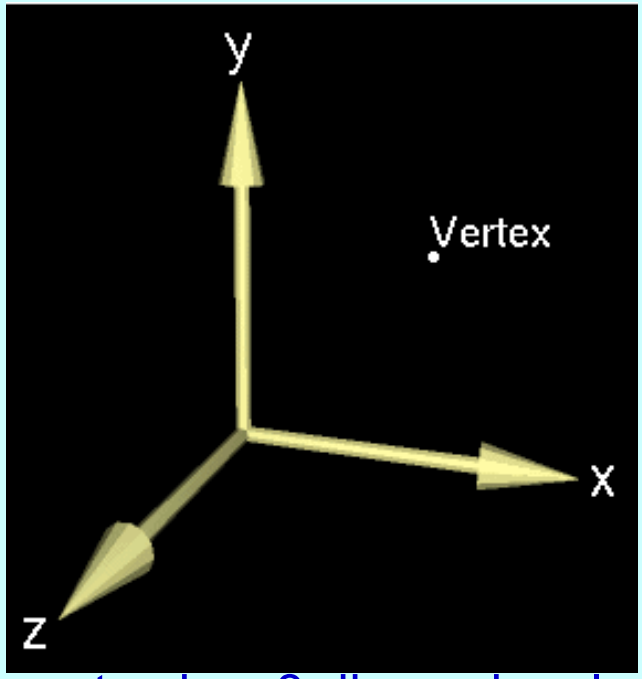
Building Models

Overview

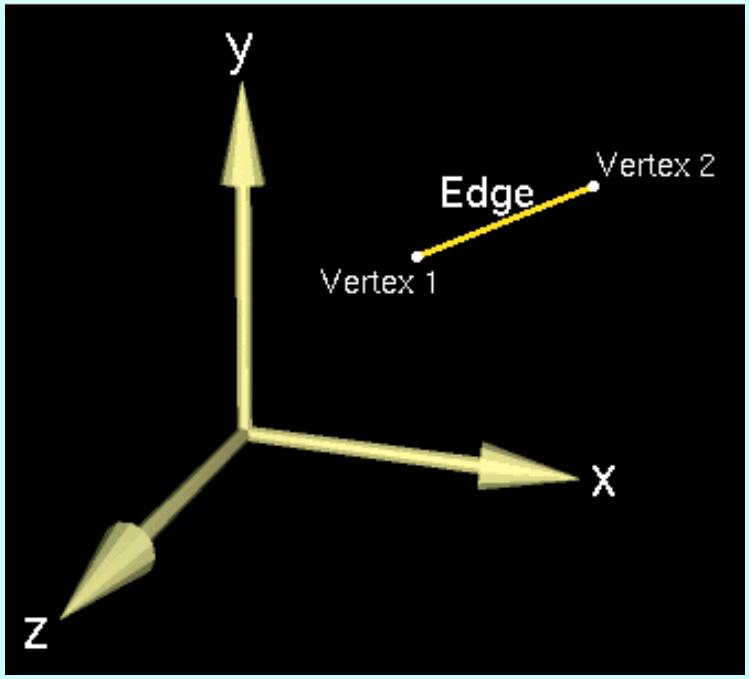
- Polygonal Meshes
- 3D Shapes
- Extruded Shapes
- Modeling Solids

Object representation

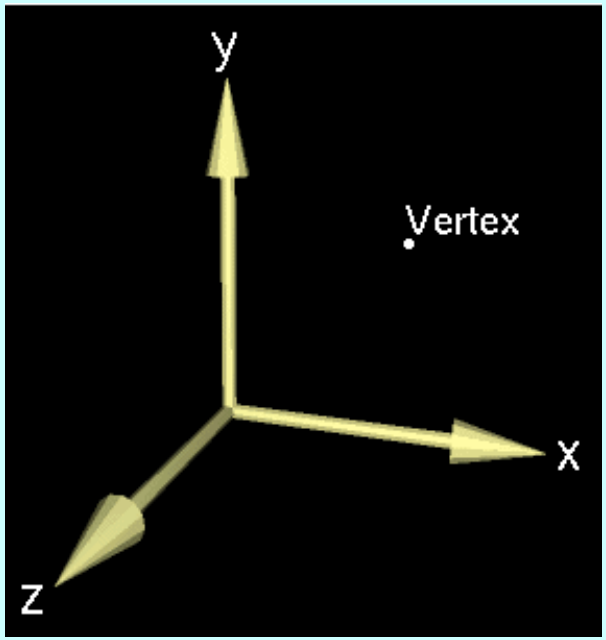
A collection of 3D objects are arranged, along with lights, to form a scene. This system uses a hierarchical scene representation:

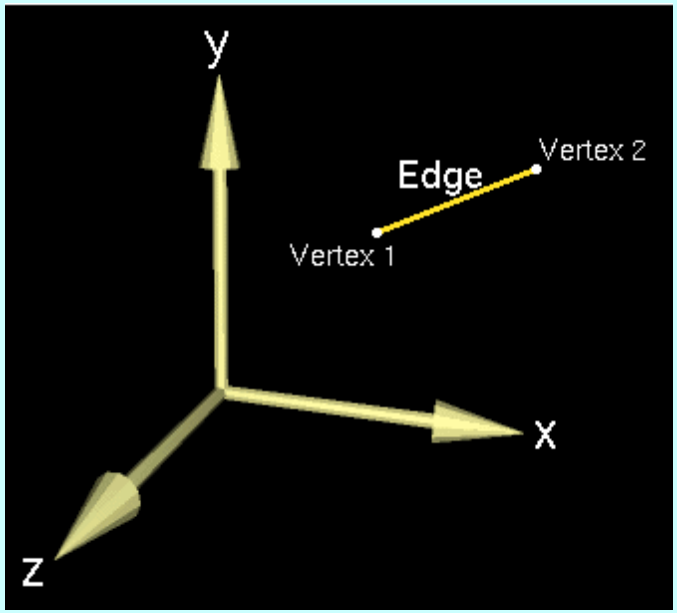


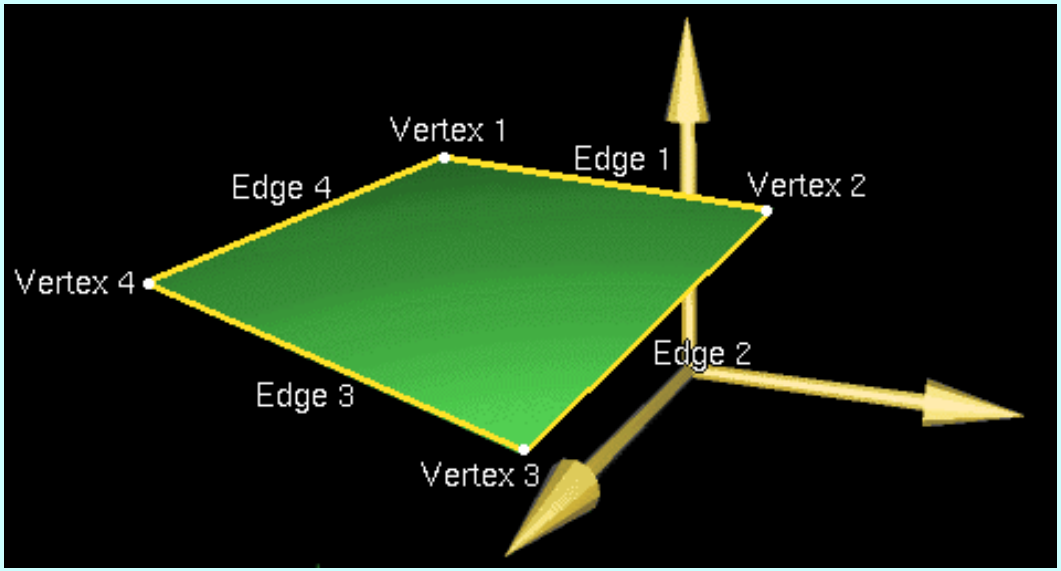
A vertex is a 3 dimensional point, with an x, y and z coordinate



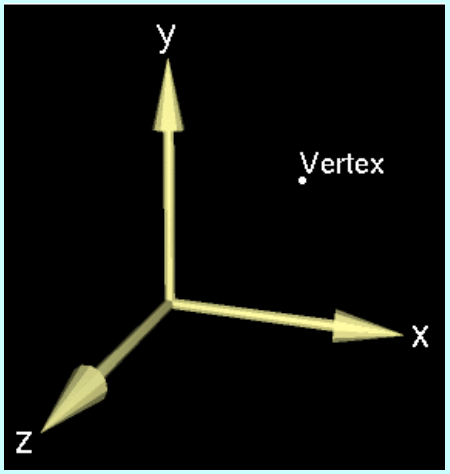
An edge is a line joining 2 vertices

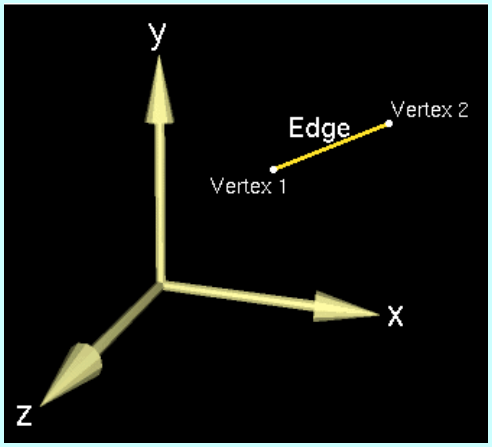


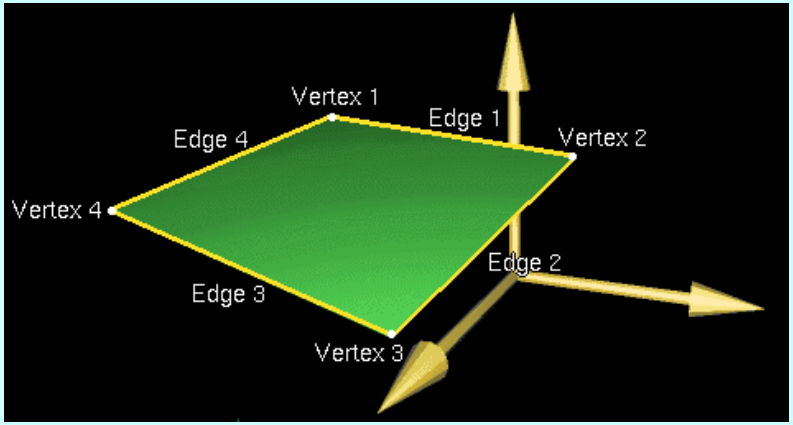


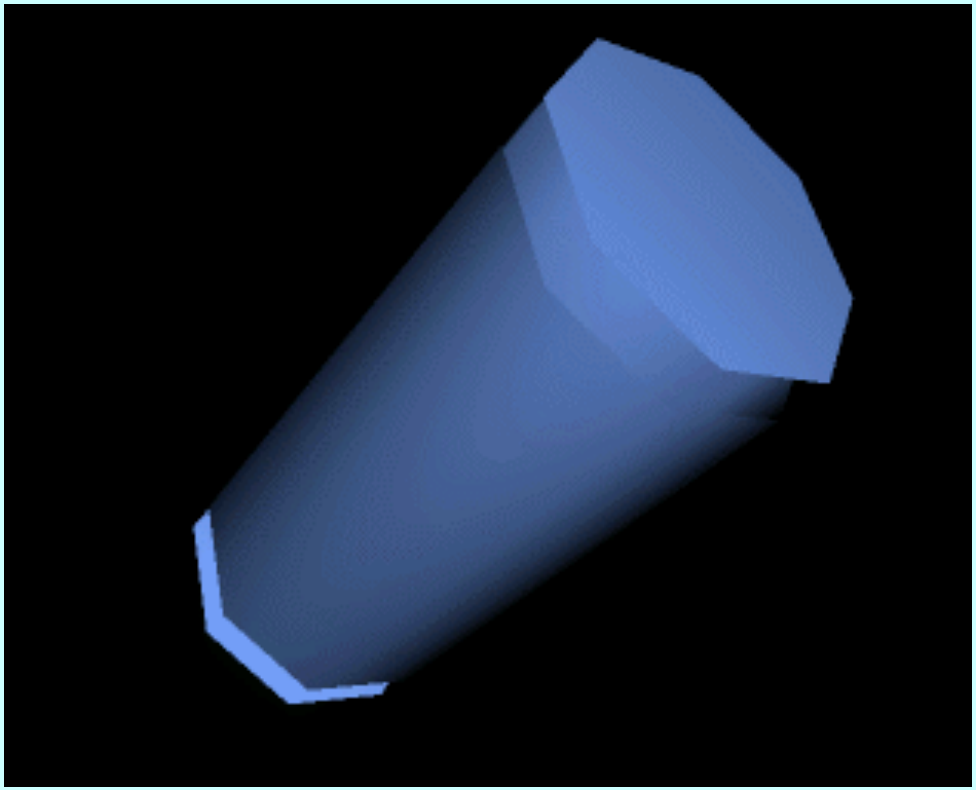


A polygon is made up of 3 (a triangle) or more edges

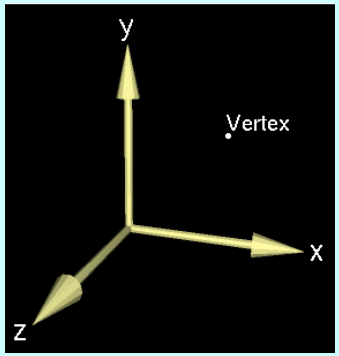


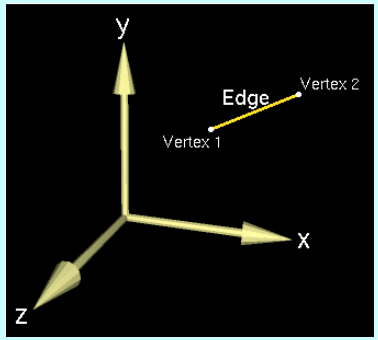


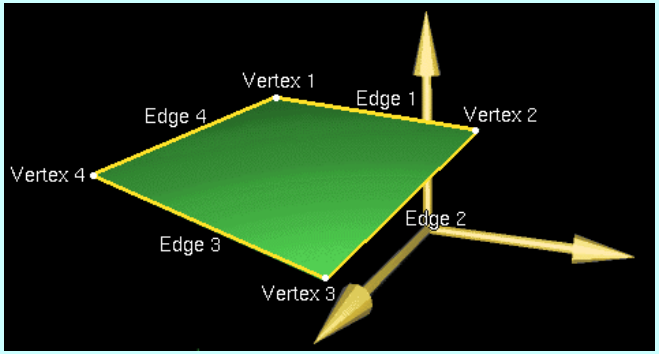


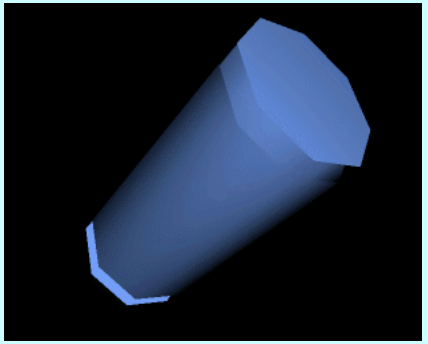


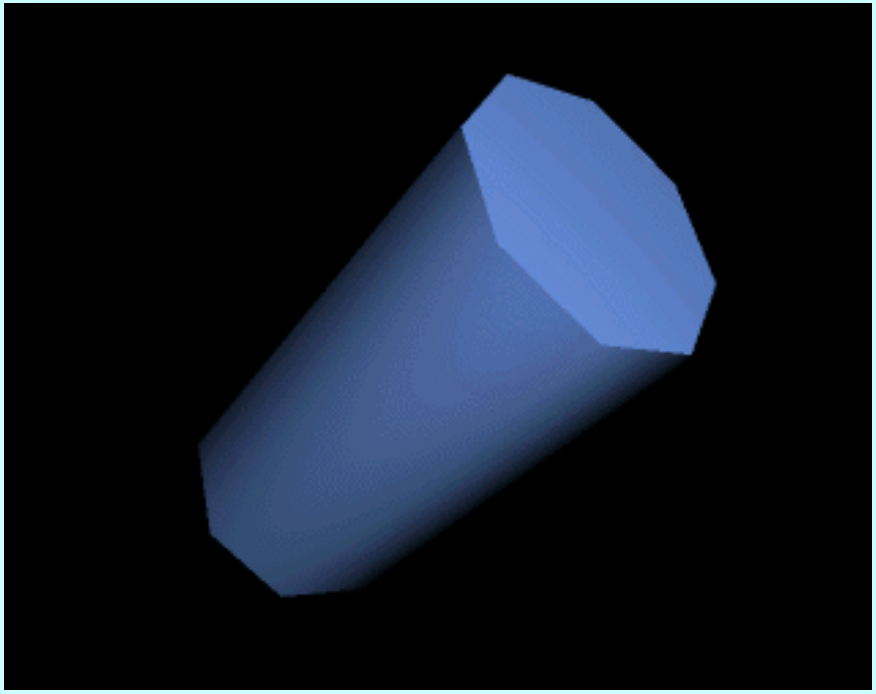
A surface is a collection of polygons which approximate a single surface of the object being modeled



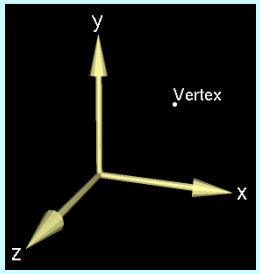


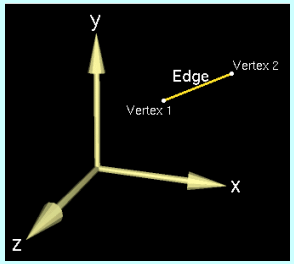


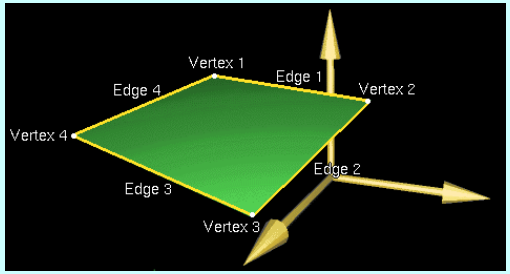


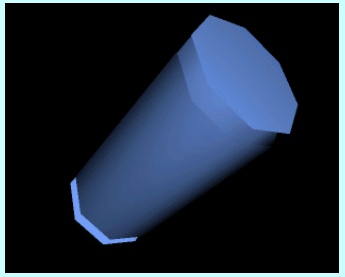


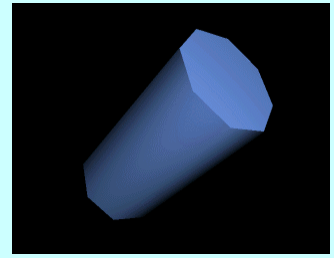
Surfaces are then grouped into an object

