Eliminarea partilor nevizibile ale scenelor 3D din imagini - 1

Prof. univ. dr. ing. Florica Moldoveanu

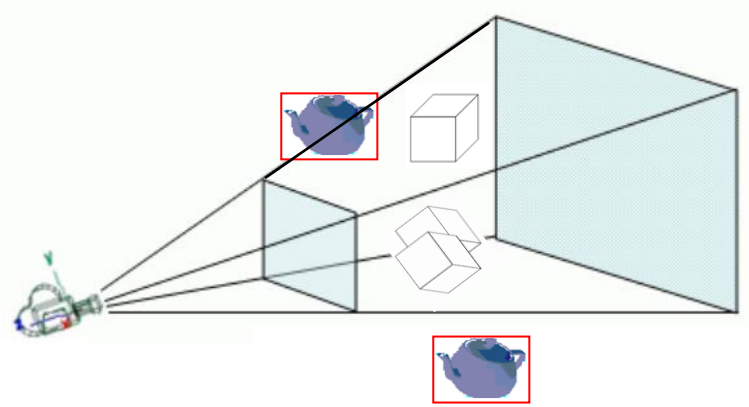
Eliminarea partilor nevizibile ale scenelor 3D, din imagini (1)

Vizibilitatea obiectelor dintr-o scena 3D, într-o imagine, depinde de:

– Pozitia şi orientarea camerei (observatorul virtual)

– Pozitionarea obiectelor din scena unul față de celalalt

– Volumul vizual



Partile unei scene 3D nevizibile într-o imagine sunt:

– Părți aflate total sau parțial în afara volumului vizual

– Fețe auto-obturate: fețele unui obiect obturate

de celelalte parti ale aceluiasi obiect

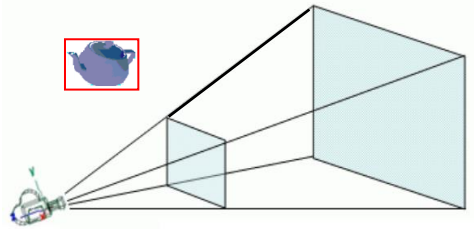
– Părți obturate (ascunse) de obiecte ale scenei,

aflate în fața lor în raport cu observatorul

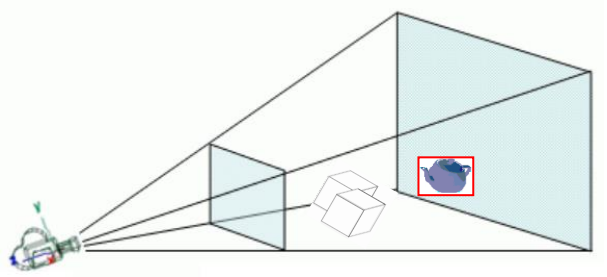
Eliminarea partilor nevizibile ale scenelor 3D, din imagini (2)

- Eliminarea părților dintr-o scenă 3D care nu sunt nevizibile într-o imagine este efectuata la mai multe niveluri:

1. **Eliminare grupuri de primitive (obiecte sau grupuri de obiecte)** - **Object culling**:



1.1. Frustum Culling - eliminarea din banda grafica a grupurilor de primitive aflate complet în afara volumului vizual: poate fi efectuata pe CPU, de biblioteca folosita de aplicatia grafica 3D (motorul grafic).



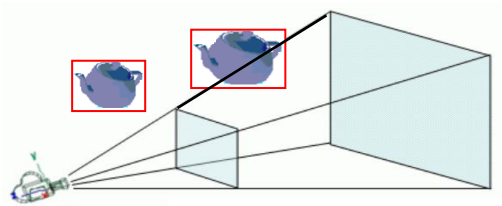
1.2. Occlusion Culling – eliminarea din banda grafica a grupurilor de primitive care sunt complet obturate de alte grupuri de primitive (obiecte): poate fi efectuata pe CPU, prin algoritmi implementati în motorul grafic folosit de aplicatia grafica.

Prin “object culling” se evita trimiterea în banda grafica a unui numar mare de primitive nevizibile în cadrul imagine curent!

