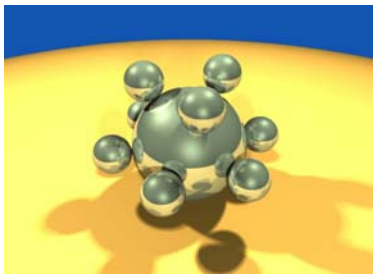
Computer Graphics using OpenGL, 3rd Edition

F. S. Hill, Jr. and S. Kelley



Chapter 6.1-3
Modeling Shapes with Polygonal Meshes

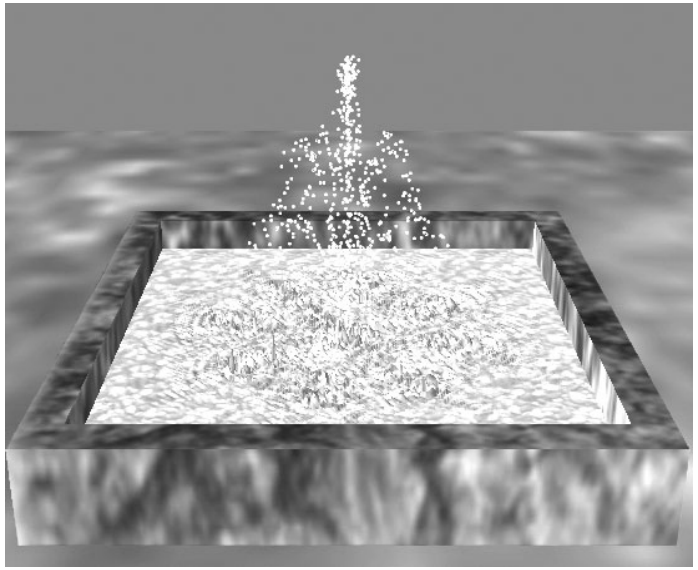
S. M. Lea
University of North Carolina at Greensboro

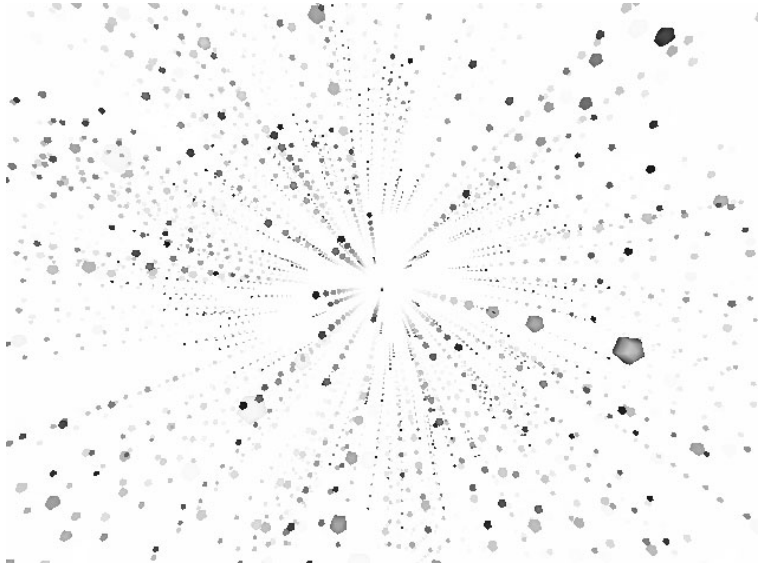
3D Modeling

- Polygonal meshes capture the shape of complex 3D objects in simple data structures.
  - Platonic solids, the Buckyball, geodesic domes, prisms.
  - Extruded or swept shapes, and surfaces of revolution.
  - Solids with smoothly curved surfaces.
- Animated Particle systems: each particle responds to conditions.
- Physically based systems: the various objects in a scene are modeled as connected by springs, gears, electrostatic forces, gravity, or other mechanisms.

Particle Systems Example

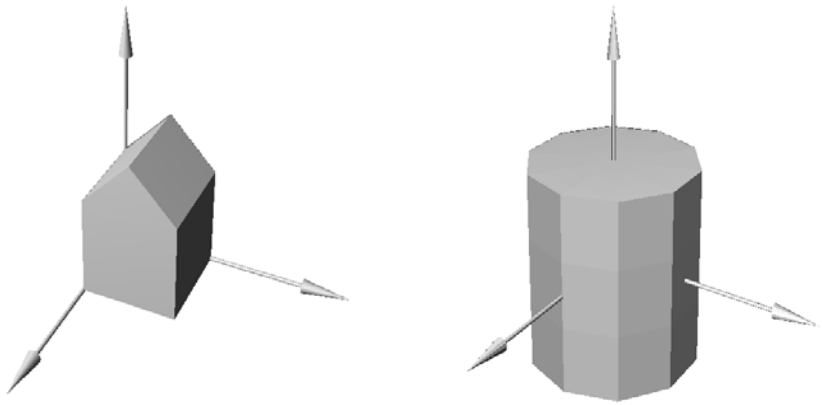
- **Particle system showing water droplets in a fountain. (Courtesy of Philipp Crocoll); Starfield simulation (Courtesy of Ge Wang)**





Polygonal Meshes

- A polygonal mesh is a collection of polygons (faces) that approximate the surface of a 3D object.
  - Examples: surfaces of sphere, cone, cylinder made of polygons (Ch. 5); barn (below).



Polygonal Meshes (2)

- Polygons are easy to represent (by a sequence of vertices) and transform.
- They have simple properties (a single normal vector, a well-defined inside and outside, etc.).
- They are easy to draw (using a polygon-fill routine, or by mapping texture onto the polygon).

Polygonal Meshes (3)

- Meshes are a standard way of representing 3D objects in graphics.
- A mesh can approximate the surface to any degree of accuracy by making the mesh finer or coarser.
- We can also smooth the polygon edges using rendering techniques.

Polygonal Meshes (4)

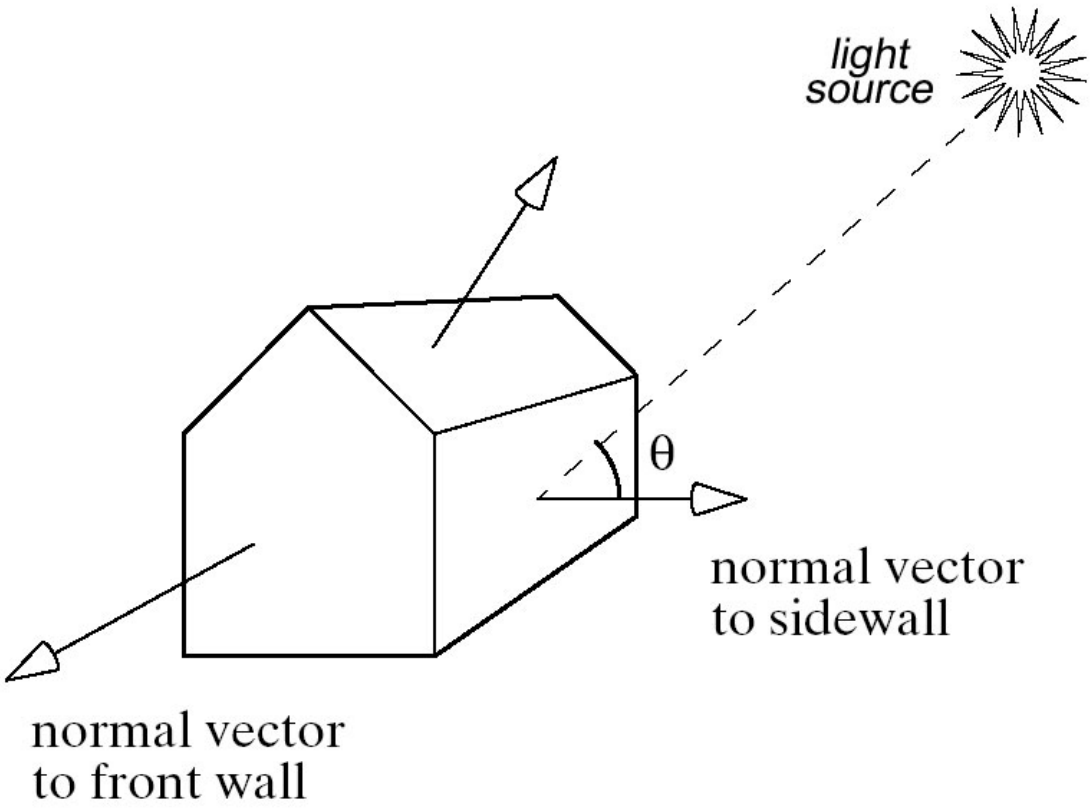
- Meshes can model both solid shapes and thin skins.
  - The object is **solid** if the polygonal faces fit together to enclose space.
  - In other cases, the faces fit together without enclosing space, and so they represent an infinitesimally thin surface.
- In both cases we call the collection of polygons a **polygonal mesh** (or simply a **mesh**).

Polygonal Meshes (5)

- A polygonal mesh is described by a list of polygons, along with information about the direction in which each polygon is facing.
- If the mesh represents a solid, each face has an inside and an outside relative to the rest of the mesh.
- In such a case, the directional information is often simply the outward pointing **normal vector** to the plane of the face used in the shading process.

Polygonal Meshes (6)

- The normal direction to a face determines its brightness.

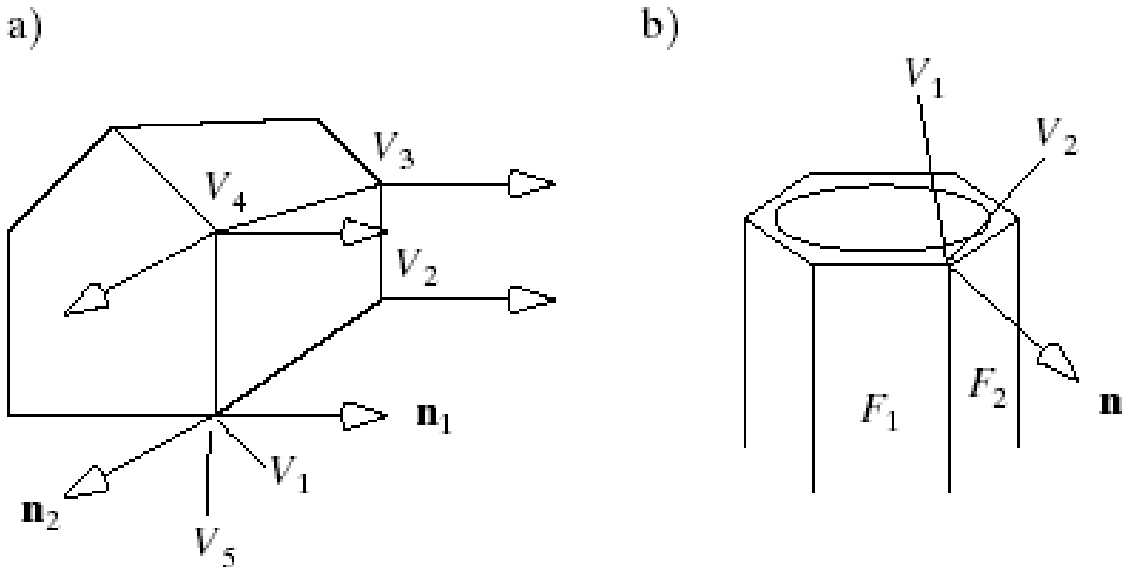


Polygonal Meshes (7)

- For some objects, we associate a *normal vector* to each vertex of a face rather than one vector to an entire face.
  - We use meshes, which represent objects with smoothly curved faces such as a sphere or cylinder. We will refer to the faces of such objects, but with the idea that there is a "*smooth-underlying surface*".
  - When we display such an object, we will want to de-emphasize the individual faces of the object in order to make the object look smooth.

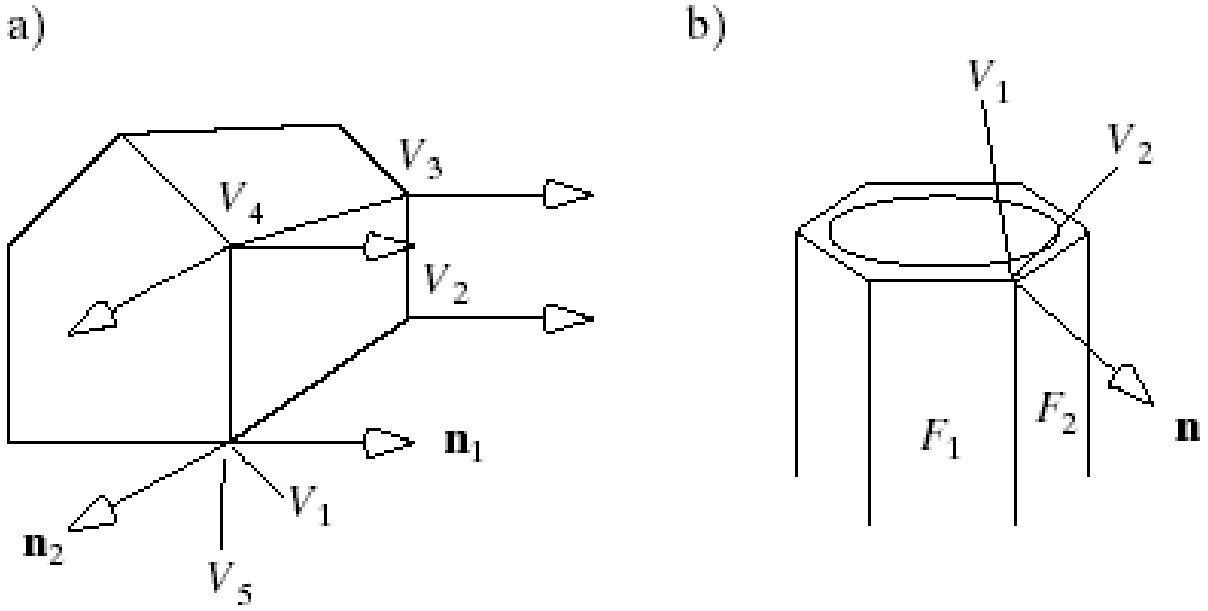
Polygonal Meshes (8)

- Each vertex V_1, V_2, V_3, and V_4 defining the side wall of the barn has the *same* normal $\mathbf{n}_1$, the normal vector to the side wall.
- But vertices of the front wall, such as V_5, will use normal $\mathbf{n}_2$. (Note that vertices V_1 and V_5 are located at the same point in space, but use different normals.)



Polygonal Meshes (9)

- For the smoothly curved surface of the cylinder, both vertex V_1 of face F_1 and vertex V_2 on face F_2 use the same normal ***n***, the vector perpendicular to the underlying smooth surface.

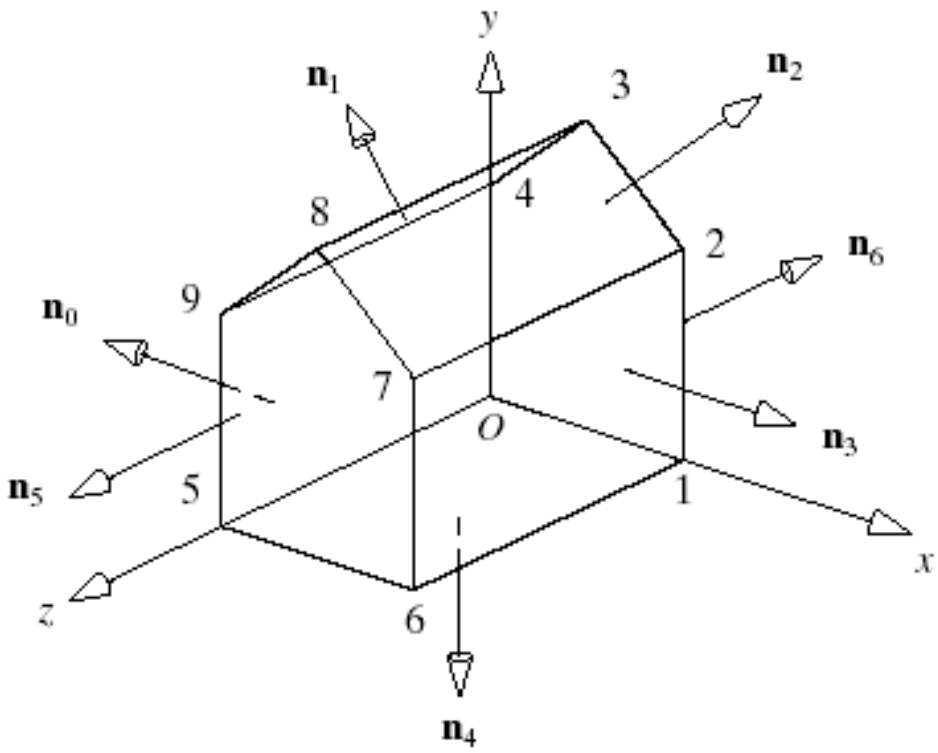


Defining a Polygonal Mesh

- A mesh consists of 3 lists: the vertices of the mesh, the outside normal at each vertex, and the faces of the mesh.
- Example: the basic barn has 7 polygonal faces and 10 vertices (each shared by 3 faces).

Defining a Polygonal Mesh (2)

- It has a square floor one unit on a side.
- Because the barn has flat walls, there are only 7 distinct normal vectors involved, the normal to each face as shown.



Defining a Polygonal Mesh (3)

- The vertex list reports the locations of the distinct vertices in the mesh.
- The list of normals reports the directions of the distinct normal vectors that occur in the model.
- The face list indexes into the vertex and normal lists.

Vertex List for the Barn

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	10
9	0	1	1

Normal List for the Barn

- The normal list (as unit vectors, to the 7 basic planes or polygons).

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 List for the Barn

Face	Vertices	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

Defining a Polygonal Mesh (4)

- Only the indices of the vertices and normals are used.
- The list of vertices for a face begins with any vertex in the face, and then proceeds around the face vertex by vertex until a complete circuit has been made.
  - There are two ways to traverse a polygon: clockwise and counterclockwise. For instance, face #5 above could be listed as (5, 6, 7, 8, 9) or (9, 8, 7, 6, 5).
  - Convention: ***Traverse the polygon counterclockwise as seen from outside the object*.**

Defining a Polygonal Mesh (5)

- Using this order, if you traverse around the face by walking from vertex to vertex, the inside of the face is on your left.
- Using the convention allows algorithms to distinguish with ease the front from the back of a face.
- If we use an underlying smooth surface, such as a cylinder, normals are computed for that surface.

3D File Formats

- There is no standard file format.
- Some formats have been found to be efficient and easy to use: for example, the .qs file format developed by the Stanford University Computer Graphics Laboratory. This particular mesh model has 2,748,318 points (about 5,500,000 triangles) and is based on 566,098 vertices.

3D File Formats (2)

- OpenGL has the capability to load a variety of 3D model formats such as (but not limited to) 3DS, VRML, PLY, MS3D, ASE and OBJ.
- A number of resources are available on the book's companion web site that cover loading 3D mesh models into OpenGL.

Calculating Normals

- Take any three non-collinear points on the face, V_1, V_2, and V_3, and compute the normal as their cross product $\mathbf{m} = (V_1 - V_2) \times (V_3 - V_2)$ and normalize it to unit length.
  - If the two vectors $V_1 - V_2$ and $V_3 - V_2$ are nearly parallel, the cross product will be very small and numerical inaccuracies may result.
  - The polygon may not be perfectly planar. Thus the surface represented by the vertices cannot be truly flat. We need to form some average value for the normal to the polygon, one that takes into consideration all of the vertices.

Newell's Method for Normals

- Given N vertices, define next(i) = ni = (i+1) mod N.
- Traverse the vertices for the face in counter-clockwise order from the outside.
- The normal given by the values on the next slide points to the outside (front) of the face.

