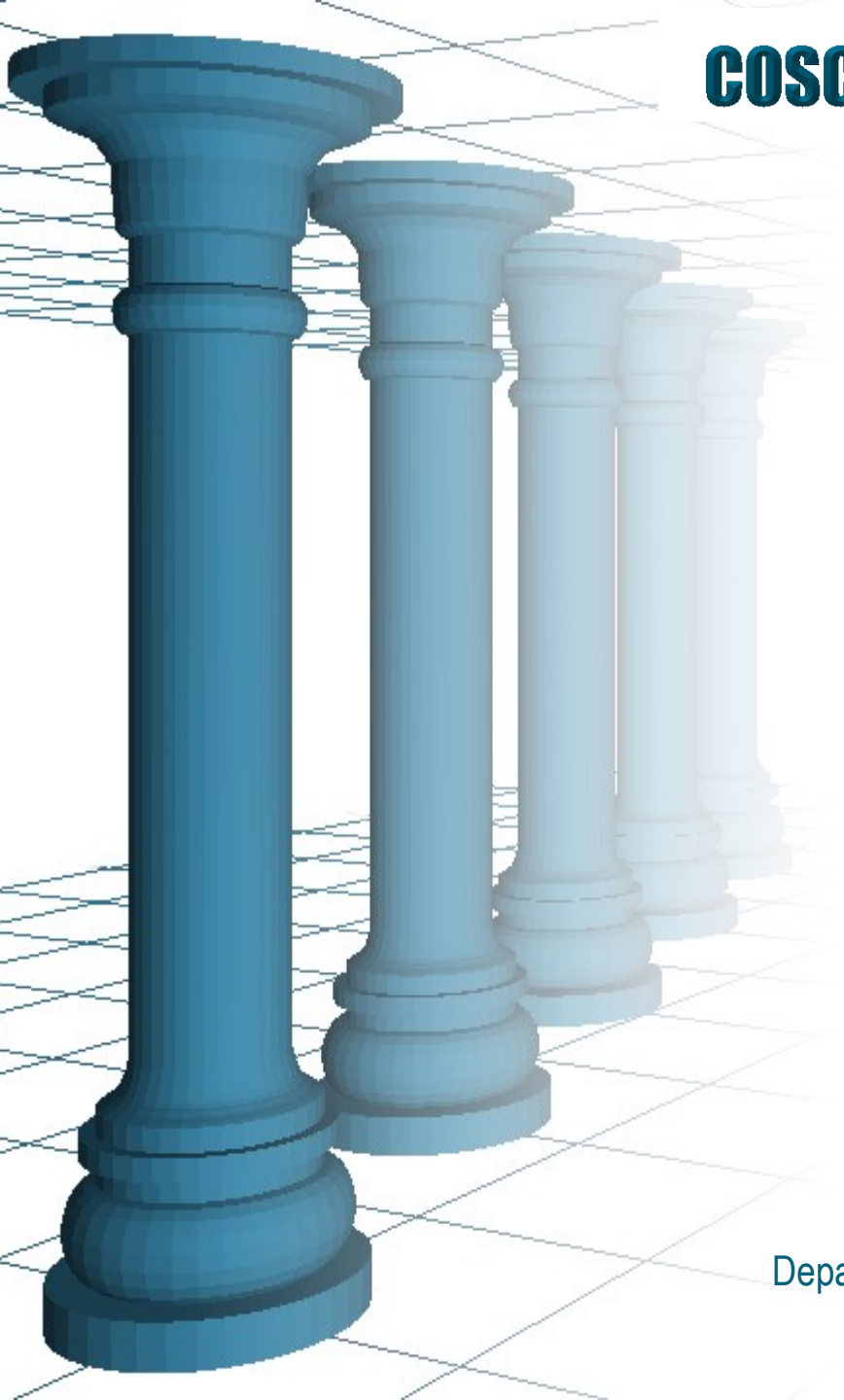


# COSC363 Computer Graphics

# 5

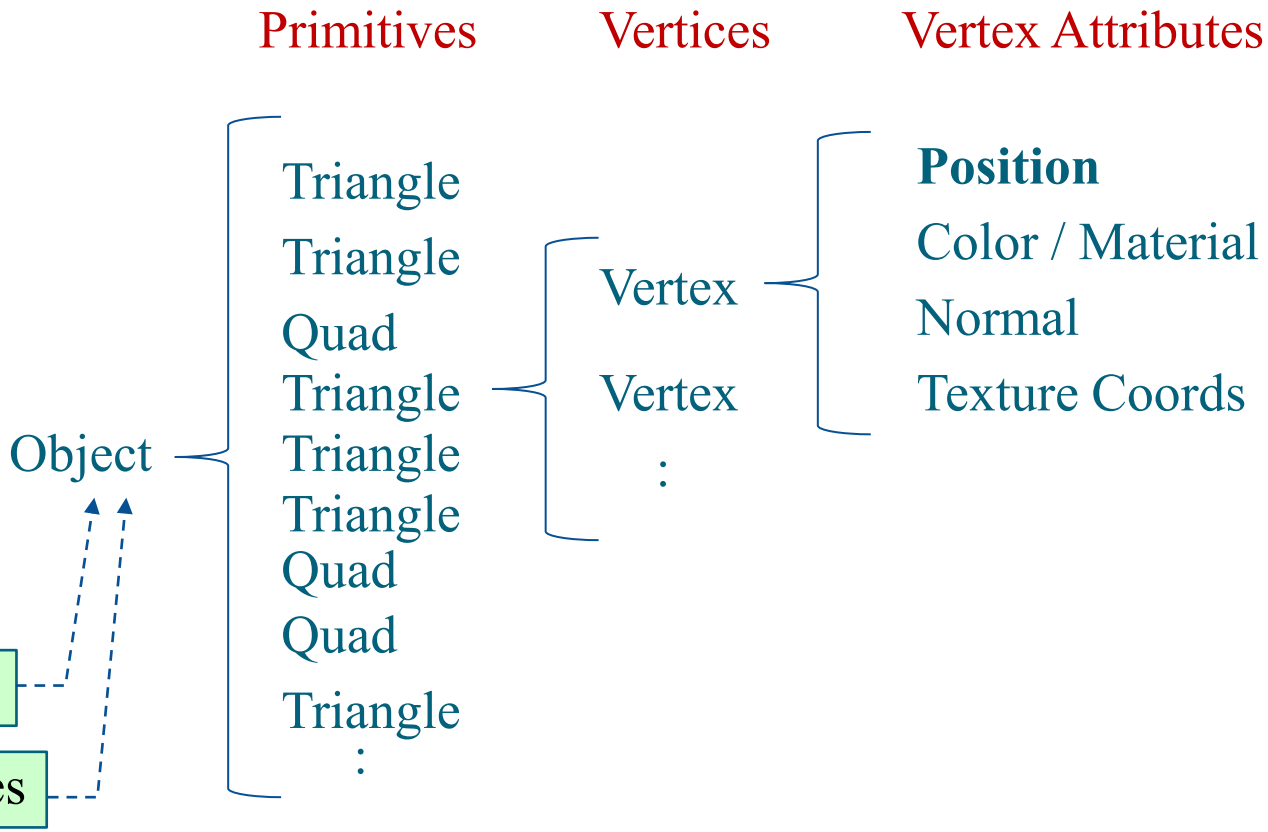
# Object Modelling

Mould your imagination!

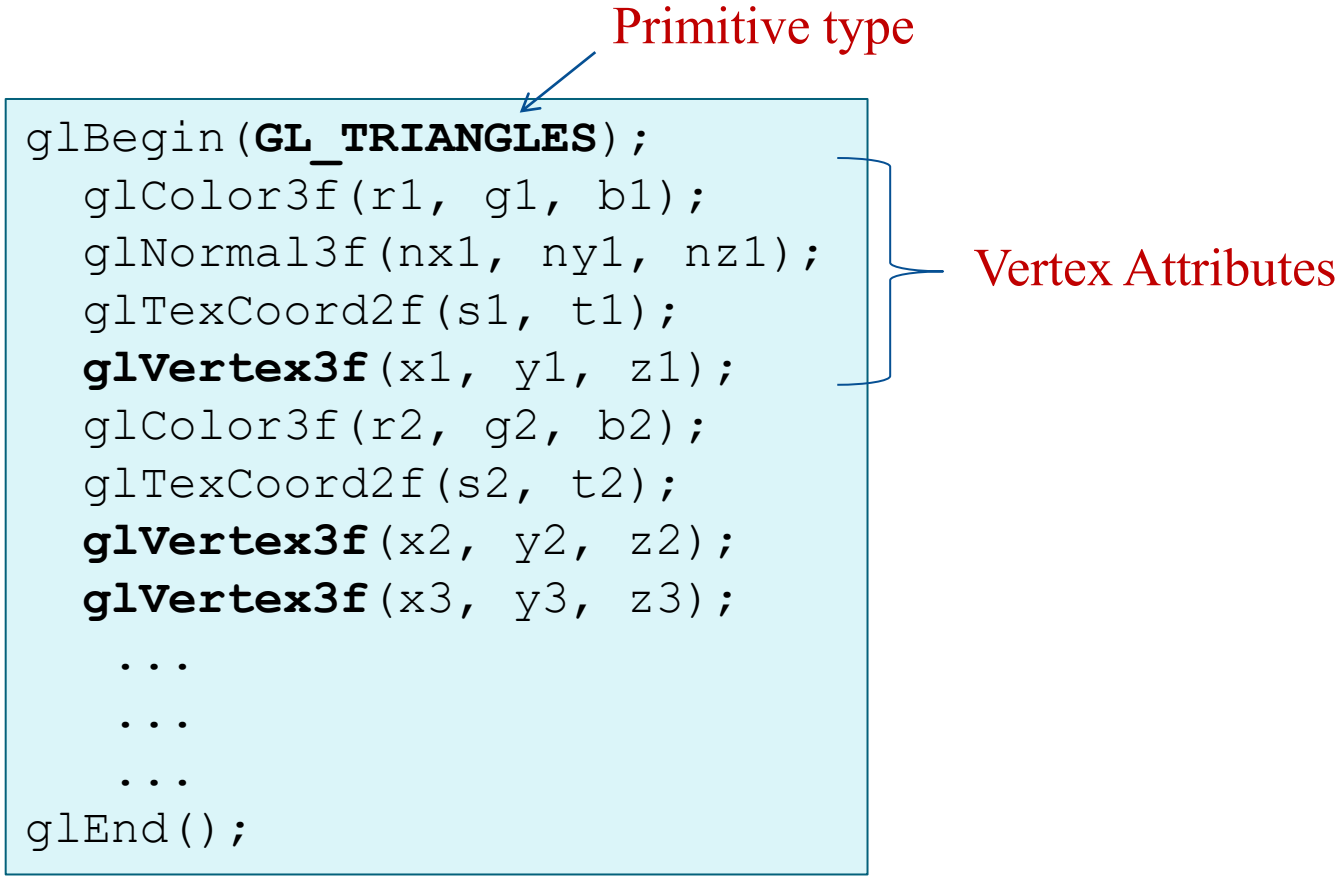
**R. Mukundan** (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

# 3D Object Models

Objects are modelled as polygonal meshes – a collection of primitives (usually triangles or quads) that together define the shape of the object.



# Primitive/Vertex Definitions: OpenGL v1, 2

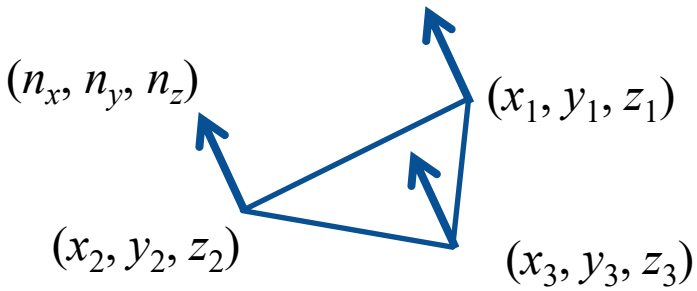



In the above example, the second and third vertices will use the normal vector defined for the first vertex.
The third vertex will use the colour and texture coordinates defined for the second vertex.