Eliminarea partilor nevizibile ale scenelor 3D, din imagini (3)

2. **Eliminarea fețelor auto-obturate ale obiectelor** - ***Back face culling:*** 

- este efectuata de GPU**.**

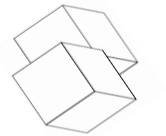


3. Eliminarea părților nevizibile ale primitivelor grafice:

3.1. Eliminarea primitivelor/partilor de primitive aflate în afara volumului vizual, prin operatia de **decupare (clipping)** efectuata de GPU.

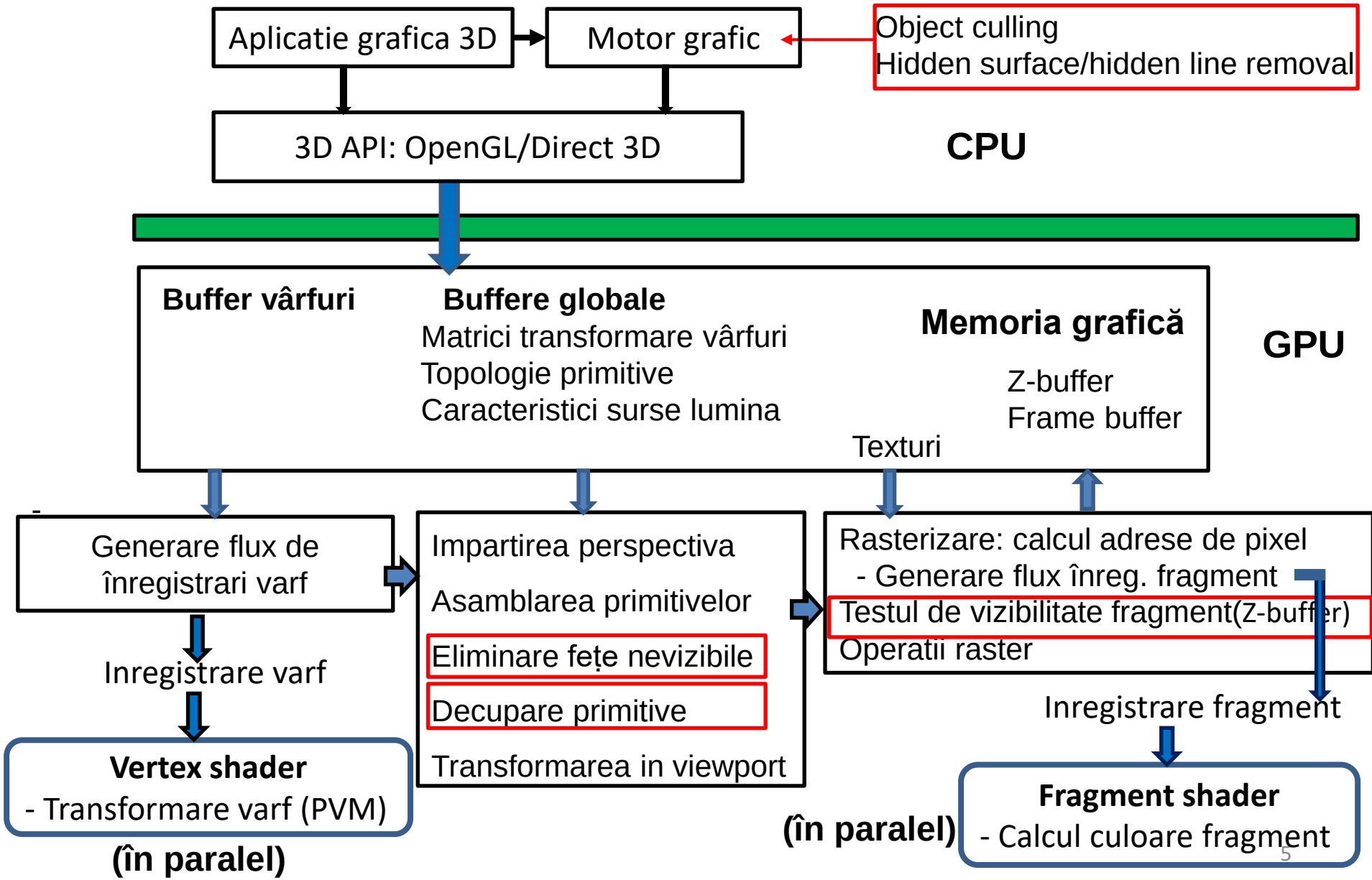
4. **Eliminarea fragmentelor nevizibile**