Normal (Newell's Method)

$$n_x = \sum_{i=0}^{N-1} (y_i - y_{ni})(z_i + z_{ni})$$

$$n_y = \sum_{i=0}^{N-1} (z_i - z_{ni})(x_i + x_{ni})$$

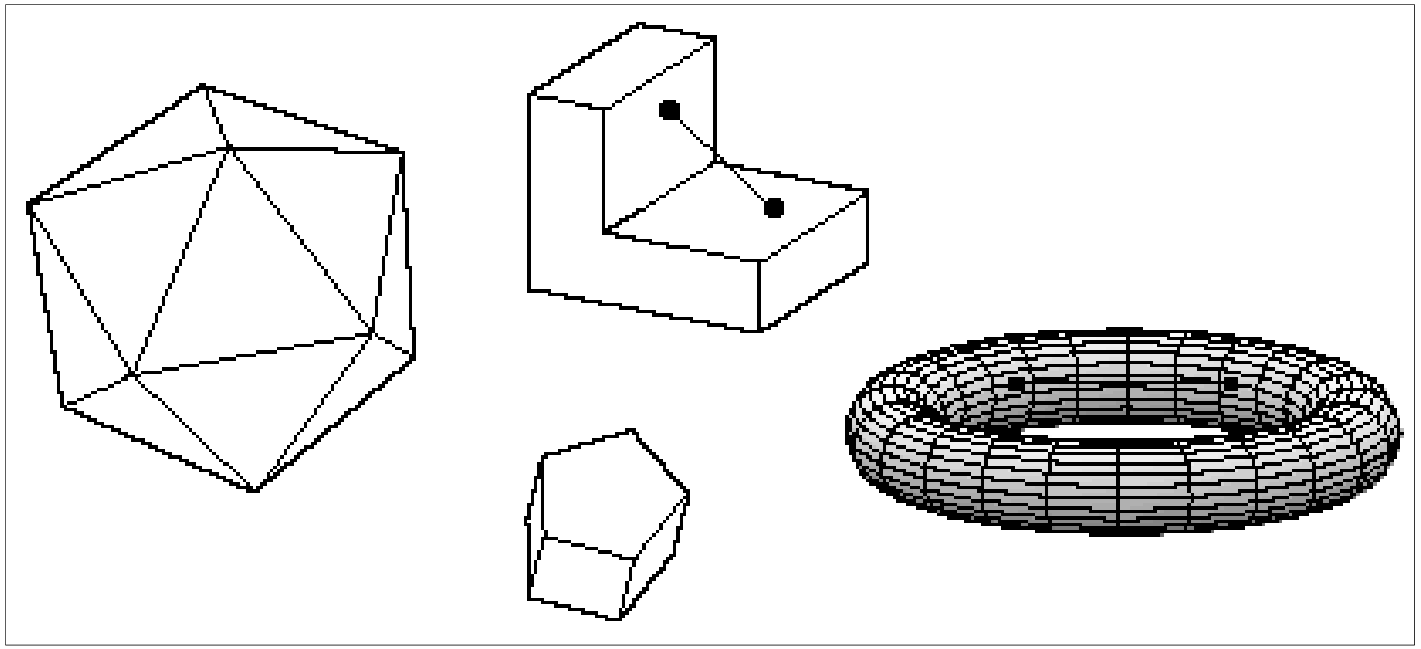
$$n_z = \sum_{i=0}^{N-1} (x_i - x_{ni})(y_i + y_{ni})$$

Properties of Meshes

- A closed mesh represents a solid object (which encloses a volume).
- A mesh is connected if there is an unbroken path along the edges of the mesh between any two vertices.
- A mesh is simple if it has no holes. Example: a sphere is simple; a torus is not.
- A mesh is planar if every face is a plane polygon. Triangular meshes are frequently used to enforce planarity.

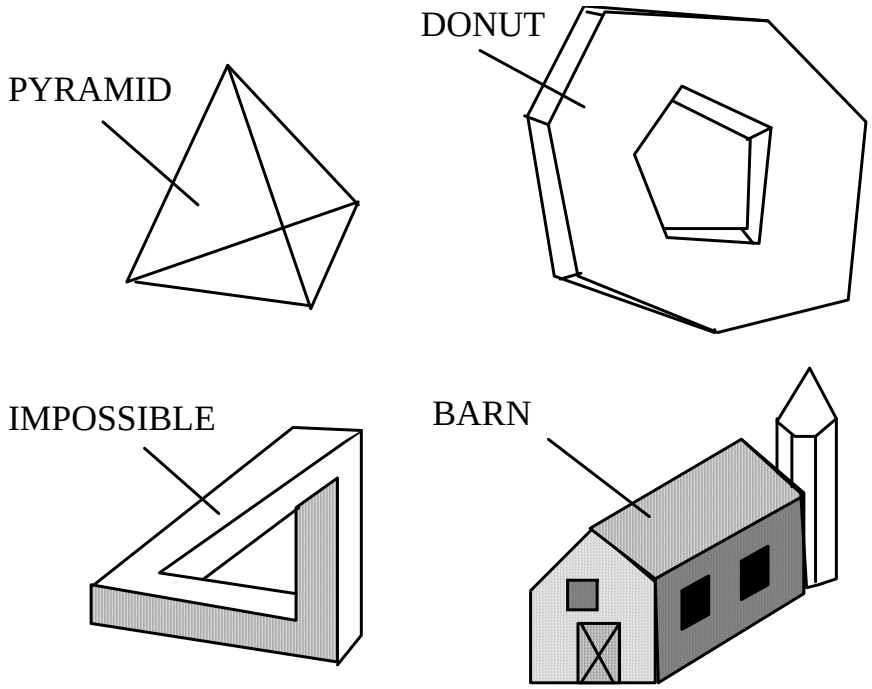
Properties of Meshes (2)

- A mesh is convex if the line connecting any two interior points is entirely inside the mesh.
- Exterior connecting lines are shown for non-convex objects below (step and torus).

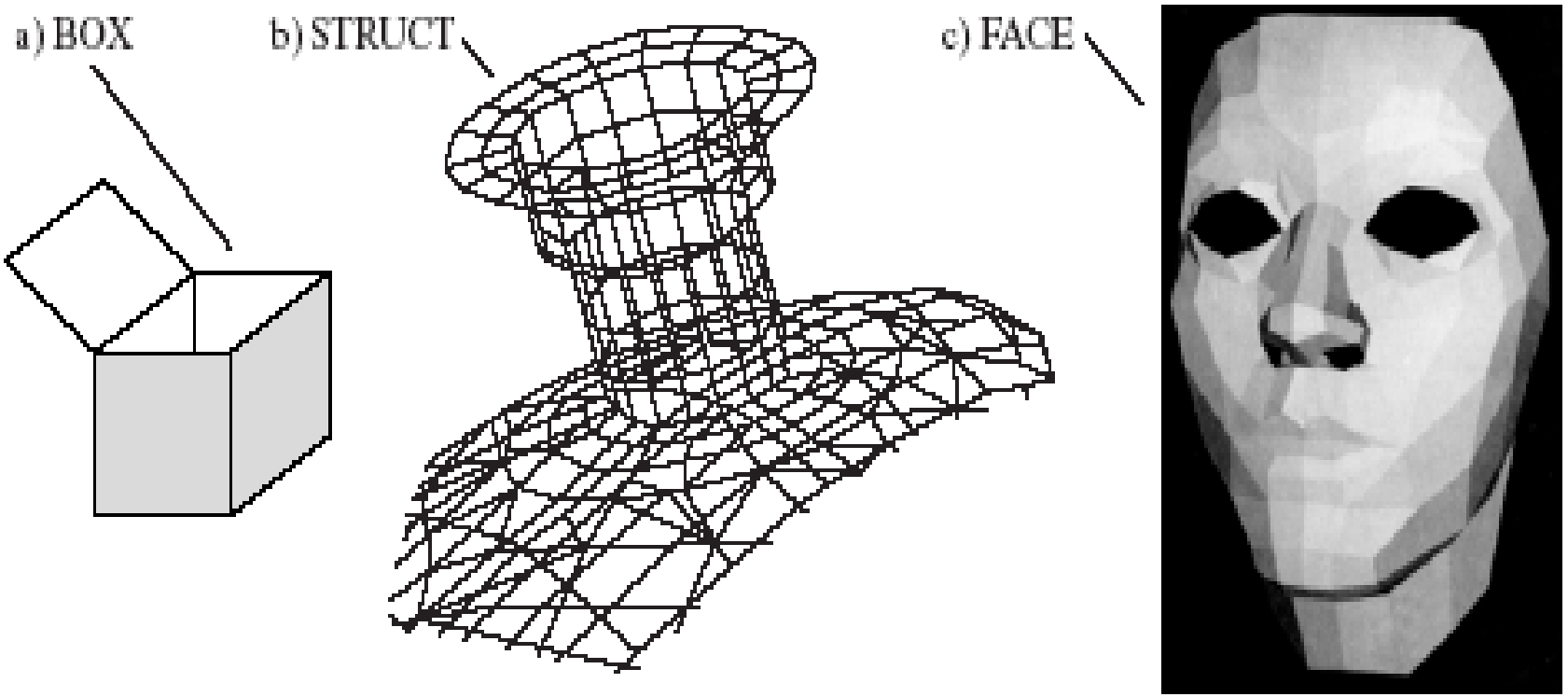


Meshes for Drawing Non-physical Objects

- The figure labeled IMPOSSIBLE looks impossible but is not.
- This object can be represented by a mesh.
- Gershon Elber's web site (http://www.cs.technion.ac.il/~gershon/EscherForReal/) presents a collection of physically impossible objects, and describes how they can be modeled and drawn.



“Thin-skin” Meshes Representing Non-solid Objects



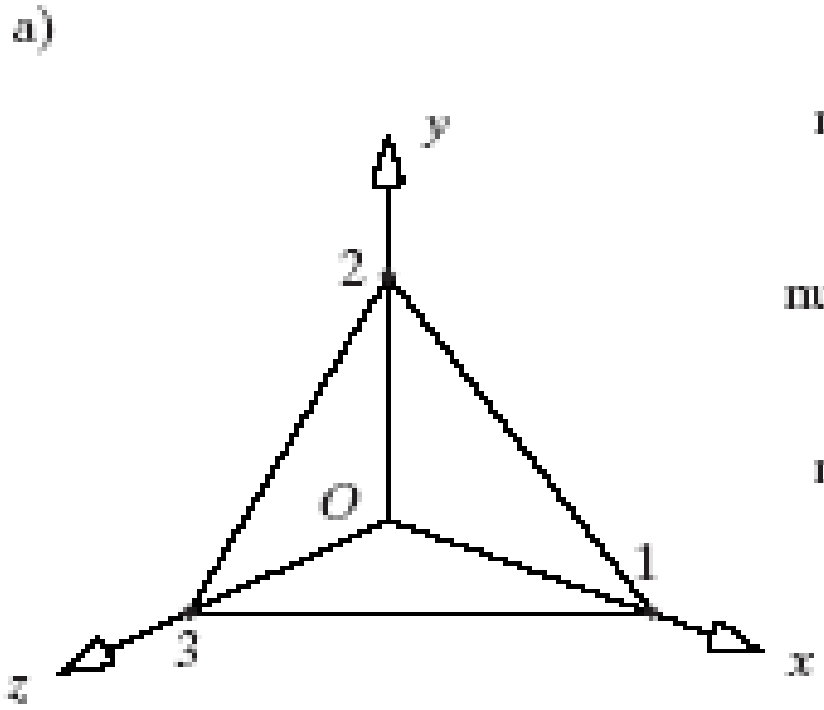
Working with Meshes in a Program

- We want an efficient Mesh class that makes it easy to create and draw the object.
- Since mesh data is frequently stored in a file, we also need simple ways to read and write mesh files.
- Code for classes VertexID, Face, and Mesh is in Fig. 6.15.

Meshes in a Program (2)

- The Face data type is a list of vertices and the normal vector associated with each vertex in the face.
- It is an array of index pairs; the normal to the v^{th} vertex of the f^{th} face has value *norm[face[f].vert[v].normIndex*].
- This indexing scheme is quite orderly and easy to manage, and it allows rapid random access indexing into the pt[ ] array.

Example (tetrahedron & representation)



b)

numVerts	4	0	1	0	0
pt		0	0	1	0
		0	0	0	1
numNorms	4	.577	0	−1	0
norm		.577	0	0	−1
		.577	−1	0	0
numFaces	4				
face		3	3	3	3

1	0	0	1	0	2	1	3
2	0	2	1	3	2	3	3
3	0	1	1	2	2	0	3

Drawing the Mesh Object

- Mesh::draw() (Fig. 6.17) traverses the array of faces in the mesh object, and for each face sends the list of vertices and their normals down the graphics pipeline.
- In OpenGL, to specify that subsequent vertices are associated with normal vector **m,** execute glNormal3f (m.x, m.y, m.z).
- For proper shading, these vectors must be normalized. Otherwise, place glEnable(GL_NORMALIZE) in the init() function. This requests that OpenGL automatically normalize all normal vectors.

SDL and Meshes

- To use SDL, simply develop the Mesh class from the Shape class (as SDL does for you) and add the method drawOpenGL(). The book's companion web site gives full details on the Shape class and SDL's supporting classes.
- The Scene class that reads SDL files is already set to accept the keyword mesh, followed by the name of the file that contains the mesh description: e.g., mesh pawn.3vn

Using SDL to Create and Draw Meshes

- The mesh data are in a file with suffix .3vn.
- The first line of the file lists the number of vertices, number of normals, and number of faces, separated by whitespace.
- The second line begins the list of vertices, giving their x, y and z coordinates separated by whitespace.
  - Multiple vertex coordinates may be listed on a single line.

Using SDL (2)

- After all vertices have been listed, the list of normals begins. A normal is specified by nx, ny, and nz, separated by whitespace.
  - Multiple normal values may be listed on a single line.
- The list of faces follows. A face is specified by the number of vertices it has, the list of vertex indices (in counter-clockwise order from outside), and the list of normal indices (same order as the vertex indices).

Using SDL (3)

- We can also use the matrix manipulation functions of SDL to position and orient the mesh drawing.
- Example:
  - push translate 3 5 4 scale 3 3 3 mesh pawn.3vn pop

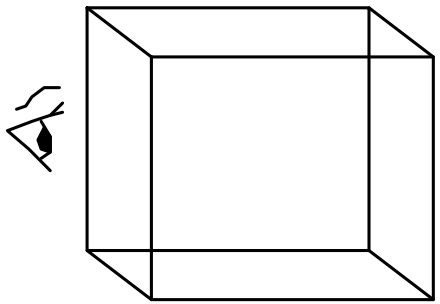
Meshes for Polyhedra

- Polyhedron: connected mesh of simple planar polygons that encloses a finite volume.
  - Every edge is shared by exactly 2 faces.
  - At least 3 edges meet at each vertex.
  - Faces do not interpenetrate. They touch either not at all, or only along their common edge.
  - Euler's formula: V + F - E = 2 for a simple polyhedron.

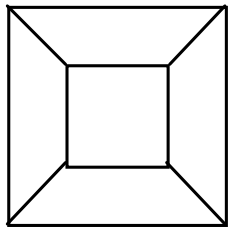
Schlegel Diagrams

- A **Schlegel diagram** reveals the structure of a polygon.
- It views the polyhedron from a point just outside the center of one of its faces.
- The front face appears as a large polygon surrounding the rest of the faces.

a).



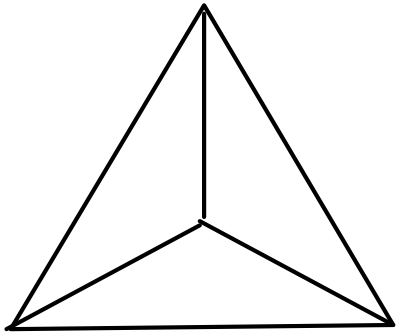
b).



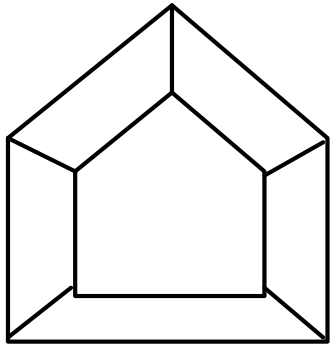
Schlegel Diagrams (2)

- Part a) shows the Schlegel diagram of a pyramid, and parts b) and c) show two different Schlegel diagrams for the basic barn. (Which faces are closest to the eye?).

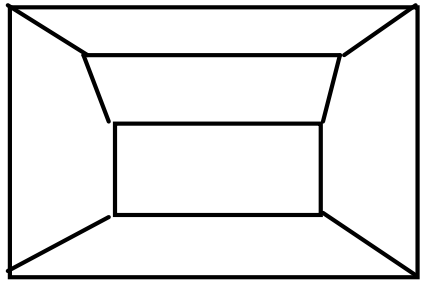
a).



b).

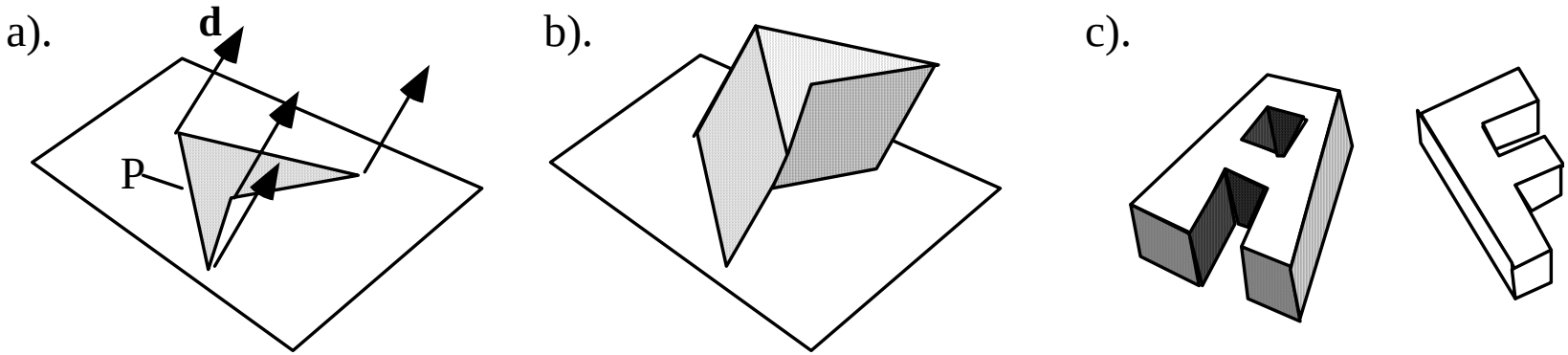


c).



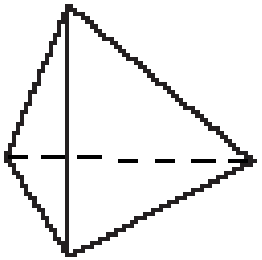
Prisms

- A prism is formed by moving a regular polygon along a straight line.
- When the line is perpendicular to the polygon, the prism is a right prism.

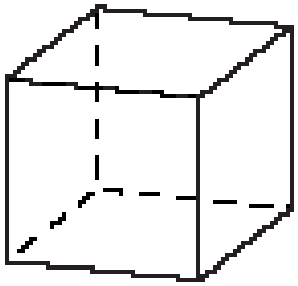


Platonic Solids

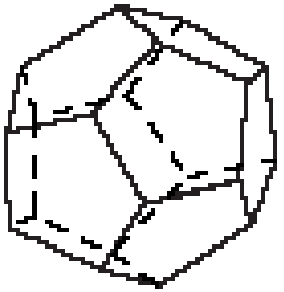
- All the faces are identical and each is a regular polygon.



