

Cursul

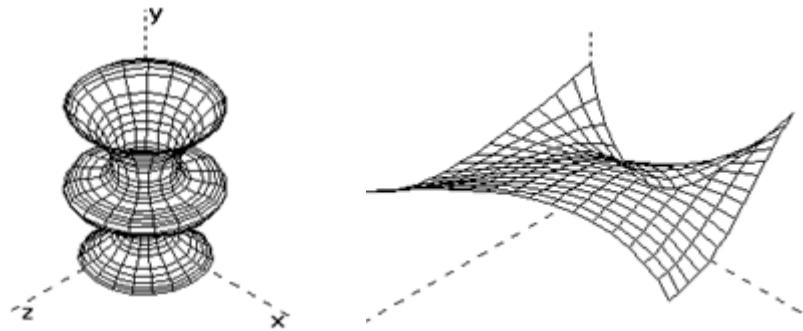
*Elemente de Grafica pe
Calculator*

Profesor Florica Moldoveanu

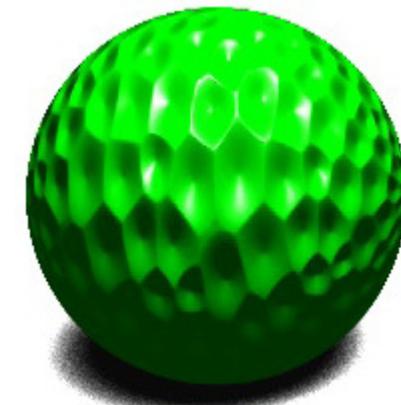
Sisteme Grafice: clasificare

- **De sinteza a imaginilor:** crearea de imagini ale unor obiecte / scene reale sau virtuale, statice sau dinamice, animatie – **Computer Graphics**
- **De prelucrare si analiza a imaginilor /Sisteme cu viziune** (Computer Vision): extragerea de informatii din imagini digitale produse de dispozitive de achizitie(camera video, scanner)
 - Aplicatii in: analiza imaginilor medicale, recunoasterea fețelor, a caracterelor de text, recunoasterea terenului din imagini satelitare, in robotica, etc.
- **De vizualizare sau reconstructie si vizualizare 3D a unor obiecte/ fenomene reale sau virtuale pornind de la date volumetrice** (valori scalare asociate unor pozitii spatiale)
 - Combina metode specifice cu metode si algoritmi de Computer Graphics si Computer Vision
 - Aplicatii in: vizualizarea 3D a organelor interne pornind de la imagini tomografice, vizualizarea 3D a dinamicii moleculare, a datelor meteo, etc.

Computer graphics (1)

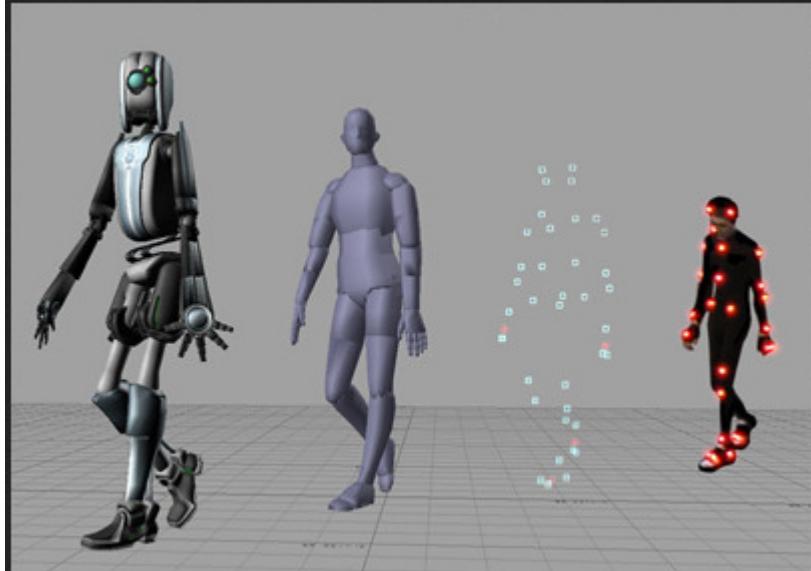


Modelarea suprafetelor 3D



Simularea detaliilor suprafetelor (texturi)

Animatia personajelor prin captura miscarii



Reflexia si refractia mediului inconjurator



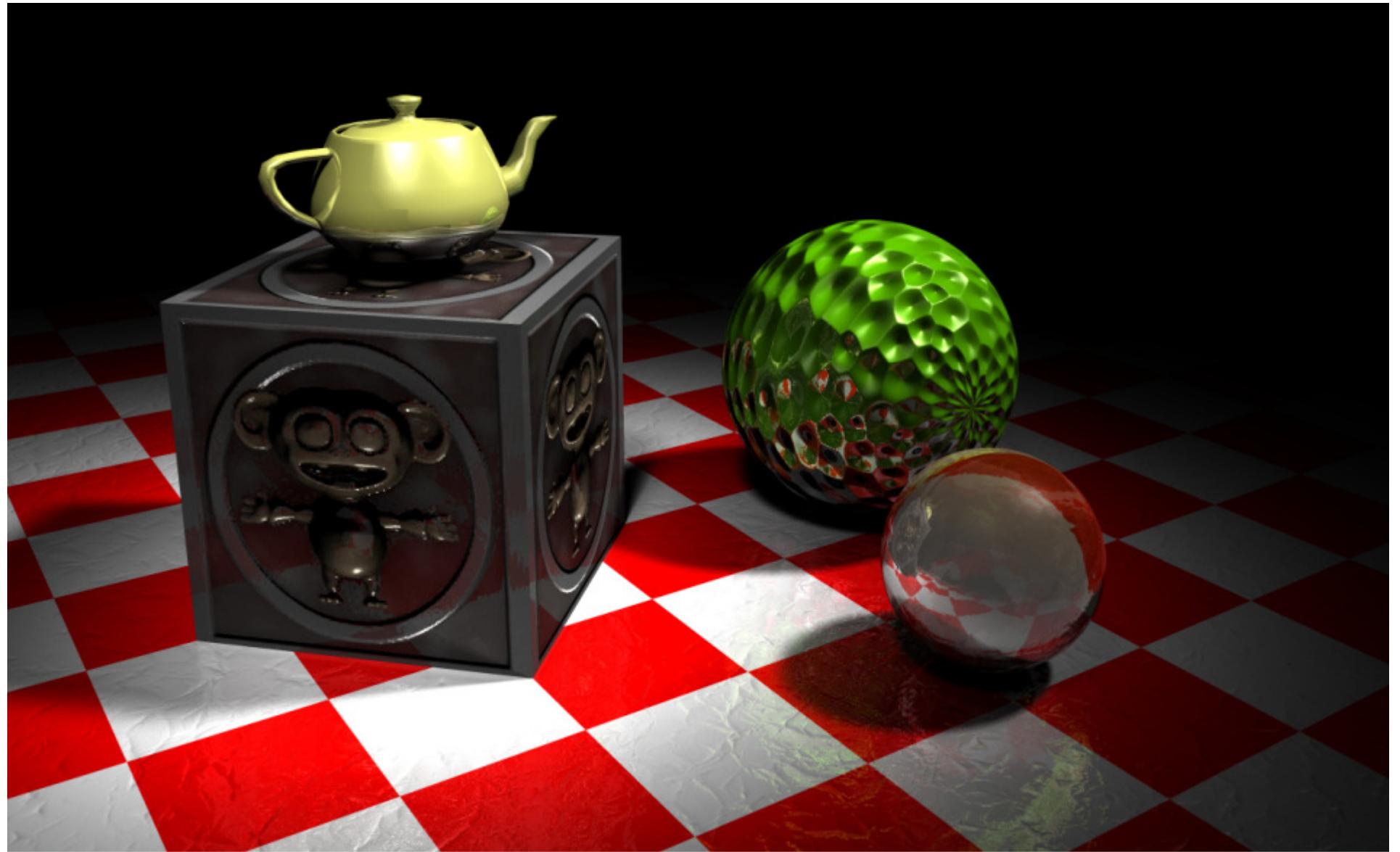
Computer graphics (2)

