EVALUADORES Y NURBS.

Computación Gráfica.

Profesora: Jhelly Pérez Núñez1

INTEGRANTES

Balboa Palma, Merly Estefany

Sebastian Vazquez

Mitchell Mirano

5 de agosto de 2022

Contenido

1 Resumen

Los evaluadores hacen splines y superficies que se basan en una base de Bézier (o Bernstein). Si se desea utilizar el evaluador para trazar curvas y superficies utilizando otras bases, se debe saber cómo convertir su base en una base Bezier. Además, al renderizar una superficie Bézier o parte de ella utilizando el evaluador, es necesario especificar el nivel de detalle de su subdivisión. La funcionalidad NURBS de GLU es una interfaz de alto nivel: los procesos NURBS encapsulan una gran cantidad de código complejo,los procesos NURBS utilizan polígonos planos para el renderizado.

2 Abstract

Evaluators make splines and surfaces that are based on a Bézier (or Bernstein) basis. If you want to use the evaluator to draw curves and surfaces using other bases, you must know how to convert your base to a Bézier base. Also, when rendering a Bezier surface or part of it using the evaluator, it is necessary to specify the level of detail of its subdivision. GLU's NURBS functionality is a high-level interface: NURBS processes encapsulate a lot of complex code, NURBS processes use flat polygons for rendering.

3 Introducción

En el nivel más bajo, el hardware de gráficos dibuja puntos, líneas y polígonos, que suelen ser triángulos y cuadriláteros.Las curvas y las superficies suaves se dibujan aproximandolas con líneas, polígonos grandes o pequeños. Sin embargo, muchas curvas y superficies útiles se pueden describir matemáticamente mediante una pequeña cantidad de parámetros, como algunos puntos de control. Registrar los 16 puntos de control de una superficie requiere mucho menos espacio que registrar 1000 triángulos con información vectorial normal en cada vértice. Además, los 1000 triángulos solo se aproximan a la superficie real, pero los puntos de control representan con precisión la superficie real.

Los evaluadores permiten especificar puntos en una curva o superficie (o parte de ella) simplemente usando puntos de control. La curva o la superficie se pueden mostrar con cualquier precisión. Además, los vectores normales se pueden calcular automáticamente para las superficies. Los puntos generados por el evaluador se pueden usar de muchas maneras: para dibujar puntos en ubicaciones de superficie, para dibujar una versión alámbrica de la superficie, para dibujar una superficie sombreada o textura.

El evaluador se puede utilizar para describir todas las splines o superficies de polinomios o polinomios racionales en cualquier grado. Esto incluye casi todas las splines y superficies de splines en uso hoy en día, incluidas B-splines, NURBS (B-Splines racionales no uniformes), curvas y superficies de Bézier y splines de Hermite. Dado que los evaluadores solo brindan descripciones de bajo nivel de puntos en curvas o superficies, a menudo se usan en bibliotecas de utilidades que brindan una interfaz de nivel superior para el programador. La función NURBS de GLU es una interfaz de alto nivel: Los procesos NURBS encapsulan una gran cantidad de

código complejo. Gran parte del renderizado final se realiza con el evaluador, pero para ciertas condiciones (por ejemplo, recorte de curvas), los procesos NURBS utilizan polígonos planos para el renderizado.

Este capítulo contiene las siguientes secciones principales: Los evaluadores, en ella se explica cómo funcionan los evaluadores y cómo controlarlos utilizando los comandos de OpenGL y la interfaz NURBS de GLU, la cual describe las rutinas de GLU para crear superficies NURBS.

4 Evaluadores

Los evaluadores permiten especificar puntos en una curva o superficie simplemente usando puntos de control. La curva o la superficie se pueden mostrar con cualquier precisión.

Una curva de Bézier es una función vectorial de una variable.

$$C(u) = [X(u)\, Y(u)\, Z(u)]$$

Donde u varía en un dominio $[0, 1]$.

