Open Inventor
7 - Superfici

Claudio Baldazzi *Maurizio Patrignani*

Università degli Studi di Roma Tre
Dip. di Informatica e Automazione
Via della Vasca Navale, 79 – 00146 Roma, Italy

13 maggio 1998

1 Introduzione

Una curva, per quanto complessa, può definire il profilo di un oggetto, ma non riesce a definire un universo credibile. Nella mondo reale gli oggetti ci si presentano come delle superfici. Per questo motivo è importante poter descrivere in maniera succinta delle superfici geometriche.

Vedremo nel seguito che l'uso di superfici in Open Inventor, seppure elastico e potente, non è forse altrettanto *friendly* ed immediato.

Le informazioni, le figure e i listati contenuti in questo documento, sono tratti, salvo diversa indicazione, dal libro "The Inventor Mentor" di Josie Wernecke edito dalla Addison Wesley.

2 Overview sulle superfici

Una superficie differisce da una curva per il solo fatto che lo spazio dei parametri è bidimensionale anziché unidimensionale (si rammenta che si fa riferimento sempre a rappresentazioni parametriche di questi oggetti geometrici). Avremo sempre due "spazi" dunque: lo spazio dei parametri e lo spazio dell'oggetto. Possiamo immaginare lo spazio dei parametri come l'area del piano u,v, nella quale le due variabili hanno valori compresi tra zero e uno. Lo spazio dell'oggetto è invece costituito dalla solita terna cartesiana x, y e z.

Si noti che i punti di controllo sono specificati nello spazio oggetto. I parametri u e v possono avere ordine differente e una differente sequenza di knot, ma nella quasi totalità dei casi essi hanno medesimo ordine e sequenza di knot. L'ordine di ogni dimensione è ottenibile come segue:

ordine = numero-dei-nodi - numero-dei-punti-di-controllo

3 Superfici di Bezier

L'esempio seguente crea una superficie di Bezier. Il vettore dei knot definisce una supercificie di Bezier cubica.

```
1 // The control points for this surface
2 const float pts[16][3] = {
3    {-4.5, -2.0,  8.0},
4    {-2.0,  1.0,  8.0},
5    { 2.0, -3.0,  6.0},
6    { 5.0, -1.0,  8.0},
7    {-3.0,  3.0,  4.0},
8    { 0.0, -1.0,  4.0},
9    { 1.0, -1.0,  4.0},
10   { 3.0,  2.0,  4.0},
11   {-5.0, -2.0, -2.0},
12   {-2.0, -4.0, -2.0},
13   { 2.0, -1.0, -2.0},
14   { 5.0,  0.0, -2.0},
15   {-4.5,  2.0, -6.0},
16   {-2.0, -4.0, -5.0},
17   { 2.0,  3.0, -5.0},
18   { 4.5, -2.0, -6.0}};

19 // The knot vector
20 float knots[8] = {
21   0, 0, 0, 0, 1, 1, 1, 1};

22 // Create the nodes needed for the Bezier surface.
23 SoSeparator *
24 makeSurface()
25 {
26   SoSeparator *surfSep = new SoSeparator();
27   surfSep->ref();

28   // Define the Bezier surface including the control
29   // points and a complexity.
30   SoComplexity  *complexity = new SoComplexity;
31   SoCoordinate3 *controlPts = new SoCoordinate3;
32   SoNurbsSurface  *surface  = new SoNurbsSurface;
33   complexity->value = 0.7;
34   controlPts->point.setValues(0, 16, pts);
35   surface->numUControlPoints = 4;
36   surface->numVControlPoints = 4;
37   surface->uKnotVector.setValues(0, 8, knots);
38   surface->vKnotVector.setValues(0, 8, knots);
39   surfSep->addChild(complexity);
40   surfSep->addChild(controlPts);
41   surfSep->addChild(surface);

42   surfSep->unrefNoDelete();
43   return surfSep;
44 }
```

linea 2 È necessaria una matrice di punti di controllo 4×4. Questi vengono qui specificati in un unico vettore.

