UNIVERSITÀ DEGLI STUDI
DI TORINO

FACOLTÀ DI SCIENZE M.F.N.

Corso di Laurea in Matematica

Algoritmo di Chaikin e sue evoluzioni

Relatore:
Prof. Catterina Dagnino
Correlatore:
Dott. Sara Remogna

Candidato:
Fabio Roman

Ottobre 2009

Indice

1 Introduzione

Una curva polinomiale o polinomiale a tratti può essere costruita sia come somma pesata di punti di controllo mediante l'impiego di opportune funzioni di miscelamento, sia con algoritmi detti di suddivisione (o di raffinamento), che permettono di generare, tra le altre, curve di Bézier e curve B-spline, tramite un processo detto di taglio degli angoli, per cui, a partire per esempio dal poligono di controllo originario, passo dopo passo si ottiene una curva poligonale che approssima la curva cercata, fino ad ottenere, in un numero finito di passi, oppure come limite di una successione di curve, la curva desiderata.

Nel 1974, George Chaikin ebbe l'idea di costruire una curva regolare a partire da un numero ridotto di punti di controllo tramite passi di raffinamento. Il principio del metodo di Chaikin è quello di partire da un insieme dato di punti di controllo $\mathbf{P}_i^0$, iterare un procedimento che risulti in un nuovo insieme di punti $\{\mathbf{P}_i^1\}$, e ripeterlo, producendo ad ogni passo k un ulteriore insieme di punti $\{\mathbf{P}_i^k\}$; in questo modo, il poligono di controllo originale viene raffinato di volta in volta.
Nella sua formulazione originaria, l'algoritmo di Chaikin veniva posto come un veloce algoritmo per la generazione di curve arbitrarie. La sua ricorsività, e il fatto che dal punto di vista computazionale utilizzasse solamente operazioni elementari, lo rendevano particolarmente adatto per la rappresentazione di curve, ma anche per l'interpolazione non lineare e per la direzione di macchine a controllo numerico nell'industria.

L'algoritmo di Chaikin è oggi alla base di molte tecniche CAGD più avanzate e correntemente utilizzate nel campo della ricerca scientifica e nell'attività produttiva.
Dagli anni '70 in poi sono stati sviluppati ulteriori metodi di suddivisione che a seconda delle necessità hanno perfezionato la tecnica originaria di Chaikin. In questa tesi presenteremo, unitamente al lavoro originale di Chaikin e alle sue applicazioni, anche alcuni suoi sviluppi, ottenuti, per esempio, per permettere la costruzione di circonferenze esatte, il che non è direttamente possibile con l'algoritmo di Chaikin, o di migliorare la velocità computazionale, dal momento che le applicazioni grafiche odierne sono molto più avanzate di quelle dei primi computer e richiedono un costo computazionale ben maggiore per fare fronte al quale potrebbe non essere sufficiente tutto il seppur enorme progresso che l'hardware ha avuto in questi ultimi trent'anni.
In particolare, nel capitolo 1 parleremo dell'algoritmo di Chaikin in sè, nel capitolo 2 tratteremo il suo uso per la generazione di curve B-spline, mentre argomento del capitolo 3 sarà un algoritmo di suddivisione multipasso che migliora quello originario. Infine, nel capitolo 4, tratteremo l'algoritmo di suddivisione razionale di Nasri-Farin e una sua evoluzione.

2 L'algoritmo di Chaikin

Chaikin considerava una curva come descritta da quattro punti P1, P2, P3, P4, dove i punti sono rappresentati da coppie di interi che stanno a specificare le loro coordinate. Come si può vedere in figura 1, la curva generata inizia in P1 ed è tangente alla retta passante per P1 e P2; interseca il punto medio del segmento tra P2 e P3 in modo tale da essere tangente ad esso; e termina in P4, tangente alla retta passante per P3 e P4. L'algoritmo può essere visto geometricamente, come la divisione della spezzata di tre segmenti descritta dai quattro punti presi nell'ordine, in due spezzate di due segmenti operando un taglio nel punto medio del segmento tra P2 e P3, che chiameremo M23; dopodiché, si considerino i punti medi tra P1 e P2 (M12), quello tra P2 e M23 (M223), quello tra M23 e P3 (M233), quello tra P3 e P4 (M34), e con questi si formi un nuovo quadrilatero. Questo procedimento viene ripetuto fino a quando i due punti del triangolo che stanno sulla curva occupano posizioni adiacenti. La curva si ottiene ora come insieme di tutti i vettori che uniscono tali punti, e risulta essere una curva B-spline quadratica. Estendendo e adattando questo procedimento, si possono ottenere curve B-spline di grado superiore e in particolare curve di Bézier.

2.1 Come fu presentato originariamente

Nell'articolo originario, comparso su [1] nel 1974, un'eternità fa se si considera la rapida evoluzione dell'informatica avvenuta negli ultimi decenni, ma relativamente poco tempo fa se si tiene conto dell'evoluzione secolare della matematica, l'algoritmo veniva presentato sotto questa forma:

- Si inseriscano i punti P4 e P3 in una pila (stack);
- Si definisca un nuovo P4 come punto medio del segmento tra P2 e P3, ovvero si ponga P4 = $\frac{1}{2} * (P2 + P3)$;
- Si effettui un test sulla distanza tra P1 e P4; se questa è maggiore di $2 * \sqrt{2}$, allora si ridefinisca P2 come $\frac{1}{2} * (P2 + P1)$, e P3 come $\frac{1}{2} * (P2 + P4)$, ripetendo il procedimento da capo fino a quando la distanza non risulti minore o uguale a $2 * \sqrt{2}$;
- Giunti alla situazione nella quale la distanza non sia maggiore di $2 * \sqrt{2}$, si tracci il segmento tra P1 e P4, si ridefinisca P1 come P4, e si svuoti la pila, chiamando i punti *poppati* P2 e P4;
- Si ritorni al test sulla distanza. Quando, ad un certo punto nel suo svolgimento, la risposta al test è no in un momento in cui la pila è vuota, l'algoritmo termina dopo l'ultima tracciatura del segmento tra P1 e P4 (e la conseguente riassegnazione di P1 che però a quel punto non fa più nulla di utile).

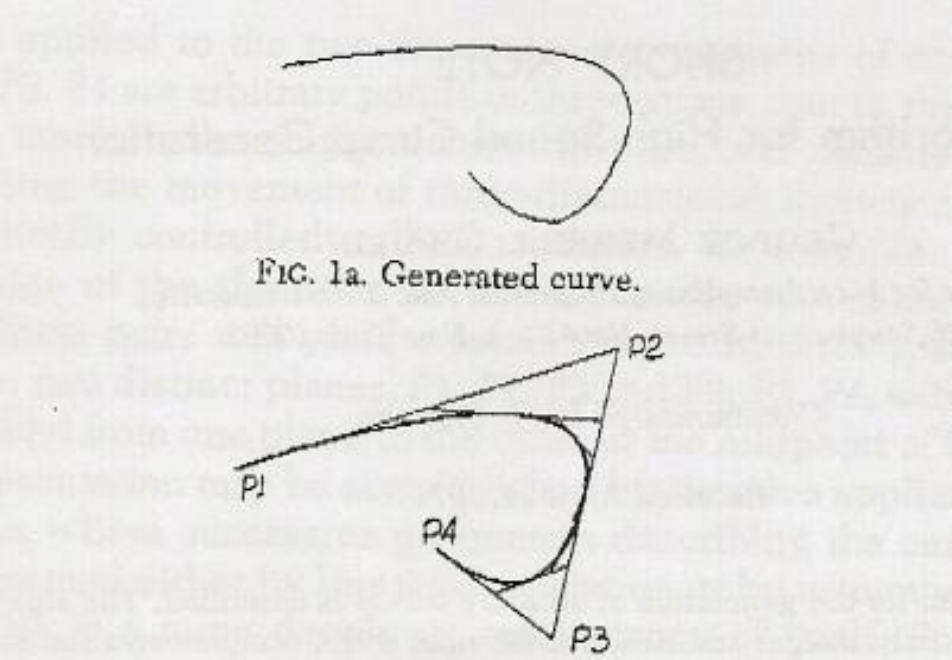

Figura 1: Un esempio molto semplice di curva generata con l'algoritmo di Chaikin. A partire da quattro punti, si generano successivamente quadrilateri, ottenendo al limite una curva regolare. [1]

Notare che il test sulla distanza, effettuato confrontando con $2 * \sqrt{2}$, è equivalente a dire che sia la coordinata x che la coordinata y devono differire meno (o non meno) di tre. Il motivo per cui il confronto viene effettuato con tre e non con due è dovuto all'aritmetica di macchina e più precisamente alla divisione tra interi: poiché questa tronca il risultato all'intero più vicino, procedendo in funzione di tale condizione, sarebbe possibile che P1 e P4 diventino identici. Nel caso in cui questo succeda, l'assegnazione P1 $\leftarrow$ P4 non farebbe nulla, e il diagramma di flusso mostra che questo farebbe risultare un loop che svuota le pile senza fare nulla di utile.

Curve più complesse di quelle descritte da quattro punti possono essere ottenute in due modi.
La prima soluzione è quella di dare direttamente i punti per generare i tratti di curva seguenti a quello generato dai primi quattro; si noti che, poiché al termine dell'esecuzione dell'algoritmo, l'ultimo punto è chiamato P1, è necessario e sufficiente dare soltanto tre punti (P2, P3, P4) al fine di generare

ogni tratto seguente reiterando l'algoritmo.
La seconda soluzione, meno scontata e che può causare qualche perdita di generalità, è talvolta preferibile grazie alla sua migliore velocità computazionale. In questo caso, per descrivere ogni segmento successivo servono due soli punti, che vengono preinseriti nella pila in ordine inverso prima che la routine sia chiamata, in modo tale che, quando il primo segmento è stato completato, l'algoritmo trova ancora punti nella pila, e continua.
In entrambi i casi, due segmenti concatenati si uniscono in maniera regolare (almeno C^1) se il primo punto che descrive il segmento successivo sta sulla retta passante per i punti P3 e P4 del segmento precedente.

Le uniche operazioni richieste da questo algoritmo sono la somma tra interi, la divisione per 2 (che per un calcolatore corrisponde ad uno spostamento verso destra di un bit), funzioni di complementazione, e confronti. Dal punto di vista hardware, sono richiesti otto registri per i quattro punti (le due coordinate di ogni punto), un registro di indici per la memoria della pila (stack), e la memoria della pila stessa. Un *raster* 1024x1024 richiede quattro pile *ten-word* (cioè da 10*16 = 160 bit), con una *word* (16 bit) addizionale in ogni pila per ogni tratto di curva in più nel caso in cui si intraprenda la strada del preinserimento.
Dal punto di vista della velocità, sono richieste n iterazioni dell'algoritmo per generare $n + 2$ punti (il punto iniziale, n punti intermedi, e il punto finale). Considerando un'implementazione hardware in grado di compiere calcolo parallelo, questo significa che sono necessari circa 15 cicli di istruzioni per punto.

Questo algoritmo, bidimensionale, può essere applicato anche alle proiezioni bidimensionali di curve i cui punti P1, P2, P3, P4 sono arbitrari nello spazio tridimensionale, nel senso che non necessitano di essere complanari. Una delle questioni aperte dell'epoca era però quella che risultava considerando la direzione di macchinari a controllo numerico: in questo caso, Chaikin propose una diretta estensione del suo algoritmo, in modo tale che potessero essere gestite terne piuttosto che coppie di (interi a rappresentazione di) coordinate, generando così una curva regolare nello spazio tridimensionale, ma contenuta in due piani, il primo quello generato da P1, P2, P3, il secondo quello generato da P2, P3, P4, con la curva che passa in maniera regolare da un piano all'altro in corrispondenza del punto medio del segmento P2, P3. Analoghe modifiche venivano proposte per la rappresentazione di superfici piuttosto che di curve.

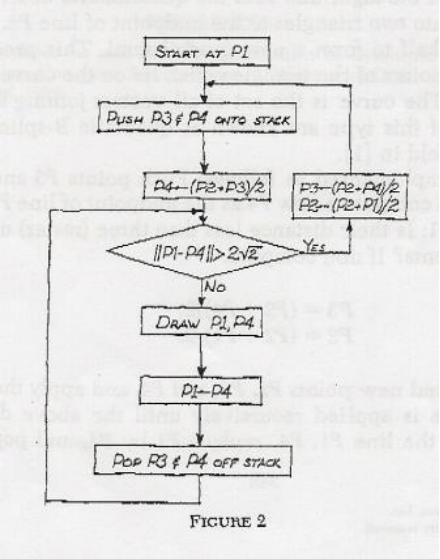

Figura 2: L'algoritmo di Chaikin sotto forma di diagramma di flusso. [1]

2.2 Formulazione attuale

Al giorno d'oggi, l'algoritmo viene generalmente descritto in maniera differente da quanto visto nella pubblicazione originaria di Chaikin. Abbiamo visto che l'idea generale è quella di partire da un insieme dato di punti di controllo $\mathbf{P}_i^0$, iterare un procedimento che risulti in un nuovo insieme di punti $\mathbf{P}_i^1$, e ripetere il procedimento, producendo di volta in volta ulteriori insiemi di punti $\mathbf{P}_i^k$. In altre parole, le sequenze di punti sono di volta in volta le seguenti:

$$\begin{gathered}\mathbf{P}_0^0, \mathbf{P}_1^0, \mathbf{P}_2^0, ..., \mathbf{P}_{n_0}^0 \\ \mathbf{P}_0^1, \mathbf{P}_1^1, \mathbf{P}_2^1, ..., \mathbf{P}_{n_1}^1 \\ \mathbf{P}_0^2, \mathbf{P}_1^2, \mathbf{P}_2^2, ..., \mathbf{P}_{n_2}^2 \\ ... \\ \mathbf{P}_0^k, \mathbf{P}_1^k, \mathbf{P}_2^k, ..., \mathbf{P}_{n_k}^k\end{gathered}$$

Ogni punto $\mathbf{P}_j^k$ viene calcolato come somma pesata dei punti $\mathbf{P}_i^{k-1}$, in altre parole:

$$\mathbf{P}_j^k = \sum_{i=0}^{n_{k-1}} a_{ijk} * \mathbf{P}_i^{k-1}$$

dove gli a_{ijk} sono coefficienti reali. Ogni iterazione produce un numero di punti $n_k + 1$ che è in generale differente da iterazione a iterazione. Se n_k decresce in maniera stretta al crescere di k, allora il numero di punti diventa

sempre più piccolo, fino a quando rimane un solo punto. Un esempio di questo caso è dato dall'algoritmo di de Casteljau, che fissati opportunamente i coefficienti di cui sopra, produce un singolo punto di una curva di Bézier (e infatti la curva intera si ottiene variando opportunamente i coefficienti). D'altra parte, è possibile che n_k cresca all'aumentare di k, producendo di volta in volta più e più punti ad ogni iterazione. A questo punto, ottenuto un numero sufficiente di punti a soddisfare le nostre condizioni di precisione, si traccia la curva congiungendo nell'ordine i punti tramite segmenti.
L'algoritmo di Chaikin può essere visto come caso particolare di questo modo di procedere.
Si parta infatti dagli $n+1$ punti di controllo che denotiamo come $\mathbf{P}_0^0$, $\mathbf{P}_1^0$, ..., $\mathbf{P}_n^0$ e si applichi la seguente regola di raffinamento, come visibile in figura 3, tratta da [5].

$$\mathbf{P}_{2j}^{k+1} = \tfrac{3}{4}\mathbf{P}_j^k + \tfrac{1}{4}\mathbf{P}_{j+1}^k$$
$$\mathbf{P}_{2j+1}^{k+1} = \tfrac{1}{4}\mathbf{P}_j^k + \tfrac{3}{4}\mathbf{P}_{j+1}^k$$

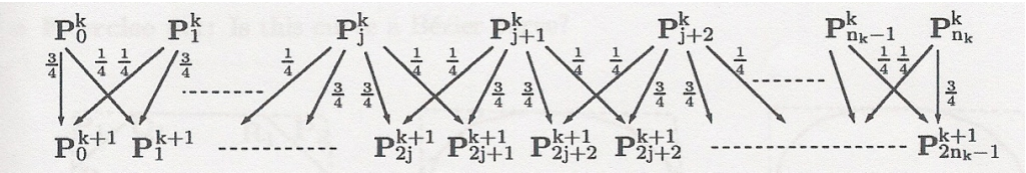

Figura 3: Regola di raffinamento dell'algoritmo di Chaikin.

