615. 777
211 17

UNIVERSITATEA "POLITEHNICA" DIN TIMISOARA
Facultatea de Automaticã si Calculatoare

ing. Voichita Muresan

MODELATOR DE SOLIDE PENTRU PROIECTAREA ASISTATĂ DE CALCULATOR

- TEZA DE DOCTORAT -

BIBLIOTECA CENTRALĂ
UNIVERSITATEA "POLITEHNICA"
TIMIŞOARA

Conducãtor stiintific:
Prof.Dr.Ing. George G. Savii

Timisoara
1996

BUPT

Cuprinsul

1. INTRODUCERE 1

1.1. MODELAREA SI VIZUALIZAREA 1
1.2. VIZUALIZAREA 1

2. MODELAREA CORPURILOR 3D 5

2.1. INTRODUCERE 5
2.2. FUNDAMENTELE MODELĂRII SOLIDELOR 7
2.2.1. GEOMETRIA SI TOPOLOGIA SOLIDELOR 7
2.2.2. INCHIDEREA GEOMETRICĂ 8
2.2.3. TEORIA MULTIMILOR 9
2.2.4. CLASIFICAREA ELEMENTELOR MULTIMII 13
2.3. GENERAREA SOLIDELOR PRIN OPERATII BOOLEENE 13
2.4. TEHNICI DE REPREZENTARE A MODELELOR GEOMETRICE 3D 14
2.4.1. REPREZENTĂRI PRIN PUNCTE 14
2.4.2. REPREZENTĂRI PRIN SEGMENTE 15
2.4.3. REPREZENTĂRI PRIN POLIGOANE 16
2.4.4. REPREZENTĂRI PRIN VOLUME 19

3. VIZUALIZAREA MODELELOR 3D 20

3.1. TEHNICI DE VIZUALIZARE A MODELELOR 3D 20
3.2. REALISMUL VIZUAL 21
3.3. ÎNLĂTURAREA LINIILOR ASCUNSE 23
3.3.1. VIZIBILITATEA REPREZENTĂRILOR OBIECTELOR 23
3.3.2. TEHNICI DE VIZIBILITATE 24
3.3.3. SORTAREA 25
3.3.4. COERENTA 26
3.3.5. EXEMPLE DE ALGORITMI PENTRU LINII ASCUNSE 27
3.3.5.1. Algoritmul cu priorităti 27
3.3.5.2. Algoritmul orientat pe arii 29
3.4. ÎNLĂTURAREA SUPRAFETELOR ASCUNSE 32
3.4.1. ALGORITMUL BUFFER-ULUI Z 32
3.4.2. ALGORITMUL LUI WARNOCK 33
3.5. ELIMINAREA SOLIDELOR ASCUNSE 34
3.5.1. ALGORITMUL CU URMĂRIREA RAZEI 35
3.6. UMBRIREA 36

4. ALGORITMI UTILIZATI ÎN MODELAREA SOLIDELOR PRIN OPERATII BOOLEENE 37

4.1. ALGORITM PENTRU CLASIFICAREA ELEMENTELOR UNEI MULTIMI ÎN CAZUL INTERSECTIEI EI CU UN SOLID 37

BUPT

4.2. ALGORITMI DE VIZUALIZARE 39
4.2.1. TESTUL MINIMAX 39
4.2.2. TESTUL DE INTERIOR 40
4.2.3. TESTUL DE ADÂNCIME 41
4.2.4. CALCULUL SILUETELOR 43
4.2.5. INTERSECTIILE MUCHIILOR 45
4.2.6. COMPARAREA SEGMENTELOR 45
4.2.7. TESTUL DE OMOGENITATE 47

5. DIFICULTATI GENERALE IN CADRUL ALGORITMILOR DE MODELARE SI REPREZENTARE A SOLIDELOR 50

5.1.INTRODUCERE 50
5.2.REDUCEREA NUMARULUI DE CALCULE 51
5.3.CAZURI DEGENERATE 54
5.4.TRATAREA ERORILOR NUMERICE 63

6. ALGORITMI UTILIZATI ÎN MODELAREA PRIN OPERATII BOOLEENE SI ÎN PROCESUL DE ELIMINARE A LINIILOR SI SUPRAFETELOR ASCUNSE 67

6.1. ALGORITMI PENTRU TESTELE DE BOUNDING-BOX 67
6.2.ALGORITM PENTRU AFLAREA UNUI PUNCT SITUAT PE UN SEGMENT DAT CÂND SE CUNOASTE RAPORTUL ÎN CARE PUNCTUL ÎMPARTE SEGMENTUL 69
6.3.ALGORITMI PENTRU CALCULUL DISTANTELOR 71
6.3.1.ALGORITM PENTRU CALCULUL DISTANTEI DE LA UN PUNCT LA O DREAPTĂ 71
6.3.2 ALGORITM PENTRU CALCULUL DISTANTEI DE LA UN PUNCT LA UN PLAN 72
6.4.ALGORITMI PENTRU DETERMINAREA POZITIEI RELATIVE A DOUĂ FIGURI GEOMETRICE 73
6.4.1.ALGORITM PENTRU DETERMINAREA POZITIEI RELATIVE A UNUI PUNCT FATĂ DE O DREAPTĂ 73
6.4.2 ALGORITM PENTRU DETERMINAREA POZITIEI RELATIVE A UNUI PUNCT FATĂ DE UN SEGMENT 74
6.4.3.ALGORITM PENTRU DETERMINAREA POZITIEI RELATIVE A UNUI PUNCT FATĂ DE UN PLAN 74
6.4.4.ALGORITM PENTRU DETERMINAREA POZITIEI RELATIVE A UNUI PUNCT FATĂ DE UN POLIGON 75
6.4.5.ALGORITM PENTRU DETERMINAREA POZITIEI RELATIVE A UNUI SEGMENT FATĂ DE UN PLAN 76
6.4.6.ALGORITM PENTRU DETERMINAREA POZITIEI RELATIVE A DOUĂ SEGMENTE 76
6.4.7.ALGORITM PENTRU DETERMINAREA POZITIEI RELATIVE A DOUĂ SEGMENTE COLINEARE 78
6.5.ALGORITMI PENTRU DETERMINAREA INTERSECTIILOR DINTRE SEGMENTE SI PLANE 80
6.5.1.ALGORITM PENTRU DETERMINAREA PUNCTULUI DE INTERSECTIE DINTRE DOUĂ SEGMENTE 80
6.5.2.ALGORITM PENTRU DETERMINAREA PUNCTULUI DE INTERSECTIE DINTRE UN SEGMENT SI UN PLAN 81
6.5.3.ALGORITM PENTRU DETERMINAREA LINIEI DE INTERSECTIE DINTRE DOUĂ PLANE 82
6.6.ALGORITM PENTRU SELECTIA UNEI MUCHII ÎN PROCESUL DE STABILIRE A CONTURURILOR POLIGONALE 83
6.6.1.ALGORITM ORIGINAL PENTRU SELECTIA UNEI MUCHII ÎN CAZUL STABILIRII CONTURULUI UNUI POLIGON ATOMIC 83

BUPT

6.6.2. ALGORITM ORIGINAL PENTRU SELECTIA UNEI MUCHII ÎN CAZUL STABILIRII CONTURULUI UNUI POLIGON OBTINUT PRIN REUNIUNEA MAI MULTOR POLIGOANE ____ 85
6.7.ALGORITMI PENTRU TESTUL DE INTERIOR ____ 85
6.7.1. ALGORITM ORIGINAL PENTRU STABILIREA POZITIEI RELATIVE A UNUI PUNCT FATĂ DE UN POLIGON ____ 85
6.7.2. ALGORITM ORIGINAL PENTRU STABILIREA POZITIEI RELATIVE A UNUI PUNCT FATĂ DE UN VOLUM ____ 87
6.8.CALCULE DE ADINCIME ____ 87
6.9.ALGORITM PENTRU COMPARAREA A DOUĂ DREPTE REZULTATE PRIN INTERSECTII DE PLANE ____ 89
6.10.ALGORITM PENTRU DETERMINAREA FETELOR POTENTIAL VIZIBILE ____ 90
6.11.ALGORITM PENTRU DETERMINAREA CONTURURILOR UNUI SOLID ____ 91
6.12.ALGORITMUL ORIGINAL DE CLASIFICARE ____ 92

7. ALGORITM ORIGINAL PENTRU MODELAREA SOLIDELOR FOLOSIND OPERATIILE BOOLEENE ____ 94

7.1.PRINCIPIUL DE BAZA AL ALGORITMULUI ____ 94
7.2.STRUCTURILE DE DATE UTILIZATE IN PROGRAMUL PENTRU EFECTUAREA OPERATIILOR BOOLEENE INTRE SOLIDE ____ 95
7.3.IMPLEMENTAREA ALGORITMULUI DE EFECTUARE A OPERATIILOR BOOLEENE INTRE SOLIDE ____ 101
7.4.FUNCTIA PRINCIPALĂ CARE IMPLEMENTEAZĂ ALGORITMUL DE EFECTUARE A OPERATIILOR BOOLEENE ____ 103
7.5.FUNCTIILE CARE IMPLEMENTEAZA ALGORITMII DIN CAPITOLUL 6 ____ 105
7.5.1. FUNCTII CARE IMPLEMENTEAZĂ ALGORITMII PENTRU TESTELE DE BOUNDING-BOX ____ 105
7.5.2. IMPLEMENTAREA ALGORITMULUI PENTRU AFLAREA UNUI PUNCT SITUAT PE UN SEGMENT DAT CÂND SE CUNOASTE RAPORTUL ÎN CARE PUNCTUL ÎMPARTE SEGMENTUL ____ 105
7.5.3. IMPLEMENTAREA ALGORITMULUI PENTRU CALCULUL DISTANTEI DE LA UN PUNCT LA O DREAPTĂ ____ 105
7.5.4. FUNCTIA CARE IMPLEMENTEAZĂ ALGORITMUL PENTRU DETERMINAREA POZITIEI RELATIVE A UNUI PUNCT FATĂ DE UN POLIGON DAT ____ 106
7.5.5. FUNCTIA CARE IMPLEMENTEAZĂ ALGORITMUL PENTRU DETERMINAREA POZITIEI RELATIVE A DOUĂ SEGMENTE ____ 106
7.5.6. IMPLEMENTAREA ALGORITMULUI PENTRU DETERMINAREA LINIEI DE INTERSECTIE DINTRE DOUA PLANE ____ 107
7.5.7. IMPLEMENTAREA ALGORITMULUI PENTRU SELECTIA UNEI MUCHII IN PROCESUL DE STABILIRE A CONTURULUI UNUI POLIGON ATOMIC ____ 107
7.5.8. IMPLEMENTAREA ALGORITMULUI PENTRU SELECTIA UNEI MUCHII IN IN CAZUL STABILIRII CONTURULUI UNUI POLIGON OBTINUT PRIN REUNIUNEA MAI MULTOR POLIGOANE ____ 107
7.5.9. IMPLEMENTAREA ALGORITMULUI PENTRU DETERMINAREA POZITIEI RELATIVE A UNUI PUNCT FATA DE UN POLIGON ____ 108
7.5.10. IMPLEMENTAREA ALGORITMULUI DE CLASIFICARE ____ 108
7.6.FUNCTIILE CARE IMPLEMENTEAZĂ CREAREA STRUCTURILOR DE DATE NECESARE PENTRU EFECTUAREA OPERATIILOR BOOLEENE ____ 112
7.7.FUNCTIILE CARE IMPLEMENTEAZA OPERATIILE DE INTERSECTIE DINTRE CELE DOUA SOLIDE ____ 114
7.8.FUNCTII PENTRU TRATAREA CAZURILOR PARTICULARE

BUPT

MUCHIE/MUCHIE 118
7.9.FUNCTIILE CARE SELECTEAZA POLIGOANELE CE TREBUIE ELIMINATE 126
7.10.FUNCTIILE CARE IMPLEMENTEAZA OPERATIILE DE CONSTRUIRE A POLIGOANELOR REZULTATE 128
7.11.FUNCTIILE CARE IMPLEMENTEAZA OPERATIILE PENTRU CONSTRUIREA SOLIDELOR REZULTATE 129

8. ALGORITM ORIGINAL PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE 136

8.1. INTRODUCERE 136
8.2. ETAPELE PARCURSE ÎN DEZVOLTAREA ALGORITMULUI 137
8.3. PRINCIPIUL DE BAZĂ AL ALGORITMULUI PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE PENTRU MODELE DE SOLIDE 141
8.4. VARIANTE ALE ALGORITMULUI PREZENTAT 144
8.4.1. ALGORITMUL LINIILOR ASCUNSE PENTRU CORPURI SUPRAPUSE ÎN SPATIU 144
8.4.2. ALGORITMUL LINIILOR ASCUNSE CU COLORAREA FETELOR VIZIBILE 145
8.5. PRINCIPALELE PROBLEME ALE ALGORITMILOR LINIILOR ASCUNSE 145
8.6. STRUCTURILE DE DATE UTILIZATE ÎN PROGRAMELE DE LINII ASCUNSE 147
8.7. FUNCTIILE CARE IMPLEMENTEAZĂ ALGORITMII PREZENTATI ÎN CAPITOLUL 6 149
8.7.1. FUNCTII CARE IMPLEMENTEAZĂ ALGORITMII PENTRU TESTELE DE BOUNDING-BOX 149
8.7.2. IMPLEMENTAREA ALGORITMULUI PENTRU AFLAREA UNUI PUNCT SITUAT PE UN SEGMENT DAT CÂND SE CUNOASTE RAPORTUL ÎN CARE PUNCTUL PARTE SEGMENTUL 150
8.7.3 FUNCTII CARE IMPLEMENTEAZĂ ALGORITMII PENTRU CALCULUL DISTANTELOR 151
8.7.3.1. Functia care implementează algoritmul pentru calculul distantei dintre două puncte 151
8.7.3.2. Implementarea algoritmului pentru calculul distantei de la un punct la o dreaptă 151
8.7.4. FUNCTII PENTRU IMPLEMENTAREA ALGORITMILOR PENTRU DETERMINAREA POZITIEI RELATIVE A DOUĂ FIGURI GEOMETRICE 152
8.7.4.1. Functia care implementează algoritmul pentru determinarea pozitiei relative a unui punct fată de un plan 152
8.7.4.2. Functia care implementează algoritmul pentru determinarea pozitiei relative a două segmente 152
8.7.4.3. Functia care implementează algoritmul pentru determinarea pozitiei relative a două segmente coliniare 154
8.7.5. IMPLEMENTAREA ALGORITMULUI PENTRU DETERMINAREA PUNCTULUI DE INTERSECTIE DINTRE DOUĂ SEGMENTE 155
8.7.6. IMPLEMENTAREA ALGORITMULUI PENTRU SELECTIA UNEI MUCHII ÎN CAZUL STABILIRII CONTURULUI UNUI POLIGON ATOMIC 155
8.7.7. IMPLEMENTAREA ALGORITMULUI PENTRU TESTUL DE INTERIOR 155
8.7.8. IMPLEMENTAREA FUNCTIILOR PENTRU CALCULELE DE ADÂNCIME 156
8.7.9. IMPLEMENTAREA ALGORITMULUI PENTRU DETERMINAREA FETELOR POTENTIAL VIZIBILE 157
8.7.10. IMPLEMENTAREA ALGORITMULUI PENTRU DETERMINAREA CONTURURILOR UNUI SOLID 157
8.8. FUNCTIILE NECESARE IMPLEMENTĂRII ALGORITMULUI PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE 158

BUPT

8.8.1. PRINCIPALELE FUNCTII CARE STAU LA BAZA IMPLEMENTĂRII 158
8.8.2. FUNCTII CARE IMPLEMENTEAZĂ CREAREA BAZEI DE DATE NECESARE ALGORITMULUI DE ELIMINARE A LINIILOR SI SUPRAFETELOR ASCUNSE 161
8.8.3. FUNCTII CARE IMPLEMENTEAZĂ OPERATIILE DE INTERSECTIE 163
8.8.4. FUNCTII CARE IMPLEMENTEAZĂ TESTELE DE VIZIBILITATE 169
8.8.5. FUNCTII CARE IMPLEMENTEAZĂ STABILIREA POLIGOANELOR ATOMICE SI TIPUL LOR 171

9. CONCLUZII 175

9.1.CONCLUZII ASUPRA PROGRAMULUI DE IMPLEMENTARE A ALGORITMULUI PENTRU EFECTUAREA OPERATIILOR BOOLEENE INTRE SOLIDE 175
9.2. CONCLUZII ASUPRA PROGRAMULUI DE IMPLEMENTARE A ALGORITMULUI PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE 177
9.3.REZULTATE SI MASURATORI 178
9.4.PERSPECTIVE 182
9.5.CONTRIBUTII ORIGINALE 183

BIBLIOGRAFIA 186

BUPT

BUPT

CAPITOLUL 1

INTRODUCERE

1.1. MODELAREA SI VIZUALIZAREA

Nucleul sistemelor CAD îl reprezintă modelarea geometrică si aplicatiile grafice. Modelarea geometrică permite abstractizarea sau reprezentarea obiectelor, înlocuind prototipurile utilizate în mod curent în practică, în timp ce simularea înlocuieste testele efectuate asupra prototipului.

Un model geometric este definit ca o reprezentare completă a unui obiect care contine atât informatie grafică cât si negrafică. Constructia modelului implică crearea bazei de date a modelului, în timp ce vizualizarea afectează modul în care modelul este afisat pe ecran. Modelarea si vizualizarea sunt legate între ele, dar tehnicile de afisare si aproximările cu care se face aceasta nu afectează reprezentarea obiectului prin baza sa de date. Vederile sunt definite prin diferitele unghiuri sub care este observat modelul. Cel care priveste îsi modifică pozitia în sistemul de coordonate al modelului, pe când modelul îsi mentine orientarea initială. Pentru observator există senzatia că modelul este cel care se miscă. Unui model i se poate defini o infinitate de vederi. Cele mai multe sisteme CAD furnizează comenzi pentru obtinerea vederilor standard (de sus, de jos, din stânga s.a.m.d) atât ca vederi bidimensionale cât si ca vederi tridimensionale [Zei91][Hof87].

1.2. VIZUALIZAREA

Una din cele mai utile facilităti ale sistemelor CAD/CAM, recunoscută de marea majoritate a utilizatorilor, o reprezintă posibilitatea furnizării imaginilor vizuale ale obiectelor si scenelor pe care acestea le modelează. Vizualizarea a fost întotdeauna recunoscută ca fiind mijlocul cel mai eficace pentru comunicarea

BUPT

de idei si proiecte noi între proiectanti, ingineri si altii. Se spune că întotdeauna o imagine are valoarea a o mie de cuvinte. Adevărul acestei afirmatii poate fi legat de faptul că un procent de 50% din neuronii creierului sunt asociati cu vederea. Virtual, toate programele de aplicatie legate de CAD/CAM îsi prezintă rezultatele utilizatorilor într-o formă grafică, convertind rezultatele numerice corespunzătoare la aceste forme [Coh93].

Vizualizarea cuprinde atât întelegerea imaginii cât si sinteza imaginii; astfel, vizualizarea este un mijloc atât pentru interpretarea datelor introduse într-un calculator, cât si pentru generarea imaginilor din seturi de date complexe multidimensionale. De aceea pot fi identificate două tipuri de vizualizare: vizualizarea în modelarea geometrică si vizualizarea în calculul stiintific. Prima este legată de reprezentarea modelelor geometrice ale obiectelor, în timp ce a doua ia în considerare reprezentarea rezultatelor stiintifice, ingineresti [Ros89].

Cele mai obisnuite metode de vizualizare a modelelor geometrice ale obiectelor sunt proiectia si umbrirea. Proiectiile ortografice au fost utilizate mult timp în desenul tehnic pentru a reprezenta proiectele noi. Proiectiile izometrice si perspective sunt folosite si în CAD/CAM datorită usurintei cu care pot fi generate chiar si automat si datorită bogătiei de informatii vizuale pe care le furnizează, informatii legate de proiecte si obiecte. Cu toate că proiectiile ortografice sunt cele mai vechi mijloace de comunicare legate de proiectele ingineresti, sunt greu de interpretat si cer o anumită experientă. De asemenea, ele nu prezintă nici o caracteristică legată de aspectul proiectelor, cum ar fi culoarea suprafetelor si textura. Imaginile umbrite cu un înalt grad de realism pot face utilizatorii să creadă că imaginile reprezintă obiecte reale si nu unele sintetice (modele matematice). Imaginile umbrite de o înaltă calitate, în multe exemple, furnizează un mod simplu, mai eficient si mai putin scump de a vedea diferite alternative ale proiectului, decât constructia modelelor si prototipurilor. Proiectele mai multor părti mecanice, caroseriile automobilelor, părti ale avioanelor, carenele navelor si altele sunt mult îmbunătătite prin studiul imaginilor umbrite corespunzătoare.

Vizualizarea în calculul stiintific este văzută ca o metodă din cadrul modelării geometrice. Ea transformă datele numerice în reprezentări sub formă de imagini, permitând utilizatorilor să observe simulările si rezultatele calculelor efectuate.

Multe aplicatii CAD/CAM existente utilizează una din formele prezentate de vizualizare. Astfel sunt simulatoarele de zbor si navigare, simularea miscării diferitelor mecanisme efectuată prin generarea si animarea diferitelor cadre ale modelelor geometrice corespunzătoare, generate în concordantă cu ecuatiile cinematice si dinamice care guvernează mecanismele. Simularea robotilor si

BUPT

planificarea si generarea traiectoriei în sistemele CAD/CAM pot fi obtinute într-un mod asemănător. Verificarea traseelor sculelor masinilor unelte cu comandă numerică este o altă aplicatie de vizualizare utilă, în care descrierea prin date numerice a traseului sculei de tăiere este folosită la generarea si reprezentarea miscării sculei pentru a-i verifica corectitudinea. Alte aplicatii ale vizualizării în calculul stiintific includ reprezentarea rezultatelor analizei prin elemente finite, analizei transferului de caldură, calculului dinamicii fluidelor si dinamicii structurale si vibratiilor.

Problema principală în vizualizare o constituie reprezentarea obiectelor si scenelor tridimensionale pe ecrane bidimensionale, adică modul în care poate fi sugerată a treia dimensiune pe ecran, adâncimea, si modul în care poate fi reprezentată, prin atribute ale imaginii, complexitatea realitătii înconjurătoare descrisă prin iluminare, culoare, umbre si textură. Ceea ce complică reprezentarea obiectelor tridimensionale si mai mult este natura centralizată a bazei de date a modelelor lor geometrice. Dacă proiectăm un model tridimensional complex pe ecran, obtinem un labirint de linii si suprafete. Pentru a interpreta acest labirint, liniile si suprafetele care nu pot fi văzute dintr-un anumit punct de observare, trebuie eliminate. Inlăturarea lor elimină ambiguitătile reprezentărilor simple ale modelelor tridimensionale si este considerată ca fiind primul pas spre obtinerea realismului vizual.

Există diferite abordări pentru obtinerea realismului vizual. Ele sunt legate direct de tipul de modele geometrice utilizate pentru reprezentările obiectelor tridimensionale. Astfel, ar fi de asteptat o tendintă pozitivă în eficienta si nivelul de automatizare al acestor abordări pe masură ce tehnicile de modelare geometrică au avansat de la reprezentări cadru, la suprafete si solide. Printre abordările existente, se află proiectiile paralele, proiectiile perspective, înlăturarea liniilor ascunse, înlăturarea suprafetelor ascunse, înlăturarea solidelor ascunse si generarea imaginilor umbrite ale modelelor si scenelor. Proiectiile paralele, în particular proiectiile ortografice, reprezintă cea mai veche abordare. Ele sunt usor de generat, folosesc cel mai putin timp de calculator pentru generare si sunt mult folosite în desenul tehnic. Proiectiile perspective sunt mult folosite în ingineria arhitecturii si constructiilor. Imaginile umbrite pot fi generate numai pentru suprafete si modele de solide. Umbrirea este un proces în doi pasi. El începe prin eliminarea suprafetelor si solidelor ascunse si apoi se efectuează umbrirea partilor vizibile. Imaginile umbrite reprezintă cel mai înalt nivel al realismului vizual.

Mărirea gradului realismului unui model prin dezvoltarea imaginilor umbrite de exemplu, duce la cresterea complexitătii lui. Procesul de creare si

BUPT

întretinere a unui astfel de model devine, de asemenea, complex. De exemplu, generarea unei imagini umbrite de înaltă rezolutie a unui model complex cu diferite tipuri de iluminări, poate dura un timp CPU de câteva minute si poate reduce performantele sistemului CAD/CAM (încetineste activitatea altor utilizatori curenti). Memorarea si/sau regăsirea modelelor cu aceste tipuri de imagini umbrite sunt de obicei lente. De altfel, generarea imaginilor în timp real, cere calculatoare puternice cu cât mai multi algoritmi de umbrire încorporati în hardware. Pentru a îmbunătăti performantele acestor algoritmi, anumite sisteme CAD/CAM sunt furnizate cu microprograme de umbrire.

Abordarea directă a înlăturării liniilor/suprafetelor/solidelor ascunse cere o cantitate mare de timp de calcul. Pozitia fiecărei linii sau suprafete trebuie comparată cu toate celelalte, lucru care conduce la o explozie combinatorială ce are la bază o lege de variatie proportională cu pătratul numărului de elemente de comparat. Din acest motiv s-au efectuat studii extensive asupra algoritmilor existenti, încercându-se stabilirea principiilor generale comune acestor algoritmi. Cele mai importante au fost identificate a fi coerenta si sortarea. Obiectele si modelele lor geometrice sunt mai mult decât un set de discontinuităti aleatoare. Ele au consistentă atât în aria imaginii cât si în timp, de la un cadru la următorul. Astfel, folosind coerenta între diferite cadre se poate îmbunătăti eficienta algoritmilor de vizualizare.

Similar, toti algoritmii de ascundere sortează sau caută prin colectiile de suprafete, muchii sau obiecte în functie de diferite criterii, pentru a găsi elementele vizibile pe care să le reprezinte. De aceea, o tehnică de sortare eficientă reprezintă cheia unui bun algoritm de eliminare a liniilor/suprafetelor/solidelor ascunse [Zei91][Tan89].

BUPT

CAPITOLUL 2

MODELAREA CORPURILOR 3D

2.1. INTRODUCERE

Reprezentarea cu succes a modelelor solidelor în calculatoare si utilizarea lor in aplicatiile ingineresti depinde atât de proprietătile lor cât si de cele ale schemelor prin care sunt reprezentate. În contextul teoriei modelării solidelor, modelul solidului, model abstract, este considerat modelul matematic al obiectului real (solidul fizic). Proprietătile acestui model matematic determină comportarea lui în momentul în care algoritmii geometrici prelucrează structura de date asociată. Un model matematic al unui obiect stabileste clasa de algoritmi care pot fi aplicati asupra lui si nivelul automatizării lor.

Proprietătile pe care trebuie să le aibe un model al solidului, din punct de vedere matematic, ar putea fi enuntate în felul următor:

1. Rigiditatea. Aceasta implică faptul că forma modelului solidului este invariantă si nu depinde de pozitia sau orientarea în spatiu a modelului.

2. Omogenitatea tridimensională. Frontierele solidului trebuie să fie în contact cu interiorul (să nu existe frontiere izolate ca în fig.2.1.).

3. Dimensiunea solidului să nu fie infinită si solidul să poată fi descris printr-o cantitate finită de informatie.

4. Prin deplasare sau operatii booleene regulate să se poată obtine alte solide valide.

5. Determinismul frontierelor. Frontiera unui solid trebuie să contină solidul si mai mult, trebuie să determine în mod clar interiorul solidului.

Implicatia matematică a acestor proprietăti sugerează faptul că modelele solidelor sunt submultimi ale lui E^3 mărginite, închise, regulate si semianalitice. Aceste submultimi se numesc multimi regulate "r-sets" (regularized sets). Multimea de puncte S care defineste un model solid si care este dată prin relatia (2.1) este totdeauna o multime regulată (r-set).

BUPT

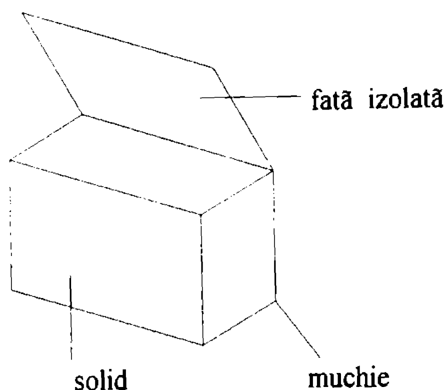


Figura 2.1.Exemplu de frontiere izolate

După cum s-au stabilit proprietătile necesare ale unui model de solid, se pot discuta si cele ale schemelor de reprezentare care operează de obicei asupra multimilor de puncte sau r-sets pentru a produce modele valide.O schemă de reprezentare este definită ca o relatie care leagă o multime de puncte valide de un model valid. De exemplu, o schemă CSG (vezi 2.4.4) leagă primitivele valide de solidele valide prin intermediul operatiilor booleene. O schemă de reprezentare este neambiguă sau completă, dar nu unică, dacă obiectul poate fi reprezentat prin mai mult de un model. Pe de altă parte, o schemă este ambiguă sau incompletă, dacă un model poate reprezenta mai mult de un obiect, ca în cazul modelelor cadru [Zei91][Hof87].

Proprietătile schemelor de reprezentare care determină utilitatea lor în modelarea solidelor ar fi următoarele:

1. **Domeniul.** Domeniul unei scheme de reprezentare este clasa de obiecte pe care schema le poate reprezenta sau este acoperirea geometrică a schemei.

2. **Validitatea.** Validitatea schemei de reprezentare este determinată de multimea de reprezentări sau modele valide pe care le poate produce. Dacă o schemă de reprezentare produce un model invalid, sistemul CAD/CAM utilizat se poate bloca, sau baza de date grafice care contine modelul poate fi pierdută sau deteriorată dacă se încearcă introducerea ei într-un algoritm. Verificările pentru corectitudine pot fi efectuate în trei moduri: testarea bazei de date rezultate prin intermediul unui algoritm, introducerea verificărilor chiar în procesul de generare

BUPT

a schemei sau proiectarea elementelor de schemă (cum ar fi primitivele) care pot fi manipulate prin intermediul unei sintaxe date.

3. **Neambiguitatea.** Această proprietate determină abilitatea schemei de a permite analiza si alte aplicatii ingineresti. O schemă completă trebuie să furnizeze modele cu date suficiente pentru efectuarea oricăror calcule geometrice [Hof87].

4. **Unicitatea.** Această proprietate este utilă pentru determinarea egalitătii obiectelor. Este dificilă dezvoltarea algoritmilor pentru detectarea echivalentei a două obiecte si dacă acesti algoritmi ar fi dezvoltati, ar consuma mult timp de calcul. Complexitatea problemei se datorează faptului că obiectele pot ocupa diferite pozitii în spatiu, pot avea diferite orientări si pot fi obtinute din aceleasi elemente, cu aceleasi operatii, dar în ordine diferită.

Există si alte proprietăti ale schemelor de reprezentare cum ar fi conciziunea, usurinta în creare si eficienta în contextul aplicatiei. Aceste proprietăti însă nu pot fi formalizate. Conciziunea este o măsură a cantitătii de date de care are nevoie o schemă pentru a descrie un obiect. Schemele de reprezentare concise generează baze de date compacte care contin putine date redundante, folosesc putină memorie [Zei91].

2.2. FUNDAMENTELE MODELĂRII SOLIDELOR

Teoria modelării solidelor are la bază elemente de geometrie, topologie, închideri geometrice, teoria multimilor, regularizarea operatiilor cu multimi, clasificarea elementelor multimii si vecinătăti. Acestea furnizează fundamente matematice riguroase pentru dezvoltarea si analiza solidelor [Zei91] [Fla87].

2.2.1. Geometria si topologia solidelor

Reprezentarea unui obiect prin frontiere contine informatia topologică si geometrică a solidului. Informatia topologică se referă la conexiunile dintre cele trei entităti de bază: fete, muchii si vertecsi. Informatia geometrică este folosită pentru a specifica informatia metrică si tipul entitătii (un vertex, un tip de fată sau muchie). Informatia metrică, la rândul ei, contine parametrii configuratiei într-un sistem local de coordonate (cum ar fi raza unei suprafete cilindrice sau semiunghiul unei suprafete conice) si parametrii miscării rigide prin care se

BUPT

pozitionează si se orientează entitatea geometrică în pozitia dorită, relativ la un sistem de coordonate global.

Problema cea mai delicată a sistemelor pentru modelarea solidelor o reprezintă obtinerea de solide corecte, oricât de sofisticate. Dacă acest lucru nu este realizat (solidul obtinut este incorect) nu mai pot fi efectuate alte operatii.

Modelatoarele de solide continuă să fie utilizate în ciuda faptului că se poate ajunge în situatia în care este refuzată crearea anumitor forme, sau în situatia în care încercarea de creare a unei forme duce la blocarea sistemului.

Conceptul principal în definirea modelelor de solide îl reprezintă frontiera unui volum, frontiera care din punct de vedere topologic împarte spatiul tridimensional în trei părti distincte: partea interioară, frontiera si partea exterioară a volumului dat.

Modelul solid al unui obiect este definit din punct de vedere matematic ca o multime S de puncte din spatiul euclidian tridimensional (E^3). Dacă se notează interiorul si frontiera multimii prin iS si respectiv bS, se poate scrie:

$$S = \mathrm{i}S \cup \mathrm{b}S \qquad (2.1)$$

si dacă definim exteriorul prin cS (complementul lui S), atunci:

$$W = \mathrm{i}S \cup \mathrm{b}S \cup \mathrm{c}S \qquad (2.2)$$

unde W este multimea totală, care în cazul lui E^3 reprezintă toate punctele tridimensionale posibile.

2.2.2. Inchiderea geometrică

Definitia solidului dată prin (2.1) introduce conceptul de închidere geometrică care implică faptul că interiorul solidului este geometric închis de frontiera sa. Astfel, relatia (2.1) poate fi rescrisă sub forma:

$$S = \mathrm{k}S \qquad (2.3)$$

unde kS reprezintă închiderea geometrică a solidului sau multimii de puncte S, si este dată de membrul drept al relatiei (2.1), adică $\mathrm{k}S = \mathrm{i}S \cup \mathrm{b}S$.

Figura 2.2 arată explicatia geometrică a ecuatiilor (2.1) si (2.3).

BUPT

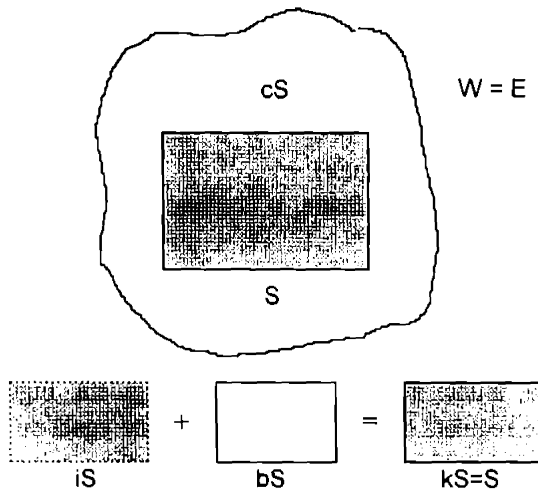


Figura 2.2.Definitiile solidului si închiderii geometrice

2.2.3.Teoria multimilor

Operatiile cu multimi (complementare, reuniune, intersectie si diferentă) se mai numesc si operatii teoretice pentru a le deosebi de operatiile folosite în cazul modelării geometrice. Dacă operatiile teoretice ar fi folosite în modelarea geometrică pentru a construi obiecte complexe plecând de la unele primitive, operatia de complementare ar esua, deoarece ar putea crea geometrii inacceptabile. Celelalte operatii ar putea genera obiecte lipsite de închidere geometrică, neadecvate pentru alte aplicatii.

Pentru a evita aceste probleme, multimea de puncte care reprezintă obiectele si operatiile cu multimi care operează asupra lor trebuie să fie regularizate. Multimile regulate si operatiile cu multimi regularizate se consideră că fac parte din algebra booleană.

O multime regulată este definită ca fiind o multime geometric închisă (relatia 2.3). Notiunea de multime regulată este introdusă în modelarea geometrică pentru a asigura validitatea obiectelor pe care le reprezintă si de aceea elimină obiectele fără sens. Sub închiderea geometrică, o multime regulată

BUPT

contine submultimile interior si frontieră. Frontiera închide interiorul si orice punct de pe frontieră este în contact cu un punct din interior. Cu alte cuvinte, frontiera se comportă ca un învelis ce acoperă interiorul. Multimea S din figura 2.2 este un exemplu de multime regulată, în timp ce figura 2.1 arată o multime neregulată deoarece contine o muchie si o fată care nu sunt în contact cu interiorul multimii (în acest caz cutia).

Din punct de vedere matematic, o multime *S* este regulată dacă si numai dacă:

$$S = \mathrm{ki}S \tag{2.4}$$

Această ecuatie arată că dacă închiderea interiorului unei multimi date este egală cu multimea, atunci această multime este regulată. Figura 2.3a arată că multimea *S* nu este regulată deoarece $S' = \mathrm{ki}S$ nu este egală cu *S*. Unele sisteme de modelare folosesc multimi regulate care sunt deschise sau nu au frontiere. O multime S este regulată deschisă dacă si numai dacă:

$$S = \mathrm{ik}S \tag{2.5}$$

Această ecuatie arată că o multime este regulată deschisă dacă interiorul

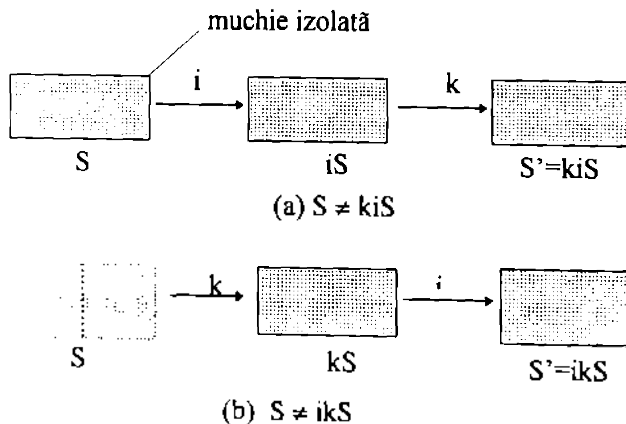


Figura 2.3. Regularitatea multimilor

BUPT

închiderii sale este multimea initială. Figura 2.3b arată că multimea S nu este regulată deschisă deoarece $S' = \text{ik}S$ nu este egală cu S.

Operatiile cu multimi, cunoscute si ca operatii booleene, trebuie regularizate pentru a se asigura generarea întotdeauna de multimi regulate. Pentru modelarea geometrică, acest lucru înseamnă că modelele solidelor construite din primitive bine definite sunt totdeauna valide si reprezintă obiecte valide. Operatiile cu multimi regularizate păstrează omogenitatea si dimensionalitatea spatială, adică prin utilizarea acestor operatii nu se pot obtine elemente care să nu aibă contact cu interiorul si nici nu se pot obtine obiecte având o dimensiune mai mică decât dimensiunea obiectelor folosite ca operanzi. Regularizarea operatorilor este în mod special utilă când utilizatorii au de lucru cu obiecte ale căror fete se suprapun (obiecte tangente).

Bazat pe descrierea de mai sus, operatorii regularizati pot fi descrisi după cum urmează:

$$P \cup^* Q = \text{ki}(P \cup Q) \quad (2.6)$$

$$P \cap^* Q = \text{ki}(P \cap Q) \quad (2.7)$$

$$P -^* Q = \text{ki}(P - Q) \quad (2.8)$$

$$\text{c}^* P = \text{ki}(\text{c}P) \quad (2.9)$$

unde simbolul * din dreapta operatorilor marchează regularizarea. Multimile P si Q din ecuatiile de mai sus sunt multimi oarecare. Dacă două multimi X si Y sunt regulate (r-sets), cum este totdeauna cazul modelelor geometrice, atunci ecuatiile (2.6) - (2.9) devin:

$$X \cup^* Y = X \cup Y \quad (2.10)$$

$$X \cap^* Y = X \cap Y \iff \text{b}X \text{ si } \text{b}Y \text{ nu se suprapun} \quad (2.11)$$

$$X -^* Y = \text{k}(X - Y) \quad (2.12)$$

$$\text{c}^* X = \text{k}(\text{c}X)$$

(2.13)

Dacă bX si bY se suprapun în ecuatia (2.11), se foloseste ecuatia (2.7) si rezultatul este obiectul nul. Figura 2.4 ilustrează geometric ecuatiile (2.6)-(2.13). Figura nu contine operatia de complementare [Zei91].

BUPT

muchii izolate

P Q ∪ i k

P ∪ Q   i(P∪Q)   ki (P ∪ Q) = P ∪* Q

∩ i k

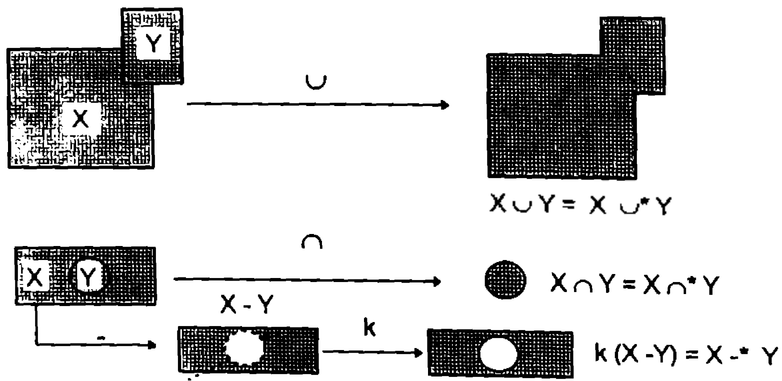
P ∪ Q   i (P ∩ Q)   ki (P ∩ Q) = P∩* Q

aceste muchii nu există

i k

P - Q   i (P – Q)   ki (P – Q) = P–* Q

(a) Multimi neregulate



această frontieră
nu există

(b) Multimi regulate

Figura 2.4. Operatii pe multimi

BUPT

2.2.4. Clasificarea elementelor multimii

În multe probleme geometrice ce implică modele ale solidelor este necesar să se cunoască ce punct, linie sau portiune a unui alt solid intersectează un solid dat. În cazul intersectiilor punct/solid, linie/solid sau solid/solid trebuie să se cunoască ce puncte, segmente sau portiuni de solide se află înăuntrul, în afară sau pe frontiera unui solid dat. Aceste probleme de intersectie geometrică au multe aplicatii practice ingineresti utile. De exemplu, intersectia linie/solid poate fi utilizată la umbrirea unui solid prin algoritmul de urmărire a razei, în timp ce intersectia solid/solid poate fi folosită pentru a verifica interferenta dintre două solide [Zei91][Ben94].

2.3. GENERAREA SOLIDELOR PRIN OPERATII BOOLEENE

Utilizarea modelării solidelor este foarte complicată si de obicei este limitată fie de complexitatea, fie de precizia geometrică. Din această cauză, în anumite situatii, constructia solidului nu poate avea loc sau ea duce la blocarea sistemului [Sav94] [Mu194] [Mu294][Mu394].

Modulul pentru modelarea geometrică reprezintă partea principală a unui sistem CAD. Modulele aplicatiei sunt direct legate si limitate de diversele moduri de reprezentare pe care sistemul le permite. Cele mai multe sisteme CAD furnizează modelarea "wire-frame", prin suprafete si prin solide, si în fiecare caz trebuie luate în considerare diferitele entităti ce stau la baza fiecărei reprezentări. De asemenea trebuie luată în considerare legatura ce trebuie asigurată între diferitele modalităti de reprezentare si aplicatiile dorite [Zei91][Ben94].

Reprezentările prin frontiere contin o componentă topologică si una geometrică. Este necesar să se poată verifica corectitudinea modelului obtinut, adică faptul că elementele topologice sunt neambiguu legate de entitătile geometrice. În sistemele de modelare algebrică, o structură topologică neambiguă poate conduce la interpretări geometrice ambigue pentru coordonate identice de vertexi, ecuatii ale planelor fetelor si descrieri de muchii (prin intersectia a două fete adiacente). Verificarea corectitudinii din punct de vedere topologic evită conversia unei descrieri neambigue a unui solid obtinut prin utilizarea geometriei constructive într-o descriere ambiguă ce foloseste reprezentarea prin frontiere. [Hof87][Ben94]

BUPT

Ambiguitatea nu se datorează în general lipsei de informatie topologică. De cele mai multe ori, ambiguitatea rezultă din faptul că muchiile se reprezintă ca intersectii de fete adiacente. Acest lucru este corect în cazul solidelor poliedrale, dar nu si în cazul solidelor mărginite de suprafete curbe.

Cele mai importante operatii efectuate de un modelator de solide sunt operatiile booleene (reuniunea, intersectia si diferenta). Aceste operatii pot produce obiecte neregulate, ca de exemplu cele care contin părti de dimensiuni diferite (un obiect tridimensional ce contine părti bidimensionale sau cu o singură dimensiune) sau cele nemultipluconexe (un obiect ce contine o muchie sau un vertex ce se sprijină pe interiorul unei fete).

Se recomandă ca în cazul muchiilor care apartin la mai mult de două fete distincte să nu se efectueze operatia booleană, iar cazurile de suprapunere muchie-muchie, muchie-fată sau fată-fată să fie tratate ca si cum entitătile geometrice se ating fără a avea puncte comune [Ben94] [Fla87].

2.4. TEHNICI DE REPREZENTARE A MODELELOR GEOMETRICE 3D

Informatiile necesare construirii unui obiect trebuie stocate într-o "bază de date", într-o structură ordonată de numere si caractere. Structura bazei de date influentează puternic viteza de lucru, memoria necesară, flexibilitatea si usurinta de scriere a programului care operează asupra obiectului.

Principalele moduri de reprezentare a informatiilor geometrice despre obiectele 3D în baza de date grafice pot fi clasificate în functie de elementul geometric de bază utilizat în descrierea informatiei. Astfel pot fi folosite reprezentarea prin puncte, prin segmente, prin poligoane sau prin volume. Fiecare dintre ele prezintă avantaje si dezavantaje în functie de contextul în care este folosită [Zei91][Dog88][Tan89].

2.4.1. Reprezentări prin puncte

In cazul corpurilor cu suprafete plane, reprezentarea poate fi făcută prin punctele care reprezintă vârfurile fetelor. Dacă corpurile au suprafete curbe, reprezentarea prin puncte poate fi făcută prin fatetare cu o precizie limitată, care să se încadreze în limita admisă pentru erori. Acest mod de reprezentare este

BUPT

foarte simplu, dar în multe aplicatii poate fi insuficient prin cantitatea de informatii oferită.

Elementul de bază utilizat pentru reprezentarea prin puncte este lista de vertexi. Un vertex este un punct de pe suprafata unui model în care se întâlnesc sau din care pornesc - una sau mai multe linii folosite în reprezentarea corpului respectiv. Termenul s-a extins si pentru reprezentarea prin puncte, unde nu apar linii ca elemente ale desenului rezultat. O listă de vertexi contine numărul total de puncte si coordonatele 3D ale fiecărui punct.

Tot prin puncte se pot descrie corpurile reprezentate prin sectiuni transversale. Acestea decurg direct din reprezentările prin puncte situate în sectiuni transversale paralele între ele. Punctele situate într-o aceeasi sectiune se unesc între ele, prin segmente de dreaptă, astfel încât să apară reprezentat conturul sectiunii respective [Dog88][Tan89].

2.4.2. Reprezentări prin segmente

Reprezentarea prin segmente, numită si reprezentarea wire-frame, nu foloseste notiunile de volum si suprafată. Un corp este reprezentat ca o multime de segmente de dreaptă sau portiuni de curbă.

Memorarea datelor pentru realizarea unei reprezentări wire-frame a unui obiect se poate face sub forma de:

a) Segmente explicite, când obiectul este văzut ca o colectie de segmente, pentru care se cunoaste numărul total de elemente, iar pentru fiecare segment se precizează coordonatele extremitătilor.

b) Segmente implicite, când fiecare segment este precizat printr-o pereche de indici care identifică capetele acestuia într-o listă de vertexi.

Obiectul va fi descris prin numărul total de puncte, numărul total de segmente, lista de vertexi cu numărul curent si coordonatele fiecărui vertex si lista de segmente cu numărul curent si numărul vertexilor care corespund capetelor segmentului.

c) Linii date prin indici. Atunci când o linie poligonală poate fi descrisă prin concatenarea unui sir de segmente, este mult mai indicat să folosim tipul de structură de date ce contine numărul total de puncte, numărul total de linii, lista punctelor cu numărul curent si coordonatele vertexilor, lista liniilor cu numărul curent si sublista numerelor de vertexi care formează linia.

BUPT

d) Sectiuni transversale si linii longitudinale. Aici se procedează ca în cazul liniilor date prin indici, dar curbele transversale sunt în marea lor majoritate chiar sectiunile transversale folosite la generarea corpului.

Desi reprezentarea wire-frame este oarecum simplistă si nu furnizează informatii complete asupra geometriei corpului, datorită usurintei de utilizare si a rapidității de afisare a reprezentării pe display ea este astăzi foarte mult folosită. Folosind tehnica wire-frame pentru vizualizare, se pot obtine viteze mari de lucru si chiar efecte interesante de animatie folosind un material putin sofisticat [Dog88] [Tan89] .

2.4.3. Reprezentări prin poligoane

Reprezentarea "wire-frame" a unui obiect 3D nu permite definirea unor suprafete si deci calcularea ariilor, volumelor, maselor, centrelor de greutate sau afisarea pe ecran a portiunii vizibile a obiectului analizat. Reprezentările simple care permit recunoasterea unor suprafete si efectuarea de calcule relative la aceste suprafete se obtin prin două procedee de modelare a corpurilor: modelarea suprafetelor si modelarea solidului. In primul caz, un corp este modelat prin precizarea frontierei sale, deci prin modelarea unei suprafete. Aceasta din urmă poate fi obtinută ca suprafată a unui poliedru, compusă dintr-o retea de fatete poligonale plane sau ca suprafată strîmbă în spatiu, compusă din portiuni de suprafete curbe [Tan89] .

Reprezentarea prin poligoane se bazează pe modelarea obiectelor 3D prin una sau mai multe suprafete de poliedre. Fiecare suprafată poliedrală este privită ca o colectie de fete poligonale plane adiacente. Dacă obiectul real are suprafete curbe, modelul său poligonal este evident aproximativ. Aproximarea poate fi făcută oricât de mică prin mărirea numărului de fete poligonale plane care modelează o suprafată curbă. Apare dezavantajul unui necesar sporit de memorie, dar acesta este redus de faptul că algoritmii care procesează suprafete poligonale plane sunt mult mai simpli decât cei pentru suprafete curbe.

Din acest motiv, majoritatea aplicatiilor care nu implică prelucrarea efectivă a corpului analizat (CAM) apelează la modelarea poliedrală, făcînd un compromis între precizie pe de o parte, memorie, simplitate si viteză de lucru pe de altă parte [Zei91].

Modelele 3D construite prin enumerarea fetelor ce separă obiectele de restul "lumii" (definite prin "coaja" lor) se numesc modele prin frontiere (B-rep). Modelul ar trebui să memoreze si pozitia interiorului obiectului fată de fete.

BUPT

Tehnica este avantajoasă pentru vizualizări, dar nu este bine adaptată pentru operatii analitice, precum calculul centrului de greutate [Ben94].

Elementul topologic central al reprezentării prin frontiere este o entitate numită flaps, care poate fi imaginată ca o portiune dintr-o fată care se sprijină pe o muchie orientată. Formal, un flaps este o reprezentare explicită a vecinătătii unei muchii 2D, o vecinătate relativă la o fată *f*, formată din puncte din interiorul fetei *f* ce contine muchia *e*.

Reprezentarea prin frontiere este alcătuită din două liste de muchii si fete. O fată este specificată printr-o listă de flaps-uri, în care fiecare flaps apartine unei singure fete, si este incident unei singure muchii. Fiecare muchie are un număr par de flapsuri care sunt incidente ei.

Structura de date aleasă pentru solidele rationale contine următoarele articole:

Articolul Solid care contine lista tuturor fetelor solidului, lista tuturor muchiilor solidului si bounding-box-ul solidului.

Articolul Fată care contine coeficientii a, b, c si d, ai ecuatiei planului fetei ($ax + by + cz + d = 0$), coeficienti exprimati prin numere rationale (se consideră plane orientate, astfel încât gradientul (a, b, c) să indice întotdeauna o normală cu sensul dinspre interiorul spre exteriorul solidului), lista tuturor flapsurilor fetei si bounding-box-ul fetei.

Articolul Muchie care contine două triplete p_1 si p_2 ale coordonatelor rationale care definesc cele două capete ale muchiei, în sensul că p_1 precede pe p_2 în ordine lexicografică (muchiile sunt unice si sunt totdeauna memorate ca perechi ordonate (p_1 si p_2)), lista pointerilor la toate flapsurile incidente si o listă a punctelor de intersectie în care muchia este divizată în timpul algoritmului de intersectie.

Articolul flaps care contine un pointer la muchia atasată *e*, un pointer la fata atasată *f* si latura *s* a flapsului, definită ca un întreg (± 1), după cum este arătat mai jos.

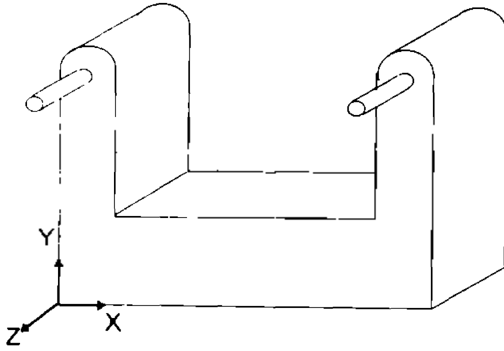
Pentru ca o reprezentare prin frontiere să fie validă, trebuie să îndeplinească urmatoarele trei conditii:

- Două muchii distincte nu trebuie nici să se suprapună, nici să se intersecteze în alt punct decât într-un capăt comun.

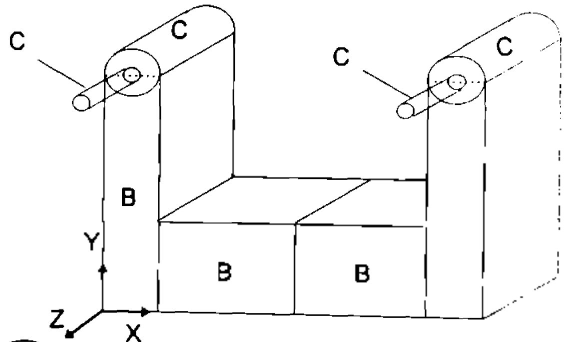
- Două fete distincte se pot intersecta numai după muchii ce apar în lista de muchii.

- O muchie trebuie să aibă un număr par de flapsuri incidente ei. Muchiile inutile care au exact două flapsuri incidente în aceeasi fată trebuie sterse din reprezentarea prin frontiere.

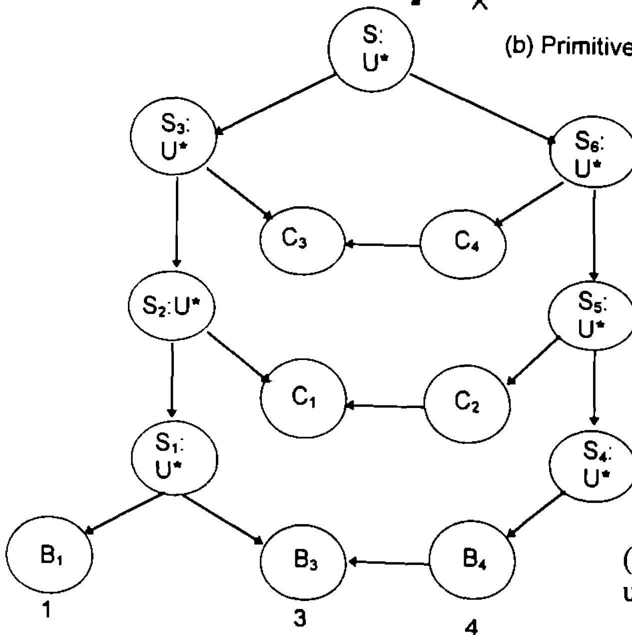
BUPT



(a) Solid tipic



(b) Primitive



(c) Graful CSG al unui solid tipic

Figura 2.5. Reprezentarea CSG a unui solid tipic

BUPT

2.4.4. Reprezentări prin volume

Există mai multe modalităti de reprezentare a obiectelor prin volume [Zei91].

