Evaluadores bidimensionales

En dos dimensiones, todo es similar al caso unidimensional, excepto que todos los comandos deben Tenga en cuenta dos parámetros, U y V,.Los puntos, colores, normales o coordenadas de textura deben ser suministrado sobre una superficie en lugar de una curva.Matemáticamente, la definición de un parche de superficie Bézier es dada por

$$S(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u) B_j^m(v) P_{ij} \tag{1}$$

donde P_{ij} son un conjunto de puntos de control $m * n$ y los B_i son los mismos polinomios de Bernstein para uno dimensión.Como antes, el P_{ij} puede representar vértices, normales, colores o coordenadas de textura.

El procedimiento para usar evaluadores bidimensionales es similar al procedimiento para una dimensión.

1. Defina los evaluadores (s) con glMap2*()
2. Habilitarlos pasando el valor apropiado a glEnable().
3. Invocarlos llamando a glEvalCoord2() entre un par glBegin() y glEnd() o por Especificar y luego aplicar una malla con glMapGrid2() y glEvalMesh2().

Definición y evaluación de un evaluador bidimensional

Use GLMAP2*() y GlevalCoord2*() para definir y luego invocar un evaluador bidimensional. void glMap2fd(GLenum target, TYPEu1, TYPEu2, GLint ustride, GLint uorder, TYPEv1, TYPEv2, GLint vstride, GLint vorder, TYPE points);

El parámetro objetivo puede tener cualquiera de los valores en la Tabla 12-1, excepto que la cadena MAP1 es reemplazado con MAP2.Como antes, estos valores también se usan con glEnable() para habilitar el evaluador correspondiente.Los valores mínimos y máximos para U y V se proporcionan como U1, U2, V1 y V2.Los parámetros de ustride y vstride indican el número de precisión única o doble valores (según corresponda) entre la configuración independiente para estos valores, lo que permite a los usuarios seleccionar un El subrectangle de control señala de una matriz mucho más grande.Por ejemplo, si los datos aparecen en el forma.

GLfloat ctlpoints[100][100][3];

y desea usar el subconjunto 4x4 que comienza en cltpoints[20][30], elija Ustride para que sea 100*3 y Vstride para ser 3. El punto de partida, puntos, debe establecerse en &*ctlpoints*[20][30][0].Finalmente, el Los parámetros de pedido, Uorder y Vorder pueden ser diferentes, lo que permite parches cúbicos en uno dirección y cuadrática en el otro, por ejemplo.

1. void glEvalCoord2fd(TYPE u, TYPE v)

2. void glEvalCoord2fdv(TYPE *values)

Causa la evaluación de los mapas bidimensionales habilitados.Los argumentos U y V son los valores (o un puntero a los valores U y V, en la versión vectorial del comando) para las coordenadas de dominio. Si se habilita cualquiera de los evaluadores de vértices ($GL_MAP2_VERTEX_3$ o $GL_MAP2_VERTEX_4$), entonces lo normal a la superficie se calcula analíticamente.Esta normal está asociada con el vértice generado si la generación normal automática se ha habilitado pasando GL_AUTO_NORMAL a glEnable().Si está deshabilitado, el mapa normal habilitado habilitado correspondiente es utilizado para producir una normal.Si no existe tal mapa, se usa la corriente actual.

Ejemplo 12-2 dibuja una superficie de estructura bézier utilizando evaluadores, como se muestra en la Figura 12-2.En esto Ejemplo, la superficie se dibuja con nueve líneas curvas en cada dirección.Cada curva se dibuja como 30 segmentos.Para obtener todo el programa, agregue las rutinas Reshape () y Main () del ejemplo 12-1.

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <stdlib.h>
#include <GL/glut.h>

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 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 init(void)
{
    glClearColor (0.0, 0.0, 0.0, 0.0);
    glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
    0, 1, 12, 4, &ctrlpoints[0][0][0]);
    glEnable(GL_MAP2_VERTEX_3);
    glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
    glEnable(GL_DEPTH_TEST);
    glShadeModel(GL_FLAT);
}


void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
    glOrtho(-5.0, 5.0, -5.0*(GLfloat)h/(GLfloat)w,
    5.0*(GLfloat)h/(GLfloat)w, -5.0, 5.0);
    else
    glOrtho(-5.0*(GLfloat)w/(GLfloat)h,
    5.0*(GLfloat)w/(GLfloat)h, -5.0, 5.0, -5.0, 5.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
}