linea 20 Il vettore `knots` ha 8 elementi, ed è utilizzato sia per il parametro u che per il parametro v.

linea 21 Si noti che è stata imposta una molteplicità 4 all'inizio ed alla fine della sequenza dei knot. La molteplicità massima dei knot è pari all'ordine. Ne segue che l'ordine è almeno quattro. Si vedrà che l'ordine è proprio 4. Imporre ad una NURBS una molteplicità quattro all'inizio ed alla fine della sequenza dei knot, con nessun nodo nel mezzo, equivale ad imporre che la curva risultante sia un Bezier cubica passante per gli estremi.

Analogamente qui imporre lo stesso vincolo su entrambi i parametri della superficie equivale ad imporre che la superficie passi per i punti di controllo "esterni" alla matrice definita alla linea 2.

linea 35 e 36 Il numero dei punti di controllo per entrambi i parametri u e v è quattro (in effetti sono condivisi). Se ne desume che l'ordine rispetto ad entrambi i parametri della superficie è 8 (numero dei knot) - 4 = 4.

linea 40 Il nodo di tipo `SoNurbsSurface` deve essere aggiunto al grafo della scena dopo il nodo di tipo `SoCoordinate3` contenente i punti di controllo. Durante il traversal del grafo della scena, i valori dei punti di controllo verranno raccolti nello stato del sistema, e poi interpretati nell'eseguire il rendering del nodo `SoNurbsSurface` grazie al valore dei suoi campi `numUControlPoints` e `numVControlPoints`.

4 Tagliare le superfici NURBS

In Open Inventor esiste la possibilità di ritagliare le superfici NURBS ed ottenere che parti di esse non siano rappresentate. Questo aiuta il programmatore a descrivere superfici che è più facile definire per differenza piuttosto che per giustapposizione di superfici diverse. Un uso piuttosto diffuso è quello di ritagliare i bordi di una superficie definita precedentemente, eliminando così gli spigoli vivi. L'operazione di "ritagliamento" viene detta in inglese "trimming", si basa sulla definizione di *profile curves* (cioè "curve profilo") che circoscrivono la zona da ritagliare. Attenzione dunque: si taglia una superficie con una curva. Il motivo di questa asimmetria è che non si va a tagliare la superficie nello spazio oggetto, cosa che sarebbe molto complessa, perché di fatto non si conoscono esplicitamente i punti per i quali la superficie passa, ma si taglia la superficie mediante la definizione di "zone proibite" nello spazio dei parametri (che come abbiamo visto è bidimensionale).

Le *profile curves* si comportano rispetto ai nodi separatori esattamente come le trasformazioni: entrando in un sottoalbero una copia dello stato (comprendente le profile curves correnti) viene memorizzata, per poi essere adottata di nuovo all'uscita del sottoalbero stesso. Mentre le trasformazioni vengono moltiplicate tra loro, le *profile curves* si accumulano nello stato, a formare una lista.

Una *profile curve* può essere lineare (`SoLinearProfile`), oppure una curva NURBS (`SoNurbsProfile`), o una combinazione delle due. Quanto alle coordinate essa può utilizzare sia `SoProfileCoordinate2` (per *profile curves* non razionali) o `SoProfileCoordinate3` (per *profile curves* razionali). La sola cosa indispensabile è che una *profile curve* composita costituisca un ciclo, con il primo punto ripetuto come ultimo punto. Inoltre il ciclo deve essere semplice, cioè non deve essere autointersecante.

La direzione usata nella definizione della sequenza dei punti di controllo di una *profile curve* è importante: se il ciclo è definito seguendo il senso orario l'area interna alla curva è scartata e l'area esterna è conservata. Altrimenti avviene esattamente l'inverso, e l'area esterna è conservata, mentre l'area contenuta nel ciclo viene eliminata. Niente vieta di annidare le *profile curves* una dentro l'altra, ma esse non si devono intersecare. Ovviamente ciò ha senso solamente per due *profile curves* di cui quella esterna è descritta in senso antiorario.