Modelarea prin geometria constructivă a solidelor (CSG - Constructive Solid Geometry) permite crearea modelelor complexe prin gruparea componentelor mai simple, denumite sub-obiecte sau primitive. Exemple de sub-obiecte frecvente sunt: cuburi, conuri, cilindri, sfere. În tehnica CSG, modelul este reprezentat printr-un arbore (sau graf, pentru modele complexe). La capetele ramurilor arborelui se află obiecte primitive parametrabile, iar în noduri - operatori pe multimi. Fiecărui nod îi corespunde un sub-obiect (chiar dacă nu este calculat).

Tehnica CSG este avantajoasă în cazul modificării geometriei, dar nu este adaptată vizualizării. De aceea, uneori se păstrează în memorie două versiuni de model, una CSG si una B-rep, pentru vizualizare. In figura 2.5c este prezentat modelul CSG al obiectului din figura 2.5a [Maz94].

Modelarea prin enumerare spatială stochează modelul sub forma unei multimi de obiecte "atomice" volumice, asemănător stocării imaginilor 2D prin descrierea multimii de pixeli constituenti. Elementul volumic ce corespunde unui pixel este denumit voxel (volume element). Deoarece enumerarea tuturor punctelor volumice ar consuma memorie enormă, se utilizează obiecte unitare speciale. De aceea s-a dezvoltat o metodă de divizare recursivă, cunoscută sub numele de quadtree în 2D si "octree" în 3D. Un mare avantaj al acestei metode il reprezintă posibilitatea de descriere a obiectelor complexe neregulate, de tipul celor existente în natură [Mu694].

Modelarea prin instantierea primitivelor se bazează pe reprezentarea analitică sau parametrică a obiectelor singulare. Un obiect va fi descris de modelul generator al întregii clase (familii) si de valorile parametrilor ce definesc obiectul particular. Pentru prelucrări mecanice sau operatii analitice, metoda este avantajoasă pentru descrierea portiunilor din obiecte, dar interfata cu utilizatorul este foarte complexă. Exemple ar fi petecele (poligoane spatiale generalizate) parametrice cubice.

BUPT

CAPITOLUL 3

VIZUALIZAREA MODELELOR 3D

3.1. TEHNICI DE VIZUALIZARE A MODELELOR 3D

Vizualizarea unui model este mijlocul cel mai convenabil de a recepta majoritatea caracteristicilor unui obiect virtual. Prin vizualizare se poate prezenta obiectul modelat operatorului (sub formă de vederi, sectiuni, izometric etc.) si se poate facilita interactiunea dintre operator si model.

Problemelele legate de vizualizarea modelelor au fost analizate si pe baza lucrarilor [Zei91][Dog88][Mu494][Mu694][MV194][MV394].

Vizualizarea obiectelor tridimensionale se poate face prin mai multe metode. Unele dintre acestea ar fi:

1.Afisarea de vederi ortogonale, ca în desenul tehnic.

2.Afisarea prin reprezentarea tuturor muchiilor (wire-frame). Vizualizarea în tehnica wire-frame are dezavantajul prezentei unor ambiguităti privind forma reală a obiectului, nefiind furnizate suficiente informatii privind forma spatială a obiectului. Are însă avantajul afisării tuturor liniilor constituente ale obiectului. Obiectul fiind transparent, se pot trage concluzii asupra corectitudinii sale si a corectei pozitionări a elementelor interne. Ambiguitatea reprezentării wireframe poate fi rezolvată prin trecerea la reprezentarea hidden lines.

3.Afisarea cu efect de perspectivă, pentru care se definesc un punct de vedere si o directie de privire, iar forma depinde de distanta fată de observator.

4.Afisarea cu simularea adâncimii (depth cueing), în care intensitatea luminoasă scade cu cresterea distantei fată de observator.

5.Afisarea cu părtile ascunse eliminate (hidden lines sau hidden surfaces).

6.Afisarea nuantată (shading), în care fetele obiectelor sunt nuantate în functie de pozitia si orientarea elementelor de suprafată relativ la sursele de

BUPT

lumină si la punctul de privire (pozitia simulată a ochiului), încercând să se simuleze efectele iluminărilor din lumea reală, inclusiv pe diferite culori si texturi ale suprafetelor.

7. Afisare stereoscopică în care se redau simultan două imagini, câte una pentru fiecare ochi, ce se privesc prin dispozitive speciale (ochelari, ecran polarizor sau cu microlentile cilindrice etc.).

8. Afisarea tip realitate virtuală în care se redau două imagini, pe două ecrane mici (câte unul pentru fiecare ochi) fixate de o cască, iar senzori speciali comandă modificarea punctului de vedere la calculul imaginii de afisat functie de miscările capului.

3.2. REALISMUL VIZUAL

Incă de la începuturile graficii pe calculator s-a dorit obtinerea de imagini cât mai bune ale obiectelor prin înlăturarea părtilor ascunse care nu ar putea fi văzute dacă în realitate obiectele ar fi construite de materiale opace. Muchiile si suprafetele unui obiect dat pot fi ascunse (invizibile) sau vizibile într-o imagine dată, în functie de directia din care este privit. Determinarea muchiilor si suprafetelor ascunse este una din cele mai disputate din domeniul graficii pe calculator. Solutia ei cere de obicei o cantitate mare de timp de calculator si spatiu de memorie. Există o serie de tehnici pentru reducerea acestor necesităti si pentru îmbunătătirea eficientei algoritmilor.

Solutia problemei de înlăturare a muchiilor si fetelor ascunse conduce la diferite concepte din domeniul calculatoarelor, în special sortarea, si din domeniul modelării geometrice, în special proiectii si intersectii. Această problemă poate fi vazută de asemenea ca o problemă de vizibilitate. De aceea, o întelegere clară a problemei si a solutiei ei este utilă si poate fi extinsă pentru a rezolva probleme ingineresti semnificative. Un exemplu ar fi problema vederii si planificării traiectoriei din aplicatiile robotilor. În problema vederii, pozitionarea si orientarea camerei furnizează directia vederii care la rândul ei poate fi folosită pentru a determina muchiile si fetele ascunse ale obiectelor întâlnite în mediul de lucru al robotului. În problema planificării traiectoriei, cunoasterea momentului în care o suprafată dată devine ascunsă din vizibilă (prin găsirea muchiilor si curbelor siluetei) poate fi folosită pentru a determina traiectoria minimă a capătului "mâinii" robotului. Punctele de pe suprafată în care se schimbă starea lor din

BUPT

vizibile în invizibile sau invers, pot fi considerate ca puncte critice care pot fi folosite ca intrări de către algoritmul de planificare a traiectoriei.

În prezent există o varietate de algoritmi pentru înlăturarea liniilor si suprafetelor ascunse. Dezvoltarea lor este influentată de tipurile de dispozitive grafice de afisare folosite (dacă sunt vector sau raster) si de tipul structurilor de date sau a modelării geometrice cu care lucrează (modelare prin cadre, suprafete sau solide). Unii algoritmi utilizează prelucrarea paralelă în locul celei seriale traditionale, pentru a le mări viteza de executie. Formalizarea si generalizarea acestor algoritmi sunt utile si sunt cerute atunci când se încearcă proiectarea si constructia de hardware dedicat care să suporte înlăturarea liniilor si suprafetelor ascunse, care să nu fie limitat la un singur algoritm. Oricum, nu este o sarcină usoară conversia diferitelor formulări ale algoritmilor într-o formă care să permită încadrarea lor într-o schemă generalizată.

Algoritmii care sunt aplicati unui set de obiecte pentru a înlătura părtile ascunse în scopul creării unei imagini mai realiste sunt de obicei clasificati în algoritmi pentru înlăturarea liniilor ascunse si algoritmi pentru înlăturarea suprafetelor ascunse. Primii folosesc dispozitive de trasare a liniilor cum ar fi display-urile de tip vector si plotter-ele, în timp ce cei din urmă folosesc display-uri de tip raster. Algoritmii de linii ascunse pot folosi display-uri de tip raster deoarece acestea permit trasarea liniilor. În schimb, algoritmii de suprafete ascunse nu pot folosi display-uri de tip vector. Din punctul de vedere al modelării geometrice, această clasificare este confuză si înselătoare. Înlăturarea liniilor ascunse nu înseamnă (după cum ar sugera numele) că este aplicabilă numai modelelor cadru. Analog, înlăturarea suprafetelor ascunse nu este aplicabilă numai modelării prin suprafete simple. De fapt, algoritmii pentru înlăturarea părtilor ascunse dintr-o imagine nu pot fi aplicati direct modelelor ce folosesc cadre sau suprafete. Ei au nevoie de o structura de date neambiguă care să reprezinte obiectul ca fiind format din fete orientate. Aceasta înseamnă că fiecare fată are o normală la suprafată cu o orientare consecventă (de exemplu pozitivă dacă muchiile sunt introduse în ordinea dată de sensul antiorar); deci, obiectele poliedrale sunt reprezentate prin poligoane pline orientate. Aceste poligoane pot fi obtinute dintr-un model ce foloseste reprezentarea prin cadre, suprafete sau solide. Utilizatorii ar trebui să introducă informatie suplimentară pentru a identifica fetele si orientarea în cazul modelelor ce folosesc cadre sau orientarea în cazul modelelor ce folosesc suprafete. Modelele solide furnizează automat aceste informatii. Această clasificare are drept scop prezentarea în ordine istorică a dezvoltării algoritmilor de vizibilitate.

BUPT

Algoritmii pentru linii si suprafete ascunse au fost clasificati în algoritmi spatiu - obiect, algoritmi spatiu - imagine si o combinatie a lor (algoritmi hibrizi). Algoritmii spatiu -imagine pot fi la rândul lor împărtiti în algoritmi raster si algoritmi vector. Algoritmii raster folosesc o reprezentare sub formă de matrice de pixeli a imaginii, iar algoritmii vector folosesc coordonatele capetelor segmentelor în reprezentarea imaginii. Un algoritm spatiu - obiect foloseste relatiile spatiale si geometrice dintre obiectele din scenă pentru a determina părtile ascunse si părtile vizibile ale acestor obiecte. Un algoritm spatiu - imagine, pe de altă parte, se concentrează pe imaginea de pe ecran pentru a determina ceea ce este vizibil, de exemplu, în fiecare pixel în cazul algoritmilor raster. Cei mai multi algoritmi de suprafete ascunse folosesc metode spatiu - imagine raster, în timp ce cei mai multi algoritmi de linii ascunse folosesc metodele spatiu - obiect.

Cele două abordări (spatiu - obiect si spatiu - imagine) pentru obtinerea realismului vizual prezintă caracteristici diferite. Algoritmii spatiu - obiect sunt mai exacti decât algoritmii spatiu - imagine. Primii efectuează calcule geometrice (cum ar fi intersectiile) folosind precizia virgulei flotante a hardware-ului calculatorului, în timp ce ultimii efectuează calcule cu precizia egală cu rezolutia ecranului diplay-ului folosit la reprezentare. De aceea, mărirea unei imagini prin algoritmii spatiu - obiect nu reduce calitatea reprezentării, asa cum se întâmplă în cazul măririi imaginii prin algoritmii spatiu - imagine. Pe măsură ce complexitatea scenei creste (un număr mai mare de obiecte în scenă), timpul de calcul creste mai repede pentru algoritmii spatiu - obiect decât pentru cei spatiu - imagine.

3.3. ÎNLĂTURAREA LINIILOR ASCUNSE

3.3.1. Vizibilitatea reprezentărilor obiectelor

Vizibilitatea părtilor obiectelor dintr-o scenă depinde de pozitia din care se priveste (pozitia ochiului), de tipul proiectiei (ortogonale sau perspective) si de adâncimea sau distanta dintre diferitele fete ale diferitelor obiecte din scenă, fată de punctul de privire (ochi). Înlăturarea liniilor ascunse din vederile perspective este o problemă destul de complexă. Trebuie luate în considerare multe linii ce pleacă din punctul din care se priveste (raze vizuale) si trebuie calculate intersectiile lor cu fetele obiectelor. Complexitatea problemei se reduce considerabil dacă se folosesc vederile ortografice deoarece nu mai sunt necesare

BUPT

intersectiile. De aceea este o practică uzuală aplicarea transformărilor perspective tuturor punctelor din scenă si apoi aplicarea algoritmilor ortografici de vizibilitate pentru linii ascunse asupra setului de puncte rezultate (transformate). Aceasta este echivalent cu a spune că vederea ortografică a obiectelor transformate (perspective) este identică cu vederea perspectivă a obiectelor originale (netransformate). Din acest motiv, interesează numai algoritmii ortografici pentru linii ascunse.

Compararea adâncimilor este criteriul principal utilizat de algoritmii liniilor ascunse pentru determinarea vizibilitătii. Comparatiile de adâncime se fac de obicei după efectuarea transformărilor de vedere corespunzătoare pentru proiectiile ortografice, respectiv perspective.

Compararea adâncimii determină dacă un punct proiectat P_{1v} (x_{1v}, y_{1v}) dintr-o vedere dată ascunde un alt punct $P_{2v}(x_{2v}, y_{2v})$. Acest lucru este echivalent cu a determina dacă cele două puncte originale corespunzătoare P_1 si P_2 se proiectează în acelasi punct, adică dacă $x_{1v} = x_{2v}$ si $y_{1v} = y_{2v}$. În acest caz compararea lui z_{1v} si z_{2v} decide care punct este mai apropiat de ochi (figura 3.1) [Zei91] [Dog88].

3.3.2. Tehnici de vizibilitate

Dacă criteriul de comparare a adâncimilor este utilizat singur, fără alte îmbunătătiri, numărul de comparatii creste rapid (pentru n puncte sunt necesare 2^n teste), lucru care conduce la dificultăti de memorare si gestiune a rezultatelor obtinute prin algoritmul de linii ascunse. Efectul este că algoritmul este încetinit din cauza calculelor pentru obtinerea imaginii finale. Există diferite tehnici de vizibilitate care îmbunătătesc acesti algoritmi. În general, aceste tehnici încearcă să stabilească relatiile dintre poligoane si muchii în planul de reprezentare. În mod normal, tehnicile verifică suprapunerea perechilor de poligoane (numite uneori si comparatii laterale) în planul de reprezentare (ecran). Dacă apar suprapuneri, comparatiile de adâncime sunt folosite pentru a determina dacă un poligon întreg sau o parte a sa este ascunsă de un alt poligon. Atât comparatiile laterale cât si comparatiile de adâncime se fac în sistemul de coordonate al reprezentării [Zei91].

BUPT

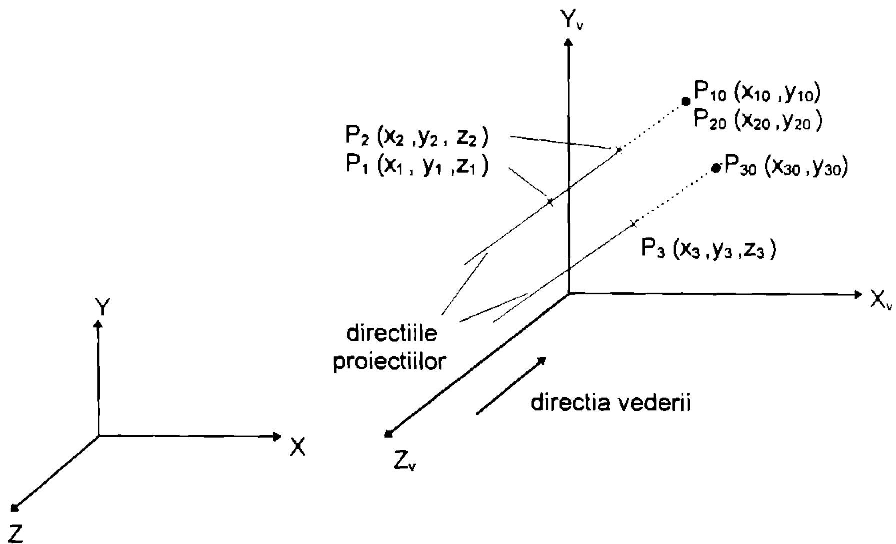


Figura 3.1.Compararea adâncimii pentru proiectii ortografice

3.3.3. Sortarea

Multi algoritmi de vizibilitate (algoritmi pentru linii, suprafete si solide ascunse) utilizează mult operatiile de sortare. Pentru acesti algoritmi, tehnicile de sortare si căutare operează asupra înregistrărilor din baza de date a scenei. Aceste înregistrări contin de obicei informatii geometrice, topologice si de vedere asupra poligoanelor si fetelor care alcătuiesc scena. Sortarea este o operatie care ordonează o multime de înregistrări date în functie de un criteriu selectat.

Timpul necesar pentru executia sortării depinde de numărul de înregistrări de prelucrat, de algoritmul care efectuează sortarea si de ordinea initială a înregistrărilor (dacă înregistrările sunt aleator sau semiordonat memorate). Există o multime de tehnici de sortare, dar acestea nu sunt prezentate aici [Zei91].

BUPT

3.3.4. Coerenta

In mod natural, elementele unei scene sau imaginea ei au unele legături între ele. Ansamblul acestor legături este numit coerentă. Algoritmii liniilor ascunse care utilizează coerenta în tehnicile lor de sortare sunt mai eficienti decât algoritmii care nu o folosesc. Coerenta este o măsură a vitezei cu care se schimbă o scenă sau imaginea ei. Ea descrie gradul în care scena sau imaginea ei este local constantă.

Coerenta unui set de date poate îmbunătăti în mod semnificativ viteza sortării. Schimbările gradate în aparenta unei scene sau a imaginii ei de la un loc la altul poate reduce cu mult numărul operatiilor de sortare.

Pot fi identificate mai multe tipuri de coerentă atât în spatiul obiect cât si în spatiul imagine:

1. *Coerenta muchiilor.* Vizibilitatea unei muchii se schimbă numai când se întretaie cu altă muchie.

2. *Coerenta fetelor.* Dacă o parte a unei fete este vizibilă, probabil că întreaga fată este vizibilă. Mai mult, penetrarea fetelor apare destul de rar, si de aceea nu este verificată în mod uzual în algoritmii de înlăturare a părtilor ascunse.

3. *Coerenta geometrică.* Muchiile care împart acelasi vertex sau fetele care împart aceeasi muchie au vizibilităti similare în cele mai multe cazuri. De exemplu, dacă trei muchii împart acelasi vertex, ele pot fi toate vizibile, toate invizibile sau două vizibile si una nu. Combinatia potrivită depinde de unghiul dintre oricare două muchii (mai mic sau mai mare decât 180°) si de pozitia oricărei muchii relativ la planul determinat de celelalte două.

4. *Coerenta cadrelor.* Un film (de exemplu animatie) nu are schimbări foarte mari de la un cadru la altul.

5. *Coerenta liniilor de scanare.* Segmentele unei scene vizibile pe o linie de scanare sunt foarte probabil vizibile si pe urmatoarea linie.

6. *Coerenta ariilor.* Un element particular (arie) al unei imagini si vecinătătile lui au probabilistic aceeasi vizibilitate si sunt influentate de aceeasi fată.

7. *Coerenta pe adâncime.* Diferitele suprafete dintr-o anumită portiune a ecranului sunt în general bine separate în adâncime, relativ la domeniul adâncimii fiecăreia.

Primele trei tipuri de coerentă au la bază reprezentarea spatiu-obiect, în timp ce restul se bazează pe reprezentarea spatiu-imagine. Dacă o imagine

BUPT

prezintă o coerentă particulară predominantă, ea va forma baza pentru algoritmul corespunzător de înlăturare a liniilor ascunse [Zei91].

3.3.5. Exemple de algoritmi pentru linii ascunse

La baza unui algoritm de vizibilitate stă testul dacă un element al unui obiect este sau nu ascuns de un altul. Un algoritm de vizibilitate poate fi descris ca o functie *va(H,C)*, unde *H* si *C* sunt multimile de muchii candidat, ascunse respectiv acoperitoare, din cadrul scenei considerate. De vreme ce fiecare muchie din multimea *H* trebuie testată pentru fiecare muchie din *C*, algoritmul necesită $H * C$ teste de vizibilitate. Diferitii algoritmi de vizibilitate sunt deosebiti în general prin strategia aplicată pentru testul de vizibilitate, care determină multimile *H* si *C*.

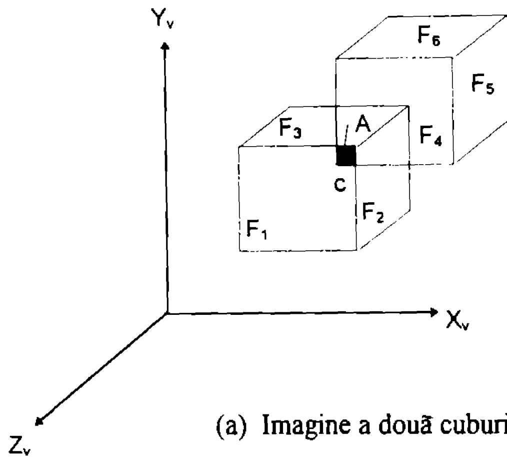
Algoritmii de înlăturare a liniilor ascunse pot folosi una din următoarele trei abordări: abordarea orientată pe muchii, abordarea orientată pe contururi sau abordarea orientată pe arii [Tan89] [Dog88].

Există o varietate de algoritmi pentru linii ascunse care utilizează una sau mai multe din tehnicile de vizibilitate discutate în 4.2 si urmează una din cele trei abordări. Acestia includ algoritmul cu priorităti, paradigma "plane-sweep", algoritmi pentru scene de mare complexitate, algoritmi pentru modele de elemente finite sau elemente plane, algoritmi orientati pe arii, algoritmul de suprapunere pentru suprafete definite parametric (retele u-v).

3.3.5.1. Algoritmul cu priorităti

Acest algoritm mai este cunoscut si ca algoritmul în adâncime sau "algoritmul z". Este bazat pe sortarea tuturor fetelor (poligoanelor) din scenă, în raport cu cea mai mare coordonată z a fiecăreia. Acest pas este cunoscut uneori ca si atribuire de priorităti. Dacă o fată intersectează mai multe fete sunt necesare si alte teste de vizibilitate pentru a rezolva orice ambiguitate. Acest pas constituie determinarea acoperirilor [Tan89] [Dog88].

BUPT

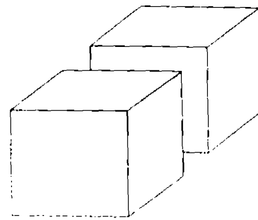


(a) Imagine a două cuburi

Fete	Lista de priorităţi	Fete	Lista de priorităţi	Fete	Lista de priorităţi	Fete	Lista de priorităţi
F_1	1	F_1	1	F_1	1	F_1	~~1~~ 2
F_2	1	F_2	1	F_2	2	F_2	~~1~~ 2
F_3	1	F_3	2	F_3		F_3	~~1~~ 2
F_4	2	F_4		F_4		F_4	1
F_5		F_5		F_5		F_5	1
F_6		F_6		F_6		F_6	1

Iteratia 1 Iteratia 2 Iteratia 3 Iteratia 4

(b) Stabilirea priorităţilor



(c) Imagine cu liniile ascunse

Figura 3.2. Algoritmul cu priorităţi

BUPT

Pentru a ilustra cum poate fi implementat algoritmul cu prioritâti, se considerâ o scenâ ca cea din figura 3.2. Figura prezintâ scena în sistemul de axe VCS (al vizualizării), unde ochiul este plasat la infinit pe directia pozitivă a axei Z_V.

In cadrul acestui algoritm se utilizează proiectia ortografică potrivită pentru a obtine vederea dorită (ale cărei linii ascunse urmează să fie îndepărtate) a scenei. Pentru a îmbunătăti eficienta algoritmului, se utilizează testul de suprafată pentru eliminarea fetelor din spate. Se obtine astfel o listă de fete care va fi sortată pentru atribuirea prioritătilor, prin compararea a câte două fete. Se reordonează listele de fete si de prioritâti astfel încât prioritatea cea mai mare este în capul listei. În cazul unui afisaj de tip rastru, eliminarea liniilor ascunse este realizată prin hardware. Trebuie doar afisate fetele în ordinea inversă a prioritătii lor. Oricare fete care vor fi ascunse de către altele vor fi astfel afisate primele, dar vor fi acoperite mai târziu, partial sau total, de fetele cu prioritate mai mare. În cazul unui afisaj vectorial, eliminarea liniilor ascunse trebuie făcută prin software, prin determinarea acoperirilor. În acest scop, muchiile unei fete sunt comparate cu toate celelalte muchii de prioritate mai mare. Poate fi creată o listă de muchii care păstrează lista tuturor segmentelor de dreaptă care vor fi desenate ca fiind vizibile. În acest caz sunt utile tehnici de vizibilitate, cum ar fi testul de continut si intersectia de muchii.

În cazul anumitor scene pot apărea ambiguitâti după aplicarea testului de prioritate. Figura 3.3 prezintă un caz în care ordinea fetelor este ciclică. Fata F_1 acoperă fata F_2, F_2 acoperă F_3, si F_3 acoperă F_1. Pentru a remedia această ambiguitate, algoritmului cu prioritâti trebuie să i se adauge software suplimentar pentru determinarea acoperirilor.

3.3.5.2. Algoritmul orientat pe arii

Algoritmul orientat pe arii divizează pas cu pas setul de date dintr-o scenă dată, până când sunt determinate si afisate toate ariile vizibile din cadrul scenei. În acest algoritm, ca si în algoritmul cu prioritâti, nu este permisă străpungerea fetelor. Considerând scena din figura 3.4a, algoritmul orientat pe arii poate fi descris după cum urmează:

BUPT

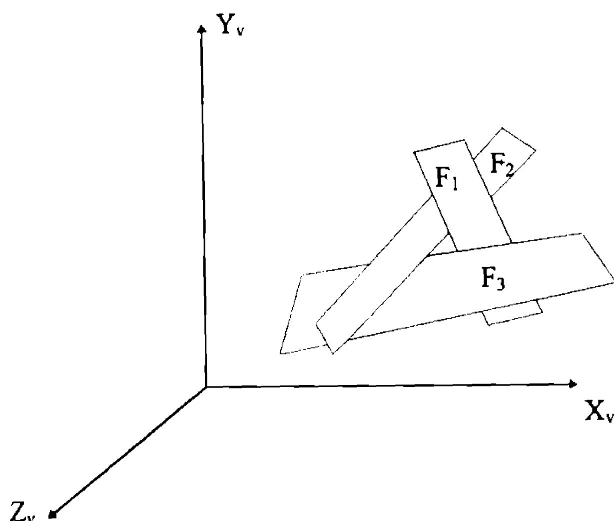


Figura 3.3. Exemplu cu ambiguitâti pentru algoritmul cu prioritâti

Se identifica contururilor poligonale. (Pentru scena prezentatâ în figura 3.4 s-au identificat douâ contururi poligonale închise S_1 si S_2.) Se atribuie valori de ascundere cantitativă (QH) muchiilor contururilor poligonale, prin intersectarea poligoanelor (poate fi utilizat mai întâi testul de continut ca si test rapid). Punctele de intersectie definesc punctele în care valoarea lui QH se poate modifica. Aplicând testul de adâncime punctelor de intersectie (P_1 si P_2 în figura 3.4) se determinâ segmentele ascunse ale muchiilor de contur. De exemplu, dacâ testul de adâncime în P_1 aratâ câ z_v în P_1 al S_1 este mai mic decât pentru S_2, muchia C_1C_2 este partial vizibilâ. Similar, testul de adâncime în P_2 aratâ câ muchia C_2C_3 este de asemenea partial vizibilâ [Zei91].

BUPT

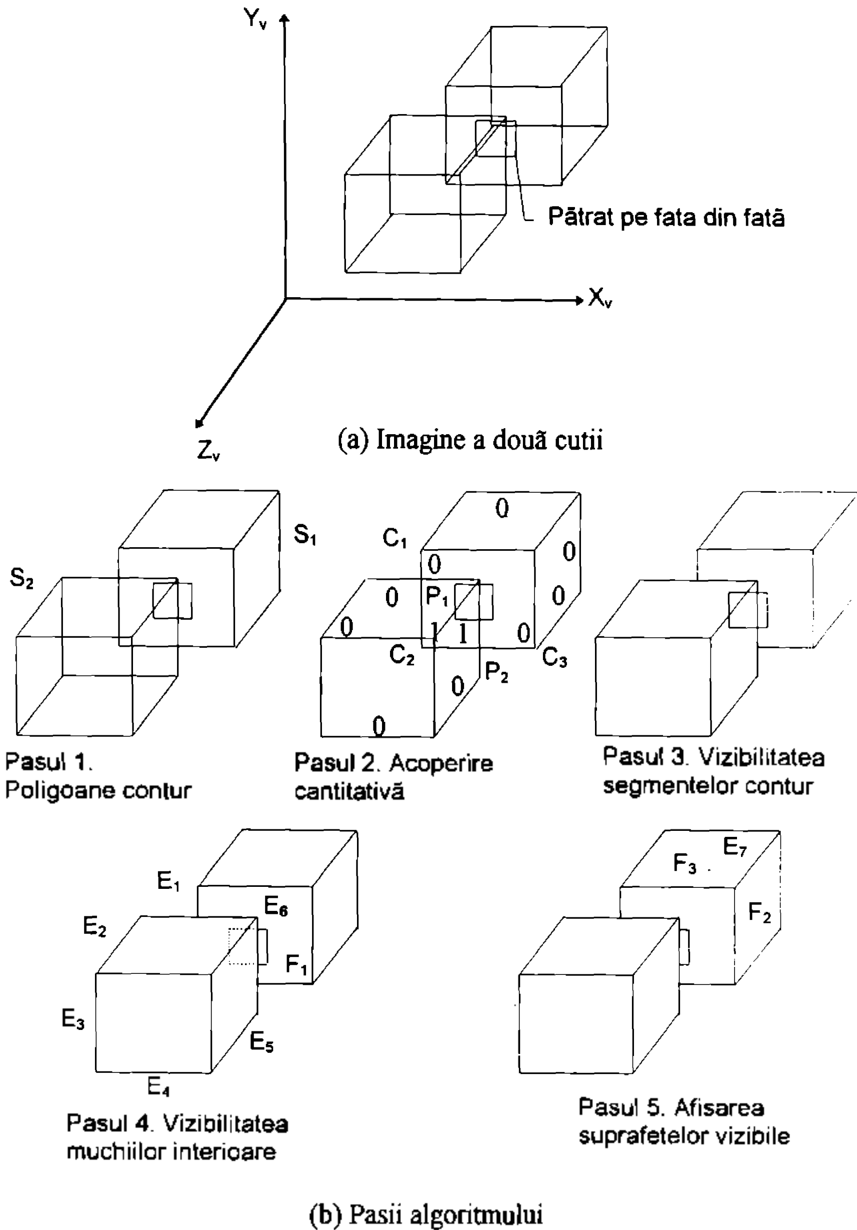


Figura 3.4. Algoritmul orientat pe arii

BUPT

Pentru a determina care segment al muchiei este vizibil poate fi utilizat testul de vizibilitate din 4.2.5. Determinarea valorilor QH pentru diferite muchii sau segmente de muchii ale contururilor poligonale este bazată pe testul de adâncime. Valoarea 0 indică faptul că o muchie sau segment este vizibilă si valoarea 1 faptul că această este invizibilă. Se determina segmentele de contur vizibile. Vizibilitatea segmentelor de contur poate fi determinată din valorile QH tinând cont de următoarele reguli. Dacă un contur poligonal închis este complet invizibil el nu trebuie luat în considerare în continuare. Altfel, sunt vizibile segmentele sale cu cele mai mici valori QH (pasul 3 în figura 3.3). Se intersecteaza segmentele de contur vizibile cu fetele partial vizibile, pentru a determina dacă segmentele de contur ascund (total sau partial) muchiile care nu sunt de contur în fetele partial vizibile. În figura 3.4 muchiile E_1..E_6 ale lui S_2 sunt intersectate cu muchiile interne (muchiile pătratului din interiorul fetei) ale lui F_1 si sunt determinate segmentele vizibile ale muchiilor interne.

Algoritmul evită orice calcule în plus, prin accesarea doar a muchiilor de contur ale contururilor poligonale acoperitoare si doar a fetelor partial vizibile. In final se afiseaza interiorul poligoanelor vizibile sau partial vizibile. Acest pas poate fi realizat folosind o stiva si enumerând toate fetele care se găsesc în interiorul unui contur poligonal. Stiva este initializată cu o fată vizibilă care are o muchie de contur. Stim că această fată apartine unei arii vizibile. Se începe extrăgând o fată (F_2) de pe stivă. Se examinează toate muchiile fetei. Dacă o muchie (E_7) nu este complet vizibilă atunci fata vecină (F_3) are de asemenea muchii vizibile [Tan89] [Zei91].

3.4. ÎNLĂTURAREA SUPRAFETELOR ASCUNSE

Există o largă varietate de asemenea algoritmi. Ei includ algoritmul buffer-ului z, algoritmul lui Watkin, algoritmul lui Warnock si algoritmul pictorului. Algoritmul Watkin este bazat pe coerenta liniilor de scanare, în timp ce algoritmul Warnock este bazat pe coerenta suprafetelor. Algoritmul pictorului este un algoritm cu prioritări [Zei91].

3.4.1. Algoritmul buffer-ului Z

Acesta este cunoscut si ca algoritmul buffer-ului de adâncime. În plus fată de buffer-ul de constructie (cadru) acest algoritm necesită un buffer z în care pot

BUPT

fi sortate valorile z pentru fiecare pixel. Buffer-ul z este initializat la cea mai mică valoare z, în timp ce buffer-ul de constructie (cadru) este initializat la valoarea pixelului de fond. Ambele buffere sunt indexate de coordonatele (x,y) ale pixelilor. Aceste coordonate sunt de fapt coordonatele ecranului. Algoritmul buffer-ului z functionează după cum urmează: Pentru fiecare poligon care face parte din scenă găsesc toti pixelii (x,y) care se găsesc pe poligon sau în interiorul acestuia atunci când poligonul este proiectat pe ecran. Pentru fiecare din acesti pixeli calculează adâncimea z a poligonului la (x,y). Dacă z > adâncime(x,y) poligonul este mai apropiat de ochi decât altele deja memorate pentru pixelul respectiv. În acest caz, buffer-ul z este actualizat setând adâncimea(x,y) la z. În mod similar, intensitatea locatiei corespunzătoare pixelului din buffer-ului de constructie este actualizată la intensitatea poligonului de la (x,y). După ce au fost prelucrate toate poligoanele, buffer-ul de constructie contine solutia [Tan89].

3.4.2. Algoritmul lui Warnock

Acesta este unul dintre primii algoritmi bazati pe coerenta suprafetelor. Acest algoritm rezolvă problema suprafetelor ascunse divizând recursiv imaginea în subimagini. Mai întâi încearcă să rezolve problema pentru o fereastră care acoperă întreaga imagine. Cazurile simple cum ar fi un poligon sau niciunul în fereastră sunt rezolvate cu usurintă. Dacă poligoanele se suprapun algoritmul încearcă să analizeze relatia dintre poligoane [Tan89].

Dacă algoritmul nu poate decide cu usurintă, subdivide fereastra în patru ferestre mai mici si aplică aceeasi tehnică pentru fiecare dintre acestea. Dacă una dintre cele patru ferestre este încă complexă ea este divizată în continuare în patru ferestre mai mici. Procesul recursiv ia sfârsit dacă problema suprafetelor ascunse poate fi rezolvată pentru toate ferestrele sau dacă fereastra se reduce la mărimea unui pixel. În acest caz, intensitatea pixelului este aleasă egală cu a poligonului vizibil în acel pixel. Rezultă astfel o structură arborescentă de ferestre.

Figura 3.5 prezintă aplicarea algoritmului lui Warnock scenei din figura 3.2a. S-ar putea enunta o regulă conform căreia orice fereastră este divizată recursiv dacă nu contine două poligoane. Într-un asemenea caz, comparând adâncimea z a poligoanelor se determină care dintre ele îl ascunde pe celălalt.

BUPT

3.5. ELIMINAREA SOLIDELOR ASCUNSE

Problema eliminării solidelor ascunse implică afisarea modelelor solidelor cu liniile sau suprafetele ascunse eliminate. Datorită complexitătii si lipsei de ambiguitate a modelelor solidelor, eliminarea solidelor ascunse este efectuată automat. De aceea, comenzile disponibile în sistemele CAD/CAM pentru eliminarea solidelor ascunse, cum ar fi comanda "ascunde solid", necesită un minim de interventie din partea utilizatorului. De fapt, tot ceea ce este necesar pentru executarea comenzii este ca utilizatorul să identifice solidul care trebuie ascuns. Structura de date a unui model solid contine toate informatiile necesare pentru rezolvarea problemei liniilor sau suprafetelor ascunse. Toti algoritmii discutati mai sus (3.3 si 3.4) sunt aplicabili eliminării solidelor ascunse pentru modelele B-rep. Algoritmi cum ar fi cel al buffer-ului z au fost extinsi la modele CSG. [Zei91]

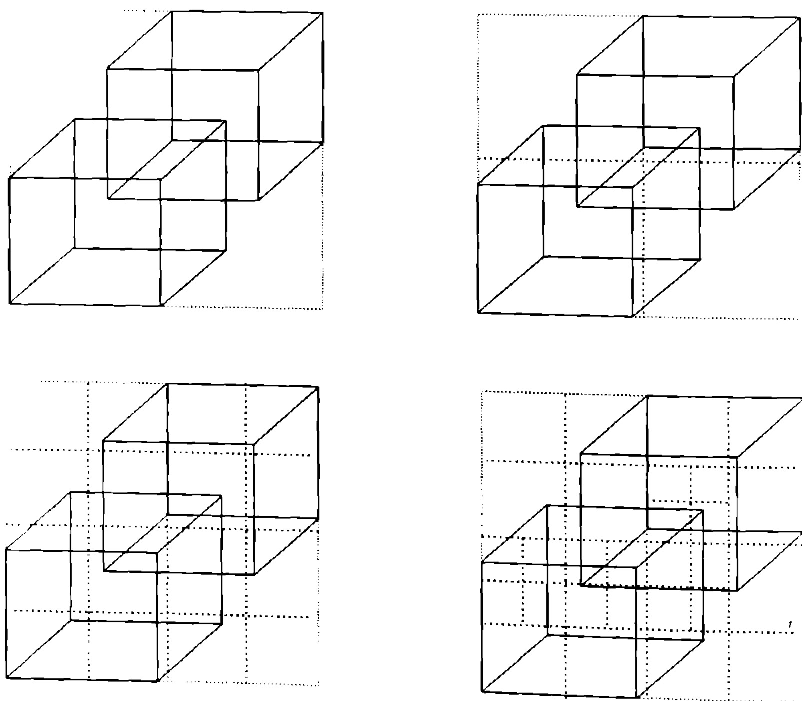


Figura 3.5. Algoritmul lui Warnock

BUPT

Pentru afisarea modelelor CSG trebuie rezolvate atât problema vizibilitătii cât si problema combinării solidelor primitive într-un model compus. Există trei abordări pentru afisarea modelelor CSG. Prima abordare converteste modelul CSG într-un model B-rep ("boundary representation model") care poate fi interpretat pentru vizualizare cu algoritmii pentru suprafete ascunse standard. A doua abordare utilizează o strategie de divizare spatială. Pentru a simplifica problemele combinatoriale, arborele CSG (semi-spatii) este "tăiat" simultan cu divizarea. Această divizare reduce evaluarea CSG la o simplă preprocesare înainte ca semi-spatiile să fie procesate cu tehnicile standard de rendering.

A treia abordare utilizează un algoritm pentru suprafete ascunse CSG care combină evaluarea CSG cu eliminarea suprafetelor ascunse pe baza clasificării razelor. Algoritmii de urmărire a razelor si de scanare a liniilor folosesc această abordare. Atractivitatea abordării stă în conversia problemei tridimensionale complexe a intersectiei solid/solid într-un calcul unidimensional de intersectie rază/solid.

3.5.1. Algoritmul cu urmărirea razei

Principalele calitătii ale acestui algoritm sunt simplitatea si posibilitatea extinderii acestuia. Cea mai complicată problemă numerică a algoritmului este găsirea punctelor în care liniile (razele) intersectează suprafetele. Urmărirea razelor (*ray tracing*) a fost utilizată în realismul vizual al solidelor pentru a genera schite cu eliminarea solidelor ascunse, animatie de solide, si desene umbrite. A fost utilizată de asemenea în analiza solidelor, în principal pentru calculul proprietătilor de masă. [Zei91]

Ideea urmăririi razei a fost folosită în anii 1970 de MAGI (Mathematic Applications Group, Inc.) pentru a genera solide umbrite. Pentru a genera aceste desene este simulat în sens invers procesul fotografic. Prin fiecare pixel al ecranului este trecută o rază de lumină către scenă pentru a identifica suprafata vizibilă. Prima suprafată intersectată de rază, gasită prin urmărirea razei, este cea vizibilă. În punctul de intersectie rază/suprafată este calculată normala la suprafată, si cunoscând pozitia sursei de lumină poate fi calculată luminozitatea pixelului.

Urmărirea razei este considerată o metoda brută de rezolvare a problemelor. Algoritmul de bază este foarte simplu, dar totusi încet. Gradul de utilizare a CPU de către algoritm creste odată cu complexitatea scenei luate în

BUPT

considerare. Algoritmul a fost rafinat pentru a i se îmbunătăti eficienta. Mai mult, algoritmul a fost implementat hardware pentru a fi mai rapid.

3.6. UMBRIREA

Desenele cu ajutorul liniilor, cele mai obisnuite modalităti de comunicare a geometriei pàrtilor mecanice, sunt limitate în posibilitatea lor de a reprezenta forme complicate. Imaginile color umbrite contin informatii asupra formelor care nu pot fi reprezentate doar prin desenarea liniilor. Ele pot să contină si alte caracteristici cum ar fi finisarea suprafetelor sau tipul de material (plastic, metal sau lemn).

Algoritmii pentru imagini umbrite filtrează informatia afisând doar suprafetele vizibile. Multe relatii spatiale nerezolvate prin afisările wire-frame devin clare prin această metodă. Imaginile umbrite sunt mai usor de interpretat, deoarece obiectele seamănă mai mult cu cele reale. Ele prezintă si probleme de vedere care nu apar in reprezentarile wire-frame. Obiecte de interes pot fi ascunse sau partial ascunse, caz in care diferite imagini umbrite pot fi obtinute din diferite puncte de vedere. Tehnici binecunoscute, cum ar fi imagini umbrite peste wireframe, transparenta si sectiuni pot fi utilizate la rezolvarea acestor probleme.

Una din cele mai provocatoare probleme in grafica pe calculator este generarea imaginilor realistice. Cererea pentru imagini umbrite a inceput in primii ani ai deceniului opt, cind preturile memoriei au scăzut suficient de mult pentru a face costul tehnologiei raster atractiv. In umbrirea unei scene (interpretarea unei scene) se pleacă de la rezolvarea problemei de eliminare a suprafetelor ascunse pentru determinarea obiectelor sau portiunilor de obiecte care sunt vizibile in scena. Odată găsite suprafetele vizibile, ele trebuie descompuse in pixeli si umbrite corect. Acest proces trebuie să ia in considerare pozitia si culoarea surselor de lumină si pozitia, orientarea si proprietătile obiectelor vizibile [Zei91].

BUPT

CAPITOLUL 4

ALGORITMI UTILIZATI ÎN MODELAREA SOLIDELOR PRIN OPERATII BOOLEENE

4.1. ALGORITM PENTRU CLASIFICAREA ELEMENTELOR UNEI MULTIMI ÎN CAZUL INTERSECTIEI EI CU UN SOLID

În fiecare din problemele de intersectie punct/solid, linie/solid sau solid/solid, se dau două multimi de puncte: o multime de referintă S si o multime candidat X. Multimea de referintă este de obicei solidul dat al cărui interior si frontieră sunt iS si respectiv bS. Exteriorul lui S este complementul său cS. Multimea candidat este entitatea geometrică care trebuie clasificată relativ la S. Procesul prin care diferite părti ale lui X (puncte, segmente de dreaptă sau portiuni de solid) sunt atribuite lui iS, bS si/sau cS se numeste clasificarea elementelor multimii.

Functia de clasificare a elementelor multimii, notată $M[X, S]$, furnizează o abordare unitară pentru studiul comportării multimii candidat X relativ la multimea de referintă S. Functia este definită cu relatia:

$$M[X, S] = (X \text{ in } S, X \text{ on } S, X \text{ out } S) \qquad (4.1)$$

Această ecuatie implică faptul că intrarea functiei o reprezintă două multimi X si S si că iesirea o reprezintă clasificarea lui X relativ la S prin in, on

BUPT

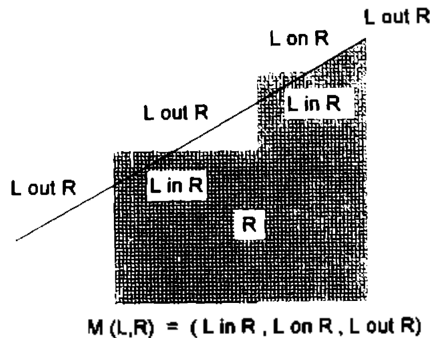


Figura 4.1.Clasificarea elementelor multimii în cazul linie/poligon

sau out S. Figura 4.1 arată un exemplu de clasificare a unei portiuni de linie L relativ la poligonul R.

Implementarea functiei de clasificare date de ecuatia (4.1) depinde în mare masură de reprezentările lui X si S si de structurile lor de date. Considerând problema clasificării linie/poligon din figura 4.2 când poligonul (solidul de referintă) este memorat prin reprezentarea prin frontiere, iar linia L, pentru simplificare nu se suprapune pe nici o portiune cu muchiile poligonului, algoritmul clasificării ar fi urmatorul:

1. Folosind o rutină pentru intersectia linie/muchie, se găsesc punctele de intersectie cu frontiera P_1si P_2.

2. Se sortează intersectiile cu frontiera după directia lui L si să presupunem că lista sortată ar fi (P_0, P_1, P_2 , P_3).

3. Se clasifică L relativ la R. În acest exemplu se stie că intersectiile impare (cum este P_1) marchează segmentele "in" si cele pare (cum este P_2)

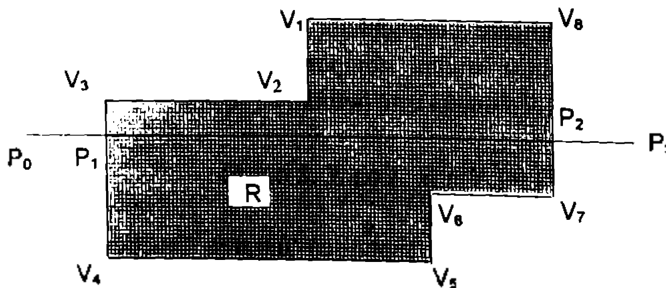


Figura 4.2.Clasificarea linie/poligon pentru B-rep

BUPT

marchează segmentele "out". Astfel, clasificarea lui L relativ la R devine

$$[P_0, P_1] \subset L \text{ out } R$$
$$[P_1, P_2] \subset L \text{ in } R$$
$$[P_2, P_3] \subset L \text{ out } R$$

Dacă linia L contine o muchie a poligonului, criteriul de clasificare de mai sus nu mai este valabil, si trebuie folosit altul.

Algoritmii pentru clasificarea multimilor candidate relativ la solide tridimensionale pot urma pasi similari celor de mai sus, dar cu detalii mai elaborate [Kri87] [Zei91].

4.2. ALGORITMI DE VIZUALIZARE

4.2.1. Testul MINIMAX

Acest test (numit si test de suprapunere sau test de bounding-box) verifică dacă două poligoane se pot suprapune sau nu. Testul furnizează o metodă rapidă pentru a determina dacă două poligoane se suprapun sau nu. Se încadrează fiecare poligon cu un dreptunghi (bounding-box BB) căutându-i extremele (minimul si maximul coordonatelor x si y) si se verifică intersectia oricăror două astfel de dreptunghiuri atât pe directia X cât si pe directia Y. Dacă două dreptunghiuri nu se intersectează, poligoanele corespunzătoare lor nu se suprapun (Fig.4.3). În astfel de cazuri, nu mai sunt necesare altfel de teste asupra muchiilor poligoanelor.

Dacă testul MINMAX esuează (două dreptunghiuri se intersectează), cele două poligoane pot să se suprapună sau nu. Atunci, fiecare muchie a unui poligon se compară cu toate muchiile celuilalt poligon pentru a determina intersectiile. Pentru a mări viteza procesului, se poate mai întâi aplica testul MINMAX asupra fiecărei perechi de muchii.

Testul MINMAX poate fi aplicat în directia Z pentru a verifica dacă există suprapuneri pe această directie. Partea cea mai critică a testului, care consumă cel mai mult timp, o reprezintă găsirea punctelor extreme. De obicei, acestea se obtin parcurgând lista coordonatelor vertexilor fiecărui poligon si înregistrând valoarea cea mai mică si cea mai mare pentru fiecare coordonată [Zei91].

BUPT

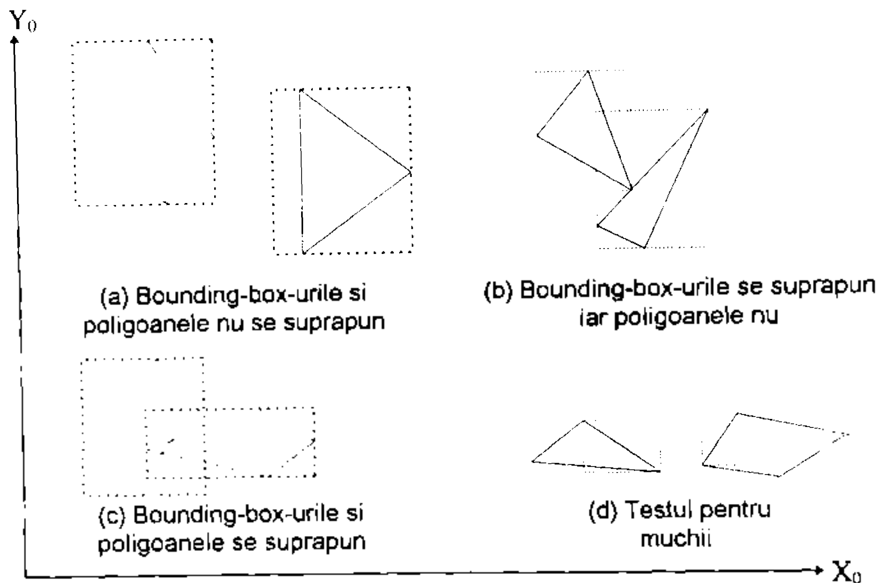


Figura 4.3.Testul MINIMAX pentru poligoane si muchii

4.2.2. Testul de interior

Unii algoritmi de linii ascunse trebuie să cunoască dacă un anumit punct este interior unui poligon. Există trei metode pentru a stabili dacă un punct se găseste în interiorul sau exteriorul unui poligon. Pentru un poligon convex, se pot substitui coordonatele x_v si y_v în ecuatia liniei fiecărei muchii. Dacă toate substitutiile dau acelasi semn, punctul se găseste de aceeasi parte a fiecărei muchii si deci este interior. Testul necesită ca semnele coeficientilor ecuatiilor liniilor să fie corect alese [Sav94] [Mu194] [Mu294][Mu294].

Pentru poligoane concave, pot fi folosite alte două metode. În prima metodă, se trasează o semidreaptă cu originea în punctul de testat si se intersectează cu toate muchiile poligonului. Dacă numărul punctelor de intersectie este par, punctul este exterior poligonului. Dacă numărul este impar, punctul este interior. Dacă vreuna din muchiile poligonului se găseste pe semidreaptă, cazul trebuie tratat în mod deosebit pentru a garanta corectitudinea rezultatului (Fig.4.4).

BUPT

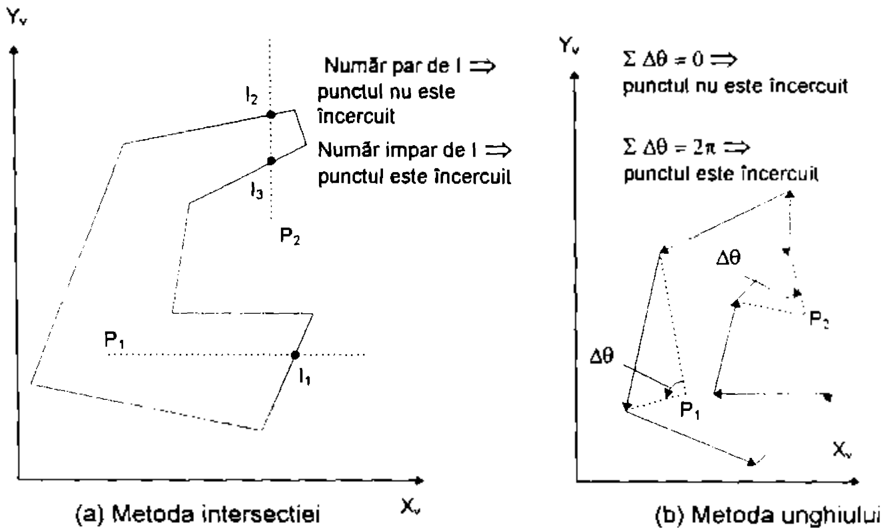


Figura 4.4.Testul de interior pentru poligoane concave

A doua metodă pentru poligoane concave calculează suma unghiurilor subântinse de fiecare muchie orientată, asa cum este văzută din punctul considerat. Dacă suma este zero, punctul este exterior poligonului. Dacă suma este π sau $-\pi$ rad, punctul este în interior. Semnul minus indică dacă vertexii poligonului sunt parcursi în sens orar în loc de sens antiorar [Zei91].

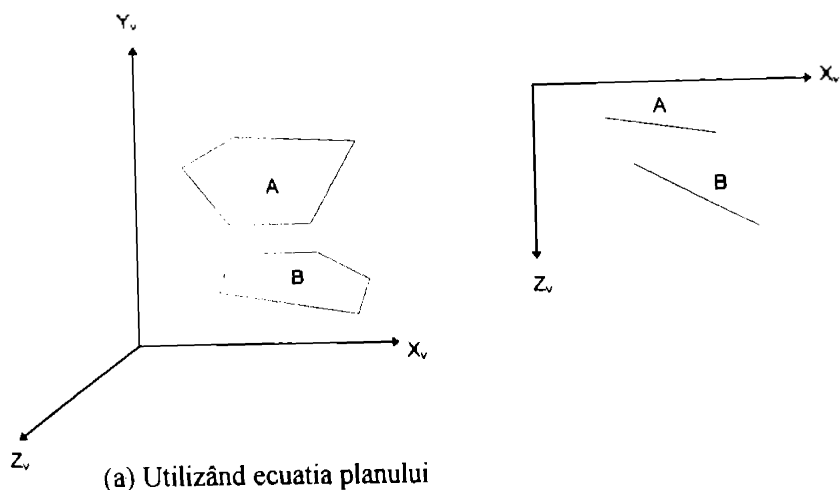
4.2.3. Testul de adâncime

Acest test (numit si test al suprafetelor) furnizează o metodă eficientă pentru implementarea comparatiei de adâncime. Fig.4.5a arată că fata B ascunde o parte a fetei A. În acest caz este folosită ecuatia planului:

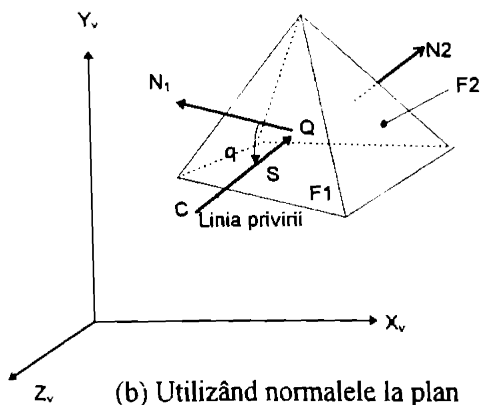
$$ax_v + by_v + cz_v + d = 0$$

Dacă un punct dat (x_v, y_v, z_v) nu este continut în plan, semnul membrului stâng al ecuatiei de mai sus este pozitiv dacă punctul se găseste de o parte a planului si negativ dacă se gaseste de cealaltă parte a planului. Coeficientii

BUPT



(a) Utilizând ecuatia planului



(b) Utilizând normalele la plan

Figura 4.5.Testarea suprafetelor

ecuatiei a, b, c si d pot fi luati astfel încât o valoare pozitivă să indice că punctul de găseste în afara planului. Deci testul foloseste ecuatia planului pentru a stabili pozitia relativă a unui punct dat fată de plan. De asemenea, ecuatia planului poate fi folosită pentru a calcula adâncimea z_v a unui punct dat (x_v, y_v). În acest fel se poate stabili care punct din două este mai apropiat de ochi.