int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (1080, 1080);
    glutInitWindowPosition (420, 0);
    glutCreateWindow("Superficie ejemplo 12-2");
    init ();
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutMainLoop();
    return 0;
}
```

Definición de valores de coordenadas espaciados uniformemente en dos dimensiones

En dos dimensiones, los comandos glMapGrid2*() y glEvalMesh2() son similares a los Versiones unidimensionales, excepto que se deben incluir información de U y V.

- void glMapGrid2fd(GLint nu, TYPEu1, TYPEu2, GLint nv, TYPEv1, TYPEv2);

- void glEvalMesh2(GLenum mode, GLint i1, GLint i2, GLint j1, GLint j2);

Define una cuadrícula de mapa bidimensional que va de U1 a U2 en pasos nu uniformemente espaciados, de V1 a V2 en pasos NV (glMapGrid2*()), y luego aplica esta cuadrícula a todos los evaluadores habilitados (glEvalMesh2()).La única diferencia significativa con las versiones unidimensionales de estos dos Los comandos son que en glevalmesh2 () el parámetro de modo puede ser GL_FILL y GL_POINT o GL_LINE GL_FILL genera polígonos rellenos utilizando el primitivo de malla cuádruple.Declarado precisamente, GlevalMesh2 () es casi equivalente a uno de los siguientes tres fragmentos de código.(Es casi equivalente porque cuando yo es igual a nu o j a nv, el parámetro es exactamente igual a u2 o v2, no a $u1 + nu * (u2 - u1)/nu$, que podría ser ligeramente diferente debido al error redondeado).

```
    glBegin(GL_POINTS); /* mode == GL_POINT */
        for (i = nu1; i <= nu2; i++)
        for (j = nv1; j <= nv2; j++)
        glEvalCoord2(u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
    glEnd();
```

o

```

    for (i = nu1; i <= nu2; i++) { /* mode == GL_LINE */
    glBegin(GL_LINES);
        for (j = nv1; j <= nv2; j++)
        glEvalCoord2(u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
    glEnd();
    }

    for (j = nv1; j <= nv2; j++) {
    glBegin(GL_LINES);
        for (i = nu1; i <= nu2; i++)
        glEvalCoord2(u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
    glEnd();
    }
```

or

```
    for (i = nu1; i < nu2; i++) { /* mode == GL_FILL */
        glBegin(GL_QUAD_STRIP);
        for (j = nv1; j <= nv2; j++) {
            glEvalCoord2(u1 + i*(u2-u1)/nu, v1+j*(v2-v1)/nv);
            glEvalCoord2(u1 + (i+1)*(u2-u1)/nu, v1+j*(v2-v1)/nv);
        glEnd();
    }
}
```

Ejemplo 12-3 muestra las diferencias necesarias para dibujar la misma superficie de Bézier que el ejemplo 12-2, pero Usando glMapGrid2() y glEvalMesh2() para subdividir el dominio cuadrado en una cuadrícula uniforme de 8x8.Este El programa también agrega iluminación y sombreado, como se muestra en la Figura 12-3.

```
#include <GL/gl.h>
#include <GL/glu.h>
#include <stdlib.h>
#include <GL/glut.h>

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 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, GL_DIFFUSE, mat_diffuse);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialfv(GL_FRONT, 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, 20, 0, 20);
    glPopMatrix();
    glFlush();
}


void init(void)
{
   glClearColor(0.0, 0.0, 0.0, 0.0);
    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(20, 0.0, 1.0, 20, 0.0, 1.0);
    initlights();
}
```