La prima iterazione inizia con gli $n+1$ punti originali producendo i $2n$ punti $\mathbf{P}_i^1$. Ogni iterazione seguente aumenta il numero di punti portandoli più vicini alla posizione limite che consiste nella posizione teorica della curva. Dopo k iterazioni, con k che dipende dalla precisione richiesta, la curva viene rappresentata tracciando segmenti tra i punti prodotti nell'ultima iterazione.
Una semplice interpretazione geometrica di questo fatto è la seguente: consideriamo per esempio un poligono di controllo composto da cinque punti, la regola di raffinamento sarà eseguita prendendo, per ogni i, il segmento tra $\mathbf{P}_i$ e $\mathbf{P}_{i+1}$, chiamando $\mathbf{Q}_i$ e $\mathbf{R}_i$ i punti che stanno rispettivamente a un quarto e a tre quarti di distanza da $\mathbf{P}_i$ sul segmento in questione (cioè in modo tale che, posto $\mathbf{M}_i$ il punto medio del segmento, i punti $\mathbf{P}_i$, $\mathbf{Q}_i$, $\mathbf{M}_i$, $\mathbf{R}_i$, $\mathbf{P}_{i+1}$ siano allineati, consecutivi e tali per cui ognuno disti dal precedente e dal successivo un quarto della lunghezza del segmento), e osservando che valgono le seguenti combinazioni lineari che sono anche baricentriche e convesse:

$$\mathbf{Q}_i = \tfrac{3}{4}\mathbf{P}_i + \tfrac{1}{4}\mathbf{P}_{i+1}$$
$$\mathbf{R}_i = \tfrac{1}{4}\mathbf{P}_i + \tfrac{3}{4}\mathbf{P}_{i+1}$$

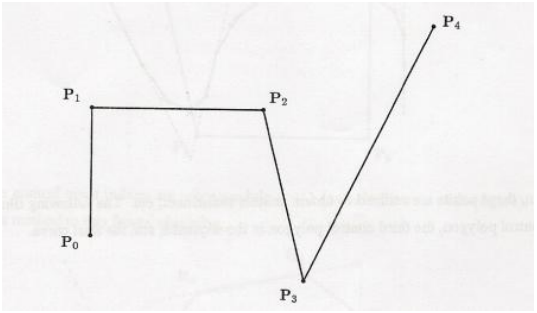

Figura 4: Poligono di controllo aperto.

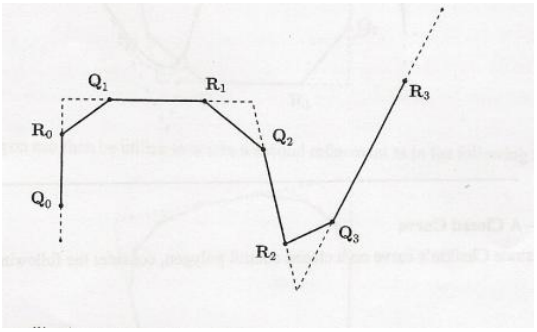

Figura 5: Primo passo dell'algoritmo di suddivisione di Chaikin.

In questo modo, partendo da $n+1$ punti di controllo che definiscono un poligono di controllo composto da n lati, arriviamo ad avere $2n$ nuovi punti $\mathbf{Q}_i$ e $\mathbf{R}_i$. Questi possono ora essere collegati in modo da formare un nuovo poligono di controllo composto da $2n-1$ lati. Ripetendo questo procedimento, i poligoni di controllo che via via possono essere formati si avvicinano sempre più alla posizione limite che consiste nella posizione della curva di Chaikin.
Notare inoltre che gli $\mathbf{M}_i$ stanno tutti sulla curva; non solo, i punti medi di tutti i segmenti generati nei successivi passi dell'algoritmo di raffinamento stanno sulla curva. Questa curva risulta essere una curva di Bézier, anche se definita da altri punti di controllo rispetto a quelli utilizzati nei passi dell'algoritmo, dal momento che essa non passa per il primo e per l'ultimo punto di controllo.

L'algoritmo può essere facilmente adattato per la generazione di curve chiuse: l'unica modifica necessaria è quella di collegare il primo e l'ultimo punto $\mathbf{P}_0$ e $\mathbf{P}_n$ e calcolarsi con essi $\mathbf{Q}_n$ e $\mathbf{R}_n$. Ciò può essere svolto in maniera del tutto naturale definendo $\mathbf{P}_{n+1}$ come duplicato di $\mathbf{P}_0$, analogamente a come si fa in generale quando si trattano curve chiuse; nelle figure 8, 9, 10, 11 è visibile il risultato dell'applicazione dell'algoritmo in tale evenienza.

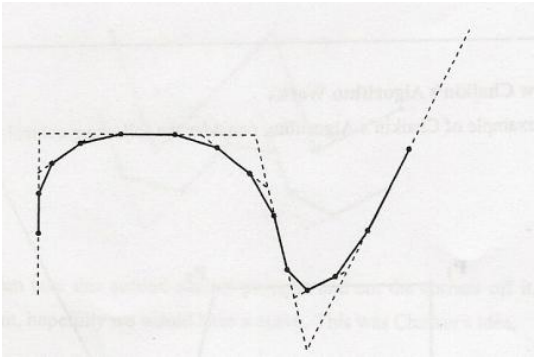

Figura 6: Secondo passo dell'algoritmo di suddivisione di Chaikin.

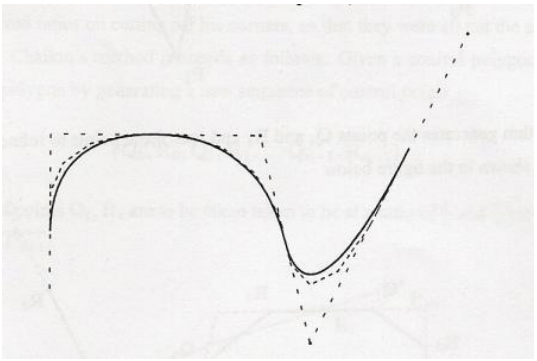

Figura 7: Successione di poligoni di controllo e curva limite.

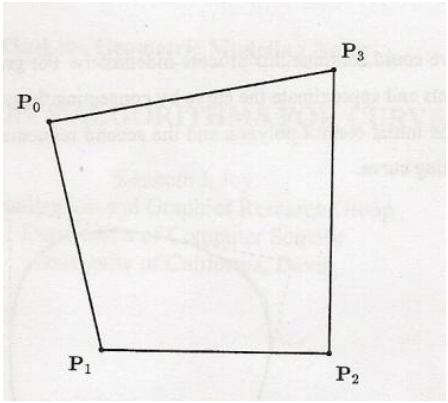

Figura 8: Poligono di controllo chiuso.

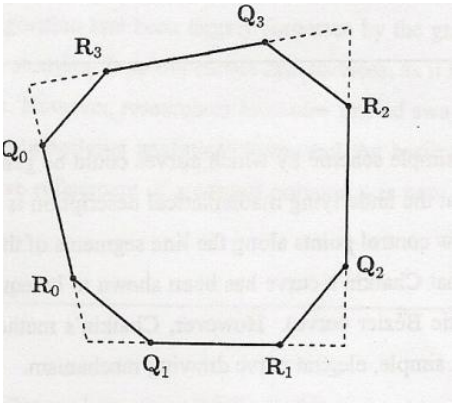

Figura 9: Primo passo nel caso di poligono chiuso.

2.3 Una procedura `MATLAB` che implementa l'algoritmo

Presentiamo ora una procedura `MATLAB` che implementa l'algoritmo di Chaikin. In input viene fornita la matrice dei punti che formano il poligono di controllo originario, dove ogni riga rappresenta un punto, e il numero di iterazioni k che si vogliono effettuare, mentre in output vengono restituiti la matrice dei punti che formano il poligono di controllo finale, ovvero l'approssimazione della curva limite, e un grafico nel quale vengono rappresentati il poligono di controllo iniziale e quello ottenuto.

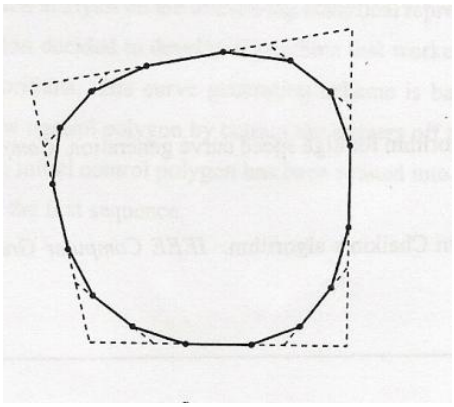

Figura 10: Secondo passo nel caso di poligono chiuso.

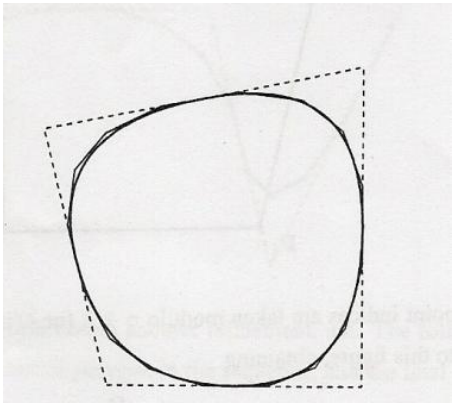

Figura 11: Successione di poligoni di controllo e curva limite, caso chiuso.

```
function Pk = chaikin(P,k)
%Applicazione classica dell'algoritmo di Chaikin.
%Parametri di input: la matrice dei punti iniziali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z), e il passo k.
%Parametri di output: la matrice dei punti finali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z) e un grafico 3D, che nel
%caso in cui tutte le coordinate z siano nulle viene plottato come 2D.
P0 = P; %teniamoci da parte il poligono originale per il grafico
n=size(P,1)-1;
for j=1:k %ci sono variazioni rispetto alla notazione originaria
    for i=1:n %dovute al fatto che Matlab indicizza da 1 e non da 0.
        Q(i,:) = (3/4)*P(i,:) + (1/4)*P(i+1,:);
        R(i,:) = (1/4)*P(i,:) + (3/4)*P(i+1,:);
    end
    for i=1:n
```

```
        P(2*i-1,:) = Q(i,:);
        P(2*i,:) = R(i,:);
    end
    n=size(P,1)-1; %bisogna ridefinire n poiché P cambia dimensione
end
Pk = P
if(norm(P0(:,3)) ~= 0) %equivalente a dire che tutte le coordinate z = 0
    plot3(P0(:,1),P0(:,2),P0(:,3),'o--')
    hold on
    plot3(Pk(:,1),Pk(:,2),Pk(:,3),'ro-')
else
    plot(P0(:,1),P0(:,2),'o--')
    hold on
    plot(Pk(:,1),Pk(:,2),'ro-')
end
```

Vediamo ora l'applicazione dell'algoritmo ad un poligono di controllo di prova: il poligono in questione è aperto e formato da 6 punti generati con casualità uniforme nel quadrato di vertici $(0,0)$, $(0,1)$, $(1,1)$, $(1,0)$, tramite la funzione `rand` di `MATLAB`. Nell'output, i punti sono stati raccolti per colonne per una questione di ottimizzazione di spazio della pagina. Altri esempi verranno mostrati nel paragrafo 3.4, quando questo algoritmo verrà confrontato con una sua evoluzione, più recente e più efficiente dal punto di vista computazionale.

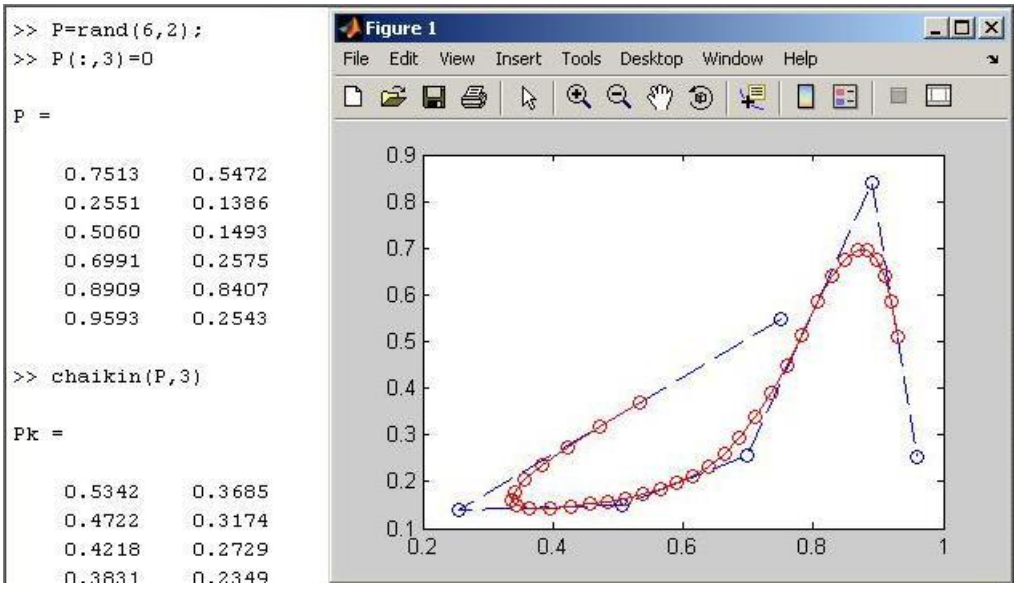

Figura 12: Applicazione della *function* `MATLAB` che svolge l'algoritmo di Chaikin su un poligono di controllo di prova.

```
>> P=rand(6,2);
>> P(:,3)=0;

P =
 0.7513    0.5472      0
 0.2551    0.1386      0
 0.5060    0.1493      0
 0.6991    0.2575      0
 0.8909    0.8407      0
 0.9593    0.2543      0

>> chaikin(P,3)

Pk =

0.5342    0.3684      0  | 0.4562    0.1519      0 | 0.7350    0.3891      0
0.4722    0.3174      0  | 0.4849    0.1578      0 | 0.7590    0.4472      0
0.4218    0.2728      0  | 0.5127    0.1652      0 | 0.7830    0.5126      0
0.3832    0.2349      0  | 0.5395    0.1742      0 | 0.8070    0.5855      0
0.3562    0.2034      0  | 0.5654    0.1846      0 | 0.8290    0.6402      0
0.3408    0.1786      0  | 0.5905    0.1966      0 | 0.8492    0.6765      0
0.3372    0.1603      0  | 0.6146    0.2102      0 | 0.8673    0.6946      0
0.3452    0.1485      0  | 0.6387    0.2311      0 | 0.8836    0.6944      0
0.3649    0.1433      0  | 0.6628    0.2595      0 | 0.8979    0.6759      0
0.3962    0.1446      0  | 0.6869    0.2953      0 | 0.9103    0.6392      0
0.4267    0.1475      0  | 0.7110    0.3385      0 | 0.9208    0.5842      0
                         |                         | 0.9294    0.5109      0
```

3 Uso dell'algoritmo di Chaikin per la generazione di curve B-spline

3.1 Curve B-spline quadratiche tramite suddivisione

La curva B-spline quadratica uniforme determinata da una sequenza di $n+1$ punti di controllo può essere generata come insieme di sezioni, ognuna delle quali sia un polinomio di secondo grado basato su tre punti di controllo. In questo frangente vedremo come l'algoritmo di Chaikin, ed in particolare la sua formulazione attuale, possa essere applicato per costruire una tale curva. Consideriamo la divisione degli $n+1$ punti di controllo di partenza in $n-1$ sequenze sovrapposte di tre punti di controllo l'una, utilizzando poi ogni sequenza per calcolare quattro nuovi punti. Le sequenze sono:

$$\mathbf{P}_0, \mathbf{P}_1, \mathbf{P}_2 \ ; \ \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3 \ ; \ \ldots \ ; \ \mathbf{P}_{n-2}, \mathbf{P}_{n-1}, \mathbf{P}_n$$

