Computer Graphics

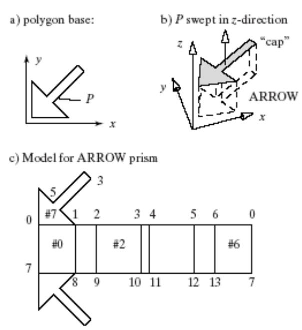
Modeling Shapes with
Polygonal Meshes
Chapter 6.4-5

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.

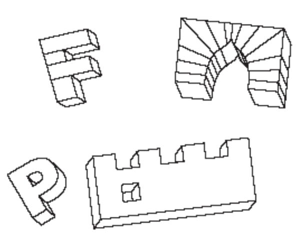
Face List for the Prism

- We first make the side faces and then add the cap and base.
- For the j-th wall (j = 0,...,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.

```
if (j < n-1)
    next = ++j;
else
    next = 0;
```


Arrays of Extruded Prisms

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

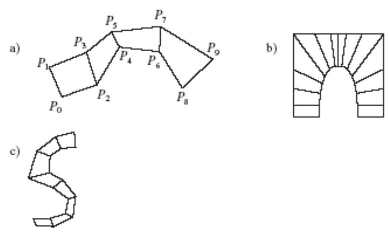


Drawing Arrays of Extruded Prisms

- 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

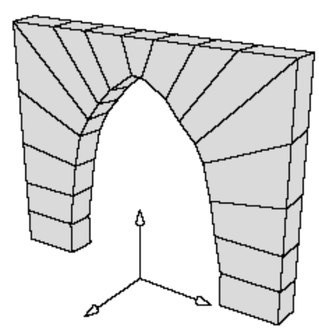
- quad-strip = {p_0, p_1, p_2, …., p_{M-1}}
- The vertices are understood to be taken in pairs, with the *odd* ones forming one edge of the quad-strip, and the *even* ones forming the other edge.
- Not every polygon can be represented as a quad-strip.

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

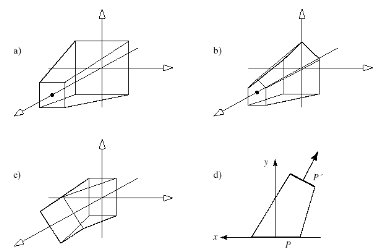


Special Case: Twisted Extrusions

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

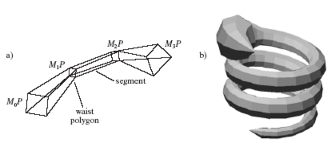
Examples

- 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.

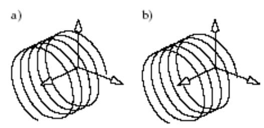


Special Case: Segmented Extrusions

- 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.

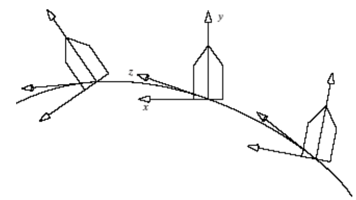
Methods for Twisted Extrusions

- 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:

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

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); and after normalization:

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.

•The normalized binormal vector to the curve:

B = **T**x**A**=(bsin(t), -bcos(t), 1).

•The normal 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**.

After normalization, N is the same as A

Method for Twisted Extrusions (extra)

Example: **C**(t) = (t, t^2, t^3)

•The tangent vector to the curve:

T = derivative of **C**(t):

T = (1, 2t, $3t^2$)

•The acceleration vector is the derivative of the tangent vector, and thus the second derivative of the curve **C**:

A = (0, 2, 6t)

• The binormal vector to the curve:

B = **T**x**A**=($6t^2$, -6t, 2).

