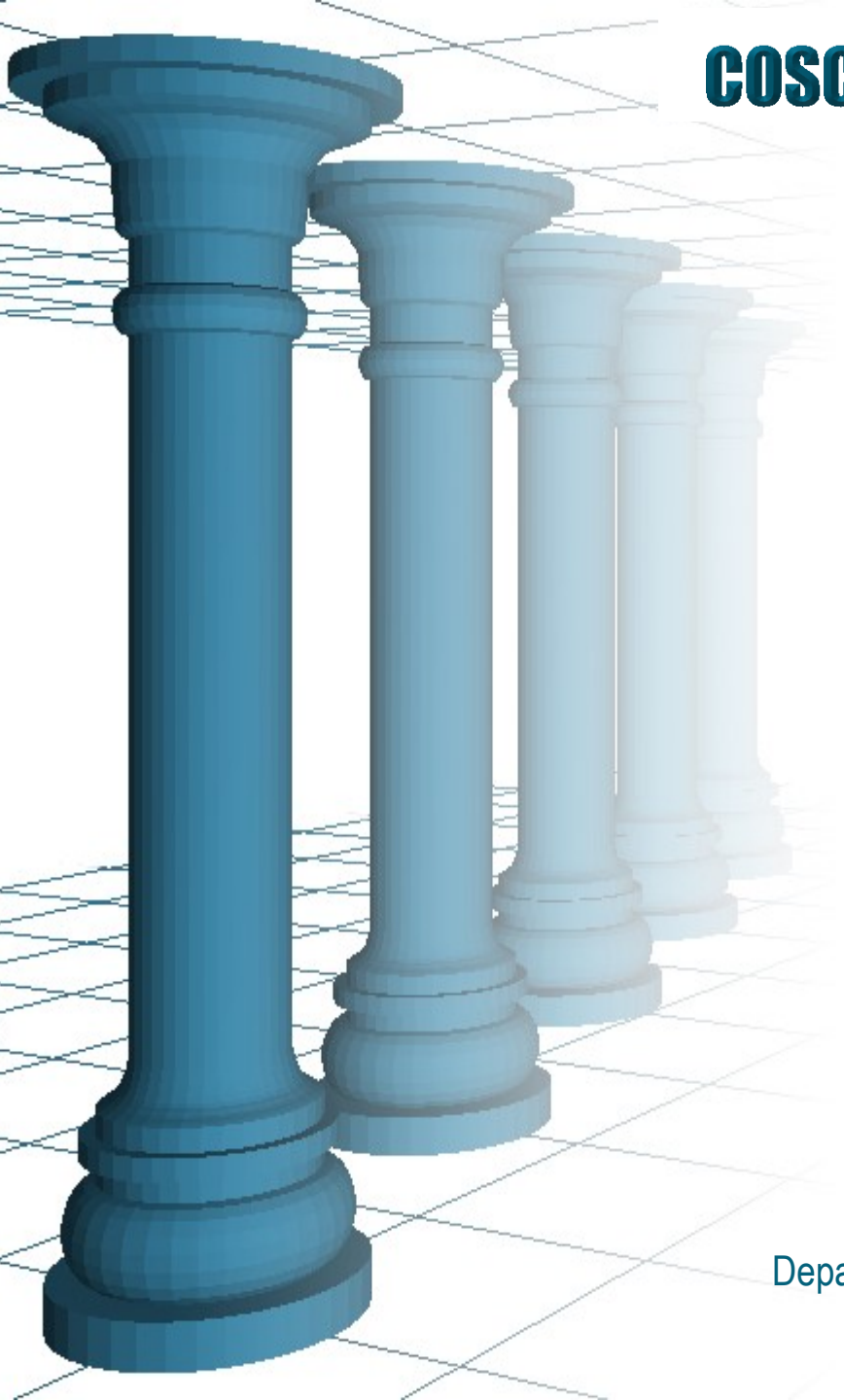


COSC363 Computer Graphics

5

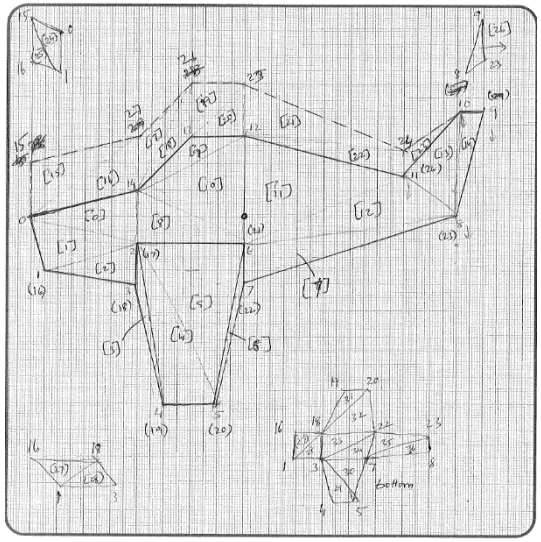
Object Modelling

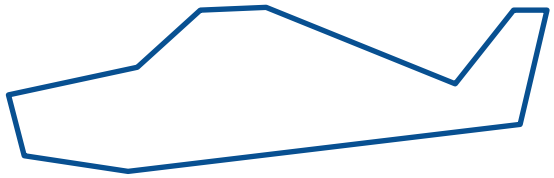
Mould your imagination!