# Primitive/Vertex Definitions

```
glBegin(GL_TRIANGLES);
  computeNormal(x1,y1,z1, x2,y2,z2, x3,y3,z3);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
  computeNormal(x4,y4,z4, x5,y5,z5, x6,y6,z6);
  glVertex3f(x4, y4, z4);
  glVertex3f(x5, y5, z5);
  glVertex3f(x6, y6, z6);
  ...
glEnd();
```

Slide [3]-9




The surface normal vector of a polygonal element may be computed using three vertices of a triangle or a quad.

In this case, the surface normal vector has the same direction at all vertices of a primitive

# Modelling by Parts

An object model can be constructed as a combination of several parts, with each part carefully scaled and transformed relative to other parts to form the whole object.

Example:

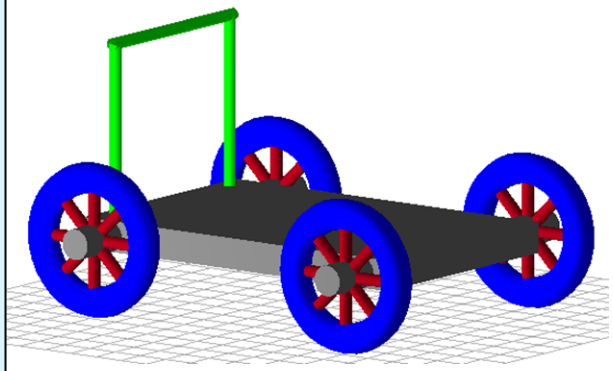
```
void display()
{

  //Cart's  motion
  glPushMatrix();
   glTranslatef(...);
   glRotatef(..);
   drawCart();
  glPopMatrix();

}
```

Object Transformations

```
void drawCart()
{
  glPushMatrix();
     glTranslatef(..);
     glScalef(..);
     drawBase();
  glPopMatrix();
  glPushMatrix();
     glTranslatef(..);
     drawHandle();
  glPopMatrix();
  for(i=0; i<4; i++)
  {
     glPushMatrix();
       glTranslatef(..);
       glRotatef(..);
       drawWheel();
     glPopMatrix();
  }
}
```

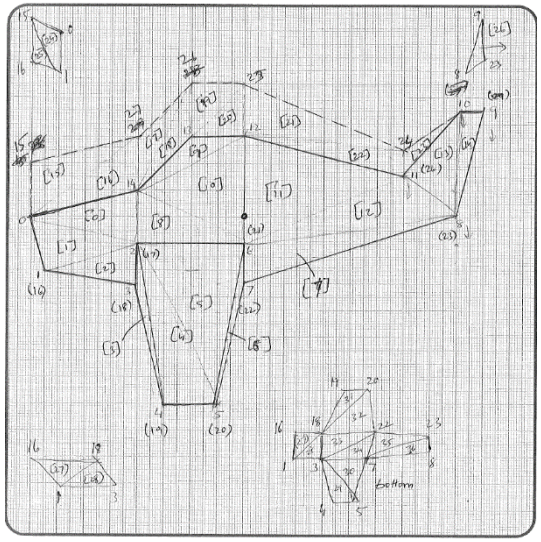


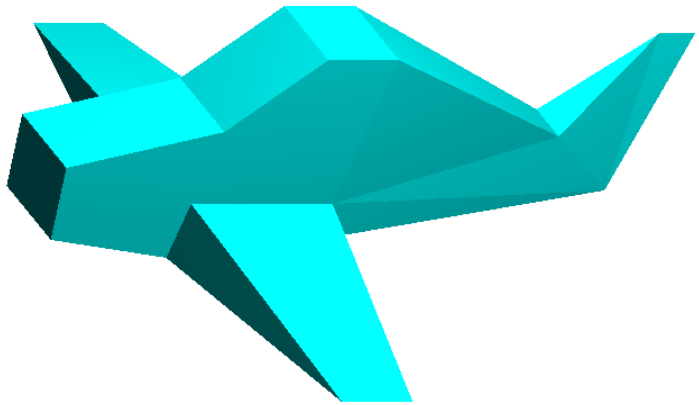
Design Transformations

Object Definition

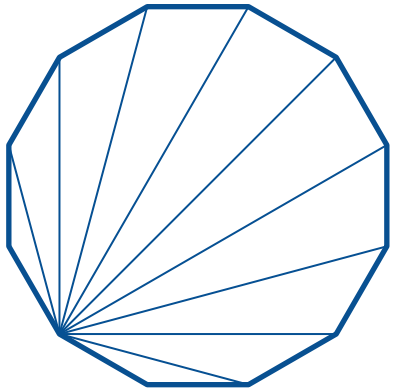
# Modelling of Simple Objects

- Create a simple sketch on a graph paper.
- With the centre point as the origin, get the *x*, *y* coordinates.
- The *z* coordinate is the out-of-plane distance (depth)
- Tessellate the object into a set of triangles
- Create the triangle definitions using an anti-clockwise ordering of vertex indices.