Nell'esempio che segue si eseguirà un "trimming" sulla superficie definita nell'esempio precedente

```
// The array of trim coordinates
const float tpts[12][2] = {
   {0.0, 0.0},
   {1.0, 0.0},
   {1.0, 1.0},
   {0.0, 1.0},
   {0.2, 0.2},
   {0.2, 0.7},
   {0.9, 0.7},
   {0.9, 0.2},
   {0.7, 0.0},
   {0.4, 0.8}};

// The 16 coordinates defining the Bezier surface.
const float pts[16][3] = {
   {-4.5, -2.0,  8.0},
   {-2.0,  1.0,  8.0},
   { 2.0, -3.0,  6.0},
   { 5.0, -1.0,  8.0},
   {-3.0,  3.0,  4.0},
   { 0.0, -1.0,  4.0},
   { 1.0, -1.0,  4.0},
   { 3.0,  2.0,  4.0},
   {-5.0, -2.0, -2.0},
   {-2.0, -4.0, -2.0},
   { 2.0, -1.0, -2.0},
   { 5.0,  0.0, -2.0},
   {-4.5,  2.0, -6.0},
   {-2.0, -4.0, -5.0},
   { 2.0,  3.0, -5.0},
   { 4.5, -2.0, -6.0}};

// The 3 knot vectors for the 3 trim curves.
float tknots1[7] = {0, 0, 1, 2, 3, 4, 4};
float tknots2[6] = {0, 0, 1, 2, 3, 3};
float tknots3[8] = {0, 0, 0, 0, 1, 1, 1, 1};

// The Bezier knot vector for the surface.
// This knot vector is used in both the U and
// V directions.
float knots[8] = {0, 0, 0, 0, 1, 1, 1, 1};

// Create the nodes needed for the Bezier patch
// and its trim curves.
SoSeparator *
makeSurface()
{
   SoSeparator *surfSep = new SoSeparator();
   surfSep->ref();

   // Define the Bezier surface including the control
   // points, trim curve, and a complexity.
   SoComplexity  *complexity     = new SoComplexity;
   SoCoordinate3 *controlPts     = new SoCoordinate3;
   SoNurbsSurface *surface       = new SoNurbsSurface;
   complexity->value = 0.7;
   controlPts->point.setValues(0, 16, pts);
   surface->numUControlPoints.setValue(4);
   surface->numVControlPoints.setValue(4);
   surface->uKnotVector.setValues(0, 8, knots);
```

```
56    surface->vKnotVector.setValues(0, 8, knots);
57    surfSep->addChild(complexity);
58    surfSep->addChild(controlPts);

59    SoProfileCoordinate2 *trimPts = new SoProfileCoordinate2;
60    SoNurbsProfile *nTrim1        = new SoNurbsProfile;
61    SoNurbsProfile *nTrim2        = new SoNurbsProfile;
62    SoNurbsProfile *nTrim3        = new SoNurbsProfile;
63    long trimInds[5];

64    trimPts->point.setValues(0, 12, tpts);
65    trimInds[0] = 0;
66    trimInds[1] = 1;
67    trimInds[2] = 2;
68    trimInds[3] = 3;
69    trimInds[4] = 0;
70    nTrim1->index.setValues(0, 5, trimInds);
71    nTrim1->knotVector.setValues(0, 7, tknots1);
72    trimInds[0] = 4;
73    trimInds[1] = 5;
74    trimInds[2] = 6;
75    trimInds[3] = 7;
76    nTrim2->linkage.setValue(SoProfile::START_NEW);
77    nTrim2->index.setValues(0, 4, trimInds);
78    nTrim2->knotVector.setValues(0, 6, tknots2);
79    trimInds[0] = 7;
80    trimInds[1] = 8;
81    trimInds[2] = 9;
82    trimInds[3] = 4;
83    nTrim3->linkage.setValue(SoProfile::ADD_TO_CURRENT);
84    nTrim3->index.setValues(0, 4, trimInds);
85    nTrim3->knotVector.setValues(0, 8, tknots3);

86    surfSep->addChild(trimPts);
87    surfSep->addChild(nTrim1);
88    surfSep->addChild(nTrim2);
89    surfSep->addChild(nTrim3);
90    surfSep->addChild(surface);

91    surfSep->unrefNoDelete();
92    return surfSep;
93 }
```