Imagine sintetizata prin Ray-tracing

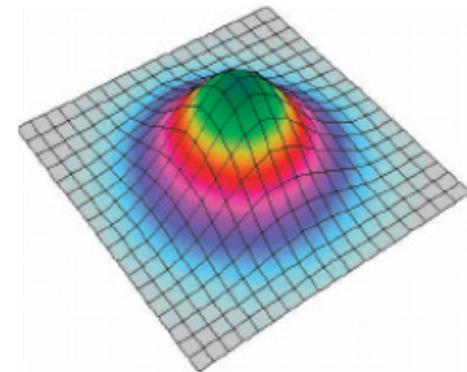


Computer graphics (3)

Imagine sintetizata prin Ray-tracing



Prelucrarea si analiza imaginilor (1)



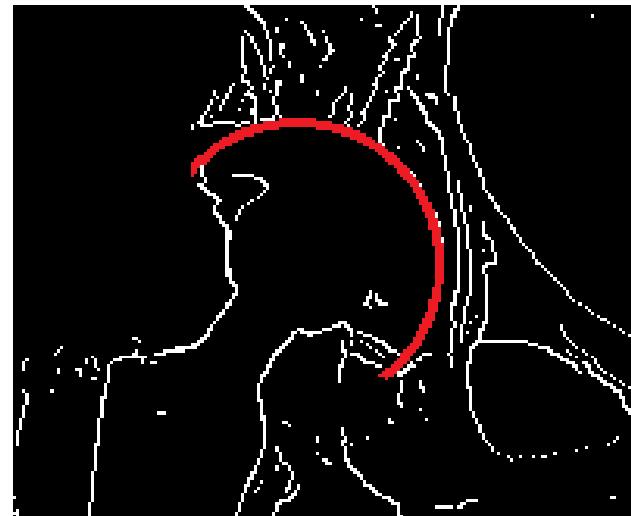
Eliminarea zgomotelor



Extragerea frontierelor →

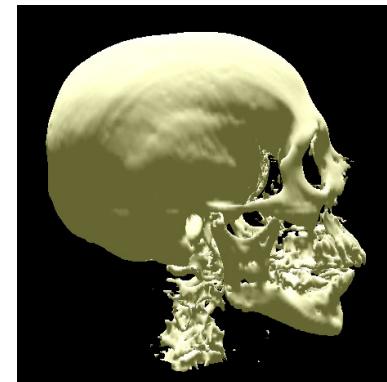
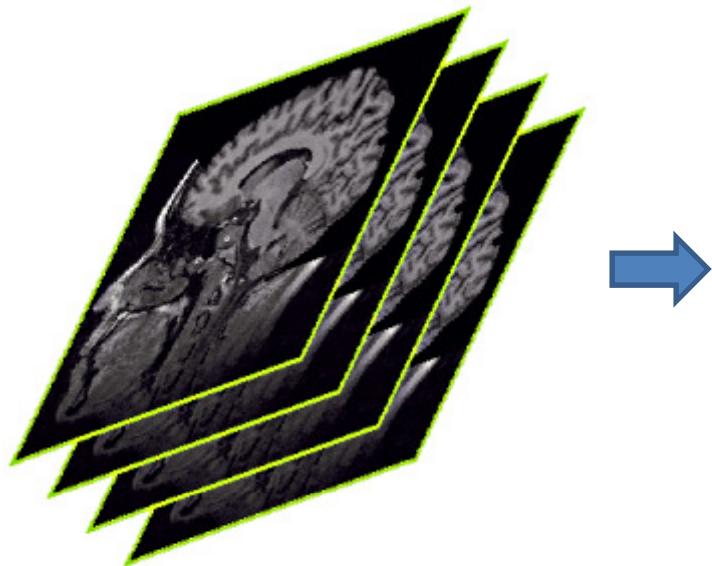


Prelucrarea si analiza imaginilor (2)

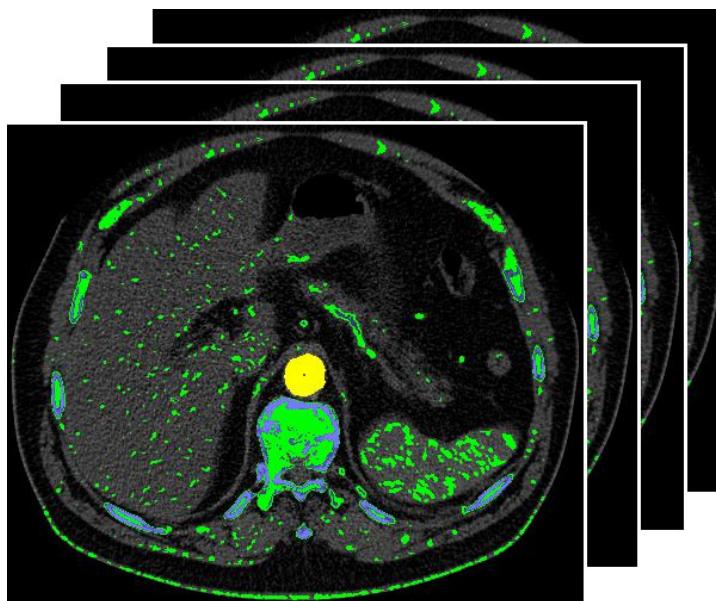


Analiza imaginilor medicale
(extragerea formei osului femural si efectuarea
de masuratori automate)

