Grafică pe calculator (MLR5060)

Elemente de grafică 3_D

1. *Transformări geometrice uzuale;*
2. *Reprezentarea curbelor, suprafețelor şi corpurilor;*
3. *Observarea unui sistem 3_D de puncte;*
4. *Modelarea corpurilor;*
5. *Creşterea realismului imaginilor tridimensionale*
   - *Eliminarea suprafețelor acoperite*

Creşterea realismului imaginilor tridimensionale

Următoarele cursuri prezintă câteva metode de îmbunătățire a imaginilor în sensul apropierii calității lor de imaginile reale.

Dintre aceste metode prezentate în literatura de specialitate cum ar fi:

a) *eliminarea suprafețelor şi muchiilor acoperite* pentru extragerea elementelor de frontieră ascunse,
b) *perspectiva* pentru informațiile de profunzime,
c) *proiecțiile dinamice* pentru reprezentarea obiectelor în mişcare,
d) *indici de intensitate* sau *variația de culoare* utilizate pentru modificarea culorilor din adâncime,
e) *texturi* şi detalii de suprafață pentru reprezentarea microstructurilor fețelor,
f) *secționarea* cu un plan frontal utilizată la vizualizarea interiorului obiectului,
g) *iluminarea curpurilor* prin *utilizarea luminilor şi umbrelor* şi
h) *stereografia* pentru redarea în relief a a obiectelor tridimensionale,

am ales doar câteva pe care le-am considerat mai importante.

… Creşterea realismului imaginilor tridimensionale

1. Eliminarea suprafeţelor acoperite
2. Texturi
3. Lumină şi umbră
4. Stereografie

1. *Eliminarea suprafețelor acoperite*

Algoritmii de eliminare a zonelor *nevizibile* sunt în general mari consumatori de resurse (*memorie* şi *timp*). Există trei clase de algoritmi şi anume:

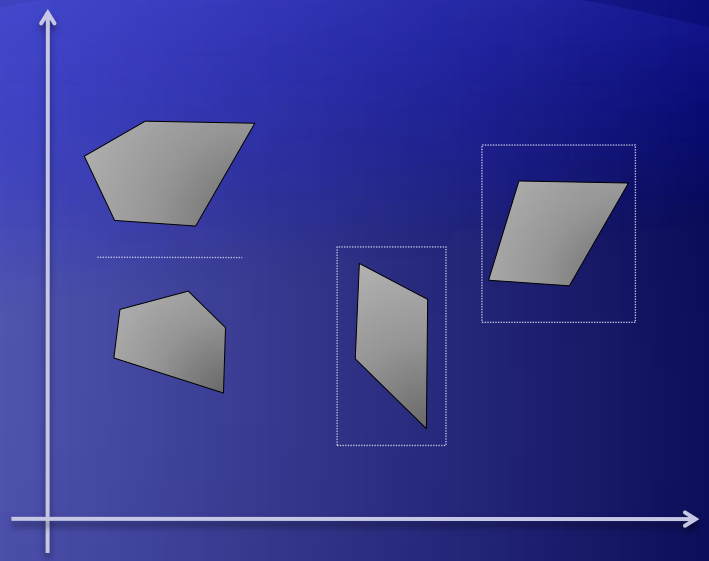
a) *Algoritmi spațiu-imagine* prin care se compară fiecare pixel cu fiecare față, deci de complexitate P*F (P=numărul de pixeli din fereastră, F=numărul de fețe ale corpului) pentru a decide cărei fețe îi aparține fiecare pixel în scopul colorării corespunzătoare;

b) *Algoritmi spațiu-obiect* care compară fețele două câte două, deci de complexitate F*F, pentru a decide relația de acoperire dintre fețe în scopul determinării unei ordini de transpunere a fețelor pe ecran astfel încât fețele mai îndepărtate se transpun primele iar cele mai apropiate vor fi desenate ultimele (se vor acoperi una pe alta).

c) *Algoritmi hibrizi* care utilizează elemente convenabile din primele două clase.

...1. *Eliminarea suprafețelor acoperite*

Pentru simplificarea comparațiilor dintre fețe sunt utilizate diverse tipuri de ferestre ecran (uni-dimensionale, bidimensionale sau chiar tridimensionale) care conțin fețele studiate şi pentru care comparațiile sunt mult mai uşoare (vezi figura alăturată).