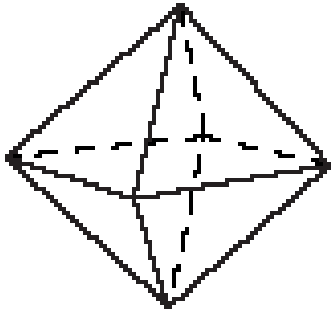
Tetrahedron



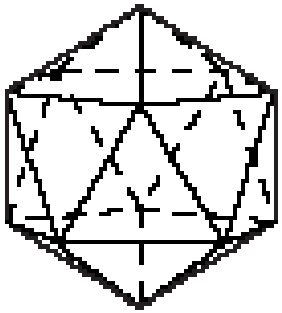
Hexahedron



Dodecahedron



Octahedron



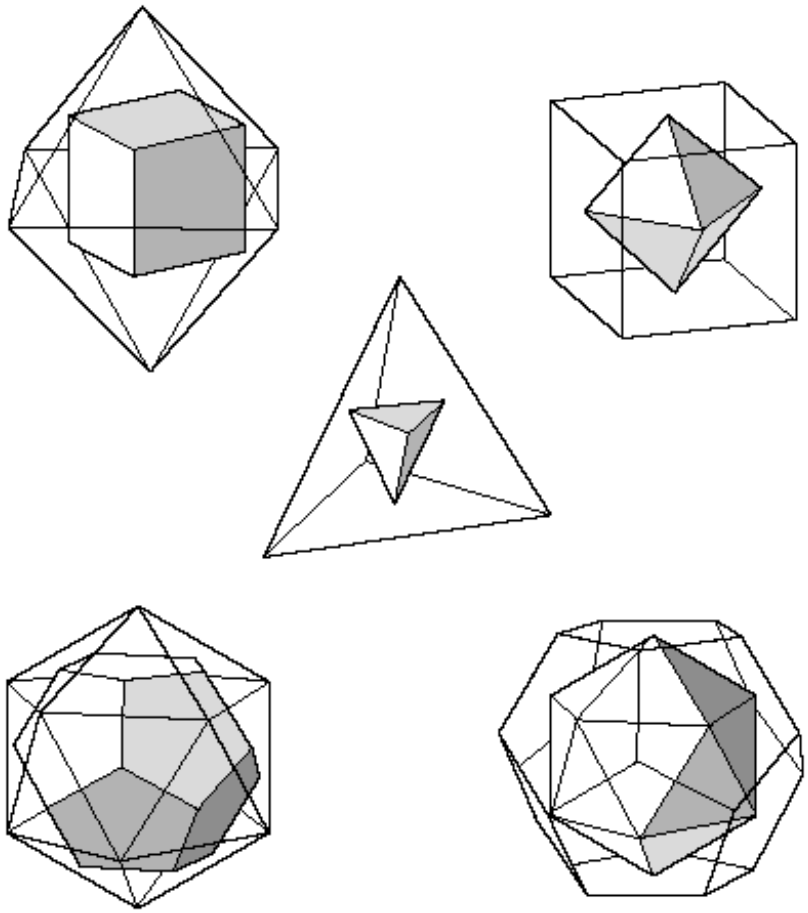
Isosahedron

Duals

- Every Platonic solid P has a dual Platonic solid D. The vertices v_i of D are the centers of faces of P, calculated as

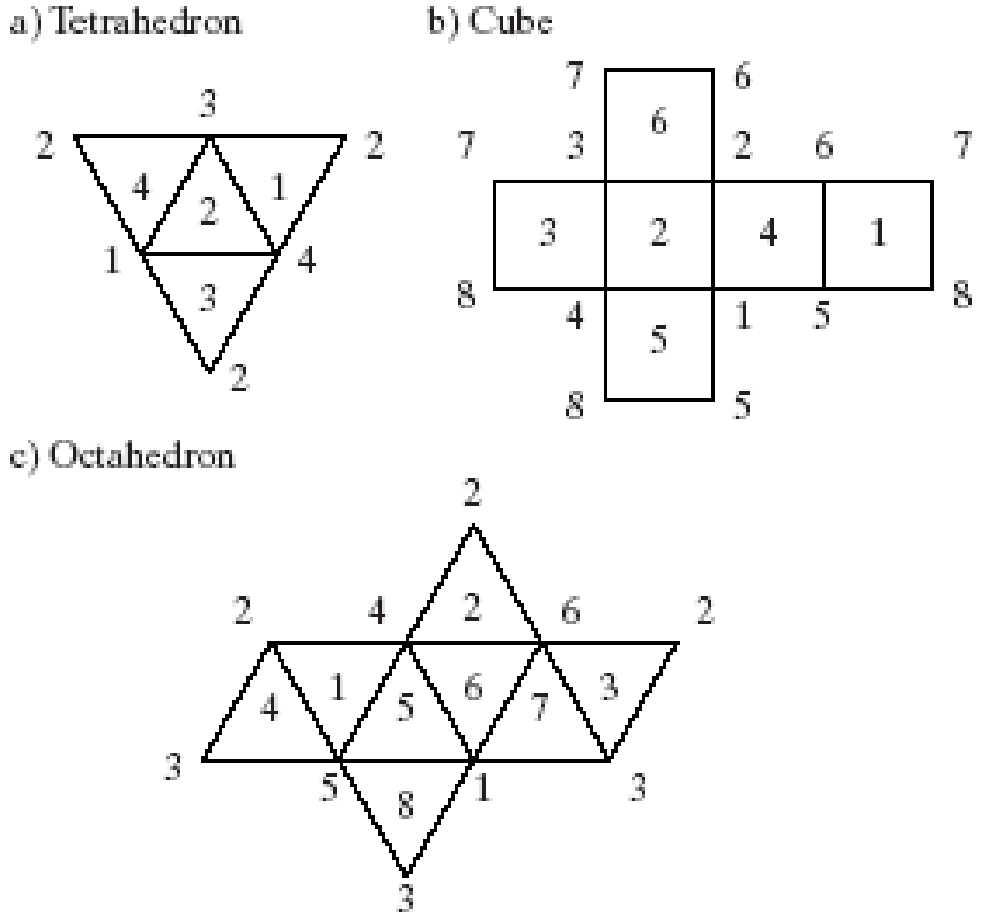
$$c = \frac{1}{n} \sum_{i=0}^{n-1} v_i$$

- The duals are tetrahedron-tetrahedron, hexahedron-octahedron, dodecahedron-icosahedron.



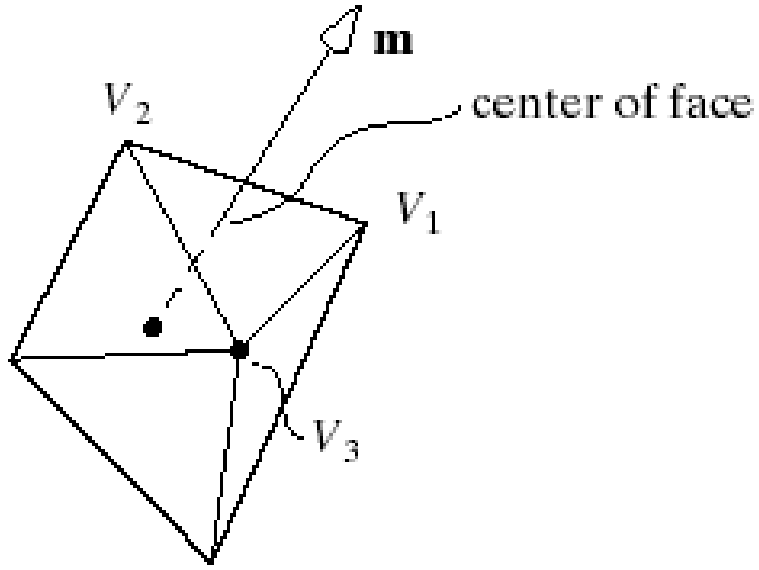
Flattened Models

- To keep track of vertex and face numbering, we use a flat **model**, which is made by cutting along certain edges of each solid and unfolding it to lie flat.



Normal Vectors for the Platonic Solids

- Normals can be found using Newell's method.
- Also, because of the high degree of symmetry of a Platonic solid, **if the solid is centered at the origin**, the normal vector to each face is the vector from the origin to the *center* of the face (the average of the vertices of the face).



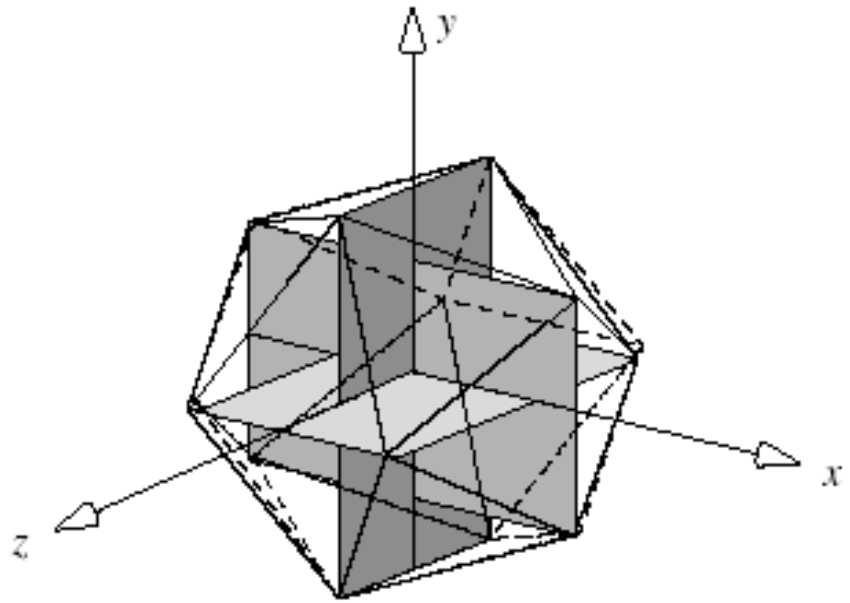
Vertex and Face lists for a Tetrahedron

- For the unit cube having vertices (±1,±1,±1), and the tetrahedron with one vertex at (1,1,1), the tetrahedron has vertex and face lists given below.

Vertex list					Face list	
vertex	x	y	z		Face #	vertices
0	1	1	1		0	1, 2, 3
1	1	-1	-1		1	0, 3, 2
2	-1	-1	1		2	0, 1, 3
3	-1	1	-1		3	0, 2, 1

Icosahedron

- This figure shows that three mutually perpendicular **golden rectangles** inscribe the icosahedron. A vertex list may be read directly from this picture.



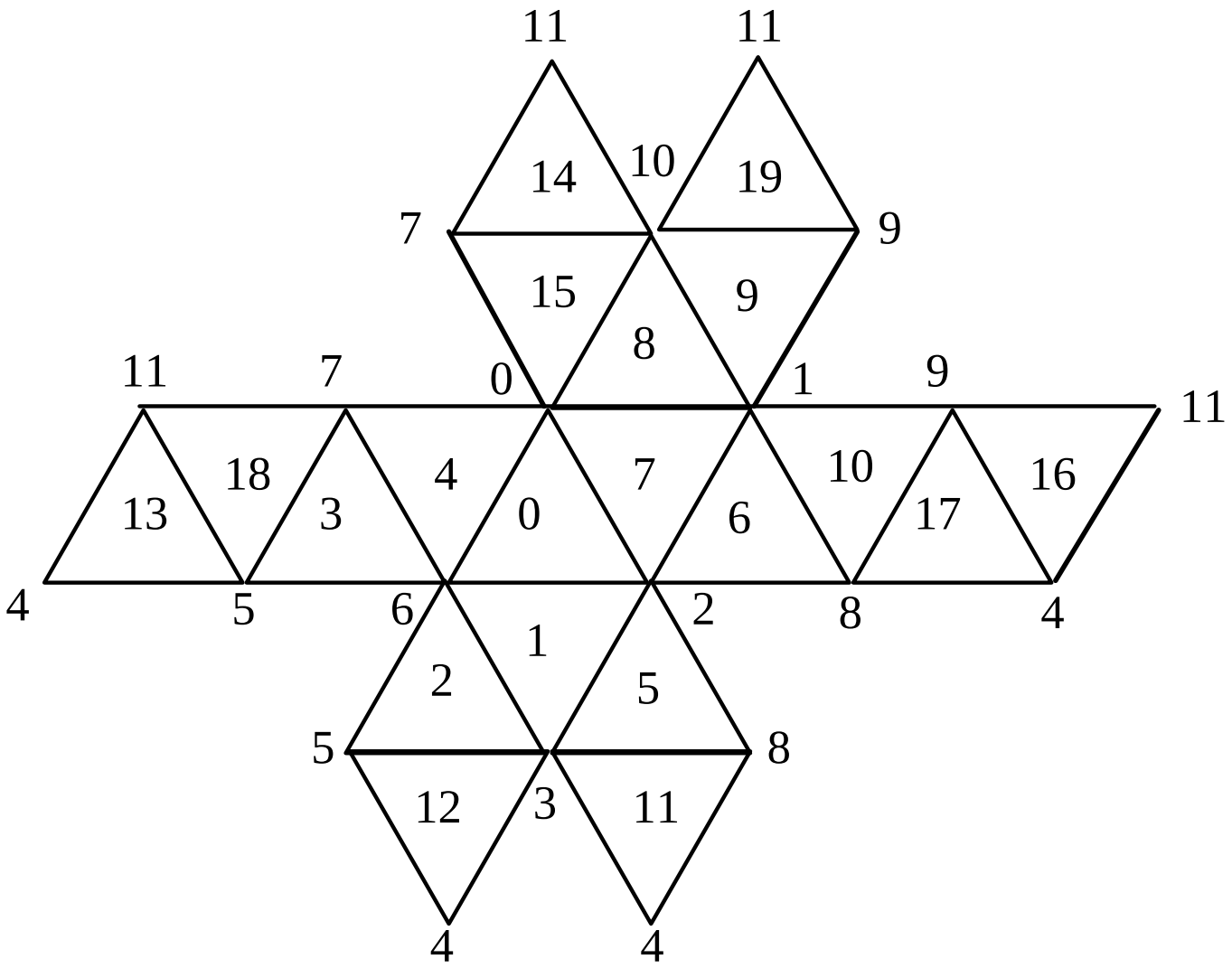
Icosahedron (2)

- We choose to align each golden rectangle with a coordinate axis. For convenience, we define one rectangle so that its longer edge extends from -1 to 1 along the x-axis, and its shorter edge extends from -φ to φ, where φ = 0.618 is the reciprocal of the golden ratio Φ.

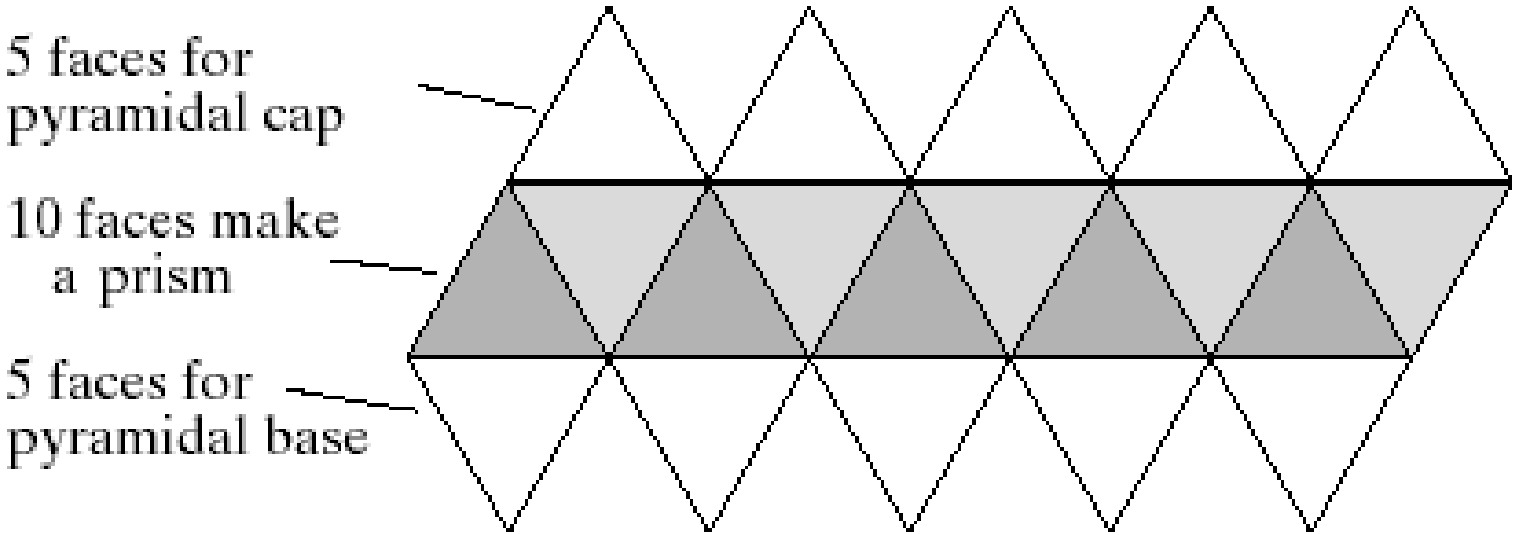
Vertex List for the Icosahedron

Vertex	x	y	z		Vertex	x	y	z
0	0	1	φ		6	φ	0	1
1	0	1	-φ		7	-φ	0	1
2	1	φ	0		8	φ	0	-1
3	1	-φ	0		9	-φ	0	-1
4	0	-1	-φ		10	-1	φ	0
5	0	-1	φ		11	-1	-φ	0

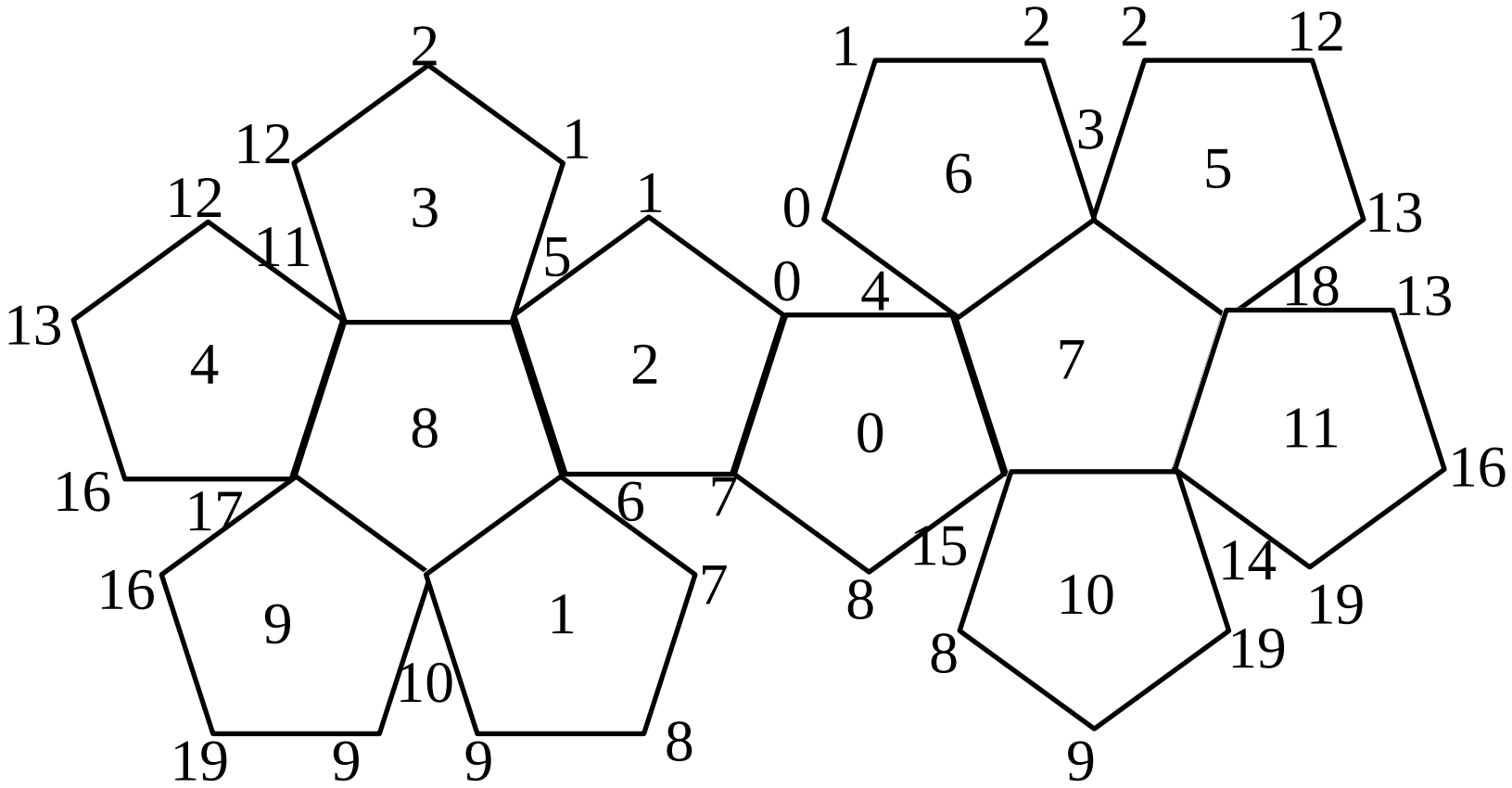
Flattened Model for Icosahedron



Prism Model for Icosahedron

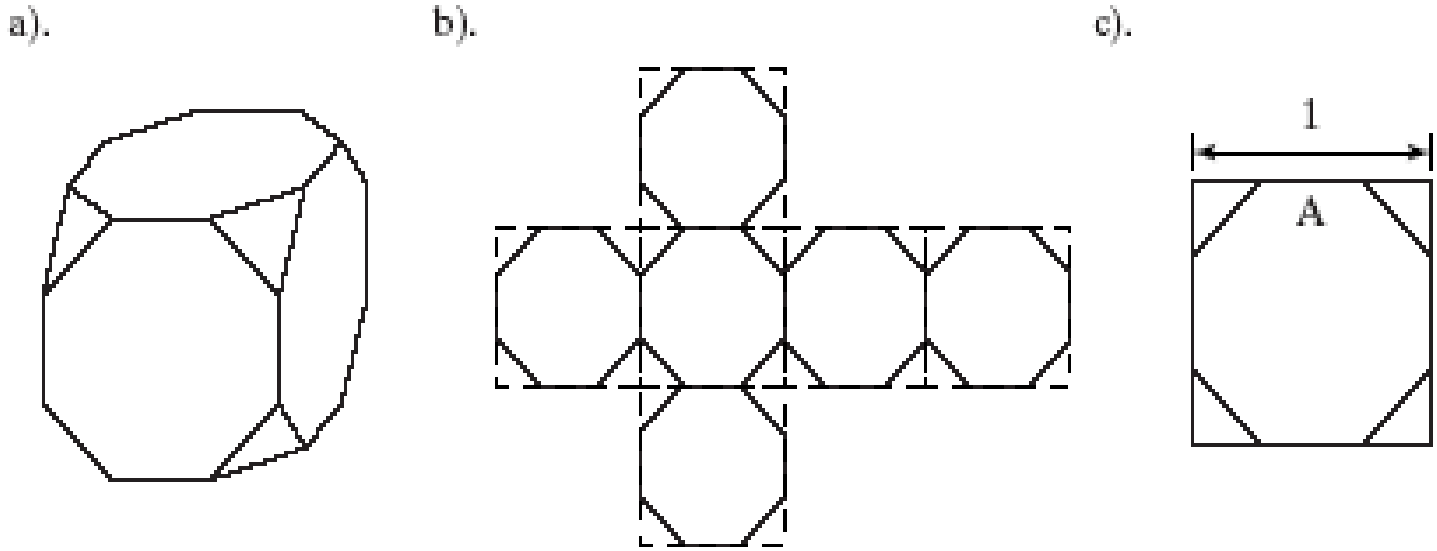


Flattened Model for Dodecahedron



Archimedean Solids

- Have more than one type of polygon as faces; semi-regular.
- Examples: truncated cube (octagon and triangle)