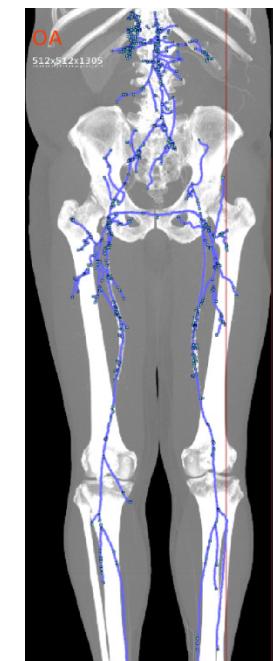
Reconstructia si vizualizarea volumelor



Reconstructia / vizualizarea
volumelor pornind de la
imagini 3D



Vizualizarea vaselor de sânge

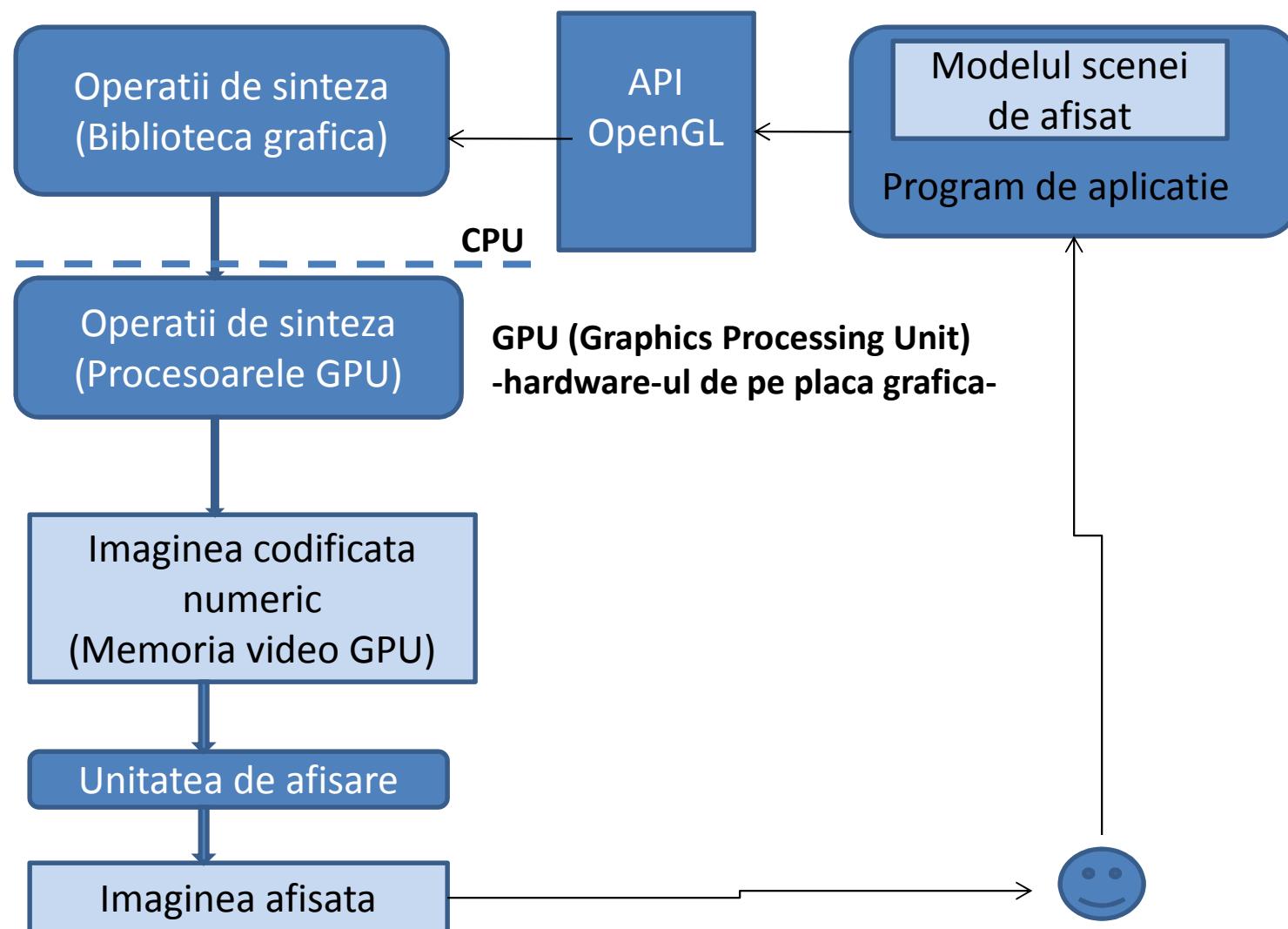


Sisteme grafice de sinteza(1)

Cursul Elemente de Grafica pe Calculator (Computer Graphics):

- Fundamente ale sintezei imaginilor digitale
- **Aplicatii ale sistemelor grafice de sinteza:**
 - Proiectarea asistata de calculator: arhitectura, constructii, design industrial, confectii
 - Cartografie
 - Productia de filme
 - Jocuri pe calculator
 - Spatii virtuale interactive,
 - Etc.