•The normal vector

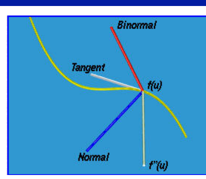
N = **B**x**T** =(4t+$18t^3$, -2+$18t^4$, -6t-$12t^3$)

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)

- Frenet Frame:
  - Tangent vector T
    - (Accelarator vector A (f' ' (u)))
  - Binomal vector
  - Normal vector



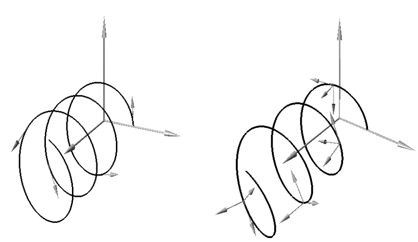
- If the curve is awkward numerically, the derivatives for the reference frame vectors may be approximated over a small distance ε by

T(t_i) = (**C**(t+ε) - **C**(t-ε))/(2 ε),

B(t_i) = (**C**(t+ε) - 2**C**(t) + **C**(t-ε))/$ε^2$.

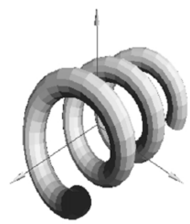
Examples

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.



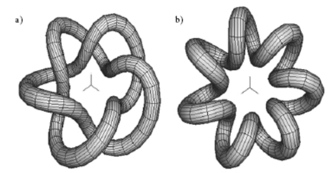
Examples (2)

- Toroidal spiral:

C(t) = (a + b cos(qt) cos(pt), (a + b cos(qt)) sin(pt), c sin(qt))

- What is the Frenet frame for this curve?

p = 2, q = 5

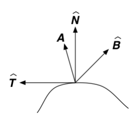


p = 1, q = 7

Frenet Frame for Toroidal spiral

Tangent:

$$\begin{aligned}\dot{x}(t) &= -p(a + b\cos qt)\sin pt - bq\sin qt\cos pt \\ &= -p\ y(t) - bq\sin qt\cos pt, \\ \dot{y}(t) &= p(a + b\cos qt)\cos pt - bq\sin qt\sin pt \\ &= p\ x(t) - bq\sin qt\sin pt, \\ \dot{z}(t) &= bq\cos qt.\end{aligned}$$



Acceleration:

$$\begin{aligned}\ddot{x}(t) &= -p\ \dot{y}(t) + bq(p\sin qt\sin pt - q\cos qt\cos pt), \\ \ddot{y}(t) &= p\ \dot{x}(t) - bq(p\sin qt\cos pt + q\cos qt\sin pt), \\ \ddot{z}(t) &= -q^2 b\sin qt.\end{aligned}$$

$$T \bullet A \neq 0$$

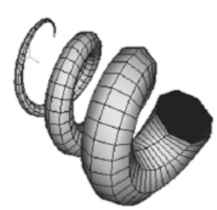
B=T x A

N=T x B

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.

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

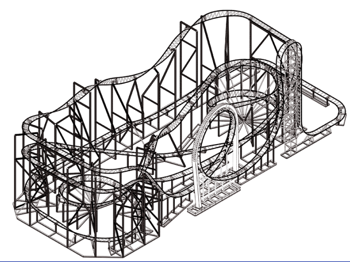


Application of Frenet Frames

- 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 ***C***(t).
- Now if suitable derivatives of ***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.

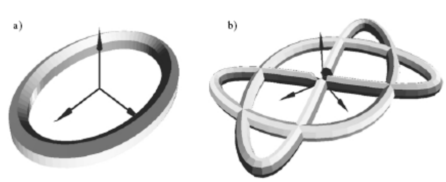
Application of Frenet Frames (2)

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



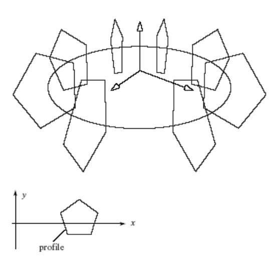
Special Case: Discretely Swept Surfaces of Revolution

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



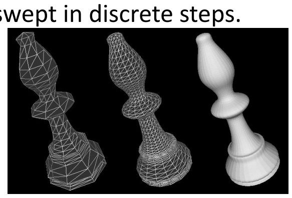
Discretely Swept Surfaces of Revolution (2)

- 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.