O altă utilizare importantă a ecuatiei planului în algoritmii de linii ascunse este obtinută prin utilizarea vectorului normal la plan. Cu ajutorul lui se poate decide vizibilitatea unei fete. Ideea de bază a testului constă în faptul că fetele ale

BUPT

căror normale sunt îndreptate spre ochi sunt vizibile, iar celelalte sunt invizibile. Acest test este implementat prin calculul produsului scalar dintre vectorul normală N si vectorul rază vizuală S (Fig.4.5.b):

$$N \cdot S = |N|\,|S| \cos a$$

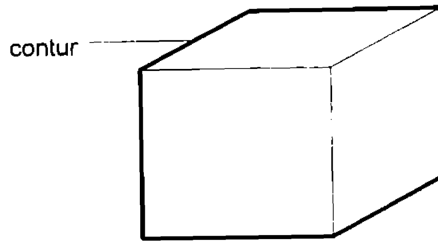
Dacă produsul scalar este pozitiv, înseamnă că normala este îndreptată spre ochi, deci componenta ei pe directia razei vizuale este îndreptată în sensul pozitiv al axei Z_v a sistemului de coordonate al reprezentării. Acest test de suprafată nu poate rezolva singur problema liniilor ascunse decât în cazul unui singur poliedru convex. Testul poate să esueze chiar în cazul poliedrelor convexe dacă sunt prezente mai multe în scenă.

4.2.4. Calculul siluetelor

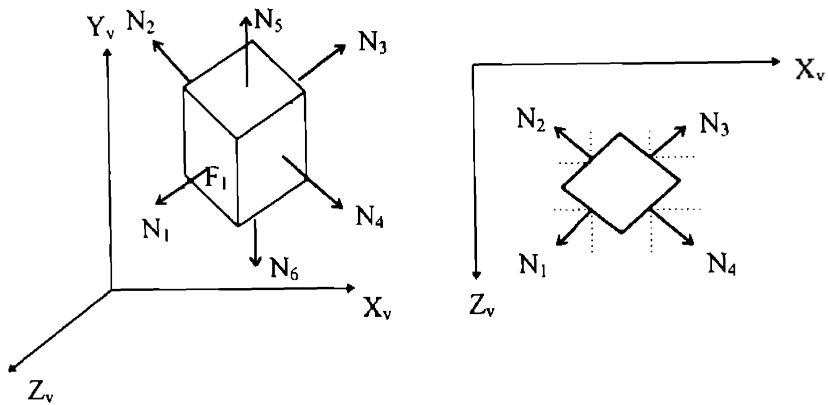
Se numesc muchii de siluetă (sau siluete) acele muchii care separă fetele vizibile de cele invizibile ale unui obiect, relativ la o anumită directie din care se priveste. Semnele componentelor pe directia Z_v a vectorilor normali la fetele obiectului pot fi folosite la determinarea siluetei. O muchie care face parte din siluetă se caracterizează prin faptul că este intersectia dintre o fată vizibilă si una invizibilă. O muchie care este intersectia dintre două fete vizibile este vizibilă dar nu face parte din siluetă. Intersectia dintre două fete invizibile produce o muchie invizibilă. În Fig.4.6 este prezentată silueta unui cub.

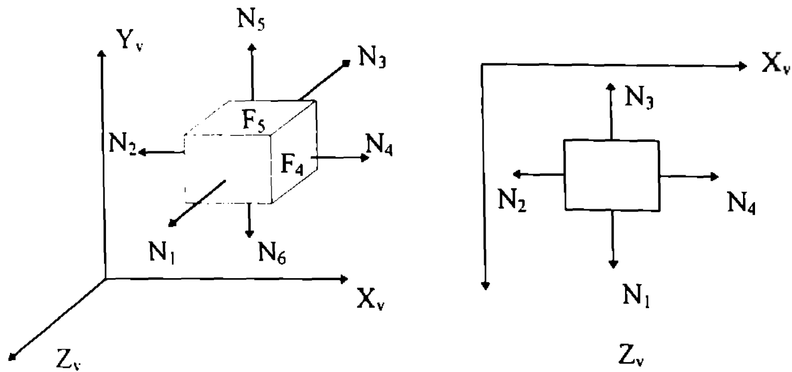
Pentru a stabili silueta unui corp se calculează componentele pe axa Z_v a fiecărei normale. Dacă o normală nu are componentă pe această axă, este nevoie de informatie suplimentară pentru a calcula silueta. O fată a cărei normală nu are componentă pe Z_v este paralelă cu aceasta axă si cu X_v sau Y_v si perpendiculară pe cealaltă axă. De aceea normala este paralelă cu una din axele sistemului de reprezentare. Dacă această normală este îndreptată în sensul pozitiv al axei, fata este vizibilă [Zei91].

BUPT



(a) Muchii de contur





(b) Determinarea muchiilor de contur

Figura 4.6.Muchiile siluetei unui obiect poliedral

BUPT

4.2.5. Intersectiile muchiilor

Nici una din tehnicile de vizibilitate prezentate nu poate determina care părti ale muchiilor partial ascunse sunt vizibile si care nu. Pentru a sti acest lucru, algoritmii de linii ascunse calculează mai întâi intersectiile muchiilor în 2D, adică în planul $X_v Y_v$ al sistemului de reprezentare. Aceste intersectii sunt folosite pentru determinarea vizibilitătii muchiilor. În Fig.4.7. se consideră muchia AB si fata F. Muchiile fetei sunt parcurse în sens antiorar. Muchia AB se intersectează cu CD în punctul I. Vizibilitatea muchiei AB relativ la fata F poate fi în una din urmatoarele trei situatii: complet vizibilă, punctul I indică disparitia muchiei AB, sau punctul I indică aparitia muchiei AB.

În primul caz, se calculează adâncimile z_v în punctele corespunzătoare lui I, situate pe muchia AB si pe fata F, după care se compară. Pentru a găsi adâncimea z_v a punctului de pe fata F, se înlocuiesc coordonatele x_v si y_v în ecuatia planului în care se găseste fata. Pentru punctul de pe AB, se foloseste ecuatia dreptei pentru a calcula adâncimea. Dacă adâncimea punctului de pe linie este mai mare, AB este complet vizibilă (Fig.4.7a). Dacă adâncimea punctului de pe fată este mai mare, muchia este partial vizibilă. Dacă muchia orientată CD subântinde din A un unghi orientat în sens orar, atunci în I muchia devine invizibilă (Fig.4.7b). În caz contrar muchia devine vizibilă din punctul I (Fig.4.7c). Dacă muchiile fetei sunt date în sens orar si criteriul unghiului se inversează [Kri87] [Zei91].

4.2.6. Compararea segmentelor

Această clasă de tehnici este folosită pentru a rezolva problema suprafetelor ascunse în spatiul imaginii (raster). Pot fi aplicate atât algoritmilor de suprafete cât si celor de solide ascunse. Segmentele ce se compară se obtin prin intersectia liniilor de scanare cu fetele din imagine. Aceste linii sunt aranjate pe ecranul dispozitivului de afisare de sus în jos si de la stânga la dreapta. De aceea, în loc să se calculeze întreaga imagine corectă dintr-o dată, ea poate fi calculată linie cu linie de scanare, adică pe segmente, si apoi segmentele sunt afisate în aceeasi ordine cu liniile de scanare.

Din punct de vedere al calculelor, planul de scanare defineste segmente acolo unde întâlneste fetele imaginii (Fig.4.8). Calculul imaginii corecte pentru o linie de scanare este destul de simplu [Zei91].

BUPT

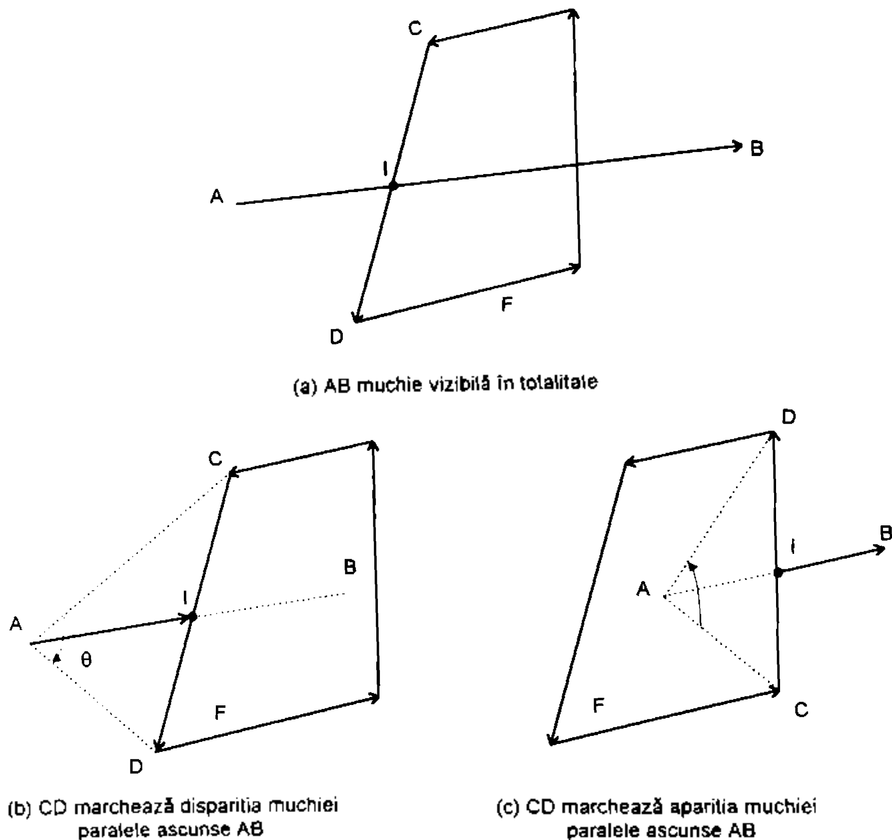


Figura 4.7.Calculul vizibilitătii muchiilor

Compararea segmentelor se face în planul X_vZ_v(Fig.4.8). Linia de scanare este împărtită în portiuni (delimitate de liniile punctate din partea de jos a figurii 4.8). Vizibilitatea este determinată pentru fiecare portiune comparând adâncimile muchiilor aflate pe portiuni. Pentru calculul acestor adâncimi se folosesc ecuatiile planelor. Segmentele cu adâncimea maximă sunt vizibile.

Strategia de a împărti o linie de scanare în portiuni este o caracteristică distinctă a oricărui algoritm de suprafete ascunse. O strategie evidentă constă în împărtirea liniei de scanare în fiecare capăt al fiecărei muchii (liniile A, B, C si D din Fig.4.8). O strategie mai bună constă în alegerea a cât mai putine portiuni. În Fig.4.8, optim ar fi să se împartă linia de scanare prin C doar în două portiuni.

BUPT

4.2.7. Testul de omogenitate

Testul de adâncime din 4.2.3. se referă la compararea adâncimilor punctelor pentru determinarea vizibilitătii. Calculul omogenitătii punctelor este un

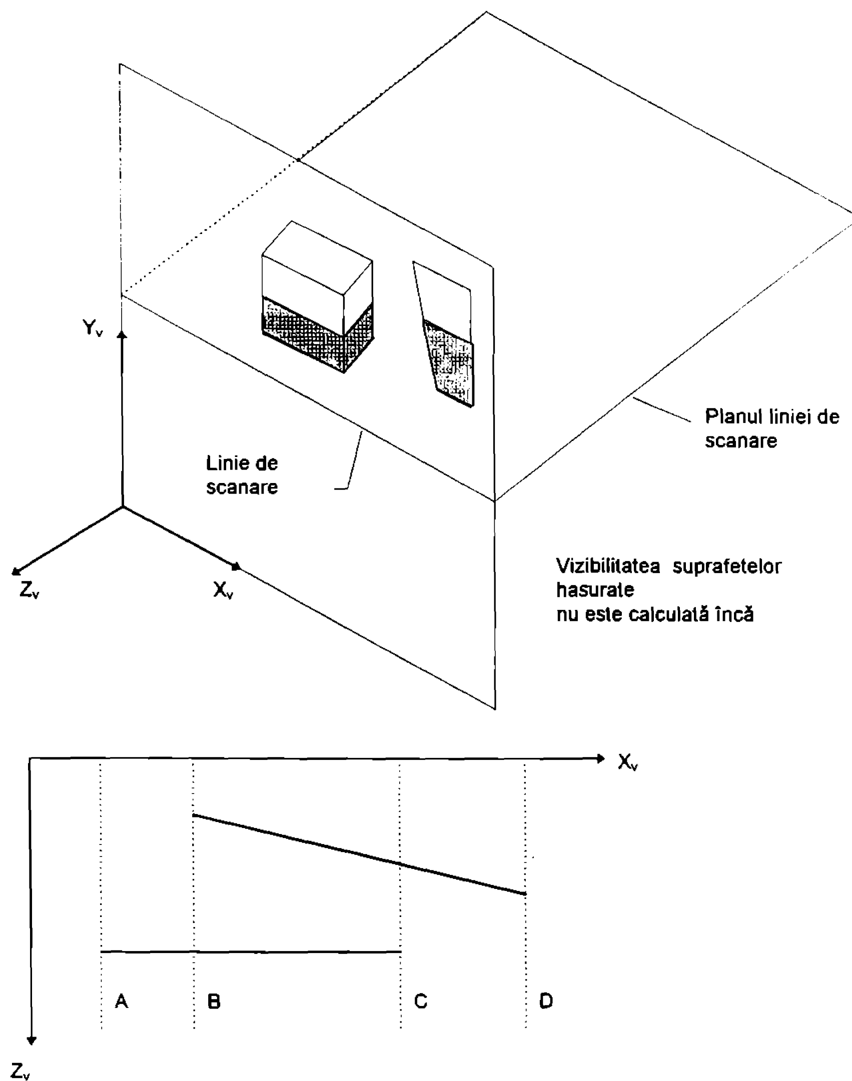


Figura 4.8.Calculul vizibilitătii folosind linii de scanare

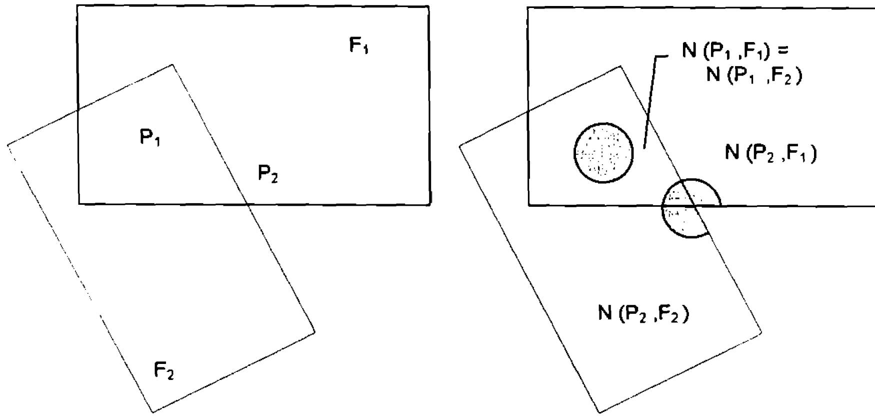
alt test pentru determinarea vizibilitătii. Pentru determinarea omogenitătii este folosită notiunea de vecinătate. Vecinătatea unui punct P, notată V(P), este multimea de puncte situate în interiorul unei sfere (sau al unui cerc în 2D) cu centrul în P.

Pentru calculul omogenitătii, se disting trei tipuri de puncte: omogen vizibile, omogen invizibile si neomogen vizibile. Dacă o vecinătate a unui punct P poate fi bijectiv proiectată într-o vecinătate a proiectiei punctului, atunci vecinătatea lui P este vizibilă sau invizibilă si P este numit omogen vizibil respectiv invizibil. Dacă se notează proiectia lui P cu pr(P), P este omogen vizibil sau invizibil dacă $pr(V(P)) = V(pr(P))$ si neomogen vizibil sau invizibil dacă $pr(V(P)) \neq V(pr(P))$. Folosind acest test, punctele interioare ale scenei sunt omogen vizibile (acoperitoare) sau invizibile (ascunse) si punctele de pe contur sunt neomogen vizibile. În Fig.4.9 este prezentat un exemplu.

S-a notat cu V(P, F) vecinătatea punctului P care apartine fetei F. Este evident că punctele de pe contur (P_2) sunt neomogen vizibile si punctele interioare sunt omogen vizibile (P_1 pe fata F_2) sau invizibile (P_1 pe fata F_1).

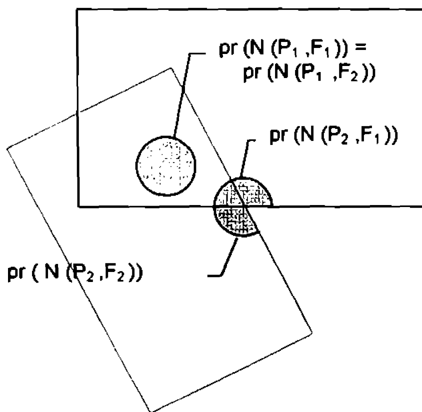
Omogenitatea este importantă atât pentru acoperire cât si pentru ascundere. Nici un punct de lângă punctele omogen vizibile nu trebuie testate si nici punctele omogen invizibile de lângă alte puncte nu trebuie testate deoarece ele sunt omogen invizibile în ambele cazuri. Mai mult, punctele omogen invizibile nu prezintă interes pentru problema vizibilitătii. Punctele omogen vizibile pot fi reprezentate fără să fie nevoie de alte teste relativ la alte puncte. De aceea, determinarea punctelor omogene reduce calculele următoare. Unii algoritmi de arii orientate folosesc acest test pentru a-si îmbunătăti eficienta.

BUPT

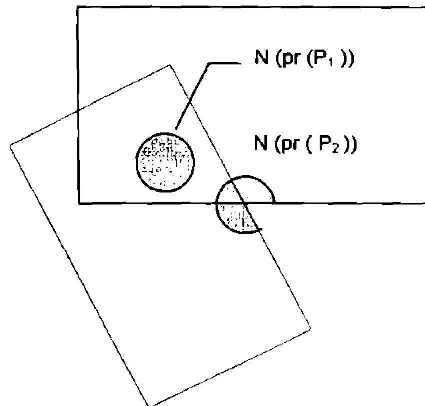


Două fete suprapuse F_1 si F_2

Vecinătăti



Proiectiile vecinătătilor punctelor



Vecinătătile punctelor proiectate

Figura 4.9.Testul de omogenitate

BUPT

CAPITOLUL 5

DIFICULTATI GENERALE IN CADRUL ALGORITMILOR DE MODELARE SI REPREZENTARE A SOLIDELOR

5.1.INTRODUCERE

Există mai multi algoritmi pentru rezolvarea operatiilor booleene pe solide poliedrale, ca si pentru reprezentarea solidelor, dar nu toti iau in considerare in mod satisfăcător problema crucială a erorilor numerice care sunt inerente in calculele cu virgulă flotantă.

Erorile numerice pot fi evitate prin utilizarea unui nou tip de aritmetică, numită "aritmetică lenesă", care efectuează cu precizie doar calculele necesare, fără ca algoritmul să trebuiască să prevadă aceste calcule [Ben94].

Principalele dificultati intilnite in aceste tipuri de algoritmi sunt urmatoarele:

1.Numărul mare de calcule geometrice

Pentru a evita calculele geometrice nenecesare (intersectiile si clasificările), algoritmii caută rapid intr-o listă de perechi de fete sau muchii candidate, lista care include perechile care se intersectează efectiv. De obicei, solidele si fetele au calculate bounding-box-urile si fetele potential intersectabile sunt determinate prin testarea intersectiilor bounding-box-urilor.

BUPT

2.Tratarea cazurilor degenerate

Problemele deosebite in algoritmii geometrici sunt date de elementele care nu se găsesc in pozitii generale. Fiecare caz particular trebuie reprezentat si tratat in mod special, ceea ce conduce la structuri de date si algoritmi complexi. Mai mult, trebuie să se tină seama de faptul că degenerările conduc adesea la aparitia inconsistentelor [Fla87][Hof87].

3.Tratarea erorilor numerice

O primă incercare in eliminarea erorilor constă in utilizarea unor valori empirice pentru precizia de calcul "epsilon". Această metodă însă nu poate garanta intotdeauna rezultate consistente.

O alta abordare a problemei eliminarii erorilor o reprezinta analiza intervalelor. Acest domeniu al matematicii aplicate este utilizat si in modelarea geometrică si in cadrul algoritmilor ce stau la baza reprezentării solidelor [Sny92].

5.2. REDUCEREA NUMARULUI DE CALCULE

In cadrul algoritmilor de modelare geometrică prin operatii booleene sunt necesare o multime de calcule care au drept scop determinarea segmentelor de intersectie dintre fetele solidelor care intervin ca operanzi. Indiferent de operatia care trebuie efectuată (reuniune, intersectie sau diferentă) trebuie calculate punctele de intersectie dintre muchiile unui solid si fetele celuilalt si pe baza lor se vor stabili segmentele de intersectie. Multe din aceste calcule s-ar putea să fie efectuate degeaba deoarece duc la concluzia inexistentei punctului de intersectie. De aceea, o reducere a numărului de intersectii se poate face pe baza testului de bounding-box aplicat la nivelul solidelor, al fetelor si chiar al muchiilor.

Abordarea directă a înlăturării liniilor/suprafetelor/solidelor ascunse cere o cantitate mare de timp de calcul. Pozitia fiecărei linii sau suprafete trebuie comparată cu toate celelalte, lucru care conduce la o explozie combinatorială ce are la bază o lege de variatie proportională cu pătratul numărului de elemente de comparat. Din acest motiv s-au efectuat studii extensive asupra algoritmilor existenti, încercându-se stabilirea principiilor generale comune acestor algoritmi. Cele mai importante au fost identificate a fi coerenta si sortarea. Obiectele si modelele lor geometrice sunt mai mult decât un set de discontinuităti aleatoare. Ele au consistentă atât în aria imaginii cât si în timp, de la un cadru la următorul.

BUPT

Astfel, folosind coerenta între diferite cadre se poate îmbunătăti eficienta algoritmilor de vizualizare.

Similar, toti algoritmii de ascundere sortează sau caută prin colectiile de suprafete, muchii sau obiecte în functie de diferite criterii, pentru a găsi elementele vizibile pe care să le reprezinte. De aceea, o tehnică de sortare eficientă reprezintă cheia unui bun algoritm de eliminare a liniilor/suprafetelor/solidelor ascunse [Dog88].

Eficienta unui **algoritm de vizibilitate** depinde de alegerea multimilor H si C, ale multimilor de muchii candidat ascunse si acoperitoare. Algoritmii de înlăturare a liniilor ascunse pot folosi una din următoarele trei abordări: abordarea orientată pe muchii, abordarea orientată pe contururi sau abordarea orientată pe arii. Pentru a examina eficienta fiecărei abordări, fie E si S numărul de muchii, respectiv contururi, ale unui desen. Abordarea orientată pe muchii stă la baza majoritătii algoritmilor pentru linii ascunse. Strategia de bază stă în calculul explicit al segmentelor vizibile ale tuturor muchiilor individuale. Calculul de vizibilitate constă din testarea tuturor muchiilor relativ la toate suprafetele. Ca urmare, H = E si C = E. Astfel sunt necesare E*E teste de vizibilitate. Această abordare este ineficientă deoarece verifică fiecare muchie cu toate celelalte, indiferent dacă se intersectează sau nu; nu recunoaste informatiile structurale ale unui desen. Sortarea tuturor muchiilor îmbunătăteste eficienta acestui algoritm.

În abordarea orientată pe contururi sunt calculate mai întâi muchiile contururilor. Calculul de vizibilitate constă în testarea tuturor muchiilor relativ doar la muchiile de contururi. Astfel, toate muchiile sunt candidate pentru multimea H, dar doar contururile sunt candidate pentru multimea C, adică H = E si C = S. Astfel sunt necesare E*S teste de vizibilitate. Această abordare este mai eficientă decât prima. Totusi, unele muchii (după sortare) vor fi totusi testate relativ la unele contururi cu care nu se intersectează. Mai mult, sunt calculate fără a fi necesar intersectiile, relativ la muchii de contururi, ale unor muchii complet ascunse. Totusi, sunt complet evitate testele muchiilor complet vizibile între ele.

Abordarea orientată pe arii are ca scop recunoasterea ariilor vizibile ale unui desen. Această abordare determină muchiile contururilor si le conectează pentru a forma poligoane închise (arii). Conectarea poate fi realizată în S*S pasi. Calculele de vizibilitate încep cu testarea tuturor muchiilor de contur între ele. Astfel, H = S si C = S, si sunt necesare S*S teste. Această abordare este mai eficientă decât cea orientată pe contururi. Sortarea este redusă doar la muchiile de contur. Doar contururile sunt testate relativ la contururi cu care nu se

BUPT

intersectează. Este evitat calculul punctelor de intersectie al muchiilor complet vizibile, respectiv complet ascunse.

Algoritmul orientat pe arii este mai eficient decât **algoritmul cu priorităti** deoarece presupune foarte rar o intersectie muchie / fată care să nu fie necesară.

În **algoritmul Warnock**, dacă divizarea ferestrei originale este guvernată de complexitatea scenei din figura 3.4, divizarea oricărei ferestre în patru ferestre egale face ca algoritmul să fie ineficient. Mai eficientă ar fi împărtirea unei ferestre în functie de complexitatea scenei din fereastră, adică divizarea unei ferestre în patru subferestre inegale.

Algoritmul cu urmărirea razei este foarte încet si gradul de utilizare a memoriei si a CPU este direct proportional cu complexitatea scenei, adică cu numărul de primitive din solid. În practică, nu atât gradul de utilizare a memoriei este o problemă, cât viteza de lucru a algoritmului.

Prin folosirea de bounding-box-uri (BB) minime în jurul solidelor din arborele CSG, căutarea exhaustivă a intersectiilor rază/solid devine o căutare binară eficientă. Aceste bounding-box-uri permit algoritmului să detecteze cazurile evidente de disjunctie dintre rază si solid. Arborele CSG poate fi văzut ca o reprezentare ierarhică a spatiului ocupat de solid. Astfel, nodurile arborelui vor avea închideri ("enclosures boxes") care sunt pozitionate în spatiu. Atunci testele rapide de intersectie rază/închidere ghidează căutarea în ierarhie. Când testul esuează într-un nod intermediar din arbore, raza poate fi în mod sigur clasificată ca fiind afară din compus; astfel, nu mai este necesară parcurgerea recursivă în jos a subarborilor solidului.

Algoritmul cu urmărirea razei are câteva avantaje. El elimină găsirea, parametrizarea, clasificarea si memorarea muchiilor curbe formate de intersectia suprafetelor. Determinarea contururilor suprafetelor curbe este un produs secundar si poate fi găsită ori de câte ori vederea se schimbă.

Principalul dezavantaj ale algoritmului este viteza. Aceasta este deosebit de importantă pentru a afisa desene umbrite într-un mediu interactiv. Dacă utilizatorul crează un arbore echilibrat al solidului din scenă eficienta algoritmului creste. Coerenta suprafetelor vizibile (este mai degrabă probabil ca suprafetele vizibile în doi pixeli vecini să fie aceleasi, decât diferite) poate mări de asemenea viteza algoritmului. În plus, muchiile solidului sunt doar căutate pentru a genera desenele. De aceea, urmărirea razei trebuie să se concentreze în jurul muchiilor si nu în regiunile deschise. Acest lucru poate fi implementat esantionând în mod împrăstiat ecranul cu raze si localizând apoi (când razele vecine identifică suprafete vizibile diferite) muchiile prin căutări binare. Pasul de esantionare este

BUPT

controlat de utilizator. Pe măsură ce esantionarea devine mai împrăstiată, cresc sansele de a nu fi luate în considerare muchii sau aschii ale solidului [Dog88].

Mărirea gradului realismului unui model prin dezvoltarea imaginilor umbrite de exemplu, duce la cresterea complexitătii lui. Procesul de creare si întretinere a unui astfel de model devine de asemenea complex. De exemplu, generarea unei imagini umbrite de înaltă rezolutie a unui model complex cu diferite tipuri de iluminări, poate dura un timp CPU de câteva minute si poate reduce performantele sistemului CAD/CAM (încetineste activitatea altor utilizatori curenti). Memorarea si/sau regăsirea modelelor cu aceste tipuri de imagini umbrite sunt de obicei lente. De altfel, generarea imaginilor în timp real, cere calculatoare puternice cu cât mai multi algoritmi de umbrire încorporati în hardware. Pentru a îmbunătăti performantele acestor algoritmi, anumite sisteme CAD/CAM sunt furnizate cu microprograme de umbrire.

5.3. CAZURI DEGENERATE

Pentru a evita generarea obiectelor neregulate in procesul de modelare, se recomanda neefectuarea operatiei booleene in cazul in care o muchie noua apartine la mai mult de doua fete distincte (figura 5.1) precum si considerarea suprapunerilor de tip muchie-muchie, muchie-fata sau fata-fata ca si cum entitatile geometrice se ating una cu alta fara a avea puncte in comun (figura 5.2) [Fla87][Hof87].

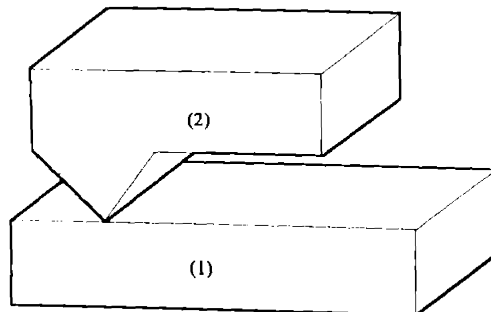


Fig.5.1. Reuniune interzisă

BUPT

In aceasta abordare, obiectele rezultate nu mai trebuie sa fie regularizate, dar trebuie acordata atentie noilor frontiere utilizind regulile exacte de formare a lor (figura 5.3).

Deoarece pentru constructia solidelor rezultate este necesară cunoasterea tipului muchiilor suprapuse, stabilirea acestora trebuie să aibă loc in timpul procesului de intersectie. Muchiile ale căror poligoane invecinate formează diedre ascutitunghice vor fi de tip "vale" (*Valley*), iar cele corespunzatoare diedrelor obtuzunghice vor fi de tip "deal" (*Hill*). Cunoscind capetele muchiei si normalele

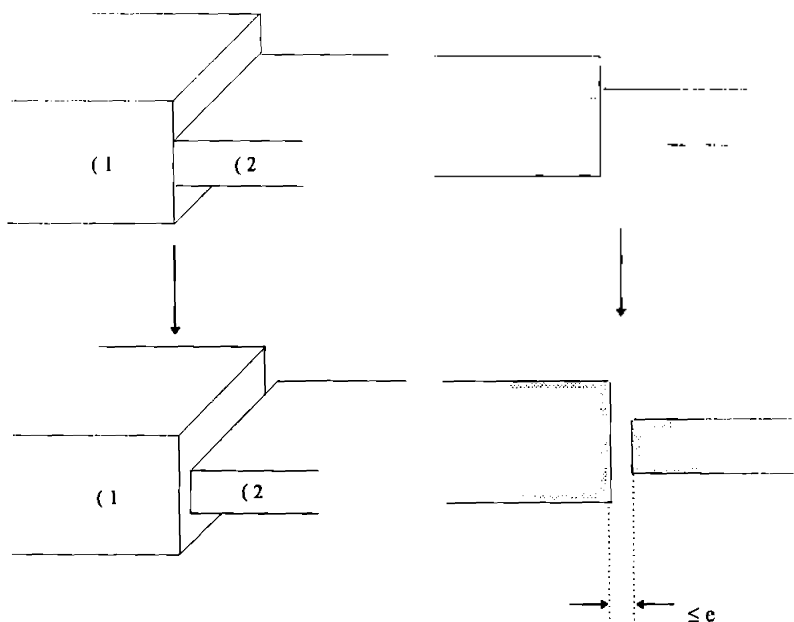


Fig.5.2. Frontiere fără puncte comune

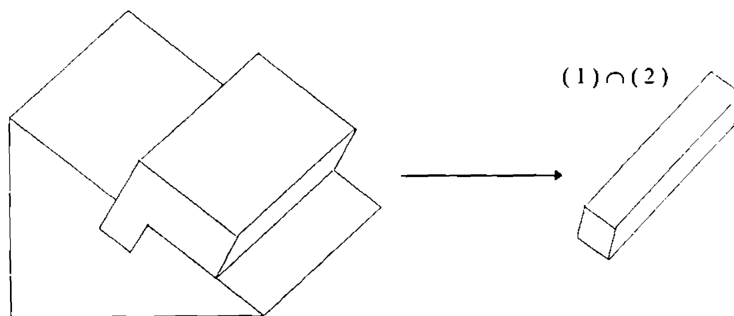


Figura 5.3. Intersectie fără necesitatea regularizării

BUPT

planelor in care se gasesc fetele alăturate ei, prin calcul vectorial (produsul mixt intre normala poligonului din stinga si dreapta muchiei si vectorul muchiei respective) se poate stabili tipul muchiei (figura 5.4).

Diferitele situatii particulare care pot sa apara sunt prezentate in figura 5.5. In astfel de cazuri in care solidele sunt prost pozitionate, opeartia booleana este

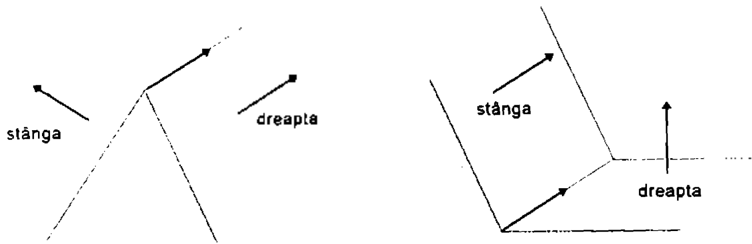


Fig.5.4. Muchii de tip "deal" si de tip "vale"

evitata

Muchii suprapuse pot să apară insă si in situatii in care operatia trebuie efectuată, cum este cazul din figura 5.6. Aici, din multimea muchiilor care apartin doar unui solid si cea a muchiilor care apartin ambelor solide (muchii suprapuse total sau partial) trebuie eliminate acelea care nu vor exista in solidul rezultat (figura 5.7). De exemplu, in cazul reuniunii, trebuie eliminate muchiile din fiecare solid care se găsesc pe frontierele celuilalt.

Atunci cand două muchii a si b se suprapun, trebuie luată decizia asupra segmentelor care se pastreaza, in functie de operatia booleana care se efectuează, de modul in care sunt situate fetele asociate muchiei si de contactul dintre muchii. In acest scop, se disting cinci actiuni diferite legate de suprapunerea a doua muchii, notate A, B, C, C' si D (figura 5.8). Figura 5.9 arata efectele fiecarei actiuni, cand se iau in considerare contactele dintre muchii, notate de la 0 la 6.

Contactele dintre fete se clasifica in patru grupe, de cate treisprezece clase fiecare, conform celor patru posibilitati de combinare a muchiilor: *hill-hill*, *valley-valley*, *hill-valley* si *valley-hill*.

Figura 5.10 prezintă actiunile ce se efectueaza in functie de contactul dintre fete si de operatia booleană de efectuat. **Q**⁻ si **Q+** indica cate normale coincid si respectiv cate sunt opuse. Coloanele corespund celor patru modele diferite **X**, **Y**, **Z** si **T**, dupa cum sunt indicate in partea de jos a figurii.

Simetria dintre operatiile booleene si cele patru posibilităti pentru muchii pot fi observate in figura 5.11. Aceasta simetrie provine din faptul ca intersectia a două multimi se poate obtine din complementul reuniunii complementelor multimilor initiale, iar diferenta mutimilor se obtine din complementul reuniunii dintre complementul primei multimi si a doua multime.

BUPT

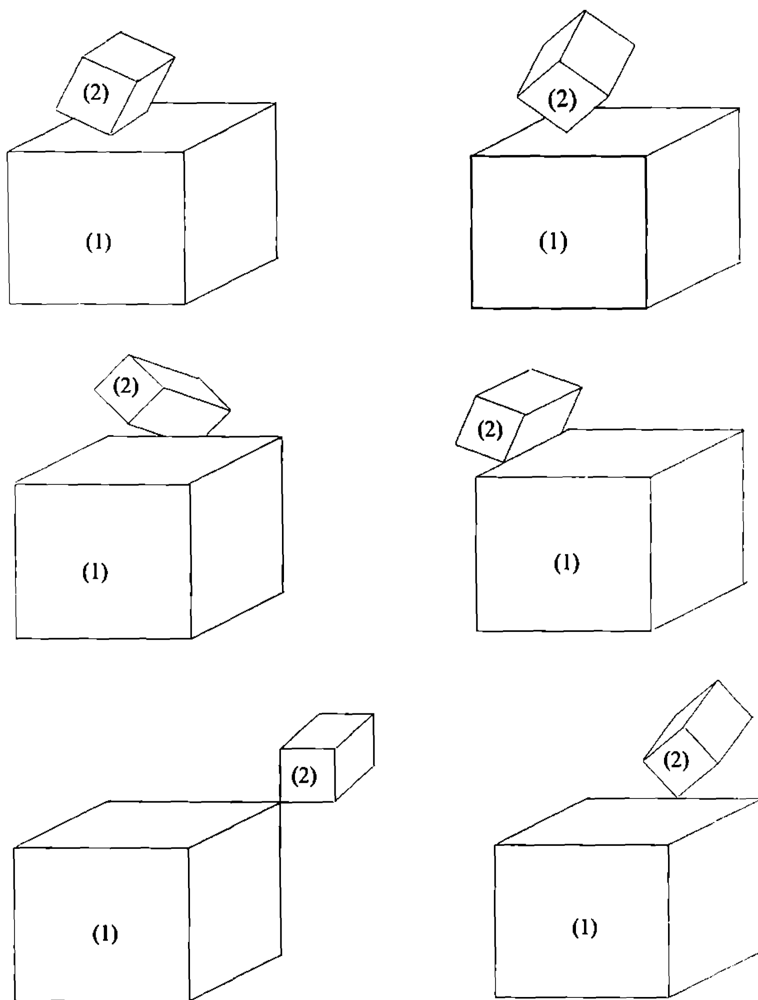


Figura 5.5. Obiecte prost pozitionate

Pentru fiecare grup se pot recunoaste usor cele treisprezece clase mentionate.Este necesar sa se cunoasca valorile lui Q^- si Q^+ si deasemenea care este relatia dintre cei patru vectori nou introdusi.

BUPT

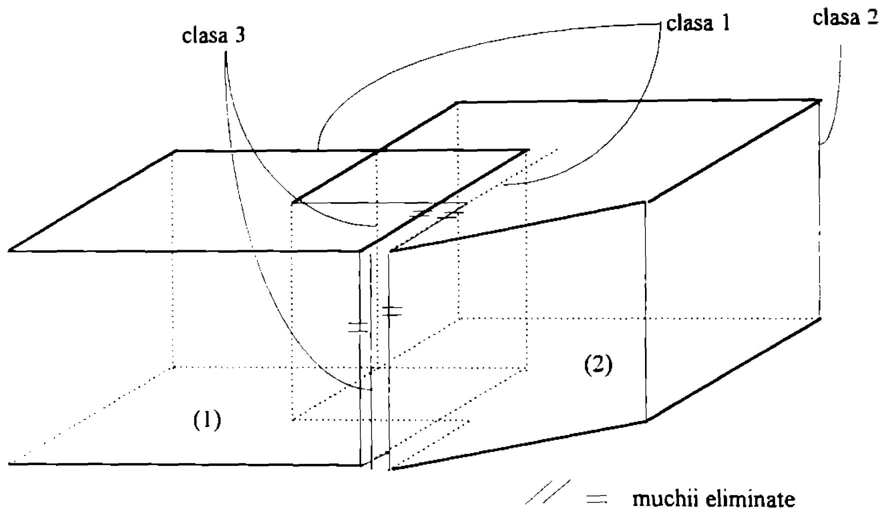


Figura 5.6. Diferite clase de muchii înainte de reuniune

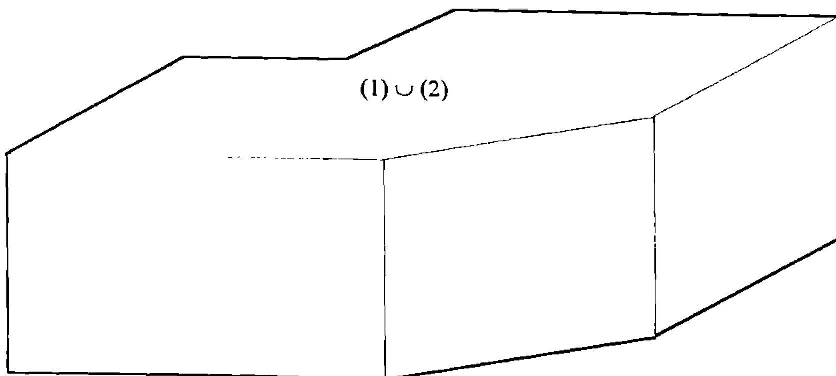


Figura 5.7. Muchiile selectate pentru reuniune

BUPT

Actiuni	Segmente rezultante			
A	a-b	b-a		
B	a-b	a-(a-b)	b-a	
B'	b-a	b-(b-a)	a-b	
C	a-b	a-(a-b)	b-(b-a)	b-a
D	a-b	$a \cap b$	b-a	

Figura 5.8. Actiuni asupra unei perechi de muchii suprapuse

O muchie *hill* sau *valley* are o pereche de vectori $L \cap E$ si $E \cap R$ (vezi figura 5.12). Există patru relatii posibile in cazul unei perechi de muchii, notate (a) - (d) in figura 5.13. Figura 5.14 arată cum se recunosc cele treisprezece clase pentru cele patru cazuri de combinare a muchiilor.

Dacă cele două muchii nu se suprapun perfect, ele trebuie divizate in punctele de intersectie, dacă există [Fla87].

BUPT

Fig.5.9. Efectele fiecărei actiuni indicând clasele obiectelor obtinute

BUPT

Clase	H-H (Q Q')	Actiuni ∪	Actiuni ∩	Actiuni -	V-V (Q Q')	Actiuni ∪	Actiuni ∩	Actiuni -	H-V (Q Q')	Actiuni ∪	Actiuni ∩	Actiuni -	V-H (Q Q')	Actiuni ∪	Actiuni ∩	Actiuni -
1	11	A	A	B	11	A	A	B'	11	B'	B	A	11	B	B'	A
2	20	D	D	A	20	D	D	A	02	A	A	D	02	A	A	D
3	02	C	A	B	02	A	C	B'	20	B'	B	A	20	B	B	C
4	10	C	A	B	10	A	C	B'	01	B'	B	A	01	B	B	C
5	01	C	A	B	01	A	C	B'	10	B'	B	A	10	B	B	C
6	00	C	A	B	00	A	C	B'	00	B'	B	A	00	B	B'	C
7	10	A	D	D	10	D	A	D	01	D	D	D	01	D	D	A
8	00	D	D	D	00	D	D	D	00	D	D	D	00	D	D	D
9	01	D	A	B	01	A	D	B'	10	B'	B	A	10	B	B	C
10	10	B	D	D	10	D	B	A	01	A	D	D	01	D	A	B
11	10	B'	D	A	10	D	B'	D	01	D	A	D	01	A	D	B
12	00	B	D	C	00	D	B	A	00	A	C	D	00	C	A	B
13	00	B	D	A	00	D	B'	C	00	C	A	D	00	A	C	B
	Modele	X	Y	T	Modele	Y	X	Z	Modele	Z	T	Y	Modele	T	Z	X

Fig.5.10. Actiuni de efectuat

BUPT

	H-H	V-V	H-V	V-H
∪	X	Y	Z	T
∩	Y	X	T	Z
—	T	Z	Y	X

Fig. 5.11. Operatii booleene si grupe de contact

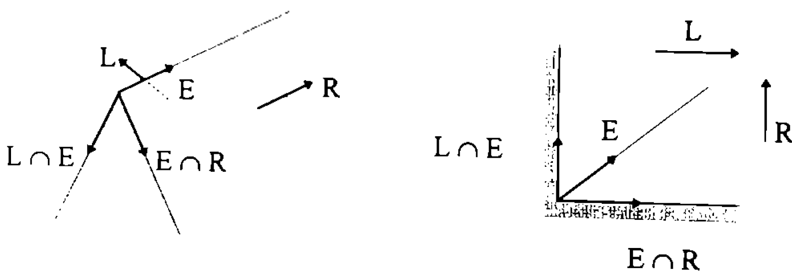


Fig.5.12. Vectorii asociati

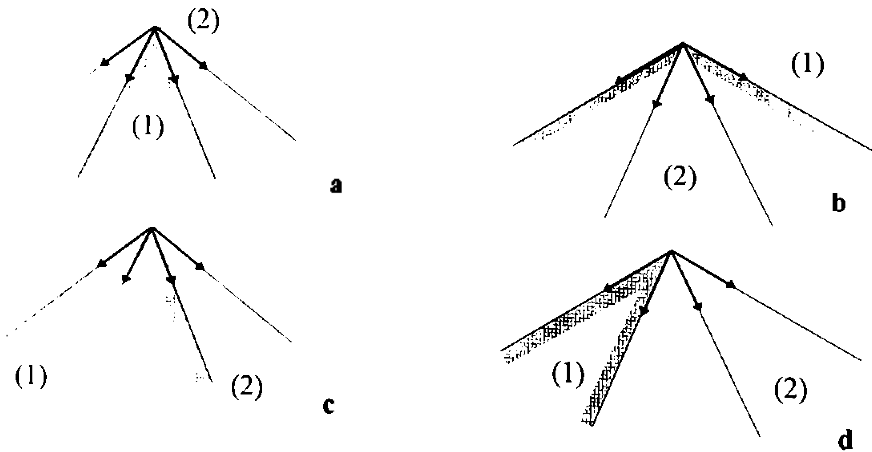


Fig.5.13. Relatii dintre două perechi de vectori asociati

BUPT

H-H sau V-V

1 1	2 0	0 2	1 0				0 1		0 0			
			(d)	(c)	(b)	(a)	(d)	(c)	(d)	(c)	(b)	(a)
1	2	3	4	7	10	11	5	9	6	8	12	13

H-V sau V-H

1 1	0 2	2 0	0 1				1 0		0 0			
			(d)	(c)	(b)	(a)	(d)	(c)	(d)	(c)	(b)	(a)
1	2	3	4	7	10	11	5	9	6	8	12	13

Figura 5.14. Clasificarea celor 13 clase

5.4.TRATAREA ERORILOR NUMERICE

Există o multime de abordări ale problemei eliminării erorilor datorate preciziei cu care lucrează calculatoarele numerice. Cateva din ele sunt prezentate mai jos.

Guibas, Salesin si Stolfi descriu un cadru teoretic care permite mentinerea in sigurantă a intervalelor in care se pot lua deciziile, dar este o problemă stabilirea acestor intervale.

Segal si Sequin impun o separare minimă intre elementele unei perechi de primitive (fete, muchii sau vertexi). Oricare două primitive care se găsesc la o distantă mai mică decît o valoare minimă aleasă, trebuie fie unite ca sa coincida, fie îndepărtate pentru a mentine separarea minimă.

Hoffman, Hopcroft si Karasick impun deasemenea o separare minimă, dar ei recurg la un rationament simbolic suplimentar pentru a se asigura că o decizie nu va contrazice niciodată pe cele anterioare.

Milenkovic acordă deasemenea o prioritate mai mare topologiei. El descrie o implementare verificabilă a unui algoritm de aranjare a liniilor care păstrează o bază de date consistentă din punct de vedere geometric.

Mäntilä si Sulonen asigură consistenta topologică a modelatorului lor de solide prin utilizarea strictă a operatorilor lui Euler. Oricum, acesti operatori nu evită contradictiile dintre datele numerice si topologice.

Sugihara si Iri observă că dacă datele geometrice originale sunt reprezentate cu o precizie finită, configuratia topologică relativă a două elemente geometrice poate fi calculată exact in cadrul preciziei finite. Ei arată cum se poate construi un modelator poliedral fără erori bazat pe primitive triedrale.

BUPT

Karasick, Lieber si Nakman se bazează pe o aritmetică ratională, o utilizare inteligentă a intervalelor (de intregi) si o proiectare atentă pentru implementarea triangularizării Delaunay fără erori.

Benouamer, Michelucci si Peroche se bazează deasemenea pe o aritmetică ratională, dar spre deosebire de abordările anterioare, modul în care sunt evitate erorile numerice este total transparent din punct de vedere al algoritmului. Mai mult, aritmetica ratională utilizată este mult mai putin costisitoare din punctul de vedere al calculelor, decît aritmeticile rationale pure traditionale, deoarece se efectuează doar calculele precise necesare, fără ca algoritmul să trebuiască să presupună aceste calcule particulare [Ben94].

In cazul in care algoritmii sunt implementati folosind o aritmetică ratională pură, executia programului este foarte lentă. Acest lucru se datorează cantitătii mari de calcule precise efectuate, chiar dacă ulterior nu mai sunt folosite. Două ar fi dezavantaje utilizării directe a aritmeticii rationale:

1. Multe date numerice (de exemplu distanta cun semn, determinanti) sunt calculate exact pentru că deciziile geometrice pot fi luate pe baza determinării unui semn (de exemplu testul care stabileste dacă un punct se găseste intr-un plan dat). Oricum, valoarea calculată nu este necesarsă fie cunoscută dacă presupunem că există o modalitate prin care semnul să poată fi obtinut corect.

2. Multe date geometrice (de exemplu coordonate de vertexi si coeficientii ecuatiilor planelor) sunt calculate si memorate in structuri de date, desi la sfirsit sunt eliminate (se ajunge la concluzia că nu sunt necesare pentru obtinerea rezultatului final).

Luind in considerare aceste aspecte, se poate recurge la intirzierea calculelor precise pină in momentul in care ele devin fie inutile, fie indispensabile. Pentru aceasta, fiecare număr va avea două reprezentări: un interval (dat de exemplu prin două numere in virgulă flotantă) care contine valoarea exactă (probabil necunoscută) si definitia simbolică care permite calculul valorii exacte atunci cind este necesar.

O definitie este fie un număr rational pur (reprezentat intr-un mod oarecare), fie o expresie neevaluată care reprezintă suma, produsul, opusul sau inversul unor numere lenese. Mai precis, numerele lenese formează un graf aciclic direct si nu un arbore, căci anumite noduri pot fi distribuite.

Fiecare operatie este efectuată intr-un timp constant. Ea constă din adaugarea unui nou nod in graful aciclic direct care memorează operatorul, pointerii la operanzi si un interval corect care contine rezultatul. Acest interval se

BUPT

obtine din cele ale operanzilor prin reguli simple de compozitie fixate in aritmetica intervalelor [Sny92].

In cele mai multe cazuri, intervalele sunt suficiente pentru a compara numere lenese. Valoarea exactă a unui număr nu este niciodată calculată, cu exceptia cazurilor urmatoare:

(a) cind este nevoie să determinăm semnul unui număr lenes al cărui interval contine zero sau, mai general, cind trebuie să comparăm două numere ale căror intervale se suprapun;

(b) cind este nevoie să se calculeze inversul unui număr lenes al cărui interval contine zero;

(c) cind este nevoie să se calculeze un predecesor din graful aciclic direct al numărului lenes considerat.

O astfel de aritmetică ratională lenesă nu consumă mai mult timp de calculator decit o aritmetică ratională pură. In cel mai rău caz se vor efectua exact toate calculele si in cel mai bun caz nu se va efectua nici un calcul. Se vor efectua numai calculele exacte necesare, fără ca algoritmul să le prevadă. Ca rezultat, evitarea erorilor numerice (care duce la evitarea inconsistentelor) este tratată in mod transparent la un nivel scăzut care este total independent de algoritm.

Abordarea analizei intervalelor pentru rezolvarea problemelor globale poate fi rezumată după cum urmează:

1. Sunt construite functii de incluziune care limitează functiile folosite în ecuatiile si inegalitătile cu restrictii si functia obiectiv (functia care trebuie minimizată).

2. Aceste functii de incluziune sunt folosite într-un algoritm "branch and bound". O regiune initială a spatiului parametrilor este divizată recursiv în regiuni mai mici. Functiile de incluziune sunt folosite pentru a testa dacă o anumită regiune poate fi o solutie pentru problema globală. De exemplu, pentru a testa dacă o regiune X poate include o solutie a ecuatiei $f(x) = 0$ este evaluată o functie de incluziune pentru f peste regiunea X. Dacă granita care rezultă nu contine 0, atunci regiunea X poate fi respinsă. Similar, o granită a unei functii de incluziune pentru functia obiectiv poate respinge o regiune dacă functia obiectiv este prea întinsă pentru a contine un minim global. Divizarea continuă până când toate regiunile sunt fie respinse, fie acceptate ca si regiuni solutie.

BUPT

În plus, astfel de metode de divizare pot fi îmbunătătite prin tehnici locale (de exemplu, metodele Newton si cvasi-Newton pentru probleme cu restrictii, metode de gradient conjugate pentru probleme de minimizare). Încorporarea acestor tehnici poate deseori îmbunătăti simtitor performantele algoritmului solutie.

Abordarea prin intermediul analizei intervalelor prezintă câteva avantaje fată de celelalte abordări ale solutionării problemelor globale. Analiza intervalelor controlează erorile de aproximare care rezultă din efectuarea pe un calculator numeric a unor calcule în virgulă flotantă imprecise. Controlul erorii este realizat prin asigurarea că functiile de incluziune sunt granite valide, chiar dacă aceste granite trebuie reprezentate folosind setul discret de numere în virgulă floatantă al calculatorului. Analiza intervalelor este o tehnică exhaustivă, adică pentru găsirea solutiilor sunt examinate toate regiunile spatiului de intrare al problemei. Abordarea prin analiza intervalelor pemite astfel găsirea minimelor globale, nu doar a unui minim local care poate fi apropiat de minimul global. Mai mult, pot fi găsite toate minimele sau toate solutiile unui sistem cu restrictii, mai degrabă decât o singură solutie [Ben94][Sny92].

BUPT

CAPITOLUL 6

ALGORITMI UTILIZATI ÎN MODELAREA PRIN OPERATII BOOLEENE SI ÎN PROCESUL DE ELIMINARE A LINIILOR SI SUPRAFETELOR ASCUNSE

6.1. ALGORITMI PENTRU TESTELE DE BOUNDING-BOX

Bounding-box-ul unei figuri geometrice poate fi un dreptunghi sau un paralelipiped, după cum prelucrarile geometrice se efectuează în plan sau în spatiul tridimensional [Zei91][MV294][MV394].