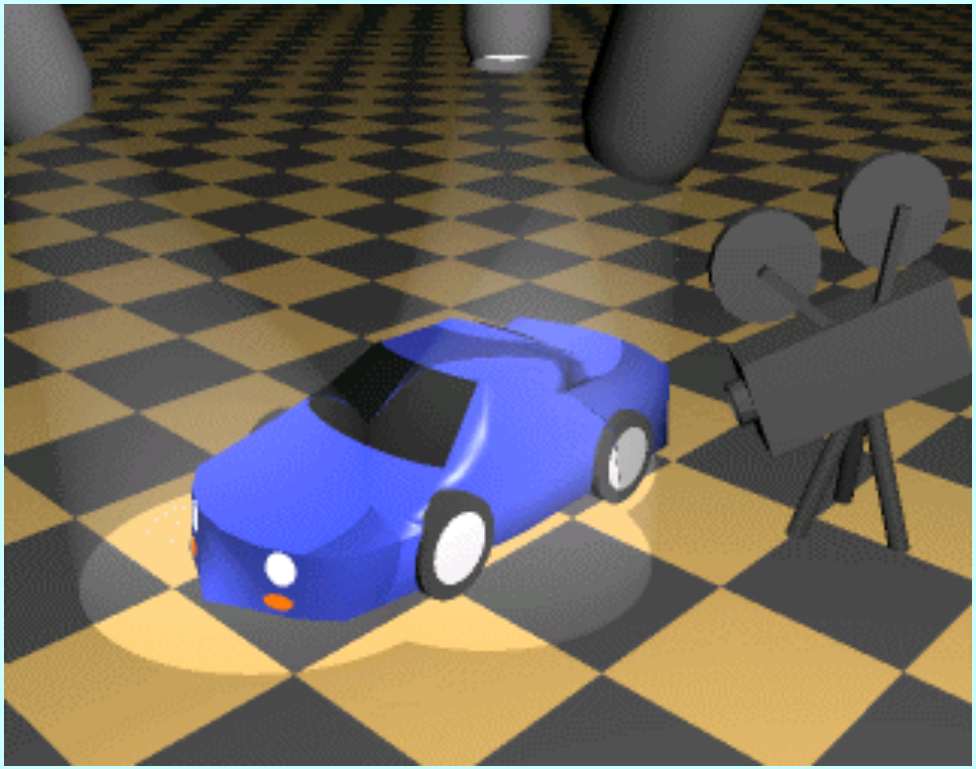




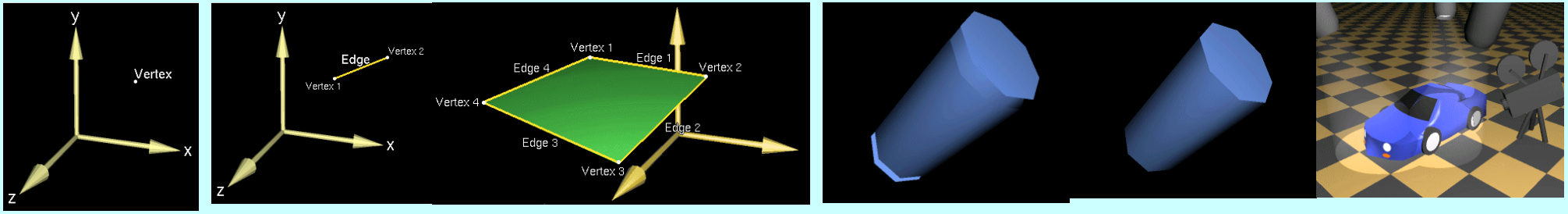






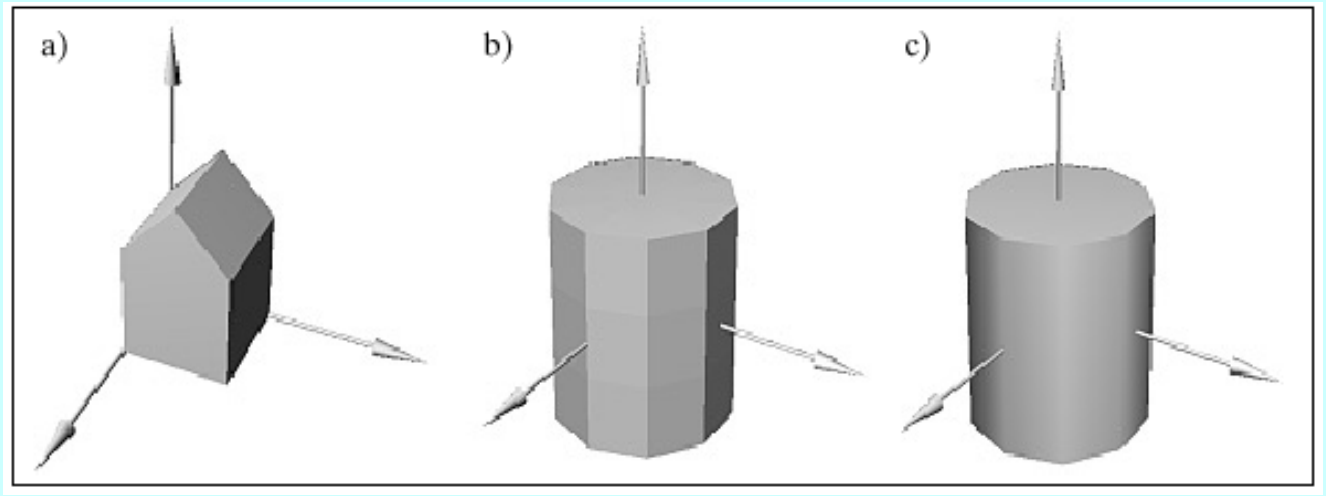


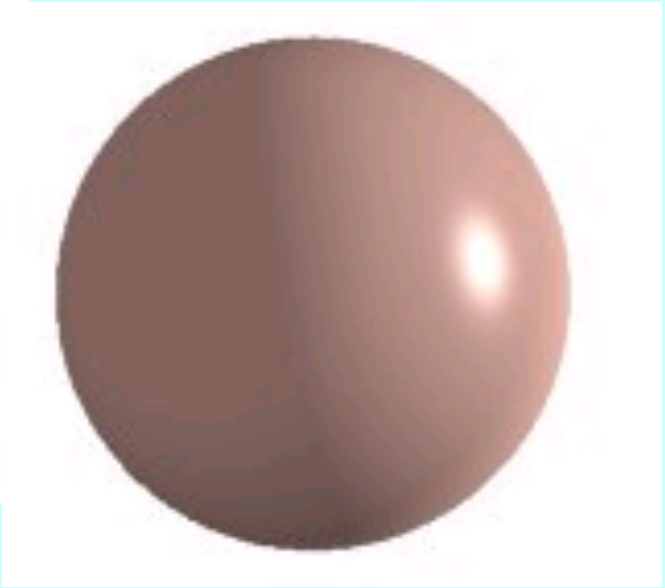
Objects are grouped into a scene, along with light sources, and a virtual camera



Once a scene has been created, it is passed into the rendering pipeline, and an image from the view of the virtual camera comes out of the other end

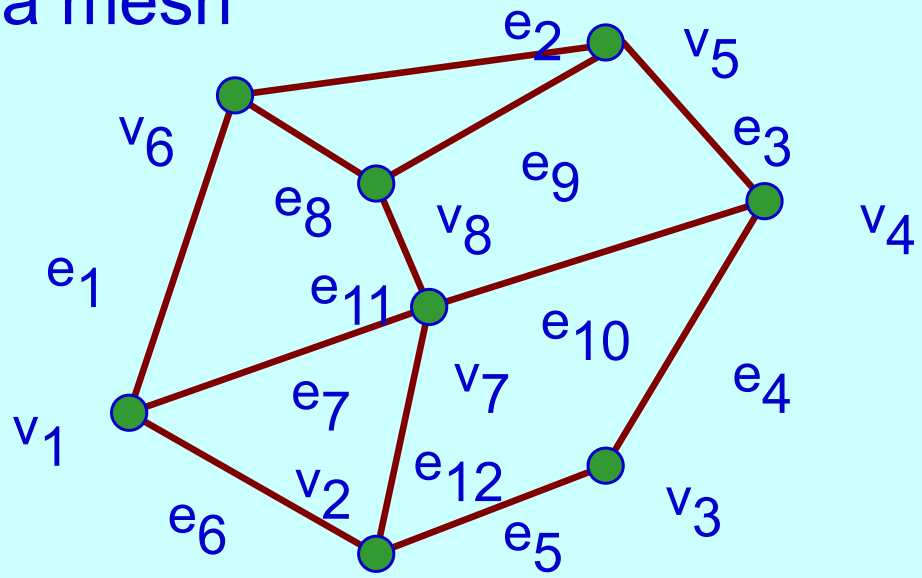
Polygonal meshes are simply collections of polygons, or ‘faces’ that together form the ‘skin’ of an object.





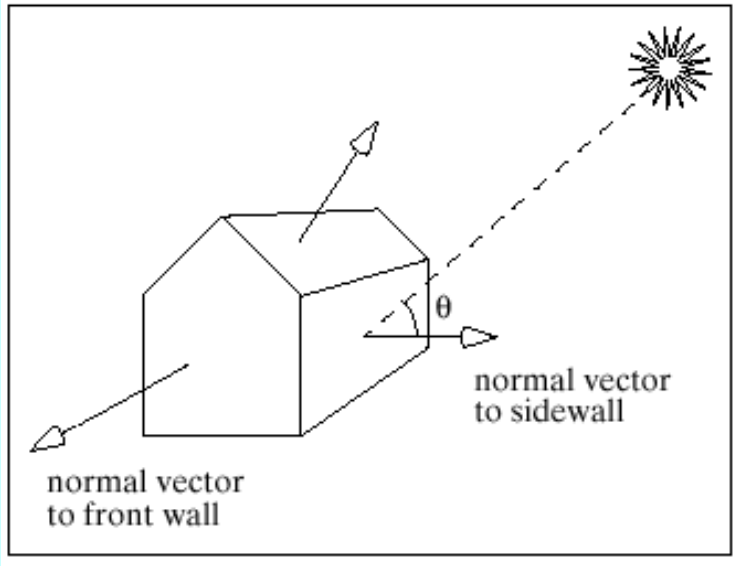
Representing a Mesh

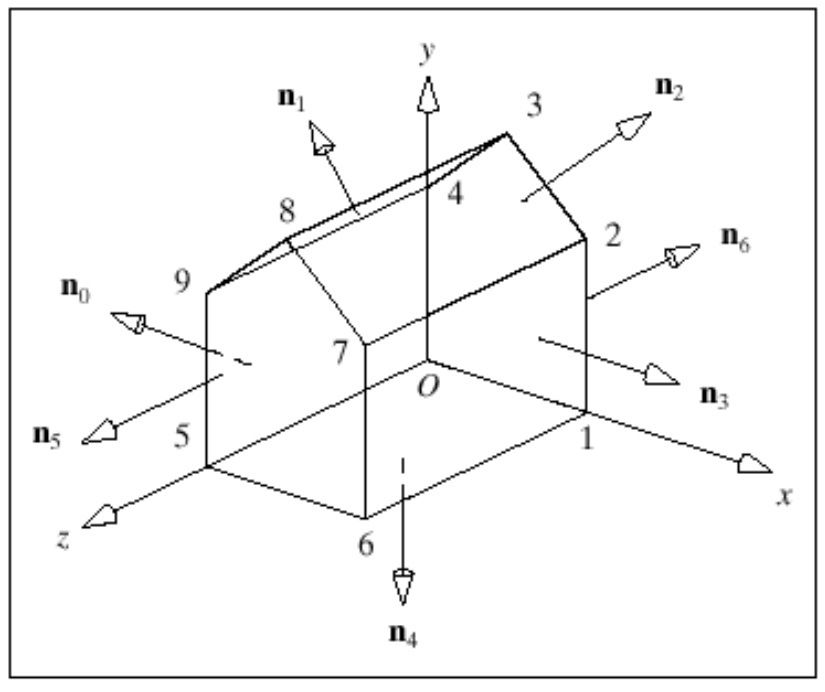
- Consider a mesh



- There are 8 nodes and 12 edges
  - 5 interior polygons
  - 6 interior (shared) edges
- Each vertex has a location $v_i = (x_i\ y_i\ z_i)$

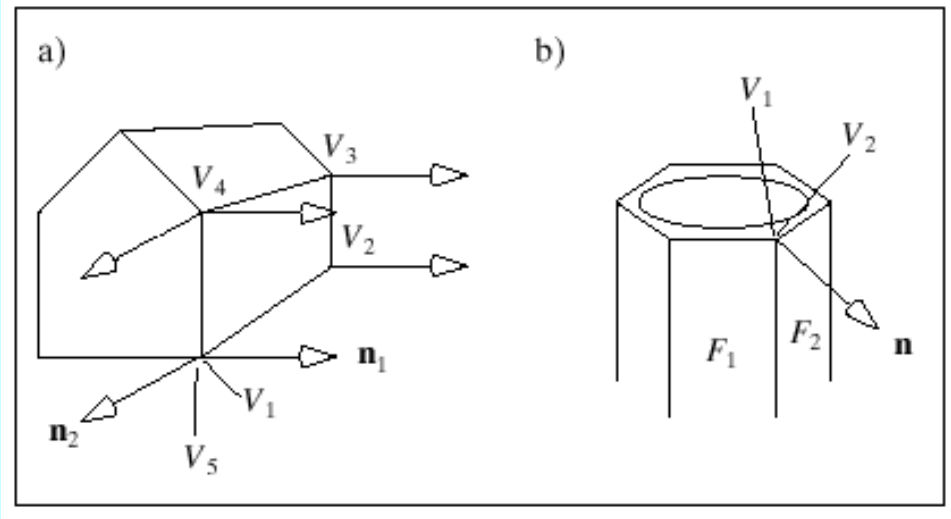
A polygonal mesh is given by a list of polygons, along with the normal vector.




y
n1
n2
3
4
8
2
n6
n0
9
7
O
n3
n5
5
1
x
z
6
n4

Vertex Normals versus Face Normals

It turns out to be highly advantageous to associate a ‘normal vector’ with each vertex of a face, rather than one vector for an entire face.



To store information about a mesh

- Use a list of all the polygons, and for each one, keep a list of vertices and a list of normals

  Redundant and bulky

- Use three separate lists: a vertex list, a normal list and a face list.

  Vertex list contains locational or **geometric** information
  The normal list contains **orientation** information
  The face list contains connectivity or **topological** information

vertex	x	y	z
0	0	0	0
1	1	0	0
2	1	1	0
3	0.5	1.5	0
4	0	1	0
5	0	0	1
6	1	0	1
7	1	1	1
8	0.5	1.5	1
9	0	1	1

normal	n_x	n_y	n_z
0	–1	0	0
1	–0.707	0.707	0
2	0.707	0.707	0
3	1	0	0
4	0	–1	0
5	0	0	1
6	0	0	–1

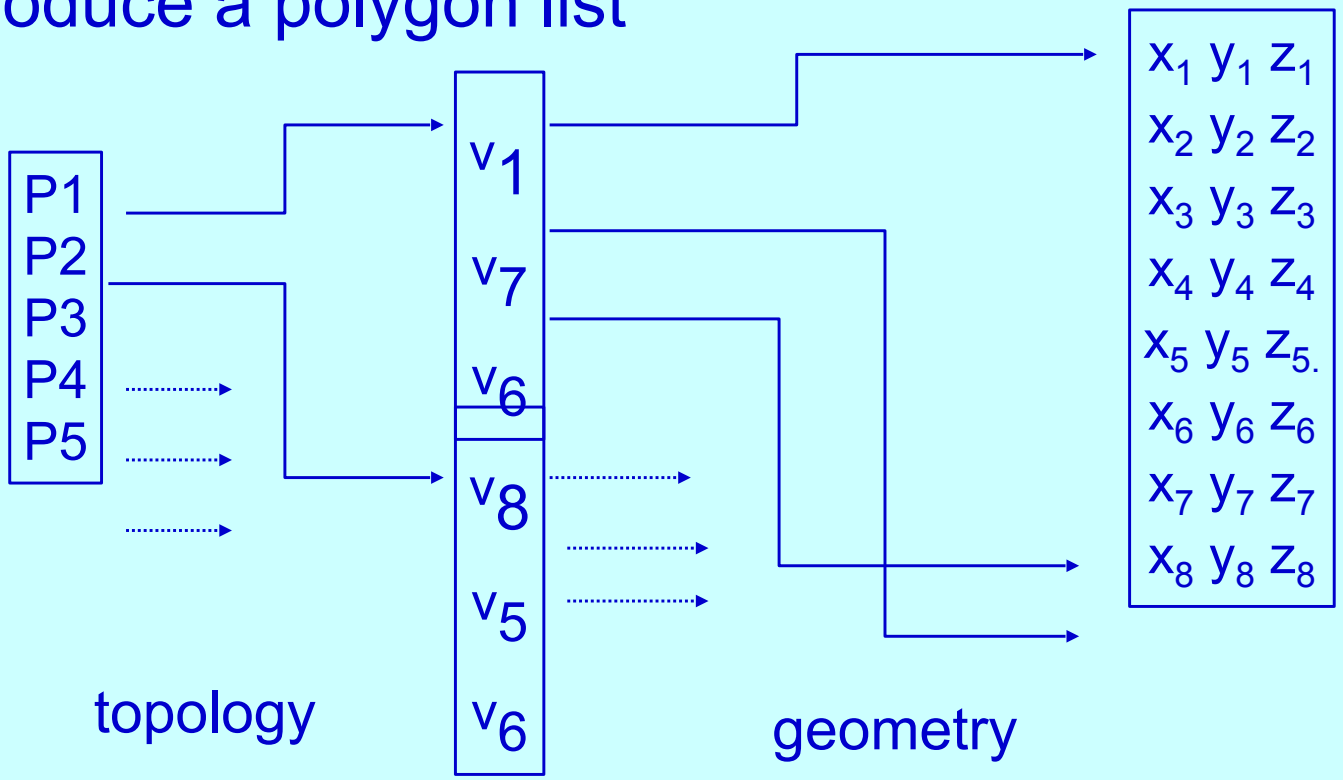
face	vertices	associated normal
0 (left)	0,5,9,4	0,0,0,0
1 (roof left)	3,4,9,8	1,1,1,1
2 (roof right)	2,3,8,7	2,2,2,2
3 (right)	1,2,7,6	3,3,3,3
4 (bottom)	0,1,6,5	4,4,4,4
5 (front)	5,6,7,8,9	5,5,5,5,5
6 (back)	0,4,3,2,1	6,6,6,6,6

Geometry vs Topology

- Generally it is a good idea to look for data structures that separate the geometry from the topology
  - Geometry: locations of the vertices
  - Topology: organization of the vertices and edges
  - Example: a polygon is an ordered list of vertices with an edge connecting successive pairs of vertices and the last to the first
  - Topology holds even if geometry changes

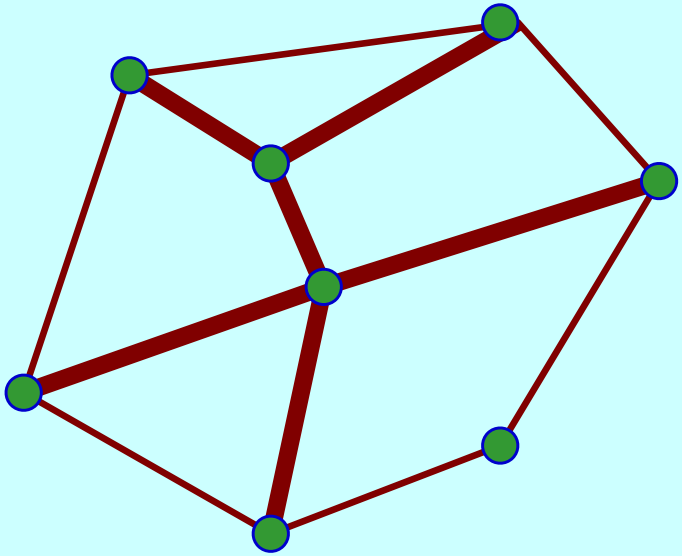
Vertex Lists

- Put the geometry in an array
- Use pointers from the vertices into this array
- Introduce a polygon list