linea 2 L'array `tpts` (probabilmente sta per "trimming points") contiene tutti i punti necessari per definire le *profile curves*. Per pescare ogni volta i punti giusti nel seguito si userà il metodo degli indici: cioè la possibilità di isolare i valori di interesse all'interno dell'array mediante la specifica della loro posizione.

linea 34 I punti definiti in questa linea sono i knot di una NURBS. La molteplicità quattro all'inizio ed alla fine (con nessun altro punto nel mezzo) è una configurazione notevole: si può asserire che la curva sia una curva di Bezier cubica. Se ci fossero stati punti (equidistanziati) nel mezzo si sarebbe trattato di una *Uniform cubic B-spline* passante per gli estremi.

linea 58 Fin qui il codice (variabili globali a parte) è analogo a quello dell'esempio precedente.

linea 60 Si definiranno due *profile curves*: la prima semplice (`nTrim1`) e la seconda composita (`nTrim2` e `nTrim3`). La prima *profile curve* costituisce un rettangolo, che fungerà da curva esterna, percorso dunque in senso antiorario.

linea 65 La variabile `trimInds` (probabilmente sta per "trimming indexes") è stata definita alla linea 63, e contiene la posizione dei punti di controllo di interesse (per la definizione della curva corrente) all'interno del vettore dei punti `tpts` definito alla linea 2.

linea 69 La prima *profile curve* `nTrim1` ha cinque punti di controllo. I vertici del rettangolo sono ovviamente quattro, ma una *profile curve* deve avere necessariamente l'ultimo punto di controllo coincidente con il primo. Essendo il numero dei knot pari a 7 (linea 32), l'ordine di questa curva sarà pari a $7 - 5 = 2$, e dunque il grado è uno (dunque è lineare).

linea 75 `nTrim2` ha quattro punti di controllo. Essendo il numero dei knot pari a 6 (linea 33), l'ordine di questa curva sarà pari a $6 - 4 = 2$, e dunque il grado è uno. Quindi si tratta di una spezzata che passa per i quattro punti di controllo. L'ultimo punto non è coincidente con il primo, e dunque si tratta di una curva aperta. La curva `nTrim3` provvederà a chiudere il ciclo.

linea 76 L'istruzione di questa linea specifica ad OpenInventor che la curva che si sta per definire andrà a far parte di una nuova *profile curve*. Si confronti questa linea con la linea 83.

linea 79 `nTrim3` ha quattro punti di controllo. Il numero dei knot questa volta è pari a 8 (linea 34). Ciò comporta un ordine pari a $8 - 4 = 4$, e dunque un grado 3 per la curva (che è dunque una cubica).

linea 76 L'istruzione di questa linea specifica ad OpenInventor che la curva che si sta per definire deve essere considerata un completamento della *profile curve* la cui definizione è tuttora in corso.

linea 86 Il nodo `SoProfileCoordinate2` deve precedere i nodi `SoNurbsProfile` che definiscono le *profile curves*. Analogamente il nodo `SoCoordinate3` che fornisce i punti di controllo della superficie (linea 58) deve precedere il nodo *SoNurbsSurface* che definisce la superficie.

Si rammenta, per concludere, che l'uso delle superfici e il "trimming" delle stesse, è generalmente considerato nei manuali di Open Inventor come "argomento avanzato", a testimoniare la possibile insorgenza di problemi dovuti all'approccio *non friendly* della libreria, ed alla grande quantità di considerazioni matematiche e geometriche che sono implicite nella teoria.

Per questo motivo si consiglia un approccio per gradi, cominciando da esperimenti semplici, e preparandosi ad un certo numero di tentativi infruttuosi dovuti ad equivoci, lacune nel proprio background matematico-geometrico o quant'altro.