Una superficie de Bézier es una función vectorial de dos variables.

$$S(u, v) = [X(u, v)\, Y(u, v)\, Z(u, v)]$$

Donde u y v pueden variar en algún dominio. El rango no es necesariamente tridimensional como se muestra. Puede tener una salida bidimensional para curvas en un plano o coordenadas de textura, o una salida cuatridimensional para especificar información RGBA. Incluso la salida unidimensional puede tener sentido para los niveles de gris.

Para cada u (u y v, en el caso de una superficie), la fórmula de **C()** o **S()** calcula un punto de la curva o superficie. Para utilizar un evaluador, primero hay que definir la función **C()** o **S()**, habilitarla y luego utilizar el comando **glEvalCoord1()** o **glEvalCoord2()** en lugar de **glVertex*()**. De este modo, los vértices de la curva o de la superficie pueden utilizarse como cualquier otro vértice, para formar puntos o líneas. Además, otros comandos generan automáticamente series de vértices que producen una malla regular uniformemente espaciada en u (o en u y v). Los evaluadores unidimensionales y bidimensionales son similares, pero la descripción es algo más sencilla en una dimensión.

4.1 Evaluadores unidimensionales

Presentamos un ejemplo del uso de evaluadores unidimensionales para dibujar una curva. A continuación, se describen los comandos y ecuaciones que controlan los evaluadores.

Ejemplo unidimensional: Una curva de Bézier simple.

El programa mostrado en el Ejemplo 1 dibuja una curva cúbica de Bézier usando cuatro puntos de control, como se muestra en la Figura 1.

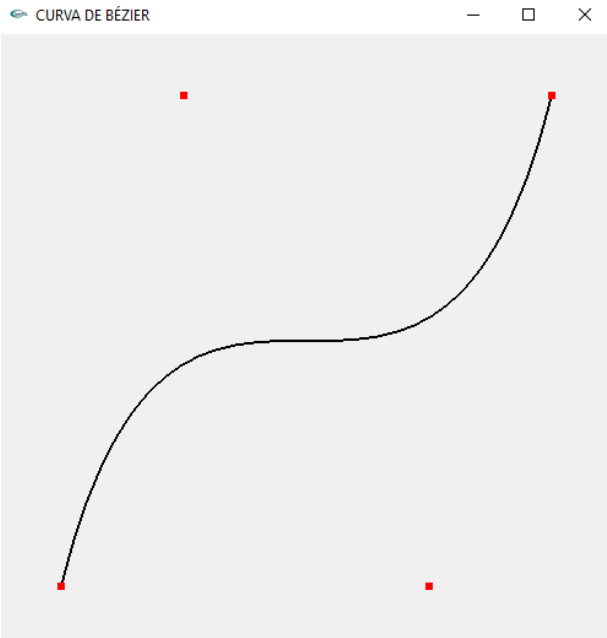

Figura 1: Curva de Bézier.

4.1.1. Código del ejemplo

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

void init(void){
  glClearColor(0.0, 0.0, 0.0, 0.0);
  glShadeModel(GL_FLAT);
  glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, &ctrlpoints[0][0]);
  glEnable(GL_MAP1_VERTEX_3);
}

void display(void){
      int i;
      glClear(GL_COLOR_BUFFER_BIT);
      glColor3f(1.0, 1.0, 1.0);

      glBegin(GL_LINE_STRIP);
        for (i = 0; i <= 30; i++)
            glEvalCoord1f((GLfloat) i/30.0);
```

