POLYGON MESHES

• list of vertices - polygon; list of edges - polygon
• list of polygons -- objects
• Plane equation from 3 vertices:
  \[ Ax + By + Cz + D = 0 \]
• Normal: \((A,B,C) = k(P_1P_2 \times P_1P_2)\)
• A, B, and C are proportional to the signed areas of the projections of the polygon onto the \((y, z)\), \((x, z)\), and \((x, y)\) planes. If the polygon is parallel to the \((x, y)\) plane, then \(A = B = 0\).
\[ C = \frac{1}{2} \sum_{i=1}^{n} (y_i + y_{i\oplus 1})(x_{i\oplus 1} - x_i) \quad \text{where the operator} \quad \oplus \]

is normal addition except that \( n \oplus 1 = 1. \)

The area for A and B are given by similar formulae.

\[ \text{• The distance } d \text{ for a vertex at } (x,y,z) \text{ is} \]

\[ d = \frac{Ax + By + Cz + D}{\sqrt{A^2 + B^2 + C^2}} \]
QUADRIC SURFACES

Quadric Surfaces

The implicit surface equation of the form defines the family of quadric surfaces:

\[ ax^2 + by^2 + cz^2 + 2(dxy + eyz + fxy + gxz + hxy + jyz) + k = 0 \]

Sphere

\[ x^2 + y^2 + z^2 = R^2; \text{ or in parametric form} \]
\[ x = r\cos\phi\cos\theta, \quad y = r\cos\phi\sin\theta, \quad z = r\sin\phi; \]

Ellipsoid

\[ \frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1; \text{ or in parametric form} \]
\[ x = a\cos\phi\cos\theta, \quad y = b\cos\phi\sin\theta, \quad z = c\sin\phi; \]

Torus

\[ \left[r - \left(\frac{x^2}{a^2} + \frac{y^2}{b^2}\right)^{1/2}\right]^2 + \frac{z^2}{c^2} = 1; \text{ or} \]
\[ x = a(r + \cos\phi)\cos\theta, \quad y = b(r + \cos\phi)\sin\theta, \quad z = c\sin\phi; \]
Superquadrics

supperellipse: \( x = a \cos^s \theta, \ y = b \sin^s \theta \)

supperellipsoid

\[
x = r \cos^s \phi \cos^t \theta, \ y = r \cos^s \phi \sin^t \theta, \ z = r \sin^s \phi;
\]

Blobby Objects

Some objects do not maintain a fixed shape, but change their surface characteristics in certain motions or when in proximity to other objects.

One way to model: combinations of Gaussian density functions, or 膊umps?

\[
\sum_k b_k e^{-a_k \left( x_k^2 + y_k^2 + z_k^2 \right)} = T \quad \text{where} \ T \ \text{is a threshold}
\]

and \( a \) and \( b \) are to adjust the amount of blobbiness
7 PARAMETRIC BICUBIC SURFACES

- General form of cubic curve: \( Q(u) = U \cdot M \cdot G \)
  where \( G \), the geometry vector, is a constant
- If we allow \( G \) to vary in 3D along some path:

\[
Q(s, t) = S \cdot M \cdot G(t) = S \cdot M \cdot \begin{bmatrix}
G_1(t) \\
G_2(t) \\
G_3(t) \\
G_4(t)
\end{bmatrix}
\]

Then, a functional description is often tesselated to produce a polygon-mesh approximation to the surface (triangular polygon patches)
- For a fixed \( t_1 \), \( Q(s, t_1) \) is a curve because \( G(t_1) \) is constant. If \( G_i(t) \) are cubics, the surface is said to be a parametric bicubic surface
Hermite Surfaces

**Curve:** \( x(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} M_h \)

\[
M_h = \begin{bmatrix}
2 & -2 & 1 & 1 \\
-3 & 3 & -2 & -1 \\
0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 
\end{bmatrix}
\]

\[
P_{1x}(t) = T \cdot M_h
\]

\[
\frac{\partial}{\partial t} x(0,0)
\]

\[
\frac{\partial}{\partial t} x(0,1)
\]

Since:

\[
\begin{bmatrix} P_1(t) & P_4(t) & R_1(t) & R_4(t) \end{bmatrix}_x = T \cdot M_h \cdot G_{Hx}^T
\]

we have:
\[ x(s,t) = S \cdot M_h \begin{bmatrix} P_1(t) \\ P_4(t) \\ R_1(t) \\ R_4(t) \end{bmatrix} = S \cdot M_h \cdot G_{Hx} \cdot M_h^T \cdot T^T \]