Sisteme grafice de sinteza(2)



Sisteme grafice de sinteza(3)

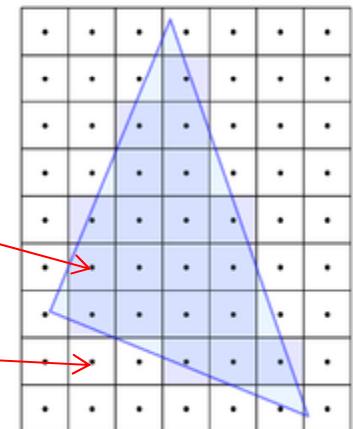
Modelul scenei:

- Inclus in programul de aplicatie
- Alcatuit din:
 - Primitive grafice 2D, 3D (linii, poligoane) sau obiecte complexe
 - Transformari de aplicat obiectelor in scopul compunerii si vizualizarii scenei
 - Proprietati ale suprafetelor obiectelor din scena(culoare, textura, reflexie/refractie a luminii)
 - Surse de lumina (pozitie, caracteristici de culoare)
 - Pozitia observatorului in scena 3D

Sisteme grafice de sinteza(4)

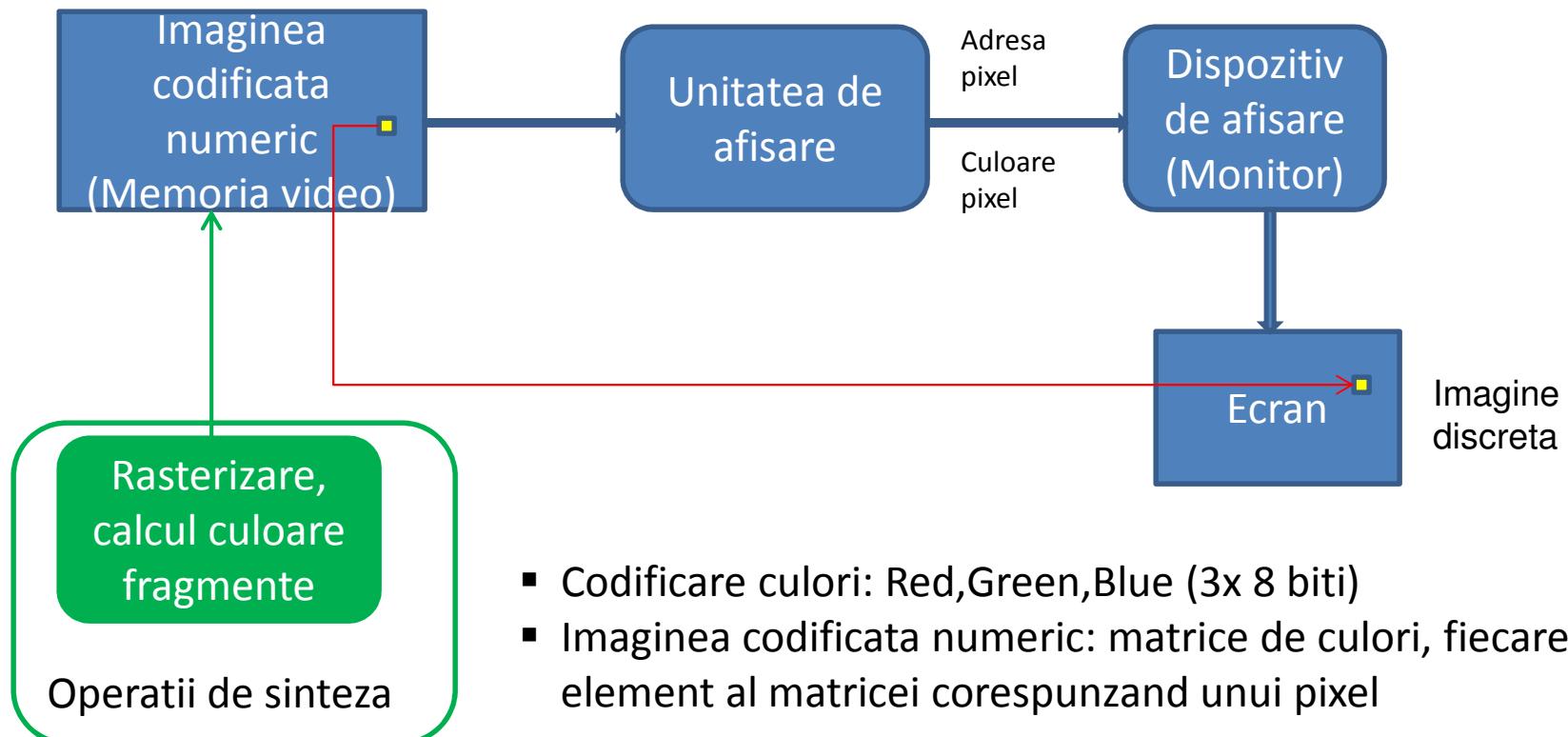
Operatii de sinteza:

- Efectuarea transformarilor care se aplica varfurilor obiectelor pentru modelarea scenei
- Efectuarea transformarilor de proiectie din spatiul 3D in spatiul 2D
- Eliminarea obiectelor nevizibile din “banda grafica”(Graphics pipeline)
- Decuparea primitivelor la marginile volumului vizual
- Eliminarea fețelor nevizibile ale obiectelor
- Rasterizarea: descompunerea primitivelor grafice in **fragmente**
care se afiseaza in **pixeli**
- Calculul culorii fiecarui fragment folosind:
modele de reflexie si refractie a luminii, calcul de umbre, texturi, s. . .



Sisteme grafice de sinteza(5)

Modelul discret al imaginii si afisarea sa



- Codificare culori: Red,Green,Blue (3x 8 biti)
 - Imaginea codificata numeric: matrice de culori, fiecare element al matricei corespunzand unui pixel

Scurt istoric al hardware-ului grafic de sinteza(1)

- **Mijlocul anilor '90:** cel mai complex hard grafic realiza numai rasterizarea primitivelor
- **Sfarsitul anilor '90:** NVIDIA a introdus termenul GPU, inlocuind termenul VGA – Video Graphics Addapter (introdus de IBM in 1987), devenit neadecvat pentru hardware-ul grafic dintr-un PC
- **Hardware de accelerare grafica:** hardware grafic specializat si scump pentru statii grafice (**SiliconGraphics** –SGI, Evans& Sutherland)
- **Prima generatie GPU: pana in 1998**
 - NVIDIA TNT2, ATI Rage
 - Rasterizare linii si triunghiuri, cu varfurile transformate de CPU
 - Aplicare 1-2 texturi

Scurt istoric al hardware-ului grafic de sinteza(2)

