Grafica pe calculator

Determinarea vizibilității in scene 3D

Victor Moraru

Vizibilitatea obiectelor dintr-o scena 3D

O scena 3D conține o informație completa despre fiecare dintre obiectele constituante. Pentru a reprezenta o instanta a acestei scene privita dintr-un anumit punct multe elemente ale obiectelor sunt invizibile și pot fi excluse din banda grafica

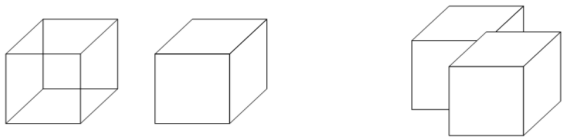
Vizibilitatea obiectelor dintr-o scena 3D

Vizibilitatea obiectelor dintr-o scena 3D depinde de:

- Poziția observatorului
- Poziționarea obiectelor din scena unul fata de celalalt
- Volumul vizual

Parțile nevizibile ale unui obiect 3D sunt:

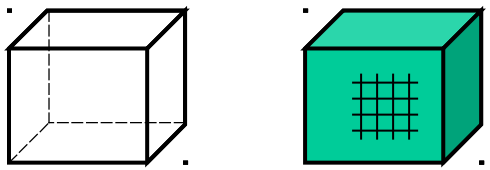
- Fețe auto-obturate
- Parți obturate de alte obiecte, aflate in fata obiectului in raport cu observatorul



Algoritmii de determinare a fețelor vizibile

Algoritmii de determinare a fețelor vizibile se împart în două categorii:

- algoritmi în spațiul imagine - care determină obiectul scenei vizibil în fiecare pixel ecran.
- algoritmi în spațiul obiect - care compară obiectele între ele, eliminând parțial sau total acele obiecte care nu sunt vizibile



Spatiu obiect Spatiu imagine

Algoritmii de determinare a feţelor vizibile

Algoritmii din spațiul imagine determină obiectul cel mai apropiat de observator de-a lungul proiectorului ce trece prin fiecare pixel. Afişarea unei scene 3D pe baza acestor algoritmi poate fi descrisă astfel:

pentru fiecare pixel ecran, P, execută

- determină obiectul O cel mai apropiat de observator de-a lungul proiectorului ce trece prin pixelul P;
- afișează pixelul P în culoarea obiectului O;

Efortul de calcul este proporțional cu (n*p), unde n este numărul de obiecte ale scenei, iar p reprezintă rezoluția ecranului.

Algoritmii de determinare a feţelor vizibile

Algoritmii în spațiul obiect au următoarea formă:

pentru fiecare obiect O al scenei execută

- determină părțile vizibile ale obiectului O (părțile care nu sunt obturate de alte parți ale sale sau de alte obiecte);
- afișează părțile vizibile ale obiectului O;

Efortul de calcul e proporțional cu (n*n), n fiind numărul de obiecte ale scenei 3D. Algoritmii în spațiul obiect sunt mai lenți (cu toate că, de regulă, n<p) şi mai dificili de implementat, fiecare pas al algoritmului fiind mai complex.

Terminologie

Object culling – eliminarea din banda grafică a obiectelor sau a grupurilor de obiecte care sunt in afara volumului vizual

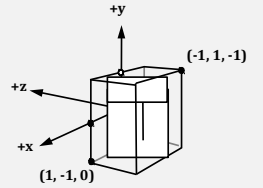
- poate fi efectuată prin algoritmi implementați într-o bibliotecă folosită de aplicația grafica 3D (motorul grafic)

Back face culling – eliminarea din banda grafică a fețelor auto-obturate ale obiectelor

Hidden surface removal – eliminarea părților nevizibile ale fețelor obiectelor:

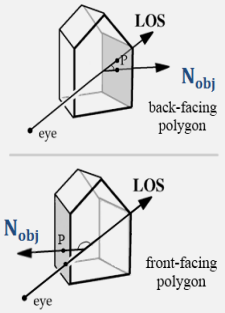
- este efectuată de GPU, ca operație raster la nivel de fragment (z-buffer)
- poate fi implementată şi la nivel de poligoane prin algoritmi implementați într-o bibliotecă folosită de aplicația grafica 3D (motorul grafic)

Fețe vizibile și invizibile



Volum de vedere canonic

Dacă normala e îndreptată in aceeași direcție in care privește observatorul fața respectivă este invizibilă

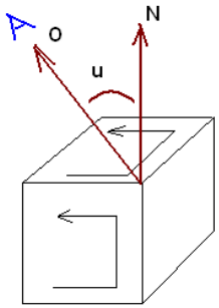


Determinarea fetelor auto-obturate pentru poliedre convexe (Back face culling)

Poate reduce substantial numarul de poligoane procesate in banda grafica.

