 & 1 & 0 & 2 \\ -\frac{\sqrt{2}}{2} & 0 & \frac{\sqrt{2}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3\sqrt{2} \\ 3 \\ 0 \\ 1 \end{bmatrix}$$

   The new point is $(3\sqrt{2}, 3, 0)$.

5. An application needs to transform a unit square whose vertices are (0,0), (1,0), (1,1) and (0,1) to a parallelogram whose vertices are (0,0), (1,0), (3,3) and (2,3) as show below. Explain how you will achieve this. Write down each individual transformation and its transformation matrix, compute the composite transformation, and verify your result. Note that the answer is *not* unique.




   **Solution:** This can be done by first scaling in the $y$-direction, which transforms the unit square to a rectangle of vertices (0,0), (1,0), (1,3) and (0,3), followed by a shearing in the $x$-direction. This procedure is shown below.



The scaling transformation matrix is

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The shear in $x$-direction is

$$H = \begin{bmatrix} 1 & \alpha & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

where $\alpha$ is an unknown. Since this shear sends (1,3) to (3,3), it satisfies the following:

$$\begin{bmatrix} 3 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & \alpha & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+3\alpha \\ 3 \\ 1 \end{bmatrix}$$

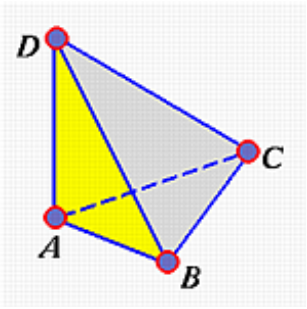
Therefore, we have $3 = 1 + 3\alpha$. Solving for $\alpha$, we have $\alpha = 2/3$ and the shear matrix

$$H = \begin{bmatrix} 1 & \frac{2}{3} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Consequently, the desired transformation matrix is

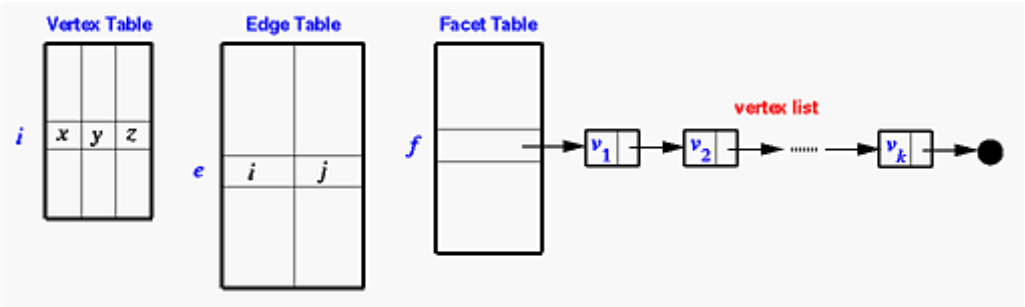
$$H \cdot S = \begin{bmatrix} 1 & \frac{2}{3} & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

6. Suppose we have a tetrahedron with four vertices $A = (0,0,0)$, $B = (1,0,0)$, $C = (0,1,0)$ and $D = (0,0,1)$, and edges $AB$, $BC$, $AC$, $DA$, $DB$ and $DC$ (see the figure below). The four facets are $ABC$ (bottom), $ABD$ (left), $BCD$ (right) and $ACD$ (back). Design a wireframe representation for this tetrahedron solid.




**Solution:** Simple. Do it yourself.

7. We can extend the wireframe data structure to include facet information as shown below. In addition to a vertex table and an edge table, we have a facet table whose $f$-th entry represents facet $f$ and contains a pointer to a linked list, a *vertex list*. Each node of a vertex list contains a vertex of that facet and the list is built in either clockwise or counter-clockwise order. This representation or its variants are frequently used in graphics programming.




Answer the following important questions:

(a) Suppose $V$, $E$ and $F$ are the number of vertices, edges, and facets. Can you come up with a *very* accurate estimation of memory usage? What is the worst case? You can assume one word for each integer or pointer.