Truncated Cube

- Each edge of the cube is divided into three parts; the middle part of length $A = \frac{1}{\left(1+\sqrt{2}\right)}$ and the middle portion of each edge is joined to its neighbors.
- Thus if an edge of the cube has endpoints *C* and *D*, two new vertices, *V* and *W,* are formed as the affine combinations

$$V = \frac{1+A}{2}C + \frac{1-A}{2}D \quad W = \frac{1-A}{2}C + \frac{1+A}{2}D$$

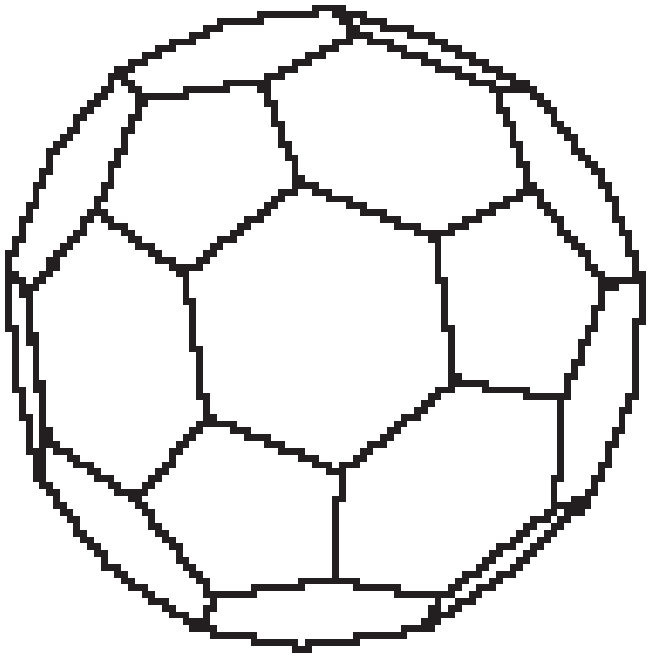
Number of Archimedean Solids

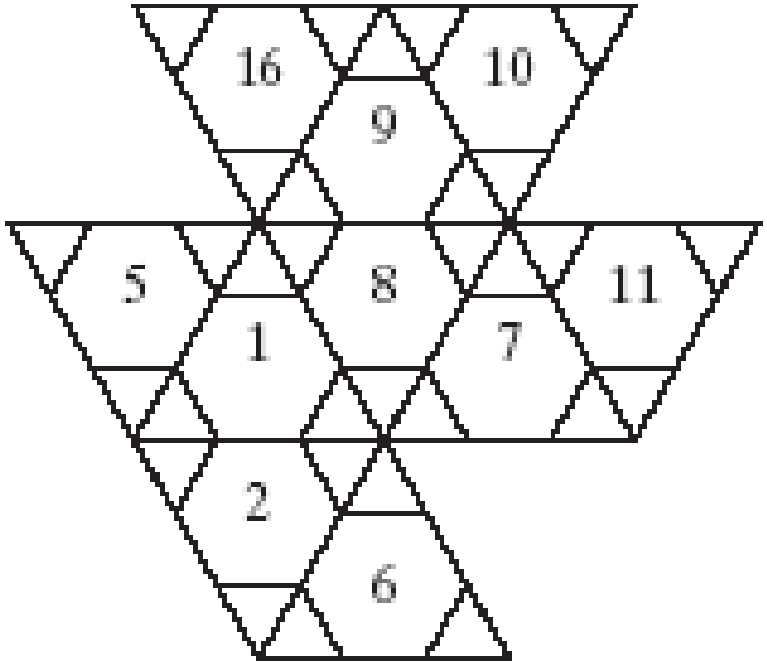
- Given the constraints that faces must be regular polygons, and that they must occur in the same arrangement about each vertex, there are only 13 possible Archimedean solids.
- Archimedean solids have sufficient symmetry that the normal vector to each face is found using the center of the face.

Truncated Icosahedron

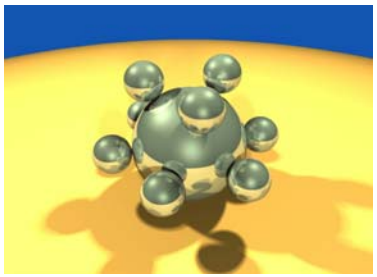
- The truncated icosahedron (soccer ball) consists of regular hexagons and pentagons.
- More recently this shape has been named the **Buckyball** after Buckminster Fuller, because of his interest in geodesic structures similar to this.
- Crystallographers have discovered that 60 atoms of carbon can be arranged at the vertices of the truncated icosahedron, producing a new kind of carbon molecule that is neither graphite nor diamond.
- The material has been named **Fullerene**.

The Buckyball and Flattened Version (Partial)





Computer Graphics using OpenGL, 3rd Edition
F. S. Hill, Jr. and S. Kelley



Chapter 6.4-5
Modeling Shapes with Polygonal Meshes

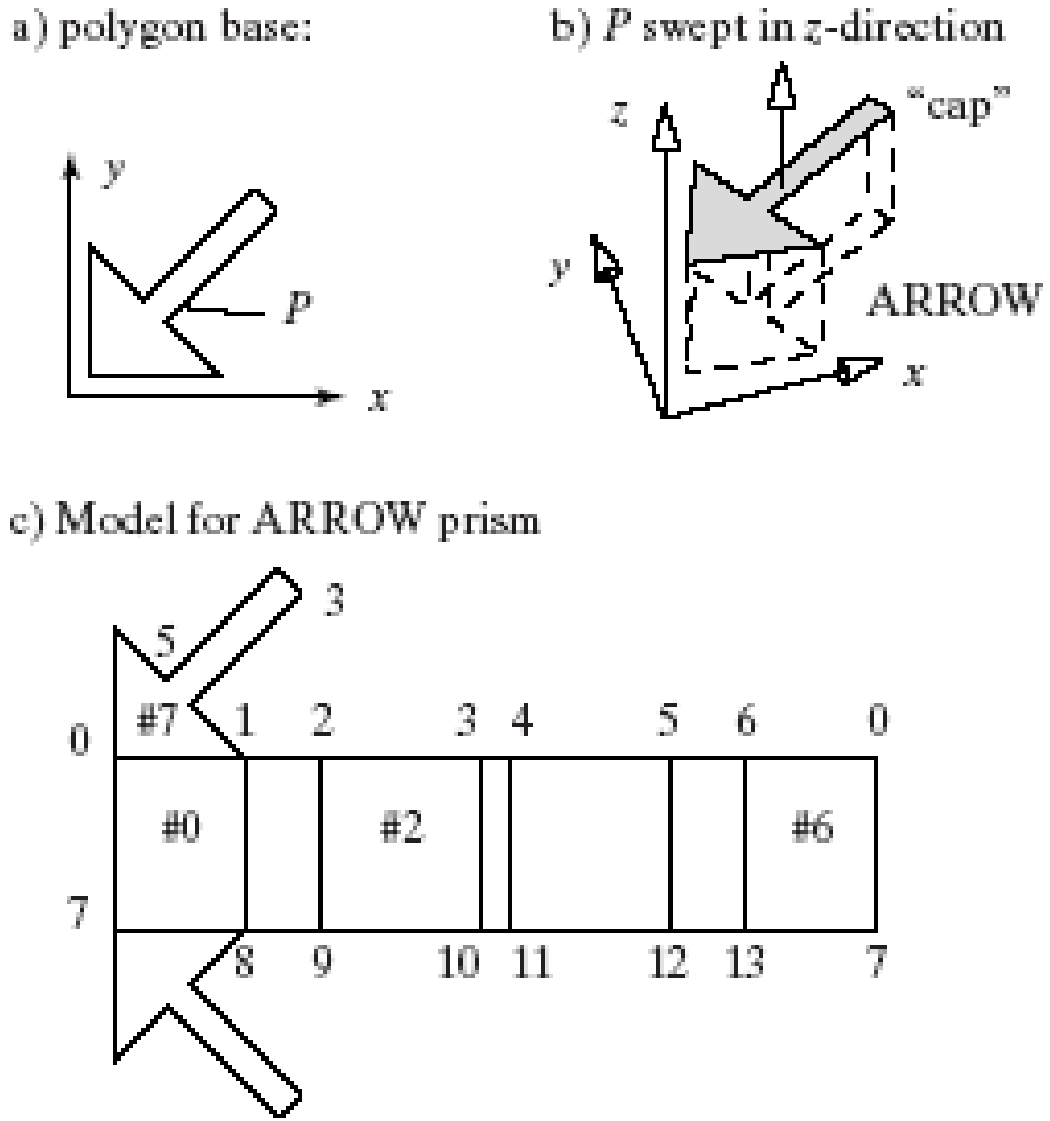
S. M. Lea
University of North Carolina at Greensboro

Extruded Shapes

- A large class of shapes can be generated by **extruding** or **sweeping** a 2D shape through space.
- In addition, surfaces of revolution can also be approximated by extrusion of a polygon once we slightly broaden the definition of extrusion.

Extruded Shapes

- Prism: formed by sweeping the arrow along a straight line.
- Flattened version.



Extruded Shapes (2)

- Base has vertices $(x_i, y_i, 0)$ and top has vertices (x_i, y_i, H).
- Each vertex (x_i, y_i, H) on the top is connected to corresponding vertex $(x_i, y_i, 0)$ on the base.
- If the polygon has n sides, then there are n vertical sides of the prism plus a top side (cap) and a bottom side (base), or n+2 faces altogether.
- The normals for the prism are the face normals. These may be obtained using the Newell method, and the normal list for the prism constructed.

Vertex List for the Prism

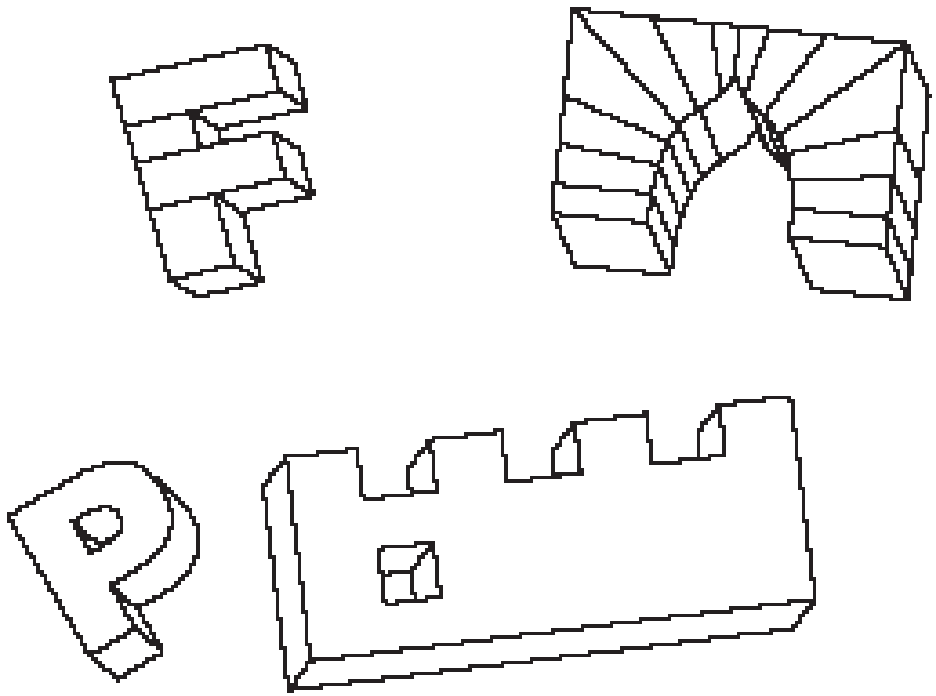
- Suppose the prism's base is a polygon with N vertices (x_i, y_i). We number the vertices of the base 0, . . . , N-1 and those of the cap N, . . ., 2N -1, so that an edge joins vertices i and $i + N$.
- The vertex list is then easily constructed to contain the points $(x_i, y_i, 0)$ and (x_i, y_i, H), for i = 0, 1, ..., N-1.

Face List for the Prism

- We first make the side faces and then add the cap and base.
- For the *j*-th wall ($j = 0,\ldots,N-1$) we create a face with the four vertices having indices j, $j + N$, $next(j) + N$, and $next(j)$ where $next(j)$ is $j+1 \ \% \ N$.
- Code: `if (j < n-1) next = ++j; else next = 0;`
- Or `j = (++j) % N;`

Arrays of Extruded Prisms

- OpenGL can reliably draw only convex polygons. For non-convex prisms, stack the parts.

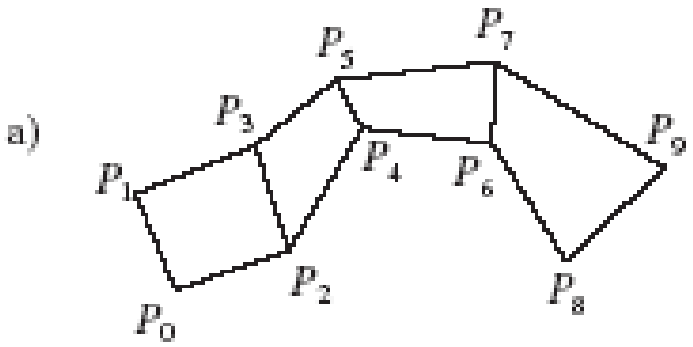


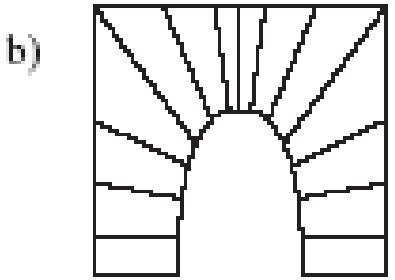
Drawing Arrays of Extruded Prisms

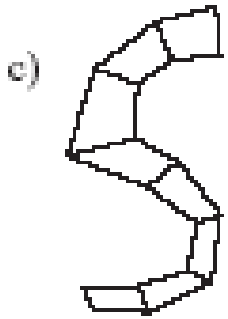
- We need to build a mesh out of an array of prisms: void Mesh:: makePrismArray(...)
- Its arguments are a list of (convex) base polygons (in the *xy*-plane), and perhaps a vector **d** that describes the direction and amount of extrusion.
- The vertex list contains the vertices of the cap and base polygons for each prism, and the individual walls, base, and cap of each prism are stored in the face list.
- Drawing such a mesh involves some wasted effort, since walls that abut would be drawn (twice), even though they are ultimately invisible.

Special Case: Extruded Quadstrips

- Quadstrip (an OpenGL primitive) can be created and then extruded as for prism.







Quadstrip Data Structure

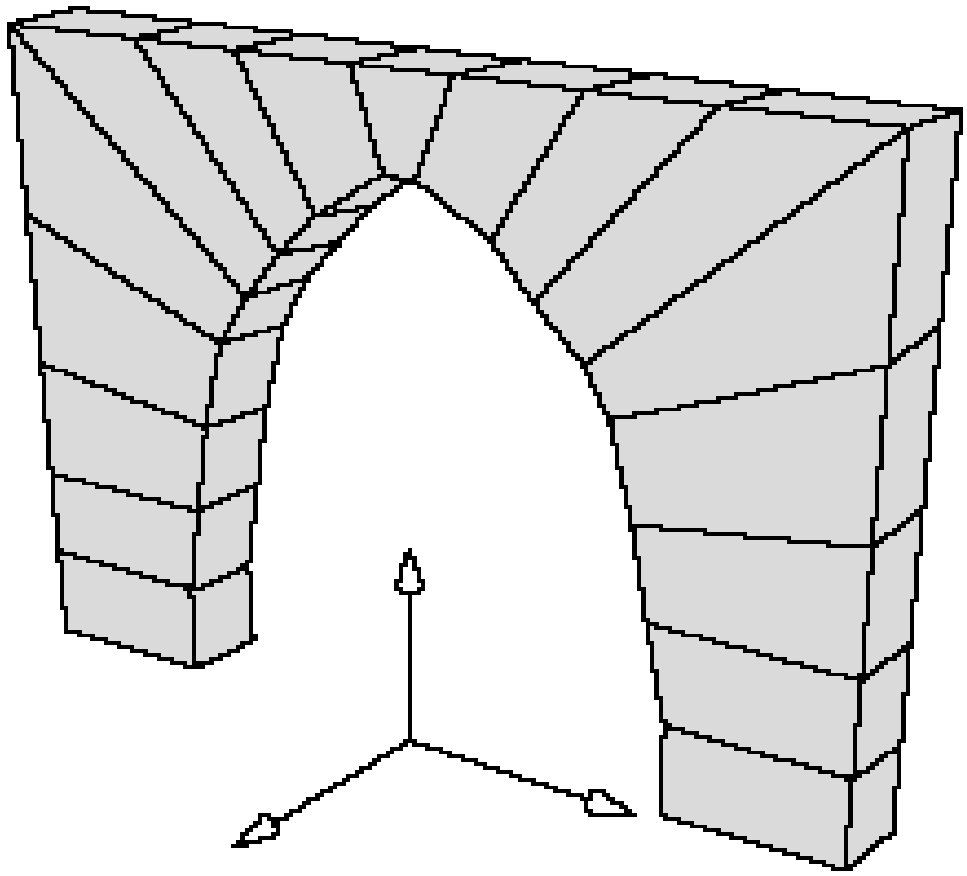
- quad-strip = $\{p_0, p_1, p_2, ..., p_{M-1}\}$
- The vertices are understood to be taken in pairs, with the *odd* ones forming one edge of the quad-strip, and the *even* ones forming the other edge.
- Not every polygon can be represented as a quad-strip.

Drawing Extruded Quadstrips

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

Example: Arch

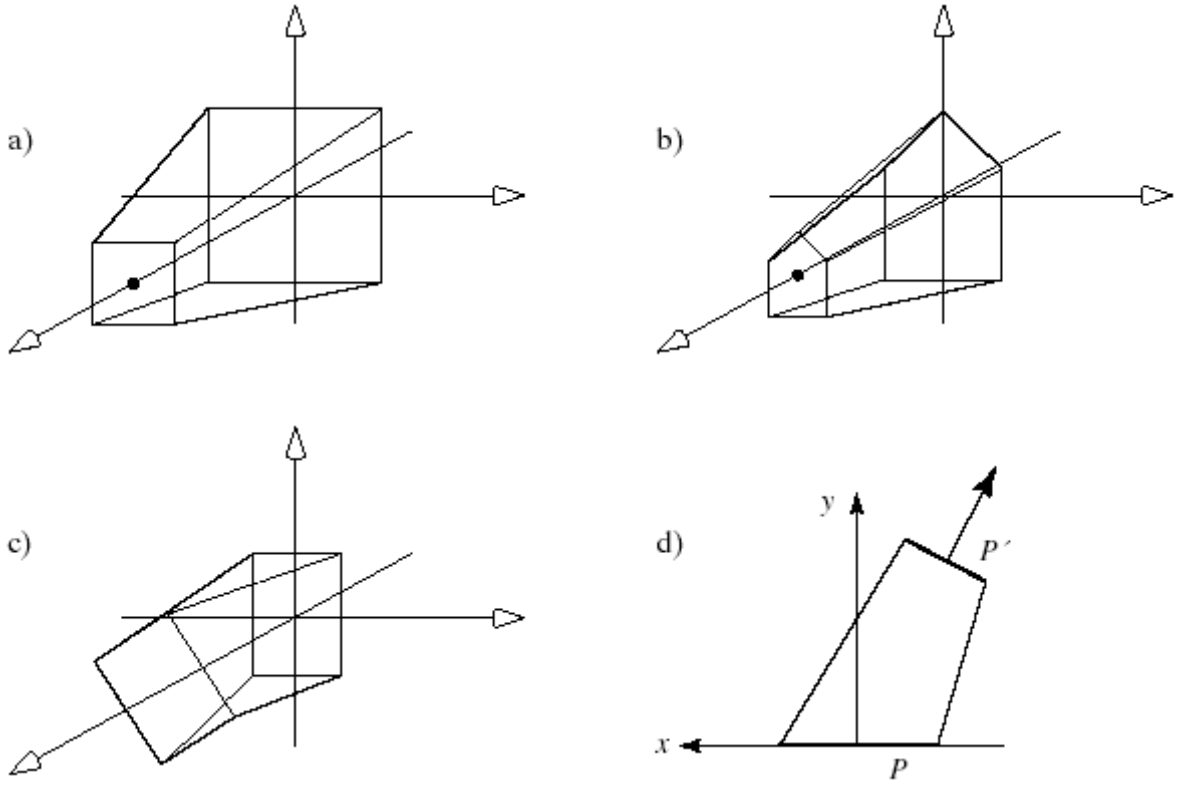


Special Case: Twisted Extrusions

- Base is n-gon, top is scaled, translated, and possibly rotated version of base.
- Specifically, if the base polygon is P, with vertices $\{p_0, p_1, ..., p_{N-1}\}$, the cap polygon has vertices $P' = \{ Mp_0, Mp_1, ..., Mp_{N-1}\}$ where M is some 4 by 4 affine transformation matrix.