L'applicazione dell'algoritmo alla prima sequenza porta ad avere i quattro seguenti nuovi punti:

$$\begin{aligned}
\mathbf{P}_0^1 &= \tfrac{3}{4}\mathbf{P}_0 + \tfrac{1}{4}\mathbf{P}_1 \\
\mathbf{P}_1^1 &= \tfrac{1}{4}\mathbf{P}_0 + \tfrac{3}{4}\mathbf{P}_1 \\
\mathbf{P}_2^1 &= \tfrac{3}{4}\mathbf{P}_1 + \tfrac{1}{4}\mathbf{P}_2 \\
\mathbf{P}_3^1 &= \tfrac{1}{4}\mathbf{P}_1 + \tfrac{3}{4}\mathbf{P}_2
\end{aligned}$$

In maniera analoga si applica poi l'algoritmo alla seconda sequenza, portando ad avere:

$$\begin{aligned}
\mathbf{P}_2^1 &= \tfrac{3}{4}\mathbf{P}_1 + \tfrac{1}{4}\mathbf{P}_2 \\
\mathbf{P}_3^1 &= \tfrac{1}{4}\mathbf{P}_1 + \tfrac{3}{4}\mathbf{P}_2 \\
\mathbf{P}_4^1 &= \tfrac{3}{4}\mathbf{P}_2 + \tfrac{1}{4}\mathbf{P}_3 \\
\mathbf{P}_5^1 &= \tfrac{1}{4}\mathbf{P}_2 + \tfrac{3}{4}\mathbf{P}_3
\end{aligned}$$

I primi due punti sono stati già ottenuti dall'applicazione precedente, gli ultimi due invece non sono stati ottenuti prima e sono pertanto nuovi. In generale, l'applicazione dell'algoritmo alla i-esima sequenza porta ad avere:

$$\begin{aligned}
\mathbf{P}_{2i-2}^1 &= \tfrac{3}{4}\mathbf{P}_{i-1} + \tfrac{1}{4}\mathbf{P}_i \\
\mathbf{P}_{2i-1}^1 &= \tfrac{1}{4}\mathbf{P}_{i-1} + \tfrac{3}{4}\mathbf{P}_i \\
\mathbf{P}_{2i}^1 &= \tfrac{3}{4}\mathbf{P}_i + \tfrac{1}{4}\mathbf{P}_{i+1} \\
\mathbf{P}_{2i+1}^1 &= \tfrac{1}{4}\mathbf{P}_i + \tfrac{3}{4}\mathbf{P}_{i+1}
\end{aligned}$$

Si noti che, ad ogni applicazione alla sequenza successiva, soltanto due dei quattro punti generati sono nuovi, dal momento che gli altri due punti sono già stati generati nella sequenza precedente.
Il procedimento viene ora reiterato sui segmenti definiti dai punti $\mathbf{P}_i^1$, portando ad avere nuovi punti $\mathbf{P}_i^2$ che definiscono nuovi segmenti. Quando il numero di punti è sufficientemente grande, la curva può essere tracciata

collegando ogni coppia di punti adiacenti tramite un segmento. La curva limite di questo procedimento è infatti la curva B-spline quadratica uniforme avente come punti di controllo gli $n+1$ punti iniziali, come mostrato da Riesenfeld (si veda [5]).

3.2 Curve B-spline cubiche tramite suddivisione

Il procedimento visto sopra può essere esteso e adattato per la generazione di curve B-spline cubiche. Mostreremo come i metodi di Chaikin possano essere applicati alla costruzione di una curva B-spline cubica uniforme basata su di un insieme di $n+1$ punti di controllo $\mathbf{P}_i$. Questi punti vengono divisi in sequenze sovrapposte di quattro punti ciascuna, e ogni sequenza viene utilizzata per calcolare tramite raffinamento un polinomio cubico che diventa un tratto dell'intera curva. Questi tratti cubici hanno continuità C^2 nei punti di passaggio da uno all'altro. Dal momento che il raffinamento è un procedimento iterativo, può essere utile identificare i punti di controllo ottenuti al passo k del processo di suddivisione come $\mathbf{P}_i^k$; di conseguenza, denotiamo i punti di controllo originari come $\mathbf{P}_i^0$. Questi vengono divisi nelle seguenti sequenze (che si sovrappongono):

$$\mathbf{P}_0^0, \mathbf{P}_1^0, \mathbf{P}_2^0, \mathbf{P}_3^0 \ ; \ \mathbf{P}_1^0, \mathbf{P}_2^0, \mathbf{P}_3^0, \mathbf{P}_4^0 \ ; \ \ldots \ ; \ \mathbf{P}_{n-3}^0, \mathbf{P}_{n-2}^0, \mathbf{P}_{n-1}^0, \mathbf{P}_n^0$$

Applichiamo ora il primo passo di raffinamento alla prima sequenza, calcolando i cinque punti $\mathbf{P}_i^1$, $i = 0, ..., 4$, nel seguente modo (in figura 13, tratta da [5], è riportato uno schema che illustra questo algoritmo):

$$\begin{aligned}
\mathbf{P}_0^1 &= \tfrac{1}{2}\mathbf{P}_0^0 + \tfrac{1}{2}\mathbf{P}_1^0 \\
\mathbf{P}_1^1 &= \tfrac{1}{8}\mathbf{P}_0^0 + \tfrac{6}{8}\mathbf{P}_1^0 + \tfrac{1}{8}\mathbf{P}_2^0 \\
\mathbf{P}_2^1 &= \tfrac{1}{2}\mathbf{P}_1^0 + \tfrac{1}{2}\mathbf{P}_2^0 \\
\mathbf{P}_3^1 &= \tfrac{1}{8}\mathbf{P}_1^0 + \tfrac{6}{8}\mathbf{P}_2^0 + \tfrac{1}{8}\mathbf{P}_3^0 \\
\mathbf{P}_4^1 &= \tfrac{1}{2}\mathbf{P}_2^0 + \tfrac{1}{2}\mathbf{P}_3^0
\end{aligned}$$

La scelta dei coefficienti è svolta in modo tale da far valere per i cinque punti risultanti le seguenti proprietà:

- Ognuno dei tre punti con indice pari, ovvero $\mathbf{P}_0^1$, $\mathbf{P}_2^1$, $\mathbf{P}_4^1$, viene posto al centro di un segmento delimitato da due dei punti di controllo originari. In questo caso, $\mathbf{P}_0^1$ sta a metà tra $\mathbf{P}_0^0$ e $\mathbf{P}_1^0$, così come $\mathbf{P}_2^1$ è punto medio del segmento tra $\mathbf{P}_1^0$ e $\mathbf{P}_2^0$, e altrettanto $\mathbf{P}_4^1$ si trova ad essere a metà strada tra $\mathbf{P}_2^0$ e $\mathbf{P}_3^0$. Questi punti vengono denominati *edge points* o *punti di spigolo*.
- Ognuno dei due punti con indice dispari, ovvero $\mathbf{P}_1^1$ e $\mathbf{P}_3^1$, viene posto al centro di un segmento le cui estremità sono localizzate nei centri di due segmenti delimitati ognuno da un nuovo punto di spigolo e da un punto di controllo originario. In questo caso, $\mathbf{P}_1^1$ si trova al centro del segmento le cui estremità sono localizzate in corrispondenza dei centri dei due segmenti $\mathbf{P}_0^1\mathbf{P}_1^0$ e $\mathbf{P}_1^0\mathbf{P}_2^1$, e analogamente $\mathbf{P}_3^1$ sta nella stessa situazione relativamente

ai due segmenti $\mathbf{P}_2^1\mathbf{P}_2^0$ e $\mathbf{P}_2^0\mathbf{P}_4^1$.

Si nota che ognuno dei punti del nuovo passo $\mathbf{P}_i^1$ è calcolato a partire da due o tre dei punti del passo precedente $\mathbf{P}_j^0$. I cinque nuovi punti vengono poi divisi in due gruppi sovrapposti, il primo che contiene i punti per i che va da 0 a 3, il secondo per i da 1 a 4, e il procedimento viene riapplicato ad ogni gruppo, in modo tale da produrre cinque nuovi punti per ogni gruppo denotati con $\mathbf{P}_i^2$. Alcuni di questi punti sono però identici tra di loro poiché si verificano casi (tre, per la precisione) di calcoli nel secondo gruppo già effettuati nel primo gruppo, per cui il numero di punti effettivamente distinti tra di loro è $2 * 5 - 3 = 7$. Infatti:

$$\begin{aligned}
\mathbf{P}_0^2 &= \tfrac{1}{2}\mathbf{P}_0^1 + \tfrac{1}{2}\mathbf{P}_1^1 \\
\mathbf{P}_1^2 &= \tfrac{1}{8}\mathbf{P}_0^1 + \tfrac{6}{8}\mathbf{P}_1^1 + \tfrac{1}{8}\mathbf{P}_2^1 \\
\mathbf{P}_2^2 &= \tfrac{1}{2}\mathbf{P}_1^1 + \tfrac{1}{2}\mathbf{P}_2^1 \\
\mathbf{P}_3^2 &= \tfrac{1}{8}\mathbf{P}_1^1 + \tfrac{6}{8}\mathbf{P}_2^1 + \tfrac{1}{8}\mathbf{P}_3^1 \\
\mathbf{P}_4^2 &= \tfrac{1}{2}\mathbf{P}_2^1 + \tfrac{1}{2}\mathbf{P}_3^1
\end{aligned}$$

$$\begin{aligned}
\mathbf{P}_2^2 &= \tfrac{1}{2}\mathbf{P}_1^1 + \tfrac{1}{2}\mathbf{P}_2^1 \\
\mathbf{P}_3^2 &= \tfrac{1}{8}\mathbf{P}_1^1 + \tfrac{6}{8}\mathbf{P}_2^1 + \tfrac{1}{8}\mathbf{P}_3^1 \\
\mathbf{P}_4^2 &= \tfrac{1}{2}\mathbf{P}_2^1 + \tfrac{1}{2}\mathbf{P}_3^1 \\
\mathbf{P}_5^2 &= \tfrac{1}{8}\mathbf{P}_2^1 + \tfrac{6}{8}\mathbf{P}_3^1 + \tfrac{1}{8}\mathbf{P}_4^1 \\
\mathbf{P}_6^2 &= \tfrac{1}{2}\mathbf{P}_3^1 + \tfrac{1}{2}\mathbf{P}_4^1
\end{aligned}$$

Procedendo ulteriormente si trova che ad ogni iterazione i punti raddoppiano, ma ogni volta ne sono da rimuovere tre in quanto provenienti da calcoli identici a computazioni già effettuate. Al passo k si ottengono dunque $3+2^k$ punti distinti: all'inizio sono 4, alla prima iterazione 5, alla seconda 7, e così via.

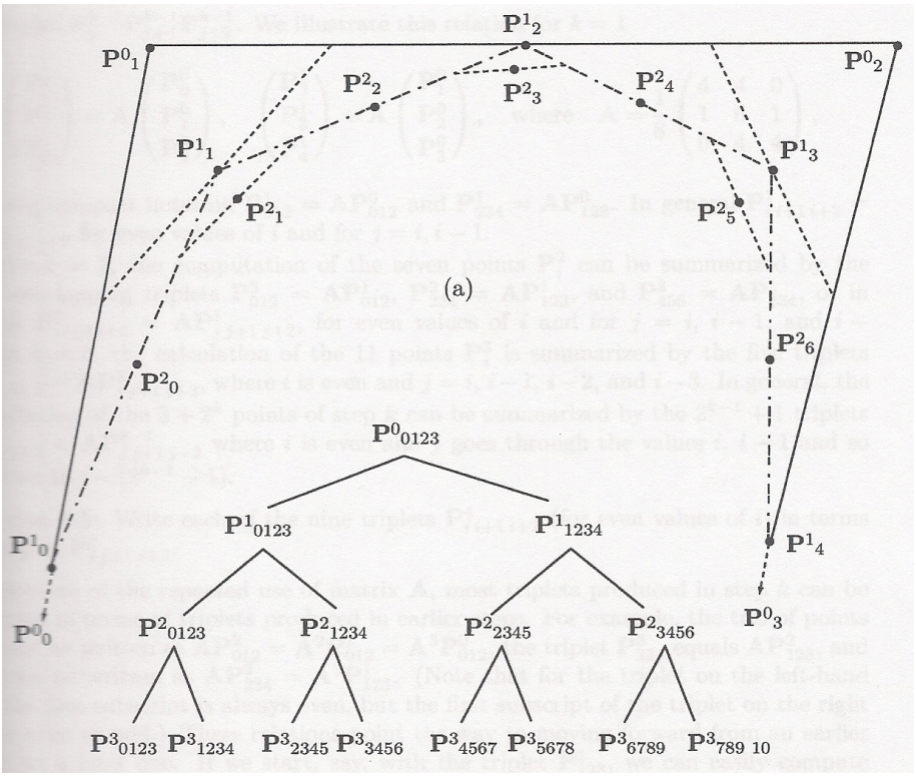

Figura 13: In (a) possiamo vedere il risultato dei primi due passi del raffinamento a partire da quattro punti, mentre in (b) vediamo come vengono raggruppati a quattro a quattro i punti passo dopo passo, per i primi tre passi. Notare che, in (a), $\mathbf{P}^0_0$ e $\mathbf{P}^0_3$ non sono localizzati nelle loro posizioni reali, ma sono più vicini agli altri per questioni di spazio.

Poiché ogni punto ottenuto al passo k è calcolato da due o tre punti ottenuti al passo $k-1$, può essere utile esprimere una terna di punti $\mathbf{P}^k_i$, $\mathbf{P}^k_{i+1}$, $\mathbf{P}^k_{i+2}$ come funzione di una terna di punti $\mathbf{P}^{k-1}_j$, $\mathbf{P}^{k-1}_{j+1}$, $\mathbf{P}^{k-1}_{j+2}$. Nel seguito di questa trattazione utilizzeremo la notazione di [5], che, benché non del tutto rigorosa, permette di esprimere con formule semplici le relazioni che ci interessano.

Per $k = 1$ la relazione può essere espressa come:

$$\left[\ \mathbf{P}^1_0, \mathbf{P}^1_1, \mathbf{P}^1_2\ \right]^T = \mathbf{A} \left[\ \mathbf{P}^0_0, \mathbf{P}^0_1, \mathbf{P}^0_2\ \right]^T$$

$$\left[\ \mathbf{P}^1_2, \mathbf{P}^1_3, \mathbf{P}^1_4\ \right]^T = \mathbf{A} \left[\ \mathbf{P}^0_1, \mathbf{P}^0_2, \mathbf{P}^0_3\ \right]^T$$

dove:

$$\mathbf{A} = \frac{1}{8} \begin{bmatrix} 4 & 4 & 0 \\ 1 & 6 & 1 \\ 0 & 4 & 4 \end{bmatrix}$$

Usando una notazione più compatta, possiamo scrivere:

$$\mathbf{P}^1_{0,1,2} = \mathbf{AP}^0_{0,1,2} \; ; \; \mathbf{P}^1_{2,3,4} = \mathbf{AP}^0_{1,2,3}$$

e cioè in generale:

$$\mathbf{P}^1_{i,i+1,i+2} = \mathbf{AP}^0_{j,j+1,j+2}$$

per valori pari di i e per $j = i,\, i-1$.

Per $k = 2$, il calcolo dei sette punti P_i^2 può essere riassunto nella seguente notazione compatta:

$$\mathbf{P}^2_{0,1,2} = \mathbf{AP}^1_{0,1,2} \; ; \; \mathbf{P}^2_{2,3,4} = \mathbf{AP}^1_{1,2,3} \; ; \; \mathbf{P}^2_{4,5,6} = \mathbf{AP}^1_{2,3,4}$$

quindi in generale:

$$\mathbf{P}^2_{i,i+1,i+2} = \mathbf{AP}^1_{j,j+1,j+2}$$

per valori pari di i e per $j = i,\, i-1,\, i-2$.