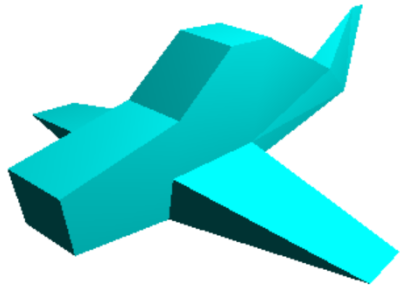
R. Mukundan (mukundan@canterbury.ac.nz)
Department of Computer Science and Software Engineering
University of Canterbury, New Zealand.

Modelling 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)
- **Subdivide** polygons into groups of triangles
- Create triangle definitions using an anti-clockwise ordering of vertex indices.







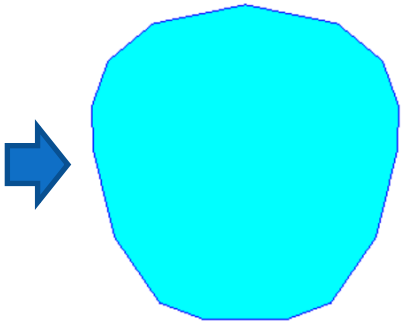
28 Vertices
52 Triangles

Polygon Definition

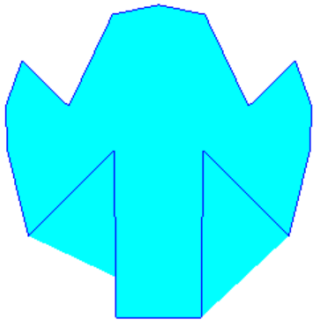
- Why do we have to triangulate polygons? Why can't we define a polygon using the primitive type GL_POLYGON ?
- A polygon defined using GL_POLYGON will be rendered correctly only if it is a **planar and convex** polygon

```
const int n =   ;   //Array size
float vx[] = ...
float vy[] = ...
float vz[n] = ...

glBegin(GL_POLYGON);
   for (int i = 0; i < n; i++)
       glVertex3f(vx[i], vy[i], vz[i]);
glEnd();
```

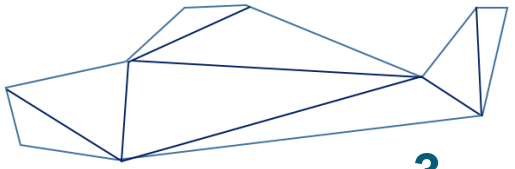


- Convex polygon
- Correctly rendered



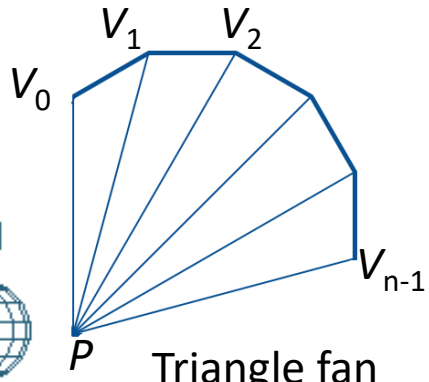
- Non-convex polygon
- Incorrectly rendered

- A polygon subdivided into triangles may be defined using GL_TRIANGLES as the primitive type.
  For large polygons, this is a time consuming and error-prone process.

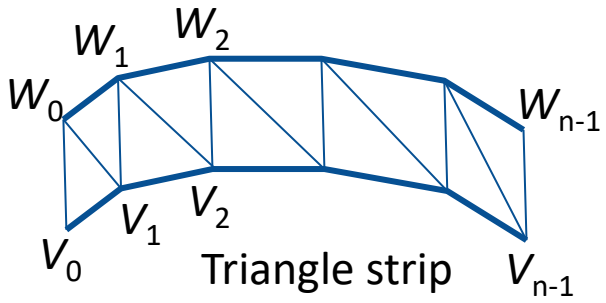


Primitive Types for Triangulation

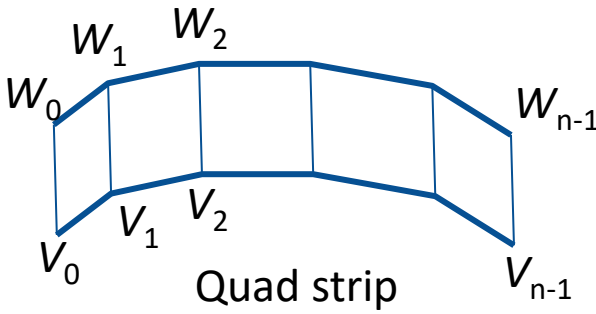
- OpenGL provides a few primitive types that are suitable for subdividing polygonal regions into triangles or quads:
  - GL_TRIANGLE_FAN
  - GL_TRIANGLE_STRIP
  - GL_QUAD_STRIP
- Advantages:
  - Provides a compact representation of a triangulation in terms of vertices of polygonal lines, without having to define each and every triangle independently.