Prin **bounding-box-ul unei figuri geometrice plane** se întelege dreptunghiul cu laturile paralele cu axele sistemului de coordonate atasat planului de (proiectie) perspectivă, laturi care trec prin punctele extreme ale figurii geometrice în cauză. În cazul algoritmilor pentru înlăturarea liniilor si suprafetelor ascunse, aceste figuri geometrice pot fi muchii ale poligoanelor ce formează fetele unui corp, contururile unui corp sau întregul corp, toate proiectate în planul de perspectivă.

Determinarea punctelor extreme ale unei figuri geometrice plane presupune utilizarea algoritmilor de aflare a minimului si maximului dintr-un sir de valori reale. Aceste valori sunt coordonatele de pe axa Ox a sistemului de coordonate al

BUPT

planului de perspectivă, si respectiv coordonatele de pe axa Oy a aceluiasi sistem, ale punctelor ce definesc figura geometrică.

O primă categorie de calcule geometrice se refera la testele de suprapunere, în planul de perspectivă, a bounding-box-urilor diferitelor elemente ale corpurilor ce trebuie reprezentate. Plecând de la nivelul cel mai de sus, s-au folosit teste de suprapunere a bounding-box-urilor proiectiilor în planul de perspectivă a corpurilor, a bounding-box-urilor suprafetelor determinate de contururile unui corp, a bounding-box-urilor a două muchii apartinând unui corp sau la două corpuri diferite, precum si situarea unui punct în interiorul bounding-box-ului unui poligon ce defineste o fată a unui corp.

Aceste teste au drept scop evitarea operatiilor de intersectie în planul de perspectivă în cazul în care bounding-box-urile elementelor de intersectat nu au puncte comune. Operatiile de intersectie pentru care se iau în considerare aceste bounding-box-uri sunt intersectiile proiectiilor a două volume, a două contururi, a două muchii, precum si intersectia dintre o semidreaptă si un poligon, necesară în cazul testului de interior efectuat asupra unui punct dat, relativ la proiectia unui poligon dat.

Câstigul obtinut prin utilizarea acestor teste a fost considerabil, timpul de executie a algoritmului reducându-se în mod evident.

Deoarece în aceste operatii de intersectie fiecare element este luat în considerare de mai multe ori, o dată pentru fiecare element dintr-o anumită categorie considerată, si valorile care definesc bounding-box-ul elementului respectiv trebuie utilizate de mai multe ori. Calculul lor repetat, ori de câte ori este necesar, ar duce la cresterea nejustificată a timpului de executie. De aceea s-a preferat efectuarea calculului o singură dată pentru fiecare element ce ar putea intra într-o operatie de intersectie, si memorarea valorilor ce definesc bounding-box-ul în structura atasată acestui element. Pierderea ce rezultă prin ocuparea memoriei alocate pentru fiecare bounding-box este compensată de cistigul mare obtinut prin reducerea timpului de executie a algoritmului în ansamblu.

În algoritmul pentru eliminarea suprafetelor ascunse, testele de bounding-box efectuează comparatiile între valorile extreme ale elementelor ce urmează a fi intersectate. Dacă este necesară stabilirea corpurilor ale căror proiectii se pot suprapune, testul se face la nivelul bounding-box-urilor lor. În cazul în care se ajunge în etapa de stabilire a intersectiilor dintre două contururi, testul se face la nivelul bounding-box-urilor acestora. Ultimul nivel la care se utilizează testul este cel al bounding-box-urilor muchiilor, pentru a evita eventualele calcule de

BUPT

intersectie, precum si la nivelul algoritmului prin care se stabileste dacă un punct ar putea să se gasească în interiorul unui poligon.

Bounding-box-ul unei figuri geometrice cu trei dimensiuni este folosit în algoritmii corespunzători modelării solidelor prin operatii booleene.

În acest caz, un rol important în reducerea timpilor de calcul îl au testele ce pot fi efectuate pentru reducerea numărului de operatii de intersectie. Aceste teste pot fi făcute la nivelul solidelor, al poligoanelor sau chiar al muchiilor care formează solidele.

Deoarece intersectiile se efectuează în spatiul tridimensional, determinarea bounding-box-ului unui element spatial presupune determinarea punctului având drept coordonate minimele coordonatelor pe cele trei directii ale sistemului de axe al modelului solidului, precum si a punctului cu coordonate maxime. Bounding-box-ul unui element geometric spatial va fi paralelipipedul cu fetele paralele cu planele sistemului de coordonate ce trece prin cele două puncte extreme.

Testul de bounding-box se face prin comparatii între coordonatele punctelor extreme. Intersectia a două bounding-box-uri este posibilă dacă cel putin una din coordonatele unui punct extrem al primului bounding-box se găseste în intervalul definit de aceeasi coordonată a punctelor extreme ale celuilalt bounding-box.

Deoarece efectuarea acestor teste este necesară în diferite momente ale algoritmului operatiilor booleene, fiecare element geometric va păstra în structura lui de date informatiile referitoare la bounding-box-ul propriu. În acest fel comparatiile pentru stabilirea coordonatelor extreme se efectuează o singură dată, dar prin memorarea lor, pot fi folosite ori de câte ori este cazul.

6.2. ALGORITM PENTRU AFLAREA UNUI PUNCT SITUAT PE UN SEGMENT DAT CÂND SE CUNOASTE RAPORTUL ÎN CARE PUNCTUL ÎMPARTE SEGMENTUL

Algoritmul este util în situatia în care se doreste aflarea punctului de intersectie dintre două segmente definite în plan sau în spatiu [MV294][MV394].

Dându-se segmentul cu capetele P_1 (x_1 , y_1) si P_2 (x_2 , y_2), se cere găsirea punctului P (x , y) situat pe segmentul dat, dacă se cunoaste raportul în care punctul împarte segmentul. Acest raport este subunitar si poate fi interpretat

BUPT

ca distanta de la punct la un capăt al segmentului raportată la lungimea segmentului.

Determinarea punctului P presupune determinarea coordonatelor x si y ale punctului. Ele pot fi stabilite bazat pe faptul că raportul în care punctul împarte segmentul este acelasi cu raportul în care proiectia punctului pe axele de coordonate împarte proiectiile segmentului pe aceste axe.

Dacă

$$r = \frac{P_1P}{P_1P_2}$$

P_1 —— P —————— P_2

atunci

$$r = \frac{x - x_1}{x_2 - x_1} \quad \text{si} \quad r = \frac{y - y_1}{y_2 - y_1}$$

De aici rezultă că:

$$x = x_1 + r * (x_2 - x_1)$$
$$y = y_1 + r * (y_2 - y_1)$$

Acest algoritm este utilizat în programul liniilor si suprafetelor ascunse în cazul în care se doreste aflarea punctului de intersectie dintre două segmente, caz in care raportul se determină pe baza distantelor de la capetele unui segment la celalalt.

În ipoteza că segmentul are capetele în punctele de coordonate (x_1, y_1, z_1) si (x_2, y_2, z_2), punctul de intersectie de coordonate (x, y, z) se va obtine prin intermediul relatiilor:

$$x = x_1 + r * (x_2 - x_1)$$
$$y = y_1 + r * (y_2 - y_1)$$
$$z = z_1 + r * (z_2 - z_1)$$

unde r este raportul în care punctul de intersectie împarte muchia dată.

BUPT

6.3. ALGORITMI PENTRU CALCULUL DISTANTELOR

6.3.1.Algoritm pentru calculul distantei de la un punct la o dreaptă

Calculul distantei de la un punct la o dreaptă se face în mod distinct pentru cazul în care se lucrează în 2D sau în 3D. De obicei dreptele sunt specificate prin două puncte [MV294][MV394].

În plan, distanta de la un punct P (a , b) la dreapta ce trece prin punctele $P_1(x_1, y_1)$ si $P_2(x_2, y)$ se obtine prin raportul dintre valoarea obtinută înlocuind coordonatele punctului P în ecuatia dreptei si radicalul sumei patratelor cosinusilor directori din ecuatia dreptei.

Dacă ecuatia dreptei ce trece prin două puncte este dată de relatia

$$\frac{x_2 - x_1}{x_1 - x} = \frac{y_2 - y_1}{y_1 - y}$$

atunci forma standard este:

$$x * (y_1 - y_2) - y * (x_1 - x_2) + x_1 * y_2 - x_2 * y_1 = 0$$

Valoarea distantei se va calcula cu relatia:

$$d = \frac{a * (y_1 - y_2) - b * (x_1 - x_2) + x_1 * y_2 - x_2 * y_1}{\mathrm{sqrt}((y_1 - y_2)^2 + (x_1 - x_2)^2)}$$

În cazul dreptei si punctului din spatiu, algoritmul foloseste ecuatia dreptei dată prin două puncte (x_1, y_1, z_1) si (x_2, y_2, z_2), în forma parametrică :

$$x = x_1 + t * (x_2 - x_1)$$
$$y = y_1 + t * (y_2 - y_1)$$
$$z = z_1 + t * (z_2 - z_1)$$

BUPT

Distanta de la punctul de coordonate (a, b, c) la dreaptă se calculează cu relatia:

$$d = sqrt(((b - y_1) * (z_2 - z_1) - (c - z_1) * (y_2 - y_1))^2 + ((c - z_1) * (x_2 - x_1) - (a - x_1) * (z_2 - z_1))^2 + ((a - x_1) * (y_2 - y_1) - (b - y_1) * (x_2 - x_1))^2) / sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)$$

Facând substitutiile:

$$rad = sqrt((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)$$
$$cx = (x_2 - x_1) / rad$$
$$cy = (y_2 - y_1) / rad$$
$$cz = (z_2 - z_1) / rad$$

relatia de calcul a distantei devine:

$$d_1 = sqrt(((y - y_1) * cz - (z - z_1) * cy) * ((y - y_1) * cz - (z - z_1) * cy) + ((z - z_1) * cx - (x - x_1) * cz) * ((z - z_1) * cx - (x - x_1) * cz) + ((x - x_1) * cy - (y - y_1) * cx) * ((x - x_1) * cy - (y - y_1) * cx)))$$

6.3.2.Algoritm pentru calculul distantei de la un punct la un plan

Valoarea distantei de la un punct la un plan foloseste ecuatia planului dată sub forma:

$$a * x + b * y + c * z + d = 0$$

Dacă punctul are coordonatele x_1, y_1, z_1, atunci distanta se calculează cu relatia

$$dist = \frac{a * x_1 + b * y_1 + c * z_1 + d}{sqrt(a^2 + b^2 + c^2)}$$

BUPT

Dacă planul este specificat prin trei puncte ale sale (x_1, y_1, z_1), (x_2, y_2, z_2) si (x_3, y_3, z_3) atunci ecuatia este poate fi scrisă sub forma:

$$\begin{vmatrix} x - x_1 & y - y_1 & z - z_1 \\ x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \end{vmatrix} = 0$$

Calculul coeficientilor ecuatiei se face cu relatiile:

$$a = (y_2 - y_1)*(z_3 - z_1) - (y_3 - y_1)*(z_2 - z_1)$$
$$b = -[(x_2 - x_1)*(z_3 - z_1) - (x_3 - x_1)*(z_2 - z_1)]$$
$$c = (x_2 - x_1)*(y_3 - y_1) - (x_3 - x_1)*(y_2 - y_1)$$

după care se foloseste expresia de mai sus pentru calculul distantei [MV294][MV394].

6.4. ALGORITMI PENTRU DETERMINAREA POZITIEI RELATIVE A DOUĂ FIGURI GEOMETRICE

Acesti algoritmi sunt necesari atât în procesul de eliminare a liniilor si suprafetelor ascunse cât si în cazul operatiilor booleene. Prin stabilirea pozitiei relative a două figuri geometrice se pot evita o serie de calcule complexe consumatoare de timp, dar si posibile surse de erori.

Prin stabilirea pozitiei relative a două segmente se poate evita calculul punctului de intersectie dintre dreptele suport ale segmentelor [Sav94] [Mu194] [Mu294][MV294][MV394].

6.4.1.Algoritm pentru determinarea pozitiei relative a unui punct fată de o dreaptă

Acest algoritm este folosit de algoritmul pentru determinarea pozitiei relative a două segmente sau a pozitiei relative a unui segment fată de un plan.

BUPT

Pozitia relativă a unui punct fată de o dreptă poate fi stabilită prin calculul distantei de la punct la dreaptă. Dacă distanta este zero, punctul se găseste pe dreaptă. O valoare diferită de zero indică faptul că punctul nu se găseste pe dreaptă [Per74].

6.4.2.Algoritm pentru determinarea pozitiei relative a unui punct fată de un segment

O primă estimare a pozitiei relative a unui punct fată de un segment se face folosind algoritmul pentru determinarea pozitiei relative a unui punct fată de dreaptă pe care se găseste segmentul. Dacă punctul nu se găseste pe dreaptă, evident că nu se poate găsi nici pe segment.

În cazul în care punctul se găseste pe suportul segmentului, trebuie să se stabilească situarea punctului între capetele segmentului sau pe prelungirea lui. Acest lucru se poate face calculând raportul în care punctul împarte segmentul. Situarea punctului în interiorul segmentului impune o valoare subunitară a raportului.

Dacă segmentul este definit prin punctele (x_1, y_1, z_1) si (x_2, y_2, z_2), iar punctul dat este de coordonate (a, b, c), atunci raportul în care punctul împarte segmentul poate fi calculat cu relatia (dacă $x_2 \neq x_1$):

$$r = \frac{a - x_1}{x_2 - x_1}$$

O valoare subunitară a raportului indică situarea punctului între capetele segmentului [Per74].

6.4.3.Algoritm pentru determinarea pozitiei relative a unui punct fată de un plan

Algoritmul pentru stabilirea pozitiei relative a unui punct fată de un plan dat este necesar atât în stabilirea pozitiei relative a unui segment fată de un plan cât si în algoritmul pentru eliminarea liniilor ascunse în momentul în care se stabilesc muchiile ce pot lăsa linii de umbra.

BUPT

Pozitia unui punct dat prin coordonatele lui în sistemul de referintă al modelului solidului de reprezentat, relativ la un plan determinat prin ecuatia sa, poate fi stabilită prin semnul expresiei obtinute înlocuind coordonatele punctului în ecuatia planului. O valoare pozitivă înseamna că punctul este situat în semispatiul indicat de normala planului, în timp ce o valoare negativă inseamna situarea în semispatiul opus. Valoarea zero a expresiei indică apartenentă punctului la planul considerat.

Dacă ecuatia planului este

$$a * x + b * y + c * z + d = 0$$

si punctul are coordonatele (p_1, p_2, p_3), atunci expresia al carui semn intereseazã este:

$$E = a * p_1 + b * p_2 + c * p_3 + d$$

Dacă $E = 0.0$ atunci punctul se găseste în planul considerat. Dacă $E > 0.0$ atunci punctul se găseste în semispatiul indicat de normala la plan, iar dacă $E < 0.0$ punctul se găseste în semispatiul opus [Per74].

6.4.4. Algoritm pentru determinarea pozitiei relative a unui punct fată de un poligon

Acest algoritm poate fi utilizat si în cazul stabilirii pozitiei relative a unui segment fată de un poligon dat, deoarece pozitiile capetelor segmentului pot da informatii legate de posibilitatea situării acestuia în interiorul sau exteriorul poligonului, sau de intersectia cu conturul acestuia.

Relativ la un poligon, un punct se poate situa în interiorul, exteriorul sau pe frontiera poligonului.

Suprapunerea unui punct peste unul din vîrfurile poligonului se poate stabili prin comparatii directe ale coordonatelor lor. Situarea punctului pe una din muchii, poate fi verificată folosind algoritmul pentru determinarea pozitiei relative a unui punct fată de un segment, în timp ce situarea punctului în interiorul poligonului trebuie verificată printr-un algoritm de interior [Per74].

BUPT

6.4.5.Algoritm pentru determinarea pozitiei relative a unui segment fată de un plan

Acest algoritm utilizează algoritmul pentru stabilirea pozitiei relative a unui punct fată de un plan dat.

Dacă ambele capete ale segmentului se găsesc de aceeasi parte a planului, rezultă inexistenta unui punct de intersectie între segment si plan.

Dacă extremitatile segmentului se găsesc de o parte si de alta a planului, există un punct de intersectie ce poate fi calculat dacă este cazul [Per74].

6.4.6.Algoritm pentru determinarea pozitiei relative a două segmente

Acest algoritm este utilizat în procesul de eliminare a liniilor si suprafetelor ascunse când segmentele în cauză sunt plasate în planul de perspectivă, dar si în cazul operatiilor boolene când segmentele sunt definite prin perechi de puncte din spatiul 3D.

În primul caz, cest algoritm este necesar deoarece duce la evitarea calculelor pentru aflarea punctului de intersectie dintre două segmente, în cazul în care acest punct nu există. Dacă testul asupra bounding-box-urilor celor două segmente de intersectat nu constată disjunctia lor, algoritmul pentru determinarea pozitiei relative a celor două segmente poate stabili existenta sau inexistenta punctului de intersectie.

Algoritmul se batează pe faptul că valoarea nulă sau semnul distantei de la un punct la un segment dă informatii asupra pozitiei punctului relativ la segment. Dacă distanta este nulă, punctul se găseste pe suportul segmentului. Dacă valoarea ei este pozitivă se găseste într-un semiplan, iar dacă este negativă se găseste în celalalt semiplan. Folosind distantele de la capetele fiecarui segment la celalalt segment, se poate stabili pozitia relativă a celor două segmente.

Dacă unul din segmente este definit prin punctele P_1 (x_1 , y_1) si P_2 (x_2 , y_2), atunci ecuatia dreptei pe care se găseste segmentul se scrie ca fiind:

$$\frac{x_2 - x_1}{x_1 - x} = \frac{y_2 - y_1}{y_1 - y}$$

De aici rezultă forma ecuatiei:

BUPT

$$x * (y_1 - y_2) - y * (x_1 - x_2) + (x_1 * y_2 - x_2 * y_1) = 0$$

Distanta de la un punct P (a , b) la dreapta ce trece prin punctele P1 si P2 se obtine prin raportul dintre valoarea obtinută inlocuind coordonatele punctului P în ecuatia dreptei si radicalul sumei patratelor cosinusilor directori din ecuatia dreptei:

$$d = \frac{a * (y_1 - y_2) - b * (x_1 - x_2) + x_1 * y_2 - x_2 * y_1}{sqrt((y_1 - y_2) * (y_1 - y_2) + (x_1 - x_2) * (x_1 - x_2))}$$

Segmentele a căror pozitie relativă trebuie stabilită pentru ca în final să se gasească eventualele puncte comune, sunt fie proiectiile în planul de perspectivă a două muchii sau portiuni de muchii, fie proiectia unei muchii sau portiuni de muchie si un segment auxiliar definit în conformitate cu anumite cerinte.

În cazul algoritmilor pentru efectuarea operatiilor booleene interesează pozitia relativă a două segmente definite prin două puncte din spatiu.

Acest algoritm este utilizat în procesul de stabilire a pozitiei relative a unei muchii fată de o dreaptă obtinută prin intersectia a două sau mai multe plane. Se evită calculele complexe pentru rezolvarea sistemului format din ecuatiile celor două drepte, prin calculul distantelor de la capetele muchiei la suportul dreptei. Dreapta este reprezentată prin două puncte situate la o distantă mare relativ la punctele care formează corpurile, pentru a putea fi considerată ca fiind o linie infinită. De aceea si dreapta poate fi considerată ca fiind un segment de dreaptă.

Algoritmul determină pozitia relativă a muchiei fată de dreptă prin pozitiile relative ale capetelor ei fată de dreaptă, deci prin calculul distantelor de la capetele ei la suportul dreptei.

Pentru a evita erorile de calcul datorită preciziei cu care se pot reprezenta numerele reale în calculator, cele două segmente de dreaptă sunt proiectate pe rând în cele trei plane ale sistemului de coordonate, astfel încât calculul distantelor se va realiza în plan.

Formula de calcul folosită în planul xOy pentru calculul distantei este urmatoarea:

$$d_1 = ((y_f - y_1) * (x_2 - x_1) - (x_f - x_1) * (y_2 - y_1)) / rad$$

BUPT

în care

$$rad = sqrt((x_2 - x_1) * (x_2 - x_1) + (y_2 - y_1) * (y_2 - y_1))$$

iar d_1 reprezintă distanta de la un capăt al muchiei (dat prin coordonatele x_f si y_f) la dreapta pe care se afla punctele de coordonate (x_1, y_1), respectiv (x_2, y_2). În cazurile particulare, când x_1 si x_2 sunt identice sau y_1 si y_2 sunt identice, distanta va avea valoarea $d_1 = x_f - x_1$ sau $d_1 = y_f - y_1$.

Folosind aceleasi formule se calculează si distanta de la celalalt capăt al muchiei la dreaptă.

Dacă ambele distante sunt diferite de zero si au acelasi semn (produsul lor este pozitiv), punctele sunt de aceeasi parte a dreptei, deci muchia nu intersectează dreapta.

Dacă ambele distante sunt diferite de zero, dar au semne diferite (produsul lor este negativ), capetele muchiei sunt de o parte si de alta a dreptei, deci muchia intersectează suportul dreptei.

Dacă una dintre distantele calculate sau chiar amindouă sunt nule, se repetă operatiile si în planele *yOz* si *xOz*.

În functie de valorile obtinute pentru distante, rezultă situatii distincte:

- dacă distanta de la un capăt al muchiei la suportul dreptei este zero, muchia se sprijină cu acel capăt pe dreaptă. (Se notează de la 1 la 4 cazurile de sprijin ale primului segment cu începutul, respectiv sfârsitul pe al doilea segment, si cazurile de spijin ale celui de-al doilea segment cu începutul, respectiv sfârsitul pe primul segment.)
- dacă toate cele sase valori calculate sunt nule, suporturile celor două segmente se suprapun. (Se notează de la 5 la 12 cazurile distincte de dispunere a celor două segmente pe suportul comun, ca în figura 6.1.)

În concluzie, algoritmul poate preciza dacă muchia se găseste pe suportul dreptei, dacă se sprijină cu unul din capete pe suportul dreptei sau dacă intersectează sau nu suportul dreptei [MV294][MV394].

6.4.7. Algoritm pentru determinarea pozitiei relative a două segmente coliniare

Dacă algoritmul pentru determinarea pozitiei relative a două segmente stabileste că ele se suprapun pe acelasi suport, atunci rămâne de stabilit dacă au

BUPT

sau nu puncte comune. Pentru aceasta se pot folosi operatii simple de comparare a coordonatelor capetelor celor două segmente, dar pentru a cunoaste perechile între care se efectuează comparatiile, este necesar să se cunoască dispunerea segmentelor pe suportul comun. Segmentele fiind date prin coordonatele capetelor lor într-o ordine oarecare si nu în ordinea crescatoare a valorilor, este necesar să se cunoască sensurile segmentelor relativ la un sens ales al suportului [MV294][MV394] .

Coordonata după care se face ordonarea punctelor ce determină segmentele (numită coordonată de ordonare), se alege ca fiind aceea care are o variatie nenulă pentru capetele primului segment.

Relativ la primul segment, coordonata de ordonare cea mai mică se notează cu "a", iar cea mai mare cu "b". Relativ la al doilea segment, coordonata de ordonare cea mai mică se notează cu "c", iar cea mai mare cu "d". Astfel se pot distinge cazurile posibile de plasare a celor două segmente pe suportul comun prezentate în figura 6.1.

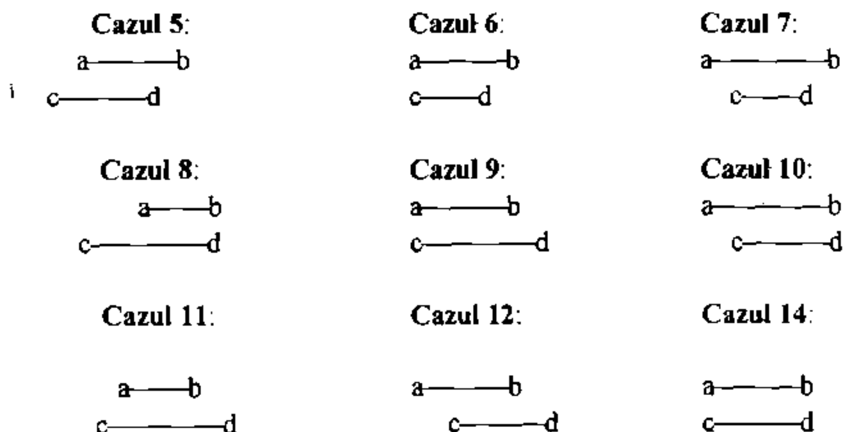


Figura 6.1. Pozitii relative ale segmentelor de dreaptă

BUPT

6.5. ALGORITMI PENTRU DETERMINAREA INTERSECTIILOR DINTRE SEGMENTE SI PLANE

6.5.1. Algoritm pentru determinarea punctului de intersectie dintre două segmente

Acest algoritm este necesar atât în cazul în care segmentele sunt definite în plan (planul de perspectivă în problema eliminării liniilor si suprafetelor ascunse), cât si în cazul segmentelor definite în spatiu (pentru operatiile booleene dintre solide) [Kri87][Per74][MV294][MV394].

Algoritmul pentru intersectia a două segmente foloseste algoritmii pentru testul de bounding-box aplicat celor două segmente si algoritmul pentru determinarea pozitiei relative a două segmente. Prin utilizarea acestor algoritmi se poate stabili existenta posibilă sau inexistenta punctului de intersectie a celor două segmente, fără a efectua în mod inutil calculele de intersectie. În acest mod se reduce timpul total necesar rezolvării operatiei de intersectie din cadrul operatiilor booleene sau a problemei liniilor si suprafetelor ascunse.

În cazul în care testul de bounding-box semnalează o posibilă intersectie între cele două segmente, se execută algoritmul pentru stabilirea pozitiei relative. Dacă se constată că este vorba de sprijinul unui segment cu un capăt pe cealalt segment, punctul de intersectie este deja cunoscut ca fiind acel capăt al segmentului. Dacă se constată că suporturile celor două segmente se intersectează, este necesar să se calculeze raportul în care punctul de intersectie împarte unul din segmente. Interesează situatia în care acest raport este subunitar, ceea ce înseamna că punctul de intersectie se afla în interiorul segmentului respectiv. În acest caz, punctul de intersectie se calculează cu ajutorul algoritmului pentru aflarea unui punct pe un segment dat, cunoscând raportul în care punctul împarte segmentul. Acest raport se determină pe baza distantelor de la capetele unui segment la celalalt, distante deja calculate în algoritmul pentru stabilirea pozitiei relative a celor două segmente.

În procesul de determinare a liniilor si suprafetelor ascunse algoritmul este necesar pentru aflarea punctelor de intersectie dintre proiectia unei muchii a unui poligon si proiectia unui alt poligon, problema ce poate fi redusa la aflarea punctului de intersectie dintre proiectiile a două muchii.

BUPT

În cazul operatiilor booleene, algoritmul este necesar în procesul de intersectie dintre două poligoane, proces ce reprezintă o etapa în operatia de intersectie dintre două solide. După ce prin testul de bounding-box se stabileste faptul că există posibilitatea intersectiei celor două poligoane după dreapta de intersectie a planelor ce le contin, algoritmul pentru stabilirea pozitiei relative a două segmente poate furniza mai multe informatii referitoare la modul în care se intersectează fiecare muchie a fiecarui poligon cu dreapta de intersectie: dacă intersectia se face într-un punct de pe muchie, în unul din capetele ei sau dacă muchia este suprapusa pe dreaptă.

În cazul în care punctul de intersectie este situat în interiorul muchiei, se efectuează calculele pentru determinarea lui. Pozitia lui pe segment se determină pe baza raportului în care punctul împarte segmentul, raport calculat cu relatia:

$$s = d_1 / (d_2 - d_1)$$

În acest scop se determină distantele d_1 si d_2 de la capetele muchiei la dreaptă folosind algoritmul pentru calculul distantei de la un punct la o dreaptă, după care se determină coordonatele punctului de intersectie, folosind algoritmul pentru aflarea unui punct când se cunoaste raportul în care împarte segmentul.

6.5.2. Algoritm pentru determinarea punctului de intersectie dintre un segment si un plan

După ce algoritmul pentru determinarea pozitiei relative a unui segment fată de un plan furnizează informatia că cele două se intersectează, în anumite situatii este necesar ca punctul să fie determinat [Per74][MV294][MV394].

Dacă planul este cunoscut prin ecuatia

$$a * x + b * y + c * z + d = 0$$

iar segmentul este dat prin capetele lui (x_1, y_1, z_1) si (x_2, y_2, z_2), atunci coordonatele (p_1, p_2, p_3) ale punctului de intersectie se vor calcula cu relatiile:

$$p_1 = [b(x_1y_2 - x_2y_1) + c(x_1z_2 - x_2z_1) - d(x_2 - x_1)]/n$$
$$p_2 = [c(y_1z_2 - y_2z_1) + a(y_1x_2 - y_2x_1) - d(y_2 - y_1)]/n$$
$$p_3 = [a(z_1x_2 - z_2x_1) + b(z_1y_2 - z_2y_1) - d(z_2 - z_1)]/n$$

BUPT

în care [Per74]:

$$n = a(x_2 - x_1) + b(y_2 - y_1) + c(z_2 - z_1)$$

6.5.3.Algoritm pentru determinarea liniei de intersectie dintre două plane

Cele două plane determinate prin normalele lor li se va calcula linia de intersectie doar după ce se verifică faptul că ele nu sunt paralele [Per74][MV294][MV394].

Ecuatiile celor două plane sunt de forma:

$$a_1*x + b_1*y + c_1*z + d_1 = 0$$
$$a_2*x + b_2*y + c_2*z + d_2 = 0 \quad (1)$$

unde a_1, b_1, c_1 respectiv a_2, b_2, c_2 sunt cosinusii directori ai normalelor celor două plane, iar d_1 si d_2 reprezintă distanta de la originea sistemului de coordonate la planul corespunzător.

Linia de intersectie este definită prin directia ei (un set de cosinusi directori) si printr-un punct prin care trece.

Un set de cosinusi directori ai liniei de intersectie dintre planele date prin ecuatiile (1) este:

$$(b_1*c_2 - b_2*c_1), (c_1*a_2 - c_2*a_1), (a_1*b_2 - a_2*b_1) \quad (2)$$

Pentru simplificare se utilizează un set de cosinusi directori normalizat, în care unul din ei are valoarea 1. Acest set se poate obtine împartând valorile din (2) la una dintre ele care este nenula.

Dacă toti cosinusii directori sunt nuli, intersectia planelor nu poate fi efectuată (este nedefinită).

Se testează pe rând care din valorile din (2) este nenula. Cu valoarea găsită se împart cosinusii directori din (2), iar valorile obtinute reprezintă directia dreptei de intersectie.

Pentru a simplifica calculele, punctul de pe dreaptă cu coordonatele notate (x_1, y_1, z_1) se alege ca fiind cel cu coordonata corespunzatoare cosinusului

BUPT

director 1.0, nulă. Celelalte 2 coordonate ale punctului se calculează rezolvând ecuatiile (1) în care s-a inlocuit coordonata cu valoarea zero.

Pentru cazul în care $a_1*b_2 - a_2*b_1$ este nenul, se alege $z_1 = 0.0$ si ecuatiile (1) devin:

$$a_1 * x_1 + b_1 * y_1 + d_1 = 0$$
$$a_2 * x_1 + b_2 * y_1 + d_2 = 0 \qquad (3)$$

Se rezolva sistemul format de aceste două ecuatii în raport cu x_1 si y_1. Se determină astfel coordonatele punctului de pe dreaptă de intersectie dintre plane:

$$x_1 = -(b_1 * d_2 - b_2 * d_1) / (a_1 * b_2 - a_2 * b_1)$$
$$y_1 = -(a_1 * d_2 - a_2 * d_1) / (a_1 * b_2 - a_2 * b_1)$$

Analog se determină punctul de pe dreaptă în cazurile în care $a_1*c_2 - a_2*c_1$, respectiv $c_1*b_2 - c_2*b_1$ sunt nenule si se aleg ca referinte [Per74].

6.6. ALGORITM PENTRU SELECTIA UNEI MUCHII ÎN PROCESUL DE STABILIRE A CONTURURILOR POLIGONALE

6.6.1.Algoritm original pentru selectia unei muchii în cazul stabilirii conturului unui poligon atomic

Si acest algoritm este necesar atât în procesul de eliminare a liniilor si suprafetelor ascunse, cât si în efectuarea operatiilor booleene dintre solide [Per74][MV294][MV394].

În urma procesului de intersectare dintre două solide, pe poligoanele lor initiale pot să apară noi muchii, iar în procesul de eliminare a liniilor si suprafetelor ascunse, pe proiectiile poligoanelor potential vizibile apar muchii noi ce vor delimita portiunile vizibile de cele ascunse. Poligoanele formate din muchiile, segmentele de muchii initiale sau muchiile noi, care nu mai pot fi descompuse în poligoane mai mici, se vor numi poligoane atomice. În procesul de constructie a solidului rezultat în urma operatiilor booleene, precum si în procesul

BUPT

de stabilire a portiunilor ascunse, sunt necesare poligoanele atomice rezultate în urma intersectiei, respectiv ascunderii. Obtinerea conturului unui poligon atomic reprezintă o problema în cazul în care există mai multe muchii ce pleaca din acelasi punct. În acest moment este necesar algoritmul de selectie care va fi prezentat mai jos.

Dându-se un astfel de poligon, o muchie a sa si o listă de muchii cu capătul de început comun identic cu sfirsitul muchiei date, trebuie aleasa muchia cea mai la stânga. Această problema ar putea fi rezolvată foarte simplu dacă muchia cea mai la stânga ar putea fi aleasa ca fiind cea care formează cu muchia dată unghiul cel mai mic. Calculul unghiurilor prin functii trigonometrice ridică însă probleme din cauza periodicitatii lor. De aceea sunt necesare mai multe operatii care să înlocuiască (să modeleze) procesul de recunoastere a muchiei celei mai din stânga, proces pe care omul îl poate efectua foarte simplu. [Kri87]

Muchia situată în pozitia cea mai din stânga se determină prin intermediul valorilor produselor mixte si scalare dintre vectorii atasati muchiei date si muchiilor din listă.

Algoritmul efectuează urmatoarele operatii:

Se construieste versorul a corespunzător muchiei date. Se iau în considerare primele două muchii din listă si se construiesc versorii b si c corespunzători. Muchia selectată va fi apoi luată în considerare cu urmatoarea din listă, dintre ele alegându-se cea mai la stânga. Procesul continua până la atingerea sfirsitului listei, moment în care muchia selectată va fi utilizată în procesul de închidere a conturului poligonal curent.

Pentru alegerea muchiei cea mai din stânga din cele două muchii luate în considerare, se determină proiectia fiecareia din ele pe muchia initiala, calculând produsul scalar al vectorilor (versorilor) corespunzători lor:

$$c_1 = a \cdot b$$
$$c_2 = a \cdot c$$

În plus mai este necesar calculul proiectiilor pe directia normalei a produselor mixte dintre vectorul muchiei initiale cu fiecare din vectorii corespunzători muchiilor în discutie si dintre vectorii celor două muchii de selectat:

$$db = (a \times b) \cdot normala_poligon$$
$$dc = (a \times c) \cdot normala_poligon$$
$$mixt = (b \times c) \cdot normala_poligon$$

BUPT

Pe baza valorilor obtinute se selectează muchia corespunzatoare vectorului *c* dacă una din conditiile de mai jos este îndeplinită:

1. db * dc >= 0 si mixt > 0 si (dc ≠ 0 sau (dc = 0 si c2 > 0))
2. db * dc >= 0 si mixt < 0 si db = 0 si c1 = 0
3. db < 0 si dc > 0

În caz contrar se selectează muchia corespunzatoare vectorului *b* [Mu194] [Mu294] [Mu394].

6.6.2. Algoritm original pentru selectia unei muchii în cazul stabilirii conturului unui poligon obtinut prin reuniunea mai multor poligoane

În cazul în care mai multe poligoane disjuncte coplanare de acelasi tip au muchii în comun, este necesar să se construiască un poligon al carui contur să fie format din muchiile poligoanelor initiale, prin eliminarea muchiilor sau portiunilor de muchii comune [MV294][MV394].

În această situatie se pune problema alegerii muchiei celei mai din dreapta dintr-un set de mai multe muchii care au punctul de început comun, relativ la o muchie care are ca punct de sfarsit punctul comun al celorlalte muchii.

Principiul de selectie a muchiei este acelasi cu cel din algoritmul anterior, cu deosebirea că stânga se înlocuieste cu dreapta si invers.

6.7. ALGORITMI PENTRU TESTUL DE INTERIOR

6.7.1. Algoritm original pentru stabilirea pozitiei relative a unui punct fată de un poligon

O operatie aparent banala pentru om, dar a carei implementare presupune serioase calcule matematice, o reprezintă testul de interior. În algoritmul pentru eliminarea liniilor ascunse interesează pozitia proiectiei unui capăt al unei muchii ce poate lăsa linie de umbra relativ la proiectia unei fete ce poate fi umbrită. În cazul algoritmilor pentru efectuarea operatiilor booleene interesează pozitia unui

BUPT

punct relativ la o fată a unui solid pentru a se stabili dacă un anumit contur poligonal se găseste în interiorul altuia (primul este sau nu gaura a celui de al doilea), precum si pozitia unui punct relativ la interiorul unui solid pentru a stabili dacă solidul contine cavităti [MV294][MV394].

Algoritmul foloseste metoda bazată pe numărul punctelor de intersectie dintre o semidreaptă ce trece prin punctul dat si poligonul sau volumul fată de care se face testul. Dacă numărul punctelor de intersectie este par, punctul se afla în exterior, iar dacă numărul este impar, punctul de afla în interior. Pentru a evita complicatiile în cazul în care semidreapta aleasa se suprapune peste o muchie a poligonului sau volumului, algoritmul permite schimbarea pantei semidreptei alese initial.

În cazul algoritmului de eliminare a liniilor si suprafetelor ascunse, punctul este considerat prin coordonatele sale în planul de perspectivă, la fel cum este considerat si poligonul fată de care se efectuează testul de interior.

Deoarece semidreapta poate fi oarecare, se va alege (pentru început) una care trece prin punctul dat si este paralela cu prima muchie a poligonului. Semidreapta va fi definită prin două puncte: primul este punctul dat, iar al doilea este un punct situat pe directia aleasa la distantă suficient de mare, astfel încât să poată fi considerat la infinit.

Semidreapta aleasa va fi intersectată cu toate muchiile poligonului, inclusiv muchiile ce formează contururile de gaură. Pentru aceasta se foloseste algoritmul pentru determinarea pozitiei relative a două segmente, în care nu interesează efectiv punctul de intersectie, ci doar dacă cele două segmente se intersectează. Intersectiile semidreptei cu acestea vor fi contorizate tinând cont si de faptul că semidreapta intră sau iese din suprafata poligonului. Orice punct de intrare incrementează contorul, în timp ce punctele de iesire îl decrementează. Cazurile de suprapunere dintre semidreaptă si muchii nu vor influenta valoarea contorului. În acest fel, în final contorul poate fi doar 0 sau 1, 0 pentru cazul în care punctul este exterior si 1 pentru cazul în care punctul este interior poligonului.

Stabilirea faptului că într-un punct de intersectie semidreapta intră sau iese din suprafata poligonului, se face prin calcul vectorial. Dacă prin produsul vectorial al vectorilor definiti de muchia curentă si semidreaptă rezultă un vector ce intră în planul de perspectivă, contorul este decrementat, iar dacă vectorul iese din planul de perspectivă, contorul este incrementat. Intersectia semidreptei se face cu muchii consecutive parcurse în sensul în care suprafata interioară a poligonului rămâne la stânga muchiilor.

BUPT

În cazul în care ar fi fost folosit un algoritm care ar lua în considerare doar numărul punctelor de intersectie, fără a se efectua calculul produsului vectorial dintre cele două segmente care se intersectează la un moment dat, ar fi necesară ordonarea acestor puncte pe directia semidreptei si marcarea portiunilor în care semidreapta se suprapune peste o muchie a poligonului. Complicatiile ce apar în astfel de situatii au condus la adoptarea variantei prezentate mai sus.

6.7.2.Algoritm original pentru stabilirea pozitiei relative a unui punct fată de un volum

În cazul operatiilor booleene punctul este considerat prin coordonatele sale spatiale, la fel ca si poligonul fată de care se efectuează testul. Principiul este acelasi ca si în cazul testului efectuat în planul de perspectivă, doar că semidreapta aleasa trebuie să se gasească în planul poligonului considerat.

Pentru testul de interior efectuat asupra unui punct relativ la un volum dat, semidreapta se alege pe baza informatiilor furnizate de bounding-box-ul volumului. În acest caz semidreapta se intersectează cu toate fetele volumului, iar numărul intersectiilor este interpretat în mod asemanător testului de interior aplicat relativ la un poligon. Acest test foloseste algoritmul pentru stabilirea pozitiei relative a unui segment fată de un plan dat, iar în cazul în care se ajunge la concluzia că segmentul intersectează planul, este necesar să se cunoască coordonatele punctului de intersectie pentru a se efectua testul de interior relativ la conturul poligonal al fetei curente. Punctul de intersectie este luat în considerare doar în cazul în care se găseste în interiorul poligonului [MV294][MV394].

6.8. CALCULE DE ADANCIME

În procesul de stabilire a liniilor de umbra (segmenetele care separă partile vizibile de cele ascunse) trebuie să se cunoască mai multe date referitoare la un punct dat, cum ar fi: distanta fată de punctul de privire; pozitia sa relativă fată de un plan (de aceeasi parte sau de parte opusa cu punctul de privire); coordonatele sale spatiale, atunci când se cunosc coordonatele proiectiei sale în planul de perspectivă si planul în care se găseste.

Pentru determinarea suprafetelor ascunse, este necesar să se stabilească muchiile ce lasă linii de umbra, precum si poligoanele pe care apar aceste linii. O muchie poate lăsa linie de umbra pe un poligon dacă se găseste de aceeasi parte

BUPT

cu punctul de privire fată de planul poligonului, dacă cel putin un segment al proiectiei sale cade în interiorul proiectiei poligonului si dacă nu există un alt poligon interpus între cel considerat si punctul de privire.

Dacă din multimea poligoanelor care contin în interiorul proiectiei lor proiectia unui capăt al muchiei în discutie, există cel putin un poligon care se interpune între punctul de privire si muchia considerată, atunci muchia nu poate lăsa umbra, deoarece ea însăsi este ascunsă.

Cunoscând că normala oricarui poligon potential vizibil este îndreptată înspre punctul de privire, înlocuind coordonatele unui capăt al muchiei în ecuatia planului în care se găseste poligonul, se obtine o valoare pozitivă dacă punctul este de aceeasi parte cu punctul de privire, respectiv o valoarea negativă în caz contrar. Valoarea zero marchează apartenenta punctului la planul considerat. În situatia în care se obtine cel putin o valoare negativă, muchia în cauza va fi marcată ca fiind ascunsă.

Algoritmul pentru ordonarea punctelor din spatiu proiectate în acelasi loc în planul de perspectivă este necesar pentru a stabili poligonul cel mai apropiat de ochi, pe care o muchie dată poate lăsa linie de umbra. Evident că din multimea poligoanelor vizibile sunt luate în considerare doar acelea care contin în interiorul lor proiectia unuia din capetele muchiei în cauză. Punctele din spatiu, pe baza carora se vor ordona poligoanele, sunt punctele în care raza vizuala ce trece printr-un capăt al muchiei intersectează poligoanele.

În acest scop se calculează o valoare egala cu patratul distantei unui punct fată de punctul de privire, valoare egala cu suma patratelor diferentelor coordonatelor celor două puncte pe cele trei directii ale sistemului de coordonate. Această valoare va reprezenta elementul după care se efectuează ordonarea în lista de poligoane. Ordonând crescător lista după acestă valoare, se obtine poligonul umbrit ca fiind acela care corespunde primului element din listă, adică cel corespunzător punctului cel mai apropiat de punctul de privire.

Dacă din multimea poligoanelor care contin în interiorul proiectiei lor proiectia unui capăt al muchiei în discutie există cel putin un poligon care se interpune între punctul de privire si muchia considerată, atunci muchia nu poate lăsa umbra, deoarece ea însăsi este ascunsă.

Cunoscând că normala oricarui poligon potential vizibil este îndreptată înspre punctul de privire, înlocuind coordonatele unui capăt al muchiei în ecuatia planului în care se găseste poligonul, se obtine o valoare pozitivă dacă punctul este de aceeasi parte cu punctul de privire, respectiv o valoarea negativă în caz

BUPT

contrar. Valoarea zero marchează apartenenta punctului la planul considerat. În situatia în care se obtine cel putin o valoare negativă, muchia în cauză va fi marcată ca fiind ascunsă.

Algoritmul pentru determinarea coordonatele spatiale ale unui punct atunci când se cunosc coordonatele proiectiei perspective si planul în care se găseste punctul, este necesar în procesul de stabilire a segmentelor ce separă portiunile vizibile de cele ascunse. În situatia în care s-a stabilit că o anumită muchie lasă linie de umbra pe un poligon, extremitatile muchiei trebuie proiectate pe acel poligon. Coordonatele de perspectivă ale capetelor muchiei coincid cu coordonatele de perspectivă ale proiectiei ei. În procesul de stabilire a poligoanelor vizibile si ascunse, sunt însă necesare coordonatele spatiale ale capetelor proiectiei muchiei.

6.9. ALGORITM PENTRU COMPARAREA A DOUĂ DREPTE REZULTATE PRIN INTERSECTII DE PLANE

Algoritmul este necesar în cazul în care fiecarei linii de intersectie dintre două sau mai multe plane trebuie să i se asocieze o singură structura de date.

În ipoteza că fiecare linie de intersectie este descrisă prin directie si un punct, si că se cunosc planele prin a căror intersectie a rezultat, compararea trebuie facută în două etape [MV294].

În primul rând se compară vectorii ce descriu directiile liniilor. Egalitatea lor duce la concluzia că liniile sunt sau paralele sau identice. A doua etapă a comparatiei are sens doar în cazul în care prima comparatie a reusit, si ea constă în verificarea suprapunerii liniilor. Acest lucru se poate face verificând faptul că punctul prin care este definită una din linii apartine planelor care au determinat cealaltă linie.

Pentru a verifica dacă punctul apartine celor două plane se inlocuiesc coordonatele punctului în ecuatiile planelor. Dacă coordonatele punctului verifică ambele ecuatii, atunci punctul apartine dreptei, si deci liniile de intersectie sunt identice.

BUPT

6.10. ALGORITM PENTRU DETERMINAREA FETELOR POTENTIAL VIZIBILE

În algoritmul pentru stabilirea caracterului vizibil sau invizibil al unei fete a unui solid dat, se tine seama de normala planului în care se găseste fata, precum si de pozitia punctului din care este privit solidul [MV394].

O fată situată într-un plan care are normala îndreptată înspre observator poate fi vizibilă dacă nu este ascunsă de alte fete ale aceluiasi sau altui solid. De aceea pentru a se stabili orientarea fetei relativ la pozitia observatorului, se poate folosi orientarea normalei planului relativ la vectorul determinat de punctul de privire si un punct al fetei în cauză. Distanta relativ mare la care este situat punctul de privire face ca alegerea punctului de pe fata considerată să nu influenteze esential orientarea vectorului folosit.

Proiectia normalei planului pe directia vectorului ales dă informatii în legatură cu orientarea planului relativ la pozitia observatorului. Dacă proiectia are sensul înspre observator, atunci fata poate fi vizibilă. În caz contrar, fata nu poate fi vazută de catre observatorul plasat în acel punct.

Notând cu N versorul normalei planului ce contine fata în cauză si cu V versorul directiei care uneste punctul de observare cu un vîrf al fetei, produsul scalar al celor doi vectori este dat de relatia:

$$p = |N| \cdot |V| \cdot \cos(a)$$

Semnul valorii lui p poate fi folosit în stabilirea vizibilitatii fetei: o valoarea pozitivă corespunde unui plan vizibil, în timp ce o valoare negativă corespunde unui plan invizibil.

Pentru ca acest algoritm să poată fi folosit si în varianta în care portiunile de fete vizibile sunt reprezentate cu diferite nuante în functie de înclinarea lor relativ la punctul de observare, este necesar să se ia în considerare valoarea unghiului a, prin cosinusul sau. Astfel, cu cât cosinusul este mai mare (unghiul a este mai mic) cu atât mai deschisă trebuie să fie nuanta cu care este colorată fata respectivă. S-a ales această regula în ipoteza că punctul de observati coincide cu punctul în care se găseste sursa de lumină. Intervalul [0, 1] în care cosinusul poate lua valori în cazul fetelor vizibile a fost împartit în cinci subintervale, carora le corespund cinci nuante.

Deoarece în stabilirea pozitiei relative a celor doi vectori intereseazã unghiul format de directiile lor, în calcule se pot folosi versorii acestora.

BUPT

$p = |N| . |V| . \cos(a) = \cos(a)$

Deoarece functia cosinus este monotona pe intervalul [0, π/2], divizarea acestui interval în subintervale poate fi înlocuită cu divizarea intervalului [0, 1] în subintervale.

Valoarea produsului scalar este chiar valoarea cosinusului unghiului a format de cele două directii, deoarece modulele versorilor sunt unitare.

6.11. ALGORITM PENTRU DETERMINAREA CONTURURILOR UNUI SOLID

Conturul unui solid este o secventă închisă de muchii, numite muchii exterioare, având proprietatea că se învecinează într-o parte cu o fată potential vizibilă si în cealaltă parte cu una invizibilă [MV294][MV394].

Plecând de la această definitie, rezultă că operatia de stabilire a caracterului vizibil sau invizibil al fiecarei fete, trebuie efectuată anterior operatiei de determinare a contururilor.

În mod obisnuit, muchiile exterioare ale unui solid închid un singur contur. În cazul în care solidele contin gauri pot exista două sau mai multe contururi.

Din punct de vedere al proiectiei solidului în planul de perspectivă, conturul poate contine sau nu bucle. O încrucisare a proiectiilor unei perechii de muchii de contur produce schimbarea vizibilitatii unei din ele. Pe această proprietate se bazează anumiti algoritmi folositi în eliminarea liniilor ascunse. Tot din cauza posibilitatii existentei buclelor formate de contururile solidelor, este necesar să se aplice algoritmul de determinare a pozitiei relative a două segmente, iar în cazul în care se stabileste că segmentele se intersectează, trebuie calculat punctul de intersectie. Cu toate că intersectie se face în planul de perspectivă la nivelul proiectiilor muchiilor de contur, este necesar să se stabilească coordonatele spatiale ale punctelor de pe muchii corespunzatoare punctului de intersectie din proiectie.

Fiind cunoscută multimea muchiilor exterioare ale unui solid, algoritmul determină succesiunea muchiilor care formează fiecare contur al solidului.

Plecând de la prima muchie exterioară întâlnită, se caută o muchie urmatoare care are capătul de început identic cu capătul de sfirsit al muchiei curente. Identitatea se stabileste la nivelul coordonatelor spatiale ale punctelor în cauză. Procedeul continua până în momentul în care capătul de sfirsit al muchiei

BUPT

curente coincide cu capătul de început al primei muchii a conturului. În cazul solidelor cu mai multe contururi, după închiderea primului contur rămân muchii exterioare neutilizate. De aceea procedeul se reia până când toate muchiile exterioare au fost cuprinse într-un astfel de contur. Efectuarea acestor operatii presupune marcarea fiecarei muchii utilizate într-un contur, deoarece doar în momentul în care nu mai există muchii nemarcate se poate lua decizia de încheiere a procesului de stabilire a contururilor solidului.

6.12.ALGORITMUL ORIGINAL DE CLASIFICARE

Algoritmul de clasificare se referă la stabilirea tipului muchiilor sau portiunilor de muchii în functie de pozitia lor relativă la un poligon sau volum dat [MV294][MV394].

Astfel de operatii de clasificare sunt necesare atât în algoritmul pentru efectuarea operatiilor booleene dintre solide, cât si în cel pentru eliminarea liniilor ascunse.

În urma procesului de intersectie dintre o dreaptă si un poligon, rezultă o multime de puncte care pot fi ordonate pe directia dreptei. Fiecare pereche de puncte consecutive determină un segment a cărui pozitie relativă la poligon determină tipul: segment interior, exterior sau suprapus pe o muchie a poligonului. Există o multitudine de metode pentru stabilirea tipului, dar luând în considerare avantajele si dezavantajele fiecăreia, s-a ajuns la următoarea solutie utilizată în ambii algoritmi.

In ipoteza că fetele solidelor sunt descrise prin muchii specificate in ordinea parcurgerii conturului fetei cu materialul la stânga, se va putea determina dacă un segment orientat intră sau iese din suprafata inchisă de poligonul fetei sau se suprapune peste o muchie a acesteia. Considerând atât segmentele de pe dreapta de intersectie cât si muchiile poligonului ca vectori, se poate calcula câte un produs vectorial pentru fiecare pereche de vectori formată dintr-un segment si muchia pe care se sprijină.(In momentul in care se efectuează această operatie, toate muchiile au fost segmentate in muchii atomice. De aceea toate intersectiile dintre muchii sunt de tip sprijin cu unul din capetele segmentelor atomice.) Sensul acestui vector dă informatii in legatură cu pozitia relativă a primului vector fată de poligon: dacă vectorul rezultat are sensul normalei la poligon, segmentul iese din interiorul poligonului, iar dacă vectorul rezultat are sens opus normalei, segmentul intră in interiorul poligonului. Fiecarui segment i se asociază un fanion care va primi valoarea *IN*, *OUT* sau *ON* in functie de situatia concretă.

BUPT

In algoritmul pentru eliminarea liniilor si suprafetelor ascunse, această operatie este suficientă deoarece interesează pozitia segmentelor unei muchii relativ la un poligon. In cazul algoritmului pentru efectuarea operatiilor booleene dintre solide, cand interesează pozitia relativă a segmentelor fată de două poligoane, clasificarea trebuie facută tinind seama de faptul că un segment de intersectie trebuie să fie interior ambelor poligoane [Zei91].

BUPT

CAPITOLUL 7

ALGORITM ORIGINAL PENTRU MODELAREA SOLIDELOR FOLOSIND OPERATIILE BOOLEENE

7.1. PRINCIPIUL DE BAZA AL ALGORITMULUI [MV194] [MV294][MV394]

Algoritmul pentru efectuarea operatiilor booleene având ca operanzi solide permite tratarea singularitătilor de tip suprapunere de muchie - muchie, muchie - fată, fată - fată, vertex - vertex sau vertex - muchie. Algoritmul poate stabili numărul obiectelor rezultate si tipul fiecăruia si reuseste să construiască obiecte regulate din punct de vedere topologic. Acest lucru este obtinut prin modul în care se efectuează operatiile.

În prima fază, se intersectează fiecare fată a unui obiect cu fiecare fată a celuilalt obiect. Acest lucru se face de fapt prin intersectiile muchiilor perechilor de fete luate pe rând în considerare, precum si prin intersectiile muchiilor unei fete cu planul în care se găseste cealaltă fată. Ca urmare a acestei operatii, se obtin muchii atomice, care nu mai pot fi divizate prin altă operatie. Acestor muchii le sunt asociate toate atributele topologice necesare constructiei obiectelor rezultante.

Se foloseste un arbore de decizie pentru a se face distinctie între intersectii si suprapuneri ale muchiilor atomice, luând în considerare precizia calculelor si

BUPT

erorile de aproximare. 99% din erorile detectate se datoresc acestor tipuri de probleme.

Descompunerea elementelor unui obiect în entităti atomice (multimea fetelor plane intersectabile descompuse în fete neintersectabile mărginite de muchii atomice) se face în patru etape logice: găsirea fetelor potential intersectabile; calcularea intersectiilor si definirea muchiilor atomice; împărtirea fetelor intersectate si definirea noilor fete atomice; redefinirea noilor muchii sau perechi multiple de muchii dintre doi vertexi asociînd perechi de fete atomice după cum se impune din punct de vedere topologic. Toate singularitătile sunt tratate într-o manieră uniformă. Pe baza acestor informatii se poate construi obiectul sau obiectele rezultate.