Dintre algoritmii mai cunoscuți amintim următorii:

- *Buffer de adâncime (Z-buffer) ,*
- *Linie de baleiaj ,*
- *Sortarea în adâncime (depth-sort) şi*
- *Subdivizarea ariilor în arii elementare disjuncte.*

- Algoritmul ***Z-buffer***

Algoritmul utilizează o matrice ***D*** care conține pentru fiecare pixel P_{ij} cota z a punctului reprezentat prin P_{ij} (în urma proiecției) şi notată cu $z(P_{ij})$. Presupunem că observatorul se află pe axa Oz, aşa cum am văzut în paragraful precedent.

*Algoritmul **Z-Buffer** este:*

Pentru fiecare pixel P_{ij} din fereastra ecran execută

Colorează (P_{ij} , Culoarea_de_fond);

$D_{i,j}$:= minimă, negativă (minus infinit);

Pentru fiecare față f a obiectului execută

Pentru fiecare pixel P_{ij} corespunzător feței f execută

*Dacă $z(P_{ij}) > D_{ij}$ Atunci // (*)*

Colorează (P_{ij} , Culoarea_feței f);

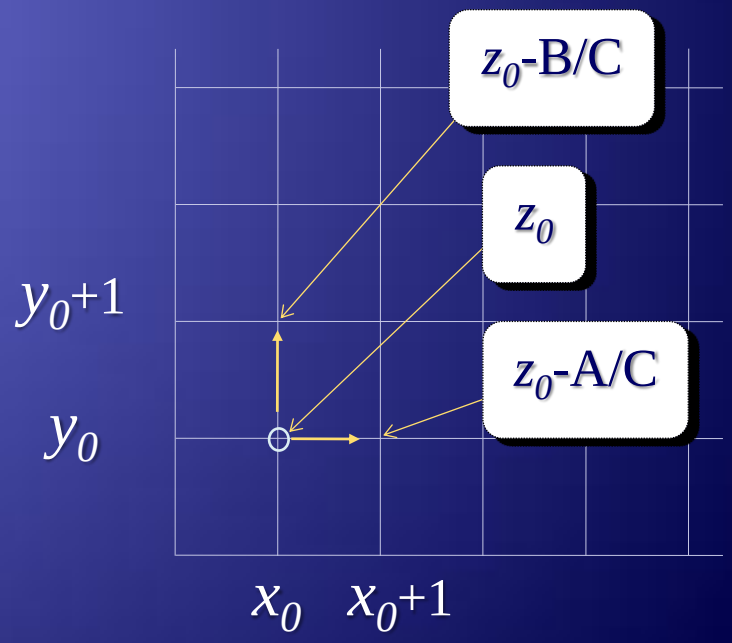
$D_{i,j}$:= $z(P_{ij})$;

*Sf_ **Z-Buffer**.*

Datorită faptului că acest algoritm utilizează multă memorie pentru a reprezenta matricea D, acesta se poate modifica pentru a rezolva această problemă pentru o linie sau pentru o fereastră ecran.

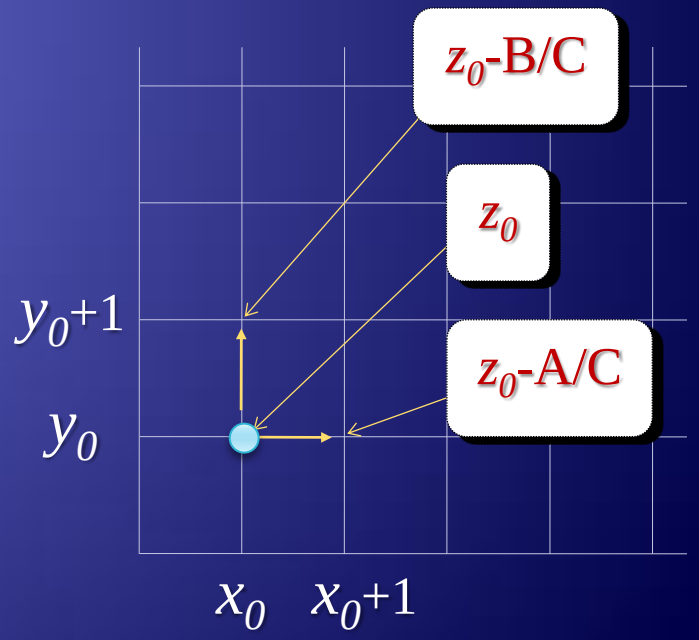
Pentru a micşora durata de execuție a algoritmului prezentat, calculele de distanță ($z(P_{ij})$) pot fi evitate prin determinarea ecuației planului ce conține fiecare față şi determinarea recurentă a cotei z. Pentru trei puncte ale unei fețe alese se poate determina ecuația planului ce conține fața :

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



Odată calculată cota z_0 a unui pixel P_{x_0,y_0} , se deduce că pentru punctele vecine, cotele sunt egale cu z_0-A/C respectiv z_0-B/C.

