Polygonal Models
- Object surfaces approximated by a mesh of planar polygons
  Scene -->
  Objects -->
  Subobjects -->
  Polygons -->
  Vertices (points)

Data structures
- Polygons represent/approximate object surfaces
- In either case we must store 3-D world coordinates of each vertex
  - Use an array of 3-D points:
    • struct point3d (float x; float y; float z)
      // a single 3-D point
    • struct point3d w_pts[]; // w_pts is the 3-D points array

3-D Modeling with Polygons
- Polygon Mesh
  - Store the polygon faces:
    • Array of vertex lists
    • One list for each polygon

Storing Polygons in a Polygon Mesh Model
- Object: Can be represented as an array of polygons
- Each polygon consists of:
  - (a) the number of vertices in the polygon
  - (b) a list of indices into the 3-D points array
    • (An index gives the position of a vertex in the 3-D points array)
struct polygon {int n; int *inds};
// n: The number of vertices
// inds: A list of indices into
// the points array.
// Specifies which vertices form
// the polygon
struct polygon object[ ];
// The object being modeled
// An array of polygons

Vertex Coordinates

vertex xw yw zw
----------------------
0 0 0 0
1 150 0 0
2 150 150 0
3 0 150 0
4 75 75 150

The Pyramid's Points Array

struct point3d w_pts[5];
// Pyramid vertices in world coords.
int b=150, h=75 ;  // Dimensions of pyramid

// Set up world coordinate points array
w_pts[0].x=w_pts[0].y=w_pts[0].z=0;
w_pts[1].x=b; w_pts[1].y=w_pts[1].z=0;
w_pts[2].x=b; w_pts[2].y=w_pts[2].z=0;
w_pts[3].x=b; w_pts[3].y=b;
w_pts[4].x=w_pts[4].y=b; w_pts[4].z=h;

Example--A Pyramid

Following pyramid has 5 vertices, 8 edges and 5 polygon faces

The Pyramids Polynons Array (Mesh)

<table>
<thead>
<tr>
<th>polygon</th>
<th># vertices</th>
<th>vertices</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>3</td>
<td>0, 1, 4</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1, 2, 4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2, 3, 4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0, 4, 3</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0, 3, 2, 1</td>
</tr>
</tbody>
</table>

Polygon array could be generated by:
struct polygon *object;
// Allocate Space:
for (i=0;i<4;i++)
{
    object[i].n=3; object[i].inds = (int *) calloc(3,sizeof(int));
}
object[4].n=4; object[4].inds = (int *) calloc(4,sizeof(int));
// Define the polygons in the object
// define the side triangles
object[0].inds[0]=0; object[0].inds[1]=1; object[0].inds[2]=4;
// define the square base
More Complex 3-D Objects

- Approximate surfaces with polygons
- Often points, edges, and/or polygons arrays can be generated procedurally

Example 1: A Cone
- Approximate with $n$ triangular sides
- $n+1$ vertices (apex + $n$ in the base)
- And a Base polygon with $n$ sides (example, $n=12$)

Cone Points Array

- Base points:
  $x = R \cos \left(i \cdot \frac{360}{n}\right)$
  $y = R \sin \left(i \cdot \frac{360}{n}\right)$
  $z = 0$
- Apex point:
  $x = y = 0$
  $z = h$ (height of cone)

Cone Polygons Array

- $\text{poly}[0] = \{12, \{12,11,10,9,8,7,6,5,4,3,2,1\}\}$
- $\text{poly}[1] = \{3, \{1,2,0\}\}$
- $\text{poly}[2] = \{3, \{2,3,0\}\}$
- $\text{poly}[3] = \{3, \{3,4,0\}\}$
- $\text{poly}[4] = \{3, \{4,5,0\}\}$
- $\ldots$
- $\text{poly}[12] = \{3,\{12,1,0\}\}$
- The triangles can be generated in a loop

Example 2: A Sphere

- Divide with $n$ lines of latitude and $m$ lines of longitude
- Gives triangles and quadrilaterals
- Latitude/Longitude intersection points used as approximating-polygon vertices
- Number of vertices = $m \cdot n + 2$
- Number of polygons = $(n+1) \cdot m$
- Example $n=3, m=8$
Example: $n=3$, $m=3$