Indiferent de numărul operanzilor care intervin intr-o sesiune de lucru, algoritmul ia in considerare la un moment dat o singura pereche. Intre cei doi operanzi nu trebuie facută nici o distinctie dacă ei intervin in operatia de reuniune sau intersectie, deoarece acestea sunt operatii comutative. In cazul diferentei insă, trebuie stabilită cu precizie ordinea operanzilor, deoarece diferenta nu este comutativă. In acest caz, primul operand se va numi corp moale, iar al doilea tare, deoarece rezultatul diferentei reprezintă acea parte din primul operand care nu are puncte comune cu al doilea (din primul operand se detasează acele părti care sunt situate in exteriorul celui de-al doilea).

Algoritmul pentru efectuarea operatiilor booleene tratează in mod unitar cazul intersectiei, reuniunii si diferentei. Indiferent ce operatie trebuie efectuată este necesară stabilirea părtilor comune ale solidelor operanzi. In cazul reuniunii, solidul rezultat va contine părtile necomune ale celor doi operanzi si partea comună luată o singura dată. In cazul intersectiei solidul rezultat va contine părtile comune ale operanzilor, iar in cazul diferentei, partea necomună a primului operand. Conform definitiei operatiilor cu multimi, s-ar putea ca multimea rezultat să nu poată reprezenta un solid corect din punct de vedere topologic. De aceea, in unele situatii, efectuarea operatiei cerute s-ar putea să nu furnizeze nici un rezultat [Sav94][Mu194][Mu294][MV294].

7.2. STRUCTURILE DE DATE UTILIZATE IN PROGRAMUL PENTRU EFECTUAREA OPERATIILOR BOOLEENE INTRE SOLIDE

In multimea structurilor de date ale pachetului de programe care stă la baza sistemului CAD utilizat, există o serie de tipuri care stau la baza descrierii

BUPT

topologiei solidelor. Pe langa acestea, programul care implementează algoritmul pentru efectuarea operatiilor booleene defineste structurile proprii necesare. Din prima categorie fac parte structurile *Volume*, *Poly* si *Vertex*.

Structura ***Volume*** cuprinde pe lângă altele, câmpuri ce pot contine informatii care interesează în problema modelarii unui solid: un pointer la o listă de blocuri ce contin pointeri la structurile de tip *Vertex* corespunzătoare vârfurilor prin care este definit modelul solidului; un pointer la o listă de blocuri ce contin pointeri la structurile de tip *Poly* corespunzătoare poligoanelor asociate fetelor din care este alcătuit volumul; un pointer la un tablou de structuri de tip *BPoly* care contine informatiile necesare modelării prin operatii booleene, precum si dimensiunea acestui tablou.

Structura ***Poly*** încapsulează date ce definesc un poligon: normala sa; distanta fată de originea sistemului de coordonate; un număr pentru identificarea sa în lista de poligoane ale volumului; numărul unui vârf al său notat *f_point* de la care plecând se poate stabili secventa vârfurilor ce formează conturul poligonului; informatii despre găurile existente în poligon.

Structura ***Vertex*** este asociată fiecărui vârf al solidului. Ea contine câmpul *number* în care se păstrează numărul vertexului si campul *Form* ce pastrează intr-un tablou informatii legate de topologia solidului: vertexii de care este legat prin muchii vertexul curent, precum si fetele din care fac parte acestea (ambele tipuri de informatii sunt reprezentate prin numerele de ordine ale vertexilor, respectiv ale poligoanelor asociate fetelor, numărul vertexilor se află la indici pari ai tabloului, iar numărul poligoanelor corespunzătoare la indici impari din tablou).

Câmpurile *coord* si *pers* păstrează coordonatele în sistemul de coordonate al modelului prin structura de tip *Vector* (tablou de trei valori reale), respectiv coordonatele corespunzătoare proiectiei punctului în planul de perspectivă prin structura de tip *Perspective* (tablou de două valori reale), numite coordonate de perspectivă.

Structurile specifice implementarii modelarii prin operatii booleene sunt prezentate mai jos.

Structura **BPoly** este o structură ce descrie o fata a solidului. Ea contine pointerul la structura *Poly* corespunzător fetei si pointerul la structura *Volume* a solidului. Exista deasemenea un tablou indicat de campul *bedge* de dimensiune

BUPT

egala cu valoarea memorata in campul *nrp* care pastreaza secventa de muchii ce formeaza conturul fetei. In cazul in care fata contine gauri, se completeaza campul *gauri* ce indica un tablou de pointeri la structuri de tip *StrHole*, tablou având dimensiunea memorată în câmpul *hbox*.

Pe parcursul operatiei booleene, pe lângă muchiile initiale ale poligonului, apar segmente de tăiere si atingere introduse în structură în ambele sensuri. Pentru fiecare dintre aceste tipuri există câte un pointer la un tablou de muchii si câte un câmp care păstrează dimensiunea tabloului (*ext*, *inter*, *fext*, *fint*, *cbox*, *ibox*, *febox*, *fibox*).

Campul *pl* păstrează pointerul la o structură de tip *PlanB* corespunzătoare planului in care se gaseste fata.

În plus, structura contine patru fanioane care sunt folosite pentru diferite operatii si indică de exemplu tipul poligonului din punct de vedere al operatiei booleene, numărul lui, etc.

Structura **Muchie** corespunzătoare unei laturi a unui poligon contine următoarele informatii: punctul de început si punctul de sfârsit ca pointeri la structurile *PVertex* corespunzătoare; poligonul din stânga, respectiv din dreapta muchiei ca pointeri la structuri de tip *BPoly*, linia de intersectie pe care se găseste această muchie, ca pointer la o structură de tipul *LinieB*; lista segmentelor în care este divizată o muchie prin campul *cedge*, pointer la un tablou de pointri la structura de tip *PVertex*, de dimensiune precizata in campul *nrc*; patru fanioane utilizate în diferite momente ale prelucrării booleene.

Structura **StrHole** contine un pointer la un tablou cu elemente de tip *Muchie* (*conturg*) si un întreg ce reprezintă dimensiunea acestui tablou (*gbox*).

Elementele tabloului corespund secventei de muchii care formează conturul găurii.

Structura **PlanB** poate păstra informatiile ce descriu planele care la rândul lor contin fetele solidelor operanzi in operatiile booleene.

Această structură de date reprezintă elementul de bază al unei liste înlăntuite.In afara câmpurilor ce contin informatia utilă, există un câmp de tip pointer care asigură înlăntuirea elementelor din listă.

Informatiile păstrate în structura *PlanB* necesare în procesul de intersectie a volumelor ce intervin într-o operatie booleană sunt următoarele: parametrii ecuatiei planului în câmpul *normal* (cosinusii directori) si *D* (termenul liber);

BUPT

poligoanele aflate în plan referite prin campul *bpoly*, pointer la un tablou de pointeri de tip *BPoly* de dimensiune specificată in campul *nrpo*.

Structura de date **LinieB** descrie o linie de intersectie dintre doua sau mai multe plane. Ea corespunde elementelor ce alcătuiesc lista înlăntuită a tuturor liniilor de intersectie formate între planele ce contin fetele solidelor ce intervin în operatia booleană dorită. De aceea, pe lângă informatia referitoare la o astfel de linie, structura mai contine si pointerul de înlăntuire.

Informatiile referitoare la linia de intersectie sunt următoarele: caracteristicile geometrice ale liniei (directia ei si un punct prin care trece, păstrate în câmpuri de tip *Vector*, *dir* si *pct*); un pointer la un tablou de pointeri la structuri de tip *PlanB* (*plane*) si un întreg ce reprezintă dimensiunea acestui tablou (*nrpl*), plane care se intersecteaza dupa linia curenta; un pointer la un tablou cu elemente de tip *McPol* (*mcpol*) si un întreg ce reprezintă dimensiunea lui (*nrmp*), tablou ce contine muchiile situate pe această linie de intersectie.

Structura **McPol** contine un pointer la un tablou de pointeri la muchii (*mc*) si un întreg care reprezintă dimensiunea acestui tablou (*mcbox*). Muchiile indicate prin pointerii continuti în tablou au proprietatea că se găsesc pe aceeasi linie de intersectie dintre două sau mai multe plane.

Sructura **LBEdge** este folosită pentru păstrarea informatiei referitoare la punctele de intersectie dintre o linie si muchiile a două poligoane. Linia este determinată de intersectia planelor în care se găsesc cele două poligoane. Ea va mai fi referita ca linie infinita.

Aceste informatii trebuie să asigure stabilirea portiunilor de pe linie care se vor marca în cele două poligoane ca muchii de tăiere (intersectie) sau de atingere (muchii care se sprijina pe o fata).

Pentru a retine toate punctele de intersectie, se crează o listă de elemente de tipul *LBEdge*. În această listă, elementele sunt ordonate în functie de coordonatele punctului de intersectie dintre o muchie a unui poligon si linia de intersectie.

Informatiile continute într-un element de tip *LBEdge* sunt următoarele: coordonatele punctului de intersectie; pointer la muchia din primul poligon cu care s-a intersectat linia infinită în acest punct; pointer la muchia din al doilea poligon cu care s-a intersectat linia infinită în acest punct; primul poligon care a intervenit în operatia de intersectie; al doilea poligon care a intervenit în operatia de intersectie; câte un fanion care specifică faptul că linia de intersectie intră sau

BUPT

iese din primul, respectiv al doilea, poligon; câte un fanion care indică faptul că muchia din primul, respectiv al doilea, poligon se găseste pe linia infinită.

Pentru muchiile aflate pe linia de intersectie, vor exista, în listă, două elemente: un element corespunzător punctului de început si un element corespunzător punctului de sfârsit al muchiei.

Structura **IntExtL** este folosită pentru crearea unei liste de muchii sau segmente de muchii ce pot forma conturul unui poligon. Astfel de contururi sunt necesare în diferite etape ale algoritmului. Muchiile pot fi muchii initiale, segmente de muchii initiale, muchii de tăiere interioare sau exterioare si muchii de atingere interioare sau exterioare sau segmente ale acestora.

Fiind folosită într-o structură dinamică de listă înlăntuită, contine un pointer pentru a indică elementul următor.

Câmpurile acestei structuri sunt următoarele: un pointer la o structură de tip *Muchie* corespunzătoare muchiei considerate sau corespunzătoare muchiei căreia îi apartine segmentul considerat; doi pointeri la structuri de tip *PVertex* care corespund punctelor de început si de sfârsit ale muchiei sau segmentului de muchie considerat; fanionul *poz* care contine numărul de ordine al segmentului de muchie care interesează; câmpul tip care este folosit pentru a specifica tipul muchiei sau al segmentului de muchie (tăietură sau atingere, interioară sau exterioară).

Structura **TabPol** este folosită la crearea unei liste înlăntuite de poligoane de tip *BPoly*, poligoane din care in final se va obtine solidul rezultat al operatiei booleene.

Structura *TabPol* are ca elemente un pointer la o structură de tip *BPoly* corespunzătoare unui poligon (*pol*) si un câmp de înlăntuire spre următorul element din listă (*next*).

Structura **ListPol** este necesară în procesul de stabilire a pozitiei relative a două unghiuri diedre. Ea este utilizată la crearea unei liste de vectori care să corespundă amplasării spatiale a celor patru poligoane care formează fetele a două diedre cu muchia comună.

Unul din diedre este format din poligoane sau portiuni de poligoane din primul volum, iar al doilea diedru din poligoane sau portiuni de poligoane din al doilea volum.

Cunoasterea pozitiei relative a acestor fete ale diedrului este esentială în procesul de selectare a poligoanelor care vor forma diedrul sau diedrele

BUPT

rezultante, în functie de operatia booleană cerută. Odată creată această listă se poate fixa si tipul poligoanelor (exterioare, interioare sau de frontieră).

Printr-o conventie se stabileste poligonul stâng si poligonul drept al fiecărui diedru si se asociază câte un vector fiecărei fete. Rezultă astfel patru vectori numiti prin conventie: stânga moale (*SM*), stânga tare (*ST*), dreapta moale (*DM*), dreapta tare (*DT*), in functie de solidul din care provin. Vectorii se aleg perpendicular pe muchia diedrului, inclusi în poligonul căruia îi sunt asociati. Ordonarea se face plecând într-un anumit sens de la vectorul stânga moale si luând în ordine toti ceilalti vectori întâlniti.

Structura *ListPol* contine informatii referitoare la fetele care se sprijina pe doua muchii suprapuse: vectorul corespunzător fiecarei fete; tipul acestui vector (din ce volum face parte si de care parte a muchiei se gaseste: *SM* - fata din stanga din volumul moale, *ST* - fata din stanga din volumul tare, *DM* - fata din dreapta din volumul moale, *DT* - fata din dreapta din volumul tare); poligonul căruia îi este asociat vectorul; pointer spre următorul element al listei; pointer spre un eventual element al listei corespunzător unui poligon coplanar poligonului curent.

Există posibilitatea ca două poligoane să fie coplanare (de tip *FRONTIERA*), caz în care vectorii corespunzători lor sunt identici. În aceasta situatie lista se ramifica pe o a doua directie cu ajutorul celui de-al al doilea pointer la o structură de tip *ListPol*. Muchia comuna in astfel de situatii se spune ca este de tip "atingere" si poate fi interioară (dacă normalele poligoanelor sunt în acelasi sens) sau exterioară (dacă normalele sunt în sensuri opuse).

Structura **CutLine** este necesară în stabilirea liniei de separatie dintre două volume. Ea este folosită la crearea unei liste formate din muchii care delimiteaza două volume.

În afara de câmpul pentru înlăntuire, structura de tip *CutLine* contine următoarele informatii: un pointer la o listă ce contine muchiile ce formează linia de separatie în elemente de tip *IntExtL*; un număr asociat fiecărei linii de separatie, care va fi utilizat în operatia de identificare a volumelor finale; un câmp care indică faptul că linia a fost sau nu tratată.

Structura de date **Class** este folosită la crearea unei liste ce contine intregi care reprezintă numerele de ordine ale liniilor de separatie echivalente. Liniile de separatie determină împărtirea volumului initial în mai multe volume. Se spune că două linii de separatie sunt echivalente dacă pe ele se sprijină poligoane din acelasi volum.

BUPT

La construirea liniilor de separatie, se atribuie fiecăreia un număr de ordine. Initial, numărul corespunzător unei linii va fi asociat tuturor poligoanelor care au cel putin o muchie pe acea linie. În etapa următoare numărul asociat unui poligon va fi atribuit si poligoanelor vecine lui, astfel că, din aproape în aproape, toate poligoanele vor ajunge să fie asociate unei anumite linii de separatie.

Dacă prin acest procedeu de propagare, se întâlnesc două poligoane vecine care au numere diferite, corespunzând la linii de separatie diferite, aceste numere se trec în aceeasi clasă, fiind echivalente. O astfel de listă de numere echivalente va conduce în final la un volum unic.

Structura de date *Class* contine doar un câmp de tip întreg corespunzător unui număr de linie de separatie si un pointer de înlănțuire.

Structura **EchClass** contine pointerul la un tablou cu elemente de tip *Class* si un întreg ce reprezintă dimensiunea acestui tablou.

În final, dimensiunea acestui tablou reprezintă numărul de volume rezultate în urma operatiei booleene efectuate [Mu294].

7.3. IMPLEMENTAREA ALGORITMULUI DE EFECTUARE A OPERATIILOR BOOLEENE INTRE SOLIDE

In continuare se face o descriere a principalelor operatii efectuate de programul ce implementeaza operatiile booleene pentru solide [Mu294].

Odata selectate corpurile operanzi si operatia de efectuat, se considera pe rand câte o pereche de solide. Pentru acesti doi operanzi se completează structurile de date necesare si se stabilesc si bounding-box-ul fiecărei fete si a solidului. Operatia booleană se efectuează numai dacă bounding-box-urile celor două corpuri se intersectează.

Se construieste lista tuturor planelor care contin fetele celor două corpuri si lista tuturor liniilor de intersectie dintre aceste plane. Aceste linii de intersectie vor fi referite si ca linii infinite.

Odată stabilite datele initiale necesare, se trece la urmatoarea etapă a algoritmului, efectuarea intersectiei celor două solide, stabilirea muchiilor si poligoanelor atomice si alegerea acelora care vor face parte din volumele rezultat, în functie de operatie. Dacă operatia este de reuniune se aleg poligoanele *EXTERIOARE* de la ambele corpuri, dacă operatia este de intersectie se aleg poligoanele *INTERIOARE* din ambele corpuri, iar dacă operatia este de diferentă

BUPT

se aleg poligoanele *EXTERIOARE* de la corpul moale si poligoanele *INTERIOARE* de la corpul tare.

Prin operatia de intersectie se obtin segmente de tăietură sau atingere pe poligoane, în functie de pozitia lor fată de poligoanele celuilalt corp. Dacă două poligoane se intersecteaza printr-un segment aflat in interiorul lor, segmentul se numeste de tăietură. Daca poligoanele se intersecteaza de-a lungul unei muchii a unuia sprijinită in interiorul celuilalt, segmentul se numeste de atingere. Obtinerea acestor segmente se face prin intersectia fiecarei linii din lista liniilor de intersectie cu perechi de poligoane din solide diferite situate in planele care au determinat linia de intersectie curentă. Pentru reducerea calculelor, intersectia se face doar daca bounding-box-urile poligoanelor se intersectează.

Intersectia dintre un poligon cu o linie infinită se efectuează prin intersectia cu linia infinită a tuturor muchiile poligonului, inclusiv muchiile de gaură. Punctele de intersectie determină niste segmente, care in functie de pozitia lor relativa la poligoanele pereche considerate, vor fi memorate intr-un anumit tablou din structura *BPoly*.

Cand procesul de intersectie este incheiat, se trece la stabilirea poligoanelor atomice ce se formează tinând seama de muchiile initiale si muchiile noi apărute pe fiecare poligon. În functie de tipul muchiilor care formează fiecare poligon, se stabileste tipul poligonului (*EXTERIOR*, *INTERIOR*, *NEUTRU* sau *GAURA*). Dacă dintr-un poligon initial rezultă mai multe poligoane, se stabilesc caror poligoane li se asociaza găurile.

O atentie specială se acordă cazurilor de suprapunere a doua sau mai multe muchii. In această situatie este util să se cunoască tipul muchiei, din punct de vedere a tipului diedrului format de poligoanele invecinate muchiei. Un diedru ascutitunghic dă tipul muchiei *HILL*, unul obtuzunghic dă tipul *VALLEY*, iar unul de 180 grade dă tipul *FISSURE*.

În cadrul functiilor de tratare a acestor cazuri, în functie de operatie si de pozitia relativa a semiplanelor care formeaza diedrele, se stabileste diedrul sau diedrele care vor face parte din volumul rezultat, si deci elementele diedrelor care trebuie eliminate (ambele muchii si deci toate patru poligoanele ce se sprijina pe muchii sau o muchie si poligoanele corespunzatoare ei sau cate un poligon de la fiecare muchie).

In vederea procesului de selectie a poligoanelor care vor face parte din volumul rezultat, se utilizeaza un algoritm de propagare a tipului poligoanelor. In acelasi scop se efectuează testul de interior, care poate detecta cavitatile.

BUPT

În cazul diferentei a două corpuri, poligoanele din lista de poligoane care au tip *INTERIOR* vor fi inversate, adică li se va inversa sensul muchiilor si al parametrilor ecuatiei planelor in care se gasesc.

Pentru a stabili dacă rezultatul operatiei booleene este reprezentat de unul sau mai multe volume, este necesar să se stabilească existenta sau inexistenta unor linii de separatie. Acestea pot fi formate din secvente de segmente de tăietură, dar si din secvente de segmente de atingere. În afară de aceste linii de separatie se încearcă închiderea unor linii pe muchiile multipluconexe. Dacă există astfel de muchii se generează două linii formate doar din două muchii pentru fiecare muchie de tip multipluconex.

Tot printr-un algoritm de propagare, plecând de la poligoanele care au muchii pe liniile de separatie, se stabileste multimea poligoanelor care formează un volum rezultat. Pentru o astfel de multime, este necesar să se stabilească dacă in acelasi plan există poligoane care au segmente comune, caz in care cu ajutorul unui algoritm de selectie a muchiei celei mai din dreapta, se va forma un nou poligon ca reuniune a poligoanelor initiale. Pentru astfel de cazuri este necesar să se verifice existenta muchiilor coliniare si inlocuirea lor cu o singură muchie.

Ultima operatie ce trebuie efectuată constă in completarea structurilor *Volume*, *Poly* si *Vertex* corespunzătoare volumelor rezultate, pe baza informatiilor continute in structurile *BPoly*, *Muchie* si *PVertex*, ale poligoanelor atomice generate prin algoritmul de efectuare a operatiilor booleene.

7.4. FUNCTIA PRINCIPALĂ CARE IMPLEMENTEAZĂ ALGORITMUL DE EFECTUARE A OPERATIILOR BOOLEENE

Functia **BooleanOp** asigură apelul functiilor care implementează principalele etape ale algoritmului pentru efectuarea operatiilor booleene dintre două solide [Mu294].

Functia primeste ca parametru un tablou cu doi pointeri la structurile *Volume* ale solidelor între care se va efectua operatia booleană. Primul pointer corespunde solidului moale, iar al doilea celui tare.Functia returnează 0 pentru caz de eroare, 1 pentru operatie executată corect sau 2 pentru operatie imposibilă, când solidele nu au puncte comune. Cazul de eroare poate fi stabilit in diferite etape ale algoritmului si de aceea se foloseste variabila globală *bool_err*, a cărei setare va provoca intreruperea executiei programului.

BUPT

Prin apelul functiei InitBool() se initializează lista care va contine liniile de intersectie între plane precum si lista planelor volumelor.

Se apelează apoi functia MakeBPoly() cu argument volumul moale, functie care creează tabloul structurilor de tip *BPoly* corespunzatoare tuturor fetelor volumului. Urmează apelul functiei GetRightPoly() care completează pointerii la poligoanele din stânga si dreapta fiecărei muchii a volumului moale. Poligonul stâng este poligonul care contine muchia respectivă cu conventia parcurgerii conturului fetei cu materialul la stânga, respectiv poligonul drept este poligonul care contine aceeasi muchie însă parcursă în sens invers.

În continuare se completează lista planelor celor două volume, prin apelul functiei MakePlaneList(), precum si lista liniilor de intersectie dintre toate planele celor două volume cu ajutorul functiei MakeLineList().

Se crează lista cu antet *tabpol* care va contine în final toate poligoanele ce vor forma volumul rezultant.

Apelul functiei PolyInter() va rezolva marea majoritate a problemelor legate de intersectia poligoanelor si constructia poligoanelor ce vor forma noul volum. Dacă această functie se termină cu succes, se continuă cu urmatoarele operatii:

Se stabilesc liniile de separatie, contururi închise formate din segmente de tăiere, cu ajutorul functiei CloseInterLine().

Se apelează functia GetDrPoly care determină poligonul drept al muchiilor tuturor poligoanelor de tip *FRONTIERA*.

Functia ExtractPolyPlan() elimină din *tabpol* poligoanele ce nu vor face parte din noul volum.

Se apelează functia RemakePoly() care la nivelul fiecărui plan formează poligoanele atomice si determină care dintre ele sunt găuri si căror poligoane le apartin. Dacă este cazul, se apelează functia Comasare() care formează o singură muchie din mai multe muchii coliniare.

În final, prin functia RemakeVolumes() se construieste noul volum plecând de la poligoanele acestuia, completind structurile de date referitoare la virfurile si fetele solidului rezultat.

BUPT

7.5. FUNCTIILE CARE IMPLEMENTEAZA ALGORITMII DIN CAPITOLUL 6

7.5.1. Functii care implementează algoritmii pentru testele de bounding-box

Functia **Test** stabileste dacă două bounding-box-uri tridimensionale au puncte comune. Dacă acest lucru nu se intimplă, operatiile de intersectie dintre elementele ale căror bounding-box-uri au fost testate, nu mai trebuie efectuate. Functia este utilizată pentru bounding-box-urile solidelor si ale fetelor.

7.5.2. Implementarea algoritmului pentru aflarea unui punct situat pe un segment dat când se cunoaste raportul în care punctul împarte segmentul

Acest algoritm a fost implementat prin intermediul macroinstructiei **GET_POINT** pentru cazul în care se lucrează în spatiul modelului.

Prin argumentele ei, se dau următoarele informatii: numele variabilei de tip *Vector* care va contine coordonatele punctului căutat, segmentul specificat printr-un punct si directia lui, de asemenea, în spatiul modelului, precum si raportul în care punctul căutat trebuie să împartă segmentul, raport exprimat printr-un număr real, subunitar.

Macroinstructia contine operatiile specificate in paragraful 6.2 care efectueaza calculele pentru determinarea coordonatelor spatiale ale punctului căutat.

7.5.3. Implementarea algoritmului pentru calculul distantei de la un punct la o dreaptă

Functia **Dist** implementează algoritmul pentru calculul distantei de la un punct la o dreaptă specificată prin două puncte. Aceste date sunt transmise functiei prin argumentele ei de tip *Vector*. Primul argument corespunde punctului dat prin coordonatele lui spatiale, iar urmatoarele argumente corespund coordonatelor spatiale a doua puncte prin care trece dreapta fata de care se calculeaza distanta.

BUPT

Functia efectueaza calculul distantei prin intermediul relatiilor prezentate in paragraful 6.3.1 pentru cazul lucrului in 3D.

7.5.4. Functia care implementează algoritmul pentru determinarea pozitiei relative a unui punct fată de un poligon dat

Functia **TestOnPoly** primeste ca argumente un pointer la structura corespunzătoare unui poligon si un pointer la structura *Vector* corespunzătoare unui punct. Functia verifică dacă punctul se găseste pe vreuna din muchiile poligonului. Valoarea returnată este pointerul la muchia care contine punctul specificat sau pointerul NULL cand punctul nu se gaseste pe nici o muchie.

Functia **TestOnLineB** stabileste daca punctul se gaseste pe o muchie data. Ea are ca argumente pointerul la structura punctului si pointerul la structura muchiei. Dacă punctul este pe muchie, returnează valoarea 1, altfel valoarea returnată este 0.

7.5.5. Functia care implementează algoritmul pentru determinarea pozitiei relative a două segmente

Functia **Verificare** implementeaza algoritmul prezentat in paragraful 6.4.6 pentru cazul in care cele doua segmente sunt precizate prin coordonate spatiale ale capetelor lor.

Parametrii functiei sunt coordonatele spatiale a patru puncte prin care sunt determinate cele două segmente de dreptă.

Functia determină pozitia relativă a două segmente de dreaptă, mai concret care este pozitia unei muchii fată de o linie de intersectie.

Functia determină distantele de la capetele muchiei (primii doi parametrii) la linia infinită (ultimii doi parametrii) si returneaza o valoare intreaga in functie de pozitia lor relativa. Functia va returna -1 daca segmentele de dreapta nu se intersectează, 0 daca există un punct de intersectie a celor două segmente, 3 dacă cele două segmente se gasesc pe acelasi suport, 1 dacă muchia se srijină cu punctul de început pe linie si 2 dacă muchia se sprijină cu punctul de sfârsit.

BUPT

7.5.6. Implementarea algoritmului pentru determinarea liniei de intersectie dintre doua plane

Functia InterLine are ca parametrii de intrare doi pointeri la elemente din lista *planeb* corespunzătoare la două plane si doi parametrii de iesire elemente de tip *Vector*.

Functia are rolul de a determina linia de intersectie dintre cele două plane care au fost transmise ca parametrii. Linia va fi returnată functiei apelante prin cei doi parametrii de iesire care reprezintă directia ei si un punct prin care trece.

Valoarea returnată de functie este un short care indică dacă operatia de intersectie a decurs normal (valoare returnată 1) sau dacă planele nu au putut fi intersectate (valoare returnată 0).

7.5.7. Implementarea algoritmului pentru selectia unei muchii in procesul de stabilire a conturului unui poligon atomic

Functia **ChooseNext** implementeaza algoritmul pentru selectia unei muchii in procesul de stabilire a conturului unui poligon atomic descris in paragraful 6.6.1.

Functia primeste ca argumente o lista de muchii în care se va insera muchia aleasa ca fiind cea mai la stânga, o lista cu muchiile dintre care se va alege muchia cea mai la stânga, poligonul în cauza si punctele de început si sfârsit ale muchiei fata de care se face alegerea.

7.5.8. Implementarea algoritmului pentru selectia unei muchii in in cazul stabilirii conturului unui poligon obtinut prin reuniunea mai multor poligoane

Functia **ChooseRightNext** implementeaza algoritmul prezentat in paragraful 6.6.2.

Functia alege din mai multe muchii, muchia situată cea mai la dreapta fată de o muchie data. Implementarea este asemanatoare cu cea din functia ChooseNext().

Functia primeste ca argumente o lista de muchii în care se va insera muchia aleasa ca fiind cea mai la dreapta, o lista cu muchiile dintre care se va alege muchia cea mai la dreapta, poligonul în cauza si punctele de început si sfârsit ale muchiei fata de care se face alegerea.

BUPT

7.5.9. Implementarea algoritmului pentru determinarea pozitiei relative a unui punct fata de un poligon

Functia **PointInter** implementeaza algoritmul descris in paragraful 6.7.1, pentru cazul in care punctul si poligonul sunt specificate in spatiul 3D.

Functia primeste ca argumente pointerul la structura *BPoly* corespunzatoare unei fete si pointerul la structura unui punct si testează dacă punctul respectiv se găseste în interiorul poligonului.

Dacă punctul se găseste în interiorul poligonului se returnează valoarea 1, iar dacă punctul se găseste în exterior se returnează valoarea 0.

7.5.10. Implementarea algoritmului de clasificare

Functiile **InfIntersect** si **ProcessList** implementeaza algoritmul de clasificare prezentat in paragraful 6.12. Prima functie stabileste punctele de intersectie dintre o linie infinita si doua poligoane si pozitia relativa a fiecarui segment al liniei fata de fiecare poligon, iar a doua functie stabileste pozitia relativa a fiecarui segment fata de ambele poligoane.

Functia InfIntersect inserează în lista punctelor de intersectie informatia referitoare la punctul de intersectie dintre o muchie si o linie de intersectie, in cazul in care acesta exista.

Parametrii functiei sunt: pointerii la structura muchiei si liniei intre care se cauta punctul de intersectie; pointerul la structura corespunzătoare poligonului din care face parte muchia; pointerul la lista *l* in care se memoreaza punctele de intersectie; coordonata dupa care se ordoneaza punctele pe linia de intersectie; sensul liniei infinite în raport cu coordonata de ordonare; un fanion prin care se precizează dacă muchia face parte din volumul moale sau tare; un punctul P de pe linia infinită.

Tipul de intersectie este furnizat de functia Verificare().

Dacă muchia nu se intersectează cu linia infinită, se părăseste functia.

Pentru celelalte situatii se determină distantele $d1$ si $d2$ de la extremele muchiei la linia infinită. Dacă cele două drepte sunt suprapuse, distantele sunt nule. Pentru cazul de intersectie, distantele sunt calculate cu functia Dist().

În functie de aceste distante apar mai multe cazuri distincte.

1. Dacă muchia se sprijină cu punctul de început pe linia infinită ($d1 = 0$, $d2 \neq 0$) se executa urmatoarea secventa de operatii:

BUPT

Se determină muchia anterioara muchiei în discutie cu functia GetPrevEdge() si se testează pozitia acestei muchii fată de linia infinită si fată de muchia în discutie. Pentru aceasta se calculează produsul mixt între linia de intersectie, muchia găsită si normala planului care le contine, respectiv produsul mixt dintre linia de intersectie, muchia în discutie si normala planului. Produsul mixt este calculat cu ajutorul macroinstructiei MIXTPRD. Dependent de valorile celor două produse apar mai multe cazuri.

Dacă muchia anterioara muchiei curente este de-a lungul liniei de intersectie (primul produs mixt este zero) si dacă muchia care se sprijină pe linia de intersectie nu este nici prima, nici ultima muchie din lista de muchii a poligonului, se trece la urmatoarea muchie din lista si se revine din functie.

Dacă cele două produse mixte au acelasii semn, ceea ce înseamna ca cele două muchii ale poligonului se află de o parte si de alta a liniei infinite si dacă sunt si negative atunci linia de intersectie intra în poligon (se pozitionează variabila *inout* pe *IN*), iar dacă sunt pozitive linia de intersectie iese din poligon (*inout* = *OUTB*).

Dacă însă cele două produse au semne contrare, deci cele două muchii se află de aceeasi parte a liniei infinite si se sprijină cu punctul lor comun pe aceasta, se calculează produsul mixt si între cele două muchii si normala planului în care se află. Dacă produsul este negativ, *inout* este pozitionat pe *IN* (linia infinită se află în interiorul poligonului, fiind în partea stânga a celor două muchii după sensul de parcurgere al acestora), iar dacă este pozitiv *inout* = *OUTB* (linia de taiere este în exteriorul poligonului, acesta sprijinîndu-se cu un vârf pe ea).

Ultima operatie o constitue inserarea în lista punctelor de intersectie a punctului în care se sprijină muchia testată pe linia de intersectie. Inserarea se face cu functia InsLBEdge().

2. Dacă ambele distante *d1* si *d2* sunt diferite de zero, caz în care punctul de intersectie se află între punctele P1 si P2, se procedează în modul urmator:

Se calculează produsul mixt între linia infinită, muchia în discutie si normala planului ce contine aceste drepte. Dacă produsul este negativ înseamna ca linia infinită intra în poligon taind muchia în discutie. În caz contrar linia infinită iese din poligon. Dacă muchia nu este segmentată, coordonatele punctului de intersectie se determină cu functia GET_POINT(), iar apoi punctul va fi inserat în lista de puncte *I* corespunzătoare liniei infinte. În schimb, dacă muchia este segmentată, este necesar să stabilim care este segmentul pe care va fi pozitionat noul punct de intersectie. Pentru aceasta se ia pe rând fiecare segment al muchiei din tabloul indicat de câmpul *cedge* al muchiei si se testează pozitia lui fată de linia infinită folosind functia Verificare(). Apar din nou mai multe situatii:

BUPT

segmentul se sprijină cu unul din vârfuri pe linie, caz în care functia InsLBEdge() va primi ca parametru vârful corespunzător; capatele segmentului se află de o parte si de alta aliniei infinite, deci punctul de intersectie este în interiorul segmentului. Pentru această situatie se calculează distantele de la capetele segmentului la linia infinită cu functia Dist() si apoi se determină punctul de intersectie cu GET_POINT(). Acesta apoi este inclus în lista *l*.

3. Dacă ambele distante *d1* si *d2* sunt nule, ceea ce înseamna ca muchia se suprapune peste linia infinită, câmpul *li* al muchiei va primi valoarea pointerului la linia infinită. Se indică în acest mod ca muchia este suprapusa peste o linie de intersectie.

Dacă sensul muchiei este contrar sensului liniei infinite, se cauta muchia care o precede pe aceasta folosind functia GetPrevEdge() si se determină pozitia muchiei găsite în raport cu linia infinită. Pentru aceasta se face produsul mixt între directia liniei infinite, muchie si normala planului care le contine. Dacă produsul este negativ, linia infinită intra în interiorul poligonului (este în partea stânga a celor două muchii) si inout primeste valoarea *IN*. În caz contrar linia se află în exteriorul poligonului, acesta s-a sprijinit doar cu o muchie pe linie (*inout* = *OUTB*). Apoi, atit punctul de început cât si cel de sfârsit al muchiei sunt inserate în lista de puncte *l* corespunzătoare liniei de intersectie.

Dacă sensul muchiei este identic cu cel al liniei, se cauta urmatoarea muchie din contur cu functia GetNEdge(). Apoi, în mod similar situatiei anterioare, se cerceteazã pozitia noii muchii găsite în raport cu linia de taiere.

Punctele de început si de sfârsit ale muchiei vor fi inserate în lista *l* a liniei infinite. La inserarea punctelor în lista *l* se precizează ca ele sunt capetele unei muchii ce se suprapune peste linia infinită. Acest lucru se face pozitionînd pe *ON* un parametru al functiei InsLBEdge(). Functia primeste tot ca parametru si variabila inout setată de catre functia InfIntersect().

Functia **ProcessList** determină toate segmentele comune rezultate în urma intersectiei dintre două poligoane si le inserează în tabelele de muchii ale poligoanelor.

Parametrii functiei sunt pointerii la cele două poligoane care au fost intersectate si pointerul la lista ce contine punctele de intersectie.

Dacă primul element din lista nu corespunde unui punct care apartine ambelor poligoane, adică nu are completate ambele câmpuri *po1* si *po2*, se parcurge lista până când se va găsi un punct care să apartină de un poligon, altul decât cel în care este inclus primul punct din lista sau până este găsit un punct care să apartină ambelor poligoane. În cazul în care nici un punct de intersectie nu

BUPT

îndeplineste una din aceste conditii, se revine în functia apelanta, deoarece poligoanele nu au segmente comune. Se verifica în acest mod dacă pe linia de intersectie există puncte din ambele poligoane.

Dacă una din conditii a fost verificată, se parcurge din nou lista de la început. Dacă primul punct nu apartine ambelor poligoane, se stabileste poligonul în care este inclus. Dacă apartine poligonului din volumul moale, se cauta în lista un punct care face parte din cel de-al doilea poligon. În cazul în care nu există nici un punct de intersectie din poligonul al doilea se părăseste functia. Analog se tratează si situatia în care primul punct din lista apartine volumui tare.

Dacă însă punctele de pe linia infinită sunt grupate unele dintr-un poligon si altele din celalalt, înseamna ca poligoanele nu se intersectează si deci procesarea listei nu mai are rost.

Urmatorul pas îl constitue verificarea existentei unei muchii a poligonului moale pe linia de intersectie. Pentru aceasta se testează câmpul *on1* al fiecărui punct. În cazul în care există o astfel de muchie, pentru toate punctele de intersectie situate în interiorul muchiei, câmpul *on1* va primi valoarea câmpului *on1* al punctului din capatul muchiei (valoarea *ON*).

Se parcurge din nou lista pentru a stabili care segmente de pe linia de intersectie se află în interiorul poligonului moale. Pentru toate punctele situate între punctul care are câmpul *inout1* pozitionat pe *IN* (punctul în care linia de intersectie patrunde în poligonul moale) si punctul al cărui câmp *inout1* are valoarea *OUTB* (punctul în care linia de intersectie iese din poligonul moale), se pozitionează câmpul *inout1* pe *IN*. Se marchează astfel toate segmentele din linia infinită care sunt incluse în poligonul moale.

Ultimele două operatii se repeta si pentru cel de-al doilea poligon, poligon ce apartine volumului tare. Se găsesc toate muchiile care sunt suprapuse peste linia de taiere precum si toate segmentele din linia de intersectie care sunt în interirul poligonului.

Se cauta apoi în lista punctele care se află atit în interiorul poligonului moale cât si în interiorul poligonului tare (puncte pentru care câmpurile *inout1* si *inout2* au valoarea *IN*). Primul punct ce îndeplineste aceste conditii este punctul de început al unui segment de intersectie dintre cele două poligoane. Segmentul determinat de acest punct si de punctul care îi succede în lista, va fi inserat în ambele poligoane. Inserarea este realizată cu ajutorul functiei CalcIntExt(). Parametrii cu care este apelată functia difera în functie de faptul ca punctul în discutie apartine sau nu unei muchii de frontiera (situată de-a lungul liniei de intersectie). Vor exista trei moduri de apel al functiei:

BUPT

1. pentru cazul când punctul apartine si unei muchii de frontiera din volumul moale;

2.pentru cazul când punctul apartine unei muchia de frontiera din volumul tare;

3.cazul când punctul nu apartine nici unei muchii de frontiera.

Procesul continua căutîndu-se în lista *l* alte segmente de intersectie dintre poligoane. Operatia ia sfârsit când toate aceste segmente au fost incluse în poligoanele în care au fost create.

Dacă în lista există puncte cu câmpurile *inout1*, *inout2*, *on1* si *on2* setate, se apelează functia CutSupr(). Aceasta determină pozitia relativa a celor două muchii suprapuse si le taie în functie de caz [Sav94] [Mu194] [Mu294] [Mu394].

7.6. FUNCTIILE CARE IMPLEMENTEAZĂ CREAREA STRUCTURILOR DE DATE NECESARE PENTRU EFECTUAREA OPERATIILOR BOOLEENE

Functia InitBool

Functia InitBool() crează lista cu antet *linii*, care va contine informatiile referitoare la liniile de intersectie dintre plane si lista cu antet *planeb*, care va contine informatiile referitoare la toate planele care contin fetele solidelor.

Variabila externă *l_linii* indică ultimul element al listei liniilor de intersectie, iar variabila externă *l_planeb*, indică ultimul element din lista *planeb*.

Functia MakeBPoly

Functia are rolul de a completa structurile de date ce trebuie sa contina informatia referitoare la fetele unui volum, in vederea efectuarii operatiilor booleene. Aceste date necesare apar ca si câmpuri în structura *BPoly* care va fi asociată fiecărui poligon al volumului.

Functia primeste ca argument un pointer la un volum. Pentru acest volum se completează câmpul *polygon* care va indica un tablou cu elemente de tip *BPoly*, fiecare element corespunzând câte unui poligon al volumului. Pentru fiecare element de tip *BPoly* se vor completa câmpurile *bedge* si *bbox*. Pointerul *bedge* va indica un tablou cu elemente de tip *Muchie*, iar câmpul *bbox* va

BUPT

reprezenta dimensiunea acestui tablou. Tabloul va păstra muchiile poligonului în ordinea în care apar plecând din vârful indicat de câmpul *f_point* al structurii *Poly* corespunzătoare poligonului curent.

Functia începe prin a verifica dacă câmpul *polygon* al volumului este completat sau nu. În caz afirmativ se paraseste functia, deoarece volumul a mai intervenit într-o operatie booleană anterioară si informatia referitoare la poligoanele lui a fost deja completată.

Se parcurg toate poligoanele volumului si pentru fiecare poligon se insereaza un nou element în tabloul indicat de câmpul *polygon* al volumului, apelând functia InsertPolyB() si se modifica campul *nrp* care indica dimensiunea tabloului. Se parcurge fiecare contur al poligonului (contur propriu-zis sau contur de gaură), determinand muchiile care il formeaza si completand cite o structura de tip *Muchie* pentru fiecare din ele. Dacă un contur este de tip *GAURA*, se insereaza un nou element in tabloul indicat de cimpul *găuri* din structura *BPoly*, si se incrementează contorul care indică numărul găurilor (câmpul *hbox*).

Aceasta functie utilizeaza o serie de functii care efectueaza gestiunea datelor pentru efectuarea operatiilor de inserare a noilor elemente, cautari in tablouri sau liste in vederea completarii diferitelor informatii necesare.

Functia MakePlaneList

Functia primeste ca parametri pointerii la cele două volume operanzi ai operatiei booleane.

Functia are drept scop introducerea în lista *planeb* a câte unui element corespunzător fiecarui plan in care se gaseste cel putin o fată a solidelor operanzi.

Daca la un moment dat se doreste introducerea unui element corespunzator unui poligon al carui plan care exista deja in lista, se efectueaza doar completarea elementului respectiv cu informatia referitoare la poligonul curent.

Se parcurge tabloul indicat de câmpul *poligon* al structurii *Volume* corespunzătoare celor două volume date ca parametrii. Pentru fiecare element al tabloului care corespunde unui poligon al volumului respectiv se apelează functia InsertPlan().

Functia MakeLineList

Functia completează lista liniilor de intersectie dintre toate planele ce contin fete ale celor două volume.

BUPT

Se ia pe rând fiecare plan din lista *planeb*. Se determină liniile de intersectie dintre acest plan si toate celelalte plane din lista *planeb*, aflate însă în listă după planul curent. Acest lucru are drept scop evitarea efectuarii intersectiei între două plane deja intersectate. Pentru determinarea liniei de intersectie între două plane se apelează functia GetLine().

Functia nu returnează nimic, deoarece rezultatul executiei ei se va regăsi în lista globală de linii unde se vor afla toate liniile de intersectie între plane.

Functia GetLine

Functia primeste ca parametri doi pointeri la cele două plane între care se doreste să se găseasca linia de intersectie.

Prima etapă constă în testarea normalelor planelor pentru a se vedea dacă planele nu sunt paralele. Dacă sunt paralele, functia îsi încheie executia. In caz contrar se apelează functia PlanInter() care va determina linia de intersectie dintre plane obtinuta prin functia InterLine() si va verifica daca in lista exista sau nu un element cu aceleasi caracteristici. In primul caz se actualizeaza informatia relativa la planele care prin intersectie genereaza linia, iar in al doilea se insereaza in lista liniilor de intersectie un nou element corespunzator liniei gasite.

7.7. FUNCTIILE CARE IMPLEMENTEAZA OPERATIILE DE INTERSECTIE DINTRE CELE DOUA SOLIDE

Intersectia dintre cele două volume se face la nivel de poligoane. Poligoanele intersectate la un moment dat se aleg din tabelele de poligoane atasate la două plane care se intersectează. Practic se efectuează intersectia dintre un poligon si linia de intersectie dintre planele ce contin cele două poligoane. După ce punctele de intersectie au fost găsite si au fost memorate în ordine într-o structura de date specială, se stabilesc segmentele situate în interiorul poligoanelor si tipul lor (segmente pe care poligoanele se intersectează sau se ating).

Functia principală care realizează intersectiile, PolyInter(), împreună cu alte functii necesare în acest proces sunt prezentate în continuare.

BUPT

Functia PolyInter

Această functie efectuează intersectiile dintre elementele componente ale celor două solide, inserând informatia în structurile de date corespunzătoare.

Functia are ca parametrii de intrare pointerii la structurile corespunzătoare celor două volume care intervin în operatia booleană selectată. Prin intermediul primului parametru de iesire, functia returnează pointerul la lista *tabpol*, iar prin al doilea parametru de iesire semnalează dacă operatia booleană se poate face sau nu. Functia semnalează prin intermediul valorii întregi returnate aparitia unei eventuale erori.

Pentru efectuarea intersectiei dintre cele două solide, se parcurge lista liniilor de intersectie dintre plane si pentru fiecare linie se iau pe rând toate planele asociate ei. Fiecare poligon dintr-un asemenea plan se intersectează cu toate poligoanele celorlalte plane asociate liniei de intersectie curente, cu conditia ca ele să nu fie din acelasi solid.

Pentru fiecare pereche de poligoane ce trebuie intersectate se verifică dacă bounding-box-urile lor se intersectează, folosind functia Test(). Dacă intersectia este posibilă, se apelează functia PolyLiInt() care va realiza intersectia efectivă dintre linia de intersectie si cele două poligoane. Ca urmare a acestei operatii, unele structuri de tip *BPoly* asociate fetelor solidelor vor contine noi muchii. Este apelată apoi functia Remake(), care formează noile poligoane atomice ce au la baza muchiile initiale si muchiile noi rezultate prin intersectie, poligoane pe care le introduce în lista *tabpol*.

Pentru fiecare muchie a fiecăruia dintre poligoanele nou create se determina poligonul din stânga si cel din dreapta muchiei, folosind functia GetRPolyTab() si tipul ei (*VALLEY*, *FISSURE* sau *HILL*) folosind functia SetEdgeType().

Urmatorul pas îl constitue apelul functiei ReconstrLinie() care insearează intr-un tablou asociat fiecarei linii infinite de intersectie, muchiile din poligoanele nou create care se suprapun pe ea, eliminind acele linii pe care nu se gaseste nici o muchie.

Functia ProcessListGlobal() analizează două cate două muchiile suprapuse, in scopul determinarii pozitiei relative a poligoanelor din care fac parte. În functie de aceasta precum si de operatia booleana ceruta (intersectie, reuniune sau diferenta) se determină care din poligoanele celor două diedre vor fi eliminate si care vor fi păstrate în structura finală a volumelor rezultate.

BUPT

Muchiile care formeaza un singur diedru sunt luate in considerare in functia ProListGlobal(), care are drept scop stabilirea tipului unui poligon vecin al muchiei in functie de tipul cunoscut al celuilalt.

In functie de tipul poligoanelor si de operatia booleana ceruta, functia ChoosePolygon() stabileste care din poligoane vor fi eliminate din lista poligoanelor finale. În lista *tabpol* vor fi păstrate doar acele poligoane din care se vor construi volumele rezultate.

Pentru a stabili tipul poligoanelor rămase fără tip, se verifică dacă acestea sunt plasate în interiorul celuilalt volum, prin functia IsCavitate(). In functie de operatia booleana curenta si tipul unor poligoane, se incearca stabilirea tipului celor ramase fara tip.

În cazul în care operatia booleeana aleasa este diferenta, functia InverseSenseB() schimba sensul de parcurgere a muchiilor poligoanelor de tip *INTERIOR*, atit pentru contururi cât si pentru gaurile asociate lor.

Functia PolyLiInt

Aceasta functie este apelata de PolyInter() in vederea efectuarii intersectiei dintre doua poligoane si linia de intersectia dintre planele in care se gasesc.

Parametrii functiei sunt: pointerul la structura unei linii de intersectie si pointerii la structurile corespunzătoare a două poligoane.

Functia returnează un număr întreg care semnalează functiei apelante dacă operatia de intersectie a reusit sau nu.

Se intersecteza linia dată ca parametru cu muchiile fiecarui poligon, inclusiv muchiile ce formeaza conturul gaurilor, apelând functia InfIntersect(). Aceasta va determina punctele de intersectie dacă există sau va stabili dacă exista muchii care se suprapun pe linia infinită. Punctele de intersectie gasite se inserează ordonat într-o lista *l* cu elemente de tip *LBEdge*.

Pasul urmator îl reprezinta prelucrarea informatiei continute în lista *l*. Aceasta este realizată de functia ProcessList(). Se determina toate segmentele de pe linia infinită care apartin ambelor poligoane. Aceste segmente vor fi inserate în listele de segmente ale fiecărui poligon.

Dacă s-a depistat vreo eroare, functia PolyLiInt() returnează 0, iar în caz contrar returnează 1. Eroarea este marcată prin valoarea variabilei *bool_err*.

BUPT

Functia IsCavitate

Functia primeste ca argumente un pointer la un poligon, doi pointeri la volume si un pointer la un element de tipul *PVertex*.

Functia testează dacă poligonul primit ca argument se găseste în interiorul unui alt volum decât volumul din care face parte.

Se aproximează un punct de la infinit, alegându-l în afara bounding-box-urile celor două volume.

Se ia pe rând fiecare plan din lista de plane *planeb* diferit de planul poligonului specificat si din acest plan se iau în discutie toate poligoanele care nu apartin volumului poligonului care se testează.

Punctul de la infinit, împreuna cu punctul primit ca argument (care este punctul de început al primei muchii care apartine poligonului testat) formează o semidreapta. Se intersectează această semidreapta cu planul curent din lista *planeb*. Această intersectie se realizează cu ajutorul functiei InterSemidrPlan(). Dacă valoarea *is* returnată de această functie este 0, se trece la urmatorul plan din *planeb*, deoarece semidreapta nu poate intersecta acest plan.

Dacă *is* are valoarea -1, se schimba panta semidreptei prin modificarea coordonatelor punctului de la infinit si se reia testul pentru toate planele din lista *planeb*.

Dacă punctul de început al semidreptei este continut în planul curent din *planeb*, se alege prin intermediul functiei NextPoint() un alt punct din poligonul pentru care se face testul de cavitate si se reia testul cu toate planele din *planeb*.

Dacă functia NextPoint() nu returnează un punct, se alege un punct din interiorul poligonului si se face acelasi test pentru acest punct. Dacă nici pentru acesta nu s-a ajuns la nici o concluzie, se verifica dacă poligonul în discutie are vreun punct comun cu vreun alt poligon din acelasi plan, si dacă are, atunci i se pune tipul *NEUTRU*. Altfel ramine *EXTERIOR*. În acest punct de program, functia îsi încheie executia si returnează valoarea 0, deoarece poligonul nu este cavitate.

Dacă valoarea returnată de functia InterSemidrPlan() este 1, se incrementează contorul care păstrează numărul de intersectii ale semidreptei cu poligoanele apartinând planelor testate si se trece la urmatorul plan din *planeb*.

După ce au fost analizate toate planele din *planeb*, se testează dacă numărul de intersectii este un număr par, caz în care functia returnează valoarea 0 (nu este cavitate). Dacă acest număr este impar se returnează 1 (poligonul este cavitate).

BUPT

Functia ProcessListGlobal

Functia primeste ca parametrii un pointer la structura de date a unei linii de intersectie, pointerul la lista poligoanelor nou create, precum si pointerii la structurile celor două volume supuse operatiei booleene.

Functia este apelată pentru fiecare linie de intersectie în parte.

Se parcurg pe rând toate tabelele de muchii asociate liniei date, pentru a lua în considerare muchiile suprapuse. În cadrul aceluiasi tablou, se tratează două câte două din muchiile lui. Dacă două astfel de muchii fac parte din volume diferite, se apelează functia CazMuchieMuchie(). În functie de pozitia relativa a poligoanelor adiacente celor două muchii si de operatia booleana ceruta, această functie apelează la rândul ei functia de tratare a cazului particular corespunzător. Dacă cele două muchii nu sunt din volume diferite, sunt marcate ca fiind de tip multiplu conex.

La încheierea tratării perechilor de muchii dintr-un tablou, se verifica existenta muchiilor multiconexe. Pentru acestea se setează variabila multiplucon pe 1.

7.8. FUNCTII PENTRU TRATAREA CAZURILOR PARTICULARE MUCHIE/MUCHIE

Aceste functii folosesc notiunile si notatiile prezentate in paragraful 5.4 si in plus se trateaza in mod distinct cazul in care unul din diedrii sau ambii sunt de π rad (tip *FISSURE*).

Functiile efectuază prelucrările corespunzătoare operatiilor booleene, asupra celor doi diedrii aflati în discutie. Datorita formelor pe care le pot lua cei doi diedrii (*HILL*, *VALLEY* sau *FISSURE*) pot să apară noua situatii distincte (*HILL-HILL*, *HILL-VALLEY*, etc), fiecare fiind rezolvată de catre o functie TratareCaz speciala.

Fiecare functie TratareCaz ia în considerare o anumita formă particulară a celor doi diedrii (în numele functiei se precizează care este aceasta).

Functia analizează pe rând toate toate cazurile posibile de pozitionare a celor doi diedrii unul fată de celalalt. Aceste cazuri au fost stabilite anterior cu functia DetCaz(), obtinându-se 13 posibilităti. Pentru fiecare caz, se cercetează mai întii tipul poligoanelor care definesc cei doi diedrii. Dacă există poligoane al căror tip nu este precizat (*NEUTRU*) se încearcă găsirea tipului lor tinînd seama de tipul celorlalte poligoane din diedre. Dependent de operatia booleană cerută,

BUPT

va fi executată o anumita secventă de instructii. Fiind cunoscute în acest moment forma diedrelor si pozitia lor relativă, în functie de operatia booleeană cerută se determină care din poligoanele celor două diedre vor fi eliminate pentru că nu vor face parte din volumele rezultate. (Felul în care se procedează în fiecare caz este prezentat în tabelele de modelare matematica a relatiilor muchie-muchie). Eliminarea efectiva din lista poligoanelor noi este realizată de functia EliminPoly().

În urma acestor operatii apar situatii care necesită modificarea tipurilor muchiilor ce stau la baza celor două diedre (muchiile pot deveni de tipul *FISSURE*), eliminarea lor totala în cazul în care poligoanele lor stâng si drept au fost eliminate, apare posibilitatea ca acestea să devină multipluconexe (vor apartine la mai mult de două poligoane) sau se va forma un nou diedru (poligoanele din dreapta si stânga muchiei vor fi schimbate). Prelucrarile efectuate asupra muchiilor se vor resfrînge si asupra muchiilor identice cu acestea si care sunt situate în poligoanele din dreapta muchiilor în discutie. Obtinerea acestor muchii se face la începutul functiilor cu ajutorul lui functiei SearchREdge(). Indicatorul flagdraw al muchiilor supuse prelucrarilor va fi pozitionat la valoarea 1 precizînd prin acesta că muchia a fost luată în considerare.