Acest algoritm, prezentat pentru eliminarea fețelor ascunse, se poate uşor adapta pentru eliminarea muchiilor ascunse şi de asemenea se poate extinde pentru fețe cu un anumit grad de transparență, modficând condiția de *vizibilitate* (*) din algoritm:



Dacă $z(P_{ij}) > D_{ij}$ *Atunci*

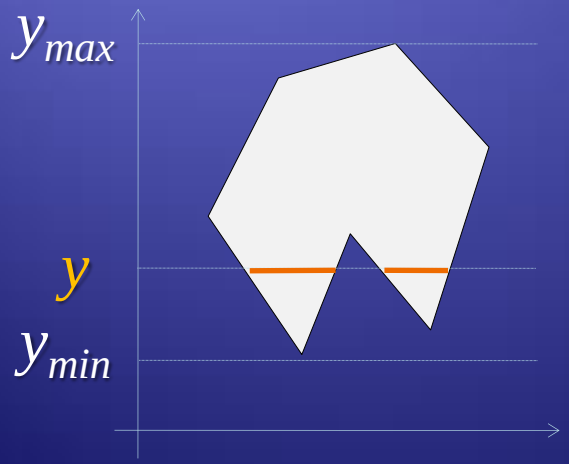
Colorează (P_{ij} , *Culoarea_feței* f);

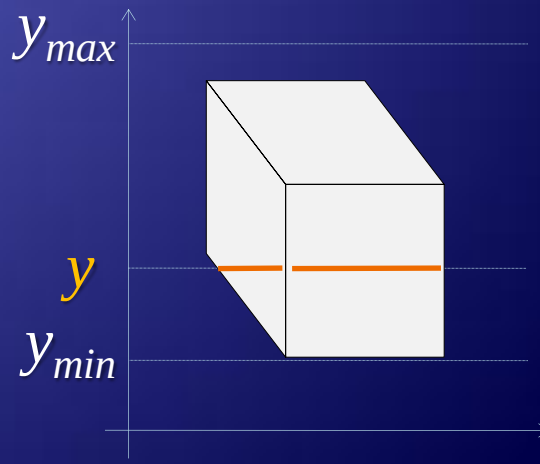
- Algoritmul *Linie de baleiaj*

Acest algoritm este o extindere a algoritmului de *umplere a interiorului uni poligon* şi poate fi utilizat în cazul reprezentărilor *poliedrale*, deoarece fețele în acest caz sunt *poligoane*.

Algoritmul de *umplere* este următorul:

Pentru fiecare linie de baleiaj $y := ymin, ymax$ *execută*
Determină intersecţiile x_i *cu fiecare latură;*
Ordonează crescător şirul x_i *;*
Pentru fiecare pereche de coordonate determinate
Trasează segmente orizontale de la x_{2*k-1} *la* x_{2*k}.

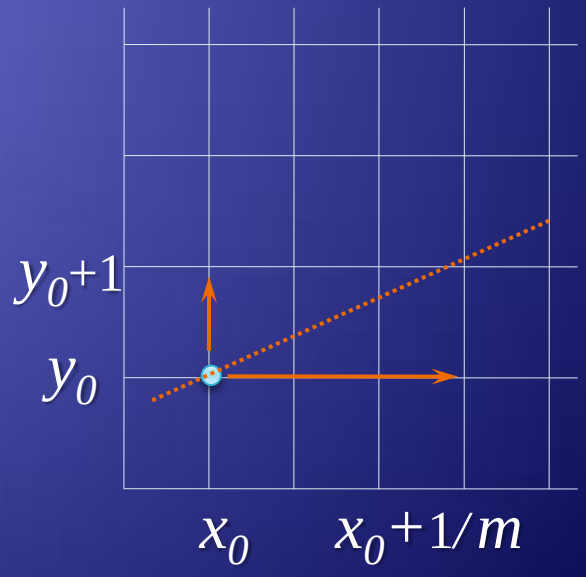




Pentru a nu mai calcula de fiecare dată intersecțiile liniei de baleiaj cu laturile poligonului se poate utiliza o listă (ordonată după x_{min}) de segmente *active* (notată cu *Lsa* care conține laturile intersectate la un moment dat de linia de baleiaj) fiecare conținând informațiile:

$$(y_{min}, x_{min}, y_{max}, 1/m)$$

unde m-reprezintă panta dreptei, iar $1/m$ creşterea pe x corespunzătoare creşterii cu o unitate pe y.



