UNIVERSITATEA POLITEHNICA TIMIŞOARA

ing. Călin Stoicanescu

SISTEME BAZATE PE CUNOAŞTERE ÎN CONDUCEREA ROBOŢILOR

TEZA DE DOCTORAT –

Conducător ştiinţific:
Prof.Dr.Ing. Doina Drăgulescu

Timişoara
2004

Cuprins

Cap 1 Introducere

Scopul abordărilor bazate pe cunoştinţe este codificarea cunoaşterii prin simboluri şi efectuarea de prelucrări asupra simbolurilor, prelucrări ce corespund proceselor de judecată ale fiinţelor umane. Aceste abordări reprezintă încercări de adaptare la strategia de rezolvare a problemelor într-un mod asemănător omului, pentru a permite utilizarea calculatoarelor în domenii în care teoriile exacte lipseau şi unde era disponibilă doar experienţa pe termen lung a "experţilor" umani. Astfel, tehnicile bazate pe cunoaştere sunt aplicate cu succes în domeniul procesării imaginii şi vorbirii, ca şi în domeniul roboticii şi inteligenţei artificiale.

Modelele virtuale ale elementelor reale înglobează toate cunoştinţele legate de aceste elemente încercând să surprindă cât mai fidel realitatea modelată. Sensul proiectării spaţiilor virtuale îl constituie determinarea comportării elementelor componente la situaţii şi stimuli complecşi greu de simulat în realitate care pot afecta în mod esenţial structura acestor elemente precum şi verificarea anumitor principii şi reguli pentru aceste entităţi care pot fi testate într-un mod simplu şi eficient înainte de a fi implementate în realitate. Simularea comportării sistemelor complexe cu ajutorul spaţiilor virtuale permite cercetarea unor concepte noi, a validităţii lor practice, felul în care acestea reacţionează în diferite situaţii, descoperirea punctelor slabe sau a erorilor de proiectare, toate acestea cu un efort mult mai redus. Modelul virtual nu poate înlocui deplin sistemul real care îi corespunde dar poate crea o imagine clară asupra acestuia şi o predicţie veridică a modului în care reacţionează.

Una din temele principale ale roboticii este planificarea mişcării roboţilor mobili. Acest proces implică multe metode specifice inteligenţei artificiale. Pentru operaţii simple de manipulare, sistemul de comandă generează traiectoriile de mişcare între două puncte ale spaţiului de lucru şi controlează mişcarea robotului pe această traiectorie. Deoarece robotul operează într-un spaţiu, populat cu obiecte fizice, trebuie să fie capabil să-şi planifice şi genereze propriile mişcări, să-şi activeze cuplele cinematice în acord cu obiectivul sarcinii, în funcţie de aranjamentul iniţial şi final al obiectelor din spaţiul de lucru. Crearea unui model virtual în acest domeniu permite simularea cât mai reală a conducerii robotului şi a comportării sale. Modul în care interacţionează cu mediul şi capacitatea sa de a-şi îndeplini sarcinile definesc calitativ sistemul de comandă al robotului.

Proiectarea unui model virtual pentru testarea mişcării roboţilor mobili necesită existenţa unui model al robotului, inclus în modelul spaţiului de lucru. Reprezentarea realistică conduce la ideea folosirii unui model tridimensional pentru robot şi mediul său, model care să surprindă cât mai exact proprietăţile geometrice ale tuturor elementelor spaţiului de lucru. Domeniul proiectării asistate de calculator CAD şi al prelucrării asistate de calculator CAM a cunoscut în ultimii ani o dezvoltare deosebită. Evoluţia de la reprezentarea wireframe la modelarea prin suprafeţe poligonale complexe a permis obţinerea unor modele corecte ale obiectelor procesate şi utile din punct de vedere practic. Anumite domenii impun însă reprezentări mai fine ale corpurilor ceea ce a condus la proiectarea prin curbe şi suprafeţe curbe. Varietatea formelor din lumea reală este modelată mult mai exact în acest mod. Aceste principii de modelare pot fi aplicate şi în domeniul modelării roboţilor. Lucrarea se constituie ca o încercare în această direcţie. Crearea unui laborator virtual populat de roboţi mobili folosindu-se metodele de proiectare prin suprafeţe

curbe permite generarea unui spațiu virtual care se apropie de realitatea modelată. Simularea mişcării roboților în cadrul laboratorului virtual poate constitui o modalitate eficientă în procesul de planificare a roboților.

Începuturile modelării geometrice prin curbe şi suprafețe au fost strâns legate de domeniul proiectării formei maşinilor. Datorită caracteristicelor lor, aceste metode de proiectare sunt utilizate acum într-un spectru larg de domenii. Editoarele de texte şi imagini includ funcții spline, modelarea virtuală 3D din orice domeniu foloseşte aceste principii, mediile CAD cuprind elemente de suprafețe curbe. Acest lucru a impus încorporarea de funcții spline în cadrul librăriilor grafice cunoscute precum şi a limbajelor standardizate de modelare virtuală. Astfel, librăria OpenGL pune la dispoziție propriile sale funcții de vizualizare a suprafețelor curbe, de asemenea şi formatul standard Virtual Reality Modeling Language VRML de reprezentare 3D include facilități de reprezentare a suprafețelor curbe.

În cadrul tezei de doctorat s-a încercat implementarea şi folosirea unor algoritmi proprii de proiectare a curbelor şi suprafețelor pornind de la baza teoretică existentă în acest domeniu. S-a folosit această abordare în locul utilizării facilităților oferite de librăriile grafice sau limbajele de modelare 3D deoarece aceste funcții au în principal o orientare vizuală, urmărind modul de reprezentare cât mai real al elementelor curbe, în timp ce în lucrare, accentul este pus pe determinarea exactă a formei obiectelor laboratorului virtual. Pentru fiecare element s-a creat un model geometric cât mai corect, păstrându-se raporturile, dimensiunile, poziția şi orientarea în spațiul de lucru. De asemenea modelarea mişcării roboților a impus utilizarea unui set de funcții de intersecție de suprafețe, funcții proiectate în strânsă legătură cu modul de generare al acestor suprafețe. Analiza stării de coliziune a corpurilor laboratorului virtual ar fi fost mai dificil de realizat prin folosirea funcțiilor librăriei grafice deoarece acestea nu conțin algoritmi matematici strict dedicați intersecțiilor de suprafețe. Toate aceste elemente au condus la modelarea geometrică a roboților şi a obiectelor laboratorului virtual prin algoritmi proprii.

Lucrarea este structurată în cinci capitole. Capitolul 1 *Introducere* descrie în linii generale tema lucrării, modul de abordare şi soluția aleasă pentru rezolvarea sa. De asemenea sunt schițate motivele pentru care a fost studiată această temă şi pentru alegerea dezvoltării ei. Toate aceste elemente sunt detaliate pe parcursul lucrării.

Capitolul 2 *Principii de modelare geometrică prin curbe* prezintă noțiunile de bază pentru generarea curbelor şi tehnicile de implementare software utilizate. Descrierea avansează de la metodele de bază şi cu un grad de complexitate mai mic către metodele mai vast elaborate dar care permit proiectarea unor forme mult mai sofisticate. O altă direcție avută în vedere pe parcursul capitolului o constituie stabilirea distincției dintre metodele care realizează o aproximare a formei inițiale şi metodele care interpolează forma dorită. Marea majoritate a metodelor de generare au fost utilizate în procesul de proiectare a laboratorului virtual, celelalte au constituit o punte necesară pentru înțelegerea şi dezvoltarea algoritmilor mai avansați. Fiecare principiu de construcție este însoțit de codul sursă corespunzător.

Capitolul 3 *Principii de modelare geometrică prin suprafețe* cuprinde metodele de proiectare a suprafețelor. Se realizează o structurare a tipurilor de suprafețe în raport cu diferite criterii. Sunt definite suprafețele analitice şi sintetice, suprafețele care aproximează harta de control şi cele interpolatoare. Toate metodele de generare a suprafețelor sintetice reprezintă dezvoltări a metodelor de construcție a curbelor descrise în capitolul 2. Sunt prezentate soluțiile software utilizate pentru implementarea acestor algoritmi. Proiectarea a fost abordată din perspectiva minimizării timpului de execuție, criteriu impus de faptul că tema lucrării cuprinde elemente de dinamică care necesită procesarea unor algoritmi matematici în perioade de timp foarte scurte.

Capitolul 4 *Proiectarea unui laborator virtual deservit de roboţi mobili* realizează o descriere detaliată a modului de proiectare a spaţiului virtual folosind baza teoretică prezentată în capitolele anterioare. Laboratorul constituie modelul virtual al Bazei de Cercetare cu Utilizatori Multiplii *Centrul de Modelare a Protezării şi Intervenţiilor Chirurgicale asupra Scheletului Uman - CMPICSU* din cadrul Universităţii „Politehnica" din Timişoara. Toate obiectele din cadrul spaţiului laboratorului, structura acestuia precum şi roboţii sunt construiţi din suprafeţe generate cu metodele dezvoltate. Implementarea software optimă pentru această aplicaţie este programarea obiectuală în sistemul Windows. Proprietăţile acestui mod de programare, mecanismul de proiectare al aplicaţiei, etapele parcurse în realizarea modelului sunt detaliate în cadrul capitolului. Vizualizarea realistă a întregului spaţiu virtual se realizează folosind capabilităţile oferite de biblioteca de funcţii grafice OpenGL. Aceasta permite construirea unei scene complexe prin organizarea efectelor de lumină şi a proprietăţilor de desenare a obiectelor şi facilitează o vizualizare completă a scenei din diverse unghiuri şi poziţii. De asemenea în cadrul capitolului este descris mecanismul de mişcare al roboţilor în spaţiul laboratorului virtual. Se permite atât o mişcare liberă a roboţilor cât şi orientată spre îndeplinirea unei anumite sarcini.

Capitolul 5 *Concluzii şi contribuţii personale* sintetizează elementele esenţiale din cuprinsul tezei de doctorat, fiind subliniate concluziile la care s-a ajuns în urma realizării lucrării şi evidenţiate contribuţiile originale din întreaga teză.

Cap 2 Principii de modelare geometrică prin curbe

2.1 Noțiuni generale

Obiectele din lumea reală pot fi modelate geometric prin intermediul curbelor şi suprafețelor. Definirea matematică a acestora este singura modalitate prin care pot fi implementate pe calculator.

Ca prim pas în modelarea matematică a unui obiect, se defineşte sistemul de coordonate al spațiului în care va fi descris obiectul. Acest spațiu nu posedă un sistem de coordonate preferențial ci el trebuie definit astfel încât să nu afecteze nici una dintre proprietățile intrinseci ale obiectelor. Metodele de modelare dezvoltate trebuie să fie independente de alegerea sistemului de coordonate [37][77][81][82][84]. Acest concept motivează distincția strictă dintre puncte şi vectori.

Un punct identifică o locație şi sunt elemente ale spațiului euclidian tridimensional E^3. Vectorii sunt elemente ale spațiului (vectorial) liniar tridimensional R^3. Atât punctele cât şi vectorii sunt descrise prin triplete de numere reale în raport cu un sistem de coordonate stabilit. Pentru puncte acestea reprezintă coordonatele lor în spațiul tridimensional, iar pentru vectori, direcția lor în cadrul aceluiaşi spațiu. Există astfel, o distincție clară între puncte şi vectori dar şi o legătură puternică. Pentru orice două puncte a şi b, se defineşte un singur vector $\mathbf{v}$ al cărui sens este de la a la b: $\mathbf{v} = b - a \quad a,b \in E^3 \quad \mathbf{v} \in R^3$. Pe de altă parte, fiind dat un vector $\mathbf{v}$, există o infinitate de perechi de puncte a şi b astfel încât $\mathbf{v} = b - a$. Dacă a şi b reprezintă o astfel de pereche iar $\mathbf{w}$ este un vector arbitrar, atunci $a + \mathbf{w}, b + \mathbf{w}$ reprezintă o altă pereche pentru care $\mathbf{v} = (b + \mathbf{w}) - (a + \mathbf{w})$ este acelaşi. Asociind punctul $a + \mathbf{w}$ oricărui punct $a \in E^3$ se realizează de fapt o translație, iar afirmația anterioară arată că vectorii sunt invarianți la translații pe când punctele nu.

Elementele spațiului E^3 pot fi doar scăzute, iar rezultatul operației este un vector. Ele nu pot fi adunate. Această operație nu este definită pentru puncte ci doar pentru vectori. O operației similară celei de adunare, pentru puncte este *combinația baricentrică* (combinația afină) [30][77][80][87]. Aceasta este suma ponderată a punctelor, unde suma ponderilor este 1:

$$b = \sum_{j=0}^{n} \alpha_j b_j \quad b_j \in E^3, \quad \alpha_0 + \cdots + \alpha_n = 1 \tag{2.1.1}$$

Relația arată ca o sumă nedefinită de puncte, dar se poate rescrie ca suma unui punct cu un vector:

$$b = b_0 + \sum_{j=0}^{n} \alpha_j (b_j - b_0) \tag{2.1.2}$$

Un caz important de combinaţii baricentrice îl reprezintă *combinaţiile convexe*. Acestea sunt combinaţii baricentrice care au toţi coeficienţii α_i pozitivi, pe lângă faptul că suma lor este 1. O combinaţie convexă de puncte este întotdeauna "in interiorul" punctelor generatoare, observaţie care prefaţează definiţia noţiunii de *acoperire convexă* a unei mulţimi de puncte. Aceasta este mulţimea alcătuită din toate combinaţiile convexe ale punctelor respective. Mai intuitiv, acoperirea convexă a unei mulţimi se obţine în felul următor: pentru o mulţime 2D se defineşte un şir de puncte care este circumscris în jurul acestei mulţimi, şir ce devine frontiera acoperirii convexe.

Acoperirea convexă a unei mulţimi de puncte este o mulţime convexă. O astfel de mulţime este caracterizată prin faptul că pentru oricare două puncte din mulţime, linia dreaptă care uneşte aceste puncte este de asemenea conţinută în această mulţime [74][87]. Exemple de mulţimi convexe sunt elipsa şi paralelogramul.

2.1.1 Relaţii afine

Cele mai multe din transformările care sunt folosite pentru a poziţiona sau scala un obiect într-un spaţiu sunt *relaţii afine*. Termenul se datorează lui L. Euler. Relaţiile afine au fost prima dată studiate în mod sistematic de F. Moebius [56][76][82].

Operaţia fundamentală pentru puncte este combinaţia baricentrică pe care se bazează definirea relaţii afine se bazează pe noţiunea de combinaţie baricentrică. *O relaţie ϕ ce relaţionează E^3 în el însuşi este numită o relaţie afină dacă combinaţiile baricentrice rămân invariante.* Dacă

$$x = \sum \alpha_j a_j \quad x, a_j \in E^3 \tag{2.1.3}$$

şi dacă ϕ este o relaţie afina, atunci

$$\phi(x) = \sum \alpha_j \phi(a_j) \quad \phi(x), \phi(a_j) \in E^3 \tag{2.1.4}$$

Această definiţie este pur abstractă, având o interpretare simplă. Expresia $x = \sum \alpha_j a_j$ reprezintă o ponderare a punctelor a_j, ponderea lor medie fiind x. Expresia este validă şi dacă se aplică o relaţie afină tuturor punctelor a_j şi lui x. Ca exemplu, punctul de mijloc al unui segment de dreaptă relaţionat la punctul de mijloc al imaginii afine al segmentului respectiv. De asemenea, centroidul unui număr de puncte poate fi relaţionat la centroidul imaginii afine a punctelor respective.

Forma generală a unei relaţii afine luând în considerare modul de reprezentare a punctelor într-un sistem de coordonate este:

$$\phi(x) = Ax + \mathbf{v} \tag{2.1.5}$$

unde A este o matrice 3 x 3 şi **v** este un vector în R^3.

Câteva exemple de relaţii afine:

Identitatea: $\mathbf{v} = 0$, vectorul zero şi $A = I$, matricea identitate.
Translaţia : $A = I$, matricea identitate şi un *vector de translaţie* **v**.
Scalarea : $\mathbf{v} = 0$ şi matricea diagonală A. Coeficienţii matricei definesc modul în care fiecare componentă a punctului este scalată.

Rotația : în jurul axei z, $\mathbf{v} = 0$ şi

$$A = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1.6}$$

Un caz important al relațiilor afine sunt relațiile euclidiene, utilizate în formalismul exprimării mişcării corpurilor rigide. Ele sunt caracterizate de matrice A ortonormale, definite de proprietatea $A^T A = I$. Relațiile euclidiene păstrează dimensiunile şi unghiurile nemodificate [84][95]. Cele mai importante sunt rotațiile şi translațiile.

Relațiile afine simple pot fi combinate, iar o relație complexă poate fi descompusă într-o secvență de relații simple. Orice relație afină poate fi compusă din translații, rotații, rotunjiri şi scalări.

Rangul matricei A are o importantă interpretare geometrică: dacă $rang(A) = 3$, atunci relația afină ϕ relaționează obiectele tridimensionale la obiecte tridimensionale. Dacă rangul este mai mic decât 3, ϕ este o proiecție paralelă într-un plan (grad = 2) sau într-o linie (grad = 1). O relație afină din E^2 în E^2 este univoc determinată de un triunghi şi de imaginea lui. Oricare două triunghiuri determină o relație afină a planului în el însuşi. În E^3, o relație afină este univoc definită de un tetraedru şi imaginea sa.

2.1.2 Interpolarea liniară

Fie a, b două puncte distincte în E^3. Determinarea tuturor punctelor $x \in E^3$ de forma :

$$x = x(s) = (1-s)a + sb; \quad s \in R \tag{2.1.7}$$

este numită *linia dreaptă* determinată de a şi b. Oricare trei puncte de pe o linie dreaptă sunt coliniare. Pentru $s = 0$ linia dreaptă trece prin a şi pentru $s = 1$ trece prin b. Pentru $0 \le s \le 1$, punctul x este între a şi b, în timp ce pentru toate celelalte valori ale lui s este în exterior. Punctul împarte linia dreaptă în raportul $s : 1-s$. Relația (2.1.7) defineşte punctul x ca o combinație baricentrică de două puncte din E^3· Aceeaşi combinație baricentrică este valabilă şi pentru cele trei valori 0, s, 1 din E^1: $s = (1-s) * 0 + s * 1$. Deci s este raportat la 0 şi 1 prin aceeaşi combinație baricentrică care leagă x de a şi b. Astfel, pornind de la definiția relațiilor afine rezultă că cele trei puncte a, x, b din spațiul tridimensional sunt o relație afină a trei puncte 0, t, 1 din spațiul unidimensional. Astfel *interpolarea liniară* este o relație afină a liniei reale cu o linie dreaptă în E^3 [11][12].

În mod evident interpolarea liniară este invariant afină. Pentru orice relație afină $\phi : E^3 \to E^3$ se obține:

$$\phi(x) = \phi((1-s)a + sb) = (1-s)\phi(a) + s\phi(b) \tag{2.1.8}$$

Strâns legat de interpolarea liniară este conceptul de *coordonate baricentrice*,

datorat lui Moebius [2][76][95] Fie a, x, b trei puncte coliniare în E^3 cu proprietatea :

$$x = \alpha a + \beta b, \quad \alpha + \beta = 1 \tag{2.1.9}$$

α şi β se numesc coordonate baricentrice ale lui x , în raport cu a şi b. Conexiunea între coordonatele baricentrice şi interpolarea liniară este evidentă: $\alpha = 1 - t, \beta = t$. Astfel coordonatele baricentrice pot lua şi valori negative. Pentru $t \notin [0,1]$ fie α, fie β sunt negative. Pentru oricare trei puncte coliniare a, b, c coordonatele baricentrice ale lui b în raport cu a şi c sunt :

$$\alpha = \frac{vol_1(b,c)}{vol_1(a,c)}, \quad \beta = \frac{vol_1(a,b)}{vol_1(a,c)} \tag{2.1.10}$$

vol_1 indică volumul unidimensional, care este distanţa cu semn între două puncte. Coordonatele baricentrice pot fi definite nu numai pe o linie dreaptă, ci şi pe un plan.

Un alt concept important în acest context este cel al raporturilor. Raportul a trei puncte coliniare a, b, c este definit de :

$$raport(a,b,c) = \frac{vol_1(a,b)}{vol_1(b,c)} = \frac{\alpha}{\beta} \tag{2.1.11}$$

α şi β fiind coordonatele baricentrice ale lui b în raport cu a şi c. Coordonatele baricentrice ale unui punct nu se modifică în urma transformărilor afine. Astfel, raportul a trei puncte coliniare :

$$raport(\phi a, \phi b, \phi c) = \frac{\beta}{\alpha} \tag{2.1.12}$$

nu este afectat de transformarea afină ϕ. Relaţia (2.1.12.) exprimă faptul că relaţiile afine sunt păstrătoare de raport. Această proprietate poate fi folosită pentru definirea relaţiilor afine. Fiecare relaţie ce conectează linii drepte cu linii drepte şi este păstrătoare de raport, este o relaţie afină.

Conceptul conservării raportului poate fi utilizat pentru a arăta o altă proprietate utilă a interpolării liniare. Segmentul de dreaptă $[a,b]$ a fost definit ca fiind imaginea afină a intervalului unitar $[0,1]$. El poate fi de asemenea considerat ca imagine afină a oricărui domeniu $[m,n]$. Intervalul $[m,n]$ poate fi obţinut printr-o relaţie afină din intervalul $[0,1]$ sau invers. Pentru $s \in [0,1], q \in [m,n]$ relaţia afină este de forma $s = (q-a)/(b-a)$. Punctul interpolat pe linia dreaptă, dat de relaţia (2.1.7) poate fi exprimat şi prin relaţia:

$$x(q) = \frac{b-q}{b-a} a + \frac{q-a}{b-a} b \tag{2.1.13}$$

Valorile m, q, n şi 0, s, 1 sunt în acelaşi raport ca şi tripletul a, x, b. Relaţiile conduc la faptul că interpolarea liniară este invariantă la transformări afine de parametrii. Parametrul s este uneori numit parametru local al intervalului $[m,n]$.

Legătura între interpolarea liniară şi conceptul raporturilor poate fi sintetizată astfel: afirmaţia geometrică, trei puncte coliniare aflate într-un anumit raport, poate fi exprimată algebric ca interpolare liniară a unui punct în raport cu celelalte două.

2.1.3 Interpolarea liniară pe porţiuni

Se consideră un poligon B dat prin vârfurile sale $b_0 \ldots b_n \in E3$. Poligonul este format dintr-o secvenţă de segmente de dreaptă, fiecare interpolând o pereche de puncte b_i, b_{i+1}. B constituie un interpolant liniar pe porţiuni (notat *PL)* al punctelor b_i. Dacă punctele b_i se află pe o curbă c, atunci B este numit *interpolant liniar pe porţiuni al curbei c* notat $B = PLc$ [13][74]. Interpolarea liniară pe porţiuni are ca proprietăţi:

- Invarianţa afină. Dacă curba c este relaţionată într-o curbă ϕc prin transformarea afină ϕ, atunci interpolantul liniar pe porţiuni al curbei ϕc este obţinut prin aplicarea transformării afine interpolantului liniar pe porţiuni original $PL\phi c = \phi PLc$
- Diminuarea variaţiei. Fie curba c , un interpolant liniar pe porţiuni PLc al curbei şi un plan arbitrar. Se defineşte *traversarea*(c) ca fiind numărul de traversări a planului de către curba c şi traversarea(PLc) numărul de traversări realizat de interpolantul liniar pe porţiuni în raport cu acel plan.
  Întotdeauna $traversarea(PLc) \le traversarea(c)$. Această proprietate se bazează pe o observaţie simplă: segmentul de dreaptă ce uneşte două puncte traversează un plan în cel mult un punct, în timp ce segmentul de curbă c care uneşte cele două puncte poate traversa planul de mai de mai multe ori.

2.1.4 Spaţii de funcţii

Pentru atingerea scopului urmărit se consideră câteva noţiuni din domeniul analizei funcţionale utilizate în cadrul modelării curbelor şi suprafeţelor [77][80][95]. Se consideră $C[a,b]$ grupul funcţiilor continue de valori reale definite pe intervalul $[a,b]$. Pentru funcţiile $f, g \in C[a,b]$ sunt definite adunarea şi înmulţirea cu o constantă prin relaţia :

$$(\alpha f + \beta g)(s) = \alpha f(s) + \beta g(s) \qquad pentru\ \forall s \in [a,b] \tag{2.1.14}$$

Luând în considerare definiţiile rezultă că $C[a,b]$ formează un spaţiu liniar. Acesta este adevărată şi pentru $C^k[a,b]$, grupul funcţiilor de valori reale definit pe $[a,b]$ de k ori continuu diferenţiabile. Pentru orice k, C^{k+1} reprezintă un subspaţiu al lui C^k.

Funcţiile $f_1, \ldots, f_n \in C[a,b]$ sunt liniar independente dacă $\sum c_i f_i = 0, \quad \forall s \in [a,b]$ implică $c_1 = \ldots = c_n = 0$.

O clasă de subspaţii ale domeniului $C[a,b]$ o reprezintă polinoamele de grad n:

$$P^n(s) = a_0 + a_1 s + a_2 s^2 + \ldots + a_n s^n \quad s \in [a,b] \tag{2.1.15}$$

Pentru n fixat, dimensiunea lui P^n este $n+1$. Fiecare polinom $p^n \in P^n$ este unic determinat de cei $n+1$ coeficienţi ai săi $a_0, \ldots, a_n$. Aceştia pot fi interpretaţi ca un vector

în spațiul liniar $n+1$ dimensional. De asemenea în spațiul P^n monoamele $1, s, s^2, \ldots, s^n$ reprezintă $n+1$ funcții liniar independente şi deci formează o bază pentru P^n [2][13][87].

O altă clasă importantă de subspații a lui $C[a,b]$ este cea a funcțiilor liniare pe porțiuni. Fie $a = s_0 < s_1 < \ldots < s_n = b$ o partiție a domeniului $[a,b]$. O funcție continuă liniară pe fiecare subinterval $[s_i, s_{i+1}]$ este numită funcție liniară pe porțiuni. Pentru o partiție fixă a domeniului $[a,b]$, funcțiile liniare pe porțiuni formează un spațiu liniar. O bază pentru acest spațiu este dată de funcțiile de acoperire $H_i(s)$, funcții liniare pe porțiuni cu proprietatea : $H_i(s_i) = 1$, $H_i(s_j) = 0$ *daca* $i \neq j$. O funcție liniară pe porțiuni f cu $f(s_j) = f_j$ poate fi scrisă :

$$f(s) = \sum_{j=0}^{n} f_j H_j(s) \tag{2.1.16}$$

Se definesc de asemenea operatorii liniari care asociază o funcție *Af* unei funcții date *f*. Un operator $A : C[a,b] \rightarrow C[a,b]$ este liniar dacă lasă combinațiile liniare invariante:

$$A(\alpha f + \beta g) = \alpha Af + \beta Ag \quad \alpha, \beta \in R \tag{2.1.17}$$

Un exemplu de operator liniar este operatorul de derivare care asociază derivata f' la o funcție dată f: $Af = f'$.

2.1.5 Implementarea software a curbelor

Proiectarea software obiectuală a elementelor 3D, la fel ca şi modelarea matematică, trebuie să pornească de la entitățile atomice ale corpurilor: punctele. Datorită similitudinilor dintre puncte şi vectori, atât la nivel de date cât şi la nivelul operațiilor permise, a fost construită o clasă de bază comună denumită Vector. Pentru utilizarea cât mai simplă a obiectelor clasei Vector, toți operatorii matematici au fost suprascriși, astfel încât operațiile matematice se realizează în acelaşi mod ca şi în cazul numerelor reale. Acest lucru uşurează scrierea în fişierele sursă a relațiilor precum şi verificarea corectitudinii lor. Definirea online a tuturor metodelor clasei reduce timpul de execuție a funcțiilor matematice care le utilizează.

```
class Vector
{
    double vect[3]; /// coordonatele punctului sau a vectorului

    public:
    Vector(double x = 0.0, double y = 0.0, double z = 0.0); /// constructorul clasei
    Vector(Vector &v);
    void operator=(double d); /// operatorul egal
    Vector operator+(Vector &v); /// operatorul de adunare cu un vector
    Vector operator+(double d);
    void operator+=(Vector &v);
    Vector operator-(Vector &v); /// operatorul de scădere
    void operator-=(Vector &v);
    Vector operator*(double d); /// operatorul de înmulțire
    void operator*=(double d);
    Vector operator/(double d); /// operatorul de împărțire
```

```
    void operator/=(double d);
    double &operator[](int i); /// operatorul de acces direct la coordonatele Vectorului
    short operator==(Vector &v); /// operatorul de identitate
    short operator==(double d);
    void VECTOR(double x = 0.0, double y = 0.0, double z = 0.0);
    double SCALPRD(Vector &v); /// produsul scalar dintre doi vectori
    double COS(Vector &v); /// cosinusul unghiului dintre doi vectori
    void VECTPRD(Vector &u, Vector &v); /// produsul vectorial dintre doi vectori
    double MIXTPRD(Vector &u, Vector &v); /// produsul mixt dintre trei vectori
    double VECT_LENGTH(void) ; /// lungimea unui vector
    void VERSOR(void); /// generarea  unui vector de lungime 1
    void ROUND(double b); /// rotunjirea coordonatelor unui vector
};
```

Punctul, entitatea de bază pentru corpurile 3D este reprezentat prin clasa CVertex derivată din clasa de bază Vector. Derivarea a fost impusă de faptul că punctul conține suplimentar proprietăți de vizualizare necesare pentru desenarea sa în spațiul de lucru.

Aceste clase şi funcțiile încapsulate în cadrul lor se constituie ca unelte principale în construcția şi proiectarea funcționalității claselor corespunzătoare obiectelor complexe. Proiectarea clasei CMeCurve, modelul obiectual al curbelor neparametrice, a încercat să surprindă totalitatea trăsăturilor caracteristice ale acestor curbe într-un mod cât mai eficient din punct de vedere al programării dar în acelaşi timp şi cât mai natural, prin păstrarea corespondențelor între elementele clasei şi semnificația lor reală.

```
class CMeCurve : public CMeElement
{
   // variabilele clasei
   protected:
    CArray<CVertex,CVertex> curvePoints;    //  poligonul de control al curbei
    CArray<CVertex,CVertex> controlPolygon;  // lista de puncte ale curbei
    CurveType  curveType;        // tipul curbei depinde de metoda de proiectare aleasă
    CurveSettings   curveSettings;    // proprietățile de desenare a curbei
    int   degree;                 // gradul curbei
    struct Spline                 //  structura cuprinde pointerii la funcțiile corespunzătoare
    {                             // fiecărui tip de curbă
      char nm[30];
      CurveFunction execute;  // funcția de construcție a curbei
     CurveProp     prop;  // funcția de setarea a proprietăților curbei
    };
    static Spline        spline[9];
    CArray<CVertex,CVertex> bezierPoints; // punctele poligonului Bézier pe porțiuni

// metodele clasei
};
```

Semnificația variabilelor clasei va fi descrisă în următoarele secțiuni ale capitolului. Fiecărui tip de curbă îi corespunde un set de funcții care conduc la generarea curbei. Accesarea acestor funcții se realizează în mod direct din tabloul static spline prin index care este identificatorul curbei – curveType. Acest mod de proiectare a clasei a permis atât individualizarea simplă a fiecărui tip de curbă, precum şi gruparea în aceeaşi clasă a tuturor metodelor de generare a curbelor. Modalitatea alternativă de proiectare, prin derivarea din clasa CMeCurve pentru fiecare tip de curbă a unor clase proprii, conduce la o creştere inutilă a cantității de cod ce trebuia scrisă şi în acelaşi timp face dificilă operația de modificare a tipului unei curbe.

2.2 Curbe Bézier

Curbele Bézier reprezintă una din cele mai importante clase de curbe polinomiale aproximante folosite curent în domeniul proiectării asistate de calculator. Ele sunt numeric cele mai stabile dintre toate curbele polinomiale şi de asemenea constituie forma geometrică ideală pentru reprezentarea curbelor polinomiale pe bucăţi.

Sunt definite două metode de generare a curbelor Bézier, proprietăţile acestor curbe precum şi principiul înfloririi aplicat asupra curbelor.

2.2.1 Algoritmul lui de Casteljau de generare a curbelor Bézier

Casteljau a dezvoltat un algoritm puternic şi foarte simplu folosit la proiectarea curbelor şi a suprafeţelor [11][15][34][36]. Acesta a fost definit mai întâi pentru parabole, iar generalizarea metodei va conduce la curbe Bézier. Se consideră trei puncte $b_0, b_1, b_2 \in E^3$. Prin interpolări liniare succesive se obţin punctele:

$$\begin{aligned} b_0^1(s) &= (1-s)b_0 + sb_1 \\ b_1^1(s) &= (1-s)b_1 + sb_2 \\ b_0^2(s) &= (1-s)b_0^1(s) + sb_1^1(s) \end{aligned} \qquad (2.2.1)$$

Înlocuind primele două relaţii în ultima rezultă:

$$b_0^2(s) = (1-s)^2 b_0 + 2s(1-s)b_1 + t^2 b_2 \qquad (2.2.2)$$

Expresia pătratică reprezintă formula unei parabole. $b_0^2(s)$ descrie parabola după cum s variază de la $-\infty$ la $+\infty$. Această construcţie constă din repetate interpolări liniare, geometria ei fiind reprezentată în figura 1. Pentru $s \in [0,1]$, $b_0^2(s)$ se află în interiorul triunghiului format de b_0, b_1, b_2; în particular $b_0^2(0) = b_0, b_0^2(1) = b_1$.

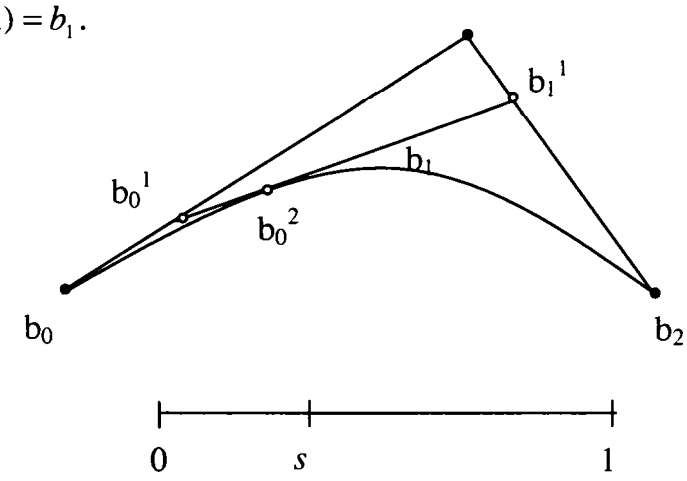

Fig. 2.2.1 Generarea unui punct al parabolei definită de punctele b_0, b_1, b_2

Construcţia parabolei este invariant afină pentru că interpolarea liniară pe

porțiuni este invariant afină. Datorită faptului că $b_0^2(s)$ este întotdeauna o combinație baricentrică a trei puncte, parabola este o curbă plană.

Deoarece aplicațiile de proiectare geometrică a corpurilor necesită curbe în spațiul real, precedenta construcție pentru o parabolă poate fi generalizată, pentru a se genera o curbă polinomială de grad oarecare n:

Algoritmul de Casteljau: Fie $b_0, b_1, b_2 \in E^3$ şi $s \in R$ şi relațiile:

$$b_i^r(s) = (1-s)b_i^{r-1}(s) + sb_{i+1}^{r-1}(s) \quad \begin{cases} r = 1,\dots n \\ i = 0,\dots n-r \end{cases} \qquad (2.2.3)$$

$$b_i^0(s) = b_i$$

Atunci $b_0^n(s)$ este punctul de pe *curba Bézier* b^n corespunzător valorii parametrice s [34][36].

Poligonul P format de $b_0,\dots,b_n$ este numit *poligonul Bézier* sau *poligonul de control* al curbei b^n. Vârfurile poligonului, sunt denumite *puncte de control* sau *puncte Bézier*. Figura 2.1. ilustrează cazul curbelor Bézier cubice. Curba $b_0^n(s)$ este *aproximarea Bernstein-Bézier* pentru poligonul de control, terminologie împrumutată din teoria aproximărilor.

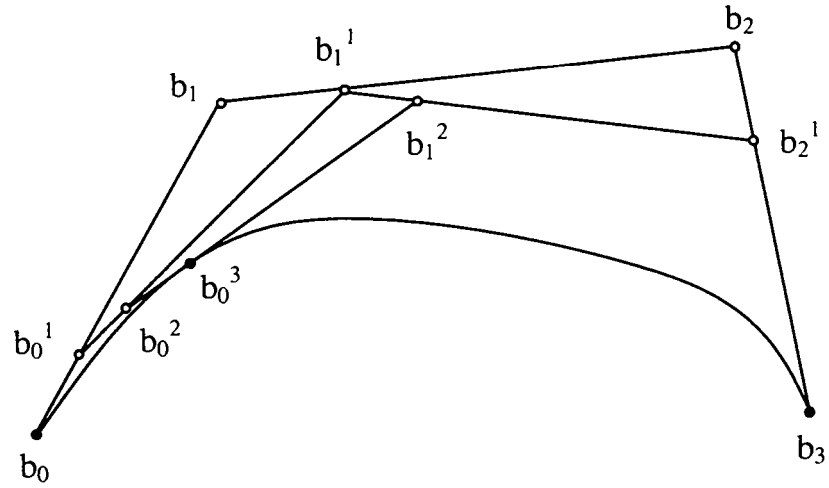

Fig. 2.2.2 Generarea unei curbe Bézier cubice

Convențional, coeficienții intermediari $b_i^r(s)$ sunt scrişi într-un tablou triunghiular de puncte, numit *schema de Casteljau*. Pentru cazul cubic:

$$\begin{array}{cccc} b_0 & & & \\ b_1 & b_0^1 & & \\ b_2 & b_1^1 & b_0^2 & \\ b_3 & b_2^1 & b_1^2 & b_0^3 \end{array}$$

Implementarea curbelor a fost realizată cu ajutorul clasei *CMeCurve* descrisă în secțiunea 2.1.5. Variabila membră *controlPolygon* a clasei conține punctele de control ale curbei iar variabila *curvePoints* punctele curbei. Gradul curbei este

memorat în CMeCurve::degree. Metoda *Bezier()* a clasei cuprinde algoritmul de Casteljau. Deşi schema de Casteljau conduce la ideea folosirii unui tablou bidimensional, acest lucru ar reprezenta consum inutil de memorie fiind suficientă folosirea numai a coloanei din stânga şi suprascrierea ei în mod corespunzător.

```
void CMeCurve::Bezier()
{
    short nr_points, degree, i, r, k;
    double t, t1;
    CArray<CVertex, CVertex> tempTab;

    nr_points =  controlPolygon.GetSize();
    degree = nr_points -1; // gradul curbei Bézier
    // setarea tabloului temporar la dimensiunea necesara
    tempTab.SetSize(nr_points);
    // setarea tabloului de puncte al curbei la dimensiunea necesara
    curvePoints.SetSize(N);
    // calculul punctelor curbei
    for(i = 0; i < N; i++)
    {     // copierea punctelor de control in tabloul temporar
          tempTab.Copy(controlPolygon);
          // calculul fiecărui punct al curbei prin metoda de Casteljau
          t = (double)(i)/(N-1);
          t1 = 1.0 - t;
          for(r = 1; r <= degree; r++)
          {     for(k = 0; k <= degree - r; k++)
                {   // interpolarea liniara realizata pentru fiecare coordonata a punctului
                    tempTab[k]  =  tempTab[k] * t1 + tempTab[k+1] * t;
                }
          }
          curvePoints[i] = tempTab[0];
          myDoc->ComputePersp(this->curvePoints[i]);
    }
}
```

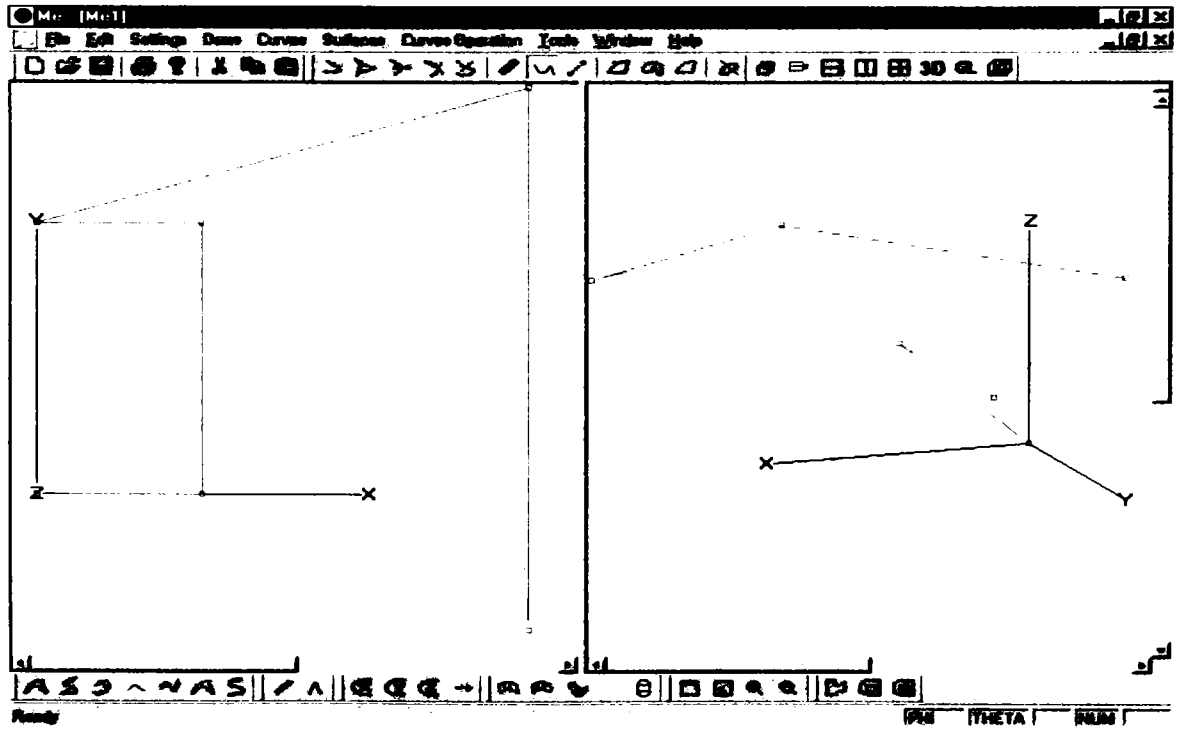

Fig. 2.2.3 Curba Bézier realizata cu algoritmul lui de Casteljau

2.2.2 Forma Bernstein a curbelor Bézier

Curbele Bézier au fost dezvoltate inițial printr-un algoritm recursiv de către de Casteljau. Fiind necesară o reprezentare explicită a acestor curbe, sub forma unei formule nerecursive, Bézier a definit forma lor polinomială prin exprimarea acestora în baza Bernstein [71].

Astfel, *polinoamele Bernstein* sunt definite în mod explicit de relația:

$$B_i^n(\mathrm{s}) = \mathrm{C}_\mathrm{n}^\mathrm{i} s^i (1-\mathrm{s})^{n-i} \tag{2.2.4}$$

coeficienții binomiali fiind:

$$C_\mathrm{n}^\mathrm{i} = \begin{cases} \dfrac{\mathrm{n!}}{\mathrm{i!(n-i)!}} \text{daca } 0 <= \mathrm{i} <= 1 \\ 0 \text{altfel} \end{cases} \tag{2.2.5}$$

Aceste polinoame au ca proprietăți:

1. satisfac relația de recursivitate :

$$\begin{aligned} & B_i^n(s) = (1-\mathrm{s})B_i^{n-1}(s) + sB_{i-1}^{n-1}(\mathrm{s}) && \text{cu} \\ & B_0^0(t) \equiv 1 && \text{și} \\ & B_j^n(s) \equiv 0 \text{ pentru } \; j \notin \{0,\ldots,n\} \end{aligned} \tag{2.2.6}$$

2. formează o *partiție a unității.*

$$\sum_{j=0}^{n} B_j^n(\mathrm{s}) \equiv 1 \tag{2.2.7}$$

3. sunt pozitive pentru s subunitar

$$B_i^n(s) \geq 0 \; \forall s \in [0,1], \forall i \in \overline{0,n} \tag{2.2.8}$$

4. pentru valorile limită ale lui s polinoamele sunt nule mai puțin polinoamele de ordin 0 respectiv n care sunt egale cu 1:

$$B_i^n(0) = \begin{cases} 0 & pt.\, i = 1,2,\ldots,n \\ 1 & pentru\ i = 0 \end{cases}, \qquad B_i^n(1) = \begin{cases} 0 & pt.\, i = 0,1,\ldots,n-1 \\ 1 & pentru\ i = n \end{cases} \tag{2.2.9}$$

5. în domeniul [0,1] au un maxim pentru $s = \dfrac{i}{n}$

Punctele de Casteljau intermediare b_i^r pot fi exprimate cu ajutorul polinoamelor Bernstein de grad r :

$$b_i^r(s) = \sum_{j=0}^{r} b_{i+j} B_j^r(s) \quad r \in \{0,...,n\}, \quad i \in \{0,...,n-r\} \tag{2.2.10}$$

Relația (2.2.10) arată exact modul în care punctele intermediare b_i^r depind de punctele Bézier date b_i. Pentru cazul $r = n$, punctul de Castelja0u corespunzător este punctul de pe curbă şi este dat de relația:

$$b^n(s) = b_0^n(s) = \sum_{j=0}^{n} b_j B_j^n(s) \tag{2.2.11}$$

care reprezintă *forma Bernstein a curbelor Bézier* [11].

Pentru implementarea formei Bernstein a curbelor Bézier se poate utiliza algoritmul Horner-Bézier. Acest algoritm realizează o regrupare a termenilor din membrul drept al relației (2.2.11):

$$\begin{aligned} b(s) &= C_n^0 q^n b_0 + C_n^1 s q^{n-1} b_1 + C_n^2 s^2 q^{n-2} b_2 + ... + C_n^n s^n b_n = \\ &= [\cdots[(C_n^0 q b_0 + C_n^1 s b_1) q + C_n^2 s^2]s + \cdots + C_n^n s^n b_n \qquad q = 1 - s \end{aligned} \tag{2.2.12}$$

Funcția BezierHorner() determină punctele curbei Bézier folosind forma polinomială a acestora.

```
void CMeCurve::BezierHorner(void)
{
    short nr_points, degree, i, j;
    double t, s, combinari, putere;

    nr_points = controlPolygon.GetSize();
    degree = nr_points -1;  // gradul curbei Bézier
    // setarea tabloului de puncte al curbei la dimensiunea necesara
    curvePoints.SetSize(N);

    // calculul punctelor curbei folosind algoritmul Horner - Bézier
    for(i = 0; i < N; i++)
    {
        t = (double)(i)/(N-1);
        s = 1.0 - t;
        combinari = 1;
        putere = 1;
        curvePoints[i]  =  controlPolygon[0] * s;
        for(j = 1; j < degree; j ++)
        {
            combinari = combinari*(degree - j + 1)/j;
            putere = t*putere;
            curvePoints[i] = (curvePoints[i]  +  controlPolygon[j] * combinari * putere) * s;
        }
        putere = t*putere;
        curvePoints[i] = curvePoints[i] +  controlPolygon[degree] * putere;
```

```
            myDoc->ComputePersp(curvePoints[i]);
        }
}
```

2.2.3 Proprietăți ale curbelor Bézier

Algoritmul de Casteljau, cu argumentele sale geometrice, şi forma polinomială Bernstein, cu argumentele sale algebrice, permit definirea proprietăților curbelor Bézier:

- *Invarianța afină.* Curbele Bézier sunt invariante în raport cu relațiile afine, ceea ce înseamnă că următoarele două proceduri dau acelaşi rezultat: (1) se calculează punctul $b^n(s)$ şi apoi acestuia i se asociază o relație afină; (2) se aplică o relație afină poligonului de control şi apoi se evaluează poligonul la valoarea parametrului s. Invarianța afină este o consecință directă a algoritmului de Casteljau: algoritmul este format dintr-o secvența de interpolări liniare. Ele sunt invariant afine şi astfel este şi o secvență finită a lor [13][14][16].

Ca aspect practic al acestei proprietăți este rotația unei curbe în jurul unei axe de rotație. Operația se realizează prin aplicarea rotației punctelor de control şi apoi, pornind de la poligonul de control modificat, se recalculează curba. Rezultatul obținut este identic cu cel care s-ar realiza prin aplicarea rotației pe rând tuturor punctelor curbei, proces care însă este mult mai costisitor din punct de vedere al timpului. Factorul timp este esențial în modelarea cât mai realistă a mişcării roboților. Funcția virtuală RotateElement() realizează operația de rotație a unei curbei folosind proprietatea de invarianță afină.

```
void CMeCurve::RotateElement(Vector &rotation_axis, double angle)
{
    double l;
    Vector origin;
   origin = 0.0;
   InitPerspBBox();
    short nr_points = controlPolygon.GetSize();
    // rotirea punctelor poligonului de control in jurul originii
    for(int i = 0; i < nr_points;  i++)
    {   l = controlPolygon[i].VECT_LENGTH();
        if(l > V_PPREC)
        {   PointRotation(controlPolygon[i], origin, rotation_axis, angle);
            controlPolygon[i].CalcPersp();  // calculul perspectivelor punctului
            SetPerspBBox(controlPolygon[i]);
        }
    }
    // setarea boundingbox-ului curbei; acesta este identic cu al poligonului de control
    // (datorita proprietății de acoperire convexa)
    SetElementBBox();
    // reconstruirea curbei in funcție de tipul său
    (this->*this->spline[this->curveType].execute)();
}
```

Curbele Bézier nu sunt *invariant proiective.* Relațiile proiective sunt utilizate în grafica pe calculator când un obiect este reprezentat realistic (reprezentarea în perspectivă). Astfel, dacă se doreşte simplificarea unei imagini de perspectivă a unei curbe Bézier prin reprezentarea mai întâi a imaginii poligonului de control, după care

se calculează curba, curba rezultată nu este imaginea de perspectivă a curbei originale.

- *Invarianța în cazul transformărilor afine de parametri.* În cadrul algoritmului de generare curbele Bézier au fost definite pe intervalul [0,1] cu toate că algoritmul de Casteljau este "insensibil" față de intervalul pe care curba este definită. Astfel, pentru o curbă definită pe un interval arbitrar ales $a \le q \le b$ pe axa reală, prin introducerea parametrului local $s = (q-a)/(b-a)$, algoritmul decurge normal. Această proprietate provine de la interpolarea liniară. Forma generalizată a algoritmului de Casteljau este:

$$b_i^r(q) = \frac{b-q}{b-a} b^{r-1}(q) + \frac{q-a}{b-a} b_{i+1}{}^{r-1} \qquad (2.2.13)$$

Tranziția de la intervalul $[0,1]$ la intervalul $[a,b]$ reprezintă o transformare afină. Deci, curbele Bézier sunt invariante în cazul transformărilor afine de parametri.

Algebric, această proprietate se exprimă:

$$\sum_{i=0}^{n} b_i B_i^n(s) = \sum_{i=0}^{n} b_i B_i^n\left(\frac{q-a}{b-a}\right) \qquad (2.2.14)$$

- *Proprietatea de acoperire convexă.* Pentru $s \in [0,1]$, $b^n(s)$ se află în acoperirea convexă a poligonului de control. Geometric, acest lucru este demonstrat, deoarece orice valoare intermediară $b_i^r(s)$ se obține ca o combinație baricentrică convexă a punctelor b_i^{r-1}, b_{i+1}^{r-1} de la pasul anterior. În nici un pas al algoritmului de Casteljau nu se creează puncte în afara acoperirii convexe a lui b_i. Pornind de la forma Bernstein, datorită faptului că polinoamele Bernstein sunt pozitive pe domeniul [0,1], curba Bézier este o combinație convexă a punctelor de control şi deci toate punctele sale se află în acoperirea convexă a acestora. O consecință imediată a proprietății de acoperire convexă este aceea că un poligon plan de control generează întotdeauna o curbă plană [11][13].

Proprietatea este folosită la operația de verificare a interferenței a două curbe. Dacă de exemplu, fiecare curbă reprezintă traiectoria unui braț de robot, pentru evitarea coliziunii efectoarelor, este necesar ca cele două căi să nu se intersecteze. În loc de a se calcula o posibilă intersecție, se circumscrie cea mai mică cutie posibilă în jurul poligonului de control al fiecărei curbe astfel încât laturile acesteia să fie paralele cu sistemul de coordonate. Aceste cutii sunt numite *cutii minmax* sau *bounding box*, deoarece fețele lor sunt generate de coordonatele minime şi maxime ale poligonului de control. Fiecare cutie minimax conține poligonul său de control şi curba Bézier corespunzătoare datorită proprietății de acoperire convexă. Dacă cele două cutii minimax nu se suprapun, atunci în mod cert cele două curbe nu se intersectează. Dacă cutiile se suprapun, este necesară aplicarea unor verificări suplimentare. Posibilitatea luării unei decizii rapide din punct de vedere al interferenței este foarte importantă, din moment ce, în practică, adesea se verifică un obiect împotriva altora, multe dintre acestea pot fi etichetate ca "neinterferate" de testul cutiei minimax.

- *Interpolarea punctului de sfârşit.* Curbele Bézier trec prin punctul de început b_0 şi de sfârşit b_n al poligonului de control, proprietate verificată de relația de Casteljau scrisă pentru cazurile $s = 0$ şi $s = 1$. Dacă $b_0 = b_n$ curba Bézier este închisă.

- *Simetrie.* Considerând exemplul din figura 2.4, se poate uşor observa că nu contează dacă punctele Bézier sunt notate $b_0, b_1, \ldots, b_n$ sau $b_n, b_{n-1}, \ldots, b_0$. Curbele care corespund celor două ordonări au acelaşi aspect, fiind diferite doar prin direcţia în care sunt parcurse. Formal se scrie identitatea:

$$\sum_{j=0}^{n} b_j B_j^n(s) = \sum_{j=0}^{n} b_{n-j} B_j^n(1-s) \qquad (2.2.15)$$

care rezultă ca o consecinţă a relaţiei $B_j^n(s) = B_{n-j}^n(1-s)$.

- *Invarianţa în cazul combinaţiilor baricentrice.* Procesul formării curbei Bézier din poligonul Bézier lasă combinaţiile baricentrice invariante. Pentru $a+b=1$, rezultă:

$$\sum_{j=0}^{n} (\alpha b_j + \beta c_j) B_j^n(s) = \alpha \sum_{j=0}^{n} b_j B_j^n(s) + \beta \sum_{j=0}^{n} c_j B_j^n(s) \qquad (2.2.16)$$

Astfel se poate construi media ponderată a două curbe Bézier fie pe baza mediei ponderate a punctelor corespunzătoare de pe curbe, fie pe baza mediei ponderate a vârfurilor de control corespunzătoare.

Proprietăţile acestor curbe constituie motivul pentru care ele reprezintă instrumente utile în desenarea curbelor oarecare. Pentru a reproduce forma unei curbe desenate cu mâna, este suficientă specificarea poligonului de control care oarecum "subliniază" forma curbei. Calculatorul va desena curba Bézier definită de poligon şi dacă este necesar se vor ajusta poziţiile vârfurilor poligonului. De obicei, o persoană experimentată va reproduce o curbă dată după doua sau trei iteraţii ale acestei proceduri *interactive*.

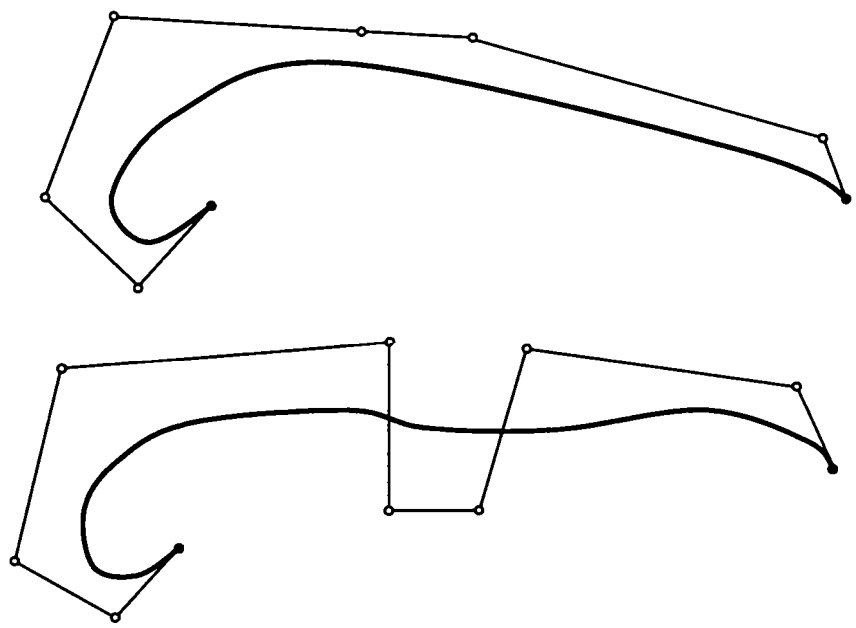

Fig. 2.2.4 Modelarea formei curbei prin modificarea poligonului său de control

- *Control pseudo-local.* Deoarece polinoamele Bernstein B_i^n au un maxim pe care îl ating la $t = i/n$ rezultă că prin modificarea doar a unuia dintre vârfurile

poligonului de control b_i, curba este afectată de schimbare în regiunea curbei din jurul valorii i/n a parametrului . Aceasta face ca efectul schimbării să fie uşor de prevăzut, cu toate că modificarea afectează întreaga curbă.

2.2.4 Derivatele unei curbe Bézier

Un rol important în studiul curbelor Bézier şi, în continuare, a curbelor B-spline cubice îl ocupă derivatele unei curbe Bézier. Influenţa formei poligonului de control asupra formei curbei se exprimă cu ajutorul derivatelor, iar generarea curbelor B-spline cubice se bazează pe condiţiile de continuitate date tot de derivatele curbei Bézier.

Derivata de ordinul 1 a unei curbe Bézier

Pentru analiza şi calculul acestora se porneşte de la determinarea derivatei unui polinom Bernstein. Derivarea formulei polinomului Bernstein conduce la relaţia:

$$\frac{d}{dt}B_i^n(\mathrm{s}) = n[B_{i-1}^{n-1}(\mathrm{s}) - B_i^{n-1}(\mathrm{s})] \qquad (2.2.17)$$

Derivata unei curbe Bézier b^n va fi:

$$\frac{d}{dt}b^n(\mathrm{s}) = n\sum_{j=0}^{n}[B_{j-1}^{n-1}(s) - B_j^{n-1}(s)]b_j \qquad (2.2.18)$$

Relaţia poate fi modificată luând în considerare proprietatea 1 a polinoamelor Bernstein şi prin introducerea *operatorului diferenţă*: $\Delta b_j = b_{j+1} - b_j$ [17][32]. Forma derivatei unei curbe Bézier va fi:

$$\frac{d}{dt}b^n(s) = n\sum_{j=0}^{n-1}\Delta b_j B_j^{n-1}(s); \qquad \Delta b_j \in R^3 \qquad (2.2.19)$$

Se observă că, derivata unei curbe Bézier este o altă curbă Bézier obţinută prin diferenţierea poligonului de control original. Această curbă Bézier derivată nu mai este definită în E^3, coeficienţii săi fiind diferenţe de puncte, *vectori*, care sunt elemente din R^3. Pentru a vizualiza curba şi curba derivatei în E^3, se poate construi un poligon ce constă din punctele $a + \Delta b_0, \ldots, a + \Delta b_{n-1}$. a este un punct arbitrar, o alegere potrivită este a = 0.

Derivate de ordin superior

Pentru calculul derivatei de ordin superior se generalizează mai întâi operatorul diferenţă:

$$\Delta^r b_j = \Delta^{r-1} b_{j+1} - \Delta^{r-1} b_j \qquad (2.2.20)$$

Formula pentru derivata de ordinul r a unei curbe Bézier este:

$$\frac{d^r}{dt^r}b^n(t) = \frac{n!}{(n-r)!}\sum_{j=0}^{n-r}\Delta^r b_j B_j^{n-r}(t) \qquad (2.2.21)$$

Relația se demonstrează prin inducție [88]. Pornind de la această formulă, se observă că derivata de ordinul r a unei curbe Bézier într-un punct de capăt ($t=0, t=1$) depinde doar de $r+1$ puncte Bézier din apropierea acelui punct de capăt (inclusiv). Pentru $r=0$ se obține proprietatea inserării punctului de sfârşit determinată anterior. Cazul $r=1$ stabileşte că b_0 şi b_1 definesc tangenta la $t=0$. În general, tangenta la b_0 este dată de b_0 şi de primul punct b_i diferit de b_0, astfel tangenta poate fi definită chiar dacă vectorul tangent este vectorul nul. În mod similar b_{n-1} şi b_n determină tangenta la $t=1$.

2.2.5 Forma polară a curbelor Bézier

Principiul înfloririi sau al polarizări a fost introdusă ca o generalizare a metodei lui de Casteljau [35][54][92][93]. Se utilizează schema lui de Casteljau, în cadrul căreia se reprezintă cei n paşi ai metodei sub formă de coloane. Înflorirea impune ca în coloana r să nu se facă un pas de Casteljau pentru valoarea parametrului s, ci să se folosească o nouă valoare s_r. Astfel pentru cazul cubic, se obține:

b_0
$b_1 \quad b_0^1[s_1]$
$b_2 \quad b_1^1[s_1] \quad b_0^2[s_1,s_2]$
$b_3 \quad b_2^1[s_1] \quad b_1^2[s_1,s_2] \quad b_0^{23}[s_1,s_2,s_3]$

Punctul rezultat $b_0^2[s_1,s_2,s_3]$ este o funcție de trei variabile independente, astfel încât acesta nu descrie o curbă, ci o regiune din E^3. Această funcție trivariabilă $b[.,.,.]$ este numită *înflorirea curbei* $b^3(s)$ [89][92][93]. Curba originală este regăsită pentru toate cele trei argumente egale: $s = s_1 = s_2 = s_3$.

Pentru exemplificare, punctele de început şi de sfârşit ale curbei sunt definite prin înflorire în forma: $b[0,0,0] = b_0$ şi $b[1,1,1] = b_3$. Pentru argumentele $[t_1,t_2,t_3] = [0,0,1]$, schema înfloririi se reduce la :

b_0
$b_1 \quad b_0$
$b_2 \quad b_1 \quad b_0$
$b_3 \quad b_2 \quad b_1 \quad b_1 = b[0,0,1]$

În mod similar se obține $b_2 = b[0,1,1]$. Se observă că punctele Bézier originale pot fi găsite prin evaluarea înfloririi curbei cu argumente constând din valori de 0 şi 1. Si celelalte puncte ale schemei de Casteljau pot fi scrise ca valori ale înfloririi pentru

argumente speciale. Astfel pentru $[s_1, s_2, s_3] = [0,0,s]$ se obține $b_0^1 = b[0,0,s]$. Schema de Casteljau completă este de forma:

$$
\begin{array}{llll}
b_0 = b[0,0,0] & & & \\
b_1 = b[0,0,1] & b_0^1 = b[0,0,s] & & \\
b_2 = b[0,1,1] & b_1^1 = b[0,1,s] & b_0^2 = b[0,s,s] & \\
b_3 = b[1,1,1] & b_2^1 = b[1,1,s] & b_1^2 = b[1,s,s] & b_0^3 = b[s,s,s]
\end{array}
$$

Pentru orice curbă de grad n oarecare, punctele Bézier pot fi exprimate ca valori ale înfloririi prin relația:

$$b_i = b[0^{<n-r-i>}, s^{<r>}, 1^{<i>}] \qquad (2.2.22)$$

$s^{<i>}$ semnifică faptul că s apare ca argument de r.

Formula recursivă de Casteljau pentru curbele Bézier poate fi acum exprimată în termenii înfloririi:

$$b[0^{<n-r-i>}, s^{<r>}, 1^{<i>}] = (1-s)b[0^{<n-r-i+1>}, s^{<r-1>}, 1^{<i>}] + sb[0^{<n-r-i>}, s^{<r-1>}, 1^{<i+1>}] \quad (2.2.23)$$

Punctul de pe curbă este $b[s^{<n>}]$.

Se poate observa că nu contează în ce ordine se utilizează argumentele t_i pentru evaluarea înfloririi. Astfel, pentru cazul cubic $b[t_1, t_2, t_3] = b[t_2, t_3, t_1]$. Datorită acestui fapt înflorirea are proprietatea de *simetrie*. Fiecare curbă polinomială are o unică înflorire asociată ei: o funcție polinomială simetrică de n variabile, ce relaționează R^n în E^3.

O altă proprietate importantă a înfloririi este *multiafinitatea*. Dacă primul argument al înfloririi este definit ca o combinație baricentrică de două (sau mai multe) numere, se pot calcula valorile înfloririi pentru fiecare număr după care se determină combinația baricentrică a lor:

$$b[\alpha r + \beta s, s_2, \ldots, s_n] = \alpha b[r, s_2, \ldots, s_n] + \beta b[s, s_2, \ldots, s_n] \quad \alpha + \beta = 1 \qquad (2.2.24)$$

Formula exprimă faptul că înflorirea este afină în raport cu primul său argument, fapt adevărat pentru oricare din argumentele rămase. Acesta este motivul pentru care înflorirea este numită multiafină. Înfloririle sunt multiafine deoarece sunt obținute prin repetarea paşilor algoritmului de Casteljau. Fiecare din aceşti paşi constă din interpolări liniare, o relație afină ea însăşi.

De asemenea, se poate determina înflorirea unei curbe Bézier definită pe un interval [a,b] din domeniul real. Procedând în mod similar şi ținând seama de forma generalizată a algoritmului de Casteljau se determină punctele Bézier b_i ca valori ale înfloririi prin relația:

$$b_i = b[a^{<n-i>}, b^{<i>}] \qquad (2.2.25)$$

După primul pas al algoritmului de Casteljau cu referire la o valoare s_1 a parametrului, punctele rezultate $b_0^1(s_1),\ldots,b_{n-1}^1(s_1)$ pot fi interpretate ca un poligon de control al unei curbe $p_1(s)$ de grad $n-1$. În terminologia înfloririi curba este definită ca:

$$p_1(s) = b[s_1, s^{<n-1>}]$$

Polinomul p_1 este numit *primul polar* al lui $b(s)$. Semnificaţia geometrică a ultimei relaţii este următoarea: tangenta în orice punct $b(t)$ intersectează polarul $p_1(s)$ în $p_1(s)$. Acest fapt nu este limitat doar la curbele din plan, ci este la fel de adevărat şi pentru curbele din spaţiu.

Un caz special este dat de $b[0, s^{<n-1>}]$: acesta este polinomul definit de $b_0,\ldots,b_{n-1}$. În mod similar, $b[1, s^{<n-1>}]$ este definit de $b_1,\ldots,b_n$.

Procesul formării polarilor se poate repeta, obţinându-se astfel un al doilea polar $p_{1,2}(t) = b[s_1, s_2, s^{<n-2>}]$, etc. Se ajunge la cel de-al *n* polar, care este şi înflorirea lui b(*s*) [11][89][92][93].

2.3 Curbe spline în forma Bézier

Curbele Bézier deşi constituie metode puternice de proiectare în CAGD prezintă anumite dezavantaje: dacă curba ce urmează să fie modelată are o formă complexă, atunci reprezentarea ei Bézier necesită folosirea unui număr mare de puncte de control şi deci curba va avea un grad mare. Pentru parametrizări de grad mare, ce depăşesc valoarea 10, intervin erori de calcul datorită numărului mare de operaţii şi deci metodele Bézier nu mai conduc la rezultate corecte. Curbele de formă complexă sunt modelate folosind curbe Bézier compuse. Astfel de curbe polinomiale pe porţiuni sunt denumite *curbe B-spline* [17][19][30][32].

O curbă Bézier compusă este obţinută prin racordarea mai multor arce Bézier de acelaşi grad. Fiecare segment de curbă este definit de propriul său poligon de control $b_i, b_{i+1}, \ldots, b_{i+n}$. Ultimul punct al fiecărui poligon coincide cu primul punct de control al poligonului următor. Datorită proprietăţii de interpolare a punctului de sfârşit, reuniunea arcelor Bézier defineşte o curbă continuă.

2.3.1 Parametrizarea globală şi locală a unei curbe

Curbele Bézier definite pe intervalul [0,1], au putut fi uşor dezvoltate teoretic. Ele pot fi însă definite pe orice domeniu real datorită proprietăţii lor de invarianţă la transformări afine de parametru. Pentru cazul curbelor polinomiale pe porţiuni, dacă fiecare segment individual al curbei poate fi definit pe intervalul [0,1], curba spline, ca întreg, este definită pe o colecţie de intervale ale căror lungimi relative joacă un rol important.

Pentru stabilirea unei parametrizări globale a unei curbe spline, se alege un domeniu arbitrar $[u_0, u_L]$ divizat de valorile $u_i, i = 1,2,\ldots L-1$ în L intervale $u_0 < u_1 < \ldots u_{L-1} < u_L$. O curbă spline c reprezintă o relaţionare a domeniului $[u_0, u_L]$ în E^3. Fiecare interval $[u_i, u_{i+1}]$ constituie spaţiul de parametrizare al unui segment Bézier al curbei spline. Numerele reale u_i sunt denumite *noduri*. Colecţia tuturor nodurilor este numită *secvenţa de noduri* a curbei spline. Oricărei valori u din domeniul $[u_0, u_L]$ îi corespunde un punct $c(u)$ pe curba c. Parametrul u se numeşte *parametrul global* al curbei. Pentru fiecare interval $[u_i, u_{i+1}]$ se defineşte *parametrul local* s prin relaţia:

$$s = \frac{u - u_i}{u_{i+1} - u_i} = \frac{u - u_i}{\Delta_i} \tag{2.3.1}$$

Δ_i este operatorul diferenţă definit în secţiunea 2.2.4. s parcurge domeniul [0,1] la o variaţie a lui u de la u_i la u_{i+1} [14][16].

Definirea întregii curbe spline se face în termenii parametrizării globale. Segmentele individuale c_i ale lui c sunt curbe Bézier care pot fi mai uşor descrise în termenii parametrizării locale. Astfel un punct al curbei spline este $c(u) = c_i(s)$.

Legătura între parametrizarea locală şi cea globală poate fi observată şi în cazul formulei derivatei curbei într-un punct $u \in [u_i, u_{i+1}]$:

$$\frac{dc(u)}{du} = \frac{dc_i(s)}{ds}\frac{ds}{du} = \frac{1}{\Delta_i}\frac{dc_i(s)}{ds} \tag{2.3.2}$$

Punctele $c(u_i) = c_i(0) = c_{i-1}(1)$ sunt denumite *puncte de joncţiune*. Colecţia tuturor poligoanelor Bézier ale segmentelor de curbe Bézier formează un poligon numit *poligonul Bézier pe porţiuni* al curbei spline c [11][30] [91].

2.3.2 Condiţii de continuitate

Două curbe Bézier s_o şi s_1 având poligoanele de control $b_0,\ldots,b_n$, respectiv $b_n,\ldots,b_{2n}$, pot fi considerate ca existând individual independente una de alta, sau ca două segmente ale unei curbe compuse s pe domeniul $[u_0, u_2]$. În primul caz nu există nici un fel de restricţie asupra punctelor de control ale celor două curbe. Pentru cea de a doua situaţie, între punctele poligoanelor de control ale celor două curbe se stabilesc anumite relaţii de legătură pentru a se menţine proprietăţile de continuitate şi derivabilitate ale curbei compuse de-a lungul întregului domeniu de definiţie.