Procedimento analogo per $k = 3$, il cui calcolo porta ad avere undici punti:

$$\mathbf{P}^3_{0,1,2} = \mathbf{AP}^2_{0,1,2} \; ; \; \mathbf{P}^3_{2,3,4} = \mathbf{AP}^2_{1,2,3} \; ; \; \mathbf{P}^3_{4,5,6} = \mathbf{AP}^2_{2,3,4}$$
$$\mathbf{P}^3_{6,7,8} = \mathbf{AP}^2_{3,4,5} \; ; \; \mathbf{P}^3_{8,9,10} = \mathbf{AP}^2_{4,5,6}$$

ovvero:

$$\mathbf{P}^3_{i,i+1,i+2} = \mathbf{AP}^2_{j,j+1,j+2}$$

per valori pari di i e per $j = i,\, i-1,\, i-2,\, i-3,\, i-4$.

In generale, per un k qualsiasi, i $3+2^k$ punti risultanti possono essere scritti in questo modo:

$$\mathbf{P}^k_{i,i+1,i+2} = \mathbf{AP}^{k-1}_{j,j+1,j+2}$$

con i che assume tutti i valori pari da 0 a 2^k, e j che assume di volta in volta valori decrescenti di un'unità da i fino a $i - 2^{k-1}$. Può essere altresì considerato che la situazione assume una connotazione ancora più semplice se si nota che $j = \frac{i}{2}$, per cui si potrà ancora più in generale scrivere:

$$\mathbf{P}^k_{2j,2j+1,2j+2} = \mathbf{AP}^{k-1}_{j,j+1,j+2}$$

per ogni k e per ogni j che va da 0 a 2^{k-1}.

Poiché in questi calcoli viene utilizzata ripetutamente la matrice $\mathbf{A}$, molte delle terne ottenute al passo k possono essere espresse in termini di terne ottenute in passi precedenti.
Per esempio, possiamo scrivere:

$$\mathbf{P}^3_{0,1,2} = \mathbf{A}\mathbf{P}^2_{0,1,2} = \mathbf{A}^2\mathbf{P}^1_{0,1,2} = \mathbf{A}^3\mathbf{P}^0_{0,1,2}$$
$$\mathbf{P}^3_{2,3,4} = \mathbf{A}\mathbf{P}^2_{1,2,3}$$
$$\mathbf{P}^3_{4,5,6} = \mathbf{A}\mathbf{P}^2_{2,3,4} = \mathbf{A}^2\mathbf{P}^1_{1,2,3}$$

in virtù di quanto visto nel caso generale. Si noti che, per ogni tripletta a sinistra di un segno di uguale, il primo indice è sempre pari, ma il primo indice di una tripletta che sta a destra di un segno di uguale, può essere pari o dispari. Questo poiché il processo in questione può essere reiterato solamente nel caso in cui il primo indice risulti essere pari, dal momento che, secondo la notazione vista sopra, esso vale $2j$, e j deve essere un numero intero.
Grazie a queste relazioni è possibile passare piuttosto agevolmente da una terna di un certo passo k ad una anche di molto successiva. Partendo per esempio con la terna $\mathbf{P}^1_{1,2,3}$, è possibile calcolare senza difficoltà le terne $\mathbf{P}^2_{2,3,4}$, $\mathbf{P}^3_{4,5,6}$, $\mathbf{P}^4_{8,9,10}$, $\mathbf{P}^5_{16,17,18}$, e quelle di seguito, semplicemente moltiplicando $\mathbf{P}^1_{1,2,3}$ per potenze successive di $\mathbf{A}$.
Seguendo [5], con un abuso di notazione dal punto di vista analitico, scriviamo:

$$\lim_{k\to\infty} \mathbf{P}^k_{i,i+1,i+2} = \mathbf{A}^\infty\mathbf{P}^k_{i,i+1,i+2}$$

dove con $\mathbf{A}^\infty$ denotiamo $\lim_{k\to\infty} \mathbf{A}^k$. Il limite indicato sta a significare il valore a cui, fissati i e k, converge la successione definita da:

$$\mathbf{P}^k_{i,i+1,i+2},\ \mathbf{A}\mathbf{P}^k_{i,i+1,i+2},\ \mathbf{A}^2\mathbf{P}^k_{i,i+1,i+2},\ ...,\ \mathbf{A}^h\mathbf{P}^k_{i,i+1,i+2},\ ...,$$

ovvero, in virtù di quanto visto precedentemente, la successione equivalentemente definita da:

$$\mathbf{P}^k_{i,i+1,i+2},\ \mathbf{P}^{k+1}_{2i,2i+1,2i+2},\ \mathbf{P}^{k+2}_{4i,4i+1,4i+2},\ ...,\ \mathbf{P}^{k+h}_{2^hi,2^hi+1,2^hi+2},\ ...$$

Ogni terna $\mathbf{P}^k_{i,i+1,i+2}$ è un'approssimazione della curva B-spline ideale, ma il limite

$$\lim_{k\to\infty} \mathbf{P}^k_{i,i+1,i+2}$$

è un punto che sta effettivamente su tale curva.
La questione è ora quella di calcolare il limite di $\mathbf{A}^k$ per k che tende all'infinito; questa può essere risolta agilmente grazie ad un noto teorema di algebra lineare, che afferma che, se $\mathbf{A}$ è una matrice quadrata di ordine n, per la quale esistono n autovettori linearmente indipendenti, allora $\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^{-1}$, dove $\mathbf{Q}$ è la matrice le cui colonne sono gli n autovettori e $\mathbf{D}$ è la matrice diagonale i cui elementi diagonali sono gli autovalori di $\mathbf{A}$. Questo teorema implica che $\mathbf{A}^2 = \mathbf{A} * \mathbf{A} = (\mathbf{Q}\mathbf{D}\mathbf{Q}^{-1})*(\mathbf{Q}\mathbf{D}\mathbf{Q}^{-1}) = \mathbf{Q}\mathbf{D}^2\mathbf{Q}^{-1}$, e in generale $\mathbf{A}^k = \mathbf{Q}\mathbf{D}^k\mathbf{Q}^{-1}$. Grazie a ciò, possiamo scrivere la matrice $\mathbf{A}$, una volta calcolati i suoi autovalori e tre suoi autovettori linearmente indipendenti,

come:

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & 1 \\ -\frac{1}{2} & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & -\frac{2}{3} & \frac{1}{3} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{bmatrix}$$

Il limite $\mathbf{A}^\infty$ è dunque:

$$\begin{bmatrix} 1 & -1 & 1 \\ -\frac{1}{2} & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \lim_{k\to\infty} \begin{bmatrix} (\frac{1}{4})^k & 0 & 0 \\ 0 & (\frac{1}{2})^k & 0 \\ 0 & 0 & 1^k \end{bmatrix} * \begin{bmatrix} \frac{1}{3} & -\frac{2}{3} & \frac{1}{3} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{bmatrix}$$

ma, poiché la matrice $\mathbf{D}$ è diagonale, abbiamo:

$$\lim_{k\to\infty} \mathbf{D}^k = \lim_{k\to\infty} \begin{bmatrix} (\frac{1}{4})^k & 0 & 0 \\ 0 & (\frac{1}{2})^k & 0 \\ 0 & 0 & 1^k \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

il che porta a:

$$\begin{bmatrix} 1 & -1 & 1 \\ -\frac{1}{2} & 0 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{3} & -\frac{2}{3} & \frac{1}{3} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{6} & \frac{2}{3} & \frac{1}{6} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 \\ 1 & 4 & 1 \\ 1 & 4 & 1 \end{bmatrix}$$

da cui si perviene a:

$$\lim_{k\to\infty} \mathbf{P}^k_{i,i+1,i+2} = \frac{1}{6} \begin{bmatrix} 1 & 4 & 1 \\ 1 & 4 & 1 \\ 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}^k_i \\ \mathbf{P}^k_{i+1} \\ \mathbf{P}^k_{i+2} \end{bmatrix} = \frac{1}{6} \begin{bmatrix} 1, 4, 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}^k_i \\ \mathbf{P}^k_{i+1} \\ \mathbf{P}^k_{i+2} \end{bmatrix} = \frac{1}{6}(\mathbf{P}^k_i + 4\mathbf{P}^k_{i+1} + \mathbf{P}^k_{i+2})$$

per ogni intero k non negativo. Notare che le tre righe della matrice limite sono uguali, il che sta a significare che i tre punti di ogni tripletta convergono sullo stesso punto della curva B-spline.
Riassumendo, possiamo eseguire i seguenti passi:

- scegliamo quattro punti di controllo $\mathbf{P}^0_{0,1,2,3}$;
- scegliamo un valore k ed eseguiamo k passi di raffinamento;
- scegliamo un valore i ed una terna $\mathbf{P}^k_{i,i+1,i+2}$;
- calcoliamo $\frac{1}{6}(\mathbf{P}^k_i + 4\mathbf{P}^k_{i+1} + \mathbf{P}^k_{i+2})$

Questo risulterà essere un punto sul segmento di curva B-spline cubica definito dai quattro punti di controllo originari. Per provare ciò, possiamo esprimere ognuno dei tre punti $\mathbf{P}^k_{i,i+1,i+2}$ in funzione dei punti di controllo originari $\mathbf{P}^0_{0,1,2,3}$, e confrontare il risultato con il segmento generale di curva B-spline cubica fornito dalla teoria. Ecco alcuni esempi:

1. Cominciamo con $k = 0$ e $i = 0$. La terna iniziale è pertanto $\mathbf{P}^0_{0,1,2}$.

$$\lim_{k\to\infty} \mathbf{P}^0_{0,1,2} = \frac{1}{6} \begin{bmatrix} 1, 4, 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}^0_0 \mathbf{P}^0_1 \mathbf{P}^0_2 \end{bmatrix}^T = \frac{1}{6}(\mathbf{P}^0_0 + 4\mathbf{P}^0_1 + \mathbf{P}^0_2)$$

che corrisponde al punto iniziale del segmento di curva B-spline, $\mathbf{P}(0)$ secondo la notazione classica.

2. I valori $k = 0$ e $i = 1$ specificano la terna $\mathbf{P}^0_{1,2,3}$; notare che i non deve essere per forza pari.

$$\lim_{k\to\infty} \mathbf{P}^0_{1,2,3} = \frac{1}{6}\left[\ 1,4,1\ \right]\left[\ \mathbf{P}^0_1\mathbf{P}^0_2\mathbf{P}^0_3\ \right]^T = \frac{1}{6}(\mathbf{P}^0_1 + 4\mathbf{P}^0_2 + \mathbf{P}^0_3)$$

che corrisponde al punto finale $\mathbf{P}(1)$ del segmento di curva B-spline, $\mathbf{P}(1)$ secondo la notazione classica.

3. Eseguiamo un passo di raffinamento e scegliamo la terna $\mathbf{P}^1_{1,2,3}$ specificata da $k = 1$ e $i = 1$. Se questa terna è espressa in funzione dei punti di controllo $\mathbf{P}^0_i$, il risultato è:

$$\lim_{k\to\infty} \mathbf{P}^1_{1,2,3} = \frac{1}{6}(\mathbf{P}^1_1+4\mathbf{P}^1_2+\mathbf{P}^1_3) = \frac{1}{6}(\frac{1}{8}(\mathbf{P}^0_0+6\mathbf{P}^0_1+\mathbf{P}^0_2)+\frac{4}{2}(\mathbf{P}^0_1+\mathbf{P}^0_2)+\frac{1}{8}(\mathbf{P}^0_1+6\mathbf{P}^0_2+\mathbf{P}^0_3))$$

$$\lim_{k\to\infty} \mathbf{P}^1_{1,2,3} = \frac{1}{48}(\mathbf{P}^0_0 + 23\mathbf{P}^0_1 + 23\mathbf{P}^0_2 + \mathbf{P}^0_3)$$

Questo corrisponde al punto intermedio del segmento di curva, $\mathbf{P}(\frac{1}{2})$ secondo la notazione classica.

4 Un algoritmo di suddivisione multipasso per curve di Chaikin

4.1 Introduzione

Nelle sezioni precedenti, abbiamo visto che una curva di Chaikin è una curva di suddivisione, e che tale procedimento ha inizio a partire da una curva di controllo poligonale. Ad ogni passo del metodo di suddivisione, gli angoli della curva polinomiale vengono tagliati, e una nuova curva viene ottenuta e data in input per il passo successivo; come limite del procedimento, si ottiene una curva di Chaikin. In questo frangente, esamineremo un algoritmo di suddivisione multipasso per la generazione di tali curve, tale per cui, dato un intero positivo arbitrario k, sia possibile realizzare la curva poligonale risultante al k-esimo passo del procedimento di suddivisione direttamente a partire dal solo poligono di controllo iniziale. Si mostrerà che questo procedimento rende più veloce la generazione di curve.

4.2 Come viene migliorato l'algoritmo originario

Nel paragrafo precedente abbiamo brevemente introdotto ciò che abbiamo intenzione di ottenere con questo algoritmo. In questo paragrafo, formalizzeremo le cose.
Classicamente, tutti i procedimenti di suddivisione, considerano un poligono di controllo iniziale, denotato per esempio come $\mathbf{L}_0$, e si occupano di raffinarlo passo dopo passo. Ad ogni passo, il procedimento viene applicato al poligono $\mathbf{L}_k$ una sola volta, restituendo un nuovo poligono $\mathbf{L}_{k+1}$, che fungerà da input per il passo successivo della suddivisione. In questo caso, $\mathbf{L}_k$, con $k = 0, 1, 2, ...$, rappresenta il poligono di controllo risultante dopo il k-esimo passo del procedimento di suddivisione. Di conseguenza, secondo gli schemi di suddivisione classici, per ogni k, prima di poter calcolare $\mathbf{L}_{k+1}$, è necessario già avere calcolato $\mathbf{L}_k$. In questo ambito troveremo invece un metodo per calcolare $\mathbf{L}_k$ direttamente a partire dal solo $\mathbf{L}_0$, secondo lo schema di suddivisione di Chaikin: in questo modo, non è necessario calcolare $\mathbf{L}_i$ per ogni $i = 1, 2, ..., k-1$ prima di generare $\mathbf{L}_k$, e a partire dallo stesso poligono di controllo $\mathbf{L}_0$, la curva poligonale $\mathbf{L}_k$ ottenuta con questo metodo è la stessa che si ottiene alla k-esima esecuzione del metodo tradizionale di Chaikin. In questo modo, è possibile innanzitutto risparmiare memoria in quanto non è necessario allocare i dati relativi ai precedenti passi della suddivisione, inoltre l'esperienza mostra che questo metodo è diverse volte più veloce dell'algoritmo tradizionale di Chaikin.

4.3 L'algoritmo nei dettagli

Esaminiamo ora nei fatti in che cosa consiste questo algoritmo, proposto in [6], fornendo formule esplicite per il calcolo diretto di $\mathbf{L}_k$ direttamente a

partire da $\mathbf{L}_0$. Partiamo dall'idea fondamentale di Chaikin, il taglio degli angoli: tutti gli angoli della curva poligonale vengono tagliati ricorsivamente, e man mano che il procedimento di suddivisione avanza, le curve poligonali che risultano di volta in volta diventano sempre più liscie; al limite, si ottiene una curva liscia, con condizioni di regolarità geometrica e parametrica determinate. Denotiamo ora in questo modo:

$$\mathbf{L}_0 = \{\mathbf{P}_{0,0}, \mathbf{P}_{1,0}, ..., \mathbf{P}_{n_0,0}\}$$

i punti di controllo iniziali, e analogamente per un k qualunque:

$$\mathbf{L}_k = \{\mathbf{P}_{0,k}, \mathbf{P}_{1,k}, ..., \mathbf{P}_{n_k,k}\}$$

Così facendo, il primo intero a pedice indica il numero ordinale del punto (cominciando da 0), mentre il secondo il passo della suddivisione relativo al punto in questione.
Tornando all'algoritmo, una volta posta questa notazione, e analizzati gli angoli della curva poligonale iniziale, troviamo che i punti $\mathbf{P}_{2^k(i-1)+j,k}$, dove $j = 1, 2, ..., 2^k$, appartenenti alla curva poligonale $\mathbf{L}_k$, dipendono soltanto da $\mathbf{P}_{i-1,0}$, $\mathbf{P}_{i,0}$, $\mathbf{P}_{i+1,0}$. Pertanto, per induzione, si dimostra il seguente teorema che costituisce di fatto la procedura dell'algoritmo.