Algoritmul ***Linie de baleiaj*** este următorul:

*Algoritmul **Linie de baleiaj** este:*

Lsa:=∅;

Pentru fiecare linie de baleiaj $y := y_{min}, y_{max}$ *execută*

Dacă ∃ o latură s a poligonului pentru care $y=y_{min}$

Atunci Lsa:=Lsa ∪{s};

Intersecţiile x_i *sunt valorile* x_{min} *din Lsa;*

Pentru fiecare pereche de coordonate x_i

Trasează segmente orizontale de la x_{2*k-1} *la* x_{2*k} *;*

Pentru fiecare segment $s \in Lsa$ *execută*

$x_{min} := x_{min} + 1/m$;

Dacă $y = y_{max}$ *Atunci Lsa:=Lsa \ {s};*

*Sf_ **Linie de baleiaj** .*

Pentru algoritmul de eliminare a fețelor acoperite vom mai reține în lista de segmente şi fețele care formează fiecare muchie.

Pentru fiecare segment orizontal desenat se va determina care este fața cea mai apropiată care îl conține şi se va colora în culoarea feței respective până la următoarea intersecție.

Dacă în listă nu mai apar segmente, atunci ordinea rămâne aceeaşi, deci nu se mai determină față cea mai apropiată.

Este de asemenea utilă o informație binară pentru fiecare față care ne va spune dacă linia de baleiaj intră sau părăseşte fața respectivă pentru a counaşte care fețe sunt active şi trebuie luate în considerare la determinarea culorii de desenare.

- Algoritmul *Depth-Sort*

Algoritmul constă în ordonarea fețelor astfel încât cea mai depărtată de observator se transpune prima iar cea mai apropiată ultima. In felul acesta fețele se vor acoperi efectiv de către fețele mai apropiate.

Pentru aceasta se vor efectua următoarele:

- Se ordonează fețele după valoarea z *maximă;*
- Se transpun prin baleiaj fețele în ordinea determinată.

Pentru a determina această ordine va trebui să determinăm o relație de acoperire între fețe, după care putem să efectuăm ordonarea.

Două fețe F_i şi F_j pot fi în următoarele situații:

I. F_i *Acoperă* F_j $(F_i \supset F_j)$;

II. F_i *Acoperită de* F_j $(F_i \subset F_j)$;

III. F_i , F_j *Disjuncte* $(F_i \cap F_j = \varnothing)$;

IV. F_i , F_j *Se intersectează* $(F_i \cap F_j \neq \varnothing)$.

În prime trei situații putem rezolva problema colorării, iar în situația IV se vor împărți fețele astfel încât putem reduce problema la unul din cele trei cazuri favorabile (I - III).

În algorim vom utiliza o matrice de *acoperire* în care un element $M_{ij} = 1$ dacă fața *i* acoperă fața *j* (I) şi $M_{ij} = 0$ în rest. Coeficientul de acoperire A_i ne va da numărul de fețe acoperite de fața *i*. A_i se poate calcula ca suma elementelor de pe linia *i* din matricea de acoperire M.

Algoritmul este următorul:

Cât timp ∃ feţe F_i pentru care $A_i \geq 0$ execută
i:=Alege(F_i) { pentru care A_i=0 };
Transpune (F_i); A_i=-1;
Pentru toate feţele F_j care acoperă faţa F_i decrementează A_j;

Pentru a determina relaţiile dintre două feţe F_i şi F_j se procedează astfel:

- Se compară extinderile ecran şi dacă acestea nu se suprapun atunci feţele vor fi în relaţia III;
- Se determină ecuaţia planului ce conţine faţa F_i apoi se verifică dacă faţa F_j şi observatorul sunt de acceaşi parte a planului, caz în care suntem în situaţia II, altfel în situaţia I. Se procedează analog pentru a determina ecuaţia planului ce conţine faţa F_j apoi se verifică dacă faţa F_i şi observatorul sunt de acceaşi parte a planului determinat, etc.;
- Pentru zona comună se poate analiza chiar fiecare pixel pentru a stabili faţă cea mai apropiată.