Curba c_0, definită pe intervalul $[u_0, u_1]$, şi c_1, definită pe intervalul $[u_1, u_2]$, sunt de r ori continuu derivabile în $u = u_1$ dacă:

$$\frac{d^r}{du^r}c_0(u_1) = \frac{d^r}{du^r}c_1(u_1) \tag{2.3.3}$$

Folosind formula derivatei de ordin superior şi transformarea de parametrii global-local, relaţia devine:

$$\left(\frac{1}{\Delta_0}\right)^i \Delta^i b_{n-i} = \left(\frac{1}{\Delta_1}\right)^i \Delta^i b_n \qquad i = 0,\ldots,r$$
$$\Delta_0 = u_1 - u_0 \tag{2.3.4}$$
$$\Delta_1 = u_2 - u_1$$

Relaţia defineşte *condiţia de continuitate* C^r pentru o curbă polinomială pe porţiuni. Continuitatea de grad 0 (C^0), indică faptul că punctul de sfârşit al poligonului de control al primei curbe Bézier este identic cu primul punct de control al celei de-a doua curbe. Cazurile de continuitate C^1 şi C^2 sunt cele mai relevante din punct de vedere practic, ele definind panta şi curbura unei spline [17][20].

Racordul de clasă C^1 a două curbe Bézier

Pornind de la relația 2.3.4 se obține condiția de racord ținând seama de faptul că derivata de ordinul întâi a parametrizării Bernstein a unei curbe Bézier pentru valorile 0 şi 1 ale parametrului depinde de cele mai apropiate două puncte de control [21].

Curba c este de clasă C^1 în u_1 dacă şi numai dacă punctele b_{n-1}, b_n, b_{n+1} sunt coliniare şi punctul b_n împarte segmentul de dreaptă $b_{n-1}b_{n+1}$ în raportul $\Delta_0 : \Delta_1$

$$b_n = \frac{\Delta_1}{\Delta_0 + \Delta_1} b_{n-1} + \frac{\Delta_0}{\Delta_0 + \Delta_1} b_{n+1} \tag{2.3.5}$$

Racordul de clasă C^2 a două curbe Bézier

Racordul de clasă C^2 în u_1 a două curbe cubice implică ca derivatele de ordinul 2 ale celor două curbe să coincidă:

$$\frac{1}{\Delta_0^2}(b_{n-2} - 2b_{n-1} + b_n) = \frac{1}{\Delta_0^2}(b_n - 2b_{n+1} + b_{n+2}) \tag{2.3.6}$$

Deoarece s este de clasă C^1, punctul b_n poate fi exprimat în raport cu b_{n-1} şi b_{n+1}. Înlocuind b_n în formula 3.6 rezultă:

$$\left(1 - \frac{1}{s_1}\right) b_{n-2} + \frac{1}{s_1} b_{n-1} = \frac{1}{1 - s_1} b_{n+1} - \frac{s_1}{1 - s_1} b_{n+2} \qquad s_1 = \frac{\Delta_0}{\Delta_0 + \Delta_1} \tag{2.3.7}$$

Ambii termeni ai relației 2.3.7, conform interpolării liniare, conduc la ideea existenței unui punct d aflat în relație de coliniariate cu punctele de control implicate. Luând în considerație acest lucru se poate defini condiția de continuitate de grad 2 [17][21].

Curba s este de clasă C^2 în punctul de joncțiune u_1, dacă şi numai dacă există un punct d astfel încât punctele b_{n-2}, b_{n-1}, d şi d, b_{n+1}, b_{n+2} să fie coliniare şi în plus punctul b_{n-1}, respectiv b_{n+1} să împartă segmentul $[b_{n-2}, d]$, respectiv $[d, b_{n+2}]$ în raportul $\frac{\Delta_0}{\Delta_1}$:

$$\begin{aligned} b_{n-1} &= (1 - s_1) b_{n-2} + s_1 d \\ b_{n+1} &= (1 - s_1) d + s_1 b_{n+2} \qquad s_1 = \frac{\Delta_0}{\Delta_0 + \Delta_1} \end{aligned} \tag{2.3.8}$$

Punctul d asociat racordului de clasă C^2 a două curbe Bézier se numeşte *punct de Boor* [17][33].

2.3.3 Cubice B-spline

Curba polinomială pe bucăți $s:[u_0,u_L]\to E^3$, pentru care este definită o secvență de noduri $u_0<u_1<\cdots<u_L$ care împart domeniul în L subintervale, este o cubică B-spline generată prin racordul de clasă C^2 în $u_i, i=\overline{1,L-1}$ a unui număr de L curbe Bézier cubice $c_i:[u_i,u_{i+1}]\to E^3$ de poligon de control $b_{3i},b_{3i+1},b_{3i+2},b_{3i+3}$.

În acest context, o cubică B-spline este determinată de secvența sa de noduri şi de poligonul Bézier pe bucăți format din punctele de control ale segmentelor de curbă Bézier, poligon ce conține $3L+1$ puncte. Luându-se în considerare condițiile de racord de clasă C^2 care introduc punctele de Boor, se poate genera o cubică B-spline pornind de la setul de puncte de Boor ce formează *poligonul de Boor* al cubicei.

Astfel, o cubică B-spline este complet determinată de poligonul de Boor alcătuit din $L+3$ puncte $d_{-1},d_0,d_1,\ldots,d_{L+1}$ şi de secvența de noduri $u_0<u_1<\cdots<u_L$ care divizează intervalul $[u_0,u_L]$. Această modalitate de definire este mai simplă şi mai intuitivă deoarece implică folosirea unui număr mai mic de puncte de control față de definirea completă a cubicei spline prin poligonul său pe porțiuni [88][94].

Punctele poligonului Bézier pe porțiuni a cubicei B-spline sunt obținute din punctele poligonului de Boor. Din definiția punctelor de Boor d_i rezultă că punctele b_{3i-2},b_{3i-1},d_i sunt coliniare şi b_{3i-1} împarte segmentul $b_{3i-2}d_i$ în raportul $\frac{\Delta_{i-1}}{\Delta_i}$. La fel, punctele $d_{i-1},b_{3i-2},b_{3i-1}$ sunt coliniare şi b_{3i-2} divide segmentul $d_{i-1}b_{3i-1}$ în raportul $\frac{\Delta_{i-2}}{\Delta_{i-1}}$. Din aceste afirmații rezultă faptul că punctele $d_{i-1},b_{3i-2},b_{3i-1},d_i$ sunt coliniare, iar punctul b_{3i-2} împarte segmentul $d_{i-1}d_i$ în raportul $\frac{\Delta_{i-2}}{\Delta_{i-1}+\Delta_i}$ (Figura 2.3.1.). Ținând seama de considerațiile făcute la interpolarea liniară şi la definirea raporturilor, punctul b_{3i-2} rezultă ca interpolant al punctelor d_{i-1},d_i:

$$b_{3i-2}=\frac{\Delta_{i-1}+\Delta_i}{\Delta}d_{i-1}+\frac{\Delta_{i-2}}{\Delta}d_i \qquad i=\overline{2,L-1} \tag{2.3.9}$$

În mod similar, punctul b_{3i-1} împarte segmentul $d_{i-1}d_i$ în raportul $\frac{\Delta_{i-2}+\Delta_{i-1}}{\Delta_i}$ şi se obține cu formula:

$$b_{3i-1}=\frac{\Delta_i}{\Delta}d_{i-1}+\frac{\Delta_{i-2}+\Delta_{i-1}}{\Delta}d_i \qquad i=\overline{2,L-1} \tag{2.3.10}$$

În cele două relații, $\Delta=\Delta_{i-2}+\Delta_{i-1}+\Delta_i$ [18][32][88].

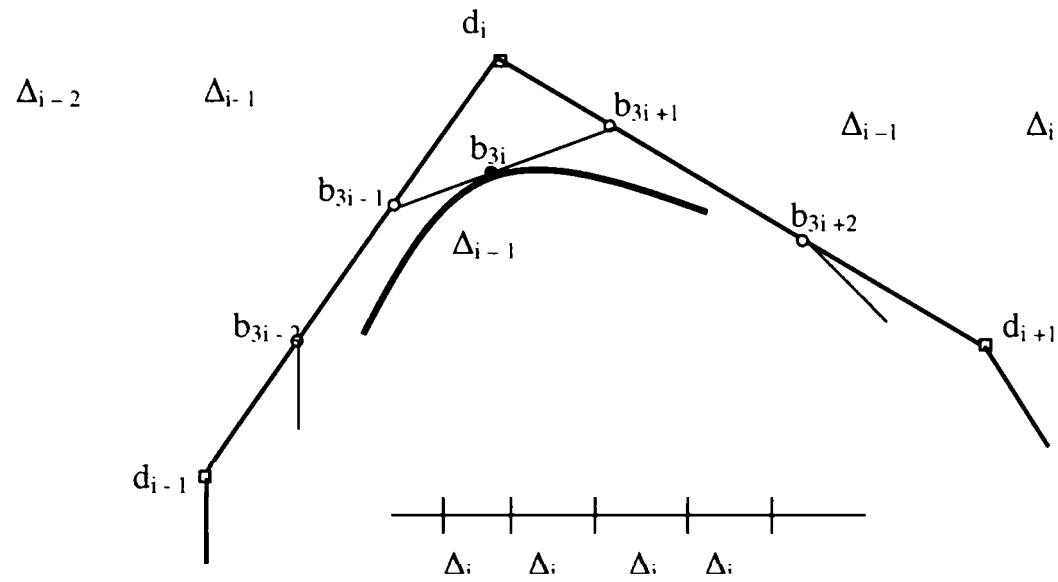

Fig. 2.3.1 Poligonul de Boor şi poligonul Bézier pe porţiuni al unei cubice spline

Cu ajutorul acestor relaţii, din punctele de Boor $d_1, d_2, \ldots, d_{L-1}$ se generează punctele $b_4, b_5, \ldots, b_{3i-2}, b_{3i-1}, \ldots, b_{3L-5}, b_{3L-4}$ ale poligonului Bézier pe bucăţi, numite şi *puncte interioare* ale acestui poligon pe porţiuni.

Punctele Bézier de joncţiune b_{3i} se determină din condiţia de racord de clasă C^1 a curbelor Bézier. Punctele $b_{3i-1}, b_{3i}, b_{3i+1}$ sunt coliniare şi b_{3i} împarte segmentul $b_{3i-1}b_{3i+1}$ în raportul $\frac{\Delta_{i-1}}{\Delta_i}$. Rezultă:

$$b_{3i} = \frac{\Delta_i}{\Delta_{i-1} + \Delta_i} b_{3i-1} + \frac{\Delta_{i-1}}{\Delta_{i-1} + \Delta_i} b_{3i+1} \qquad i = \overline{3, L-2} \tag{2.3.11}$$

Punctele de sfârşit ale poligonul Bézier pe porţiuni sunt calculate cu relaţiile:

$$\begin{aligned} b_0 &= d_{-1} \\ b_1 &= d_0 \\ b_2 &= \frac{\Delta_1}{\Delta_0 + \Delta_1} d_0 + \frac{\Delta_0}{\Delta_0 + \Delta_1} d_1 \end{aligned} \tag{2.3.12}$$

$$\begin{aligned} b_{3L-2} &= \frac{\Delta_{L-1} + \Delta_L}{\Delta} d_{L-1} + \frac{\Delta_{L-2}}{\Delta} d_L, \qquad \Delta = \Delta_{L-2} + \Delta_{L-1} + \Delta_L \\ b_{3L-1} &= d_L \\ b_{3L} &= d_{L+1} \end{aligned}$$

Punctele b_3 şi b_{3L-3} rezultă aplicând formula de calcul a punctelor de joncţiune pentru $i = 1$, respectiv $i = L-1$.

Această construcţie a punctelor poligonului Bézier pe bucăţi a unei cubice B-spline, pornind de la poligonul de Boor şi secvenţa de noduri a fost realizată de către Boehm şi este cunoscută în literatura CAGD ca *metoda de Boor - Boehm* [17][18]. Curba B-spline este generată prin aplicarea algoritmului de Casteljau pentru fiecare

secvență de 4 puncte Bézier, corespunzătoare unei porțiuni polinomiale a curbei compuse.

Din modul de generare a unei cubice B-spline se pot deduce câteva din proprietățile acestor curbe:

- proprietatea de acoperire convexă
- precizie liniară
- invarianță în raport cu relațiile afine
- simetrie
- interpolarea punctului de sfârșit
- proprietatea de diminuare a variației
- control local

O proprietate importantă de care curbele Bézier nu se bucură este *controlul local*. În cazul curbelor Bézier, modificarea unui punct al poligonului de control afectează întreaga curbă, ceea ce indică un control global. Pentru o cubică B-spline, dacă un punct al poligonului de Boor este modificat, va implica o schimbare asupra a 4 segmente ale curbei din vecinătatea punctului respectiv, curba păstrându-și forma nemodificată în celelalte porțiuni. Această calitate conferă o valoare deosebită curbelor B-spline în domeniul proiectării. Dacă o parte a curbei a fost complet generată se poate realiza modificarea curbei în alte regiuni fără teama de a afecta porțiunea corectă [30][32][98].

Figura 2.3.2 prezintă o curbă B-spline complexă generată cu ajutorul algoritmului descris.

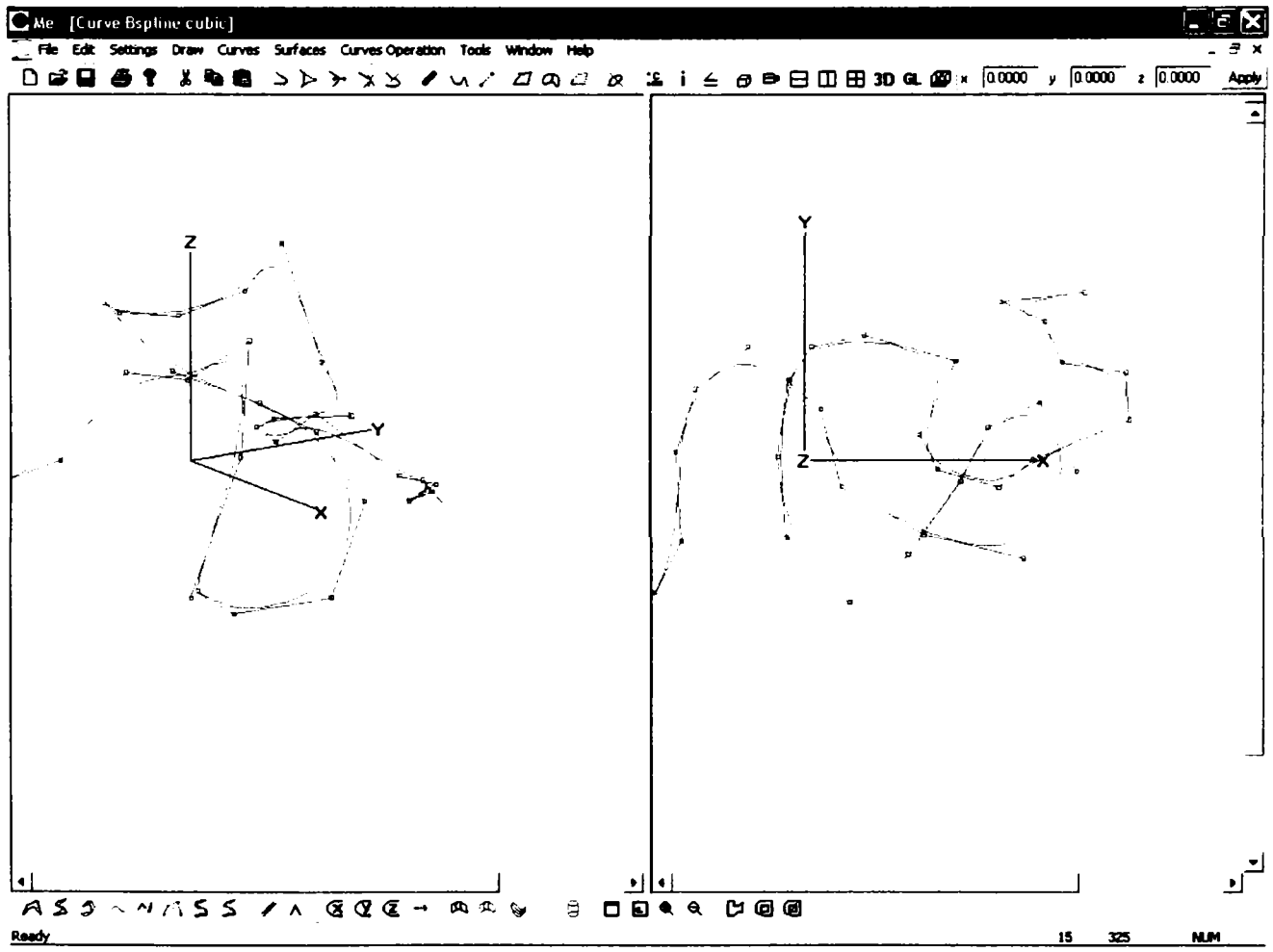

Fig. 2.3.2 Curbă B-spline

Curba urmăreşte fidel silueta poligonului de control pe porţiuni. Dacă pentru generarea curbei se utilează metoda Bézier sau Horner-Bézier rezultatul este cu totul diferit si aproximarea poligonului de control este mult mai slabă.

Funcţia membră BsplineCubicDeschis() a clasei CMeCurve cuprinde algoritmul de generare a poligonului Bézier pe porţiuni pentru a curbă spline cubică deschisă. Primul pas îl constituie generarea secvenţei de noduri corespunzătoare poligonului de Boor stocat în variabila membră controlPoints. Se determină apoi dimensiunea fiecărui interval dintre noduri. Prin apelarea metodei CalculPuncteBezierDeschis() se obţine poligonul Bézier pe porţiuni în tabloul bezierPoints.

```
void CMeCurve::CalculPuncteBezierDeschis(double delta[], short nr_points)
{
    short i, k;
    double delt;
    // calcul puncte b3i-2, b3i-1
    for(i = 2, k = 3; i <= nr_points-1; i ++, k++)
    {
        delt = delta[i-2] + delta[i-1] + delta[i];
        bezierPoints[3*i-2] =    controlPolygon[k-1] * (delta[i-1] + delta[i])/delt +
                                 controlPolygon[k] * delta[i-2]/delt;
        bezierPoints[3*i-1] =    controlPolygon[k-1] * delta[i]/delt +
                                 controlPolygon[k] * (delta[i-2] + delta[i-1])/delt;
    }
    // calcul puncte b3i
    for(i = 2; i <= nr_points-2; i ++)
        bezierPoints[3*i] =      bezierPoints[3*i-1] * delta[i]/(delta[i-1] + delta[i]) +
                                 bezierPoints[3*i+1] * delta[i-1]/(delta[i-1] + delta[i]);
    bezierPoints[0] = controlPolygon[0]; // punctul d-1 din punctele de Boor
    bezierPoints[1] =   controlPolygon[1]; // punctul d0 din punctele de Boor
    bezierPoints[2] =   controlPolygon[1] * delta[1]/(delta[0] + delta[1]) +
                        controlPolygon[2] * delta[0]/(delta[0]+delta[1]);
    bezierPoints[3] =   bezierPoints[2] * delta[1]/(delta[0] + delta[1]) +
                        bezierPoints[4] * delta[0]/(delta[0] + delta[1]);
    bezierPoints[3*nr_points-2]  = controlPolygon[nr_points] * delta[nr_points-1]/
                                   (delta[nr_points-2] + delta[nr_points-1]) +
                                   controlPolygon[nr_points+1] * delta[nr_points-2]/
                                   (delta[nr_points-2] + delta[nr_points-1]);
    bezierPoints[3*nr_points-1] = controlPolygon[nr_points+1];
    bezierPoints[3*nr_points] = controlPolygon[nr_points+2]; // punctul de Boor dL+1
    bezierPoints[3*(nr_points-1)]  = bezierPoints[3*nr_points-4] * delta[nr_points-1]/
                                     (delta[nr_points-2] + delta[nr_points-1]) +
                                     bezierPoints[3*nr_points-2] * delta[nr_points-2]/
                                     (delta[nr_points-2] + delta[nr_points-1]);
}

void CMeCurve::BSplineCubicDeschis(short L)
{
    double *delta, *knots;
    // setarea dimensiunii tabloului de noduri si de diferenţe
    knots = new double[L+1];
    delta = new double[L+1];
    // calculul secvenţei de noduri a curbei
    DetKnots1(knots, 1, L);
    // calculul diferenţelor ui - ui-1
    DetDif(knots,delta, L);
    // calculul punctelor poligonului Bézier pe porţiuni
```

```
    CalculPuncteBezierDeschis(delta, L);
    delete []knots;
    delete []delta;
}
```

2.3.4 Parametrizarea cubicei B-spline

Construcția unei cubice B-spline pornește de la poligonul de Boor și secvența de noduri corespunzătoare. În domeniul proiectării formelor libere, așa cum este cazul la construirea spațiilor cu geometrie complexă cum sunt roboții, punctele de Boor sunt generate interactiv. Alegerea intervalului de parametrizare al curbei și a secvenței de noduri constituie o problemă, deoarece acestea joacă un rol important în generarea curbei. O metodă simplă, este de a realiza o împărțire uniformă a spațiului de definiție prin $u_i = i$, însă această metodă este prea rigidă în cele mai multe cazuri. Cele mai adecvate metode sunt cele care încorporează în secvența de noduri caracteristicile geometrice ale poligonului de Boor. Una din cele mai uzuale metode de parametrizare este metoda *lungimii corzii*, care ia în calcul lungimea segmentelor poligonului de Boor [32][55]:

$$\begin{aligned} u_0 &= 0 \\ u_1 &= \|d_{-1}d_1\| \\ u_i &= u_{i-1} + \|d_{i-1}d_i\| \qquad i = 2,\ldots,L-1 \\ u_L &= u_{L-1} + \|d_{L-1}d_{L+1}\| \end{aligned} \tag{2.3.13}$$

Prin această parametrizare $\Delta_{i-1} = u_i - u_{i-1}$ este egal cu lungimea corzii de extremități $d_{i-1}d_i$.

Secvența de noduri pentru curbele spline cubice deschise a fost generată cu metoda DetKnots1(), iar calculul lungimii domeniului fiecărui poligon pe porțiuni a fost realizat cu metoda DetDif().

```
void CMeCurve::DetKnots1(double knots[], short id, short nr_points)
{
    Vector v;
    short i, k;
    knots[0] = 0;
    for(i = id + 1, k = 1; k <= nr_points; i++, k++)
    {
        v  = controlPolygon[i] - controlPolygon[i-1];
        knots[k] = knots[k-1] + v.VECT_LENGTH();
    }
}

void CMeCurve::DetDif(double knots[], double delta[], short nr_points)
{
    short i;
    for(i = 0; i <= nr_points-1; i++)
        delta[i] = knots[i+1] - knots[i];
}
```

2.3.5 Cubice B-spline închise

Curba polinomială parametrizată de $c:[u_0,u_L]\to E^3$ este o curbă închisă dacă punctul de început al curbei este identic cu punctul de sfârşit: $c(u_0)=c(u_L)$. În cazul cubicelor B-spline, datorită proprietăţii de interpolare a punctelor de capăt, poligonul de control de Boor al unei B-spline închise este de asemenea închis: $d_0=d_L$.

Punctele de control de Boor $d_0,d_1,\ldots,d_{L-1},d_L=d_0$ şi secvenţa de noduri $u_0<u_1<\ldots<u_{L-1}<u_L$ determină complet o curbă cubică B-spline închisă.

La fel ca şi în cazul curbelor deschise, algoritmul de generare se bazează pe rezultatele relaţiilor dintre punctele de Boor şi punctele poligonului Bézier pe bucăţi. Arcele de curbă Bézier generate de poligonul de control $b_{3i},b_{3i+1},b_{3i+2},b_{3i+3}$, $i=\overline{0,L-1}$ sunt racordate în b_{3i} printr-un racord de clasă C^2. Acesta implică existenţa unui punct d_i astfel încât punctele b_{3i-2},b_{3i-1},d_i şi respectiv d_i,b_{3i+1},b_{3i+2} sunt coliniare iar punctul b_{3i-1} împarte segmentul $b_{3i-2}d_i$ în raportul $\frac{\Delta_{i-1}}{\Delta_i}$, respectiv b_{3i+1} împarte segmentul d_ib_{3i+2} în raportul $\frac{\Delta_i}{\Delta_{i+1}}$. În plus, faţă de curbele deschise, datorită identităţii $b_0=b_{3L}$ există o conexiune de clasă C^2 şi între cubicele Bézier $b_{3L-3},b_{3L-2},b_{3L-1},b_{3L}$ şi b_0,b_1,b_2,b_3, ceea ce conduce la existenţa unui punct d cu proprietatea că punctele b_{3L-2},b_{3L-1},d, respectiv d,b_1,b_2 sunt coliniare. Acest punct este identic cu punctul de început şi de sfârşit al poligonului de Boor $d=d_0=d_L$ [17].

Astfel, la fel ca şi în cazul general, punctele Bézier interioare b_{3i-2},b_{3i-1} sunt obţinute cu ajutorul relaţiilor 2.3.9 şi 2.3.10, iar punctele de joncţiune b_{3i} cu formula 2.3.11.

Se determină punctele Bézier corespunzătoare subpoligonului de Boor $d_{L-1},d_L=d_0,d_1$. Punctul d_{L-1} rezultă din racordul arcelor $s([u_{L-2},u_{L-1}])$ şi $s([u_{L-1},u_L])$. Conform acestei conexiuni, punctele $d_{L-1},b_{3L-2},b_{3L-1}$ sunt coliniare iar b_{3L-2} împarte segmentul $d_{L-1}b_{3L-1}$ în raportul $\frac{\Delta_{L-2}}{\Delta_{L-1}}$. În mod similar, b_1 rezultă din racordul arcelor $s([u_0,u_1])$ şi $s([u_1,u_2])$ şi punctele d_1,b_1,b_2 sunt coliniare iar punctul b_1 împarte segmentul d_1b_2 în raportul $\frac{\Delta_0}{\Delta_1}$.

Punctele de control Bézier interioare ale primului arc al curbei spline rezultă din relaţiile:

$$\begin{aligned} b_1&=\frac{\Delta_0+\Delta_1}{\Delta}d_0+\frac{\Delta_{L-1}}{\Delta}d_1\\ b_2&=\frac{\Delta_1}{\Delta}d_0+\frac{\Delta_{L-1}+\Delta_0}{\Delta}d_1\\ \Delta&=\Delta_{L-1}+\Delta_0+\Delta_1 \end{aligned} \qquad (2.3.14)$$

iar punctele Bézier ale ultimului arc al cubicei spline:

$$b_{3L-2} = \frac{\Delta_{L-1} + \Delta_0}{\Delta} d_{L-1} + \frac{\Delta_{L-2}}{\Delta} d_L$$
$$b_{3L-1} = \frac{\Delta_0}{\Delta} d_{L-1} + \frac{\Delta_{L-2} + \Delta_{L-1}}{\Delta} d_L \qquad (2.3.15)$$
$$\Delta = \Delta_{L-2} + \Delta_{L-1} + \Delta_0$$

Punctele de joncțiune $b_{3L-3}, b_{3L} = b_0, b_3$ se calculează cu formulele [17][18]:

$$b_{3L-3} = \frac{\Delta_{L-1}}{\Delta_{L-2} + \Delta_{L-1}} b_{3L-2} + \frac{\Delta_{L-2}}{\Delta_{L-2} + \Delta_{L-1}} b_{3L-4}$$
$$b_{3L} = b_0 = \frac{\Delta_0}{\Delta_{L-1} + \Delta_0} b_{3L-1} + \frac{\Delta_{L-1}}{\Delta_{L-1} + \Delta_0} b_1 \qquad (2.3.16)$$
$$b_3 = \frac{\Delta_1}{\Delta_0 + \Delta_1} b_2 + \frac{\Delta_0}{\Delta_0 + \Delta_1} b_4$$

Cunoscând punctele poligonului Bézier pe porțiuni, curba B-spline închisă se generează aplicând algoritmul de Casteljau pentru fiecare poligon Bézier.

În general secvența de noduri nu este cunoscută. O alegere optimă a nodurilor care ia în considerare geometria curbei este:

$$u_0 = 0$$
$$u_i = u_{i-1} + dist(d_{i-1}, d_i), \quad i = \overline{1, L} \qquad (2.3.17)$$

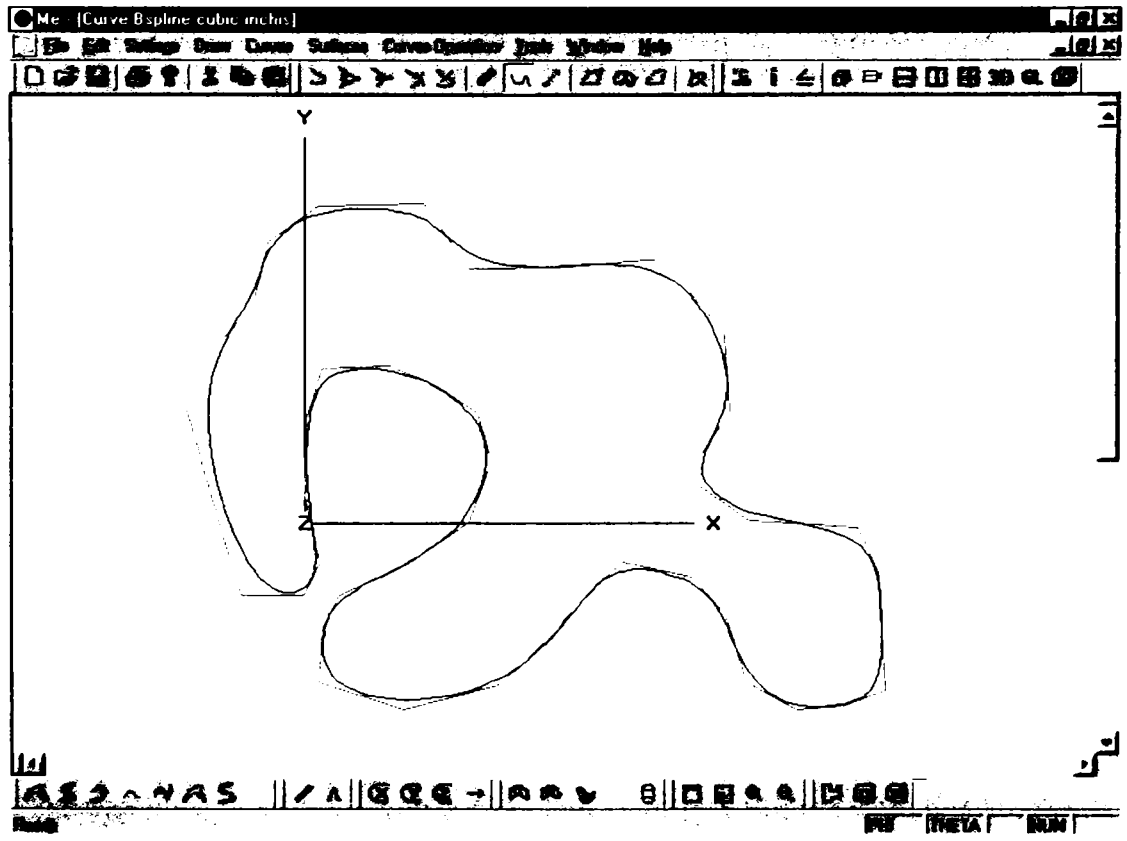

Fig. 2.3.3 Curbă B-spline cubică închisă

La fel ca şi în cazul curbelor spline generale, metodele BsplineCubicInchis() şi CalculPuncteBezierInchis() realizează calculul punctelor poligonului Bézier pe porţiuni pentru curbele spline închise. Modul de construire a secvenţei de noduri este acelaşi ca şi în
cazul curbelor spline deschise.

```
void CMeCurve::CalculPuncteBezierInchis(double delta[], short nr_points)
{
    short i;
    double delt;
    for(i = 2; i <= nr_points-1; i ++)
    {
        delt = delta[i-2] + delta[i-1] + delta[i];
        bezierPoints[3*i-2] = controlPolygon[i-1] * (delta[i-1] + delta[i])/delt +
                              controlPolygon[i] * delta[i-2]/delt;
        bezierPoints[3*i-1] =  controlPolygon[i-1] * delta[i]/delt +
                               controlPolygon[i] * (delta[i-2] + delta[i-1])/delt;
    }
    // calcul puncte b3i
    for(i = 2; i <= nr_points-2; i ++)
        bezierPoints[3*i] = bezierPoints[3*i-1] * delta[i]/(delta[i-1] + delta[i]) +
                            bezierPoints[3*i+1] * delta[i-1]/(delta[i-1] + delta[i]);
    // calculul primelor si ultimelor puncte Bézier
    bezierPoints[3*nr_points-2] = (controlPolygon[nr_points-1] * (delta[nr_points-1] +
                                   delta[0]) + controlPolygon[nr_points] * delta[nr_points-2])/
                                  (delta[nr_points-2] + delta[nr_points-1] + delta[0]);

    bezierPoints[3*nr_points-1] = (controlPolygon[nr_points-1] * delta[0] +
                                   controlPolygon[nr_points] * (delta[nr_points-2] + delta[nr_points-1]))/
                                   (delta[nr_points-2] + delta[nr_points-1] + delta[0]);
    bezierPoints[1] = (controlPolygon[0] * (delta[0] + delta[1]) + controlPolygon[1] *
                       delta[nr_points-1])/(delta[nr_points-1] + delta[0] + delta[1]);
    bezierPoints[2] = (controlPolygon[0] * delta[1] + controlPolygon[1] * (delta[nr_points-1] +
                       delta[0]))/ (delta[nr_points-1] + delta[0] + delta[1]);
    bezierPoints[3] = (bezierPoints[2] * delta[1] + bezierPoints[4] * delta[0])/
                       (delta[0] + delta[1]);
    bezierPoints[3*nr_points-3] = (bezierPoints[3*nr_points-4] * delta[nr_points-1] +
                                   bezierPoints[3*nr_points-2] * delta[nr_points-2])/
                                   (delta[nr_points-2] + delta[nr_points-1]);
    bezierPoints[3*nr_points] = (bezierPoints[3*nr_points-1] * delta[0] + bezierPoints[1] *
                                 delta[nr_points-1])/(delta[nr_points-1] + delta[0]);
    bezierPoints[0] = bezierPoints[3*nr_points];
}

void CMeCurve::BSplineCubicInchis(short L)
{
    double *knots, *delta;
    knots = new double[L+1];
    delta = new double[L+1];
    // calculul secvenţei de noduri
    DetKnots1(knots, 0, L);
    // calculul diferenţelor ui - ui-1
    DetDif(knots,delta, L);
    // calculul punctelor poligonului Bézier pe porţiuni închis
    CalculPuncteBezierInchis(delta, L);
    delete []knots;
    delete []delta;
}
```

Funcția principală care generează o curbă spline cubică, BSplineCubic(), determină mai întâi tipul poligonului de Boor al curbei şi, în funcție de acesta construieşte poligonul Bézier pe porțiuni. Numărul de segmente poligonale diferă pentru un spline închis față de o cubică spline generală. Ultimul pas îl constituie generarea punctelor curbei prin aplicarea algoritmului de Casteljau (metoda deCasteljau()) pentru fiecare segment poligonal (alcătuit din 4 puncte Bézier) generat.

```
void CMeCurve::BSplineCubic(void)
{
    short nr_puncte, L;

    nr_puncte = controlPolygon.GetSize();
    ASSERT(nr_puncte < N);
    // determinarea tipului de poligon de control: închis sau deschis
    if(controlPolygon[0] == controlPolygon[nr_puncte-1])
    {
        // poligonul de control este închis
        L = nr_puncte -1;
        // setarea tabloului temporar la dimensiunea necesara
        bezierPoints.SetSize(3*L +1);
        BSplineCubicInchis(L);
    }
    else
    {
        // poligonul de control este deschis
        L = nr_puncte-3;
        if(L < 2)
        {
            // nu se poate genera o curba spline cubica
            return ;
        }
        // setarea tabloului temporar la dimensiunea necesara
        bezierPoints.SetSize(3*L +1);
        BSplineCubicDeschis(L);
    }
    // construirea punctelor curbei pornind de la punctele de control Bézier pe porțiuni
    deCasteljau(L,10, 0);
    bezierPoints.RemoveAll();
}
```

2.4 Metode de interpolare

Curbele Bézier şi cubicele B-spline pe porţiuni sunt elemente de proiectare care aproximează poligonul de control generator. Forma lor urmăreşte conturul poligonului de control fiind mereu în înfăşurătoarea convexă a acestuia. Singurele puncte interpolate de curbe sunt punctele de început şi de sfârşit ale poligonului generator. Există domenii de proiectare în care curba trebuie, în mod necesar, să treacă prin anumite puncte şi nu doar în apropierea lor. Pentru rezolvarea acestor probleme, de-a lungul timpului, au fost dezvoltate metodele de interpolare de la interpolarea polinomială globală la interpolarea polinomială pe porţiuni. Aceste metode conduc la curbe care trec prin punctele de control ale poligonului generator.

2.4.1 Interpolarea polinomială

Interpolarea polinomială constituie un concept fundamental. Printre algoritmii rapizi şi simpli care se bazează intrinsec pe metodele polinomiale se pot prezenta:

Algoritmul lui Aitken

Pentru interpolarea unei secvenţe de n puncte p_i este necesară definirea unui *interpolant polinomial* p care să satisfacă constrângerile de interpolare $p(s_i) = p_i$, s_i este valoarea parametrului pentru care interpolantul trece prin punctul p_i [55][75].

Aitken a dezvoltat o metodă recursivă care determină un punct de pe interpolantul polinomial prin intermediul unei secvenţe de interpolări liniare repetate definite prin relaţia:

$$p_i^1(s) = \frac{s_{i+1} - s}{s_{i+1} - s_i} p_i + \frac{s - s_i}{s_{i+1} - s_i} p_{i+1} \qquad i = 0,\ldots,n-1 \qquad (2.4.1)$$

Se consideră că problema a fost rezolvată pentru cazul $n-1$, astfel încât s-a determinat funcţia polinomială p_0^{n-1} care interpolează primele n puncte $p_0,\ldots,p_{n-1}$ şi de asemenea a fost obţinut un polinomial p_1^{n-1} care interpolează ultimele n puncte $p_1,\ldots,p_n$. Pornind de la aceste consideraţii, forma interpolantului final p_0^n este:

$$p_0^n(s) = \frac{s_n - s}{s_n - s_0} p_0^{n-1}(s) + \frac{s - s_0}{s_n - s_0} p_1^{n-1}(s) \qquad (2.4.2)$$

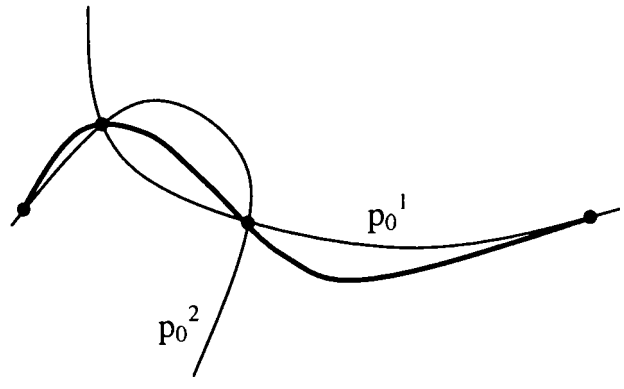

Fig. 2.4.1 Interpolantul polinomial cubic

Pentru $n = 3$, un polinomial de interpolare cubic este obținut ca o combinație a doi interpolanți cvadratici (figura 2.4.1).

Relația se poate generaliza astfel încât să rezolve problema interpolării polinomiale. Se numeşte *curbă interpolatoare Aitken* definită de punctele p_i, curba polinomială parametrizată de interpolantul polinomial $p:[s_0,s_n]\rightarrow E^3$, $p(s_i)=p_i$ definit prin:

$$p_i^r(s)=\frac{s_{i+r}-s}{s_{i+r}-s_i}p_i^{r-1}(s)+\frac{s-s_i}{s_{i+r}-s_i}p_{i+1}^{r-1}(s)\begin{cases} r=1,\ldots,n \\ i=0,\ldots,n-r \end{cases} \qquad (2.4.3)$$

$p_0^n(s)$ reprezintă un punct al curbei interpolatoare [74].

Pornind de la algoritmul lui Aitken, se pot deduce câteva caracteristici importante ale interpolării polinomiale:

- *Invarianța afină.* Proprietatea rezultă din faptul că algoritmul lui Aitken utilizează doar combinații baricentrice.

- *Precizia liniară.* Dacă toate punctele p_i sunt uniform distribuite pe un segment de dreaptă, toate punctele intermediare $p_i^r(s)$ sunt identice pentru r > 0. Astfel segmentul de dreaptă este reprodus de curbă.

Interpolarea polinomială nu prezintă proprietatea de acoperire convexă. Valoarea parametrului s în formula recursivă a lui Aitken nu este limitată la domeniul $[s_i,s_{i+r}]$. Astfel, algoritmul nu utilizează doar combinații convexe: $p_0^n(s)$ nu se află în mod cert în interiorul acoperirii convexe produse de p_i. Nici o metodă de interpolare a curbelor plane nu prezintă proprietatea de acoperire convexă.

Interpolarea polinomială nu are proprietatea de diminuare a variației. Din aceleaşi motive ca şi în cazul anterior această proprietate lipseşte, interpolarea polinomială poate prezenta o creştere a variației, o întindere a curbei ceea ce duce la o mai mică utilizare a metodei în problemele practice.

Implementarea algoritmului a fost realizată prin intermediul metodei CMeCurve::Aitken(). Pentru fiecare valoare a parametrului local *s*, prin interpolări succesive, se determină punctul corespunzător al curbei.

```
void CMeCurve::Aitken(void)
{
    short i,degree;
    double t, t1;
    CArray<CVertex, CVertex> tempTab;
    curvePoints.SetSize(N+1);
    // determinarea gradului curbei
    degree = controlPolygon.GetSize() - 1;
    tempTab.SetSize(degree + 1);
    for(i = 0; i <= N; i++)
    {
        tempTab.Copy(controlPolygon);
        t = (double)(i)/N;
        for (short r = 1; r <= degree; r++)
        {
            for (short k=0; k <= degree - r; k++)
            {
                t1=(degree*t-k)/((double) r);
                tempTab[k]= tempTab[k] * (1.0-t1) + tempTab[k+1] * t1;
            }
```

```
        }
        curvePoints[i]= tempTab[0];
        myDoc->ComputePersp(curvePoints[i]);
        SetPerspBBox(curvePoints[i]);
    }
}
```

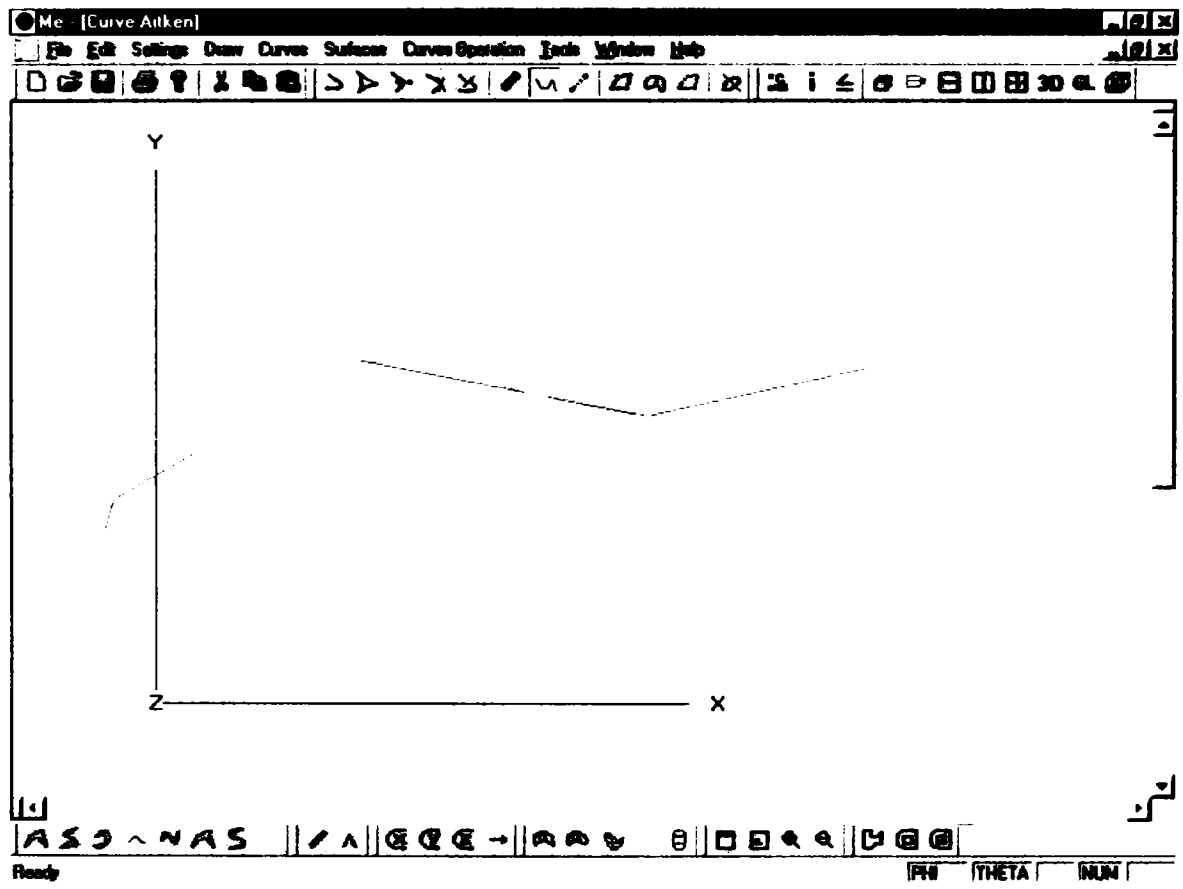

Fig. 2.4.2 Generarea unei curbe Aitken

Interpolarea Hermitică cubică

Interpolarea polinomială nu se restrânge doar la interpolarea punctelor (este vorba de datele inițiale de la care se pleacă în procesul de interpolare). Interpolarea se poate realiza pornind și de la alte informații cum ar fi derivatele. Aceasta conduce la o schemă de interpolare numită *interpolarea hermitică* [48][58][62]. Mai frecvent este cazul cubic, pentru care sunt date două puncte p_0, p_1, vectorii tangenți m_0 și m_1, se urmărește determinarea unei curbe polinomiale cubice p care să interpoleze aceste elemente:

$$\begin{aligned} p(0) &= p_0 \quad p(1) = p_1 \\ \dot{p}(0) &= m_0 \quad \dot{p}(1) = m_1 \end{aligned} \tag{2.4.4}$$

Pentru a scrie curba în forma Bézier trebuie determinate cele 4 puncte Bézier $b_0,\ldots,b_3$. Deoarece o curbă Bézier interpolează punctele de sfârșit ale poligonului de control rezultă: $b_0 = p_0$, $b_3 = p_1$. Derivata unei curbe Bézier ( ) în punctul de început este definită în raport cu primele două puncte Bézier: $\dot{b}_0 = 3(b_1 - b_0) = m_0$. Înlocuind în relație $b_0 = p_1$ rezultă $b_1 = p_0 + \frac{1}{3} m_0$.

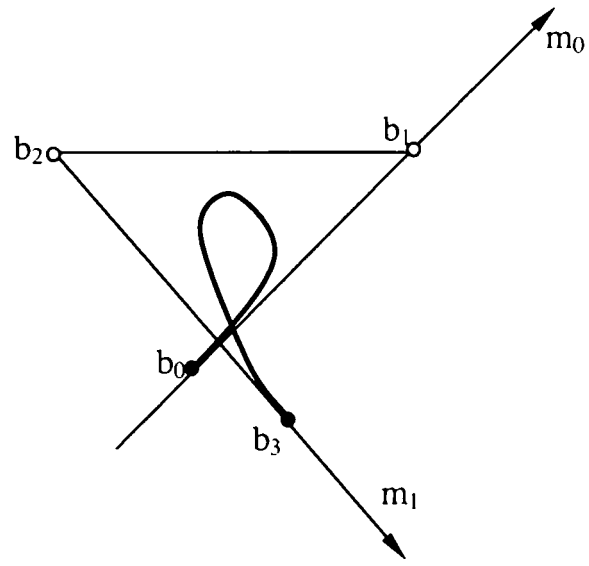

Fig. 2.4.3 Generarea punctelor Bézier în funcție de datele de interpolare.

Formula curbei polinomiale scrisă în baza Bernstein este:

$$p(s)=p_0B_0^3(s)+\left(p_0+\frac{1}{3}m_0\right)B_1^3(s)+\left(p_1-\frac{1}{3}m_1\right)B_2^3(s)+p_1B_3^3(s) \qquad (2.4.5)$$

Relația reprezintă forma interpolantului polinomial al punctelor p_0, p_1 şi a derivatelor m_0, m_1. Pentru ca datele ce intervin în operația de interpolare să apară în mod explicit în formula interpolantului este necesară rearanjarea relației (2.4.5) într-o *formă cardinală*:

$$p(t)=p_0H_0^3(t)+m_0H_1^3(t)+m_1H_2^3(t)+p_1H_3^3(t) \qquad (2.4.6)$$

$$H_0^3(t)=B_0^3(t)+B_1^3(t)$$

$$H_1^3(t)=\frac{1}{3}B_1^3(t)$$

$$H_2^3(t)=-\frac{1}{3}B_2^3(t)$$

$$H_3^3(t)=B_2^3(t)+B_3^3(t)$$

Relația definește *interpolantul Hermite cubic*. H_i^3 sunt denumite *funcții polinomiale Hermite* şi sunt ilustrate în figura 2.4.4.

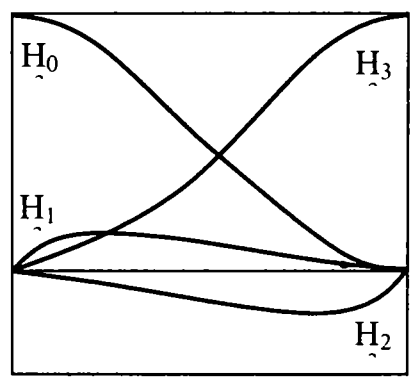

Fig. 2.4.4 Reprezentarea funcțiilor polinomiale Hermite

Aceste funcții sunt cardinale în raport cu evaluarea lor şi a derivatelor lor pentru $t=0$ şi $t=1$. Astfel, fiecare din funcțiile H_i^3 va fi egală cu 1 pentru una din aceste operații şi 0 pentru celelalte trei operații:

$$\begin{aligned}
&H_0^3(0)=1 \quad \frac{d}{dt}H_0^3(0)=0 \quad \frac{d}{dt}H_0^3(1)=0 \quad H_0^3(1)=0\\
&H_1^3(0)=0 \quad \frac{d}{dt}H_1^3(0)=1 \quad \frac{d}{dt}H_1^3(1)=0 \quad H_1^3(1)=0\\
&H_2^3(0)=0 \quad \frac{d}{dt}H_2^3(0)=0 \quad \frac{d}{dt}H_2^3(1)=1 \quad H_2^3(1)=0\\
&H_3^3(0)=0 \quad \frac{d}{dt}H_3^3(0)=0 \quad \frac{d}{dt}H_3^3(1)=0 \quad H_3^3(1)=1
\end{aligned} \tag{2.4.7}$$

O altă proprietate importantă a funcțiilor Hermite rezultă din geometria problemei de interpolare. Formula interpolantului Hermite constă dintr-o combinație de puncte şi vectori. Pentru ca relația să aibă sens geometric, suma coeficienților punctelor trebuie să fie 1 (adunarea punctelor are sens numai sub formă de combinație baricentrică):

$$H_0^3(s)+H_3^3(s)\equiv 1 \tag{2.4.8}$$

Acest lucru poate fi uşor verificat ținând seama de forma funcțiilor Hermite.

Interpolarea Hermite cubică prezintă încă o particularitate: nu este invariantă în raport cu transformarea afină a domeniului curbei. Interpolantul cubic a fost definit pe intervalul $[0,1]$. Se aplică o transformare afină a domeniului prin schimbarea lui s în $\hat{s}=(1-s)a+sb$, realizându-se astfel trecerea de la intervalul $[0,1]$ la intervalul $[a,b]$. Interpolantul Hermite devine:

$$\hat{p}(\hat{s})=p_0\hat{H}_0^3(\hat{s})+m_0\hat{H}_1^3(\hat{s})+m_1\hat{H}_2^3(\hat{s})+p_1\hat{H}_3^3(\hat{s})$$

unde $\hat{H}_i^3(\hat{s})$ sunt definite prin proprietățile lor de cardinalitate. Pentru a satisface aceste cerințe noul $\hat{H}_i^3(\hat{s})$ trebuie să difere de funcțiile originale $H_i^3(s)$. Se obține:

$$\begin{aligned}
&\hat{H}_0^3(\hat{s})=H_0^3(s)\\
&\hat{H}_1^3(\hat{s})=(b-a)H_1^3(s)\\
&\hat{H}_2^3(\hat{s})=(b-a)H_2^3(s)\\
&\hat{H}_3^3(\hat{s})=H_3^3(s)
\end{aligned}$$

unde $s\in[0,1]$ este un parametru local al intervalului $[a,b]$. Evaluarea formulei interpolantului pentru $\hat{s}=a$ şi $\hat{s}=b$ conduce la $\hat{p}(a)=p_0$, $\hat{p}(b)=p_1$. Derivatele însă s-au modificat. În punctul de început derivata este $\frac{d\hat{p}(a)}{dt}=(b-a)m_0$ şi în mod similar, în punctul de sfârşit $\frac{d\hat{p}(b)}{dt}=(b-a)m_1$. Astfel o transformare afină de domeniu schimbă curba dacă vectorii tangenți nu sunt modificați în mod corespunzător.

Pentru a menține aceeaşi curbă după o transformare afină a domeniului, trebuie schimbată lungimea vectorilor tangenți: dacă lungimea domeniului s-a modificat printr-un factor α, este necesară înlocuirea lui m_0 şi m_1 cu $\frac{m_0}{\alpha}$ respectiv $\frac{m_1}{\alpha}$. Există o explicație intuitivă pentru această transformare .Interpretând parametrul curbei ca timp, se consideră că este nevoie de o singură unitate de timp pentru a traversa curba. După modificarea lungimii intervalului printr-un factor de 10, de exemplu, vor trebui 10 unități de timp pentru a traversa aceeaşi curbă, ceea ce conduce la o viteză mult mai mică de traversare. Întrucât mărimea derivatei indică viteza, aceasta trebuie micşorată cu factorul 10. Se produce o scurtare a vectorului tangent al curbei parametrice.

De asemenea curbele Hermite nu sunt simetrice. Dacă se înlocuieşte *t* prin 1 - *t* (se consideră din nou intervalul $[0,1]$ ca domeniu), coeficienții curbei nu pot fi doar renumerotați, aşa cum se întâmplă în cazul curbei Bézier. Vectorii tangenți trebuiesc *inversați*. Aceasta rezultă din transformarea de parametru, realizată prin aplicarea unei relații afine intervalului $[0,1]$ care îi asociază intervalul $[1,0]$, astfel încât se inversează direcția acestuia.

Dependența formei cubice Hermite de domeniu este un dezavantaj important, iar ignorarea lui poate conduce la erori de programare. Se preferă utilizarea formei Bézier în toate aplicațiile posibile.

Generarea unui interpolant Hermite cubic a fost realizată cu funcția membră CMeCurve::Hermite(). Variabila membră controlPolygon stochează atât punctele p_0, p_1 care sunt interpolate cât şi valorile tangentelor în aceste puncte m_0, m_1. Pornind de la constrângerile de interpolare, sunt calculate punctele poligonului Bézier corespunzător interpolantului, şi prin apelul algoritmului lui de Casteljau se construieşte curba interpolatoare.

```
void CMeCurve::Hermite(void)
{
    // controlPolygon conține date de interpolat p0, m0, m1, p1
    // calculul poligonului Bézier corespunzător interpolantului Hermite
    bezierPoints.SetSize(4);
    bezierPoints[0].coord = controlPolygon[0].coord;
    bezierPoints[3].coord = controlPolygon[3].coord;
    bezierPoints[1].coord = controlPolygon[0].coord + controlPolygon[1].coord*1/3;
    bezierPoints[2].coord = controlPolygon[3].coord - controlPolygon[2].coord*1/3;
    // generarea curbei prin aplicarea algoritmului lui deCasteljau
    deCasteljau(4, 20, 0);
}
```

Forma de monom a curbelor Hermite cubice

O altă modalitate de reprezentare a curbelor interpolatoare Hermite o constituie forma de monom în baza canonică $B = (1, s, s^2, s^3)$:

$$p(s) = \sum_{i=0}^{3} a_i s^i, \quad st \in [0,1] \tag{2.4.9}$$

Deşi în această formă, coeficienții c_i ai relației nu au o semnificație geometrică $(1 + t + t^2 + t^3 \neq 1)$, scrierea în baza canonică permite determinarea unei reprezentări

matriciale a curbelor Hermite care poate fi implementată uşor pe calculator. Matricial formula devine:

$$p(s)=SA,\quad S=\left[s^3,s^2,s,1\right],\ A=\left[a_3,a_2,a_1,a_0\right]^T \tag{2.4.10}$$

C este denumită *matricea coeficienţilor algebrici.*

Vectorul tangent la curba Hermite în punctul u este obţinut prin derivarea relaţiei 2.4.11. Determinarea coeficienţilor c_i în funcţie de datele de interpolat, se face evaluând relaţia şi derivata sa în punctele de capăt $u=0$, $u=1$:

$$\begin{aligned}
&p(0)=a_0=p_0\\
&p(1)=a_0+a_1+a_2+a_3=p_1\\
&\dot{p}(0)=a_1=m_0\\
&\dot{p}(1)=a_1+2a_2+3a_3=m_1
\end{aligned}$$

Rezultă:

$$\begin{aligned}
&a_0=p_0\\
&a_1=m_0\\
&a_2=3(p_1-p_0)-2m_0-m_1\\
&a_3=2(p_0-p_1)+m_0+m_1
\end{aligned} \tag{2.4.11}$$

Pornind de la aceste relaţii, formula curbei poate fi scrisă:

$$p(s)=(2s^3-3s^2+1)p_0+(-2s^3+3s^2)p_1+(s^3-2s^2+s)m_0+(s^3-s^2)m_1 \tag{2.4.12}$$

De asemenea formula tangentei devine:

$$\dot{p}(s)=(6s^2-6s)p_0+(-6s^2+6s)p_1+(3s^2-4s+1)m_0+(3s^2-2s)m_1 \tag{2.4.13}$$

Funcţiile în s din cele două relaţii sunt numite *funcţii de amestecare* (blending function). Primele două funcţii "amestecă" punctele p_0 şi p_1, în timp ce ultimele combină tangentele m_0 şi m_1 pentru a produce membrul drept al fiecărei relaţii.

Forma matriceală poate fi rescrisă:

$$p(t)=S[M_H]V \tag{2.4.14}$$

unde $[M_H]=\begin{bmatrix}2&-2&1&1\\-3&3&-2&-1\\0&0&1&0\\1&0&0&0\end{bmatrix}$ este *matricea Hermite* iar

$V=[p_0,p_1,m_0,m_1]$ este *vectorul geometric* [60][62].

Comparând formele matriciale ale relației se obține $A = [M_H]V$. Similar, formula tangentei poate fi scrisă:

$$\dot{p}(s) = A[M_H]V \tag{2.4.15}$$

$[M_H] = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 6 & -6 & 3 & 3 \\ -6 & 6 & -4 & -2 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ este *matricea Hermite derivată*

Relațiile obținute descriu curba Hermite interpolatoare în funcție de vectorii şi tangentele de capăt. Forma curbei este controlată de aceste elemente geometrice. Dacă punctele de capăt sunt fixe, forma curbei se poate schimba prin modificarea lungimii sau a direcției tangentelor.

Metoda CMeCurve::HermiteMonomial() realizează calculul curbei interpolante Hermite cubice folosind forma monomială a acesteia. Clasa Vector care păstrează coordonatele unui punct, clasă agregată în cadrul clasei CVertex (ce defineşte în mod complet punctul), conține un set de operatori matematici suprascrişi astfel încât operațiile matematice cu puncte 3D să se realizeze cât mai simplu.

```
void CMeCurve::HermiteMonomial(void)
{
   double t;
   // controlPolygon conține datele de interpolat p0, m0, m1, p1
   curvePoints.SetSize(N);
   for(short i = 0; i < N; i++)
   {
      t = (double)(i)/(N-1);
      curvePoints[i].coord = controlPolygon[0].coord*(2*pow(t,3) - 3*pow(t,2) + 1) +
                             controlPolygon[3].coord*(-2*pow(t,3) + 3*pow(t,2)) +
                             controlPolygon[1].coord*(pow(t,3) -2*pow(t,2) + t) +
                            controlPolygon[2].coord*(pow(t,3) - pow(t,2));
      myDoc->ComputePersp(curvePoints[i]);
   }
}
```

2.4.2 Interpolarea polinomială pe porțiuni

Interpolarea polinomială reprezintă un concept fundamental, însă pentru elemente complexe au fost dezvoltate metode mai precise. Cea mai cunoscută clasă de metode este a schemelor polinomiale pe porțiuni [17][31][32][91]. Toate aceste metode construiesc curbe alcătuite din porțiuni polinomiale de acelaşi grad şi care prezintă anumite condiții de continuitate precise. Datele de la care se porneşte sunt secvența de puncte de interpolat şi valorile corespunzătoare a parametrilor, uneori fiind adăugate informațiile legate de tangente.

Cele mai des întâlnite curbe interpolatoare sunt curbele cubice pe porțiuni de clasă de continuitate C^1 sau C^2. Elementul care caracterizează categoria curbelor cu grad mic de continuitate C^1 este *caracterul local*: dacă un punct de control este modificat, curba de interpolare se modifică doar în vecinătatea acestui punct.

Sunt definite noțiunile de bază referitoare la interpolarea cubică pe porțiuni. Există o mare varietate de metode de interpolare destinate rezolvării diverselor probleme de proiectare. Multe din aceste metode încearcă să păstreze caracteristicile formei moştenite din datele primite, de exemplu, convexitatea sau monotonia.

Interpolarea Hermite cubică pe porțiuni C^1

Conceptual aceasta este cea mai simplă dintre metodele de interpolare pe porțiuni. Se consideră secvența de puncte de interpolat $x_0,\ldots,x_L$ corespunzând valorilor parametrice $u_0,\ldots,u_L$ şi vectorii tangenți $m_0,\ldots,m_L$ în aceste puncte. Se doreşte determinarea curbei polinomiale cubice pe porțiuni de clasă C^1 care să interpoleze aceste date:

$$c(u_i)=x_i,\frac{d}{du}c(u_i)=m_i;i=0,...,L \qquad (2.4.16)$$

Soluția problemei o constituie o curbă Bézier pe porțiuni aşa cum este ilustrată în figura 2.4.5.

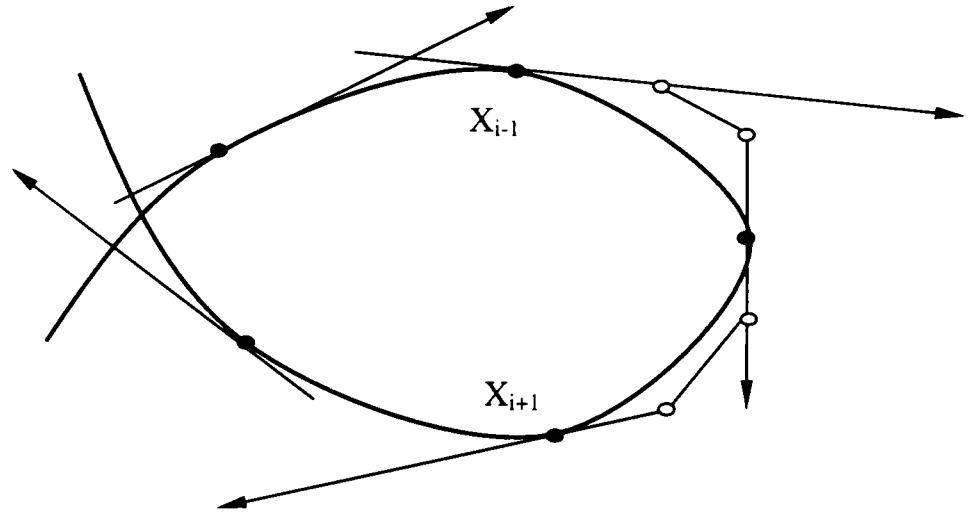

Fig. 2.4.5 Curba interpolatoare Hermite cubică pe porțiuni

Pentru aceasta se determină punctele Bézier ale poligonului de control pe bucăți. Punctele de joncțiune se obțin imediat ca fiind $b_{3i} = x_i$. Calculul punctelor Bézier interioare porneşte de la formula derivatei curbei Bézier :

$$\frac{d}{du}c(u_i)=\frac{3}{\Delta_{i-1}}(b_{3i}-b_{3i-1})=\frac{3}{\Delta_i}(b_{3i+1}-b_{3i})$$

unde $\Delta_i = \Delta_{ui}$. Astfel, punctele Bézier interioare b_{3i+1}, b_{3i-1}, $i=\overline{0,L-1}$ sunt date de relațiile:

$$b_{3i+1}=b_{3i}+\frac{\Delta_i}{3}m_i,\quad b_{3i-1}=b_{3i}-\frac{\Delta_{i-1}}{3}m_i \qquad (2.4.17)$$

Luând în considerare aceste formule, se poate construi forma Bézier a interpolantului Hermite cubic pe porțiuni C^1 [56][65][74]. Prin rearanjarea termenilor se obține interpolantul scris în baza Hermite. Pe intervalul $[u_i, u_{i+1}]$ interpolantul este definit:

$$c(u) = x_i \hat{H}_0^3(u) + m_i \hat{H}_1^3(u) + m_{i+1} \hat{H}_2^3(u) + x_{i+1} \hat{H}_3^3(u) \tag{2.4.18}$$

Funcțiile polinomiale Hermite sunt date de relațiile:

$$\begin{aligned} \hat{H}_0^3(u) &= B_0^3(s) + B_1^3(s) \\ \hat{H}_1^3(u) &= \frac{\Delta_i}{3} B_1^3(s) \\ \hat{H}_2^3(u) &= -\frac{\Delta_i}{3} B_2^3(s) \\ \hat{H}_3^3(u) &= B_2^3(s) + B_3^3(s) \end{aligned} \tag{2.4.19}$$

unde $s = \dfrac{(u - u_i)}{\Delta_i}$ este parametrul local al intervalului $[u_i, u_{i+1}]$.

Interpolarea cubică pe porțiuni C^1

Metoda generează o curbă cubică pe porțiuni de clasă C^1 care interpolează un set de puncte $x_0, \ldots, x_L$ ce corespund valorilor parametrice $u_0, \ldots, u_L$. Algoritmul se bazează pe determinarea tangentelor în punctele interpolate şi calculul apoi a punctelor poligoanelor Bézier pe porțiuni.

Cea mai simplă modalitate de estimarea a tangentei este cunoscută sub numele FMILL [61][66]. Metoda construieşte direcția tangentei l_i din punctul x_i astfel încât să fie paralelă cu coarda ce trece prin punctele x_{i-1}, x_{i+1}: $l_i = x_{i+1} - x_{i-1}$, $i = \overline{0, L-1}$ (Figura 2.4.6). Punctele Bézier interioare sunt plasate pe această direcție:

$$b_{3i-1} = b_{3i} - \frac{\Delta_{i-1}}{3(\Delta_{i-1} + \Delta_i)} l_i, \; b_{3i+1} = b_{3i} + \frac{\Delta_i}{3(\Delta_{i-1} + \Delta_i)} l_i \tag{2.4.20}$$

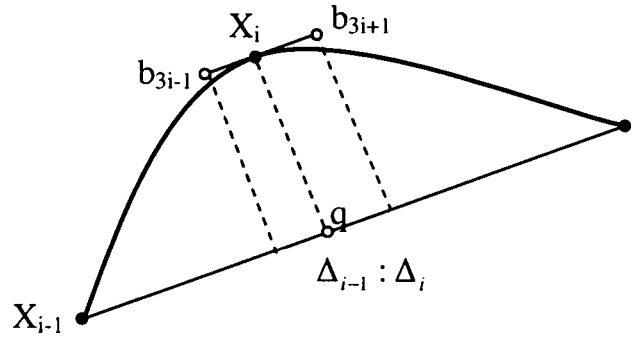

Fig. 2.4.6 Calculul punctelor Bézier interioare pentru interpolantul cubic C^1

Interpolantul este de asemenea cunoscut sub numele de spline Catmull-Rom. Această modalitate de construcție a punctelor Bézier interioare nu este valabilă în x_0 şi x_L. Pentru determinarea tangentelor în aceste puncte se folosesc relațiile Bessel [55][88].

Construcția tangentelor Bessel se bazează pe parabola $q_i(u)$ ce trece prin punctele x_{i-1}, x_i, x_{i+1} ce corespund valorilor parametrice u_{i-1}, u_i, u_{i+1}. Vectorul tangent m_i în punctul x_i va fi obținut ca derivata lui q_i în u_i. Scrisă în termenii datelor de intrare a problemei, formula derivatei este:

$$m_i = \frac{(1-\alpha_i)}{\Delta_{i-1}}\Delta x_{i-1} + \frac{\alpha_i}{\Delta_i}\Delta x_i; i = 1,\ldots,L-1$$
$$\alpha_i = \frac{\Delta_{i-1}}{\Delta_{i-1}+\Delta_i} \qquad (2.4.21)$$

Pentru punctele de sfârşit tangentele sunt:

$$m_0 = 2\frac{\Delta x_0}{\Delta_0} - m_1$$
$$m_L = 2\frac{\Delta x_{L-1}}{\Delta_{L-1}} - m_{L-1} \qquad (2.4.22)$$

Pornind de la aceste relații, sunt obținute punctele interioare pentru primul şi ultimul poligon Bézier pe porțiuni. Considerând punctele de joncțiune ca fiind $b_{3i} = x_i$, poligonul Bézier al interpolantului este complet determinat. Aplicarea algoritmului lui de Casteljau sau a schemei Horner-Bézier pentru fiecare segment poligonal format din 4 puncte conducerea la generarea curbei cubice interpolatoare.

2.4.3 Interpolarea spline cubică

Curbele B-spline cubice reprezintă un instrument puternic de proiectare, capabile să modeleze uşor forme complexe. Curbele au fost dezvoltate ca metode de aproximare, ce urmăresc forma poligonului de control, dar pot fi de asemenea folosite la rezolvarea problemei interpolării unui set de puncte date. Interpolarea spline cubică a fost introdusă în literatura CAGD de J. Ferguson în 1964 în timp ce baza matematică a fost studiată de teoria aproximărilor [60][94] .

Forma B-spline

Se consideră un set de L puncte $x_0,\ldots,x_L$ şi secvența de noduri corespunzătoare punctelor $u_0,\ldots,u_L$. Există o curbă B-spline cubică s, determinată de aceste noduri şi de $L+3$ puncte de control de Boor $d_{-1},\ldots,d_{L+1}$ astfel încât să interpoleze datele inițiale $c(u_i) = x_i$.

Soluția acestei probleme constă în determinarea relației dintre punctele x_i şi punctele de Boor d_i. Conexiunea între aceste puncte se realizează cu ajutorul poligonului

Bézier pe porţiuni. Orice curbă B-spline poate fi scrisă ca o curbă Bézier pe porţiuni de clasă C^2. Fiecare arc al cubicei spline este o curbă Bézier definită de un set de 4 puncte de control $b_{3i}, b_{3i+1}, b_{3i+2}, b_{3i+3}$, care împreună alcătuiesc poligonul de control Bézier pe porţiuni al cubicei B-spline. Deoarece o curbă Bézier interpolează punctele de capăt ale poligonului de control, secvenţa de puncte de interpolat poate fi interpretată ca puncte de joncţiune a poligonului Bézier pe porţiuni: $x_i = b_{3i} \quad i = \overline{0, L}$.

Din condiţia de racord de clasă C^1 a două cubice Bézier rezultă că punctele $b_{3i-1}, b_{3i}, b_{3i+1}$ sunt coliniare, iar punctul b_{3i} împarte segmentul $b_{3i-1}b_{3i+1}$ în raportul $\frac{\Delta_{i-1}}{\Delta_i}$.