Algoritmul:

- Observatorul si obiectul sunt raportati la acelasi sistem de coordonate.
- Conturul fiecarei fețe a obiectului este orientat in sens trigonometric atunci cand poliedrul este vazut din exterior.
- Observatorul este situat in afara obiectului.



Fața este vizibila daca : $0 <= u < 90^0$

sau

$0 < \cos(u) <= 1$

N: normala la fața

O: vectorul orientat către observator

Determinarea fetelor auto-obturate pentru poliedre convexe (Back face culling)

Produsul scalar:

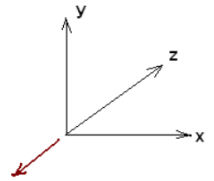
N•O = ||N|| * ||O|| * cos(u) ---> cos(u) = N• O / (||N|| * ||O||) = Nu•Ou

Conditia de vizibilitate: Nu • Ou > 0 (produsul scalar la versorilor)

Determinarea feţelor auto-obturate poate fi efectuată:

1. In sistemul coordonatelor globale (in care este definit si observatorul)
2. In sistemul de coordonate observator
3. In sistemul coordonatelor de decupare (după aplicarea transformării “model-view-projection”)

In cazul 3:



O[0, 0, -1] - este acelasi pentru orice punct al unui obiect

Nu • Ou = [nx, ny, nz] • [0, 0, -1] = - nz

Conditia de vizibilitate devine: nz <0

Observatorul este la infinit, pe axa z negativă, si privește în direcţia axei Z pozitive

Determinarea fetelor auto-obturate pentru poliedre convexe (Back face culling)

Calculul normalei la o faţă a poliedrului

Fie V1(x1, y1, z1), V2(x2, y2, z2), V3(x3, y3, z3), 3 varfuri succesive ale conturului fetei.

Normala la față se poate obține calculând produsul vectorial: (V1-V2)x(V2-V3)

N = (V1-V2)x(V2-V3) = [(z3-z2)*(y2-y1) – (z2-z1)*(y3-y2)]*i

- [(x2-x1)*(z3-z2) –(x3-x2)*(z2-z1)]*j

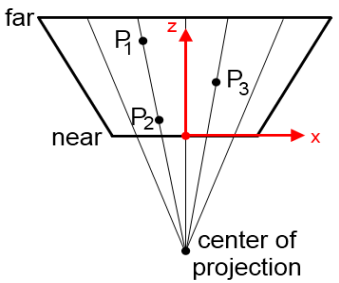
+[(x2-x1)*(y3-y2)- (x3-x2)*(y2-y1)]*k

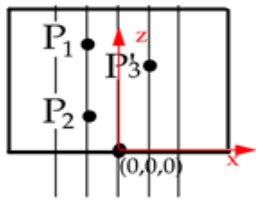
unde i, j, k sunt versorii direcțiilor axelor sistemului de coordonate carteziene 3D

Algoritmi din spațiul imagine

Aplicați în momentul rasterizării scenelor

- Care sunt pixelii care se suprapun?
- Cum calculăm distanța pană la pixelul cel mai apropiat?



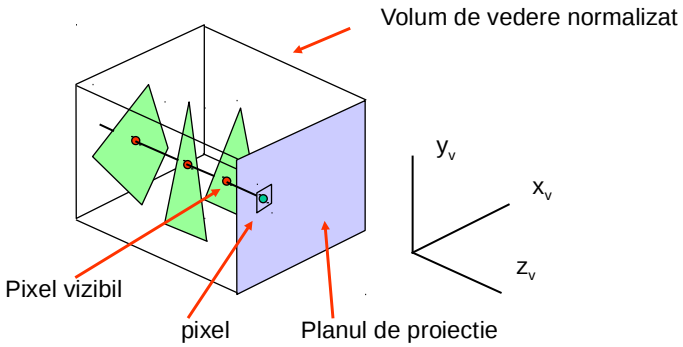


Algoritmul z-buffer

Algoritm de eliminare a parților nevizibile ale poligoanelor (fețelor obiectelor)

Executat in timpul rasterizării primitivelor (după transformarea de proiecție), de GPU:

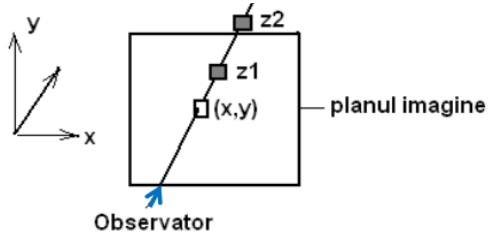
- Observatorul este la infinit, pe axa z negativa
- Asupra primitivelor se efectuează o proiecție ortografică în planul XOY



Algoritmul z-buffer

Principiu:

- Doua fragmente aflate pe același proiector, rezultate din rasterizarea a două primitive diferite, au coordonata z diferita.
- Fragmentul având coordonata z mai mica va fi afișat in pixelul (x,y).



Algoritmul z-buffer

Algoritmul z-buffer:

- Inițializează buffer-ul imagine la culoarea de fond
- Inițializează Z-buffer la coordonata z a planului din spate al volumului vizual (z=1)

Pentru fiecare fragment f(x,y,z) rezultat din rasterizarea unei primitive

dacă z< Z-buffer[y][x] atunci

Z-buffer[y][x] = z

actualizează culoarea pixelului (x,y) in buffer-ul imagine folosind culoarea fragmentului f

Algoritmul z-buffer

Primitivele grafice (poligoane, linii) in care a fost descompusa scena 3D sunt rasterizate in ordinea in care au fost transmise de programul de aplicatei

Buffer-ul imagine este actualizat pe măsura rasterizarii primitivelor, astfel:

dacă (fragmentul curent se proiectează in pixelul (x,y) si
z-fragment < z-fragment afișat in pixelul (x,y))
atunci
actualizează culoarea pixelului (x,y) in buffer-ul imagine, la culoarea fragmentului curent

Rezultat: este necesar sa se memoreze coordonatele z ale fragmentelor afișate in pixelii imaginii

Z-buffer :

- tablou bidimensional cu număr de elemente egal cu numărul de pixeli ai suprafeței de afișare
- in memoria plăcii grafice

Algoritmul z-buffer

P1(xP1, yr, zP1), P2(xP2, yr+1, zP2)

m- panta laturii (x1,y1,z1) – (x2,y2,z2)

xP2 = xP1 + 1/m

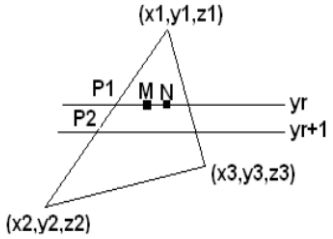
A*x + B*y + C*z + D = 0

zP1 = (-A*xP1 - B*yr –D)/C

zP2 = (-A*xP2 - B*(yr +1) – D)/C = (-A(xP1 + 1/m) - B*yr –B – D)/C = zP1 + (-A/m –B) = zP1 + K1

M(xM, yr, zM), N(xM + 1, yr, zN)

zN = (-A*(xM + 1) – B*yr – D)/ C = zM + (-A/C) = zM + k2

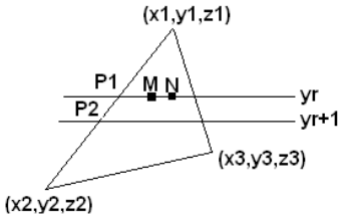


Algoritmul z-buffer

Calculul coordonatei z a unui fragment

Ultima transformare din lanțul de transformări efectuate asupra vârfurilor 3D conserva coordonata z a fiecărui vârf

Coordonatele z ale punctelor (fragmentelor) P1, P2, M, N se obțin prin calcul incremental, in algoritmul de rasterizare.



Algoritmul z-buffer

Se stie ca un punct (x, y, z) este in spatele unui poligon daca :

$$Ax + By + Cz + D < 0$$

unde A, B, C si D sunt parametrii planului

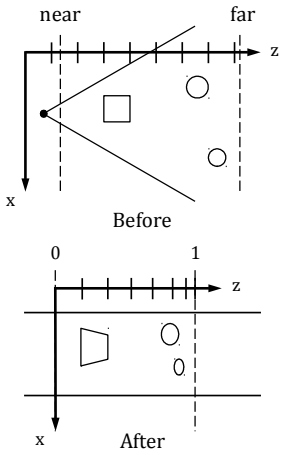
Daca punctul apartine planului:

$$Ax + By + Cz + D = 0$$

Algoritmul z-buffer

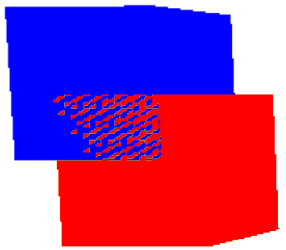
Probleme ale algoritmului z-buffer

- Compresia axei din spațiul post-perspectiva
- Obiectele îndepărtate au valori z foarte apropiate
- Pierderea preciziei despre profunzime
- Situație cunoscută ca z-fighting



