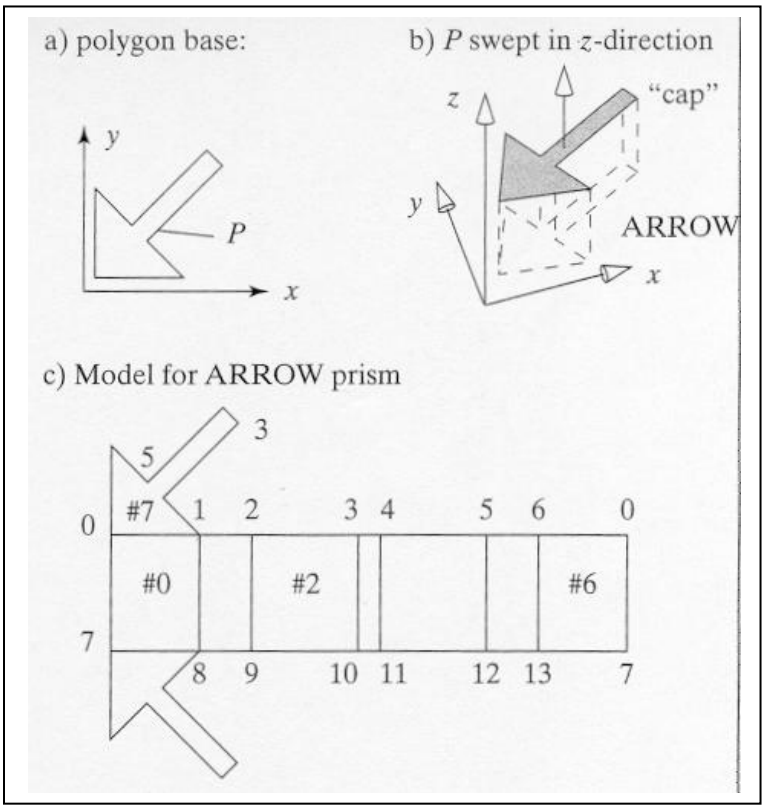
Mesh data for an extruded shape

A large number of shapes can be generated by **extruding** or **sweeping**, a 2D shape through space. The figure in **b** to the left shows such a shape. Each extruded shape starts with a 2D shape which is the base of the final shape. Figure **a** shows the polygonal base for the 3D shape shown in **b**. The 3D shape is found by sweeping the 2D shape in a straight line. In the figure, the shape is swept along the z-axis to a certain height. The resulting shape has a base and a cap (the polygon at the top of the shape) that look exactly the same. The other polygons making up the extruded shape come from the sides of the extruded shape.



Thus the shape starts with a 2D polygon with a certain number of vertices. Using figure **a** as our example, the original shape has 7 vertices 0, 1, 2, . . ., 6. After the figure is swept up the z-axis, we have 7 more vertices 7, 8, 9, . . . 13. Each of these vertices are found by making the z value equal to the height. For example if vertex 0 had the coordinates $(x_0, y_0, 0)$, then vertex 7 would be (x_0, y_0, H). Thus vertices 7 – 13 can be found easily from vertices 0 – 6.

In general, if the vertices for the base are labeled 0, 1, …, N-1 (N is the number of vertices in the base), the next vertices are labeled N, N+1, . . ., 2N – 1. Thus each edge joins vertices i and i + N as in figure **c** above. The vertex list is constructed to contain the original points $(x_i, y_i, 0)$ for i = 0, . . ., N-1 and (x_i, y_i, H) for i = N, N+1, . . ., 2N – 1. Code to fill the vertex array might appear as follows:

```
//set the total number of vertices to N
numVerts = 2*N;

//allocate enough space for the vertex array
pt = new Point3[2*N];

//create the vertex array
//the first N vertices come from the base
for (int i = 0; i < N; i++)
          pt[i] = array[i];

//the next N vertices come from the cap
for (int i = 0; i < N ; i++)
{
          pt[i+N] = array[i];
          pt[i+N].z += height;
}
```

The normal list is created by determining a normal to each face using cross products of vectors or Newell's method. The normal list will have a normal for the base, the cap, and the N faces that make up the sides of the surface.

The face list must contain indices for each vertices and normals in each face. Obviously, the base has vertices 0, 1, . . ., N-1 in it. The cap has vertices N, N+1, . . ., 2N-1 in it. For each face in the side, the jth face for j = 0, 1, . . ., N-1 there are four vertices in the face (see Figure c above). The jth face has vertices with indices j, j+N, next(j) + N, and next (j) where next(j) = (j+1)%N. The face list contains the faces in the side of the extruded shape in the list first then the base and cap face last. Code to produce the normals using Newell's method and to create the N faces might look like:

```
//set the number of normals and faces and
//allocate space for the normal vectors list and
//face array.
numNorms = N + 2;
numFaces = N + 2;
norm = new Vector3[N + 2];
face = new Face[N + 2];

//Create the face list:
//Create side faces first.  N quadrilaterals
//with 4 vertices each make up the side faces.
int nextJ;

for (int j = 0; j < N; j++)
{
          //each side face has four vertices
          face[j].nVerts = 4;

          //store indices of vertices for
          //each vertex in the face
          face[j].vert = new VertexID[4];
          face[j].vert[0].vertIndex = j;
          face[j].vert[1].vertIndex = j+N;
          nextJ = (j+1)%N;
          face[j].vert[2].vertIndex = nextJ + N;
          face[j].vert[3].vertIndex = nextJ;

          //use Newell's method to determine the
          //normal vector to each face
          norm[j] = newellMethod(j);
          for (int i = 0; i <= 3; i++)
                    //all vertices in the same face
                    //have the same normal
                    face[j].vert[i].normIndex = j;
}
//add the base face to the face list as the Nth face
//the base face has N vertices
face[N].nVerts = N;
face[N].vert = new VertexID[N];
for (int i = 0; i < N; i++)
          face[N].vert[i].vertIndex = i;

//find the normal of the Nth face
norm[N] = newellMethod(N);

//each vertex in the Nth face has the same
//normal
for (int i = 0; i < N; i++)
          face[N].vert[i].normIndex = N;

//add the cap face to the face list as the N+1st face
face[N + 1].nVerts = N;
face[N + 1].vert = new VertexID[N];
for (int i = 0; i < N; i++)
          face[N + 1].vert[i].vertIndex = N + i;
norm[N + 1] = newellMethod(N);
for (int i = 0; i < N; i++)
          face[N + 1].vert[i].normIndex = N + 1;
```

When writing the code for an extruded mesh, since an extruded mesh is a mesh, the ExtrudedMesh class inherits from the Mesh class as follows:

```
class ExtrudedMesh: public Mesh
```

```
{
public:
          ExtrudedMesh();
          ExtrudedMesh(Point3[], int numPts, int height);
}
```

The readmesh() method can be used to read the data for an extruded mesh from a file. However, since the only data usually known for the extruded mesh is in the base, it is better to have a constructor that receives an array of points that are vertices in the base plus the height the mesh should be swept to in z. The constructor can then create mesh data using the code above and the draw() method in the mesh class can be used to draw the extruded surface.

To create an extruded mesh, you might use:

```
Point3 ar[N];                          //array of N vertices for the base
. . .                                  // fill the array, ar with vertices
ExtrudedMesh extMesh(ar, N, 4);        //create an extruded mesh of height 4
```

To draw the mesh, you would use:

```
extMesh.draw();
```

To see the entire extruded mesh class, see the code on the class web site.