- Algoritmul *Subdivizarea ariilor elementare*

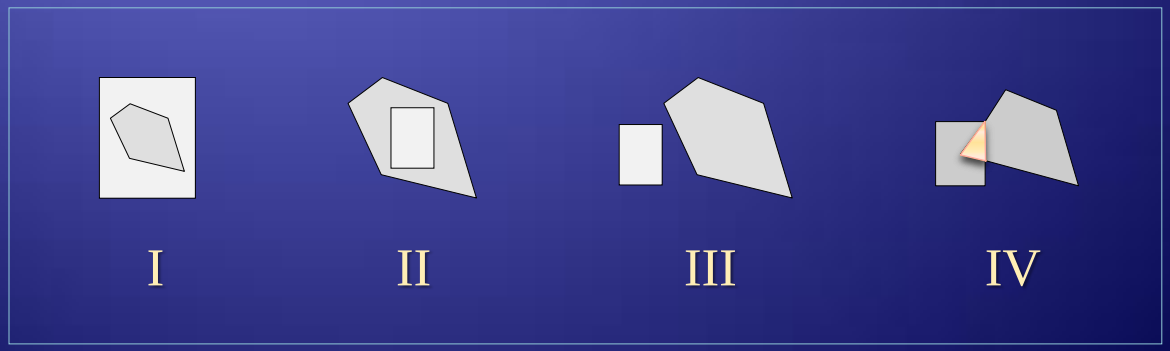
Această metodă compară zone dreptunghiulare ale ferestrei (numite *arii elementare* notate în cele ce urmează cu A_E) cu proiecții ale fețelor corpului. Relațiile dintre o arie elementară A_E şi o față F_i pot fi:

I. A_E *Conține* F_i $(A_E \supset F_i)$;

II. A_E *Conținută în* F_i $(A_E \subset F_i)$;

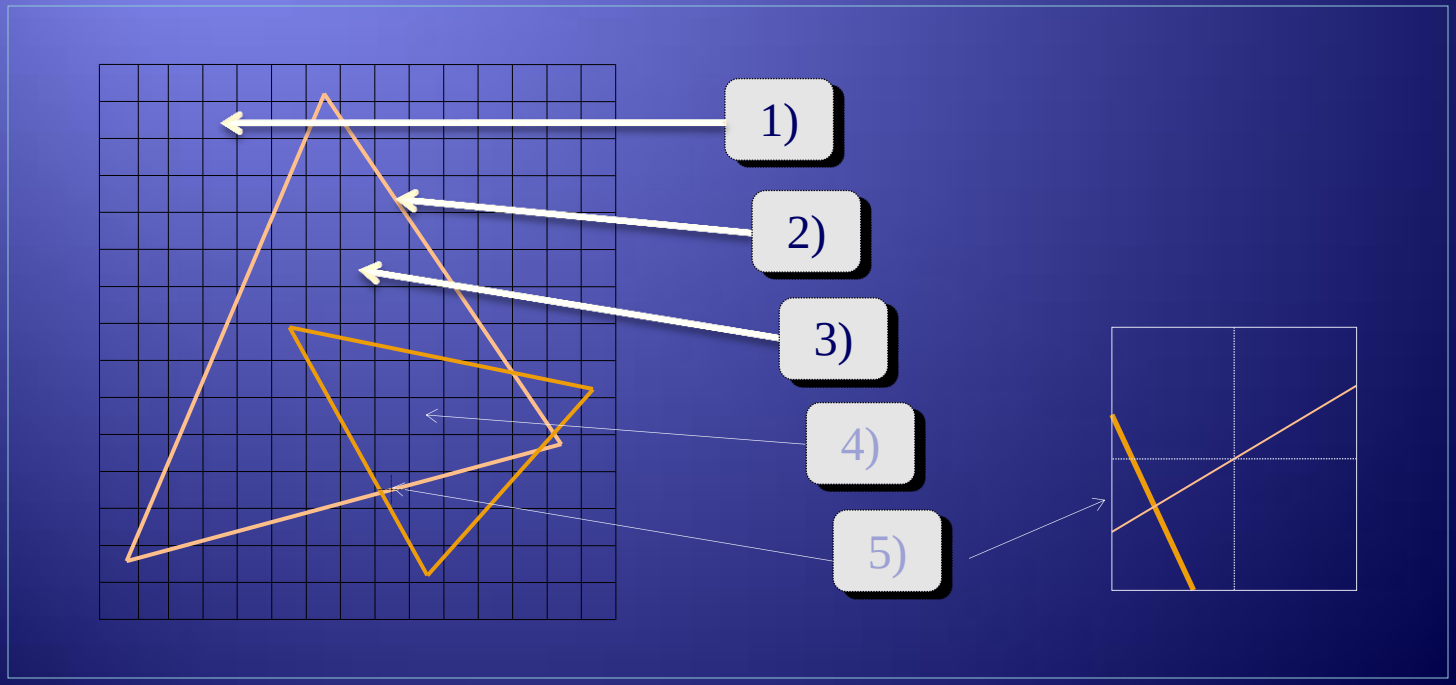
III. A_E , F_i *Disjuncte* $(A_E \cap F_i = \varnothing)$;