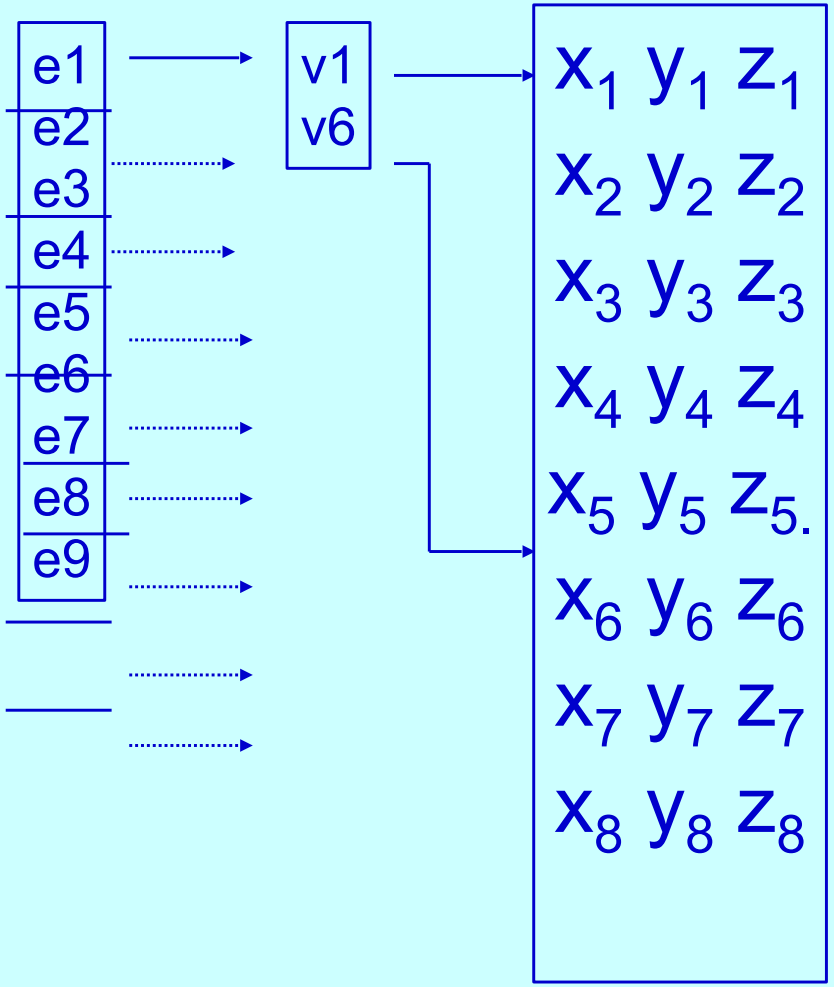
Shared Edges

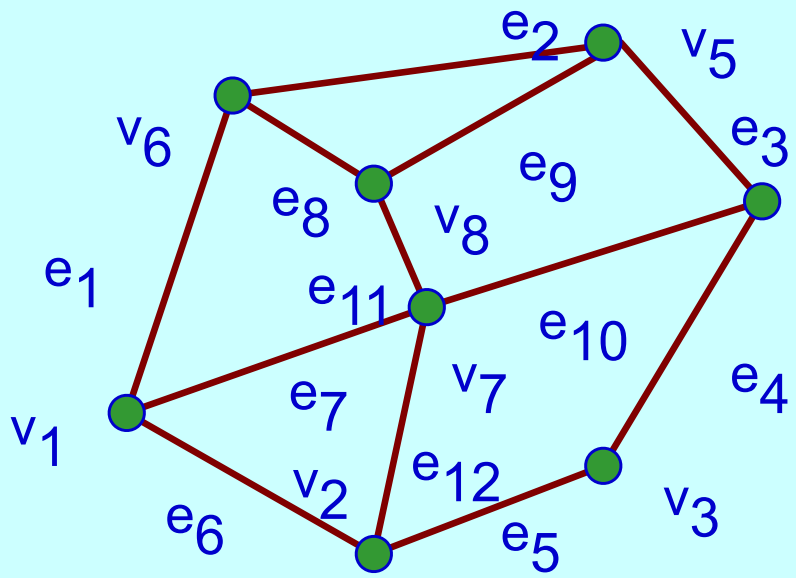
- Vertex lists will draw filled polygons correctly but if we draw the polygon by its edges, shared edges are drawn twice



- Can store mesh by *edge list*

Edge List

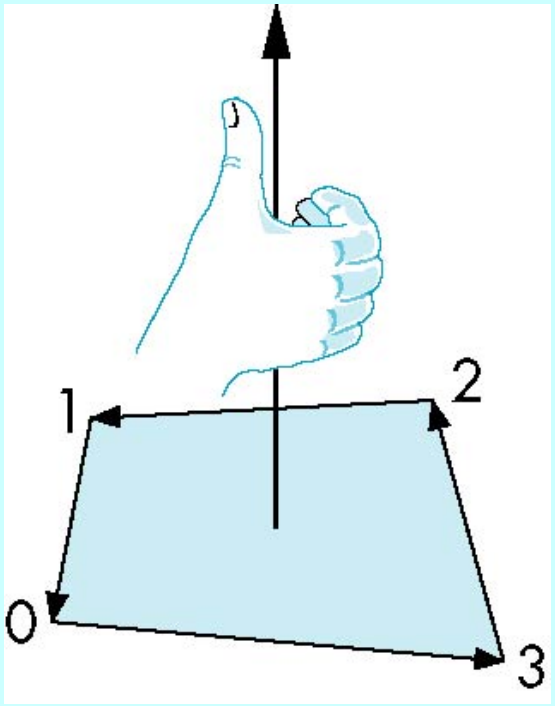




Note polygons are not represented

Inward and Outward Facing Polygons

- The order $\{v_1, v_6, v_7\}$ and $\{v_6, v_7, v_1\}$ are equivalent in that the same polygon will be rendered by OpenGL but the order $\{v_1, v_7, v_6\}$ is different
- The first two describe *outwardly facing* polygons
- Use the *right-hand rule* = counter-clockwise encirclement of outward-pointing normal
- OpenGL treats inward and outward facing polygons differently



Finding the Normal Vectors

Take any three adjacent points on the face, V_1, V_2, and V_3.
Compute the normal as the cross product $m = (V_1 - V_2) \times (V_3 - V_2)$

Problems:

- Vectors might be nearly parallel (cross products will be small)
- Face might not be planar

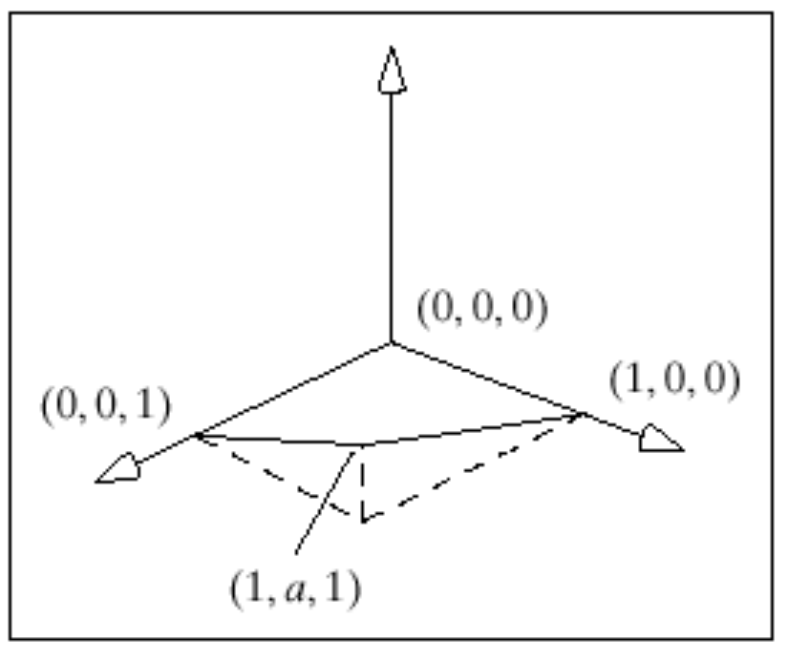
Finding the Normal Vectors

Take any three adjacent points on the face, V_1, V_2, and V_3.
Compute the normal as the cross product $m = (V_1 - V_2) \times (V_3 - V_2)$

$$m_x = \sum_{i=0}^{N-1} (y_i - y_{next(i)})(z_i + z_{next(i)}),$$

$$m_y = \sum_{i=0}^{N-1} (z_i - z_{next(i)})(x_i + x_{next(i)}),$$

$$m_z = \sum_{i=0}^{N-1} (x_i - x_{next(i)})(y_i + y_{next(i)}),$$



A nonplanar polygon

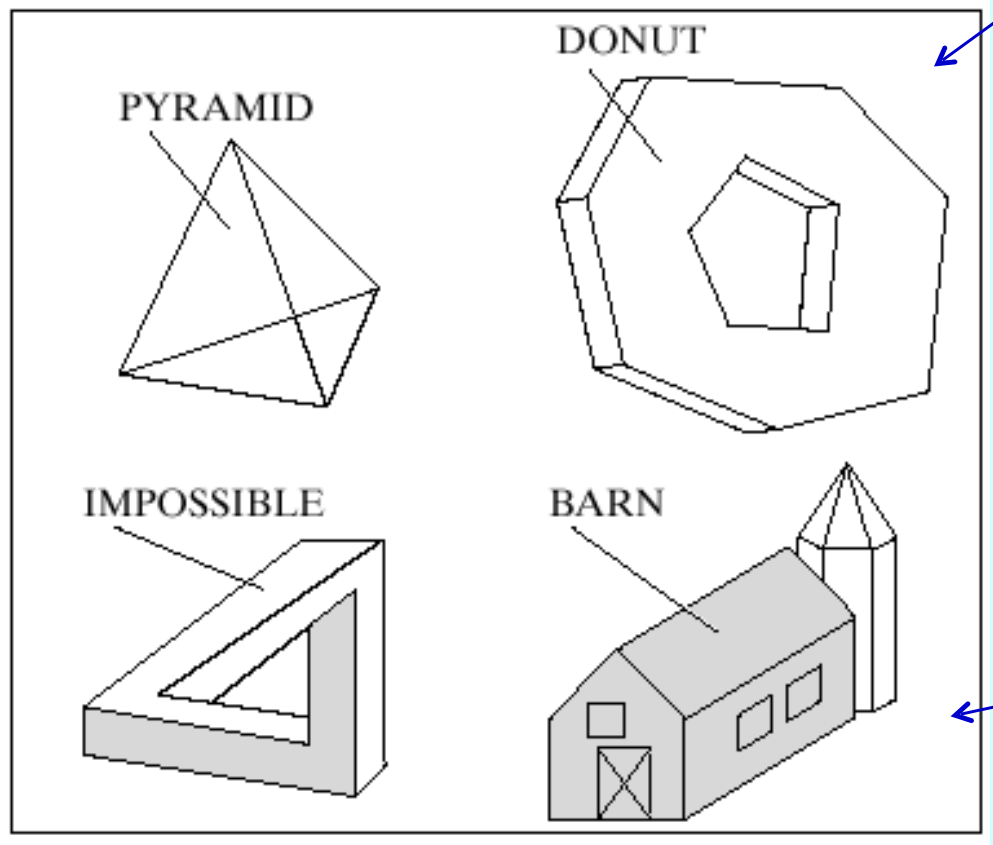
If the vertices of the polygon are traversed (as i increases) in a **CounterClockWise** direction as seen from the outside the polygon, then the normal m points toward the outside of the face.

Properties of Meshes

- **Solidity** – A mesh represents a solid object if its faces together enclose a positive and finite amount of space.
  - **Connectedness** – A mesh is **connected** if an unbroken path along polygon edges exists between any two vertices.
- **Simplicity** – A mesh is **simple** if the object it represents is solid and has no holes through it.
- **Planarity** – A mesh is **planar** if every face of the object it represents is a **planar** polygon: The vertices of each face lie in a single plane.
- **Convexity** – The mesh represents a **convex** object if the line connecting any two points within the object lies wholly inside the object.

connected and solid,
not simple, not convex

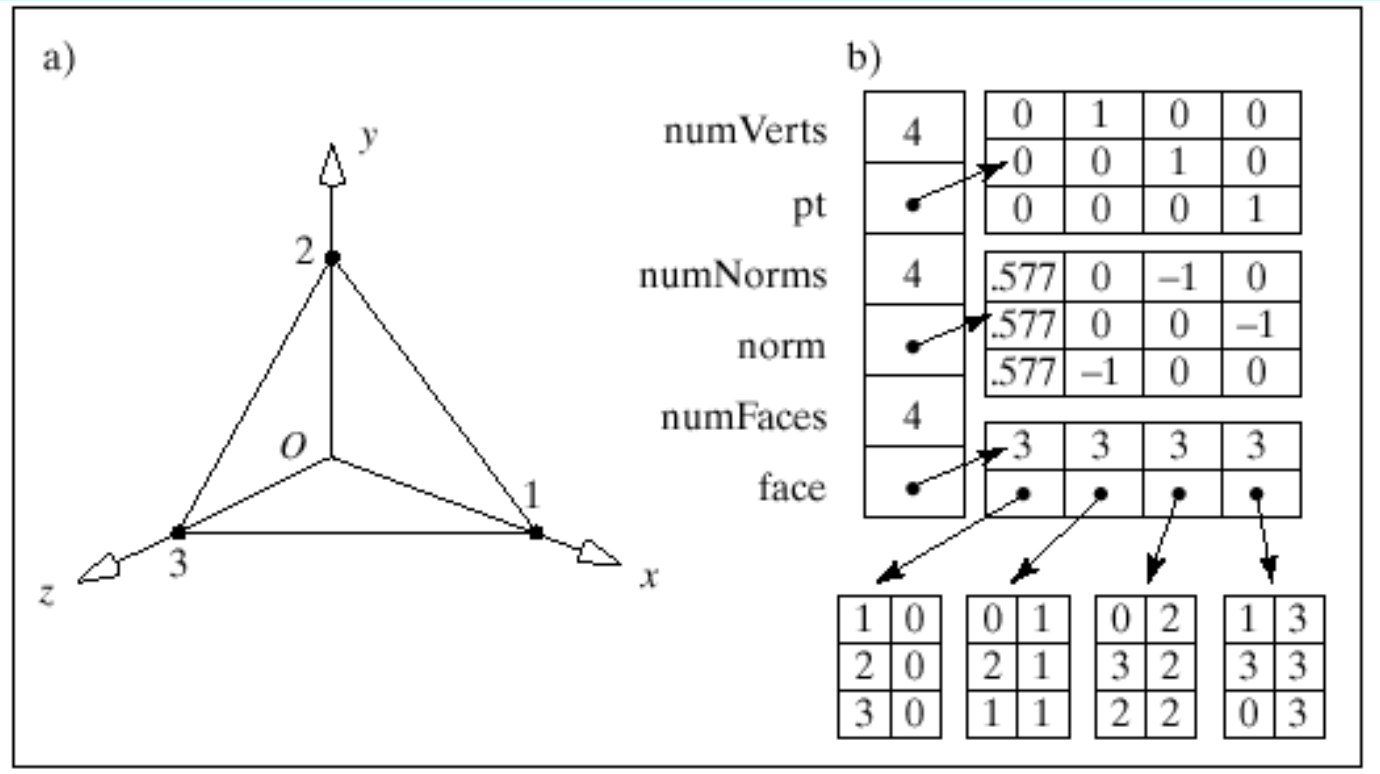
has all the properties



not connected

Mesh Models for Nonsolid Objects

Check book for a mesh data structure !



Drawing a mesh object

- Traverse the array of faces in the mesh object
- For each face, send the list of vertices and their normals down the graphics pipeline.

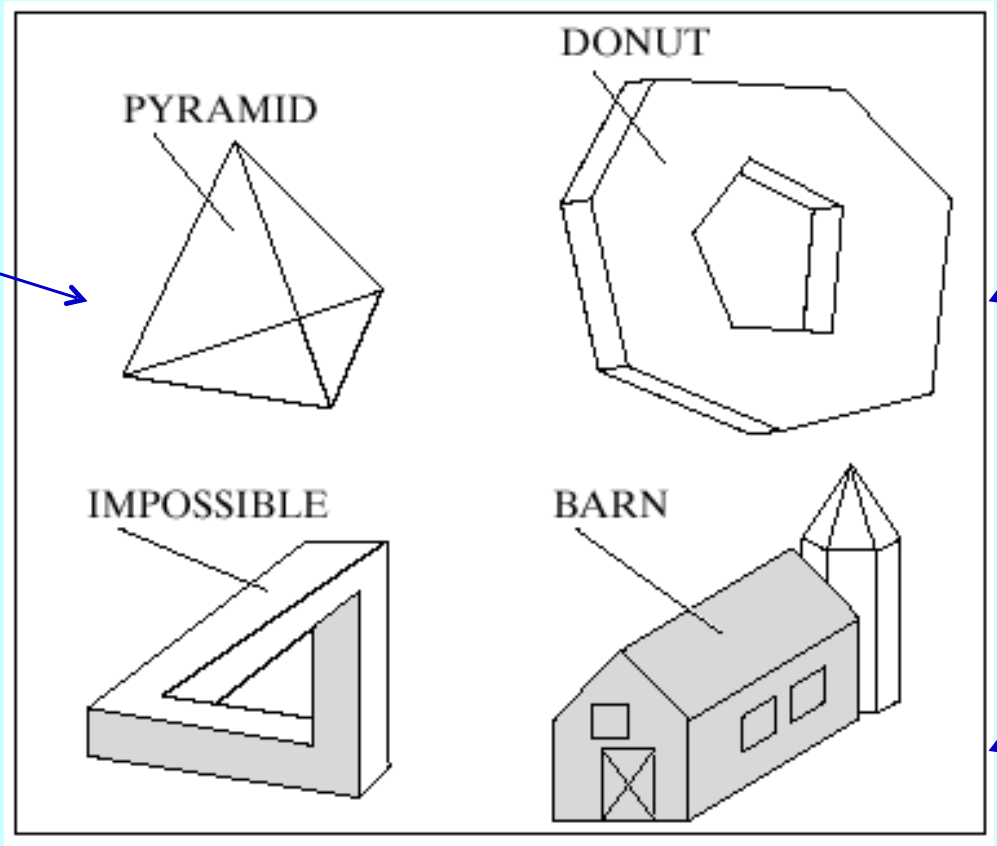
```
void Mesh:: draw()      // use OpenGL to draw this mesh
{
   for (int f = 0; f < numFaces; f++)      // draw each face
   {
      glBegin (GL_POLYGON);
         for (int v = 0; v < face[f].nVerts; v++)      // for each one..
         {
            int in = face[f].vert[v].normIndex;      // index of this normal
            int iv = face[f].vert[v].vertIndex;      // index of this vertex
            glNormal3f(norm[in].x, norm[in].y, norm[in].z);
            glVertex3f(pt[iv].x, pt[iv].y, pt[iv].z);
         }
      glEnd();
   }
}
```

Polyhedron: Connected mesh of simple planar polygons that encloses a finite amount of space

- every edge is shared by exactly two faces
- at least three edges meet at each vertex
- Faces do not interpenetrate. Two faces either do not touch at all, or they touch only along their common edge.