Where

\[
G_{Hx} = \begin{bmatrix}
x(0,0) & x(0,1) & \frac{\partial}{\partial t} x(0,0) & \frac{\partial}{\partial t} x(0,1) \\
x(1,0) & x(1,1) & \frac{\partial}{\partial t} x(1,0) & \frac{\partial}{\partial t} x(1,1) \\
\frac{\partial}{\partial s} x(0,0) & \frac{\partial}{\partial s} x(0,1) & \frac{\partial^2}{\partial s \partial t} x(0,0) & \frac{\partial^2}{\partial s \partial t} x(0,1) \\
\frac{\partial}{\partial s} x(1,0) & \frac{\partial}{\partial s} x(1,1) & \frac{\partial^2}{\partial s \partial t} x(1,0) & \frac{\partial^2}{\partial s \partial t} x(1,1) \\
\end{bmatrix}
\]

Where x coordinates, coordinates of the tangent vectors and twists are specified.
• Just as the Hermite cubic curves, the Hermite bicubic permits $C^1$ and $G^1$ continuity from one patch to the next.

• 1st, to have $C^0$ continuity, the matching curves of the two patches must be identical, which means the control points for the two surfaces must be identical along the edge.

• To have $C^1$ continuity, the control points along the edge and the tangent and twist vectors across the edge be equal.

• To have $G^1$ continuity, the tangent and twist vectors across the edge be in the same direction, but do not need to have the same magnitude.
Beziers Surfaces

The Beziers bicubic formulation can be derived in exactly the same way as above. The results are:

\[ x(s, t) = S \cdot M_b \cdot G_{Bx} \cdot M_b^T \cdot T^T \]

B-Spline Surfaces

The B-Spline bicubic formulation can be derived in exactly the same way also. The results are:

\[ x(s, t) = S \cdot M_{Bs} \cdot G_{BSx} \cdot M_{Bs}^T \cdot T^T \]

Normals to Surfaces

The cross product between the \( s \) and \( t \) tangent vectors of the surface \( Q(s, t) \) results in the normal at given \( s \) and \( t \):

\[ \frac{\partial}{\partial s} Q(s, t) \times \frac{\partial}{\partial t} Q(s, t) \]
EVALUATORS AND NURBS in OpenGL

One-Dimensional Evaluators

\[ B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i} \]  
and  
\[ C(u) = \sum_{i=0}^{n} B_i^n(u)P_i \]

Example Drawing a Bézier Curve
Using Four Control Points: bezcurve.c

```c
#include <GL/glut.h>
#include <stdlib.h>

GLfloat ctrlpoints[4][3] = {
    {-4.0, -4.0, 0.0},      
    {-2.0, 4.0, 0.0},       
    {2.0, -4.0, 0.0},       
    {4.0, 4.0, 0.0}         
};
```

Copyright © 2002 by Jim X. Chen:
jchen@cs.gmu.edu
void myinit(void) {
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3,
            4,&ctrlpoints[0][0]);
    glEnable(GL_MAP1_VERTEX_3);
    glShadeModel(GL_FLAT);
}

GL_MAP1_VERTEX_3: Three-dimensional vertices are produced
0--Low value of parameter u
1--High value of parameter u
3--The number of floating-point values to advance in the data between one control point and the next
4--The order of the spline, which is the degree+1;
&ctrlpoints[0][0]--Pointer to the first control point’s data
void display(void) {
    int i;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 30; i++)
            glEvalCoord1f((GLfloat) i/30.0);
    glEnd();

    /* The following code displays the control points as dots. */
    glPointSize(5.0);
    glColor3f(1.0, 1.0, 0.0);
    glBegin(GL_POINTS);
        for (i = 0; i < 4; i++)
            glVertex3fv(&ctrlpoints[i][0]);
    glEnd();
    glFlush();
}
Two-Dimensional Evaluators

• Mathematically, the definition of a Bézier surface patch is given by

\[ S(u, v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u) B_j^m(v) P_{ij} \]

Example 11-2 Drawing a Bézier Surface: bezsurf.c

```c
GLfloat ctrlpoints[4][4][3] = {
    {{-1.5, -1.5, 4.0}, {-0.5, -1.5, 2.0}, {0.5, -1.5, -1.0}, {1.5, -1.5,2.0}},
    {{-1.5, -0.5, 1.0}, {-0.5, -0.5, 3.0}, {0.5, -0.5, 0.0}, {1.5, -0.5,-1.0}},
    {{-1.5, 0.5, 4.0}, {-0.5, 0.5, 0.0},{0.5, 0.5, 3.0}, {1.5, 0.5, 4.0}},
    {{-1.5, 1.5, -2.0}, {-0.5, 1.5, -2.0}, {0.5, 1.5, 0.0}, {1.5, 1.5, -1.0}}
};

void myinit(void)
{
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 
        0, 1, 12, 4, &ctrlpoints[0][0][0]);
}```