Astfel punctele Bézier interioare sunt raportate la punctele x_i prin relaţia:

$$x_i = b_{3i} = \frac{\Delta_i b_{3i-1} + \Delta_{i-1} b_{3i+1}}{\Delta_{i-1} + \Delta_i} \quad i = 1, \ldots, L-1 \tag{2.4.23}$$

unde $\Delta_i = u_{i+1} - u_i$.

De asemenea din teoria cubicelor B-spline, punctele interioare pot fi scrise în raport cu punctele de Boor, conform formulelor 3.9 şi 3.10. Pentru punctele interioare de sfârşit ale poligonului pe porţiuni b_2, b_{3L-2}, situaţia fiind mai specială, se aplică formulele 3.12. Pornind de la aceste relaţii se poate scrie legătura dintre punctele de Boor d_i şi punctele x_i prin eliminarea punctelor Bézier. Se obţine:

$$(\Delta_{i-1} + \Delta_i) x_i = \alpha_i d_{i-1} + \beta_i d_i + \gamma_i d_{i+1} \tag{2.4.24}$$

$$\alpha_i = \frac{\Delta_i^2}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i}$$

$$\beta_i = \frac{\Delta_i(\Delta_{i-2} + \Delta_{i-1})}{\Delta_{i-2} + \Delta_{i-1} + \Delta_i} + \frac{\Delta_{i-1}(\Delta_i + \Delta_{i+1})}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}} \qquad i = \overline{2, L-2}$$

$$\gamma_i = \frac{\Delta_{i-1}^2}{\Delta_{i-1} + \Delta_i + \Delta_{i+1}}$$

S-a considerat $\Delta_{-1} = \Delta_L = 0$.

Pentru $i = 1$, luându-se în considerare forma specială a punctului b_2 se setează:

$$\alpha_1 = \frac{\Delta_1^2}{\Delta_0 + \Delta_1}$$

$$\beta_1 = \frac{\Delta_0 \Delta_1}{\Delta_0 + \Delta_1} + \frac{\Delta_0(\Delta_1 + \Delta_2)}{\Delta_0 + \Delta_1 + \Delta_2}$$

$$\gamma_1 = \frac{\Delta_0^2}{\Delta_0 + \Delta_1 + \Delta_2}$$

iar pentru $i = L-1$ coeficienţii sunt:

$$\alpha_{L-1} = \frac{\Delta_{L-1}^2}{\Delta_{L-3} + \Delta_{L-2} + \Delta_{L-1}}$$
$$\beta_{L-1} = \frac{\Delta_{L-1}(\Delta_{L-3} + \Delta_{L-2})}{\Delta_{L-3} + \Delta_{L-2} + \Delta_{L-1}} + \frac{\Delta_{L-2}\Delta_{L-1}}{\Delta_{L-2} + \Delta_{L-1}}$$
$$\gamma_{L-1} = \frac{\Delta_{L-2}^2}{\Delta_{L-2} + \Delta_{L-1}}$$

Dacă se aleg arbitrar punctele b_1 şi b_{3L-1}, formula ce relaţionează punctele interpolate x_i cu punctele de Boor se poate scrie sub forma unui sistem liniar:

$$\begin{bmatrix} 1 & & & & & \\ \alpha_1 & \beta_1 & \gamma_1 & & & \\ & & & & & \\ & & & \alpha_{L-1} & \beta_{L-1} & \gamma_{L-1} \\ & & & & & 1 \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{L-1} \\ d_L \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{L-1} \\ r_L \end{bmatrix} \qquad (2.4.25)$$

În această formulă:

$$r_0 = b_1$$
$$r_i = (\Delta_{i-1} + \Delta_i)x_i \qquad i = \overline{1, L-1}$$
$$r_L = b_{3L-1}$$

Sistemul este compatibil determinat fiind diagonal dominant. Prin rezolvarea sistemului sunt obţinute, din datele iniţiale, secvenţa de puncte de Boor $d_0, \ldots, d_L$. Primul şi ultimul punct al poligonului de control sunt date ca fiind punctele de extrem ale setului de interpolat $d_{-1} = x_0 d_{L+1} = x_L$.

În cadrul algoritmului de generare a curbelor B-spline cubice interpolatoare punctele Bézier b_1 şi b_{3L-1} au fost alese în mod arbitrar. Poziţia acestor puncte influenţează curbura primului şi ultimului arc al curbei B-spline şi deci, determinarea acestor puncte are un rol important. O metodă de calcul a punctelor se bazează pe tangentele Bessel [31][56][88].

Generarea curbelor B-spline cubice interpolatoare a fost realizată cu funcţia CMeCurve::BsplineCubicInterpolation(). Fiecărei etape a algoritmului îi corespunde o funcţie specifică. Secvenţa de noduri este construită cu funcţia membră CMeCurve::DetKnots2(), iar valorile Δ pentru fiecare interval sunt calculate cu CMeCurve::DetDifV(). Următorul pas îl constituie determinarea poligonului de control de Boor prin intermediul metodei CMeCurve::CalculPuncteDeBoor(). Folosind algoritmul de generare a curbelor B-spline cubice se construiesc punctele poligonului Bézier pe porţiuni CMeCurve::CalculPuncteDeBoorBezier(). Curba spline cubică interpolatoare este generată folosind funcţia CMeCurve::deCasteljau();

```
d CMeCurve::DetDifV(Vector knots[],double delta[], short L)

for(short i = 0; i < L; i++)
    delta[i] = knots[i+1][0] - knots[i][0];


d CMeCurve::CalculPuncteDeBoor(short L, double delta[], Vector deBoor[])

Vector *r, v1, v2, v3;
double *alfa, *beta, *gama;
short i;
double A, B, C, E, F, G;

alfa = new double[L+1];
beta = new double[L+1];
gama = new double[L+1];

r = new Vector[L+1];
// calcul puncte ri
for(i = 1; i < L; i++)
    r[i] = controlPolygon[i] * (delta[i-1] + delta[i]);
// r0 este punctul b1
A = (delta[0]+2*delta[1])/(3*(delta[0]+delta[1]));
B = (delta[0]+delta[1])/(3*delta[1]);
C = -(delta[0]*delta[0])/(3*(delta[0]+delta[1])*delta[1]);
v1 = controlPolygon[0] * A;
v2 = controlPolygon[1] * B;
v3 = controlPolygon[2] * C;
r[0] = v1 + v2;
r[0] += v3;
// rL este punctul b3L-1
E = -(delta[L-1]*delta[L-1])/(3*delta[L-2]*(delta[L-2]+delta[L-1]));
F = (delta[L-2]+delta[L-1])/(3*delta[L-2]);
G = (delta[L-1]+2*delta[L-2])/(3*(delta[L-2]+delta[L-1]));
v1 = controlPolygon[L-2] * E;
v2 = controlPolygon[L-1] * F;
v3 = controlPolygon[L] * G;
r[L] = v1 + v2;
r[L] += v3;

// calculul coeficienților sistemului sistemului tridiagonal
for(i = 2; i < L-1; i ++)
{
    alfa[i] = (delta[i]*delta[i])/(delta[i-2]+delta[i-1]+delta[i]);
    beta[i] = (delta[i]*(delta[i-2]+delta[i-1]))/(delta[i-2]+delta[i-1]+delta[i]) +
            (delta[i-1]*(delta[i]+delta[i+1]))/(delta[i-1]+delta[i]+delta[i+1]);
    gama[i] = (delta[i-1]*delta[i-1])/(delta[i-1]+delta[i]+delta[i+1]);
}
alfa[0] = 0;
beta[0] = 1;
gama[0] = 0;
alfa[1] = delta[1]*delta[1]/(delta[0]+delta[1]);
beta[1] = delta[0]*delta[1]/(delta[0]+delta[1]) +
        delta[0]*(delta[1]+delta[2])/(delta[0]+delta[1]+delta[2]);
gama[1] = delta[0]*delta[0]/(delta[0]+delta[1]+delta[2]);
alfa[L-1] = delta[L-1]*delta[L-1]/(delta[L-3]+delta[L-2]+delta[L-1]);
beta[L-1] = delta[L-1]*(delta[L-3]+delta[L-2])/(delta[L-3]+delta[L-2]+delta[L-1])+
            delta[L-2]*delta[L-1]/(delta[L-2]+delta[L-1]);
gama[L-1] = delta[L-2]*delta[L-2]/(delta[L-2]+delta[L-1]);
alfa[L] = 0;
```

```
    beta[L] = 1;
    gama[L] = 0;

    // rezolvarea sistemului tridiagonal
    CalculSistemTridiagonal(alfa, beta, gama, r, L+1, deBoor+1);
    deBoor[0] = controlPolygon[0];
    deBoor[L+2] = controlPolygon[L];

    delete []alfa;
    delete []beta;
    delete []gama;
    delete []r;
}

void CMeCurve::CalculPuncteDeBoorBezier(short L, Vector deBoor[], double delta[])
{
    short i;
    double delt;

    // calcul puncte b3i
    for(i = 0; i <= L; i++)
        bezierPoints[3*i] = controlPolygon[i];

    // calcul puncte b3i-2, b3i-1
    for(i = 2; i <= L-1; i ++)
    {
        delt = delta[i-2] + delta[i-1] + delta[i];
        bezierPoints[3*i-2] = deBoor[i-1] * (delta[i-1] + delta[i])/delt + deBoor[i] * delta[i-2]/delt;
        bezierPoints[3*i-1] = deBoor[i-1] * delta[i]/delt + deBoor[i] * (delta[i-2] + delta[i-1])/delt;
    }
    bezierPoints[2] = deBoor[0] * delta[1]/(delta[0] + delta[1]) +
                      deBoor[1] * delta[0]/(delta[0]+delta[1]);
    bezierPoints[3*L-2] = deBoor[L-1] * delta[L-1]/(delta[L-2] + delta[L-1])+
                      deBoor[L] * delta[L-2]/(delta[L-2] + delta[L-1]);
    bezierPoints[1] = deBoor[0];
    bezierPoints[3*L-1] = deBoor[L];
}

// funcția de interpolare prin curbe B-spline a unui set de puncte
void CMeCurve::BSplineCubicInterpolation(void)
{
    double *delta;
    Vector *knots, *deBoor;
    short L;
    CVertex vert;

    L = controlPolygon.GetSize() -1;
    // setarea tabloului temporar la dimensiunea necesara
    bezierPoints.SetSize(3*L +1);

    deBoor =  new Vector[L+3];
    knots = new Vector[L+1];
    delta = new double[L+1];

    // calcul noduri
    DetKnots2(knots, L);
    // calculul diferențelor ui - ui-1
    DetDifV(knots,delta, L);
    // calculul punctelor deBoor
```

```
    CalculPuncteDeBoor(L, delta, deBoor);
    // calcul puncte poligoane Bézier pe bucăți
    CalculPuncteDeBoorBezier(L, deBoor+1, delta);
    // construirea punctelor curbei pornind de la punctele de control Bézier pe porțiuni
    deCasteljau(L,40, 1);

    delete []delta;
    delete []knots;
    delete []deBoor;
}
```

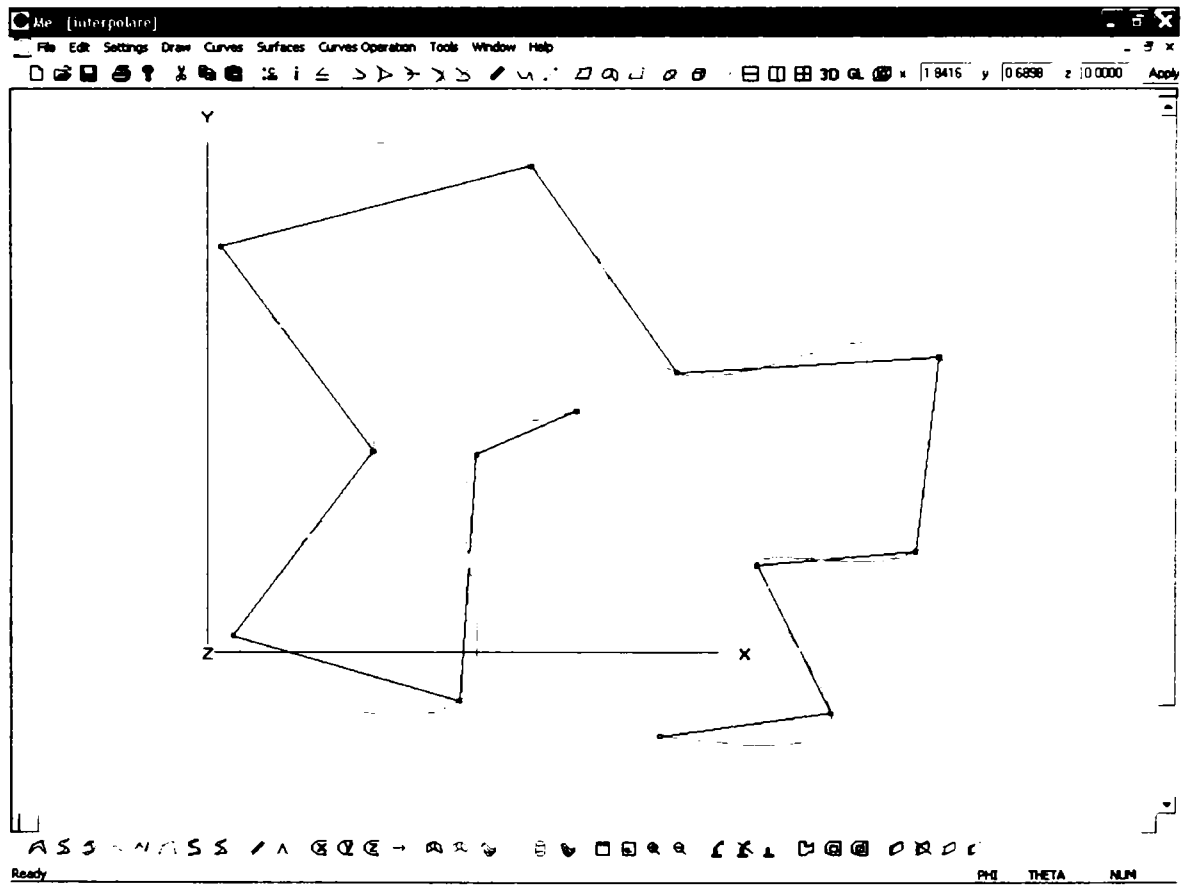

Fig. 2.4.7 Curbă B-spline cubică interpolatoare

2.4.4 Parametrizarea curbelor interpolatoare B-spline cubice

Datele inițiale de la care se pornește în procesul de interpolare sunt punctele de interpolat și secvența de noduri corespunzătoare lor. În general, valorile nodurilor nu sunt cunoscute și de aceea este necesară determinarea lor în mod automat. Calculul secvenței de noduri a curbei spline interpolatoare este important deoarece forma curbei depinde de modul în care acești parametrii au fost determinați.

O modalitate simplă de calcul a valorilor u_i este de a seta $u_i = i$. Metoda poartă numele de *parametrizare uniformă sau echidistantă*. Dezavantajul ei constă în faptul că este independentă de geometria punctelor de interpolat. Condiția de racord de clasă C^2 implică ca viteza și accelerația în punctele de interpolat să fie continue. Deoarece în cele mai multe cazuri lungimea arcelor curbei nu este egală, pentru o parametrizarea uniformă a acestora, condițiile de continuitate nu pot fi îndeplinite.

Pentru ca aceste cerinţe să fie satisfăcute este necesar ca divizarea domeniului curbei să se realizeze în mod proporţional cu distanţele dintre puncte:

$$\frac{\Delta_i}{\Delta_{i+1}} = \frac{\|\Delta x_i\|}{\|\Delta x_{i+1}\|}, \qquad i = \overline{0, L-2} \tag{2.4.26}$$

Parametrizarea este denumită *metoda lungimii corzii*. Formula nu defineşte în mod unic o secvenţă de noduri, de aceea în practică se setează $u_0 = 0, u_L = 1$ sau $u_0 = 0, u_L = L$. Rezultă un sistem diagonal dominant [88]. Metoda conduce la rezultate mai bune decât divizarea uniformă.

O altă parametrizare este *metoda centripetă*. Secvenţa de noduri rezultă din relaţia:

$$\frac{\Delta_i}{\Delta_{i+1}} = \sqrt{\frac{\|\Delta x_i\|}{\|\Delta x_{i+1}\|}}, \qquad i = \overline{0, L-2} \tag{2.4.27}$$

La fel ca şi în cazul metodei lungimii corzii nodurile extreme se setează $u_0 = 0, u_L = 1$.

Pentru generarea curbelor spline cubice interpolatoare, secvenţa de noduri a fost construită folosindu-se metoda lungimii corzii implementată prin metoda CMeCurve::DetKonts2(). Rezolvarea sistemului diagonal a fost realizată cu funcţia CMeCurve::CalculSistemTridiagonal().

```
void CMeCurve::CalculSistemTridiagonal(double a[], double b[], double c[], Vector r[], short n,
                                       Vector x[])
{
    for(short k = 1; k < n; k++)
    {
        b[k] = b[k] - c[k-1] * a[k]/b[k-1];
        r[k] = r[k] - r[k-1] * a[k]/b[k-1];
    }
    x[n-1] = r[n-1]/b[n-1];
    for(k = n-2; k >= 0; k--)
    x[k] = (r[k] - x[k+1] * c[k])/b[k];
}

void CMeCurve::DetKnots2(Vector knots[], short L)
{
    double *a, *b, *c, *val;
    short i;
    Vector vi, vii;
    double li, lii;
    Vector *r;

    a = new double[L];
    memset(a,0,L*sizeof(double));
    b = new double[L];
    memset(b,0,L*sizeof(double));
    c = new double[L];
    memset(c,0,L*sizeof(double));
    val = new double[L];
    memset(val,0,L*sizeof(double));
    r = new Vector[L];
    memset(r,0,L*sizeof(Vector));
```

```
    for(i = 0; i < L - 1; i++)
    {
        vi = controlPolygon[i+1] - controlPolygon[i];
        li = vi.VECT_LENGTH();
        vii = controlPolygon[i+2] - controlPolygon[i+1];
        lii = vii.VECT_LENGTH();
        val[i] = li/lii;
    }
    a[0] = 0;
    for(i = 1; i < L-1; i++)
        a[i] = -1;
    for(i = 0; i < L-1; i++)
        b[i] = 1+val[i];
    for(i = 0; i < L-2; i++)
        c[i] = -val[i];
    c[L-2] = 0;
    r[L-2][0] = val[L-2];
    CalculSistemTridiagonal(a, b, c, r, L-1, knots+1);
    knots[0][0] = 0;
    knots[L][0] = 1;

    delete []a;
    delete []b;
    delete []c;
    delete []val;
    delete []r;
}
```

2.4.5 Interpolarea curbelor închise

În mod frecvent în proiectare sunt utilizate curbe închise (Figura 2.4.8). Secvența de interpolare are punctul de început identic cu punctul de sfârşit. Din teoria generală a curbelor B-spline interpolatoare se deduce sistemul liniar care relaționează punctele de interpolat cu poligonul de control de Boor. Numărul de relații este redus întru-cât condițiile C^2 în $x_0 = x_L$ nu vor apărea de două ori în sistemul liniar. Acesta va lua forma:

$$\begin{bmatrix} \beta_0 & \gamma_0 & & & & \alpha_0 \\ \alpha_1 & \beta_1 & \gamma_1 & & & \\ & & & & & \\ & & & \alpha_{L-2} & \beta_{L-2} & \gamma_{L-2} \\ \gamma_{L-1} & & & & \alpha_{L-1} & \beta_{L-1} \end{bmatrix} \begin{bmatrix} d_0 \\ d_1 \\ \vdots \\ d_{L-2} \\ d_{L-1} \end{bmatrix} = \begin{bmatrix} r_0 \\ r_1 \\ \vdots \\ r_{L-2} \\ r_{L-1} \end{bmatrix} \tag{2.4.28}$$

Formulele pentru calculul valorilor $\alpha_i, \beta_i, \gamma_i, i = \overline{1, L-2}$ sunt similare cu cele din cazul general (4.26). De asemenea valorile din partea dreaptă a sistemului sunt date formula $r_i = (\Delta_{i-1} + \Delta_i)x_i, \quad i = \overline{1, L-2}$. Pentru ca aceste relații să aibă sens, se defineşte o *continuare periodică* a secvenței de noduri: $\Delta_{-1} = \Delta_{L-1}, \quad \Delta_{-2} = \Delta_{L-2}$.

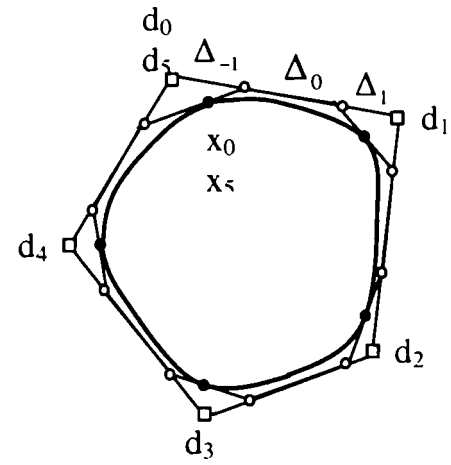

Figura 2.4.8 Curbă spline interpolatoarea cubică închisă

Luând în considerare rezultatele obținute la generarea cubicelor B-spline închise se calculează prima şi ultima relație a sistemului. Pentru prima formulă se porneşte de la relația $x_L = x_0 = b_0$. Punctul b_0 poate fi determinat în raport cu punctele b_1, b_{3L-1} din relația 2.3.16. Deoarece aceste puncte pot fi exprimate în raport cu punctele de Boor prin formulele 2.3.14 şi 3.15, rezultă:

$$x_0(\Delta_{L-1}+\Delta_0) = \frac{\Delta_0^2}{\Delta} d_{L-1} + \left[\frac{\Delta_0(\Delta_{L-2}+\Delta_{L-1})}{\Delta} + \frac{\Delta_{L-1}(\Delta_0+\Delta_1)}{\tilde{\Delta}}\right] d_0 + \frac{\Delta_{L-1}^2}{\tilde{\Delta}} d_1 \qquad (2.4.29)$$

$$\Delta = \Delta_{L-2}+\Delta_{L-1}+\Delta_0, \quad \tilde{\Delta} = \Delta_{L-1}+\Delta_0+\Delta_1$$

Relația conține formulele pentru valorile $r_0, \alpha_0, \beta_0, \gamma_0$.

Similar, pentru ultima relație pornind de la $x_{L-1} = b_{3L-3}$ şi ținând cont de aceleaşi relații se obține:

$$x_{L-1}(\Delta_{L-2}+\Delta_{L-1}) = \frac{\Delta_{L-1}^2}{\Delta} d_{L-2} + \frac{\Delta_{L-1}(\Delta_{L-3}+\Delta_{L-2})}{\Delta} d_{L-1} + \frac{\Delta_{L-2}(\Delta_{L-1}+\Delta_0)}{\tilde{\Delta}} d_{L-1} + \frac{\Delta_{L-2}^2}{\tilde{\Delta}} d_L$$

$$= \frac{\Delta_{L-1}^2}{\Delta} d_{L-2} + \left[\frac{\Delta_{L-1}(\Delta_{L-3}+\Delta_{L-2})}{\Delta} + \frac{\Delta_{L-2}(\Delta_{L-1}+\Delta_0)}{\tilde{\Delta}}\right] d_{L-1} + \frac{\Delta_{L-2}^2}{\tilde{\Delta}} d_0 \qquad (2.4.30)$$

Din formulă rezultă formulele pentru $r_{L-1}, \alpha_{L-1}, \beta_{L-1}, \gamma_{L-1}$.

Sistemul are L necunoscute $d_0, d_1, \ldots, d_{L-1}$ şi L relații. Nu este un sistem tridiagonal dar este în mod evident diagonal dominant [17][18][30][31].

Fiind determinate punctele poligonului de control de Boor, conform teoriei cubicelor B-spline închise se construieşte poligonul Bézier pe bucăți. Curba B-spline interpolatoare este generată prin aplicarea algoritmului de Casteljau fiecărui poligon de control Bézier $b_{3i-3}, b_{3i-2}, b_{3i-1}, b_{3i}$, $i = \overline{1,L}$.

2.5 Curbe B-spline de grad n

Curbele B-spline au fost investigate la începutul sec XIX de către N. Lobacevski. Ele au fost construite ca şi convoluţii ale distribuţiilor de probabilitate certe. În 1946 I. J. Schoenberg foloseşte curbele B-spline pentru liniarizarea (netezirea) statistică a datelor şi lucrarea sa a reprezentat începutul teoriei moderne a aproximărilor spline. Descoperirea relaţiilor de recurenţă pentru B-spline de către C. de Boor, M Cox si L. Mansfield reprezintă unul dintre cele mai importante elemente în cadrul teoriei legate de curbe [19][28][33].

Capitolul prezintă noţiunile teoretice de bază a curbele B-spline de grad arbitrar. Prima parte se referă la o definiţie geometrică a curbelor B-spline, bazată pe un principiu de subdivizare numit metoda de inserţie a unui nod, datorată lui W Boehm. Demonstraţia se restrânge la curbe neparametrice datorită faptului că o mare parte din teoria B-spline este explicată mai natural în acest mod. Construcţia geometrică permite un control direct asupra formei curbei.

Definiţia analitică a curbelor B-spline dezvoltată în ultima parte a capitolului oferă posibilitatea de a deduce o serie de proprietăţi afine şi diferenţiale ale acestor curbe.

2.5.1 Curbe Neparametrice. Funcţii Bézier, Funcţii B-spline

Metodele de proiectare descrise în capitolele anterioare conduc la curbe tridimensionale parametrice. O clasă particulară de curbe o reprezintă curbele care sunt grafice de funcţii polinomiale $y = f(x)$. Aceste *curbe planare neparametrice* $(x, f(x))$ pot fi scrise într-o formă parametrică:

$$r(s) = \begin{bmatrix} x(s) \\ y(s) \end{bmatrix} = \begin{bmatrix} s \\ f(s) \end{bmatrix} \tag{2.5.1}$$

Funcţia polinomială de grad n, $f:[0,1]\to R$ exprimată în baza Bernstein este de forma $f(s) = b_0 B_0^n(s) + \ldots + b_n B_n^n(s)$. Coeficienţii b_i sunt numere reale şi nu puncte, deci nu pot alcătui un poligon. Deoarece curbele neparametrice reprezintă un subset al curbelor parametrice se poate defini un poligon de control şi pentru ele. Luând în considerare proprietatea de precizie liniară a curbelor Bézier rezultă că $r(s) = (s, f(s))$ este parametrizarea în baza Bernstein a unei curbe Bézier ale cărei puncte de control sunt $\left(\frac{i}{n}, b_i\right)$:

$$r(s) = \sum_{i=0}^{n} \begin{bmatrix} i/n \\ b_i \end{bmatrix} B_i^n(s) \tag{2.5.2}$$

Funcţia $f(s)$ poartă numele de *funcţie Bézier* iar numerele b_i sunt *ordonatele Bézier* ale punctelor de control. Datorită proprietăţii de invarianţă la transformări afine de parametrii a curbelor Bézier, funcţiile Bézier pot fi definite pe orice interval $[a,b] \in R$. În acest caz valorile absciselor punctelor Bézier devin $a + i(b-a)/n, i = \overline{0,n}$. Formula se obţine prin

aplicarea inversei transformării afine $\varphi(u)=\frac{u-a}{b-a}$ absciselor inițiale. Astfel punctele poligonului de control ale graficului funcției Bézier sunt $\left(a+\frac{b-a}{n}i, b_i\right)$.

În mod similar, *funcțiile spline* $F:[u_0,u_L]\to R$ sunt definite de secvența de noduri $u_0,u_1,\ldots,u_L$ şi de L+3 numere reale $d_{-1},d_0,d_1,\ldots,d_L,d_{L+1}$ numite *ordonate de Boor* [17][19][32]. Funcția spline, fiind polinomială pe porțiuni, de-a lungul fiecărui subinterval $[u_{i-1},u_i]$ este definită ca o funcție Bézier. Pentru determinarea poligonului de control al graficului funcției spline se calculează abscisele ordonatelor de Boor. Datorită condiției de racord de clasă C^2 dintre două arce de curbă se defineşte relația dintre b_{3i-1} şi b_{3i-2}, d_i (relația 2.3.8). Abscisele punctelor b_{3i-1} şi b_{3i-2}, conform observațiilor de mai sus sunt $u_{i-1}+\frac{\Delta_{i-1}}{3}2$ respectiv $u_{i-1}+\frac{\Delta_{i-1}}{3}$. Aplicând relația 2.3.8 pentru abscise se obține valoarea abscisei pentru ordonatele de Boor:

$$\xi_i=\frac{1}{3}(u_{i-1}+u_i+u_{i+1}) \tag{2.5.3}$$

ξ_i poartă numele de *abscise Greville* [64]. Pentru punctele de Boor extreme, determinarea absciselor Greville se face în funcție de relațiile lor cu punctele Bézier ale arcelor de capăt. Astfel, deoarece punctul $d_{-1}=b_0$ abscisa sa este u_0 iar $d_0=b_1$ are abscisa $u_0+\frac{\Delta_0}{3}$. Punctul de Boor d_L este identic cu b_{3L-2}, iar d_{L+1} este identic cu b_{3L} şi deci absicele lor sunt $u_{L-1}+\frac{\Delta_{L-1}}{3}2$ respectiv u_L. Punctele (ξ_i,d_i) alcătuiesc *poligonul de control de Boor* al funcției B-spline.

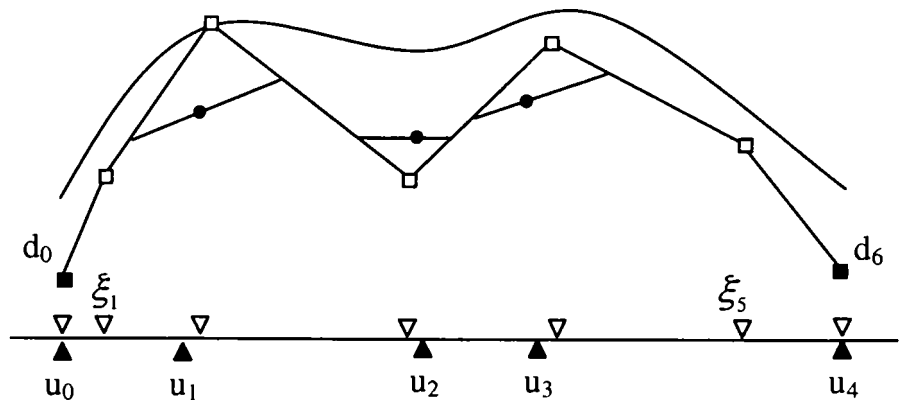

Fig. 2.5.1 Reprezentarea unei curbe spline cubice C^2 şi a poligonului său de control

Poligonul de Boor, fiind reuniunea segmentelor determinate de câte două puncte de Boor, poate fi definit ca şi graficul unei funcții polinomiale pe bucăți $P:[\xi_{-1},\xi_{L+1}]\to R$:

$$P_{|[\xi_i,\xi_{i+1}]}(u)=d_i+\frac{d_{i+1}-d_i}{\xi_{i+1}-\xi_i}(u-\xi_i) \tag{2.5.4}$$

Deoarece punctele $d_{i-1}, d_i, b_{3i-2}, b_{3i-1}$ sunt coliniare, ordonatele punctelor Bézier interioare se calculează prin evaluarea funcției P în abscisele acestor puncte. Poligonul rezultat prin includerea punctelor Bézier reprezintă o rafinare a poligonului de control de Boor.

Pornind de la aceste date se poate defini un algoritm pentru proiectarea unei curbe B-spline cubice definite pe domeniul $[u_0, u_L]$:

1. Se defineşte secvența de noduri u_i
2. Se determină abscisele Greville ξ_i
3. Se definesc numerele reale d_i pentru a obține un poligon ce trece prin punctele (ξ_i, d_i)
4. Evaluarea acestui poligon (o funcției liniare pe porțiuni P) pentru abscisele punctelor Bézier interioare. Acest lucru conduce la un poligon mai rafinat alcătuit din punctele Bézier interioare.
5. Evaluarea noului poligon în nodurile u_i, abscisele punctelor Bézier de joncțiune. Se obțin astfel punctele Bézier de joncțiune.
6. Se construieşte funcția B-spline cubică prin aplicarea algoritmului de Casteljau poligonului Bézier pe porțiuni determinat.

Într-o manieră similară se poate genera o curbă Bézier neparametrică cvadratică C^1 pe porțiuni. În continuare, se doreşte o generalizare a funcțiilor B-spline pentru a include grade arbitrare şi clase diferențiale arbitrare.

2.5.2 Inserția de noduri

Se defineşte un algoritm de rafinare a unei funcții liniare pe porțiuni. Mai târziu această funcție liniară pe porțiuni va fi interpretată ca un poligon B-spline, dar în acest punct se va considera doar un algoritm care produce o funcție liniară pe porțiuni din altă funcție. Algoritmul se numeşte *metoda Boehm de inserție a unui nod* [17][18][19][30][65].

Fie un număr n (va reprezenta gradul curbei B-spline) şi un număr L (care se referă la numărul segmentelor polinomiale ale curbei B-spline). Tot ca şi date inițiale, este definită o secvență crescătoare de noduri $u_0, \ldots, u_{L+2n-2}$.

În şir pot exista noduri identice. Dacă r noduri succesive coincid, $u_i = \ldots = u_{i+r-1}$, u_i este un nod de multiplicitate r. Dacă un nod nu coincide cu nici un alt nod, acesta este un nod simplu sau de multiplicitate 1.

O curbă B-spline va fi definită doar pe domeniul $[u_{n-1}, u_{L+n-1}]$. Aceste noduri sunt denumite *noduri ale domeniului*. L reprezintă numărul potențial de segmente de curbă. Dacă toate nodurile domeniului sunt simple, L constituie numărul de intervale de diviziune ale domeniului. Pentru fiecare nod al domeniului cu grad de multiplicitate numărul de intervale ale domeniului scade. Suma tuturor gradelor de multiplicitate a nodurilor domeniului este relaționată la L prin expresia $\sum_{i=n-1}^{L+n-1} r_i = L+1$.

Legătura dintre multiplicitatea nodurilor şi numărul de intervale ale domeniului este exprimată prin următoarele exemple:

Pentru $n=3, L=3$ este definită secvența de noduri:

$(u_0, \ldots, u_7) = (0,1,2,3,4,5,6,7)$

Toate nodurile sunt simple, deci numărul de intervale ale domeniului este 3.

Păstrând n şi L neschimbate se consideră secvenţa de noduri:

$(u_0,\ldots,u_7)=(0,1,2,3,3,5,6,7)$

În acest caz $u_3=u_4$ deci nodul 3 are gradul de multiplicitate 2, vor fi doar 2 intervale ale domeniului.

Multiplicitatea nodurilor care nu aparţin domeniului nu afectează numărul de intervale. Dacă se consideră :

$(u_0,\ldots,u_7)=(0,0,0,3,4,7,7,7)$

u_0 şi u_5 au gradul de multiplicitate 3. Dar ele apar, fiecare, doar o singură dată în secvenţa de noduri ale domeniului care este $(u_2,u_3,u_4,u_5)=(0,3,4,7)$. Vor exista 3 intervale ale domeniului.

Pornind de la secvenţa de noduri, şi generalizând relaţia 2.5.3. se determina $n+L$ abscise Greville:

$$\xi_i=\frac{1}{n}(u_i+\ldots+u_{i+n-1}),\quad i=\overline{0,L+n-1} \tag{2.5.5}$$

Abscisele Greville sunt date de media nodurilor. Numărul absciselor este egal cu numărul de perechi de n noduri succesive în secvenţa de noduri.

Fiind dat un set de ordonate de Boor d_i, se construieşte un poligon planar P alcătuit din punctele (ξ_i,d_i), $i=\overline{0,L+n-1}$ (relaţia 2.5.4). Acest poligon este o funcţie liniară pe porţiuni cu puncte de rupere la abscisele Greville:

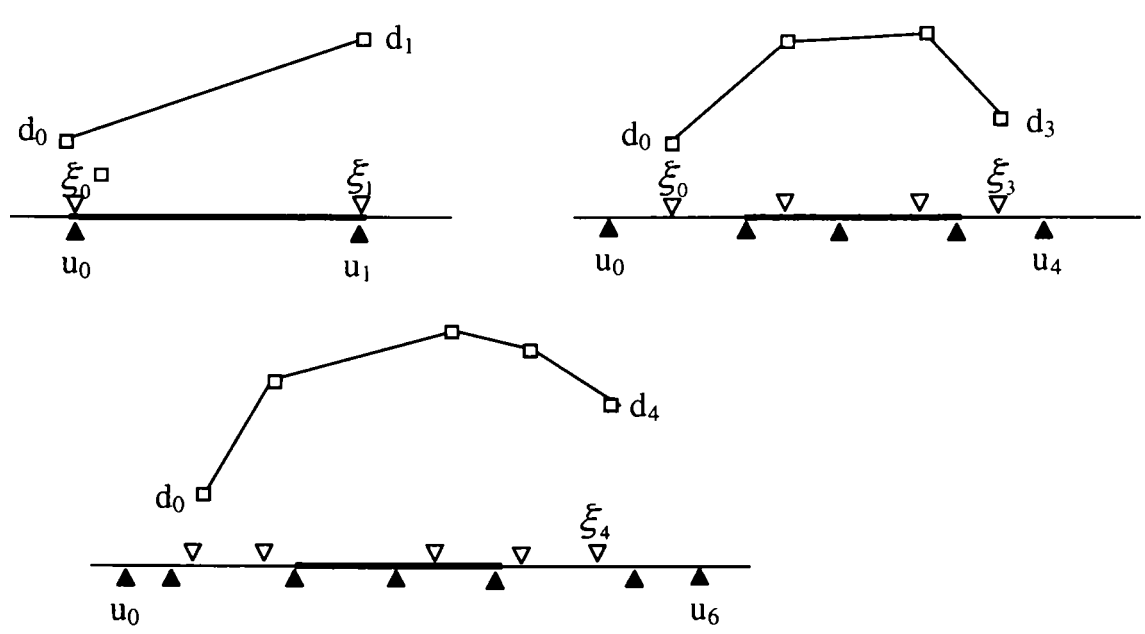

Fig. 2.5.2 Exemple de poligoane planare de Boor

Rafinarea acestui poligon se realizează prin inserţia unui nod $u\in[u_{n-1},u_{L+n-1}]$ în secvenţa de noduri. Noului şir de noduri i se asociază abscisele Greville notate ξ_i^u. Se evaluează P pentru calculul noilor ordonate de Boor $d_i^u=P(\xi_i^u)$. Poligonul rafinat va fi definit de punctele (ξ_i^u,d_i^u). Aceasta reprezintă descrierea formală a algoritmului Boehm de inserţie a unui nod. Algoritmul determină poligonul rafinat din datele iniţiale şi nodul inserat [19][65][74].

Figura 2.5.3. reprezintă un exemplu pentru procedura de inserţie a nodurilor pentru cazul cvadratic. Este posibil de a insera nodul u din nou - acesta va deveni un nod dublu sau un nod de multiplicitate 2 (va apare de 2 ori în secvenţa de noduri).

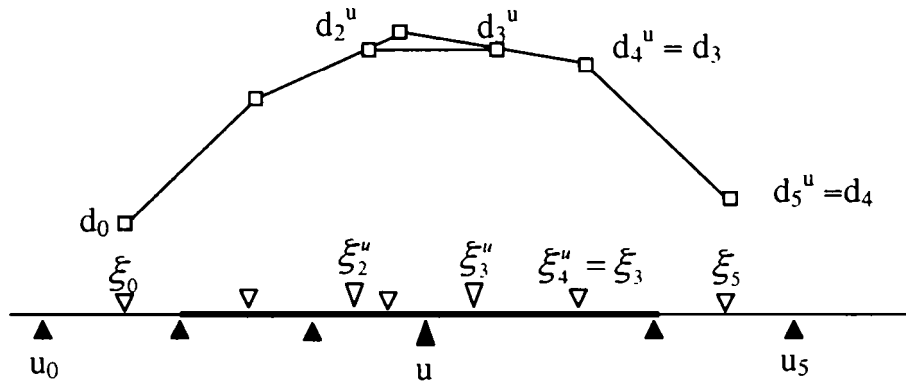

Fig. 2.5.3 Operaţia de inserţie a unui nod într-un poligon de Boor

Pot fi deduse câteva proprietăţi ale algoritmului de inserţie a unui nod:

- Poligonul P^u este obţinut din P printr-o interpolare liniară pe porţiuni.
- Ca şi consecinţă, inserţia nodurilor este un proces de diminuare a variaţiei. O linie dreaptă nu intersectează P^u de mai multe ori decât P.
- Inserţia nodurilor păstrează convexitatea. Dacă P este convex atunci şi P^u este convex.
- Inserţia nodurilor este un proces local. P diferă de P^u doar în vecinătatea lui u (definiţia exactă a vecinătăţii fiind o funcţie de grad n).

Se poate da o definiţie algoritmică a inserţiei unui nod. Aceasta este mai uşor de implementat în cadrul unui program. Descrierea formală de mai sus exprimă aceeaşi informaţie.

Algoritmul de inserţie a unui nod

Fiind dată o valoare reală $u \in [u_{n-1},\ldots,u_{L+n-1}]$ se doreşte determinarea poligonului P^u definit pentru o secvenţă rafinată de noduri care include pe u.

- este determinată cea mai mare valoare I cu proprietatea $u_I \le u \le u_{I+1}$. Dacă $u = u_I$ şi u_I are gradul de multiplicitate n algoritmul se opreşte deoarece reinserarea nodului nu conduce la rafinarea poligonului P, deoarece se obţine de două ori aceeaşi abscisă Greville. Altfel:
- Pentru $i = \overline{0, I-n+1}$ se setează
$$\xi_i^u = \xi_i$$
- Pentru $i = \overline{I-n+2, I+1}$ abscisele Greville sunt calculate cu relaţia
$$\xi_i^u = \frac{1}{n}\left(u_i + \ldots + u_{i+n-2}\right) + \frac{1}{n}u$$
- Pentru $i = \overline{I+2, L+n}$ se setează
$$\xi_i^u = \xi_{i-1}$$
- Se calculează ordonatele de Boor asociate noile abscise Greville
$$d_i^u = P\left(\xi_i^u\right), \quad i = \overline{0, L+n}$$
- Secvenţa de noduri este renumerotată pentru a include pe u ca u_{i+1}

- L este înlocuit cu $L+1$.

Evaluarea funcției $P_{[[\xi_{i-1},\xi_i]}$ în abscisa ξ_i^u conduce la relația:

$$d_i^u = \frac{\xi_i - \xi_i^u}{\xi_i - \xi_{i-1}} d_{i-1} + \frac{\xi_i^u - \xi_{i-1}}{\xi_i - \xi_{i-1}} d_i \qquad (2.5.6)$$

d_i^u este obținut prin interpolare liniară. Înlocuind în relație abscisele Greville rezultă:

$$d_i^u = \frac{u_{i+n-1} - u}{u_{i+n-1} - u_{i-1}} d_{i-1} + \frac{u - u_{i-1}}{u_{i+n-1} - u_{i-1}} d_i \qquad (2.5.7)$$

Formula reprezintă relația de calcul a ordonatelor de Boor în urma inserției nodului u.

Se poate demonstra că procesul de inserare a nodurilor este *consistent* [19][30][56]. Dacă se inserează două noduri rezultatul final trebuie să fie independent de ordinea în care acestea au fost inserate. Mai precis, se consideră u şi v nodurile ce urmează a fi inserate. Prin inserția lui u rezultă poligonul rafinat P^u, apoi inserția lui v conduce la P^{uv}. Dacă inserția se face în ordine inversă se obține $P^{vu} = P^{uv}$.

2.5.3 Algoritmul de Boor de generare a curbelor B-spline

Algoritmul de inserție a nodurilor se aplică funcțiilor liniare pe porțiuni. Metoda va fi utilizată la definirea curbelor B-spline. În general pentru gradul n, inserarea repetată a unui nod u nu va mai produce modificări ale poligonului după ce gradul de multiplicitate al nodului u a atins valoarea n (nu vor mai fi generate alte abscise Greville). Acest lucru va fi utilizat în definirea algoritmică a unor funcții speciale denumite curbe B-spline [33].

Algoritmul de Boor: pentru a evalua o curbă B-spline de gradul n (dată prin ordonatele sale de Boor şi prin secvența de noduri) pentru o valoare a parametrului u, se inserează u în secvența de noduri până când atinge gradul de multiplicitate n. Punctul din poligon corespunzător lui u aparține curbei B-spline.

Dacă un nod u_i este de multiplicitate n, atunci una din abscisele Greville coincide cu u_i: $\xi = \frac{1}{n}(u_i + ... + u_{i+n-1}) = u_i$. În consecință, poligonul are un punct (u_i, d_i), iar d_i este valoarea funcției B-spline în u_i. Figura 2.5.4 ilustrează acest lucru pentru o funcție spline de grad 3.

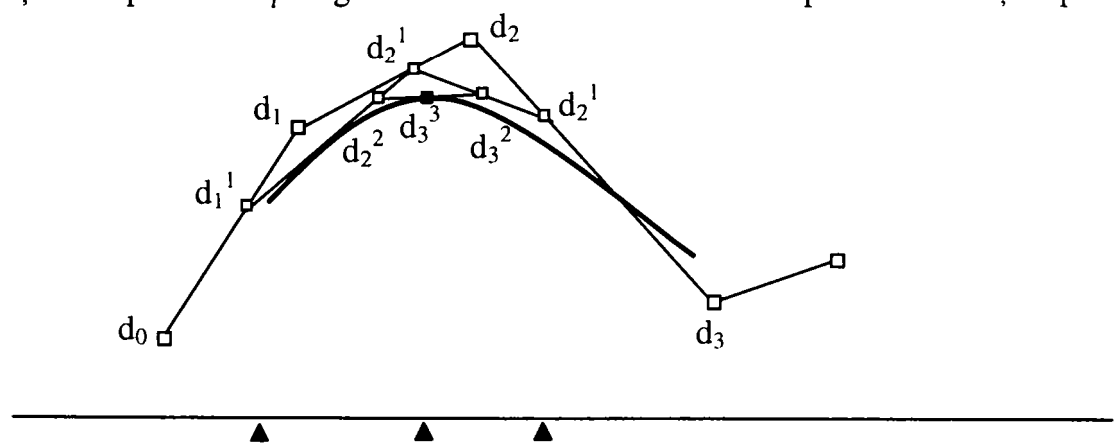

Fig. 2.5.4 Inserarea repetată a nodul u pentru o funcție de ordinul 3

Algoritmul de Boor necesită mai puține inserții dacă parametrul u este deja conținut în secvența de noduri. Dacă nodul u are grad de multiplicitate r, atunci sunt necesare doar $n-r$ inserții pentru ca nodul să atingă gradul de multiplicitate n.

Se poate da acum o definiție formală a algoritmului. Pentru aceasta, o curbă B-spline de grad n şi poligon de control P se notează B_nP iar valoarea curbei în u este $[B_nP](u)$. Curba va fi definită doar pentru valori ale lui u cuprinse între u_{n-1} şi u_{L+n-1}.

Algoritmul de Boor. Fie $u \in [u_I, u_{I+1}) \subset [u_{n-1}, u_{L+n-1}]$. Pentru $k = \overline{1, n-r}$ şi $i = \overline{I-n+k+1, I+1}$ se calculează:

$$d_i^k(u) = \frac{u_{i+n-k} - u}{u_{i+n-k} - u_{i-1}} d_{i-1}^{k-1}(u) + \frac{u - u_{i-1}}{u_{i+n-k} - u_{i-1}} d_i^{k-1}(u) \tag{2.5.8}$$

Valoarea curbei B-spline în nodul u este

$$[B_nP](u) = d_{I+1}^{n-r}(u) \tag{2.5.9}$$

r este gradul de multiplicitate a nodului u în cazul în care el aparține secvenței de noduri. Dacă u este inserat pentru prima dată $r = 0$. Ordonatele inițiale de Boor sunt notate $d_i^0(u) = d_i$. C. de Boor a publicat acest algoritm în 1972 [32][33]. Algoritmul este asemănător cu algoritmului lui de Casteljau. Figura 2.5.5 prezintă schematic punctele d_i care sunt implicate în relația de calcul.

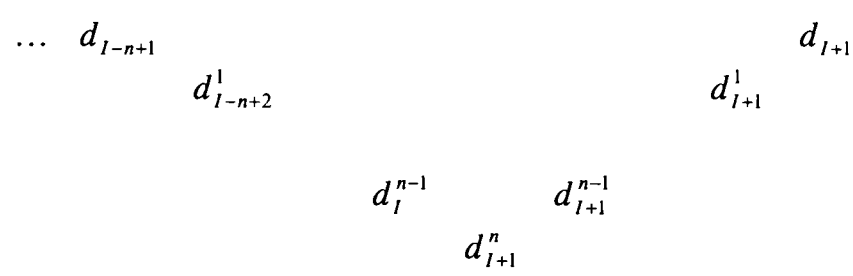

Fig. 2.5.5 Reprezentarea punctelor de Boor succesiv calculate în cadrul algoritmului

În descrierea algoritmului de Boor, nu s-a realizat renumerotarea secvenței de noduri şi a punctelor de control la fiecare nivel, deoarece este important doar rezultatul final $d_{I+1}^{n-r}(u)$. La fiecare nivel k, se generează un nou poligon de control care descrie aceeaşi curbă B-spline ca şi poligonul de control anterior. În particular pentru $k-1$, se obține algoritmul de inserție a nodurilor.

Figura 2.5.5. constituie un exemplu al metodei de calcul. În acest context, au fost construite mai multe poligoane care descriu aceeaşi curbă B-spline:

- $k = 1$: ordonatele de Boor $d_0, d_1^1, d_2^1, d_3^1, d_3, d_4$ corespund secvenței de noduri $u_0, u_1, u_2, u, u_3, u_4, u_5, u_6$;
- $k = 2$: ordonatele de Boor $d_0, d_1^1, d_2^2, d_3^2, d_3^1, d_3, d_4$ corespund secvenței de noduri $u_0, u_1, u_2, u, u, u_3, u_4, u_5, u_6$

- $k=3$: ordonatele de Boor $d_0, d_1^1, d_2^2, d_3^3, d_3^2, d_3^1, d_3, d_4$ corespund secvenței de noduri $u_0, u_1, u_2, u, u, u, u_3, u_4, u_5, u_6$

O curbă B-spline specială este definită pentru secvența de noduri:

$$0 = u_0 = u_1 = \ldots = u_{n-1} < u_n = u_{n+1} = \ldots = u_{2n-1} = 1.$$

Atât u_0 cât şi u_n au gradul de multiplicitate n. Abscisele Greville vor fi:

$$\xi = \frac{1}{n}\sum_{j=i}^{i+n-1} u_i = \frac{i}{n}; \quad i = \overline{0,n} \tag{2.5.10}$$

Întru-cât $n-1 \geq i-k \geq 0$ rezultă $u_{i+n-k} = 1, u_{i-1} = 0 \ \forall i,k$. Astfel relația 2.5.8 devine:

$$d_i^k(u) = (1-u)d_i^{k-1} + ud_i^{k-1}, \quad k = \overline{1,n} \tag{2.5.11}$$

Formula reprezintă algoritmul lui de Casteljau [34]. Schoenberg a observat primul acest lucru în 1967. Pornind de la acest caz special se deduc câteva concluzii importante.

Restricția la intervalul $[0,1]$ nu este esențială: toate construcțiile sunt invariante la transformări afine de parametrii.

Dacă două noduri adiacente în orice secvență de noduri au amândouă gradul de multiplicitate n, curba B-spline corespondentă este o curbă Bézier între cele 2 noduri. Poligonul de control B-spline este poligonul Bézier; abscisele Greville sunt în mod egal spațiate între cele două noduri.

După inserarea lui u până a atins gradul de multiplicitate n, poligonul inițial de Boor (ori poligonul Bézier, în acest caz) a fost transformat în două poligoane Bézier ce definesc aceeaşi curbă ca şi poligonul inițial. Astfel acest lucru reprezintă o altă demonstrație a faptului că algoritmul de Casteljau subdivide curbele Bézier [91].

Fiecare nod al secvenței de noduri a unei curbe B-spline poate fi inserat repetat până atinge gradul de multiplicitate n. Poligonul B-spline corespunzător acestei secvențe de noduri este poligonul Bézier pe porțiuni al curbei. Acest lucru demonstrează că B-spline sunt curbe polinomiale pe porțiuni pe domeniul $[u_{n-1}, u_{L+n-1}]$. Metoda de construcție a poligonului Bézier pe porțiuni din poligonul B-spline prin inserarea de noduri a fost dezvoltată de W. Boehm [19].

Funcția membră BsplineCoxDeBoor() generează o curbă B-spline folosind algoritmul de Boor, de inserție repetată a unui nod. Se construieşte mai întâi secvența de noduri a curbei, funcția GenKnotVectorBspline(). Pentru fiecare valoare a parametrului u din domeniul de existență al curbei se determină intervalul corespunzător lui, gradul său de multiplicitate, iar apoi prin aplicarea algoritmului de Boor se obține punctul de pe curba B-spline.

```
void CMeCurve::BSplineCoxDeBoor(void)
{
    double *knots, u, dim;
    short i, J, p, nr, nrCurvePoints, dense = 10;

    nr = controlPolygon.GetSize();
    // determinarea numărului de puncte ale curbei
    nrCurvePoints = (dense+1)*(nr-degree);
```

```
    // generarea vectorului nodurilor
    knots = new double[nr+degree+1];
    GenKnotVectorBspline(nr-1, degree+1, knots);
    dim = knots[nr+degree] - knots[0];
    // setarea dimensiunii tabloului de puncte al curbei
    curvePoints.SetSize(nrCurvePoints);
    // generarea punctelor curbei
    for(i = 0; i < nrCurvePoints; i++)
    {
        u = (double)(i)/(nrCurvePoints-1)*dim;
        J = CautareBinaraInterval(knots, nr+degree, nr-1, u);
        p = DetOrdinMultiplicitate(knots, nr+degree+1, u);
        if(p > degree)
            p = 0;
        CoxDeBoor(degree, nr, controlPolygon, knots, p, u, J, curvePoints[i]);
        myDoc->ComputePersp(curvePoints[i]);
    }
    delete []knots;
}
```

Funcția CautareBinaraInterval() returnează indicele din şirul nodurilor pentru care $u_c \in [u_J, u_{J+1}]$. Parametrii funcției sunt vectorul nodurilor, lungimea acestui vector, valoarea maximă pe care indicele poate să o obțină şi valoarea u din secvența de noduri pentru care se determină indicele.

```
short CautareBinaraInterval(double *u, short lung, short n, double uc)
{
    short j, k, J;

    if(uc < u[0] || uc > u[lung])
        return -1; // punctul este in exteriorul intervalului
    if(fabs(uc - u[lung]) < V_PPREC)
        return n;
    J = 0;
    j = lung;
    while( j - J > 1)
    {
        k = (J + j)/2;
        if(uc < u[k])
            j = k;
        else
            J = k;
    }
    return J;
}
```

Funcția DetOrdinMultiplicitate() determină ordinul de multiplicitate a parametrului u_c prin parcurgerea secvenței de noduri u, de lungime n şi identificarea numărului de apariții al acestuia în cadrul vectorului de noduri

```
short DetOrdinMultiplicitate(double *u, short n, double uc)
{
    short nr = 0;

    for(short i = 0; i < n; i++)
    {
```

```
        if(fabs(u[i] - uc) < V_PPREC)
        nr++;
    }
    return nr;
}
```

Funcția CoxDeBoor() implementează algoritmul de Boor de calcul al unui punct al curbei B-spline, prin inserarea repetată a nodului respectiv. Parametrii funcției sunt gradul curbei, numărul de puncte ale poligonului de Boor, poligonul de Boor, secvența de noduri a curbei, gradul de multiplicitate al nodului pentru care se calculează punctul, valoarea nodului, indicele intervalului corespunzător acestuia în secvența de noduri. Funcția returnează prin intermediul parametrului v, punctul de pe curba B-spline calculat pentru valoarea u_c din domeniul de existență al curbei.

```
void CoxDeBoor(short grad, short n, ControlPoints &d, double *u, short p, double uc,
                    short J, Vector &v)
{
    int r, j, h;
    double omega, uomega;
    ControlPoints da;

    da.SetSize(n);
    // copierea tabloului de Boor d in tabloul temporar da (in jurul intervalului de interes)
    for( j = J - grad; j <= J - p; j++)
        da[j] = d[j];
    // operația de inserție repetata a nodului uc si calculul noului poligon de Boor rezultat
    // prin inserție; inserția se încheie atunci când gradul de multiplicitate a lui uc este egal
    // cu gradul curbei
    h = grad - p;
    for(r = 1; r <= h; r++)
    {
        for(j = J - p; j >= J - grad + r; j--)
        {
            omega = (uc - u[j]) / (u[j+grad-r+1] - u[j]);
            uomega = 1.0 - omega;
            da[j] = da[j-1] * uomega + da[j] * omega;
        }
    }
    v = da[J-p];
}
```

2.5.4 Continuitatea curbelor B-spline

Curbele B-spline fiind polinomiale pe porțiuni, trebuie investigată continuitatea lor în noduri: de câte ori este diferențiabilă o curbă B-spline în nodul u. Problema se pune doar în punctele de rupere deoarece curba este infinit diferențiabilă în toate celelalte puncte.

Pentru a răspunde la această întrebare se porneşte de la cazul special descris anterior. Nodul u_n ce urmează să fie inserat repetat este definit ca având gradul de multiplicitate r. Secvența de noduri va fi:

$$0 = u_0 = u_1 = ... = u_{n-1} < u_n = u_{n+1} = ... = u_{n+r-1} < u_{n+r} = u_{n+r+1} = ... = u_{2n+r-1} = 1$$

Algoritmul de Boor va consta doar din n-r nivele. Calculul punctului curbei pentru nodul u_n se realizează aplicând relația 2.5.11 succesiv pentru $k = \overline{1, n-r}$. Acest lucru

reprezintă ultimele *n-r* nivele ale algoritmului de Casteljau. Astfel, două segmente de curbe polinomiale care se întâlnesc în u_n sunt cel puțin de *n-r* ori diferențiabile în acest punct.

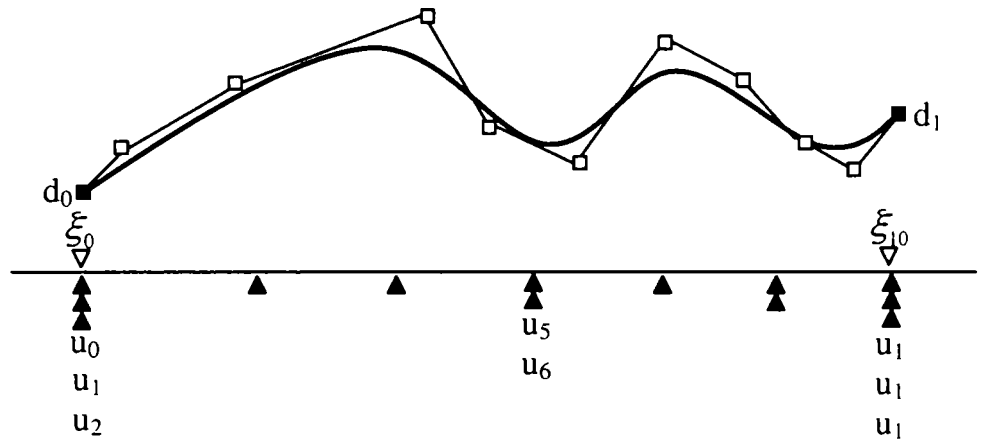

Fig. 2.5.6 Curbă B-spline cubică ce conține noduri multiple

Restricția la intervalul [0,1] nu este esențială. Dacă se dorește investigarea continuității unei curbe B-spline într-un nod, se forțează ca cei doi vecini ai săi să fie de multiplicitate *n* (acest lucru nu conduce la modificare curbei) iar apoi se aplică argumentele de mai sus.

În concluzie, o curbă B-spline este continuă C^{n-r} în nodurile cu grad de multiplicitate r. Curba este de *n-1* ori continuu diferențiabilă dacă toate nodurile sunt simple (de multiplicitate 1). Figura 2.5.6. prezintă o curbă B-spline cubică $n = 3$, $L = 8$ dată pentru o secvență de noduri care conține mai multe puncte multiple. Nodurile triple de la capete forțează d_0 şi d_{10} să fie pe curbă.

2.5.5 Baza unei curbe B-spline

Dezvoltarea anterioară constituie o definiție geometrică a curbelor B-spline de grad *n*, bazată pe principiul inserției de noduri. Definiția analitică a curbelor B-spline presupune construirea unei baze pentru spațiul funcțiilor polinomiale pe bucăți, definite pe un interval dat, divizat de un şir de noduri. Diferența dintre aceste funcții şi baza Bernstein folosită pentru definirea curbelor Bézier constă în faptul că domeniul de definiție este alcătuit dintr-o reuniune de subintervale iar funcțiile bazei vor fi nenule doar pe câteva subintervale adiacente, ceea ce le conferă caracterul de funcții locale.

Fie $u_0,\ldots,u_K$ o secvență de noduri şi N_i^n un set de polinoame pe porțiuni de grad *n* definite pentru aceste noduri. Fiecare funcție în acest set este de $n-r_i$ ori continuu diferențiabilă în nodul u_i. Toate aceste porțiuni polinomiale formează un spațiu liniar de dimensiunea:

$$\dim = (n+1) + \sum_{i=1}^{K-1} r_i \qquad (2.5.12)$$

Pentru demonstrația relației se consideră construirea unui element al spațiului liniar pe porțiuni. Numărul de constrângeri independente care pot fi impuse pentru un element arbitrar, sau numărul său de *grade de libertate*, este egal cu dimensiunea spațiului liniar considerat. Se începe printr-o completă specificare a primului segment polinomial, definit pe domeniul $[u_0,u_1]$. Acest lucru se poate face în $n+1$ moduri, $n+1$ fiind numărul

coeficienților care pot fi specificați pentru un poligon de grad n. Următorul segment polinomial definit pe $[u_1,u_2]$ trebuie să coincidă în poziție și $n-r_1$ derivate în u_1 cu segmentul anterior, ceea ce înseamnă că doar r_1 coeficienți trebuiesc aleși pentru cel de-al doilea segment. Continuând raționamentul se obține relația de mai sus.

Funcțiile polinomiale pe porțiuni de grad n $N_i^n:[u_0,u_K]\to R,\ i=\overline{0,K-n-1}$ sunt definite recursiv prin relațiile:

$$N_i^0(u)=\begin{cases}1 & daca\ u\in[u_i,u_{i+1})\\ 0 & in\ rest\end{cases}$$
$$N_i^n(u)=\frac{u-u_i}{u_{i+n}-u_i}N_i^{n-1}(u)+\frac{u_{i+n+1}-u}{u_{i+n+1}-u_{i+1}}N_{i+1}^{n-1}(u)\quad daca\ u_i<u_{i+n},u_i<u_{i+n+1} \tag{2.5.13}$$

Notând

$$\omega_i^n(u)=\begin{cases}\dfrac{u-u_i}{u_{i+n}-u_i} & daca\ u_i<u_{i+n}\\ 0 & in\ rest\end{cases}$$

formula devine:

$$N_i^n(u)=\omega_i^n(u)N_i^{n-1}(u)+\left(i-\omega_{i+1}^n(u)\right)N_{i+1}^{n-1}(u) \tag{2.5.14}$$

Aceste funcții polinomiale poartă numele de *funcții B-spline* [30][63][95]. Definiția recursivă are sens doar dacă fiecare nod din secvența de noduri are ordinul de multiplicitate cel mult n. Pentru u_i sau u_{i+1} de multiplicitate $n+1$, adică $u_i=u_{i+1}=\ldots=u_{i+n}$ sau $u_{i+1}=u_{i+2}=\ldots=u_{i+n+1}$, există convenția de a se considera cele două fracții din relația de recursivitate ca fiind egale cu 0.

Luând în considerare modul de definire a funcțiilor B-spline se pot deduce câteva proprietăți importante ale acestor funcții.

- Suportul funcției N_i^n este $[u_i,u_{i+n+1}]$.
- $N_i^n(u)\geq 0,\ \forall u\in[u_0,u_K]$
- $N_i^n(u)>0$ pentru $u\in(u_i,u_{i+n+1})$, $N_i^n(u_i)=0$ mai puțin cazul în care u_i are gradul de multiplicitate $n+1$.
- Dacă $K>2n\Leftrightarrow n<K-n$, atunci funcțiile N_i^n formează o partiție a unității pentru $\forall u\in[u_n,u_{K-n}]$.

Ultima proprietate este foarte importantă pentru cazul curbelor B-spline. Demonstrația se realizează prin inducție. Din definiția funcțiilor de grad 0 rezultă simplu că $\sum_{i=0}^{K-1}N_i^0(u)=1$. Se consideră că funcțiile B-spline de grad $n-1$ formează o partiție a unității: $\sum_{i=0}^{K-n-1}N_i^{n-1}(u)=1,\ u\in[u_{n-1},u_{K-n-1}]$. Fiind dat $u\in[u_n,u_{K-n})$ se definește j ca cea mai mare valoare pentru care u aparține domeniului $[u_j,u_{j+1})$. Dacă $u=u_j$ și $N_j^n(u_j)=1$, datorită

penultimei proprietăți, relația este satisfăcută. Pentru $u \in (u_j, u_{j+1})$:

$$\sum_{i=0}^{K-n-1} N_i^n(u) = \sum_{i=j-n}^{j} N_i^n(u)$$

$$= \sum_{i=j-n+1}^{j} \frac{u-u_i}{u_{i+n}-u_i} N_i^{n-1} + \sum_{i=j-n}^{j-1} \frac{u_{i+n+1}-u}{u_{i+n+1}-u_{i+1}} N_{i+1}^{n-1}$$

$$= \sum_{i=j-n+1}^{j} \left(\frac{u-u_i}{u_{i+n}-u_i} N_i^{n-1} + \frac{u_{i+n}-u}{u_{i+n}-u_i} N_i^{n-1} \right)$$

$$= \sum_{i=j-n+1}^{j} \left(\frac{u-u_i}{u_{i+n}-u_i} + \frac{u_{i+n}-u}{u_{i+n}-u_i} \right) N_i^{n-1}$$

$$= \sum_{i=j-n+1}^{j} N_i^{n-1}(u) = \sum_{i=0}^{K-n-1} N_i^{n-1}(u) = 1$$

Pornind de la această proprietate, dacă d_i $i = \overline{0, K-n-1}$ constituie o mulțime ordonată de puncte, relația $\sum_{i=0}^{K-n-1} d_i N_i^n(u)$ reprezintă o combinație convexă a punctelor d_i [91].

Având în vedere aceste rezultate, se poate demonstra că funcțiile polinomiale $(N_0^n, N_1^n, \ldots, N_{K-n-1}^n)$ formează o bază a spațiului $P_{n,u}$ al funcțiilor polinomiale pe bucăți de grad mai mic sau egal cu *n*. Acest lucru este reflectat şi în numele funcțiilor, B derivând de la bază. Orice funcție $f \in P_{n,u}$ se va exprima ca o combinație liniară a funcțiilor bazei:

$$f(u) = \sum_{j=0}^{K-n-1} d_j N_j^n(u),\ d_j \in R,\ u \in [u_n, u_{K-n}) \tag{2.5.15}$$

d_j se numesc ordonate de Boor ale funcției *f*. Deşi funcțiile B-spline sunt definite pe întreg intervalul $[u_0, u_K]$ ele formează o bază doar pe domeniul $[u_n, u_{K-n})$.

De asemenea se poate demonstra că funcția polinomială pe bucăți de grad *n* definită pentru secvența de noduri $(u_0, \ldots, u_K)$ şi ordonatele de Boor $(d_0, \ldots, d_{K-n-1})$ coincide cu funcția definită de aceleaşi date inițiale cu ajutorul algoritmului Cox–de Boor [28].

2.5.6 Curbe B-spline. Definiție analitică

$L+1$ puncte de control $(d_0, \ldots, d_L)$, secvența crescătoare de noduri $(u_0, \ldots, u_K)$ şi *n*, un număr natural cu proprietatea $L = K - n - 1$, $n \le L$, definesc o curbă B-spline de grad *n* $s: [u_n, u_{K-n}] \to E^3$:

$$s(u) = \sum_{i=0}^{K-n-1} d_i N_i^n(u) \tag{2.5.16}$$

Poligonul de control al curbei este poligonul de Boor [32][97]. Definiția analitică a curbelor B-spline este asemănătoare cu cea a curbelor Bézier definite în baza Bernstein,

existând însă diferențe: gradul curbei B-spline este setat de către utilizator, în timp ce gradul unei curbe Bézier depinde de numărul punctelor de control.

Forma unei curbe B-spline depinde de poligonul de control, secvența de noduri şi de gradul curbei. Astfel geometria curbei poate fi modificată prin schimbarea unuia din aceşti parametrii de control ai curbei.

Din definiția curbelor B-spline şi din caracteristicile funcțiilor B-spline se pot deduce o serie de proprietăți ale curbelor.

Curba B-spline este inclusă în înfăşurătoarea convexă a punctelor poligonului de Boor, iar arcul de curbă $\gamma_i = s([u_i, u_{i+1}))$, $u_i \neq u_{i+1}$ este inclus în înfăşurătoarea convexă a punctelor $d_{i-n},\ldots,d_{i-1},d_i$. Datorită proprietății de partiție a unității a funcțiilor B-spline, curba $s(u)$ este o combinație convexă a punctelor de control d_i, şi deci, orice punct al curbei se va afla în înfăşurătoarea convexă a punctelor de Boor. De asemenea, dacă $u \in [u_i, u_{i+1})$, conform primei proprietăți a funcțiilor B-spline $N_j^n(u) = 0$ pentru $i+1 \le j \le i-n-1$. Deci curba B-spline se poate scrie $s(u) = \sum_{j=i-n}^{i} d_j N_j^n(u)$ ceea ce conduce la faptul că arcul γ_i este inclus în înfăşurătoarea convexă a punctelor $d_{i-n},\ldots,d_{i-1},d_i$. Această caracteristică se numeşte *proprietate de convexitate tare* a unei curbe B-spline.

Curbele B-spline, asemenea curbelor Bézier, sunt invariante la transformări afine. Dacă $\phi: R^3 \to R^3$ este o transformare afină, prin aplicarea ei curbei $s(u)$ se obține o nouă curbă B-spline $\phi(s)$ de grad n ale cărei puncte de control sunt $\phi(d_0),\phi(d_1),\ldots,\phi(d_{K-n-1})$.

O proprietate importantă de care se bucură curbele B-spline este caracterul local al acestor curbe. Astfel, un punct al curbei B-spline corespunzător valorii u a parametrului nu depinde decât de cel mult $n+1$ puncte de control din vecinătatea sa. In sens invers, fiecare punct de control d_i influențează doar forma arcului corespunzător parametrilor u pentru care $N_i^n(u) \neq 0$, deci arcul definit pe domeniul $[u_i, u_{i+n+1})$.