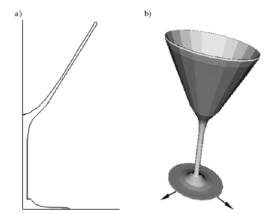


Discretely Swept Surfaces of Revolution (3)

- Glass: polyline with P_j = (x_j, y_j, 0).
- To rotate the polyline to *K* equal-spaced angles about the *y*-axis:

θ_i = 2π *i/K, i = 0, 1, 2, …, K, and

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



Surfaces of Revolution

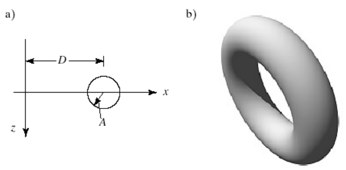
- Produced by rotational sweep of profile curve C around an axis.
- Curve C(v) = (X(v), Z(v)) is revolved, generally around the z axis.
- u is the angle of rotation, and v determines the shape of the curve.
- When point (X(v), 0, Z(v)) is rotated by angle u, it becomes ((X(v)cos(u), X(v)sin(u), Z(v)).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



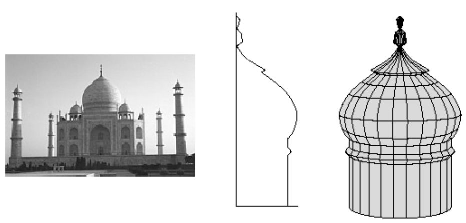
V: angle of the points on the circle having radius=A
U: angle, as it sweeps around Z

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.



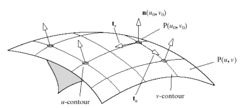
a) b) c)

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)

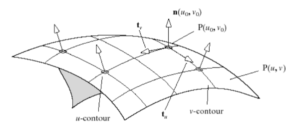
- In Ch. 4.5, we used the planar **patch** given parametrically by *P* (*u*, *v*) = *C* + **a***u* + **b***v,* where *C* is a point, **a** and **b** are vectors, and *u* and *v* are in [0, 1].
  - This patch is a parallelogram in 3D with corner vertices *C*, *C* + **a**, *C* + **b**, and *C* + **a** + **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.



For example:
P(u, v) = (cos v cos u, cos v sin u, sin v),
with 0 ≤u ≤ 2π, -π/2 ≤ v ≤ π/2

Mesh Approximations to Smooth Objects (4)

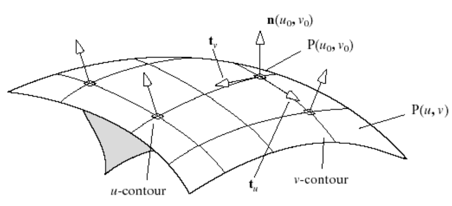
- Different surfaces are characterized by different functions: *X*, *Y*, and *Z*.
  - The notion is that the surface is at (*X*(0, 0), *Y*(0, 0), *Z*(0, 0)) when both *u* and *v* are zero, at (*X*(1, 0), *Y*(1, 0), *Z*(1, 0)) when *u* = 1 and *v* = 0, and so on.
- Letting *u* vary while keeping *v* constant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



For example:
P(u, v) = (cos v cos u, cos v sin u, sin v),
with 0 ≤u ≤ 2π, -π/2 ≤ v ≤ π/2

Mesh Approximations to Smooth Objects (3)

- 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 (4)

- The normal vector in parametric or gradient form is

n (u_0, v_0) =

$$\left(\frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v}\right)\Bigg|_{u_0, v_0}$$

n (x_0, y_0, z_0) =

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

- Normalize **n**

For example: For sphere,
P(u, v) = (cos v cos u, cos v sin u, sin v),
with 0 ≤u ≤ 2π, -π/2 ≤ v ≤ π/2
$F(x, y, z) = x^2 + y^2 + z^2 - 1 = 0$
The normal vector to (*x*, *y*, *z*) is radially outward.