Teorema 1. *Tutti i punti in* $\mathbf{L}_k$*, con* $k = 1, 2, ...,$ *possono essere calcolati direttamente a partire unicamente da* $\mathbf{L}_0$ *tramite le seguenti formule:*

$$\begin{aligned} \mathbf{P}_{0,k} &= (2^{-1} + 2^{-(k+1)})\mathbf{P}_{0,0} + (2^{-1} - 2^{-(k+1)})\mathbf{P}_{1,0} \\ \mathbf{P}_{2^k(i-1)+j,k} &= F(j,k)\mathbf{P}_{i-1,0} + G(j,k)\mathbf{P}_{i,0} + H(j,k)\mathbf{P}_{i+1,0} \qquad (1) \\ \mathbf{P}_{2^k n_0 - 2^k + 1,k} &= (2^{-1} - 2^{-(k+1)})\mathbf{P}_{n_0-1,0} + (2^{-1} + 2^{-(k+1)})\mathbf{P}_{n_0,0} \end{aligned}$$

dove $i = 1, 2, ..., n_0 - 1$*,* $j = 1, 2, ..., 2^k$*, e:*

$$\begin{aligned} F(j,k) &= 2^{-1} - 2^{-(k+1)} - (j-1)(2^{-k} - 2^{-2k-1}j) \\ G(j,k) &= 2^{-1} + 2^{-(k+1)} + (j-1)(2^{-k} - 2^{-2k}j) \qquad (2) \\ H(j,k) &= j(j-1)(2^{-2k-1}) \end{aligned}$$

Dimostrazione. Useremo l'induzione: le formule valgono per $k = 1$, infatti:

$$\begin{aligned} F(1,1) = \tfrac{1}{4}\ ;\ G(1,1) = \tfrac{3}{4}\ ;\ H(1,1) = 0 \\ F(2,1) = 0\ ;\ G(2,1) = \tfrac{3}{4}\ ;\ H(2,1) = \tfrac{1}{4} \end{aligned}$$

da cui, applicandole alla terna di equazioni (1) per i punti, arriviamo a:

$$\begin{aligned} \mathbf{P}_{2i,1} &= \tfrac{3}{4}\mathbf{P}_{i,0} + \tfrac{1}{4}\mathbf{P}_{i+1,0} \\ \mathbf{P}_{2i+1,1} &= \tfrac{1}{4}\mathbf{P}_{i,0} + \tfrac{3}{4}\mathbf{P}_{i+1,0} \end{aligned}$$

per ogni $i = 0, 1, ..., n_0 - 1$. Si nota che queste due equazioni ricalcano fedelmente lo schema di Chaikin; pertanto, le formule valgono per $k = 1$. Supponiamo ora le formule valide per $k = m$ e dimostriamo che esse valgono anche per $k = m + 1$. Per provarlo, applichiamo l'algoritmo di Chaikin ai punti ottenuti al passo m-esimo della suddivisione:

$$\begin{aligned}
\mathbf{P}_{0,m+1} &= \tfrac{3}{4}\mathbf{P}_{0,m} + \tfrac{1}{4}\mathbf{P}_{1,m} \\
\mathbf{P}_{2^{(m+1)}(i-1)+j,m+1} &= \tfrac{1}{4}\mathbf{P}_{2^m(i-1)+\frac{j-1}{2},m} + \tfrac{3}{4}\mathbf{P}_{2^m(i-1)+\frac{j+1}{2},m} \quad \text{per } j \text{ dispari} \\
\mathbf{P}_{2^{(m+1)}(i-1)+j,m+1} &= \tfrac{3}{4}\mathbf{P}_{2^m(i-1)+\frac{j}{2},m} + \tfrac{1}{4}\mathbf{P}_{2^m(i-1)+\frac{j}{2}+1,m} \quad \text{per } j \text{ pari} \\
\mathbf{P}_{2^{(m+1)}n_0-2^{(m+1)}+1,m+1} &= \tfrac{1}{4}\mathbf{P}_{2^m n_0-2^m,m} + \tfrac{3}{4}\mathbf{P}_{2^m n_0-2^m+1,m}
\end{aligned}$$

dove $i = 1, 2, ..., n_0 - 1$, e $j = 1, 2, ..., 2^{m+1}$. Si sostituiscano ora tutti i punti in $\mathbf{L}_m$ con le espressioni che contengono solo punti in $\mathbf{L}_0$; ciò è possibile poichè abbiamo supposto che le formule fossero valide per $k = m$, come ipotesi induttiva. Svolgendo i calcoli, si ottengono le formule che si otterrebbero sostituendo $k = m+1$, e ciò prova la validità di esse anche per il passo successivo. Tramite induzione si è dunque provato che tali espressioni valgono per ogni k.

□

Poichè $F(j,k)$, $G(j,k)$, e $H(j,k)$ non dipendono dagli indici i, è possibile calcolarli preventivamente e allocarli in strutture dati di dimensioni 2^k in modo tale da rendere più veloci le operazioni. L'algoritmo può essere pertanto sintetizzato nei seguenti passi:

1. Si calcolino $F(j,k)$, $G(j,k)$, e $H(j,k)$, per ogni $j = 1, 2, ..., 2^k$, e si allochino i risultati in tre strutture dati (per esempio vettori);

2. Si calcolino $\mathbf{P}_{0,k}$ e $\mathbf{P}_{2^k n_0-2^k+1,k}$;

3. Si calcoli $\mathbf{P}_{2^k(i-1)+j,k}$ per ogni coppia (i,j) da $(1,1)$ a $(n_0 - 1, 2^k)$, dove si fa variare più internamente j e più esternamente i.

In metalinguaggio il tutto diventa:

Algoritmo Calcolo diretto di $\mathbf{L}_k$ da $\mathbf{L}_0$.
Input. $\mathbf{L}_0 = \{\mathbf{P}_{0,0}, \mathbf{P}_{1,0}, ..., \mathbf{P}_{n_0,0}\}$
Output. $\mathbf{L}_k = \{\mathbf{P}_{0,k}, \mathbf{P}_{1,k}, ..., \mathbf{P}_{n_k,k}\}$

(1) Calcolare $F(j,k)$, $G(j,k)$, e $H(j,k)$ per ogni $j = 1, 2, ..., 2^k$, tramite le equazioni (2), e allocare i risultati in tre strutture dati (per esempio vettori);

(2) Calcolare $\mathbf{P}_{0,k}$ e $\mathbf{P}_{2^k n_0-2^k+1,k}$ tramite le equazioni (1);

(3) **Per** $i = 1$ **fino a** $n_0 - 1$, **fare**
Per $j = 1$ **fino a** 2^k, **fare**
Calcolare $\mathbf{P}_{2^k(i-1)+j,k}$ tramite le equazioni (1);

(4) Fine dell'algoritmo.

Nelle figure 14 e 15 è mostrata graficamente la differenza tra l'applicazione iterativa del metodo classico di Chaikin e l'applicazione immediata di questo nuovo metodo.

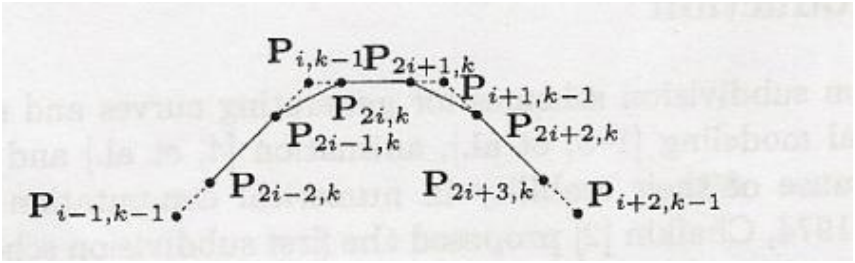

Figura 14: Passo di suddivisione semplice: $\mathbf{L}_k$ è calcolato a partire da $\mathbf{L}_{k-1}$. [6]

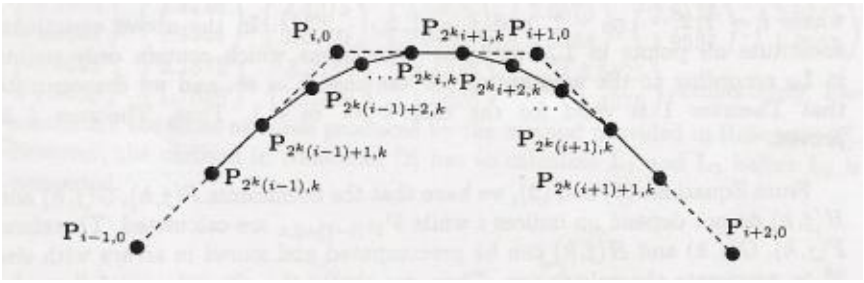

Figura 15: Passo di suddivisione multiplo: $\mathbf{L}_k$ è calcolato direttamente a partire da $\mathbf{L}_0$. [6]

4.4 Confronto computazionale con l'algoritmo originale

Svolgiamo ora un confronto in termini computazionali tra questo algoritmo e quello originale di Chaikin, ponendoci come obiettivo quello di provare sul campo che questo nuovo algoritmo migliora la velocità dell'originale (di diverse volte).

4.4.1 Presentazione degli algoritmi MATLAB

Riconsideriamo la già vista nel paragrafo 1.4 *function* MATLAB che svolge l'algoritmo di Chaikin:

```
function Pk = chaikin(P,k)
%Applicazione classica dell'algoritmo di Chaikin.
%Parametri di input: la matrice dei punti iniziali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z), e il passo k.
%Parametri di output: la matrice dei punti finali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z) e un grafico 3D, che nel
%caso in cui tutte le coordinate z siano nulle viene plottato come 2D.
P0 = P; %teniamoci da parte il poligono originale per il grafico
n=size(P,1)-1;
for j=1:k %ci sono variazioni rispetto alla notazione originaria
    for i=1:n %dovute al fatto che Matlab indicizza da 1 e non da 0.
        Q(i,:) = (3/4)*P(i,:) + (1/4)*P(i+1,:);
        R(i,:) = (1/4)*P(i,:) + (3/4)*P(i+1,:);
    end
    for i=1:n
        P(2*i-1,:) = Q(i,:);
        P(2*i,:) = R(i,:);
    end
    n=size(P,1)-1; %bisogna ridefinire n poiché P cambia dimensione
end
Pk = P
if(norm(P0(:,3)) ~= 0) %equivalente a dire che tutte le coordinate z = 0
    plot3(P0(:,1),P0(:,2),P0(:,3),'o--')
    hold on
    plot3(Pk(:,1),Pk(:,2),Pk(:,3),'ro-')
else
    plot(P0(:,1),P0(:,2),'o--')
    hold on
    plot(Pk(:,1),Pk(:,2),'ro-')
end
```

Consideriamo ora questa *function* che svolge l'algoritmo oggetto di questa sezione:

```
function Pk = zhang(P0,k)
%Funzione che, dato un poligono di controllo composto da n_0+1 punti,
%genera il poligono di controllo del k-esimo passo di suddivisione di
%Chaikin tramite l'algoritmo di Wu, Yong, Zhang, e Zhang.
%Parametri di input: la matrice dei punti iniziali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z), e il passo k.
%Parametri di output: la matrice dei punti finali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z) e un grafico 3D, che nel
%caso in cui tutte le coordinate z siano nulle viene plottato come 2D.
n0 = size(P0,1)-1;
for j=1:2^k
    F(j,k)= 2^(-1) - 2^(-k-1) - (j-1)*(2^(-k) - j*2^(-2*k-1));
    G(j,k)= 2^(-1) + 2^(-k-1) + (j-1)*(2^(-k) - j*2^(-2*k));
    H(j,k)= (j-1)*j*2^(-2*k-1);
end
Pk(1,:) = ( 2^(-1) + 2^(-k-1) )*P0(1,:) + ( 2^(-1) - 2^(-k-1) )*P0(2,:);
%il primo indice va aumentato di 1 poiché Matlab indicizza da 1 e non da 0
Pk(2^k*n0-2^k+2,:) = (2^(-1)-2^(-k-1))*P0(n0,:) + (2^(-1)+2^(-k-1)) * P0(n0+1,:);
for i=1:n0-1
    for j=1:2^k
        Pk(2^k*(i-1)+j+1,:) = F(j,k)*P0(i,:)+G(j,k)*P0(i+1,:)+H(j,k)*P0(i+2,:);
    end
end
if(norm(P0(:,3)) ~= 0) %equivalente a dire che tutte le coordinate z = 0
    plot3(P0(:,1),P0(:,2),P0(:,3),'o--')
    hold on
    plot3(Pk(:,1),Pk(:,2),Pk(:,3),'ro-')
else
    plot(P0(:,1),P0(:,2),'o--')
    hold on
    plot(Pk(:,1),Pk(:,2),'ro-')
end
```

4.4.2 Esempi grafici

Vediamo ora cosa restituiscono i due algoritmi se applicati ad un poligono di controllo di prova. Il poligono in questione è quello utilizzato nell'esempio del paragrafo 1.4: aperto e formato da 6 punti generati con casualità uniforme nel quadrato di vertici $(0,0)$, $(0,1)$, $(1,1)$, $(1,0)$, tramite la funzione `rand` di `MATLAB`. Si noti che il fatto che il poligono non si intrecci è solo un caso dovuto alla generazione dei punti: in teoria esso può anche essere intrecciato. Nell'esempio 1a (figura 16) applicheremo la *function* relativa all'algoritmo di Chaikin, mentre nell'esempio 1b (figura 17) quella relativa all'algoritmo multipasso. Si noti che il risultato è identico; con linea blu trat-

teggiata viene rappresentato il poligono di controllo, mentre con linea rossa viene rappresentata l'approssimante della curva. In entrambi gli esempi abbiamo preso $k = 3$, che costituisce un buon compromesso tra un'esecuzione dei calcoli pressoché immediata e una già visibile ad occhio approssimazione della curva ideale.

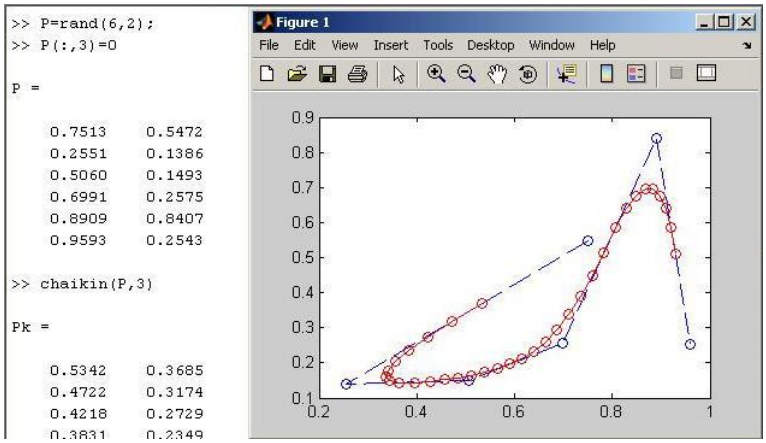

Figura 16: Esempio 1a - Applicazione della *function* `MATLAB` che svolge l'algoritmo di Chaikin su un poligono di controllo di prova.

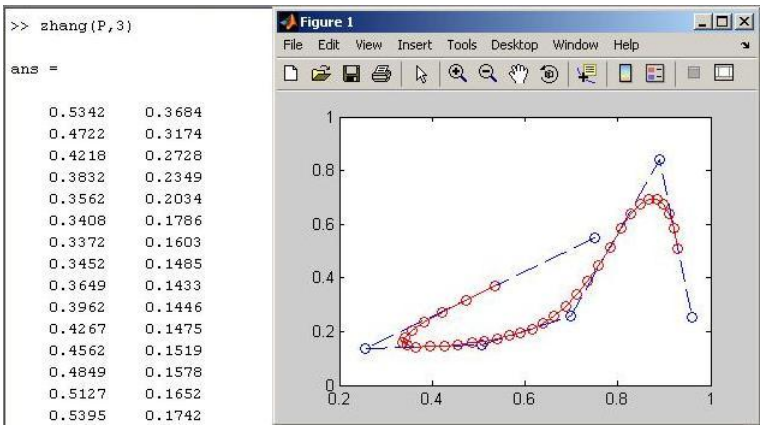

Figura 17: Esempio 1b - Applicazione della *function* `MATLAB` che svolge l'algoritmo multipasso su un poligono di controllo di prova.