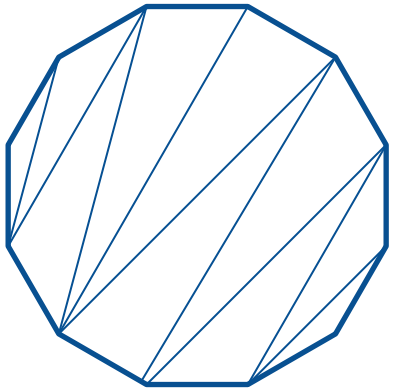




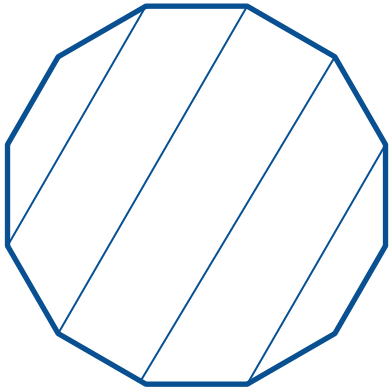
# Tessellating Polygonal Shapes



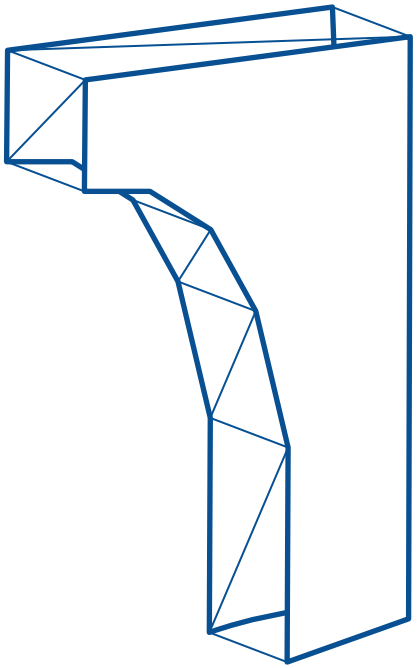
Triangle fan



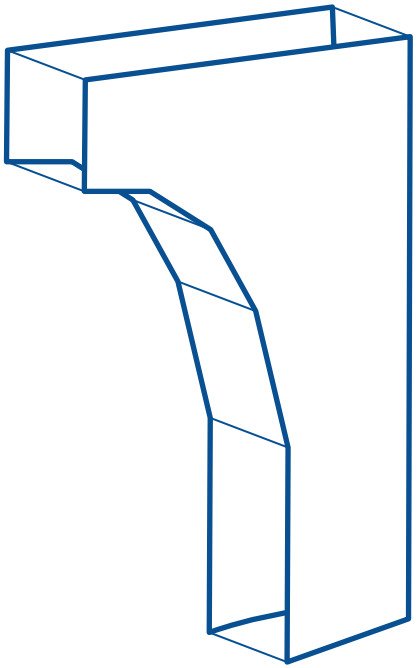
Triangles



Quads



Triangle strip

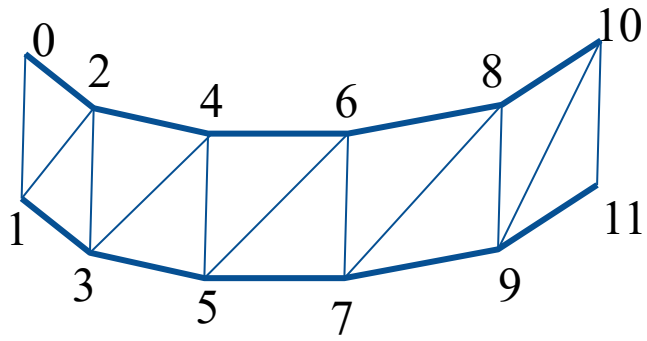


Quad strip

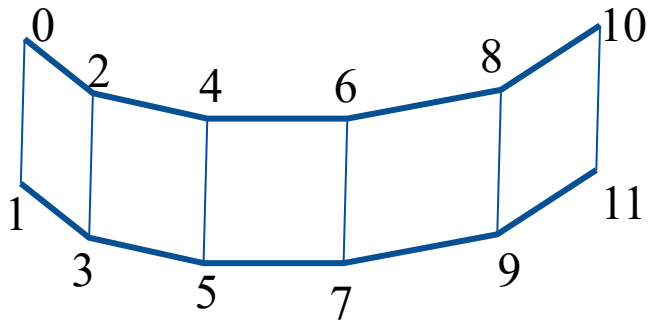
Note: Triangles are generally preferred to quads, since in the most general case, quads can be non-convex and non-planar.

# Polygon Strips

Triangle and quad strips are convenient primitive types for the construction of surfaces between two polygonal edges with the same number of vertices.




Triangle strip

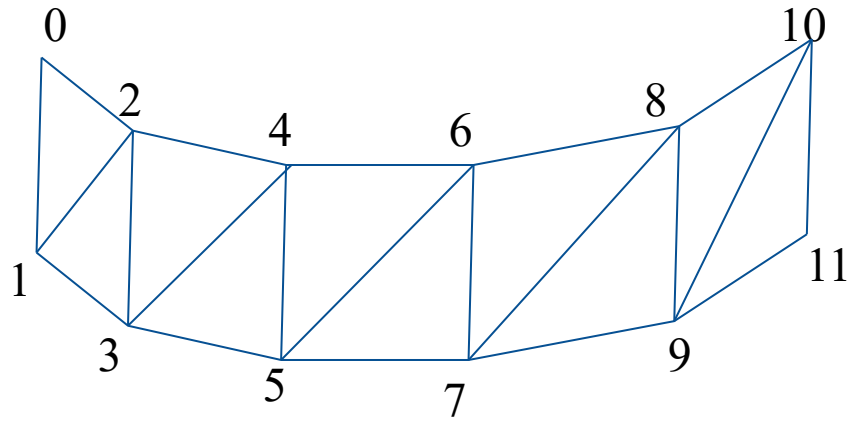



Quad strip

The vertices of a strip are specified in a sequence as shown above, alternating between the two polygonal edges.

# Triangle Strips

Triangle strips allow memory efficient data storage




Vertex array for **GL_TRIANGLES**: 30 vertices

0, 1, 2, 2, 1, 3, 2, 3, 4, 4, 3, 5, 4, 5, 6, 6, 5, 7, 6, 7, 8, 8, 7, 9, 8, 9, 10, 10, 9, 11

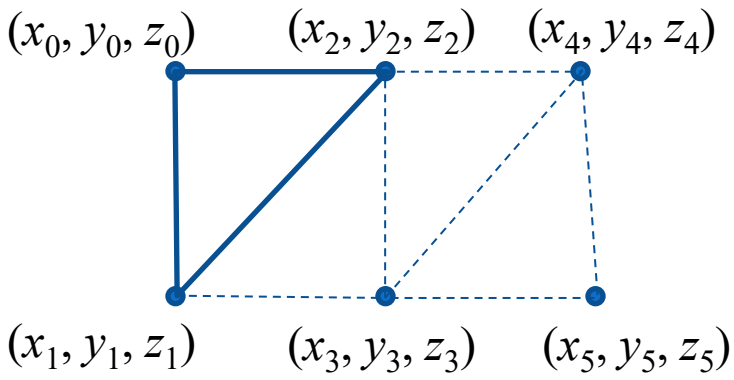
Vertex array for **GL_TRIANGLE_STRIP**: 12 vertices