```c
void display(void) {
    int i, j;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glPushMatrix();
    glRotatef(85.0, 1.0, 1.0, 1.0);
    for (j = 0; j <= 8; j++) {
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 30; i++)
            glEvalCoord2f((GLfloat)i/30.0, (GLfloat)j/8.0);
        glEnd();
        glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 30; i++)
            glEvalCoord2f((GLfloat)j/8.0, (GLfloat)i/30.0);
        glEnd();
    }
    glPopMatrix();
    glFlush();
}
```

void initLights(void) {
    GLfloat ambient[] = { 0.2, 0.2, 0.2, 1.0 };  
    GLfloat position[] = { 0.0, 0.0, 2.0, 1.0 }; 

    GLfloat mat_diffuse[] = {0.6, 0.6, 0.6, 1.0 }; 
    GLfloat mat_specular[] = {1.0,1.0, 1.0, 1.0 };  
    GLfloat mat_shininess[] = { 50.0 };  
    glEnable(GL_LIGHTING);  
    glEnable(GL_LIGHT0);  
    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);  
    glLightfv(GL_LIGHT0, GL_POSITION, position);  
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, mat_diffuse);  
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, mat_specular);  
    glMaterialfv(GL_FRONT_AND_BACK, GL_SHININESS, mat_shininess); 
}
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    
    glPushMatrix();
    glRotatef(85.0, 1.0, 1.0, 1.0);
    glEvalMesh2(GL_FILL, 0, 8, 0, 8);
    glPopMatrix();
    glFlush();
}

void myinit(void) {
    glClearColor (0.0, 0.0, 0.0, 1.0);
    glEnable(GL_DEPTH_TEST);
    
glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4, 0, 1, 12, 4, &ctrlpoints[0][0][0]);
    glEnable(GL_MAP2_VERTEX_3);
    glEnable(GL_AUTO_NORMAL);
    glMapGrid2f(8, 0.0, 1.0, 8, 0.0, 1.0);
    initlights();
}
The GLU NURBS Interface

• The GLU provides a NURBS (Non-Uniform Rational B-Spline) interface built on top of the OpenGL evaluator commands.

Example 11-5 Drawing a NURBS Surface: surface.c

```c
#include <GL/glu.h>
#include <stdlib.h>
#include <stdio.h>

GLfloat ctlpoints[4][4][3];
GLUnurbsObj *theNurb;
```
void init_surface(void)
{
    int u, v;
    for (u = 0; u < 4; u++) {
        for (v = 0; v < 4; v++) {
            ctlpoints[u][v][0] = 2.0*((GLfloat)u - 1.5);
            ctlpoints[u][v][1] = 2.0*((GLfloat)v - 1.5);
            if ( (u == 1 || u == 2) && (v == 1 || v == 2))
                ctlpoints[u][v][2] = 3.0;
            else
                ctlpoints[u][v][2] = -3.0;
        }
    }
}

void myinit(void)
{
    GLfloat mat_diffuse[] = { 0.7, 0.7, 0.7, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_shininess[] = { 100.0 };
    glClearColor (0.0, 0.0, 0.0, 1.0);
glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);

glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);

glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

.glEnable (GL_LIGHTING);

.glEnable (GL_LIGHT0);

.glDepthFunc(GL_LEQUAL);

.glEnable (GL_DEPTH_TEST);

.glEnable (GL_AUTO_NORMAL);

.glEnable (GL_NORMALIZE);

init_surface();

theNurb = gluNewNurbsRenderer();

.gluNurbsProperty (theNurb, GLU_SAMPLING_TOLERANCE, 25.0);

.gluNurbsProperty (theNurb, GLU_DISPLAY_MODE, GLU_FILL);

}
void display(void)
{
    GLfloat knots[8] = {0.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0,

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glPushMatrix();
    glRotatef(330.0, 1.,0.,0.);
    glScalef (0.5, 0.5, 0.5);
    gluBeginSurface(theNurb);
    gluNurbsSurface(theNurb,
        8, knots,
        8, knots,
        4 * 3,
        3,
        &ctlpoints[0][0][0][0],
        4, 4,
        GL_MAP2_VERTEX_3);
    gluEndSurface(theNurb);
    glPopMatrix();
    glFlush();
}