Nel seguito verranno presentati altri esempi, per i quali eseguiremo soltanto l'algoritmo multipasso, avendone già verificata l'equivalenza con l'algoritmo di Chaikin. Gli esempi sono tratti da [6]; nell'immagine viene mostrata parte dell'esecuzione `MATLAB` e la finestra grafica contenente il risultato finale. Per il terzo esempio (figura 20), è stato utilizzato un opportuno algoritmo `MATLAB`

per generare il poligono di controllo del fiore, sfruttando poi le varie simmetrie della figura, in modo da evitare la necessità di definire analiticamente ogni punto.

```
L0(1,1:3)=[-0.05 0.1 0]; L0(2,1:3)=[-0.2 0.3 0]; L0(3,1:3)=[-0.3 0.6 0];
L0(4,1:3)=[-0.3 0.8 0]; L0(5,1:3)=[-0.15 1 0]; L0(6,1:3)=[0.15 1 0];
L0(7,1:3)=[0.3 0.8 0]; L0(8,1:3)=[0.3 0.6 0]; L0(9,1:3)=[0.2 0.3 0];
L0(10,1:3)=[0.05 0.1 0];
t=2*pi/5;
for i=1:4
    for j=1:10
    L0(10*i+j,1:3)=L0(j,1:3)*[cos(i*t) -sin(i*t) 0; sin(i*t) cos(i*t) 0; 0 0 1];
    end
end
L0(51,1:3)=L0(1,1:3);
L0(52,1:3)=L0(2,1:3); %servono per chiudere la stella
```

Si noti il prodotto vettore per matrice definito diversamente dal solito, la cosa è dovuta al fatto che in questo caso i punti sono rappresentati da vettori riga.

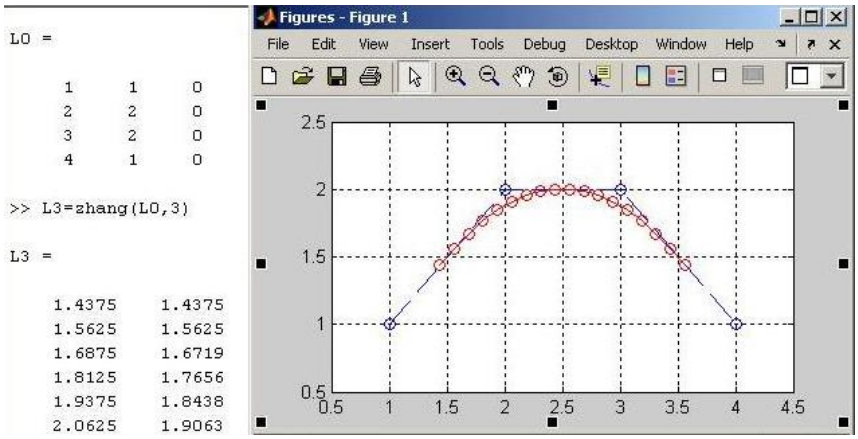

Figura 18: Esempio 2 - quattro punti di controllo presi a coordinate intere.

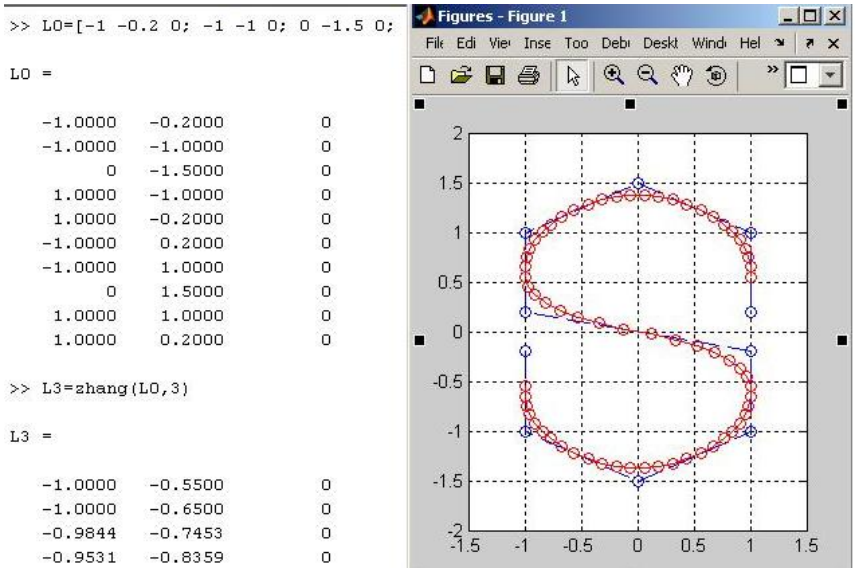

Figura 19: Esempio 3 - una lettera S.

4.4.3 Versioni delle procedure per timing, test e risultati

Queste due *function* possono essere opportunamente modificate per facilitare le operazioni di rilevamento della tempistica computazionale, sopprimendo tutti gli output non necessari a questo tipo di verifica.

Questa è la versione modificata della *function* che svolge Chaikin:

```
function Pk = chaikin_time(P,k)
%Applicazione classica dell'algoritmo di Chaikin.
%Parametri di input: la matrice dei punti iniziali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z), e il passo k.
%Parametri di output: la matrice dei punti finali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z) e un grafico 3D, che nel
%caso in cui tutte le coordinate z siano nulle viene plottato come 2D.

%QUESTA VERSIONE DELLA FUNCTION E' PER TEST DI VELOCITA' DA EFFETTUARSI CON
%TIC; CHAIKIN_TIME(P0,K); TOC. NON RESTITUISCE NULLA IN OUTPUT.

P0 = P; %teniamoci da parte il poligono originale per il grafico
n=size(P,1)-1;
for j=1:k %ci sono variazioni rispetto alla notazione originaria
    for i=1:n %dovute al fatto che Matlab indicizza da 1 e non da 0.
        Q(i,:) = (3/4)*P(i,:) + (1/4)*P(i+1,:);
        R(i,:) = (1/4)*P(i,:) + (3/4)*P(i+1,:);
    end
```

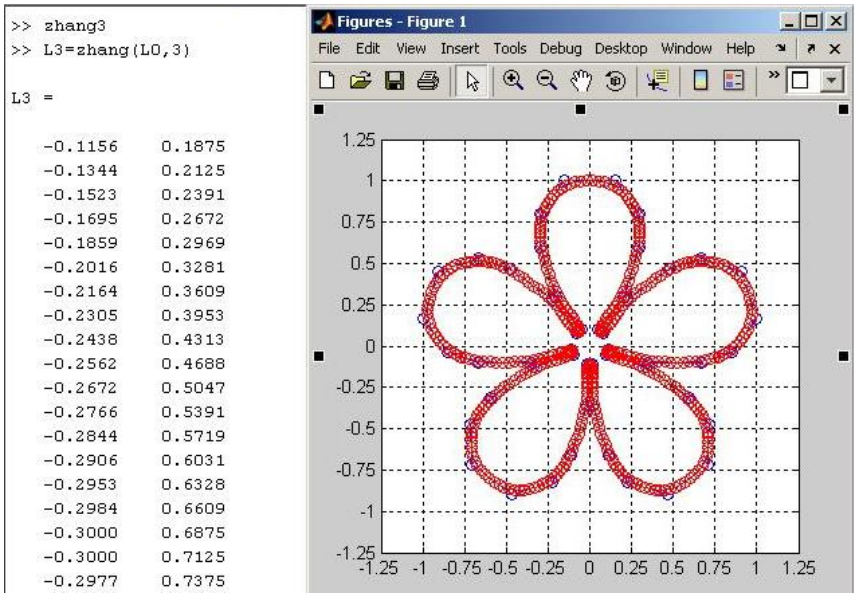

Figura 20: Esempio 4 - un fiore a cinque petali.

```
    for i=1:n
        P(2*i-1,:) = Q(i,:);
        P(2*i,:) = R(i,:);
    end
    n=size(P,1)-1; %bisogna ridefinire n poiché P cambia dimensione
end
Pk = P;
```

e questa è la versione modificata della *function* che svolge l'algoritmo multipasso:

```
function Pk = zhang_time(P0,k)
%Funzione che, dato un poligono di controllo composto da n_0+1 punti,
%genera il poligono di controllo del k-esimo passo di suddivisione di
%Chaikin tramite l'algoritmo di Wu, Yong, Zhang, e Zhang.
%Parametri di input: la matrice dei punti iniziali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z), e il passo k.
%Parametri di output: la matrice dei punti finali (ogni riga un punto,
%prima colonna coordinata x, seconda y, terza z) e un grafico 3D, che nel
%caso in cui tutte le coordinate z siano nulle viene plottato come 2D.

%QUESTA VERSIONE DELLA FUNCTION E' PER TEST DI VELOCITA' DA EFFETTUARSI CON
%TIC; ZHANG_TIME(P0,K); TOC. NON RESTITUISCE NULLA IN OUTPUT.

n0 = size(P0,1)-1;
```

```
for j=1:2^k
    F(j,k)= 2^(-1) - 2^(-k-1) - (j-1)*(2^(-k) - j*2^(-2*k-1));
    G(j,k)= 2^(-1) + 2^(-k-1) + (j-1)*(2^(-k) - j*2^(-2*k));
    H(j,k)= (j-1)*j*2^(-2*k-1);
end
Pk(1,:) = ( 2^(-1) + 2^(-k-1) )*P0(1,:) + ( 2^(-1) - 2^(-k-1) )*P0(2,:);
%il primo indice va aumentato di 1 poiché Matlab indicizza da 1 e non da 0
Pk(2^k*n0-2^k+2,:) = (2^(-1)-2^(-k-1))*P0(n0,:) + (2^(-1)+2^(-k-1)) * P0(n0+1,:);
for i=1:n0-1
    for j=1:2^k
        Pk(2^k*(i-1)+j+1,:) = F(j,k)*P0(i,:)+G(j,k)*P0(i+1,:)+H(j,k)*P0(i+2,:);
    end
end
```

Eseguiamo ora una serie di test e osserviamone i risultati. Consideriamo 15 punti sul piano, distribuiti con casualità uniforme nel quadrato di vertici $(0,0)$, $(0,1)$, $(1,1)$, $(1,0)$:

```
>> P=rand(15,2);
>> P(:,3)=0;
```

e, variando opportunamente k di volta in volta, applichiamo i due algoritmi. Per ogni valore di k, vengono svolte più misurazioni del tempo; i valori variano leggermente di esecuzione in esecuzione, qui riportiamo il migliore ottenuto, considerando che le differenze sono percentualmente piccole e che resta chiaro come il secondo algoritmo sia uniformemente più veloce del primo.

```
>> tic; chaikin_time(P,3); toc
Elapsed time is 0.001858 seconds.
>> tic; zhang_time(P,3); toc
Elapsed time is 0.000940 seconds.

>> tic; chaikin_time(P,6); toc
Elapsed time is 0.017422 seconds.
>> tic; zhang_time(P,6); toc
Elapsed time is 0.006039 seconds.

>> tic; chaikin_time(P,9); toc
Elapsed time is 0.286480 seconds.
>> tic; zhang_time(P,9); toc
Elapsed time is 0.037700 seconds.

>> tic; chaikin_time(P,10); toc
Elapsed time is 0.998293 seconds.
```

```
>> tic; zhang_time(P,10); toc
Elapsed time is 0.090961 seconds.
```

Possiamo notare inoltre come, al crescere di k, il rapporto tra il tempo impiegato dall'algoritmo di Chaikin e quello impiegato dall'algoritmo multipasso aumenti: mentre per $k = 3$ il primo impiega circa il doppio del tempo del secondo:

$$\frac{0.001858}{0.000940} \cong 1.98$$

per $k = 6$ il tempo impiegato diventa già il triplo:

$$\frac{0.017422}{0.006039} \cong 2.88$$

per diventare di quasi otto volte per $k = 9$:

$$\frac{0.286480}{0.037700} \cong 7.60$$

e addirittura di undici per $k = 10$:

$$\frac{0.998293}{0.090961} \cong 10.97$$

I dati sono stati calcolati su di un computer desktop Pentium 4 con una CPU di 3 Ghz e 2048 MB di RAM; i tempi risultanti sono infatti inferiori a quelli riscontrati dagli ideatori dell'algoritmo che utilizzarono una macchina meno performante, ma viene confermato il minore e migliore utilizzo delle risorse di sistema da parte di questo nuovo algoritmo, il che è pienamente dimostrato dal minor tempo impiegato nello svolgimento di ogni procedura. L'algoritmo multipasso si propone dunque come un miglioramento di quello originario di Chaikin in termini computazionali, dal momento che restituisce lo stesso risultato ad un costo computazionale tuttavia notevolmente inferiore.

5 L'algoritmo di suddivisione razionale di Nasri-Farin e una sua evoluzione

5.1 Introduzione

Sin dalla formulazione originaria di Bézier e di De Casteljau, la generazione di curve a partire da un dato poligono di controllo è divenuto uno standard molto affermato nel disegno geometrico computerizzato (CAGD). A partire da un poligono di controllo, si iniziarono a sviluppare algoritmi per generare curve di Bézier, curve B-spline, B-spline razionali, e altri tipi di curve. Come abbiamo visto prima, la suddivisione fu semplificata da Chaikin mediante il suo sistema di taglio degli angoli; l'algoritmo fu presentato in una conferenza al termine della quale Riesenfeld e Forrest mostrarono come la curva fosse una B-spline quadratica (si veda [4]). Nonostante la semplicità dell'idea, la sua versatilità era tale da permettere una rapida estensione alla generazione di superfici B-spline tensore prodotto a partire da un'arbitraria rete di punti (poliedro di controllo), grazie soprattutto ai lavori di Doo-Sabin e di Catmull-Clark. Da allora, la suddivisione è diventata una delle aree principali di ricerca nel CAGD e la letteratura matematica è ricca di risultati interessanti. Le tecniche di suddivisione basate sulla ricorsione hanno il vantaggio di permettere di definire superfici di forma molto variabile a partire da reti topologicamente arbitrarie, andando oltre la necessità di una topologia rettangolare richiesta dai metodi tensore prodotto caratteristici del CAD classico; inoltre, sono numericamente stabili e semplici da implementare. A causa delle limitazioni imposte dalla rappresentazione uniforme delle B-spline, la ricerca nel campo degli algoritmi di suddivisione razionali per la generazione di curve e superfici razionali è tuttora attiva. Alla fine degli anni '90, Sederberg e altri presentarono un interessante approccio inerente una suddivisione ricorsiva non uniforme. Di particolare interesse è la generazione di superfici delimitate da curve razionali: per esempio, spesso nelle applicazioni può essere utile unire superfici di suddivisione con altri tipi di superfici delimitate da circonferenze.

Al fine di raggiungere questo risultato, lo schema di suddivisione deve generare contorni circolari, a partire dal poligono di controllo che delimita il poliedro di controllo che definisce la superficie. Diversi ricercatori hanno trattato il problema delle spline circolari, e un risultato notevole fu annunciato quando Nasri e Farin proposero in [3] uno schema di suddivisione razionale per generare una curva di classe G^1 a partire da un dato poligono di controllo. In generale, le parti che coinvolgono la curva non risultano essere circolari, ma se i segmenti incidenti su di un vertice sono della stessa lunghezza, allora la parte di curva corrispondente è un arco circolare. Dunque, per generare una spline circolare, l'applicazione dell'algoritmo è limitata a poligoni di controllo con segmenti di eguale misura.

Questa limitazione fu superata e l'annuncio fu dato nel gennaio del 2001 quando lo stesso Nasri, Overveld e Wyvill proposero in [4] un algoritmo di suddivisione che fosse in grado di generare una curva circolare a tratti a partire da un qualsiasi poligono di controllo; per ogni angolo del poligono di controllo, l'algoritmo genera un biarco circolare. Non si tratta del primo algoritmo a fare uso degli archi doppi, tuttavia è il primo che può interpolare un insieme arbitrario di punti: ha il vantaggio di poter essere utilizzato nella generazione di superfici di suddivisione delimitate da circonferenze, e inoltre i suoi tratti circolari sono parametrizzati secondo la lunghezza della curva, cioè secondo la loro ascissa curvilinea.