Algorithms for processing a mesh can be greatly **simplified** if they need only process meshes that represent a polyhedron.

polyhedron



Polyhedron if faces are planar

Two Polyhedra (without texture)

Euler's formula

For a simple polyhedron:

$$V+F-E = 2$$

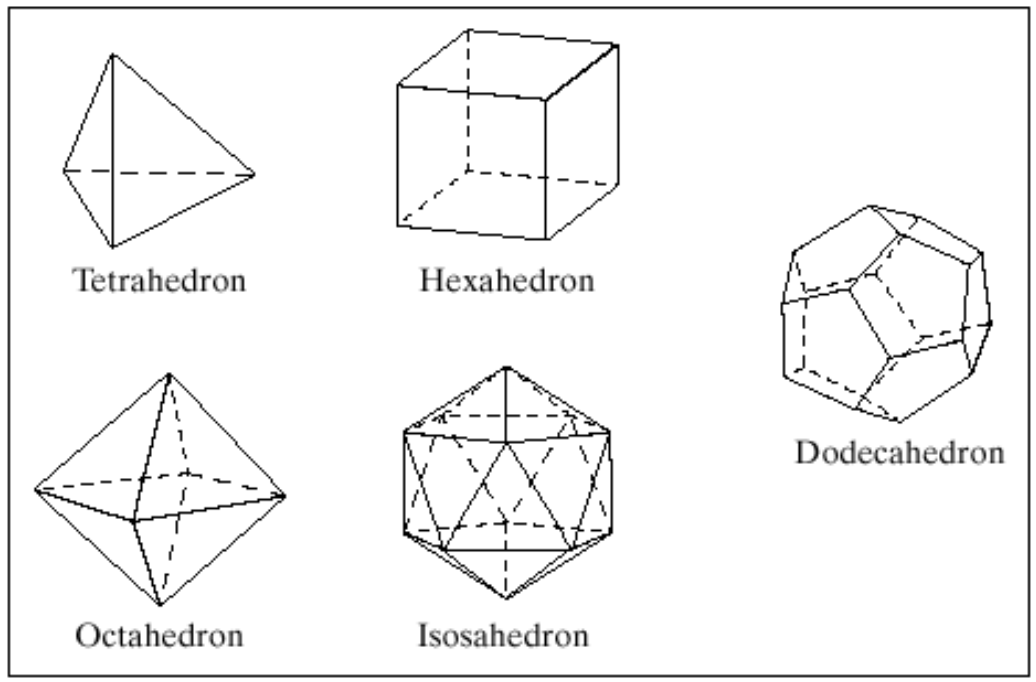
e.g. a cube has V = 8, F=6, E=12

F... Number of Faces
E... Number of Edges
V... Number of Vertices

The Platonic Solids

If all the faces of a polyhedron are identical and each is a regular polygon, the object is a **regular polyhedron**.

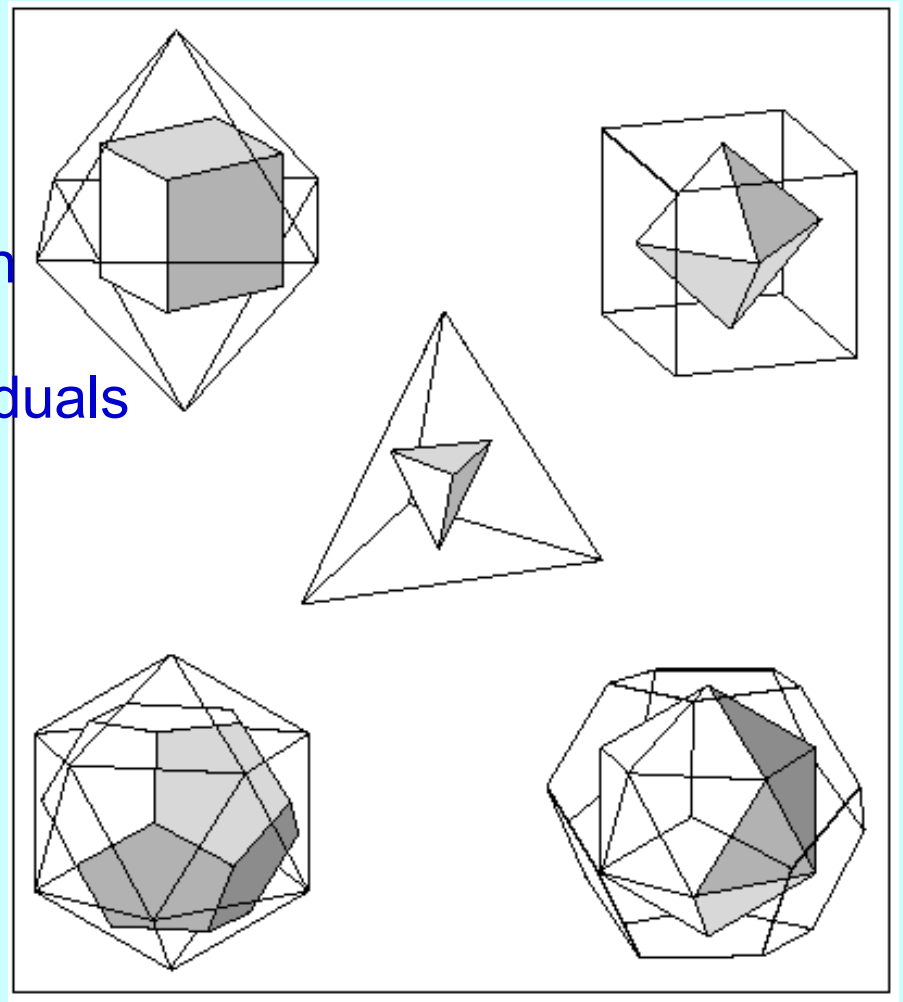
These symmetry constraints are so severe that only five such objects exist: the **Platonic solids**



Dual polyhedra

Each Platonic solid P has a dual polyhedron D. The vertices of D are centers of the faces of P, so edges of D connect the midpoints of adjacent faces of P.

- Dual of a tetrahedron is a tetrahedron
- The cube and octahedron are duals
- The icosahedron and dodecahedron are duals

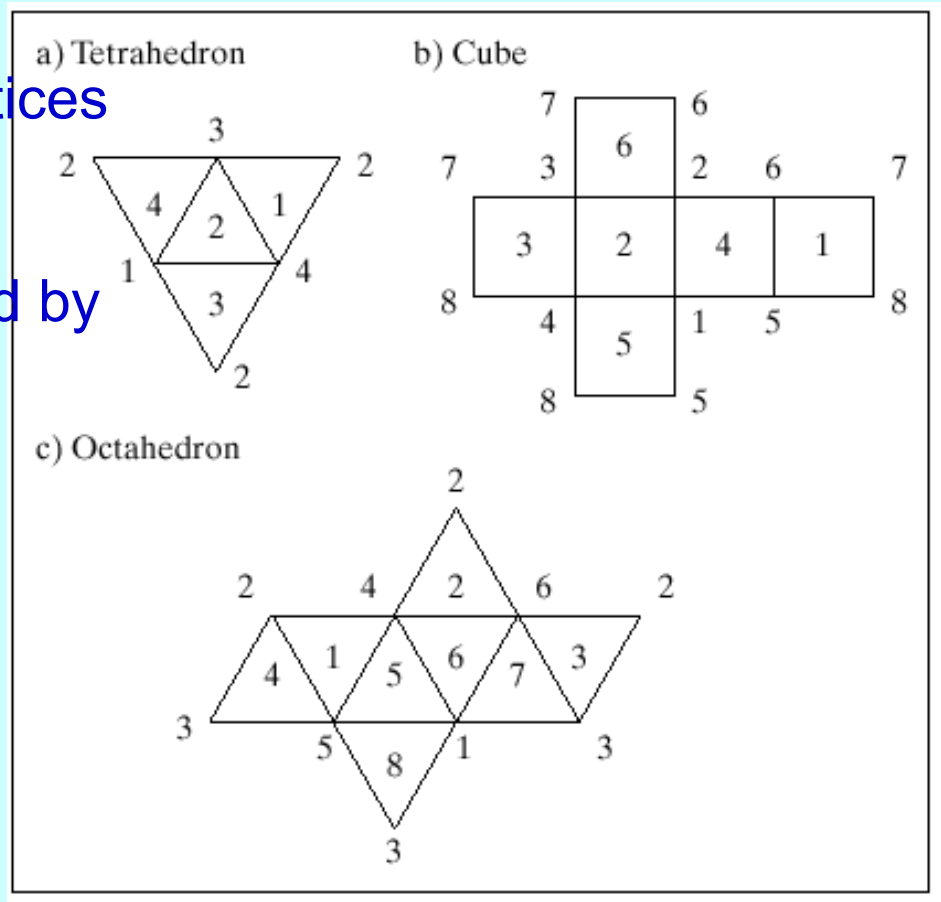


A model is obtained by slicing along certain edges of each solid and "unfolding" it to lie flat, so that all the faces are seen from the outside.

Consider the dual pair, cube and the octahedron:

Face 4 of the cube is surrounded by vertices 1,5,6,2

Vertex 4 of the octahedron is surrounded by faces 1,5,6,2.

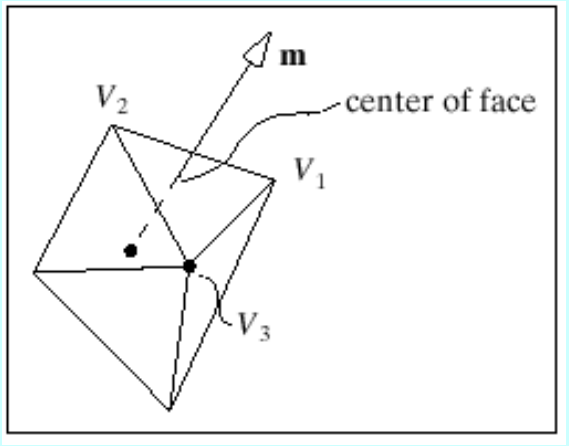


Normal Vectors for the Platonic Solids

Use Newell's method

High degree of symmetry of a Platonic solid offers a much simpler approach

$$\mathbf{m} = \frac{V_1 + V_2 + V_3}{3}$$



Extruded Shapes

A large class of shapes can be generated by **extruding** or **sweeping**.

Extruded Shapes

Let's start with a rectangle drawn on the surface of the screen

Let's then place this 2-D object, a flat polygon, in 3-D space

We extrude it a little, pulling it out in the third dimension

We pull it out even more

Extrusion allows us to break the process of modeling three-dimensional objects into **two 2-D steps**.

First a flat object is created on a plane.

Then the flat object is extruded out along a path.

In the simplest and most common case, the path is a straight line that is normal to the polygon being extruded. Thus the front and back faces of the extruded object have the same normals. In other words, the two faces are parallel to each other.

An extrusion path need not be a straight line. It can be a curve

We can close the extrusion curve to make a ring or torus

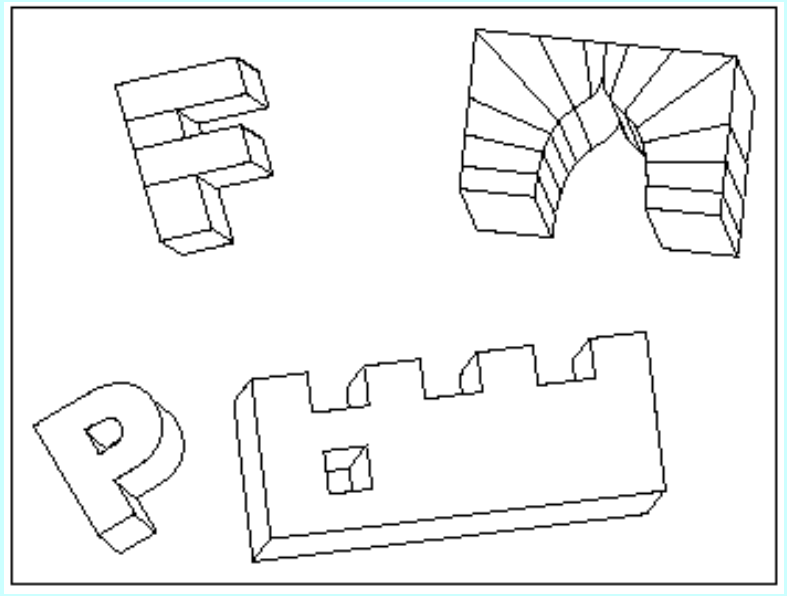
here is a perspective view

If the extrusion path is changed from a circle to an oval

Arrays of Extruded Prisms

Some rendering tools like OpenGL can reliably draw only convex polygons.

For this case, you can decompose (tessellate) the polygon into a set of convex polygons and extrude each one.



For this family of shapes, we need a method that builds a mesh out of an array of prisms,

```
void Mesh:: makePrismArray(…)
```

which would take as its arguments

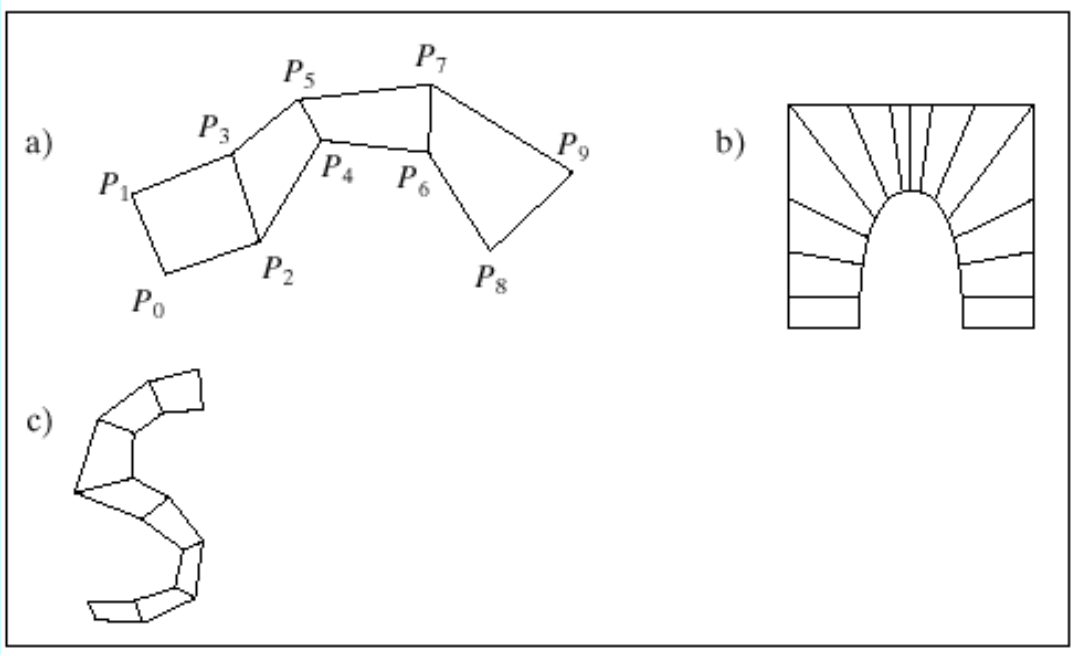
a suitable list of (convex) base polygons and

a vector d that describes the direction and amount of extrusion.

- Drawing such a mesh would involve some wasted effort, since walls that abut would be drawn (twice), even though they are ultimately invisible.

Special case: Extruded Quad Strips

A simpler but very interesting family of prisms are for which the base polygon can be represented by a *quad-strip*.



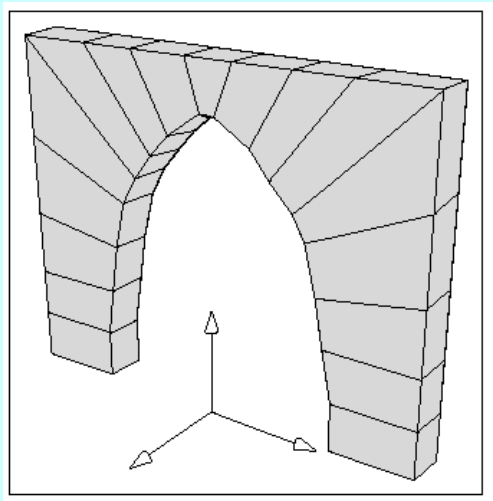
When a mesh is formed as an extruded quad-strip, only 2M vertices are placed in the vertex list, and only the “outside walls” are included in the face list.

There are 2M-2 faces in all. Thus, no redundant walls are drawn when the mesh is rendered.

A method for creating a mesh for an extruded quad-strip would take an array of 2D points and an extrusion vector as its parameters:

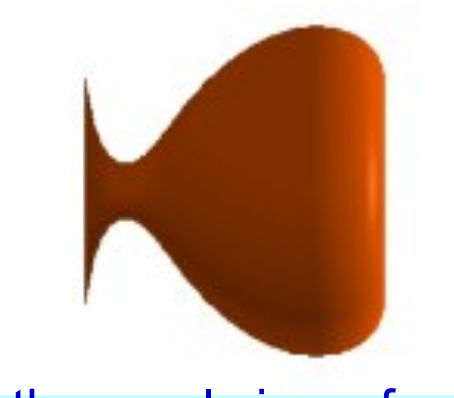
```
void Mesh:: makeExtrudedQuadStrip(Point2 p[], int numPts, Vector3 d);
```

Quad-strip = {P0, P1, P2, …, PM-1}

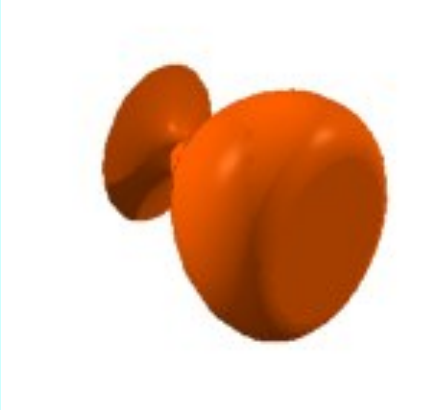


The concept of extruding a polygon in a closed path, merges into the concept of "lathing."

On a wood-turning lathe, as on a pottery wheel, the curved profile is cut into a solid object by turning the object. In a similar way, we can rotate a flat shape around an axis to create a solid object.

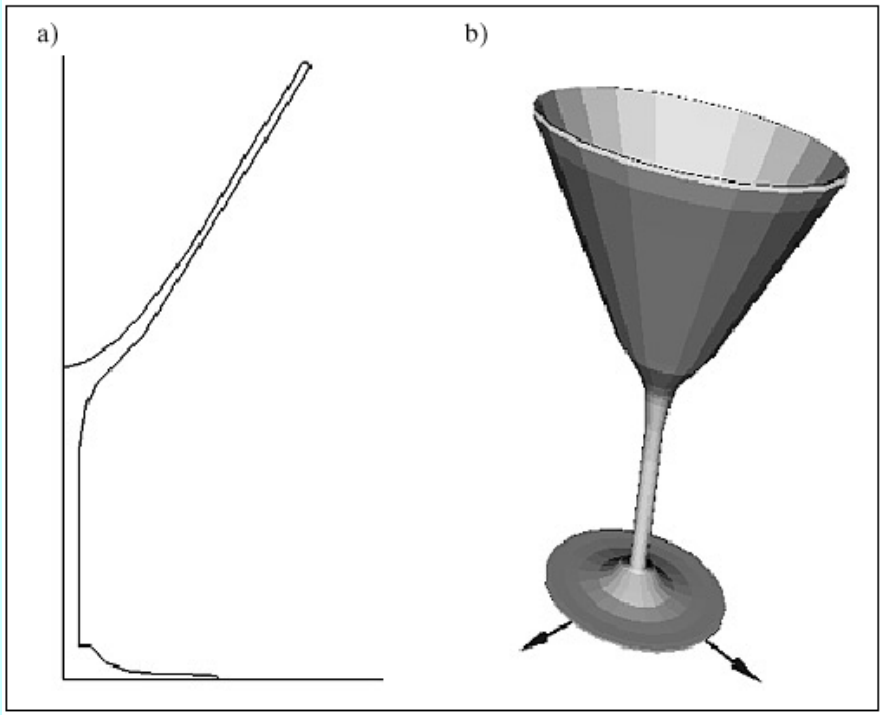


A side, orthogonal view of a lathed object reveals the original profile



A perspective view from a different angle gives us a basic goblet shape

Swept surfaces of revolution



The profile is a simple polyline based on points $P_j = (x_j, y_j, 0)$

$$M_i = \begin{pmatrix} \cos(\theta_i) & 0 & \sin(\theta_i) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta_i) & 0 & \cos(\theta_i) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where $\theta_i = 2\pi i / K, \quad i = 0,1,\ldots,K-1$

Mesh Approximations to Smooth Objects

The basic approach for each type of surface is to “polygonalize” or “tessellate” it into a collection of flat faces.

If the faces are small enough and there is a graceful change in direction from one face to the next, the resulting mesh will provide a good approximation to the underlying surface.

A mesh is created by building a vertex list and face list in the usual way, Except here **the vertices are computed from formulas**.