Triangle fan



Triangle strip

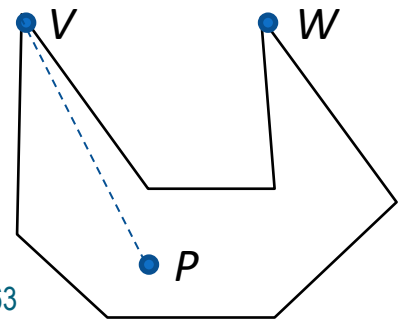


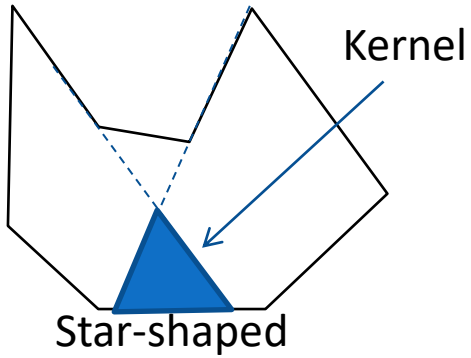
Quad strip

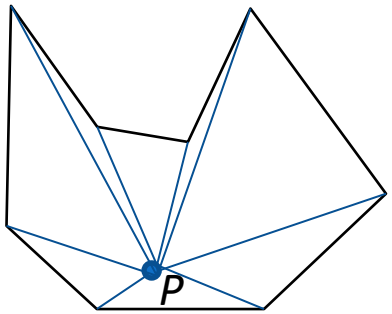
n = size of the *V* (or *W*) array.

Visibility and Kernel of a Polygon

- A point *P* inside a polygon is visible to a vertex *V* if the line segment *PV* lies entirely within the polygon.
- In the example below, *P* is visible to *V* but not to *W*.
- The set of interior points of a polygon that are visible to *all* vertices of the polygon is called the polygon's **kernel**.
- The polygon in the first figure below does not have a kernel. The second polygon has a non-empty kernel.
- A polygon with a non-empty kernel is called **star-shaped**.
- A star-shaped polygon can be triangulated from any point *P* inside the kernel.

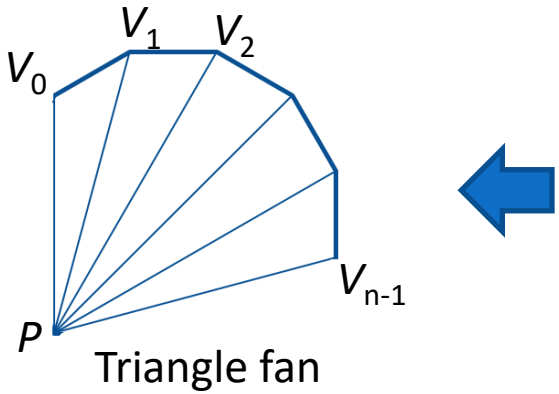






Triangulating from a Point: Triangle Fan

- The primitive type suitable for triangulating a polygon from a point is GL_TRIANGLE_FAN



Triangle fan

```
float vx[] = ...
float vy[] = ...
float vz[] = ...
glBegin(GL_TRIANGLE_FAN);
   glVertex3f(px, py, pz);
   for (int i = 0; i < n; i++)
      glVertex3f(vx[i], vy[i], vz[i]);
glEnd();
```

P

$V_0, V_1, \ldots$

Triangulating a **closed** polygon from an interior point *P*:

Primitive definition:
$P, V_0, V_1, \ldots V_{n-1}$

Primitive definition:
$P, V_0, V_1, \ldots V_{n-1}, \boldsymbol{V_0}$

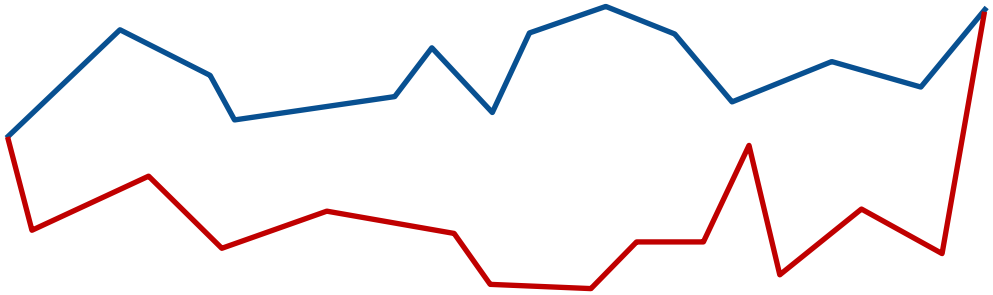
Monotone Polygons

- A polygonal chain is called **x-monotone** if the x-coordinates of its vertices increase (or stay the same) as you traverse the polygon from left to right.