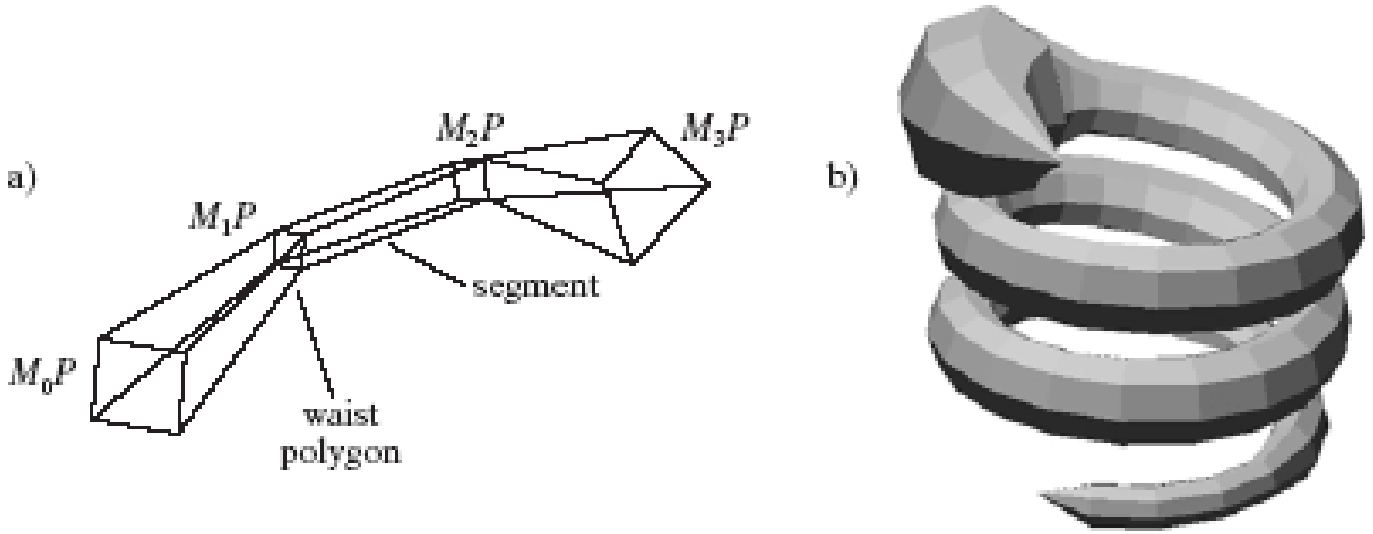
Examples

- A), B): cap is smaller version of base.
- C): cap is rotated through θ about *z*-axis before translation.
- D): cap *P*' is rotated arbitrarily before translation.



Segmented Extrusions

- Below: a square P extruded three times, in different directions with different tapers and twists. The first segment has end polygons M_0P and M_1P, where the initial matrix M_0 positions and orients the starting end of the tube. The second segment has end polygons M_1P and M_2P, etc.

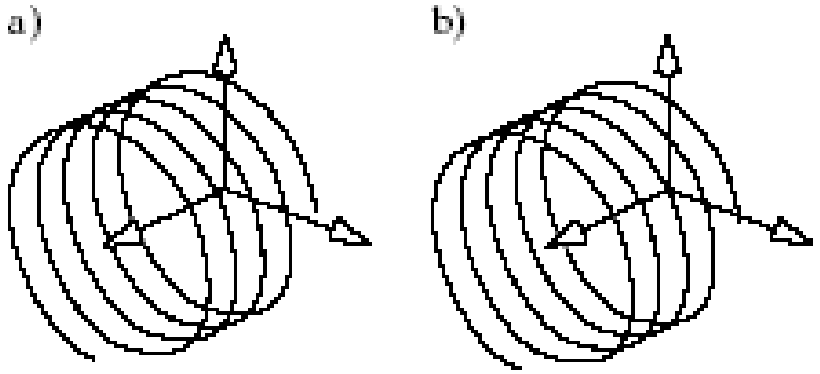


Special Case: Segmented Extrusions

- We shall call the various transformed squares the "**waists**" of the tube.
- In this example the vertex list of the mesh contains the 16 vertices M_0p_0, $M_0\,p_1$, $M_0\,p_2$, $M_0\,p_3$, M_1p_0, M_1p_1, M_1p_2, M_1p_3, ..., M_3p0, M_3p_1, M_3p_2, M_3p_3.
- The "snake" used the matrices M_i to grow and shrink the tube to represent the body and head of a snake.

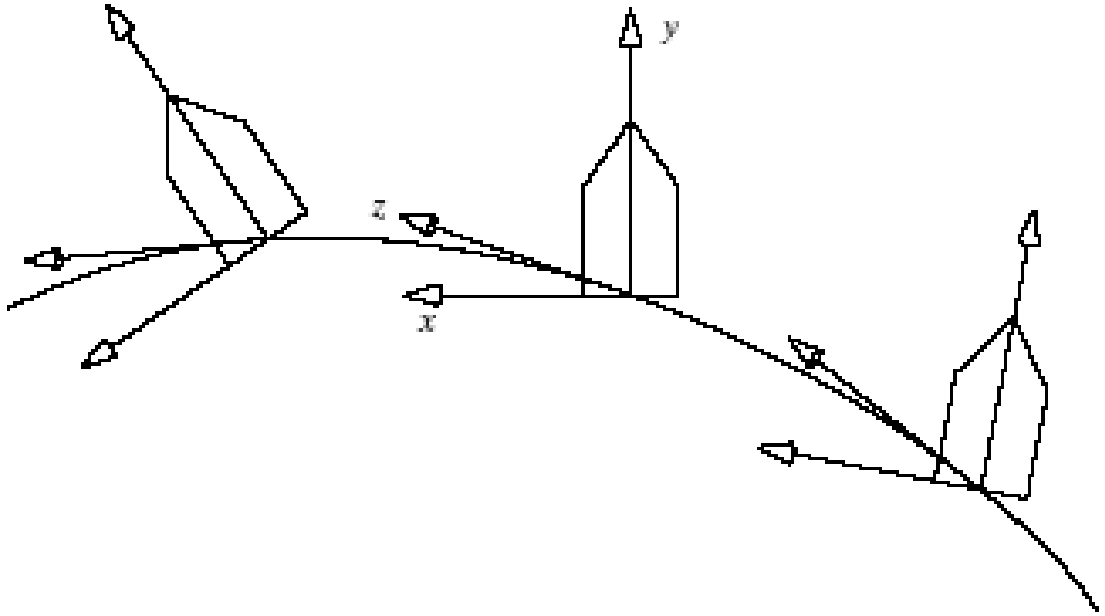
Methods for Twisted Extrusions

- Multiple extrusions used, each with its own transformation. The extrusions are joined together end to end.
- The extrusion tube is wrapped around a space curve C, the spine of the extrusion (e.g., helix $C(t) = (cos(t), sin(t), bt)$).



Method for Twisted Extrusions (2)

- We get the curve values at various points t_i and then build a polygon perpendicular to the curve at $C(t_i)$ using a Frenet frame.



Method for Twisted Extrusions (3)

- We create the **Frenet frame** at each point along the curve: at each value t_i a normalized vector $\boldsymbol{T}(t_i)$ tangent to the curve is computed. It is given by $C'(t_i)$, the derivative of $C(t_i)$.
- Then two normalized vectors, $\mathbf{N}(t_i)$ and $\mathbf{B}(t_i)$, which are perpendicular to $\mathbf{T}(t_i)$ and to each other, are computed. These three vectors constitute the **Frenet frame** at t_i.

Method for Twisted Extrusions (5)

- Once the Frenet Frame is computed, the transformation matrix *M* that transforms the base polygon of the tube to its position and orientation in this frame is the transformation that carries the world coordinate system **i**, **j**, and **k** into this new coordinate system $\mathbf{N}(t_i)$, $\mathbf{B}(t_i)$, $\mathbf{T}(t_i)$, and the origin of the world into the spine point $C(t_i)$.
- Thus the matrix has columns consisting directly of $\mathbf{N}(t_i)$, $\mathbf{B}(t_i)$, $\mathbf{T}(t_i)$, and $C(t_i)$ expressed in homogeneous coordinates:

$$M = (\mathbf{N}(t_i)|\ \mathbf{B}(t_i)|\ \mathbf{T}(t_i)|\ C(t_i))$$

Method for Twisted Extrusions (4)

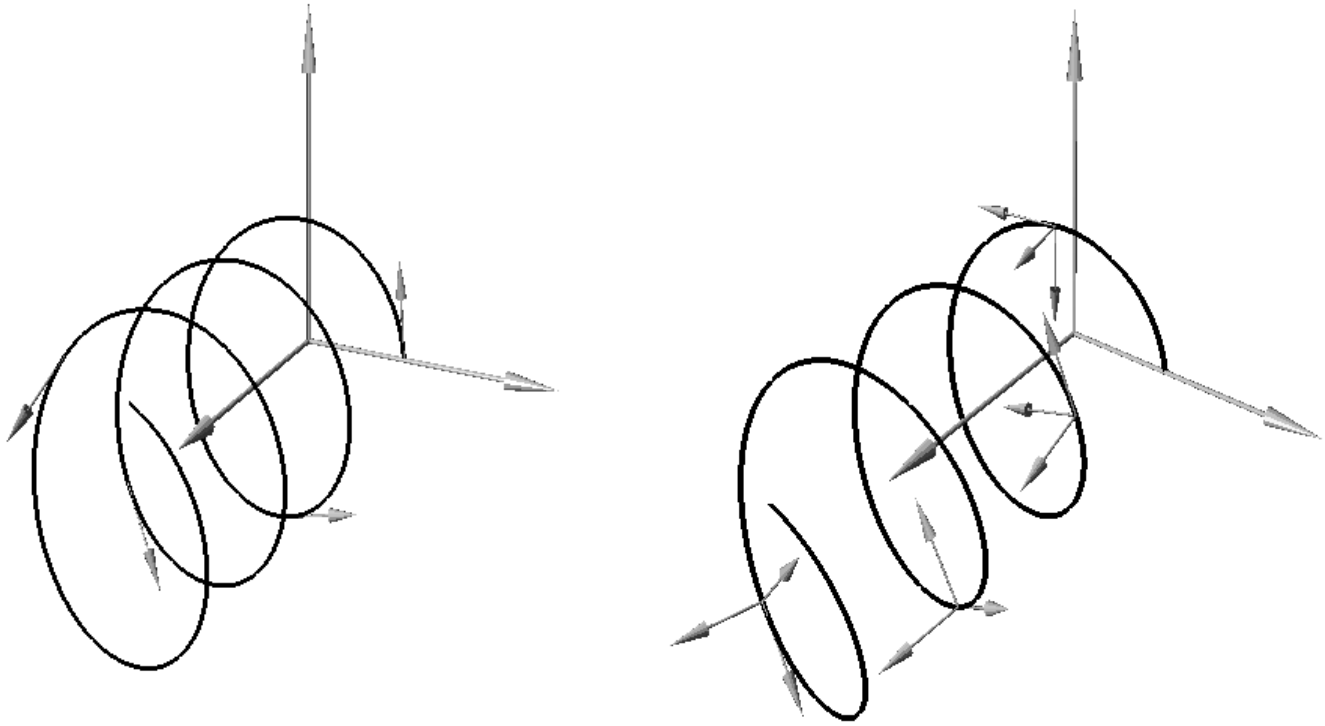
- Example: **C**(t) = (cos (t), sin(t), bt) (a helix)
- The tangent vector to the curve: **T** = derivative of **C**(t); specifically, $\mathbf{T} = (1 + b^2)^{-1}$ (-sin(t), cos(t), b)
- The acceleration vector is the derivative of the tangent vector, and thus the second derivative of the curve **C**: **A** = (-cos(t), -sin(t), 0). **A** is perpendicular to **T**, because **A**·**T** = 0. We let **B** = **T**x**A**/|**T**x**A**|. **B** is called the binormal vector to the curve.
- A final vector **N** = **B**x**T** forms a reference frame, the Frenet frame, at point t_i on the curve. **N** is perpendicular to both **B** and **T**.

Method for Twisted Extrusions (5)

- If the curve is awkward numerically, the derivatives for the reference frame vectors may be approximated over a small distance ε by $\mathbf{T}(t_i) = (\mathbf{C}(t+\varepsilon) - \mathbf{C}(t-\varepsilon))/(2\,\varepsilon)$, $\mathbf{B}(t_i) = (\mathbf{C}(t+\varepsilon) - 2\mathbf{C}(t) + \mathbf{C}(t-\varepsilon))/\varepsilon^2$.

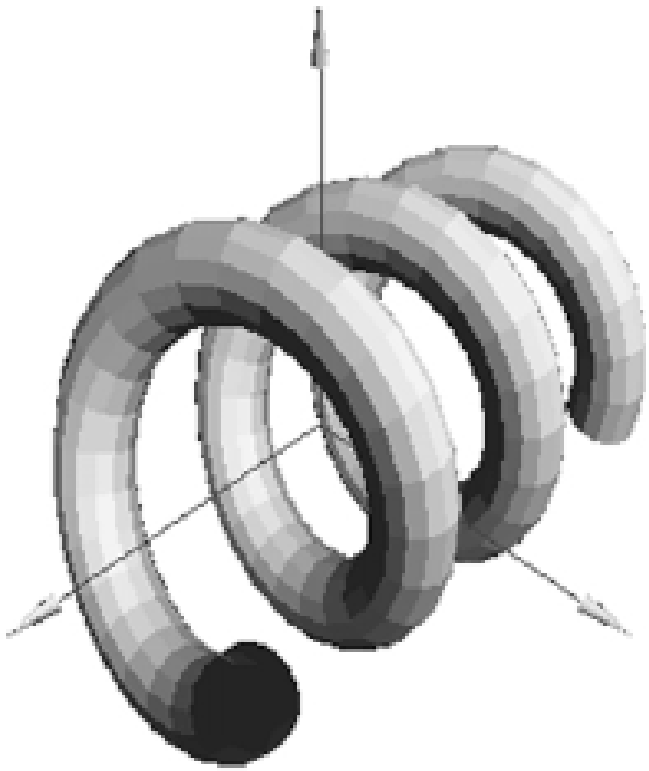
Examples

a). Tangents to the helix. b). Frenet frame at various values of *t*, for the helix.



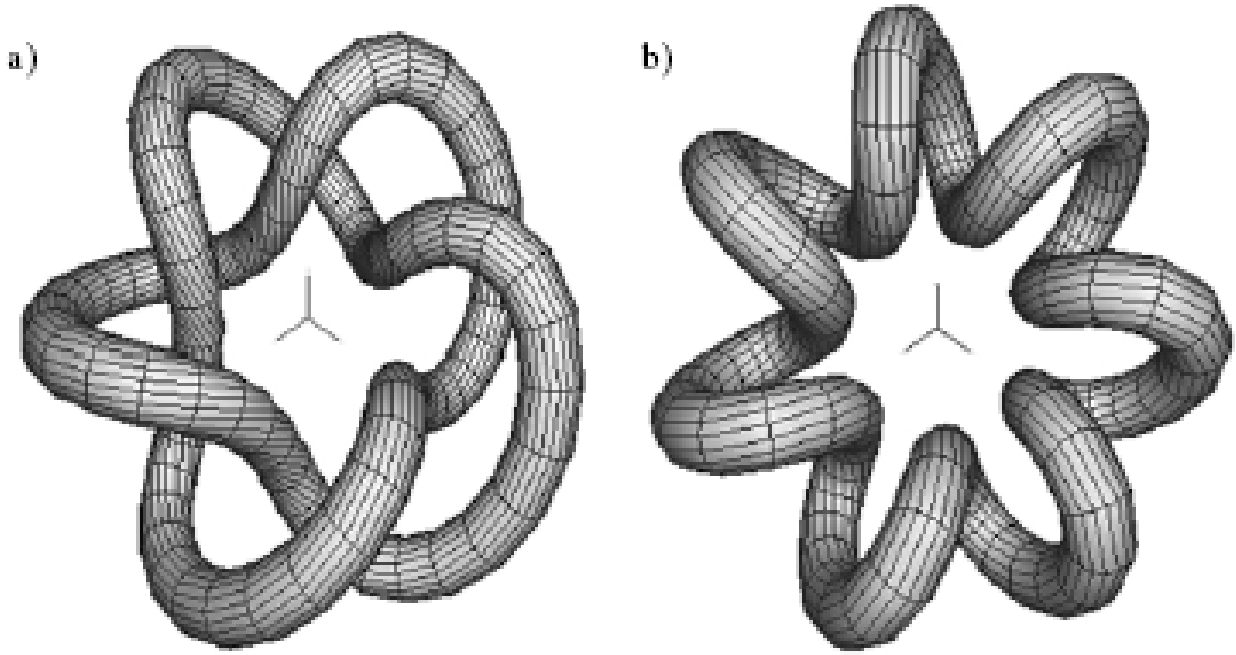
Examples

- Helix, C(t) = (cos t, sin t, bt). A decagon (10 sides) is wrapped around the helix.



Examples (2)

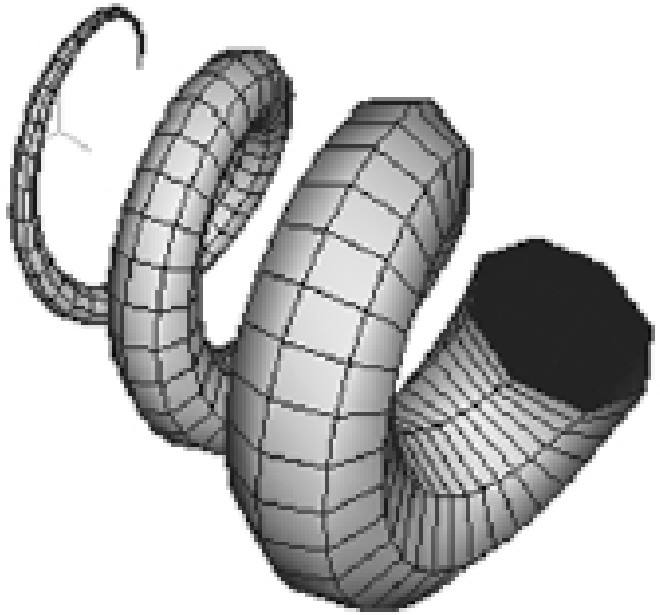
- Toroidal spiral, C(t) = (a + b cos(qt) cos(pt), (a + b cos(qt)) sin(pt), c sin(qt)).
- A) p = 2, q = 5; B) p = 1, q = 7.