Representation of Surfaces

Examine a planar patch, given **parametrically**,

$$P(u,v) = C + \mathbf{a}u + \mathbf{b}v$$

C is a Point, a,b are vectors

$$P(u,v) = (X(u,v), Y(u,v), Z(u,v))$$

Implicitly,

$$F(x,y,z) = 0$$

For all the points that lie on the surface

$$F(x,y,z) = n_x x + n_y y + n_z z - D$$

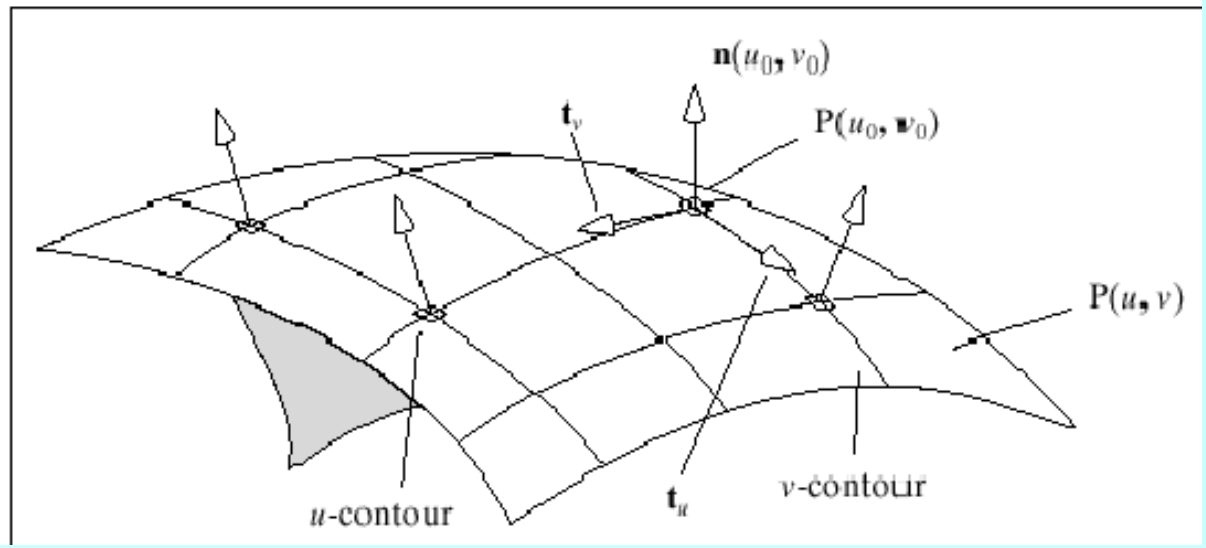
n is the normal vector,

$$F(P) = \mathbf{n}.(P - B)$$

The Normal Vector to a Surface

The normal direction to a surface can be defined at a point $P(u_0,v_0)$ on the surface by considering a very small region of the surface around $P(u_0,v_0)$.

If the region is small enough and the surface varies “smoothly” in the vicinity, the region will be essentially flat.



The Normal Vector to a Surface Given Parametrically

$$\mathbf{n}(u_0, v_0) = \left(\frac{\partial \mathbf{p}}{\partial u} x \frac{\partial \mathbf{p}}{\partial v} \right) \Big|_{u=u_0, v=v_0}$$

Example: Consider the plane given parametrically by P(u,v) = C + **a**u + **b**v. Find the normal to the plane.

Does the partial derivative work for the plane ?

The partial derivative of P with respect to u is a

The partial derivative of P with respect to v is b

$$N(u,v) = a \times b$$

Partial derivatives of **p**(u,v) exist whenever the surface is ‘smooth enough’.

$$\mathbf{p}(u,v) = X(u,v)\mathbf{i} + Y(u,v)\mathbf{j} + Z(u,v)\mathbf{k}$$

$$\frac{\partial \mathbf{p}(u,v)}{\partial u} = \left(\frac{\partial X(u,v)}{\partial u}, \frac{\partial Y(u,v)}{\partial u}, \frac{\partial Z(u,v)}{\partial u} \right)$$

The Normal Vector to a Surface Given Implicitly

The normal direction at the surface point (x,y,z) is found using the gradient, ∇F of F, which is given by

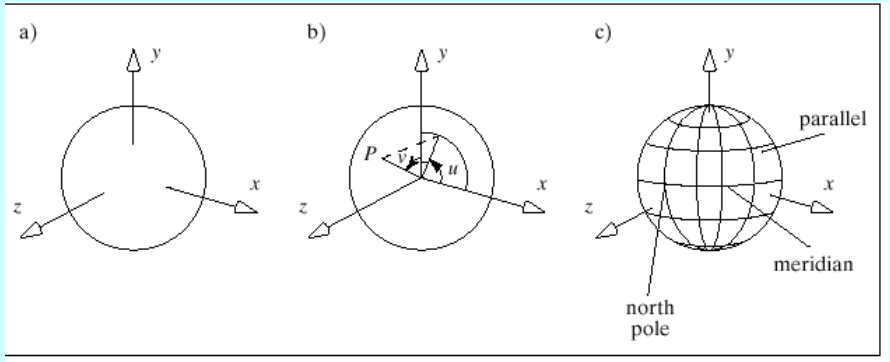
$$\mathbf{n}(x_0, y_0, z_0) = \nabla F\Big|_{x=x_0, y=y_0, z=z_0} = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right)\Bigg|_{x=x_0, y=y_0, z=z_0}$$

Three Generic Shapes: The sphere, cylinder, and cone

We develop the implicit form and parametric form for each one and see how one might make meshes to approximate them.

The Generic Sphere

$$F(x,y,z) = x^2 + y^2 + z^2 - 1$$



In the alternative notation,

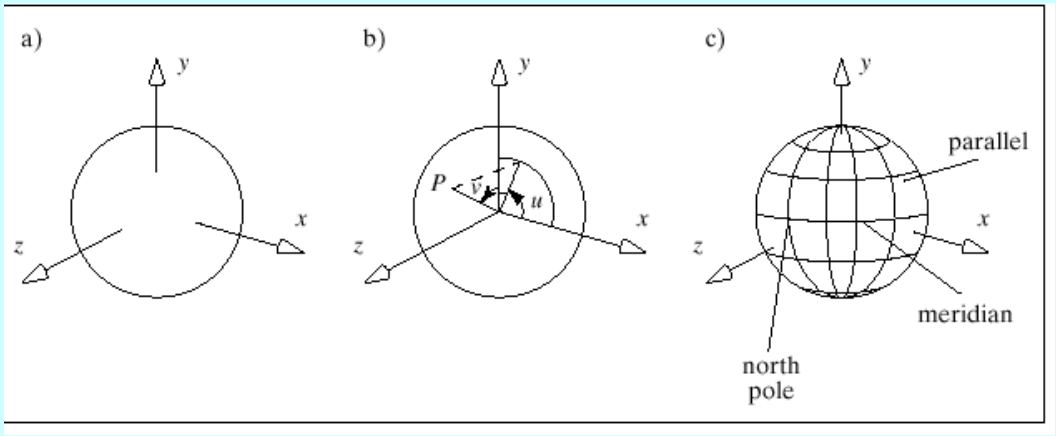
$$F(P) = |P|^2 - 1$$

A parametric form of the sphere:

$$P(u,v) = (\cos(v)\cos(u), \cos(v)\sin(u), \sin(v))$$

What is the normal direction **n**(u,v) to the sphere's surface at the point specified by the pair of parameters (u,v)?

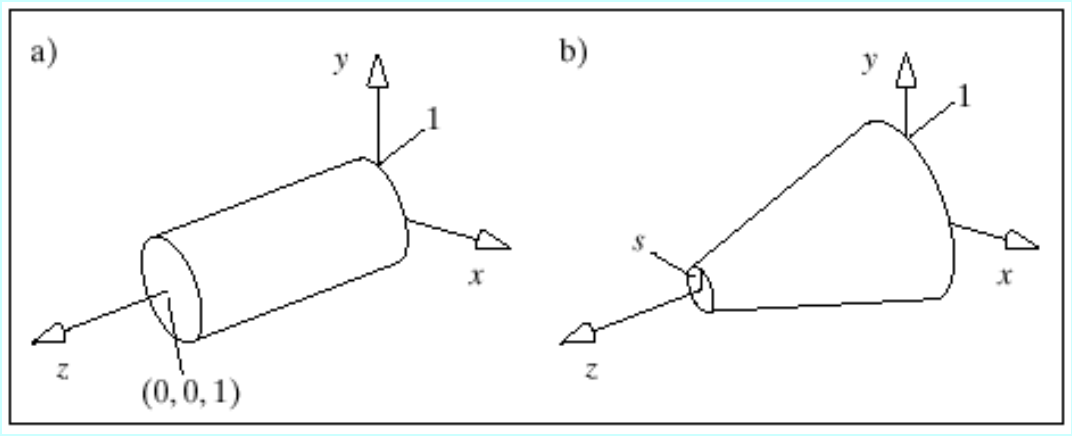
What is the gradient ?



The Generic Cylinder

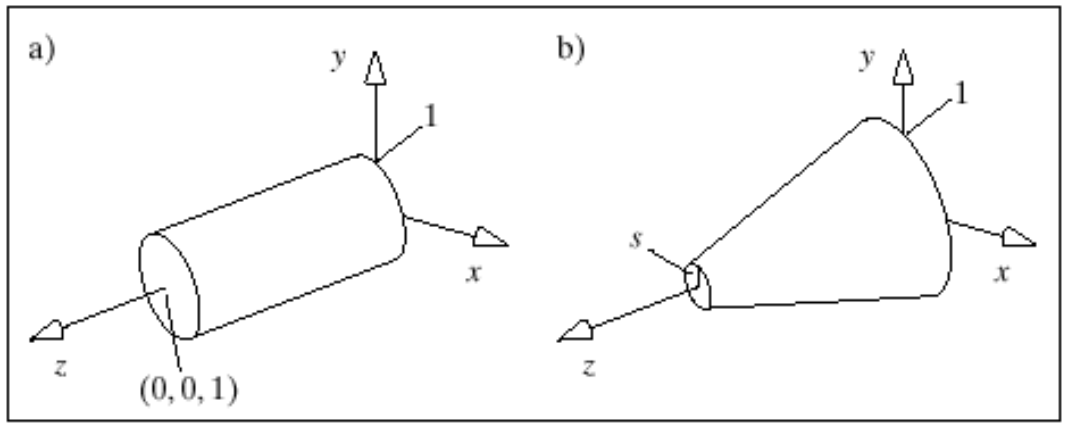
$$F(x, y, z) = x^2 + y^2 - (1 + (s-1)z)^2 \quad \text{for } 0 < z < 1$$

Generic cylinder is a member of the family of tapered cylinder having a ‘small radius’ of s when z = 1



The parametric form is:

$$P(u,v) = ((1+(s-1)v)\cos(u),(1+(s-1)v)\sin(u),v)$$

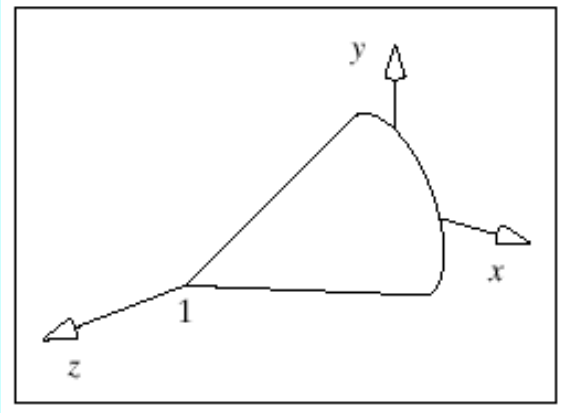


The Generic Cone

$$F(x, y, z) = x^2 + y^2 - (1 - z)^2 \quad \text{for } 0 < z < 1$$

and the parametric form is:

$$P(u, v) = ((1 - v)\cos(u), (1 - v)\sin(u), v)$$



Forming a Polygonal Mesh for a Curved Surface

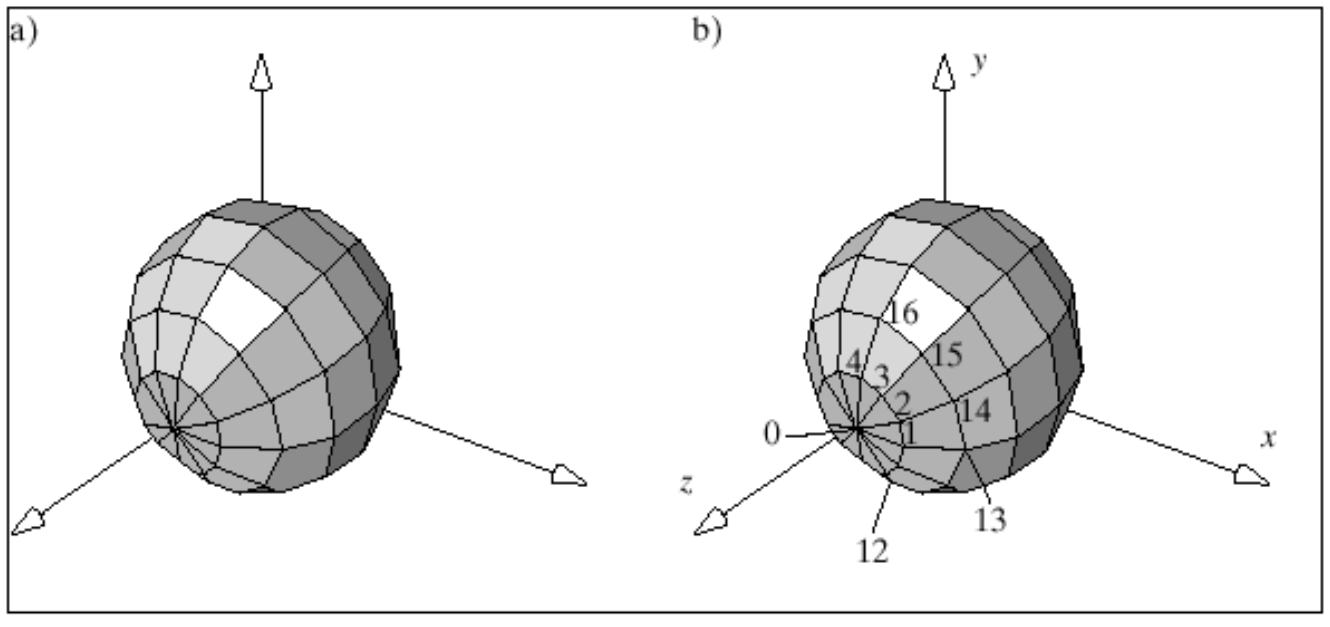
The process is called '**polygonalization**' or '**tessellation**' and it involves replacing the surface by a collection of triangles and quadrilaterals.

- Choose a number o values of u and v and "**sample**" the parametric form for the surface at these values to obtain a collection of vertices.

- Place the vertices in a vertex list

- Create a face list. Each face consists of three or four indices pointing to suitable vertices in the vertex list.

- Associated with each vertex in a face is the normal vector to the surface.



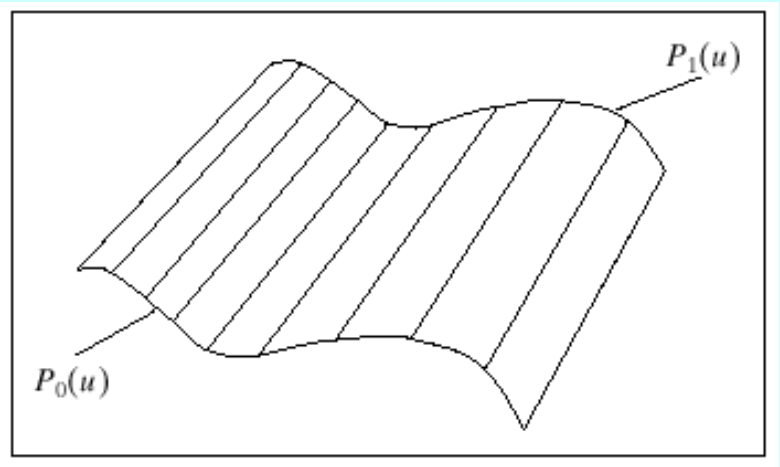
Slice up the sphere along azimuth lines and latitude lines.

$$u_i = \frac{2\pi i}{\text{nSlices}} \qquad i = 0,1,\ldots,nSlices\text{-}1$$

$$v_j = \frac{\pi}{2} - \frac{\pi j}{\text{nStacks}} \qquad i = 0,1,\ldots,nStacks\text{-}1$$

Ruled Surfaces

Ruled surfaces are swept out by moving a straight line along a particular trajectory.



$$P(u,v) = (1-v)\,P_0(u) + vP_1(u)$$

The ruled surface consists of one straight line joining each pair of corresponding points, $P_0(u')$ and $P_1(u')$

A ruled surface is easily “polygonalized”:

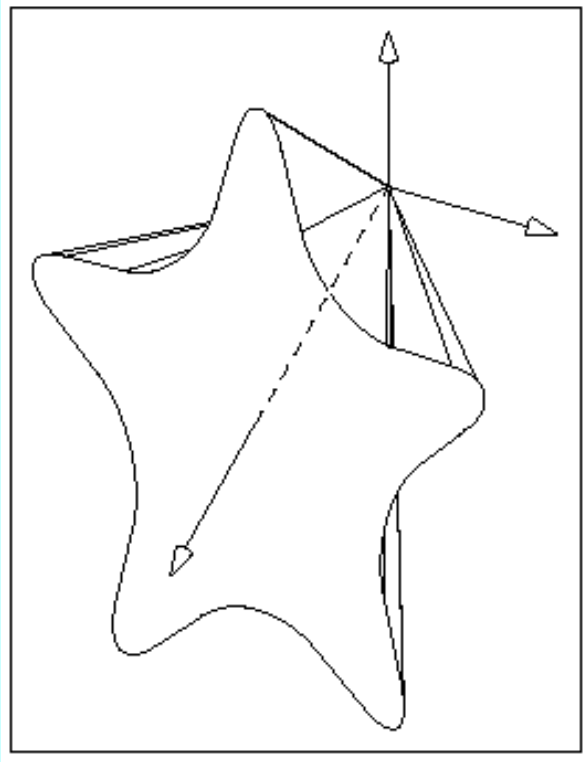
Choose a set of samples u_i and v_j and compute the position $P(u_i,v_j)$ and $\mathbf{n}(u_i,v_j)$ at each.

Three important ruled surfaces:

- The cone
- The cylinder
- The bilinear patch

Cones

A cone is a ruled surface for which one of the curves, say, $P_0(u)$, is a single point: the apex of the cone.



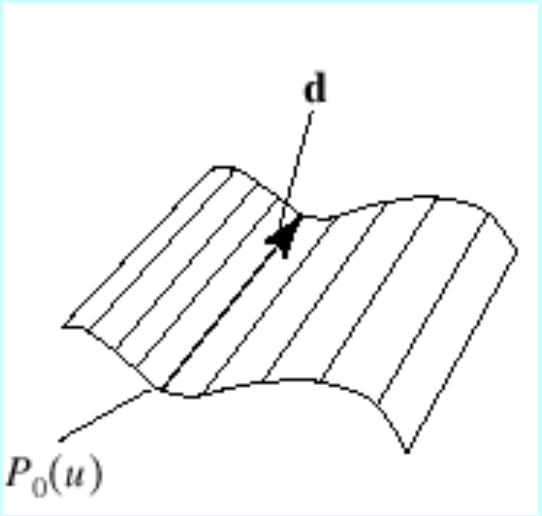
$P(u,v) = (1-v)\, P_0 + vP_1(u)$ where P_0 is the apex.

All lines pass through P_0 at v=0 and through $P_1(u)$ at v=1.