x-monotone

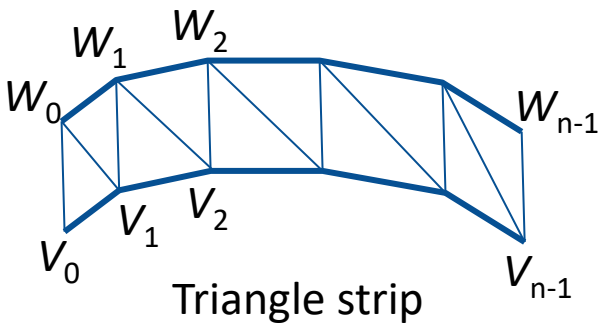
not x-monotone

- A polygon is x-monotone if its boundary can be split into two x-monotone chains.



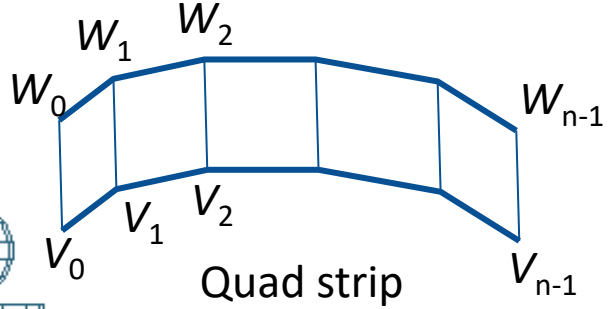
Polygon Strips

- Triangle and quad strips are convenient primitive types for the construction of surfaces between two polygonal chains with the *same number of vertices*. These primitive types are useful for triangulating monotone polygons.



Triangle strip

```
float vx[] = ...;  float wx[] = ...;
float vy[] = ...;  float wy[] = ...;
float vz[] = ...;  float wz[] = ...;
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();
```

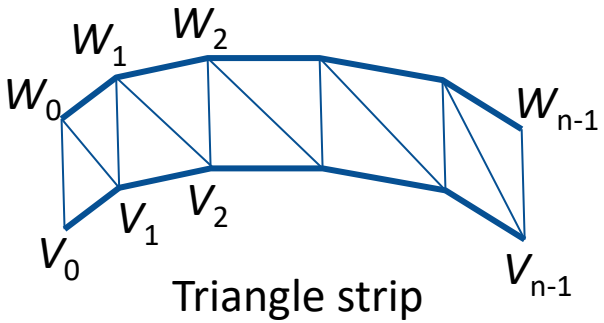


Quad strip

```
float vx[] = ...;  float wx[] = ...;
float vy[] = ...;  float wy[] = ...;
float vz[] = ...;  float wz[] = ...;
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();
```

Triangle Strip Example

- Triangle Strip:

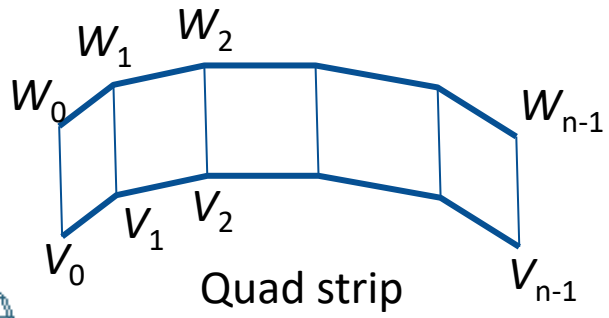


Triangle strip

Primitive Definition:

V_0
W_0
$V_1 \rightarrow$ Triangle $V_0 W_0 V_1$
$W_1 \rightarrow$ Triangle $W_0 V_1 W_1$
$V_2 \rightarrow$ Triangle $V_1 W_1 V_2$
$W_2 \rightarrow$ Triangle $W_1 V_2 W_2$
...

- Quad Strip

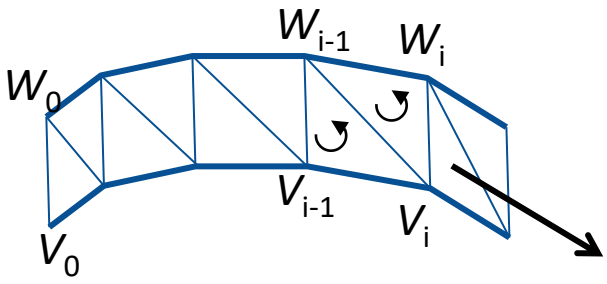


Quad strip

V_0
W_0
V_1
$W_1 \rightarrow$ Quad $W_0 V_0 V_1 W_1$
V_2
$W_2 \rightarrow$ Quad $W_1 V_1 V_2 W_2$
...