- **A 2-a generatie GPU (1999-2000):**
 - NVIDIA GeForce 256, Geforce2, ATI Radeon 7500
 - Transformare varfuri 3D si calcule de iluminare standard la nivel de varfuri
 - OpenGL1.x si DirectX7 suporta transformarea hardware a varfurilor
- **A 3-a generatie GPU:2001-2002**
 - NVIDIA GeForce 3, Geforce 4, ATI Radeon 8500
 - Transformare varfuri 3D si calcule de iluminare standard la nivel de varfuri
 - Posibilitatea programarii calculelor de iluminare la nivel de varfuri → VERTEX SHADER (program scris de programator si transferat la GPU; executat in paralel de mai multe procesoare pentru varfuri diferite)
 - DirectX8 – shader model 1

Scurt istoric al hardware-ului grafic de sinteza(3)

- **A 4-a generatie GPU (→2003):**

- NVIDIA GeForce FX, Cine Fx, ATI Radeon 9700
- Procesoare programabile la nivel de varfuri si de fragment (pixel)
- Procesoare specializate pentru calcule cu vectori si matrici
- Calcul paralel la nivel de varfuri si fragmente de primitive
- OpenGL 2.x si DirectX9 permit scrierea de programe VERTEX SHADER si PIXEL SHADER si transferul lor la GPU – shader model 2, 3
- OpenGL 3.x si DirectX10 adauga etapa programabila GEOMETRY SHADER – shader model 4

- **A 5-a generatie GPU (→2008):**

- NVIDIA GeForce G80, Fermi, Kepler, ATI – AMD Radeon incepand cu seria 4000
- In plus fata de generatia a 4-a:
 - Procesoare programabile pentru teselare (marirea rezolutiei geometrice) – TESSALLATION SHADER

Scurt istoric al hardware-ului grafic de sinteza(4)

GPGPU (General purpose Graphics Processing Unit): programare de aplicatii ne-grafice pentru executie pe placa grafica – stream-processing, paralelism inalt, circuite specializate.

Programare in CUDA, OpenCL, DirectCompute, ComputeShader (tip de shader in OpenGL 4.4 si DirectX11).

Evaluare la disciplina EGC

- **Laborator:**
 - 1.5 p – activitate la laborator
 - 4.5 p – teme (4 teme) prezentate la laborator
- **Examen:**
 - 4 puncte
- Se acorda bonusuri pentru:
 - Teme cu realizare deosebita (peste cerinte)
 - Prezenta la curs
- **Conditii de promovare:**
 - **Minim 50% punctaj pentru activitatea pe parcurs**
 - **Minim 50% punctaj in examen**

Transformari GEOMETRICE 2D

Prof. univ. dr. ing. Florica Moldoveanu

Transformari GEOMETRICE(1)

2

Obiectele 2D/3D sunt reprezentate prin:

- Coordonatele varfurilor, raportate la un sistem de coordinate carteziene 2D sau 3D;
- Atribute topologice (laturi, ciclul de laturi al unei fete, s.a.);
- Atribute de aspect: culoare, tipul de interior pentru suprafete 2D, atribute de material(ex.: reflexia/refractia luminii de catre suprafata), texturi, s.a.

Transformarile geometrice se aplică (coordonatelor) varfurilor obiectului și nu afectează atributele sale!

Transformari GEOMETRICE(2)

3

- Sunt operatii fundamentale in sinteza imaginilor
- Folosite pentru:
 - Redarea desenelor la diferite marimi
 - Compunerea desenelor sau a scenelor 3D
 - Realizarea animatiei
 - Transformarea obiectelor dintr-un spatiu logic, in care sunt definite, in spatiul fizic de afisare
 - Etc.

Transformari GEOMETRICE 2D(1)

4

1. TRANSFORMARI GEOMETRICE ELEMENTARE

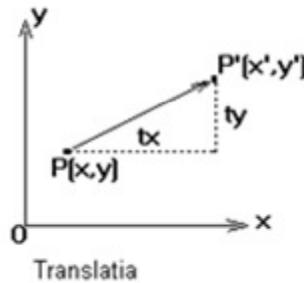
- Translatia
- Scalarea fata de origine
- Rotatia fata de origine
- Forfecarea fata de origine
- Oglindiri fata de axele principale, fata de origine

Transformari geometrice 2D elementare(1)

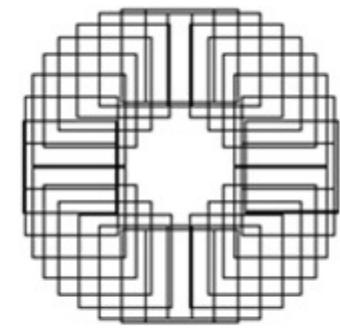
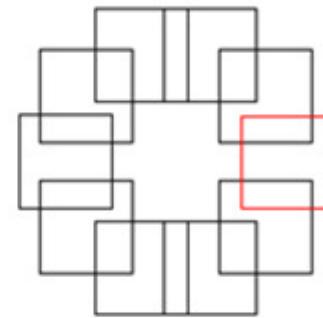
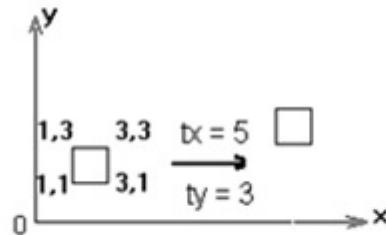
5

• Translatia

Este definita printr-un vector, $T[tx,ty]$.



$$\begin{aligned}x' &= x + tx \\y' &= y + ty\end{aligned}$$



Translatia patratului pe circumferinta unui cerc

Se doreste o reprezentare matriciala a transformarilor, necesara pentru compunerea lor.

$$P(x,y) \rightarrow [x,y] \text{ si } P(x',y') \rightarrow [x',y']$$

Se doreste: $[x',y'] = [x,y] * M \rightarrow$ Nu exista o matrice de 2×2 pt exprimarea translatiei in coordonate carteziene