Proprietățile de diferențiabilitate ale curbelor B-spline se deduc pornind de la derivatele funcțiilor B-spline. Acestea sunt date de relațiile:

$$\begin{aligned}
&\frac{d}{du}N_i^n(u) = \frac{n}{u_{i+n}-u_i}N_i^{n-1}(u) - \frac{n}{u_{i+n+1}-u_{i+1}}N_{i+1}^{n-1}(u), \quad pentru \ \ u \in (u_{i+1}, u_{i+n}) \\
&\frac{d}{du}N_i^n(u) = \frac{n}{u_{i+n}-u_i}N_i^{n-1}(u), \quad pentru \ \ u \le u_{i+1} \\
&\frac{d}{du}N_i^n(u) = \frac{n}{u_{i+n+1}-u_{i+1}}N_{i+1}^{n-1}(u), \quad pentru \ \ u \ge u_{i+n}
\end{aligned} \tag{2.5.17}$$

Demonstrația acestor formule se realizează prin inducție. Vectorul tangent într-un punct al unei curbe B-spline va fi:

$$\begin{aligned}
&\dot{s}(u) = \sum_{i=0}^{K-n-2} v_i N_i^{n-1} \\
&v_i = \begin{cases} k\dfrac{d_i - d_{i-1}}{u_{i+n}-u_i} & u_i < u_{i+n} \\ 0 & in\ caz\ contrar \end{cases}
\end{aligned} \tag{2.5.18}$$

Dacă $s(u)$ este o curbă B-spline de grad n și nodul u_{i+1} este multiplu de ordin n, $u_{i+1} = u_{i+2} = \ldots = u_{i+n} = \bar{u}$, $i \in \{0,\ldots,K-n-1\}$, conform algoritmului de inserție a unui nod punctul $d_i = s(\bar{u})$ aparține curbei și

- vectorul tangent în d_i la arcul $s([u_n, \bar{u}])$ este coliniar cu vectorul $\overrightarrow{d_{i-1}d_i}$ dacă $j > 0$, $d_{i-1} \neq d_i$.
- vectorul tangent în d_i la arcul $s([\bar{u}, u_{K-n}])$ este coliniar cu vectorul $\overrightarrow{d_i d_{i+1}}$ dacă $j < n$, $d_i \neq d_{i+1}$

Luând în considerare aceste afirmații, dacă curba B-spline interpolează extremitățile poligonului de control, deci nodurile de capăt au ordin de multiplicitate $n+1$, curba va fi tangentă în aceste extremități la primul și respectiv ultimul segment al poligonului de control.

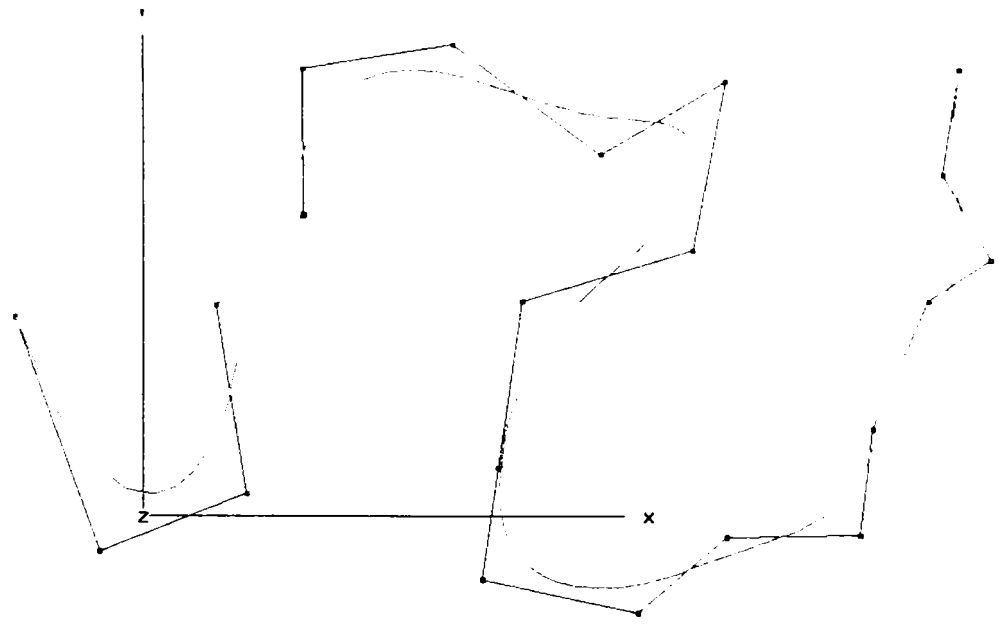

Fig. 2.5.7. Curbă B-spline de grad 5

Metoda CMeCurve::BSpline() construiește o curbă B-spline de grad arbitrar utilizând funcțiile B-spline $N_i^n(u)$. La fel ca și în cazul metodei de Boor de generare a unei curbe B-spline, se determină mai întâi secvența de noduri a curbei. Pentru fiecare segment al curbei al cărui $\Delta u > 0$ se calculează punctele curbei folosindu-se formula recursivă a funcțiilor B-spline.

```
void CMeCurve::BSpline(ControlPoints &controlPolygon)
{
    short i, j, ii, dense = 10, n;
    double *knots, u;
    n = controlPolygon.GetSize() - 2; // numărul de segmente de curba
    // numărul de puncte ale curbei depinde de numărul de segmente ale curbei pentru care
    // diferența dintre noduri este du != 0
    curvePoints.SetSize((dense+1)*(n-degree+2));
    // generarea vectorului nodurilor
    knots = new double[n+degree+1];
    GenKnotVectorBspline(n,degree,knots);
    // generarea punctelor curbei
    // se ia fiecare segment al curbei care prezintă o variație du != 0 si se generează punctele
```

```
    // curbei corespunzătoare acestuia
    j = 0;
    for (i=degree-1; i<n+degree-1; i++)
    {   if(knots[i+1]>knots[i])
        {   for(ii = 0; ii <= dense; ii++)
            {   // valoarea pentru care se calculează punctul de pe curba
                u=knots[i]+ii*(knots[i+1]-knots[i])/dense;
                // determinarea punctului curbei
                GenerateBsplinePoint(controlPolygon, degree,knots,u,i, curvePoints[j]);
                 if(myDoc)
                myDoc->ComputePersp(curvePoints[j]);
                j++;
            }
        }
    }
    delete []knots;
}

void GenerateBsplinePoint(ControlPoints &controlPoints, int degree, double *knots, double u,
    int i, Vector &bsplinePoint)
{
    int k,j, l;
    double t1,t2;
    Vector *points;
    points = new Vector[degree+1];
    for(j=i-degree+1, k=0; j<=i+1; j++, k++)
        points[k] = controlPoints[j];
    for(k=1; k <= degree; k++)
    {   for(j=i+1, l = degree; j >= i-degree+k+1; j--, l--)
        {   t1= (knots[j+degree-k] - u )/(knots[j+degree-k]-knots[j-1]);
            t2= 1.0-t1;
            points[l]= points[l-1] * t1 + points[l] * t2;
        }
    }
    bsplinePoint = points[degree];
    delete []points;
}
```

Parametrizarea curbei B-spline

În general, la fel ca şi în cazul celorlalte curbe, secvența de noduri nu este dată şi deci este necesară găsirea unei metode de parametrizare automată a curbelor B-spline. Parametrizarea trebuie să țină cont de faptul că aceste curbe sunt polinomiale pe bucăți iar domeniul lor suport este $[u_n, u_{K-n}]$.

O metodă de parametrizare care ține cont şi de gradul curbei generate este următoarea:

$$u_i = \begin{cases} 0 & i < n \\ i-n+1 & n \le i \le K-n \\ K-2n+2 & i > K-n \end{cases} \qquad (2.5.19)$$

Nodurile aflate la extremitățile secvenței de noduri au gradul de multiplicitate n, astfel încât curba B-spline va interpola capetele poligonului de control de Boor [63].

Cap 3 Principii de modelare geometrică prin suprafețe

3.1 Noțiuni generale

Un *model al suprafeței* unui obiect este o reprezentare matematică exactă a formei obiectului respectiv. Modelarea suprafețelor a fost dezvoltată rapid datorită inconvenienței modelării cu ajutorul *wireframe-urilor*. Prima este considerată o extensie a celei de-a doua metode. În general, un model wireframe poate fi extras dintr-un model de suprafață prin ştergerea sau eliminarea tuturor entităților suprafeței.

Modelele realizate cu ajutorul suprafețelor au avantaje considerabile asupra modelelor wireframe. Ele sunt mult mai complexe în conținut geometric şi mai puțin ambigue. De asemenea, modelarea suprafețelor oferă algoritmi pentru suprafețe şi algoritmi de umbrire pentru a aduce realism în geometria afişată. Modelele suprafețelor pot fi utilizate în calculul proprietăților de masă şi a volumului, în secționări de materiale, la modelarea elementelor finite precum şi la detecția intersecțiilor. Ultimele două domenii sunt atinse şi în cazul modelării laboratorului virtual de robotică, spațiu alcătuit din corpuri geometrice bine definite unde mişcarea roboților implică determinarea coliziunii dintre ei şi obiectele laboratorului virtual.

Modelarea suprafețelor prezintă însă şi anumite dezavantaje. Fiind în general un proces complex necesită mult timp procesor şi capacitate de memorare mare. De asemenea, în multe aplicații apar ambiguități la alegerea suprafețelor unui obiect care îi definesc volumul. Modelele de suprafață sunt uneori dificil de creat şi pot necesita manipulări inutile ale unor entități wireframe.

Descrierea matematică a suprafeței trebuie să cuprindă cât mai multe din proprietățile geometrice ale acesteia. O modalitate complet ineficientă de descriere este utilizarea unei liste suficient de mare de puncte ale suprafeței. Această abordare este costisitoare şi nu poate fi folosită pentru a determina toate caracteristicile suprafeței. Modelele matematice ale suprafețelor trebuie să fie uşor diferențiabile pentru obținerea tangentelor, normalelor şi curburilor suprafeței.

Au fost dezvoltate mai multe tehnici de modelare a suprafețelor pentru a se putea surprinde marea diversitate de forme din lumea reală. Alegerea metodei de reprezentare depinde de caracteristicile specifice ale aplicației.

3.1.1 Reprezentarea suprafețelor

Reprezentarea suprafețelor este considerată, prin multe aspecte, ca o extensie a reprezentării curbelor. Formele parametrice şi neparametrice ale curbelor pot fi extinse la suprafețe. Tratarea suprafețelor prin intermediul graficii pe calculator necesită dezvoltarea algoritmilor şi a relațiilor potrivite atât pentru calcul cât şi pentru programare.

Suprafețele pot fi descrise matematic în spațiul tridimensional prin relații parametrice sau neparametrice [26][55][57][59]. Există mai multe metode pentru a asocia suprafețe neparametrice unei mulțimi de puncte considerate ca date de intrare. Aceste metode se împart în două categorii. În prima, o relație este astfel realizată încât să treacă

prin toate punctele, în timp ce în a doua, punctele sunt folosite pentru dezvoltarea unei serii de petice de suprafețe care sunt apoi conectate între ele cu păstrarea continuității de poziție şi a derivabilității. În ambele categorii, formula suprafeței sau a peticelor de suprafețe este dată prin:

$$P = [x \quad y \quad z]^T = [x \quad y \quad f(x,y)]^T \tag{3.1.1}$$

P este vectorul de poziție al unui punct de pe suprafață. Forma neparametrică a funcției $f(x,y)$ utilizată pentru a interpola toate punctele este cea polinomială:

$$z = f(x,y) = \sum_{m=0}^{p}\sum_{n=0}^{q} a_{mn} x^m y^n \tag{3.1.2}$$

Suprafața este descrisă printr-o rețea xy de puncte de dimensiunea $(p+1)\times(q+1)$. Funcțiile polinomiale constituie o alegere potrivită pentru suprafețele neparametrice. Funcțiile de ordin ridicat nu sunt corespunzătoare din punct de vedere al proiectării suprafețelor datorită numărului mare de coeficienți care pot să producă dificultăți în controlul suprafeței rezultate sau pot introduce oscilații nedorite în suprafață. Pentru cele mai multe aplicații practice ale suprafețelor, funcțiile polinomiale cubice sunt suficiente.

Reprezentarea parametrică a unei suprafețe înseamnă o funcție continuă $P(u,v)$ de două variabile, sau parametrii u şi v:

$$P(u,v) = [x \quad y \quad z]^T = [x(u,v) \quad y(u,v) \quad z(u,v)]^T, \quad u_{\min} \le u \le u_{\max}, v_{\min} \le v \le v_{\max} \tag{3.1.3}$$

Relația determină coordonatele unui punct de pe suprafață, prin intermediul componentelor vectorului lui de poziție. Acest vector pune în corespondență unică spațiul parametric E^2 (în u şi v) şi spațiul cartezian E^3 (în x, y, z). Domeniul de existență al parametrilor este $[u_{\min}, u_{\max}]$ respectiv $[v_{\min}, v_{\max}]$. Pentru cele mai multe suprafețe, aceste intervale sunt [0,1].

Relațiile (3.1.1) şi (3.1.3) sugerează că un spațiu tridimensional general poate fi modelat prin împărțirea lui într-un ansamblu de *petice topologice* [26]. Un petic este considerat un element matematic de bază pentru a modela suprafețe compuse. Unele suprafețe pot să fie constituite doar dintr-un singur petic, în timp ce altele pot să fie constituite din mai multe petice conectate între ele. Topologia unui petic poate fi triunghiulară sau dreptunghiulară. Peticele triunghiulare prezintă mai multă flexibilitate în modelarea suprafețelor, deoarece nu necesită tablouri dreptunghiulare ordonate de puncte pentru crearea suprafeței, pe când cele dreptunghiulare necesită.

Se pot defini două categorii de suprafețe: *analitice* şi *sintetice*. Suprafețele analitice sunt bazate pe entități wireframe şi cuprind suprafețele plane, suprafețele riglate, suprafețe de revoluție şi cilindrii generalizați. Suprafețele sintetice sunt formate dintr-o mulțime dată de puncte sau curbe, şi includ petice Bézier, B-spline şi Coons. Există câteva metode pentru generarea suprafețelor sintetice, cum ar fi metoda produsului tensorial, metoda rațională, şi metoda de amestecare (blending). *Metoda rațională* descrie o suprafață rațională care este o extensie a curbelor raționale. *Metoda de amestecare* aproximează o suprafață prin bucăți de suprafețe [13][57].

Metoda produsului tensorial este cea mai folosită metodă în modelarea solidelor.

Utilizarea metodei se datorează naturii ei separabile simple, care implică doar produse de funcții de bază univariante, de regulă polinoame. Ea nu introduce complicații conceptuale noi datorită dimensiunii mult mai mari a unei suprafețe față de o curbă. Proprietățile suprafețelor produsului tensorial pot fi uşor deduse din proprietățile curbelor. Formularea produsului tensorial este o punere în corespondență a unui domeniu dreptunghiular descris de valorile u şi v; de exemplu $[0,1]\times[0,1]$ cu spațiul real. Suprafețele produs tensorial se potrivesc natural peste peticele dreptunghiulare. În plus, ele au o orientare explicită unică (triangulația unei suprafețe nu este unică) şi direcții parametrice speciale sau coordonate asociate fiecărei variabile parametrice independente [94].

Pentru un petic dreptunghiular este definit un set de condiții de frontieră alcătuit din 16 vectori şi 4 curbe de frontieră. Vectorii sunt : patru vectori de poziție pentru cele 4 colțuri ale peticului $P(0,0), P(0,1), P(1,0), P(1,1)$, opt vectori tangenți (doi pentru fiecare colț) şi patru vectori de twist la punctele de colț. Cele patru curbe de frontieră sunt obținute prin păstrarea unei variabile parametrice fixată pentru fiecare valoare limită $u=0, u=1, v=0, v=1$.

Pentru generarea curbelor pe un petic de suprafață, se menține unul din parametrii suprafeței constant, celălalt parametru parcurgând întreg domeniul său de definiție. Rezultă în acest mod o rețea de două familii de curbe parametrice (fiecare corespunzătoare unei variabile) numite *curbe isoparametrice*. Doar o singură curbă din fiecare familie trece prin orice punct $P(u,v)$ de pe suprafață. Sensul pozitiv al oricărei astfel de curbe este sensul de creştere al parametrului liber. Reprezentarea grafică a unui petic de suprafață se realizează prin intermediul acestei rețele de curbe isoparametrice. Curbele din fiecare familie sunt de obicei echidistante în intervalul de lucru permis al variabilei parametrice corespondente.

3.1.2 Caracteristicile geometrice ale unei suprafețe

Analiza geometrică a suprafețelor constituie un factor important în folosirea suprafețelor în cadrul unor aplicații. Cunoaşterea normalelor vectorilor la suprafață oferă direcția corectă de apropiere şi îndepărtare de aceasta, lucru esențial în procesul de mişcare al roboților pentru stabilirea poziției acestora în raport cu celelalte obiecte ale spațiului virtual. Geometria diferențială joacă un rol central în analiza suprafețelor.

Conceptul de vector tangent introdus la curbe poate fi extins şi la suprafețe. *Vectorul tangent* al oricărui punct $P(u,v)$ de pe suprafață este obținut prin păstrarea unui parametru constant şi diferențiind în raport cu celălalt. În fiecare punct vor exista doi vectori tangenți, pentru fiecare curbă isoparametrică care trece prin acel punct. Aceşti vectori sunt dați de relațiile:

$$P_u(u,v)=\frac{\delta P}{\delta u}=\frac{\delta x}{\delta u}\bar{i}+\frac{\delta y}{\delta u}\bar{j}+\frac{\delta z}{\delta u}\bar{k}, \qquad v=const.$$
$$P_v(u,v)=\frac{\delta P}{\delta v}=\frac{\delta x}{\delta v}\bar{i}+\frac{\delta y}{\delta v}\bar{j}+\frac{\delta z}{\delta v}\bar{k}, \qquad u=const. \tag{3.1.4}$$

Vectorii tangenți sunt necesari la determinarea condițiilor de frontieră pentru conectarea peticelor, la fel cum definesc şi mişcarea de-a lungul suprafeței [91][103].

Vectorul de twist pentru un punct de pe suprafață se defineşte ca fiind măsurătoarea twistului suprafeței în acel punct. El este rata de modificare a vectorului tangent P_u în

raport cu v, sau a vectorului P_v în raport cu u, sau este vectorul derivator mixt al punctului. Dacă u şi v variază cu Δu şi respectiv Δv, rata incrementală de modificare a vectorilor tangenți P_u şi P_v este $\frac{\Delta P_u}{\Delta v}$ respectiv $\frac{\Delta P_v}{\Delta u}$ iar rata infinitezimala de schimbare este dată de următoarele limite:

$$\underset{\Delta v\to 0}{Limit}\frac{\Delta P_u}{\Delta v}=\frac{\delta P_u}{\delta v}=\frac{\delta^2 P}{\delta u\delta v}=P_{uv} \qquad \underset{\Delta u\to 0}{Limit}\frac{\Delta P_v}{\Delta u}=\frac{\delta P_v}{\delta u}=\frac{\delta^2 P}{\delta u\delta v}=P_{uv} \tag{3.1.5}$$

Interpretarea geometrică a vectorului de twist se realizează prin construirea a două triunghiuri formate din P_u şi $P_u(u,v+\Delta v)$ respectiv P_v şi $P_v(u+\Delta u,v)$ în punctul $P(u,v)$ al suprafeței [63].

Vectorul de twist depinde atât de caracteristicile geometrice ale suprafeței cât şi de parametrizare.

Normala unei suprafețe este o altă proprietate analitică importantă. Ea este folosită pentru determinarea direcției de apropiere sau depărtare de suprafață, calcule de volume, umbrirea unui model de suprafață. Normala suprafeței într-un punct este un vector perpendicular pe ambii vectori tangenți ai punctului:

$$\mathrm{N}(u,v)=\frac{\delta P}{\delta u}\times\frac{\delta P}{\delta v}=P_u\times P_v \tag{3.1.6}$$

Versorul acestui vector este:

$$\bar{n}=\frac{\mathrm{N}}{|\mathrm{N}|}=\frac{\mathrm{P_u}\times\mathrm{P_v}}{|\mathrm{P_u}\times\mathrm{P_v}|} \tag{3.1.7}$$

Ordinea vectorilor din cadrul produsului vectorial poate fi inversată, iar produsul rezultat defineşte tot un vector normal. Sensul lui N, sau n, poate fi ales în mod convenabil aplicației. În practică, sensul lui n se alege astfel încât să indice exteriorul suprafeței [15][16].

Normala la o suprafață este zero dacă $P_u\times P_v=0$. Acest lucru se întâmplă în cazul unei coame, sau în cazul intersecției suprafeței cu ea însăşi. Poate de asemenea să apară în cazul în care cele două derivate P_u şi P_v sunt paralele, sau una dintre ele are modulul 0. Cazurile din urmă corespund unei parametrizări greşite, care însă poate fi refăcută.

3.1.3 Implementarea software a suprafețelor

Proiectarea software a suprafețelor a fost abordată în mod diferit față de cea a curbelor datorită trăsăturilor specifice. În timp ce pentru curbe implementarea optimă a constituit-o utilizarea unei singure clase generale, la suprafețe, datorită diversității modurilor de construcție, a fost necesară realizarea unei structuri ierarhice de clase, fiecărui tip de suprafață fiindu-i asociată o clasă proprie. Elementele comune tuturor acestor suprafețe au fost grupate la nivelul unei clase de bază care reprezintă punctul de plecare pentru toate celelalte clase.

Reprezentarea tuturor suprafețelor curbe se realizează printr-o rețea de curbe isoparametrice longitudinale şi transversale. Acestea au structură mult mai simplă decât

curbele independente, fiecare curbă nefiind definită prin propriul său poligon de control şi nici printr-un mecanism specific de generare. Clasa corespunzătoare curbelor isoparametrice este CMeSurfaceCurve. Ea încapsulează lista de puncte a curbei şi funcțiile de manipulare a acesteia.

```
class CMeSurfaceCurve : public CObject
{
   // variabilele clasei
public:
   CArray<CVertex,CVertex> curve;  //  tablou cu punctele curbei isoparametrice a suprafeței
   // metodele clasei
   CMeSurfaceCurve(){};
   short   Select(CMeView *view, double persp[2]);
   void    Draw(CMeView *view, CDC* pDC);
   void    SetBBox(ElementBBox &bbox);
   void    operator=(CMeSurfaceCurve &surfaceCurve);
   bool    IntersectRobot(CMeOpenGLRobot *pRobot);
};
```

Metoda IntersectRobot() procesează operația de verificare a coliziunii dintre curba suprafeței şi robot.

Pornind de la această structură se defineşte clasa principală pentru suprafețe curbe CMeStandardSurface. Curbele longitudinale şi transversale ale suprafeței sunt grupate într-un tablou bidimensional de liste de obiecte CMeSurfaceCurve. Acest tablou, proprietățile de desenare ale unei suprafețe, împreună cu funcțiile de manipulare specifice, constituie elementele comune pentru toate tipurile de suprafețe.

```
class CMeStandardSurface : public CMeElement
{
protected:
    // variabilele clasei
    SurfSettings  surfaceSettings;
    CTypedPtrList<CObList,CMeSurfaceCurve*> surfaceCurves[2]; //  lista de curbe isoparametrice
    // metodele clasei
    CMeStandardSurface();
    DECLARE_SERIAL(CMeStandardSurface)
public:
    CMeStandardSurface(CMeDoc *pDoc);
    CMeStandardSurface(Settings &surfSettings, CMeDoc *pDoc);
    // funcțiile virtuale suprascrise pentru clasa CMeStandardSurface
    virtual void       DrawElement(CMeView *view, CDC *pDC, short drawControlPoints);
    virtual void       CalculatePerspElement(CMeView *view);
    virtual void       CreatePerspElement(CMeView *view);
    virtual void       DeletePerspElement(CMeView *view);
    virtual void       DeleteContentElement(short mode=0);
    virtual short      SelectElement(CMeView *view, double persp[2]);
    virtual void       SetElementBBox(void);
    virtual CMeElement *CreateCopyElement(void);
    void SetElementSettings(unsigned int lineWidth, unsigned long color, char *texture , short mode);
    void               SetSettings(Settings &settings){surfaceSettings = settings;}
    void               operator=(CMeStandardSurface &surface);
    bool               IntersectRobot(CMeOpenGLRobot *pRobot);
};
```

Toate clasele care modelează întreaga tipologie de suprafețe sunt derivate din această clasă de bază. Fiecare din aceste clase situate ierarhic pe o treaptă superioară încapsulează mecanismul propriu de construcție a suprafețelor.

Pentru suprafețele sintetice a fost definită o clasă unică denumită CMeSurface deoarece atât datele inițiale cât şi modul de reprezentare al acestor suprafețe este identic. Această clasă are în fapt un rol structural căci întreg mecanism de generare al suprafețelor este implementat la nivelul peticelor de suprafață.

Elementul de bază în proiectarea acestor suprafețe este harta de control care este o extensie a poligonului de control corespunzător curbelor. Pentru aceasta a fost definită clasa CMeControlMap care conține tabloul de liste de puncte de control. Fiecare listă reprezintă o linie de puncte iar toate punctele cu acelaşi index din aceste listă formează o coloană din harta de control.

```
class CMeControlMap : public CObject
{
 // variabilele clasei
public:
 short   nr_row, nr_col;
    CArray< ControlPoints, ControlPoints> controlPoints;  //  harta punctelor de control
    short   selPoint[2]; // indecşii punctului selectat de pe harta de control

    // metodele clasei
public:
    CMeControlMap();
    DECLARE_SERIAL(CMeControlMap)
    CMeControlMap(short nr_r,  short nr_c);
    void     Draw(CMeView *view, CDC* pDC, SurfaceSettings &surfSettings);
    short    Select(CMeView *view, double persp[2]);
    void     CopyLineCol(ControlPoints &l_c, short index, RowCol r_c);
    void     SetBBox(ElementBBox &bbox);
    void     RacordPatch(CMeControlMap &controlMap, short mode);
    void     Resize(short nr_r, short nr_c); // redimensionarea rețelei de control
    void     Serialize(CArchive& ar);
    void     operator=(CMeControlMap &controlMap);
};
```

Deoarece orice suprafață sintetică este concepută ca o colecție de petice, a fost necesară definirea clasei CMeSurfacePatch proprie acestei entități. Fiecare petic este în sine o suprafață curbă, care conține harta sa de control şi curbele isoparametrice ale peticului. Funcțiile de construcție al diverselor tipuri de suprafețe sintetice sunt incluse în această clasă.

```
class CMeSurfacePatch : public CMeStandardSurface
{
    // variabilele clasei
private:
    CMeSurface                  *parentSurface;   //  suprafața părinte a peticului
    SurfaceType                 surfaceType;      //  tipul peticului
    CMeControlMap                 controlMap;     //  mapa de control a peticului
    struct Surface
    {   char nm[30];
        SurfaceFunction execute;  // pointer la funcția de construcție a peticului
    };
    static Surface       surface[4];

    // metodele clasei
```

```
protected:
    CMeSurfacePatch();
    DECLARE_SERIAL(CMeSurfacePatch)

public:
    CMeSurfacePatch(CMeDoc *pDoc, CMeSurface *surface, short nrRow, short nrCol);
    void       SetSurface(SurfaceType type){surfaceType = type;};
    short      SetControlMapPoint(short row, short col, Vector &v);
    short      DetControlMapPoint(short row, short col, Vector &v);
    // funcțiile virtual suprascrise pentru clasa CMeSurfacePatch
    void       DrawElement(CMeView *view, CDC *pDC, short drawControlPoints,
                           SurfaceSettings &surfaceSettings);
    void       CalculatePerspElement(CMeView *view);
    void       CreatePerspElement(CMeView *view);
    void       DeletePerspElement(CMeView *view);
    void       DeleteContentElement(short mode=0);
    short      SelectElementControlPoints(CMeView *view, double persp[2]);
    short      SelectElement(CMeView *view, double persp[2]);
    void       TranslateElement(Vector &depl);
    void       RotateElement(Vector &axa_rotatie, double angle);
    CVertex*   DetSelectedVertex(void);
    short      ConstructElement(short mode = 0)
               {(this->*this->surface[surfaceType].execute)();return 0;};
    void       SetElementBBox(void);
    CMeElement *CreateCopyElement(void);
    void       Serialize(CArchive& ar);
    void       GenerateOpenGLRepresentation(CMeOpenGLDoc *pDoc, HGLRC glRC);
    void       RacordPatch(short row, short &col);
    void       SetParent(CMeSurface *surface){parentSurface = surface;}
    void       Resize(short nr_r, short nr_c){controlMap.Resize(nr_r, nr_c);}
    void       GetDimension(short &nr_r, short &nr_c);
    void       operator=(CMeSurfacePatch &surfacePatch);

private:
    void       NoSurface(void){};
    void       BezierSurface(void);
    void       DecasteljauCurve(ControlPoints &points, CMeSurfaceCurve **surf);
    void       DecasteljauPoint(ControlPoints &points, double t, Vector &v);
    void       BSplineSurface(void);
    void       BSplineSurface(CMeControlMap &controlMap);
    void       BSplineSurfaceInterpolation(void);
    void       BSplineCurve(ControlPoints &controlPoints, short degree, double *knots,
                            CMeSurfaceCurve **surf);
};
```

Tot la nivelul acestei clase a fost implementat şi mecanismul de conexiune al peticelor pentru păstrarea regulilor de continuitate, element esențial în proiectarea corectă a oricărei suprafețe. Funcția RacordPatch() determină automat o parte din punctele mapei de control al noului petic în acord cu clasa de continuitate considerată.

Având definite aceste structuri de date, clasa principală a suprafețelor sintetice este proiectată ca o matrice bidimensională de petice de suprafață. Implementarea software a matricei a fost realizată prin intermediul unei liste de liste de petice. Funcția membră AddPatch() realizează operația de ataşare a unui petic la suprafață în poziția dorită, petic generat respectând regulile de racord. De asemenea funcția IntersectRobot() stabileşte dacă suprafața se afla în contact cu suprafață.

3.2 Reprezentarea parametrică a suprafețelor analitice

Secțiunea descrie elementele fundamentale pentru formulele parametrice ale suprafețelor analitice întâlnite în proiectarea şi modelarea suprafețelor. Se stabilesc în acest mod posibilitățile şi limitările de modelare ale fiecărui tip de suprafață. Suprafețele analitice cuprind suprafețe plane, suprafețe riglate, suprafețe de revoluție, cilindrii generalizați [74][94].

Termenii *suprafață* şi *petic*, în această secțiune, sunt folosiți pentru a desemna aceeaşi entitate.

3.2.1 Suprafața plană

Formula parametrică a unui plan poate lua diferite forme în funcție de datele disponibile. Cel mai simplu mod de definire a unui plan este prin trei puncte P_0, P_1, P_2 ce aparțin planului. Se consideră că punctul P_0 corespunde valorilor parametrice $u = 0, v = 0$, iar vectorii P_0P_1 şi P_0P_2 definesc direcțiile u respectiv v. Domeniul suport pentru parametrii este $[0,1]$. Pornind de la aceste date, formula parametrică a planului este:

$$P = P_0 + u \cdot (P_1 - P_0) + v \cdot (P_2 - P_0), \qquad u, v \in [0,1] \tag{3.2.1}$$

Vectorii tangenți în punctul P la plan sunt:

$$P_u(u,v) = P_1 - P_0 \qquad P_v(u,v) = P_2 - P_0 \qquad u, v \in [0,1] \tag{3.2.2}$$

iar versorul normal la suprafață este:

$$\bar{n}(u,v) = \frac{(P_1 - P_0) \times (P_2 - P_0)}{\left|(P_1 - P_0) \times (P_2 - P_0)\right|}, \quad u, v \in [0,1] \tag{3.2.3}$$

care este constant pentru orice punct din plan. Curbura planului este de asemenea egală cu zero deoarece toți coeficienții fundamentali secunzi pentru acest plan sunt nuli.

Un alt mod de definire a unei suprafețe plane este printr-un punct P_0 şi două direcții date prin versorii r şi s. Similar cazului anterior, formula planului devine:

$$P(u,v) = P_0 + uL_u\bar{r} + vL_v\bar{s}, \qquad u, v \in [0,1] \tag{3.2.4}$$

Relația presupune un plan de dimensiuni L_u şi L_v care pot fi egale cu unitatea.

Cazurile anterioare pot fi combinate pentru a furniza formula unei suprafețe plane care trece prin două puncte P_0 , P_1 şi este paralelă cu versorul r:

$$P(u,v) = P_0 + u(P_1 - P_0) + vL_v\bar{r}, \qquad u, v \in [0,1] \tag{3.2.5}$$

O modalitate obişnuită de construcție a unui plan este printr-un punct P_0 şi normala sa n:

$$(P - P_0) \cdot \bar{n} = 0 \tag{3.2.6}$$

Relația este o formă neparametrică a suprafeței plane [2][74]. Cu ajutorul acestei relații se pot genera însă două puncte ale suprafeței, care pot fi folosite împreună cu P_0 în prima formulă parametrică. Planele care sunt perpendiculare pe axele sistemului de coordonate curent reprezintă un caz special al relației anterioare. Astfel, în cazul unui plan care este perpendicular pe axa X, normala n este (1,0,0) iar ecuația planului $x = x_0$.

Elementele de construcție a unei suprafețe planare au fost utilizate la definirea unui obiect de tip poligon. Clasa CMePolygon corespunzătoare acestui element geometric conține, ca variabile membre, normala si distanța planului în care se află conturul poligonal. Această clasă constituie o excepție față de celelalte clase de suprafețe prin faptul că nu este derivată din clasa principală descrisă în secțiunea 3.1.3. Contururile poligonului, atât cel exterior cât şi cele interioare, sunt cuprinse într-un tablou de liste de puncte al clasei reprezentat de variabila polygon.

Poziția relativă a unei entități geometrice, curbă sau suprafață, față de poligon poate fi determinată prin aplicarea ecuației planului punctelor entității geometrice. Astfel, dacă pentru o curbă, rezultatul este pozitiv pentru toate punctele sale, aceasta este situată în semispațiul pozitiv delimitat de planul conturului poligonal. Valori pozitive şi negative indică faptul că suprafață planară suport a poligonului străbate curba. Calculul normalei se realizează cu funcția CalcND(). Funcția primeşte ca parametrii 3 puncte ale conturului poligonal şi determină normala şi distanța planului suport.

```
short CalcND(Vector &P1, Vector &P2, Vector &P3, Vector &normal, double &D)
{
    Vector v1, v2;
    double l1, l2, l;
    v1 = P2 - P1;
    v2 = P3 - P2;
    l1 = v1.VECT_LENGTH();
    if(fabs(l1) < V_PPREC)
        return 1;
    v1.VERSOR();
    l2 = v2.VECT_LENGTH();
    if(fabs(l2) < V_PPREC)
        return 1;
    v2.VERSOR();
    normal.VECTPRD(v1, v2);
    l = normal.VECT_LENGTH();
    if(fabs(l) < V_PPREC)
        return 1;
    normal.VERSOR();
    D = - normal.SCALPRD(P1);
    return 0;
}
```

3.2.2 Suprafața riglată

Suprafețele rulate (riglate) sunt elemente simple şi fundamentale ale domeniului modelării suprafețelor. O suprafață riglată este generată prin conectarea punctelor corespondente a două curbe spațiale $G(u)$ şi $Q(u)$, definite pe domeniul $[0,1]$, prin linii drepte numite *generatoare*. Caracteristica principală a unei suprafețe riglate este aceea că există cel puțin o linie dreaptă care trece prin punctul $P(u,v)$ şi care străbate întreaga suprafață. Cilindrii şi conurile sunt exemple de suprafețe riglate, iar suprafața plană descrisă anterior este una dintre cele mai simple suprafețe riglate.

Se consideră că generatoarea suprafeței $u = u_i$ unește punctele G_i și Q_i situate la extremele curbelor $G(u)$ respectiv $Q(u)$. Formula generatoarei va fi:

$$P(u_i, v) = G_i + v(Q_i - G_i) \tag{3.2.7}$$

v este parametrul definit de-a lungul riglei. Generalizând relația pentru orice generatoare a suprafeței, se obține formula parametrică pentru suprafața riglată definită prin două curbe spațiale:

$$P(u,v) = G(u) + v[Q(u) - G(u)] = (1-v)G(u) + vQ(u), \qquad u, v \in [0,1] \tag{3.2.8}$$

Păstrând constantă valoarea lui u în formulă, se determină generatoarele suprafeței $P(u_i, v)$ pe direcția lui v de pe suprafață, în timp ce dacă se menține fixă valoarea parametrului v se generează curbe pe suprafață pe direcția u, curbe care sunt o combinație liniară a granițelor. Granițele suprafeței riglate, $G(u)$ și $Q(u)$ pot fi considerate ca fiind $P(u,0)$ respectiv $P(u,1)$. Pentru valori mici ale lui v (spre zero) influența curbei $G(u)$ este mai mare, iar pentru valori mari ale lui v, ce tind spre 1, creste influența lui $Q(u)$ asupra geometriei suprafeței riglate.

O suprafață riglată realizează interpolarea completă a curbelor și nu doar a unei secvențe discretizate de puncte, de aceea acest proces este adesea referit ca și *interpolare transfinită* [66].

Din modul de generare a unei suprafețe riglate se poate observa faptul că aceasta permite curburi doar în direcția parametrului u a suprafeței, dacă granițele prezintă curburi. Curbura suprafeței în direcția v (în lungul generatoarelor) este zero, și astfel o suprafață riglată nu poate fi folosită pentru modelarea peticelor de suprafețe care au curburi în două direcții.

Un important aspect al suprafețelor rulate îl constituie generalitatea tipului admis pentru curbele generatoare $P(u,0)$ și $P(u,1)$. Acestea pot fi construite folosind oricare din metodele de proiectare a curbelor descrise. Singura restricție este dată de domeniul comun de parametrizare pentru curbe, care însă poate fi orice interval $[a,b] \in R$ și nu în mod obligatoriu intervalul unitate. Astfel, de exemplu, suprafața rulată poate fi generată de o curbă polinomială cubică și o curbă spline sau chiar un poligon.

Suprafețele riglate au fost modelate soft prin clasa CMeRuledSurface. Clasa cuprinde cele două curbe de graniță care definesc forma suprafeței. Generarea curbelor se poate realiza prin oricare din metodele descrise.

Funcția CMeRuledSurface::ConstructElement() conține algoritmul de construcție al suprafeței riglate. Sunt generate liniile generatoare iar apoi curbele transversale.

```
short CMeRuledSurface::ConstructElement(short mode)
{
    ControlPoints firstCurve, secondCurve;
    CMeSurfaceCurve *newSurfCurve;
    double t;
    short Nr = 20;
    DeleteContentElement();
    // obținerea punctelor curbelor de graniță
    firstBorderCurve->GetPoints(firstCurve, 1);
    secondBorderCurve->GetPoints(secondCurve, 1);
    int nr = firstCurve.GetSize();
    // generarea curbelor generatoare ale suprafețelor
```

```
    for(short i = 0; i < nr; i++)
    {   newSurfCurve = new CMeSurfaceCurve;
        newSurfCurve->curve.SetSize(2);
        newSurfCurve->curve[0] = firstCurve[i];
        myDoc->ComputePersp(newSurfCurve->curve[0]);
        SetPerspBBox(newSurfCurve->curve[0]);
        newSurfCurve->curve[1] = secondCurve[i];
        myDoc->ComputePersp(newSurfCurve->curve[1]);
        SetPerspBBox(newSurfCurve->curve[1]);
        surfaceCurves[0].AddTail(newSurfCurve);
    }
    // generarea curbelor care reprezintă o combinație liniara a curbelor date
    for(i = 0; i < Nr; i++)
    {   t = (double)(i)/(Nr-1);
        newSurfCurve = new CMeSurfaceCurve;
        newSurfCurve->curve.SetSize(nr);
        for(short j = 0; j < nr; j++)
        {   newSurfCurve->curve[j] = firstCurve[j] * (1- t) + secondCurve[j] *t;
            myDoc->ComputePersp(newSurfCurve->curve[j]);
            SetPerspBBox(newSurfCurve->curve[j]);
            surfaceCurves[1].AddTail(newSurfCurve);
        }
    }
    return 0;
}
```

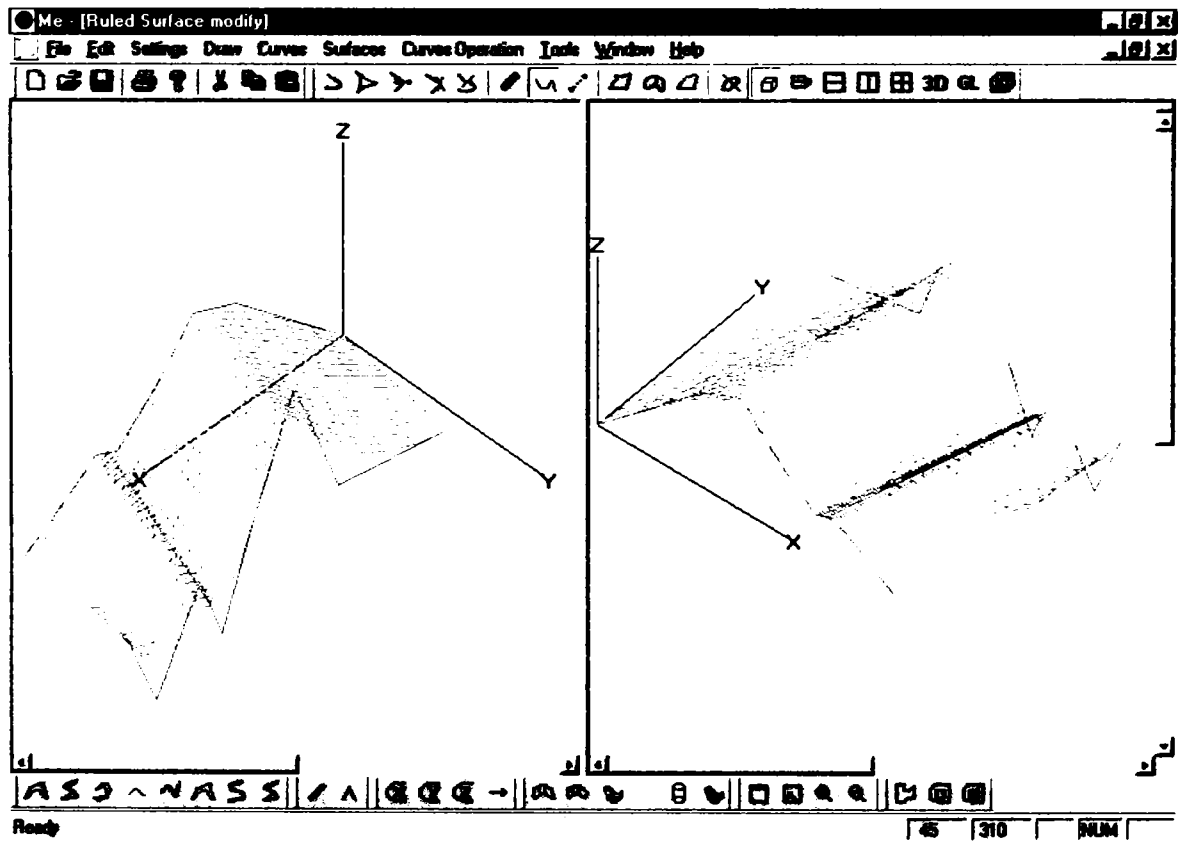

Fig. 3.2.1 Suprafață riglată

3.2.3 Suprafața conică

O subclasă a suprafețelor rulate o reprezintă *pânzele conice*. Pentru acest tip de suprafețe, una din curbele generatoare este redusă la un punct. Datele inițiale, necesare în

construcție sunt curba $c(u)$, $u \in [0,1]$ inclusă într-un plan $c \in \pi$, și un punct $P(x, y, z) \notin \pi$. Suprafața conică de vârf P și curbă c este generată de dreptele ce trec prin P și $c(u)$:

$$s(u,v) = (1-v)P + vc(u) \tag{3.2.9}$$

Curba planară c este denumită *curbă directoare* a conicei. La fel ca și în cazul suprafețelor rulate, curba directoare poate fi proiectată apelându-se la oricare din metodele descrise obține o suprafață înfășurătoare închisă, care poate alcătui un solid prin adăugarea suprafeței plane limitate de curba directoare [26][74].

Construcția pânzelor conice s-a realizat prin aceeași clasă CMeRuledSurface. Curba secundară de graniță este alcătuită dintr-un singur punct.

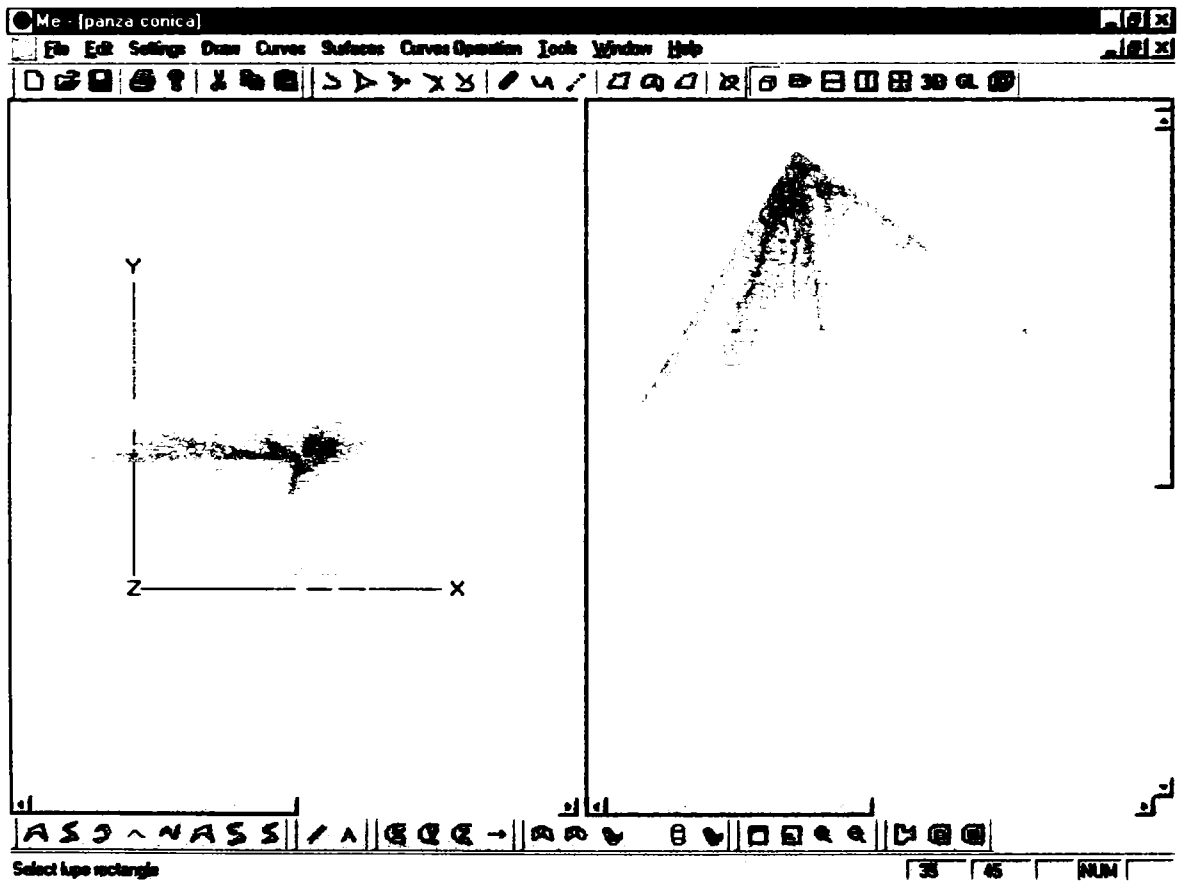

Fig. 3.2.2 Reprezentarea unei suprafețe conice

3.2.4 Suprafața de revoluție

Suprafața de revoluție se generează prin rotirea unei curbe planare $G(u)$ cu un unghi v în jurul unei *axe de rotație*. Rotirea curbei creează un cerc (dacă $v = 360$) pentru fiecare punct de pe curbă, a cărui centru trece prin axa de rotație și a cărui rază $r_z(u)$ este variabilă în funcție de distanța punctului la axa de rotație. Curba planară și cercurile sunt denumite *profil* și respectiv *paralele* în timp ce pozițiile diferite ale profilului în jurul axei sunt numite *meridiane* [59].

Curba planară și axa de rotație formează planul de unghi zero, care este $v = 0$. Relația parametrică a suprafeței de revoluție este definită în raport cu un sistem local atașat suprafeței. Acest sistem local (indicat prin indicii L) poate fi creat în felul următor: axa Z_L

a sistemului este identică cu axa de rotație, axa X_L este aleasă direcția perpendiculară din punctul $u=0$ pe planul curbei profil iar punctul de intersecție dintre X_L şi Z_L reprezintă originea sistemului local. Axa Y_L este determinată automat cu ajutorul *regulii mâinii drepte*. Se consideră punctul $G(u)=P(u,0)$ de pe profil care se roteşte cu un unghi v în jurul axei Z_L odată cu întreg profilul. Pornind de la triunghiul format din punctul inițial, punctul rotit $P(u,v)$ şi originea curbei paralele corespunzătoare, triunghi perpendicular pe axa Z_L, relația parametrică a suprafeței de revoluție este:

$$P(u,v)=r_z(u)\cos v\bar{n}_1+r_z(u)\sin v\bar{n}_2+z_L(u)\bar{n}_3, \qquad 0\le u\le 1,\ \ 0\le v\le 2\pi \quad (3.2.10)$$

Dacă se setează $z_L(u)=u$ pentru fiecare punct de pe profil, formula oferă coordonatele locale $(x_L,y_L,z_L)=(r_z(u)\cos v, r_z(u)\sin v, u)$ ale unui punct $P(u,v)$ al suprafeței riglate. Pentru reprezentarea reală a suprafeței este necesară transformarea coordonatelor locale în coordonate globale [16].

Baza de date inițială a suprafeței de revoluție trebuie să includă profilul acesteia, axa de rotație şi unghiul de rotație, acesta fiind specificat prin unghiul de început şi unghiul de sfârşit. Pentru afişarea suprafeței prin intermediul unei rețele de dimensiunea *m x n*, domeniul variabilei *u* este împărțit în mod echidistant în $m-1$ părți. În mod similar, domeniul lui *v* este împărțit în n valori echidistante. Având aceste valori parametrice, prin aplicarea relației sunt generate punctele situate pe curbele paralele ale suprafeței de revoluție.

Pentru generarea acestor suprafețe s-a construit clasa CMeRevolutionSurface derivată din clasa CMeStandardSurface. Variabilele membre externalCurve şi internalCurve reprezintă pointeri la curbele planare care constituie profilul suprafeței. De asemenea sistemul local al suprafeței este păstrat în variabila coordSystem. Axa de rotație este axa Z globala a spațiului de lucru. Metoda CMeRevolutionSurface::GenerateSurface() implementează algoritmul de construcție al suprafeței. Funcția este apelata consecutiv pentru ambele profile ale suprafeței, în cazul în care aceasta conține şi un profil interior. Sunt generate, în sistemul local al suprafeței, curbele verticale şi orizontale ale acesteia memorate în variabila membră CMeStandardSurface::surfaceCurves[2].

Metoda CMeRevolutionSurface::ConstructElement() realizează construcția întregii suprafețe în sistemul său local şi trecerea punctelor în sistemul global.

```
short CMeRevolutionSurface::ConstructElement(short mode)
{
   DeleteContentElement();
   // generarea curbelor pentru suprafața exterioară
   GenerateSurface(externalCurve);
   if(internalCurve)
      GenerateSurface(internalCurve);  // generarea curbelor pentru suprafața interioară
   // trecerea tuturor curbelor in sistemul global
   for(short i = 0; i < 2; i++)
   {  POSITION pos = surfaceCurves[i].GetHeadPosition();
      while(pos != NULL)
      {  CMeSurfaceCurve *surfCurve;
         surfCurve = (CMeSurfaceCurve *)surfaceCurves[i].GetNext(pos);
         TrecereSistemControlPoints(surfCurve->curve, coordSystem, FALSE);
         for(short j = 0; j < surfCurve->curve.GetSize(); j++)
         { myDoc->ComputePersp(surfCurve->curve[j]);
           SetPerspBBox(surfCurve->curve[j]);
```

```
            }
        }
    }
    return 0;
}

short CMeRevolutionSurface::GenerateSurface(CMeCurve *curve)
{
    CMeSurfaceCurve *newSurfCurve;
    short dense = 30, nr, i, j, idBegin, idEnd;
    double angle, cosAngle, sinAngle, r, pas;
    Vector dir;
    ControlPoints genCurve;
    // prima curba de pe suprafață este tocmai curba generatoare
    curve->GetPoints(genCurve, 1);
    // trecerea punctelor in sistemul local al suprafeței
    TrecereSistemControlPoints(genCurve, coordSystem, TRUE);
    // generarea curbelor verticale de pe suprafața
    nr = genCurve.GetSize();
    pas = 360/dense;
    for(i = 0; i < dense; i++)
    {   angle = i*pas;
        cosAngle = COSINUS(angle);
        sinAngle = SINUS(angle);
        newSurfCurve = new CMeSurfaceCurve;
        newSurfCurve->curve.SetSize(nr);
        for(j = 0; j < nr; j++)
        {   dir = genCurve[j];
            dir[Z] = 0.0;
            r = dir.VECT_LENGTH();
            newSurfCurve->curve[j][X] = r *cosAngle;
            newSurfCurve->curve[j][Y] = r *sinAngle;
            newSurfCurve->curve[j][Z] = genCurve[j][Z];
        }
        surfaceCurves[0].AddTail(newSurfCurve);
}
// generarea curvelor orizontale de pe suprafață
pas = 360/dense;
if(deplAxis)
{   idBegin = 0;
    idEnd = nr;
}
else
{    idBegin = 1;
    idEnd = nr -1;
}
for(j = idBegin; j < idEnd; j++)
{   dir = genCurve[j];
    dir[Z] = 0.0;
    r = dir.VECT_LENGTH();
    newSurfCurve = new CMeSurfaceCurve;
    newSurfCurve->curve.SetSize(dense+1);
    for(i = 0; i <= dense; i++)
    {    angle = i*pas;
        cosAngle = COSINUS(angle);
        sinAngle = SINUS(angle);
        newSurfCurve->curve[i][X] = r *cosAngle;
        newSurfCurve->curve[i][Y] = r *sinAngle;
        newSurfCurve->curve[i][Z] = genCurve[j][Z];
     }
```

```
        surfaceCurves[1].AddTail(newSurfCurve);
    }
    return 0;
}
```

Funcția globală TrecereSistemControlPoints() realizează trecerea unui tablou de puncte dintr-un sistem local în sistemul global sau invers.

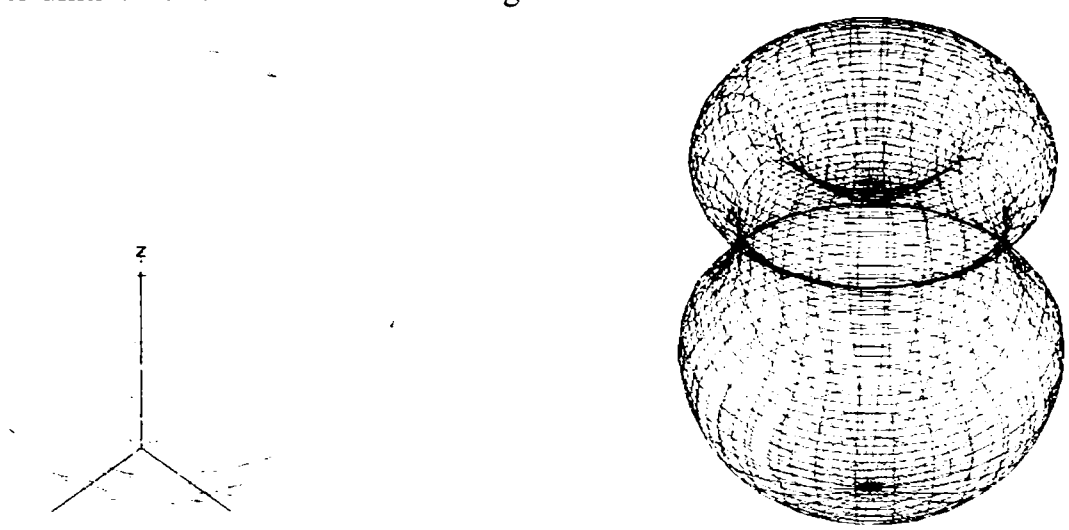

Fig. 3.2.3 Suprafețe de revoluție

3.2.5 Cilindrul generalizat

Cilindrul generalizat este o suprafață generată prin translația unei curbe planare strâmbe în lungul unei direcții date. De asemenea, cilindrul poate fi construit prin deplasarea unei linii drepte (numită generatoare) de-a lungul unei curbe planare date (numită directoare). Linia dreaptă rămâne tot timpul paralelă cu un vector fix dat care definește direcția v a cilindrului. Curba planară directoare $G(u)$ poate reprezenta orice fel de curbă sau entitate wireframe. Vectorul de poziție al oricărui punct $P(u,v)$ de pe suprafața cilindrului este:

$$P(u,v) = G(u) + v\bar{n}_v, \qquad 0 \le u \le u_{max}, \quad 0 \le v \le v_{max} \tag{3.2.11}$$

Datele inițiale necesare pentru generarea unei suprafețe cilindrice sunt *curba directoare* $G(u)$, *axa cilindrului* n_v și *lungimea cilindrului* v descrisă prin frontierele minimă și maximă [57][66]. O valoare 0 pentru marginea de jos indică planul director. Axa cilindrului poate fi dată prin două puncte, cu ajutorul cărora se va calcula versorul n_v al axei. Afișarea unui cilindru generalizat cu printr-o rețea *m x n* urmează aceeași abordare ca și în cazul suprafețelor de revoluție.

Clasa corespunzătoare acestui tip de suprafață este CMeCylindricalSurface a cărei clasă de bază este de asemenea CMeStandardSurface. În cadrul clasei sunt stocate toate datele corespunzătoare suprafeței, variabila directrixCurve conține curba directoare, genDir păstrează axa cilindrului iar lungimea lui este dată de length. Funcția membră ConstructElement() generează suprafața cilindrică.

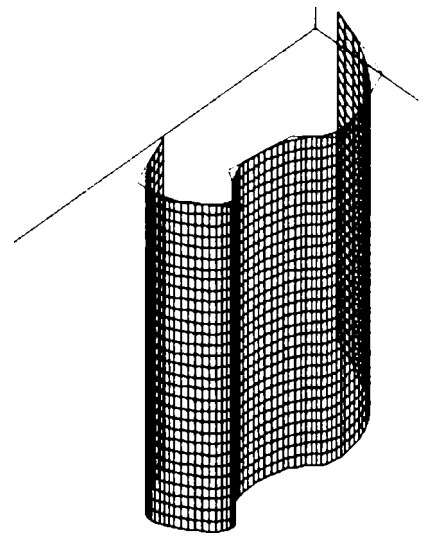

Fig. 3.2.5 Suprafețe cilindrice

```
short CMeCylindricalSurface::ConstructElement(short mode)
{
    short i, j, dense = 30;
    double v;
    ControlPoints curvePoints;
    DeleteContentElement();
    InitPerspBBox();
    directrixCurve->GetPoints(curvePoints, 1);
    // generarea curbelor orizontale ale suprafeței
    short nr = curvePoints.GetSize();
    for(i = 0; i <= dense; i++)
    {
        v = i*length/dense;
        CMeSurfaceCurve *newSurfCurve = new CMeSurfaceCurve;
        newSurfCurve->curve.SetSize(nr);
        for(j = 0; j < nr; j++)
        {
            newSurfCurve->curve[j] = curvePoints[j] + genDir*v;
            myDoc->ComputePersp(newSurfCurve->curve[j]);
            SetPerspBBox(newSurfCurve->curve[j]);
        }
        surfaceCurves[0].AddTail(newSurfCurve);
    }
    // generarea curbelor verticale
    for(j = 0; j < nr; j++)
    {
        CMeSurfaceCurve *newSurfCurve = new CMeSurfaceCurve;
        newSurfCurve->curve.SetSize(dense+1);
        for(i = 0; i <= dense; i++)
        {
            v = i*length/dense;
            newSurfCurve->curve[i] = curvePoints[j] + genDir*v;
            myDoc->ComputePersp(newSurfCurve->curve[i]);
            SetPerspBBox(newSurfCurve->curve[i]);
        }
        surfaceCurves[1].AddTail(newSurfCurve);
    }
    return 0;
}
```

Axa cilindrului este determinată ca fiind normala planului ce conține curba directoare.

Suprafețele analitice se regăsesc în forma unora din elementele complexe care compun spațiul laboratorului virtual. Un exemplu concludent este proiectarea aparatului de radiografie ortopanoramică. Suportul acestuia a fost realizat cu suprafețe poligonale, iar partea centrală cuprinde suprafețe rulate, cilindrice și de revoluție.

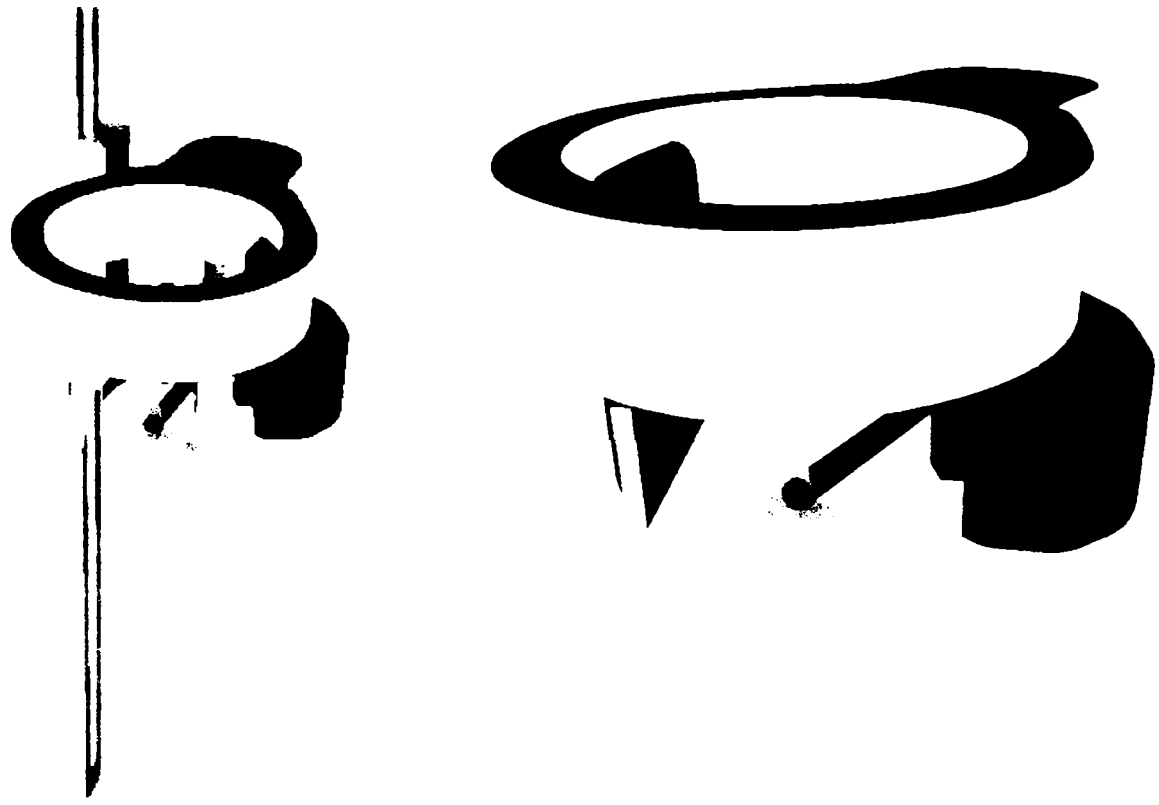

Fig. 3.2.6 Modelul geometric al aparatului de radiografie ortopanoramică

Elementele cu forme mai complexe ale aparatului au fost generate cu suprafețe analitice. De asemenea pereții și podeaua au fost realizate exclusiv cu elemente poligonale, iar suprafețele cilindrice, rulate și de revoluție sunt folosite la proiectarea formei roboților mobili din cadrul laboratorului.

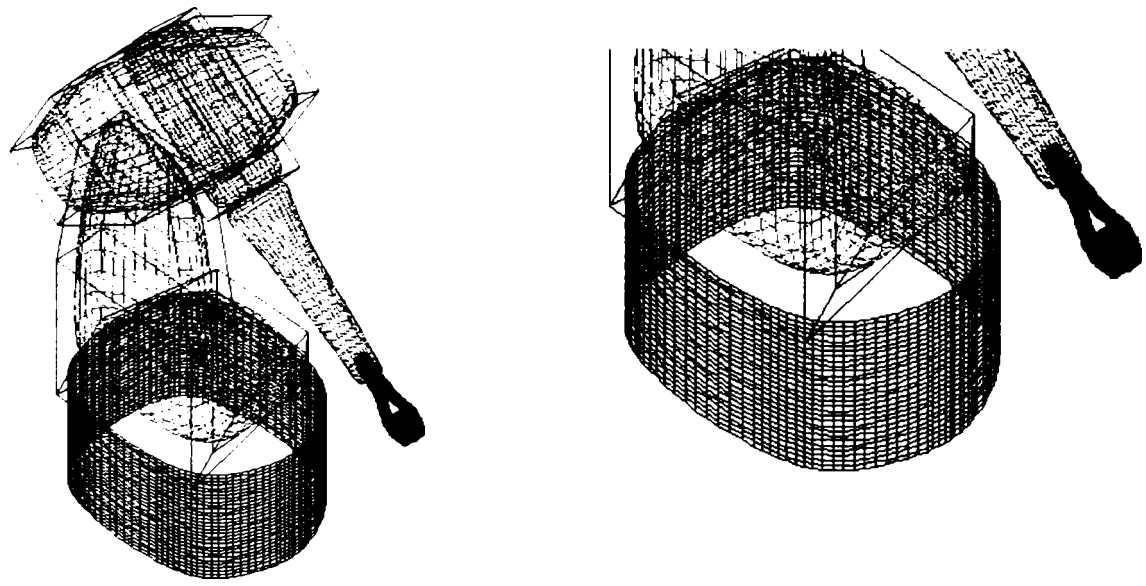

Fig. 3.2.7 Modelarea formei robotului utilizând și suprafețe analitice

3.3 Suprafețe Bézier produs tensorial

Suprafețele Bézier produs tensorial reprezintă una dintre cele mai generale clase de suprafețe sintetice folosite în domeniul proiectării. Primul care a dezvoltat aceste suprafețe a fost de Casteljau, între 1959 şi 1963, dar popularitatea lor se datorează muncii lui Bézier [13][36]. Inițial, peticele Bézier au fost folosite la aproximarea unor suprafețe date. Datorită însă caracteristicilor lor, orice suprafață B-spline poate fi scrisă în forma peticelor Bézier.

Exemplul suprafețelor Bézier demonstrează utilitatea produsului tensorial în cadrul modelării suprafețelor [11][12]. Odată ce acest principiu este prezentat, va fi uşoară generalizarea altor scheme de curbe pentru construcția de suprafețe produs tensorial.

3.3.1 Interpolarea biliniară

Metoda interpolării liniare în spațiul E^3, descrisă în capitolul introductiv, a fost folosită pentru dezvoltarea curbelor Bézier. Într-un mod asemănător, teoria suprafețelor Bézier produs tensorial se bazează pe conceptul de *interpolare biliniară*. În timp ce interpolarea liniară se aplică curbelor "simple" între două puncte, interpolarea biliniară se aplică suprafețelor "simple" între patru puncte.

Fie $b_{00}, b_{01}, b_{10}, b_{11}$ patru puncte distincte în E^3. Mulțimea tuturor punctelor $x \in E^3$ de forma:

$$\mathrm{x}(u,v) = \sum_{i=0}^{1}\sum_{j=0}^{1} \mathrm{b}_{i,j} B_i^1(u) B_j^1(v) \tag{3.3.1}$$

este numită paraboloidul hiperbolic prin cele patru puncte. În forma matricială:

$$\mathrm{x}(u,v) = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} \mathrm{b}_{00} & \mathrm{b}_{01} \\ \mathrm{b}_{10} & \mathrm{b}_{11} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} \tag{3.3.2}$$

Deoarece relația este liniară atât în u cât şi în v şi interpolează punctele de intrare, suprafața $x(u,v)$ poartă numele de *interpolant biliniar* [14].

Pătratul unitate $u, v \in [0,1]$ este domeniul interpolantului, în timp ce suprafața x este limita lui. O linie paralelă cu una din axele domeniului corespunde unei curbe din domeniu numită curbă isoparametrică. Orice curbă isoparametrică fiind o linie dreaptă, paraboloidul hiperbolic reprezintă o suprafața riglată. Curba isoparametrică $u = 0$ este interpolantul liniar al punctelor b_{00}, b_{01}. În mod similar, acest lucru este valabil şi pentru celelalte trei curbe de frontieră.

Evaluarea interpolantului biliniar se realizează în două etape. Procedeul este folosit mai târziu în contextul interpolării produs tensorial. Sunt calculate punctele intermediare:

$$\begin{aligned} \mathrm{b}_{0,0}^{0,1} &= (1-v)\mathrm{b}_{0,0} + v\mathrm{b}_{0,1} \\ \mathrm{b}_{1,0}^{0,1} &= (1-v)\mathrm{b}_{1,0} + v\mathrm{b}_{1,1} \end{aligned} \tag{3.3.3}$$

Interpolantul biliniar este definit în raport cu aceste puncte:

$$x(u,v) = b_{0,0}^{1,1}(u,v) = (1-u)b_{0,0}^{1,1} + ub_{1,0}^{0,1} \tag{3.3.4}$$

Se calculează mai întâi coeficienții liniei isoparametrice $v = const$ şi apoi se evaluează linia isoparametrică în u. Acelaşi rezultat se obține şi pentru o linie isoparametrică $u = const$. Deoarece interpolarea liniară este o transformare afină, interpolarea biliniară alcătuită din interpolări liniare atât pe direcția u cât şi pe direcția v, poate fi denumită transformare biafină.

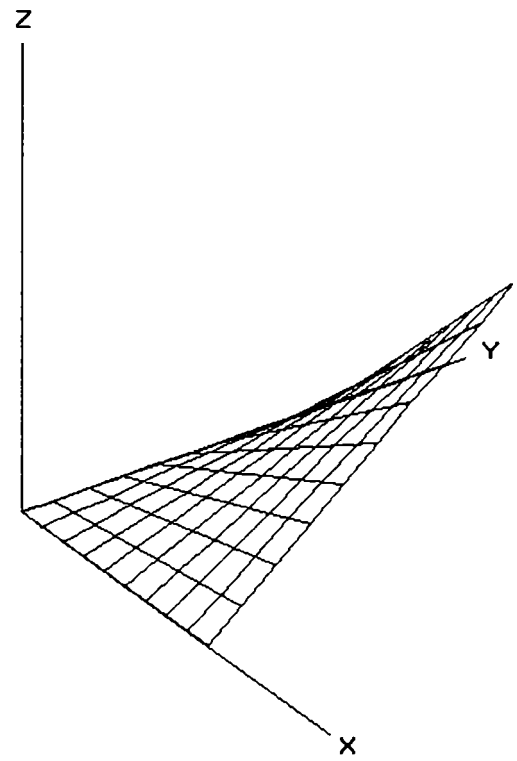

Fig. 3.3.1 Interpolarea biliniară

Termenul de paraboloid hiperbolic dat interpolării biliniare vine din geometria analitică. Se consideră suprafața neparametrică $z = xy$, suprafață ce poate fi interpretată ca şi un interpolant biliniar al punctelor *(0,0,0), (1,0,0), (0,1,0), (1,1,1)*. Dacă suprafața este intersectată cu un plan paralel cu planul *xy*, curba rezultantă este o *hiperbolă*. Dacă este intersectată cu un plan care conține axa *z*, curba rezultantă este o *parabolă*.

3.3.2 Algoritmul de Casteljau direct

Curbele Bézier pot fi obținute prin aplicarea repetată a interpolării liniare. Similar, construcția suprafețelor Bézier se realizează prin aplicarea repetată a interpolării biliniare.

Fiind dat un tablou dreptunghiular de puncte $b_{i,j}$, $0 \le i, j \le n$ şi valorile parametrice (u,v), suprafața generată de aceste puncte este dată de relația:

$$b_{i,j}^{r,r} = \begin{bmatrix} 1-u & u \end{bmatrix} \begin{bmatrix} b_{i,j}^{r-1,r-1} & b_{i,j+1}^{r-1,r-1} \\ b_{i+1,j}^{r-1,r-1} & b_{i+1,j+1}^{r-1,r-1} \end{bmatrix} \begin{bmatrix} 1-v \\ v \end{bmatrix} \qquad r = 1,\ldots,n \quad i,j = 0,\ldots,n-r \tag{3.3.5}$$

Punctele $b_{i,j}^{0,0}$ sunt punctele tabloului inițial $b_{i,j}^{0,0} = b_{i,j}$. $b_{0,0}^{n,n}$ este punctul corespunzător valorilor parametrice (u,v) de pe suprafața Bézier $b^{n,n}$. Tabloul de puncte $b_{i,j}$ este denumit *rețeaua de control sau poliedrul de control al suprafeței* [13].