- este efectuata de GPU: fragmentele de primitive obturate de alte fragmente (aflate mai aproape de observator) sunt suprascrise la momentul rasterizarii (algoritmul Z-buffer)

Algoritmi "hidden surface/ hidden line removal": 

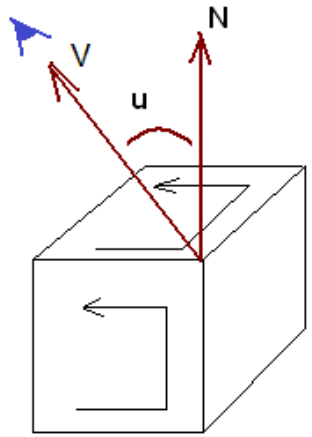
- implementati în motorul graphic; sunt eliminate (parti de) primitive, la nivel de fragment
- scop: cresterea vitezei de redare a scenei (algoritmul BSP, algoritmul Pictorului, ş.a.)

Eliminarea partilor nevizibile in banda grafica



Eliminarea fețelor auto-obturate pentru poliedre convexe (Back face culling)(1)

- Poate reduce substantial numarul de poligoane rasterizate.
- Clasificarea fetelor unui obiect in “fete din față (vizibile)” şi “fețe din spate” se face pe baza pozitiei obiectului fata de observator (camera)



Algoritmul se bazeaza pe urmatoarele presupuneri:

1. Observatorul si obiectul sunt raportati la acelasi sistem de coordonate.

2. Conturul fiecarei fețe a obiectului este orientat in sens trigonometric atunci cand obiectul este vazut din exterior; conditia asigura ca normala fiecarei fete vazuta din exterior este orientata catre spatiul exterior obiectului.

3. Observatorul este situat in afara obiectului.

O față este vizibila daca : $0 <= u < 90^o$ sau $0 < \cos(u) <= 1$ sau $\cos(u) > 0$

u: unghiul dintre normala la față si vectorul orientat catre observator

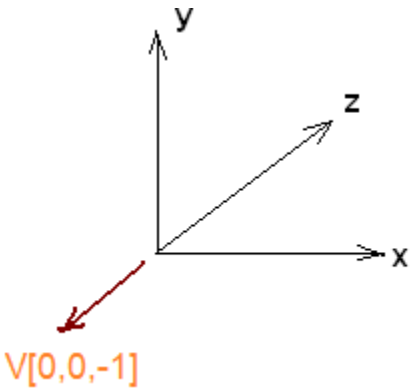
Produsul scalar: $N \bullet V = |N| * |V| * \cos(u) \rightarrow \cos(u) = N \bullet V / (|N| * |V|) = Nu \bullet Vu$

Conditia de vizibilitate: $Nu \bullet Vu > 0$ (produsul scalar al versorilor)

Determinarea fetelor auto-obturate pentru poliedre convexe(2)

Determinarea feţelor auto-obturate poate fi efectuata:

1. In sistemul coordonatelor globale (in care este definit si observatorul – camera virtuala)
2. In sistemul de coordonatelor observator (camera).
3. In sistemul coord. de decupare, transformand pozitia observatorului cu matricea P*V. In acest caz:



Sistemul coordonatelor de decupare

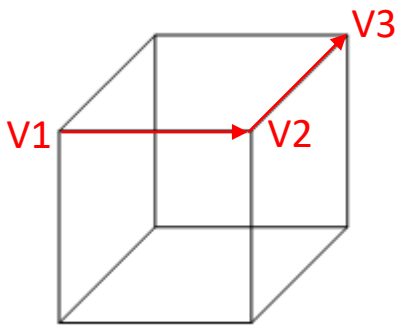
Nu • Vu = [nx, ny, nz] • [0, 0, -1] = - nz

Conditia de vizibilitate devine: nz <0

Dupa transformarea de proiectie pozitia observatorului este la infinit, pe axa z negativa.

Vectorul orientat catre observator este acelasi pentru orice punct al unui obiect.