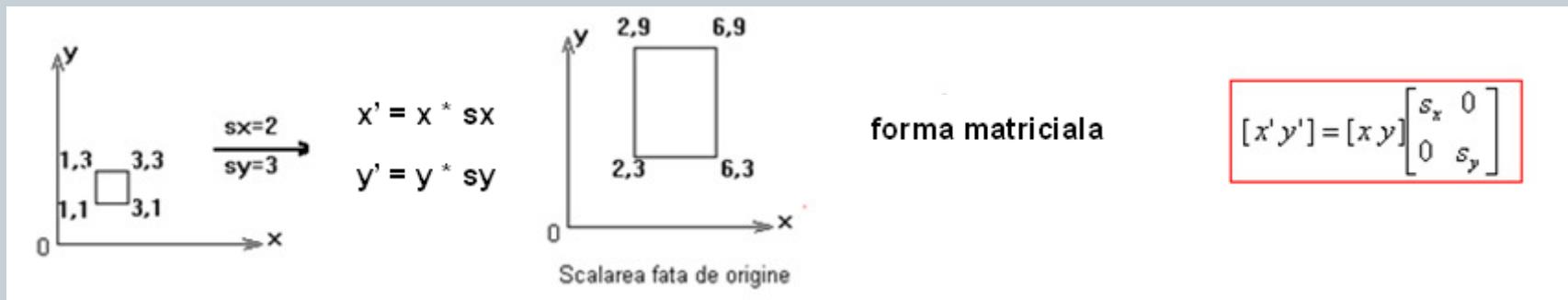
Transformari geometrice 2D elementare(2)

6

Scalarea fata de origine

Este definita prin 2 numere reale, de regula pozitive:

- s_x - scalarea de-a lungul axei OX
- s_y - scalarea de-a lungul axei OY



Efecte.....

- $s_x = s_y$, scalare uniforma

Transformari geometrice 2D elementare(3)

7

Rotatia fata de origine



$$\begin{aligned}x &= r * \cos(t) \\y &= r * \sin(t)\end{aligned}$$

Relatia dintre coordonatele carteziene si
coordonatele polare ale unui punct

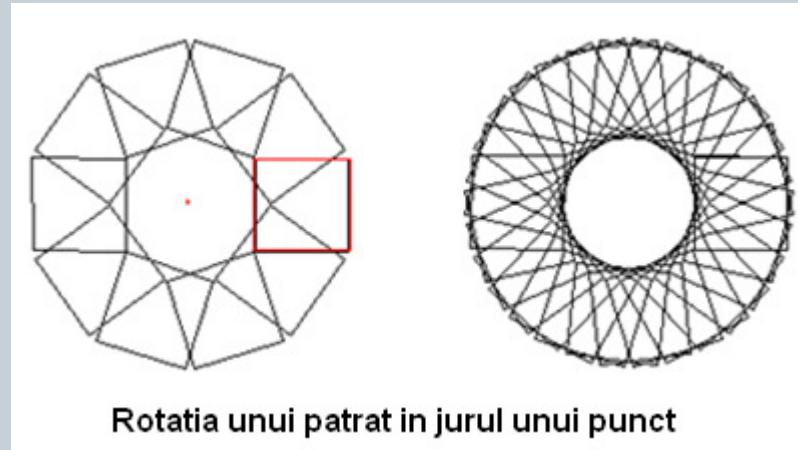
$$x' = r * \cos(t+u) = r * (\cos(t) * \cos(u) - \sin(t) * \sin(u)) = x * \cos(u) - y * \sin(u)$$

$$y' = r * \sin(t+u) = r * (\cos(t) * \sin(u) + \sin(t) * \cos(u)) = x * \sin(u) + y * \cos(u)$$

forma matriciala

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} \begin{bmatrix} \cos(u) & \sin(u) \\ -\sin(u) & \cos(u) \end{bmatrix}$$

Rotatia fata de origine



Transformari geometrice 2D fata de un punct oarecare din plan(1)

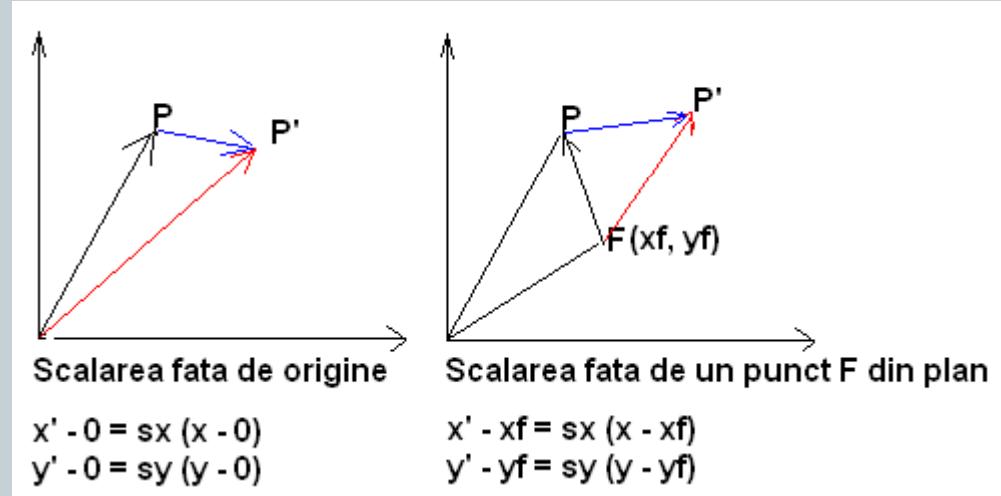
8

Scalarea fata de un punct oarecare din plan

- Punctul fix al transformarii este un punct oarecare $F(x_f, y_f)$ – coordonatele sale nu se modifica prin aplicarea transformarii.
- Scalarea se aplica vectorului FP :
 $x' - x_f = sx^*(x - x_f)$
 $y' - y_f = sy^*(y - y_f)$

Rezulta:

$$\begin{aligned}x' &= x^*sx + x_f - x_f^*sx \\y' &= y^*sy + y_f - y_f^*sy\end{aligned}$$



Nu poate fi exprimata printr-o matrice de 2×2 !

Transformari geometrice 2D fata de un punct oarecare din plan(2)

9

Rotatia fata de un punct oarecare din plan

- Punctul fix al transformarii este $F(x_f, y_f)$
- Rotatia se aplica vectorului FP , in jurul punctului F :

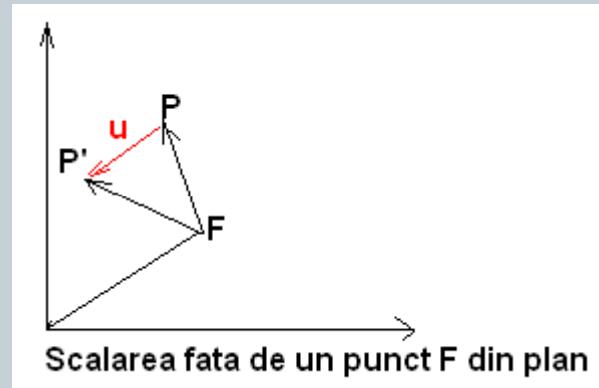
$$x' - x_f = (x - x_f) * \cos(u) - (y - y_f) * \sin(u)$$

$$y' - y_f = (x - x_f) * \sin(u) + (y - y_f) * \cos(u)$$

Rezulta:

$$x' = x * \cos(u) - y * \sin(u) + x_f - x_f * \cos(u) + y_f * \sin(u)$$

$$y' = x * \sin(u) + y * \cos(u) + y_f - x_f * \sin(u) - y_f * \cos(u)$$



Nu poate fi exprimata printr-o matrice de 2×2 !