$8 \times 3 + 2 = 26$ vertices

Can get $x$, $y$, $z$ from spherical coordinates

Loop $j$: $0 \rightarrow n-1$ (latitudes), $i$: $0 \rightarrow m-1$ (longitudes)

$$x = R \times \sin(i\times \theta) \times \cos(j\times \phi)$$

$$y = R \times \sin(i\times \theta) \times \sin(j\times \phi)$$

$$z = R \times \cos(j\times \phi)$$

$(3+1) \times 8 = 32$ polygons

Number them in a consistent way

poly[0] = {4, {1,2,10,9}}  
Upper Hemisphere
poly[1] = {4, {2,3,11,10}}  
etc.

poly[8] = {3, {0,9,10}}

poly[9] = {3, {0,10,11}}  
etc.

These can be generated in a loop.

---

**3-D Surfaces**

**Explicit Representation**

$z = f(x,y)$

**Plotting**

- Fix values of $y$ and vary $x$
- Gives a family of curves
  
  $z_0 = f(x,0)$
  $z_1 = f(x,1)$
  $z_2 = f(x,2)$
  $z_3 = f(x,3)$
  etc.

---

**Plotting 3D Surfaces, continued**

- Then fix values of $x$ and vary $y$
- Gives another family of curves
  
  $z_0' = f(0,y)$
  $z_1' = f(1,y)$
  $z_2' = f(2,y)$
  $z_3' = f(3,y)$
  etc.

---

**Plotting 3D Surfaces, continued**

- Result is a wireframe that represents the surface
- Could be broken up into polygons

---

**Parametric Representation of 3D Surfaces**

- Need two parameters, say $t$ and $s$
  
  $x = x(t,s)$, $y = y(t,s)$, $z = z(t,s)$
- both $t$ and $s$ vary over a range (0 to 1)

- To plot:
  - Fix values of $s$ and for each vary $t$ over range
    - gives one family of isoparametric curves
  - Fix values of $t$ and for each vary $s$ over range
    - gives another family of isoparametric curves
Bicubic Bezier Surface Patches

- Define 4-vectors S and T:
  \[
  S = \begin{bmatrix} s^3 & s^2 & s & 1 \end{bmatrix}, \quad 0 \leq s \leq 1
  \]
  \[
  T = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}, \quad 0 \leq t \leq 1
  \]
- Define points on surface patch \( Q(s,t) \) \([ \rightarrow x,y,z \)] as:
  \[
  Q(s,t) = S \cdot M_b \cdot T
  \]
  where \( M_b \) is the Bezier Geometry Matrix we’ve seen before.

A Bicubic Bezier Surface Patch

- Compute and store \( x(s,t), y(s,t), z(s,t) \)
- Project to screen and store \( x(s,t), y(s,t) \)
- For \((s=0; s+=1; s=s)\)
  - Compute & store \( x(s,0), y(s,0), z(s,0) \)
  - Project to screen and store \( x(s,0), y(s,0) \)
  - MoveTo \( x(s,0), y(s,0) \)
  - For \((t=0; t+=1; t=t)\)
    - Compute & store \( x(s,t), y(s,t), z(s,t) \)
    - Project to screen and store \( x(s,t), y(s,t) \)
    - LineTo \( x(s,t), y(s,t) \)

Expanding and Rearranging Terms -- \( x(s,t) \) Equation

- Expand and rearrange terms:
  \[
  x(s,t) = x_0 (1-t)^3 + 3x_0 (1-t)^2 t + 3x_0 (1-t) t^2 + x_0 t^3
  \]
  \[
  + y_0 (1-t)^3 + 3y_0 (1-t)^2 t + 3y_0 (1-t) t^2 + y_0 t^3
  \]
  \[
  + z_0 (1-t)^3 + 3z_0 (1-t)^2 t + 3z_0 (1-t) t^2 + z_0 t^3
  \]
  \[
  + x_1 (1-t)^3 + 3x_1 (1-t)^2 t + 3x_1 (1-t) t^2 + x_1 t^3
  \]
  \[
  + y_1 (1-t)^3 + 3y_1 (1-t)^2 t + 3y_1 (1-t) t^2 + y_1 t^3
  \]
  \[
  + z_1 (1-t)^3 + 3z_1 (1-t)^2 t + 3z_1 (1-t) t^2 + z_1 t^3
  \]