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11

# Triangle Strips

From the third vertex onwards, *every vertex* forms a new triangle with the previous two vertices

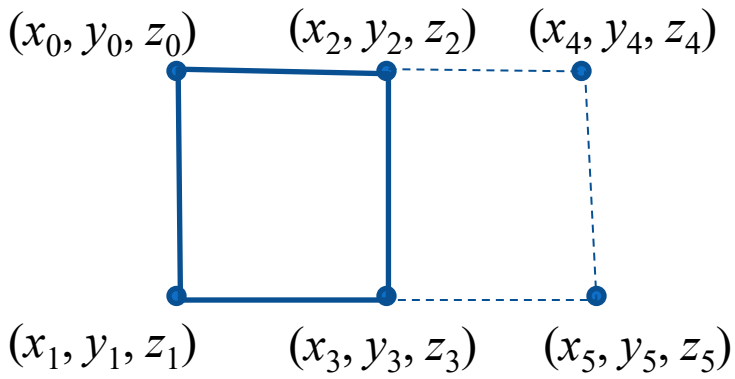
```
glBegin(GL_TRIANGLE_STRIP);
  glVertex3f(x0, y0, z0);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
  glVertex3f(x4, y4, z4);
  glVertex3f(x5, y5, z5);
glEnd();
```



# Quad Strips

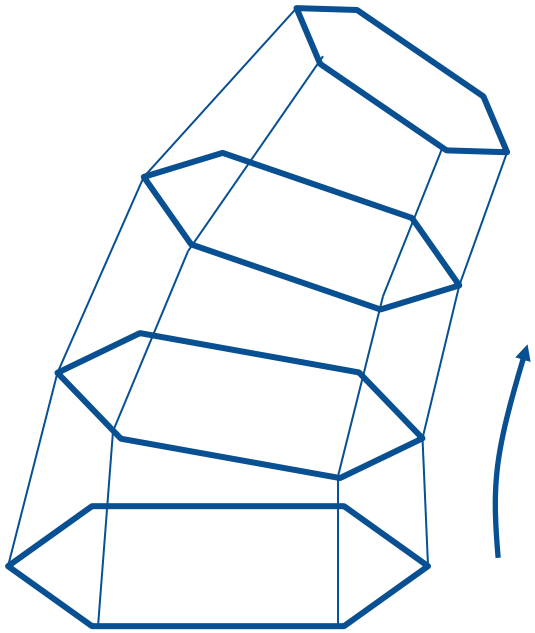
- The first four vertices form the first quad. **Note**: The quad is formed joining the first, second, fourth and third vertices.
- From the third vertex onwards, *every pair of vertices* form a new quad with the previous two vertices.

```
glBegin(GL_QUAD_STRIP);
  glVertex3f(x0, y0, z0);
  glVertex3f(x1, y1, z1);
  glVertex3f(x2, y2, z2);
  glVertex3f(x3, y3, z3);
  glVertex3f(x4, y4, z4);
  glVertex3f(x5, y5, z5);
glEnd();
```

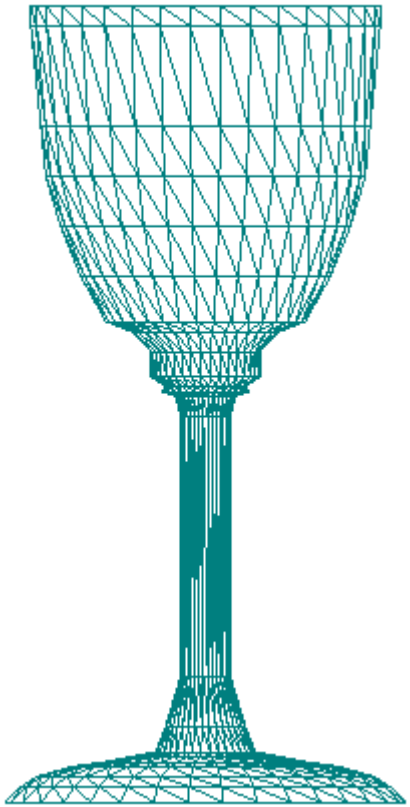


# Sweep Surfaces

Both triangle strips and quad strips are convenient primitive types for generating surfaces formed by sweeping a polygonal element in three-dimensional space.



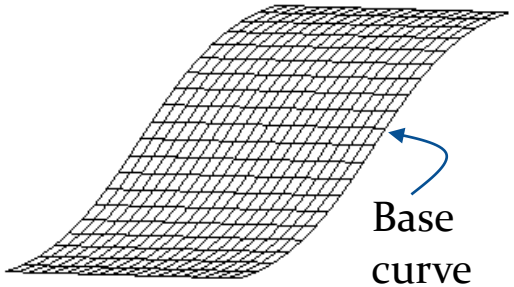
Sweeping a polygon



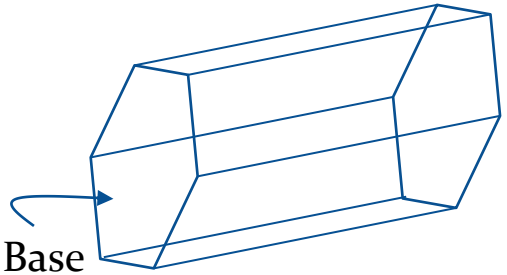
Sweeping a curve

# Sweep Representations

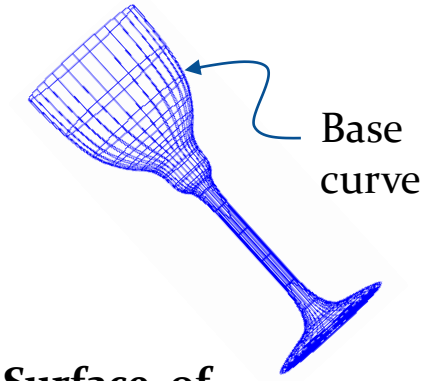
- Sweep representations are useful for both surface modeling and solid modeling.
- A large class of shapes (both surfaces and solid models) can be formed by *sweeping* or *extruding* a 2D shape (**base polygon** or **base curve**) through space.
- A **ruled surface** is generated by moving a straight line in a particular trajectory.
- A polyhedron obtained by sweeping (extruding) a polygon along a straight line is called a **prism**.