Compunerea transformarilor geometrice 2D

10

De ce este necesara?

- Pentru a aplica o singura transformare care inglobeaza o secventa de transformari elementare, in locul aplicarii in secventa a transformarilor elementare; de ex., se aplica tuturor varfurilor o transformare care inglobeaza scalare, rotatie si translatie in loc sa se aplice fiecarui varf secventa de transformari elementare.
- Matricea unei transformari compuse se obtine prin inmultirea matricilor transformarilor elementare. Exemplu:

$$RS = R^*S = \begin{bmatrix} \cos(u) & \sin(u) \\ -\sin(u) & \cos(u) \end{bmatrix} \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \quad RS \neq SR$$

$$SR = S^*R = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} \cos(u) & \sin(u) \\ -\sin(u) & \cos(u) \end{bmatrix}$$

- **Translatia fata de origine nu poate fi reprezentata printr-o matrice de 2x2!**
- **Aceasta impune reprezentarea transformarilor in coordonate omogene:**

Reprezentarea transformarilor geometrice 2D in coordonate omogene(1)

11

Coordonate omogene:

Un punct din plan, $P(x,y)$, se reprezinta in coordonate omogene printr-un vector

$$[xw, yw, w] \text{ sau } \begin{bmatrix} xw \\ yw \\ w \end{bmatrix} \quad xw = x * w; yw = y * w; w - orice numar real$$

Exemplu: $P(2, 0.5) \rightarrow [2, 0.5, 1], [4, 1, 2], [20, 5, 10]$

Transformarea din coordonate omogene in coordonate carteziene:

$[xw \ yw \ w] \rightarrow P(x, y)$, unde:

- pentru $w \neq 0$, $x = xw/w$, $y = yw/w$
- pentru $w = 0$, P este un punct la infinit

Reprezentarea transformarilor elementare 2D în coordonate omogene(2)

12

Translația

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix} \text{ sau } \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scalarea față de origine

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ sau } \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Rotatia față de origine

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ sau } \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Transformarile inverse ale transformarilor elementare

13

$T(tx, ty)$: matricea translatiei, in coordonate omogene

$S(0, 0, sx, sy)$: matricea scalarii fata de origine, in coordonate omogene

$R(0,0,u)$: matricea rotatiei fata de origine, in coordonate omogene

$$T(tx, ty)^{-1} = T(-tx, -ty)$$

$$S(0, 0, sx, sy)^{-1} = S(0, 0, 1/sx, 1/sy)$$

$$R(0,0, u)^{-1} = R(0, 0, -u)$$

Transformari geometrice 2D compuse(1)

14

Exemple de transformari compuse:

- Expresiile matematice ale scalării și rotației față de un punct oarecare din plan se pot obține prin compunerea următoarelor transformări:
 - Translația prin care punctul fix al transformării ajunge în origine: $T(-xf, -yf)$;
 - Scalarea / rotația față de origine: $S(o, o, sx, sy)/R(o, o, u)$;
 - Translația inversă celei de la punctul 1: $T(xf, yf)$.

Transformari geometrice 2D compuse(2)

15

Scalarea față de punctul F (xf, yf)

$$[x' \ y' \ 1] = [x \ y \ 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_f & -y_f & 1 \end{bmatrix} \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_f & y_f & 1 \end{bmatrix}$$

$T(-x_f, -y_f)$ $S(0, 0, s_x, s_y)$ $T(x_f, y_f)$

Matricea transformării compuse:

$$M = T(-x_f, -y_f) * S(0, 0, s_x, s_y) * T(x_f, y_f)$$

Aplicarea transformării compuse:

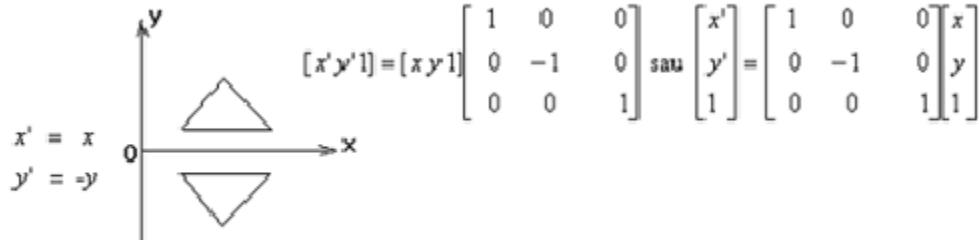
$$[x' \ y' \ 1] = [x \ y \ 1] * M$$

Alte transformari geometrice 2D(1)