```
25        glEnd();
26
27   /* El siguiente codigo muestra los puntos de control como puntos. */
28        glPointSize(5.0);
29        glColor3f(1.0, 1.0, 0.0);
30        glBegin(GL_POINTS);
31          for (i = 0; i < 4; i++)
32            glVertex3fv(&ctrlpoints[i][0]);
33        glEnd();
34
35  glFlush();
36  }
37
38  void reshape(int w, int h) {
39        glViewport(0, 0, (GLsizei) w, (GLsizei) h);
40        glMatrixMode(GL_PROJECTION);
41        glLoadIdentity();
42
43        if (w <= h)
44          glOrtho(-5.0, 5.0, -5.0*(GLfloat)h/(GLfloat)w,
45          5.0*(GLfloat)h/(GLfloat)w, -5.0, 5.0);
46        else
47        glOrtho(-5.0*(GLfloat)w/(GLfloat)h,
48        5.0*(GLfloat)w/(GLfloat)h, -5.0, 5.0, -5.0, 5.0);
49        glMatrixMode(GL_MODELVIEW);
50        glLoadIdentity();
51  }
52
53  int main(int argc, char** argv) {
54        glutInit(&argc, argv);
55        glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
56        glutInitWindowSize (500, 500);
57        glutInitWindowPosition (100, 100);
58        glutCreateWindow (argv[0]);
59        init ();
60        glutDisplayFunc(display);
61        glutReshapeFunc(reshape);
62        glutMainLoop();
63        return 0;
64  }
```

La curva cúbica de Bézier se describe mediante cuatro puntos de control, que aparecen en este ejemplo en la matriz **ctrlpoints[][]**. Esta matriz es uno de los argumentos de **glMap1f()**. Todos los argumentos de este comando son los siguientes:

- $GL\_MAP1\_VERTEX\_3$ **:** Proporciona puntos de control tridimensionales y se producen vértices tridimensionales.
- **0.0:** Valor bajo del parámetro u
- **1.0:** Valor elevado del parámetro u
- **3:** El número de valores de punto flotante para avanzar en los datos entre un punto de control y el siguiente.

- **4:** El orden de la spline, que es el grado + 1: en este caso, el grado es 3 (ya que se trata de una curva cúbica).
- **&ctrlpoints[0][0]:** Puntero a los datos del primer punto de control

Nótese que el segundo y tercer argumento controlan la parametrización de la curva ya que la variable u va de 0 a 1, la curva va de un extremo a otro. La llamada a **glEnable()** habilita el evaluador unidimensional para vértices tridimensionales.

La curva se dibuja en la rutina **display()** entre las llamadas **glBegin()** y **glEnd()**. Como el evaluador está habilitado, el comando **glEvalCoord1f()** es como emitir un comando **glVertex()** con las coordenadas de un vértice de la curva correspondiente al parámetro de entrada u.

4.2 Definición y evaluación de un evaluador unidimensional

El polinomio de Bernstein de grado n (u orden $n+1$) aproxima una función en un intervalo y viene dado por:

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

Si Pi representa un conjunto de puntos de control (unidimensionales, bidimensionales, tridimensionales o incluso cuatridimensionales), entonces la ecuación:

$$C(u) = \sum_{i=0}^{n} B_i^n(u) P_i$$

Donde los $B_i^n(u)$ son elementos de la distribución binomial respecto de la variable u y los P_i son valores de la función que queremos aproximar.

Representa una curva de Bézier cuando u varía de 0 a 1. Para representar la misma curva, pero permitiendo que, u varíe entre $u1$ y $u2$ en lugar de 0 y 1, evaluamos:

$$C\left(\frac{u-u_1}{u_2-u_1}\right)$$

El comando **glMap1()** define un evaluador unidimensional que utiliza estas ecuaciones.