**Ruled Surface**



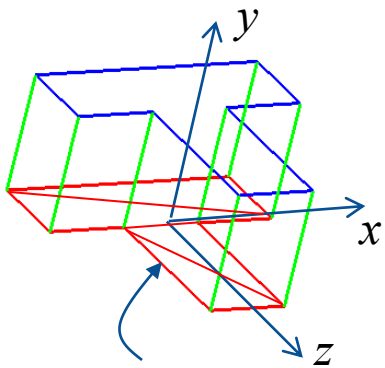

**Prism**



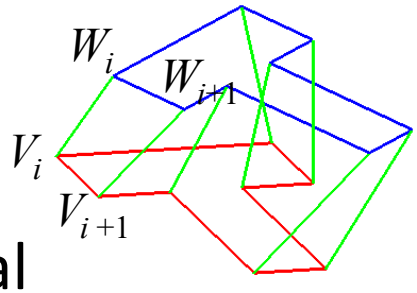

**Surface of Revolution**

# Extruded Surfaces

- Create a base polygon, centred at the origin. For proper rendering of this polygon, subdivide it into a set of triangles.
  Vertex List: $V_i = (x_i, y_i, z_i), \quad i = 0..\, n_v - 1.$
- Transform the above vertices using a matrix $\boldsymbol{M}$ to get the vertices of the extruded shape:
  Vertex List: $W_i = \boldsymbol{M} V_i, \quad i = 0..\, n_v - 1.$
- Generate the boundary surface using quadrilaterals $V_i\ V_{i+1}\ W_{i+1}\ W_i$
- Compute normal vectors of each quadrilateral using vertex coordinates.



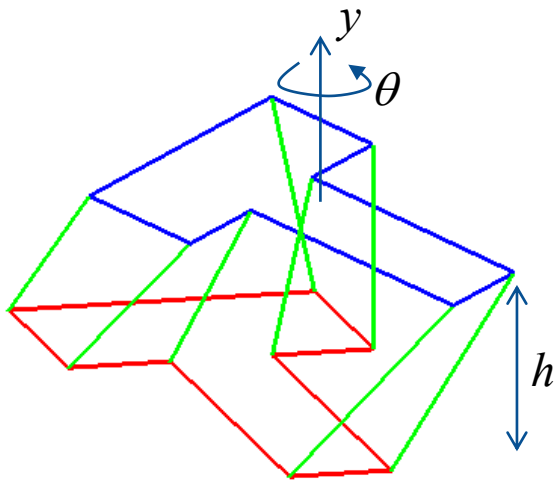
Base Polygon



# Extruded Surfaces

General transformation:

The base polygon is rotated and then translated to get the polygon for the next stage.




$(x_i, y_i, z_i)$ = Vertices of the base polygon

$(x_i', y_i', z_i')$ = Vertices of the base polygon

$\theta$ = Angle of rotation about *y*-axis

*h* = Translation of the polygon along *y*-axis.

$$x' = x_i \cos\theta + z_i \sin\theta$$

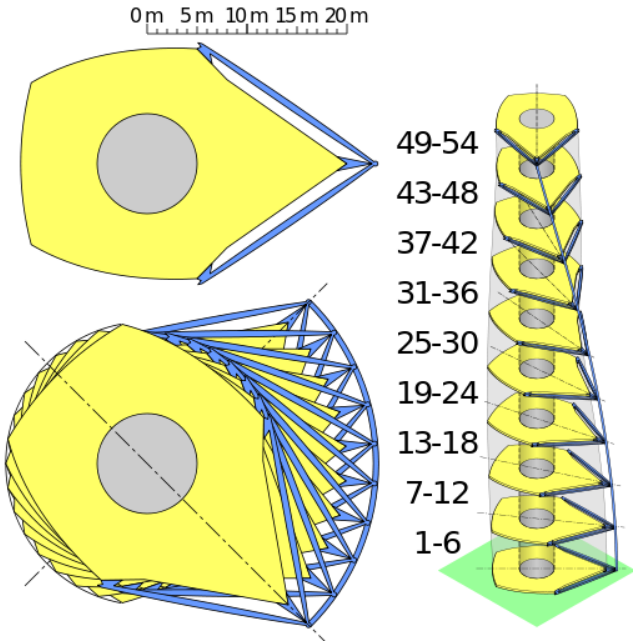
$$y' = y + h$$

$$z' = -x_i \sin\theta + z_i \cos\theta$$

# Extruded Shape: Architectural Example

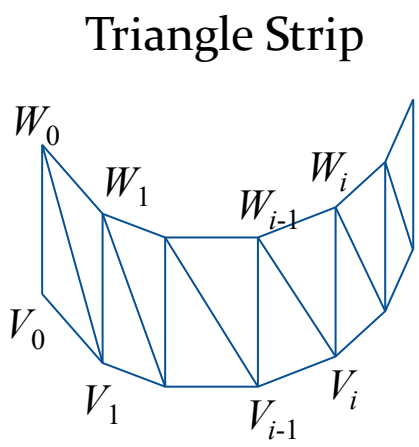
"Turning Torso", the twisted tower in Malmo, Sweden.

[Source: Wikipedia]




3D Model

# Extruded Surfaces Using Triangle Strips




```
glBegin(GL_TRIANGLE_STRIP);
  for(int i = 0; i < N; i++)
  {
    glVertex3f(vx[i], vy[i], vz[i]);
    glVertex3f(wx[i], wy[i], wz[i]);
  }
glEnd();
```

Triangle strip:

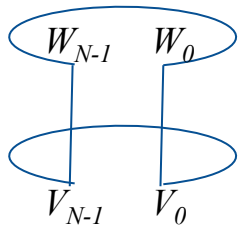
- Every point $V_i$ $(i > 0)$ generates a new triangle $W_{i-1}V_{i-1}V_i$.
- Every point $W_i$ $(i > 0)$ generates a new triangle $W_{i-1}V_iW_i$.