Examples (3)

- Helix with t-dependent scaling: Matrix M_i is multiplied by a matrix which provides t-dependent scaling (g(t) = t) along the local x and y.

$$M' = M \begin{pmatrix} g(t) & 0 & 0 & 0 \\ 0 & g(t) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

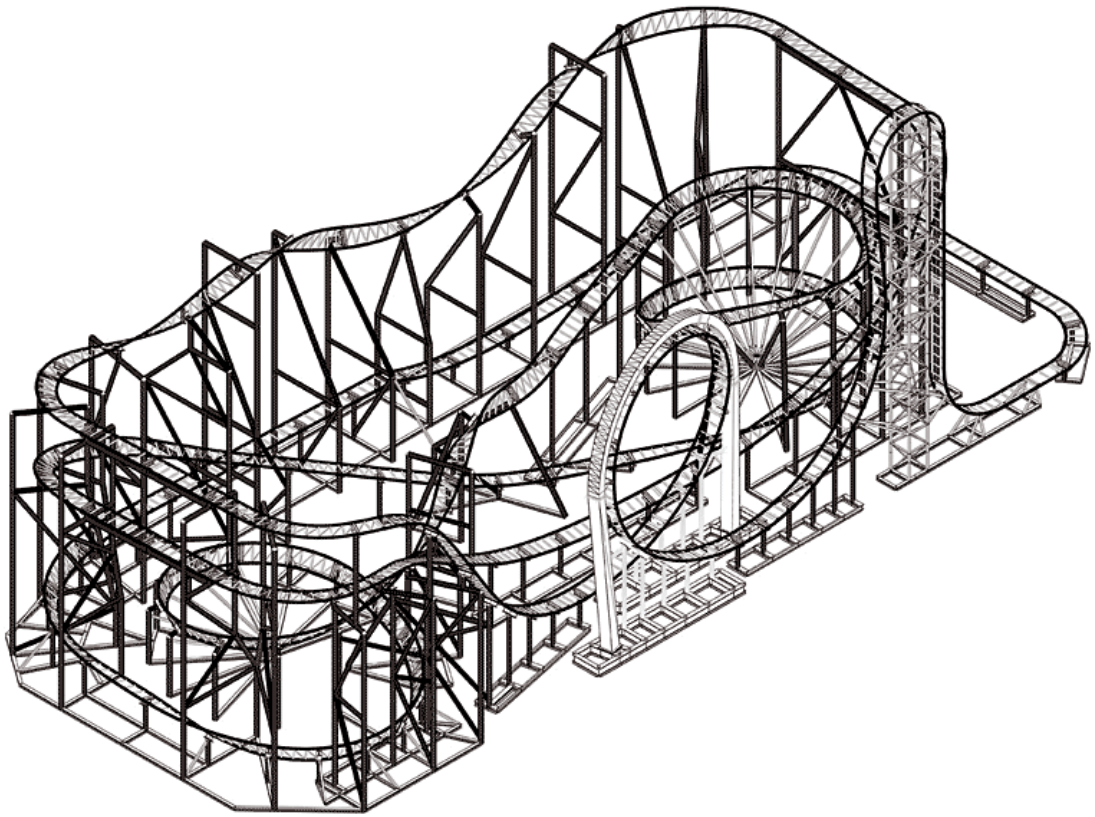


Application of Frenet Frames

- Another application for Frenet frames is analyzing the motion of a car moving along a roller coaster.
- If we assume a motor within the car is able to control its speed at any instant, then knowing the shape of the car's path is enough to specify $\boldsymbol{C}(t)$.
- Now if suitable derivatives of $\boldsymbol{C}(t)$ can be taken, the normal and binormal vectors for the car's motion can be found and a Frenet frame for the car can be constructed for each relevant value of t.

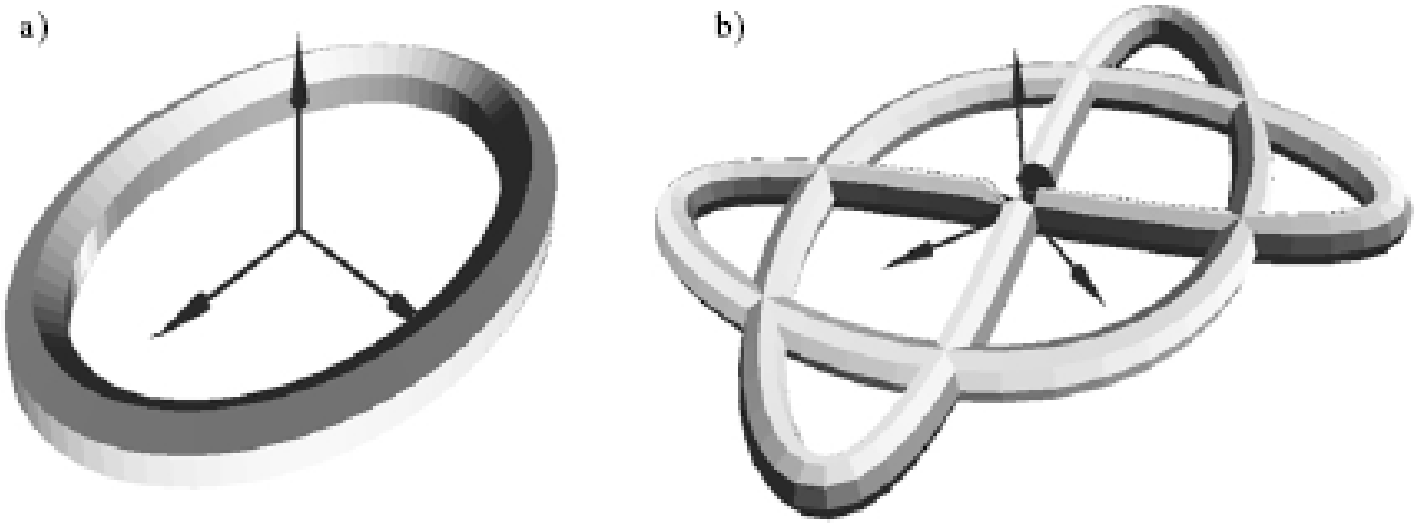
Application of Frenet Frames (2)

- This allows us to find the forces operating on the wheels of each car and the passengers.



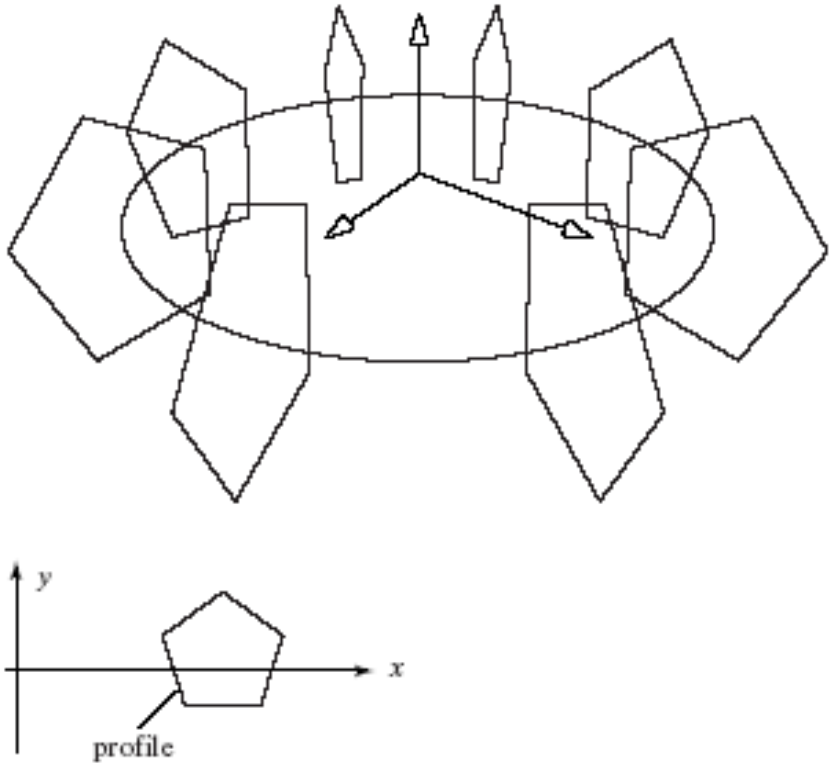
Special Case: Discretely Swept Surfaces of Revolution

- Example: polygon positioned away from y axis and then rotated around y axis along some curve ((a) circle, (b) Lissajous figure).



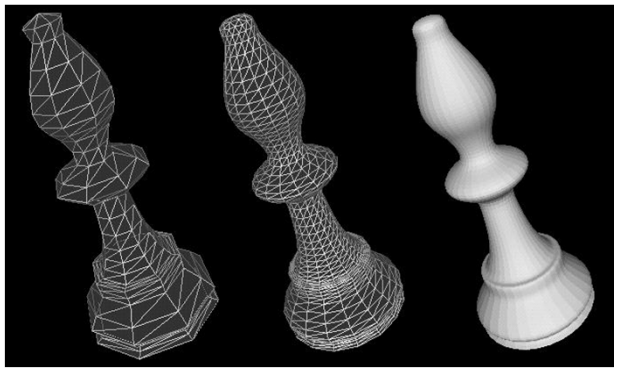
Discretely Swept Surfaces of Revolution (2)

- Example: rotating a polyline around an axis to produce a 3D figure.



Discretely Swept Surfaces of Revolution (3)

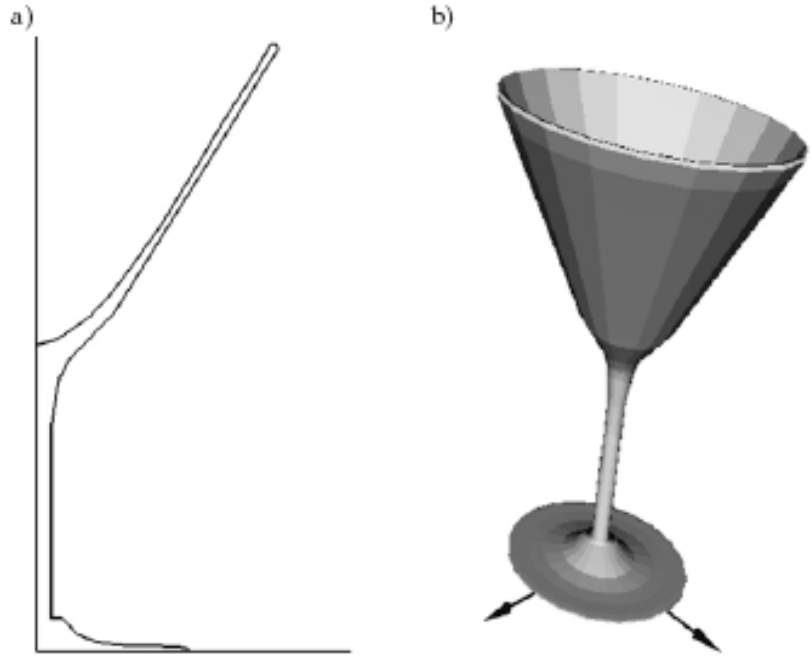
- This is equivalent to **circularly sweeping** a shape about an axis.
- The resulting shape is often called a **surface of revolution**. Below: 3 versions of a pawn based on a mesh that is swept in discrete steps.



Discretely Swept Surfaces of Revolution (3)

- Glass: polyline with $P_j = (x_j, y_j, 0)$.
- To rotate the polyline to K equispaced angles about the y-axis: $\theta_i = 2\pi i/K$, $i = 0, 1, 2, \ldots, K$, and

$$\tilde{M} = \begin{pmatrix} \cos(\vartheta_i) & 0 & \sin(\vartheta_i) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\vartheta_i) & 0 & \cos(\vartheta_i) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Mesh Approximations to Smooth Objects

- Given a smooth surface, tesselate it: approximate it by triangles or quadrilaterals with vertices on the smooth surface, connected by straight lines not on the surface.
- If the mesh is fine enough (the number of faces is large enough), shading can make the surface look smooth.

Mesh Approximations to Smooth Objects (2)

- The faces have vertices that are found by evaluating the surface's parametric representation at discrete points.
- A mesh is created by building a vertex list and face list in the usual way, except now the vertices are computed from formulas.
- The vertex normal vectors are computed by evaluating formulas for the **normal to the smooth surface** at discrete points.

Mesh Approximations to Smooth Objects (3)

- In Ch. 4.5, we used the planar **patch** given parametrically by $P(u, v) = C + \mathbf{a}u + \mathbf{b}v$, where C is a point, **a** and **b** are vectors, and u and v are in [0, 1].
  - This patch is a parallelogram in 3D with corner vertices C, $C + \mathbf{a}$, $C + \mathbf{b}$, and $C + \mathbf{a} + \mathbf{b}$.
- More general surface shapes require three functions $X()$, $Y()$, and $Z()$ so that the surface has parametric representation in point form $P(u, v) = (X(u, v), Y(u, v), Z(u, v))$ with u and v restricted to suitable intervals.

Mesh Approximations to Smooth Objects (4)

- Different surfaces are characterized by different functions: *X*, *Y*, and *Z*.
  - The notion is that the surface is at (*X*(0, 0), *Y*(0, 0), *Z*(0, 0)) when both *u* and *v* are zero, at (*X*(1, 0), *Y*(1, 0), *Z*(1, 0)) when *u* = 1 and *v* = 0, and so on.
- Letting *u* vary while keeping *v* constant generates a curve called a ***v*-contour**. Similarly, letting *v* vary while holding *u* constant produces a ***u*-contour**.

Mesh Approximations to Smooth Objects (2)

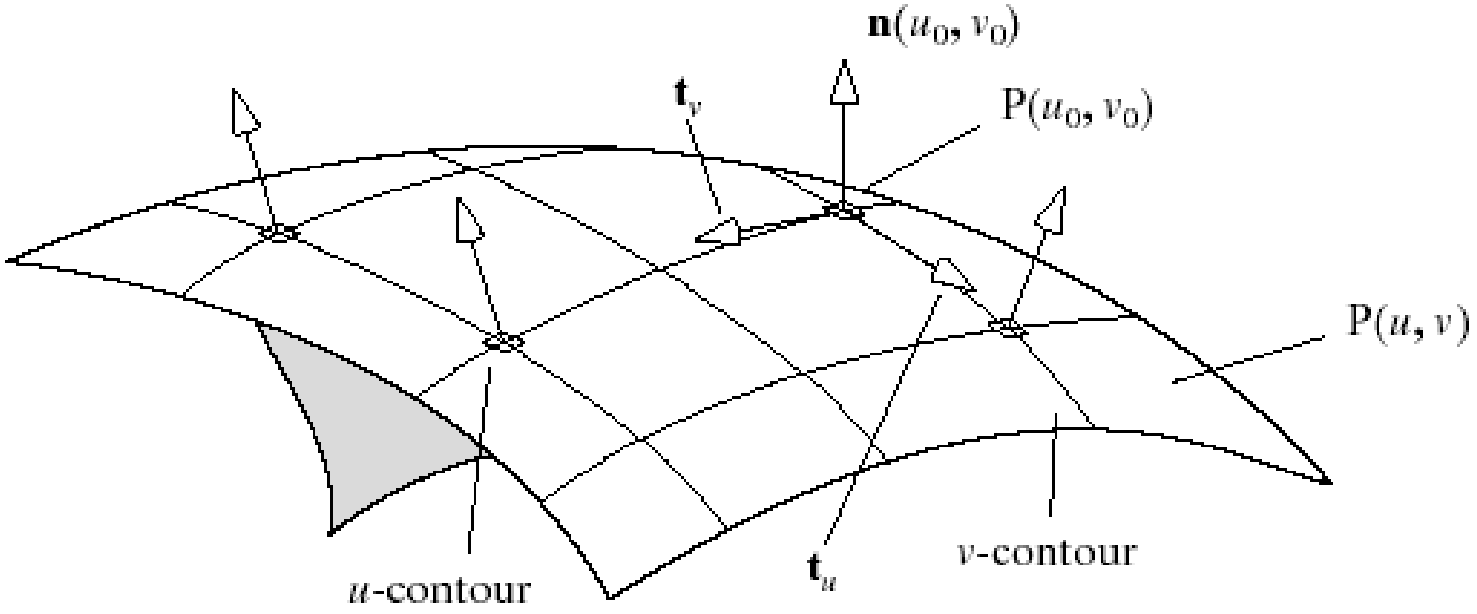
- Method: describe surface as P(u, v) = (X(u, v), Y(u, v), Z(u, v)) (parametric form) and use the implicit form for the equation of the surface: F(x, y, z) = 0.
- Given a point Q, Q is inside the object if F(Q) < 0, on it if F(Q) = 0, and outside it if F(Q) > 0.
- Example: the plane that passes through point *B* and has normal vector **n** is described by the equation $n_x\, x + n_y\, y + n_z\, z = D$ (where $D = \mathbf{n} \bullet B$), so the implicit form for this plane is $F(x, y, z) = n_x\, x + n_y\, y + n_z\, z - D$.

Mesh Approximations to Smooth Objects (3)

- Sometimes it is more convenient to think of *F* as a function of a point *P*, rather than a function of three variables *x*, *y*, and *z*, and we write $F(P) = 0$ to describe all points that lie on the surface.
- For the plane, we define $F(P) = \mathbf{n}\cdot(P - B)$ and say that *P* lies in the plane if and only if $F(P) = \mathbf{n}\cdot(P-B)$ is zero.
- If we wish to work with coordinate frames with P = (x, y, z, $1^{)T}$, the implicit form for a plane is even simpler: $F(P) = \mathbf{n}\cdot P$, where $\mathbf{n} = (nx, ny, nz\ -D)^T$.

Mesh Approximations to Smooth Objects (4)

- The normal to a surface at a point $P(u_0, v_0)$ on the surface is found 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, the region will be essentially flat and will have a well-defined normal direction.



Mesh Approximations to Smooth Objects (5)

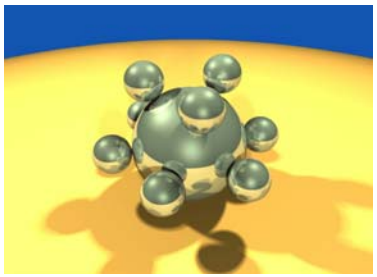
- The normal vector in parametric or gradient form is
  $\mathbf{n}(u_0,v_0) = \left(\frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v} \right) \Big|_{u0,v0}$

  $\mathbf{n}(x_0, y_0, z_0) = \nabla F = \left(\frac{\partial F}{\partial x}, \frac{\partial F}{\partial y}, \frac{\partial F}{\partial z} \right) \Big|_{x0,y0,z0}$

- (Normalize **n**.)
- The result of applying an affine transformation M to the surface P(u,v) or F(x, y, z) is P(u, v) → MP(u, v), F(P) → F(M^{-1}P) and **n**(u,v) → M^{-1}(**n**(u,v)).