Cyclinders

A cyclinder is a ruled surface for which $P_1(u)$ is simply a translated version of $P_0(u)$

$$P_1(u) = P_0(u) + \mathbf{d}$$



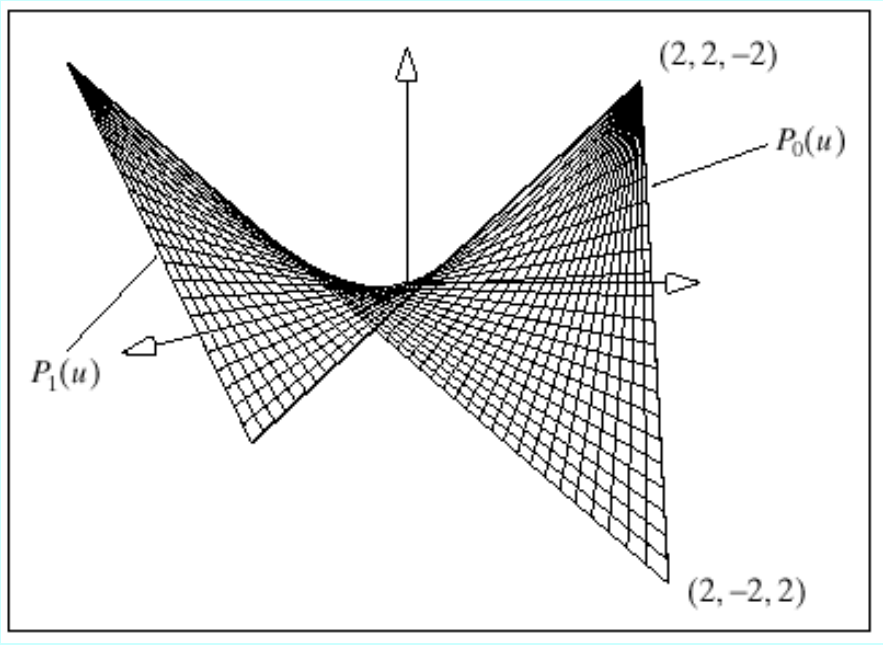
$$P(u,v) = P_0(u) + \mathbf{d}v$$

Bilinear Patches

A bilinear patch is formed when both $P_0(u)$ and $P_1(u)$ are straight-line segments defined over the same interval in u, say, 0 to 1.

$P_0(u) = (1-u)P_{00} + uP_{01}$

$P_1(u) = (1-u)\,P_{10} + u\,P_{11}$.

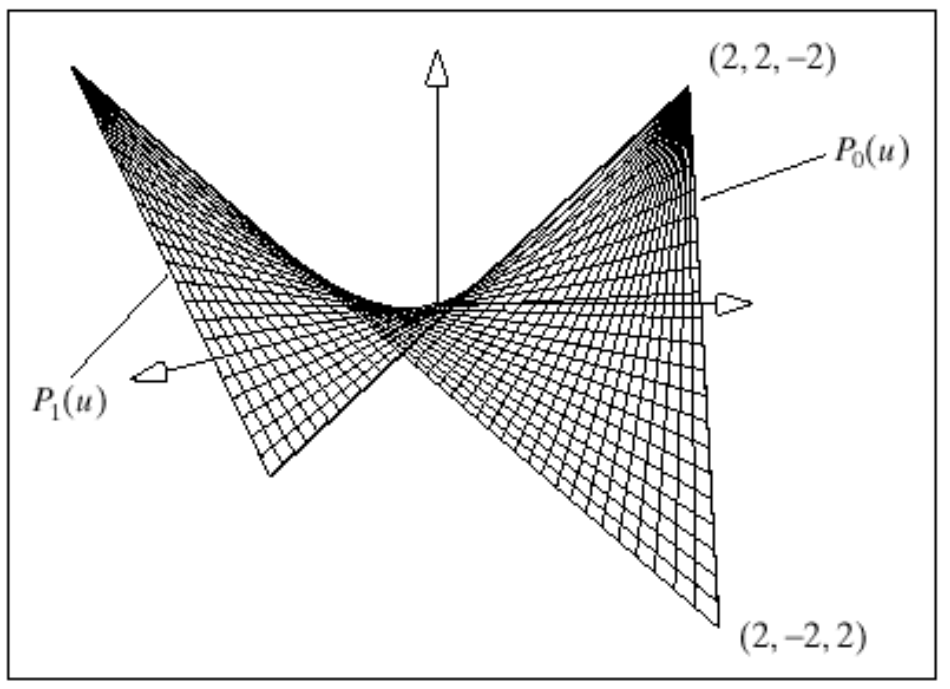


$$P(u,v) = (1-v)(1-u)P_{00} + (1-v)uP_{01} + v(1-u)P_{10} + uvP_{11}$$

This surface is bilinear, because its dependence is linear in u and linear in v.

Bilinear Patches

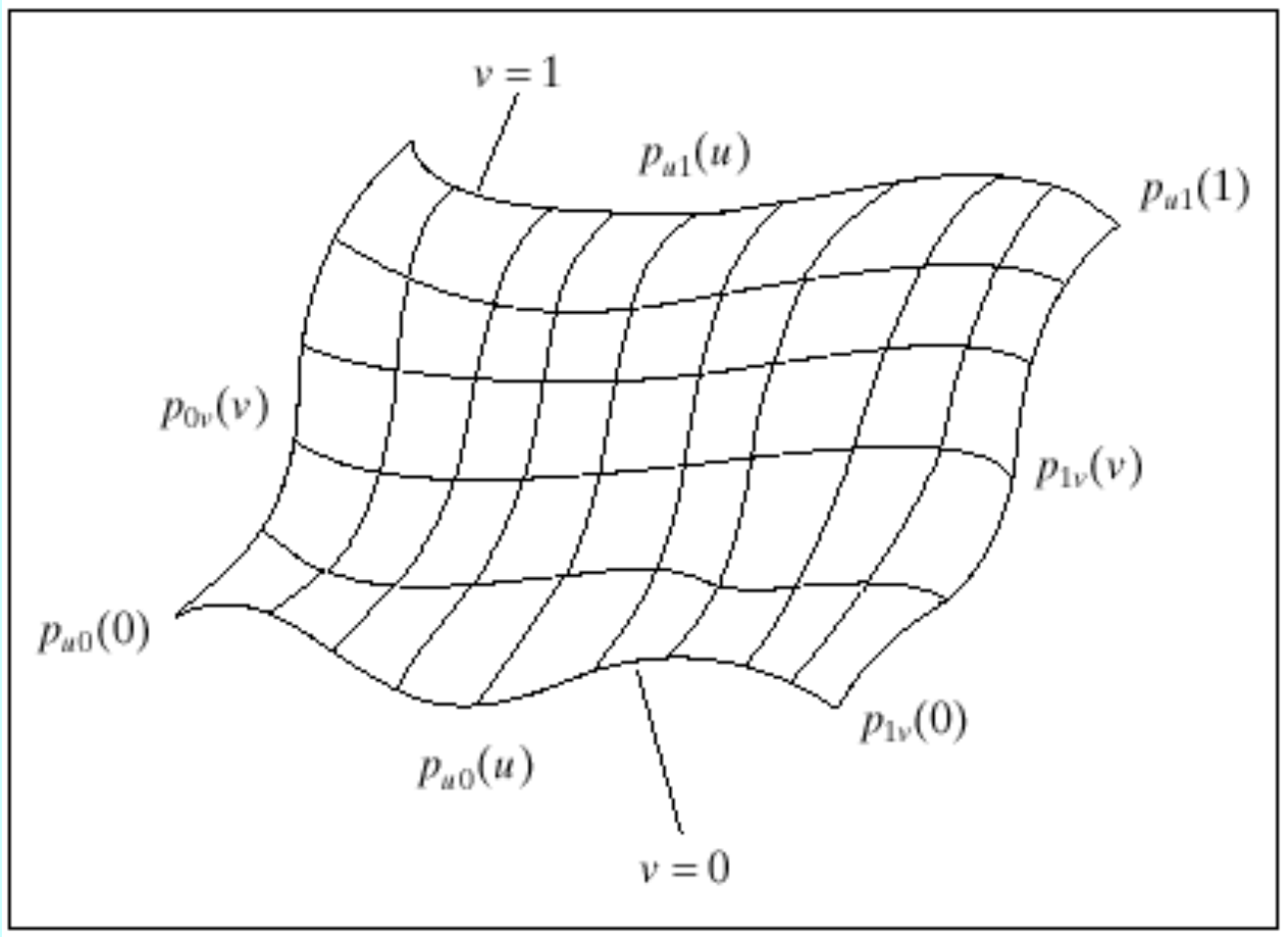
$P_0(u)$ is the line from (2,-2,2) to (2,2,-2)
$P_1(u)$ is the line from (-2,-2,-2) to (-2,2,2) .



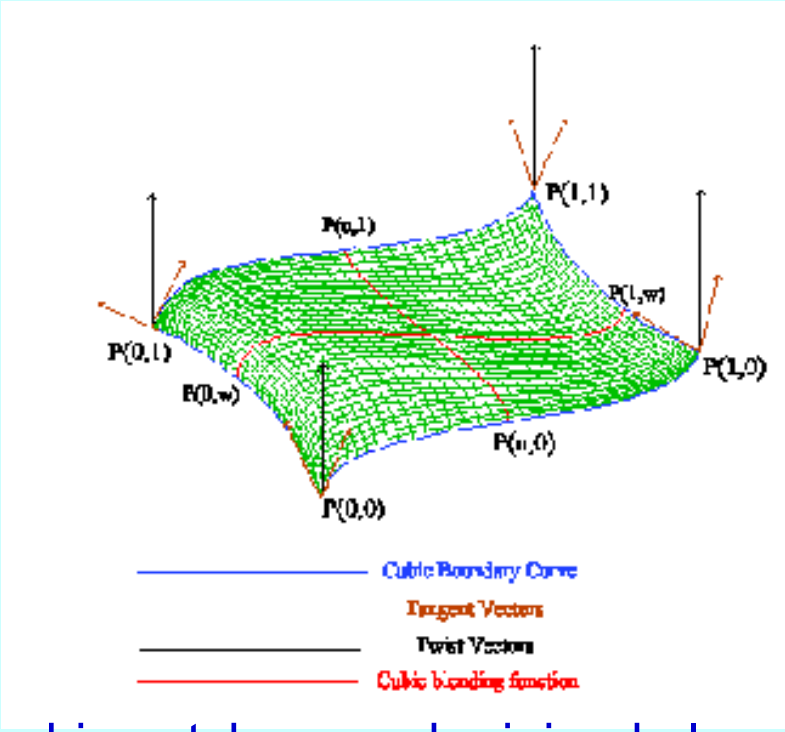
Bilinearly Blended Surfaces: Coons Patches

A generalization of a ruled surface that interpolates two boundary curves $P_0(u)$ and $P_1(u)$ is a bilinearly blended patch that interpolates to four boundary curves.

Coons patch

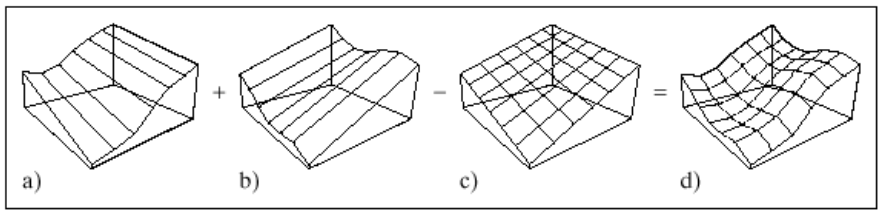


We need to know the four **position vectors** at the corners, eight **tangent vectors**, two at each corner and the four **twist vectors** at the corners.

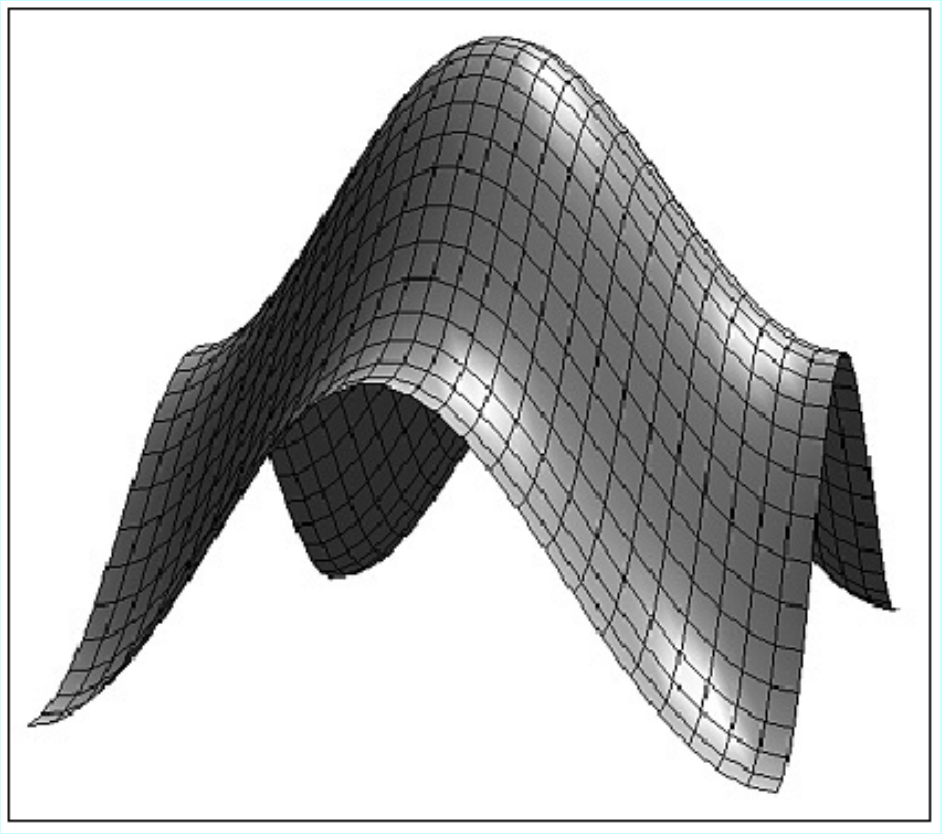


Multiple Coons bicubic patches can be joined along their boundaries provided certain **continuity conditions** are satisfied to yield more complex surfaces.

A natural first guess is to somehow combine a ruled patch built out of $p_{u0}(u)$ and $p_{u1}(u)$ with a ruled patch built out of $p_{0v}(v)$ and $p_{1v}(v)$



$$P(u,v) = \left[p_{0v}(v)(1-u) + p_{1v}(v)(u)\right] + \left[p_{u0}(u)(1-v) + p_{u1}(u)(v)\right]$$
$$-\left[(1-u)(1-v)\,p_{0v}(0) + u(1-v)\,p_{1v}(0)u + p_{0v}(1)v(1-u) + p_{1v}(1)uv\right]$$



Sample coons patch bounded by curves that have a sinusoidal oscillation.

Surfaces of revolution

A surface of revolution is formed by a **rotational sweep** of a profile curve C around an axis.

Suppose we place the profile in the xz plane and represent it parametrically by C(v) = (X(v), Z(v)). To generate the surface of revolution, we sweep the profile about the z axis.

The different positions of the curve C around the axis are called meridians.

A general point on the surface:

Surfaces of revolution

$$P(u,v) = \left[X(v)\cos(u), X(v)\sin(u), Z(v)\right]$$

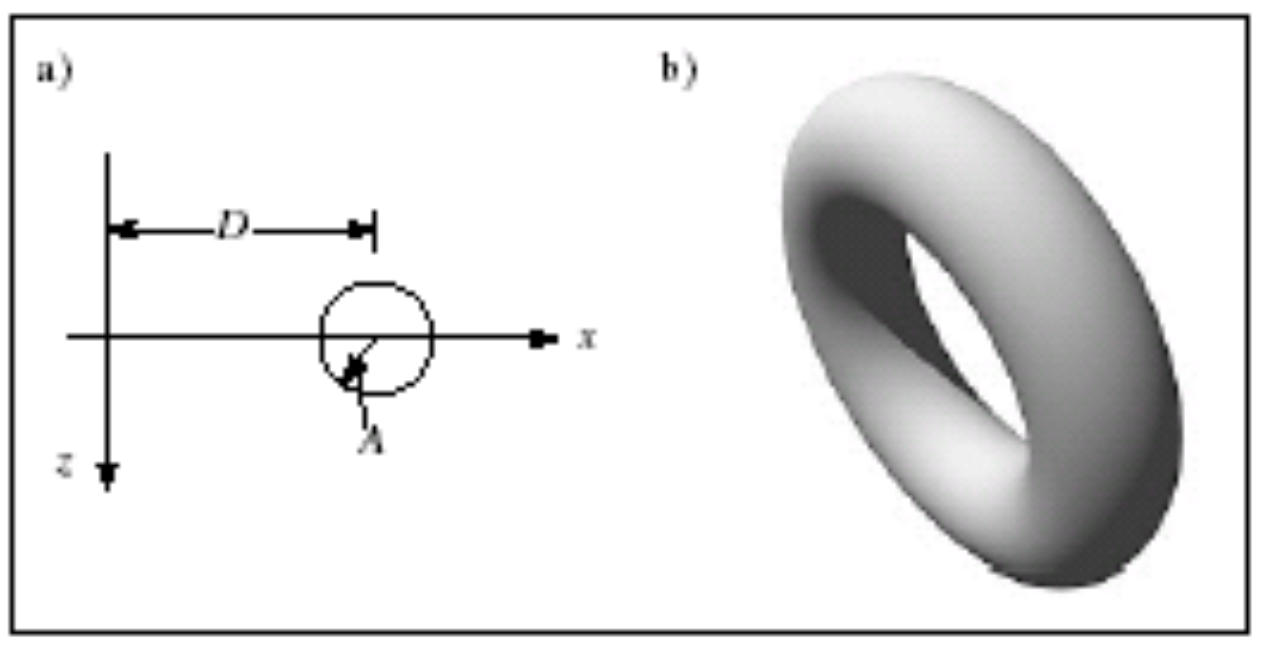
The normal vector to a surface of revolution is found by applying:

$$\mathbf{n}(u_0, v_0) = \left(\frac{\partial \mathbf{p}}{\partial u} x \frac{\partial \mathbf{p}}{\partial v} \right) \Big|_{u=u_0, v=v_0}$$

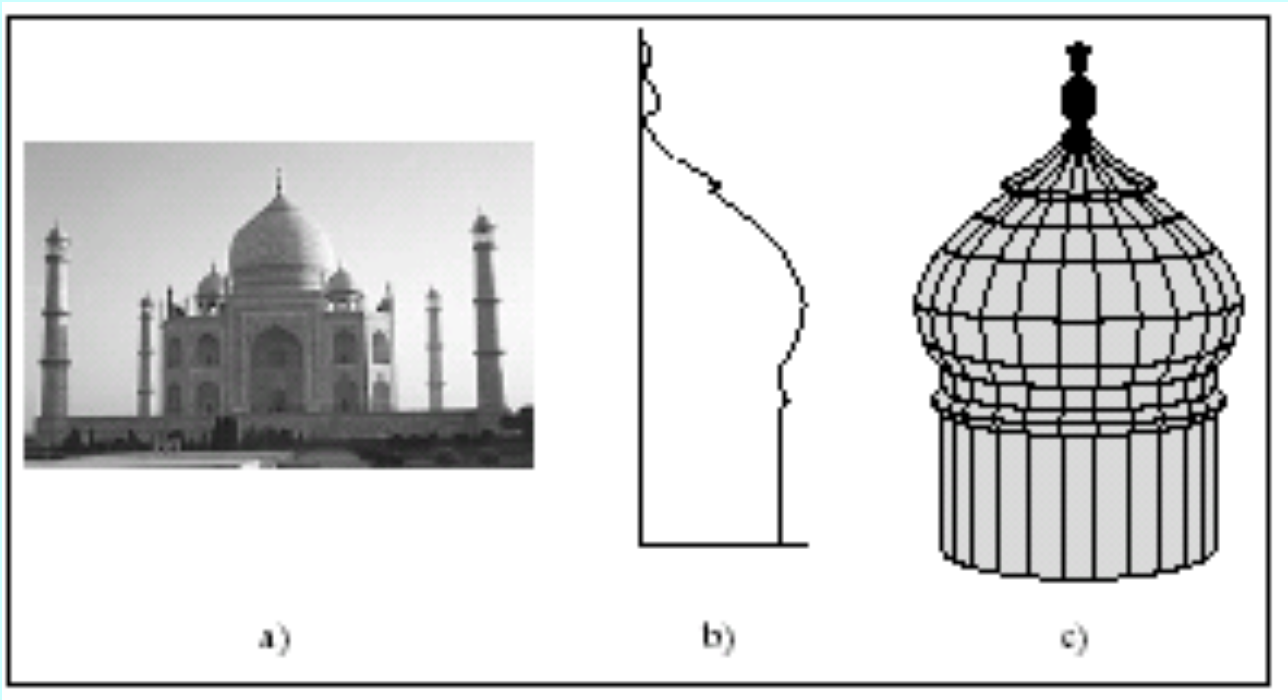
$$\mathbf{n}(u_0, v_0) = X(v)(\dot{Z}(v)\cos(u), \dot{Z}(v)\sin(u), -\dot{X}(v))$$