IV. A_E , F_i *Se intersectează parțial* $(A_E \cap F_i \neq \varnothing)$.



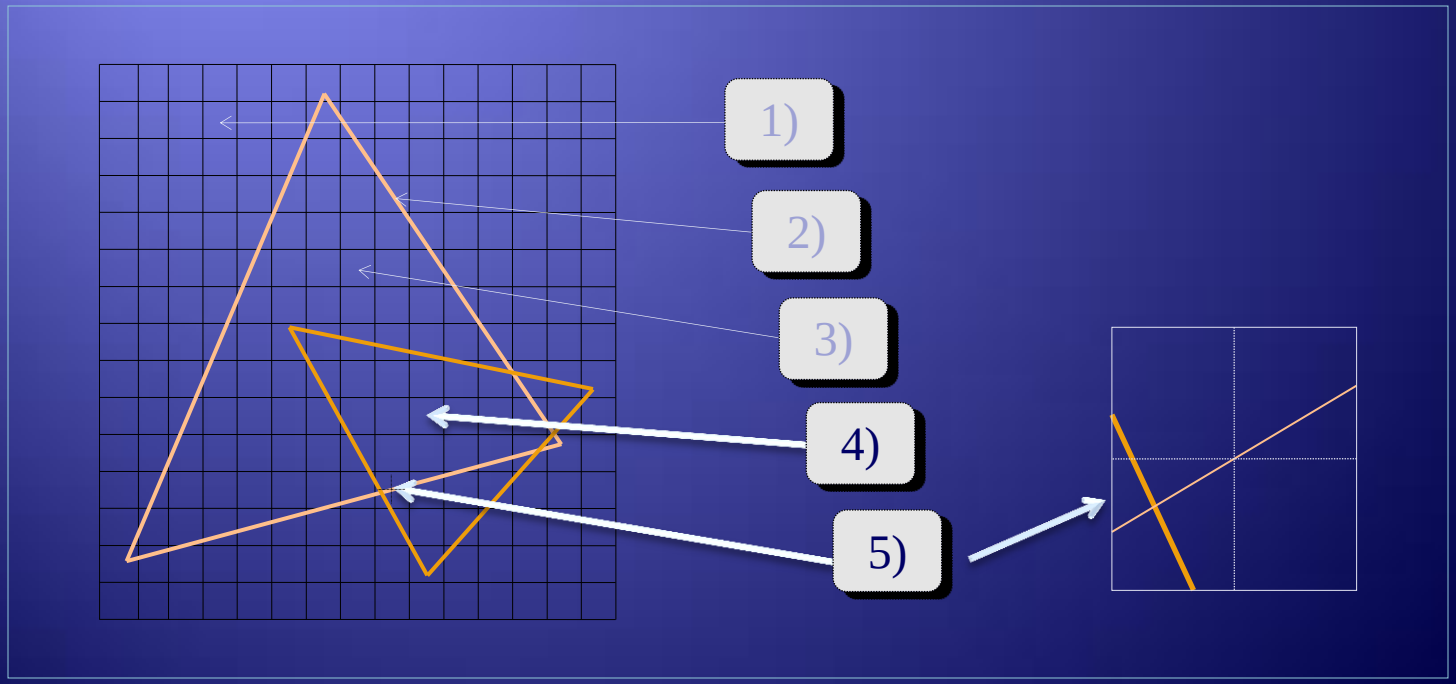
Desenarea conţinutului unei *arii elementare* se realizează astfel:

1) Dacă $A_E \cap F_i = \varnothing$ (III), $\forall F_i$ atunci A_E se colorează cu culoarea de fond;
2) Dacă există o singură faţă F_i pentru care $A_E \cap F_i \neq \varnothing$ (IV), atunci exteriorul $(A_E \setminus F_i)$ se colorează cu culoarea de fond, iar interiorul $(A_E \cap F_i)$ cu culoarea feţei F_i;
3) Dacă există o singură faţă F_i pentru care $A_E \subset F_i \neq \varnothing$ (I), atunci A_E se colorează cu culoarea feţei F_i;



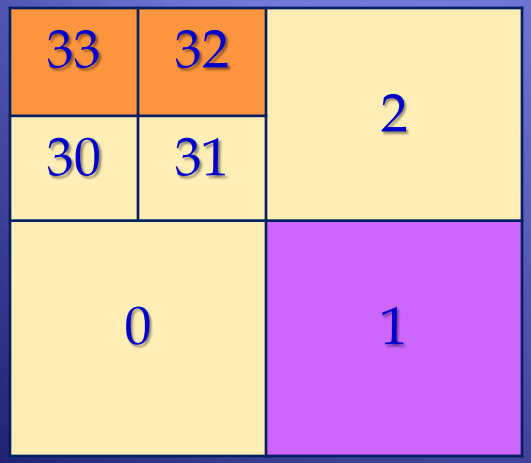
... Desenarea conţinutului unei *arii elementare* se realizează astfel:

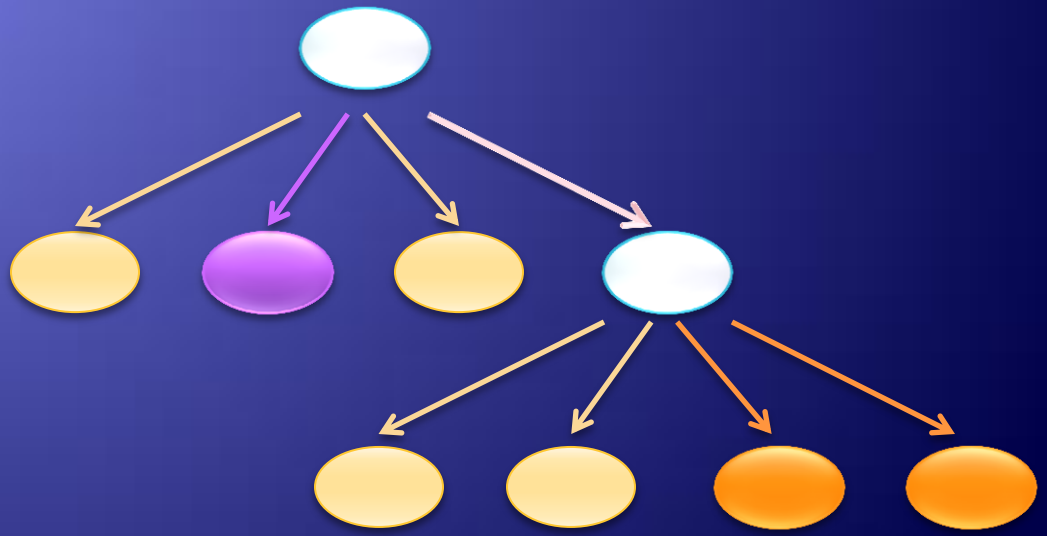
4) Dacă există mai multe feţe F_i pentru care avem $A_E \cap F_i \neq \varnothing$ (IV) dar există o singură faţă F_j pentru care $A_E \subset F_j$ şi F_j este cea mai apropiată, atunci A_E se colorează cu culoarea feţei F_j;
5) Dacă nu se poate aplica nici o regulă dintre primele patru descrise, atunci se împarte A_E în subarii (de exemplu în patru) şi se verifică din nou condiţiile 1-4. Dacă nici atunci nu sunt verificate, se împarte din nou, şi tot aşa până se ajunge la arie egală cu un pixel, situaţie în care preblema poate fi rezolvată.