Plotting the Other Set of Isoparametric Curves

- For \((t=0; t+=1; t=t)\)
  - MoveTo \( x(0,t), y(0,t) \)
  - For \((s=0; s+=1; s=s)\)
    - LineTo \( x(s,t), y(s,t) \)
3-D Geometric Transformations
- Move objects in a 3-D scene
- Extension of 2-D Affine Transformations
- Three important ones:
  - Translation
  - Scaling
  - Rotations

Representing 3-D Points
- Homogeneous coordinates
- \[ P(x,y,z) \rightarrow P'(x',y',z') \]

Translations
- Given 3-D translation vector \( T=\langle tx, ty, tz \rangle \)
- Component equations
  - \( x'=x+tx \)
  - \( y'=y+ty \)
  - \( z'=z+tz \)
- Represent translation as matrix equation
  - \[ P' = T \cdot P \]
- \( T \) is a 4 X 4 Homogeneous Matrix

Homogeneous Translation Matrix

Scaling with respect to origin
- Given three scaling factors \( sx, sy, sz \)
- \[ P' = S \cdot P \]
- \( S \) is the following 4 X 4 scaling matrix:

Rotations
- Need to specify angle of rotation
- And axis about which the rotation is to be performed
- Infinite number of possible rotation axes
  - Rotation about any axis: linear combinations of rotations about x-axis, y-axis, z-axis
Rotations about z-axis

Consider rotation of point \( P=(x,y,z) \) by angle \( \theta \) about the z-axis giving rotated point \( P'=(x',y',z') \)

- Same \( x, y \) equations as in the 2-D case
- \( z \) will not change

**Z-Axis Rotation Component Equations**

\[
\begin{align*}
x' &= x \cos(\theta) - y \sin(\theta) \\
y' &= x \sin(\theta) + y \cos(\theta) \\
z' &= z
\end{align*}
\]

Represented as homogeneous matrix equation:

\[
P' = R_z \cdot P
\]

**Z-Axis Rotation Matrix**

\[
R_z = \begin{bmatrix}
\cos(\theta) & -\sin(\theta) & 0 & 0 \\
\sin(\theta) & \cos(\theta) & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Rx Matrix for rotations about x-axis

**Symmetry argument**

\[
y \quad z \\
| \quad | \\
| \quad | \\
| \quad | \\
x \quad y
\]

Make replacements:

\[
x \rightarrow y \\
y \rightarrow z \\
z \rightarrow x
\]

**Resulting equations:**

\[
\begin{align*}
y' &= y \cos(\theta) - z \sin(\theta) \\
z' &= y \sin(\theta) + z \cos(\theta) \\
x' &= x
\end{align*}
\]

Represented as matrix equation:

\[
P' = R_x \cdot P
\]

\[
R_x = \begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & \cos(\theta) & -\sin(\theta) & 0 \\
0 & \sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]

Ry Rotation Matrix

**Symmetry:**

\[
y \quad x \\
| \quad | \\
| \quad | \\
z \quad y
\]

Replacements:

\[
x \rightarrow z \\
y \rightarrow x \\
z \rightarrow y
\]

\[
P' = R_y \cdot P
\]

\[
R_y = \begin{bmatrix}
0 & 0 & 0 & 0 \\
0 & \cos(\theta) & -\sin(\theta) & 0 \\
0 & \sin(\theta) & \cos(\theta) & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
\]
\[ x \rightarrow z \]
\[ y \rightarrow x \]
\[ z \rightarrow y \]

\[ z' = z\cos(\theta) - x\sin(\theta) \]
\[ x' = z\sin(\theta) + x\cos(\theta) \]
\[ y' = y \]

\[ P' = R_y \cdot P \]
\[
\begin{pmatrix}
\cos(\theta) & 0 & \sin(\theta) & 0 \\
0 & 1 & 0 & 0 \\
-sin(\theta) & 0 & \cos(\theta) & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

**Rotation Sense**

- Positive sense
  - Defined as counter clockwise as we look down the rotation axis toward the origin