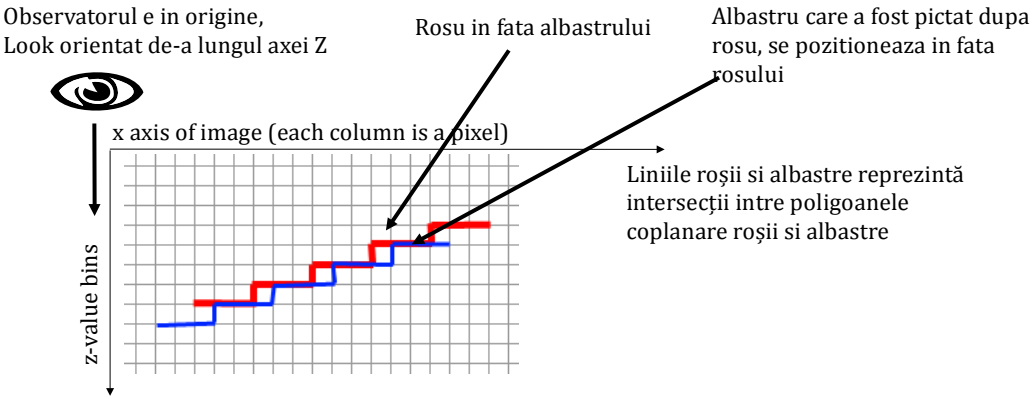
Probleme ale algoritmului z-buffer: z-fighting

- Situația poate fi observată atunci când două primitive au informații similare în z-buffer



Doua cuburi care se inersecteaza

Probleme ale algoritmului z-buffer: z-fighting



Algoritmul z-buffer

Aprecieri asupra algoritmului z-buffer

1) Numărul de comparații de valori z pentru un pixel: numărul de fragmente care se proiectează in acel pixel.

Timpul de calcul tinde sa devină independent de numărul de poligoane: în medie, numărul de pixeli acoperiți de un poligon este invers proporțional numărului de poligoane.

2) “Depth complexity”: numărul de suprascrieri ale unui pixel in buffer-ul imagine la generarea unui cadru imagine (calcul culoare fragment: model iluminare, utilizare texturi)

3) Valorile z (in z-buffer) sunt reprezentate prin numere întregi: 16, 32 biți - pierdere precizie

Algoritmul BSP (Binary Space Partitioning)

Poate fi folosit pentru:

- Eliminarea obiectelor aflate in afara volumului vizual (object culling)
- Eliminarea părților nevizibile ale fețelor obiectelor (hidden surface removal)

Intrarea: lista poligoanelor care compun scena 3D

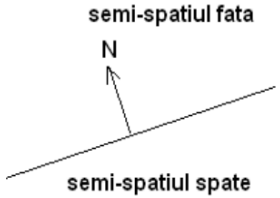
(nu are importanta din care obiect face parte fiecare poligon)

- Scena 3D este reprezentată printr-un arbore binar: arborele BSP
- Arborele BSP al scenei este independent de poziția observatorului (reprezintă scena in sistemul coordonatelor globale)

Algoritmul BSP (Binary Space Partitioning)

Construirea arborelui BSP al unei scene

Fiecare nod al arborelui corespunde unui plan de partiționare a spațiului 3D (de regula, planul unui poligon al scenei)



Fiecare plan de partiționare împarte spațiul in 2 semi-spatii:

– cel din faţa planului (de aceeași parte cu normala la plan)

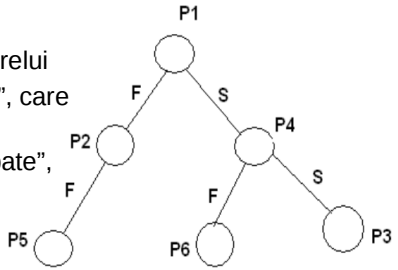
– cel din spatele planului

Se începe cu un poligon oarecare din listă (de regulă primul), pentru care se creează nodul rădăcină al arborelui

Poligoanele din semi-spațiul “faţă” formează “lista-față”, care va genera sub-arborele “față” al nodului

Poligoanele din semi-spațiul “spate” formează “lista-spate”, care va genera sub-arborele “spate” al nodului

Se alege un poligon din lista-față și se creează nodul rădăcină al sub-arborelui “față, etc.