Suprafețele generate cu algoritmul de Casteljau direct au același grad în u şi v. Dacă poliedrul de control $b_{i,j}$, $i = \overline{0,m}$, $j = \overline{0,n}$ conține un număr diferit de puncte pe cele două direcții, algoritmul de Casteljau direct nu poate fi aplicat până când este obținut punctul de pe suprafață. Construcția suprafețelor care au grade diferite în u şi v se realizează printr-o succesiune de etape. Prin aplicarea, mai întâi, a algoritmului de Casteljau direct de $k = \min(m,n)$ ori se obțin punctele $b_{i,j}^{k,k}$ care formează poligonul de control al unei curbe. Punctul de pe suprafață rezultă folosind algoritmul univariant de Casteljau pentru poligonul de control determinat. Această distincție de caz este puțin neobișnuită şi nu va fi întâlnită în abordarea produsului tensorial.

3.3.3 Suprafețe Bézier produs tensorial

Stiliştii creează fizic suprafețe prin aplicarea de şabloane asupra modelelor de lut pentru eliminarea surplusului de material. Sculptarea suprafețelor complexe în lut necesită folosirea mai multor şabloane diferite. Analizând acest proces din punct de vedere teoretic, se ajunge la următoarea definiție intuitivă pentru suprafață: O suprafață este locul geometric al unei curbe care se deplasează în spațiu şi îşi schimbă forma.

Pornind de la acest concept intuitiv se realizează o descriere matematică a unei suprafețe. Se consideră curba care se deplasează ca fiind o curbă Bézier de grad m. (Aceasta constituie o restricție asupra claselor de suprafețe care pot fi reprezentate cu ajutorul produsului tensorial.) In orice moment, curba care se deplasează şi se deformează este determinată de un set de puncte de control. Fiecare punct inițial de control se deplasează prin spațiu urmărind curba. Traiectoriile parcurse de aceste puncte sunt reprezentate de asemenea prin curbe Bézier de grad n. Fie $b^m(u)$ curba Bézier inițială de grad m:

$$b^m(u) = \sum_{i=0}^{m} b_i B_i^m(u) \tag{3.3.6}$$

Traiectoria fiecărui punct de control este o curbă Bézier de grad n:

$$b_i = b_i(v) = \sum_{j=0}^{n} b_{i,j} B_j^n(v) \tag{3.3.7}$$

Suprafața descrisă de curba inițială rezultă din combinarea celor două relații:

$$b^{m,n}(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{i,j} B_i^m(u) B_j^n(v) \tag{3.3.8}$$

Suprafața astfel generată şi parametrizată se numeşte *suprafață Bézier produs tensorial*. Suprafața este complet determinată de o rețea de puncte $b_{i,j}$, $i = \overline{0,m}$ $j = \overline{0,n}$

[15]. Se poate demonstra simplu că definiția unei suprafețe Bézier produs tensorial precum și definiția dată de algoritmul de Casteljau direct sunt echivalente.

Suprafața Bézier a fost obținută prin deplasarea curbei isoparametrice corespondente lui $v=0$. Oricare din cele trei frontiere rămase pot fi folosite de asemenea ca și curbă inițială.

O curbă isoparametrică oarecare $v=const$ a unei suprafețe Bézier $b^{m,n}$ este o curbă Bézier de gradul m în u, iar cele $m+1$ puncte Bézier ale sale sunt obținute prin evaluarea rândurilor rețelei de control la $v=const$, prin formula:

$$b_{i,0}^{0n}(v)=\sum_{j=0}^{n} b_{ij} B_j^n(v); \qquad i=\overline{0,m} \tag{3.3.9}$$

Coeficienții curbei isoparametrice pot fi obținuți prin aplicarea de $m+1$ ori a algoritmului de Casteljau univariant. Astfel, un punct de pe suprafață este obținut prin aplicarea succesivă a algoritmului de Casteljau.

Curbele isoparametrice $u=const$ sunt tratate în mod analog. De remarcat că alte linii drepte din domeniu sunt relaționate la curbe de grad $m+n$ de pe petic. Două exemple speciale sunt diagonalele unui domeniu dreptunghiular.

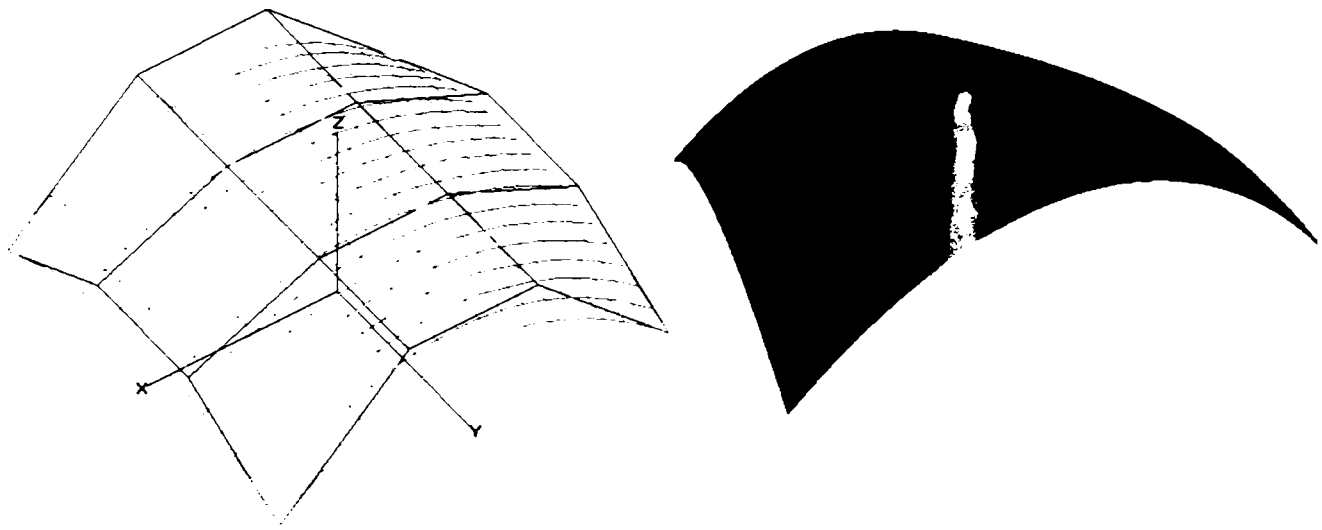

Fig. 3.3.2 Reprezentarea unei suprafețe Bézier sub formă de cadru și realistic

Suprafețele de tip produs tensorial sunt implementate cu ajutorul clasei CMeSurface descrisă în secțiunea 3.1.3. Pornind de la punctele de control și, în funcție de tipul suprafeței produs tensorial, se generează curbele orizontale și verticale ale peticului. Metoda ConstructElement() a clasei CMeSurface apelează succesiv pentru fiecare petic al suprafeței funcția proprie de construcție a peticului CMeSurfacePatch::ConstructElement().

```
short CMeSurface::ConstructElement(short mode)
{
    POSITION posPatchLine, posPatch;

   // setarea gradului curbei
   if((this->*this->surfDegree[surfaceType])(mode))
       return 1;
   // parcurgerea tabloului de petice al suprafeței și apelarea funcției de generare a
   // fiecărui petic
   posPatchLine = surfacePatchs.GetHeadPosition();
```

```
    while (posPatchLine != NULL)
    {
        CMeSurfacePatchLine* pSurfPatchLine;
        pSurfPatchLine = (CMeSurfacePatchLine *)surfacePatchs.GetNext(posPatchLine);
        posPatch = pSurfPatchLine->surfacePatchLine.GetHeadPosition();
        while(posPatch != NULL)
        {
            CMeSurfacePatch *pSurfacePatch =
                (CMeSurfacePatch *)pSurfPatchLine->surfacePatchLine.GetNext(posPatch);
            pSurfacePatch->ConstructElement();
        }
    }
    return 0;
}
```

Pentru fiecare tip de suprafață produs tensorial a fost definită o funcție membră a clasei CMeSurfacePatch care generează suprafața. Algoritmul de construcție a suprafețelor Bézier este implementat de funcția BezierSurface();

```
// construirea punctelor suprafeței Bézier
void CMeSurfacePatch::BezierSurface(void)
{
    short nr_curves = 20, i, j, k;
    double v0, u0;
    ControlPoints tempTab, controlP;
    CMeSurfaceCurve *surf;

    // eliberarea punctelor suprafeței daca aceasta a fost deja generata
    CMeStandardSurface::DeleteContentElement();
    // generarea curbelor suprafeței pe linii
    tempTab.SetSize(controlMap.nr_row);
    controlP.SetSize(controlMap.nr_col);
    // prima curba este generata de punctele de control de pe prima linie
    controlMap.CopyLineCol(controlP, 0, ROW);
    DecasteljauCurve(controlP, &surf);
    surfaceCurves[0].AddTail(surf);
    // construirea curbelor intermediare
    for(k =1; k < nr_curves-1; k++)
    {
        v0 = (double)k/(nr_curves - 2);
        // generarea punctelor de control pentru v0
        for(j = 0; j < controlMap.nr_col; j++)
        {
            // copierea coloanei j in tabloul tempTab
            controlMap.CopyLineCol(tempTab, j, COL);
            DecasteljauPoint(tempTab, v0, controlP[j]);
        }
        // construcția curbei pornind de la punctele de control controlP
        DecasteljauCurve(controlP, &surf);
        surfaceCurves[0].AddTail(surf);
    }
    // ultima curba este generata de punctele de control de pe ultima linie
    controlMap.CopyLineCol(controlP, controlMap.nr_row-1, ROW);
    DecasteljauCurve(controlP, &surf);
    surfaceCurves[0].AddTail(surf);

    // generarea curbelor suprafeței de pe coloane
    tempTab.SetSize(controlMap.nr_col);
    controlP.SetSize(controlMap.nr_row);
```

```
    // prima curba este generata de punctele de control de pe prima coloana
    controlMap.CopyLineCol(controlP, 0, COL);
    DecasteljauCurve(controlP, &surf);
    surfaceCurves[1].AddTail(surf);
    // construirea curbelor intermediare
    for(k =1; k < nr_curves-1; k++)
    {
        u0 = (double)k/(nr_curves - 2);
        // generarea punctelor de control pentru v0
        for(i = 0; i < controlMap.nr_row; i++)
        {
            // copierea coloanei j in tabloul tempTab
            controlMap.CopyLineCol(tempTab, i, ROW);
            DecasteljauPoint(tempTab, u0, controlP[i]);
        }
        // construcția curbei pornind de la punctele de control controlP
        DecasteljauCurve(controlP, &surf);
        surfaceCurves[1].AddTail(surf);
    }
    // ultima curba este generata de punctele de control de pe ultima linie
    controlMap.CopyLineCol(controlP, controlMap.nr_col-1, COL);
    DecasteljauCurve(controlP, &surf);
    surfaceCurves[1].AddTail(surf);
}
```

Caracteristici ale suprafețelor Bézier produs tensorial

Cele mai multe proprietăți ale peticelor Bézier derivă în mod direct din proprietățile curbelor Bézier.

Invarianța afină. Algoritmul de Casteljau direct constă din interpolări biliniare repetate şi posibile subsecvențe de interpolări liniare. Toate aceste operații sunt invariant afine deci şi compunerea lor este invariant afină Acest lucru conduce la faptul că suprafața Bézier este o combinație baricentrică, şi deci invariant afină, a poliedrului de control. Polinoamele $B_i^m(u)B_j^n(v)$ realizează o partiție a unității [71]. Această proprietate determină o creştere a vitezei operațiilor de manipulare a suprafețelor. Deplasarea sau rotirea unei suprafețe se realizează mai rapid dacă operația este aplicată punctelor de control şi apoi se recalculează suprafața. Mişcarea unui robot, proiectat ca un ansamblu complex de suprafețe şi curbe se va executa mai rapid aplicând acest principiu.

Proprietatea de acoperire convexă. Polinoamele Bernstein $B_i^m(u), B_j^n(u)$ sunt pozitive pentru $u,v \in [0,1]$. Luând în considerare faptul că $\sum_{i=0}^{m}\sum_{j=0}^{n} B_i^m(u)B_j^n(u) = 1$, suprafața $b^{m,n}(u,v)$ este o combinație convexă a punctelor $b_{i,j}$, şi deci este inclusă în înfăşurătoarea convexă a punctelor sale de control [11][13]. Acest lucru, la fel ca şi în cazul curbelor, este foarte important pentru testul brut de coliziune realizat cu ajutorul cutiei minmax. Calculul cutiei minmax se realizează cu harta de control şi nu cu suprafața însăşi ceea ce conduce la un timp de calcul mai redus. Dacă cutiile minmax a două suprafețe nu se intersectează, suprafețele sunt în mod cert disjuncte. Acest test al cutiilor minmax elimină rapid situațiile în care corpurile nu au nimic în comun ceea ce are un rol important în procesul de mişcare a unui robot pentru verificarea poziției acestuia față de obiectele înconjurătoare.

Vectorii suprafeței Bézier

Analiza geometrică a unei suprafețe, așa cum s-a prezentat în capitolul anterior, implică calculul vectorilor tangenți, a vectorilor normali și a vectorilor twist ai suprafeței cu ajutorul geometriei diferențiale.

Vectori tangenți

În cazul curbelor, derivatele au fost considerate prin diferențierea punctelor de control. Acest lucru este corect și în cazul pânzelor. Derivatele unei suprafețe într-un punct sunt *derivatele parțiale* $\frac{\partial}{\partial u}, \frac{\partial}{\partial v}$. Derivata parțială reprezintă vectorul tangent la o curbă isoparametrică, și deci vectorul tangent la suprafață pe direcția respectivă. Calculul tangentei la suprafață pe direcția u se realizează derivând relația suprafeței produs tensorial în raport cu u:

$$\frac{\partial}{\partial u} b^{m,n}(u,v) = \sum_{j=0}^{n} \left[\frac{\partial}{\partial u} \sum b_{i,j} B_i^m(u) \right] B_j^n(v) \tag{3.3.10}$$

Termenul dintre parantezele drepte depinde doar de u, și astfel se poate aplica formula de derivare a unei curbe Bézier rezultând:

$$\frac{\partial}{\partial u} b^{m,n}(u,v) = m \sum_{j=0}^{n} \sum_{i=0}^{m-1} \Delta^{1,0} b_{i,j} B_i^{m-1}(u) B_j^u(v) \tag{3.3.11}$$

Indecșii superiori $(1,0)$ ai operatorului standard diferențial Δ indică faptul că diferența este realizată doar pentru primul indice inferior: $\Delta^{(1,0)} b_{i,j} = b_{i+1,j} - b_{i,j}$. Dacă se calculează derivata parțială în raport cu v se folosește operatorul diferențial care acționează doar asupra celui de al doilea indice inferior: $\Delta^{0,1} b_{i,j} = b_{i,j+1} - b_{i,j}$ [57].

Ca și în alte situații, problema derivatelor unei suprafețe poate fi descompusă în mai multe probleme univariante. Astfel, pentru calcularea derivatei parțiale în raport cu u, toate coloanele rețelei de control sunt interpretate ca și curbe Bézier de grad m și se calculează derivatele lor, fiind evaluate pentru o anumită valoare u. Aceste derivate sunt interpretate ca și coeficienți ai unei curbe Bézier de grad n și se calculează valoarea curbei pentru v dorit. Se obține în acest mod tangenta la suprafață în punctul $P(u,v)$ pe direcția u.

Formula derivatelor parțiale în raport cu u de ordin superior este:

$$\frac{\partial^r}{\partial u^r} b^{m,n}(u,v) = \frac{m!}{(m-r)!} \sum_{j=0}^{n} \sum_{i=0}^{m-r} \Delta^{r,0} b_{i,j} B_i^{m-r}(u) B_j^n(v) \tag{3.3.12}$$

Operatorul diferențial este definit prin formula:

$$\Delta^{r,0} b_{i,j} = \Delta^{r-1,0} b_{i+1,j} - \Delta^{r-1,0} b_{i,j} \tag{3.3.13}$$

Relațiile pentru derivatele parțiale în raport cu v de ordin superior sunt similare. Pornind de la aceste formule se pot determina derivatele parțiale mixte de ordin arbitrar ale suprafeței:

$$\frac{\delta^{r+s}}{\delta u^r \delta u^s} b^{m,n}(u,v) = \frac{m!n!}{(m-r)!(n-s)!} \sum_{i=0}^{m-r} \sum_{j=0}^{n-s} \Delta^{r,s} b_{i,j} B_i^{m-r}(u) B_j^{n-s}(v) \tag{3.3.14}$$

Coeficienții operatorului $\Delta^{r,s} b_{i,j}$ sunt vectori şi deci nu aparțin spațiului E^3.

Derivatele parțiale asociate curbelor de frontieră sunt denumite *derivate de traversare a frontierei* [21]. Formula derivatei parțiale restricționată pentru $u = 0$ poate fi scrisă:

$$\frac{\partial^r}{\partial u^r} b^{m,n}(0,v) = \frac{m!}{(m-r)!} \sum_{j=0}^{n} \Delta^{r,0} b_{0,j} B_j^n(v) \tag{3.3.15}$$

Se poate observa că derivatele de traversare a frontierei de ordinul r depind doar de $r+1$ rânduri (coloane) de puncte Bézier din vecinătatea frontierei. Acest lucru va fi important la formularea condițiilor de continuitate între petice adiacente.

Vectorii normali

Vectorul normal n al unei suprafețe este vectorul normalizat care este perpendicular pe suprafață într-un punct dat. Normala poate fi calculată ca produs vectorial al oricăror doi vectori care sunt tangenți la suprafață în acel punct [34]. Deoarece derivatele parțiale în raport cu u şi v sunt doi vectori tangenți, vectorul normal este:

$$n(u,v) = \frac{\frac{\partial}{\partial u} b^{m,n}(u,v) \times \frac{\partial}{\partial v} b^{m,n}(u,v)}{\left\| \frac{\partial}{\partial u} b^{m,n}(u,v) \times \frac{\partial}{\partial v} b^{m,n}(u,v) \right\|} \tag{3.3.16}$$

Pentru cele 4 colțuri ale peticului, derivatele parțiale implicate în calcule sunt simple diferențe ale punctelor de graniță. Astfel:

$$n(0,0) = \frac{\Delta^{1,0} b_{0,0} \times \Delta^{0,1} b_{0,0}}{\left\| \Delta^{1,0} b_{0,0} \times \Delta^{0,1} b_{0,0} \right\|} \tag{3.3.17}$$

Vectorul normal într-unul din colțuri (de exemplu $b_{0,0}$) este nedefinit dacă $\Delta^{1,0} b_{0,0}$ şi $\Delta^{0,1} b_{0,0}$ sunt liniar dependenți: în acest caz formula de mai sus va degenera într-o expresie de forma 0/0. Colțul corespunzător al peticului este denumit degenerat. Două cazuri deosebite pot conduce la o asemenea situație.

În primul exemplu întreaga frontieră este redusă la un singur punct $b_{0,0} = b_{1,0} = \ldots = b_{m,0} = c$. Vectorul normal în acest punct poate, sau nu, să fie definit. Se consideră tangentele la liniile isoparametrice u, evaluate pentru $v = 0$. Aceste tangente trebuie să fie perpendiculare pe vectorul normal, dacă acesta există. Deci o condiție pentru existența vectorului normal în punctul c este ca toate derivatele parțiale în raport cu v

evaluate la $v=0$ să fie coplanare. Acest lucru este echivalent cu faptul că punctele $b_{0,1}, b_{1,1}, \ldots, b_{m,1}$ şi c să fie coplanare.

O altă posibilitate de a crea petice degenerate este ca cele două derivate parțiale ale unui colț să fie coliniare. Pentru punctul $P(0,0)$ acest lucru înseamnă că punctele $b_{1,0}, b_{0,1}, b_{0,0}$ sunt coliniare. Dacă normala în $b_{0,0}$ este definită conduce la faptul că punctul $b_{1,1}$ nu este coliniar cu celelalte 3 puncte. Planul tangent în punctul $b_{0,0}$ este planul ce conține aceste puncte coplanare. Normala în $b_{0,0}$ este perpendiculară pe plan.

Vectorii twist

Twist-ul unei suprafețe este dat de derivata parțială mixtă $\partial^2 / \partial u \partial v$. Ținând cont de formulele derivatelor parțiale, suprafața twist a pânzei $b^{m,n}$ este o suprafață Bézier de grad $(m-1, n-1)$, iar coeficienții săi au forma $mn\Delta^{1,1}b_{i,j}$ [11][12]. Interpretarea geometrică a acestor coeficienți este simplă Figura 3.3.3 Punctul $p_{i,j}$ din figură este cel de-al patrulea punct al paralelogramului definit de $b_{i,j}, b_{i+1,j}, b_{i,j+1}$. Punctul este definit de relația:

$$p_{i,j} - b_{i+1,j} = b_{i,j+1} - b_{i,j} \tag{3.3.18}$$

Deoarece $\Delta^{1,1}b_{i,j} = (b_{i+1,j+1} - b_{i+1,j}) - (b_{i,j+1} - b_{i,j})$ rezultă:

$$\Delta^{1,1}b_{i,j} = b_{i+1,j+1} - p_{i,j} \tag{3.3.19}$$

Astfel, termenul $\Delta^{1,1}b_{i,j}$ măsoară deviația fiecărui subpatrulater a hărții Bézier de la forma de paralelogram.

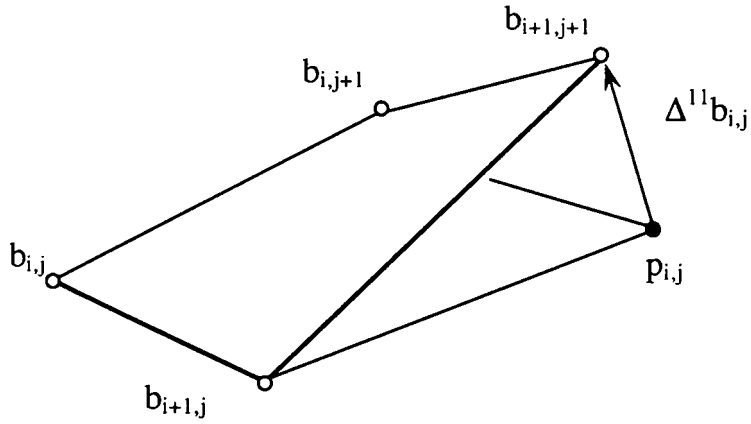

Fig. 3.3.3 Twist: interpretarea geometrică a coeficienților twist

Valorile twistului în cele patru colțuri ale peticului determină deviația respectivelor colțuri ale rețelei de control de la forma de paralelogram. Ca exemplu:

$$\frac{\partial^2}{\partial u \partial v} b^{m,n}(0,0) = mn\Delta^{1,1}b_{00} \tag{3.3.20}$$

Acest vector twist este măsura deviației punctului $b_{1,1}$ de la planul tangent în $b_{0,0}$.

O clasă deosebită de suprafețe este obținută dacă toate subpatrulaterele $b_{i,j}, b_{i+1,j}, b_{i,j+1}, b_{i+1,j+1}$ sunt paralelograme. În acest caz vectorul twist dispare peste tot. O astfel de suprafață este denumită *suprafață translațională.*

3.3.4 Racordarea pânzelor dreptunghiulare Bézier

Suprafețele mari şi cu forme complexe sunt generate pornind de la hărți de control ce conțin un număr foarte mare de puncte. Proiectarea lor cu ajutorul pânzelor Bézier conduce la construcția unor petice de grad mare care, la fel ca şi în cazul curbelor, tind să deformeze suprafața. Reprezentarea structurilor complexe necesită folosirea unor suprafețe compuse alcătuite din mai multe petice conectate între ele cu respectarea condițiilor de continuitate de-a lungul frontierelor. Gradul suprafeței este dat de cel al peticelor sale, fiind independent de numărul de puncte ale poliedrului de control pe porțiuni generator. Astfel, suprafețele Bézier bicubice pe porțiuni permit generarea unor suprafețe oricât de complexe.

Racordarea peticelor de suprafață este o problemă importantă în construcția unei suprafețe complexe pentru că ea influențează calitatea suprafeței. În general, la racordarea a două petice se urmăreşte realizarea continuității de poziție şi a planului tangent [26][74].

Continuitatea de poziție implică coincidența celor două petice de-a lungul frontierei comune, fie în întregime, fie doar pe o porțiune. Se consideră două petice $s_1(u,v)$ şi $s_2(u,v)$ definite pe domeniile $[u_{I-1},u_I]\times[u_J,u_{J+1}]$ şi respectiv $[u_I,u_{I+1}]\times[u_J,u_{J+1}]$, ce aparțin suprafeței compuse $s = s_1 \cup s_2$. Continuitate de poziție a peticelor este dată de relația $s_1(u_I,v) = s_2(u_I,v)$, $\forall v \in [u_J,u_{J+1}]$. Curba de frontieră comună este dată prin intermediul a două parametrizări identice. Faptul că cele două petice nu au acelaşi grad pe direcția v nu constituie o problemă deoarece se poate realiza mărirea gradului peticului mai mic, astfel încât peticele să aibă acelaşi grad relativ la v.

Continuitatea planului tangent într-un punct al frontierei, înseamnă coincidența planului tangent al peticului s_1 cu planul tangent al peticului s_2 calculate în punctul respectiv.

Pentru suprafețe se pot defini două clase de continuitate: continuitate C^r ce reprezintă o dezvoltare a tipurilor de continuitate descrise la curbe, şi continuitatea de tip G^1.

Continuitatea C^r a peticelor Bézier

Peticele considerate, $s_1(u,v)$ şi $s_2(u,v)$, sunt de r ori continuu diferențiabile de-a lungul curbei de frontieră comune $s_1(u_I,v) = s_2(u_I v)$ dacă toate derivatele parțiale în raport cu u până la ordinul r satisfac relația:

$$\frac{\delta^r}{\delta u^r} s_1(u,v)\Big|_{u=u_I} = \frac{\delta^r}{\delta u^r} s_2(u,v)\Big|_{u=u_I} \qquad (3.3.21)$$

Cele două petice sunt date în forma Bézier [20]. Rețeaua punctelor de control a peticului $s_1(u,v)$ este $\{b_{i,j}\}; 0 \le i \le m, 0 \le j \le n$, iar pentru peticul $s_2(u,v)$ $\{b_{i,j}\}; m \le i \le m+p, 0 \le j \le n$. Coincidența punctelor de control aflate pe ultima linie

respectiv prima linie a rețelelor de control indică continuitatea de poziție a celor două petice. Valoarea derivatelor de-a lungul frontierei este dată de relația 7.17 Această formulă este scrisă în coordonate locale. Pentru a se realiza translația la coordonate globale, se utilizează acelaşi algoritm ca şi în cazul compunerii curbelor. Relația dintre derivatele parțiale ale celor două petice devine:

$$\left(\frac{1}{\Delta_{l-1}}\right)^r \sum_{j=0}^{n} \Delta^{r,0} b_{m-r,j} B_j^n(v) = \left(\frac{1}{\Delta_l}\right)^r \sum_{j=0}^{n} \Delta^{r,0} b_{m,j} B_j^n(v) \quad \Delta_l = u_{l+1} - u_l \tag{3.3.22}$$

Deoarece polinoamele Bernstein sunt liniar independente relația se poate reduce la compararea coeficienților:

$$\left(\frac{1}{\Delta_{l-1}}\right)^r \Delta^{r,0} b_{m-r,j} B_j^n(v) = \left(\frac{1}{\Delta_l}\right)^r \Delta^{r,0} b_{m,j} \quad j = 0,....,n \tag{3.3.23}$$

Formula reprezintă condiția de continuitate C^r pentru curbe Bézier [16][56]. Relația poate fi aplicată tuturor celor $n+1$ rânduri ale rețelei Bézier compuse. Rezultă astfel definiția racordului de clasă C^r pentru suprafețe Bézier compuse: două petice adiacente sunt C^r continue de-a lungul frontierei lor comune dacă şi numai dacă toate rândurile rețelei punctelor de control pot fi interpretate ca poligoane a curbelor Bézier C^r pe porțiuni. S-a obținut din nou reducerea problemei suprafeței la câteva probleme de curbe. Condițiile de continuitate se aplică în mod identic şi pe direcția v.

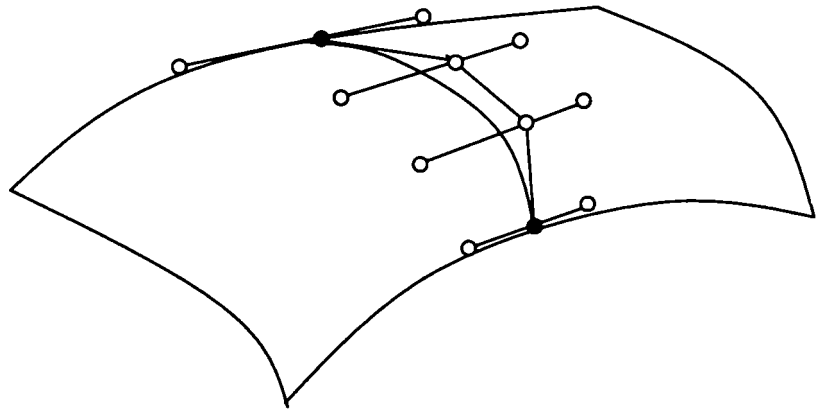

Fig. 3.3.4 Reprezentarea punctelor de control a peticelor Bézier pentru condiția de continuitate C^1

Cazul de racord C^1 este ilustrat în figura 3.3.4. Condiția de continuitate conduce la faptul că pentru orice j, poligonul format din $b_{0j},\ldots,b_{mj},\ldots,b_{m+pj}$ este poligonul de control pentru o curbă Bézier C^1 pe porțiuni. Pentru aceasta, punctele b_{m-1j},b_{mj},b_{m+1j} trebuie să fie coliniare iar segmentul $b_{m-1j}b_{m+1j}$ este împărțit de punctul b_{mj} în raportul $\Delta_l : \Delta_{l+1}$. Acest raport trebuie să fie acelaşi pentru toate rândurile rețelei. Simpla coliniaritate nu este suficientă, suprafețele compuse care au punctele b_{m-1j},b_{mj},b_{m+1j} coliniare dar nu în acest raport nu vor fi de tip C^1. Mai mult, aceste suprafețe nu vor avea un plan tangent continuu. Rigiditatea condiției C^1 poate constitui o problemă în proiectarea suprafețelor care constau

dintr-o rețea de petice Bézier, sau a peticelor polinomiale pe porțiuni date în altă reprezentare.

Continuitatea G^1 a peticelor Bézier

O modalitate mai bună de rezolvare a problemei conectării peticelor unei suprafețe o constituie relaxarea condițiilor impuse de continuitatea C^1, ceea ce conduce la o continuitate de tip G^1. Două petice ce au o curbă de frontieră comună sunt continue G^1 dacă prezintă un plan tangent ce variază continuu de-a lungul curbei de frontieră. Un aspect important al acestui tip de continuitate este independența sa față de domeniile peticelor conectate. Pentru continuitatea de tip C^1, legătura dintre rangul suprafeței și geometria domeniului este esențială [15][16].

Racordul de clasă G^1 în lungul curbei de frontieră presupune, pe lângă continuitate de poziție asigurată de identitatea punctelor de control aflate pe liniile extreme corespunzătoare ale poliedrelor de control, continuitatea planului tangent la cele două pânze în punctele curbei, ceea ce este echivalent cu coliniaritatea vectorilor normali la cele două petice în fiecare punct al curbei de frontieră. Acest lucru va permite determinarea relației dintre punctele de control ale celor două pânze din liniile "vecine" liniei care generează curba comună. În contextul considerat, curbele coordonate $u = const$ de pe suprafața compusă sunt de clasă C^∞. Curbele coordonate $v = const$ sunt parametrizate:

$$s_v : [u_{I-1}, u_{I+1}] \to E^3,\ s_v(u) = \begin{cases} s_1(u,v) & u \in [u_{I-1}, u_I] \\ s_2(u,v) & u \in [u_I, u_{I+1}] \end{cases} \tag{3.3.24}$$

Vectorul tangent la aceste curbe este $\dot{s}_v(u)$. Curbele sunt de clasă G^1 în $u = u_I$ dacă există funcția α astfel încât $\dot{s}_v(u_I - 0) = \alpha(v)\dot{s}_v(u_I + 0)$ relație ce poate fi rescrisă $\dot{s}_{1u}(u_I, v) = \alpha(v)\dot{s}_{2u}(u_I, v)$. Dacă relația este îndeplinită pentru toate curbele coordonate v, normalele la cele două pânze de-a lungul curbei de frontieră sunt coliniare şi deci suprafața compusă este de tip G^1 [13].

Continuitatea de clasă G^1 a curbelor coordonate v implică coliniaritatea vectorilor $\dot{s}_{1u}(u_I, v)$ şi $\dot{s}_{2u}(u_I, v)$. Aceştia sunt definiți de relațiile:

$$\dot{s}_{1u}(u_I, v) = m\sum_{j=0}^{n}(b_{mj} - b_{m-1\,j})B_j^n(v) \tag{3.3.25}$$

$$\dot{s}_{2u}(u_I, v) = p\sum_{j=0}^{n}(b_{m+1\,j} - b_{mj})B_j^n(v)$$

Vectorii fiind coliniari şi de acelaşi sens, există $\lambda > 0$ astfel încât $\dot{s}_{1u}(u_I, v) = \lambda\dot{s}_{2u}(u_I, v)$. Înlocuind formulele derivatelor în această relație se obține:

$$\sum_{j=0}^{n}(m(b_{mj} - b_{m-1\,j} - \lambda p(b_{m+1\,j} - b_{mj}))B_j^n(v) = 0 \tag{3.3.26}$$

Polinoamele Bernstein fiind liniar independente, relația de mai sus are loc dacă și numai dacă

$$m\left(b_{mj}-b_{m-1j}\right)=\lambda p\left(b_{m+1j}-b_{mj}\right),\quad \forall j=\overline{0,n} \tag{3.3.27}$$

relație care poate fi rescrisă:

$$b_{mj}=\frac{m}{m+\lambda p}b_{m-1j}+\frac{\lambda p}{m+\lambda p}b_{m+1j} \tag{3.3.28}$$

Formula exprimă coliniaritatea punctelor b_{m-1j},b_{mj},b_{m+1j} și independența de j a raportului în care b_{mj} divide segmentul $b_{m-1j}b_{m+1j}$. Astfel, dacă pânzele s_1 și s_2 au racord de clasă G^1 în lungul curbei de frontieră $s_1(u_1,v)=s_2(u_1,v)$, și curbele coordonate v sunt de clasă G^1 atunci linia m a rețelei ce definește s_1 coincide cu linia 0 a rețelei ce definește s_2, iar punctele din linia 1 a rețelei s_2 se exprimă prin relația:

$$b_{m+1j}=b_{mj}+\mu\frac{m}{p}\left(b_{mj}-b_{m-1j}\right),\quad \forall j=\overline{0,n},\quad \mu=\frac{1}{\lambda} \tag{3.3.29}$$

Relația stabilește raportul în care se află punctele liniilor vecine frontierei comune a celor două petice.

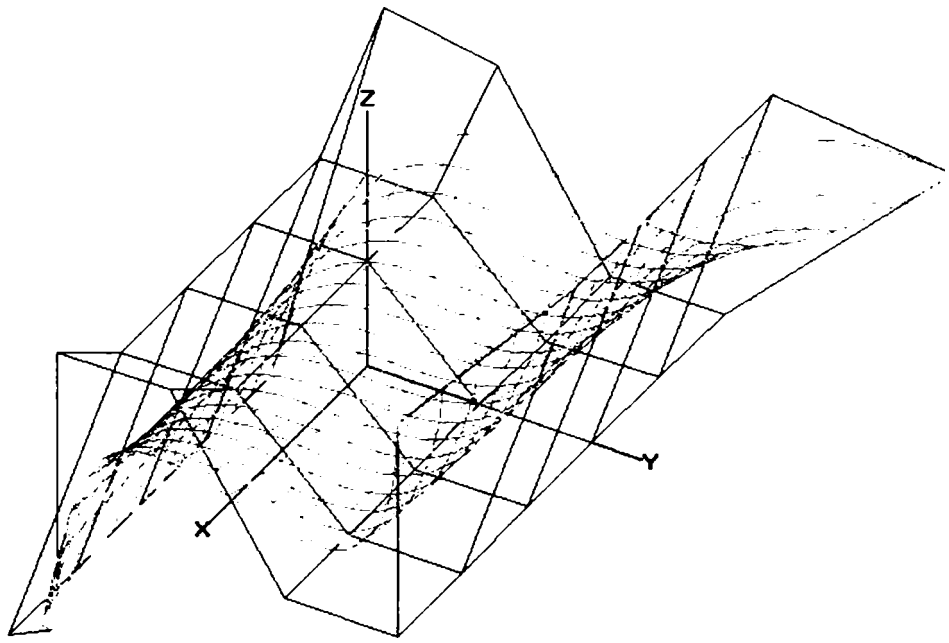

Fig. 3.3.5 Suprafață Bézier compusă și rețeaua sa de puncte de control

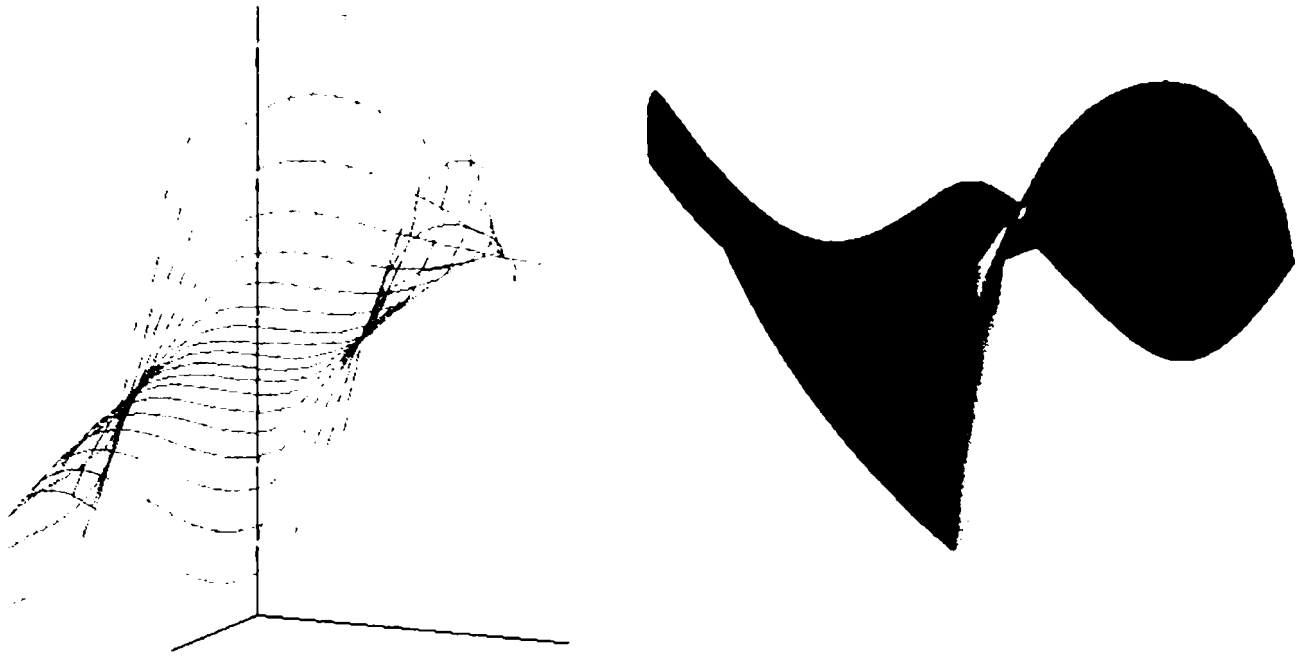

Fig. 3.3.6 Suprafață Bézier compusă și reprezentarea sa folosind librăria OpenGL

Determinarea liniilor vecine pentru un nou petic se realizează cu funcția membră a clasei CMeSurfacePatch, RacordPatch(). Sunt determinate mai întâi peticele vecine noului petic. Pentru fiecare vecin este apelată funcția CMeControlMap::RacordPatch() care generează în mod corespunzător racordului de clasă G^1 punctele de control ale noului petic.

```
void CMeSurfacePatch::RacordPatch(short row, short &col)
{
    CMeSurfacePatch *patchInf = NULL, *patchSup = NULL, *patchLeft = NULL;
    col = 0;
    // identificarea liniei pe care se dorește inserarea peticului
    CMeSurfacePatchLine *pSurfacePatchLine = parentSurface->GetPatchLine(row);
    // determinarea coloanei pe care se inserează peticul: inserarea se face pe ultima poziție
    if(pSurfacePatchLine)
        col = pSurfacePatchLine->surfacePatchLine.GetCount();
    // determinarea peticului de pe rândul inferior
    if(row > 0)
        patchInf = parentSurface->GetPatch(row - 1, col);
    // determinarea peticului superior
    patchSup = parentSurface->GetPatch(row + 1, col);
    // determinarea peticului din stânga
    if(col)
        patchLeft = parentSurface->GetPatch(row, col-1);

    // generarea hărții de control a peticului în funcție de conexiunea sa cu peticele vecine
    if(patchInf)
        controlMap.RacordPatch(patchInf->controlMap, 0);
    if(patchSup)
        controlMap.RacordPatch(patchSup->controlMap, 1);
    if(patchLeft)
        controlMap.RacordPatch(patchLeft->controlMap, 2);
    for(short i = 0; i < controlMap.nr_row; i++)
        for(short j = 0; j < controlMap.nr_col; j++)
            CreatePersp(controlMap.controlPoints[i][j]);
}

void CMeControlMap::RacordPatch(CMeControlMap &controlMap, short mode)
```

```
{
    short nrCol, nrLine;
    double beta = 1.0;

    if(mode == 0)
    {
        // racordul se face pe prima linie a peticului cu ultima linie a peticului anterior
        // construcția primei linii
        nrLine = controlMap.nr_row - 1;
        for(short i = 0; i < nr_col, i < controlMap.nr_col; i++)
            controlPoints[0][i] = controlMap.controlPoints[nrLine][i];
        // construcția celei de a doua linii
        for(i = 0; i < nr_col, i < controlMap.nr_col; i++)
            controlPoints[1][i] = controlMap.controlPoints[nrLine][i] +
          (controlMap.controlPoints[nrLine][i] – controlMap.controlPoints[nrLine-1][i])*beta;
    }
    else if(mode == 1)
    {
        // racordul se face pe ultima linie a peticului cu prima linie a peticului următor
        nrLine = nr_row - 1;
        // construcția primei linii
        for(short i = 0; i < nr_col, i < controlMap.nr_col; i++)
            controlPoints[nrLine][i] = controlMap.controlPoints[0][i];
        // construcția celei de a doua linii
        for(i = 0; i < nr_col, i < controlMap.nr_col; i++)
            controlPoints[nrLine-1][i] = controlMap.controlPoints[0][i] +
                        (controlMap.controlPoints[0][i] - controlMap.controlPoints[1][i])*beta;
    }
    else
    {
        // racordul se face pe prima coloana a peticului cu ultima coloana a
        // peticului din stânga
        nrCol = controlMap.nr_col - 1;
        // construcția primei linii
        for(short i = 0; i < nr_row, i < controlMap.nr_row; i++)
            controlPoints[i][0] = controlMap.controlPoints[i][nrCol];
        // construcția celei de a doua linii
        for(i = 0; i < nr_row, i < controlMap.nr_row; i++)
            controlPoints[i][1] = controlMap.controlPoints[i][nrCol] +
           (controlMap.controlPoints[i][nrCol] - controlMap.controlPoints[i][nrCol-1])*beta;
    }
}
```

Formele complexe din cadrul laboratorului virtual au fost modelate cu ajutorul suprafețelor sintetice. Datorită proprietăților lor, suprafețele Bézier produs tensorial au permis proiectarea cu uşurință a părților curbe ale obiectelor. Metoda a fost folosită în general pentru generarea suprafețelor ce prezintă curburi mai line şi care nu au necesitat un număr mare de petice bicubice de suprafață. Un rol important în procesul de modelare este atribuit construirii poliedrului de control al întregii suprafețe. Pentru această metodă de proiectare, harta punctelor de control este generată cu uşurință deoarece numărul de puncte ale fiecărui petic este relativ mic, fiind utilizate petice bicubice, iar punctele liniilor sau coloanelor situate în vecinătatea altui petic sunt generate în mod automat pentru a se respecta condițiile de continuitate al suprafeței. Fiecare suprafață creată necesită ajustări succesive ale punctelor poliedrului de control pentru a se obține forma dorită. Această operație este eficientă în cazul suprafețelor compuse deoarece permite modificarea locală a formei fără a se afecta întreaga suprafață. Odată forma realizată într-o anumită parte a

suprafeței se poate trece la modelarea altei porțiuni a suprafeței fără a influența zonele finalizate.

În cadrul modelului laboratorului virtual suprafețele Bézier produs tensorial au fost folosite la modelarea geometrică a calculatoarelor, monitoarelor, scaunelor, a patului, precum și a roboților.

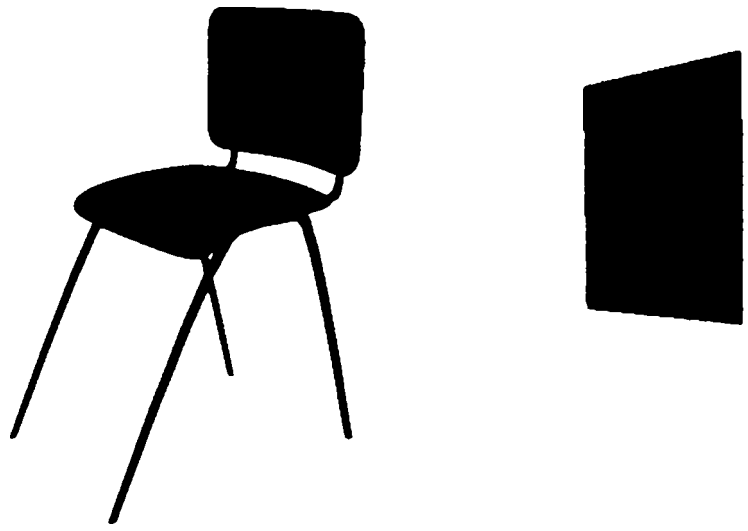

Fig. 3.3.7. Reprezentarea virtuală a monitorului calculatorului și a unui scaun

Modelul geometric al robotului cuprinde ca majoritatea elementelor complexe o varietate mare de tipuri de suprafețe pentru a se obține o formă cât mai reală a acestuia. Alegerea tipurilor de suprafețe optime pentru modelare se face în funcție de caracteristicile geometrice ale corpului. Astfel, robotul a fost modelat atât prin suprafețe produs tensorial cât și prin alte tipuri de suprafețe.

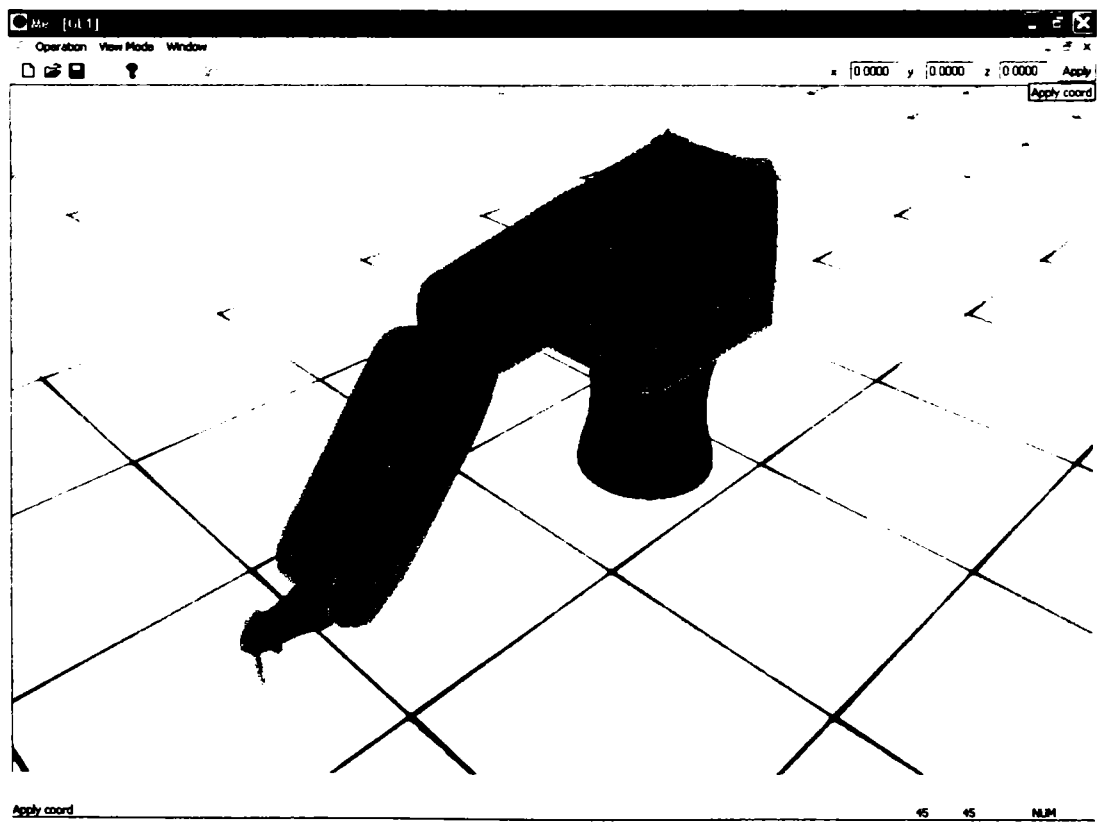

Fig.3.3.8 Modelul geometric virtual al unuia din roboții mobili

3.4 Suprafețe bicubice Hermite

3.4.1 Forma canonică a suprafețelor Hermite

Aplicarea metodei produsului tensorial domeniului curbelor Hermite a condus la dezvoltarea unui nou tip de suprafețe, peticele Hermite. La fel ca şi în cazul curbelor, suprafețele Hermite sunt determinate de secvența de puncte de control şi de proprietățile de derivabilitate ale suprafeței în aceste puncte (derivatele parțiale şi derivatele mixte).

Un petic de suprafață bicubic Hermite va realiza conectarea unei rețele dreptunghiulare de patru puncte. Fiecare din puncte, este definit, pe lângă vectorul său de poziție, de derivatele parțiale în raport cu cele două direcții parametrice u şi v ale suprafeței precum şi de vectorul său de twist. Cele 16 condiții vectoriale, 4 puncte, 8 vectori tangenți şi 4 vectori de twist, caracterizează geometric complet suprafața. Relația peticului scrisă în baza canonică este:

$$P(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}C_{ij}u^{i}v^{j},\ \ u,v\in[0,1] \tag{3.4.1}$$

Forma matriceală a formulei este:

$$P(u,v)=U[C]V \tag{3.4.2}$$
$$U=[u^3,u^2,u,1],\ \ V=[v^3,v^2,v,1]^T$$

Matricea $[C]$ se numeşte *matricea coeficienților algebrici* [26][61], deoarece punctele c_{ij} nu au o semnificație geometrică. Similar curbelor Hermite, calculul coeficienților c_{ij} se face prin aplicarea condițiilor de frontieră $u=0$, $u=1$ şi $v=0$, $v=1$. Rezolvând succesiv relația pentru aceste valori se obține:

$$P(u,v)=U[M_H][B][M_H]^T V,\quad u,v\in[0,1] \tag{3.4.3}$$

$[M_H]$ este matricea Hermite determinată la curbele interpolatoare cubice, iar $[B]$ reprezintă *matricea condițiilor geometrice de frontieră*:

$$[B]=\begin{bmatrix} P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1)\\ P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1)\\ P_u(0,0) & P_u(0,1) & P_{uv}(0,0) & P_{uv}(0,1)\\ P_u(1,0) & P_u(1,1) & P_{uv}(1,0) & P_{uv}(1,1) \end{bmatrix} \tag{3.4.4}$$

Matricea [B] este partiționată în 2×2 submatrici, pentru a indica gruparea condițiilor similare de frontieră. Ea poate să fie de exemplu scrisă:

$$[B]=\begin{bmatrix} [P] & [P_v]\\ [P_u] & [P_{uv}] \end{bmatrix} \tag{3.4.5}$$

$[P]$, $[P_u]$, $[P_v]$, $[P_{uv}]$ sunt submatricile punctelor de colț, a vectorilor de colț tangenți pe

direcția u, a vectorilor tangenți pe direcția v, și respectiv a vectorilor de twist în colțuri [66]. Vectorii tangenți și de twist în orice punct (u,v) al suprafeței sunt :

$$\begin{aligned} P_u(u,v) &= U[M_H]^u[B][M_H]^T V \\ P_v(u,v) &= U[M_H][B][M_H]^{vT} V \\ P_{uv}(u,v) &= U[M_H]^u[B][M_H]^{vT} V \end{aligned} \tag{3.4.6}$$

$[M_H]^u$ și $[M_H]^v$ sunt matricele Hermite derivate întâlnite de asemenea la curbele Hermite.

Pentru determinarea influenței vectorilor de poziție și a vectorilor tangenți asupra formei suprafeței relațiile pot fi rescrise sub următoarea formă:

$$\begin{aligned} &P(u,v) = F(u)[B]F(v)^T, \quad P_u(u,v) = G(u)[B]F(v)^T \\ &P_v(u,v) = F(u)[B]G(v)^T, \quad P_{uv}(u,v) = G(u)[B]G(v)^T \\ &F(x) = [F_1(x)\ F_2(x)\ F_3(x)\ F_4(x)] \\ &G(x) = [G_1(x)\ G_2(x)\ G_3(x)\ G_4(x)], \quad x = u,v \end{aligned} \tag{3.4.7}$$

Funcțiile polinomiale $F_i(x)$ și $G_i(x)$ sunt date de relațiile

$$\begin{aligned} F_1(x) &= 2x^3 - 3x^2 + 1 \\ F_2(x) &= -2x^3 + 3x^2 \\ F_3(x) &= x^3 - 2x^2 + x \\ F_4(x) &= x^3 - x^2 \end{aligned} \quad \text{și} \quad \begin{aligned} G_1(x) &= 6x^2 - 6x \\ G_2(x) &= -6x^2 + 6x \\ G_3(x) &= 3x^2 - 4x + 1 \\ G_4(x) &= 3x^2 - 2x \end{aligned} \tag{3.4.8}$$

Pentru muchiile $u = 0$ și $u = 1$, relațiile 3.4.7 devin:

$$\begin{bmatrix} P(0,v) \\ P(1,v) \\ P_u(0.v) \\ P_u(1,v) \end{bmatrix} = [B] \begin{bmatrix} F_1(v) \\ F_2(v) \\ F_3(v) \\ F_4(v) \end{bmatrix} \tag{3.4.9}$$

Un rezultat similar se obține pentru muchiile $v = 0$ și $v = 1$. Relația indică faptul că vectorii v-tangenți în colțuri și vectorii de twist afectează vectorii de poziție și vectorii tangenți de-a lungul muchiilor $u = 0$ și $u = 1$ cu excepția punctelor $v = 0$ și $v = 1$, pentru care funcțiile F_3 și F_4 sunt ambele zero. Un caz special al suprafețelor bicubice îl constituie peticele Ferguson (numite și petice F-suprafață) care sunt caracterizate de vectori de twist nuli în extremitățile peticului. Matricea condițiilor de frontieră pentru peticul F-suprafață devine:

$$[B] = \begin{bmatrix} P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1) \\ P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1) \\ P_u(0,0) & P_u(0,1) & 0 & 0 \\ P_u(1,0) & P_u(1,1) & 0 & 0 \end{bmatrix} \tag{3.4.10}$$

Această suprafață specială este utilă în aplicații de proiectare şi planificare. Vectorii tangenți în punctele de colț pot fi aproximați în funcție de poziția acestor colțuri, folosind direcția şi lungimea liniilor de coardă care unesc colțurile. Deci proiectantul nu trebuie să introducă informații despre vectorii tangenți, iar calculele necesare pentru determinarea parametrilor suprafeței sunt simplificate. Aceasta este util, de exemplu, atunci când o maşină unealtă este folosită pentru netezirea unei suprafeței.

Suprafețele bicubice Hermite păstrează aceleaşi caracteristici ca şi cubicele spline. Controlul suprafeței rezultate este global şi nu este intuitiv pe baza datelor de intrare. În plus, cerința ca vectorii de twist şi tangenți să fie specificați ca date de intrare nu corespunde foarte bine într-un mediu de proiectare deoarece simțul intuitiv pentru astfel de date nu este suficient de dezvoltat [57][58].

3.4.2 Reprezentarea în baza Hermite

Un petic bicubic în forma Hermite este dat de relația:

$$P(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}h_{ij}H_i^3(u)H_j^3(v);\quad 0\le u,v\le 1 \tag{3.4.11}$$

H_i^3 sunt funcțiile cubice Hermite. Coeficienții matricei h_{ij} caracterizează geometric suprafața. Determinarea acestor coeficienți se realizează prin reducerea problemei la cazul univariant, al curbelor Hermite cubice şi aplicare condițiilor de frontieră. Curba cubică $v=v_0$ de pe suprafață, scrisă în baza Hermite este:

$$P(u,v_0)=P(0,v_0)H_0^3(u)+P(1,v_0)H_1^3(u)+P_u(0,v_0)H_2^3(u)+P_u(1,v_0)H_3^3(u) \tag{3.4.12}$$

$P(0,v_0)$ şi $P(1,v_0)$ sunt punctele de capăt ale curbei iar $P_u(0,v_0),P(1,v_0)$ sunt tangentele la curbă în aceste puncte. Dacă relația se generalizează pentru $v_0=v$ arbitrar, iar apoi se derivează în raport cu v rezultă:

$$P_v(u,v)=P_v(0,v)H_0^3(u)+P_v(1,v)H_1^3(u)+P_{uv}(0,v)H_2^3(u)+P_{uv}(1,v)H_3^3(u) \tag{3.4.13}$$

În mod similar se obțin formulele pentru curbele Hermite $u=const$ de pe suprafață:

$$\begin{aligned}P(u,v)&=P(u,0)H_0^3(u)+P(u,1)H_1^3(u)+P_v(u,0)H_2^3(u)+P_v(u,1)H_3^3(u)\\P_u(u,v)&=P_u(u,0)H_0^3(u)+P_u(u,1)H_1^3(u)+P_{uv}(u,0)H_2^3(u)+P_{uv}(u,1)H_3^3(u)\end{aligned} \tag{3.4.14}$$

Matricial relația $P(u,v)$ poate fi scrisă:

$$P(u,v)=\begin{bmatrix}P(u,0) & P(u,1) & P_v(u,0) & P_v(u,1)\end{bmatrix}\begin{bmatrix}H_0^3(v)\\H_1^3(v)\\H_2^3(v)\\H_3^3(v)\end{bmatrix} \tag{3.4.15}$$

Coeficienții primei matrice pot fi rescrişi prin aplicarea relațiilor curbelor isoparametrice v (3.4.12) şi (3.4.13), pentru cazurile de frontieră $v=0$, $v=1$. Astfel $P(u,0)$ este dat de:

$$P(u,0)=\begin{bmatrix}H_0^3(u) & H_1^3(u) & H_2^3(u) & H_3^3(u)\end{bmatrix}\begin{bmatrix}P(0,0)\\P(1,0)\\P_u(0,0)\\P_u(1,0)\end{bmatrix} \tag{3.4.16}$$

Similar din relația (8.14) se obține formulele pentru $P(u,1)$, $P_v(u,0)$, $P_v(u,1)$. Înlocuind aceşti termeni în (8.15) rezultă sistemul:

$$P(u,v)=\begin{bmatrix}H_0^3(u) & H_1^3(u) & H_2^3(u) & H_3^3(u)\end{bmatrix}\begin{bmatrix}P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1)\\P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1)\\P_u(0,0) & P_u(0,1) & P_{uv}(0,0) & P_{uv}(0,1)\\P_u(1,0) & P_u(1,1) & P_{uv}(1,0) & P_{uv}(1,1)\end{bmatrix}\begin{bmatrix}H_0^3(v)\\H_1^3(v)\\H_2^3(v)\\H_3^3(v)\end{bmatrix} \tag{3.4.17}$$

care reprezintă forma matricială a unei suprafețe Hermite [26]. Matricea coeficienților h_{ij} este determinată de datele inițiale ale pânzei, date ce sunt interpolate de către aceasta.

Determinarea curbei isoparametrice $u=u_0$ de pe suprafața Hermite:

$$P(u_0,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}h_{ij}H_i^3(u_0)H_j^3(v) \tag{3.4.18}$$

se realizează prin calculul mai întâi a patru puncte $d_j(u_0)$ aflate pe curbele Hermite:

$$d_j(u_0)=\sum_{i=0}^{3}h_{ij}H_i^3(u_0),\qquad j=0,1,2,3 \tag{3.4.19}$$

Aşa cum a fost descris la interpolarea Hermite cubică, evaluarea acestor puncte se face prin determinarea punctelor Bézier din elementele ce definesc curba şi apoi aplicarea algoritmului de Casteljau poligonului de control rezultat [34][35][61].

Curba ordonată $u=u_0$ scrisă în raport cu punctele d_i calculate este:

$$P(u_0,v)=\sum_{j=0}^{3}d_jH_j^3(v),\qquad v=\overline{0,1} \tag{3.4.20}$$

Relația descrie o curbă Hermite cubică de parametru v, a cărei formă se obține prin calculul punctelor Bézier din punctele d_j, şi aplicare algoritmului de Casteljau în raport cu v.

Prin aplicarea unui algoritm identic se generează curbele isoparametrice $v=const$ ale peticului Hermite.

3.4.3 Racordul de clasă G^1 a două suprafețe Hermite bicubice

În contextul suprafețelor compuse, se definește o rețea de petice bicubice. Formula peticului (I,J) din rețea este:

$$P(u,v)=\sum_{i=0}^{3}\sum_{j=0}^{3}h_{ij}H_i^3(s)H_j^3(t);\quad 0\le s,t\le 1 \tag{3.4.21}$$

În această relație s şi t sunt coordonate locale ale domeniilor $[u_I,u_{I+1}]$ respectiv $[v_J,v_{J+1}]$. Coeficienții matricei h_{ij} se modifică, ținând cont de reprezentarea în coordonate globale:

$$h_{ij}=\begin{bmatrix} P(0,0) & P(0,1) & \Delta_J P_v(0,0) & \Delta_J P_v(0,1)\\ P(1,0) & P(1,1) & \Delta_J P_v(1,0) & \Delta_J P_v(1,1)\\ \Delta_I P_u(0,0) & \Delta_I P_u(0,1) & \Delta_I\Delta_J P_{uv}(0,0) & \Delta_I\Delta_J P_{uv}(0,1)\\ \Delta_I P_u(1,0) & \Delta_I P_u(1,1) & \Delta_I\Delta_J P_{uv}(1,0) & \Delta_I\Delta_J P_{uv}(1,1) \end{bmatrix} \tag{3.4.22}$$

Condițiile de continuitatea G^1 între două petice bicubice Hermite vecine $P_1(u,v)$ şi $P_2(u,v)$ sunt:

$$\begin{aligned} &P_1(1,v)=P_2(0,v)\\ &P_{1u}(1,v)=KP_{2u}(0,v),\quad K=const. \end{aligned} \tag{3.4.23}$$

Prima relație constituie condiția de continuitate C^0 (de poziție) a celor două pânze [12][30]. Curba ordonată $u=1$ a primului petic este identică cu ordonata cubică $u=0$ a celui de al doilea petic. Acest lucru conduce la următoarele relații între coeficienții geometrici ai celor două pânze:

$$\begin{aligned} &P_1(1,0)=P_2(0,0),\quad P_1(1,1)=P_2(0,1)\\ &P_{1v}(1,0)=P_{2v}(0,0),\quad P_{1v}(1,1)=P_{2v}(0,1) \end{aligned} \tag{3.4.24}$$

A doua relație determină raporturile care se stabilesc între curbele isoparametrice ale celor două petice care pornesc de pe frontiera comună [91]. Relația poate fi interpretată ca şi un amestec (blending) de un număr finit de segmente spline cubice de pe petice, fiecare corespunzând unei valori particulare a lui v. Curba ordonată $v=v_0$ a suprafeței compuse:

$$P(u,v_0)=\begin{cases} P_1(u,v_0) & u\in[u_I,u_{I+1}]\\ P_2(u,v_0) & u\in[u_{I+1},u_{I+2}] \end{cases} \tag{3.4.25}$$

trebuie să prezinte o continuitate G^1 de-a lungul întregului domeniu pentru ca suprafața compusă să fie de clasă G^1. Acest lucru implică faptul ca tangentele curbei în punctul u_I să aibă aceeaşi direcție şi să se afle într-un raport de proporționalitate. Relațiile dintre

coeficienții geometrici ai celor două pânze vor fi:

$$\begin{aligned} &P_{1u}(1,0)=KP_{2u}(0,0), \quad P_{1u}(1,1)=KP_{2u}(0,1) \\ &P_{1uv}(1,0)=KP_{2uv}(0,0) \quad P_{1uv}(1,1)=KP_{2uv}(0,1) \end{aligned} \tag{3.4.26}$$

Peticele bicubice în forma Hermite pot fi utilizate pentru interpolarea spline bicubică, întru-un mod asemănător cu forma B-spline. Mai întâi sunt rezolvate $(m+1)$ probleme de interpolare univariante în scopul de a se obține derivatele parțiale în raport cu v în punctele de interpolat. Apoi, se rezolvă $(n+1)$ probleme de interpolare pentru a obține derivatele u-parțiale, care vor fi parcurse rând cu rând pentru a rezulta vectorii twist (sau sunt luate derivatele v-parțiale şi rezolvate coloană cu coloană pentru a determina vectorii twist). Astfel, procesul de interpolare Hermite constă din trei etape față de interpolarea B-spline care necesită doar două.

3.5 Suprafețe B-spline. Interpolare spline

3.5.1 Suprafețe B-spline

Pânzele Bézier produs tensorial au fost dezvoltate la începutul anilor 1960. Tot în aceeaşi perioadă s-a pus şi problema proiectării suprafețelor compuse. Una din primele lucrări din domeniu a fost a lui de Boor referitoare la suprafețele spline bicubice în anul 1962. Aproape simultan şi aparent necunoscând munca lui de Boor, J. Ferguson a implementat suprafețele interpolatoare spline bicubice pe porțiuni la Boeing. Metoda sa a fost larg utilizată, cu toate că a prezentat un neajuns important prin folosirea doar a vectorilor twist în colțuri de valoare zero [60].

În secțiunile anterioare au fost prezentate principiile de generare a suprafețelor Bézier şi Hermite. Capitolul descrie suprafețele compuse B-spline şi principiile care stau la baza interpolării spline.

Una dintre cele mai importante clase de suprafețe folosite la proiectarea formelor complexe o reprezintă suprafețele B-spline. Aceeaşi metodă a produsului tensorial dezvoltată la suprafețele Bézier foloseşte acum curbele B-spline pentru a descrie suprafețe. Caracteristicile funcțiilor spline moştenite de către suprafață, stabilitate numerică, flexibilitate, control local, conferă acesteia o valoare specială. Gradul suprafeței este independent de numărul punctelor poliedrului de control, iar continuitatea este menținută automat în cadrul suprafeței în virtutea formei funcțiilor de amestecare. Ca şi rezultat, intersecțiile de suprafețe pot fi uşor manipulate. Secvența punctelor de control este fie aproximată fie interpolată de către suprafața B-spline [17][65].

Utilizând notațiile folosite la curbe, o suprafață parametrică B-spline produs tensorial, definită printr-un tablou $(n+1)\times(m+1)$ de puncte de control, poate fi scrisă:

$$P(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m} d_{ij}N_{i,k}(u)N_{j,l}(v),0\le u\le u_{max},0\le v\le v_{max} \qquad (3.5.1)$$

Construcția suprafeței necesită folosirea unor secvențe de noduri pe direcția u şi respectiv v. Vectorii nodurilor sunt constanți pe cele două direcții, dar nu în mod necesar egali. Pentru curbe, noduri de sfârşit triple înseamnă că primele şi ultimele două puncte de control B-spline sunt de asemenea puncte de control ale poligonului pe porțiuni Bézier, iar curba interpolează punctele de capăt. Acest lucru este adevărat şi în cazul suprafețelor. Punctele de control B-spline d_{ij} pentru care indecşii i sau j sunt egali cu 0 sau 1, sunt puncte de control ale rețelei Bézier pe porțiuni a suprafeței. De asemenea, aceste puncte determină curbele de frontieră şi derivatele de frontieră.

Suprafețele B-spline păstrează aceleaşi caracteristici ca şi curbele B-spline. Avantajul lor major față de suprafețele Bézier este controlul local. Modificarea unor puncte ale poliedrului de control afectează doar regiunea din vecinătatea lor, în rest, forma suprafeței rămânând neschimbată. Suprafețele B-spline compuse de clasă de continuitate C^0 sau C^1 pot fi generate în acelaşi mod ca şi suprafețele alcătuite din petice Bézier.

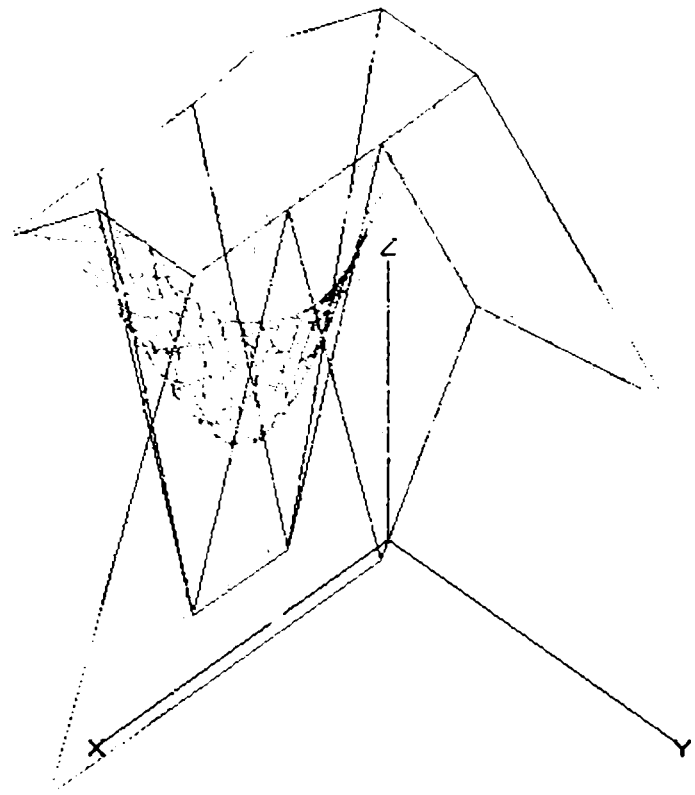

Fig. 3.5.1 Suprafață B-spline reprezentată prin rețeaua de curbe ale suprafeței

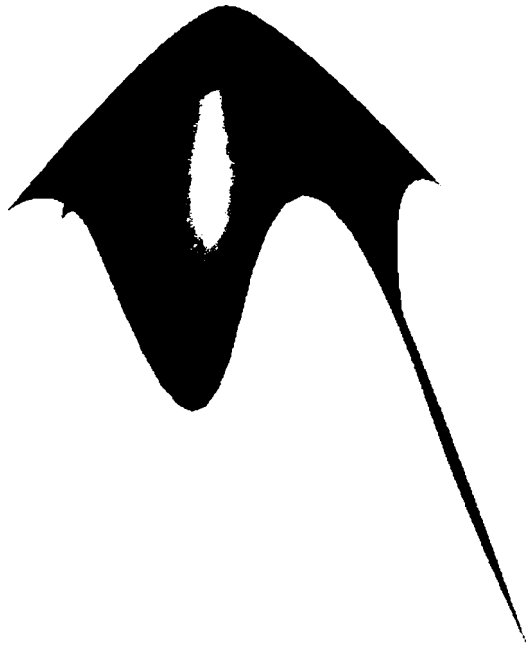

Fig. 3.5.2 Reprezentarea realistă a suprafaței B-spline folosindu-se OpenGL

Fig. 3.5.3 Reprezentarea realistă a suprafaței B-spline folosindu-se Direct3D

Suprafața B-spline urmărește cu mai multă rigurozitate forma rețelei de control față de o suprafață Bézier. Acest lucru poate fi observat comparând figura 3.5.1 a unei suprafețe B-spline cu figura 3.5.4. care reprezintă o suprafață Bézier generată pornind de la aceeași hartă a punctelor de control.