Determinarea fetelor auto-obturate pentru poliedre convexe(3)



Calculul normalei la o faţă a unui poliedru

Fie V1(x1, y1, z1), V2(x2, y2, z2), V3(x3, y3, z3),
3 varfuri succesive ale conturului feţei.
Normala la faţă se poate obtine calculand produsul vectorial: (V1-V2)x(V2-V3)

$$\mathbf{N} = (V1-V2)\times(V2-V3) = \begin{vmatrix} i & j & k \\ x2-x1 & y2-y1 & z2-z1 \\ x3-x2 & y3-y2 & z3-z2 \end{vmatrix} =$$

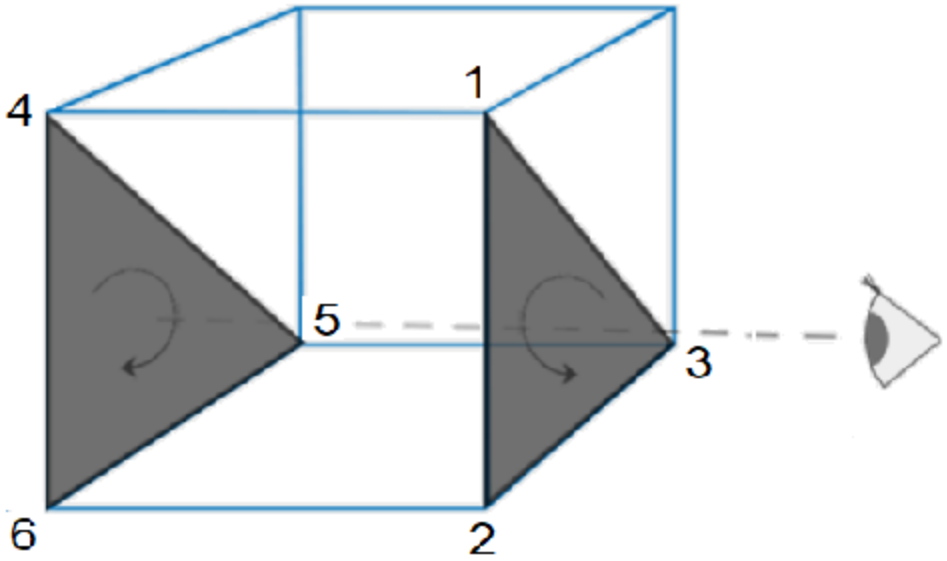
$$[(z3-z2)*(y2-y1) - (z2-z1)*(y3-y2)]*\mathbf{i} - [(x2-x1)*(z3-z2) - (x3-x2)*(z2-z1)*\mathbf{j} + [(x2-x1)*(y3-y2) - (x3-x2)*(y2-y1)]*\mathbf{k}$$

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

Determinarea fetelor nevizibile pe GPU (1)

- Presupunem ca pentru toate fetele unui cub s-a definit în aplicatie orientarea trigonometrica a contururilor lor, atunci and sunt vazute din exteriorul cubului: 1,2,3; 4,5,6, etc.
- Dupa transformarea de proiectie, in raport cu pozitia observatorului, ordinea varfurilor fețelor proiectate este:

- trigonometrica pentru fețele “din fata” (vizibile); ex. 1,2,3;
- anti-trigonometrica pentru fetele “din spate”; ex. ordinea 4,5,6 din pozitia observatorului este anti-trigonometrica – corespunde feței vazute din interior



Determinarea fetelor nevizibile pe GPU (2)

OpenGL da posibilitatea programatorului sa specifice care este orientarea fețelor “din față” şi care fețe trebuie sa fie eliminate (sunt nevizibile).

1. **Pentru activarea operatiei de eliminare a fetelor nevizibile (“face culling”) se apeleaza:**

   glEnable(GL_CULL_FACE); //implicit, operatia este dezactivata

2. **Pentru a specifica orientarea feţelor “din faţă”**: trigonometrica (GL_CCW)/sensul acelor de ceas (GL_CW): **glFrontFace(GL_CCW/GL_CW);**

3. **Pentru a specifica fețele care se doreste a fi eliminate:** din faţă / din spate/toate

   glCullFace(GL_FRONT/ GL_BACK/GL_FRONT_AND_BACK);

   In cazul **GL_FRONT_AND_BACK** nu vor fi afisate fețe, ci numai alte primitive: puncte, linii.

Exemplu: daca toate feţele unui obiect sunt orientate trigonometric, apeland glFrontFace(GL_CCW) şi glCullFace (GL_BACK) sau glFrontFace(GL_CW) şi glCullFace (GL_FRONT) se elimina feţele ale caror proiectii nu sunt orientate trigonometric.

Algoritmul z-buffer(1)

- Algoritm de eliminare a fragmentelor nevizibile ale primitivelor grafice.