Functia MuchieSuprap

Functia primeste ca parametrii pointerii la poligoanele ce urmează a fi intersectate precum si pointerul la linia infinită după care se realizează intersectia dintre planele ce contin cele două poligoane. Cele două poligoane apartin aceluiasi volum.

Functia parcurge lista de muchii de pe conturul primului poligon si pentru fiecare muchie care nu are actualizat câmpul *li*, se cercetează dacă punctele de început si de sfârsit ale muchiei apartin si celui de-al doilea plan. Dacă coordonatele celor două puncte verifica ecuatia planului al doilea, înseamna ca muchia apartine si acestui plan, deci ea nu se poate afla decât pe linia de intersectie dintre cele două plane. În acest caz câmpul *li* al muchiei va primi ca valoare adresa structurii de date corespunzătoare liniei de intersectie.

Aceeasi verificare se face si pentru muchiile care formează contururile gaurilor poligonului, în cazul în care acesta contine gauri.

BUPT

Functia SetEdgeType

Functia primeste ca parametru pointerul la lista de poligoane nou create.

Functia returnează un număr întreg care semnalează succesul sau esecul operatiilor efectuate.

Prin intermediul acestei functii muchiilor de tip *UNDEF* li se dă unul din tipurile *VALLEY*, *FISSURE* sau *HILL*.

Se parcurge lista de poligoane ce a fost primita ca parametru si se considera pe rând fiecare poligon. Interesează doar muchiile din conturul poligonului si din contururile gaurilor care sunt de tip *UNDEF*. În acest caz se calculează produsul mixt între normala poligonului de care apartine muchia, normala poligonului aflat în dreapta muchiei si vectorul muchiei respective. Se determină în poligonul din dreapta muchia opusa apelând functia SearchRedge(). Muchiei si opusei ei li se completează tipul în functie de valoarea produsului mixt. Dacă produsul mixt este negativ, tipul muchiilor este *VALLEY*. Dacă este pozitiv muchiile vor fi de tip *HILL*, iar dacă produsul este zero atunci muchiile sunt de tip *FISSURE*.

Dacă vreunei muchii nu i s-a putut stabili muchia opusa, functia returnează valoarea 0. Dacă operatiile s-au executat corect functia returnează 1.

Functia CazMuchieMuchie

Parametrii functiei sunt pointerii la structurile de date a două muchii si pointerul la lista poligoanelor nou create.

Functia returnează un număr întreg pentru a semnala succesul sau esecul operatiilor efectuate.

Functia stabileste cazul particular corespunzător pozitiei relative a poligoanelor adiacente muchiilor si apelează functia corespunzătoare care să stabileasca în functie de operatia booleana ceruta, care muchii si care poligoane vor ramane în componenta volumelor rezultate.

Este necesar să se creeze o lista ordonată a celor patru poligoane adiacente muchiilor, conformă cu pozitia lor în spatiu. Se stabileste dacă între cele patru poligoane există suprapuneri si dacă da, câte din normalele aflate pe aceeasi directie au acelasii sens si câte au sens opus.

Pentru aceasta se calculează produsul vectorial între normalele planelor ce contin cele două muchii. Dacă produsul este zero, adică cele două normale au aceeasi directie, se va calcula si produsul lor scalar pentru a se stabili dacă normalele sunt de acelasii sens sau nu. Deoarece interesează care poligoane

BUPT

apartin aceluiasi plan, se va folosi un tablou de patru întregi, vpr, ale cărui elemente iau valoarea 1 dacă poligoanele corespunzătoare sunt coplanare si valoarea 0 în caz contrar. Elementele tabloului corespund în ordine perechilor: (*SM*, *ST*), (*SM*, *DT*), (*DM*, *ST*), (*DM*, *DT*). Numărul poligoanelor cu normalele în acelasii sens este păstrat în variabila *q1*, iar numărul poligoanelor cu normale în sens contrar în *q2*.

Se apelează functia OrdonareMuchii() care foloseste aceste variabile pentru a crea lista ordonată a poligoanelor. Functia atasează câte un vector fiecărui poligon ce formează diedrele si îi inserează în lista *edg* în ordinea în care sunt plasati în spatiu. În functie de tipul celor două muchii apar nouă situatii distincte, rezultate din combinatiile posibile ale tipurilor celor două muchii, fiecare muchie putînd avea unul din tipurile *HILL*, *VALLEY* sau *FISSURE*. În functie de aceasta, variabila *hv* va primi unul din tipurile *HH*, *HV*, *VH*, *VV*, *HF*, *FH*, *VF*, *FV* sau *FF*.

Dacă numărul de elemente din lista *edg* este patru, deci poligoanele se află în plane distincte se apelează functia DetOrdabcd(). Această functie determină pozitia în care se află cele două diedre unul în raport cu celalalt (vezi figura 5.13). Pentru fiecare din cele nouă situatii pot apare patru cazuri: diedrul moale include în interiorul său diedrul tare, cazul invers, diedrele sunt exterioare având doar muchia comună si situatia în care diedrele se intersectează (au o portiune comuna). Dacă DetOrdabcd() returnează -1 înseamnă că a fost detectată o eroare si aceasta duce la întreruperea functiei.

Pe lînga cele patru cazuri generale, pentru fiecare din cele noua situatii, pot apare si cazuri particulare datorita faptului ca două poligoane în diedre diferite se pot afla în acelasii plan. Astfel numărul total de cazuri ce pot apare este 13. Identificarea cazului se realizează cu ajutorul functiei DetCaz() care primeste ca parametrii variabilele *q1*, *q2* si ord determinate anterior precum si numărul de elemnte din lista *edg* si forma particulara pe care o au cei doi diedrii (exprimată prin variabila *hv*).

Este eliberată memoria alocată pentru lista *edg* (functia FreeEdg), iar dacă functia DetCaz() returnează valoarea -1, ceea ce înseamnă că a aparut o eroare, se va întrerupe executia functiei.

În functie de valoarea variabilei *hv* (forma pe care o au cei doi diedrii) se va apela una din cele noua functii de mai jos: TratareCazHH, TratareCazHV, TratareCazVH, TratareCazVV, TratareCazHF, TratareCazFH, TratareCazVF, TratareCazFV, TratareCazFF, functia primind ca parametru si o valoare prin care se indică pozitia relativa a celor doi diedri. În functie de aceste date si tinîndu-se

BUPT

cont si de operatia booleeana ce se doreste a fi realizată, urmează tratarea cazurilor.

În final functia va returna valoarea 1 dacă operatiile au fost executate corect.

Functia OrdonareMuchii

Parametrii de intrare ai functiei sunt pointerul la lista *edg* cu elemente de tip *ListPol*, pointerii la structurile a două muchii, *m1* si *m2*, un tablou de întregi, *vpr* si doi întregi *q1* si *q2*.

Functia stabileste ordinea în care se succed poligoanele ce se sprijină pe muchiile date ca parametrii, începând cu poligonul drept al muchiei din volumul moale si mergând în sens invers trigonometric. În final returnează numărul elementelor din lista.

În lista vor exista elemente distincte pentru poligoane aflate în plane distincte. Două poligoane situate în acelasii plan vor forma o sublista al cărui cap este un element din lista *edg*.

Functia calculează la început vectorii pe care îi asociaza fiecărui poligon, vectori obtinuti prin produsul vectorial dintre muchie si normala poligoanelor care se sprijină pe ea (vezi figura 5.4).

Primul element inserat în lista este cel corespunzător poligonului din dreapta al muchiei din volumul moale. Se alocă apoi o zona de memorie pentru un element de tip *ListPol* în care se memorează pointerul la poligonul aflat în dreapta primei muchii, vectorul asociat acestui poligon si valoarea *DM* (dreapta moale), după care elementul este inserat în lista *edg*.

Operatiile continua diferentiat în functie de numărul poligoanelor aflate în acelasii plan, informatii transmise prin parametrii *vpr*, *q1* si *q2*.

Dacă poligoanele *SM* si *ST* se află în acelasii plan, este apelată functia InsertListPol() pentru a crea un element de tip *ListPol* corespunzător primului poligon. Acest element se inserează în lista după elementul corespunzător lui *DM*. În lista deci vor fi memorate toate informatiile legate de diedrul moale. Este apelată apoi functia InsertListPol pentru crearea elementului corepunzator poligonului stâng din diedrul tare (*ST*). Se stabileste dacă normalele acestor poligoane sunt de acelasii sens calculând produsul scalar între vectorii asociati poligoanelor. Dacă produsul este pozitiv, cele două poligoane sunt de tip frontiera si al doilea element va fi inserat în sublista primului, folosind câmpul ndown al nodului *SM*. Tipul poligoanelor este setat la valoarea *FRONTIERA* apelând functia SetEdg().

BUPT

Într-un tablou de întregi numit *place* se memorează poligoanele pentru care a fost creat un element în lista *edg*.

Dacă poligonul *SM* si *DT* se află în acelasii plan se crează la fel ca si în cazul anterior elementul de tip ListPol corespunzător primului poligon, dacă acesta nu a fost creat deja. În situatia în care acesta există în lista *edg*, el este căutat cu ajutorul functiei SearchInPlace(). Se crează apoi elementul corespunzător poligonului *DT*. În functie de valoarea produsului scalar dintre vectorii corespunzători celor două poligoane, se face inserarea în lista ca în cazul anterior, cu completarea tipului *FRONTIERA*.

Procedeul se repetă si pentru perechile (*DM*, *ST*) si (*DM*, *DT*).

Dacă în urma operatiilor anterioare în lista nu apar toate elementele corespunzătoare celor patru poligoane, acestea sunt incluse în lista *edg* la locul potrivit. Mai întii se va crea pentru poligonul în cauza un element de tip ListPol cu functia InsertListPol, iar apoi noul element creat este inserat în lista apelând functia InsertInPlace.

Dacă nici una din nu sunt situate în acelasii plan se procedează în felul urmator:

Elementul *DM* fiind inserat în lista, se construieste un nou element corespunzător poligonului *SM*, ce va fi apoi inclus si el în lista. Astfel toate informatiile legate de diedrul moale sunt inserate în *edg*. Se crează apoi elemente de tip ListPol si pentru poligoanele *DT* si *DT*. Crearea noilor elemente se face prin apelul functiei InsertListPol. Acestea vor fi incluse în lista în pozitiile corespunzătoare determinate cu ajutorul functiei InsertInPlace.

Ultima operatie realizată de functie este de a închide lista *edg* formînd o lista circulara. Ultimul element al listei trebuie să indice spre primul element din lista.

Functia returnează numărul de elemente aflate în lista *edg*.

Functia SetEdg

Functia primeste ca parametrii doi pointeri la structuri de date de tip *ListPol*.

Functia completează în structurile date informatii referitoare la poligoanele specificate în ele.

Poligoanele păstrate în aceste două elemente sunt poligoane de tip frontiera, adică se suprapun unul peste altul. Câmpul *tip* al acestor două poligoane se marchează ca fiind *FRONTIERA*. Câmpul *vpoly* al fiecărui poligon se completează în asa fel încât să indice spre celalalt polygon. Acest câmp are rolul

BUPT

de a indica faptul ca poligonul este de tip frontiera, specificându-se totodată si poligonul peste care este suprapus.

Se calculează apoi produsul scalar al normalelor celor două poligoane, pentru a se stabili dacă au sau nu acelasii sens, completând corespunzător câmpul *sens* cu valoarea 1, respectiv -1.

Functia DetOrdabcd

Functia primeste ca parametrii pointerul la lista *edg*, pointerii la structurile a două muchii si un număr întreg care precizează tipul celor două muchii (perechi de *HILL*, *VALLEY*, *FISSURE*).

Functia returnează un număr întreg cuprins între 0 si 3 care precizează pozitia relativa a celor două diedre corespunzătoare muchiilor parametru. În cazul detectarii unei erori, functia returnează -1.

Pentru fiecare din cele noua cazuri ce pot apare în functie de tipul celor două muchii, tratarea se face în mod distinct. Selectia se face în functie de variabila primita ca parametru.

Procedeul urmat în unul din aceste cazuri este urmatorul:

Se parcurge lista *edg* urmarindu-se ordinea în care apar poligoanele ce formează cele două diedre. În functie de aceasta se completează tipul poligoanelor ce constitue cele două diedre (*INTERIOR*, EXTERIOR sau *NEUTRU*) si se returnează numărul întreg corespunzător uneia din urmatoarele situatii (vezi figura 5.13):

a) - Diedrul moale este inclus în interiorul diedrului tare. Poligoanele diedrului moale în acest caz sunt de tip interior, iar cele ale diedrului tare sunt exterioare. În lista *edg* elementele *DM* si *DT* trebuie să se afle unul după altul. Functia returnează 0.

b) - Diedrul tare este inclus în interiorul diedrului moale. Poligoanele ce constitue diedrul tare sunt de tip *INTERIOR*, iar cele ale diedrului moale au tipul EXTERIOR. Functia returnează 1.

c) - Diedrele se intersectează, fiecare diedru avînd un poligon de tip *INTERIOR* si unul de tip *EXTERIOR*. Functia returnează 2.

d) - diedrele sunt exterioare unul altuia, singurul element comun reprezentîndu-l muchiile suprapuse. Pentru acest caz toate cele patru poligoane au tipul *EXTERIOR*. Functia returnează 3.

BUPT

Functia DetCaz

Functia primeste ca parametrii: lista *edg*, numărul de elemente aflate în lista, doi parametrii întregi *q1* si *q2* care indică câte perechi de poligoane aflate în acelasi plan au normalele în acelasi sens (*q1*) respectiv în sens contrar, o variabila întrega ord care reprezinta valoarea returnată de functia DetOrdabcd() si un întreg *hv* prin care se precizează tipul celor două muchii comune diedrelor.

Folosind informatiile transmise prin intermediul parametrilor, functia stabileste situatia concreta ca fiind una din 13 posibile (vezi figura 5.10).

Plecând de la tipul indicat de parametrul *hv* (*HH* pentru ambele muchii *HILL, HV pentru prima muchie HILL si a două VALLEY, HF* pentru prima muchie HILL si a două *FISSURE*, etc.) si în functie de valorile celorlalte patru variabile primite ca paramerii, functia va returna un număr întreg ce caracterizează situatia particulara întilnita. În acest scop vor fi apelate si functiile DetCaz1011() si DetCazNepart(). În functia DetCaz1011 care diferentiaza cazurile 10 si 11, se tine cont si de pozitia elementelor în lista *edg*. Pot să apare 13 cazuri pentru fiecare din cele noua situatii ce definesc forma diedrelor.

1. Fiecare poligon al primului diedru se află în acelasi plan cu un poligon al diedrului al doilea, o pereche de astfel de poligoane avînd normalele de acelasi sens, iar cealaltă de sens contrar.

2. Există tot două perechi de poligoane fiecare aflîndu-se în câte un plan, dar atit poligoanele din prima pereche cât si cele din a două pereche au normalele de acelasi sens.

3. Diedrele au două perechi de poligoane aflate în câte un plan, dar pentru fiecare pereche poligoanele au normalele de sens contrar.

4. Cele două diedre nu se intersectează însă au două poligoane aflate în acelasi plan cu normalele avînd acelasi sens.

5. Caz similar cu 4. doar ca poligoanele au normalele de sens contrar.

6. Corespunde cazului general d) când cele două diedre sunt distincte intersectîndu-se doar pe muchiile suprapuse.

7. Cele două diedre se intersectează avînd două poligoane în acelasi plan si normalele de acelasi sens.

8. Corespunde cazului de intersectie c).

9. Cele două diedre au două poligoane frontiera în acelasi plan cu normalele de sens contrar.

10. Diedrul tare se află în interiorul diedrului moale si două din poligoanele lor sunt de tip frontiera interioara (se află în acelasi plan si normalele au acelasi sens).

BUPT

11. Cazul invers cazului 10, când diedrul moale se află în interiorul diedrului tare.

12. Diedrul tare se află în interiorul diedrului moale corespunzător cazului b).

13. Diedrul moale se află în interiorul diedrului tare (cazul a).

7.9. FUNCTIILE CARE SELECTEAZA POLIGOANELE CE TREBUIE ELIMINATE

Functia ProListGlobal

Parametrii functiei sunt un pointer la o linie de intersectie, pointerul la lista poligoanelor nou create precum si pointerii la cele două volume supuse operatiei booleene.

Functia elimină din lista poligoanele care îndeplinesc anumite conditii legate de tipul lor, de volumul din care provin si de operatia booleana ceruta.

Functia ia în considerare muchiile simple de pe linie de intersectie, muchii care fac parte doar din două poligoane.

Pentru acestea functia stabileste tipul poligonului stang sau drept în functie de cel cunoscut si elimita unul din ele sau ambele.

Pentru aceasta se parcurg tablourile de muchii corespunzătoare liniei primite ca parametru si se iau în considerare doar acele tablouri care contin o singura muchie. Pentru fiecare muchie dintr-un astfel de tablou se verifica mai dacă tipul poligonului stang si drept este *NEUTRU*. În acest caz se trece la urmatorul tablou ce contine doar o muchie. Dacă însă pentru muchia curenta unul din cele două poligoane are tip diferit de neutru, în timp ce celalalt este neutru sau exterior, tipul primlui este transferat si celuilalt.

Apoi pentru fiecare astfel de muchie este apelată functia ChoosePolygon() prin intermediul căreia sunt eliminate din lista acele poligoane care contin muchia respectiva si care în acelasi timp satisfac anumite conditii impuse de operatia booleana aleasa (reuniune, intersectie sau diferenta).

BUPT

Functia ChoosePolygon

Parametrii functiei sunt un pointer la structura unei muchii, pointerul la lista poligoanelor noi si pointerii la cele două volume ce intervin în operatia booleana.

Functia elimină din lista poligoanelor noi pe acelea care îndeplinesc o anumita conditie legată de tipul lor interior sau exterior, în functie de operatia booleana ceruta.

Muchia transmisa ca parametru se găseste pe o linie de intersectie si apare doar în două poligoane. Conditiile care trebuie îndeplinite de aceste poligoane pentru a fi eliminate sunt urmatoarele:

- la reuniune (*UNI*) se aleg poligoanele care sunt de tip *INTERIOR*;
- la intersectie (*INT*) se aleg poligoanele sunt de tip *EXTERIOR*;
- la diferenta (*SUB*) se aleg poligoanele care sunt din volumul tare si sunt de tip *EXTERIOR* si poligoanele care sunt din volumul moale si sunt de tip *INTERIOR*.

Poligoanele alese vor fi eliminate din lista poligoanelor mici apelând functia EliminPoly().

Functia TestInt

Parametrul functiei este lista *tabpol* în care se păstrează toate poligoanele nou create în urma intersectiei dintre cele două volume.

Functia testează dacă între poligoanele noi există un poligon interior, caz în care returnează 1. Altfel returnează 0.

Se cauta în lista poligoanelor noi unul care are tipul *INTERIOR*. Dacă un astfel de poligon a fost găsit, functia returnează valoarea 1. Dacă a fost parcursa întreaga lista de poligoane si nu a fost găsit nici un poligon de tip *INTERIOR* functia returnează valoarea 0.

Functia InverseSenseB

Functia primeste ca parametru pointerul la lista poligoanelor nou create în urma intersectiilor celor două volume.

Functia inversează sensul de parcurgere a muchiilor poligoanelor de tip interior, precum si sensul normalei.

Functia este apelată doar dacă operatia booleana aleasa este diferenta.

Se parcurge lista poligoanelor noi. Pentru fiecare poligon din lista care este de tip *INTERIOR* se schimba sensul de parcurgere al laturilor poligonului.

BUPT

Pentru aceasta se schimba între ei cei doi vertexi care definesc începutul si sfârsitul muchiei.

Acelasi lucru se face si pentru muchiile ce formează gaurile poligonului. Se modifica de asemenea parametrii din ecuatia planului din care face parte poligonul, schimbând semnul cosinusilor directori ai normalei si valoarea termenului liber (D).

7.10. FUNCTIILE CARE IMPLEMENTEAZA OPERATIILE DE CONSTRUIRE A POLIGOANELOR REZULTATE

Plecând de la muchiile initiale ale poligoanelor si folosind muchiile noi de taiere sau de atingere, se construiesc secventele de muchii care vor forma noile poligoane. Functia care realizează acest lucru este functia Remake().

După ce se elimină din tablourile asociate planelor din lista *plane* acele poligoane care nu vor exista în volumele finale, se verifica dacă există poligoane coplanare care au muchii comune. În acest caz, din două poligoane trebuie construit unul singur. Acest lucru este stabilit si efectuat de functia RemakePoly(). Pentru aceste cazuri se apelează functia Comasare() care înlocuieste două sau mai multe muchii coliniare cu una singura.

Functia Remake

Functia are ca parametrii de intrare un pointer la un plan si un pointer la lista finala de poligoane - *tabpol*. Parametrul de iesire este un pointer la un fanion care va indica dacă operatia booleana este imposibila sau nu. O operatie imposibila apare când se încearca să se construiasca un poligon format din mai putin de trei muchii.

Functia determină toate poligoanele noi care se găsesc în planul specificat ca parametru.

Se ia pe rând în discutie fiecare poligon din plan. Dacă poligonul curent nu are taieturi sau atingeri interioare, se inserează în lista *tabpol*, prin apelul functiei InsPolygonTabpol().

Dacă însă poligonul are atingeri sau taieturi interioare, se construiesc poligoanele atomice, alegand din doua sau mai multe muchii cu inceput comun pe cea situata cel mai la stânga. Alegerea acestei muchii se face cu ajutorul functiei ChooseNext().

BUPT

După închiderea fiecarui poligon, se apelează functia MakePolygon, care pe baza listei de muchii, crează structura corespunzatoare unui poligon. Dacă noul poligon are toate muchiile de tip *ATINGERE* se apelează functia MakePolygonFiss.

Poligoanele astfel create se vor introduce într-o lista temporara de poligoane - *tabpol1*.

După ce au fost creata toate poligoanele atomice corespunzatoare unui poligon initial, se apelează functia ProcessPolygon() care, pe baza listei temporare *tabpol1*, completează lista finala de poligoane *tabpol*. Completarea se face tinînd cont si de posibilitatea ca un poligon să fie gaura pentru alt poligon

La sfârsit, se inserează poligoanele din *tabpol* în tabloul de poligoane corespunzător planului specificat ca si agument.

Functia returnează valoarea 1 în cazul în care a aparut o eroare, iar în caz contrar returnează 0.

7.11. FUNCTIILE CARE IMPLEMENTEAZA OPERATIILE PENTRU CONSTRUIREA SOLIDELOR REZULTATE

După ce au fost construite noile poligoane, trebuie stabilit numărul volumelor rezultate si poligoanele ce vor apartine fiecăruia din acestea. Pentru aceasta se stabilesc liniile de separatie dintre cele două volume prin intermediul functiei CloseInterLine(). Pe baza acestor linii se stabilesc toate poligoanele care vor apartine unui anumit volum.

Operatiile necesare sunt rezolvate în functia RemakeVolumes() si în functiile apelate de ea.

Liniile de separatie dintre volume sunt secvente închise de muchii de taiere sau muchii de atingere. În continuare se prezintă functia CloseInterLine care găseste acceste linii si functiile care sunt apelate de ea.

Functia CloseInterLine

Functia realizează închiderea liniilor de separatie dintre volumele ce rezulta din operatia booleană.

Functia este apelată indiferent de operatie deoarece chiar si operatia de reuniune poate genera mai multe volume (cazul în care două volume se ating pe o muchie comună).

BUPT

Functia primeste ca parametrii un pointer la volumul moale, un pointer *clin* la lista în care se vor păstra liniile de separatie, un pointer *tabpol* la lista poligoanelor atomice obtinute în urma intersectiilor si tipul liniei de separatie (linii de separatie determinate de muchii de tăiere, de atingere sau multipluconexe).

Lista *clin* este o listă simplu înlantuită, fiecare nod indicând începutul unei liste cu elemente de tip *IntExtL* care va include toate muchiile ce apartin unei linii de separatie. Liniile de separatie formeaza bucle inchise.

O muchie poate apartine unei linii de separatie dacă pe linia de intersectie pe care se găseste acea muchie există cel putin două muchii suprapuse provenind de la volume diferite. De aceea căutarea muchiilor ce ar putea forma linia de separatie se face doar în tablourile de tip *McPol* care contin mai multe muchii.

Dacă este găsită o astfel de muchie, se inserează un nou element în lista *clin* a liniilor de separatie, iar muchia va fi inserată în lista care va forma o linie de separatie. Stabilirea secventei complete de muchii ce va forma linia de separatie curentă se face prin apelul functiei SearchBlin().

Procedeul se repetă ciclic pana cand nu se mai gaseste nici o muchie care sa poata apartine unei linii de separatie.

Functia CloseMultipluCon

În cazul în care există muchii multiplu conexe, se consideră ca linii de separatie si liniile închise formate de cele două muchii multiplu conexe de sens contrar. Crearea lor are loc în felul următor.

Se parcurge lista liniilor de intersectie dintre plane si se caută pe aceste linii o portiune comună mai multor muchii (numărul de muchii dintr-un tablou de muchii de tip *McPol* mai mare decât 1), dar nu toate apartinând volumului moale. Se cauta printre aceste muchii dacă există vreuna care să nu aibă campul *flagdraw* pozitionat pe 1, poligonul din stânga muchiei să apartină volumului moale si care să fie de tip *MULTCON*. În cazul în care s-a găsit o astfel de muchie se retine muchia si linia infinită de care apartine, se setează câmpul *flagdraw* al acesteia pe 1 si apoi se inserează un nou nod în lista liniilor de separatie. În nod va fi inserată noua linie a cărei prima muchie o reprezintă muchia recent găsită. Crearea elementului de tip *Cutline* pentru linia de separatie se realizează cu functia InsertClin().

Se inserează muchia apelând functia InsertEdgeLine() în lista de muchii a liniei de separatie, iar apoi, apelând functia SRightSoftVol() se cauta în volumul moale muchia care este identica cu ea dar de sens contrar, deasemenea de tip

BUPT

MULTCON. Dacă nu s-a găsit o astfel de muchie înseamna ca s-a detectat o eroare si se iese din functie cu returnarea valorii 0. Dacă muchia este găsită, i se pozitionează si acesteia campul *flagdraw* pe 1 si se inserează în lista liniei de separatie.

Procedeul se repeta până la epuizarea elementelor listei *linii*, apelând functia SearchLinLeftFissure() care determină existenta unei muchii cu caracteristicele prezentate anterior. Dacă este găsită o astfel de muchie, se inserează un nou element în lista de linii de separatie *clin* corespunzatoare muchiei recent determinate. Numărul noii linii de separatie este incrementat cu doi fată de numărul liniei de separatie găsite anterior. Campul *flagdraw* al muchiei este si el pozitionat pe 1, iar muchia va fi si ea inserată în noua linie de separatie (functia InsertEdgeLine). Se reia algoritmul căutându-se si pentru această muchie corespondenta ei ce are sensul opus, pentru a fi inserată în linia de separatie. Procesul continua până când functia SearchLinLeftFissure() nu va mai găsi nici o muchie, deci nu mai există nici o noua linie de separatie.

Functia RemakeVolumes

Functia primeste ca argumente pointerul *tabpol* la lista finala de poligoane, pointerul la lista liniilor de separatie si pointerul la volumul moale.

In cadrul acestei functii se construiesc volumele rezultate în urma operatiei booleene.

Dacă nu există linii de taiere, se inserează vertexii si poligoanele finale în volumul initial. Această inserare se face prin intermediul functiilor InsVertNewVol() (pentru vertexi) si InsPolyNewVol() (pentru poligoane). Se realizează legaturile dintre vertexi prin intermediul functiei ConnVertNewVol() si legaturile dintre poligoane prin intermediul functiei LinkPolygons().

Dacă însă există cel putin două linii de separatie, atunci se completează pentru fiecare muchie din fiecare poligon existent în *tabpol* câmpul corespunzător poligonului din dreapta. Această completare se realizează prin apelul functiei GetRPolyTabtyp().

In functie de *tabpol* si lista liniilor de separatie *clin*, poligoanele se împart în clase de echivalentă. Această împartire este facuta de functia PropagareB(). Fiecare clasa de echivalentă, va conduce la crearea unui volum.

Se ia apoi pe rând fiecare clasa de echivalentă. Se alocă memorie pentru volumul corespunzător acestei clase de echivalentă si se inserează acest nou volum (cu ajutorul functiei InsertBOList) în lista volumelor finale.

BUPT

Pentru volumul corespunzător clasei de echivalentă curente, se inserează în cadrul acestei structuri de volum vertexii si poligoanele corespunzătoare lui (prin intermediul functiilor InsVertNewVol respectiv InsPolyNewVol). Se completează câmpurile *Form* pentru acesti vertexi cu ajutorul functiei ConnVertNewVol() si se realizează legaturile dintre poligoane, prin intermediul functiei LinkPolygons().

Functia PropagareB

Functia primeste ca argumente tabloul claselor de echivalentă, lista poligoanelor rezultate în urma operatiei booleane si lista liniilor de separatie.

Operatia executată de functie este aceea de numerotare a fiecărui poligon plecând de la liniile de separatie, pentru a se stabili în functie de aceste numere volumul final din care va face parte.

Se ia pe rând fiecare linie de separatie din lista de linii primita ca argument. Dacă prima muchie a sa este multipluconexa, atunci poligoanelor din stânga si din dreapta acestei muchii li se atasează numărul liniei de separatie, iar pentru urmatoarea muchie din această linie (muchie care e inversa primeia) poligoanelor din stânga si din dreapta ei li se atribuie un număr mai mare cu 1 decât numărul liniei de separatie. În acest caz, linia de separatie e marcată ca si *MULTCON*.

Dacă prima muchie a liniei de separatie nu are marcat tipul *MULTCON*, se verifica prin intermediul functiei VerifCutLine() dacă există vreo muchie în cadrul liniei de separatie care a fost marcată ca fiind multipluconexa în procesul de tratare a cazurilor prin intermediul câmpului pers[0] al punctului de început si sfârsit. Dacă există o asemenea muchie, câmpului *tratat* al acestei linii de separatie i se atribuie valoarea -1 si se părăseste tratarea acestei linii de separatie. Dacă nu există o asemenea muchie, atunci pentru fiecare muchie a liniei de separatie se apelează functia SearchMTabPol().

Această functie cauta în lista de poligoane, acele poligoane care contin muchia curenta (parcursa în orice sens) si le atribuie numărul liniei de separatie, în cazul în care acesta nu este completat.

Pe baza numărului atribuit poligoanelor care au muchii pe liniile de separatie are loc un proces de propagare, pentru stabilirea numărului celorlalte poligoane, număr care de fapt stabileste volumul din care vor face parte. Propagarea() este realizată de apelul recursiv al functiei FindNumarB(), care prin căutare la dreapta în lista poligoanelor pentru fiecare muchie, completează câmpul *numar* al tuturor poligoanelor.

Se face apoi o trecere prin lista poligoanelor. Se completează tabloul claselor de echivalentă prin apelul functiei BuildEchClassB() si apoi se

BUPT

prelucrează lista claselor de echivalentă prin intermediul functiei ProcessEchClass. Această functie elimină partile comune din cadrul unei clase de echivalentă.

După ce a fost construita si prelucrată fiecare clasa de echivalentă, se apelează functia VerifEchclassB(), care realizează legatura între liniile de separatie si clasele de echivalentă, determinându-se astfel numărul final de volume.

Functia SearchMTabPol

Functia primeste ca argumente doi pointeri la structurile *PVertex* a doua puncte, un pointer la lista finala de poligoane rezultate în operatia booleana, un număr care va indica clasa de echivalentă din care va face parte volumul si un pointer la o structura de tip *EchClass*.

Functia cauta în lista de poligoane, pe acelea care contin muchia dată prin punctele transmise ca parametru si le atribuie numărul liniei de separatie, în cazul în care acesta nu este completat.

Primele două argumente al functiei (cele două puncte) reprezinta extremitătile unei muchii.

Se cauta în listă acele poligoane care contin muchia dată sau inversa ei si se completează câmpul *numar* al acestor poligoane în cazul în care nu este completat. Dacă însă poligonul are campul *numar* completat (adică una din muchiile sale a fost găsită si pe o altă linie de separatie), atunci se inserează în tabloul claselor de echivalentă un nou element. Această operatie de inserare se realizează prin apelul functiei BuildEchClassB(). Se apelează apoi functia ProcessEchClassB() pentru prelucrarea acestui tablou al claselor de echivalentă.

Dacă muchia respectiva nu e găsită în nici un poligon, atunci ea este căutată si pe conturul gaurilor poligoanelor, după care se executa aceleasi operatii specificate mai sus.

Functia FindNumarB

Functia primeste ca argument un pointer la un poligon si realizează atribuirea unui număr poligonului curent.

Atribuirea unui număr unui poligon se face prin propagare, în urmatorul mod: se parcurg muchiile poligonului curent si dacă se găseste una pentru care poligonul din dreapta ei are un număr diferit de *TEMP* (1000), atunci acest număr este atribuit poligonului dat.

BUPT

Dacă însă nici una din muchiile poligonului nu verifica această conditie, atunci se parcurg din nou muchiile poligonului în cauza, iar dacă poligonul din dreapta muchiei curente nu are un număr se apelează recursiv functia FindNumarB cu argument poligonul din dreapta muchiei curente.

Deci pentru a da un număr unui poligon când nici una dintre muchiile lui nu are în dreapta un poligon cu un număr diferit de *TEMP*, se merge din aproape în aproape tot timpul pe poligonul din dreapta al unei muchii, până când se întilneste o muchie care are poligonul din dreapta cu număr diferit de *TEMP*. În acest moment, prin procesul de propagare înapoi, toate poligoanele prin care s-a trecut pentru a ajunge la acel poligon cu număr diferit de *TEMP*, vor primi numărul acestui poligon.

Acest proces de atribuire a unui număr unui poligon se repeta pentru toate poligoanele din lista poligoanelor ce vor constitui rezultatul final, care nu au număr. Deci, în final, toate poligoanele vor avea asociat un număr.

Functia returnează numărul pe care îl va primi poligonul specificat ca parametru.

Functia BuildEchClassB

Functia primeste ca argumente un pointer la o clasa de echivalentă, numărul poligonului din stânga si numărul poligonului din dreapta muchiei curente.

Functia construieste o clasa de echivalentă din altele două, în cazul în care poligonul din dreapta unei muchii are număr si acest număr e diferit de numărul poligonului stâng.

Operatiile efectuate sunt urmatoarele:

Se verifica dacă poligonul curent are număr. Dacă nu, se alocă memorie pentru încă un element din clasa de echivalentă specificată. Elementele sunt păstrate în clasa de echivalentă într-un tablou. Tot în cazul în care poligonul stâng nu are număr, acesta va primi numărul poligonului din dreapta, număr primit ca argument în functia BuildEchClassB (ultimul argument).

Dacă poligonul stâng are număr, dar în clasa de echivalentă nu există elemente, se introduc numerele celor două poligoane în această clasa de echivalentă si se revine în functia apelanta.

Dacă poligonul stâng are număr si în clasa de echivalentă există elemente, atunci se cauta numerele poligoanelor stâng si drept în această clasa. Dacă se găsesc, se încheie executia functiei. Dacă se găseste doar un număr din cele două, se inserează cel de-al doilea număr în clasa.

BUPT

Dacă nu este găsit nici unul dintre numere, se inserează amândouă în clasa de echivalentă curentă.

Functia ProcessEchClassB

Functia primeste ca argument tabloul claselor de echivalentă pentru a-l reorganiza, în cazul în care în timpul procesului de propagare s-a constatat că s-au atribuit numere diferite poligoanelor din aceeasi clasă.

Pentru fiecare clasa de echivalentă se verifica dacă are parti comune cu vreuna din celelalte. Dacă există clase de echivalentă cu părti comune, atunci aceste părti se comasează.

După comasare, se verifică dacă în cadrul unei clase se găseste de mai multe ori numărul unei linii de separatie. Această verificare se face prin apelul functiei DelDoubleNrB(), functie care, dacă găseste un număr de mai multe ori într-o clasa, sterge toate aparitiile multiple ale acestui număr, asa încât în final să avem într-o clasa numai un element care să aiba un anumit număr.

La sfârsitul executiei functiei, nu vor exista părti comune între oricare două clase de echivalentă.

BUPT

CAPITOLUL 8

ALGORITM ORIGINAL PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE

8.1. INTRODUCERE

În procesul de proiectare în sistemul DICAM - mediu avansat de proiectare si fabricatie asistată de calculator în domeniul constructiilor în lemn - a apărut necesitatea reprezentării corpurilor cu eliminarea liniilor si suprafetelor ascunse. Reprezentarea proiectelor mari, formate dintr-un număr foarte mare de corpuri, este deosebit de încărcată dacă corpurile apar cu toate muchiile ce delimitează fetele lor. Se preferă o reprezentare cât mai apropiată de imaginea percepută de ochiul omului, imagine care în cazul corpurilor opace, nu contine portiunile ascunse de fete ale aceluiasi sau ale altui corp [Emm94] [MV194] [MV394][Ada89].

Pentru rezolvarea acestei probleme au fost implementate mai multe variante de algoritmi, în scopul rezolvării generale si a unor situatii particulare deosebite si a îmbunătătirii performantelor fiecăruia [Mic87].

Algoritmii pentru eliminarea liniilor si suprafetelor ascunse au drept scop stabilirea părtilor ascunse ale fetelor corpurilor de reprezentat si furnizarea reprezentării lor pe ecranul monitorului sau la plotter numai prin fetele sau portiunile de fete vizibile. Corpurile ce formează proiectul sunt corpuri cu fete plane. Din multimea lor se poate selecta doar o parte, pentru a fi reprezentate [Art85][Arv91][Gla90][Kir92].

BUPT

8.2. ETAPELE PARCURSE ÎN DEZVOLTAREA ALGORITMULUI

1. Prima variantă a algoritmului se baza pe o metodă de ordonare a poligoanelor ce descriau fiecare corp si functiona corect doar în situatia în care toate corpurile erau convexe,. În afara acestei limitări, un alt dezavantaj îl reprezenta faptul că era foarte lent din cauza numeroaselor operatii efectuate pentru stabilirea părtilor vizibile. Corpurile se tratau două câte două, ca si cum doar ele ar fi trebuit reprezentate. În această situatie fiecare poligon ajungea să contină portiuni ascunse, chiar suprapuse, datorate diferitelor poligoane ale celorlalte corpuri care le puteau acoperi. Astfel, un poligon ce era complet ascuns de un altul, putea ascunde la rândul lui alte poligoane sau portiuni de poligoane. În această situatie, în procesul de stabilire a poligoanelor ascunse se efectuau multe operatii inutile.

2. În scopul cresterii vitezei de executie a fost proiectat si implementat algoritmul suprafetelor ascunse, denumit "algoritmul pictorului". Acesta constă în reprezentarea fetelor corpurilor în ordinea depărtării de punctul de vedere, fetele mai depărtate fiind reprezentate mai întâi. Obtinerea rezultatului dorit se baza pe faptul că poligoanele situate mai aproape de punctul de vedere acopereau pe cele mai îndepărtate, deoarece interiorul lor era colorat cu o nuantă ce depindea de unghiul de vedere.

3. O variantă îmbunătătită a algoritmului liniilor ascunse s-a obtinut prin modificări pentru admiterea corpurilor concave. În cazul acestor corpuri, anumite fete sau portiuni de fete pot fi ascunse de alte fete ale aceluiasi corp si în plus poate să apară problema intercalării corpurilor, când anumite fete ale unuia pot ascunde portiuni ale altuia si anumite portiuni ale celui de-al doilea pot să ascundă portiuni ale primului. În figura 8.1 este reprezentat un corp concav, atât prin toate fetele care îl alcătuiesc - *wireframe* (a), cât si prin utilizarea algoritmului de eliminare a liniilor ascunse (b). Se observă aglomerarea de linii din primul caz si dificultatea examinării corpului în reprezentarea respectivă. Ne putem imagina complexitatea reprezentării în acest mod a corpului din figura 8.2.

BUPT

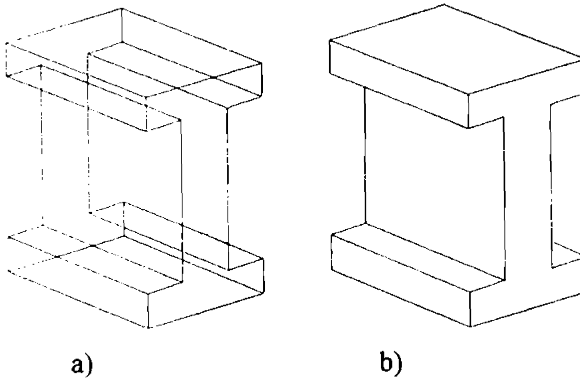


Figura 8.1.Reprezentări ale unui corp concav

Acest algoritm se bazează pe faptul că singurele muchii care pot să delimiteze zonele vizibile de cele ascunse, sunt muchiile de pe conturul (silueta) unui corp. Suprafetele pe care pot avea loc aceste delimitări sunt cele mai apropiate de muchia de contur în cauză.

Pe lângă faptul că se elimină restrictia relativă la tipul corpurilor ce pot fi reprezentate, algoritmul a câstigat în viteză prin faptul că pleacă de la o simplă observare a realitătii: oricare punct al unei fete (deci toată fata sau o portiune a ei) nu poate fi umbrit de mai multe fete; singura fată care îl poate umbri este fata cea

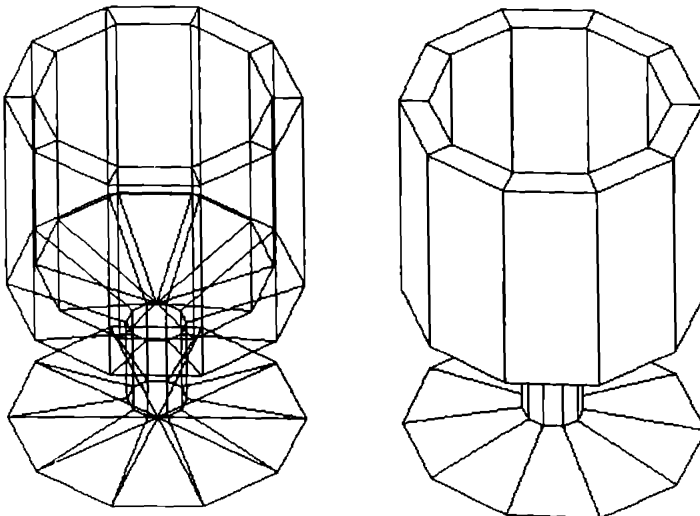


Figura 8.2.Corp complex

BUPT

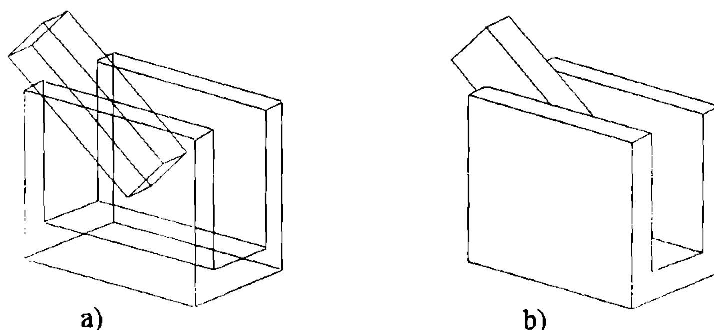


Figura 8.3.Reprezentări ale unui ansamblu de corpuri

mai apropiată de punctul de vedere si care are proiectia în planul de perspectivă suprapusă (total sau partial) proiectiei fetei umbrite.

În figura 8.3 sunt reprezentate două corpuri, unul convex si unul concav. Figura 8.3a nu furnizează informatie suficientă care să permită stabilirea pozitiei lor relative, pozitie care în figura 8.3b este evidentă. Acesta este un exemplu în care un corp concav ascunde atât părti proprii cât si părti ale altui corp. Cazul corpurilor concave cu părti întrepătrunse apare în figura 8.4.

4. O variantă a acestui algoritm a dus la transformarea lui în algoritm de suprafete ascunse. Plecând de la rezultatul generat de algoritmul precedent, se face reprezentarea poligoanelor vizibile colorate cu o anumită nuantă. Aceasta se stabileste în functie de înclinarea planului poligonului în raport cu directia razei vizuale. Corpurile ce vor fi realizate din lemn se colorează în nuante de maro, iar cele din metal în nuante de gri.

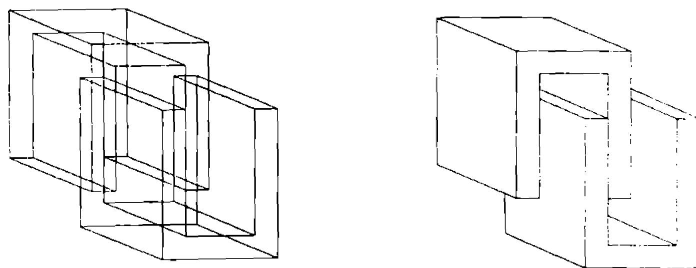


Figura 8.4.Corpuri concave întrepătrunse

BUPT

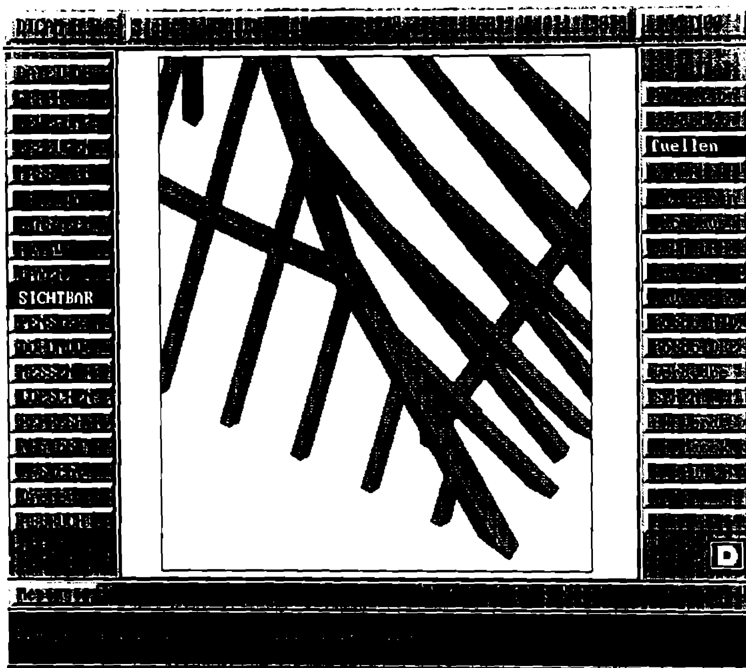


Figura 8.5.Reprezentare nuantată

O imagine rezultată prin executia programului ce implementează acest algoritm este prezentată în monocrom în figura 8.5. Se observă claritatea acestei reprezentări comparativ cu reprezentările anterioare.

5. Algoritmii prezentati functionează în ipoteza că ansamblul de corpuri de reprezentat este real, adică nu există puncte din spatiul 3D în care să existe mai mult de un corp. În procesul de proiectare însă apare necesitatea reprezentării în 2D si a unui ansamblu de corpuri virtuale (ocupând acelasi loc în spatiu). Astfel de corpuri s-ar transforma în corpuri reale dacă asupra lor s-ar efectua operatiile booleene (de reuniune, intersectie sau diferentă), în functie de ceea ce urmăreste proiectantul. Pentru a se permite reprezentarea si în astfel de situatii, a fost proiectat algoritmul denumit "no_bool_hidden_lines". Ca o etapă premergătoare celei de stabilire a suprafetelor ascunse datorită intercalării unui corp pe directia razei vizuale, are loc procesul de stabilire a muchiilor sau portiunilor de muchii situate în interiorul altor corpuri.

O reprezentare obtinută cu acest algoritm apare în figura 8.6.

BUPT

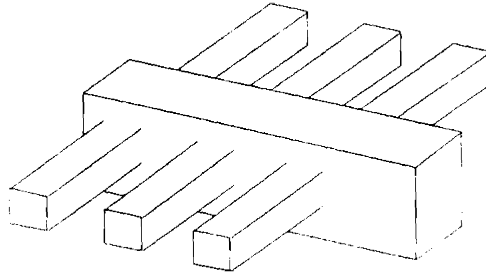


Figura 8.6.Reprezentare "no_bool_hidden_lines"

6. Ca o variantă a reprezentării corpurilor cu eliminarea liniilor ascunse, există posibilitatea trasării cu linie continuă a liniilor vizibile si cu linie punctată a celor ascunse. Fetele invizibile, adică acelea aflate în plane a căror normală nu este îndreptată spre punctul de observatie, nu sunt reprezentate. Imaginea rezultată furnizează mai multe informatii legate de forma corpurilor, fară însă a aglomera prea mult imaginea. Un exemplu este prezentat în figura 8.7.

8.3. PRINCIPIUL DE BAZĂ AL ALGORITMULUI PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE PENTRU MODELE DE SOLIDE [MV194][MV294][MV394]

Algoritmul determină, pentru un ansamblu de corpuri, modul în care se ascund unele pe altele. În urma desenării ansamblului la plotter sau pe ecranul monitorului, se obtine o imagine asemănătoare cu cea reală, asa cum este văzută

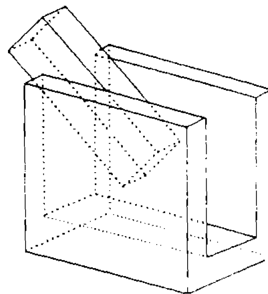


Figura 8.7.Reprezentare cu evidentierea liniilor ascunse

BUPT

dintr-un anumit punct.

Algoritmul a fost proiectat în ideea ca în anumite situatii utilizatorul să poată cere reprezentarea ansamblului de corpuri privit dintr-un punct diferit de punctul în care se găseste sursa de lumină, caz în care pe lângă portiuni ascunse pot să apară si portiuni umbrite. În cele ce urmează se face referire la situatia în care punctul din care se priveste este identic cu punctul în care se găseste sursa de lumină.

Algoritmul a impus structurarea datelor referitoare la un corp în felul următor:

Fiecare corp este descris prin poligoanele asociate fiecărei fete. Fiecare poligon este descris prin planul în care se găseste si prin lista laturilor lui. Fiecare latură (muchie) este descrisă prin capetele ei, doi vertexi de coordonate date atât în 3D cât si în 2D, în planul de proiectie. Muchiile pot fi muchii duble dacă fac parte din două poligoane vizibile, sau muchii de contur (muchii exterioare) dacă fac parte dintr-un poligon potential vizibil si unul invizibil.

Ascunderea fetelor sau portiunilor de fete ale unui corp poate fi datorată propriilor fete (vezi fig.8.1) sau fetelor altor corpuri (vezi fig. 8.3).

Principiul de bază al algoritmului constă în faptul că portiunile ascunse sunt căutate ca fiind delimitate de liniile de umbră lăsate de muchiile de contur ale corpurilor.

Întâi se stabilesc fetele potential vizibile si cele invizibile, calculând unghiul format de directia razei vizuale cu directia normalei fiecărei fete.

Întregul proces care urmează se referă la fetele potential vizibile. Dintre acestea unele sunt complet vizibile, unele sunt complet ascunse de alte fete, iar altele sunt partial vizibile, partial ascunse. Algoritmul are rolul de a determina portiunile ascunse, pentru ca în final, reprezentând fetele vizibile si portiunile de fete vizibile, să se obtină o imagine asemănătoare cu cea reală.

Pentru a stabili muchiile ce pot lăsa linii de umbră, trebuie determinate liniile de contur ale fiecărui corp, adică secventele închise de muchii de contur. O linie de contur contine bucle (anumite muchii se pot intersecta) în cazul în care corpul este concav. Pentru un corp se pot obtine mai multe contururi în cazul în care acesta contine găuri.

Pentru ca în procesul de autoascundere să se lucreze cu segmente atomice (segmente care să nu mai poată fi divizate printr-o operatie de intersectie), se efectuează intersectia tuturor muchiilor de contur ale unui corp. Prin intersectia a două muchii se întelege de fapt intersectia proiectiilor lor în planul de perspectivă. Se intersectează întâi muchiile aceluiasi contur, după care se intersectează muchiile din contururi diferite. Punctele de intersectie determinate în acest proces

BUPT

sunt marcate pe muchii si sunt memorate de asemenea într-o structură de date de tip listă înlănțuită împreună cu perechea de muchii care le-au determinat. Această listă va fi utilizată într-o etapă ulterioară a algoritmului.

Pe lângă intersectia muchiilor de contur între ele, se efectuează si intersectia muchiilor de contur cu muchiile duble ale aceluiasi volum, deoarece în procesul de stabilire a portiunilor ascunse, intervin si aceste muchii.

În etapa următoare se iau în considerare toate solidele ce trebuie reprezentate. Principiul este acelasi ca în cazul unui singur corp, dar pentru a reduce timpul de calcul al punctelor de intersectie se face o selectie prealabilă a poligoanelor potential suprapuse în planul de perspectivă. Se partajează portiunea utilă din planul de perspectivă într-un număr de dreptunghiuri, si se stabilesc corespunzător fiecărui dreptunghi, poligoanele ce se proiectează în portiunea respectivă. Cu aceste poligoane se construieste o structură de date de tip listă înlănțuită. În acest mod, operatia de intersectie se va face numai între muchiile poligoanelor aflate în aceeasi listă.

Pentru contururile ale căror bounding-box-uri se suprapun, se realizează intersectia proiectiilor muchiilor de contur ale fiecărui volum cu proiectiile tuturor muchiilor de contur ale celorlalte volume, precum si intersectia cu proiectiile muchiilor duble ale tuturor volumelor. Intersectia are loc după ce în prealabil s-a stabilit că bounding-box-rile proiectiilor celor două muchii se suprapun. Punctele de intersectie obtinute vor fi marcate pe muchii si memorate în lista punctelor de intersectie împreună cu muchiile care le-au determinat.

În acest moment toate muchiile volumelor sunt descrise prin segmente atomice si se poate trece la procesul de stabilire a segmentelor de umbră. Se va tine seama de faptul că un segment de contur poate lăsa un singur segment de umbră, pe fata cea mai apropiată în partea opusă ochiului. Portiunile de fete delimitate de astfel de segmente vor fi ascunse. Pentru a putea delimita portiunile rămase vizibile, fiecărui segment de umbră i se va asocia un segment opus, numit segment de lumină.

În acest scop, trebuie luate în considerare toate muchiile de contur ale tuturor volumelor pentru a stabili dacă pot lăsa linii de umbră si în caz afirmativ, care este poligonul pe care lasă această umbră. În această etapă se folosesc listele de poligoane proiectate într-o anumită portiune a planului de perspectivă. Odată stabilită muchia care lasă umbră si poligonul pe care apare această umbră, se calculează proiectiile capetelor muchiei ce lasă umbră pe poligonul în cauză si segmentul determinat de aceste puncte este memorat în structura de date corespunzătoare poligonului umbrit. Totodată se memorează ca segment de lumină în acelasi poligon, segmentul opus celui de umbră.

BUPT

În finalul acestui proces, poligoanele vor putea contine în structura lor de date, pe lângă muchiile initiale, toate muchiile datorate procesului de autoascundere si ascundere. Astfel dintr-un singur poligon initial ar putea rezulta mai multe poligoane noi, formate din muchiile initiale, portiuni de muchii initiale si muchii noi. Cunoscând tipul muchiilor care formează un astfel de poligon nou, se poate stabili tipul fiecărui poligon nou, poligon ascuns sau poligon vizibil. Portiunile ascunse dintr-un poligon initial sunt referite ca poligoane de umbră, iar portiunile ce rămân vizibile, ca poligoane de lumină. Dacă în multimea tuturor poligoanelor rezultate există si poligoane al căror tip nu este încă cunoscut (vizibil sau ascuns), atunci, prin intermediul unui algoritm de propagare, se stabileste si tipul acestora. Odată cunoscute poligoanele vizibile si cele ascunse, ansamblul tuturor corpurilor poate fi reprezentat prin cele vizibile, furnizând astfel imaginea corpurilor asa cum ar fi ele văzute în realitate, cu portiuni vizibile, fără portiunile invizibile sau ascunse.