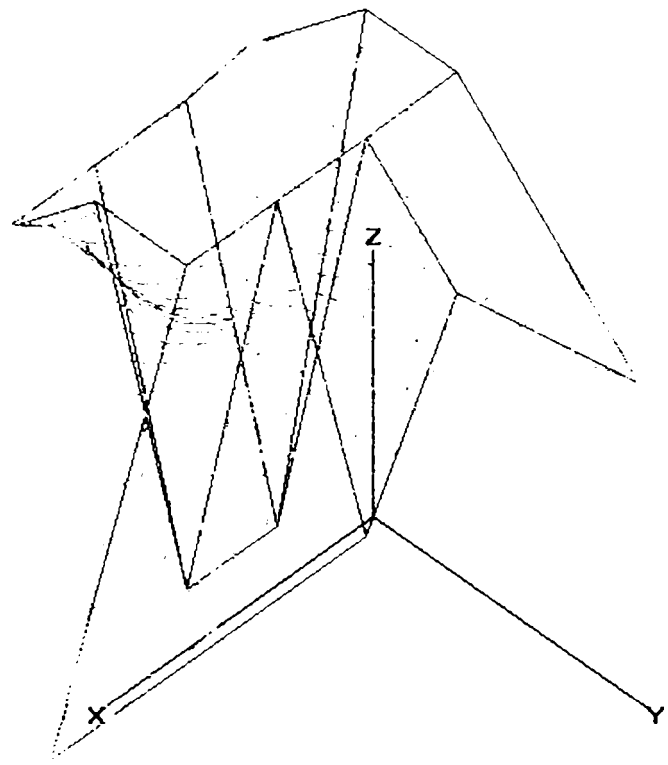

Fig. 3.5.4 Suprafață Bézier generată pornind de la a aceeași rețea

La fel ca și la curbele B-spline cubice pe porțiuni, o suprafață B-spline bicubică poate fi dată ca o colecție de petice bicubice. Acest lucru conduce la problema determinării rețelei de puncte de control Bézier pentru fiecare petic. Rezolvarea problemei este utilă la conversia unei suprafețe B-spline într-o formă Bézier pe porțiuni și de asemenea dacă se dorește evaluarea suprafeței descompunând-o mai întâi în petice bicubice. Soluția se obține, așa cum este în mod uzual la produsele tensoriale, prin transformarea problemei în mai multe probleme de curbe. Formula anterioară poate fi scrisă:

$$x(u,v) = \sum_i N_i^3(v)\left[\sum_j d_{ij} N_j^3(u)\right] \qquad (3.5.2)$$

Din relație se poate observa că suma dintre parantezele drepte descrie o curbă B-spline în variabila *u*. Aceasta poate fi convertită în forma Bézier prin utilizarea uneia din metodele descrise în capitolele referitoare la curbe B-spline. Toate acestea conduc la interpretarea rețelei de control B-spline rând cu rând ca poligoane B-spline univariante și apoi convertirea lor la forma Bézier pe porțiuni. Punctele Bézier astfel obținute, pe fiecare coloană, pot fi interpretate ca poligoane B-spline, care pot fi din nou transformate în format Bézier unul câte unul. Această familie de poligoane Bézier constituie rețeaua Bézier pe porțiuni a suprafeței [18].

Conversia B-spline – Bézier a fost realizată pornind de la rândurile rețelei de control. Un rezultat similar se obține și dacă sunt prelucrate mai întâi coloanele rețelei. Forma Bézier pe porțiuni construită poate fi transformată în orice altă formă polinomială pe

porţiuni, cum ar fi forma monomială sau forma Hermite pe porţiuni.

Funcţia BSplineSurface(), membră a clasei CMeSurfacePatch construieşte o suprafaţă B-spline.

```
void CMeSurfacePatch::BSplineSurface(CMeControlMap &controlMap)
{
    short i, j, ii;
    short m, n, mDense = 20, nDense = 20;
    double u, v;
    ControlPoints tempTab, controlP;
    CMeSurfaceCurve *surf;
    double *uKnots, *vKnots;
    short degree;
    // gradul suprafeţei este setat de către utilizator
    degree = parentSurface->GetDegree();
    n = min(controlMap.nr_row, controlMap.nr_col) - 2;
    if(degree < 2 || degree >= n +2)
        return; // gradul suprafeţei nu este corect
    // eliberarea punctelor suprafeţei daca aceasta a fost deja generata
    CMeStandardSurface::DeleteContentElement();
    // numărul de segmente de curba generate pe direcţia coloanelor
    m = controlMap.nr_row - 2;
    // numărul de segmente de curba generate pe direcţia liniilor
    n = controlMap.nr_col - 2;
    // generarea secvenţei de noduri pentru linii si pentru coloane
    uKnots = new double[m+degree+1];
    GenKnotVectorBspline(m, degree, uKnots);
    vKnots = new double[n+degree+1];
    GenKnotVectorBspline(n, degree, vKnots);
    // densitatea de curbe intermediare calculate pe fiecare interval
    mDense /= m - degree + 2;
    nDense /= n - degree + 2;
    // generarea curbelor de-a lungul liniilor punctelor de control
    tempTab.SetSize(controlMap.nr_row);
    controlP.SetSize(controlMap.nr_col);
    for (i=degree-1; i<m+degree-1; i++)
    {
        if(uKnots[i+1] > uKnots[i])
        {
            for(ii = 0; ii <= mDense; ii++)
            {
                // valoarea pentru care se calculează punctul de pe curba
                u = uKnots[i] + ii*(uKnots[i+1] - uKnots[i])/mDense;
                // generarea punctelor de control pentru u
                for(j = 0; j < controlMap.nr_col; j++)
                {
                    // copierea coloanei j in tabloul tempTab
                    controlMap.CopyLineCol(tempTab, j, COL);
                    GenerateBsplinePoint(tempTab, degree, uKnots, u, i, controlP[j]);
                }
                // construcţia curbei pornind de la punctele de control controlP
                BSplineCurve(controlP, degree, vKnots, &surf);
                surfaceCurves[0].AddTail(surf);
            }
        }
    }
    // generarea curbelor de-a lungul coloanelor punctelor de control
    tempTab.SetSize(controlMap.nr_col);
    controlP.SetSize(controlMap.nr_row);
```

```
    for (i=degree-1; i<n+degree-1; i++)
    {
        if(vKnots[i+1] > vKnots[i])
        {
            for(ii = 0; ii <= nDense; ii++)
            {
                // valoarea pentru care se calculează punctul de pe curba
                v = vKnots[i] + ii*(vKnots[i+1] - vKnots[i])/nDense;
                // generarea punctelor de control pentru u
                for(j = 0; j < controlMap.nr_row; j++)
                {
                    // copierea coloanei j in tabloul tempTab
                    controlMap.CopyLineCol(tempTab, j, ROW);
                GenerateBsplinePoint(tempTab, degree, vKnots, v, i, controlP[j]);
                }
                // construcția curbei pornind de la punctele de control controlP
                BSplineCurve(controlP, degree, uKnots, &surf);
                surfaceCurves[1].AddTail(surf);
            }
        }
    }
    delete []uKnots;
    delete []vKnots;
}
```

3.5.2 Suprafețe interpolatoare bicubice

Se consideră o rețea rectangulară de puncte x_{IJ}; $0 \le I \le M$, $0 \le J \le N$ şi două seturi de valori pentru parametrii u_I şi v_J. Pornind de la aceste date inițiale se dorește construcția unei suprafețe cubice pe porțiuni de clasă C^1 care să interpoleze punctele rețelei:

$$x(u_I, v_J) = x_{IJ}$$

Rezolvarea problemei cuprinde mai multe etape. Primul pas se bazează pe metodele dezvoltate la curbe [60][62]. Astfel, pentru toate rândurile şi coloanele rețelei de puncte se construiesc curbe spline cubice folosind interpolarea cubică pe porțiuni a curbelor. Trebuie ținut cont de faptul că toate curbele de pe direcția u au aceeaşi parametrizare dată de secvența de noduri u_I, iar toate curbele isoparametrice v sunt definite de vectorul de noduri v_J.

Crearea unei rețele de curbe cubice pe porțiuni C^1 sau C^2 de-a lungul punctelor date nu determină în mod complet suprafața interpolatoare. Rețeaua de curbe reprezintă doar frontierele pentru fiecare petic al suprafeței şi nu întreaga suprafață compusă bicubică pe porțiuni C^1. Ele permit obținerea a 12 din cele 16 elemente necesare pentru fiecare petic. În format Bézier, acestea reprezintă punctele rețelei Bézier situate pe frontierele peticului. Mai lipsesc încă patru puncte interioare pentru fiecare petic: $b_{11}, b_{21}, b_{12}, b_{22}$. În termeni derivativi, mai trebuiesc determinați vectorii twist din colțuri pentru fiecare petic. Pentru obținerea acestor vectori au fost dezvoltate mai multe metode calcul.

Vectori twist zero

Istoric, aceasta reprezintă prima "metodă" de estimare de vectori twist. Ea apare, ascunsă într-un set de formule scrise în pseudo-cod, în lucrările lui Ferguson. Autorul însă nu a prezentat efectele pe care alegerea unor astfel de vectori twist le poate avea.

Aşa cum a fost prezentat în secțiunile anteriore, există suprafețe a căror vectori twist sunt zero. Acestea sunt suprafețele de translație. Dacă curbele de frontieră paralele ale peticelor sunt identice două câte două, rezultate prin translație, atunci asocierea de vectori twist zero este o soluție bună, dar nu şi altfel. În alte cazuri, curbele de frontieră nu sunt curbe generatoare ale unei suprafețe de translație. Dacă pentru acestea se asociază vectori twist zero, peticele rezultate vor prezenta o comportare locală similară cu a suprafețelor de translație, dând naştere la forme plate, care deformează forma suprafeței interpolatoare[55][63].

Dacă se doreşte crearea unei rețele de petice, alegerea vectorilor twist zero conduce la o continuitate de tip C^1 de-a lungul întregii suprafețe. Matematic este corect, dar alegerea nu garantează obținerea unei forme frumoase a suprafeței.

Vectori twist Adini

Metoda a fost introdusă în domeniul CAGD prin intermediul lucrărilor lui Barnhill, Brown şi Klucewicz, şi este bazată pe schema "dreptunghiului lui Adini" din domeniul elementelor finite. Ideea principală a metodei este: patru curbe de frontieră cubice definesc un petic Coons, care este el însuşi un petic bicubic. Vectorii twist din colțurile peticului reprezintă vectorii twist căutați.

Dacă este generată o rețea de petice, vectorii twist Adini descrişi în acest mod nu garantează continuitatea C^1 a suprafeței. Este necesară o transformare a metodei. Se consideră că patru petice ale unei suprafețe se întâlnesc întru-un punct. Curbele de frontieră exterioare ale celor 4 petice definesc din nou un petic Coons de îmbinarea biliniar. Acest petic, alcătuit din 4 bicubice are definit un vector twist în punctul de întâlnire ale celor patru petice. Acesta va fi vectorul twist pentru peticele în cauză. El este dat de formula:

$$x_{uv}(u_i,v_j)=\frac{x_v(u_{i+1},v_j)-x_v(u_{i-1},v_j)}{u_{i+1}-u_{i-1}}+\frac{x_u(u_i,v_{j+1})-x_u(u_i,v_{j-1})}{v_{j+1}-v_{j-1}} - \frac{x(u_{i+1},v_{j+1})-x(u_{i-1},v_{j+1})-x(u_{i+1},v_{j-1})+x(u_{i-1},v_{j-1})}{(u_{i+1}-u_{i-1})(v_{j+1}-v_{j-1})} \tag{3.5.3}$$

Se poate demonstra simplu faptul că metoda Adini aplicată peticelor de frontieră ale unei suprafețe de translație conduce la vectori twist zero [26][88][103]. Metoda poate fi considerată o bună alegere, deoarece, considerată ca un interpolant, ea reproduce toate polinoamele bivariante de forma uv^i, u^iv, $i,j=0,1,2,3$ ceea ce înseamnă un set foarte larg. Vectorii twist zero conduc la distorsiuni nedorite ale suprafeței rezultate.

Vectori twist Bessel

Metoda estimează vectorul twist în punctul $x(u_I,v_J)$ ca fiind vectorul twist al interpolantului bipătrat a nouă puncte $x(u_{I+r},v_{J+s});\ r,s\in\{-1,0,1\}$. Deoarece peticele bipătratice au un twist biliniar, vectorul Bessel este interpolantul biliniar a vectorilor twist a patru petice biliniare formate de nouă puncte. Aceşti vectori twist sunt dați de formula:

$$q_{I,J}=\frac{\Delta^{1,1}x(u_I,v_J)}{\Delta_I\Delta_J} \tag{3.5.4}$$

Vectorul twist Bessel poate fi scris:

$$x_{uv}(u_I, v_J) = \begin{bmatrix} 1-\alpha_I & \alpha_I \end{bmatrix} \begin{bmatrix} q_{I-1,J-1} & q_{I-1,J} \\ q_{I,J-1} & q_{I,J} \end{bmatrix} \begin{bmatrix} 1-\beta_J \\ \beta_J \end{bmatrix}$$
$$\alpha_I = \frac{\Delta_{I-1}}{u_{I+1} - u_{I-1}}, \quad \beta_J = \frac{\Delta_{J-1}}{v_{J+1} - v_{J-1}} \tag{3.5.5}$$

Vectorul twist Brunet

Brunet a dezvoltat o metodă diferită de cea anterioară prin faptul că nu porneşte de la o rețea de petice mărginite de curbe, ci de la o rețea C^1 de petice bicubice. Pentru această suprafață se încearcă modificarea vectorilor twist existenți pentru a se ajunge la o suprafață care să difere cât mai puțin de suprafața originală. În acest sens, metoda este mai degrabă o operație de corectare a suprafeței decât o procedură de estimare a vectorilor twist.

Brunet a considerat că peticul Coons de îmbinare biliniar al unui petic mărginit dat constituie forma optimă pentru suprafață. Pornind de la remarcile făcute la vectorii twist Adini, suprafața Coons este obținută prin aplicarea metodei Adini rețelei de curbe [26][99]. Suprafața dată se modifică pentru a se apropia cât mai mult de "suprafața Adini". Metoda Brunet în termenii formei Bézier defineşte o combinație convexă între punctele Bézier interioare ale suprafeței date şi cele corespunzătoare vectorului twist zero. Brunet consideră că relația:

$$b_{11}^{new} = .25 b_{11}^{zero} + .75 b_{11}^{old} \tag{3.5.6}$$

produce suprafețe care prezintă puține modificări față de suprafața originală.

3.5.3 Interpolanții produs tensorial

Metoda produs tensorial, dezvoltată pentru suprafețe Bézier, Hermite şi B-spline poate fi folosită pentru a se obține suprafețe de interpolare rectangulare. Fie:

$$x(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} c_{ij} A_i(u) B_j(v) \tag{3.5.7}$$

o suprafață produs tensorial. Funcțiile A_i şi B_j sunt arbitrare: polinoame Bernstein, funcții Hermite, funcții B-spline, polinoame Lagrange sau alte funcții. Ecuația poate fi scrisă în formă matricială:

$$x(u,v) = \begin{bmatrix} A_0(u) & \dots & A_m(u) \end{bmatrix} \begin{bmatrix} c_{oo} & & c_{on} \\ \vdots & & \vdots \\ c_{mo} & \dots & c_{mn} \end{bmatrix} \begin{bmatrix} B_0(v) \\ \vdots \\ B_n(v) \end{bmatrix} \tag{3.5.8}$$

Dacă suprafața $x(u,v)$ interpolează un şir de $(m+1)\times(n+1)$ puncte x_{ij}, relația trebuie să fie adevărată pentru fiecare pereche (u_i,u_j). Rezultă astfel $(m+1)\times(n+1)$ ecuații care pot fi scrise sub forma:

$$X = ACB \tag{3.5.9}$$

$$X = \begin{bmatrix} x_{oo} & x_{on} \\ \vdots & \vdots \\ x_{mo} & x_{mn} \end{bmatrix} \qquad A = \begin{bmatrix} A_o(u_0) & A_o(u_m) \\ \vdots & \vdots \\ A_m(u_0) & A_m(u_m) \end{bmatrix}$$

$$C = \begin{bmatrix} c_{00} & c_{on} \\ \vdots & \vdots \\ c_{m0} & c_{mn} \end{bmatrix} \qquad B = \begin{bmatrix} B_0(v_0) & B_0(v_n) \\ \vdots & \vdots \\ B_n(v_0) & \dots \quad B_n(v_n) \end{bmatrix}$$

Matricele funcțiilor de bază A şi B sunt denumite *matricele Vandermode* [2][22][87]. În contextul interpolării, punctele x_{ij} sunt cunoscute în timp ce coeficienții c_{ij} sunt necunoscuți. Aceştia rezultă din relația:

$$C = A^{-1}XB^{-1} \tag{3.5.10}$$

Existența matricelor inverse din relație indică faptul că funcțiile A_i şi B_j sunt liniar independente (acest lucru este valabil în toate cazurile practice de interes). Ecuația prezintă modul în care se poate obține o soluție a problemei de interpolare, care însă implică inversarea explicită a matricelor A şi B.

Pentru a se înțelege mai bine problema interpolării produs tensorial, ecuația matricială 9.9 poate fi rescrisă:

$$\begin{aligned} X &= DB \\ D &= AC \end{aligned} \tag{3.5.11}$$

Matricea D este alcătuită din $m+1$ rânduri şi $n+1$ coloane. Ecuația $X = DB$ poate fi reinterpretată ca o familie de $m+1$ probleme de interpolare univariante, una pentru fiecare rând a lui X şi D, matricea D conținând necunoscutele. Rezolvând cele $m+1$ probleme, toate având aceeaşi matrice a coeficienților B, se poate trece la determinarea ecuației $D = AC$ deoarece D a fost deja calculat. Această ecuație poate fi interpretată ca o familie de $n+1$ probleme de interpolare univariante, matricea coeficienților fiind, pentru toate, A.

Din aceste relații se poate observa faptul că produsul tensorial permite o importantă compactizare a procesului de interpolare. Fără structura produsului tensorial, ar fi fost necesară rezolvarea unui sistem liniar de ordinul $(m+1)(n+1)\times(m+1)(n+1)$. Acesta este un ordin de o mărime mult mai mare decât rezolvarea a $m+1$ probleme cu aceeaşi matrice $(n+1)\times(n+1)$ şi apoi rezolvarea a $n+1$ probleme cu aceeaşi matrice $(m+1)\times(m+1)$. Dacă $m=n$, în cazul celei mai simple abordări sunt necesare $O(m^6)$ calcule, în timp ce produsul tensorial necesită doar $O(m^4)$.

Un exemplu al acestei metode este interpolarea B-spline cubică. Se consideră o rețea de puncte x_{ij} de dimensiune $(K+1)\times(L+1)$ şi două secvențe de noduri $u_0,\dots,u_k$ şi $v_0,\dots,v_L$. Construcția suprafeței se bazează pe rezultatele obținute la interpolarea cubică

spline. Pentru fiecare linie de puncte se prescriu două condiții de sfârșit (fie prin specificarea vectorilor tangenți fie a punctelor Bézier) și se rezolvă problema interpolării B-spline univariante folosind metodele descrise la curbe. Aceasta produce elementele matricei *D*. Apoi se ia fiecare rând al matricei *D* și se aplică asupra lui interpolarea B-spline univariantă, prin prescrierea din nou a condițiilor de sfârșit ca și tangente de capăt. De notat faptul că *D* are două rânduri mai mult decât *X*. Acest lucru este datorat condițiilor de sfârșit. Pentru a se rezolva această discrepanță, se consideră că *X* are două rânduri adiționale care constituie condiții de sfârșit. Pentru cele 4 colțuri, numărul condițiilor crește cu vectorii twist [57][88].

Cu toate că matematic sunt echivalente, procesele 1: mai întâi rând cu rând, apoi coloană cu coloană, 2: mai întâi coloană cu coloană, apoi rând cu rând, nu conduc la aceeași cantitate de calcul.

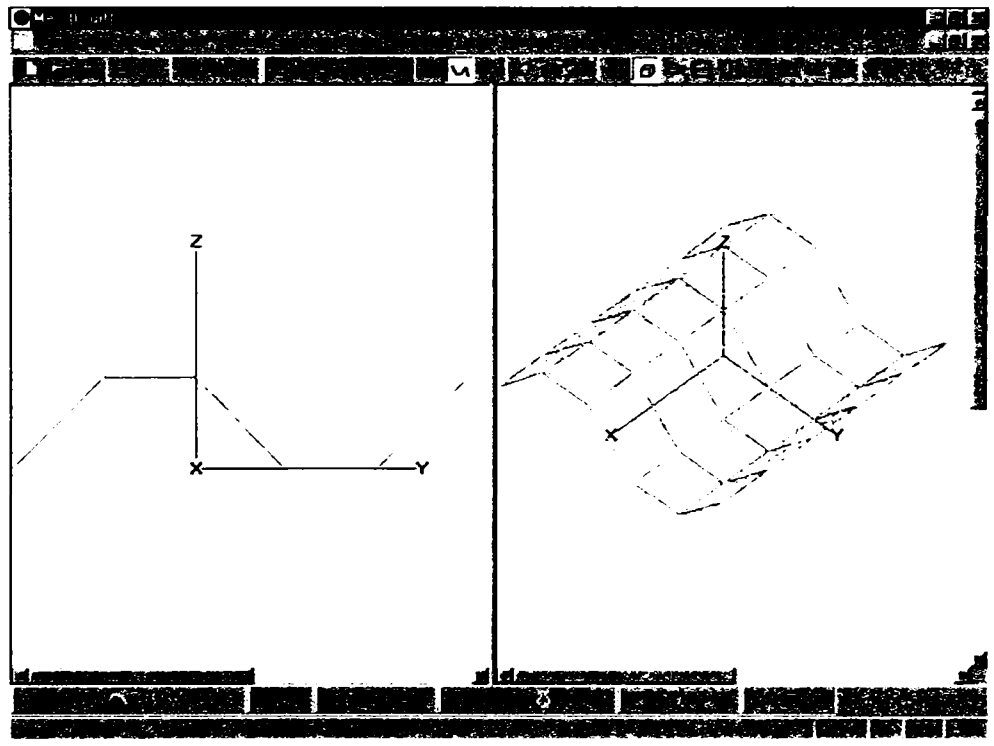

Fig. 3.5.5 Interpolare B-spline

Ilustrarea algoritmului de generare unei suprafețe interpolatoare B-spline a fost realizată cu metoda BSplineSurfaceInterpolation()

```
void CMeSurfacePatch::BSplineSurfaceInterpolation(void)
{
    short i, j, k, degree, s;;
    short m, n, mDense = 20, nDense = 20;
    ControlPoints tempTab, controlP;
    double *uKnots, *vKnots, *xi, *yi;
    double **matrixM, **matrixA, **matrixN, **matrixB;
    typedef double* pDouble;
    // gradul curbei interpolatoare este definit de către utilizator
    degree = parentSurface->GetDegree();
    n = min(controlMap.nr_row, controlMap.nr_col) - 2;
    if(degree < 2 || degree >= n +2)
        return; // gradul suprafeței nu este corect
    CMeStandardSurface::DeleteContentElement();
    m = controlMap.nr_row - 1;
    n = controlMap.nr_col - 1;
    // generarea nodurilor suprafețe
```

```
uKnots = new double[m + degree +2];
GenKnotVectorBspline(m, degree+1, uKnots);
vKnots = new double[n + degree +2];
GenKnotVectorBspline(n, degree+1, vKnots);
// calculul absciselor Greville pe cele doua direcții
 xi = new double[m+1];
CalculPuncteEvaluare(uKnots, xi, m, degree);
yi = new double[n+1];
CalculPuncteEvaluare(vKnots, yi, n, degree);
// calculul matricelor A si B
matrixM = new pDouble[m+1];
matrixA = new pDouble[m+1];
for(i = 0; i <= m; i++)
{   matrixM[i] = new double[m+1];
    memset(matrixM[i], 0, (m+1)*sizeof(double));
    matrixA[i] = new double[m+1];
    memset(matrixA[i], 0, (m+1)*sizeof(double));
}
// calculul coeficienților matricei sistemului
EvaluareBazaBSpline(uKnots, xi, m, degree, matrixM);
// A este transpusa matricei M
for(i = 0; i <= m; i++)
    for(j = 0; j <= m; j++)
        matrixA[j][i] = matrixM[i][j];
matrixN = new pDouble[n+1];
matrixB = new pDouble[n+1];
for(i = 0; i <= n; i++)
{   matrixN[i] = new double[n+1];
    memset(matrixN[i], 0, (n+1)*sizeof(double));
    matrixB[i] = new double[n+1];
    memset(matrixB[i], 0, (n+1)*sizeof(double));
}
// calculul coeficienților matricei sistemului
EvaluareBazaBSpline(vKnots, yi, n, degree, matrixN);
// B este transpusa matricei N
for(i = 0; i <= n; i++)
    for(j = 0; j <= n; j++)
        matrixB[j][i] = matrixN[i][j];
// determinarea matricei punctelor de Boor
double *c = new double[m+1];
short *perm = new short[m+1];
DescompunereLU(matrixA, m+1, perm, &s);
CMeControlMap deBoor(controlMap.nr_row, controlMap.nr_col);
for(j = 0; j <= n; j++)
{   for(short index = 0; index < 3; index++)
    {   for(k = 0; k <= m; k++)
        c[k] = controlMap.controlPoints[k][j][index];
        SistemLU(matrixA, m+1, perm, c);
        for(k = 0; k <= m; k++)
        deBoor.controlPoints[k][j][index] = c[k];
    }
}
delete []c;
delete []perm;
c = new double[n+1];
perm = new short[n+1];
DescompunereLU(matrixB, n+1, perm, &s);
CMeControlMap tmpDeBoor(controlMap.nr_col, controlMap.nr_row);
for(j = 0; j <= m; j++)
{   for(short index = 0; index < 3; index++)
```

```
    {   for(k = 0; k <= n; k++)
        c[k] = deBoor.controlPoints[j][k][index];
        SistemLU(matrixB, n+1, perm, c);
        for(k = 0; k <= n; k++)
        tmpDeBoor.controlPoints[k][j][index] = c[k];
    }
}
// matricea punctelor de Boor este transpusa  matricei tmpDeBoor
for(i = 0; i <= m; i++)
    for(j = 0; j <=n; j++)
        deBoor.controlPoints[i][j] = tmpDeBoor.controlPoints[j][i];
delete []uKnots;
delete []vKnots;
delete []xi;
delete []yi;
for(i = 0; i <= n; i++)
{   delete []matrixN[i];
    delete []matrixB[i];
}
delete []matrixN;
delete []matrixB;
for(i = 0; i <= m; i++)
{   delete []matrixM[i];
    delete []matrixA[i];
}
delete []matrixM;
delete []matrixA;
delete []c;
delete []perm;
// generarea suprafeței B-spline interpolatoare
    BSplineSurface(deBoor);
}
```

3.5.4 Parametrizarea suprafețelor produs tensorial interpolatoare

Interpolarea produs tensorial este o metodă elegantă, însă utilizarea ei este limitată la cazul în care punctele date prezintă o structură rectangulară. Atunci când punctele deviază de la forma de grid, apare o problemă asociată procesului de interpolare spline bicubic (în fapt aproape pentru toate tipurile de interpolare produs tensorial): problema determinării unei parametrizări bune. Pentru curbe, a fost posibilă crearea mai multor metode care asociază valori parametrice punctelor date. Se pune întrebarea de ce nu se aplică aceste metode şi produsului tensorial aşa cum alte metode de la curbe au fost generalizate pentru corespondentele lor de la produsul tensorial. Problema o constituie faptul că trebuie produs un singur set de valori parametrice pentru toate curbele isoparametrice de pe direcția u, şi la fel pe direcția v [64][94].

Fiecare curbă isoparametrică poate fi înzestrată cu o parametrizare. Pentru a se ajunge la o singură parametrizare pentru toate, este necesar un proces de determinare a unei medii a acestor parametrizări. O astfel de abordare va produce rezultate acceptabile doar dacă curbele isoparametrice au aceleaşi caracteristici ale formelor, ceea ce permite o parametrizare asemănătoare a lor.

Problema determinării unei parametrizări unice este fără soluție pentru cazul în care datele de interpolare prezintă o distribuție neregulată şi se doreşte aplicarea interpolării spline cubice. Pentru suprafețe de grad mai mare şi dacă se înlocuieşte continuitatea C^1 sau C^2 cu continuitate G^1 există mai multe metode de rezolvare a problemei.

3.5.5 Deformări de suprafețe

Pentru anumite situații, controlul local al suprafeței nu este util, nu este ceea ce se doreşte. Astfel de situații de proiectare sunt întinderea unei suprafețe într-o anumită direcție sau curbarea suprafeței. Acestea sunt deformări globale ale formei, iar modificarea repetată a punctelor poligonului de control este oarecum greoaie pentru asemenea sarcini. P. Bézier a creat o metodă de deformare a peticelor Bézier care să rezolve problema deformațiilor globale pentru suprafețe. Metoda este de asemenea aplicabilă suprafețelor B-spline.

Pentru a se ilustra acest principiu se consideră mai întâi cazul 2D. Fie $x(t)$ o curbă planară (Bézier, B-spline) care, fără a se pierde din generalitate, este poziționată în interiorul pătratului unitate. În acest pătrat se defineşte un grid regulat de puncte $b_{i,j} = [i/m, j/n]^T ; i = 0,...,m;\ j = 0,...,n$. Fiecare punct (u,v) al suprafeței pătrate se poate determina astfel:

$$(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} b_{i,j} B_i^m(u) B_j^n(v) \qquad (3.5.12)$$

Acest lucru rezultă din proprietatea de precizie liniară a polinoamelor Bernstein. Dacă se realizează o distorsionare a gridului rezultând un nou grid $b'_{i,j}$, punctul (u,v) va fi transformat în punctul (u',v'):

$$(u',v') = \sum_{i=0}^{m}\sum_{j=0}^{n} b'_{i,j} B_i^m(u) B_j^n(v) \qquad (3.5.13)$$

Cu alte cuvinte se realizează o transformare a spațiului E^2 întru-un alt spațiu E^2. În particular, punctelor de control ale curbei $x(t)$ li se vor asocia noi puncte de control care vor determina o nouă curbă $y(t)$. De remarcat că *y* este doar o aproximare a imaginii lui *x* după relația definită. Acest lucru este subliniat de faptul că imaginea poligonului de control al curbei *x* prin aplicarea relației este o colecție de arce de curbă şi nu un alt poligon liniar pe porțiuni [56][97].

Se poate da în acest fel o metodă indirectă de proiectare a curbelor: prin modificarea punctelor $b_{i,j}$ se produce o deformare globală a suprafeței. Această tehnică poate facilita anumite sarcini de proiectare care altfel ar deveni prea obositoare.

Tehnica poate fi generalizată. Se poate înlocui distorsiunea Bézier printr-o distorsiune B-spline produs tensorial analoagă. Acest lucru va reintroduce anumite forme ale controlului local în cadrul schemei de proiectare. Următorul nivel de generalizare este trecerea la spațiul E^3: se utilizează petice Bézier trivariante:

$$(u',v',w') = \sum_{i=0}^{m}\sum_{j=0}^{n}\sum_{k=0}^{l} b'_{i,j,k} B_i^m(u) B_j^n(v) B_k^l(w) \qquad (3.5.14)$$

care constituie o deformare a spațiului 3D. Relația poate fi folosită pentru deformarea unei rețele de control a unei suprafețe inclusă în cubul unitate. Din nou, utilizarea peticelor Bézier pentru deformare poate fi înlocuită cu petice trivariante B-spline, sau alte metode, în scopul de a se introduce anumite posibilități de control local în cadrul metodei.

Metodele de deformare volumice duc la modificarea întregului ansamblu al suprafeței deodată, într-un mod care permite răspândirea modificărilor în fiecare parte a

ansamblului foarte armonios. Prin modificarea punctelor de control unul câte unul, nu se poate obține o astfel de armonizare a deformărilor.

Complexitatea formelor obiectelor din spațiul laboratorului virtual este redată cu o mare exactitate cu ajutorul suprafețelor B-spline. Proiectarea suprafețelor prin aceste metode este mai dificilă decât cu metoda Bézier produs tensorial deoarece poliedrul de control este mai dificil de creat, fiind alcătuit dintr-un număr mare de puncte interconectate. Aceste metode însă pot surprinde curburile complexe ale suprafețelor modelate.

Un exemplu de utilizare a suprafețelor B-spline este proiectarea tomografului din cadrul laboratorului de imagistică (Figura 3.5.6). Curburile formei aparatului sunt redate prin suprafețe B-spline aproximante și interpolatoare, celelalte elemente fiind realizate cu suprafețe analitice.

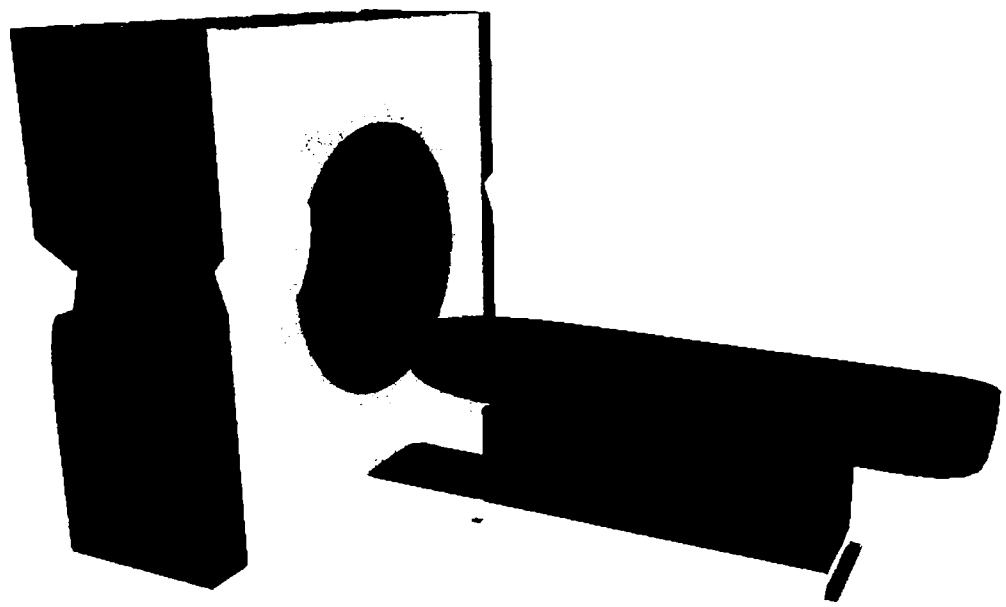

Fig. 3.5.6. Modelul virtual al tomografului

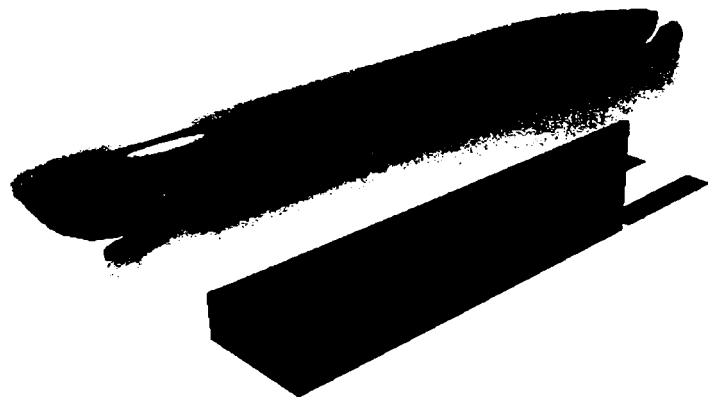

Fig. 3.5.7. Detaliu al modelului tomografului realizat cu suprafețe B-spline

3.6 Suprafețe Coons

Toate metodele de proiectare a suprafețelor descrise au în comun următorul fapt: fiecare metodă necesită un număr finit de puncte (rețeaua de control) pentru generarea suprafețelor respective. Această trăsătură nu se regăseşte şi în cazul peticelor de *suprafață Coons* care se constituie ca forme ale "*interpolării transfinite*" [66] [103]. Suprafața nu este descrisă prin rețeaua punctelor de control ci printr-un set de curbe. Conectarea sau "amestecarea" acestor curbe generează suprafața. Astfel, aceste metode de proiectare interpolează un număr infinit de puncte, aflate pe segmente de curbă, pentru a construi forma suprafeței. Cazul cel mai frecvent întâlnit este cel al peticele de suprafață Coons care realizează conectarea a patru curbe adiacente care se intersectează şi formează o frontieră închisă.

3.6.1 Suprafețe Coons biliniare

Se consideră patru curbe de frontieră $P(u,0), P(u,1), P(0,v), P(1,v)$ definite pe domeniul $[0,1]$, fiecare pereche de curbe de frontieră opuse fiind parametrizate identic. Suprafața Coons $P(u,v)$ realizează conectarea celor patru curbe date respectând condițiile de frontieră, adică reducerea la o curbă de frontieră corectă pentru $u = 0,1$ şi $v = 0,1$.

Se defineşte mai întâi cazul unui petic Coons biliniar care interpolează cele patru curbe de frontieră. O suprafață riglată a fost definită ca un interpolant liniar a două curbe de frontieră date într-o singură direcție. Astfel, "îmbinarea" a două suprafețe riglate care leagă două perechi de curbe de frontieră poate satisface condițiile impuse de curbele de frontieră şi de asemenea să conducă la generarea unui petic Coons [26][58]. Formulele celor două suprafețe riglate construite pe direcțiile u şi v sunt:

$$\begin{aligned} P_1(u,v) &= (1-u)P(0,v) + uP(1,v) \\ P_2(u,v) &= (1-v)P(u,0) + vP(u,1) \end{aligned} \tag{3.6.1}$$

Prin adunarea acestor două formule se obține suprafața:

$$P(u,v) = P_1(u,v) + P_2(u,v) \tag{3.6.2}$$

Peticul de suprafață rezultat nu îndeplineşte condițiile de frontieră. De exemplu, pentru $v = 0, v = 1$ relația devine:

$$\begin{aligned} P(u,0) &= P(u,0) + [(1-u)P(0,0) + uP(1,0)] \\ P(u,1) &= P(u,1) + [(1-u)P(0,1) + uP(1,1)] \end{aligned} \tag{3.6.3}$$

În aceste formule termenii din parantezele drepte sunt în plus şi trebuiesc eliminați pentru a se obține curbele de frontieră inițiale. Aceşti termeni definesc frontierele unei suprafețe $P_3(u,v)$ nedorite care poate fi definită prin interpolare liniară după direcția v, astfel:

$$P_3(u,v) = (1-v)[(1-u)P(0,0) + uP(1,0)] + v[(1-u)P(0,1) + uP(1,1)] \tag{3.6.4}$$

Eliminând $P_3(u,v)$, numită şi *suprafață de corecție*, din relația peticului rezultă:

$$P(u,v)=P_1(u,v)+P_2(u,v)-P_3(u,v)$$
$$\text{sau} \qquad (3.6.5)$$
$$P(u,v)=P_1(u,v)\oplus P_2(u,v)$$

unde $\oplus$ exprimă "suma booleeană": $P_1+P_2-P_3$. Suprafața $P(u,v)$ generată definește *peticul biliniar Coons* care leagă cele patru curbe de frontieră [56][61]. Forma matricială a relației este:

$$P(u,v)=-\begin{bmatrix}-1 & (1-u) & u\end{bmatrix}\left[\begin{array}{c|cc} 0 & P(u,0) & P(u,1) \\ \hline P(0,v) & P(0,0) & P(0,1) \\ P(1,v) & P(1,0) & P(1,1)\end{array}\right]\begin{bmatrix}-1\\ 1-v\\ v\end{bmatrix} \qquad (3.6.6)$$

Coloana din stânga şi rândul de sus reprezintă $P_1(u,v)$ respectiv $P_2(u,v)$ în timp ce blocul din dreapta jos corespunde suprafeței de corecție $P_3(u,v)$. Funcțiile $-1, 1-u, u, 1-v, v$ sunt numite *funcții de amestecare* deoarece ele combină (amestecă) împreună patru curbe distincte de frontieră pentru a obține o singură suprafață [26].

Dacă curbele de frontieră ale unui petic biliniar Coons sunt coplanare, peticul rezultat este de asemenea planar. Vectorii tangenți la suprafață sunt:

$$\begin{aligned} P_u(u,v)&=[P_u(u,0)+P(1,v)-P(0,v)+P(0,0)-P(1,0)]+\\ &\quad v[P_u(u,1)-P_u(u,0)+P(1,0)-P(0,0)-P(1,1)+P(0,1)] \qquad (3.6.7)\\ &=A+vB\\ P_v(u,v)&=[P_v(0,v)+P(u,1)-P(u,0)+P(0,0)-P(0,1)]+\\ &\quad u[P_v(1,v)-P_v(0,v)-P(0,0)+P(0,1)+P(1,0)-P(1,1)]\\ &=C+uD \end{aligned}$$

Deoarece curbele sunt coplanare vectorii tangenți $P_u(u,0), P_u(u,1), P_v(0,v), P_v(1,v)$ sunt conținuți în planul curbelor. Ceilalți vectori care intervin în formule sunt de asemenea conținuți în plan, fiind puncte ale curbelor. De aceea, vectorii A şi B, respectiv C şi D sunt situați în planul curbelor de frontieră. În consecință, vectorii tangenți $P_u(u,v)$ şi $P_v(u,v)$ în orice punct (u,v) al suprafeței sunt conținuți în acest plan.

Normala suprafeței este dată de relația:

$$N(u,v)=P_u\times P_v=(A+vB)\times(C+uD) \qquad (3.6.8)$$

Normala este perpendiculară pe vectorii tangenții P_u şi P_v, deci şi pe planul curbelor de frontieră. Astfel, pentru orice punct al suprafeței, direcția normalei suprafeței este constantă, amplitudinea depinzând de punct. Acest lucru este caracteristic doar pentru o suprafață plană. Astfel, un petic Coons biliniar degenerează într-un plan dacă curbele lui de frontieră sunt coplanare. Acest petic poate fi folosit pentru a crea plane cu frontiere curbe. De remarcat faptul că un petic bicubic Coons sau orice alt petic care are funcții de amestecare

neliniare nu se reduce la un plan când toate frontierele lui sunt coplanare.

Principalul dezavantaj al peticului biliniar Coons îl constituie faptul că oferă doar continuitate C^0, continuitate de poziție, între petice adiacente, chiar dacă curbele lor de frontieră formează o rețea de continuitate C^1 [20]. De exemplu, dacă două petice sunt conectate după curba de frontiera $P(1,v)$ a primului petic şi $P(0,v)$ a celui de-al doilea, se poate arăta că derivatele frontierelor comune ale celor două petice $P_u(1,v)$ şi $P_u(0,v)$ nu sunt egale.

3.6.2 Suprafețe Coons bicubice

Continuitatea planului tangent de-a lungul frontierelor peticelor unei suprafețe compuse Coons este esențială pentru aplicațiile practice. Dacă este dată o rețea de curbe C^1 este important să se poată genera o suprafață compusă Coons care este de continuitate C^1. Din formula pânzei Coons se poate observa că alegerea funcțiilor de amestecare controlează comportarea peticului. Dacă sunt folosite polinoamele Hermite $H_0^3(x)$ şi $H_1^3(x)$ în locul polinoamelor liniare $(1-u)$ şi respectiv u, rezultă un petic Coons bicubic de amestecare [26][103]. Acest petic asigură continuitatea C^1 dintre petice. Relațiile suprafețelor riglate pot fi rescrise:

$$\begin{aligned} P_1(u,v) &= H_0^3(u)P(0,v) + H_1^3(u)P(1,v) \\ P_2(u,v) &= H_0^3(v)P(u,0) + H_1^3(v)P(u,1) \end{aligned} \tag{3.6.9}$$

La fel ca şi în cazul peticului biliniar, poate fi formată suma booleeană $P_1 \oplus P_2$, iar forma matriceală a peticului Coons bicubic devine:

$$P(u,v) = -\begin{bmatrix} -1 & H_0^3(u) & H_1^3(u) \end{bmatrix} \left[\begin{array}{c|cc} 0 & P(u,0) & P(u,1) \\ \hline P(0,v) & P(0,0) & P(0,1) \\ P(1,v) & P(1,0) & P(1,1) \end{array}\right] \begin{bmatrix} -1 \\ H_0^3(v) \\ H_1^3(v) \end{bmatrix} \tag{3.6.10}$$

Aşa cum se poate observa, suprafața de corecție este în mod uzual formată prin aplicarea funcțiilor de amestecare doar datelor referitoare la colțuri.

Racordul de clasă C^1 a două petice Coons $P_1(u,v)$ şi $P_2(u,v)$ implică:

$$\begin{aligned} P_1(0,v) &= P_2(1,v) \\ P_{1u}(0,v) &= P_{2u}(1,v) \end{aligned} \tag{3.6.11}$$

Continuitatea de poziție C^0 este satisfăcută automat între două petice deoarece ele împart aceeaşi curbă de frontieră. Continuitatea C^1, se stabileşte derivând relațiile celor două petice Coons în raport cu u:

$$\begin{aligned} P_{1u}(0,v) &= H_0^3(v)P_{1u}(0,0) + H_1^3(v)P_{1u}(0,1) \\ P_{2u}(1,v) &= H_0^3(v)P_{2u}(1,0) + H_1^3(v)P_{2u}(1,1) \end{aligned} \tag{3.6.12}$$

Dacă rețeaua de curbe de frontieră este de clasă C^1, derivatele $P_{1u}(0,0)$, $P_{2u}(1,0)$ respectiv

$P_{1u}(0,1)$, $P_{2u}(1,1)$ sunt egale, ceea ce conduce la un racord de clasă C^1 între cele două petice. Astfel, continuitatea suprafeței compuse bicubice Coons este îndeplinită dacă curbele de frontieră sunt C^1 continue [20].

Peticul bicubic Coons este uşor de folosit într-un mediu de proiectare deoarece sunt necesare numai patru curbe de frontieră. O suprafață bicubică Coons compusă mai flexibilă poate fi dezvoltată, dacă, pe lângă curbele de frontieră, sunt cunoscute şi derivatele în punctele de tăiere ale frontierelor $P_u(0,v), P_u(1,v)$, $P_v(u,0), P_v(u,1)$. Aceste derivate trebuie să fie compatibile cu informațiile curbei. Similar relațiilor anterioare, se pot defini următorii interpolanți cubici Hermite:

$$\begin{aligned} P_1(u,v) &= H_0^3(u)P(0,v)+H_1^3(u)P(1,v)+H_2^3(u)P_u(0,v)+H_3^3(u)P_u(1,v) \\ P_2(u,v) &= H_0^3(v)P(u,0)+H_1^3(v)P(u,1)+H_2^3(v)P_v(u,0)+H_3^3(v)P_v(u,1) \end{aligned} \tag{3.6.13}$$

$H_0^3 \ldots H_3^3$ sunt polinoamele Hermite cubice. Formând suma booleeană $P_1 \oplus P_2$ a celor două relații rezultă peticul bicubic Coons care conține derivatele frontierelor. Introducerea acestor derivate determină apariția vectorilor de twist în colțurile peticelor. Formula matriceală a peticul Coons în acest caz este:

$$P(u,v) = -\begin{bmatrix} -1 & F_1(u) & F_2(u) & F_3(u) & F_4(u) \end{bmatrix} \times \left[\begin{array}{c|cc|cc} 0 & P(u,0) & P(u,1) & P_v(u,0) & P_v(u,1) \\ \hline P(0,v) & P(0,0) & P(0,1) & P_v(0,0) & P_v(0,1) \\ P(1,v) & P(1,0) & P(1,1) & P_v(1,0) & P_v(1,1) \\ \hline P_u(0,v) & P_u(0,0) & P_u(0,1) & P_{uv}(0,0) & P_{uv}(0,1) \\ P_u(1,v) & P_u(1,0) & P_u(1,1) & P_{uv}(1,0) & P_{uv}(1,1) \end{array}\right] \begin{bmatrix} -1 \\ F_1(v) \\ F_2(v) \\ F_3(v) \\ F_4(v) \end{bmatrix} \tag{3.6.14}$$

Matricea 3×3 din colțul stânga sus reprezintă peticul definit anterior fără a fi date derivatele frontierelor. Coloana din stânga şi rândul de sus reprezintă interpolanții $P_1(u,v)$ şi respectiv $P_2(u,v)$, în timp ce matricea 4×4 de jos reprezintă suprafața bicubică produs tensorial Hermite [55][57]. Relația demonstrează faptul că fiecare suprafață Coons de amestecare bicubică reproduce o suprafață bicubică. Însă, peticele Coons bicubice de amestecare nu pot fi descrise în general prin suprafețe produs tensorial bicubice. Astfel, metoda de proiectare Coons poate descrie o varietate mai bogată de suprafețe decât o pot face suprafețele produs tensorial.

Cap 4 Proiectarea unui laborator virtual deservit de roboţi mobili

Pornind de la metodele de generare a curbelor şi suprafeţelor descrise în capitolele anterioare s-a construit un spaţiu virtual tridimensional al laboratorului de imagistică în care este prevăzută deservirea diferitelor operaţii ale personalului medical şi pacienţilor prin intermediul unor roboţi mobili. Întregul model se constituie ca o copie „virtuală" al laboratorului de imagistică din cadrul Bazei de Cercetare cu Utilizatori Multiplii: Centrul de Modelare a Protezării şi Intervenţiilor Chirurgicale asupra Scheletului Uman. Roboţii mobili, spaţiul laboratorului şi obiectele situate în el sunt construiţi ca ansambluri de suprafeţe curbe. Roboţii se deplasează fie liber în spaţiul laboratorul evitând atât obiectele cât şi ceilalţi roboţi, fie către o anumită ţintă cu respectarea principiilor de planificare a mişcării [38][39][48][49].

Capitolul cuprinde o descriere a tehnicilor de programare utilizate, motivarea alegerii lor, modul în care s-a realizat proiectarea software. Sunt descrise modalităţile de reprezentare realistică a laboratorului virtual şi mecanismul de implementare a mişcării roboţilor.

4.1 Tehnicile de implementare software

Toţi algoritmii descrişi în lucrare şi întreaga aplicaţie a fost dezvoltată în limbajul Visual C++ [10][67]. S-a ales acest mediu de programare deoarece îmbină *tehnica de programare orientată pe obiect* [8][29] cu *programarea sub Windows* bazată pe mesaje, elemente necesare şi potrivite proiectului.

4.1.1 Programarea orientată pe obiect

Tehnica orientată pe obiect reprezintă o metodă de proiectare şi implementare software bazată pe definirea unei colecţii de *tipuri de date abstracte* [9] ce modelează elementele complexe din lumea reală şi realizarea interconexiunii acestora într-un ansamblu coerent. Termenul *de tip de dată abstractă* cuprinde o încapsulare de date şi operaţii asupra acestor date într-un tot unitar. În acest mod proprietăţile şi comportamentul unui element sunt legate împreună. Tipurile de date abstracte sunt utilizate ca specificaţii în construcţia *obiectelor* şi *claselor* [67][85][101]. Un obiect modelează o entitate din lumea reală sau imaginară fiind caracterizat de starea, comportamentul si identitatea sa. O *clasă* este o implementare care poate fi instanţiată în vederea creării de obiecte multiple având acelaşi comportament. O clasă se constituie astfel ca o reprezentare a unui tip de dată abstract.

Proiectarea obiectuala cuprinde alături de abstractizarea datelor încă două

concepte esențiale: moştenirea şi polimorfismul. Moştenirea este o relație între entități caracterizată prin trecerea atributelor şi funcțiilor de la o clasă de bază, la o altă clasă derivată. Strâns legat de noțiunea de moştenire, polimorfismul se referă la faptul că o anumită operație este definită în mod specific pentru diferite obiecte. Noțiunea de polimorfism este proprie unei structuri ierarhice de clase, alcătuită pe principiul moştenirii, pentru care funcțiile aparținând clasei de bază sunt redefinite în clasele derivate conform comportamentului acestora.

Programarea orientată pe obiect, succesor al programării structurate, este cea mai recentă etapă în strategiile de dezvoltare a produselor software. Ea reprezintă un rezultat firesc al procesului de dezvoltare a ingineriei software în condițiile în care complexitatea, amploarea şi răspândirea produselor software este în creştere în toate domeniile. Această abordare oferă posibilitatea unei avansări mai uşoare de-a lungul întregului proces de dezvoltare a unui produs software. Prin evitarea discontinuităților ce existau la modelele structurate, se înlesneşte urmărirea proiectării unui produs, la toate nivelele ciclului de viață asociat.

Principalele etape parcurse în procesul de dezvoltare a unei aplicații utilizând tehnologia orientată pe obiect sunt:

- Analiza orientată pe obiect OOA;
- Proiectarea orientată pe obiect OOD;
- Implementarea.

Abordarea unui sistem complex presupune o descompunere a acestuia. Această descompunere poate fi:

- funcțională (procedurală sau algoritmică) care generează nivele ierarhice de funcții
- orientată pe obiect care generează entități ce modelează obiectele lumii reale. Reuniunea acestor entități este însuşi sistemul ce se implementează.

Analiza domeniului presupune identificarea şi clasificarea subsistemelor complete ce compun sistemul precum şi a componentelor reutilizabile.

Analiza cerințelor sistemului presupune înțelegerea cerințelor sistemului global, a perspectivei şi aşteptărilor utilizatorului. De aici va rezulta un ansamblu de concepte operaționale şi cerințe de performanță.

Proiectarea sistemului implică descompunerea acestuia în subsisteme. Rezultă o mulțime de entități configurabile şi reutilizabile ca bază pentru proiectarea software. Identificarea interfețelor între componente are loc de asemenea în această etapă. Se realizează astfel descrierea componentelor software şi a subsistemelor potențiale cât şi a interfețelor dintre ele [9][90].

Analiza cerințelor software defineşte o submulțime de cerințe atribuite fiecărui element de configurare software rezultat din planificarea temporală. Datele inițiale ale acestei etape sunt reprezentate de setul de cerințe sistem asociate fiecărui element de configurare software.

Etapa de analiză a generat produse de analiză ca: liste de abstractizări, clase, obiecte pentru domeniul aplicației, etc.

Proiectarea software include etapele descrise de figura 4.1.

În cadrul tranziției de la OOA clasele şi obiectele de nivel înalt devin clase şi obiecte la nivel de proiectare.

Structurarea proceselor este o etapă specifică pentru sistemele de timp real şi implică stabilirea relaţiilor de concurenţă aparentă sau reală între taskuri.

Etapa OOD creează arhitectura software. Rezultă module software utilizând clasele şi obiectele la nivel de proiectare. Acum se stabilesc interfeţele între obiecte şi se creează modelul de abstractizare a procesului rezultând o imagine de obiecte implementabile într-un limbaj de programare orientat pe obiect. Accentul se pune în acest moment pe crearea de obiecte, nu de clase. Documentul principal ce însoţeşte această etapă este documentul de proiectare software ce descrie arhitectura obiectelor şi interfaţa cu alte componente software şi dispozitive hardware.

Etapa de evaluare a proiectării are loc pe parcursul întregii faze de proiectare şi realizează o analiză a fezabilităţii şi performanţei. Această etapă este puternic iterativă cu etapa OOD, având ca scop realizarea unei arhitecturi cât mai performante. Etapa de evaluare se desfăşoară în paralel cu toţi subpaşii fazei de proiectare. Evaluarea structurii obiectuale are loc din punctul de vedere al reutilizabilităţii, portabilităţii, încapsulării, a relaţiilor între obiecte şi a tratării excepţiilor.

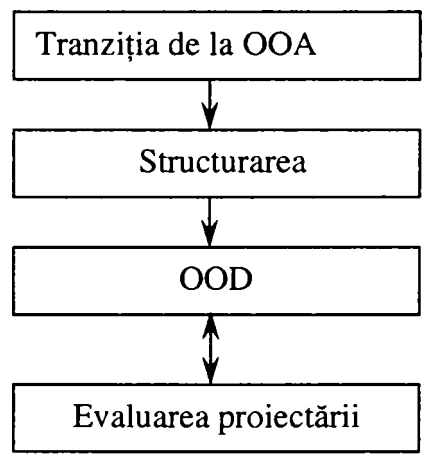

Fig. 4.1 Etapele proiectării software

Implementarea include testarea şi integrarea modulelor de programare. În cadrul sistemelor complexe dezvoltarea are loc incremental [27].

Programarea orientată pe obiect se constituie ca o nouă modalitate de organizare a codului şi datelor care determină o creştere a controlului asupra complexităţi procesului de dezvoltare software [100][102]. Această abordare oferă următoarele avantaje în raport cu programarea structurală:

- uşurinţa proiectării şi a înţelegerii programului: OOP permite generarea rapidă a unui prototip apropiat de conceptele modelate şi care, prin similitudinile sale cu elementele reale, poate fi uşor de urmărit; programele orientate spre obiecte sunt mai naturale, reflectă mai corect problema care este implementată şi utilizează concepte derivate din acesta;

- nivel înalt de reutilizare: entitățile orientate pe obiect pot fi structurate cu interfețe care permit utilizarea lor de către mai multe aplicații din acelaşi domeniu [86];
- uşurința dezvoltării aplicației: printr-o corespondență apropiată între conceptele modelate si entitățile implementate software, modificările şi extensiile ulterioare pot fi izolate la câteva obiecte, ceea ce conduce la reducerea codului si a efortului de testare;
- creşterea abstractizării: aceasta permite o privire de ansamblu asupra proiectului, punând temporar în paranteză detaliile, pot fi evidențiate astfel elementele esențiale ale sistemului;
- creşterea capacității de încapsulare: obiectele pot fi privite ca şi entități independente asupra cărora se poate acționa doar prin intermediul funcțiilor proprii;
- ascunderea datelor: tratarea obiectelor drept „cutii negre" permite utilizarea eficientă a acestora în relație cu alte obiecte trecând peste detaliile de implementare a funcționalităților lor;
- scurtarea timpului de elaborare, datorită uşurării activităților de codificare şi depanare, precum şi datorită unei senzitivități mult mai reduse a programului față de eventualele schimbări în specificațiile proiectului inițial;

4.1.2 Programarea în Windows, Visual C++

Modelul de programare sub Windows este complet diferit de programarea secvențială caracteristică sistemului de operare MS-DOS. Diferența esențiala dintre cele două modele o constă faptul că programarea sub Windows se bazează pe mesaje [10][69]. Un program scris în MS-DOS utilizează funcții ce accesează explicit sistemul de operare pentru comunicarea cu utilizatorul. În cazul unui program scris sub Windows, interconectarea cu utilizatorul este realizată prin sistemul de mesaje primite de către aplicație din partea sistemului de operare, fără a fi necesară apelarea directă a sistemului. Elementul central al oricărui program Windows este *fereastra principală a aplicației* care conține propriul său cod de procesare a mesajelor primite din partea sistemului [3]. Fiecărui eveniment ce are loc îi este dedicat un mesaj unic, care este tratat de către aplicație în mod specific. Mediile de dezvoltare a aplicațiilor Windows pun la dispoziția programatorilor un set de elemente care permit construcția automată a ferestrei principale a aplicației împreună cu mecanismul de procesare a mesajelor.

Un alt element esențial al modelului de programare sub Windows este *unitatea grafică de interfață GDI* [10]. În programarea clasică, operațiile grafice erau implementate în general prin lucrul direct cu memoria video a sistemului de calcul ceea ce implica probleme de portabilitate a programelor. Modelul Windows introduce un nivel de abstractizare denumit *unitatea grafică de interfață*, care accesează suportul grafic în mod unitar, independent de platforma grafică avută la dispoziție. În locul accesării directe a hardware-ului, aplicația foloseşte funcțiile GDI care se referă la o structură de date denumită *unitate contextuală* care cuprinde toate informațiile grafice necesare operațiilor de vizualizare. Sistemul de operare Windows realizează conexiunea între unitatea grafică

de interfață şi unitatea fizică a sistemului de calcul. Acest model de acces la suportul grafic este aproape la fel de rapid ca cel direct.

O trăsătură caracteristică modelului de programare Windows o constituie *programarea bazată pe resurse*. Toate datele şi resursele necesare unei aplicații Windows sunt cuprinse în cadrul programului prin intermediul unor fişiere de resurse care au un format prestabilit. Acest lucru permite legarea directă a datelor în programul executabil şi în acelaşi timp o separare a lor față de codul sursă. Fişierele de resurse pot cuprinde ferestre pentru introducerea de date denumite *dialogbox-uri*, meniuri, poze, texte precum şi date specifice aplicației. Mediile de dezvoltare a aplicațiilor Windows cuprind editoare de resurse care simplifică procesul de generare de resurse.

Modelul de programare Windows a introdus de asemenea şi conceptul de *librării de funcții legate dinamic* [69]. Acest lucru se referă la faptul că librăriile de funcții utilizate de către o aplicație sunt ataşate acesteia doar în procesul de execuție al aplicației şi nu în mod static atunci când se generează programul executabil. Mai multe aplicații pot utiliza în acelaşi timp librăria dinamică, ceea ce conduce la un efort mai redus de memorie şi de spațiu pe disc.

Complexitatea procesului de programare sub Windows a impus dezvoltarea unor medii de programare care să cumuleze toate elementele specifice acestui model. Microsoft Visual Studio C++ se constituie ca un sistem complet de dezvoltare a aplicațiilor Windows folosind tehnologia orientată pe obiect. Având la bază limbajul C++, acest mediu oferă posibilitatea abordării tehnicii orientate pe obiect pentru proiectarea unei aplicații precum şi a modelului de programare Windows. Întrepătrunderea acestor două componente permite dezvoltarea unor aplicații complexe în mod eficient şi foarte sigur. Pe linia îmbinării acestor modele de proiectare şi programare, în cadrul mediului Visual C++, a fost creată o librărie complexă de clase C++ denumită *Microsoft Foundation Class Library* [3]. Librăria cuprinde entități grupate pe structuri ierarhice complexe folosite pentru generarea cadrului de lucru a unei aplicații Windows, a sistemului de ferestre şi unităților contextuale, utilizarea resurselor, pentru crearea de aplicații specifice diferitelor domenii software: baze de date, rețele, Internet, sisteme server-client, CAD. O aplicație Windows bazată pe Microsoft Foundation Class Library prezintă anumite trăsături specifice. Mecanismul de tratare a mesajelor este ascuns în interiorul claselor de ferestre CWnd, CWindow, CView, CDialog, ceea ce permite o proiectare clară a funcțiilor specifice fiecărui eveniment, dar în acelaşi timp realizează şi o anumită îngrădire a modului de gestionare a mesajelor. Structura aplicației este concepută fie pe noțiunea de *document* sau *multiplu document*, fie pe cea de *cutie de date*. Documentul constituie ansamblul de date pe care o aplicație le procesează sau le generează. Aplicațiile care operează asupra mai multor documente în acelaşi timp poartă numele de multi-document. Modelul obiectual pentru acest concept în cadrul MFC este clasa CDocument. Aceasta cuprinde propria sa listă de ferestre, definite prin intermediul clasei CView, cu ajutorul cărora se realizează vizualizarea documentului în forma sa specifică [10][69].

Cutia de date constituie elementul de bază pentru comunicarea dintre aplicație şi utilizator. Introducerea de date sau afişarea de informații este realizată în mod eficient şi uşor de procesat. Mediul oferă un mecanism de proiectare a cutiilor de date, denumit *generatorul de resurse*, precum şi pentru implementarea obiectuală a acestora. Clasa asociată unei cutii de date este derivată din CDialog. Elementul atomic pentru o cutie de

date este câmpul de date care conceptual, este o fereastra copil a cutiei de date. Sunt definite o suită de tipuri de câmpuri dar exista posibilitatea de proiectare a unor forme proprii specifice cerinţelor aplicaţiei. Mecanismul de prelucrare şi dispecerizare a mesajelor este încorporat în clasa de bază.

4.1.3 Librăria grafică OpenGL

OpenGL este o interfaţă software pentru componentele hardware grafice ale calculatorului. Librăria OpenGL cuprinde o multitudine de comenzi necesare pentru specificarea obiectelor şi a operaţiilor necesare unei aplicaţii tridimensionale interactive. Vizualizarea scenelor 3D proiectate se apropie foarte mult de realitatea modelată, încorporând o mare parte din elementele specifice ale acesteia [6][73]. Interfaţa este proiectată ca o structură de sine stătătoare, independentă de elementele hard pentru a putea fi utilizată pe diferite platforme. În funcţie de potenţialul sistemului de calcul, a plăcii video şi a acceleratorului grafic, comenzile OpenGL sunt procesate fie hard, fie sunt simulate software în cazul în care nu există suport hard pentru ele. Utilizarea eficientă a capabilităţilor hard conduce la o viteză de calcul mare şi deci posibilitatea de proiectare a unor scene de o complexitate foarte ridicată, precum şi a posibilităţii încorporării mişcării corpurilor şi a interacţiunilor dintre acestea.

Reprezentarea cea mai simplă realizată de interfaţa OpenGL este *modelul wireframe* [1]. Obiectele sunt desenate în totalitatea lor, fiecare muchie fiind reprezentată printr-o linie. Modelarea poate fi rafinată prin definirea mai clară a efectului de tridimensionalitate printr-o simulare a adâncimii. Liniile mai îndepărtate de punctul din care se priveşte sunt mai estompate. OpenGL utilizează efectele atmosferice referite prin noţiunea de *fog* pentru a realiza simularea adâncimii. De asemenea un alt pas important îl constituie modelarea *antialias*. Aceasta reprezintă tehnica de reducere a liniilor zigzagate datorate aproximării muchiilor drepte prin pixeli. Un nivel mai complex de reprezentare îl constituie modelului *flat-shaded*. Obiectele sunt definite ca solide, fiecare suprafaţă vizibilă este desenată folosindu-se culoarea proprie a obiectului, porţiunile ascunse nefiind luate în considerare. O impresie mult mai realistică şi tridimensională este produsă de modelul *smooth-shaded*. Acest model include faţă de modelul anterior efectele luminii asupra corpurilor. Suprafeţele vizibile ale obiectelor primesc nuanţe de culori diferite în funcţie de poziţia lor în raport cu sursele de lumină din cadrul scenei. Reprezentarea nu mai este plată ci are o structură rotunjită, lină care accentuează efectul de tridimensionalitate. Adăugarea de *texturi* şi umbre acestui model întăreşte şi mai mult impresia realistică a scenei realizate. Texturile sunt imagini bidimensionale ataşate obiectelor 3D care înlocuiesc culoarea în procesul de desenarea al suprafeţelor corpurilor. Imaginile texturilor simulează structura diferitelor materiale din care sunt concepute obiectele, lemn, piatră, plastic, metal, ceea ce conferă o anumită materialitate acestora. Varietatea şi complexitatea modurilor de reprezentare OpenGL este motivul pentru care am utilizat librăria în cadrul aplicaţiei la desenarea modelului virtual al laboratorului [68].

Datorită independenţei sale faţă de platforma hard utilizată, în cadrul interfeţei OpenGL, nu se regăsesc comenzi pentru introducerea de date sau de procesare a operaţiilor cu ferestrele de lucru. Aceste procese vor fi implementate prin intermediul sistemului de control al ferestrelor existent pe sistemul de calcul. De asemenea OpenGL

nu furnizează funcţii de nivel înalt pentru descrierea modelelor 3D complexe ce aproximează lumea reală. Pentru a putea fi reprezentat un corp, modelul său geometric trebuie redus la un set de *primitive geometrice* alcătuit din puncte, linii şi poligoane.

În general aplicaţiile OpenGL prezintă o structură similară de creare a unei scene, desfăşurată pe mai multe etape de procesare a datelor. În prima parte, datele geometrice urmează o cale diferită de prelucrare faţă de *datele în pixeli*. Etapele finale de procesare se vor aplica asupra tuturor elementelor. Datele în pixeli se referă la texturi, imagini, bitmap-uri ce pot fi inserate în ansamblul scenei [73].

Operaţiile grafice principale pentru proiectarea unei scene în OpenGL sunt:

- **Descrierea modelului obiectelor scenei** prin primitive geometrice, generarea normalelor suprafeţelor, setarea culorilor şi a texturilor corespunzătoare corpurilor. Toate aceste date vor fi reţinute într-o structură denumită *lista de afişare*.
- **Pregătirea scenei** prin alegerea poziţiei din care se doreşte vizualizarea scenei – *punctul de vedere* şi a domeniului în care se va reprezenta denumit *viewport*.
- OpenGL va procesa intern datele geometrice prin:
  - **Transformarea coordonatelor spaţiale în coordonate ecran**, generarea coordonatelor texturilor şi transformarea lor, calculul culorilor de reprezentare a tuturor obiectelor. Acestea rezultă ca o combinaţie dintre culorile proprii ale obiectelor, a texturilor ataşate acestora precum şi a condiţiilor de lumină setate pentru scenă. Există posibilitatea introducerii mai multor surse de lumină iar poziţia acestora, normalele suprafeţelor, proprietăţile de reflexie ale corpurilor şi a mediului vor fi luate în considerare la calculul culorilor. Toate acestea corespund etapei *operaţii pe vertecşi*.
  - **Eliminarea porţiunilor primitivelor** aflate în exteriorul semispaţiului delimitat de planul care defineşte adâncimea scenei şi apoi aplicarea operaţiilor de viewport asupra lor operaţie denumită *asamblare a primitivelor*. Rezultatul acestei etape este lista primitivelor geometrice transformate împreună cu toate informaţiilor lor referitoare la culoare, adâncime şi valori ale coordonatelor texturilor.
- **Prelucrarea datelor furnizate în pixeli**. Secvenţa de pixeli a unei imagini este convertită din formatul propriu, scalată, orientată în mod corespunzător şi apoi memorată într-o zonă destinată texturilor. Fiecărei imagini îi este ataşat un obiect textură pentru o mai eficientă procesare a lor. Această operaţie este denumită *asamblarea texturilor*.
- **Convertirea descrierii matematice a obiectelor şi a datelor** prin rasterizare într-o secvenţă de *fragmente*. Fiecare fragment corespunde unui pixel de pe ecran. Calculul fragmentelor ia în considerare modelul de vizualizare ales, tipul liniilor, dimensiunea punctelor, modul de desenare a poligoanelor, efectul antialias.
- **Procesarea fragmentelor** realizează aplicarea ultimelor transformări asupra structurii de pixeli înainte de vizualizarea lor pe ecran. Aceste transformări sunt maparea elementelor de textură şi a elementelor de fog asupra fiecărui fragment. Rezultatul final este harta de pixeli a scenei.

4.2 Laborator virtual deservit de roboți

Scopul aplicației este proiectarea unui laborator virtual deservit de roboți mobili analizându-se comportarea mișcării acestora într-un mediu de testare care cuprinde diferite obstacole. Totalitatea entităților care alcătuiesc spațiul virtual precum și structura roboților au fost create cu ajutorul suprafețelor și a curbelor dezvoltate în capitolele precedente. Mișcarea roboților în cadrul scenei este subordonată cerinței de evitare a coliziunilor cu celelalte entități. Testul de coliziune se bazează pe operațiile de intersecții de curbe și cele de suprafețe. Pentru ca simulare sa fie cât mai veridică am utilizat pentru reprezentarea întregii scene biblioteca grafică OpenGL.