**Solution:** The vertex and edge tables require $3V$ and $2E$ words, respectively. The facet table needs $F$ words. But, because we do not have any information about the length of each vertex list (*i.e.*, the number of vertices of each facet), it is difficult to have an accurate estimate. Since a facet has at most $V-1$ vertices (why?), each vertex list has $2(V-1)$ words, and, as a result, $F \times 2(V-1) + F = F(2V-1)$ words are required to store the facet information. The total memory requirement is at most $3V + 2E + F(2V-1)$ words.

(b) It is also possible that the nodes of a vertex list store edges rather than vertices. Would this help solve the previous problem?

**Solution:** Similar to the previous one. Do it yourself.

(c) Suppose we know that on average each facet has $k$ vertices (*e.g.*, $k = 3$ for triangles). Would this information help?

**Solution:** Yes, it does help. In this case, each vertex list has $k$ nodes and requires $2k$ words. The total memory requirement is about $3V + 2E + (F + F \times 2k) = 3V + 2E + F(2k+1)$ words.

(d) Given a vertex $v$, develop an algorithm that lists all adjacent edges of $v$. An edge is adjacent to $v$ if that edge has $v$ as one of its vertices. How many comparisons are required (in worst case)?

**Solution:** Because all adjacent information between vertices and edges are recorded in the edge table, we can do a look-up and extract the desired information:

```
procedure ListEdges(v);
begin
    for e := 1 to E do
        if edge e contains vertex v then
            print edge e
        end if
end
```

In the above code, $E$ is the number of edges. Each edge in the edge table is examined, and is printed if that edge contains the given vertex. Since two comparisons are required for examining each edge, the total number of comparisons required is $2E$.

(e) Given a vertex $v$, develop an algorithm that lists all adjacent facets of $v$. A facet is adjacent to $v$ if that facet has $v$ as its vertex. How many comparisons are required (in worst case)?

**Solution:** This is similar to the previous problem. However, we search *all* vertex lists of the facet table.

```
procedure ListFacets(v);
begin
    for f := 1 to F do
        for each node of the vertex list of facet f do
            if that node is vertex v then
                print facet f
                exit the inner for loop;
            end if
end
```

Since every vertex list that is part of the facet table has to be searched, the number of comparisons is equal to the *total* length of all vertex lists. It is difficult to estimate a worst case; but, if all facets are triangles, then $3F$ comparisons are required.

(f) Given an edge $e$, develop an algorithm that lists all adjacent facets of $e$. A facet is adjacent to $e$ if that facet has $e$ as its edge. How many comparisons are required (in worst case)?

**Solution:** There are exactly two facets adjacent to an edge. An edge has two end vertices, and, they should be adjacent to each other in a vertex list. The following is a possible algorithm:

```
procedure ListFacets(e);
begin
    for each facet f do
        begin
            This := the first node of f's vertex list;
            Next := This's next node;
            while Next is not null do
                begin
                    if the end vertices of e are This and Next then
                        print facet f and go for next facet f;
                    end if
                    This := Next;
                    Next := This's next node;
                end
            Next := the first node of f's vertex list; /* wrap around */
            if the end vertices of e are This and Next then
                print facet f;
            end if
        end
end
```

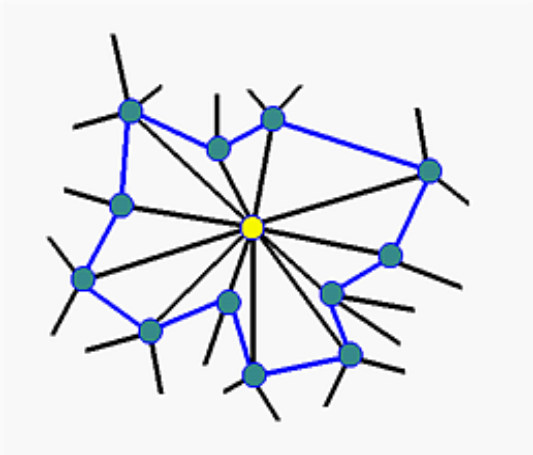
Note that wrapping around is necessary when reaching the end of a vertex list. You can also make this algorithm more efficient. Because an edge has *at most* two facets, the algorithm stops after printing the second facet. Again, it is difficult to accurately estimate the number of comparisons required because we don't know the structure of facets. But, if all facets are triangles, each vertex list contains three nodes. Since determining if an edge contains two vertices requires two comparisons, determining if a vertex list contains an edge requires $3 \times 2 = 6$ comparisons. Thus, the worst case is the case in which all vertex lists are scanned, and the total number of comparisons is $6F$.