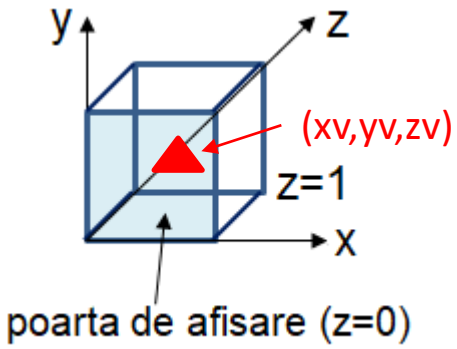
- Executat de GPU, fiind integrat in procesul de rasterizare a primitivelor.

❖ Primitivele rasterizate au coordonatele raportate la spatiul ecran

❖ Observatorul este la infinit, pe axa z negativa

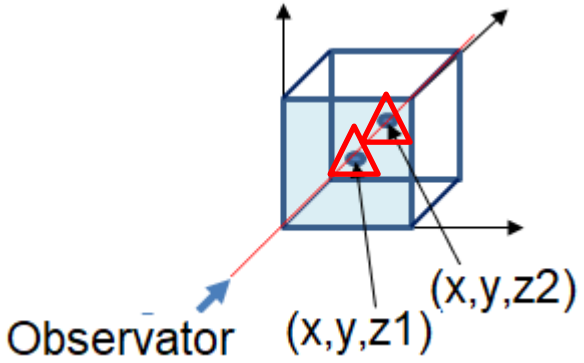
❖ Asupra primitivelor se efectueaza proiectie ortografica

in planul XOY (poarta de afisare)



Daca doua fragmente, rezultate din rasterizarea a doua primitive, se proiecteaza în acelaşi pixel, (x,y):

➢ Fragmentul avand coordonata z mai mica va fi afisat în pixelul (x,y).



Algoritmul z-bufer(2)

- Primitivele grafice (triunghiuri, linii) in care a fost descompusa scena 3D sunt rasterizate in ordinea in care au fost transmise din programul de aplicatie (nu sunt sortate).

- Buffer-ul imagine (frame buffer) este actualizat pe masura rasterizarii primitivelor, astfel:

  daca (fragmentul curent se proiecteaza in pixelul (x,y) si
  z-fragment < z-fragment afisat in pixelul (x,y))
  atunci
  actualizeaza culoarea pixelului (x,y) in bufer-ul imagine, la culoarea fragmentului curent

Rezulta: este necesar sa se memoreze coordonatele z ale fragmentelor afisate in pixelii imaginii

Z-buffer: - tablou bidimensional cu numar de elemente egal cu numarul de pixeli ai suprafetei de afisare

- în memoria grafica

Algoritmul z-bufer(3)

Algoritmul z-buffer (integrat in procesul de rasterizare):

*Initializeaza buffer-ul imagine (frame buffer) la culoarea de fond
*Initializeaza Z-buffer la coordonata z maxima (z=1)

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

* calculeaza culoarea fragmentului (in fragment shader)

//testul de vizibilitate (adancime)

daca z< Z-buffer[y][x] atunci

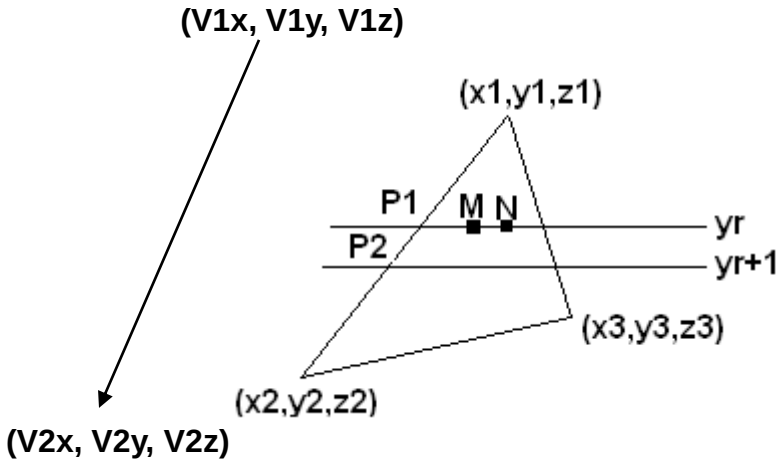
{ Z-buffer[y][x] = z

*actualizeaza culoarea pixelului (x,y) in buffer-ul imagine la culoarea fragmentului f