# Extruded Surfaces (Example)

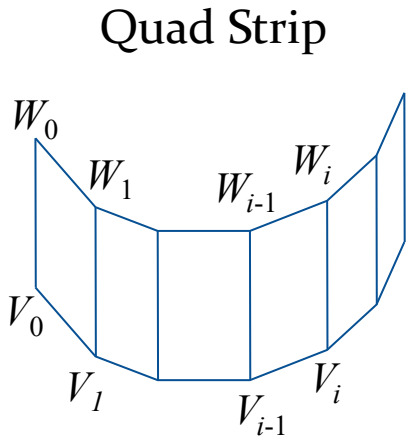
Triangle Strip (including normal computation):

```
glBegin(GL_TRIANGLE_STRIP);
  for(int i = 0; i < N; i++)
  {
      if(i > 0) normal( wx[i-1], wy[i-1], wz[i-1],
                        vx[i-1], vy[i-1], vz[i-1],
                        vx[i], vy[i], vz[i] );
      glVertex3f(vx[i], vy[i], vz[i]);
      if(i > 0) normal( wx[i-1], wy[i-1], wz[i-1],
                        vx[i], vy[i], vz[i],
                        wx[i], wy[i], wz[i] );
      glVertex3f(wx[i], wy[i], wz[i]);
  }
glEnd();
```

← Need to close the surface here.



# Extruded Surfaces Using Quad Strips




```
glBegin(GL_QUAD_STRIP);
  for(int i = 0; i < N; i++)
  {
    glVertex3f(vx[i], vy[i], vz[i]);
    glVertex3f(wx[i], wy[i], wz[i]);
  }
glEnd();
```

Quad strip:

- Every pair of points $V_i$, $W_i$ ($i > 0$) generates a new quad $W_{i-1}V_{i-1}V_iW_i$ .
- Define normal vectors (in the above example) using points $V_{i-1}V_iW_i$ .

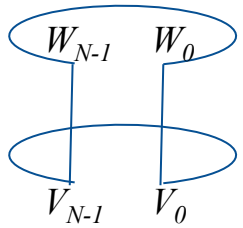
# Extruded Surfaces (Example)

Quad Strip:

- Number of vertices: *N*
- Vertices: $V_i$ = (vx[i], vy[i], vz[i]), $W_i$ = (wx[i], wy[i], wz[i]),

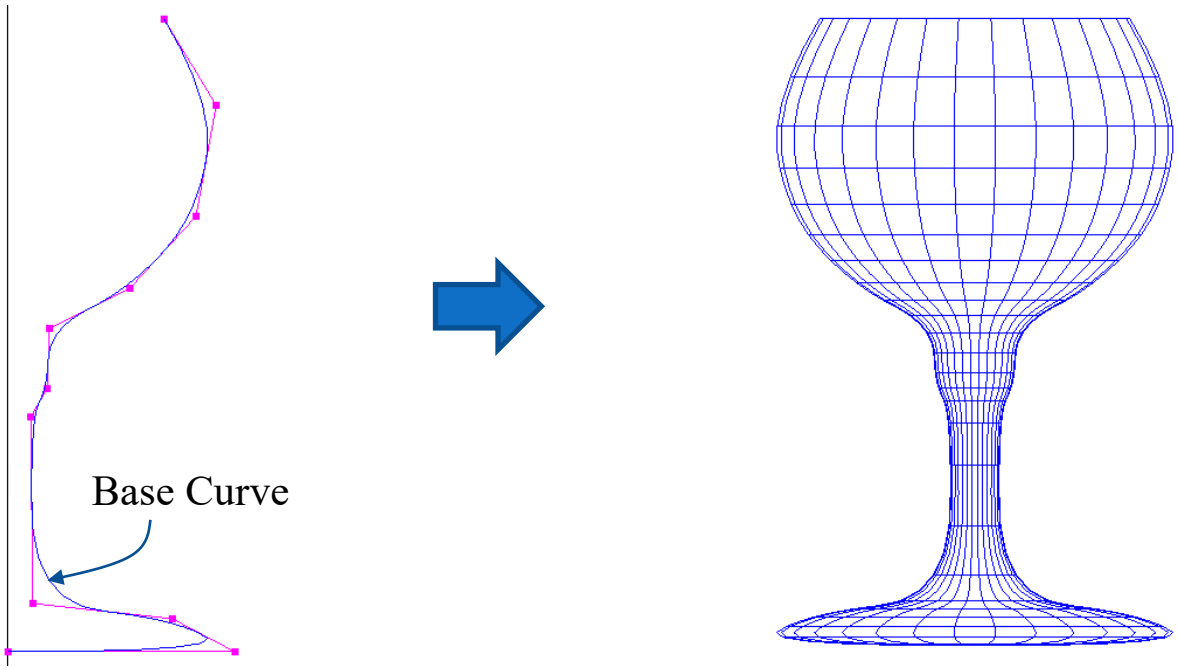
```
glBegin(GL_QUAD_STRIP);
  for(int i = 0; i < N; i++)
  {
    if(i > 0) normal( vx[i-1], vy[i-1], vz[i-1],
                      vx[i], vy[i], vz[i],
                      wx[i], wy[i], wz[i] );
    glVertex3f(vx[i], vy[i], vz[i]);
    glVertex3f(wx[i], wy[i], wz[i]);
  }
glEnd();
```

← Need to close the surface here.