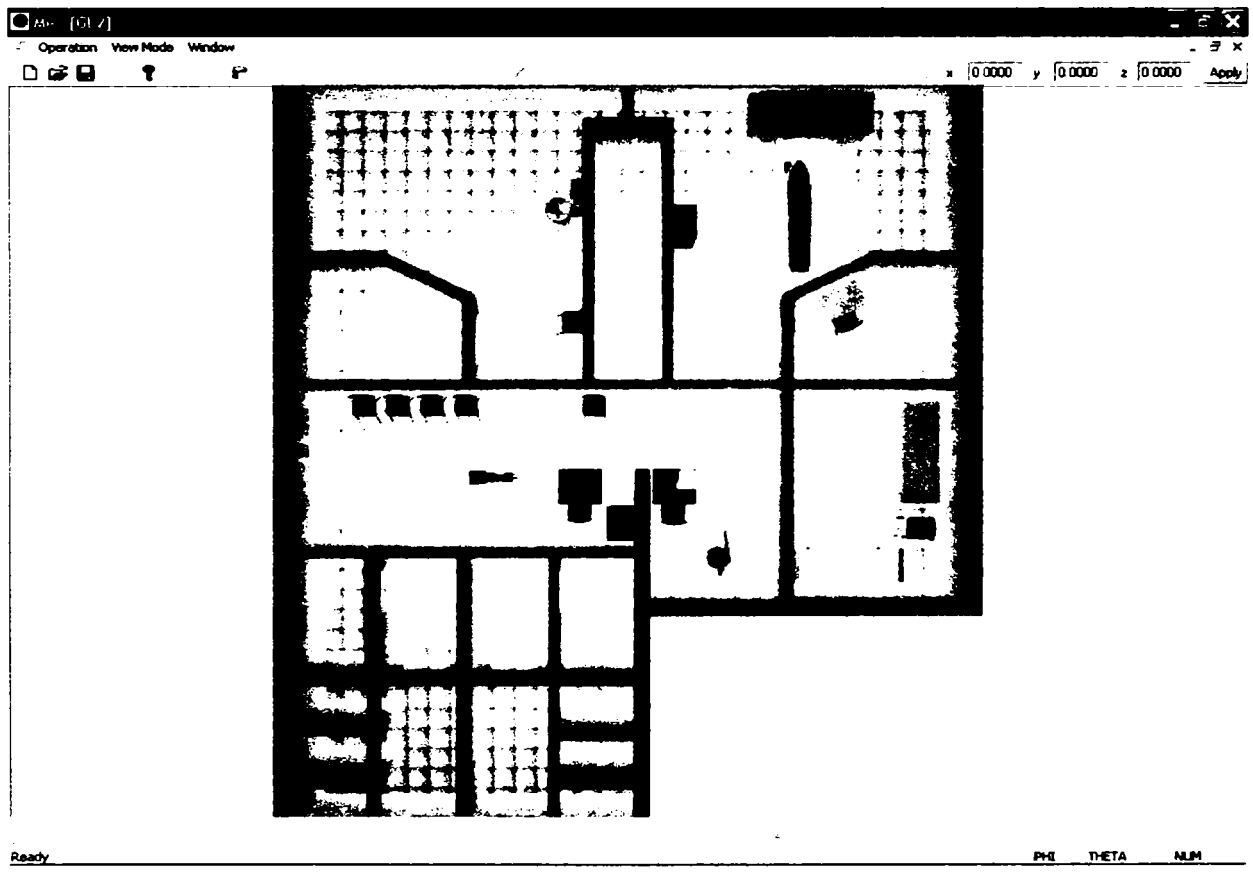

Fig 4.2 Imagine de ansamblu a laboratorului virtual

Întregul proces de proiectare a laboratorului virtual a fost cuprins într-o aplicație creată în Visual C++. Programarea obiectuală îmbinată cu programarea sub Windows au constituit mediul ideal pentru acest gen de proiect. Aplicația este structurată pe trei etape principale. Prima etapă o constituie generarea entităților geometrice care alcătuiesc spațiul laboratorului precum și construcția formei roboților. Următoarea etapă este reprezentarea întregii scene într-un mod cât mai real. Programarea mișcării roboților reprezintă ultimul element esențial al aplicației.

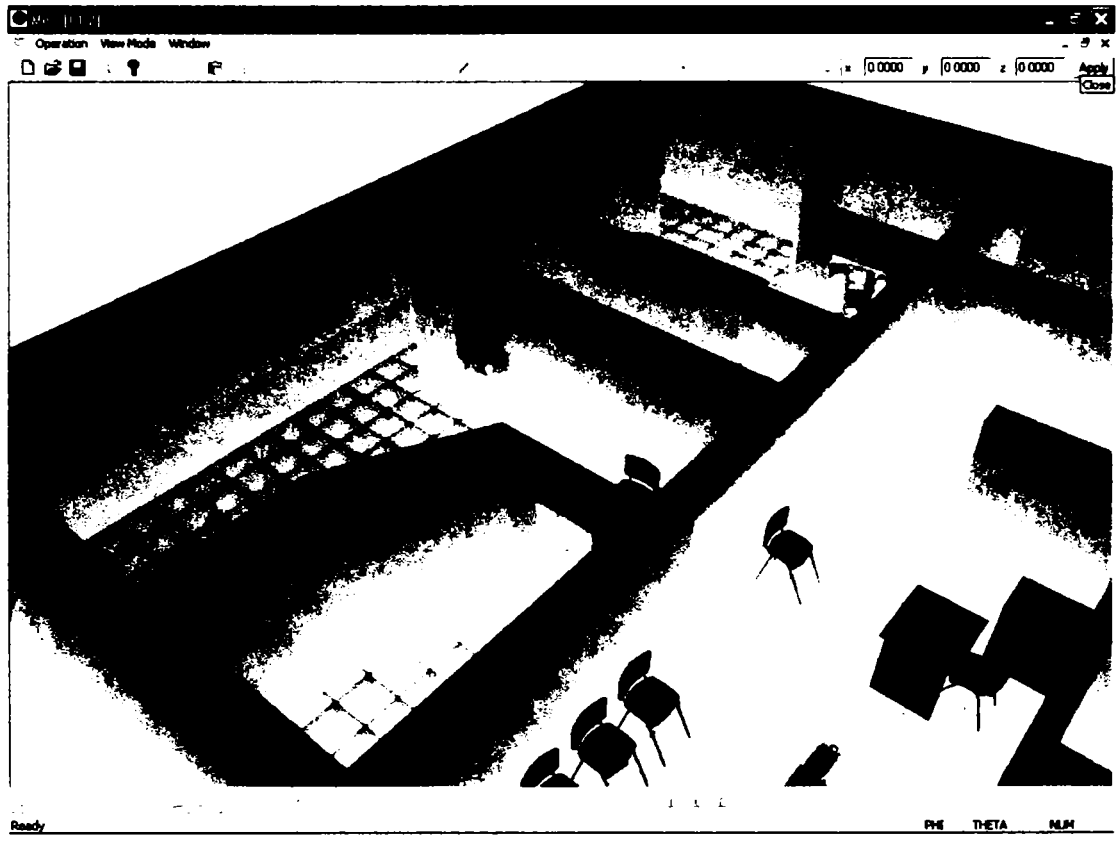

Fig 4.3 Vizualizarea panoramică a încăperilor laboratorului virtual

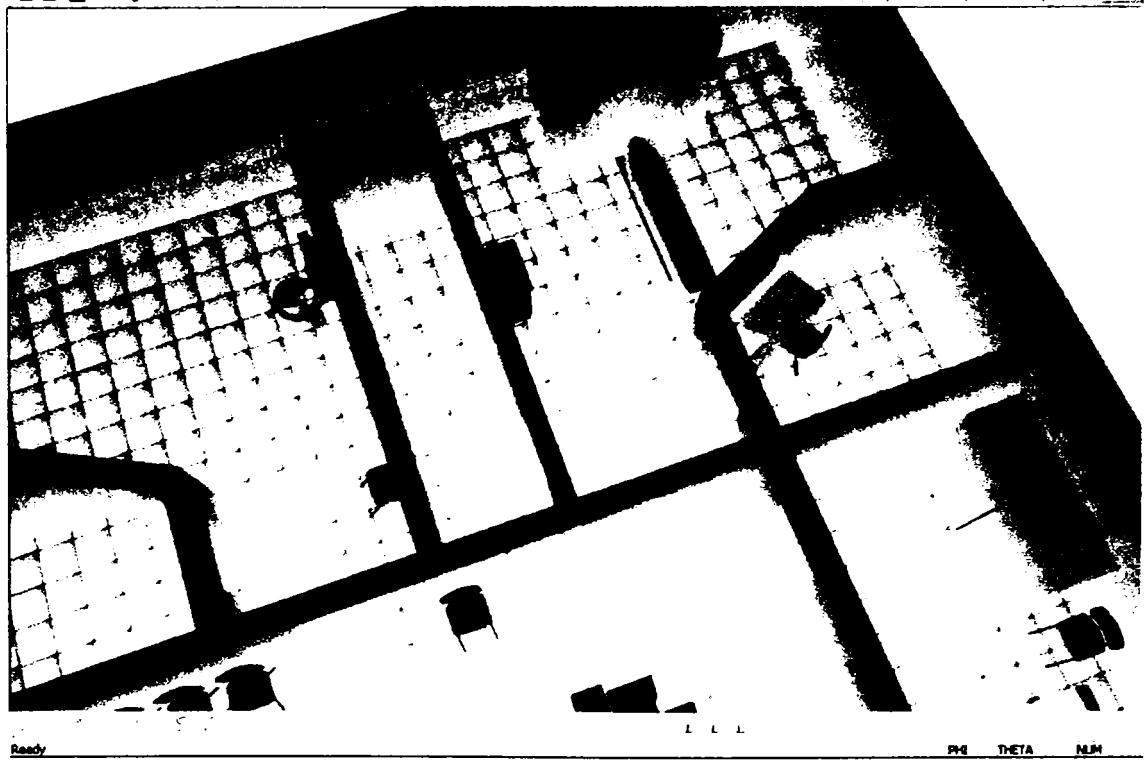

Fig 4.4 Vizualizarea panoramică a tomografului

4.2.1 Construcţia aplicaţiei de proiectare a spaţiului virtual

Aplicaţia a fost programată cu ajutorul mediului Microsoft Visual C++. Alegerea acestui limbaj a fost determinată de caracteristicile sale perfect adecvate pentru un astfel de program. Aşa cum s-a prezentat anterior, capabilităţile superioare de proiectare oferite de programarea obiectuală completate de tehnica de proiectare sub Windows şi de librăria Microsoft Foundation Classes au permis atât crearea unei structuri de clase de entităţi geometrice necesare construcţiei spaţiale cât şi vizualizarea şi realizarea dinamismului roboţilor.

Aplicaţia a fost concepută ca un model multiplu document. Necesitatea acestui tip de program a fost determinată de două elemente. Primul a fost disjuncţia efectuată între etapa de construcţie geometrică a scenei şi etapa de vizualizare realistică a ei. Pentru generarea entităţilor geometrice care în final alcătuiau elementele concrete ale scenei era necesară realizarea unui editor interactiv care să uşureze operaţiile de construcţie şi mai puţin utilizarea de modalităţi superioare de reprezentare. Operaţia de vizualizare procesează geometria elementelor scenei fiind independentă de modul de construcţie al lor. Pentru fiecare etapă a fost definit un tip de document propriu, clasa CMeDocument corespunde etapei de construcţie geometrică iar clasa CMeOpenglDocument celei de vizualizare realistică . Între aceste tipuri de documente există o legătură strânsă deoarece un document OpenGL prelucrează datele unui document CMeDocument. Fiecare dintre aceste tipuri de documente conţin propria lor listă de ferestre de vizualizare şi modalităţile specifice de reprezentare [69].

Un alt element determinant în alegerea acestui model de aplicaţie l-a constituit posibilitatea de proiectare modulară a scenei. Obiectele din cadrul laboratorului precum şi roboţii au fost proiectate în documente separate, salvate în fişiere separate şi apoi copiate şi poziţionate în mod corespunzător în documentul final pentru a alcătui ansamblul laboratorului. Acest lucru a uşurat modelarea scenei, fiecare element putând fi observat mai întâi independent cât şi în conexiune cu celelalte elemente.

Fiecare tip de document conţine propria sa listă de funcţionalităţi ce pot fi apelate atât din meniuri cât şi din toolbar-uri sau meniuri contextuale.

Generarea entităţilor laboratorului şi a formei roboţilor

Prima etapă importantă din cadrul aplicaţiei este generarea geometriei obiectelor constitutive ale laboratorului virtual. Entităţile de bază în acest proces sunt curbele şi suprafeţele descrise în capitolele anterioare. Alături de acestea, pentru construcţia suprafeţelor plane au fost definite elementele poligonale. Obiectele din laborator au fost construite prin conexiunea acestor entităţi formând elemente geometrice complexe. O ierarhie a claselor geometrice utilizate şi a relaţiilor dintre ele este reprezentată în figura 4.2.

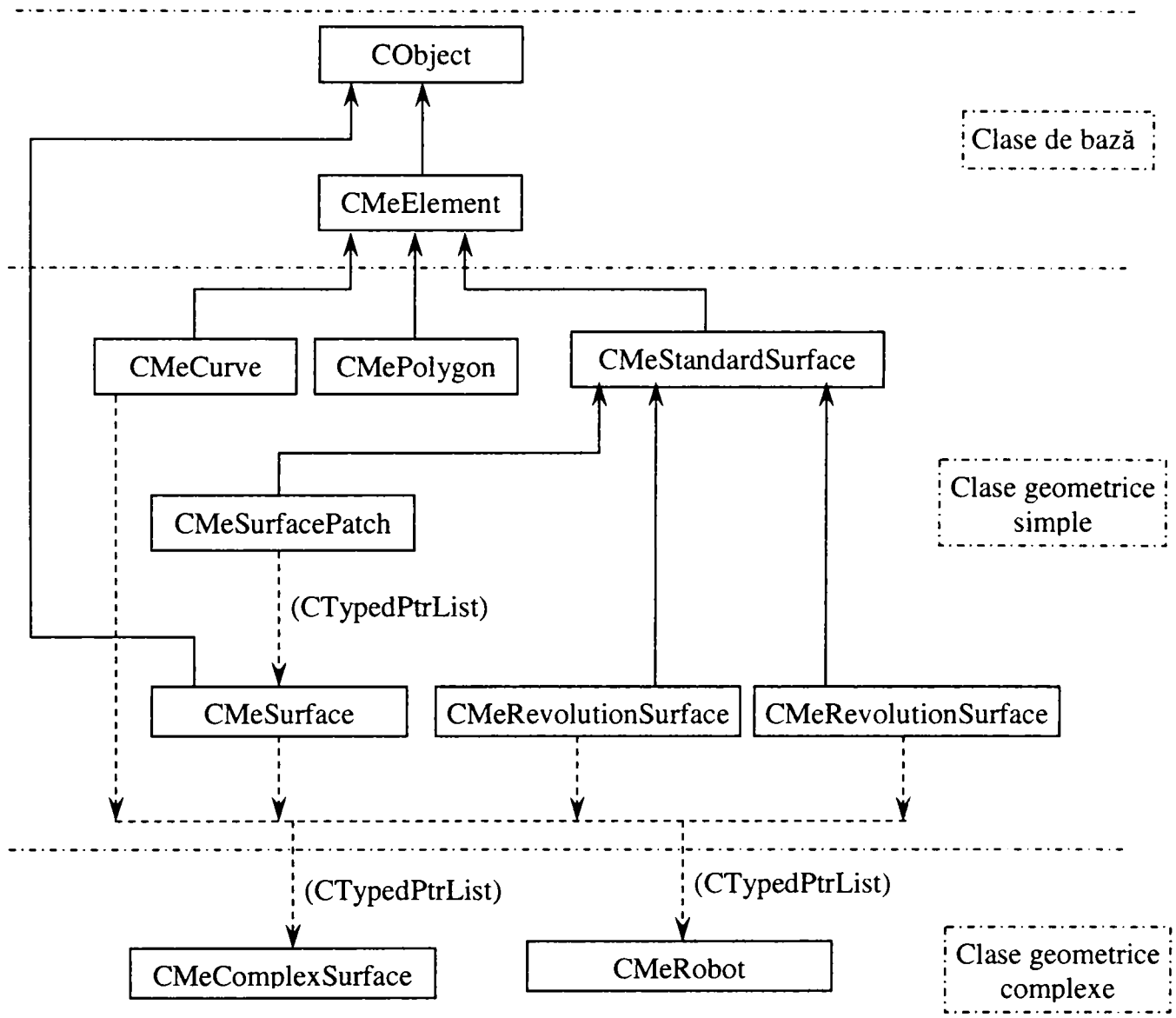

Fig. 4.5 Schema ierarhiei claselor elementelor de construcţie

Structura geometrică a spaţiului virtual ce modelează laboratorul a fost definită ca o *listă polimorfică* [90] de elemente geometrice. Listele polimorfice reprezintă o grupare de obiecte de tipuri diferite dar care prezintă un comportament similar. Caracteristicile comune sunt surprinse într-o clasă de bază, iar celelalte tipuri sunt derivate din aceasta. Esenţa unei liste polimorfice o constituie tratarea în mod unitar a tuturor obiectelor din cadrul listei indiferent de tipul lor. Deoarece toate corpurile care compun ansamblul laboratorului vor fi supuse aceloraşi modelări sau reprezentări s-a considerat că modalitatea de structurare liniară a datelor este soluţia optimă de implementare software [81][82][84].

Elementul geometric de bază este definit de clasa abstractă CMeElement. Aceasta este clasa părinte pentru toate entităţile geometrice construite în cadrul aplicaţiei şi înglobează toate funcţionalităţile virtuale comune tuturor acestor entităţi. Între aceste funcţionalităţi pot fi: desenarea, translaţia, rotaţia, trecerea dintr-un sistem de coordonate

în altul, intersecția, conversia în primitive OpenGL, etc.

CMeElement a fost derivată din clasa abstractă CObject, derivare impusă de proprietățile intrinseci ale acestei clase. CObject reprezintă clasa nucleu a librăriei MFC la care se raportează toate celelalte clase şi elemente ale librăriei. Dintre caracteristicile multiple ale clasei, cele care mi-au atras atenția au fost posibilitatea de *serializare* şi de utilizare a *listelor polimorfice*.

Serializarea reprezintă operația de salvare şi citire a informațiilor unui obiect al clasei într-un format de fişier. Derivarea a permis utilizarea acestui mecanism de păstrare a datelor la nivelul fiecărui element geometric prin suprascrierea funcției virtuale *Serialize()*. În acest mod, totalitatea elementelor geometrice ale unui document, sunt salvate într-un fişier propriu cu extensia *doc*.

De asemenea MFC pune la dispoziție un mecanism de gestionare a listelor polimorfice de obiecte derivate din CObject. Structura liniară a datelor ce compun spațiul laboratorului s-a bazat pe aceste liste. Implementarea listelor polimorfice de către MFC a fost realizată prin clase *template*, clase care permit realizarea unei suite de operații asupra unor obiecte indiferent de tipul acestora. Pentru lista polimorfică de elemente geometrice am folosit clasa template CTypedPtrList<CObList,CMeElement *>.

Primul nivel de derivare este constituit din entitățile geometrice de bază curbe şi suprafețe. Algoritmii şi modalitățile de generare precum şi tipurile acestor entități au fost descrise pe larg în capitolele anterioare.

Deoarece majoritatea curbelor sunt construite pornind de la un poligon de referință, implementarea lor a fost realizată prin intermediul unei singure clase CMeCurve care însumează toți algoritmii de generare dezvoltați. În acest mod, trecerea de la un tip de curbă la altul se realizează foarte simplu prin setarea tipului curbei şi reconstruirea ei prin intermediul funcției membre ConstructElement(). Proiectarea clasei a fost realizată în special din perspectiva creşterii vitezei de execuție şi mai puțin din cea a optimizării resurselor de memorie. De acest deziderat s-a ținut seama atât la nivelul gestionării algoritmilor cât şi la modul de implementare al acestora. Pentru o accesare rapidă a funcțiilor specifice fiecărui tip de curbă, a fost realizată o organizare într-un tablou de structuri a acestor funcții, apelul acestora fiind efectuat prin indexul corespunzător care este definit ca fiind tocmai tipul curbei. De asemenea, pentru organizarea datelor care constituie punctele poligonului de referință şi punctele curbei s-au utilizat structuri tabelare care permit accesarea rapidă a acestora. O modalitate optimă de gestionare a tablourilor este prin intermediul clasei template CArray<>.

Clasa nucleu pentru suprafețe este clasa abstractă CMeStandardSurface. Aceasta cuprinde lista curbelor longitudinale şi transversale care definesc suprafața şi virtual, funcționalitățile specifice tuturor tipurilor de suprafețe. Pentru curbele suprafețelor nu s-a folosit clasa generală de curbe deoarece modul de construcție este diferit. Clasa corespunzătoare acestor curbe este CMeSurfaceCurve. Suprafețele sintetice sunt generate ca o colecție de petice. Fiecare petic, definit prin clasa CMeSurfacePatch, conține propria sa rețea de puncte de control precum şi lista de curbe ale suprafeței peticului fiind derivat din clasa nucleu. Numele clasei suprafețelor sintetice este CMeSurface. Asemănător curbelor, clasa cuprinde toți algoritmii de generare, modificarea tipului suprafeței realizându-se fără a se necesita schimbarea tipului de clasă. Pentru fiecare tip de suprafață analitică am definit o clasă proprie deoarece modul de construcție şi parametrii inițiali diferă, singurele similitudini regăsindu-se în clasa nucleu, clasa părinte şi pentru aceste

suprafeţe. Deşi în cazul suprafeţelor mai apar câteva trepte de derivare intermediare am considerat clasele finale care definesc suprafeţele sintetice şi analitice ca nivel de derivare a entităţilor de bază.

Tot ca element atomic de bază se consideră şi clasa CMePolygon utilizată la implementarea suprafeţelor planare mărginite de un contur poligonal închis. Necesitatea acestui tip de suprafaţă rezidă din elementele de planaritate existente în spaţiul laboratorului. Acestea puteau fi definite ca şi cazuri particulare ale suprafeţelor curbe dar, s-a considerat că sunt mai corect definite în acest mod prin care se realizează şi o simplificare a reprezentării lor.

Următorul nivel de organizare corespunde elementelor complexe din cadrul laboratorului virtual. Acestea sunt grupate în două categorii distincte: roboţi şi obiecte care alcătuiesc scena. Ambele tipuri sunt definite ca ansambluri compuse din elemente de bază care alcătuiesc un tot unitar. Clasa prin care sunt implementate obiectele ce formează ambientul laboratorului se numeşte CMeComplexElement. Necesitatea introducerii acestui nivel de organizare se datorează nevoii de modularizare având un echivalent concret în obiectele spaţiului modelat. Fiecare obiect constitutiv al laboratorului a fost proiectat independent iar amplasarea sa în cadrul ansamblului s-a putut realiza simplu. Clasa corespunzătoare roboţilor poartă numele de CMeRobot. Aceasta însumează toate proprietăţile determinante pentru modelarea geometrică a roboţilor. Un punct definitoriu pentru aceştia îl constituie funcţionalităţile dinamice incluse în cadrul clasei.

Laboratorul virtual se constituie astfel ca un ansamblu de elemente de bază şi complexe organizate într-un spaţiu virtual. Lista polimorfică de elemente geometrice ce corespunde acestui ansamblu este ataşată clasei CMeDocument. Modelul de reprezentare ales pentru acest document este wireframe. Deoarece punctele de control a curbelor şi suprafeţelor pot fi introduse atât prin valorile coordonatelor cât şi cu ajutorul mouse-ului a fost necesară pe lângă reprezentarea 3D de perspectivă şi realizarea reprezentării 2D într-unul din planele globale ale sistemului de referinţă. În acest mod punctele generate cu mouse-ul au valori metrice corecte, cea de a treia coordonată a lor putând fi modificată ulterior. Pentru uşurarea operaţiilor de generare a elementelor geometrice au fost create funcţionalităţi suplimentare de vizualizare şi gestionare a acestora. Operaţii precum modificarea direcţiei de privire, interschimbarea modurilor de reprezentare 2D şi 3D, vizualizarea simultană din diferite perspective, rotirea şi translatarea, copierea, ştergerea, compunerea, facilitează procesul complex de modelare a spaţiului virtual al laboratorului. Alături de aceste operaţii se regăsesc algoritmii de generare a curbelor, suprafeţelor şi poligoanelor.

4.2.2 Reprezentarea realistică a spaţiului laboratorului

OpenGL reprezintă un motor puternic de proiectare a realităţii virtuale, procesul de construcţie a scenei OpenGL fiind structurat pe mai multe etape. Principalii paşi sunt:

- **Generarea ferestrei de lucru**. In concordanţă cu principiile de proiectare Microsoft Visual C++ am construit un document CMeOpenGLDoc dedicat domeniul OpenGL. In acest mod se realizează o disjuncţie completă a proceselor în cadrul aplicaţiei. Ferestrele CMeOpenGLView ataşate de noul document sunt create astfel încât să suporte operaţii OpenGL. Setarea

parametrilor de vizualizare a ferestrei se realizează în funcția OnSize(). Operația constă în calcularea *matricei de proiecție a domeniului*, pe baza dimensiunilor spațiului tridimensional al scenei, matrice care se va aplica tuturor elementelor ce sunt vizualizate. De asemenea se realizează și o mapare a spațiului scenei la spațiu bidimensional al ferestrei de lucru.

Fig 4.6 Vizualizarea echipamentelor de achiziție de imagine, printere și pupitrul de înregistrare pacienți

- **Construcția geometriei scenei**. Pentru implementarea acestei operații a fost necesar ca la nivelul fiecărui element geometric să existe o funcție GenerateOpenGLRepresentation() care să convertească în format OpenGL geometria acestuia. Pentru reprezentarea curbelor se utilizează tipul linie frântă GL_LINE_STRIP. Pentru descrierea optimă a suprafețelor în OpenGL am folosit tipul linie frântă cvadratică GL_QUAD_STRIP. Curbele longitudinale ale peticelor de suprafețe sunt parcurse succesiv două câte două pentru a se genera linia frântă cvadratică. Formele poligonale sunt definite cu funcțiile de *tesselare* pentru poligoane gluTessBeginPolygon() gluTessEndPolygon(). Toate elementele geometrice ale scenei sunt grupate într-o listă prin funcțiile glNewList() și glEndList(). Tot în acest mod se grupează și părțile constitutive ale fiecărui robot în liste diferite. Acest principiu de organizare a geometriei permite accesarea simplă a acestor date în

momentul desenării prin funcția glCallList(). Un proces complex în construcția scenei îl constituie operația de generare a texturilor atașate de elemente. Un pas important al acestui proces îl constituie determinarea tuturor texturilor folosite în cadrul scenei și construcția obiectelor de tip textură pentru fiecare în parte. Maparea texturilor pe fiecare suprafață se realizează prin generarea de coordonate textură 2D pentru fiecare punct al suprafeței. În acest mod, textura va fi întinsă pe întreaga suprafață. De asemenea, tot în cadrul acestei etape se determină dimensiunea scenei, valorile de translație și unghiurile de rotație ale acesteia încât reprezentarea să păstreze aceleași setări ca și în desenarea normală [68].

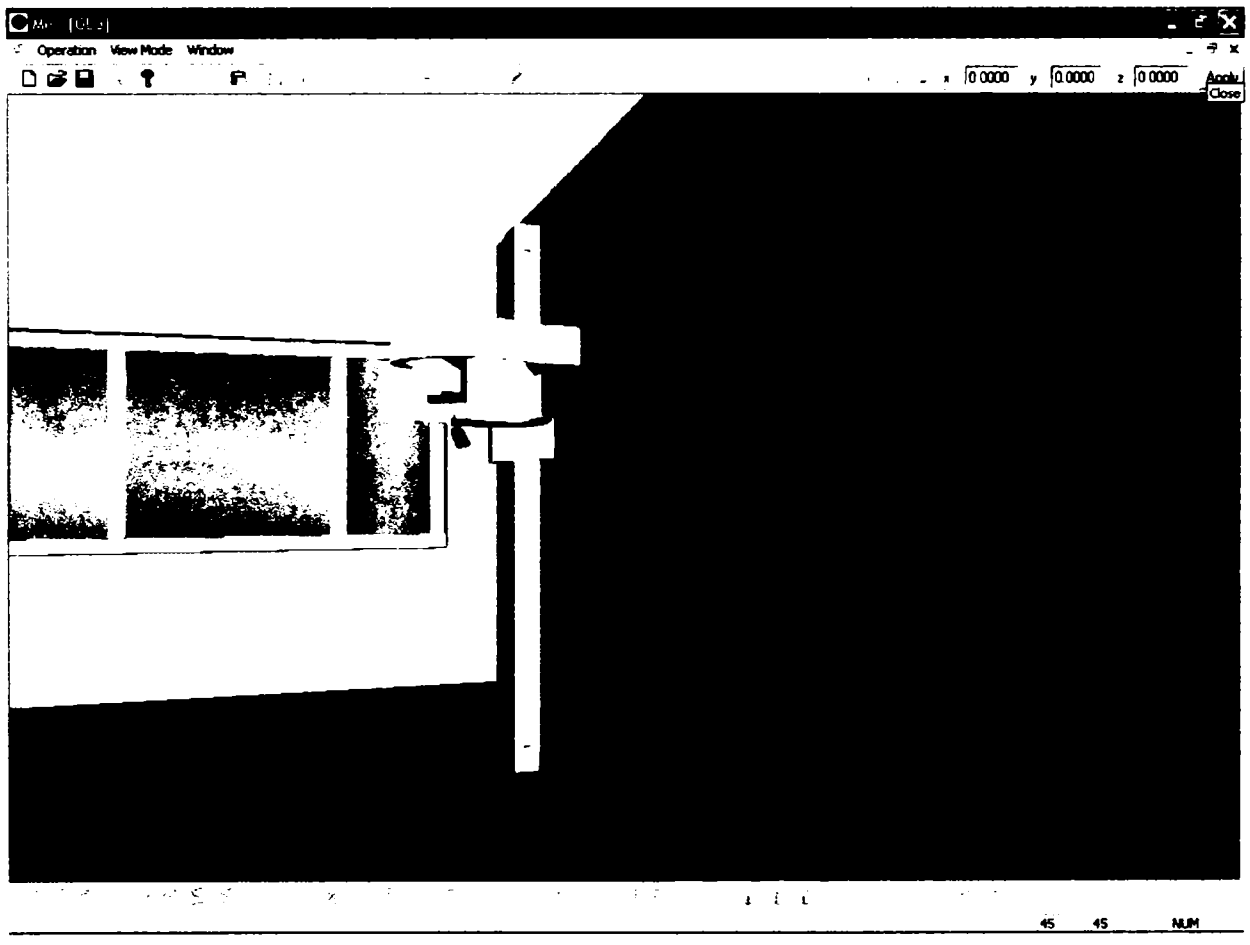

Fig 4.7 Reprezentarea camerei echipamentului de radiografiere ortopanoramică

- **Desenarea scenei**. Procesul cel mai complex în reprezentarea OpenGL este vizualizarea tuturor elementelor scenei. Acest lucru este influențat de modul în care sunt construite sursele de lumină, proprietățile materialelor, tipul de rendering folosit. Sursele de lumină sunt definite prin poziția și direcția lor, prin proprietățile lor intrinseci: strălucire, difuzie, model. Caracteristicile materialelor se referă la reflexie, strălucire, specularitate. Combinația acestor parametrii permite obținerea unei reprezentări cât mai aproape de realitate. Tot în cadrul acestui proces se construiesc matricele modelului care se aplică

elementelor scenei pentru o poziționare corectă. Aceste matrice sunt definite printr-o succesiune de translații și rotații care vor fi aplicate elementelor geometrice. Modificarea poziției scenei față de punctul de vedere implică schimbarea matricelor modelului. Parametrii care definesc în mod complet poziția scenei și care stau la baza calculului matricelor modelului sunt vectorul de translație pe cele trei direcții și unghiurile de rotație în raport cu fiecare axă. Toți acești parametrii au fost grupați în structura GLViewParam. Deoarece roboții din cadrul scenei prezintă o mișcare proprie, a fost necesară definirea pentru fiecare dintre ei a unui set propriu de valori de poziționare de tipul respectiv care vor fi aplicate suplimentar matricelor modelului. Mecanismul de gestionare a matricelor pus la dispoziție de către librăria OpenGL permite salvarea matricelor într-o stivă și citirea lor ulterioară prin perechea de funcții glPushMatrix(), glPopMatrix(). În acest mod matricea scenei este salvată iar calculul matricelor fiecărui robot vor porni de la această matrice de bază la care se adaugă translațiile și rotațiile proprii robotului.

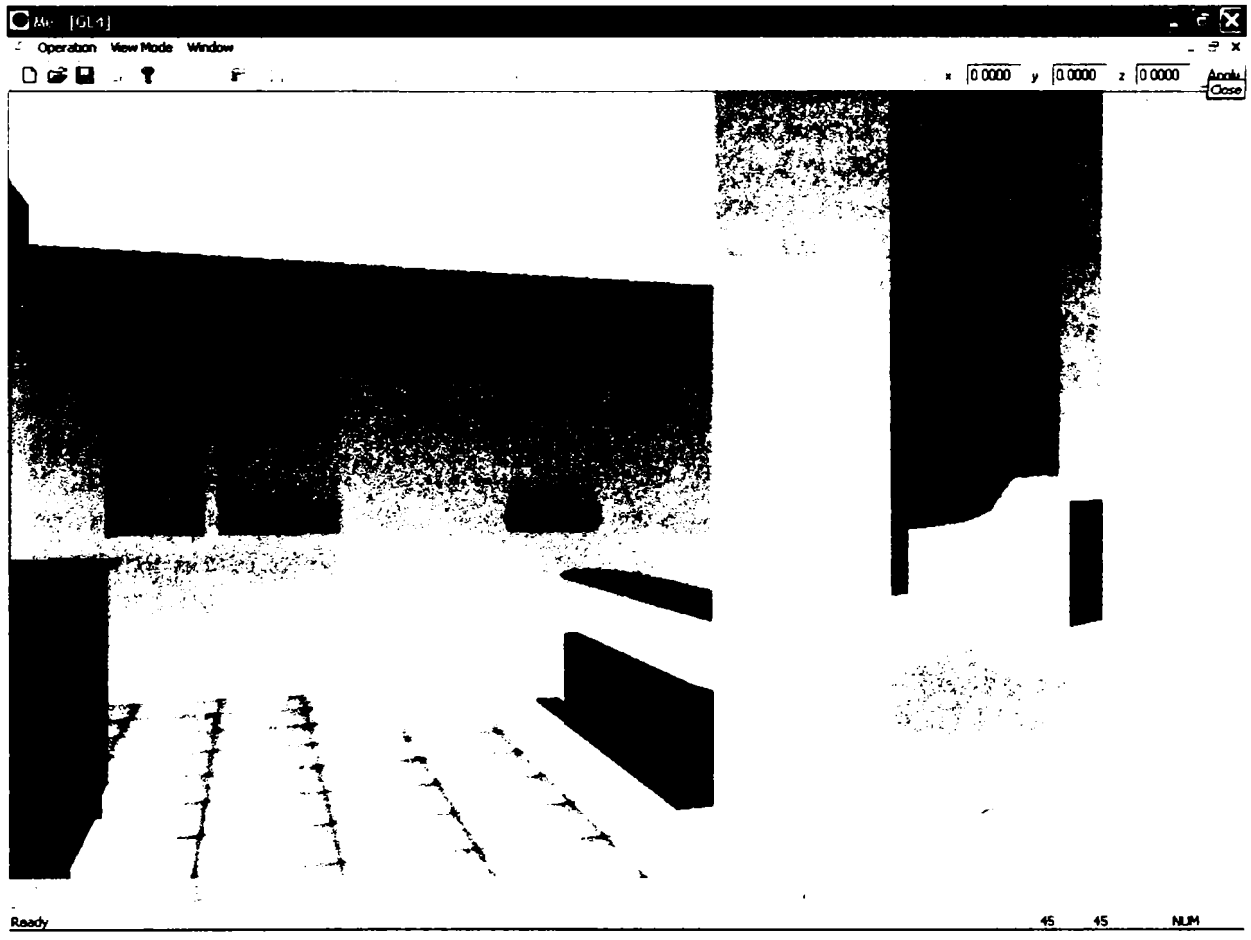

Fig 4.8 Reprezentarea camerei tomografului și a pupitrului de comandă

4.2.3 Proiectarea dinamicii spațiului virtual

Simularea mişcării roboților în spațiul virtual a fost realizată în două moduri. Prima modalitate este subordonată principiului mişcării libere în spațiul de lucru, fără îndeplinirea nici unei sarcini, singura restricție fiind evitarea obstacolelor. A doua modalitate se bazează pe algoritmii de planificare a mişcării roboților pentru atingerea unei ținte.

Mişcarea liberă a roboților

Laboratorul virtual cuprinde doi roboți care se pot deplasa liber în spațiul scenei. Mişcarea fiecărui robot este independentă de dinamica celorlalte corpuri din spațiul virtual. Fiecare robot îşi continuă deplasarea pe direcția sa atât timp cât nu întâlneşte un obstacol, un alt robot sau pereții laboratorului. La întâlnirea unui obstacol, robotul îşi defineşte o traiectorie de ocolire. Întâlnirea unui perete conduce la generarea unei traiectorii de reflexie pentru revenirea în spațiul scenei. Dacă robotul are o direcție perpendiculară pe direcția peretelui unghiul de revenire este ales aleator. La intersecția cu alt robot se încearcă determinarea unei traiectorii de evitare.

Deplasarea fiecărui robot este văzută ca un proces separat care se desfăşoară în paralel cu procesele corespunzătoare celorlalți roboți. O implementare software potrivită pentru sistemele multiproces o constituie mecanismul de funcții OnTimer() conținut de ferestrele Windows. Un robot îşi lansează propriul proces prin intermediul funcției SetTimer(). În acest mod la intervale de timp predefinite este chemată funcția de gestionare a mişcării robotului. Fiecărui robot mobil din spațiul scenei îi este alocat un obiect de tip CMeOpenGLRobot care grupează toate datele necesare construcției şi deplasării robotului. Acestea sunt identificatorul listei de elemente OpenGL a robotului direcția şi poziția inițială şi curente ale acestuia, unghiul de rotație, unghiul de reflexie, starea curentă a robotului în timpul mişcării, stiva cu pozițiile intermediare prin care a trecut robotul.

Deoarece verificarea poziției robotului față de corpurile din cadrul scenei virtuale se realizează prin operații de intersecții de elemente geometrice, care pentru forme complexe atât ale robotului cât şi ale obstacolelor consumă un timp procesor mare, s-a considerat robotul în cadrul acestor operații ca fiind definit de tetraedrul care înfăşoară forma robotului. În acest mod operațiile complexe se reduc la intersecții dintre suprafețe poligonale şi suprafețe curbe care necesită un timp de calcul mai mic. Lista celor şase poligoane ale tetraedrului este cuprinsă de asemenea în clasa CMeOpenGLRobot.

Mişcarea liberă a robotului a fost caracterizată prin cinci stări:

- FORWARD_STATE corespunzătoare deplasării înainte
- BACKWARD_STATE corespunzătoare deplasării înapoi în situația în care nu mai este posibilă înaintarea
- LEFT_ROTATION_STATE rotirea robotului la stânga pentru ocolirea obstacolelor
- RIGHT_ROTATION_STATE evitarea obstacolelor prin rotirea robotului la dreapta
- REFLEXION_STATE întâlnirea obstacolului conduce la schimbarea direcției de mers prin proces de reflexie

Trecerea sa dintr-o stare în altă stare este determinată de procesul de evitare a unui corp. Starea inițială a robotului este FORWARD_STATE, starea normală de deplasare către înainte, robotul nefiind în contact cu nici un alt corp din spațiul virtual. Dacă în timpul acestei mișcări de translație robotul întâlnește un obstacol, se încearcă ocolirea sa printr-o rotație la stânga sau la dreapta dacă este posibil. Starea robotului se modifică în mod corespunzător ea devenind fie LEFT_ROTATION_STATE fie RIGHT_ROTATION_STATE. Fiecare pas al evoluției robotului este memorat într-o stivă care stochează întreaga istorie a deplasării acestuia. Un pas de mișcare este definit prin starea curentă, poziția, direcția de mișcare și unghiul de rotație. Procesul de evitare prin rotirea la stânga sau la dreapta a robotului implică atât verificarea noncoliziunii în această nouă poziție cu elementele spațiului virtual cât și preîntâmpinarea mișcării robotului pe traiectorii deja încercate lucru care poate conduce la intrarea acestuia în cicluri infinite din care nu mai poate reveni. Verificarea noncoliziunii implică operații de intersecție cu toate elementele spațiului de lucru. Evitarea ciclurilor infinite se realizează prin verificarea poziției candidat în stiva traiectoriei. Dacă este regăsită, poziția este considerată invalidă.

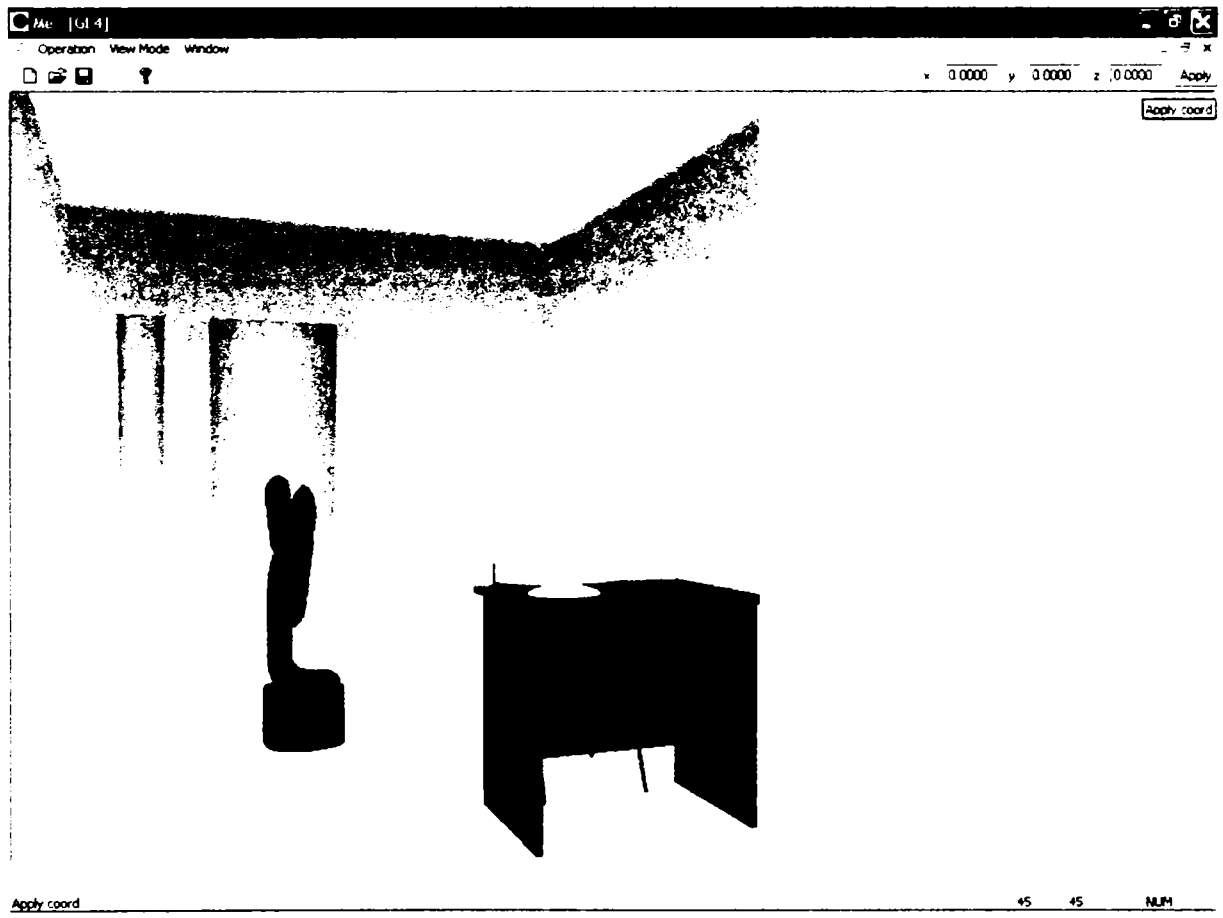

Fig 4.9 Deplasarea robotului în spațiul laboratorului virtual

În situația în care este imposibilă ocolirea obstacolului, ajungându-se într-o zonă închisă în ambele părți și toate încercările de evitare au eșuat, robotul revine pe traiectorie înapoi, aceasta fiind singura posibilitate de mișcare a sa în acest moment. Revenirea la

poziţia anterioară este denumită starea BACKWARD_STATE şi se face prin citirea din stivă a datelor pasului anterior. Noua stare a robotului este de asemenea salvată în stiva traiectoriei.

O stare specială o constituie starea de reflexie REFLECTION_STATE . Această stare este caracteristică situaţiei de coliziune a robotului cu un perete al laboratorului virtual. Identificarea calităţii de „perete" pentru un element geometric din spaţiul de lucru este realizată prin setarea tipului de interacţiune a elementului respectiv cu valoarea REFLECTION_INTERSECTION. Prin acest identificator, fiecărui element geometric construit i se stabileşte modul de comportare faţă de roboţii „mişcători". Se pot genera elemente care nu vor fi luate în considerare în procesul de mişcare, elemente care sunt considerate obstacole şi elemente care conduc la generarea unei mişcări de reflexie a robotului. În momentul în care coliziunea robotului s-a realizat cu un „perete" robotul trece în starea REFLECTION_STATE, se calculează unghiul de reflexie şi se încearcă rotirea treptată a robotului până atinge noua direcţie determinată când robotul îşi reia starea normală de mişcare către înainte. Dacă în timpul procesului de rotire se întâlneşte un obstacol robotul va trece într-una din stările de evitare descrise.

Metoda clasei document CMeOpenGLDoc care implementează mişcarea liberă a robotului este RobotAnimation(UINT nIDEvent). Desfăşurarea funcţiei urmăreşte etapele de modificare a stării robotului descrise. Funcţia VerifyRobotPosition() realizează testul de intersecţie dintre un robot şi toate elementele geometrice inclusiv ceilalţi roboţi din laboratorul virtual. Fiecare tip de element derivat din clasa CMeElement efectuează operaţia de intersecţie cu robotul prin suprascrierea funcţiei virtuale IntersectRobot(). Intersecţia unei curbe cu elementele constitutive ale robotului se reduce la operaţia de intersecţie dintre două curbe. Unul din operanzi este curba testată, al doilea operand fiind una din curbele longitudinale sau transversale a suprafeţelor care dau forma robotului. O metodă mai rapidă o constituie funcţia de verificare a coliziunii dintre tetraedrul înfăşurător al robotului şi curbă. Această funcţie testează dacă unul din punctele curbei se află în interiorul planelor ce alcătuiesc înfăşurătoarea robotului.

```
bool CMeCurve::IntersectRobot(CMeOpenGLRobot *pRobot)
{
    Vector n;
   double D;
   for(int i=0; i<curvePoints.GetSize(); i++)
   {
        bool intersect = true;
        for(int j=0;j<pRobot->robotPolygons.GetSize();j++)
        {
            pRobot->robotPolygons[j].GetND(n, D);
            if (curvePoints[i].SCALPRD(n) + D > -V_PPREC)
            {
                intersect = false;
                break;
            }
        }
        if (intersect)
            return true;
    }
    return false;
}
```

Intersecția dintre robot și o suprafață a fost implementată ca intersecția tetraedrului înfășurător al robotului cu suprafața, operație care de asemenea se realizează prin verificarea intersecției curbelor longitudinale și transversale ale suprafeței cu tetraedrul.

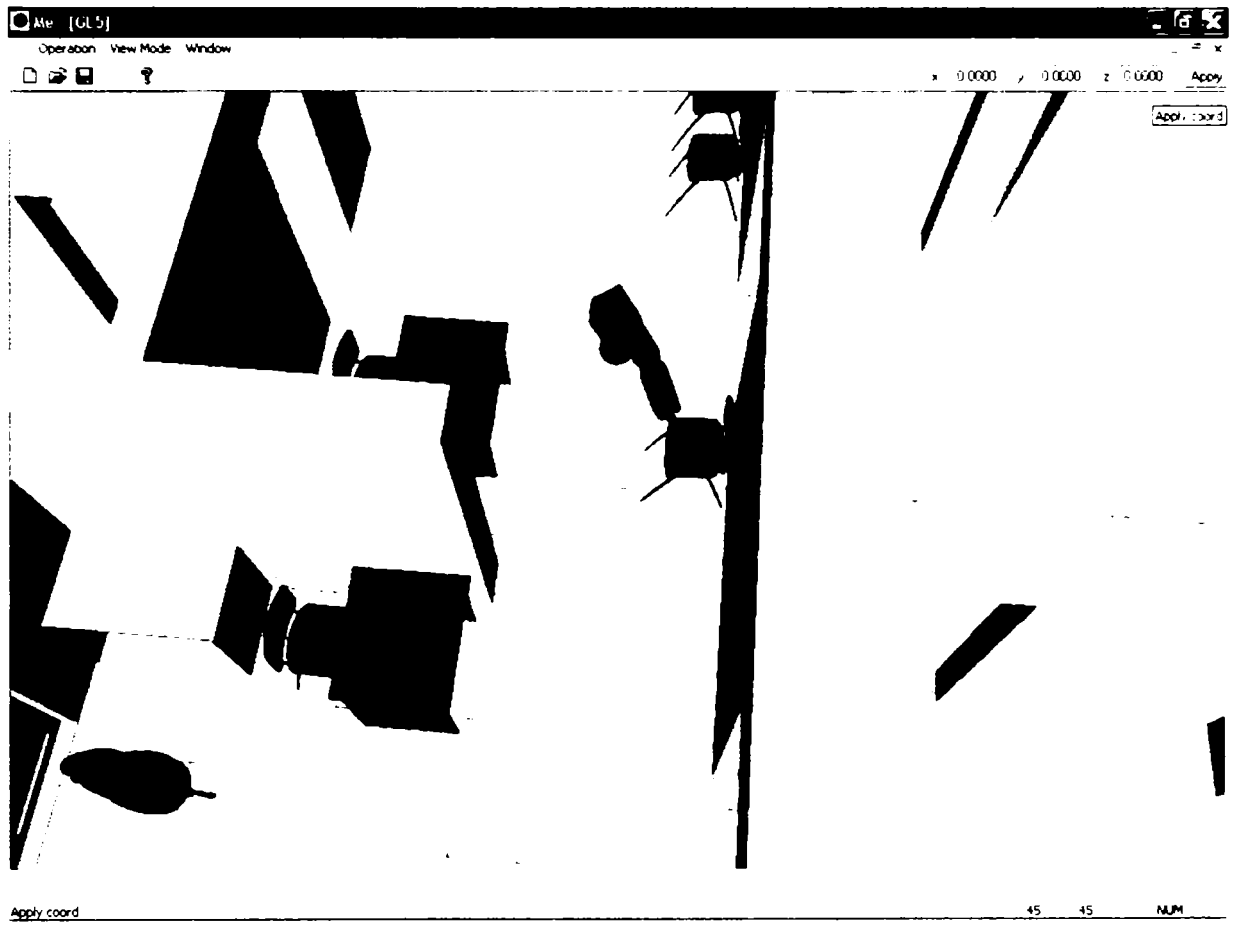

Fig 4.10 Intersecția dintre robot și suprafețele scaunului

```
bool CMeStandardSurface::IntersectRobot(CMeOpenGLRobot *pRobot)
{
    // testul de boundingbox a celor doua elemente
    if(pRobot->robotBBox & bbox)
        return false; // boundingbox-urile celor doua elemente nu se intersectează
    POSITION pos = surfaceCurves[0].GetHeadPosition();
    while (pos)
    {
        CMeSurfaceCurve *curve = surfaceCurves[0].GetNext(pos);
        if (curve->IntersectRobot(pRobot))
            return true;
    }
    pos = surfaceCurves[1].GetHeadPosition();
    while (pos)
    {
        CMeSurfaceCurve *curve = surfaceCurves[1].GetNext(pos);
        if (curve->IntersectRobot(pRobot))
            return true;
```

```
    }
    return false;
}
```

Algoritmul de intersecție a robotului cu suprafețele poligonale planare se bazează pe operația de intersecție dintre suprafața poligonală cu poligoanele înfășurătoare ale robotului. Dacă între unul din aceste poligoane și suprafață există o conexiune se consideră că robotul interacționează cu suprafața respectivă. Operația de intersecție dintre două poligoane este implementată de funcția membră a clasei CMePolygon, IntersectPolygon().

```
bool CMePolygon::IntersectPolygon(CMePolygon &pPolygon)
{
    Vector pi;
    for(int i=0;i<pPolygon.polygon.GetSize();i++)
    {
        for(int j = 0; j< pPolygon.polygon[i].GetSize()-1; j++)
        {
            double d1 = normal.SCALPRD(pPolygon.polygon[i][j]) + D;
            double d2 = normal.SCALPRD(pPolygon.polygon[i][j+1]) + D;
            if (d1 * d2 > 0)
                continue;
            else if (CalculPctInt_segment_plan(pi, pPolygon.polygon[i][j],
                pPolygon.polygon[i][j+1], normal, D))
            if (PointInter(pi) || PointOnPoly(pi))
                return true;
        }
    }
    return false;
}
```

Fig 4.11 Intersecția dintre robot și zidurile laboratorului

Funcția parcurge toate muchiile poligonului primit ca parametru şi pentru fiecare muchie testează dacă se intersectează cu planul suport al poligonului curent. În caz de intersecție se determină punctul de contact şi se verifică dacă acest punct aparține uneia din muchiile poligonului curent cu funcția PointOnPoly() sau în interiorul acestuia cu funcția PointInter(). În caz afirmativ, rezultă că există intersecție între cele două poligoane.

Toate aceste funcții de intersecție, mai puțin algoritmul de intersecție dintre două curbe, nu determină rezultatul intersecției elementelor geometrice ci faptul că acestea se intersectează sau nu. De asemenea fiecare din aceste funcții apelează ca o primă verificare la testul cutiilor minmax ale celor doi operanzi [1][77][80]. Neîndeplinirea acestui test exclude posibilitatea intersecției dintre elemente. Folosirea sa conduce la o îmbunătățire considerabilă a timpului de calcul al funcției VerifyPositionRobot().

Intersecția curbelor

Una dintre operațiile cele mai importante realizate între curbe este intersecția acestora. Procesul de intersecție permite determinarea poziției relative a celor două curbe una în raport cu cealaltă precum şi elementele comune ale acestora. Rezultatul intersecției este reprezentat de mulțimea punctelor comune celor două curbe. Principiul care stă la baza acestei operații este *subdiviziunea curbelor* [14][16].

Subdiviziunea curbelor Bézier

Subdiviziunea unei curbe este operația prin care o curbă este împărțită în două segmente de curbe mai mici. Deoarece majoritatea metodelor de proiectare descrise în lucrare pot fi reduse la curbe Bézier, operația de subdiviziune este dezvoltată pentru acest tip de curbe.

O curbă Bézier b^n este definită în general pe intervalul [0,1], dar domeniul său de variație poate fi orice interval [a,b]. Pentru orice valoare $c \in [0,1]$ se poate realiza împărțirea curbei originale în două segmente de curbă definite pe domeniile $[0,c]$, respectiv $[c,1]$. Curbele sunt complet caracterizate dacă pentru fiecare se determină poligonul Bézier de control propriu. Obținerea acestor poligoane este referită ca o subdiviziune a curbei Bézier.

Punctele Bézier necunoscute c_i sunt găsite simplu dacă este folosit principiul înfloririi (blossoming) [92][93]. Relația determină punctele Bézier ale unei curbe polinomiale definită pe intervalul $[a,b]$. Pentru domeniul $[0,c]$ punctele Bézier sunt:

$$c_i = b[0^{<n-i>}, c^{<i>}] \tag{4.1}$$

Fiecare punct c_i al poligonului de control este obținut îndeplinind i paşi ai algoritmului de Casteljau, cu referire la valoarea c, în notație neînflorită rezultă:

$$c_i = b_0^i(s) \tag{4.2}$$

Aceasta relație este numită *formula de subdiviziune* pentru curbele Bézier

[11][12][96]. Algoritmul de Casteljau nu realizează doar calculul punctelor curbei $b^n(c)$, ci furnizează şi punctele de control ale acesteia. Datorită proprietății de simetrie, punctele de control pentru segmentul de curbă corespunzător intervalului $[c,1]$ sunt date de b_j^{n-j}.

Metoda SubdivisionBezierCurve a clasei CMeCurve implementează algoritmul de subdiviziune pentru o curbă Bézier. Segmentele de curbă rezultate sunt returnate prin intermediul parametrilor leftCurve şi rightCurve.

```
short CMeCurve::SubdivisionBezierCurve(CMeCurve **leftCurve, CMeCurve **rightCurve)
{
    short nr_points, degree;
    double c;
    CArray<CVertex, CVertex> b;
    if(curveType != BEZIER)
        return 1;
    // crearea poligoanelor de control pentru cele doua subdiviziuni ale curbei
    MeSettings &settings = ((CMeApp *)AfxGetApp())->meSettings;
    (*leftCurve) = new CMeCurve(settings, myDoc);
    (*rightCurve) = new CMeCurve(settings, myDoc);
    // numărul de puncte ale poligonului de control
    nr_points = controlPolygon.GetSize();
    degree = nr_points-1;
    //setarea dimensiunilor poligoanelor de control rezultate
    (*leftCurve)->controlPolygon.SetSize(nr_points);
    // inițializarea boundingbox-ului
    (*leftCurve)->InitPerspBBox();
    (*rightCurve)->controlPolygon.SetSize(nr_points);
    // inițializarea boundingbox-ului
    (*rightCurve)->InitPerspBBox();
    // setarea dimensiunii tabloului temporar
    b.SetSize(nr_points);
    b.Copy(controlPolygon);
    // subdivizarea se face la mijlocul intervalului [0,1]
    c = 0.5;
    // determinarea punctelor de control pentru cele doua curbe subdivizate
    (*leftCurve)->controlPolygon[0] = controlPolygon[0];
    (*leftCurve)->CreatePersp((*leftCurve)->controlPolygon[0]);
    (*rightCurve)->controlPolygon[degree] = controlPolygon[degree];
    (*rightCurve)->CreatePersp((*rightCurve)->controlPolygon[degree]);
    for(int i = 1; i <= degree; i++)
    {
        for(int j = 0; j <= degree - i; j++)
            b[j] = b[j] * (1-c) +  b[j+1] * c;
        (*leftCurve)->controlPolygon[i] = b[0];
        (*leftCurve)->CreatePersp((*leftCurve)->controlPolygon[i]);
        // datorita simetriei punctele poligonului din dreapta sunt bj, n-j
        (*rightCurve)->controlPolygon[degree - i] = b[degree-i];
        (*rightCurve)->CreatePersp((*rightCurve)->controlPolygon[degree - i]);
    }
    // test de coliniaritate pentru punctele poligoanelor de control rezultate
    (*leftCurve)->TestCollinear();
    (*rightCurve)->TestCollinear();
    return 0;
}
```

În locul subdivizării curbei Bézier aceasta poate fi *extrapolată* prin generarea unui segment de curbă în prelungirea curbei originale pe un domeniu $[l,d]$ Punctele poligonului de control sunt date de relația :

$$d_j = b[l^{<n-j>}, d^{<j>}] = b^{l}_{n-j}(d) \qquad (4.3)$$

Subdiviziunea pentru curbele Bézier, deși menționată de de Casteljau, a fost demonstrata riguros de E. Staerk [35] [93].

Intersecția curbelor Bézier și a curbelor B-spline cubice

Subdivizarea poate fi realizată în mod repetat: se poate subdiviza o curbă la $t = 0.5$ si apoi curbele rezultate pot fi de asemenea împărțite din nou la jumătatea domeniului lor. Dacă operația este continuată, după k nivele de subdiviziune, vor exista 2^k poligoane Bézier, fiecare descriind un mic arc din curba originala. Aceste poligoane conduc la curba inițială dacă numărul de nivele de subdiviziune crește după cum a fost demonstrat de Lane si Riesenfeld [63]. Convergența acestui proces repetat de subdiviziune este foarte rapidă [13] si are multe aplicații practice. Un rezultat important îl constituie intersecția curbelor.

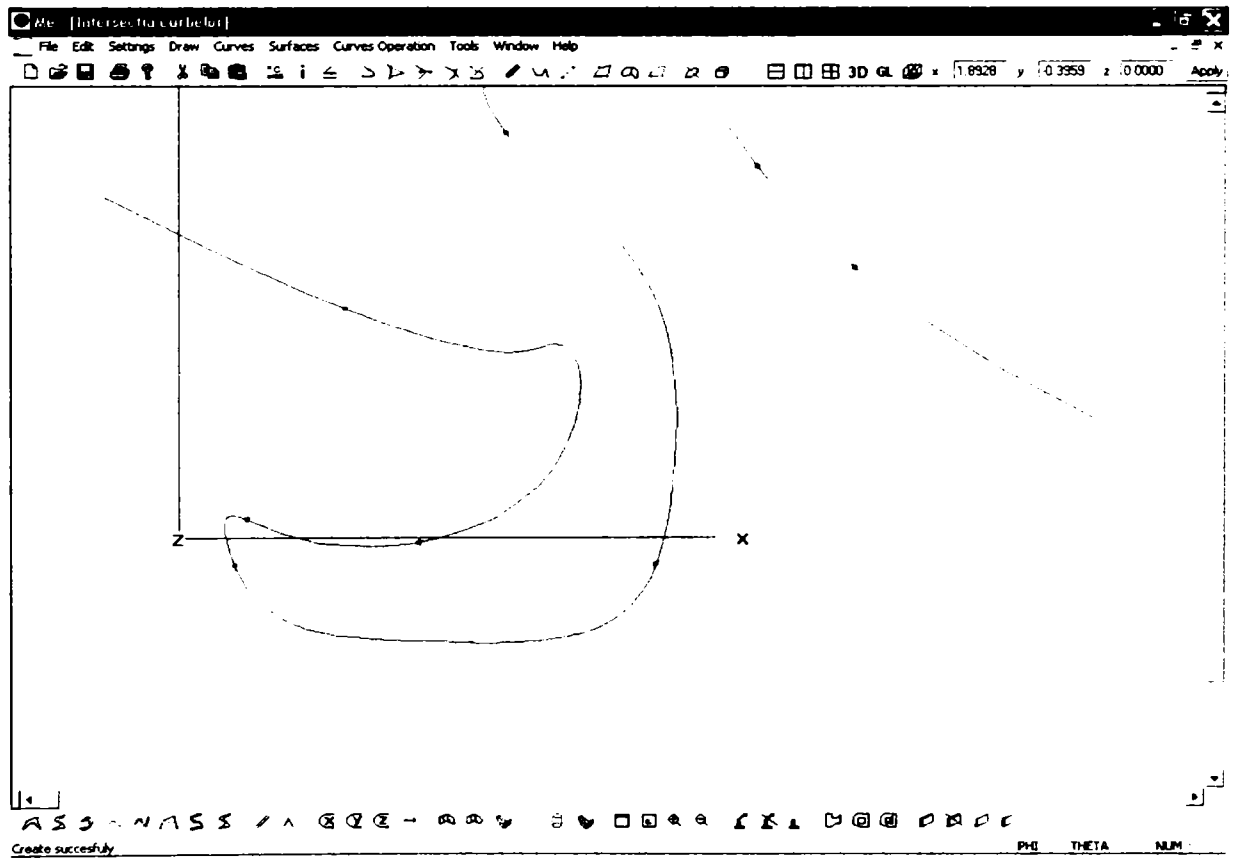

Fig 4.12 Determinarea punctelor de intersecție a curbelor B-spline

Un alt element semnificativ în procesul de intersecție este testul de *respingere banală* sau *testul de boundingbox* aplicat celor doi operanzi [80]. Acest test a fost descris la definirea proprietății de acoperire convexă a curbelor (2.2). Prin calculul cutiilor minmax a celor două curbe şi verificarea interferenței lor se poate stabili cu certitudine nonintersecția acestora. Dacă cele două cutii minmax sunt complet disjuncte nu va exista nici o intersecție între curbe.

Algoritmul de intersecție foloseşte testul de respingere banală ca element decizional [87]. Dacă cei doi operanzi interferează, există posibilitatea de intersecție. Ambele curbe sunt subdivizate la $t=\frac{1}{2}$ iar testul de boundingbox este aplicat pe rând tuturor segmentelor rezultate. Dacă două dintre aceste segmente interferează algoritmul se repetă din nou. În cadrul acestui proces recursiv pot rezulta puncte de control aproape coliniare ce pot fi înlocuite cu o dreaptă. Astfel operația de intersecție a curbelor se va reduce la intersecții de segmente minime ce aproximează poligoanele de control ale subdiviziunilor. Punctele de intersecție sunt determinate din aceste intersecții liniare.

Curbele B-spline cubice pot fi definite ca o colecție de curbe Bézier pe porțiuni. Procesul de intersecție dintre două curbe B-spline extinde algoritmul de intersecție al curbelor Bézier la întreaga colecție de curbe pe porțiuni ale fiecărui operand. Fiecare segment Bézier al primei curbe este intersectat cu toate segmentele Bézier ale celeilalte curbe. De asemenea algoritmul de intersecție al curbelor Bézier poate fi utilizat pentru toate curbele care pot fi reduse la forma Bézier sau Bézier pe porțiuni.

Funcția IntersectCurves() realizează operația de intersecție a două curbe, punctele rezultate fiind returnate prin parametrul intersectPoints.

```
void CMeCurve::IntersectCurves(CMeCurve *curve, CArray<CVertex,CVertex> &intersectPoints)
{
    short i = 0, j = 0, k;
    CMeCurve fCurve, sCurve;

    fCurve.myDoc = myDoc;
    sCurve.myDoc = myDoc;
    if((curveType == BEZIER || curveType == BEZIER_HORNER) &&
        (curve->curveType == BEZIER || curve->curveType == BEZIER_HORNER))
    {
        // ambele curbe sunt de tip Bézier
        IntersectBezierCurves(curve, intersectPoints);
        return;
    }
    fCurve.controlPolygon.SetSize(4);
    sCurve.controlPolygon.SetSize(4);
    if(curveType == BEZIER || curveType == BEZIER_HORNER)
    {
        j = 0;
        while(j < curve->bezierPoints.GetSize() - 1)
        {
            for(k = 0; k < 4; k++)
                sCurve.controlPolygon[k] = curve->bezierPoints[k+j];
            j += 3;
            IntersectBezierCurves(&sCurve, intersectPoints);
        }
```

```
}
else   if(curve->curveType == BEZIER || curve->curveType == BEZIER_HORNER)
{
    while(i < bezierPoints.GetSize() - 1)
    {
        for(k = 0; k < 4; k++)
            fCurve.controlPolygon[k] = bezierPoints[k+i];
        i += 3;
        fCurve.IntersectBezierCurves(curve, intersectPoints);
    }
}
else
{
    // se iau pe rând poligoanele Bézier pe porțiuni ale fiecărei curbe
    // si se intersectează
    while(i < bezierPoints.GetSize() - 1)
    {
        for(k = 0; k < 4; k++)
            fCurve.controlPolygon[k] = bezierPoints[k+i];
        i += 3;
        j = 0;
        while(j < curve->bezierPoints.GetSize() - 1)
        {
            for(k = 0; k < 4; k++)
                sCurve.controlPolygon[k] = curve->bezierPoints[k+j];
            j += 3;
            fCurve.IntersectBezierCurves(&sCurve, intersectPoints);
        }
    }
}
```

Algoritmul de intersecție a curbelor Bézier este implementat de funcția IntersectBezierCurves.

```
void CMeCurve::IntersectBezierCurves(CMeCurve *curve, CArray<CVertex,CVertex>
&intersectPoints)
{
    short rez, del1 = 0, del2 = 0;
    CVertex P;
CMeCurve *leftCurve1 = NULL, *leftCurve2 = NULL, *rightCurve1 = NULL, *rightCurve2 = NULL;
    // determinarea boundingbox-ului celor doua curbe
    SetElementBBox();
    curve->SetElementBBox();
    // testul de rejecție banala
    if(bbox & curve->bbox)
    {
        // cele doua curbe nu se intersectează
        return;
    }
    // se verifica daca poligoanele  de control a celor doua curve nu
    // s-au redus la cate un segment de dreapta
    if(controlPolygon.GetSize() == 2 && curve->controlPolygon.GetSize() == 2)
    {
        rez = IntersectEdge(controlPolygon[0], controlPolygon[1],
```

```
        curve->controlPolygon[0], curve->controlPolygon[1], P);
        if(!rez)
        {
            // s-a găsit un punct de intersecție între cele doua curbe
            myDoc->ComputePersp(P);
            intersectPoints.Add(P);
        }
        return;
    }
    // se împarte prima curba
    if(controlPolygon.GetSize() == 2)
        leftCurve1 = this;
    else
    {
        // se realizează subdiviziunea curbei
        SubdivisionCurve(&leftCurve1, &rightCurve1);
        del1 = 1;
    }
    // se împarte cea de a doua curba
    if(curve->controlPolygon.GetSize() == 2)
        leftCurve2 = curve;
    else
    {
        // se realizează subdiviziunea curbei
        curve->SubdivisionCurve(&leftCurve2, &rightCurve2);
        del2 = 1;
    }
    // se reia operația de intersecție
    leftCurve1->IntersectBezierCurves(leftCurve2, intersectPoints);
    if(rightCurve2)
        leftCurve1->IntersectBezierCurves(rightCurve2, intersectPoints);
    if(rightCurve1)
    {
        rightCurve1->IntersectBezierCurves(leftCurve2, intersectPoints);
        if(rightCurve2)
            rightCurve1->IntersectBezierCurves(rightCurve2, intersectPoints);
    }
    // eliberarea memoriei alocate in urma divizării curbelor
    if(del1)
    {
        delete leftCurve1;
        delete rightCurve1;
    }
    if(del2)
    {
        delete leftCurve2;
        delete rightCurve2;
    }
}
```

Planificarea mişcării robotului

Cea de a doua modalitate de mişcare a roboţilor implementată în cadrul aplicaţiei o reprezintă calcul traiectoriei roboţilor pentru atingerea unei ţinte din spaţiul virtual.

În general, robotul inteligent trebuie să fie capabil să-şi planifice propriile mişcări, şi anume să decidă automat ce mişcări să execute pentru a realiza o sarcină specificată prin aranjamentul iniţial şi final al obiectelor din spaţiul de lucru [7] [44]. Crearea roboţilor autonomi este o sarcină majoră în Robotică. Cu excepţia anumitor domenii limitate, nu este real să se anticipeze şi să se descrie explicit toate mişcările pe care robotul trebuie să le execute pentru a realiza sarcina cerută. Chiar şi în cazurile când o asemenea descriere este posibilă, este util să se încorporeze dispozitive de planificare automată a mişcărilor în sistemele de programare off-line ale robotului. Aceasta permite utilizatorului să specifice sarcini stabilind mai degrabă ce doreşte să realizeze, decât cum să o facă. Deci, robotul trebuie să realizeze mişcarea dorită şi să-şi activeze diferitele mecanisme în acord cu sarcina cerută [4][23] [70].

Planificarea mişcărilor unui robot prezintă o varietate neaşteptată de aspecte dificile de calcul. De fapt, inteligenţa operativă pe care oamenii o utilizează inconştient pentru a interacţiona cu mediul înconjurător, necesară percepţiei şi planificării mişcării, este foarte dificil de reprodus într-un program de calcul. Astfel, o problemă importantă în planificarea mişcărilor este complexitatea algoritmilor de calcul [5][41].