8.4. VARIANTE ALE ALGORITMULUI PREZENTAT

8.4.1. Algoritmul liniilor ascunse pentru corpuri suprapuse în spatiu

În procesul de proiectare apare necesitatea reprezentării corpurilor în 2D, înainte ca procesul de modelare să fie încheiat. Acest lucru nu exclude posibilitatea ca două sau mai multe corpuri sau portiuni de corpuri să fie situate în acelasi loc în spatiu. În această situatie algoritmul prezentat nu mai functionează corect, deoarece el se bazează pe ipoteza că se reprezintă doar ansambluri de corpuri care pot exista în realitate. Pentru a satisface nevoia proiectantului de a-si verifica munca, a fost generată o variantă a algoritmului, care admite ca în ansamblul de corpuri de reprezentat să fie si corpuri ce ocupă portiuni comune din spatiu (vezi fig.8.6).

Această variantă a algoritmului stabileste în primul rând muchiile sau portiunile de muchii ale fetelor potential vizibile care sunt situate în interiorul altor corpuri. Fetele formate în întregime din astfel de muchii sunt fete ascunse.

În final se reprezintă acele portiuni de muchii ale fetelor ce nu sunt complet ascunse, portiuni care nu au fost marcate ca situate în interiorul altui volum.

În această reprezentare nu există doar contururi închise ca în cazul algoritmului anterior. Doar fetele complet vizibile vor apărea ca si contururi

BUPT

închise. Fetele care pătrund în interiorul altui volum vor fi reprezentate printr-o linie frântă deschisă.

8.4.2. Algoritmul liniilor ascunse cu colorarea fetelor vizibile

Această variantă a algoritmului are la bază rezultatul furnizat de primul algoritm prezentat, rezultat ce constă din multimea poligoanelor vizibile date prin muchiile lor.

Fetele si portiunile de fete vizibile sunt colorate în nuante de maro sau gri. Nuantele sunt stabilite în functie de unghiul format cu directia punctului de vedere de către normala fiecărei fete vizibile. Cu cât planul în care se află fata este mai înclinat, cu atât nuanta este mai închisă la culoare.

În procesul de reprezentare în 2D, se trasează pe rând contururile închise corespunzătoare poligoanelor vizibile si se declansează procesul de umplere cu culoare a suprafetei delimitate de fiecare contur, plecând de la un punct interior determinat prin calcul.

8.5. PRINCIPALELE PROBLEME ALE ALGORITMILOR LINIILOR ASCUNSE

1. O parte importantă a algoritmilor originali creati pentru ascunderea liniilor o ocupă tratarea cazurilor topologice particulare. Probleme deosebite a ridicat reprezentarea corpurilor cu fete, muchii sau vârfuri suprapuse, muchii aflate pe fete, cu vârfuri pe fete sau muchii, precum si reprezentarea corpurilor care în proiectia de perspectivă se aflau în una din aceste situatii.

În figura 8.8 este prezentat cazul a două corpuri care au două fete identice suprapuse spatial. Aici, suprafata ascunsă a unuia din corpuri este delimitată chiar de muchiile ei.

Figura 8.9 scoate în evidentă situatia în care două muchii ale aceluiasi corp sunt suprapuse în spatiu, si deci implicit apare situatia de vârf pe vârf, precum si suprapunerea muchiilor în proiectia prespectivă.

2. O altă problemă care a trebuit să fie tratată în mod special a fost cea a corpurilor cu găuri. În acest caz, pe lângă faptul că au trebuit prevăzute modalităti speciale de păstrare a informatiei relative la găuri, s-au efectuat adaptări în multe părti ale algoritmilor. Un exemplu este prezentat în figura 8.10.

BUPT

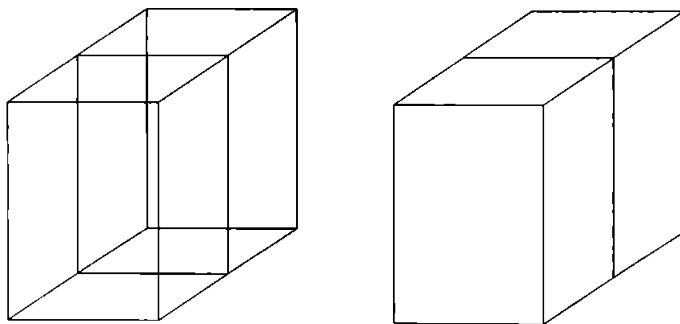


Figura 8.8. Corpuri cu fete suprapuse

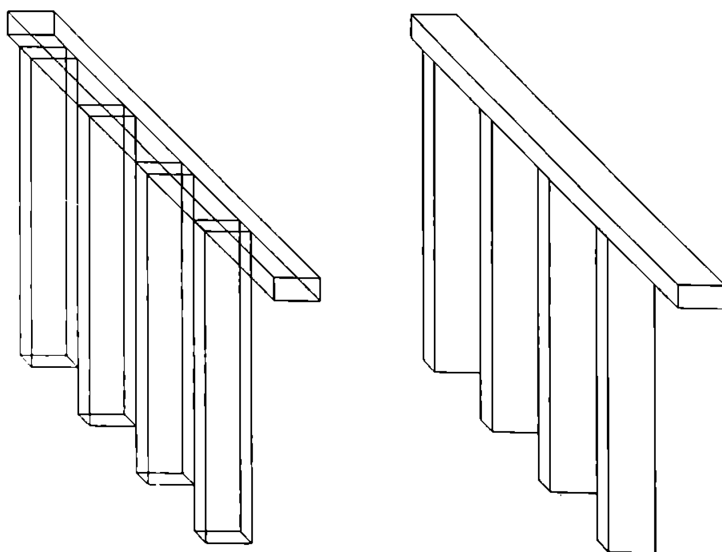


Figura 8.9. Corp cu muchii suprapuse

BUPT

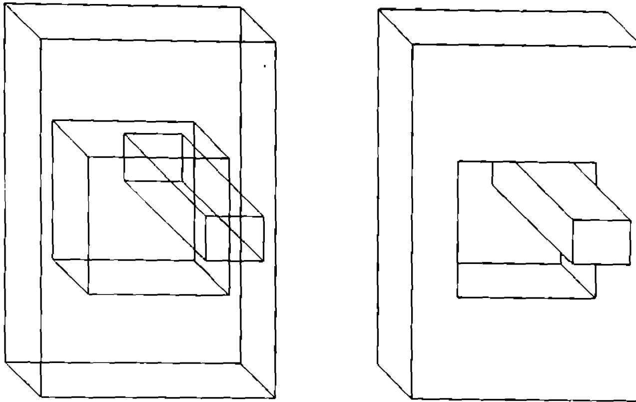


Figura 8.10. Corp cu gaură

8.6. STRUCTURILE DE DATE UTILIZATE ÎN PROGRAMELE DE LINII ASCUNSE

Structurile de date utilizate în programele ce implementează algoritmii pentru eliminarea liniilor si suprafetelor ascunse au fost create în scopul păstrării informatiei necesare temporar sau pe toată durata executiei programului.

Structura denumită ***Window*** defineste tipul fereastră. Fereastra este o portiune din ecran ce poate fi creată de utilizator si ea reprezintă spatiul de lucru efectiv. Se pot crea mai multe ferestre, dar doar una dintre ele este activă la un moment dat, doar asupra elementelor din cadrul ei pot fi executate operatii de prelucrare. Din punctul de vedere al programului pentru eliminarea liniilor si suprafetelor ascunse, structura cuprinde tablouri cu volumele aflate în fereastră si câmpurile "teta" si "fi" care indică unghiul din care se priveste întregul ansamblu de corpuri din fereastră.

Structura ***Volume*** se referă la datele unui solid si cuprinde pe lângă altele, câmpuri ce pot contine informatii care interesează în problema modelarii unui solid: un pointer la o listă de blocuri ce contin pointeri la structurile de tip *Vertex* corespunzătoare vârfurilor prin care este definit modelul solidului; un pointer la o listă de blocuri ce contin pointeri la structurile de tip *Poly* corespunzătoare poligoanelor asociate fetelor din care este alcătuit volumul; un pointer la o listă cu elemente de tip *PVertex* corespunzătoare punctelor prin care se va face reprezentarea solidului; un pointer la o listă cu elemente de tip *PictPoly*

BUPT

corespunzătoare poligoanelor prin care se va face reprezentarea solidului; un pointer la un tablou cu elemente de tip *ConturList* corespunzătoare contururilor solidului.

Structura ***PVertex*** este asociata fiecarui punct prin care solidul va fi reprezentat in planul de perspectiva. Câmpurile *coord* si *pers* au aceeasi semnificatie ca in cazul structurii *Vertex*.

Tipul de dată ***PictPoly*** definit tot ca structură, se referă la poligoanele corespunzătoare fetelor potential vizibile ale unui volum. Cuprinde informatii despre muchiile unui poligon, volumul din care face parte precum si strutura de tip *Poly* corespunzătoare acestuia (cărei fete a solidului îi corespunde). Prezintă ca si câmpuri pointeri la tablouri de tip *PEdge* în care sunt grupate muchiile exterioare (*extcontur*), muchiile duble ale poligonului (*doubl*), muchii noi datorate liniilor (segmentelor) de umbră (*shadowcut*) si de lumina (*lightcut*).

În structura ***PEdge*** sunt păstrate date despre o muchie a unui poligon: pointeri la structura *PVertex* corespunzătoare punctului de început si sfârsit al acesteia (*fv* si *sv*), un pointer la un tablou cu segmentele în care ea poate fi divizată de alte laturi în procesul de intersectie (*cedge*), segmente date prin punctele lor de început si sfârsit, precum si un câmp ce păstrează dimensiunea acestui tablou (*cbox*). Punctul de sfârsit al unui segment este identic cu punctul de început al următorului segment. Orice tăietură în latură conduce la inserarea punctului de tăiere de două ori în tabloul de segmente al laturii. Dacă latura nu prezintă nici o tăietură, în tablou nu vor exista decât punctul de început si de sfârsit al acesteia. Se păstrează numărul tăieturilor din latură.

Structura de tip ***PairEdge*** prin care se implementează o listă simplu înlăntuită, este creată pentru a păstra toate punctele de intersectie dintre proiectiile muchiilor ce intervin în procesul de reprezentare a solidelor. Intersectia se datoreste proiectării în planul de perspectivă a tuturor volumelor active (selectate). Pentru fiecare punct de intersectie din planul de perspectivă se memorează muchiile ale căror proiectii l-au determinat, poligoanele din care acestea fac parte precum si punctele de pe aceste muchii care corespund prin proiectie punctului de intersectie. În cazul în care muchiile se suprapun în perspectivă, vor exista două puncte prin care se delimitează segmentul comun si deci patru puncte reale pe cele două muchii.

Pentru a construi si utiliza contururile unui volum s-au definit tipurile de date *ConturEdge*, *ConturPoly*, *ConturList* si *Contur*.

Structura denumită ***ConturEdge*** păstrează date despre segmentele unei laturi a conturului, segmente obtinute în urma intersectiei laturii cu alte laturi.

BUPT

Pentru fiecare segment se păstrează punctele lui de început si de sfârsit si un fanion *v* care indică vizibilitatea sa (0 - vizibil, -1 - ascuns).

ConturPoly este o structură corespunzătoare unei laturi a conturului. Prin intermediul ei sunt stocate informatii privitoare la aceasta, precum si poligonul din care face parte, vertexii de început si de sfârsit ai laturii si un tablou de elemente de tip *ConturEdge* în care se pastrează toate segmentele în care se divizează latura în urma intersectiilor.

Structura ***ConturList*** este atasată unui contur al volumului. Contine un pointer la un tablou de elemente *ConturPoly*, în care se află în ordine laturile ce definesc conturul, un câmp pentru memorarea numărului de laturi si bounding-box-ul proiectiei conturului.

Tipul de dată ***Contur*** va contine toate contururile unui volum. Structura memorează numărul de contururi pe care le posedă volumul, un pointer la tabloul cu elemente de tip *ConturList* si bounding-box-ul proiectiei solidului.

Structura denumită ***Tabl*** a fost creată pentru a introduce un criteriu în selectia elemetelor ce intervin în diferitele etape ale procesului de intersectie. Planul de perspectivă este împărtit ca si tabla de sah, dar în 8*8 dreptunghiuri. Această împărtire are drept scop gruparea proiectiilor poligoanelor în functie de pozitia lor în acest plan. Structura va contine un tablou bidimensional de pointeri la liste de pointeri la structuri *PictPoly*. Fiecare listă va contine pointerii la toate poligoanele care sunt proiectate în dreptunghiul corespunzător. Structura mai contine si valorile extreme care definesc tabla de sah. Prin utilizarea partitionării tip "tablă de sah" se reduce numărul de operatii, prelucrările fiind făcute pe segmente de imagine.

8.7. FUNCTIILE CARE IMPLEMENTEAZĂ ALGORITMII PREZENTATI ÎN CAPITOLUL 6

8.7.1. Functii care implementează algoritmii pentru testele de bounding-box

Functia TestVolBoundBox( ) stabileste dacă proiectiile în planul de perspectivă a două solide specificate prin parametri de tip pointer la structurile de volum corespunzătoare ar putea să se suprapună sau nu. Functia este utilă în stabilirea perechilor de solide care în planul de perspectivă nu au proiectii

BUPT

suprapuse. În această situatie nu se mai pune problema intersectiei contururilor lor, deci se evită o serie de calcule inutile.

Functia TestBoundBox( ) stabileste dacă două contururi transmise ca parametrii ar putea să se suprapună în planul de proiectie. Cele două contururi pot fi ale aceluiasi solid sau a două solide distincte. Intersectia contururilor este necesară în procesul de determinare a segmentelor atomice de muchii.

Functia TestEdgeBBox( ) stabileste dacă proiectiile în planul de perspectivă a două muchii specificate prin parametrii de tip pointer la structurile *PEdge* corespunzătoare ar putea să se intersecteze sau nu. Acest test este util în algoritmul pentru stabilirea pozitiei relative a două segmente, pentru ca în final să se găsească un eventual punct de intersetie.

Functia TestPctPolyBBox( ) stabileste dacă proiectia în planul de perspectivă a unui punct specificat prin parametrul de tip *Perspective* corespunzător ar putea să se găsească în interiorul bounding-box-ului proiectiei unei fete vizibile transmisă ca parametru prin intermediul pointerului la structura de tip *PictPoly* corespunzătoare. Functia va fi apelată în cea care stabileste dacă un punct dat este în interiorul unui poligon dat.

8.7.2. Implementarea algoritmului pentru aflarea unui punct situat pe un segment dat când se cunoaste raportul în care punctul împarte segmentul

Acest algoritm a fost implementat prin intermediul macroinstructiei **PGET_POINT** pentru cazul în care se lucrează în planul de perspectivă. Prin argumentele ei, se dau următoarele informatii: numele variabilei de tip *Perspective* care va contine coordonatele punctului căutat, segmentul specificat printr-un punct si directia lui, de asemenea, în planul de perspectivă, precum si raportul în care punctul căutat trebuie să împartă segmentul, raport exprimat printr-un număr real, subunitar.

Pentru cazul în care se lucrează în spatiul modelului, acest algoritm a fost implementat prin intermediul macroinstructiei **GET_POINT**. Prin argumentele ei, se dau următoarele informatii: numele variabilei de tip *Vector* care va contine coordonatele punctului căutat, segmentul specificat printr-un punct si directia lui, de asemenea, în spatiul modelului, precum si raportul în care punctul căutat trebuie să împartă segmentul, raport exprimat printr-un număr real, subunitar.

BUPT

8.7.3. Functii care implementează algoritmii pentru calculul distantelor

8.7.3.1. Functia care implementează algoritmiul pentru calculul distantei dintre două puncte

În cazul algoritmului pentru eliminarea liniilor ascunse, este necesară cunoasterea distantei dintre punctul de privire (ochi) si anumite puncte din spatiul modelului, cu scopul de a realiza o ordonare a punctelor în functie de această distantă.

Dacă coordonatele punctului de privire sunt (ox, oy, oz), iar ale punctului în cauză sunt (px, py, pz), atunci distanta se calculează cu relatia:

$$d = sqrt((px - ox)^2 + (py - oy)^2 + (pz - oz)^2)$$

Pentru a reduce timpul de calcul precum si erorile ce pot să apară, se renuntă la extragerea radicalului, deoarece functia radical fiind monotonă pe domeniul de interes păstrează relatia de ordine dintre argumente si la nivelul valorilor functiei.

De aceea functia **Distance( )** calculează pătratul distantei:

$$l = (px - ox)^2 + (py - oy)^2 + (pz - oz)^2$$

Functia primeste ca parametru un pointer la structura *PVertex* atasată punctului fată de care trebuie calculată distanta de la ochi si returnează valoarea expresiei de mai sus.

8.7.3.2. Implementarea algoritmului pentru calculul distantei de la un punct la o dreaptă

Functia **PDIST** implementează algoritmul pentru calculul distantei de la un punct la o dreaptă dată prin două puncte în cazul în care acestea se găsesc în planul de perspectivă.

Functia returnează o valoare reală care reprezintă distanta de la un punct dat ca argument prin coordonatele sale în cadrul structurii *Perspective* si o dreaptă dată prin două puncte ale sale specificate de asemenea prin intermediul

BUPT

structurilor corespunzătoare de tip *Perspective*. Operatiile pe care le efectuează functia sunt precizate în descrierea algoritmului.

Functia **Dist( )** implementează algoritmul pentru calculul distantei de la un punct dat prin coordonatele lui spatiale la o dreaptă specificată prin două puncte în cazul în care acestea sunt date prin coordonatelor lor în sistemul de referintă al modelului solidului.

8.7.4. Functii pentru implementarea algoritmilor pentru determinarea pozitiei relative a două figuri geometrice

8.7.4.1. Functia care implementează algoritmul pentru determinarea pozitiei relative a unui punct fată de un plan

Functia **SPlan** implementează acest algoritm în conditiile în care trebuie să se cunoască pozitia relativă a unui punct dat prin coordonatele lui din sistemul de referintă al modelului solidului fată de planul în care se găseste o anumită fată.

Functia primeste ca argumente punctul specificat prin pointerul la structura *PVertex* corespunzătoare si planul fată de care trebuie stabilită pozitia punctului, prin pointerul la structura *PictPoly* corespunzătoare unei fetei.

Este folosită macroinstructiunea POS_P_PL care calculează expresia obtinută prin înlocuirea în ecuatia planului a coordonatelor punctului dat. Argumentele macroinstructiunii sunt cele ale functiei. Functia returnează una din valorile întregi 1, 0 sau -1, după cum semnul expresiei este pozitiv, nul sau negativ, respectiv după cum punctul se găseste de o parte a planului (indicată de sensul normalei), în plan, sau de cealaltă parte a lui.

8.7.4.2. Functia care implementează algoritmul pentru determinarea pozitiei relative a două segmente

O astfel de functie este necesară în situatia în care se verifică dacă două muchii din planul de perspectivă se intersectează sau nu.

Functia **GetIntP( )** determină pozitia relativă a două segmente de dreaptă date prin coordonatele capetelor lor în planul de perspectivă. În functie de pozitia lor relativă functia returnează o anumită valoare întreagă.

Stabilirea pozitiei relative se face în functie de valorile calculate pentru distanta de la fiecare capăt al unui segment la suportul celuilalt.

BUPT

Dacă capetele segmentelor sunt notate cu P_1, P_2, respectiv P_3 si P_4 si distantele de la aceste puncte la suportul corespunzător celuilalt segment sunt d_1, d_2, respectiv d_3 si d_4, pot exista următoarele situatii:

1.Segmente neintersectate în cazurile în care:

- $d_1 * d_2 > 0$ si $d_3 * d_4 > 0$ (suporturile segmentelor se intersectează în afara segmentelor);
- $d_1 * d_2 < 0$ si $d_3 * d_4 > 0$ (suporturile segmentelor se intersectează în interiorul primului segment dar în afara celui de al doilea);
- $d_1 * d_2 > 0$ si $d_3 * d_4 < 0$ (suporturile segmentelor se intersectează în afara primului segment dar în interiorul celui de al doilea);
- $d_1 = 0$ sau $d_2 = 0$ si $d_3 * d_4 > 0$ (primul segment se sprijină cu unul din cele două capete pe suportul celui de al doilea, dar în afara segmentului);
- $d3 = 0$ sau $d4 = 0$ si $d1 * d2 < 0$ (al doilea segment se sprijină cu unul din cele două capete pe suportul primului, dar în afara segmentului);
- $d1 = d2$ si nenule (suporturile celor două segmente sunt paralele).

În toate cazurile de mai sus functia returnează valoarea -1.

2.Segmente intersectate în cazurile în care:

- $d1 * d2 < 0$ si $d3 * d4 < 0$ cu un punct de intersectie situat în interiorul ambelor segmente, punct ce poate fi calculat folosind algoritmul de intersectie; functia returnează 0;
- $d1 = 0$ sau $d2 = 0$ si $d3 * d4 < 0$ cu punctul de intersectie P1 sau P2; functia returnează 1 respectiv 2;
- $d3 = 0$ sau $d4 = 0$ si $d1 * d2 < 0$ cu punctul de intersectie P1 sau P2; functia returnează 3 respectiv 4;

3.Segmente situate pe acelasi suport în cazul în care

- $d1 = d2 = 0$ cu cazurile distincte tratate de algoritmul pentru determinarea pozitiei relative a două segmente coliniare; functia returnează o valoare întreaga din intervalul 5 - 14 (inclusiv).

BUPT

8.7.4.3. Functia care implementează algoritmul pentru determinarea pozitiei relative a două segmente coliniare

În procesul de stabilire a segmentelor atomice de muchii, este necesar să se cunoască modul în care sunt situate relativ unul la celălalt două segmente coliniare, adică dacă au sau nu o portiune comună.

Functia **OrdSegSupr( )** implementează algoritmul pentru determinarea pozitiei relative a două segmente coliniare.

Parametrii functiei sunt punctele extreme a două muchii, precum si doi pointeri la întregi în care se vor returna sensurile segmentelor, relativ la un sens conventional de ordonare după una din cele două coordonate de perspectivă. Functia returnează o valoare întreagă corespunzătoare cazului stabilit.

Sensul unei muchii se determină comparând valorile coordonatei alese pentru ordonare ale capetelor fiecărui segment. Dacă coordonata corespunzătoare primului capăt al segmentului este mai mică decât cea a celuilalt capăt, atunci variabila care va returna sensul va primi valoarea 1. În caz contrar valoarea va fi -1.

În raport cu coordonatele plane ale celor patru puncte se disting următoarele cazuri:

- Muchiile se află pe aceeasi directie, dar nu se suprapun sau cel mult au câte un capăt comun; functia va returna -1.
- Muchiile sunt de aceeasi dimensiune si se suprapun perfect; functia va returna 14.
- Punctele de început ale celor două muchii (orientate) coincid, existând două cazuri distincte: prima muchie o include pe a doua si situatia inversă; functia va returna 6 respectiv 9.
- Punctele de sfârsit ale celor două muchii coincid, existând două cazuri în functie de care muchie o include pe cealaltă; functia va returna 8 respectiv 10.
- Segmentul comun al celor două muchii este delimitat de vertexul primei laturi care are perspectivele cele mai mici si vertexul laturii a doua cu perspectivele cele mai mari; functia va returna 5.
- Muchia a doua este complet inclusă în prima muchie; functia va returna 7.
- Prima muchie este complet inclusă în muchia a doua; functia va returna 11.
- Segmentul comun al celor două muchii este delimitat de vertexul celei de-a doua laturi care are perspectivele cele mai mici si vertexul primei laturi cu perspectivele cele mai mari; functia va returna 12.

BUPT

În plus, functia stabileste un sens relativ exprimat prin valorile 1 sau -1, dacă extremitătile fiecărui segment sunt sau nu date în ordinea crescătoare a coordonatei luate în considerare pentru studiul cazului de suprapunere.

8.7.5. Implementarea algoritmului pentru determinarea punctului de intersectie dintre două segmente

Deoarece determinarea punctului de intersectie se face pe baza distantelor de la fiecare capăt al unui segment la suportul celuilalt, valori folosite si în algoritmul pentru determinarea pozitiei relative a două segmente, algoritmul nu este implementat ca o functie de sine stătătoare, ci printr-o secventă de instructii care calculează raportul în care punctul de intersectie împarte unul din segmente si care determină punctul prin intermediul macroinstructiunii PGET_POINT.

8.7.6. Implementarea algoritmului pentru selectia unei muchii în cazul stabilirii conturului unui poligon atomic

Functia SelectNextEdge

Functia SelectNextEdge() stabileste acea muchie dintr-o lista de muchii ce pleaca din acelasi punct, pe cea care este situată cel mai la stânga în raport cu o muchie ce are sfirsitul in punctul comun al muchiilor din listă.

În cazul în care functia nu poate identifica o astfel de muchie, se returneaza -1, ca semn de eroare.

8.7.7. Implementarea algoritmului pentru testul de interior

Functia **PInterTest1** stabileste daca un punct se gaseste in interiorul proiectiei unui poligon in planul de perspectiva.

Functia primeste ca parametrii de intrare structura corespunzatoare punctului de testat, pointerul la structura muchiei al carei inceput este punctul dat si pointerul la structura poligonului fata de care se stabileste pozitia punctului.

Functia returneaza o valoare intreaga care specifica pozitia relativa a punctului fata de poligon: 1 daca punctul este in interior, 0 daca punctul este exterior.

BUPT

8.7.8. Implementarea functiilor pentru calculele de adâncime

Functia **pVisibilityTest** determină un punct dintr-un plan dat, atunci când se cunosc coordonatele lui de perspectivă.

Ca parametrii de intrare, functia are un pointer la structura *PVertex* corespunzătoare unui punct si un pointer la structura *PictPoly* corespunzătoare unui poligon.

Functia returnează un pointer la structura *PVertex* care contine punctul determinat, prin coordonatele spatiale si cele de perspectivă.

Functia pVisibilityTest() stabileste punctul din planul poligonului transmis ca parametru care are aceeasi proiectie în planul de perspectivă cu punctul transmis ca parametru.

Pentru aceasta foloseste macroinstructiile VECTOR, VECTADD, VECTSUB, GET_POINT si VECTCOPY care rezolvă practic intersectia dintre planul poligonului dat si dreapta determinată de ochi si punctul dat ca parametru.

Functia **mVisibilityTest** are rolul de a determina două puncte din spatiul modelului aflate pe două muchii, atunci când se cunoaste proiectia lor comună din planul de perspectivă. Ea utilizează algoritmul pentru determinarea coordonatelor spatiale ale unui punct atunci când se cunosc coordonatele proiectiei perspective si planul în care se găseste punctul.

Această functie are ca parametrii de intrare următoarele date:

- coordonatele de perspectivă ale unui punct ;

- pointerii la poligoanele în care se găsesc muchiile pe care sunt situate punctele căutate.

Parametrii de iesire sunt pointeri la structurile de tip *Vector* care vor contine coordonatele spatiale ale punctelor căutate.

Functia foloseste o serie de macroinstructii pentru crearea unui vector (VECTOR), adunarea (VECTADD) si scăderea (VECTSUB) a doi vectori, calculul produsului scalar a doi vectori (SCALPRD) si determinarea unui punct de pe o dreaptă dată atunci când se cunoaste pozitia sa relativă la capetele segmentului dat pe acea dreaptă (GET_POINT).

BUPT

8.7.9. Implementarea algoritmului pentru determinarea fetelor potential vizibile

Functia **HPolyVisibility** implementează algoritmul pentru determinarea fetelor potential vizibile.

Parametrii functiei sunt un pointer la structura corespunzătoare unui volum si un pointer la structura corespunzătoare unui poligon al volumului.

Functia returnează un număr întreg în functie de gradul de vizibilitate al poligonului, relativ la punctul din care este privit corpul.

Se construieste vectorul ce uneste un vârf al poligonului cu punctul din care se priveste corpul. În functie de valoarea produsului scalar dintre acest vector si vectorul normal fetei, se obtine gradul de vizibilitate a fetei în raport cu ochiul.

Dacă produsul este mai mic ca zero poligonul se află în partea invizibilă a corpului si functia va returna -1.

Dacă produsul este mai mare ca zero poligonul se află în partea vizibilă a corpului si functia va returna o valoare pozitivă dependentă de înclinatia fetei în raport cu ochiul. Această înclinatie este invers proportională cu cosinusul unghiului format de cei doi vectori.

8.7.10. Implementarea algoritmului pentru determinarea contururilor unui solid

Functia **GetContur( )** implementează algoritmul pentru determinarea contururilor unui solid. Ea stabileste contururile volumului primit ca parametru. Contururile găsite se păstrează într-o zonă de memorie special rezervată, atasată structurii de tip *Volum* a solidului considerat.

Se alocă dinamic memorie pentru un tablou de elemente de tip *ConturPoly* în care se copiază toate laturile exterioare ale poligoanelor vizibile din volum (se parcurge tabloul de pictpoly-uri al volumului si se copiază listele de laturi exterioare ale fiecărui poligon). Se creează un al doilea tablou asemănător cu primul dar neinitializat. În acestă situatie se vor construi contururile volumului. Din tabloul de bază este copiată o latură în cel de al doilea tablou. Se parcurge lista de laturi exterioare si se determină succesoarea primei laturi (primul vertex al acestei laturi este identic cu ultimul vertex al laturii anterioare). Este memorată si această latură în cel de al doilea tablou. Secventa se repetă până în momentul în care lantul de laturi s-a închis, ultima latură găsită având punctul de sfârsit identic cu punctul de început al primei laturi. Pentru acest contur obtinut se creează o

BUPT

dată de tip *ConturList* în care se va păstra conturul. În cazul în care nu toate laturile exterioare ale volumului sunt incluse în conturul găsit, se reiau operatiile pentru a se determina un alt contur. Toate contururile vor fi incluse în structura de tip *Contur* corespunzătoare volumului.

8.8. FUNCTIILE NECESARE IMPLEMENTĂRII ALGORITMULUI PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE

8.8.1. Principalele functii care stau la baza implementării[Mic87]

Functia HiddenList

Functia principală a programului este denumită **HiddenList** si are ca unic parametru o variabilă de tip short prin care se precizează optiunea de desenare a ansamblului de corpuri selectate de utilizator.

Ca iesire, functia furnizează reprezentarea corpurilor într-o imagine asemănătoare cu cea reală în care liniile ascunse nu sunt desenate.

În mediul pentru care s-a făcut implementarea, corpurile care vor fi reprezentate sunt cele corespunzătoare volumelor active din fereastra activă.

Functia foloseste valorile unghiulare *teta* si *fi*, definitorii pentru punctul din care se priveste.

Pentru reducerea duratei de executie a programului, structurile de date utilizate care dau informatii referitoare la liniile ascunse sunt memorate într-un fisier. În cazul în care se solicită optiunea pentru linii ascunse fără modificarea unghiurilor sub care sunt privite corpurile, nu se mai refac calculele, ci se extrag informatiile din fisier. De aceea, prima operatie efectuată este compararea unghiurilor curente cu cele memorate în tablourile statice în locatia corespunzătoare ferestrei. În caz de identitate si în cazul în care operatia a mai fost făcută (fanionul *Hidden_Lines* are valoarea *TRUE*), se execută secventa de desenare a volumelor fără a se mai repeta calculele.

Procesul propriu-zis de ascundere a liniilor începe prin apelul ciclic al functiei VolHidden() pentru fiecare din volumele care interesează (selectate). Functia stabileste liniile de contur ale volumului dat ca parametru. Conturul unui corp reprezintă o înlănțuire de muchii exterioare asezate în ordinea succesiunii lor

BUPT

astfel încât să formeze un lant închis. Pentru un corp se pot obtine mai multe contururi în cazul în care volumul este concav. O eroare apărută în timpul acestui proces va determina reprezentarea solidului respectiv prin toate muchiile sale, vizibile si invizibile.

Pe lângă aceasta, functia VolHidden() determină intersectiile în planul de perspectivă dintre muchiile unui contur precum si cele dintre muchiile a două contururi prin functia CutContur(), în cazul în care un anumit volum are mai multe contururi. Tot aici se stabilesc intersectiile în planul de perspectivă dintre muchiile conturului si muchiile duble (dintre două fete potential vizibile).

Cu ajutorul functiei CreateTabl() planul de reprezentare se împarte în mici dreptunghiuri, ca o tablă de sah. Fiecărui dreptunghi i se asociază o listă simplu înlăntuită care cuprinde pointeri la toate poligoanele ce au proiectia în acea portiune a planului de perspectivă, indiferent de volumul de care apartin.

Cu ajutorul functiei CutConturTwoVol() se determină punctele de intersectie dintre proiectiile contururilor tuturor volumelor luate în considerare precum si cele dintre muchiile de contur si muchiile duble.

Pentru fiecare volum din lista de volume active se parcurg toate contururile sale, latură cu latură, apelându-se functia SearchPoly(). Această functie returnează poligonul pe care segmentul din contur lasă umbră. Dacă acest poligon există, este apelată functia InterPol(). Aceasta introduce în poligon segmentul lăsat ca linie de umbră si un segment opus lui ca linie de lumină. Astfel, pentru fiecare poligon se va obtine o listă cu segmente de umbră si de lumină. Functia RemakeVol() construieste poligoanele vizibile si invizibile ale fiecărui volum, determinând care poligoane sau portiuni din ele vor fi reprezentate.

După fiecare etapă a functiei HiddenList() există o secventă de program prin care se permite întreruperea executiei programului. În cazul în care se doreste întreruperea se tastează ESC, se eliberează memoria alocată dinamic pentru variabilele din program, se pozitionează indicatorul Hidden_Lines pe FALSE (operatia de ascundere a liniilor nu s-a realizat) si se revine în functia apelantă.

Functia VolHidden

Functia are rolul de a pregăti informatiile referitoare la volumul curent în vederea operatiilor necesare în procesul de eliminare a liniilor si suprafetelor ascunse.

BUPT

Functia primeste ca parametri un pointer la structura de date corespunzătoare unui volum si un pointer la o listă simplu înlănțuită de tip *PairEdge*.

Functia returnează o valoare întreagă pentru a preciza succesul sau esecul operatiilor pe care le-a efectuat.

În această functie se apelează functia SelectVisiblePolygons() care stabileste fetele potential vizibile ale corpului si functia MakePictPolyTable() care completează structurile de date ale acestora. Odată cunoscute muchiile fiecărei fete si tipul lor, se poate trece la determinarea contururilor solidului. În acest scop se apelează functia GetContur().

Deoarece în cazul corpurilor concave contururile pot forma bucle, este necesar să se efectueze eventualele intersectii dintre proiectiile muchiilor fiecărui contur.

Se stabileste bounding-box-ul proiectiei volumului prin apelul functiei GetVolBoundBox().

În cazul corpurilor cu găuri care pot avea mai multe contururi, exită posibilitatea suprapunerii proiectiilor contururilor în planul de perspectivă. Deoarece în procesul de stabilire a liniilor ascunse este necesar să se lucreze cu segmente atomice, se apelează functia CutContur() care efectuează intersectiile dintre muchiile diferitelor contururi ale solidului precum si dintre muchiile de contur si muchiile duble ale volumului.

Functia SearchPoly

Functia SearchPoly() are rolul de a stabili pe ce poligon lasă umbră un segment al unei muchii de contur. Poligonul stabilit este returnat functiei apelante.

Functia are ca parametrii de intrare pointerul la structura corespunzătoare unei muchii de contur si la cea a unui segment al ei precum si pointerul la lista punctelor de intersectie.

Dacă portiunea de muchie aflată în continuarea celei transmise ca parametru se află în spatele a cel putin unui poligon, functia o marchează ca ascunsă si returnează 0, semn că ea nu poate lăsa umbră pe nici un poligon. În caz contrar însă, trebuie determinat poligonul cel mai apropiat pe care se va marca linia de umbră.

Pentru a stabili acest lucru se apelează functiile PInterTest1() si SPlan(). Functia PInterTest1() determină dacă sfârsitul segmentului de muchie următor se proiectează în planul de perspectivă în interiorul unui poligon dat. Functia SPlan()

BUPT

stabileste dacă acest punct este de aceeasi parte cu punctul de vedere relativ la un poligon dat sau este în partea opusă.

În cazul în care muchia nu este ascunsă, ea poate lăsa o umbră. Pentru a determina poligonul pe care lasă umbră, se caută în lista punctelor de intersectie muchiile cu care se intersectează muchia curentă si care fac parte din poligoanele situate în aceeasi portiune a planului de perspectivă. Aceste poligoane se determină pe baza coordonatelor de perspectivă ale punctului de sfârsit al muchiei următoare si a tabloului de 64 de liste. Aceste poligoane se ordonează într-o listă în functie de distanta lor fată de punctul de privire. Umbra va fi lăsată pe poligonul cel mai apropiat care are proprietatea că sfârsitul portiunii de muchie care lasă umbră se proiectează în planul de perspectivă în interiorul poligonului.

8.8.2. Functii care implementează crearea bazei de date necesare algoritmului de eliminare a liniilor si suprafetelor ascunse

Functia SelectVisiblePolygons

Functia primeste ca parametru un pointer la structura de tip *Volume* corespunzătoare unui corp si stabileste care sunt fetele lui potential vizibile.

Functia ia în considerare fiecare poligon al volumului dat ca parametru si testează dacă este o fată potential vizibilă sau invizibilă, cu ajutorul functiei HPolyVisibility(). În cazul în care fata este potential vizibilă, ea este inserată în tabloul cu elemente de tip *PictPoly* al volumului si gradul de vizibilitate este marcat într-un câmp al structurii *PictPoly*. Această valoare este folosită ulterior în cazul în care se doreste reprezentarea colorată a fetelor vizibile, pentru a stabili nuanta corespunzătoare fiecărui poligon.

Fiecărei fete potential visibile îi va corespunde un element al tabloului. Crearea tabloului se face cu functia InsertPictPoly(). Fetele invizibile nu vor avea un corespondent în acest tablou, ci doar vor fi marcate ca fiind de acest tip.

Functia MakePictPolyTabel

Această functie are drept scop completarea câmpurilor din structura *Volume* corespunzătoare modelului unui solid, cu informatiile referitoare la poligoanele potential vizibile. Acestea sunt legate de conturul poligonal al fiecărei fete, tipul fiecărei muchii si calculul parametrilor ce definesc bounding-box-ul proiectiei fiecărei muchii si fiecărei fete.

BUPT

Functia parcurge tabloul poligoanelor vizibile ale volumului dat ca parametru. Pentru fiecare poligon în parte se porneste de la un vârf al său (vertexul cu numărul indicat de câmpul *f_point* din structura *Poly* corespunzătoare poligonului). Se testează prin functia SearchPictVertex() dacă vârful se află în lista de blocuri de vertexi indicată de câmpul *f_pvtx* al volumului. Dacă nu este, se inserează în listă prin functia InsertPictVertex(). Prin intermediul functiei StartPolyRead() se determină vecinul primului vertex din poligonul respectiv. Cei doi vertexi reprezintă extremitătile unei muchii a poligonului. Se verifică dacă această muchie este dublă sau exterioară. Cu functia GetRightPolyNumber() se obtine poligonul aflat în dreapta muchiei. Dacă acest poligon este potential vizibil latura va fi inserată în lista de muchii duble, iar în caz contrar în lista de muchii exterioare. Aceste liste sunt indicate de câmpurile *doubl* si *extcontur* ale structurii *PictPoly* corespunzătoare poligonului.

Se determină următorul vertex al poligonului cu ajutorul functiei GetNextPolyVertex(), iar muchia determinată de acesta si vertexul anterior lui este supusă aceleiasi secvente de operatii ca cea precedentă pentru stabilirea tipului ei si a poligonului din dreapta.

Aceste operatii se reiau până la închiderea conturului poligonal al fetei. Inserarea lor în tablourile de muchii se face prin functia InsertDoubSegment(), respectiv prin functia InsertExtSegment().

În cazul în care poligonul prezintă găuri, întreaga operatie trebuie reluată pentru determinarea conturului acestora, a tipului fiecărei muchii si a poligonului din dreapta fiecărei muchii. Vertexii pe baza cărora se determină secventa de muchii a fiecărei găuri se găsesc precizati tot în structura *Poly* corespunzătoare fetei curente.

Fiecare vârf al fiecărei fete vizibile este supus testului prin care se verifică dacă structura de tip *PVertex* corespunzătoare este inserată în lista indicată de câmpul *f_pvtx* din structura de tip *Volume* asociată volumului. Testul se face prin functia SearchPictVertex(). Dacă nu este, se inserează în listă prin functia InsertPictVertex().

În final fiecare poligon vizibil al volumului va avea în structura *PictPoly* corespunzătoare o listă ordonată a muchiilor sale exterioare precum si o listă a muchiilor duble.

Functia CreateTabl

Functia are rolul de a stabili lista poligoanelor ale căror proiectii se găsesc

BUPT

în fiecare din cele 8*8 = 64 de dreptunghiuri în care a fost divizat planul de proiectie.

Functia determină mai întâi coordonatele extreme ale portiunii din planul de perspectivă ocupate de proiectiile modelelor solidelor de reprezentat. Pentru aceasta se compară valorile bounding-box-urilor proiectiilor tuturor volumelor, astfel încât să se obtină valorile extreme pentru punctele de perspectivă. Aceste valori notate cu *xmin, ymin, xmax, ymax* vor fi memorate în structura de tip *Tabl* corespunzătoare ansamblului de corpuri de reprezentat. Divizarea planului de proiectie în cele 64 dreptunghiuri se face pe baza acestor valori maxime si minime.

Pentru fiecare volum din ansamblul de corpuri, se parcurge tabloul corespunzător fetelor potential vizibile. Pe baza bounding-box-ului fiecăruia se vor stabili dreptunghiurile în care se găsesc portiuni ale proiectiei sale si în listele atasate acestor dreptunghiuri se vor insera elemente care să indice poligonul curent.

Astfel în urma parcurgerii tuturor poligoanelor se obtine gruparea lor în functie de pozitia proiectiilor lor în planul de perspectivă, independent de volumele din care fac parte.

8.8.3. Functii care implementează operatiile de intersectie

Functia CutContur

Functia are rolul de a stabili posibilele puncte de intersectie dintre proiectiile muchiilor a două contururi ale aceluiasi volum, precum si dintre proiectiile muchiilor de contur si proiectiile muchiilor duble.

Functia are ca argument pointerul la structura corespunzătoare volumului în cauză. Efectul operatiilor continute în această functie se va regăsi în lista punctelor de intersectie calculate, precum si în structura muchiilor de contur si a muchiilor ce definesc conturul fiecărui poligon.

Pentru realizarea obiectivului functiei se apelează functia GetIntContur() care în cazul în care i se transmit ca parametrii pointerii la structurile de tip *ConturList* corespunzătoare la două contururi, efectuează intersectia dintre muchiile acestora, iar în cazul în care i se transmite un singur astfel de pointer, efectuează intersectia dintre muchiile acestui contur si muchiile duble ale volumului.

BUPT

Functia GetIntContur

Functia poate fi apelată pentru intersectia a două contururi, sau intersectia unui contur cu muchiile duble ale unui volum. Alegerea între cele două variante se face în functie de valoarea unui argument (1 sau 0).

În primul caz, parametrii functiei sunt doi pointeri la tablouri de elemente de tip *ConturList* corespunzătoare celor două contururi ce urmează a fi intersectate si un pointer la lista în care se păstrează punctele de intersectie.

În al doilea caz, parametrii functiei sunt un pointer la tabloul de elemente de tip *ConturList* corespunzător unui contur, un pointer la structura *Volume* a solidului cu are cărui muchii trebuie intersectate muchiile conturului, precum si pointerul la lista punctelor de intersectie. Deoarece muchiile duble apar de două ori în ansamblul muchiilor care definesc contururile fetelor vizibile si pentru ca să nu se efectueze de două ori aceleasi operatii, fiecare muchie dublă este verificată dacă a fost sau nu luată deja în considerare.

Perechile de muchii din cele două categorii sunt tratate în vederea intersectiei cu ajutorul functiei TwoEdgeInt().

Functia TwoEdgeInt

Functia asigură introducerea în structurile de date corespunzătoare a informatiilor referitoare la eventuala intersectie a proiectiilor celor două muchii precizate în lista de argumente a functiei.

Muchiile care sunt considerate în vederea operatiei de intersectie pot fi două muchii de contur sau o muchie de contur si o muchie dublă. Alegerea variantei se realizează în functie de parametrii primiti de functie. Parametrii formali ai functiei sunt pointerii la structurile corespunzătoare muchiilor de intersectat, pointerul la structura poligonului căruia îi apartine muchia dublă si un pointer la lista simplu înlăntuită în care se vor păstra punctele de intersectie.

Pentru operatia de intersectie dintre proiectiile a două muchii de contur, parametrii efectivi vor fi cei doi pointeri la structurile muchiilor de contur, iar pointerii la structura corespunzătoare muchiei duble si a poligonului sunt pozitionati pe *NULL*. La intersectia între proiectia unei muchii de contur si proiectia unei muchii duble parametrii efectivi vor fi pointerul la structura unei muchii de contur si a unei muchii duble, pointerul la structura poligonului în care se găseste această muchie dublă, iar cel de al doilea pointer la structura de muchie de contur este pozitionat pe *NULL*.

BUPT

Prin apelul functiei Intersect() se stabileste pozitia relativă a două segmente, proiectiile celor două muchii. În functie de valoarea returnată de această functie, adică de pozitia relativă a celor două segmente, tratarea problemei va urma unul din următoarele cazuri:

- Dacă laturile nu se intersectează se părăseste functia, returnându-se valoarea -1.
- Dacă segmentele se intersectează, se determină punctele de pe muchii care au proiectia în punctul de intersectie a proiectiilor muchiilor, cu ajutorul functiei mVisibilityTest(). Depinde de tipul muchiilor ce operatii se fac în continuare si ce informatie se introduce în lista punctelor de intersectie. În cazul a două muchii de contur, în listă se introduce un element ce contine pointeri la cele *două muchii, la poligoanele din care fac parte, precum si la structurile PVertex* corespunzătoare noilor puncte găsite. De asemenea, aceste puncte sunt folosite la divizarea muchiilor pe care se află. Deoarece muchiile de contur apar si în structurile de tip *ConturEdge* si în *PEdge*, este necesar ca divizarea unei muchii să fie reflectată în ambele. Pentru aceasta se apelează functiile CutConturPoly() si CutEdge1().
  - ◊ Dacă punctul de intersectie este între P_1 si P_2 si între Q_1 si Q_2 , iar Q_1Q_2 este latură a conturului, se execută următoarea secventă de program: cunoscând proiectiile de perspectivă ale punctului de intersectie, prin intermediul functiei mVisibilityTest() se vor determina coordonatele punctelor reale de pe cele două laturi care se "intersectează"; pentru fiecare din puncte, se apelează functia InsertVer() care creează un vertex având coordonatele determinate cu functia mVisibilityTest() si proiectiile identice cu ale punctului de intersectie; acest punct este introdus atât în lista de segmente a laturii din contur căreia îi apartine cât si în lista de segmente a laturii exterioare corespunzătoare, din tabloul de laturi exterioare a poligonului din care face parte latura. Cele două operatii sunt realizate de functia CutConturPoly(), respectiv functia CutEdge1(); apoi cei doi vertexi astfel obtinuti sunt introdusi în elementul din lista punctelor de intersectie corespunzător acestui punct de intersectie.
  - ◊ Dacă latura Q_1Q_2 este latură dublă, secventa de program este putin diferită: pentru punctul determinat de intersectie pe latura dublă se calculează coordonatele sale reale cu functia mVisibilityTest(); se crează vertexul acestui punct; se calculează distanta de la ochi la punct si se compară cu distanta de la ochi la punctul de intersectie pe cealaltă latură, testându-se în

BUPT

acest mod care din cele două laturi este mai apropiată de ochi. Dacă latura dublă este deasupra, tăietura provocată de intersectie pe latura din contur nu va mai fi memorată în lista de segmente a acesteia si nici în lista de segmente a laturii exterioare corespunzătoare acestei laturi, iar punctul de pe latura dublă va fi păstrat în ambele câmpuri q1 si q2 ale structurii *PairEdge* care îi corespunde acestui punct de intersectie. În schimb dacă latura din contur este deasupra, tăietura va fi memorată în listele de segmente ale laturii de contur, punctul de pe latura dublă va fi păstrat doar în câmpul q1 al structurii PairEdge; se va determina în poligonul din dreapta laturii duble latura care îi corespunde acesteia si se va memora atât în lista de segmente a acesteia cât si în cea a laturii duble punctul de tăietură obtinut. Dacă P_1 sau P_2 se sprijină în interiorul muchiei Q_1Q_2, iar aceasta apartine conturului, cu ajutorul functiilor mVisibilityTest() si InsertVer() se obtine punctul de pe muchia Q_1Q_2 care va avea aceleasi perspective ca si punctul P_1 sau P_2. Pentru acest punct se apelează apoi functiile CutConturPoly() si CutEdge1() care actualizează lista de segmente a muchiei respective, apoi se completează informatia si în elementul din tabloul punctelor de intersectie corespunzător acestui punct de intersectie.

◊ Dacă însă muchia Q_1Q_2 este muchie dublă vor apare din nou două situatii: muchia dublă este deasupra muchiei de contur sau invers. Pentru prima situatie tăietura de pe latura dublă va fi inclusă atât în câmpul q1 cât si în câmpul q2 al elementului corespunzător acestui punct de intersectie din lista el, dar nu va fi păstrată si în lista de segmente a laturii duble. Pentru situatia a doua, în poligonul din dreapta laturii duble se determină latura identică cu aceasta dar de sens contrar, tăietura fiind inclusă în lista de segmente a ambelor laturi. Se completează informatia si în elementul din lista puntelor de intersectie(lista el). Latura identică cu latura dublă dar de sens invers, este obtinută cu ajutorul functiei FindRDoub().

◊ Dacă Q_1Q_2 este latura de contur si Q_1 sau Q_2 se sprijină în interiorul muchiei P_1P_2 se reia întreaga secventă de operatii pentru punctul de pe muchia P_1P_2 ce corespunde, la intersectie, punctului Q_1 sau Q_2 de pe muchia Q_1Q_2. Dacă însă cea de a doua muchie este dublă si este deasupra muchiei de contur, tăietura de pe latura de contur nu va fi memorată în lista de segmente a acestei laturi realizându-se doar inserarea informatiei în lista el. Pentru situatia inversată tăietura datorată intersectiei este inclusă si în listele de segmente ale laturii de contur folosind functiile CutConturPoly() si CutEdge1(). Dacă cele două muchii se suprapun având ca zona comună

BUPT

unul din segmentele: P_1Q_2, P_1Q_1, P_2Q_1, P_2Q_2 si punctele corespondente fiecărei extremităti a segmentului comun pe latura opusă nu se cunosc, plecând de la perspectivele extremitătilor se vor determina aceste puncte, fiind supuse aceleiasi secvente de operatii ca orice punct real de intersectie necunoscut. În cazul în care latura Q_1Q_2 este latură dublă, se include tăietura si pe muchia corespunzătoare laturii duble din poligonul aflat în dreapta acesteia.

- ◊ Dacă zona comună celor două muchii este unul din segmentele de la cazul anterior la care se adaugă situatiile în care muchia Q_1Q_2 este complet inclusă în interiorul primeia si doar unul din cele patru puncte care caracterizează situatiile de suprapunere este necunoscut; acest punct este cel de pe latura P_1P_2 corespunzător uneia din extremitătile laturii Q_1Q_2. Punctul este supus întregului lant de operatii deja cunoscute fără a exista cazuri distincte dacă Q_1Q_2 este latură de contur sau latură dublă.
- ◊ Cazul simetric celui precedent: pentru aceleasi segmente comune la care se adaugă situatiile în care muchia P_1P_2 este inclusă în interiorul muchiei Q_1Q_2. Punctul necunoscut este punctul de pe latura Q_1Q_2 corespunzător unuia din vârfurile P_1 sau P_2 (în functie de sensul de suprapunere al celor două muchii). Plecând de la proiectiile de perspectivă ale unuia din vârfuri, după caz si urmând întreaga secventă de operatii prezentată, punctul va fi găsit si va fi inclus unde este necesar. Dacă Q_1Q_2 este muchie dublă, tăietura va fi inclusă si pe latura corespunzătoare laturii duble din poligonul aflat în dreapta acesteia.
- ◊ Dacă muchia Q_1Q_2 se află în interiorul muchiei P_1P_2, iar punctele de pe prima muchie corespunzătoare vertexilor Q_1 si Q_2 sunt necunoscute, se determină aceste puncte plecând de la proiectiile de perspectivă ale punctelor Q_1 si Q_2. Se procedează asemănător cazurilor anterioare.
- ◊ Muchia P_1P_2 se află în interiorul muchiei Q_1Q_2. Situatia este similară celei precedente, rezolvarea urmând aceeasi cale. În plus dacă muchia Q_1Q_2 este muchie dublă, cele două puncte de tăietură vor trebui să fie incluse si pe muchia corespondentă muchiei duble aflată în poligonul din dreapta muchiei duble.

- Dacă muchiile care se intersectează sunt tocmai laturile vecine ale unui poligon, punctul de intersectie fiind vârful poligonului, elementul din lista punctelor de intersectie care memorează acest punct va fi sters.

BUPT

La sfârsitul functiei sunt eliberate toate zonele de memorie alocate dinamic pentru variabilele utilizate în functie.

Functia returnează tocmai valoarea returnată de functia Intersect() apelată în interiorul ei.

Functia Intersect

Functia **Intersect** are drept scop stabilirea existentei unui punct de intersectie între proiectiile în planul de perspectivă a două muchii si memorarea informatiilor corespunzătoare într-un element al unei liste destinate special unor astfel de situatii, adică muchiile ale căror proiectii se intersectează precum si coordonatele spatiale ale punctelor de pe muchii corespunzătoare punctului de intersectie.

Functia foloseste algoritmul pentru determinarea pozitiei relative a două segmente, iar în cazul în care cele două segmente se intersectează foloseste algoritmul pentru determinarea punctului de intersectie.

Functia primeste ca parametrii pointerii la structurile de date corespunzătoare a două muchii, a poligoanelor din care fac parte aceste muchii, si un pointer la lista în care se memorează eventualul punct de intersectie.

Prima operatie în vederea stabilirii unui eventual punct de intersectie constă în efectuarea testului de bounding-box, prin apelul fuctiei TestEdgeBoundB(). Dacă această functie indică faptul că bounding-box-urile celor două muchii nu au nici un punct comun, este sigur ca nici muchiile nu se intersectează.

Dacă functia pentru stabilirea pozitiei relative a celor două muchii găseste faptul că cele două segmente nu au nici un punct comun, functia Intersect() este părăsită. In caz contrar, elementele care dau informatii referitoare la punctul de intersectie sunt memorate într-un element al listei special destinate acestui scop.

În cazurile în care punctul de intersectie este unul din capetele celor două segmente, se testează cu functia PtSearch(), dacă pe muchia pe care are loc sprijinul există deja un punct ale cărui perspective sunt identice cu perspectivele punctului de intersectie. În functie de rezultatul testului functia InsList() va fi apelată cu sau fără acel punct ca parametru.

Dacă punctul de intersectie este în interiorul celor două segmente, coordonatele proiectiei perspectivă ale acestui punct sunt determinate în functie de perspectivele extremitătilor celor două muchii, prin intermediul macroinstructiei PGET_POINT. Si în acest caz se apelează functia InsList() pentru introducerea în listă a informatiilor referitoare la punctul de intersectie.