The circle has radius A and is displaced along the x axis by D, so its profile is:

$$C(v) = (D + A\cos(v), A\sin(v))$$

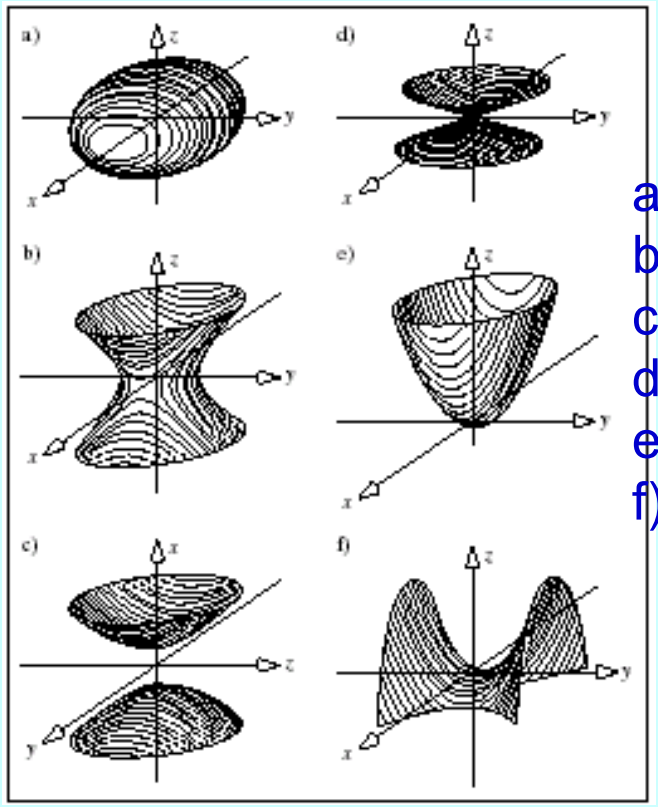


$$P(u,v) = \left[(D + A\cos(v))\cos(u), (D + A\cos(v))\sin(u), A\sin(v)\right]$$


a)
b)
c)

The Quadric Surfaces

Quadric surfaces are the 3D analogs of conic sections (the ellipse, parabola, hyperbola)



a) Ellipsoid,
b) Hyperboloid of one sheet
c) Hyperboloid of two sheets
d) Elliptic cone
e) Elliptic paraboloid
f) Hyperbolic paraboloid

Check Page 340 for Implicit & Parametric Forms

Name of quadric	*Implicit form*	*Parametric form*	*v-range, u-range*
Ellipsoid	$x^2+y^2+z^2-1$	$(\cos(v)\cos(u), \cos(v)\sin(u), \sin(v))$	$(-\pi/2, \pi/2), (-\pi, \pi)$
Hyperboloid of one sheet	$x^2+y^2-z^2-1$	$(\sec(v)\cos(u), \sec(v)\sin(u), \tan(v))$	$(-\pi/2, \pi/2), (-\pi, \pi)$
Hyperboloid of two sheets	$x^2-y^2-z^2-1$	$(\sec(v)\cos(u), \sec(v)\tan(u), \tan(v))$	$(-\pi/2, \pi/2)^*$
Elliptic cone	$x^2+y^2-z^2$	$(v\cos(u), v\sin(u), v)$	any real numbers, $(-\pi, \pi)$
Elliptic paraboloid	x^2+y^2-z	$(v\cos(u), v\sin(u), v^2)$	$v \geq 0, (-\pi, \pi)$
Hyperbolic paraboloid	$-x^2+y^2-z$	$(v\tan(u), v\sec(u), v^2)$	$v \geq 0, (-\pi, \pi)$

Some Notes on Quadric Surfaces

All traces of a quadric surface are conic sections.

- All traces of the ellipsoid are ellipses.
  - For hyperboloids of one sheet, the principle traces for the plane z=k are ellipses, the principle traces for the planes x=k and y=k are hyperbolas
  - For elliptic cones, the principle traces for the plane z=k are ellipses.

.
.
.

Normal Vectors to Quadric Surfaces

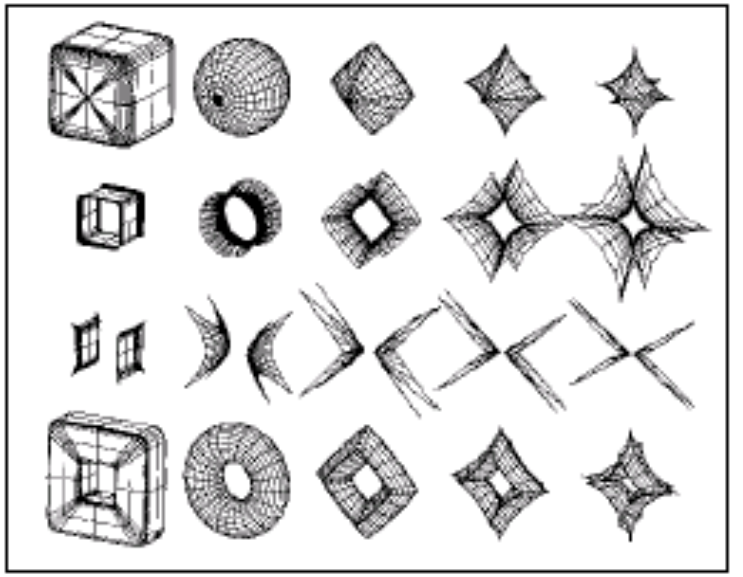
The gradient of F(x,y,z) for the ellipsoid is
F = (2x,2y,2z)

in parametric form, the normal is:

$$n(u,v) = \left[(\cos(v)\cos(u), \cos(v)\sin(u), \sin(v)\right]$$

.
.
.

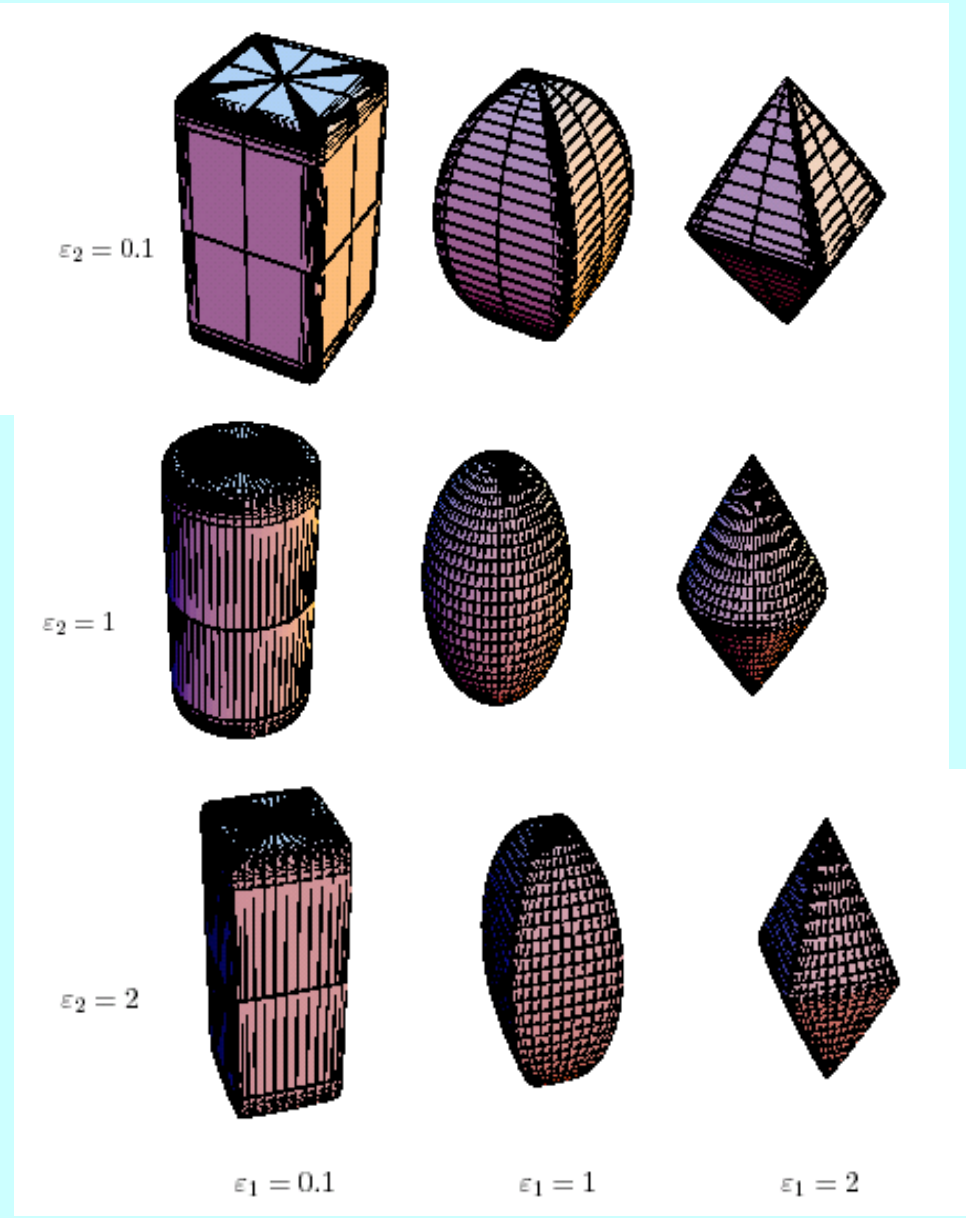
Superquadrics

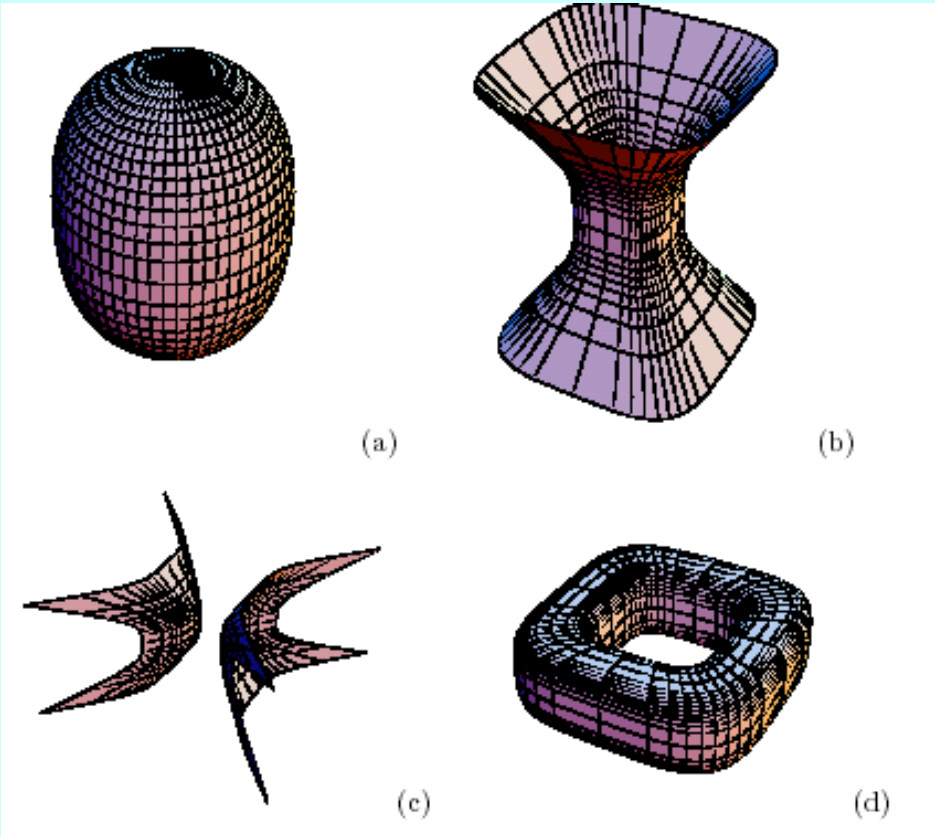


Name of quadric	*Implicit form*	*Parametric form*	*v-range, u-range*
Superellipsoid	$(x^{n} + y^{n})^{m/n} + z^{m} - 1$	$(\cos^{2/m}(v)\cos^{2/n}(u), \cos^{2/m}(v)\sin^{2/n}(u), \sin^{2/m}(v))$	$[-\pi/2, \pi/2], [-\pi, \pi]$
Superhyperboloid of one sheet	$(x^{n} + y^{n})^{m/n} - z^{m} - 1$	$(\sec^{2/m}(v)\cos^{2/n}(u), \sec^{2/m}(v)\sin^{2/n}(u), \tan^{2/m}(v))$	$(-\pi/2, \pi/2), [-\pi, \pi]$
Superhyperboloid of two sheets	$(x^{n} - y^{n})^{m/n} - z^{m} - 1$	$(\sec^{2/m}(v)\sec^{2/n}(u), \sec^{2/m}(v)\tan^{2/n}(u), \tan^{2/m}(v))$	$(-\pi/2, \pi/2)$
Supertoroid	$((x^{n} - y^{n})^{1/n} - d)^{m} + z^{m} - 1$	$((d + \cos^{2/m}(v))\cos^{2/n}(u), (d + \cos^{2/m}(v))\sin^{2/n}(u), \sin^{2/m}(v))$	$[-\pi, \pi), [-\pi, \pi)$

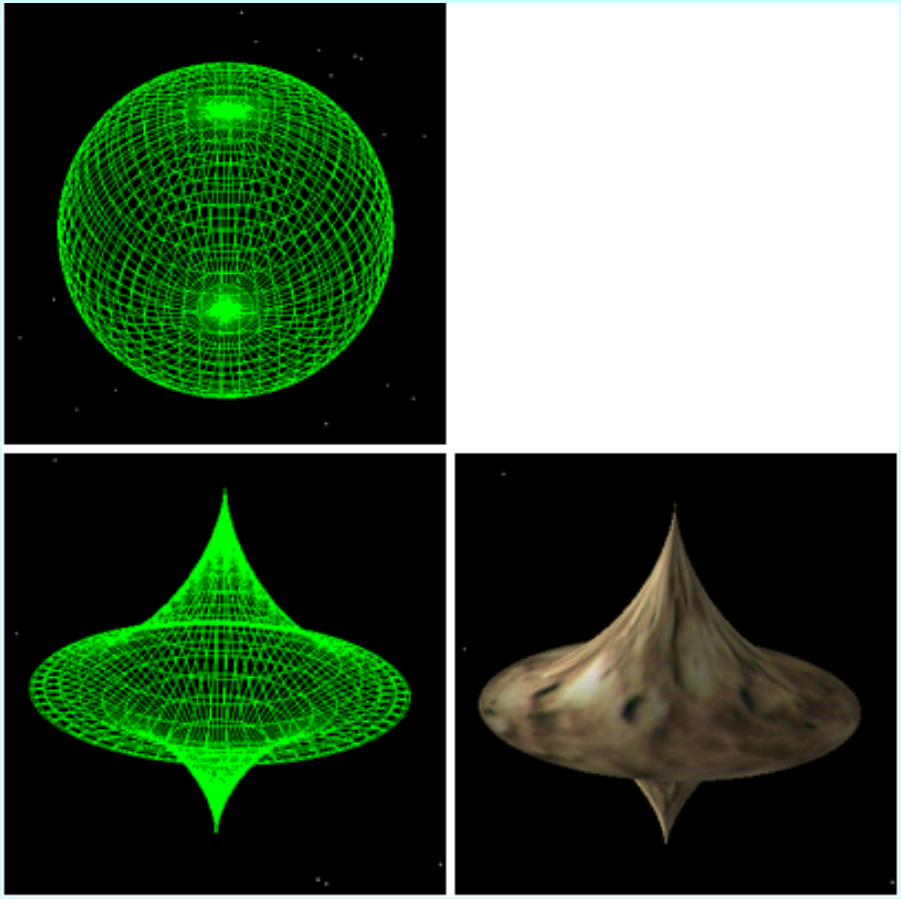
The implicit superquadric equation

$$\left(\left(\frac{x}{a_1}\right)^{\frac{2}{\varepsilon_2}} + \left(\frac{y}{a_2}\right)^{\frac{2}{\varepsilon_2}}\right)^{\frac{\varepsilon_2}{\varepsilon_1}} + \left(\frac{z}{a_3}\right)^{\frac{2}{\varepsilon_1}} = 1.$$



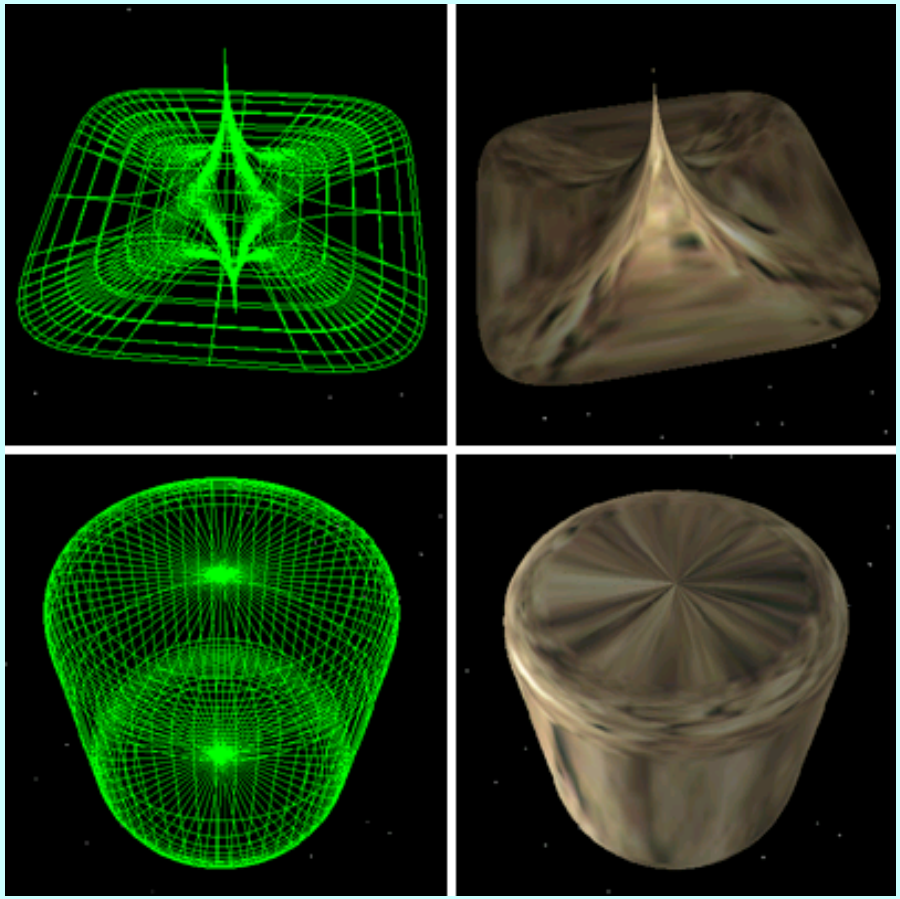


Superquadrics are a family of shapes that includes a) superellipsoids, b) hyperellipsoids of
one, and c) of two pieces and d) supertoroids