Algoritmul BSP (Binary Space Partitioning)

Afișarea arborelui BSP al unei scene

- Ţine cont de poziția observatorului
- Afișare “back-to-front”: se începe cu poligonul cel mai îndepărtat de observator

- Se pornește din rădăcina arborelui și se avansează in arbore pana la frunze, în funcție de poziția observatorului față de planul atașat fiecărui nod

- Exemplu: se adaugă observatorul in scena 3D a arborelui din figura precedenta

Eliminarea obiectelor nevizibile din banda grafica (Object culling):

Dacă planul de partiționare al unui nod nu intersectează volumul vizual, atunci numai sub-arborele aflat de aceeași parte cu volumul vizual va fi afișat, celalalt sub-arbore fiind eliminat din banda grafică.

Algoritmul de creare a arborelui BSP

```
Arbore * creareBSP (Poligon * LP)
{ Poligon P, * ListaFata, * ListaSpate, * ListaNod;
  daca (LP este vida) return NULL;
  * alege un poligon P din LP;
  * elimina P din LP;
  ListaFata = ListaSpate = NULL;
  pentru (fiecare poligon Q din LP) executa
  {  daca (Q este in semispatiul fata al planului lui P) atunci
          * adauga Q in ListaFata
     altfel
        daca (Q este in semispatiul spate al planului lui P) atunci
         * adauga Q in ListaSpate
        altfel
           daca Q este in acelasi plan cu P atunci
              * adauga  in ListaNod
```

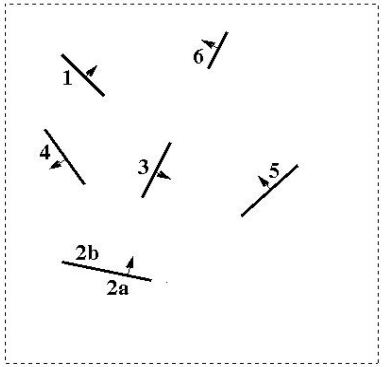
Algoritmul de creare a arborelui BSP

```
            altfel // Q este intersectat de planul lui P
               * divizeaza Q cu planul lui P
               * adauga fiecare poligon rezultat in ListaFata sau ListaSpate, in functie de pozitia sa
   } // pentru fiecare poligon
return combina(creareBSP(ListaFata), creareBSP(ListaSpate);
}
```

Afisarea arborelui BSP

```
void afisareBSP(arbore * A, Pozitie Observator)
{ daca (! A) return;
  daca (Observator este in semispatiul fata al planului radacinii) atunci
    { afisareBSP(A->spate); *afisare poligoane din nodul radacina; afisareBSP(A->fata);}
  altfel
    { afisareBSP(A->fata); *afisare poligoane din nodul radacina; afisareBSP(A->spate);}
}
```


Exemplu BSP Tree (Recursiv)



{1, 2, 3, 4, 5, 6}

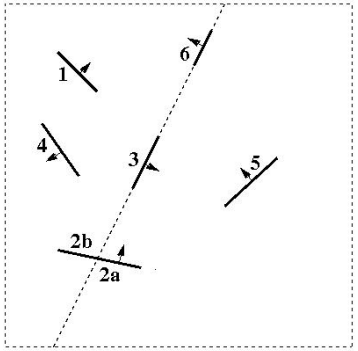
Arborele

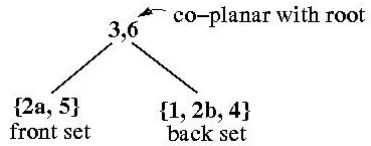
Un set de poligoane

Exemplu BSP Tree (Recursiv)

- Incepem cu un set de poligoane si cu un arbore vid
- Alegem un poligon drept radacina
- Impartim celelalate polifoane in 3 seturi: fata, spate, coplanar.
  - Fiecare poligon traversat de un plan este fractionat
- Repetam procesul recursiv construind arborele BSP