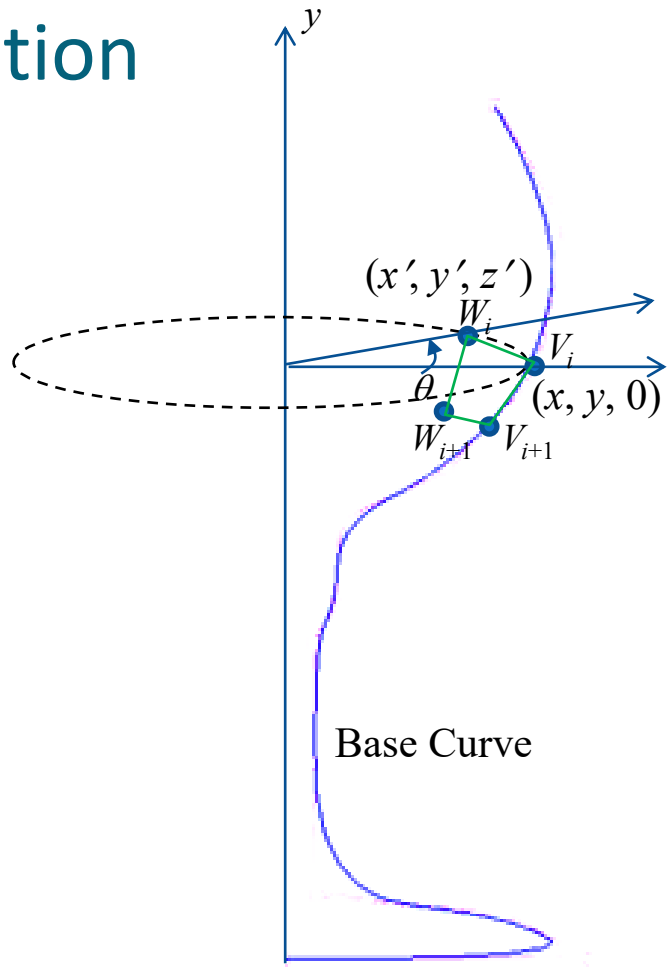
# Surface of Revolution

- A surface of revolution is obtained by revolving a planar curve about an axis, typically the *y*-axis.
- The curve (called the base curve) is usually generated using a parametric equation, such as Bezier polynomials or other types of splines (eg. Basis splines).



# Surface of Revolution

- The vertices on the base curve are given by $V_i = (x_i, y_i, 0)$ , $i = 0.. n_v - 1$.
- Transform the above vertices using a rotation matrix $\boldsymbol{M}$ to get the vertices of the revolved shape:
  $W_i = \boldsymbol{M} V_i$, $i = 0.. n_v - 1$.
- Generate the boundary surface using quadrilaterals $V_i\ V_{i+1}\ W_{i+1}\ W_i$ or quad/triangle strips $V_i\ W_i$
- Transform the surface normal vectors also by the same matrix $\boldsymbol{M}$.




$$x' = x_i \cos\theta + z_i \sin\theta$$

$$y' = y$$

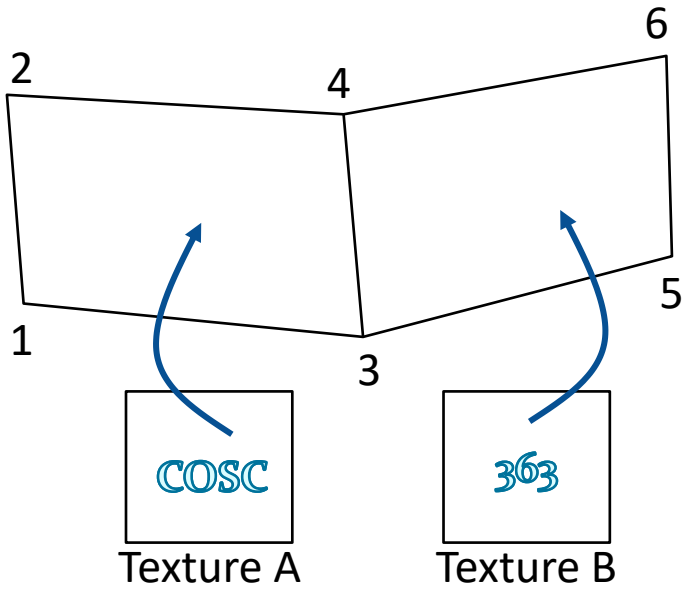
$$z' = -x_i \sin\theta + z_i \cos\theta$$

# Texturing Quad Strips: Problem

Consider a quad strip defined using six vertices:

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texId);

glBegin(GL_QUAD_STRIP);
  glTexCoord2f(0., 0.);
  glVertex3f(x1, y1, z1);
  glTexCoord2f(0., 1.);
  glVertex3f(x2, y2, z2);
  glTexCoord2f(1., 0.);
  glVertex3f(x3, y3, z3);
  glTexCoord2f(1., 1.);
  glVertex3f(x4, y4, z4);
  glVertex3f(x5, y5, z5);
  glVertex3f(x6, y6, z6);
glEnd();
```



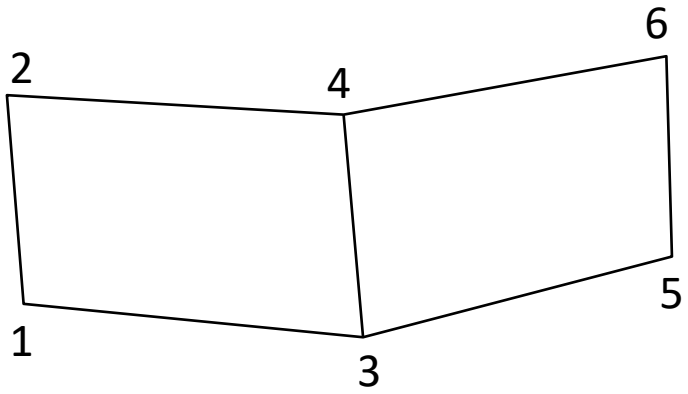

Problem: Vertices 3, 4 are shared by two quads, and each quad has an independent set of texture coords.

# Texturing Quad Strips: Solution

Create a combined texture having the same structure as the quad strip.

```
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, texId);

glBegin(GL_QUAD_STRIP);
  glTexCoord2f(0., 0.);
  glVertex3f(x1, y1, z1);
  glTexCoord2f(0., 1.);
  glVertex3f(x2, y2, z2);
  glTexCoord2f(0.5, 0.);
  glVertex3f(x3, y3, z3);
  glTexCoord2f(0.5, 1.);
  glVertex3f(x4, y4, z4);
  glTexCoord2f(1., 0.);
  glVertex3f(x5, y5, z5);
  glTexCoord2f(1., 1.);
  glVertex3f(x6, y6, z6);
glEnd();
```





A single texture image