Computer Graphics using OpenGL, 3rd Edition

F. S. Hill, Jr. and S. Kelley

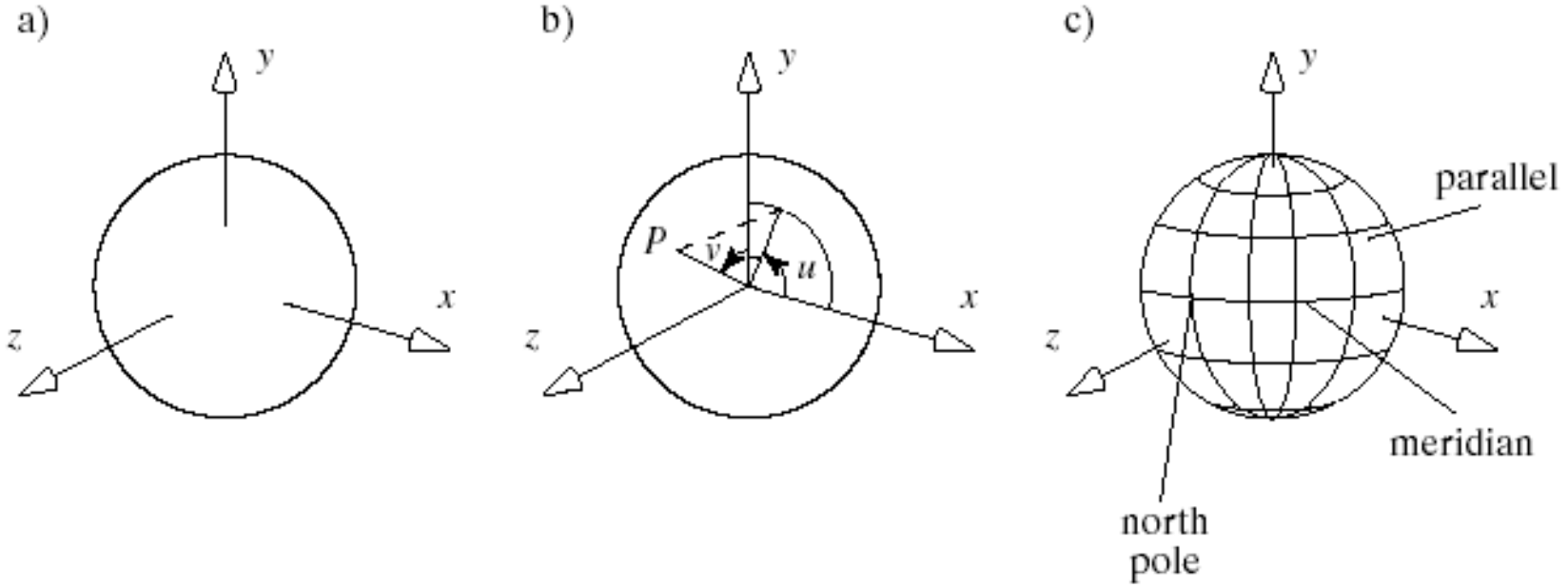


Chapter 6.5-6
Modeling Shapes with Polygonal Meshes

S. M. Lea
University of North Carolina at Greensboro

Generic Sphere

- Center (0, 0, 0), radius 1;
- $F(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$, or $F(P)=|P|^2-1$.
- $P(u, v) = (\cos v \cos u, \cos v \sin u, \sin v)$, with $0 \le v \le 2\pi$, $-\pi/2 \le u \le \pi/2$

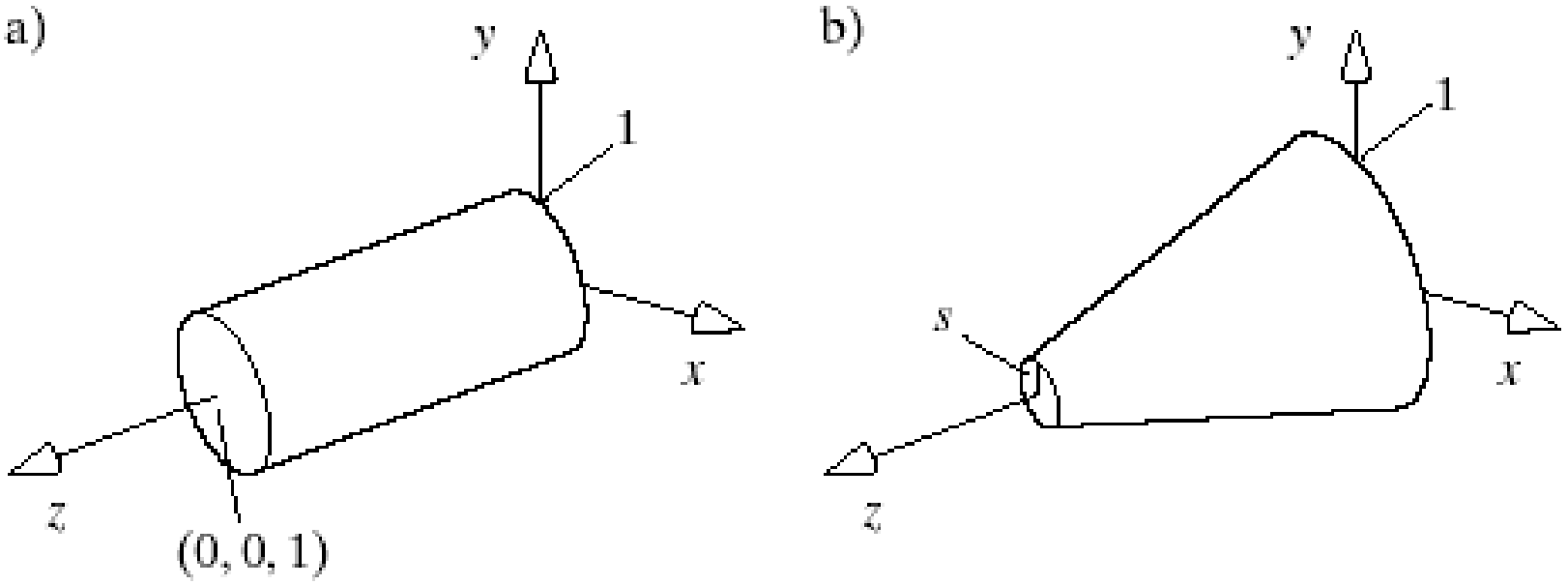


Sphere (2)

- u-contours are longitude lines (meridians), v-contours are latitude lines (parallels).
- The normal vector (gradient) $2(x, y, z)$ is radially outward.
- The parametric form is $\mathbf{n}(u, v) = -cos(v)\mathbf{p}(u, v)$, also radially outward. The scale factor $-cos(v)$ will disappear when we normalize **n.**
- We must use $\mathbf{p}(u,v)$ rather than $-\mathbf{p}(u,v)$ for the normal, so that it does indeed point radially outward.

Generic Tapered Cylinder

- Axis coincides with *z*-axis; circular cross section of radius 1 at base, s when z = 1; extends in *z* from 0 to 1.
- The tapered cylinder with an arbitrary value of *s* provides formulas for the generic cylinder and cone by setting *s* to 1 or 0, respectively.



Generic Tapered Cylinder (2)

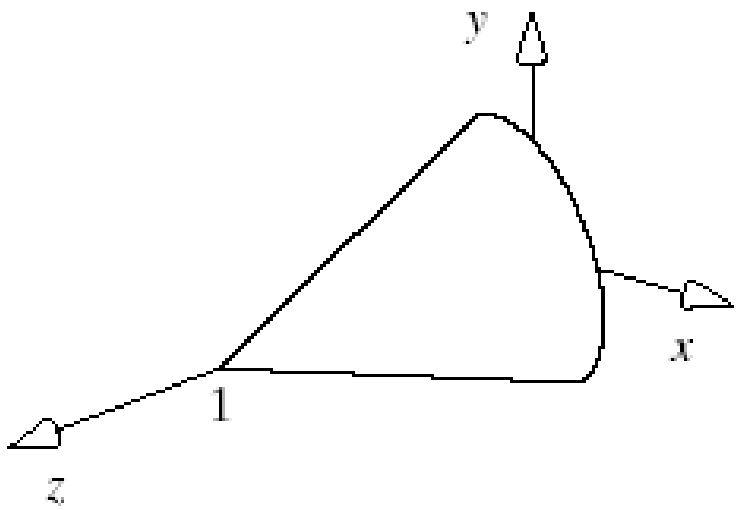
- The wall of the tapered cylinder is given by the implicit form $F(x,y,z) = x^2 + y^2 - (1+(s-1)z)^2$ for 0 < z < 1, and by the parametric form $P(u,v) = ((1+(s-1)v)\cos(u), (1+(s-1)v)\sin(u), v)$
- When the tapered cylinder is a solid object, we add two circular discs at its ends: a **base** and a **cap**. The cap is a circular portion of the plane $z = 1$, characterized by the inequality $x^2 + y^2 < s^2$, or given parametrically by $P(u, v) = (v\cos(u), v\sin(u), 1)$ for v in $[0, s]$.

Generic Tapered Cylinder (3)

- The normal vector to the wall of the tapered cylinder is $\mathbf{n}(x, y, z) = (x, y, -(s - 1)(1+ (s - 1)z))$, or in parametric form $\mathbf{n}(u, v) = (cos(u), sin(u), 1 – s)$.
- For the generic cylinder the normal is simply $(cos(u), sin(u), 0)$.
- The normal is directed radially away from the axis of the cylinder. For the tapered cylinder it is also directed radially, but shifted by a constant z-component.

Generic Cone

- A cone whose axis coincides with the *z*-axis, has a circular cross section of maximum radius 1, and extends in *z* from 0 to 1. It is a tapered cylinder with small radius of $s = 0$.



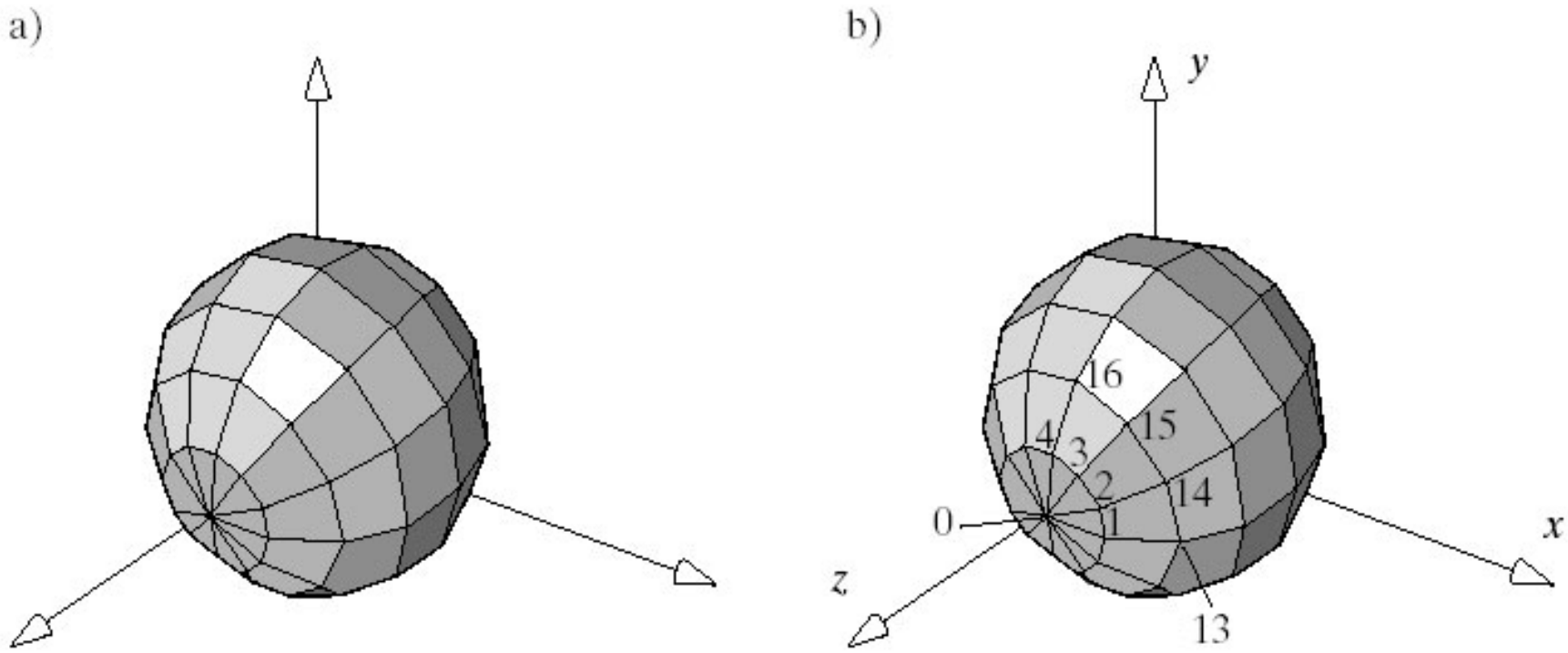
Generic Cone (2)

- Wall: $F(x, y, z) = x^2 + y^2 - (1 - z)^2 = 0$ for $0 < z < 1$; parametric form $P(u, v) = ((1-v)\,cos(u), (1- v)\,sin(u), v)$ for azimuth u in $[0, 2\pi]$ and v in $[0, 1]$.
- Using the results for the tapered cylinder again, the normal vector to the wall of the cone is $(x, y, 1-z)$.
- Fig. 6.54 shows normal vectors for all generic surfaces.

Mesh for the Generic Sphere

- We slice the sphere along azimuth lines and latitude lines.
- We slice the sphere into nSlices slices around the equator and nStacks stacks from the South Pole to the North Pole.
- The figure (next slide) shows the example of 10 slices and 8 stacks.
- The larger nSlices and nStacks are, the better the mesh approximates a true sphere.

Mesh for the Generic Sphere (2)



Mesh for the Generic Sphere (3)

- To make slices we need nSlices values of *u* around the equator between 0 and 2π. Usually these are chosen to be equispaced: u_i = 2πi/nSlices, *i* = 0, 1, ..., nSlices -1.
- We put half of the stacks above the equator and half below. The top and bottom stacks will consist of triangles; all other faces will be quadrilaterals.
- This requires we define (nStacks + 1) values of latitude: v_j = π - π j/nStacks, j = 0, 1, ..., nStacks.

Mesh for the Generic Sphere (4)

- The vertex list: put the north pole in pt[0], the bottom points of the top stack into the next 12 vertices, etc. There will be 98 points.
- The normal vector list: norm[k] is the normal for the sphere at vertex pt[k] in parametric form; $\mathbf{n}(u,v)$ is evaluated at (u,v) used for the points.
  - For the sphere this is particularly easy since norm[k] is the same as pt[k].

Mesh for the Generic Sphere (5)

- The face list: Put the top triangles in the first 12 faces, the 12 quadrilaterals of the next stack down in the next 12 faces, etc.
- The first few entries in the face list will contain the data

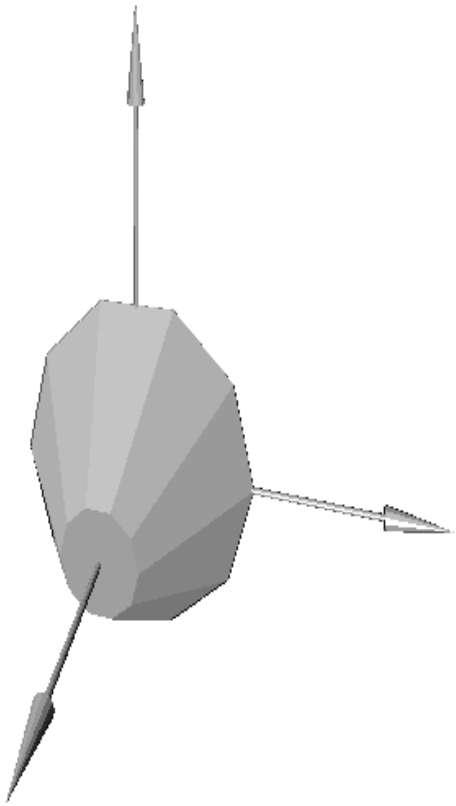
number of vertices:	3	3	3 ...
vertex indices:	0 1 2	0 2 3	0 3 4 ...
normal indices:	0 1 2	0 2 3	0 3 4 ...

General Meshes

- Ultimately we need a method, such as makeSurfaceMesh(), that generates appropriate meshes for a given surface $P(u, v)$.
- Some graphics packages have routines that are highly optimized for triangles, making triangular meshes preferable to quadrilateral ones.
- We can use the same vertices, but alter the face list by replacing each quadrilateral with two triangles.
  - For instance, a face that uses vertices 2, 3, 15, 14 might be subdivided into two triangles, one using 2, 3, 15 and the other using 2, 15, 14.

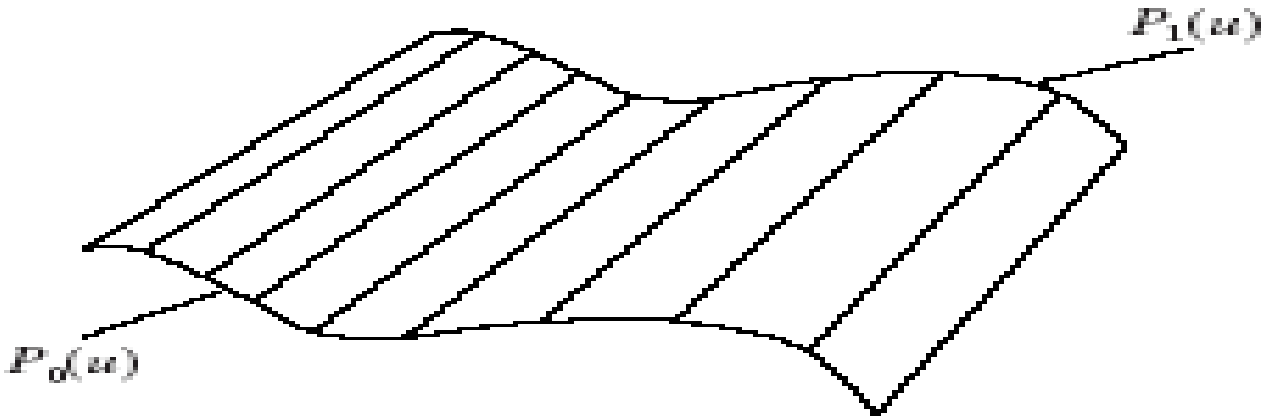
Mesh for the Tapered Cylinder

- We use nSlices = 10 and nStacks = 1.
- A decagon is used for the cap and base.
- If you prefer to use only triangles, the walls, the cap, and the base could be dissected into triangles.



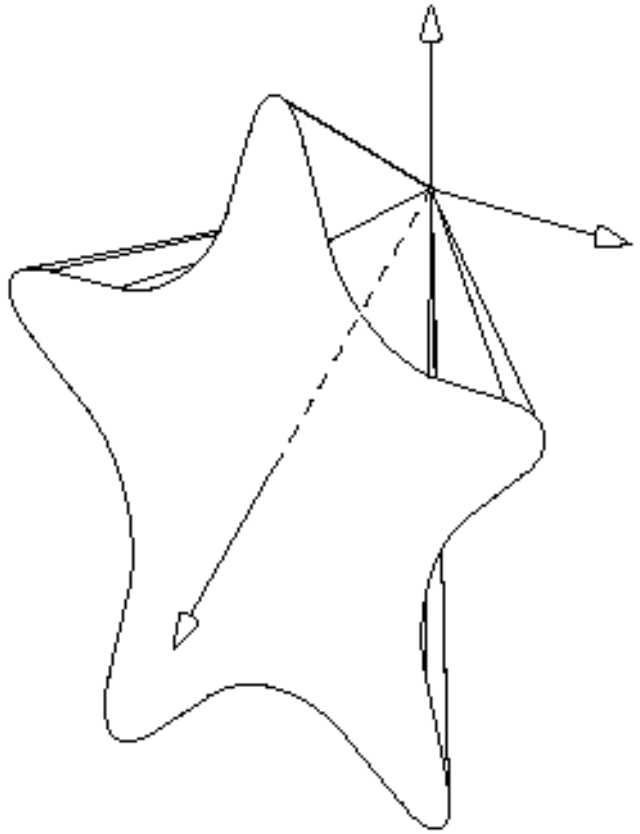
Ruled Surfaces

- Ruled Surface: through every point, there passes at least one straight line lying entirely on the surface.
- Made by moving the ends of a straight line along curves.



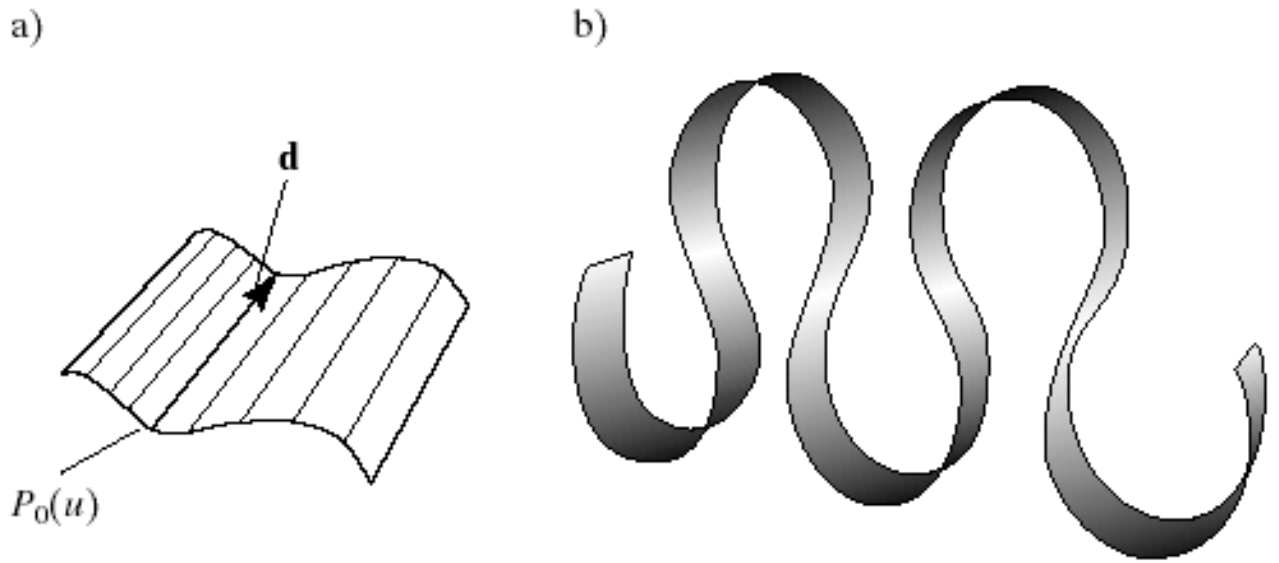
Ruled Surfaces (2)

- A cone is a ruled surface for which one of the curves, say, $P_0(u)$, is a *single* point $P_0(u) = P_0$, the apex of the cone,.
- $P(u, v) = (1 - v) P_0 + v P_1(u)$ {a general cone}



Ruled Surfaces (3)

- A cylinder is a ruled surface for which $P_1(u)$ is a translated version of $P_0(u)$: $P_1(u) = P_0(u) + \mathbf{d}$, for some vector $\mathbf{d}$.