```
58
59
60 void reshape(int w, int h)
61 {
62     glViewport(0, 0, (GLsizei) w, (GLsizei) h);
63     glMatrixMode(GL_PROJECTION);
64     glLoadIdentity();
65     if (w <= h)
66     glOrtho(-5.0, 5.0, -5.0*(GLfloat)h/(GLfloat)w,
67     5.0*(GLfloat)h/(GLfloat)w, -5.0, 5.0);
68     else
69     glOrtho(-5.0*(GLfloat)w/(GLfloat)h,
70     5.0*(GLfloat)w/(GLfloat)h, -5.0, 5.0, -5.0, 5.0);
71     glMatrixMode(GL_MODELVIEW);
72     glLoadIdentity();
73 }
74
75
76 int main(int argc, char** argv)
77 {
78     glutInit(&argc, argv);
79     glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
80     glutInitWindowSize (1080, 1080);
81     glutInitWindowPosition (420, 0);
82     glutCreateWindow("Superficie ejemplo 12-3");
83     init ();
84     glutDisplayFunc(display);
85     glutReshapeFunc(reshape);
86     glutMainLoop();
87     return 0;
88 }
```

Usar evaluadores para texturas

Ejemplo 12-4 habilita dos evaluadores al mismo tiempo: el primero genera puntos tridimensionales en La misma superficie de Bézier que el ejemplo 12-3, y la segunda genera coordenadas de textura.En este caso, el Las coordenadas de textura son las mismas que las coordenadas U y V de la superficie, pero un parche especial de Bézier debe crearse para hacer esto. El parche plano se define sobre un cuadrado con esquinas en (0, 0), (0, 1), (1, 0) y (1, 1);genera (0, 0) en esquina (0, 0), (0, 1) en la esquina (0, 1), y así sucesivamente.Ya que es de orden dos (grado lineal más uno), evaluando

Esta textura en el punto (U, V) genera coordenadas de textura (S, T).Está habilitado al mismo tiempo que el Evaluador de vértice, por lo que ambos surtan efecto cuando se dibuja la superficie.(Ver "Placa 19" en el Apéndice I.) Si Desea que la textura repita tres veces en cada dirección, cambie cada 1.0 en la matriz texpts[][][] a 3.0. Dado que la textura se envuelve en este ejemplo, la superficie se representa con nueve copias del mapa de textura.

```
1 #include <GL/gl.h>
2 #include <GL/glu.h>
3 #include <stdlib.h>
4 #include <GL/glut.h>
```

```
#include <math.h>

#define imageWidth 64
#define imageHeight 64
GLubyte image[3*imageWidth*imageHeight];

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}}
    };

GLfloat texpts[2][2][2] = {{{0.0, 0.0}, {0.0, 1.0}},
    {{1.0, 0.0}, {1.0, 1.0}}};



void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glEvalMesh2(GL_FILL, 0, 20, 0, 20);
    glFlush();
}

void makeImage(void){
    int i, j;
    float ti, tj;
    for (i = 0; i < imageWidth; i++) {
        ti = 2.0*3.14159265*i/imageWidth;
        for (j = 0; j < imageHeight; j++) {
            tj = 2.0*3.14159265*j/imageHeight;
            image[3*(imageHeight*i+j)] =

            (GLubyte) 127*(1.0+sin(ti));
            image[3*(imageHeight*i+j)+1] =
            (GLubyte) 127*(1.0+cos(2*tj));
            image[3*(imageHeight*i+j)+2] =
            (GLubyte) 127*(1.0+cos(ti+tj));
        }
    }
}


void init(void)
{
    glMap2f(GL_MAP2_VERTEX_3, 0, 1, 3, 4,
    0, 1, 12, 4, &ctrlpoints[0][0][0]);
    glMap2f(GL_MAP2_TEXTURE_COORD_2, 0, 1, 2, 2,
    0, 1, 4, 2, &texpts[0][0][0]);
    glEnable(GL_MAP2_TEXTURE_COORD_2);
    glEnable(GL_MAP2_VERTEX_3);
```

```
    glMapGrid2f(20, 0.0, 1.0, 20, 0.0, 1.0);
    makeImage();
    glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_DECAL);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
    GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
    GL_NEAREST);
    glTexImage2D(GL_TEXTURE_2D, 0, 3, imageWidth, imageHeight, 0,
    GL_RGB, GL_UNSIGNED_BYTE, image);
    glEnable(GL_TEXTURE_2D);
    glEnable(GL_DEPTH_TEST);
    glShadeModel (GL_FLAT);
}


void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei) w, (GLsizei) h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if (w <= h)
    glOrtho(-4.0, 4.0, -4.0*(GLfloat)h/(GLfloat)w,
    4.0*(GLfloat)h/(GLfloat)w, -4.0, 4.0);
    else
    glOrtho(-4.0*(GLfloat)w/(GLfloat)h,
    4.0*(GLfloat)w/(GLfloat)h, -4.0, 4.0, -4.0, 4.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    glRotatef(85.0, 1.0, 1.0, 1.0);
}


int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
glutInitWindowSize (1080, 1080);
glutInitWindowPosition (420,0);
glutCreateWindow ("Superficie ejemplo 12-4");
init ();
glutDisplayFunc(display);
glutReshapeFunc(reshape);
glutMainLoop();
return 0;
}
```