(g) Given a facet $f$, develop an algorithm that lists all adjacent facets of $f$. A facet is adjacent to $f$ if they have a common edge or a common vertex. How many comparisons are required (in worst case)?

**Solution:** This is an extension to the previous problem. First, find all edges of the given facet by going through its vertex list. Second, for each edge, print out its adjacent facet that is not the

given one.

(h) The concepts of *star* and *link* are very important in polyhedron modeling and editing. Suppose a polyhedron's facets are *all* triangles. Given a vertex, the yellow one in the figure below, the *star* of this vertex consists of *all* triangles that are adjacent to this vertex listed in clockwise or counter-clockwise order. The *link* of this vertex consists of all vertices, listed in either clockwise or counter-clockwise order, that are on the edges adjacent to but different from the given vertex. Note that once we have a properly ordered vertices list, we can connect them together to form a *link* as shown in blue in the figure below.

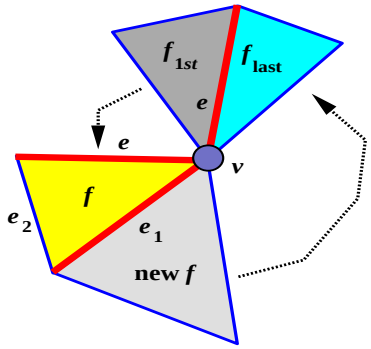


Develop an algorithm that accepts a vertex and reports the link (in either vertex or edge form) of this vertex. From this link information, you can easily generate the star.

**Solution:** This problem is an involved one. In particular, you need to generate the vertices (or edges) of the link in order. The idea of solving this problem is that we go around all facets that contain the given vertex $v$. Here is a possible algorithm:

```
procedure FindLink(v);
begin
    Find an edge e that contains v using ListEdge(v);
    Find the adjacent facets of e, f_1st and f_last using ListFacets(e);
    Let f := f_1st;
    while f ≠ f_last do
        begin
            Find the edges e, e_1 and e_2 of facet f;
            /* note that facet f contains edge e */
            Let e_1 be the edge that contains the given vertex v;
            print edge e_2; /* found one edge in the link */
            Let e := e_1; /* move to the next triangle */
            Find the adjacent facets f_1 and f_2 of edge e;
            /* note that one of f_1 and f_2 is f! */
            if f_1 = f then
                f := f_2
            else
                f := f_1
            end if
        end
end
```

Let us see how this algorithm works. See the figure below. Facets $f_{1st}$ and $f_{\text{last}}$ mark the "first" and "last" facets that share a common edge $e$ that is adjacent to vertex $v$. Initially, $f_{1st}$ is the "current" facet $f$. As long as $f$ is not the last facet $f_{\text{last}}$, we keep going so that all facets surrounding the given vertex $v$ can be traversed.




Now, when we are processing facet $f$, since it is a triangle, it has three edges, one of them must be $e$ and one of the other two that does not contain $v$ is on the link! Therefore, we print out the edge that does not contain $v$, and move on to the edge that contains $v$ but is not $e$. This is the new edge $e$.

From this new $e$, we can determine its adjacent facets, one of these two facets is $f$, which has been processed, and the other becomes the new $f$. In this way, with an edge $e$ and its adjacent facet $f$, we can traverse the link in order, although we do not know if this order is clockwise or counter-clockwise because there is no orientation information stored in the tables.

This algorithm looks simple; but, the complexity lies in the calls to find adjacent facets of a given edge. Keep in mind that the number of comparisons depends on the number of facets (*i.e.*, triangles) surrounding $v$. So, an accurate estimate is difficult to obtain.