}

Algoritmul z-bufer(4)

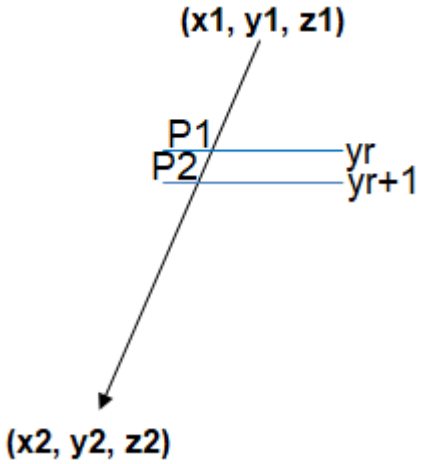
Calculul coordonatei z a unui fragment



Coordonatele z ale fragmentelor rezultate la rasterizarea vectorilor si a suprafetelor triunghiulare se obtin prin calcul incremental, in algoritmul de rasterizare.

Algoritmul z-bufer(5)

Calculul coordonatei z la rasterizarea vectorilor



x = x1 + t(x2-x1)
y = y1 + t(y2-y1) Ecuatiile parametrice ale vectorului
z = z1 + t(z2-z1)

yr, yr+1: ordonatele a 2 linii raster (ecran) care intersecteaza vectorul

$y(P1) = yr = y1 + t_{P1}(y2-y1) \rightarrow t_{P1} = (yr-y1)/(y2-y1)$

$z(P1) = z1 + (yr-y1)(z2-z1)/(y2-y1) = z1 + (yr-y1)*K,$
K este o constanta la rasterizarea vectorului

$z(P2) = z1 + (yr+1-y1)*K = z1 + (yr-y1)*K + K$

Z(P2) = z(P1) + K → coordonatele z ale fragmentelor de pe vector se obtin prin calcul incremental.

Algoritmul z-bufer(6)

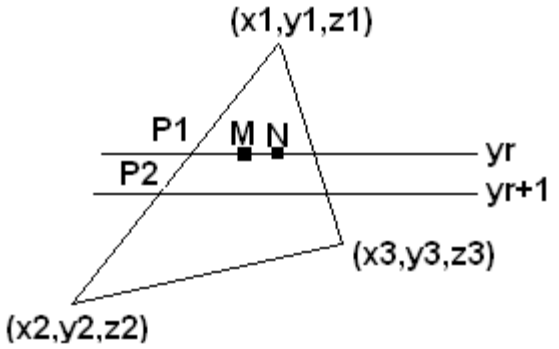
Calculul coordonatei z la rasterizarea unei suprafete triunghiulare

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

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

xP2 = xP1 + 1/m (curs - rasterizare)

A*x + B*y + C*z + D = 0 ec planului triunghiului



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) →

zP2 = zP1 + K1

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

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

Observatie: atunci cand un vector se suprapune pe suprafata unui poligon pot apărea defecte datorita modului diferit in care se calculeaza coordonatele z ale fragmentelor care se afiseaza in aceiasi pixeli!

Algoritmul z-bufer(7)

Aprecieri asupra algoritmului z-Buffer

1) Numarul de comparatii de valori z pentru un pixel = numarul de fragmente care se proiecteaza in acel pixel.

   Timpul de calcul al unui cadru imagine tinde sa devina independent de numarul de poligoane: in medie, numarul de pixeli acoperiti de un poligon este invers proportional cu numarul de poligoane.

2) Bufferul Z este un buffer de numere intregi: pentru o precizie buna, este necesara o reprezentare pe 32 biţi a valorilor din z-buffer.

3) "Depth fighting": doua fragmente care se afiseaza in acelasi pixel, cu z diferit in spatiul camerei (z-numere reale), au acelasi z in Z-buffer (z- numere intregi)→ culoarea pixelului depinde de ordinea rasterizarii primitivelor din care fac parte cele 2 fragmente.

4) "Depth complexity": numarul de suprascrieri ale unui pixel in buffer-ul imagine, la generarea unui cadru imagine (incluzand calcul culoare fragment, utilizare texturi)

- poate fi redusa prin tehnica "deffered rendering": se calculeaza culoarea numai pentru fragmentele vizibile in pixelii imaginii.