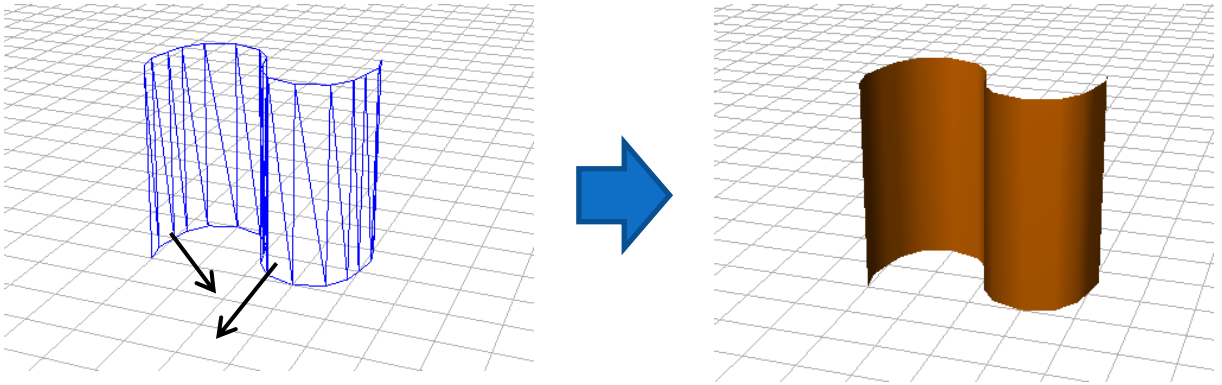
Modelling Surfaces in 3D

- Rendering in 3D space requires normal vector definitions for each triangle in a triangle strip



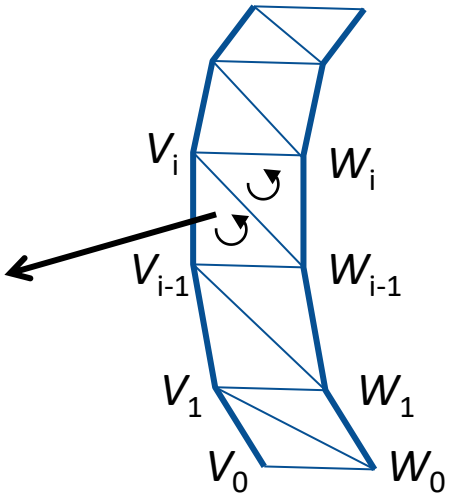
$V_i \rightarrow$ Triangle $V_{i-1}V_iW_{i-1}$
$W_i \rightarrow$ Triangle $W_{i-1}V_iW_i$

```
glBegin(GL_TRIANGLE_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-1], wy[i-1], wz[i-1]);
   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();
```



Modelling Surfaces in 3D

- Normal vectors for triangle strips: Another example to show that the order of vertices in the computation of the normal vector depends on the way in which the polygonal vertices V_i, W_i are specified.

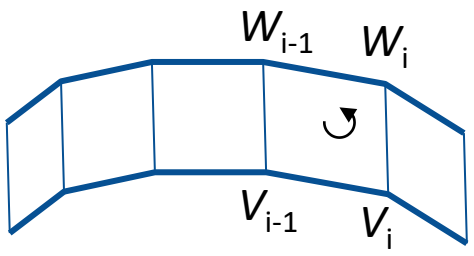



```
glBegin(GL_TRIANGLE_STRIP);
for (int i = 0; i < n; i++) {
   if(i > 0) normal(vx[i-1], vy[i-1], vz[i-1],
                    wx[i-1], wy[i-1], wz[i-1],
                    vx[i],   vy[i],   vz[i]);
   glVertex3f(vx[i], vy[i], vz[i]);
   if(i > 0) normal(wx[i-1], wy[i-1], wz[i-1],
               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

$V_i \rightarrow$ Triangle $V_{i-1}W_{i-1}V_i$
$W_i \rightarrow$ Triangle $W_{i-1}W_iV_i$

Modelling Surfaces in 3D

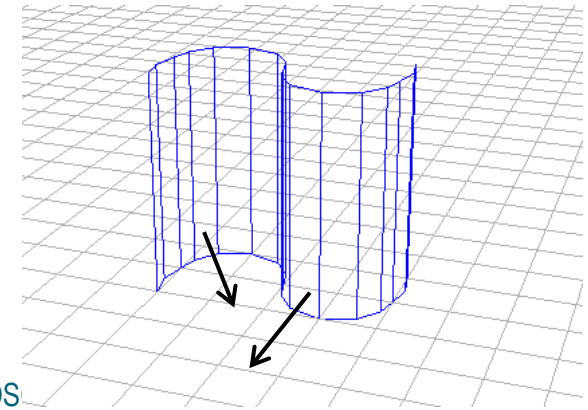
- Rendering in 3D space requires normal vector definitions for each triangle in a triangle strip

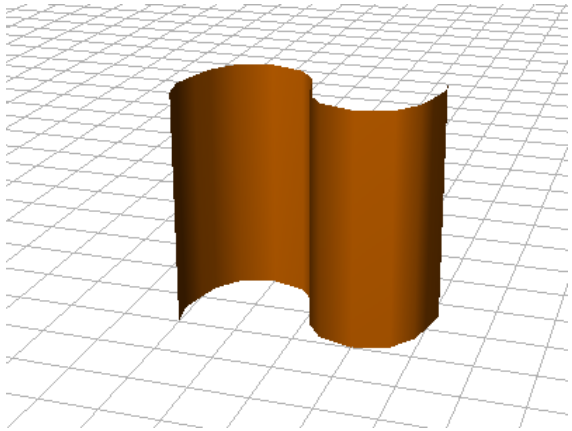


Quad strip

$W_i \rightarrow$ Quad $W_{i-1}V_{i-1}V_iW_i$