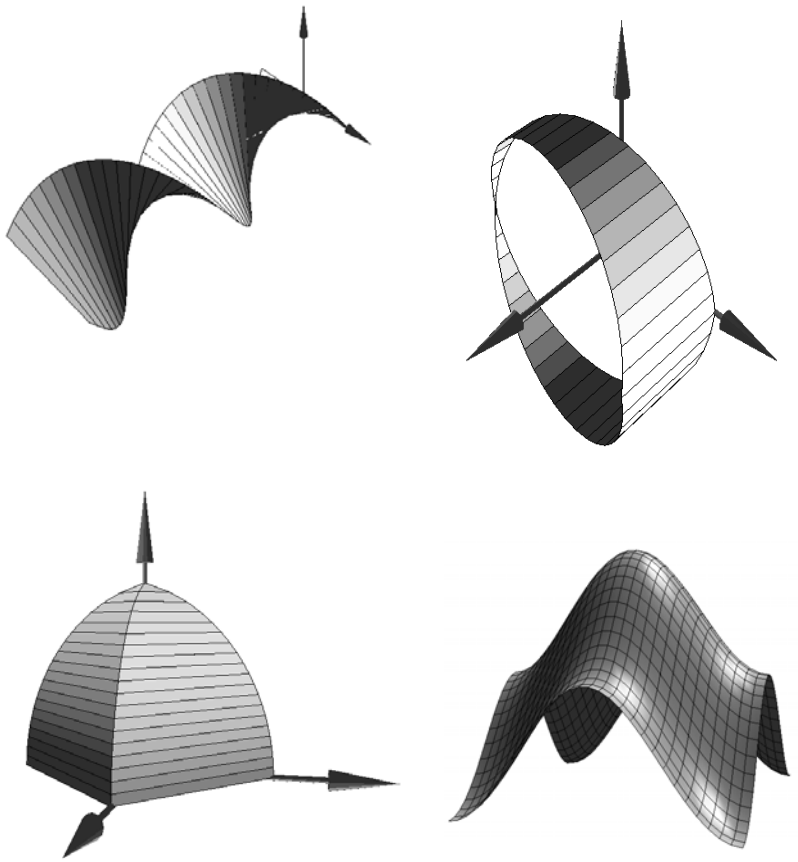


Ruled Surfaces (4)

- The general cylinder has the parametric form $P(u, v) = P_0(u) + \mathbf{d}v$.
- To be a true cylinder, the curve $P_0(u)$ is confined to lie in a plane.
  - If $P_0(u)$ is a circle the cylinder is a **circular cylinder**.
  - The direction **d** need not be perpendicular to this plane, but if it is, the surface is called a **right cylinder**.

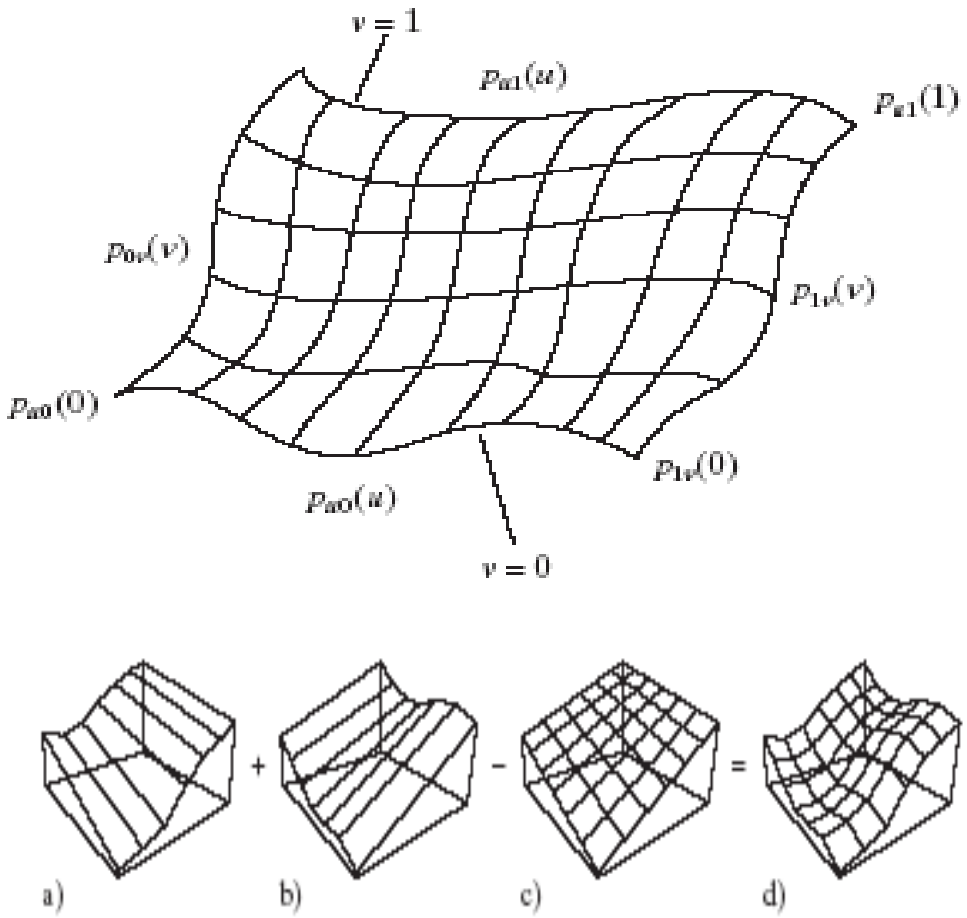
Ruled Surfaces (5)

- Double helix: $P_0(u)$ and $P_1(u)$ are both helixes that wind around each other.
- Möbius strip (has only one edge).
- Vaulted roof made up of four ruled surfaces.
- Coons patch named after the legendary graphicist Steven Coons.



Coons Patches

- Interpolates 4 boundary curves.
- $P(u, v) = [p_{0v}(v)(1-u) + p_{1v}(v)u] + [p_{u0}(u)(1-v) + p_{u1}(u)v] - [(1-u)(1-v)p_{0v}(0) + u(1-v)p_{1v}(0) + v(1-u)p_{0v}(1) + uv\ p_{1v}(1)]$



Surfaces of Revolution

- Produced by rotational sweep of profile curve C around an axis.
- Curve C(v) = (X(v), Z(v)) is revolved, generally around the z axis.
- u is the angle of rotation, and v determines the shape of the curve.
- When point (X(v), 0, Z(v)) is rotated by angle u, it becomes ((X(v)cos(u), X(v)sin(u), Z(v)).
- P(u, v) = (X(v)cos(u), X(v)sin(u), Z(v))

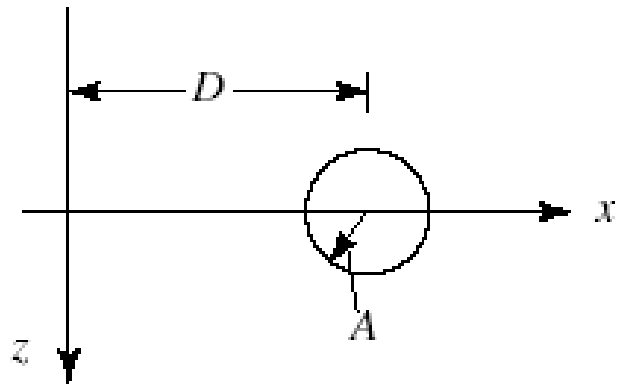
Surfaces of Revolution (2)

- The different positions of the curve *C* around the axis are called **meridians**.
- Sweeping C completely around generates a full circle, so contours of constant *v* are circles, called **parallels**.
- The normal vector is **n** (u, v) = X(v) [Ż(v)cos(u), Ż(v)sin(u), -X(v)].

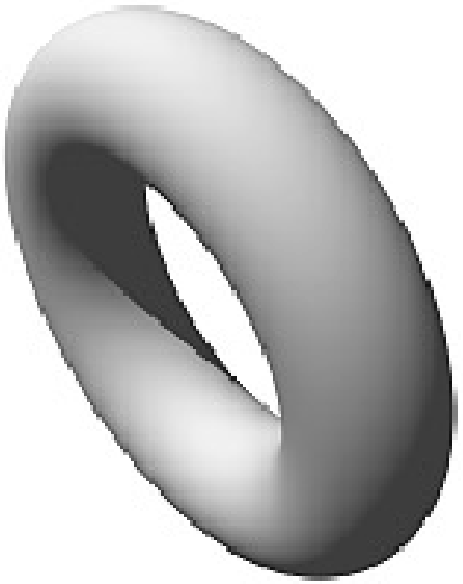
Example

- The **torus** is generated by sweeping a circle displaced by a distance D along the x-axis about the *z*-axis. The circle has radius *A*, so its profile is C(v) = (*D* + *A cos*(*v*), *A sin*(*v*)). The torus has representation *P*(*u*, *v*) = ((*D* + *A cos*(*v*)) *cos*(*u*), (*D* + *A cos*(*v*)) *sin*(*u*), *A sin*(*v*))

a)



b)



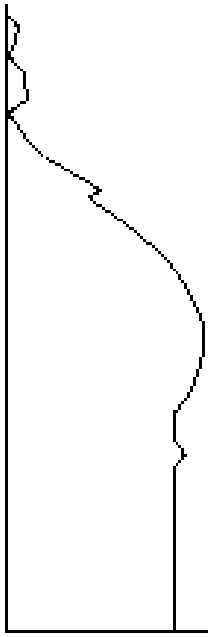
Surfaces of Revolution (3)

- A mesh for a surface of revolution is built in a program in the usual way.
- We choose a set of u and v values, $\{u_i\}$ and $\{v_j\}$, and compute a vertex at each from $P(u_i, v_j)$, and a normal direction from $\mathbf{n}(u_i, v_j)$. Polygonal faces are built by joining four adjacent vertices with straight lines.

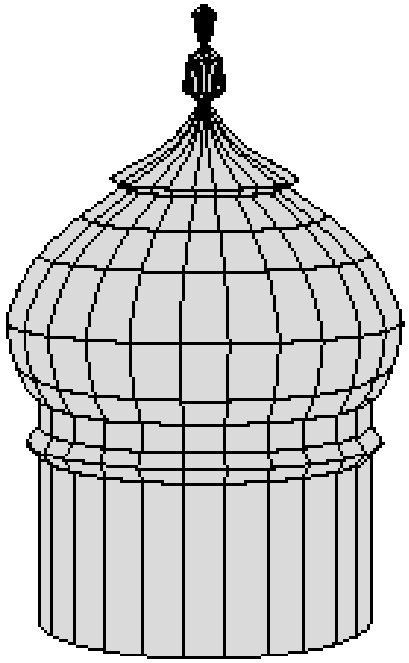
Example

- A model of the dome of the Taj Mahal in Agra, India.

a)



b)



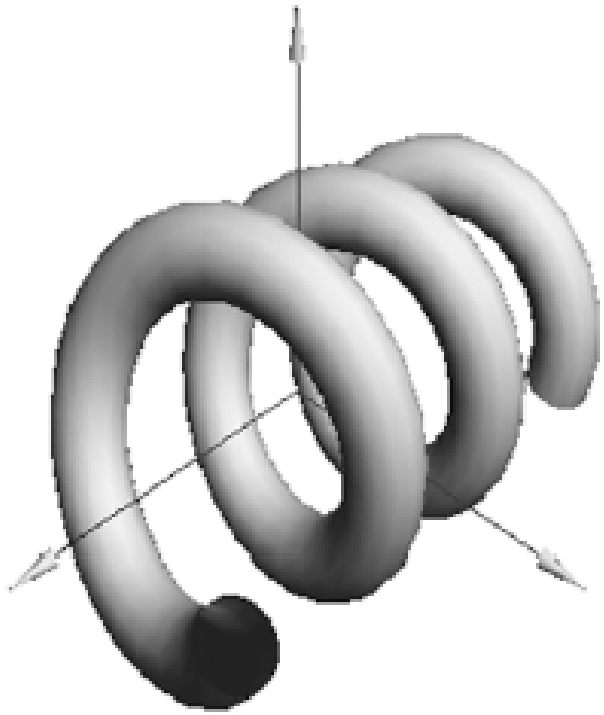
c)

Tubes Based on 3D Curves

- We discussed tubes based on a “spine” curve $C(t)$ meandering through 3D space in 6.4.3.
- A polygon was placed at each of a selection of spine points and oriented according to the Frenet frame computed there.
- Then corresponding points on adjacent polygons were connected to form a flat-faced tube along the spine.
- Here we do the same thing, except we compute the normal to the surface at each vertex so that smooth shading can be performed.

Tubes based on 3-d Curves

- C(t) is a curve in space that forms the spine of a polygon translated along the curve.
- Circle (cos(u), sin (u), 0) moves along helix C.

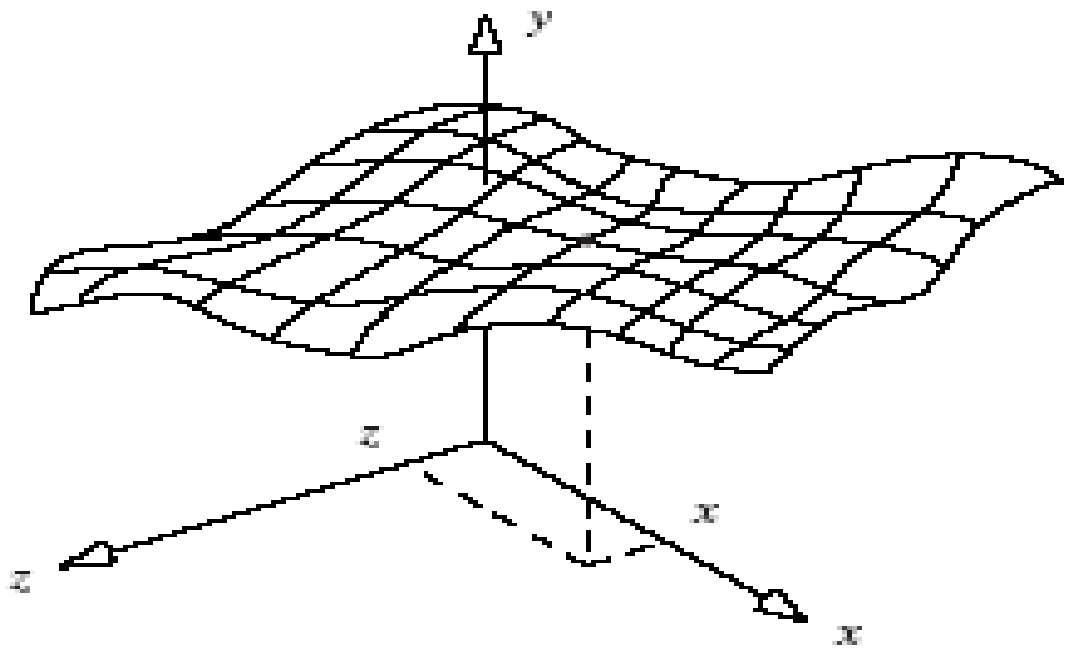


Tubes based on 3-d Curves (2)

- The parametric equation is P(u, v) = C(v) + **N**(v)cos(u) + **B**(v)sin(u), where **N** and **B** are the Frenet frame vectors.
- Then we can build a mesh by sampling P(u, v) and building vertex, normal and face lists.

Surfaces which are Functions of Two Variables

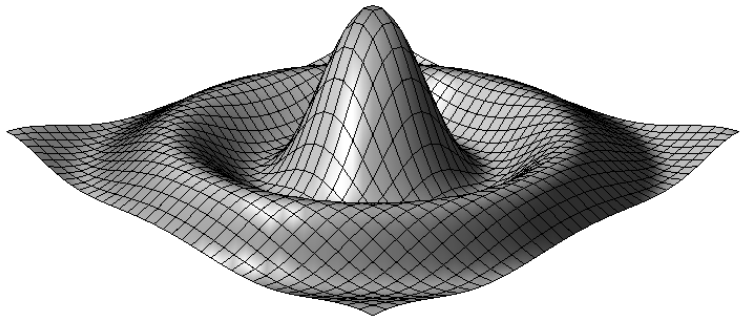
- Define a single-valued height function y = f(x, z) of any sort you wish.



Surfaces which are Functions of Two Variables (2)

- Parametric form: P(u, v) = (u, f(u, v), v)
- Normal vector **n**(u, v) = $-\left(\frac{\partial f}{\partial u}, -1, \frac{\partial f}{\partial v}\right)$

- Thus *u*-contours lie in planes of constant *x*, and *v*-contours lie in planes of constant *z*.

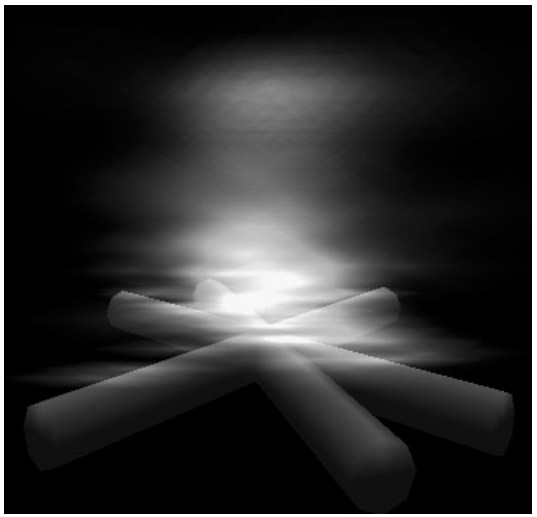


Particle Systems

- A particle system can keep track of an enormous number of particles, each with a position, a velocity, and perhaps a color, lifetime, size, degree of transparency, and shape.
- Any of these attributes might be randomly chosen by a random number generator, depending on the needs of the application.

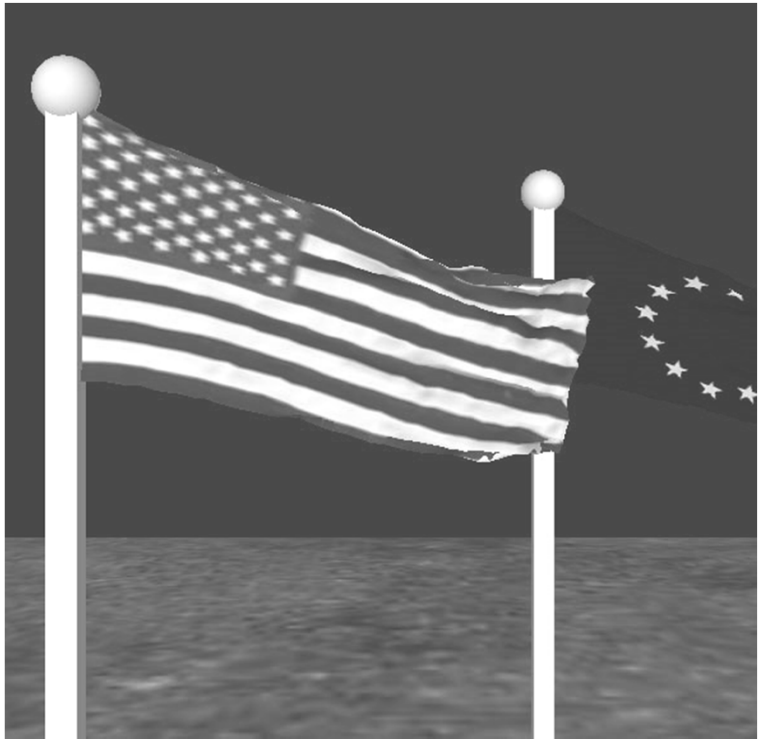
Example

- An animation that simulates fire using a particle system.
- The central issue in particle systems is that each particle must be modeled with its own set of parameters, and as time moves on, the position and velocity of the particle must be tracked correctly.
- Particle systems require a large amount of memory to store this information and a long time to update it all.



Physically Based Systems

- In physically based modeling, we describe in mathematical terms how the various forces in a system of objects interact to control the motion of these objects.
- Example: flags in the wind.



Physically Based Systems (2)

- One of the key ingredients is that different objects collide with one another and are possibly deformed in the process.
- Describing such systems leads to very complex mathematics (ordinary differential equations, partial differential equations, etc.).
- These often must be solved numerically which tends to make the associated algorithms rather slow.

Physically Based Systems (3)

- In computer games, animation allows the modeling and viewing of terrain. The observer can fly over simulated terrain.
- A number of such animations are located on the book's accompanying web site.