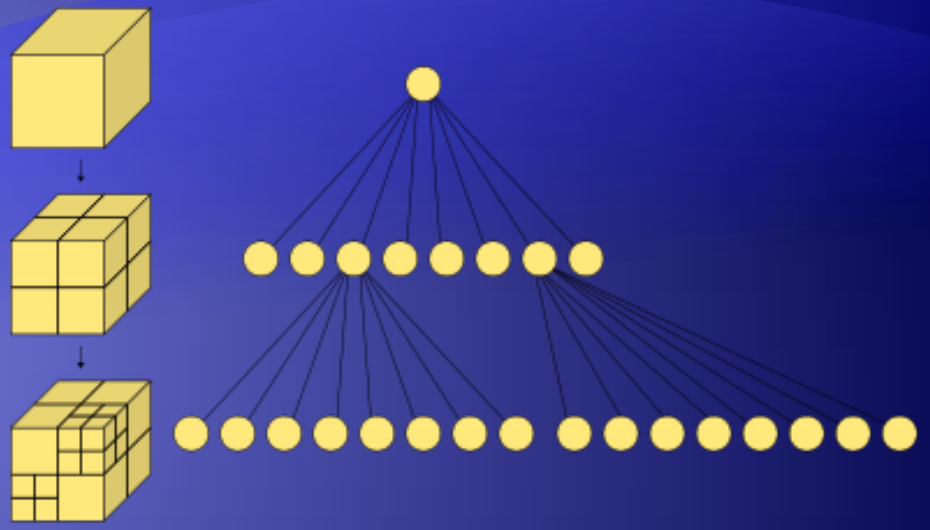
- *Metodele OCTREE*

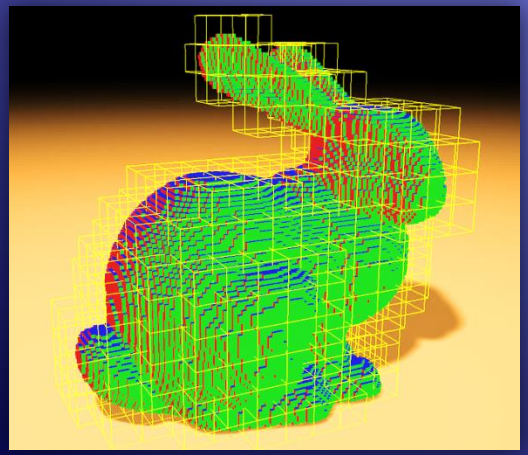
Aceste metode sunt utilizate pentru corpuri reprezentate prin *octree* (arbore de octanti – structura arborescenta in care nodurile corespund unor regini 3D, similara cu *quadtree* din 2D).

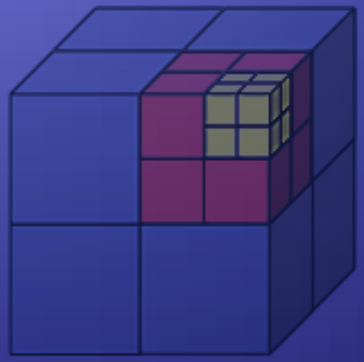


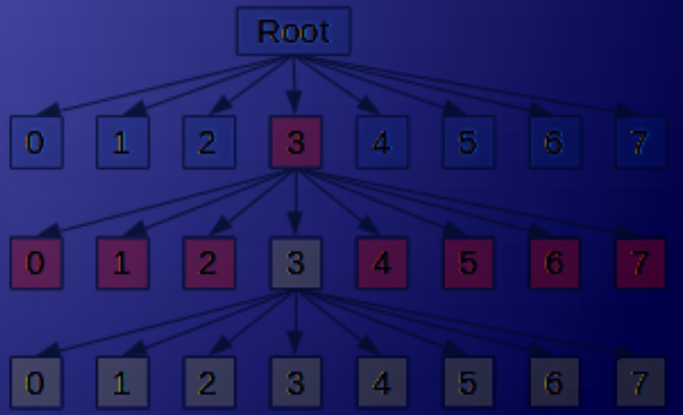


In reprezentarea 3D, spatiul este impartit in 8 octanti (0-7 sau combinatii dupa cele 3 directii: U/D, L/R, F/B). Daca toti voxelii dintr-un octant sunt de acelasi tip nodul va fi terminal (frunza) altfel se va reprezenta printr-un subarbore cu 8 fii, pana se ajunge la un voxel.









Compunerea obiectelor reprezentate prin metoda *Octree*.
Operații booleene: *Reuniune*, *Intersecție*, …

Pentru construirea obiectului rezultat se vor parcurge în paralel cei doi arbori (de la radacină spre frunze) comparând fiecare pereche de noduri.

O problemă importantă este găsirea nodului vecin. Un nod poate avea 26 de noduri vecine (6 fete + 12 laturi + 8 varfuri). Algoritmul propus de Samet determină un vecin pe o direcție precizată în două etape:

a) Se merge ascendent de la nodul inițial, până se găseste nodul părinte comun nodului inițial şi al celui verificat (posibil vecin).
b) Se merge descendent spre nodul vecin.

Acest algoritm caută nodul părinte pentru ambele noduri, apoi caută nodul vecin printre descendenții nodului părinte (determinat anterior).

Operațiile sunt similare cu cele 2-D (*quadtree*).

Referinte

Rodica Nabiu, Daniel Volovici , *Sisteme de Prelucrare Grafică*,

Editura *Albastră*, Cluj-Napoca, 1999.

W. G. Aref, H. Samet , *An algorithm for perspective viewing of objects represented by octrees*, Computer Graphics Forum, 14(1):59-66, March 1995, Also University of Maryland Computer Science Technical Report TR-2757, September 1991

http://www.spiralgraphics.biz/genetica/help/index.htm?environment_maps_explained.htm

Eric Andrew Lewis, *How a Quadtree Works ~ Watch a Quadtree in action*

http://ericandrewlewis.github.io/how-a-quadtree-works/

Temă

Pentru *Eliminarea suprafețelor acoperite* aplicați unul dintre următorii algoritmi:

- *Buffer de adâncime (Z-buffer) ,*
- *Linie de baleiaj ,*
- *Sortarea în adâncime (depth-sort) sau*
- *Subdivizarea ariilor în arii elementare disjuncte.*

Succes!