```
glBegin(GL_QUAD_STRIP);
for (int i = 0; i < n; 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();
```



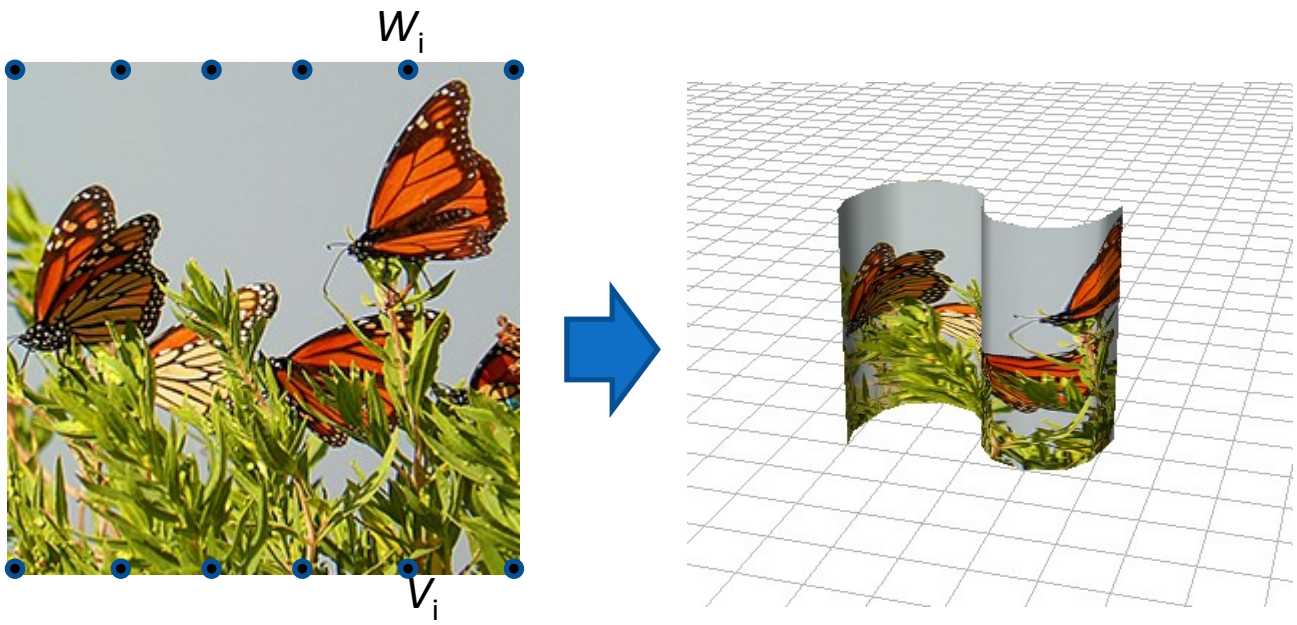


Modelling Surfaces in 3D

- Texturing: Suppose we map the points V_i and W_i evenly along the bottom and top edges of the texture, then
  - Texture coordinates for V_i : $i/(n\text{-}1)$, 0
  - Texture coordinates for W_i : $i/(n\text{-}1)$, 1

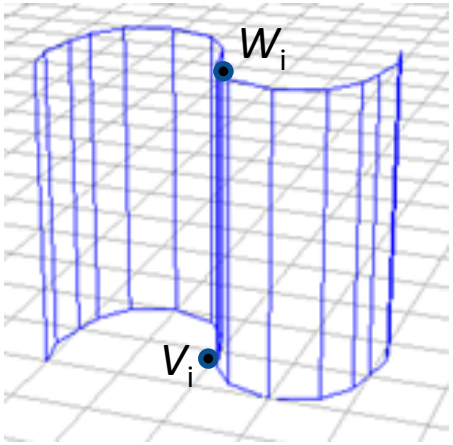
```
glTexCoord2f( (float)i/(float)(n-1), 0);
glVertex3f(vx[i], vy[i], vz[i]);