Aşa cum s-a precizat, planificarea mişcării este o problemă în crearea roboţilor autonomi, dar nu este singura. Aceasta interacţionează cu alte probleme importante, ca de exemplu: analiza cinematică directă şi inversă, modelul dinamic direct şi invers, generarea traiectoriilor, controlul mişcării în timp real, sistemul şi planificarea sarcinii [45][47].

Scopul definirii problemei de bază a planificării mişcărilor este de a izola anumite probleme centrale şi de a le studia în profunzime înainte de a considera anumite dificultăţi adiţionale.

Cea mai simplă problemă de planificare presupune că robotul este singurul obiect în mişcare în spaţiul de lucru, care nu posedă proprietăţi dinamice evitând astfel problemele temporale. Aceste consideraţii transformă problema planificării "fizice" a mişcării într-o problemă pur geometrică. Mai mult, se consideră că robotul este singurul solid rigid a cărui mişcare este limitată doar de obstacole.

Cu aceste simplificări, problema de bază a planificării traiectoriilor unui robot se poate formula astfel:

- Fie A un singur solid rigid (robotul) care se mişcă într-un spaţiu Euclidian W, numit spaţiu de lucru, reprezentat prin R^N, cu N=2 sau 3; mişcarea lui A nu este limitată de nici o restricţie cinematică.
- Fie $B_1,\ldots,B_q$ obiecte rigide fixe (obstacole) distribuite în poziţii bine determinate în spaţiul de lucru W.
- Cunoscând poziţia şi orientarea, la momentul iniţial, ale robotului, precum şi poziţia finală şi orientarea finală pentru acesta în spaţiul de lucru W, să se genereze o traiectorie T, care să specifice o secvenţă continuă de poziţii şi orientări ale lui A, pornind de la configuraţia (poziţia şi orientare) iniţială, evitând contactul cu obstacolele B_j şi terminând în configuraţia finală.
- Dacă o astfel de traiectorie nu există, trebuie să se raporteze eroare.

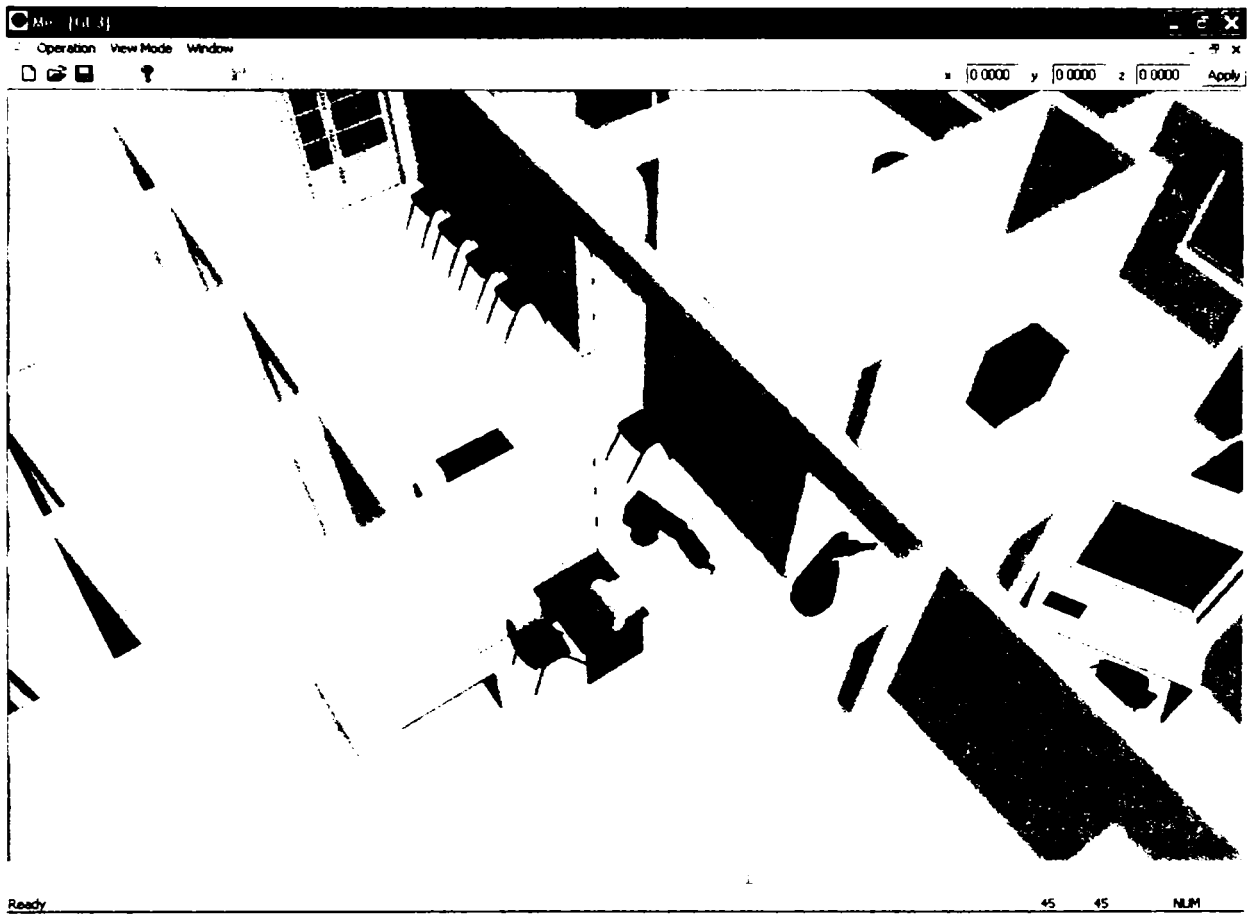

Fig 4.13 Mișcarea roboților în spațiul de lucru

Este evident că, deși problema de bază a planificării este super simplificată, ea este totuși o problemă dificilă cu mai multe soluții și cu extensii directe spre probleme mai complicate [52].

Formularea problemei de bază a planificării traiectoriilor se bazează pe anumite presupuneri care limitează semnificativ utilizarea soluțiilor. Este foarte dificil să se reducă o problemă reală de robotică la problema de bază, să se rezolve această problemă și să se adapteze soluțiile obținute astfel încât să se potrivească condițiilor problemei reale.

Problema de bază a planificării traiectoriei presupune că robotul parcurge cu exactitate traiectoria generată de planificator. De asemenea, se presupune că, atât geometria robotului, cât și geometria și pozițiile obstacolelor sunt cunoscute cu exactitate. În realitate, nici o problemă de planificare nu satisface aceste ipoteze. Mai mult, controlul roboților și modelele geometrice ale acestora nu sunt exacte. În condițiile în care robotul nu deține informații apriorice despre spațiul de lucru, acesta trebuie să se bazeze, în timpul execuției, pe sistemul său senzorial pentru înregistrarea informațiilor necesare realizării sarcinii. Se impune astfel explorarea spațiului de lucru și rezolvarea problemei de planificare în prezența incertitudinilor [23][24] [51][53].

Stabilirea unei hărți de navigare în spațiul considerat se poate face fie independent de orice acțiune a robotului prin memorarea unei configurații date, care de regulă,

reprezintă starea inițială a spațiului de lucru, fir pe baza informațiilor înregistrate de sistemul senzorial.

Pe baza cunoştinţelor acumulate despre spațiul de lucru, acesta se împarte în celule independente. Aceste celule reprezintă zonele admise respectiv interzise pentru robot. Această operație se numeşte *modelarea spațiului de lucru al robotului.* Pe o astfel de modelare se bazează procesul propriu-zis de planificare a mişcărilor [43][46].

Pentru a se realiza o modelare corectă a spațiului de lucru, unul din principalele obiective care trebuie luate în considerare este ansamblul dimensiunilor robotului. Acestea vor modifica dimensiunile obstacolelor, rezultând aşa numitele obstacole expandate, după care robotul se poate trata ca un punct în mişcare [50]. Având reprezentate obstacolele în forma în care ele se iau în considerare, se poate proceda la împărțirea spațiului de lucru în zone accesibile şi zone interzise.

Deoarece în cadrul aplicației elementul central îl constituie modelarea cât mai exactă a formei roboților şi a obiectelor componente ale spațiului virtual prin suprafețe curbe, transformarea obstacolelor în obstacole expandate prin luarea în considerare a dimensiunilor robotului conduce la o pierdere a proprietăților structurii acestora dobândite prin acest tip de modelare. Din acest motiv în teză la modelarea spațiului de lucru al robotului au fost luate în considerare obstacolele reale, neexpandate.

Pentru determinarea punctelor esențiale ale traiectoriei metoda de planificare utilizată a fost metoda descompunerii celulare exacte [39][42]. Principiul care stă la baza acestui algoritm constă în:

- descompunerea spațiului liber al robotului într-o colecție de regiuni care nu se suprapun, a căror reuniune este exact spațiul virtual al laboratorului
- construirea şi verificarea grafului conexității care reprezintă relațiile de adiacență între celule; în cazul reuşitei, rezultatul verificării este o secvență de celule, denumită *canal*, care conectează celula ce conține configurația inițială cu celula ce conține configurația finală
- extragerea traiectoriei considerate favorabilă.

Traiectoria finală de mişcare a robotului se obține prin interpolarea acestor puncte cu una din metodele de generare a curbelor interpolatoare descrise în capitolele anterioare. Deoarece spațiul de lucru nu a fost modelat prin expandare, deplasarea robotului pe curba rezultată conduce la coliziuni nedorite cu obiectele laboratorului virtual. Pentru rezolvarea acestei probleme, la fel ca şi în cazul mişcării libere, robotul intră într-un proces de evitare a obstacolelor. Elementul distinctiv îl constituie faptul că după ocolirea obiectelor robotul încearcă să revină pe traiectoria calculată pentru a-şi îndeplini sarcina. Dacă devierea de la traiectoria planificată este semnificativă, situație care indică o imposibilitate de continuare a mişcării comandate, secvența aleasă este greşită iar robotul încearcă să-şi îndeplinească sarcina alegând un alt canal determinat de metoda de planificare.

Descrierea algoritmului de mişcare al robotului

Primul pas al procesului de mişcare orientată al robotului îl constituie descompunerea spațiului virtual în regiuni disjuncte şi generarea grafului conexității. Funcția care realizează aceste operații se numeşte GenerateGrahpConnection(). Pornind

de la structura laboratorului virtual se determină mai întâi conturul spațiului de lucru delimitat de pereți și contururile obstacolelor din interiorul său. Acest lucru se realizează prin proiecția 2D a tuturor obiectelor în planul suprafeței podelei laboratorului. Se obține o formă poligonală cu o secvență de găuri reprezentate de poligoanele obstacolelor. Descompunerea convex poligonală a spațiului liber se efectuează prin operația de triunghiularizare aplicată poligonului obținut. Rezultă o succesiune de suprafețe triunghiulare disjuncte a căror reuniune constituie spațiul liber al robotului. Având această descompunere se generează graful de conexiune al spațiului de lucru. Nodurile grafului sunt suprafețele triunghiulare. Implementarea grafului de conexiune este realizată printr-un tablou de obiecte de tipul GraphConnectionNode asociate nodurilor. Clasa cuprinde geometria formei triunghiulare, centrul de greutate precum și pointerii către nodurile adiacente. Un nod obișnuit conține trei conexiuni. Pentru un nod situat pe frontiera spațiului de lucru sau lângă un obstacol, numărul de conexiuni este mai mic. Funcția care construiește graful conexiunilor se numește GenerateConnectionGraph().

```
void RobotWorkspace::GenerateConnectionGraph(CArray<GraphConnectionNode,
    GraphConnectionNode> &graphConnection)
{
GraphConnectionNode nod;
short nrPcte;
int i,j,k,l, kk[2], ll[2], mat[3][3] = {{-1,0,2},{0,-1,1},{2,1,-1}};
graphConnection.SetSize(nr_triangles);
for(i=0;i<nr_triangles;i++)
{
   for(j=0;j<3;j++)
  {
      nod.point[j][0] = vertices[triangles[i][j]][0];
      nod.point[j][1] = vertices[triangles[i][j]][1];
  }
  nod.SetG();
  graphConnection.SetAt(i,nod);
}
for(i=0;i<nr_triangles-1;i++)
{
   if (graphConnection[i].HasAllNeighbors())
      continue;
  for(j=i+1;j<nr_triangles;j++)
  {
      if (graphConnection[j].HasAllNeighbors())
         continue;
      nrPcte = 0;
      for(k=0;k<3;k++)
      {
         for(l=0;l<3;l++)
         {
            Vector pk = graphConnection[i].point[k];
            Vector pl = graphConnection[j].point[l];
            if (graphConnection[i].point[k] == graphConnection[j].point[l])
            {
               kk[nrPcte]=k;
               ll[nrPcte]=l;
               nrPcte++;
```

```
                    break;
                }
            }
            if (nrPcte==2)
            {   graphConnection[i].neighbor[mat[kk[0]][kk[1]]] = &graphConnection[j];
                graphConnection[j].neighbor[mat[ll[0]][ll[1]]] = &graphConnection[i];
                break;
            }
        }
    }
}
```

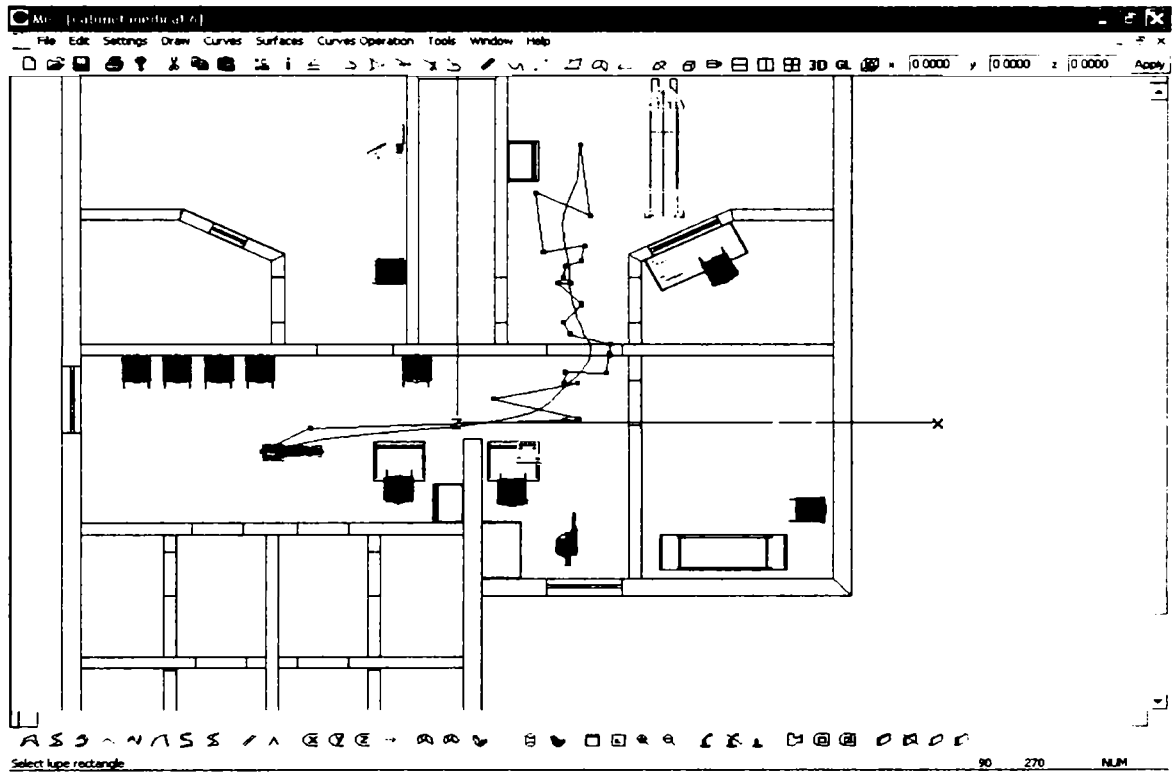

Fig 4.14 Traiectoriei robotului în deplasarea sa către tomograf

Următorul pas este determinarea unei secvențe de noduri adiacente care să unească poziția curentă a robotului cu punctul țintă. Trecerea de la secvența de noduri la puncte este simplă, punctele sunt determinate pe muchia comună a nodurilor adiacente la o distanță egală cu lungimea robotului față de cel mai apropiat punct al muchiei în raport cu punctul țintă. Având aceste puncte, calea simplă de rezolvare a problemei o constituie calculul unei curbe interpolatoare sau aproximante a lor și deplasarea robotului de-a lungul acestei curbe. Funcția care realizează aceste operații este GenerateRobotTrack(), iar deplasarea robotului de-a lungul curbei este implementată prin funcția MoveRobotToDestination(). Curba traiectoriei este salvată în cadrul proiectului pentru a putea fi vizualizată cu ușurință.

O implementare mai realistă necesită o procesare suplimentară care constă în evitarea obstacolelor care intervin de-a lungul traiectoriei alese. Funcția corespunzătoare acestei operații este RobotAnimationOnTrack(). Principiul care stă la baza acestui proces este descompunerea sa în subprocese, fiecare din aceste subprocese având ca țintă

atingerea unui punct esențial de pe traiectorie. Fiecare subproces gestionează mișcarea robotului de la punctul anterior al traiectoriei către următorul punct al traiectoriei cu restricția de a ocoli obstacolele existente. Trecerea de la o etapă la alta se face printr-o reorientare a robotului către următoarea țintă intermediară. Dacă unul din aceste subprocese nu a fost finalizat se reia algoritmul prin alegerea unei noi secvențe de noduri care să conecteze poziția inițială cu ținta robotului.

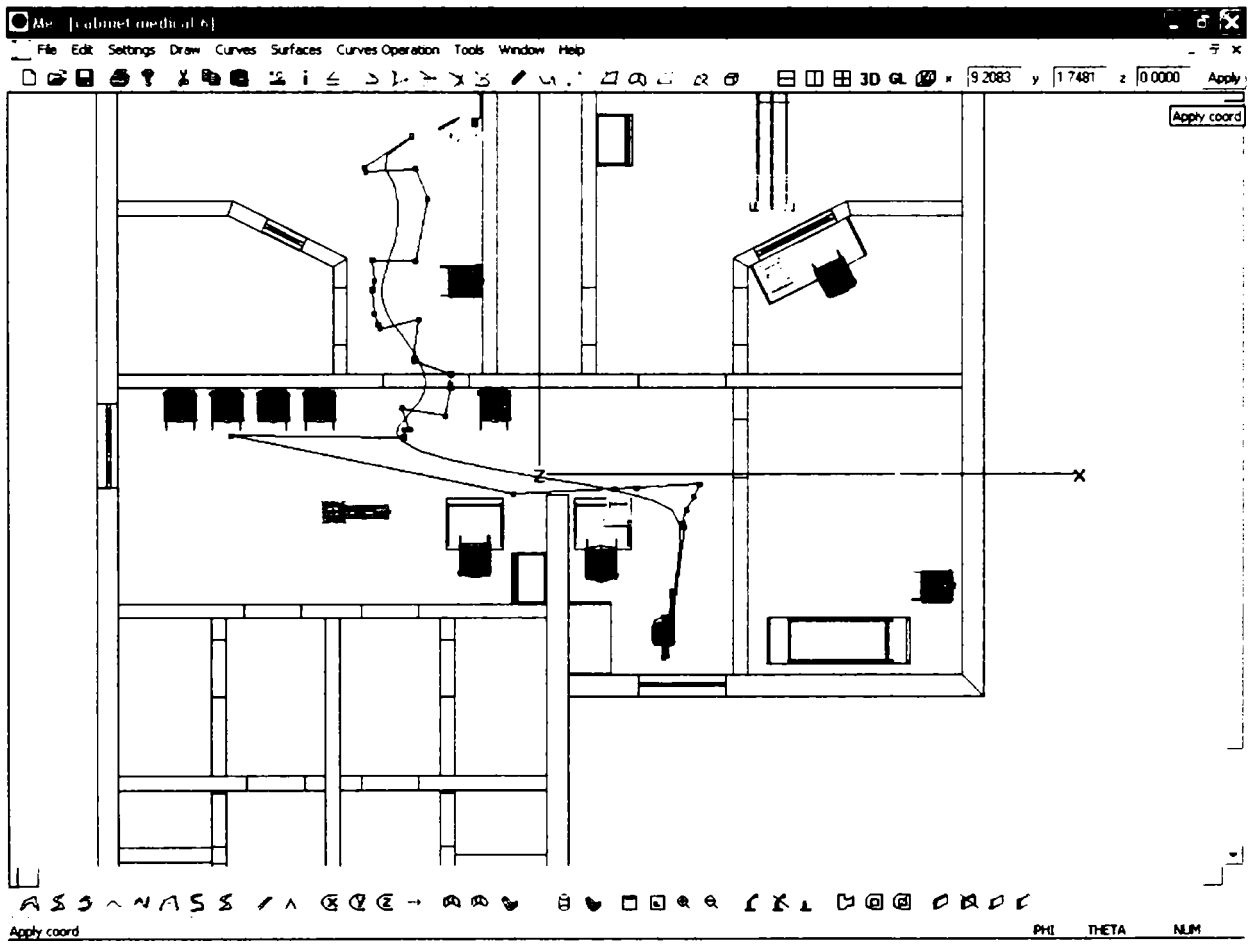

Fig 4.15 Traiectoria robotului în deplasare sa către aparatul de radiografiere

Luând în considerare aceste lucruri, la fel ca și la mișcarea liberă, deplasarea robotului este organizată în mai multe stări:

- REORIENTATION_STATE starea de revenire a robotului pe direcția de mișcare corectă
- NORMAL_STATE starea de mișcare liniară a robotului pe drumul calculat
- TRANSITION_STATE stare provocată de întâlnirea unui obstacol, prin care robotul încearcă să de depărteze de acesta
- AVOID_STATE stare cauzată de ciocnirea cu un obstacol în timpul procesul de tranziție, în care robotul încearcă, prin rotiri la stânga sau dreapta să ocolească obstacolul
- ABORT_STATE stare determinată de eșuarea procesului de

mișcare a robotului

- SUCCES_STATE stare determinată de întâlnirea punctelor intermediare din drumul calculat sau a țintei finale

. Starea inițială a robotului, la pornirea procesului de mișcare este reorientarea denumită REORIENTATION_STATE. Robotul are o anumită direcție inițială care diferă de direcția de deplasare determinată, iar acest lucru necesită o reorientare a sa pe direcția cerută. Această operație constă dintr-o rotire treptată până când se întâlnește direcția dorită. Dacă operația se încheie cu succes robotul trece în starea normală NORMAL_STATE, de mișcare liniară către punctul țintă intermediar de pe traiectorie. Dacă în timpul procesului de reorientare se întâlnește un obstacol robotul trece în starea de tranziție TRANSITION_STATE, stare în care robotul încearcă să se îndepărteze de obstacol printr-o deplasare liniară pe direcția sa curentă. Dacă starea de tranziție s-a încheiat cu succes robotul revine în starea de reorientare, dacă însă se constată că robotul a deviat prea mult față de direcția pe care dorește să ajungă, se consideră că operația de orientare a robotului a eșuat și nu este posibilă deplasarea robotului pe direcția dorită, procesul de deplasare este întrerupt și se încearcă determinarea unei noi căi către punctul țintă al robotului. Dacă în timpul procesului de tranziție robotul întâlnește un obstacol, se trece în starea de evitare a obstacolului AVOID_STATE. În această stare robotul ocolește prin rotiri la stânga sau la dreapta obstacolul întâlnit sau dacă acest lucru nu este posibil robotul se retrage. Operația se finalizează cu succes în cazul evitării corpului iar robotul trece în starea de tranziție pentru a se îndepărta de obstacol. Întregul proces de mișcare a robotului până la atingerea punctelor țintă intermediare sau a punctului final, precum și tranzițiile robotului dintr-o stare de mișcare în altă stare sunt implementate prin funcția RobotAnimationOnTrack().

Fiecare dintre roboții mobili are de îndeplinit o sumă de sarcini specifice. Primul robot deservește pacientul pe care în conduce pe traseul de investigare. Pacientul este preluat de la intrarea în laboratorul de imagistică și condus către pupitrul de înregistrare pacienți figura 4.16.

Fig 4.16 Vizualizarea primelor etape ale procesului de investigare a unui pacient

Pacientul este apoi condus spre echipamentul de radiografiere ortopanoramică pentru a i se realiza investigațiile specifice figura 4.17

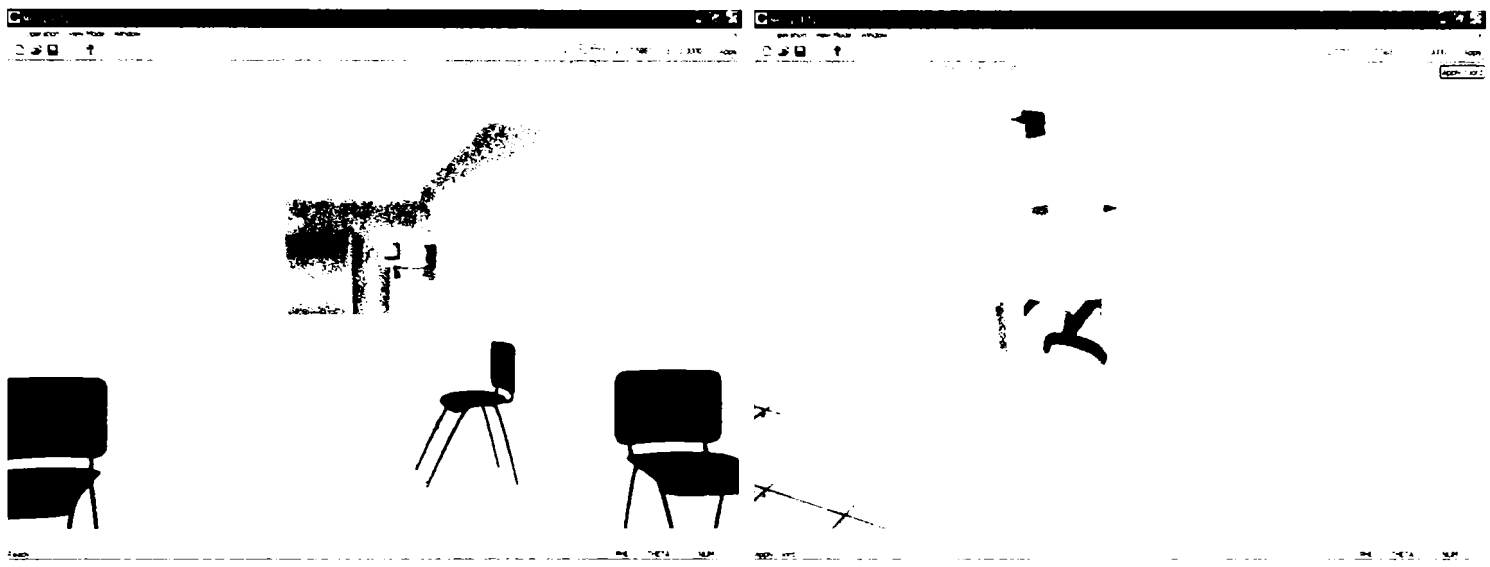

Fig 4.17 Realizarea investigațiilor cu echipamentul de radiografie

Următorul pas în procesul de investigație îl constituie deplasarea pacientului către computerul tomograf, iar apoi realizarea tomografiei acestuia figurile 4.18, 4.19. Fiecare din acești pași au fost definiți ca ținte intermediare pentru robotul mobil. Acesta își calculează traiectoria de mișcare la fiecare pas, și evitând obstacolele fixe și mobile încearcă să-și îndeplinească ținta.

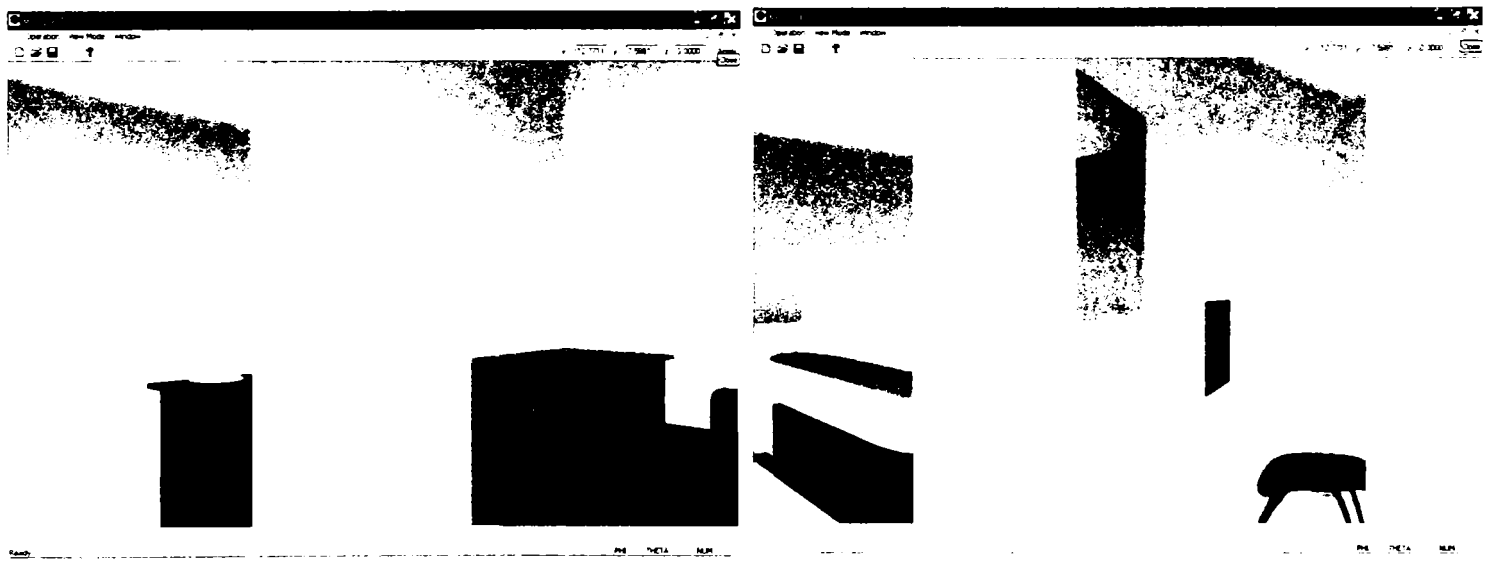

Fig 4.18 Vizualizarea deplasării robotului către computerul tomografului

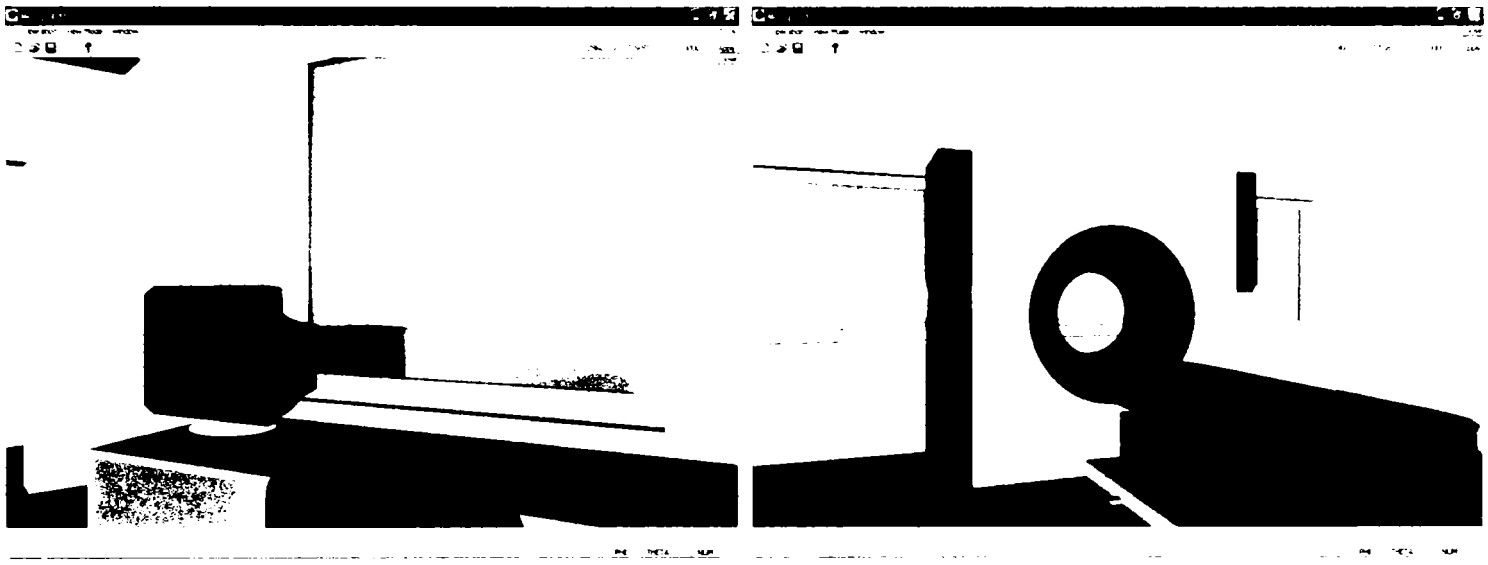

Fig 4.19 Investigarea pacientului cu ajutorul tomografului

Ultima etapă în traseul de investigație al pacientului o constituie vizualizarea rezultatelor testelor cu ajutorul echipamentelor de achiziție de imagine, printere, etc. Pacientul este condus de către robot către această zonă a laboratorului virtual figura 4.20.

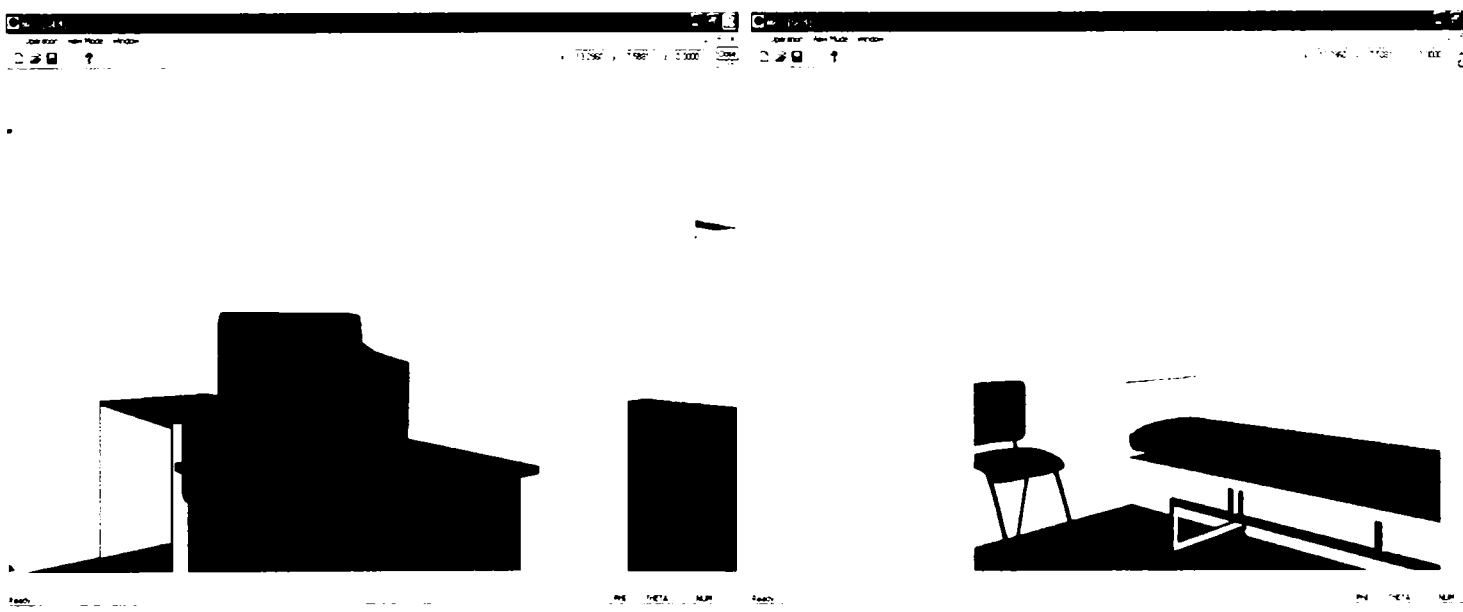

Fig 4.20 Obținerea rezultatelor investigațiilor sau refacerea pacientului în camera ATI

În situația în care pacientul se simte rău în timpul procesul de investigație este condus de către robot spre camere ATI pentru refacere figura 4.20.

Cel de-al doilea robot mobil deserveşte personalul medical şi cel auxiliar deplasându-se între diferitele echipamente de achiziţie de imagine, printere şi pupitrul de înregistrare pacienţi.

Cap 5. Concluzii şi contribuţii personale

5.1. Concluzii

Prezenta teză de doctorat se constituie ca o pledoarie în favoarea modelării cu ajutorul calculatorului a elementelor din lumea reală. Modelele virtuale reprezintă „copii" cât mai fidele ale obiectelor încercând să surprindă cât mai exact proprietăţile esenţiale ale acestora. Importanţa proiectării spaţiilor virtuale este dată de capacitatea de a testa comportarea elementelor reale în diferite situaţii precum şi verificarea unor noi principii şi reguli ce se doresc a fi proiectate.

Simularea software a comportării sistemelor complexe prezintă o dezvoltare deosebită în ultimul timp datorate în special creşterii capacităţii de calcul şi a posibilităţilor de integrare a sistemelor de calcul, fiind prezentă în aproape toate domeniile lumii reale. Robotica constituie un spaţiu în care metodele de modelare au un rol foarte important.

Teza de doctorat se prezintă ca o nouă încercare în acest domeniu vast prin realizarea modelării formei roboţilor cu ajutorul curbelor şi suprafeţelor curbe. S-a dorit stabilirea unei conexiuni între sfera CAD/CAM a proiectării asistate de calculator şi lumea roboţilor. Scopul tezei îl constituie aplicarea principiilor modelării geometrice la proiectarea formei roboţilor precum şi posibilitatea utilizării acestor modele la simularea mişcării roboţilor în cadrul unui spaţiu de lucru virtual proiectat de asemenea prin aceleaşi metode.

Lucrarea propune o analiză şi o sinteză riguroasă a principiilor de proiectare cele mai cunoscute în domeniul curbelor şi suprafeţelor. Partea teoretică a tezei a fost structurată pe două capitole realizându-se o separaţie între metodele de modelare a curbelor şi cele de suprafeţe. Stabilirea acestei distincţii a fost realizată atât pentru claritatea dezvoltării analizei cât şi datorită faptului că proiectarea suprafeţelor se fundamentează pe metodele dezvoltate pentru curbe. Algoritmii matematici şi principiile de proiectare au fost selectate astfel încât să servească scopului final ales. Alături de metodele implementate sunt prezentate din punct de vedere teoretic şi alte metode pentru scoaterea în evidenţă a avantajelor şi dezavantajelor metodelor utilizate. Partea practică a lucrării surprinde exemplificarea tuturor metodele descrise prin construirea unui spaţiu virtual care modelează Baza de Cercetare cu Utilizatori Multiplii *Centrul de Modelare a Protezării şi Intervenţiilor Chirurgicale asupra Scheletului Uman - CMPICSU* din cadrul Universităţii „Politehnica" din Timişoara. Atât elementele componente ale Laboratorului de Imagistică cât şi roboţii, sunt alcătuite din suprafeţe curbe sau plane. Pornind de la modelul matematic obţinut, s-a încercat realizarea proiectării mişcării roboţilor în scopul îndeplinirii sarcinilor stabilite pentru aceştia. Procesul de mişcare se bazează în mod fundamental pe metodele de determinare a interacţiunii dintre elementele spaţiului de lucru. Aceasta a implicat studierea şi definirea algoritmilor de intersecţie dintre curbe şi suprafeţe.

Utilizarea practică a principiului de modelare prin curbe şi suprafeţe în cazul concret al realizării spaţiului virtual al CMPICSU reprezintă o probă a viabilităţii şi posibilităţilor acestei încercări de modelare.

5.2. Contribuţii

Contribuţia personală în cadrul tezei de doctorat este legată de realizarea unei modelări geometrice prin curbe şi suprafeţe curbe a roboţilor în contextul creării unui Laborator Virtual deservit de roboţi mobili. Această temă generoasa prin aspectele teoretice şi practice pe care le pune, a urmărit să demonstreze că orice ambient şi orice instalaţie complexă de tip robot, pot fi modelate prin aceleaşi tehnici. Algoritmii de modelare şi conducere a mişcării au rezultat în urma unui studiu bibliografic amănunţit al metodelor de proiectare asistate de calculator şi prin implementarea software adecvată orientată pe obiect. Tematica abordată, şi contribuţiile personale ale autorului se pot sintetiza astfel:

Capitolul I, Introducere cuprinde o definire a tematicii lucrării şi o descriere succintă a modului de abordare şi structurare a tezei. *Autorul a realizat o analiză riguroasă a domeniului modelării geometrice orientate cu precădere spre scopul final al lucrării.*

Capitolul II, Principii de modelare geometrică prin curbe tratează domeniul proiectării curbelor. Structurarea capitolului a fost realizată pe 5 secţiuni urmărindu-se o ierarhizare a metodelor după complexitatea şi funcţionalitatea lor. Studiul bibliografic al fiecărui algoritm este însoţit de soluţia software proprie, aleasă pentru materializarea sa.

Domeniile abordate şi rezolvate original în această parte a lucrării sunt:

- **Noţiuni introductive** aduce ca principală contribuţie *Identificarea şi definirea principalelor noţiuni care sunt utilizate în domeniul modelării geometrice* atât din spaţiul matematicii vectoriale cât şi al analizei matematice.

- **Curbe Bézier,** defineşte una dintre cele mai generale şi puternice clase de curbe. Pot fi evidenţiate câteva contribuţii originale ale autorului.
  - *Se realizează o abordare bidirecţională a acestor curbe atât geometrică cât şi polinomială.* Dezvoltarea geometrică este mai intuitivă şi oferă o imagine mai clară asupra modului de construcţie în timp ce abordarea polinomială oferă o cantitate mare de informaţii despre proprietăţile acestor curbe
  - *Au fost create funcţiile care implementează algoritmul de Casteljau şi algoritmul Bezier-Horner.* Metoda de Casteljau a fost gândită spre a minimiza timpul de execuţie în pofida unui consum mai mare de memorie. Algoritmul Bezier-Horner utilizează metoda de regrupare a termenilor din forma Bernstein a curbei Bézier.
  - *Autorul a subliniat importanţa proprietăţii de invarianţă afină în cadrul unui mediu de proiectate CAD,* proprietate care permite aplicarea rapidă a transformărilor geometrice asupra curbelor
  - *Studiul principiului înfloririi* constituie un element important pentru procesul de intersecţie al curbelor.

- **Curbe spline în forma Bézier,** descrie fundamentele teoretice ale generării curbelor polinomiale complexe. Contribuţiile originale ale autorului materializate de-a lungul secţiunii sunt sintetizate în continuare.

- *Studiul principiilor de racordare a curbelor Bézier* reprezintă baza proiectării curbelor spline. Generarea corectă a formei curbelor compuse implică menţinerea proprietăţilor de continuitate şi derivabilitate de-a lungul întregului domeniu de definiţie
- *O altă contribuţie este identificarea şi analiza relaţiilor dintre punctele de Boor şi punctele poligonului Bézier pe porţiuni.* Aceste relaţii permit abordarea curbelor spline cubice ca o colecţie de curbe Bézier şi utilizarea algoritmilor definiţi pentru aceste curbe
- *Funcţia de generare a curbelor spline cubice* cuprinde formulele de calcul a punctelor poligonului pe porţiuni atât pentru curbe deschise cât şi pentru curbe închise decizia de alegere a algoritmului fiind dată de tipul poligonului de Boor.

- **Metode de interpolare**, în care sunt prezentate unele din principalele metode de construcţie a curbelor interpolatoare. Această clasă de curbe permite urmărirea precisă a formei poligonului de control, toate vârfurile acestuia fiind situate pe curbă.
  - *Autorul realizează o sinteză originală a principiilor de interpolare cu definirea avantajelor şi dezavantajelor lor.* Accentul a fost pus pe metodele geometrice de proiectare a curbelor interpolatoare.
  - *Unificarea tratării curbelor Hermite constituie o contribuţie originală a autorului,* prin definirea formei polinomiale şi matriciale a acestui tip de curbe
  - *Implementarea originală a algoritmului de interpolarea cubică pe porţiuni,* metodă care constituie una dintre cele mai importante principii de proiectare a curbelor. Funcţia de calcul a unui sistem tridiagonal ocupă un rol important în cadrul algoritmului

- **Curbe B-spline de grad n,** în cadrul căreia se dezvoltă una dintre cele mai generale metode de proiectare a curbelor eficientă în cazul modelării formelor complexe care implică un număr mare de puncte de control. O calitate esenţială a acestor curbe o constituie controlul local al formei acestora, modificările realizate într-un punct al curbei nu afectează întreaga curbă. Se pot remarca câteva dintre contribuţiile proprii ale autorului.
  - *Proiectarea curbelor B-spline de grad n a fost urmărită atât din perspectivă geometrică cât şi din perspectivă analitică.* A fost realizată o descriere atentă a acestor principii de construcţie a curbelor de grad n necesară în procesul de implementare pentru o transpunere optimă pe calculator.
  - *Studiul procesului de inserţie de noduri şi a algoritmului de Boor de generare B-spline* reprezintă un element esenţial în cadrul algoritmului de Boor. Inserţia repetată a aceluiaşi nod de un număr egal cu gradul curbei, conduce la obţinerea punctului corespunzător de pe curba B-spline

- *A fost creată funcţia care implementează algoritmului CoxDeBoor*, algoritm bazat pe principiul inserţiei de noduri. Alegerea nodurilor curbei constituie un element esenţial în stabilirea formei acesteia
- *Procesul de generare a curbelor B-spline a fost completat cu elemente specifice matematicii polinomiale*, prin definirea unei baze pentru o curbă de grad n. Acest lucru a permis definirea unui nou algoritm de generare a curbelor B-spline.
- *Funcţia de construcţie a curbelor B-spline de grad n* ilustrează o modalitate de transpunere software a algoritmului bazat pe funcţiile polinomiale spline. Pentru fiecare segment al curbei, punctele curbei se calculează folosind formula recursivă a funcţiilor B-spline.

Capitolul III, Principii de modelare geometrică prin suprafeţe tratează ansamblul metodelor de construcţie a suprafeţelor. Capitolul este structurat pe 6 secţiuni principale în cadrul cărora sunt sintetizate unele dintre metodele de proiectare consacrate în acest domeniu. Studiul bibliografic al teoriei a condus la ideea grupării metodelor în două clase distincte, cea a suprafeţelor parametrice şi a suprafeţelor neparametrice (produs tensorial). O abordare detaliată şi riguroasă a fost realizată asupra celei de-a doua categorii, fiind impusă de capacitatea superioară a acestor algoritmi de a modela suprafeţele complexe. Analizele şi sintezele originale conţinute în acest capitol se referă la:

- **Noţiuni generale,** secţiune care creionează aspectele generale legate de modelele suprafeţelor
  - *Autorul realizează identificarea şi definirea principalelor paradigme ale construcţiei suprafeţelor*, fiind realizată o clasificare a tipurilor de suprafeţe precum şi o descriere a caracteristicilor geometrice ale acestora: vectori twist, vectori normali la suprafeţe, etc.
  - *Stabilirea şi proiectarea claselor de suprafeţe şi a modalităţilor de reprezentare a acestora sunt contribuţii originale ale autorului*, acestea permit includerea suprafeţelor în ansamblul aplicaţiei. Se realizează o prezentare a claselor de bază care cuprind elementele definitorii ale suprafeţelor. Autorul a încercat generarea unor entităţi reutilizabile care să satisfacă într-o proporţie cât mai mare conceptele legate de abstractizarea datelor, încapsularea şi ascunderea informaţiei

- **Reprezentarea parametrică a suprafeţelor analitice** abordează elementele fundamentale ale suprafeţelor parametrice întâlnite în proiectarea şi modelarea suprafeţelor.
  - *Autorul a creat o serie de clase şi metode originale pentru implementarea următoarelor tipuri de suprafeţe parametrice:*
    - *suprafeţe plane, poligonale*
    - *suprafeţe riglate*
    - *suprafeţe conice*
    - *suprafeţe de revoluţie*
    - *suprafeţe cilindrice*

Introducerea suprafeţelor poligonale a fost impusă de necesitatea modelării obiectelor planare din cadrul spaţiului virtual: pereţi, geamuri, uşi, etc. Celelalte suprafeţe parametrice prezintă ca date de intrare una sau mai multe curbe ceea ce conduce la generarea unor suprafeţe cu grad de complexitate ridicat pe o anumită direcţie.

- *Setul de imagini care exemplifică fiecare tip de suprafaţă, realizate cu ajutorul mediul CAD de proiectare a suprafeţelor, este de asemenea o contribuţie originală.*

- **Suprafeţe Bézier produs tensorial** reprezintă una din cele mai generale clase de suprafeţe sintetice utilizate în domeniul proiectării. Sunt subliniate elementele abordate şi rezolvate în mod original de autor.
  - *S-a realizat studiul interpolării biliniare ca punct de plecare în generarea suprafeţelor aproximante.* Se stabileşte un paralelism între modul de analiză al curbelor şi cel al suprafeţelor, evidenţiindu-se similitudinile şi diferenţele. Modelarea curbelor constituie în mod cert punctul de plecare în procesul de proiectare al suprafeţelor
  - *Autorul a proiectat şi implementat ierarhia claselor* corespunzătoare suprafeţei produs tensorial şi a peticelor de suprafaţă.
  - *Prezentarea principiilor de racordare a peticelor de suprafaţă şi transpunerea lor software* este un punct important în ansamblul proiectării suprafeţelor produs tensorial. La fel ca şi în cazul curbelor suprafeţele trebuie să-şi conserve proprietăţile de continuitate şi derivabilitate. S-a utilizat o modalitate originală de conexiune a peticelor sub forma unei hărţi de petice.

- **Suprafeţe bicubice Hermite,** descrie principiile teoretice de construcţie a clasei de suprafeţe Hermite. Principala trăsătură a acestui tip de suprafaţă o constituie faptul că datele iniţiale cuprind alături de harta punctelor de control şi informaţii de derivabilitate în aceste puncte.
  - *Studiul şi analiza matriceală a suprafeţelor bicubice Hermite reprezintă contribuţia proprie a autorului.* Calculul matricial oferă avantaje deosebite la implementarea software prin realizarea unor funcţii rapide, element important în cazul simulării dinamicii unui sistem.

- **Suprafeţe B-spline. Interpolare B-spline**, în cadrul secţiunii se realizează o sinteză a metodelor de proiectare a suprafeţelor B-spline aproximante şi interpolatoare care permit modelarea suprafeţelor de complexe.
  - *O contribuţie originală a autorului o constituie implementarea software a pânzelor B-spline de grad n,* implementare care se defineşte ca o extindere bidirecţională a algoritmului de construcţie a curbelor B-spline de grad arbitrar.
  - *Autorul realizează unu studiu comparativ a suprafeţelor B-spline şi Bézier,* un rol important fiind atribuit analizei conversiei B-spline cubic – petice

Bézier. Acest lucru stabileşte legătura intrinsecă dintre aceste clase de suprafeţe.

- *Studiul tipurilor de vectori twist este de asemenea o contribuţie proprie a autorului.* Crearea unei reţele de curbe cubice pe porţiuni C^1 sau C^2 de-a lungul punctelor interpolatoare nu determină în mod complet suprafaţa. Este necesară construcţia vectorilor twist pentru a se obţine o proiectare completă a suprafeţei interpolatoare
- *Autorul a utilizat noţiunile teoretice pentru a crea o funcţie originală de interpolare B-spline bicubică.* Algoritmul este conceput matricial, construcţia suprafeţei bazându-se pe rezultatele obţinute la interpolarea cubică spline

- **Suprafeţe Coons,** secţiunea cuprinde o analiză succintă a clasei de suprafeţe Coons, care se constituie ca forme ale "*interpolării transfinite*".
  Autorul descrie principalele modele de suprafeţe Coons fundamentele matematice şi caracteristicile acestei clase de suprafeţe.

Capitolul IV, Proiectarea unui laborator virtual deservit de roboţi mobili tratează modul în care s-a proiectat laboratorul virtual deservit de roboţi mobili. Sunt descrise tehnicile de implementare software abordate, caracteristicile de proiectare şi vizualizare ale laboratorului, modul de realizare a dinamicii spaţiului virtual şi regulile de planificare a mişcării roboţilor. Capitolul are un caracter preponderent practic, în cadrul său fiind ilustrat modul de aplicare a principiilor de modelare prezentate în capitolele anterioare. Componentele a căror dezvoltare se urmăreşte în cadrul acestei părţi a tezei sunt:

- **Tehnicile de implementare software,** în cadrul acestei secţiuni sunt descrise fundamentele programării orientate pe obiect şi a trăsăturilor specifice programării sub Windows, Visual C++, tehnici care constituie un suport perfect pentru dezvoltarea aplicaţiei de modelare.
  - *Autorul defineşte principalele paradigme ale orientării spre obiecte.* Un punct important în programarea obiectuală este structurarea procesului de dezvoltare a unei aplicaţii utilizând tehnologia orientată pe obiecte. Dezvoltarea corectă a proiectului implică urmărirea tuturor etapelor procesului de construcţie.
  - *Sinteza elementelor esenţiale ale programării sub Window din perspectiva aplicaţiei de modelare geometrică reprezintă o contribuţie originală a autorului.* Programarea sub Windows a condus la un grad înalt de standardizare a tuturor elementelor componente: ierarhia ferestrelor, dispecerizarea mesajelor, unitatea grafică de interfaţă, orientarea pe resurse, etc.
  - *Studiul sintetic original al mediului de programare Microsoft Visual Studio C++ şi al librărie grafice OpenGL aparţine de asemenea autorului.* Nucleul principal al mediului Visual C++ este librăria de clase şi structuri *Microsoft Foundation Class Library* care oferă o varietate mare de obiecte

ce pot fi utilizate în cadrul unei aplicaţii. OpenGL se constituie ca unul din cele mai puternice mecanisme de vizualizare a elementelor grafice. Algoritmii incluşi în această librărie permit realizarea unor reprezentări realiste. În cadrul secţiunii sunt subliniate principalele operaţii grafice pentru proiectarea unei scene cu librăria grafică OpenGL.

- **Laboratorul virtual de imagistică** sintetizează elementele importante ale proiectării şi implementării aplicaţiei.
  - *Ideea generării entităţilor laboratorului şi organizarea ierarhică a acestora sunt contribuţii originale ale autorului.* Varietatea mare de elemente ce pot fi vizualizate precum şi necesitatea tratării unitare a acestora a condus la stabilirea unei structuri polimorfice de date şi elaborarea unei ierarhii de clase care să surprindă la fiecare nivel de ierarhizare proprietăţile specifice fiecărui element.
  - *Modul original de implementarea a mişcării libere a roboţilor mobili în spaţiul virtual* se bazează pe algoritmii de intersecţie creaţi pentru toate tipurile de elemente ce populează acest spaţiu. Mişcare se constituie ca o succesiune de stări prin care trece robotul în funcţie de poziţia sa în raport cu celelalte elemente ale spaţiului de lucru.
  - *Autorul a implementat un set original de algoritmi de intersecţie dintre robot şi suprafeţele curbe şi cele poligonale.* Gradul de generalitate şi reutilizabilitate a funcţiilor permite folosirea lor pentru orice tip de suprafaţă modelată în cadrul aplicaţiei. Algoritmii stabilesc doar existenţa sau nu a intersecţiei fără calculul curbelor de intersecţie. În cadrul fiecărui algoritm se aplică mai întâi testul de coliziune prin verificarea cutiilor minmax a celor doi operanzi. Acest test constituie o metodă rapidă de verificare a non coliziunii elementelor.
  - *Funcţia de intersecţie a curbelor Bézier şi a curbelor B-spline cubice este de asemenea o realizare originală a autorului.* Algoritmul recursiv de calcul al intersecţiei se bazează pe principiul subdiviziunii curbelor Bézier, derivat din forma polară a acestora. Funcţia de intersecţie este dezvoltată şi pentru curbe B-spline cubice, care pot fi convertite la o secvenţă de curbe Bézier pe porţiuni

- **Sinteza noţiunilor de bază a planificării mişcării roboţilor** descrie modul de în care s-a proiectat şi implementat mişcarea planificată a roboţilor în spaţiul virtual.
  - *Pornind de la noţiunile teoretice abordate, autorul a proiectat un algoritm original de mişcare al roboţilor liberi.* Fiecare robot are de îndeplinit sarcini exacte. Primul robot deserveşte pacientul pe care în conduce pe traseul de investigare: de la intrarea în laboratorul de imagistică către pupitrul de înregistrare, echipamentul de radiografiere ortopanoramică, computerul tomograf sau camera ATI, şi în final către echipamentele de achiziţie de imagine şi printere. Cel de-al doilea deserveşte personalul medical şi auxiliar deplasându-se între diferitele echipamente de achiziţie de imagine, printere şi pupitrul de înregistrare pacienţi. Rezolvarea acestor sarcini

implică stabilirea unei traiectorii posibile de mişcare pentru fiecare robot dar şi realizarea unei verificări permanente a poziţiei robotului în raport cu celelalte obiecte ale spaţiului virtual pentru întâmpinarea unor coliziuni.

- *Procesul de evitare inter-roboţi şi între aceştia şi obstacolele fixe* este realizat în mod original prin definirea unor traiectorii de ocolire şi de stabilire a unei rute de revenire la direcţia calculată. Ca şi în cazul mişcării libere procesul deplasării orientate către o ţintă este definit printr-un graf al stărilor robotului liber.

Contribuţiile originale ale autorului obţinute pe parcursul realizării tezei se referă deci, la toate aspectele întâlnite pe parcursul rezolvării temei, parcurgând etapele fireşti de *studiu bibliografic - alegerea metodelor de modelare – stabilirea algoritmilor – modelarea corpurilor – stabilirea sarcinilor roboţilor în spaţiul dedicat mişcării lor.*

Aşa cum a fost concepută, teza de doctorat permite dezvoltarea viitoare a tematicii abordate, la spaţii de orice complexitate, pentru un număr mai mare de roboţi mobili, etc. Algoritmii creaţi şi implementaţi de autor pot fi generalizaţi pentru alte tipuri de aplicaţii cu grad de complexitate sporit, oferind soluţii pentru rezolvarea multor probleme practice de genul: controlul şi comanda autovehiculelor în spaţii foarte aglomerate, autoconducerea mişcărilor unor mobile în spaţii cu topografie cunoscută în prealabil, dar şi în spaţii cu configuraţii variabile. O altă perspectivă, oferită de rezultatele obţinute prin cercetarea în cadrul tezei de doctorat, ar fi modelarea spatiilor şi sistemelor complexe prin corpuri geometrice pline, nu numai prin suprafeţe şi curbe. Perspectiva este generoasă şi din punct de vedere teoretic, dar şi al algoritmilor de calcul şi al implementării practice.

Bibliografie

1. *** - "3D-Calc Software for IBM-PC and Compatibles". Markt & Technik
2. *** - „Mică enciclopedie matematică" Editura tehnică Bucureşti 1980
3. *** - „Microsoft Developer Network Library Help"
4. *** Grant CNCSIS cod 820, Tema 41 (2001 - 2003) – Modelarea sistemului complex robot-spaţiu de lucru pentru crearea unui laborator virtual (Stoicanescu C. membru în colectivul de cercetare)
5. *** Grant CNCSIS cod 480, Tema 15 (1998 - 2000) - Planificarea şi generarea mişcării roboţilor mobili într-un spaţiu cu obstacole (Stoicanescu C. membru în colectivul de cercetare)
6. Adams L.- "High Performance CAD Graphics". Windcrest
7. Andreiciuc D., cond. şt. T.Mureşan - "Teză de doctorat – Optimizarea conduceri vehiculelor ghidate automat". 1999
8. Awad M., Kuusela J. - "Object-Oriented Technology for Real Time Systems". Prentice Hall, 1996
9. Barkakati Naba -„Object-Oriented Programming in C++". SAMS 1991
10. Bates J., Tompkins T. –"Utilizare Visual C++" Toera 1999
11. Bezier P. – "Definition numerique des courbes et surfaces I". Automatisme, XI 625-632, 1966
12. Bezier P. – "Definition numerique des courbes et surfaces II". Automatisme, XII 17-21, 1967
13. Bezier P. – "Essay de Definition Numerique des Courbes et des Surfaces Experimentales". PhD thesis, University of Paris VI, 1977.
14. Bezier P. – „Numerical Control: Mathematics and Applications". Wiley, 1972
15. Bezier P. – „The first years of CAD/CAM and UNISURF CAD System". Fundamental developments of Computer-Aided Geometric Modeling, Academic Press 1993

16. Bezier P. – „The Mathematical Basis of the UNISURF CAD System". Butterworths, London, 1986

17. Boehm W. – "Cubic B-spline curves and surfaces in computer aided geometric design". Computing, 19(1), 1977
18. Boehm W. – "Generating the Bezier points of B-spline curves and surfaces". CAGD, 13(6), 1981
19. Boehm W. - "Inserting new knots into B-splines curves". CAGD 12 (1980)
20. Boehm W. – „Curvature continuous curves and surfaces". Computer Aided Geometric Design, 1985
21. Boehm W. – „On the definition of geometric continuity". Computer Aided Design, 1988
22. Boehm W., Farin G. – „A survey of curve and surface methods in CAGD". Computer Aided Geometric Design, 1984
23. Bouilly B., T.Simeon, R.Alami - "A numerical technique for planning motion strategies of a mobile robot in presence of uncertainty". IEEE International Conference on robotics and Automation, 1995
24. Canudas C., H.Khennouf, C.Samson, and O.J. Sordalen - "Nonlinear ControlDesign for Mobile Robots". World Scientific Series in Robotics and Automated Systems, Vol. 11, 1993
25. Chaikin G.– "An algorithm for high speed curve generation". Computer Graphics and Image Processing 3
26. Coons S.– Surfaces for Computer Aided Design, Technical Report MIT, 1974
27. Cooplien J.O. – „Advanced C++ Programming Syles and Idioms". Addison-Wesley Publishing Company 1992
28. Cox. M. – „The numerical evaluation of B-splines". J. Inst. Maths. Applics. 1972
29. Cunningham St. – "Computer Graphics Using Obiect-Oriented Programming". Wiley&Sons
30. de Boor C. - "A practical guide to splines". Springer-Verlag, 1978
31. de Boor C. – "Bicubic spline interpolation".
32. de Boor C.- "On calculating with B-splines". J. Approx. Theory, 61 (1972)
33. de Boor C.– „B-form basics". G Farin, editor, Geometric Modeling: Algorithms and New Trends SIAM, 1987
34. de Casteljau P. – „Shape Mathematics and CAD". Kogan Page, London, 1986

35. de Casteljau P. - " Polar Forms for curve and surface modeling as used at Citroen". Fundamental developments of Cumputer-Aided Geometric Modeling, ed. Les Piegl, Academic Press
36. de Casteljau P. – „Outillages methodes calcul". Technical Report, A. Citroen, Paris 1963
37. Drăgulescu D.- "Dinamica Roboților". Ed. Didactică şi Pedagogică, Bucureşti, 1997
38. Drăgulescu D., Couturier C.- "Cours de modelisation des robots". Lito. Universite d'Artois, Franța, 1995
39. Drăgulescu D., Moldovan F, Toth-Taşcău M., Crivacucea C.- "Metode de planificare a traiectoriilor la roboți. Sinteză".Analele universității Eftimie Murgu, Reşița, anul III, 1996
40. Drăgulescu D., Moldovan F., Moldovan H.- "Considerations about the critical curves at the exact cell decomposition method". International Conference on Technical Informatics, Conti'94, Timişoara, 1994
41. Drăgulescu D., Moldovan F., Moldovan H.- "Metoda de determinare a marginilor unui C-obstacol". Al XII-lea Simpozion Național de Roboți Ind., Timişoara, 1994
42. Drăgulescu D., Toth-Taşcău M. - "Contributions to cell decomposition method for a two-dimensional work space". International Conference on Technical Informatics, Conti'96, Timişoara, 1996
43. Drăgulescu D., Toth-Taşcău M. - "Determinarea preciziei de parcurgere a taiectoriei unui robot". A XXVII-a Ses. De Comunicări Ştiințifice cu Part. Intern., Ac. Tehnică Militară, Bucureşti, 1997
44. Drăgulescu D., Toth-Taşcău M. - "Method to generate an imposed trajectory for a robot having 6 degrees of fredom". International Conference on Technical Informatics, Conti'94, Timişoara, 1994
45. Drăgulescu D., Toth-Taşcău M. - "Path planning for a robot by exact cell decomposition method". II-nd International Conference on Advanced Robotics, Viena, 1996
46. Drăgulescu D., Toth-Taşcău M. - "Studiul dinamicii unui robot tip KUKA". The VII-th International Conference of Manufacturing Engineering Techno'95, Timişoara, 1995

47. Drăgulescu D., Toth-Taşcău M. - "Workspace modelling by non-homogeneous grid and tree method". Buletinul Şt. Şi Tehnic al UTT, seria Mecanică, Tom 42(56), 1997
48. Drăgulescu D., Toth-Taşcău M.- "Interpolarea traiectoriilor unui robot cu funcţii spline şi restricţii cinematice şi dinamice". The VII-th Symposium of Mathematics and its Applic., univ. Politehnica Timişoara, 1997
49. Drăgulescu D., Toth-Taşcău M., Moldovan F. - „Planificarea mişcării roboţilor industriali". Universitatea Tehnică Timişoara
50. Drăgulescu D., Toth-Taşcău M., Moldovan F. - "Metoda de planificare a traiectorilor în spaţiul configuraţiilor bidimensional. Metoda grafului vizibilităţii". Analele Universităţii Oradea, 1995
51. Drăgulescu D., Toth-Taşcău M., Moldovan F. - "Metodă şi algortim de planificare a triectorilor plane". A II sesiune de comunicări ştiinţifice cu participare internaţională "Realizări tehnice şi cultural-ştiinţifice pe meleaguri arădene", Arad, 1994
52. Drăgulescu D., Toth-Taşcău M., Moldovan F. - "Planificarea mişcării roboţilor", Ed. Helicon, Timişoara, 1995
53. Drăgulescu D., Toth-Taşcău M., Pasco G.- "Path planning by the exactcell decomposition method". Buletinul Şt. Şi Tehnic al UTT, seria Mecanică, Tom 41(55), 1996
54. Farin G. - "Algorithms for rational Bezier curves". CAGD 15 (1983)
55. Farin G. – "Curves and Surfaces for CAGD, 5th ed.", Morgan Kaufmann 2001
56. Farin G.- "Curves and surfaces for CAGD" Academic Press, Inc 1993.
57. Farin G.– "NURBS Curves and Surfaces". AK Peters, Boston, 1995
58. Farin G., Hansford D. – "The essentials of CAGD" AK Peters, Ltd
59. Faux I. D., Prat M. J. – "Computational Geometry for Design and Manufacture". John Wiley & Sons, 1987
60. Ferguson J. – "Multivariable curve interpolation" J. ACM, 1964
61. Gallier J. – „Curves and Surfaces in Geometric Modeling: Theory and Algorithms". Morgan Kaufmann 2000
62. Gordon W. – "Free-form Surface Interpolation through Curve Networks". Technical Report, General Motors Research Laboratories, 1969
63. Gordon W., Riesenfeld R. – „B-spline curves and surfaces". Computer Aided Design, Academic Press, 1974

64. Greville T. – "Introduction to spline functions". Theory and Applications of Spline Functions, Academic Press, 1969
65. Hagen. H. – "Geometric spline curves". CAGD 2 (1-3) 1985
66. Hoschek J., Lasser D.– "Fundamentals of Computer Aided Geometric Design". A. K. Peters, 1993
67. Jamsa K., Klander L. – "Manualul fundamental de programare în C şi C++" Teora 1997
68. Kempf Renate, Frazier C. – „OpengGL Reference Manual" Second Edition. Addison-Wesley 1999
69. Kruglinski C. – "Inside Visual C++" Microsoft Press 1996
70. Lambert A., N.Le Fort-Piat - "Robot tasks planning integrating uncertainties and local maps". AVCS'98, Amiens, France, 1998
71. Lorentz G.– "Bernstein Polynomials". Toronto Press 1986
72. Lungu O, Stoicanescu C, Pocol C – ICAD Approach for EDM Technology Conti 94
73. Mason Woo, Neider J., Davis T., Shreiner D. – „OpenGL Programming Guide" Third Edition. Addison-Wesley 1999
74. Michelli C. A. - "Mathematical aspects of Geometric Modeling". SIAM 1995
75. Micula G., Micula S.– "Handbook of splines". Kluwer Academic Publ., 1998
76. Moebius E. – "August Ferdinand Moebius, Gesammlte Werke" Wiesbaden 1967
77. Mortenson, M.E. –"Computer Graphics Handbook: Geometry and Mathematics". Industrial Press, 1990
78. Mureşan I, Mureşan V., Fara H., Stoicanescu C. Raţiu D.- Abordare orientată pe obiect pentru implementarea unui sistem integrat de proiectare a construcţiilor din lemn, Sesiune de comunicări ştiinţifice "Zilele academice timişene" 1999
79. Mureşan I, Pocol C, Linte M, Stoicanescu C – Roof structure CAD Conti 94
80. Mureşan I. – "Solid modeler for CAD". International Conference on Technical Informatics-Conti'94, Tehnical University of Timişoara, 1994
81. Mureşan I., Savii G – „Abordări moderne în integrarea sistemelor CAD/CAM" Presa Universitară Română, 1996
82. Mureşan I., Savii G. – "Fundamentele CAD/CAM. Principii şi metode" Presa Universitară Română, 1996
83. Mureşan I., Stoicanescu C. – "Wall Construction in Roof Structure Applications". Sesiunea de comunicări ştiinţifice ACADRES'96, 16 nov 1996, Univ. Banatului

84. Mureşan V – "Modelarea de solide pentru proiectarea asistată de calculator" 1996
85. Muşlea I. – „Programarea orientată pe obiecte" Biblioteca PC 1992
86. Nielsen K. W. - "Software development with C++, Maximizing Reuse with Object Technology". Academic Press 1995
87. Pearson, C.E. – "Handbook of applied mathematics" Van Nostrand Reinhold Company, Litton Educational Publishing Inc., 1994
88. Petrişor E. - "Modelare geometrică algoritmică". Editura Tehnică
89. Petrişor E. - "The polar form of a polynomial surface". Applied Sciences, 2 (2000),
90. Popovici M. D., Popovici M.I. – "C++ Tehnologia orientată spre obiecte Aplicații" Teora 2000
91. Ramshaw L. – "Beziers and B-splines as multiaffine maps". Theoretical Foundations of Computer Graphics and CAD, Springer Verlag, 1988
92. Ramshaw L. - "Blossoming: a connect-the-dots approach to splines". Technical Report, Digital Systems Research Center 19, 1987
93. Ramshaw L. - "Blossoms are polar forms". CAGD, 6 (1989)
94. Riesenfeld R. – "Applications of B-spline Approximation to Geometric Problems of Computer-aided Design". PhD thesis, Dept. of Computer Science, Syracuse U., 1973
95. Risler J.J. – "Mathematical Methods for CAD". Cambridge University Press, 1992
96. Sabin M. – "Recursive subdivision". The mathematics of Surfaces pages 269-281 J. Gregory.
97. Schumaker L. – "Spline Functions: Basic Theory". Wiley 1981
98. Seidel H – "Computing B-spline control points using polar forms". Computer Aided Design, 23(9), 1991
99. Seidel H. P. – "A new multiaffine approach to B-splines". Computer-Aided Geometric Design, 6
100. Steve M.- "Writing Solid Code". Microsoft Press
101. Stevens Rogers T. - "Object – Oriented Graphics Programming in C++". AP Professional 1994
102. Stoicanescu C. – "Monitorizarea erorilor în timpul rulării unui program C". Sesiunea de comunicări ştiinţifice ACADRES'95, 5 mai 1995 Univ. Banatului
103. Yamaguchi F.– "Curves and Surfaces in Computer Aided Geometric Deisgn". Springer 1988