5.2 L'algoritmo di suddivisione razionale di Nasri-Farin

5.2.1 I progressi rispetto a Chaikin

Dato un poligono di controllo, l'algoritmo di Chaikin [1] genera una curva tramite il noto processo di taglio degli angoli; questo procedimento, che può essere considerato come una reiterazione di inserimento di nodi, fornisce come limite la B-spline quadratica basata sul poligono di controllo originale. Uno degli svantaggi dell'algoritmo di Chaikin è l'incapacità di ottenere una circonferenza; la curva limite risulta infatti essere una sequenza di parabole. Anche utilizzando una versione quadridimensionale dell'algoritmo di Chaikin, ovvero applicando il procedimento nello spazio a quattro dimensioni, e poi proiettando i risultati in tre dimensioni, ponendo come quarta coordinata $w = 1$, è impossibile ottenere curve circolari a tratti.
Fin dagli anni '90, esistevano già diversi metodi per generare circonferenze esatte, come le curve razionali di Bézier e le spline di Chebychev. Tuttavia, la forma razionale non restituisce una curva parametrizzata secondo la sua ascissa curvilinea, mentre la forma di Chebychev è computazionalmente troppo onerosa.

5.2.2 Sviluppo dell'algoritmo

Illustreremo ora nei suoi dettagli l'algoritmo di suddivisione razionale di Nasri-Farin. Il metodo è basato su una versione modificata del metodo di Chaikin per la generazione di curve, e l'idea è quella di generare un arco circolare come una NURBS quadratica a partire da tre vertici i cui corrispondenti pesi sono $(1, \cos\alpha, 1)$, dove α è l'angolo alla base dei triangoli isosceli formati da questi vertici. Dunque i due segmenti che connettono i tre vertici devono avere la stessa lunghezza.

Dato un poligono di controllo $\widehat{P_0}$, una curva a tratti può essere generata nel seguente modo:

1. Si consideri il punto medio di ogni segmento del poligono iniziale $\widehat{P_0}$ formando un nuovo poligono P_0.

2. Per ogni vertice b_i del nuovo poligono P_0, si calcoli il peso w_i come:

$$w_i = \sin \frac{\beta_i}{2}$$

dove β_i è l'angolo compreso tra b_{i-1}, b_i, b_{i+1}, che indichiamo con $\angle(b_{i-1}, b_i, b_{i+1})$. Ad ognuno dei punti medi inseriti viene assegnato il peso 1 e al punto b_i viene assegnato il peso w_i.

3. Si applichi un processo di taglio degli angoli a P_0, rimpiazzando ogni vertice che non sia punto medio b_i, con due vertici $b_{i,0}$ e $b_{i,1}$ definiti da:

$$b_{i,0} = (1 - s_i)b_{i-1} + s_i b_i$$
$$b_{i,1} = (1 - s_i)b_{i+1} + s_i b_i$$

dove s_i è dato da:

$$s_i = \frac{w_i}{1 + w_i} = \frac{\sin \frac{\beta_i}{2}}{1 + \sin \frac{\beta_i}{2}}$$

4. Si inserisca il punto medio di tutti i V-tagli razionali, cioè dei segmenti congiungenti i vari $b_{i,0}$ con i rispettivi $b_{i,1}$, e si rinumerino i vertici formando un nuovo poligono P_1.

5. Si ripeta ricorsivamente dal punto 2 utilizzando il poligono suddiviso P_1 come P_0.

In base a questo algoritmo di suddivisione, una sequenza di poligoni P_k, ad ognuno dei quali può essere attribuito un vettore di pesi W_k, è generato. Il vettore W_k è dato da:

$$\{..., 1, \sin \frac{\beta_{i-1}^k}{2}, 1, \sin \frac{\beta_{i+2}^k}{2}, 1, ...\}$$

dove β_i^k è l'angolo compreso tra i corrispondenti segmenti del poligono suddiviso P_k incidenti in b_i^k. Può essere dimostrato che se i due segmenti successivi sono di eguale lunghezza, allora la corrispondente curva razionale sarà circolare e parametrizzata dalla sua ascissa curvilinea.

Usando un quadrato come poligono di controllo, ed un fattore s_i definito da:

$$s_i = \frac{\cos \frac{\pi}{2^{i+1}}}{1 + \cos \frac{\pi}{2^{i+1}}}$$

l'algoritmo genererà una circonferenza. In figura 21 viene mostrato un esempio di tale applicazione.

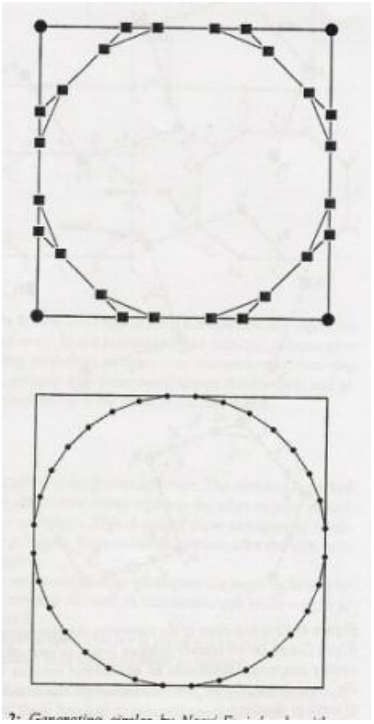

Figura 21: Generazione di circonferenze tramite l'algoritmo di Nasri-Farin: prima e seconda suddivisione a partire da un quadrato, e circonferenza limite. [4]

I vettori di pesi dei corrispondenti poligoni P_k saranno dati da:

$$W_k = (..., 1, \cos\frac{\pi}{2^{k+1}}, 1, \cos\frac{\pi}{2^{k+1}}, ...)$$

pertanto i pesi nel caso della circonferenza possono essere facilmente calcolati ad ogni iterazione.

5.3 Un algoritmo di suddivisione ricorsivo per spline circolari a tratti

L'algoritmo visto sopra può essere esteso per generare una curva spline circolare a tratti a partire da un poligono di controllo arbitrariamente dato. L'idea consiste nel costruire un biarco circolare per ogni angolo del poligono di controllo dato, come segue.

5.3.1 Costruzione di un biarco circolare

Per prima cosa, mostreremo come costruire segmenti di eguale lunghezza a partire da due segmenti di un poligono, non necessariamente della stessa

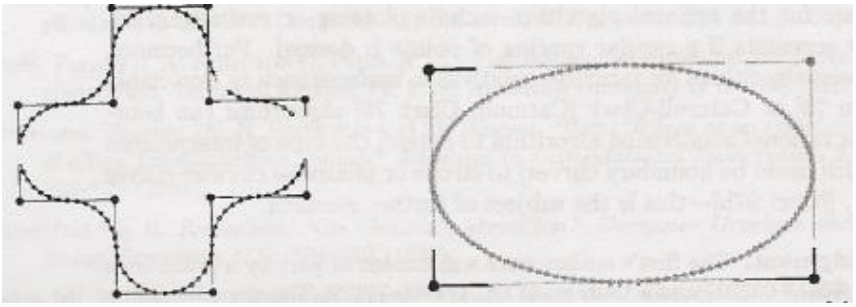

Figura 22: Applicato ad un poligono equilatero, l'algoritmo restituisce una curva circolare a tratti (sinistra); nel caso di un rettangolo, i cui lati consecutivi non sono della stessa lunghezza, viene generato un ellisse (destra). [3]

lunghezza, che condividano un vertice b_i. Siano b_{i-1} e b_{i+1} gli altri vertici di questi segmenti, come visibile in figura 23.

Ovviamente, non è in generale possibile costruire un singolo arco circolare c_i passante per b_{i-1} e b_{i+1} e tangente ad entrambi i segmenti in questi punti, dal momento che ciò richiederebbe che i due segmenti siano della stessa lunghezza. Invece, possiamo generare un biarco circolare ponendo tre punti: v_0 su b_{i-1} b_i, v_2 su b_i b_{i+1}, e v_1 su v_0 v_2, in modo tale che i due triangoli b_{i-1} v_0 v_1 e v_1 v_2 b_{i+1} siano isosceli. Questi triangoli definiscono due archi circolari c_i^0 e c_i^1 che insieme formano il biarco c_i.

Troviamo ora un modo pratico di rendere possibile questa costruzione.

In generale, i punti v_0, v_1, v_2 possono esere determinati mappando gli intervalli [0, λ_i, 1], [0, γ_i, 1], e [0, β_i, 1] rispettivamente in [b_{i-1} b_i], [b_i b_{i+1}], e [v_0 v_2], dove λ_i, γ_i e β_i sono scalari compresi tra 0 e 1. In questo modo si giunge a:

$$\begin{aligned} v_0 &= b_{i-1} + \lambda_i(b_i - b_{i-1}) \\ v_2 &= b_i + \gamma_i(b_{i+1} - b_i) \\ v_1 &= v_0 + \beta_i(v_2 - v_0) \end{aligned}$$

Tra tutte le soluzioni di questo sistema, l'imposizione della condizione $\lambda_i = \gamma_i$ ci garantisce di trovarne una unica per la quale si ha semplicità di costruzione e simmetria nello scambiare b_{i-1} e b_{i+1}. Di conseguenza:

$$v_0 v_2 = \lambda_i b_{i-1} b_{i+1}$$

Usando le condizioni:

$$|v_0 v_1| = |b_{i-1} v_0|$$

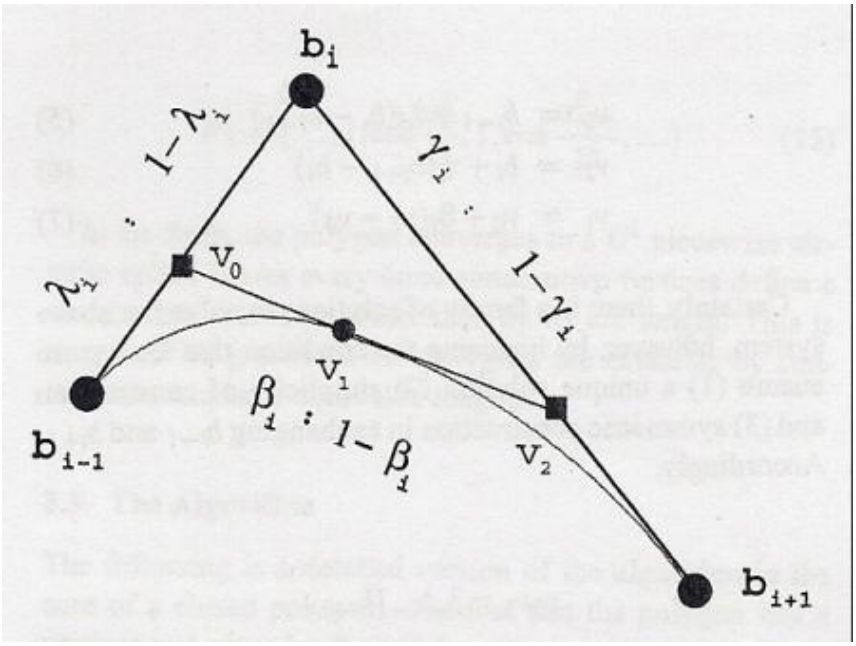

Figura 23: Costruzione di biarchi: i due archi sono definiti da b_{i-1}, v_0, v_1, e v_1, v_2, b_{i+1}. [4]

$$|v_1v_2| = |v_2b_{i+1}|$$

possiamo scrivere:

$$|v_0v_2| = |b_{i-1}v_0| + |v_2b_{i+1}|$$

oppure:

$$\begin{aligned} \lambda_i|b_{i-1}b_{i+1}| &= |b_{i-1}b_i| - |v_0b_i| + |b_{i+1}b_i| - |v_2b_i| \\ &= |b_{i-1}b_i| - \lambda_i|b_{i-1}b_i| + |b_{i+1}b_i| - \lambda_i|b_{i+1}b_i| \end{aligned}$$

o semplicemente:

$$\lambda_i(|b_{i-1}b_{i+1}| + |b_{i-1}b_i| + |b_{i+1}b_i|) = |b_{i-1}b_i| + |b_{i+1}b_i|$$

il che dà:

$$\lambda_i = \frac{|b_{i-1}b_i| + |b_{i+1}b_i|}{|b_{i-1}b_{i+1}| + |b_{i-1}b_i| + |b_{i+1}b_i|}$$

I punti v_0 e v_2 risultano ora determinati se sappiamo trovare v_1 evincendo β_i che è dato da:

$$\beta_i = \frac{(1-\lambda_i)|b_{i-1}b_i|}{\lambda_i|b_{i-1}b_{i+1}|}$$

Si noti che, se $|b_{i-1}b_i| = |b_ib_{i+1}|$, allora i due triangoli $b_{i-1}v_0v_1$ e $v_1v_2b_{i+1}$ sono congruenti, ed i due archi c_i^0 e c_i^1 sono due sottoarchi equivalenti di c_i.

5.3.2 L'algoritmo che genera la spline circolare

Dato un poligono di controllo P_0, un algoritmo di suddivisione per generare una spline circolare a tratti di classe G^1 può essere realizzato usando la costruzione dei biarchi e l'algoritmo di suddivisione razionale di Nasri-Farin ([3]).

I passi principali sono i seguenti:

1. Si marchi ogni vertice, eccetto il primo e l'ultimo nel caso in cui il poligono in questione sia aperto, come vertice d'angolo.

2. Si inserisca il punto medio di ogni segmento, formando un nuovo poligono P_0, e si marchino questi punti come vertici non di angolo. Abbiamo definito questi punti in quanto ci serviranno in seguito per le suddivisioni.

3. Si applichi il procedimento di costruzione dei biarchi che rimpiazza ogni vertice d'angolo di P_0 con tre vertici v_0, v_1 e v_2 come visto prima. Il vertice v_2 sia marcato come non di angolo, mentre gli altri due siano marcati come di angolo. Si rinumerino i vertici dopo la costruzione dei biarchi.

4. Per ogni vertice di angolo b_i, si calcoli l'angolo θ_i formato dai due segmenti incidenti in quel vertice, e si ponga il suo peso w_i come $\sin \frac{\theta_i}{2}$.

5. Ad ogni vertice non di angolo, si attribuisca peso 1.

6. Si ripetano k passi della suddivisione come segue:

   a. Si rimpiazzi ogni vertice d'angolo b_i con tre vertici: due di questi, marcati come vertici d'angolo, siano i vertici del V-taglio razionale di b_i, e il terzo, marcato come vertice non di angolo, sia il punto medio di questo V-taglio. Si tratta di un punto temporaneo appartenente a questo lato.

   b. Si ricalcoli il vettore peso del nuovo poligono. Notare che dalla geometria della figura il peso del vertice b_i^0 può essere dato da $\cos \alpha_i^0$, che è l'angolo alla base, in quanto abbiamo che $2\alpha_i^0 + \beta_i^0 = \pi$. Di conseguenza, al passo k del procedimento di suddivisione, il peso w_i^k dei nuovi angoli di Nasri-Farin può essere facilmente calcolato come:
   $$w_i^k = \cos \frac{\alpha_i^0}{2^{k+1}}$$

Dopo k suddivisioni, l'algoritmo genererà un poligono P_k con un vettore peso w_k dato da:

$$W_k = (..., 1, \cos \frac{\alpha_i^0}{2^k}, 1, \cos \frac{\alpha_{i+1}^0}{2^k}, ...)$$

Al limite per k tendente ad infinito, il poligono converge ad una spline circolare a tratti di classe G^1, dove ogni insieme di tre vertici consecutivi definisce un arco circolare parametrizzato dalla sua ascissa curvilinea. Questo è verificato dal momento che tutti i sottopoligoni generati sono isosceli per costruzione e hanno eguale lunghezza dei lati.

Descriviamo ora in maniera dettagliata l'algoritmo nel caso di un poligono di controllo chiuso. Si assuma che il poligono abbia n vertici e sia dato da $P_0 = (b_i)$, con $0 \leq i \leq n-1$.

1. Si inserisca il punto medio m_i di ogni segmento $b_i b_{i+1}$ del poligono di controllo (nel caso di un poligono aperto, si escludano il primo e l'ultimo segmento).