glTexCoord2f( (float)i/(float)(n-1), 1);
glVertex3f(wx[i], wy[i], wz[i]);
```

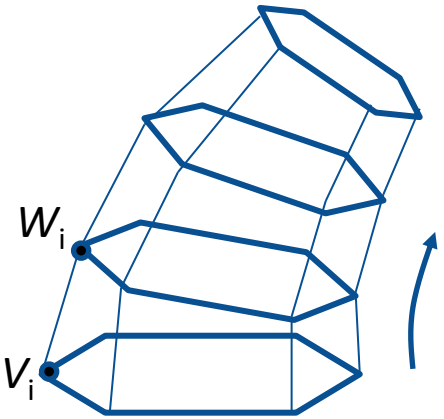


Modelling Surfaces in 3D

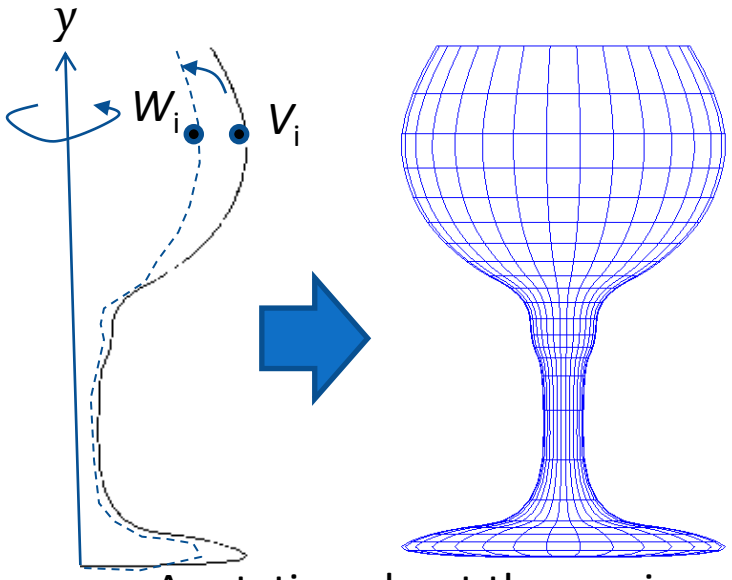
In many surface modelling applications, the polygonal line W_i is a transformed version of V_i. The transformation is repeated multiple times (with the W_i polygon as the new V_i) to generate different layers or slices of a 3D model.



A translation along y-direction



A scale transformation and a rotation



A rotation about the y-axis

Surfaces generated by repeatedly transforming a polygonal element are known as **Sweep Surfaces.**

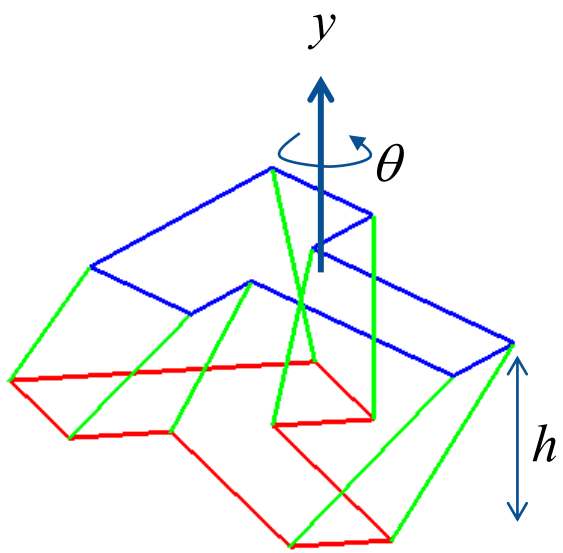
Sweep Surfaces

- In general, the vertices V_i of a polygonal line may be transformed using a matrix $\boldsymbol{M}$ to get the vertices W_i :

  $W_i = \boldsymbol{M}V_i, \quad i = 0..\, n-1.$

- We require the coordinates of W_i , and therefore we cannot use OpenGL transformation functions.
- Most commonly used transformation is a rotation about the *y*-axis by an angle θ :

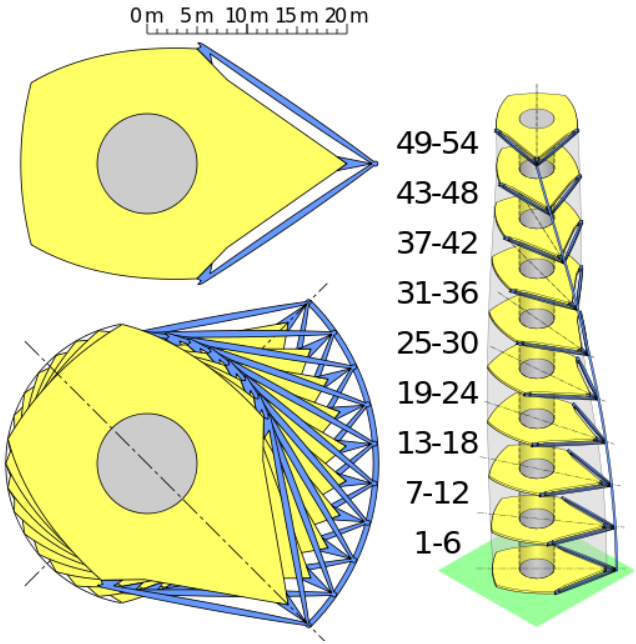
$$x' = x_i \cos\theta + z_i \sin\theta$$
$$y' = y$$
$$z' = -x_i \sin\theta + z_i \cos\theta$$



Sweep Surface: Architectural Example