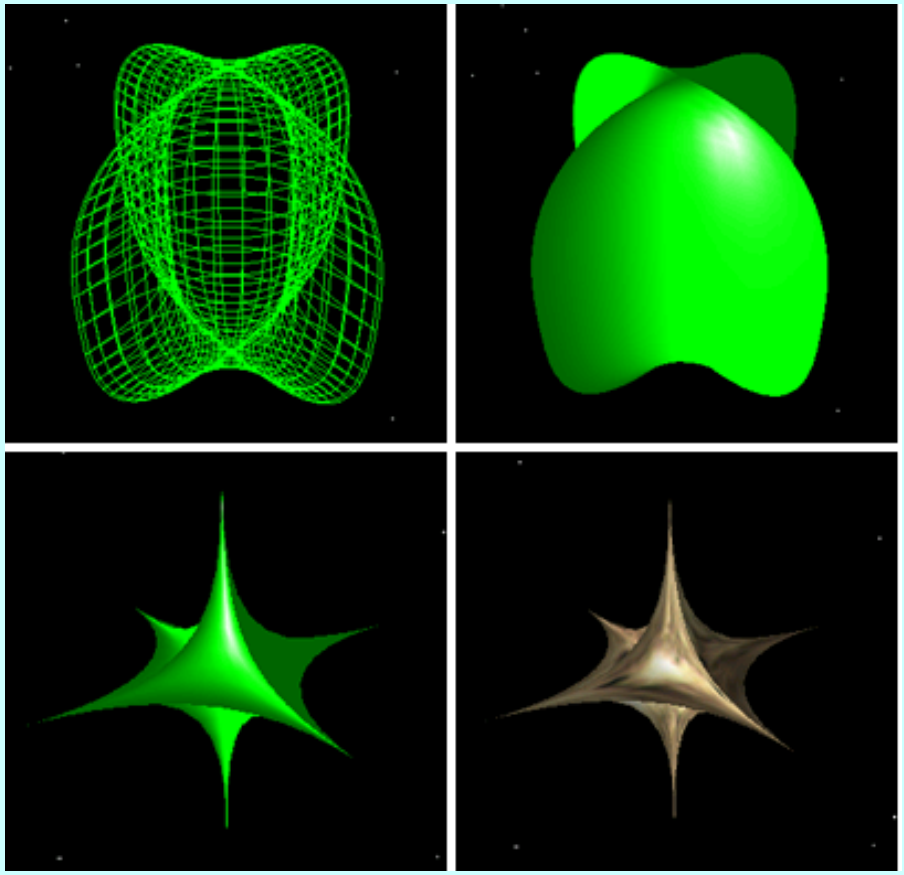
The two parameters =

First parameter = 1
Second parameter large



First parameter = small
Second parameter large

First parameter = small
Second parameter = 1



First parameter = large
Second parameter = 1

First parameter = large
Second parameter large

The code below creates a unit superellipsoid at the origin, it correctly assigns normals and
texture coordinates.

```
/* Create a superellipse "method" is 0 for quads, 1 for triangles (quads look nicer in
wireframe mode)/ This is a "unit" ellipsoid (-1 to 1) at the origin, glTranslate() and
glScale()
as required. */

void CreateSuperEllipse(double power1,double power2,int n,int method)
{
int i,j;
double theta1,theta2,theta3;
XYZ p,p1,p2,en; double delta;
/* Shall we just draw a point? */
if (n < 4) {
glBegin(GL_POINTS);
glVertex3f(0.0,0.0,0.0); glEnd(); return;
}
```

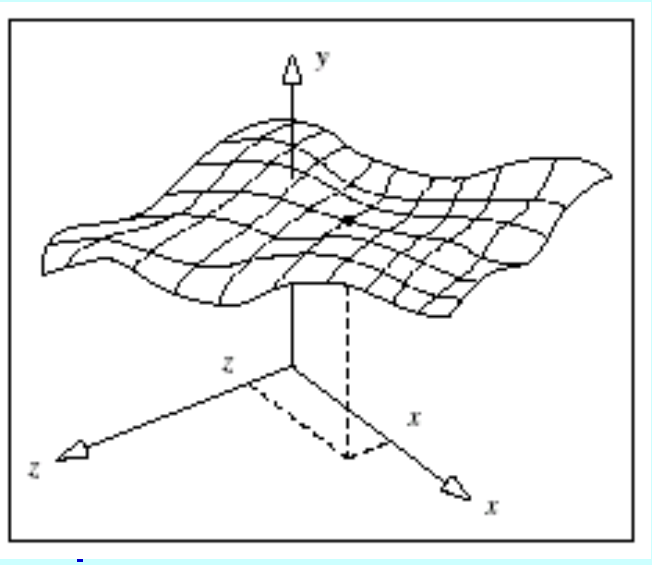
```
/* Shall we just draw a plus */
if (power1 > 10 && power2 > 10)
                {
        glBegin(GL_LINES);
      glVertex3f(-1.0, 0.0, 0.0);
      glVertex3f( 1.0, 0.0, 0.0);
      glVertex3f( 0.0,-1.0, 0.0);
      glVertex3f( 0.0, 1.0, 0.0);
      glVertex3f( 0.0, 0.0,-1.0);
      glVertex3f( 0.0, 0.0, 1.0);
            glEnd(); return;
                }

        delta = 0.01 * TWOPI / n;
            for (j=0;j<n/2;j++) {
  theta1 = j * TWOPI / (double)n - PID2;
theta2 = (j + 1) * TWOPI / (double)n - PID2;
```

```
if (method == 0)
    glBegin(GL_QUAD_STRIP);
else
    glBegin(GL_TRIANGLE_STRIP);
for (i=0;i<=n;i++) {
    if (i == 0 || i == n)
        theta3 = 0;
    else
        theta3 = i * TWOPI / n;
EvalSuperEllipse(theta2,theta3,power1,power2,&p);
EvalSuperEllipse(theta2+delta,theta3,power1,power2,&p1);
EvalSuperEllipse(theta2,theta3+delta,power1,power2,&p2);
en = CalcNormal(p1,p,p2);
glNormal3f(en.x,en.y,en.z);
glTexCoord2f(i/(double)n,2*(j+1)/(double)n);
glVertex3f(p.x,p.y,p.z);
```

```
      EvalSuperEllipse(theta1,theta3,power1,power2,&p);
      EvalSuperEllipse(theta1+delta,theta3,power1,power2,&p1);
      EvalSuperEllipse(theta1,theta3+delta,power1,power2,&p2);
      en = CalcNormal(p1,p,p2); glNormal3f(en.x,en.y,en.z);
      glTexCoord2f(i/(double)n,2*j/(double)n); glVertex3f(p.x,p.y,p.z);
    }
    glEnd();
  }
}
void EvalSuperEllipse(double t1,double t2,double p1,double p2,XYZ *p)
{
   double tmp;
   double ct1,ct2,st1,st2;
   ct1 = cos(t1); ct2 = cos(t2);
   st1 = sin(t1);
   st2 = sin(t2);
   tmp = SIGN(ct1) * pow(fabs(ct1),p1);
   p->x = tmp * SIGN(ct2) * pow(fabs(ct2),p2);
   p->y = SIGN(st1) * pow(fabs(st1),p1);
   p->z = tmp * SIGN(st2) * pow(fabs(st2),p2);
}
```

Surfaces Based on Explicit functions of two variables

Single valued functions: their position can be represented as an explicit function of two of the independent variables.



$$f(x,z) = e^{-ax^2 - bz^2}$$

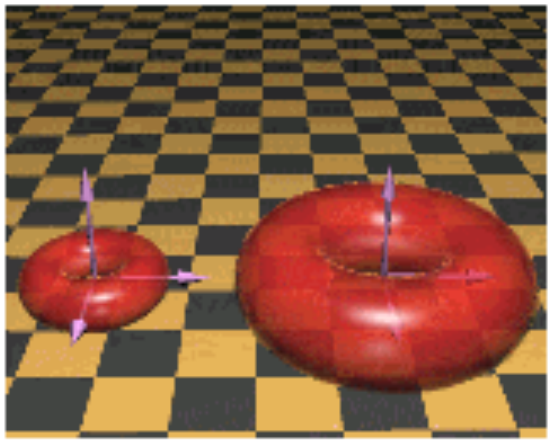
Deformations

There are two classes of 3D transformation.

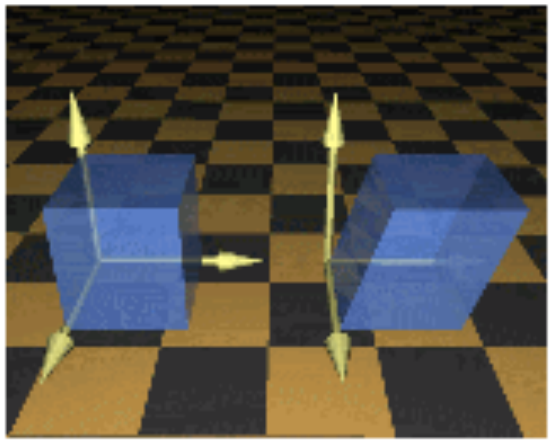
- *Affine transformations*
- *Structure deformating transformations*.

Affine transformations can be represented by matrices, and include *rotation*, *scaling*, *translation* and *shearing*.
They are all linear transformations

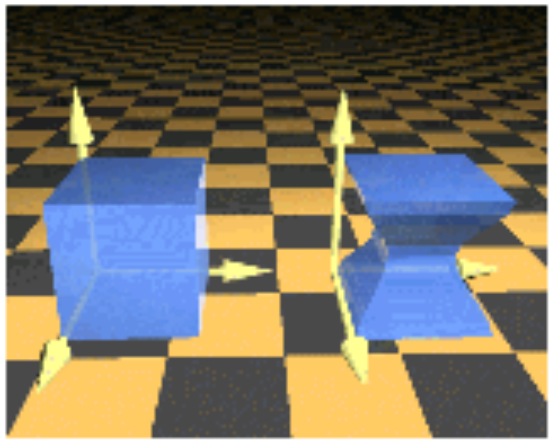
Structure deforming transformations are non-linear, and cannot be represented by a matrix.
They include *tapering*, *twisting*, *bending* and *pinching*.



Scaling



Shearing

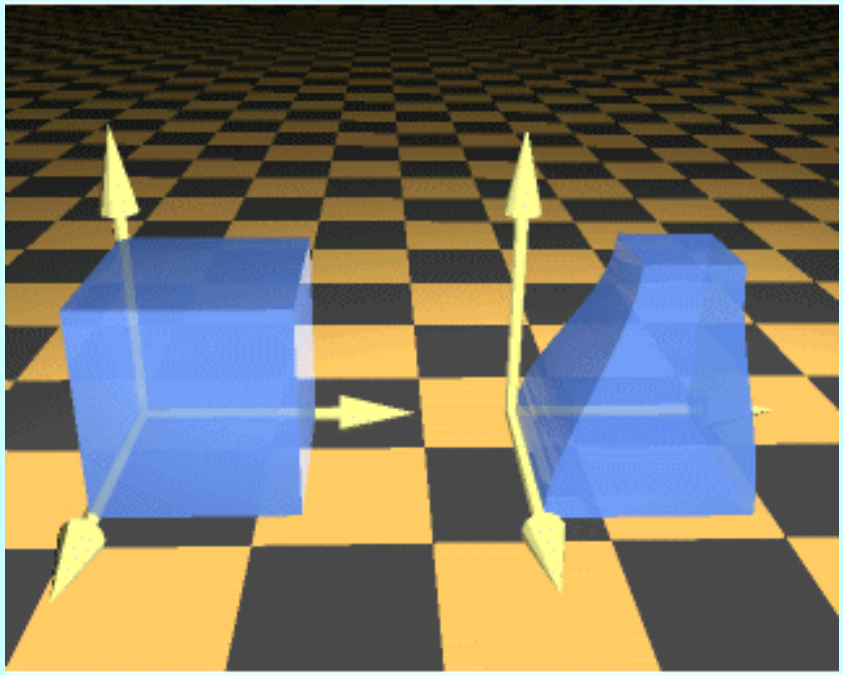


Pinching

Tapering

Tapering is nonuniform scaling

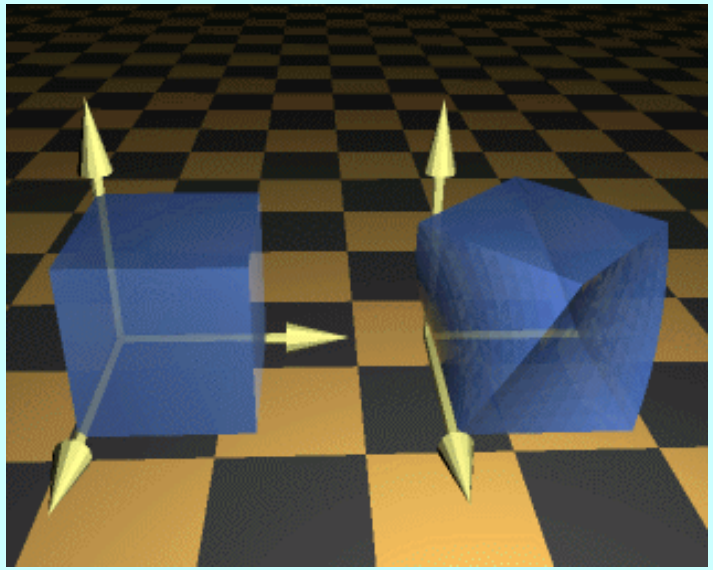
$$X = x$$
$$Y = ry$$
$$Z = rz \qquad \text{where } r = f(x)$$



Twisting

Can be thought of as a differential rotation

$$X = x\cos(\theta) - y\sin(\theta)$$
$$Y = x\cos(\theta) + y\sin(\theta)$$
$$Z = z \qquad \text{where } \theta = f(z)$$



Bending

Bend region: $y_{min} \le y \le y_{max}$

Bending angle: $\theta = k(y' - y_0)$

Radius of curvature: 1/k

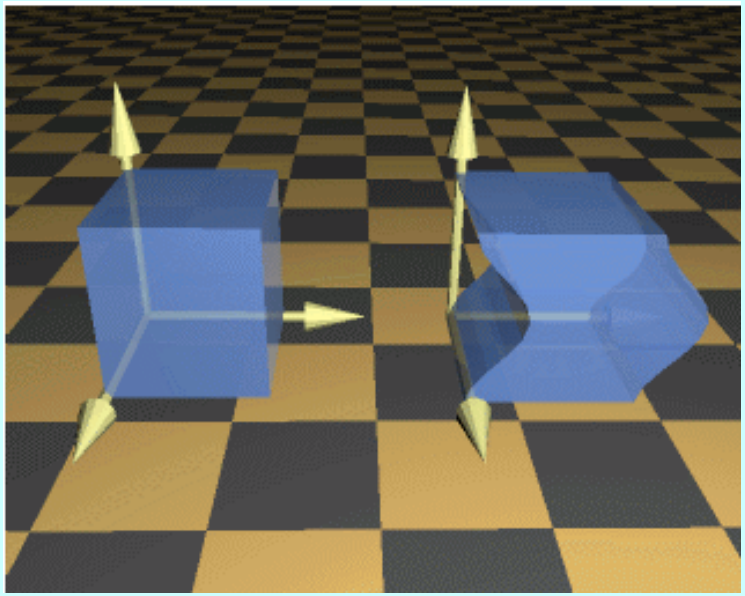
Center of bend: y_0

$$y' = \begin{cases} y_{min} & y \le y_{min} \\ y & y_{min} \le y \le y_{max} \\ y_{max} & y \ge y_{max} \end{cases}$$

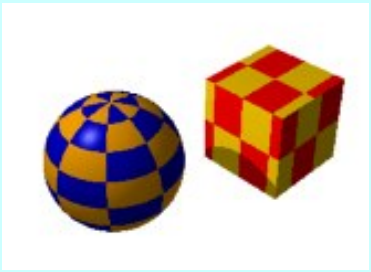
$$X = x$$

$$Y = \begin{cases} -\sin\theta(z - 1/k) + y_0 & y_{min} \le y \le y_{max} \\ -\sin\theta(z - 1/k) + y_0 + \cos\theta(y - y_{min}) & y < y_{min} \\ -\sin\theta(z - 1/k) + y_0 + \cos\theta(y - y_{max}) & y \ge y_{max} \end{cases}$$

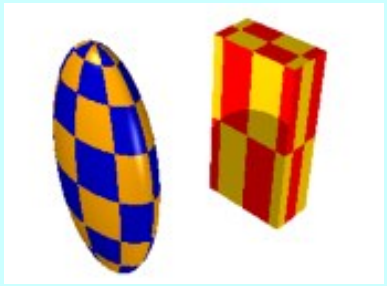
$$Z = \begin{cases} \cos\theta(z - 1/k) + 1/k & y_{min} \le y \le y_{max} \\ \cos\theta(z - 1/k) + 1/k + \sin\theta(y - y_{min}) & y < y_{min} \\ \cos\theta(z - 1/k) + 1/k + \cos\theta(y - y_{max}) & y \ge y_{max} \end{cases}$$



Start with two primitives, a cube and a sphere



scale the objects disproportionately

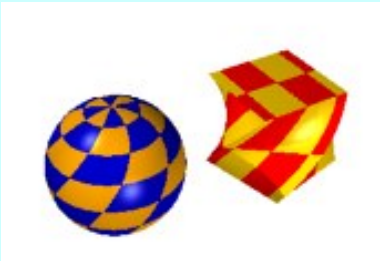


stretch of the objects.

squash the objects down

bending

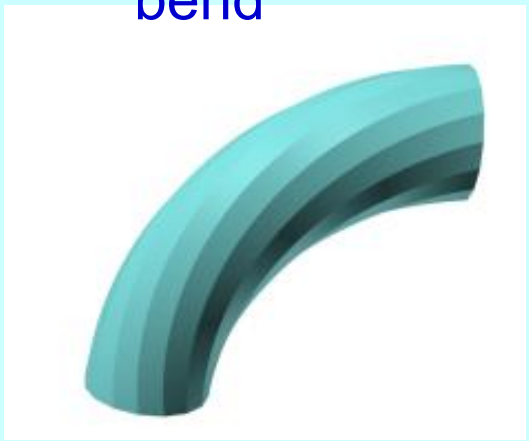
bend with a twist



create a 10-sided regular polygon and extrude

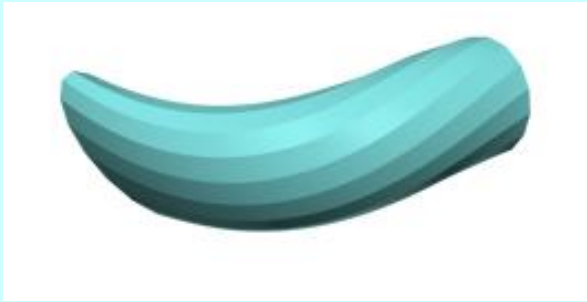
twist

bend



bend

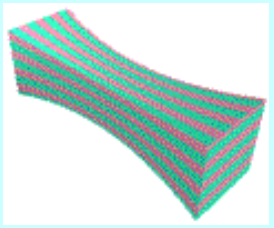
twist



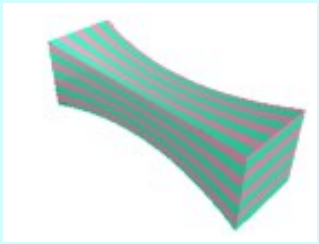
Animation

create an elongated cube

create a twist, and then move the twist through the length of the taffy over the sequence



stretch it out



squash it down

Animation

At the start of the loop we set a keyframe for the stretched version.
In the middle of the loop we set a key for the squashed version.
Finally, at the end of the sequence, we set another keyframe (or just "key") for the stretched version.

Thus, as the animated sequence is played in a repeating loop, the taffy squashes and
stretches in a cycle.