2. Si marchino gli m_i come vertici non di angolo e i b_i come vertici d'angolo.

3. Si rinumerino i vertici del nuovo poligono come $P_0 = (b_i)$, con $0 \leq i \leq 2n-1$.

4. Per ogni vertice d'angolo b_{2i} (nel caso di un poligono aperto, si usi b_{2i+1}) si effettuino i seguenti passi:

   a. Posto $Q(j) = j \ mod \ 2n$, si calcolino:

$$\lambda_{2i} = \frac{|b_{Q(2i-1)}b_{2i}| + |b_{Q(2i+1)}b_{2i}|}{|b_{Q(2i-1)}b_{Q(2i+1)}| + |b_{Q(2i-1)}b_{2i}| + |b_{Q(2i+1)}b_{2i}|}$$

$$\beta_{2i} = \frac{(1-\lambda_{2i})|b_{Q(2i-1)}b_{2i}|}{\lambda_{2i}|b_{Q(2i-1)}b_{Q(2i+1)}|}$$

$$\gamma_{2i} = \lambda_{2i}$$

   b. Si calcolino i punti b^0_{2i}, b^1_{2i}, e b^2_{2i} come:

$$b^0_{2i} = b_{2i} + \lambda_{2i}(b_{Q(2i-1)} - b_{2i})$$

$$b^2_{2i} = b_{2i} + \lambda_{2i}(b_{Q(2i+1)} - b_{2i})$$

$$b^1_{2i} = b^0_{2i} + \beta_{2i}(b^2_{2i} - b^0_{2i})$$

   c. Si rimpiazzi b_{2i} con i tre vertici b^0_{2i}, b^1_{2i}, b^2_{2i}. Si noti che $b^0_{2i}b^2_{2i}$ sarà il V-taglio razionale di b_{2i}.

   d. Si marchino b^0_{2i} e b^2_{2i} come vertici d'angolo e b^1_{2i} come vertici non di angolo.

5. Si ponga $n_1 = 4n$ (nel caso di poligono aperto $n_1 = 4n-7$), cioè il numero di vertici del poligono di controllo risultante.

6. Si rinumerino i vertici e si dia ad essi un apice 0 cominciando da b_0^2 che diventa b_0^0. Il poligono risultante P_0 sarà dato da: $P_0 = (b_i^0)$, con $0 \leq i \leq n_1 - 1$.

7. Si definisca l'operatore R come $R(i) = i \; mod \; n_1$. Nel caso del poligono di controllo aperto, si ponga R come la funzione identica.

8. Si ponga per iniziare $i = 0$, e, aumentando ogni volta di 2 il valore dell'indice, con la condizione $i < n_1 - 1$, si eseguano le seguenti operazioni. i abbia come valore iniziale 1 nel caso di un poligono di controllo aperto.
$$\alpha_i^0 = \langle (b_{R(i-1)}^0, b_i^0, b_{i+1}^0)$$
$$w_i^0 = \cos \frac{\alpha_i}{2}$$
$$w_{R(i-1)}^0 = 1$$

9. Nel caso di poligono di controllo aperto, si ponga $w_{n_1-1}^0 = 1$

10. Si ponga come valore iniziale $k = 1$ e lo si aumenti di 1 ogni volta, fino a raggiungere il numero di suddivisioni. Per ogni valore di k, si definisca innanzitutto $S(j) = j \; mod \; (2n_k - 1)$, dove abbiamo già definito n_1, e definiremo gli altri n_k all'interno del ciclo, in modo tale che al momento in cui viene richiamato per un dato k sia già stato definito. Nel caso di poligono di controllo aperto, sia S la funzione identica.
    A questo punto, per ogni i che va da 0 a $n_k - 1$, si effettuino i seguenti passi:

    a. Si definisca $t(i) = i \; mod \; n_k$

    b. Si ponga: $b_{S(4i-1)}^k = \frac{w_{t(2i-1)}^{k-1} b_{t(2i-1)}^{k-1} + w_{2i}^{k-1} b_{2i}^{k-1}}{w_{t(2i-1)}^{k-1} + w_{2i}^{k-1}}$

    c. $w_{S(4i-1)}^k = \cos \frac{\alpha_i^0}{2^{k+1}}$

    d. $b_{4i+1}^k = \frac{w_{2i}^{k-1} b_{2i}^{k-1} + w_{2i+1}^{k-1} b_{2i+1}^{k-1}}{w_{2i}^{k-1} + w_{2i+1}^{k-1}}$

    e. $w_{S(4i+1)}^k = w_{S(4i-1)}^k$

    f. $b_{4i+2}^k = b_{2i+1}^{k-1}$, in modo tale da preservare i punti.

    g. $b_{4i}^k = \frac{1}{2}(b_{S(4i-1)}^k + b_{4i+1}^k)$

    h. $w_{4i}^k = w_{4i+2}^k = 1$

    Terminato il ciclo su i, si ponga $n_k = 2n_{k-1}$ e si proceda per il valore di k successivo, iniziando con la ridefinizione di $S(j)$.

    Nel caso di un poligono di controllo aperto, il ciclo interno viene sostituito dal seguente, per ogni i che va da 0 a $n_k - 3$:

a. $b^k_{4i+1} = \frac{w^{k-1}_{2i} b^{k-1}_{2i} + w^{k-1}_{2i+1} b^{k-1}_{2i+1}}{w^{k-1}_{2i} + w^{k-1}_{2i+1}}$

b. $w^k_{S(4i+1)} = \cos \frac{\alpha^0_{2i+1}}{2^{k+1}}$

c. $b^k_{4i+3} = \frac{w^{k-1}_{2i+1} b^{k-1}_{2i+1} + w^{k-1}_{2i+2} b^{k-1}_{2i+2}}{w^{k-1}_{2i+1} + w^{k-1}_{2i+2}}$

d. $w^k_{S(4i+3)} = w^k_{S(4i+1)}$

e. $b^k_{4i} = b^{k-1}_{2i}$, in modo tale da preservare i punti.

f. $b^k_{4i+1} = \frac{1}{2}(b^k_{4i+1} + b^k_{4i+3})$

g. $w^k_{4i} = w^k_{4i+2} = 1$

Terminato il ciclo su i, si ponga $n_k = \frac{3n_{k-1}+2}{2}$ e si proceda per il valore di k successivo.

Si noti che, fondamentalmente, le differenze principali tra il caso di un poligono chiuso, e quello di un poligono aperto, consistono nel fatto che, nel caso aperto, il primo e l'ultimo vertice vengono considerati vertici non di angolo, e non viene richiesto alcun inserimento di punto medio per il primo e per l'ultimo segmento del poligono di controllo. Inoltre, i vertici d'angolo vengono indicati come b_{2i+1}, al fine che i passi successivi dell'algoritmo diano risultato corretto.
In figura 24 è mostrato un esempio grafico dell'applicazione di tale algoritmo, sia nel caso di poligono di controllo aperto che nel caso chiuso.

5.4 Alcune applicazioni

L'algoritmo proposto ha diverse applicazioni, alcune delle quali immediate. Un esempio potrebbe essere quello di produrre una curva a tratti composta da biarchi che interpoli una sequenza arbitraria di punti; un altro quello di generare superfici delimitate da spline circolari a tratti. Approfondiamo il primo caso.

5.4.1 Generazione di una curva interpolatoria a tratti composta da biarchi

Si consideri una sequenza di punti (b_{2i}), con $0 \leq i \leq n$, e un vettore tangente dato $\overrightarrow{t_0}$ in b_0. Si eseguano i seguenti passi:

1. Si inserisca un punto b_1 sulla tangente $\overrightarrow{t_0}$. Si noti che si ha un grado di libertà nell'inserimento del punto; una possibile scelta nel posizionarlo potrebbe essere quella effettuata per minimizzare la variazione di curvatura in corrispondenza delle giunzioni dei tratti di arco.

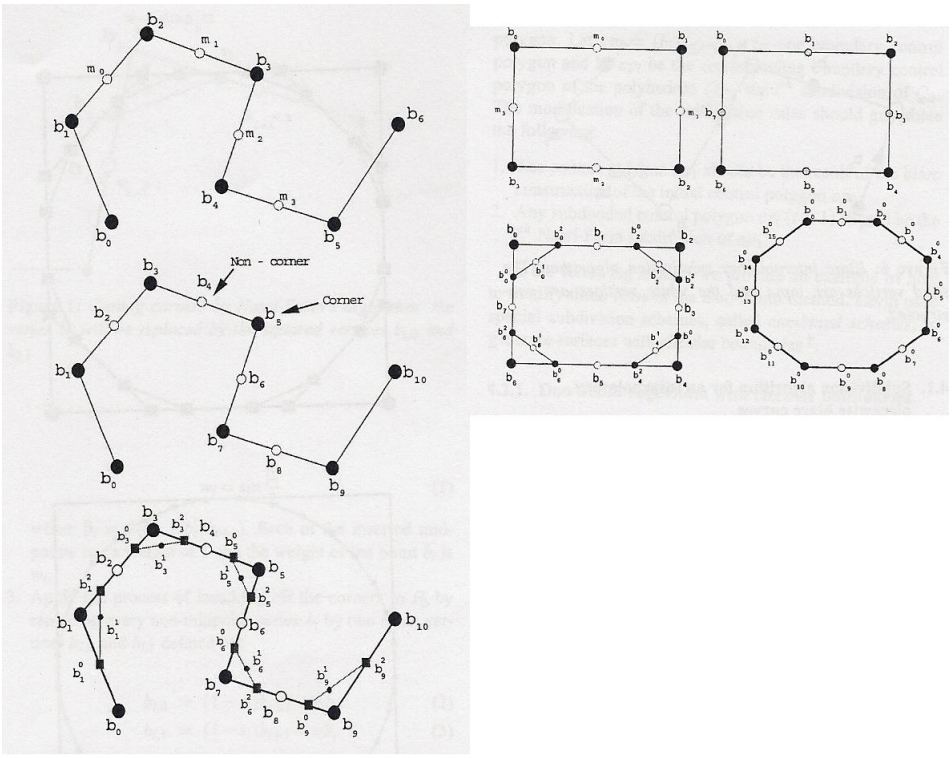

Figura 24: Primi due passi dell'algoritmo, nel caso aperto a sinistra e nel caso chiuso a destra. Il poligono di controllo iniziale, quello dopo l'inserimento dei punti medi temporanei come vertici non di angolo, e quello dopo la costruzione dei biarchi e la rinumerazione dei vertici. [4]

2. Per ogni i successivo, si inseriscano punti b_{2i+1} sulla retta passante per b_{2i-1} e b_{2i}.

3. Si costruisca un biarco per ogni insieme di tre punti consecutivi b_{2i}, b_{2i+1}, b_{2i+2}, ottenendo un nuovo poligono.

4. Si applichi l'algoritmo di suddivisione al poligono di controllo risultante.

I passi visti sopra descrivono un algoritmo di suddivisione che genera una curva a tratti composta da biarchi che interpola una sequenza arbitraria di punti. La curva risultante è di classe G^1 per costruzione. Un vantaggio dello svolgimento di questo procedimento tramite suddivisione è la possibilità di incorporare il nuovo algoritmo in un sistema ricorsivo di suddivisione, in modo tale da generare contorni circolari da generare nella costruzione di superfici.

5.4.2 Uso pratico degli algoritmi

Oltre alle applicazioni degli algoritmi proposti di cui abbiamo parlato prima, ampio uso pratico di questi procedimenti viene fatto nel CAD, nel CAGD e nel CAM. Ne mostreremo alcuni:

1. Un'operazione che talvolta può essere utile, anche durante la fase progettuale, è la costruzione di curve di offset esatte. Questo procedimento permette di produrre in maniera semplice curve spesse o renderizzate con effetti come per esempio l'ombra; nè per le B-spline, nè per le B-spline razionali, la curva di offset risulta essere una B-spline, e pertanto la sua costruzione è in generale di difficoltà superiore, sempre ammesso che sia possibile svolgerla. Tuttavia, nel caso dei biarchi circolari, la costruzione di una curva di offset è molto semplice, dal momento che è sufficiente cambiarne il raggio mantenendo fisso il centro.

2. Le B-spline, ma anche le B-spline razionali, non godono di una parametrizzazione basata sulla lunghezza della corda descrivibile in maniera semplice. Pertanto, prendendo una qualsiasi parametrizzazione di facile utilizzo, la lunghezza del vettore tangente varia continuamente. Nel caso dei biarchi circolari, tuttavia, la lunghezza del vettore tangente è costante a tratti, ovvero, per ogni tratto di curva, la parametrizzazione secondo l'ascissa curvilinea assume una forma particolarmente semplice. Ciò è particolarmente utile nel caso in cui i vettori tangenti siano utilizzati per la costruzione della curva.

3. In molti contesti di progettazione, è necessario tenere in considerazione vincoli di tipo geometrico. Nel caso di vincoli olonomi, è possibile utilizzare spline polinomiali o razionali, ma nel caso di vincoli anolonomi, come le relazioni di tangenza, non sono consigliate, e d'altro canto sistemi di equazioni non lineari da essere risolti numericamente possono essere computazionalmente pesanti. Tuttavia, nel caso di segmenti circolari a tratti, dove la posizione dei centri è conosciuta analiticamente, queste relazioni di tangenza possono essere espresse in maniera semplice.

4. Abbiamo visto che l'algoritmo restituisce, oltre che i tratti d'arco, anche i loro centri. Possiamo sfruttare questa proprietà per modificare passo passo una curva fino a farle raggiungere la forma desiderata: a partire da una forma iniziale, che può essere specificata da una sequenza di punti di controllo, la si può modificare spostando uno o più centri, con eventuali limitazioni se si desidera che lo spostamento di uno dei centri faccia mantenere la continuità G^1 con uno dei due segmenti adiacenti. Ulteriormente, si suppongano i vari segmenti identificati con

i, dove i varia da 1 a n definito come il numero di segmenti. Posta in questo modo la notazione, il procedimento per lo spostamento del segmento j, dove $1 < j < n$, può essere implementato in modo tale che una delle due metà della curva, sia essa quella che contiene i segmenti da 1 a j, oppure quella che contiene quelli da j a n, sia fatta slittare in modo tale che l'intera curva mantenga la proprietà di essere di classe G^1.

Questo genere di operazioni è di difficile applicazione, o talvolta perfino non praticabile, su B-spline a tratti o su B-spline razionali a tratti, mentre è invece ben più facilmente implementabile su questo tipo di curve.

6 Conclusioni

In questa tesi abbiamo analizzato l'algoritmo di Chaikin, alcune sue applicazioni, un suo diretto miglioramento, e infine un algoritmo da esso derivato. Abbiamo visto come dall'idea originaria nata in ambito informatico agli albori dell'era dei computer sia stato possibile ottenere procedure per la costruzione di curve B-spline quadratiche e cubiche, algoritmi per la generazione di curve circolari, e altri tipi di operazioni molto utili nel campo della grafica computerizzata. Nonostante per un certo periodo di tempo successivo alla pubblicazione, questo algoritmo fosse stato piuttosto accantonato, dal momento che la ricerca era concentrata sullo studio diretto delle B-spline, i ricercatori sono successivamente andati oltre la rigidità di modelli per curve e superfici basati su una stretta forma analitica, permettendo di sviluppare nuove procedure a partire da questo seppur semplice ma all'epoca rivoluzionario algoritmo.

Riferimenti bibliografici

[1] CHAIKIN George Merrill. *An algorithm for high speed curve generation*, Computer Graphics and Image Processing, 346-349, 1974.

[2] JOY Kenneth I., *Chaikin's Algorithms for Curves*, http://www.cipic.ucdavis.edu/education/CAGDNotes/Chaikins-Algorithm.pdf .

[3] NASRI Ahmad H., FARIN Gerald. *A Subdivision Algorithm for Generating Rational Curves*, Journal of Graphics Tools 6(1), 35-47, 2001.

[4] NASRI Ahmad H., VAN OVERVELD C.W.A.M., WYVILL Brian. *A Recursive Subdivision Algorithm for Piecewise Circular Spline*, Computer Graphics Forum 20(1), 35-45, 2001.

[5] SALOMON David. *Curves and Surfaces for Computer Graphics*, Springer-Verlag, 2006.

[6] WU Ling, YONG Jun-Hai, ZHANG You-Wei, ZHANG Li. *Multi-step Subdivision Algorithm for Chaikin Curves*, CIS 2004, Springer-Verlag, 1232-1238, 2004.