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

[Source: Wikipedia]

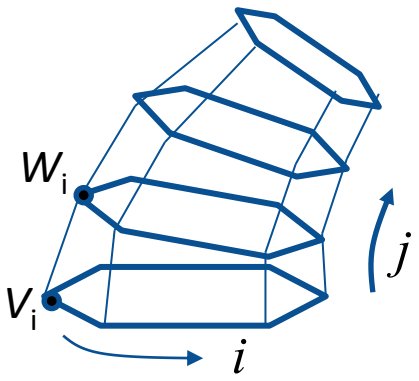


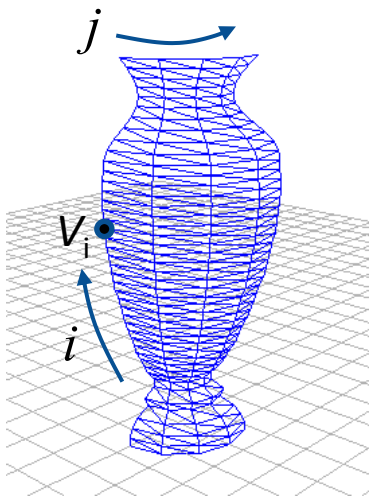
A rotation about the y-axis and a translation along y-direction.

3D Model

Sweep Surfaces

- Multiple layers of a sweep surface are generated by copying the transformed array *W* back to *V*, and repeating the processes of transforming vertices and creating triangle strips using the transformed points. Pseudo-code:
  - Initialize V_i array
  - For $j = 0 \ldots$ Nslice-1: (Nslice = number of slices)
  - Transform V_i into W_i ($i = 0 \ldots n\text{-}1$)
  - Construct triangle (or quad) strip using V_i and W_i
  - Copy W_i to V_i

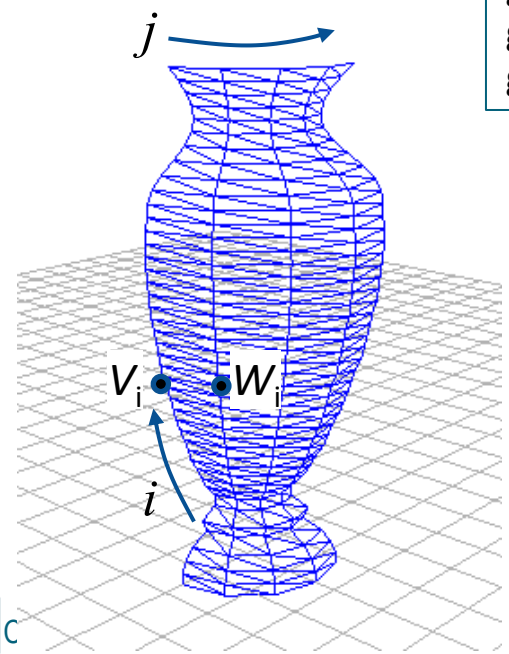


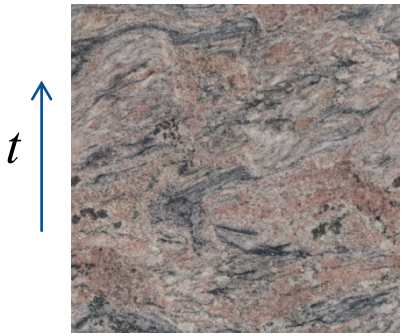


Texturing Sweep Surfaces

- Two parameters (indices) associated with sweep surfaces:
  - *i* (vertex index) varies along each polygonal line: *i* = 0 ... *n*-1.
  - *j* (slice index): *j* = 0...Nslice – 1.
- Map the above values to *s* and *t*:
  - E.g: *s* = *j*/Nslice, *t* = *i* /(n-1)

```
glTexCoord2f((float)j/NSlice,  (float)i/(N-1));
glVertex3f(vx[i], vy[i], vz[i]);
glTexCoord2f((float)(j+1)/NSlice,  (float)i/(N-1));
glVertex3f(wx[i], wy[i], wz[i]);
```





Texture

Trivia Quiz