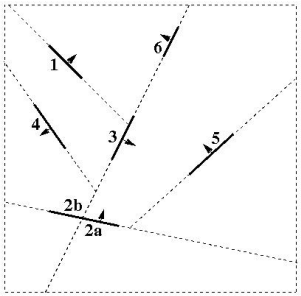
Exemplu BSP Tree (Recursiv)

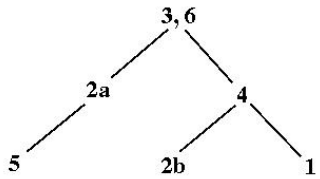




Selectam poligonul radacina

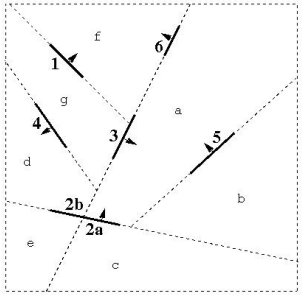
Exemplu BSP Tree (Recursiv)

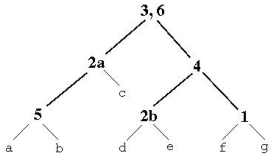




Recursively partition each sub-tree until all polygons are used up

BSP : o ierarhie a spatiilor





- Fiecare nod corespunde unei regiuni a spatiului

Algoritmul BSP: Aprecieri

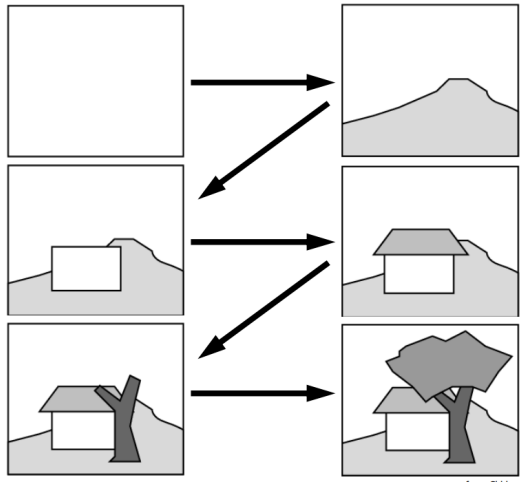
- Arborele nu trebuie sa fie construit pentru fiecare cadru imagine: avantaj pentru scenele statice
- Generarea unui cadru = execuția funcției de afișare
- Permite eliminarea din banda grafică a unui număr mare de poligoane care sunt in afara volumului vizual (Object culling)
- Diverse adaptări ale arborelui pentru cazul obiectelor in mișcare in scena 3D
- Numărul de suprascrieri ale unui pixel imagine este mai redus decât în algoritmul z-buffer

Sistemele grafice actuale, OpenGL și Direct3D, nu construiesc arborele BSP al unei scene

Construirea și afișarea arborelui BSP sunt funcții de management al scenei 3D, implementate de regula într-un motor grafic 3D

Algoritmul pictorului

Se desenează primitivele începând cu cea mai îndepărtată spre cea mai apropiata evitând astfel compararea profunzimii



Algoritmul pictorului

Intrare: lista poligoanelor care alcătuiesc scena 3D, transformate in spațiul de afișare

Forma generală a algoritmului

1. Se calculează “extensia” fiecărui poligon din listă pe axele OX, OY, OZ: paralelipipedul încadrator al poligonului, cu fețele paralele cu planele principale ale sistemului de coordonate

2. Se ordonează poligoanele crescător după coordonata z_{min} a fiecărui poligon: primul in lista va fi cel mai apropiat de observator

3. Se descompun poligoanele ale căror extensii pe axa OZ se suprapun, astfel încât extensiile lor pe axa OZ sa fie disjuncte

4. Se afișează (transmit in banda grafica) poligoanele începând cu ultimul din lista

Algoritmul pictorului

Pasul 3. Se descompun poligoanele ale căror extensii pe axa OZ se suprapun

- Este necesar numai dacă extensiile pe axa OZ se suprapun
- Sunt multe aplicații in care acest pas nu este necesar, poligoanele fiind amplasate in plane de Z constant: cartografie, generarea straturilor circuitelor imprimate, etc.
- Poate fi optimizat, știind ca nu întotdeauna atunci când extensiile pe axa OZ se suprapun este necesara descompunerea poligoanelor; testele se efectuează progresiv, in funcție de complexitatea calculelor presupuse

Principalul efort de calcul: sortarea listei de poligoane și descompunerea, dacă este necesara.

Întrebări ?