BUPT

În cazul în care muchiile se găsesc pe acelasi suport, functia OrdSegSupr( ) va determina cazul concret de plasare a lor. Un capăt al unei muchii situat în interiorul celeilalte determină un nou punct pe aceasta si el poate fi considerat un punct de intersectie. De aceea si în acest caz trebuie efectuat testul asupra existentei acestui punct pe muchia respectivă, apelând functia PtSearch(). Dacă muchiile se suprapun perfect, având si capetele suprapuse, toate cele patru puncte reale care corespund celor două puncte care delimitează segmentul comun sunt cunoscute. Si în acest caz se apelează functia InsList() pentru a completa un element al listei cu informatiile referitoare la punctele de intersectie.

8.8.4. Functii care implementează testele de vizibilitate

Functia SegCPVisib

Aceasta functie determină daca proiectia unei muchii date patrunde in interiorul proiectiei poligonului din care face parte muchi de contur data, iese din acesta sau se suprapune cu proiectia muchiei de contur in planul de perspectiva.

Functia este folosita apoi la determinarea portiunilor ascunse ale muchiei. ascundere datorată poligonului din care face parte muchia de contur data. Functia are ca parametrii pointeri la structura asociată muchiei de contur si a muchiei de analizat.

Functia returneaza 0 in cazul suprapunerii in planul de perspectiva a dreptelor pe care se găsesc cele două muchii, -1 in cazul in care muchia are sensul spre interiorul poligonului si 1 când are sensul spre exteriorul acestuia.

Cele trei situatii distincte sunt determinate de semnul produsului vectorial efectuat intre vectorii asociati celor două muchii transmise ca parametri.

Functia GetVisib

Functia este folosita in cadrul functiei SearchPoly() care implementeaza algoritmul prin care se determina poligonul pe care o anumita muchie de contur lasa umbra.

Functia stabileste daca o pereche de muchii consecutive patrunde in interiorul proiectiei unui poligon dat printr-o muchie a sa de contur. In acest caz, poligonul ce contine muchiile consecutive este introdus intr-o lista ordonată dupa distanta fată de punctul de privire. Aceasta lista este folosita pentru a determina poligonul cel mai apropiat de ochi, poligon pe care muchia de contur lasă umbra.

BUPT

Functia are ca parametri pointeri la structurile asociate muchiei de contur si muchiilor consecutive de analizat.

Stabilirea pozitiei celor doua muchii relativ la interiorul poligonului in care se gaseste muchia de contur se bazeaza pe rezultatul furnizat de functia SegVisib().

Functia returneaza *IN* in cazul in care ambele muchii consecutive patrund in interiorul proiectiei poligonului ce contine muchia de contur. In caz contrar returneaaza *OUT*.

Functia SegVisib

Aceasta functie calculeaza produsul vectorial a doi vectori specificati prin origine si virf, puncte din planul de perspectiva.

Pentru a se putea efectua produsul vectorial a doi vectori bidimensionali, se genereaza a treia dimensiune ca fiind zero. Pentru acest calcul se foloseste macroinstructia PVECTPRD. Vectorul rezultat este perpendicular pe planul de perspectiva, iar sensul lui specifica directia (intra sau iese din plan). Daca vectorul rezultat intra in planul de proiectie, primul vector intra in interioul poligonului delimitat de al doilea vector. Daca vectorul rezultat iese din planul de proiectie, primul vector iese din interiorul poligonului delimitat de al doilea vector.

Deoarece intereseaza doar sensul acestui produs, functia returneaza o valoare intreaga: -1 daca sensul este negativ, 0 daca rezultatul este nul si 1 daca este pozitiv.

Functia SegScal

Functia SegScal() calculează produsul scalar a doi vectori corespunzatori unor muchii.

Functia SegScal() primeste ca parametrii de intrare pointerii la structura *ConturPoly* corespunzătoare unei muchii de contur si la structura *PEdge* corespunzătoare unei muchii si returnează valoarea produsului scalar calculat.

Pentru a construi cei doi vectori se foloseste macroinstructia PVECTCOPY care crează vectorii corespunzători proiectiilor în planul de perspectivă ale punctelor de început si de sfarsit ale muchiilor date ca parametrii si macroinstructia PVECTSUB care crează vectorii corespunzători celor două muchii, de asemenea proiectate în planul de perspectivă. Acesti doi vectori sunt

BUPT

transmisi ca parametrii macroinstructiei PSCALPRD care calculează produsul lor scalar.

8.8.5. Functii care implementează stabilirea poligoanelor atomice si tipul lor

Functia RemakeVol

Functia are drept scop stabilirea noilor poligoane umbrite respectiv vizibile, formate pe baza informatiei introduse in structura fiecarui poligon vizibil initial.

Pentru volumul *V* transmis ca parametru, se obtine multimea de poligoane noi de tip *PictPoly* rezultate din cauza ascunderii unora de către altele. Dintre acestea unele au tipul *tLIGHT* - poligoane vizibile, iar altele au tipul *tSHADOW* - poligoane ascunse (umbrite).

Aceste poligoane se obtin ca succesiuni de muchii care au început identic cu sfârsitul muchiei anterioare. Determinarea unui astfel de contur închis este referită ca închiderea unui poligon.

În cazul în care operatia de închidere a acestor poligoane noi esuează, functia returnează -1. Pentru tratarea acestei situatii este necesar ca tabloul cu noile poligoane si dimensiunea lui să fie transmise ca parametri.

Functia RemakeVol execută următoarele operatii:

Completează câmpul type al poligoanelor care nu sunt ascunse cu *tUNDEF*.

Apelează functia MoveVolShadLight() care efectuează de fapt închiderea noilor poligoane si completarea tipului, dacă acest lucru este posibil. În cazul în care operatia de închidere esuează la unul din poligoane, functia returnează -1 pentru a semnala eroarea.

Apelează functia GetNewRPictp() care determină poligonul din dreapta fiecărei muchii a fiecărui poligon nou. Pe baza informatiei stabilite aici se va putea face distinctie între muchii exterioare (cu câmpul ce indică poligonul din dreapta nul) si muchii duble (cu câmpul corespunzător poligonului din dreapta completat).

Se eliberează memoria ce contine informatia referitoare la poligoanele initiale apelând functia FreeTp() si se completează structura volumului cu noile poligoane.

Se apelează functia GetPictPType() cu scopul de a completa câmpul type al tuturor poligoanelor rămase încă fără tip. Stabilirea tipului se face propagând

BUPT

tipul completat cu *tLIGHT* sau *tSHADOW* al unor poligoane peste muchiile lor duble la poligoanele din dreapta lor. Astfel fiecare poligon nou va fi cunoscut ca poligon vizibil (de tip *tLIGHT*) sau poligon ascuns (de tip *tSHADOW*).

Functia MoveVolShadLight

Această functie are rolul de a determina noile poligoane formate din muchiile poligoanelor initiale si liniile de umbră si lumină stabilite anterior. În plus, completează tipul acestor poligoane în cazul în care acest lucru este posibil.

Închiderea noilor poligoane se face la nivelul fiecărui poligon initial. Toate operatiile se vor repeta pentru fiecare poligon al volumului.

Pentru determinarea poligoanelor de un anumit tip, se introduc într-o lista muchiile de la care plecând se pot închide contururile lor. În cazul poligoanelor ascunse, acestea sunt muchiile de umbră si muchiile exterioare care s-a stabilit că sunt ascunse (au vizibilitatea marcată în câmpul *v* egală cu -1). În cazul celorlalte poligoane sunt muchiile de lumină, exterioare si duble, care nu au fost folosite în cazul poligoanelor de umbră. Pe măsură ce muchiile sunt folosite la închiderea unui contur, ele sunt eliminate din listă . În acest fel, când lista este goală, se stie că s-a terminat procesul de închidere a poligoanelor de acel tip. Muchiile care formează un contur începând cu o muchie din listă sunt introduse într-o listă si marcate pentru a nu mai fi folosite si în cazul altor poligoane.

Poligoanelor care contin muchii de umbră precum si poligoanelor ce provin dintr-un poligon având acest tip li se dă tipul *tSHADOW*. Celelalte vor primi tipul *tUNDEF*.

Functia tratează separat cazul poligoanelor cu muchii exterioare si cazul celor fără muchii exterioare, pentru a reduce numărul de căutari de muchii.

În primul caz se execută următoarele operatii:

Se apelează functia ShadLightList() care construieste lista tuturor muchiilor de umbră si muchiilor exterioare ascunse.

În cazul în care lista nu este goală se apelează functia MoveShadLight() care va găsi poligoanele noi pornind de la elementele listei *shl*. Acestor poligoane li se dă tipul *tSHADOW*. În cazul în care lista este goală dar poligonul curent este de tip *tSHADOW*, se apelează functia ShadLightList() care va construi lista din muchiile exterioare si duble. Poligoanele noi se vor obtine apelând MoveShadLight(). Acestora li se va da tipul *tSHADOW*.

După închiderea poligoanelor de umbră se trece la închiderea celorlalte poligoane. Functia ShadLightList() va construi lista cu muchiile de lumină si

BUPT

muchiile exterioare si duble care nu au fost încă folosite, iar functia MoveShadLight() va închide poligoanele pornind de la elementele acestei liste.

În cazul în care poligonul curent nu are muchii exterioare tratarea este similară cazului poligoanelor cu muchii exterioare, cu deosebirea că nu mai are loc căutarea lor.

Functia MoveShadLight

Plecând de la elementele listei *shl* primite ca parametru, functia închide toate contururile posibile. În cazul în care unei muchii îi urmează mai multe, se alege aceea care este situată cel mai la stânga. În acest fel se închid poligoanele cele mai mici posibile.

Pentru a evita ciclurile infinite în cazul în care un contur nu poate fi închis, se foloseste, ca variabilă de control al ciclului, numărul segmentelor din care sunt formate muchiile poligonului initial.

Numărul de repetări ale ciclului de închidere a unui contur nu este cunoscut initial.

Operatiile efectuate în acest ciclu sunt următoarele:

Se transferă primul element din lista *shl* în lista *el* care va contine muchiile ce formează conturul închis. Restul operatiilor se vor repeta pentru fiecare muchie ce va face parte din contur.

Se stabilesc toate muchiile care pornesc din punctul în care se termină ultima muchie introdusă în el si se introduc în lista *sl*.

Se apelează functia SelectNextEdge() pentru a stabili acea muchie din *sl* care este situată cel mai la stânga în raport cu ultima muchie din el. În cazul în care nu s-a putut identifica o astfel de muchie, se părăseste functia returnând -1, semn de eroare. Muchia aleasă este eliminată din *shl*, dacă există aici, si este marcată în câmpul *pos* pentru a nu fi reutilizată la închiderea unui alt contur.

Odată ce conturul a fost închis, noul poligon se inserează în tabela poligoanelor noi apelând functia InsertNewPictp(). I se completează tipul *tSHADOW* dacă este cazul. Apelând functia MakeNewPictp(), i se completează muchiile pe baza listei *el* în *extcontur* si *doubl* după cum muchiile au câmpul referitor la poligonul din dreapta nul sau nu.

BUPT

Functia ShadLightList

Functia ShadLightList() are rolul de a construi lista muchiilor de la care pornind se vor închide contururile noilor poligoane. Elementele acestei liste se determină pe baza parametrilor transmisi.

Functia trebuie să cunoască poligonul ce se descompune în poligoane noi, volumul din care face el parte, dacă contururile ce se vor închide trebuie să plece de pe linii de umbră sau nu si dacă poligonul initial are muchii exterioare sau nu. Toate acestea sunt transmise prin lista de parametrii.

Functia execută următoarele operatii:

Dacă conturul trebuie să plece de pe o muchie de umbră, se vor introduce în lista *shl* toate acele segmente din *shadowcut*. Introducerea în listă se face cu ajutorul functiei InEdgeList(). În cazul în care poligonul are muchii exterioare, se mai introduc si acelea dintre ele care sunt marcate ca fiind ascunse. Acest lucru este rezolvat de functia FindConturEdge().

Dacă nu este vorba de un contur ce pleacă de pe muchii de umbră, se vor insera în lista *shl* muchiile de lumină, duble si exterioare care nu au fost marcate ca folosite deja în alte contururi.

Functia GetPictPType

Prin această functie se stabileste tipul poligoanelor noi care nu au câmpul *type* completat la crearea lor. Problema se rezolvă printr-un algoritm de propagare a tipului cunoscut al unor poligoane peste muchiile lor duble la poligoanele din dreapta. Acest algoritm are la bază principiul potrivit căruia schimbarea tipului poligoanelor de la vizibil la ascuns sau invers are loc doar pe muchiile de umbră, respectiv de lumină.

BUPT

CAPITOLUL 9

CONCLUZII

Aceasta lucrare reprezintă finalizarea unei activităti desfăsurate pe o perioadă de aproape patru ani. Fiecare variantă a celor două programe a fost succesiv îmbunătătită si satisfactia benenficiarului de a avea un produs performant l-a determinat să achizitioneze si sisteme de calcul tot mai performante pentru implementarea aplicatiilor. Progresele realizate în dezvoltarea algoritmilor, atât în partea de conceptie, cât si în cea de implementare, au fost vizibile prin complexitatea proiectelor realizate prin modelarea geometrică folosind operatii booleene si prin corectitudinea reprezentării lor prin eliminarea liniilor si suprafetelor ascunse.

Analizând multimea de algoritmi prezentati în literatura de specialitate, am putut evidentia limitările algoritmilor respectivi si, încercând rezolvarea principalelor probleme ridicate, am reusit să obtinem rezultate remarcabile.

Ca o concluzie, se poate afirma cu certitudine că produsele realizate sunt originale, de la conceptie până la realizare practică. Dacă unii din algoritmii utilizati sunt originali chiar prin modul în care rezolvă o anumită problemă, altii sunt originali prin modul în care au fost implementati.

Rezultatele obtinute prin utilizarea programelor originale create în cadrul tezei au confirmat justetea solutiilor adoptate si corectitudinea modului de aplicare în practică.

9.1. CONCLUZII ASUPRA PROGRAMULUI DE IMPLEMENTARE A ALGORITMULUI PENTRU EFECTUAREA OPERATIILOR BOOLEENE INTRE SOLIDE

Pe parcursul procesului de implementare si testare a algoritmului, au fost efectuate o serie de îmbunătătiri care să conducă la reducerea timpului de

BUPT

executie, îmbunătătiri atât la nivelul conceptiei, cât si la nivelul tehnicilor de programare folosite. In situatia în care numărul solidelor ce intervin în operatii booleene poate ajunge să fie de ordinul zecilor, este de înteles dorinta utilizatorilor de a avea un produs performant din punct de vedere al vitezei de executie. Din acest motiv au fost preferate acele metode de implementare care păstrau în structurile de date o cantitate mai mare de informatii necesare în mai multe etape ale algoritmului, desi sunt consumatoare de memorie, în locul celor care refac calculele oricând este nevoie de anumite date.

Utilizarea programului într-o perioadă lungă de timp a scos în evidentă faptul că functionează corect într-un procent foarte apropiat de 100%. Rarele situatii în care operatia nu reuseste sunt datorate cazurilor în care nu au putut fi evitate erorile cauzate de precizia limitată. Corectitudinea se referă inclusiv la situatiile în care apar ambiguităti topologice.

In afara faptului că executia programului furnizează rezultate corecte, un alt avantaj îl reprezintă viteza cu care este executat chiar si în cazurile cele mai complexe, când trebuie efectuate operatii booleene având mai multi operanzi. Pe măsură ce un solid intervine în mai multe operatii, numărul de fete creste si deci timpul de efectuare a operatiilor creste. Acest lucru se întâmplă în cazul în care un solid considerat moale ("tăiabil") trebuie deformat de mai multe solide considerate tari.

Prin măsurările efectuate, s-a constatat că operatia de construire a poligoanelor care vor forma noul volum este o mare consumatoare de timp. Din acest motiv s-a ajuns la implementarea unei variante a algoritmului prezentat, variantă care să amâne procesul de formare a poligoanelor noi, până în momentul în care solidul moale a suferit toate operatiile de intersectie cu solidele tari. Avantajul acestei metode este important în situatiile în care un solid trebuie să fie deformat de mai multe alte solide. Altfel, economia de timp de executie nu este evidentă.

Concluzia la care s-a ajuns în urma efectuării de măsurări la nivelul diferitelor variante ale algoritmului asupra diferitelor tipuri de proiecte, este faptul că performantele algoritmului sunt dependente de genul de operatii care trebuie efectuate asupra solidelor ce alcătuiesc proiectul.

BUPT

9.2. CONCLUZII ASUPRA PROGRAMULUI DE IMPLEMENTARE A ALGORITMULUI PENTRU ELIMINAREA LINIILOR SI SUPRAFETELOR ASCUNSE

Programul pentru eliminarea liniilor si suprafetelor ascunse a fost testat si pus la punct pentru o mare varietate de situatii, modificând atât configuratia proiectului de reprezentat, ordinea de selectare a obiectelor de reprezentat cât si pozitia punctului de privire. O problemă deosebită a reprezentat-o tratarea cazurilor particulare, când solidele se ating prin muchii sau chiar suprafete sau când proiectiile muchiilor lor se suprapun.

De mare folos în procesul de punere la punct a programului a fost optiunea de modificare a punctului pe privire si cea de desenare cu linie punctată a muchiilor invizibile. Deplasând punctul de privire, utilizatorul are impresia că proiectul este cel care se roteste. Reprezentarea proiectului în pozitii consecutive apropiate permite sesizarea cu mai multă usurintă a posibilelor erori (absenta unor segmente vizibile sau prezenta unora ascunse).

Pe parcursul punerii la punct a programului a fost necesară utilizarea diferitelor tehnici de depanare care să scoată în evidentă eventuale blocări în cicluri infinite, esecuri în operatiile de alocare dinamică a memoriei, continuarea utilizării unor zone de memorie după eliberarea lor. Cauza cea mai frecventă a ciclurilor infinite a reprezentat-o imposibilitatea găsirii unei muchii care să permită inchiderea unui contur de poligon. Pentru a evita terminarea anormală a programului, a fost necesară includerea în aproape fiecare functie a unei secvente de tratare a erorilor depistate. Cu tot dezavantajul faptului că acestea au mărit semnificativ codul sursă, au fost păstrate si în varianta finală a programului.

În procesul de implementare a acestui algoritm a fost necesară luarea în considerare a preciziei calculelor efectuate. Au apărut numeroase situatii în care calculul unui punct de intersectie dintre două drepte furniza rezultate diferite, ca si cum operatia nu ar fi comutativă. Pentru evitarea acestui tip de eroare s-au introdus o serie de măsuri la nivelul functiilor ce efectuează calculele matematice. Comparatiile dintre două valori reale reprezentate în virgulă flotantă au fost făcute cu aceeasi precizie, aleasă ca fiind 10^{-6} . Valorile calculate au fost rotunjite cu aceeasi precizie. In situatiile în care nici în acest mod nu s-a putut asigura functionarea corectă a algoritmului, s-au introdus măsuri suplimentare. Acestea au constat în crearea unei baze de date care să fie mentinută pe toată perioada de efectuare a operatiilor de intersectie si utilizarea ei înainte de executia altei

BUPT

operatii. Avantajul oferit de eliminarea erorilor de calcul anihilează dezavantajul suplimentării cantitătii de memorie utilizate si a timpului necesar operatiei de căutare în baza de date.

Timpul de executie de ordinul minutelor necesar în cazul proiectelor mari, formate din zeci de solide, a impus o căutare a mijloacelor de îmbunătătire a acestui parametru, atât la nivel de organizare a operatiilor din cadrul algoritmului, cât si în procesul de implementare.

O primă modalitate prin care s-a putut obtine cresterea vitezei a fost reducerea numărului operatiilor de intersectie. Pentru aceasta au fost folosite testele de *bounding-box* la toate nivelele si algoritmul pentru stabilirea pozitiei relative a două elemente geometrice. Astfel, timpul necesar pentru efectuarea calculelor ce furnizează punctul de intersectie este înlocuit cu timpul necesar efectuării unor comparatii, eliminând calculul intersectiilor în cazul muchiilor cu vârf comun, sau în cazul muchiilor plasate în planul de proiectie în zone distincte.

Îmbunătătirea timpului de executie în faza de implementare a impus modificări în sensul păstrării pe tot parcursul procesului de prelucrare a unor date în structurile folosite, chiar dacă aceasta ducea la cresterea memoriei ocupate. Astfel s-a redus timpul de calcul al datelor necesare si implicit timpul total de executie. Este vorba de introducerea în structurile de date corespunzătoare solidului, contururilor solidului, fetelor si chiar muchiilor, a informatiilor legate de bounding-box-ul fiecărui element geometric.

9.3. REZULTATE SI MASURATORI

Deoarece aplicatia cea mai des utilizată în cadrul sistemului DICAM o reprezintă proiectarea structurii din lemn a acoperisurilor, măsurările au fost efectuate în astfel de cazuri.

In cazul primului program, măsurările au fost efectuate pentru două variante. Prima variantă efectuează operatiile booleene asupra perechilor de solide, in timp ce a doua variantă ia în considerare un singur solid ce trebuie deformat împreună cu toate solidele care îl deformează.

In cazul proiectelor de acoperis există anumite tipuri de bare care sunt deformate de altele, ambele categorii fiind fixate chiar din momentul proiectării. Prima variantă alege pe rând câte două solide, unul din lista celor care se deformează si unul din lista celor care deformează, după care se apelează programul de efectuare a operatiilor booleene având ca parametri pointerii la structurile acestei perechi de solide. Prelucrarea barelor ce formează un acoperis este încheiată în momentul în care au fost tratate toate aceste perechi de solide. In

BUPT

urma efectuării operatiei booleene asupra unei perechi de operanzi rezultă solide cu structură corectă din toate punctele de vedere. O astfel de bară din acoperis nu este deformată de obicei decât de câteva bare si de aceea performantele acestei variante sunt suficient de bune.

In cazul aplicatiilor în care o bară este deformată de mai multe alte bare este mai convenabilă varianta a doua. Dacă operatia booleeană s-ar efectua o dată pentru fiecare solid tare cu cel deformabil, numărul elementelor geometrice ale acestuia din urmă crescând de fiecare dată, si operatiile s-ar efectua tot mai lent. Constructia solidului rezultat în urma fiecărei operatii consumă tot mai mult timp pe măsură ce acesta are tot mai multe fete. Varianta a doua asigură efectuarea operatiei de constructie a rezultatului o singură dată, după ce au fost stabilite elementele noi datorate tuturor solidelor care deformează operandul comun.

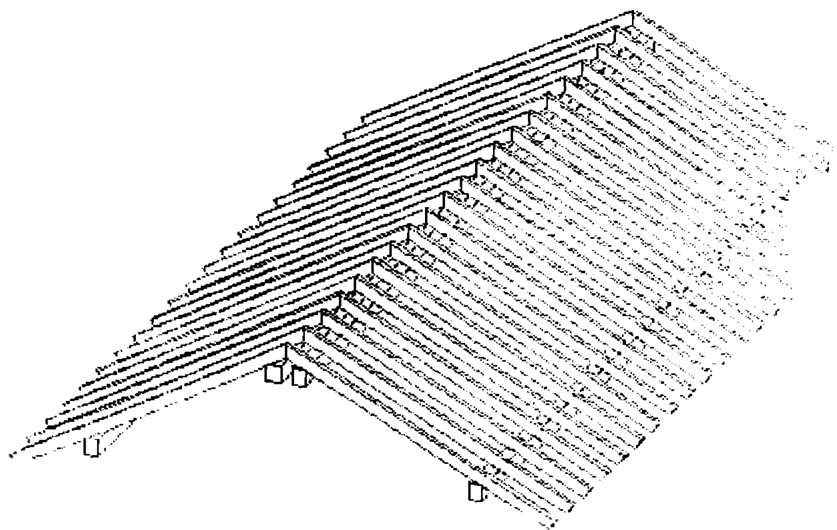


Fig.9.1.Acoperisul asupra căruia s-au efectuat măsurătorile

BUPT

Pentru proiectul de acoperis din figura 9.1 s-au efectuat măsurătorile reprezentate în graficul din figura 9.2 pentru cazul în care una din barele orizontale deformează barele ce se sprijină pe ea, folosind prima variantă a algoritmului pentru efectuarea operatiilor booleene dintre solide. Măsurătorile s-au efectuat pe două tipuri de calculatoare, unul cu procesor 486DX la 33 Mhz si altul cu procesor Pentium la 60 Mhz.

In figura 9.3 sunt reprezentate măsurătorile efectuate pentru cazul în care una din barele orizontale este deformată de celelalte bare care se sprijină pe ea, utilizând a doua variantă a algoritmului.

Executia programului de eliminare a liniilor si suprafetelor ascunse a fost testată prin măsurarea timpului de calcul pentru perechi de bare din acoperisul din figura 9.1. S-a constatat o crestere exponentială a timpului relativ la o crestere

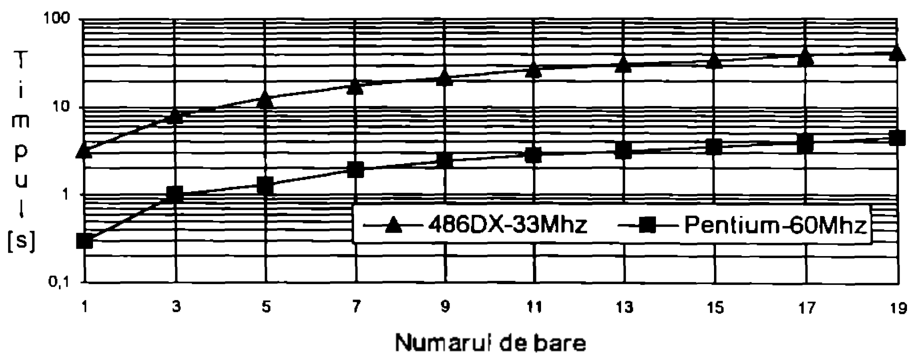


Figura 9.2. Dependenta duratei calculelor de complexitatea proiectului si tipul procesorului pentru prima variantă de algoritm

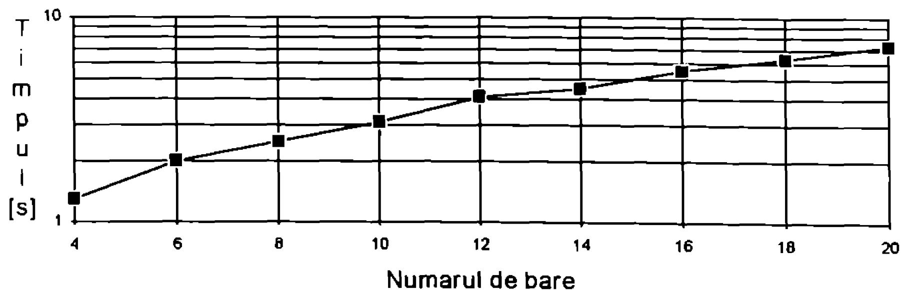


Figura 9.3. Dependenta duratei calculelor de numărul barelor care deformează, pentru a doua variantă de algoritm

BUPT

liniară a numărului de perechi de bare considerate în reprezentare.

Comparatia făcută între durata de executie a programului pentru eliminarea liniilor si suprafetelor ascunse si programul pentru reprezentarea nuantată (vezi fig.9.4, unde timpul este reprezentat logaritmic, pentru o mai usoară apreciere relativă a valorilor) arată că diferentele (relative/procentuale) sunt mai importante în cazul unor numere mai reduse de perechi de bare, în timp ce devin aproape insesizabile în cazul reprezentării mai multor solide.
Rezultă că timpul necesar calculelor de nuantare creste mai lent decât cel necesar pentru ascunderea liniilor si suprafetelor, ceea ce confirmă previziunile teoretice (dependentă patratică la ascundere si lineară la nuantare).

In figurile 9.2, 9.3 si 9.5 s-a dorit scoaterea în evidentă a influentei mari pe care o are tipul procesorului calculatorului folosit asupra duratei de executie a unor operatii identice. Sunt prezentate rezultatele obtinute pe un calculator cu procesor 486DX la 33 Mhz si pe un calculator cu procesor Pentium la 60 Mhz. Se observă că avantajul procesorului mai puternic este mai important la număr mare de elemente prelucrate.

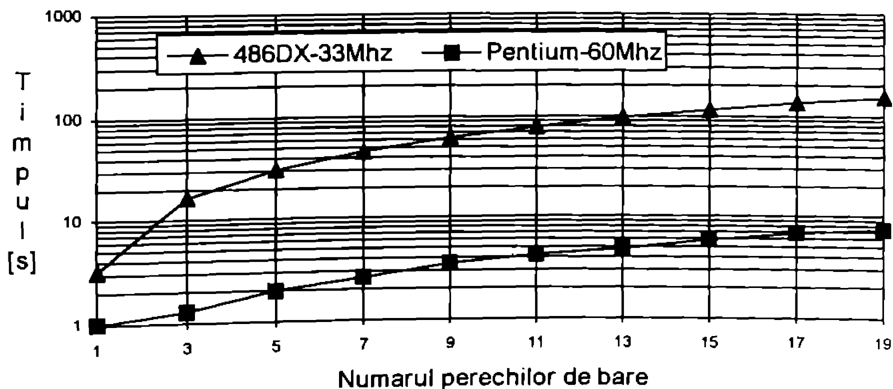


Figura 9.4. Dependenta duratei calculelor de complexitatea proiectului

BUPT

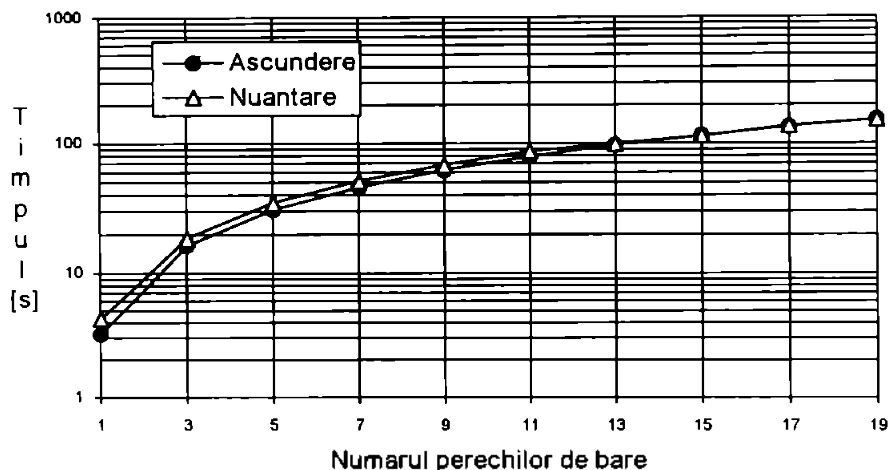


Figura 9.5. Dependenta duratei calculelor de procesorul calculatorului

9.4.PERSPECTIVE

Rezolvarea celor două probleme, cea a modelării solidelor prin operatii booleene si cea a eliminării liniilor si suprafetelor ascunse, a scos în evidentă numeroase elemente comune. Printre acestea se pot enumera: generarea structurilor de date cu informatii asemănătoare corespunzătoare fetelor si muchiilor, necesitatea utilizării testelor de bounding-box, operatiile de intersectie dintre diferite elemente geometrice, determinarea pozitiilor relative dintre acestea, problemele de clasificare sau de obtinere de contururi poligonale pornind de la un set de muchii date. Acest lucru a condus la ideea unei unificări a structurilor de date si a functiilor care efectuează operatii similare, sau chiar la cea a unei implementări a algoritmilor într-un limbaj orientat pe obiecte, cum ar fi C++. Avantajele deosebite ale definirii tipurilor de date abstracte, ale ascunderii datelor si polimorfismului, ar contrabalansa efortul proiectării orientate pe obiecte si implementarea noii versiuni. In situatia în care se doreste standardizarea descrierilor operatiilor ce trebuie efectuate pe masinile unelte, crearea unei baze de date care să contină o serie de informatii referitoare la prelucrările suferite de fiecare solid, proiectarea orientată pe obiecte ar fi chiar necesară.

BUPT

9.5. CONTRIBUTII ORIGINALE

Principalele contributii originale aduse în cadrul tezei sunt:

1. Sintetizarea rezultatelor cercetărilor pe plan mondial în domeniul abordat de teză, astfel încât să se realizeze o unealtă utilă celor ce doresc să abordeze în continuare acest domeniu.
2. Analiza avantajelor si limitelor diferitilor algoritmi prezentati în literatura de specialitate, în scopul conceperii unor algoritmi superiori, care să functioneze corect fără restrictii aplicate solidelor operanzi.
3. Analiza avantajelor si limitelor diferitelor metode de modelare a solidelor prezentate în literatura de specialitate, din punct de vedere al problemelor de rezolvat, efectuarea operatiilor booleene si reprezentarea cu eliminarea liniilor si suprafetelor ascunse, cu enuntarea unor concluzii cu privire la oportunitatea utilizării.
4. Abordarea sistematică si unitară a celor două aspecte din cadrul tezei eliminarea liniilor si suprafetelor ascunse si realizarea operatiilor booleene între solide, ceea ce a condus inclusiv la o organizare superioară a programelor din punct de vedere structural si al eficientei.
5. Analiza unitară a subalgoritmilor necesari în diferitele etape de rezolvare a problemei operatiilor booleene si a eliminării liniilor si suprafetelor ascunse. Incă din etapa de conceptie a algoritmilor au rezultat similitudinile dintre modalitătile de rezolvare a celor două probleme la primul nivel de subprobleme, cum ar fi crearea bazei de date specifice problemei plecând de la informatiile disponibile referitoare la solidele operanzi, determinarea elementelor pe baza cărora se va obtine rezultatul dorit si constructia bazei de date corespunzătoare rezultatului care să permită integrarea acestuia în oricare alt proces al sistemului CAD utilizat.
6. Determinarea si alegerea reprezentării modelelor solidelor celei mai convenabile, din punctul de vedere al prelucrărilor de efectuat, cunoscut fiind faptul că nu orice fel de reprezentare permite orice fel de manipulare a obiectelor reprezentate.
7. Alegerea structurilor de date cele mai convenabile pentru stocarea informatiilor ce descriu un model de solid, structuri care să asigure accesul simplu, cât mai direct la oricare informatie necesară, în oricare moment al executiei algoritmilor.
8. Analiza datelor necesare algoritmilor si a modalitătilor de organizare a informatiei, în scopul asigurării eficientei procesului de prelucrare a lor. Această etapă a procesului de proiectare a fost necesar să fie efectuată de mai multe ori, o dată pentru fiecare mod de abordare a rezolvării problemelor, lucru care a evidentiat dezavantajul utilizării metodei de proiectare structurate comparativ cu cea orientată pe obiect.

BUPT

9. Abordarea sistematică si unitară a metodelor de rezolvare a problemelor geometrice de intersectie dintre diferitele elemente (plane, poligoane, drepte, semidrepte, segmente de dreaptă) în spatiul 2D si în cel 3D.
10. Abordarea sistematică si unitară a metodelor de rezolvare a problemelor geometrice de clasificare a segmentelor de dreaptă relativ la un poligon (în spatiul 2D), relativ la două poligoane (în spatiul 3D) si relativ la un corp.
11. Analiza unitară a cazurilor particulare de contacte dintre solide în spatiul 3D, respectiv dintre proiectiile solidelor în 2D. A fost necesară tratarea acestor situatii cu o atentie deosebită, deoarece abordarea generală nu furniza rezultate corecte.
12. Luarea în considerare în toate etapele de proiectare a influentei pe care poate să o aibă existenta concavitătilor sau a găurilor asupra modului de evolutie a algoritmilor utilizati. Admiterea solidelor concave sau cu găuri în lista operanzilor a dus la o complicare importantă a subalgoritmilor, dar a fost necesară pentru a putea prelucra toate tipurile de corpuri reale.
13. Abordarea sistematică si unitară a căilor de evitare a erorilor datorate preciziei de calcul în cazul reprezentării datelor în virgulă flotantă. Deoarece solidelor rezultate prin operatii booleene trebuie să li se poată efectua aceleasi prelucrări ca si solidelor initiale, inclusiv operatia de reprezentare cu eliminarea liniilor si suprafetelor ascunse, a fost necesar ca în procesul de identificare a diferitelor elemente geometrice să se lucreze cu aceeasi precizie în ambele programe.
14. Conceperea algoritmului de clasificare pentru determinarea fetelor ce vor face parte din solidul sau solidele rezultate în urma operatiei booleene. Din multimea acestor fete trebuie să se poată stabili submultimile corespunzătoare fiecărui solid rezultat, în situatia în care rezultatul nu este fomat dintr-un singur solid.
15. Conceperea algoritmului de propagare a proprietătii unor fete de a apartine unui solid rezultat, spre fetele învecinate cu apartenentă necunoscută, în scopul clasificării lor relativ la solidele rezultate.
16. Conceperea algoritmului de clasificare pentru determinarea fetelor si portiunilor de fete vizibile prin care se vor reprezenta solidele selectate pentru operatia de vizualizare cu eliminarea liniilor si suprafetelor ascunse.
17. Conceperea algoritmului de propagare a proprietătii unui poligon atomic de a fi vizibil spre poligoanele vecine, în scopul stabilirii vizibilitătii, respectiv invizibilitătii tuturor poligoanelor atomice determinate prin procesul de intersectie a elementelor geometrice în planul de prespectivă.
18. Stabilirea algoritmului de determinare a vizibilitătii unui segment de muchie relativ la o fată a unui solid.
19. Stabilirea strategiei optime pentru modul de alegere a operanzilor în cazul operatiilor booleene care contin mai mult de doi termeni. Criteriul luat în considerare este cel al timpului de executie minim.

BUPT

20. Implemetarea algoritmilor pentru intersectia diferitelor elemente geometrice, în spatiul 2D pentru proiectiile acestora în planul de perspectivă, respectiv în spatiul 3D pentru efectuarea operatiilor booleene dintre solide.
21. Implementarea algoritmilor pentru obtinerea contururilor de poligoane formate dintr-un set de muchii coplanare, în cazul în care se doreste determinarea poligoanelor cele mai mici sau cele mai mari.
22. Implementarea algoritmului pentru asocierea contururilor găurilor la conturul poligonului căruia îi apartin.
23. Implementarea algoritmilor pentru stabilirea pozitiei relative a unui punct fată de un poligon, respectiv fată de un volum.
24. Implementarea algoritmilor pentru clasificarea segmentelor unei drepte relativ la un poligon, respectiv relativ la un volum.
25. Implementarea algoritmului pentru stabilirea pozitiei relative a două segmente coliniare date prin capetele lor exprimate prin coordonate în planul de proiectie, respectiv prin coordonate în spatiul modelului solidului.

BUPT

BIBLIOGRAFIA

[3DC] *3D-Calc Software for IBM-PC And Compatibles*, Markt & Technik

[Ada89] Adams, L. - *High Performance CAD Graphics*, Windcrest, 1989

[Ada91] Adams, L. - *Lee Adams' Visualisation Graphics in C* Windcrest, 1991

[Ada92] Adams, T. - *Writing DOS Device Drivers In C*, Academic Press, 1992

[Ang90] Angell, I.O.: *High Resolution Computer Graphics Using C* Macmillan, London, 1990

[Art85] Artwick, Bruce A. - *Microcomputer Displays, Graphics and Animation*, Prentice - Hall, Inc; Englewood Cliffs, New Jersey, 1985

[Arv91] Arvo, J. - *Graphics Gems II*, Academic Press, Inc, 1991

[Bar191] Barkakati, N. - *Macro Assembler Bible*, The Waite Group Inc, 1991

[Bar192] Barkakati, - *Microsoft Macro Assembler Bible*, Academic Press, 1992

[Bar291] Barkakati, N. - *Object Oriented Programming In C++*, Sams, 1991

[Bar292] Barkakati, N. - *Object-Oriented Programming In C++*, Academic Press, 1992

[Bar90] Barkakati, N. - *Microsoft C Bible*, Sams Publishing, 1990

[Ben94] Benouamer, M.O., Mechelucci, D., Peroche, B. - Error-Free Boundary Evaluation Based On A Lazy Rational Arithmetic: A Detailed Implementation, *Computer-Aided Design*, Butterworths, Volume 26, Number 6 June, 1994

[Boe92] Boersma - *Inside Autocad For Windows*, Academic Press, 1992

[Bor91] Borland - *Borland C++*, Kosel Gmbh, 1991

[Bou92] Bousquet, H. - *Autocad 3D Design And Presentation*, Academic Press, 1992

[Bou94] Bouma, W.J., Vanecek Jr, G. - Modelling contacts in a physically based simulation, *Computer-Aided Design*, Butterworths, Volume 26, Number 6 June, 1994

[Bow88] Bowerman, R.G. - *Putting Expert Systems Into Practice*, Van Nostrand Reinhold Company, 1988

BUPT

[Coh93] Cohen, Michael F. si Wallace, John R. - *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Harcourt Brace & Company, Boston, 1993

[Com94] *Computer Aided Design Report*. CAD/CAM Publishing, Vol.14, No.7, July 1994

[Con92] Conner - *Autocad Student Workbook*, Academic Press, 1992

[Cox94] Cox, E. - *The Fuzzy Systems Handbook* Academic Press, 1994

[Cre89] Cretu, V.I. - *Structuri de date si tehnici de programare avansate* UTT, 1989

[Cun92] Cunningham, St. - *Computer Graphics Using Object-Oriented Programming*. Wiley & Sons, 1992

[Dav91] Davies, B.L.; Robotham, A.J.; Yarwood, A. - *Computer Aided Drawing and Design*. Chapman & Hall, 1991

[DeS89] De Silva, C.W. - *Knowledge Based/Control With Application To Robots*, Springer Verlag, 1989

[Dit91] Dittrich, K. - *On Object Oriented Database Systems*, Springer Verlag, 1991

[Dog88] Dogaru, Dorian - *Elemente de grafica 3D*, Editura stiintifică si enciclopedică, Bucuresti, 1988.

[Don93] Donikian,S., Hegron,G.: A Declarative design method for 3D Scene Sketch Modelling. In: *Computer Graphics Forum*, vol.12, no.3, 1993, pag.C223.

[Elb94] Elber, G., Cohen, E. - Toolpath generation for freedom surface models, *Computer-Aided Design*, Butterworths, Volume 26, Number 6 June, 1994

[Emm94] Emmer, U., Gerner, W., Muresan, V. - DICAM - Wood-manufactering CAD. *International Conference on Technical Informatics-Conti '94* , Tehnical University of Timisoara, 1994

[Ele91] Eles,P. - *Programarea concurentă în limbaje de nivel înalt*, Editura Stiintifică si Enciclopedică, Bucuresti, 1991

[Fai92] Faison - *Borland C++ 3 Object Oriented Programming*, Academic Press, 1992

[Fer93] Fernando, T., Fa, M., Dew, P.M.: Direct 3D Manipulation Techniques for Interactive Constraint-Based Solid Modelling. In: *Computer Graphics Forum*, vol.12, no.3, 1993, pg.C237

[Fla87] Flaquer, J.; Carbajal, A. si Mendez, M.A. - Edge - Edge Relationships in Geometric Modelling, *Computer-Aided Design*, Butterworths, Volume 19, Number 5 June, 1987

BUPT

[Fol90] Foley, J.D.: *Computer Graphics: Principles andPractice*. 2nd ed. Addison-Wesley, Reading, 1990.

[Fog90] Foger, M.L. - *Using Intergraph MicroStation PC*. Delmar, 1990

[Geh85] Gehani N. - *C For Personal Computers*, Computer Science Press, 1985

[Ges92] Gestner - *Maximizing Autocad* Vol.I, Academic Press, 1992

[Gla89] Glassner A.S. - *An Introduction To Ray Tracing*, Academic Press, 1989

[Gla90] Glassner, A.S. - *Graphics Gems*, Academic Press, Inc., 1990

[Gui94] Gui, J., Mäntylä, M. - Functional Understanding Of Assembly Modelling, *Computer-Aided Design*, Butterworths, Volume 26, Number 6 June, 1994

[Hel94] Hel-Or, Rappoport, A., Werman, M. - Relaxed Parametric Design With Probabilistic Constraints, *Computer-Aided Design*, Butterworths, Volume 26, Number 6 June, 1994

[Hal92] Halliday - *The First Book Of The Norton Desktop For Windows*, Academic Press, 1992

[Har92] Harley H. - *Assembler Inside And Outside*, McGraw Hill, 1992

[Hea92] Head - *Autocad 3D Book*, Academic Press, 1992

[Hof87] Hoffman, Cristiph M. si Hopcroft, John E. - Geometric Ambiguities in Boundary Representations, *Computer-Aided Design*, Butterworths, Volume 19, Number 3 April, 1987

[Hol192] Holzner, *C. - With Assembly Language*, Academic Press, 1992

[Hol292] Holzner, - *Quick C For Windows*, Academic Press, 1992

[Hol90] Holtz, F. - *Expert Systems In Engineerig*, Springer Verlag, 1990

[HP75] HP7475a Graphics Plotter - *Interfacing And Programming Manual*, Rs-232-C/Cci

[HuD89] Hu D. - *C/C++ For Expert Systems*, MIS Press, 1989

[IBM190] IBM - *Getting Started With PS/1 Printer*, International Business Machines, 1990

[IBM290] IBM - *User's Reference For PS/1 Printer*, International Business Machines, 1990

[Ion90] Ionescu, T.C. - *Structuri arborescente cu aplicatiile lor*, Editura Tehnica, Bucuresti, 1990

[Jur92] Jurcă, - I. *Programarea orientată pe obiecte în limbajul C++*, Eurobit, 1992

BUPT

[Ken93] Kent, - *Autocad Reference Guide*, Academic Press, 1993

[Kir92] Kirk, D. - *Graphics Gems III*, Academic Press, Inc., 1992

[Kre92] Kreite, A. - *Visualisation In Biomedical Microscopies, 3-D Imaging And Computer Applications* Vch, 1992

[Kri87] Kripac, Jiri - Algorithm for Splitting Planar Faces, *Computer-AidedDesign*, Butterworths, Volume 19, Number 6 July/Aug., 1987

[Maz94] Mazzetti, M., Ciminiera, L. - Computing CSG-Tree Boundaries as Algebraic Expresions, *Computer-Aided Design*, Butterworths, Volume 26, Number 6 Juni, 1994

[Kus88] Kussiak, A. - *Artificial Intelligence. Implications For CIM*, Springer Verlag, 1988

[Lat91] Latham, R. - *Dictionary of Computer Graphics Technology and Applications*, Springer, 1991

[Law90] Lawrence, B. - *Using Novell Netware*, Que Corporation, 1990

[Lei93] Leigh - *Solid Modeling With Autocad*, Academic Press, 1993

[Mag93] Maguire S. - *Writing Solid Code*, Microsoft Press, 1993

[Mar87] Marinescu, Gh. - *Probleme de analiza numerica rezolvate cu calculatorul*, Editura Academiei, 1987

[Mic187] Microsoft - *Microsoft C Optimizing Compiler For MS-DOS Operating System*, Vol.1 Microsoft Corporation, 1987

[Mic193] Microsoft - *Microsoft MASM - Environment And Tools*, Microsoft Corporation, 1993

[Mic287] Microsoft - *Microsoft C Optimizing Compiler For MS-DOS Operating System*, Vol.2 Microsoft, 1987

[Mic293] Microsoft - *The Programmer's PC Book*, Academic Press, 1993

[Mic387] Microsoft - *Microsoft C Optimizing Compiler For MS-DOS Operating System*, Vol.3 Microsoft, 1987

[Mic393] Microsoft - *Excel Software Development Kit*, Academic Press, 1993

[Mic487] Microsoft - *C for the MS-DOS Operating System*, Language Reference, 1984-1987

[Mic91] Microsoft - *C++ Tutorial*, Microsoft Corporation, 1991

[Mic92] Microsoft - *Microsoft C/C++ Run-Time Library Reference* Microsoft Press, 1992

BUPT

[Mor90] Mortenson, M.E. - *Computer Graphics Handbook: Geometry and Mathematics*. Industrial Press, 1990

[Mu194] Muresan, I. - Solid modeler for CAD, *International Conference on Technical Informatics-Conti '94*, Tehnical University of Timisoara, 1994

[Mu294] Muresan, I., s.a. - Roof Structure CAD, *International Conference on Technical Informatics-Conti '94,* Tehnical University of Timisoara, 1994

[Mu195] Muresan, I., Savii G. - *Abordari moderne in intergrarea sistemelor CAD/CAM,* Presa Universitară Română, 1995

[Mu295] Muresan, I., Savii, G. - *Concepte grafice in CAD/CAM. Tehnici interactive,* Presa Universitară Română, 1995

[Mu395] Muresan, I., Savii, G. - *Fundamentele CAD/CAM. Principii si metode,* Presa Universitară Română, 1995

[Mu495] Muresan, I., Savii, G. - *Grafica 3D*, UTT, 1995

[Mu595] Muresan, I., Marchis, D. - *Sisteme fuzzy bazate pe cunoastere,* Presa Universitara Romāna, 1995

[Mu695] Muresan, I., Muresan, V.- *Umbre si siluete in grafica 3D,* Presa Universitara Romāna, 1995

[Mu795] Muresan, I - *Tehnici pentru dezvoltarea de programe fără erori in limbajul C*. Presa Universitară Română, 1995

[Mur394] Muresan, I. - *Ingineria cunoasterii,* UTT, 1994

[MV194] Muresan, V. - Fast Hidden Lines Algorithm for CAGD, *International Conference on Technical Informatics-Conti '94*, Tehnical University of Timisoara, 1994

[MV294] Muresan, V. - *Sisteme expert cu aplicatii in CSG,* Referat nr.1, Universitatea Tehnică Timisoara,1994

[MV394] Muresan, V. - *Metasisteme expert cu aplicatii in rendering,* Referat nr.2, Universitatea Tehnică Timisoara,1994

[MV494] Muresan, V., Muresan, I. - *Inteligenta artificială*, UTT, 1994

[MV594] Muresan, V. - *Limbaje de nivel inalt orientate pe obiect*, UTT, 1994

[MV95] Muresan, V. - *Inteligenta artificiala*, Indrumător UTT, 1995

BUPT

[Nie95] Nielsen, K. - *Software Development With C++,* Academic Press, Profesional, 1995

[Ono90] Onodera, T; Kawai, S. - A Formal Model of Visualisation in Computer Graphic Systems. In: *Lecture Notes in Computer Science.* Springer, Berlin, 1990.

[Ove93] Oven, Jon: Computer Graphics and Computer-Aided Design Literature: A Keyword-indexed Bibliography for the year 1992. In: *Computer Graphics Forum,* vol.12, no.4, oct. 1993, pg.231-242.

[Per74] Pearson, C.E., - *Handbook of applied mathematics,* Van Nostrand Reinhold Company, Litton Educational Publishing, Inc., 1994

[Pha191] Pharlap - *286/DOS Extender Configuration Guide And Release Notes,* Pharlap Software Inc., 1991

[Pha291] Pharlap *386/Asm Reference Manual,* Pharlap Software Inc., 1991

[Pha391] Pharlap - *C/C++ User's Guide To Tnt DOS-Extender,* Pharlap Software Inc., 1993

[Pha90] Pham, D.T. - *Expert Systems In Engineering,* Springer Verlag, 1990

[Pit93] Pitter - *Application Software Tutorial,* Academic Press, 1993

[Pla93] Pla-Garcia, N. -Boolean Operations and Spatial Complexity of Face Octrees. In *Computer Graphics Forum,* vol.12, no.3, 1993, pag.C153.

[Pra93] Pratt - *Pascal-An Intro To Computer Science,* Academic Press, 1993

[Que193] Que - *1-2-3 Beyond The Basics,* Academic Press, 1993

[Que293] Que - *C Programmer's Tool Kit*, Academic Press, 1993

[Que393] Que - *Upgrading To Windows 3.1,* Academic Press, 1993

[Que493] Que - *Using Animator,* Academic Press, 1993

[Que593] Que - *Using Autocad,* Academic Press, 1993

[Que693] Que - *Using Microsoft Windows*, Academic Press, 1993

[Que793] Que - *Using PC Tool 7.1* , Academic Press, 1993

[Rak94] Raker, R. - *Inside Autocad,* Academic Press, 1994

[Req92] Requicha, A.A.G., Rossignac, J.R.: Solid Modelling and Beyond. In: *IEEE Computer Graphics and Applications,* 12 (5), pp.31-44, Sept.1992

[Rhe91] Rheingold, H. - *Virtual Reality.* Summit Books, 1991

BUPT

[Riv94] Rivest, L. Fortin, C., Morel, C. - Tolerancing a Solid Model With a Kinematic Formulation, *Computer-Aided Design*, Butterworths, Volume 26-Number 6 Juni, 1994

[Roc88] Rochkind M. - *Advanced C Programming For Displays* Prentice Hale, 1988

[Rog88] Rogers, D.F. - *Procedural Elements for Computer Graphics*, McGraw-Hill International Editions, 1988

[Ros89] Rosenblum, L. s.a.(editori) - *Scientific Visualisation. Advances and Challenges.* Academic Press Professional, Harcourt Brace & Company, Boston, 1993

[Ros92] Rossignac,J., Megahed,A., Schneider,B.D.: Interactive Inspection of Solids: Cross-sections and Interferences. In: *Computer Graphics* 26 (2), pp.353-360, July 1992.

[Sal87] Salmon, Rod; Slater, Mel - *Computer Graphics. Systems & Concepts*, Addison - Wesley Publishing Company, 1987

[Sam90] Samet, H. - *Application of Spatial Data Structures: Computer Graphics, Image Processing and GIS.* Addison-Wesley, 1990

[Sav94] Savii, G., Ogârcin, G., - Solid Boolean Operations Implementation on Microstation Shell, *International Conference on Technical Informatics-Conti '94*, Tehnical University of Timisoara, 1994

[Sch188] Schildt H. - *C - The Pocket Reference*, Mcgraw Hill , 1988

[Sch189] Schildt H. - *Borne To Code In C*, Osborne Mcgraw Hill, 1989

[Sch288] Schildt H. - *C Power User's Guide*, Osborne Mcgraw Hill, 1988

[Sch289] Schnupp P - *Expert Systems Lab Course*, Springer-Verlag, 1989

[Sch87] Schildt H. - *Artificial Intelligence Using C*, Osborne Mcgraw Hill, 1987

[Sha85] Shafer S. - *Shadows And Silhoettes In Computer Vision*, Kluwer-Academic Publishers, 1985

[Shi92] Shirley, R.S. - *Computer Graphics for Industrial Applications.* Prentice Hall, 1992

[Smi94] Smith - *Advanced Turbo* C, Academic Press, 1994

[Sny92] Snyder, J.M. - *Generative Modeling for Computer Graphics and CAD*, Academic Press, Inc., 1992

[Som94] Sommerson - *DOS Power Tools*, Academic Press, 1994

[Ste92] Stevens R. - *The C Graphics Handbook* Academic Press, 1992

BUPT

[Ste94] Stevens R. - *Object-Oriented Graphics Programing In C++*, Academic Press, 1994

[Tan89] Tănăsescu, A, Marinescu, I.D. - *Grafică asistată. Programe FORTRAN pentru reprezentări geometrice*, Editura tehnică, Bucuresti, 1989

[Tha90] Thalmann, D. - *Scientific Visualisation and Graphics Simulation*. Wiley, Chichester, 1990.

[Tha91] Thalmann, N.M. - *New Trends in Animation and Visualisation*. Wiley, 1991

[Wac94] Waco, D.L., Kim, Y.S. - Geometric Reasoning for Machining Decomposition, *Computer-Aided Design*, Butterworths, Volume 26, Number 6 Juni, 1994

[Web94] *Webster'dictionary*, Academic Press, 1994

[Wei91] Weiskamp, K.; Heiny, L.: *Power Graphics Using Turbo C*. Wiley & Sons, New York, 1991.

[Wri94] Write - *The Best Book Of Autocad*, Academic Press, 1994

[Yam93] Yamaguchi,F. Niizeki,M.: A New Paradigm for Geometric Processing. In: *Computer Graphics Forum*, vol.12, no.3, 1993, pag.C177.

[Yan85] Yankee, H.W. - *Engineering Graphics*, PWS Engineering ,1985

[Zei91] Zeid, Ibrahim - CAD/CAM Theory and Practice, McGraw-Hill, 1991

BUPT