```
1    void glMap1(GLenum target,TYPE u1,TYPE u2,GLint stride,
2
3                    GLint order,const TYPE *points);
```

El parámetro objetivo especifica lo que representan los puntos de control, como se muestra en la Tabla 1, y por lo tanto cuántos valores deben ser suministrados en puntos. Los puntos pueden representar vértices, datos de color RGBA, vectores normales o coordenadas de textura. Por ejemplo, con $GL\_MAP1\_COLOR\_4$, el evaluador genera datos de color a lo largo de una curva en el espacio de color de cuatro dimensiones (RGBA). También se utilizan los

valores de los parámetros listados en la Tabla 1 para activar cada evaluador definido antes de invocarlo. Pase el valor apropiado a **glEnable()** o **glDisable()** para activar o desactivar el evaluador.

Los dos segundos parámetros de **glMap1*()**, $u1$ y $u2$, indican el rango de la variable u. La variable stride es el número de valores de precisión simple o doble (según el caso) en cada bloque de almacenamiento. Por lo tanto, es un valor de desplazamiento entre el comienzo de un punto de control y el comienzo del siguiente.

El orden es el grado más uno, y debe coincidir con el número de puntos de control. Los puntos apuntan a la primera coordenada del primer punto de control. Utilizando la estructura de datos de ejemplo para **glMap1*()**, utilice lo siguiente para los puntos:

$$(GLfloat\ *)(\&ctrlpoints[0].x)$$

Tabla 1: Tipos de puntos de control para **glMap1*()**.

Parámetro	Significado
GL_MAP1_VERTEX_3	coordenadas de los vértices x, y, z.
GL_MAP1_VERTEX_4	coordenadas de los vértices x, y, z, w.
GL_MAP1_INDEX	índice de color.
GL_MAP1_COLOR_4	R, G, B, A.
GL_MAP1_NORMAL	coordenadas normales.
GL_MAP1_TEXTURE_COORD_1	s coordenadas de textura.
GL_MAP1_TEXTURE_COORD_2	s, t coordenadas de textura.
GL_MAP1_TEXTURE_COORD_3	s, t, r coordenadas de textura
GL_MAP1_TEXTURE_COORD_4	s, t, r, q coordenadas de textura.

Se puede evaluar más de un evaluador a la vez. Se tiene un evaluador GL_MAP1_VERTEX_3 y un evaluador GL_MAP1_COLOR_4 definidos y habilitados, entonces las llamadas a **glEvalCoord1()** generan tanto una posición como un color. Sólo uno de los evaluadores de vértices puede estar habilitado a la vez, aunque se hayan definido ambos. Del mismo modo, sólo uno de los evaluadores de textura puede estar activo. Sin embargo, aparte de esto, los evaluadores pueden utilizarse para generar cualquier combinación de datos de vértices, vectores normales, colores y coordenadas de textura. Si se define y activa más de un evaluador del mismo tipo, se utiliza el de mayor dimensión. Utilizamos **glEvalCoord1*()** para evaluar un mapa unidimensional definido y habilitado.

```
1  void glEvalCoord1{fd}(TYPE u);
2  void glEvalCoord1{fd} v (TYPE *u);
```

Provoca la evaluación de los mapas unidimensionales habilitados. El argumento u es el valor (o un puntero al valor, en la versión vectorial del comando) de la coordenada del dominio.

4.3 Definición de valores de coordenadas uniformes en una dimensión

Se utiliza **glEvalCoord1()** con cualquier valor para u, pero el uso más común es con valores espaciados uniformemente, como se muestra en el Ejemplo 1. Para obtener valores uniformemente espaciados, definimos una unidimensional usando **glMapGrid1*()** y luego aplíquelo usando **glEvalMesh1()**.

```
void glMapGrid1{fd} (GLint n, TYPE u1, TYPE u2);
```

Define una cuadrícula que va de $u1$ a $u2$ en n pasos, que están espaciados uniformemente.

```
void glEvalMesh1(GLenum mode, GLint p1, GLint p2);
```

Aplicamos la cuadrícula de mapa definida actualmente a todos los evaluadores habilitados. El modo puede ser **GL_POINT o GL_LINE**, dependiendo de si desea dibujar puntos o una línea conectada a lo largo de la curva. La llamada tiene exactamente el mismo efecto que emitir una **glEvalCoord1()** para cada uno de los pasos, incluyendo $p1$ y $p2$, donde $0 \leq p1,\ p2 \leq n$. Es equivalente a lo siguiente:

```
glBegin(GL_POINTS); /* OR glBegin(GL_LINE_STRIP); */
      for (i = p1; i <= p2; i++)
          glEvalCoord1(u1 + i*(u2-u1)/n);
glEnd();
```

Excepto que si $i = 0$ o $i = n$, entonces se llama a **glEvalCoord1()** con exactamente $u1$ o $u2$ como parámetro.

Referencias