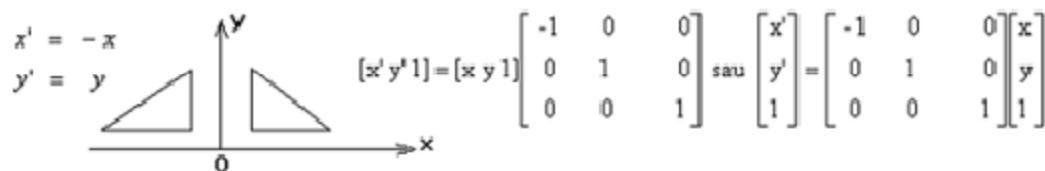
16

Oglindirea

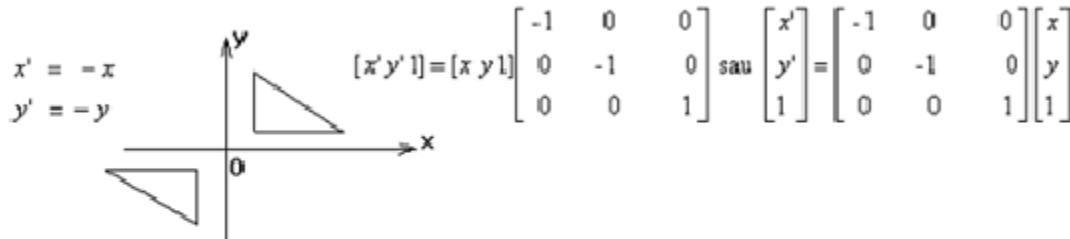
Fata de axa OX



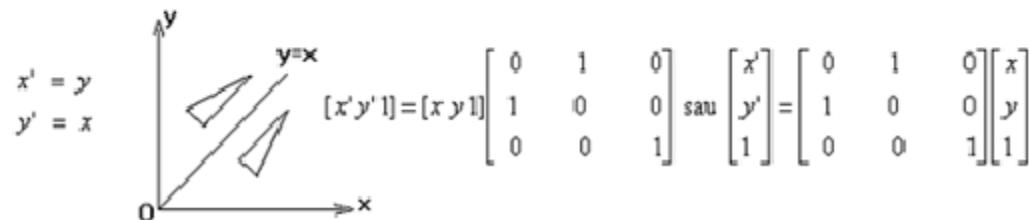
Fata de axa OY



Fata de origine



Fata de dreapta $x=y$



Alte transformari geometrice 2D(2)

17

Oglindirea față de o dreaptă oarecare

Se exprima ca transformare compusa prin inmultirea matricilor care exprima urmatoarele transformari:

1. O translație, astfel încât dreapta sa treaca prin origine.
2. O rotație față de origine, a.î. dreapta să se suprapună peste una dintre axele principale.
3. Oglindirea față de axa principală peste care a fost suprapusă dreapta.
4. Rotația inversă celei de la punctul 2.
5. Translația inversă celei de la punctul 1.

În notație matricială:

$M = T * R^* O^* R^{-1} * T^{-1}$ (folosind vectori linie) sau $M = T^{-1} * R^{-1} * O * R * T$ (folosind vectori coloana)

Deduceti T, R, O, atunci cand dreapta este data printr-un punct, (x_d, y_d) si o directie, $D[a, b]$.

Alte transformari geometrice 2D(4)

18

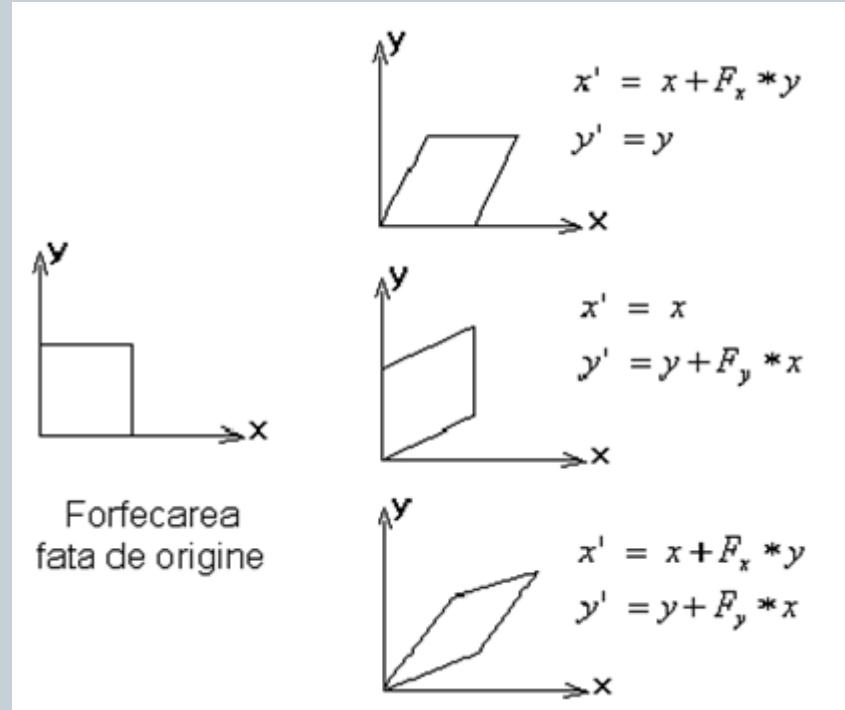
Forfecarea

Este definita prin 2 numere reale:

Fx: factorul de forfecare pe axa OX

Fy: factorul de forfecare pe axa OY

Deduceti formele matriciale ale transformarilor de forfecare:



Forfecarea fata de un punct oarecare din plan, (xf,yf) , exprimata ca transformare compusa:

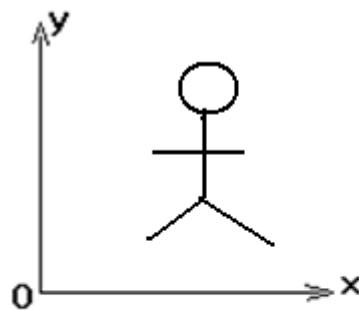
1. Translatie prin care punctul (xf, yf) ajunge in origine
2. Forfecarea fata de origine
3. Translatia inversa celei de la pasul 1

Transformarea de vizualizare 2D

Prof. univ. dr. ing. Florica Moldoveanu

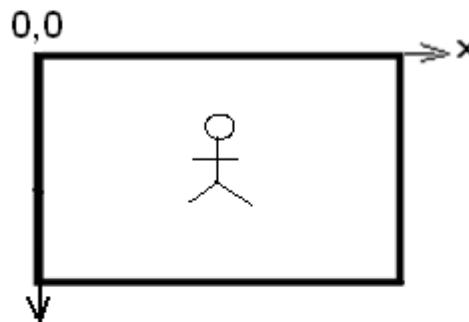
Transformarea de vizualizare 2D(1)

- Desenele reprezentate intr-un program de aplicatie grafica 2D sunt, de regula, raportate la un sistem de coordonate diferit de cel al suprafetei de afisare.
- Exemple: planul unei case, un desen tehnic, graficul unei functii, etc.



coordonate logice
(metri, milimetri, viteza, timp, etc.)

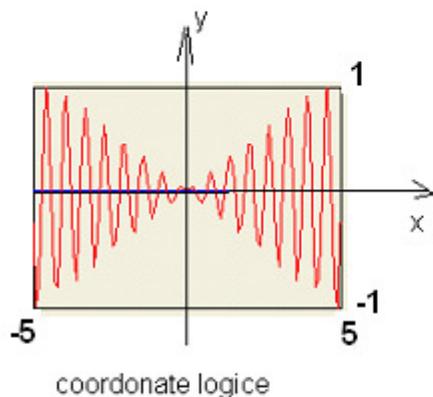
Sistemul de coordonate utilizator



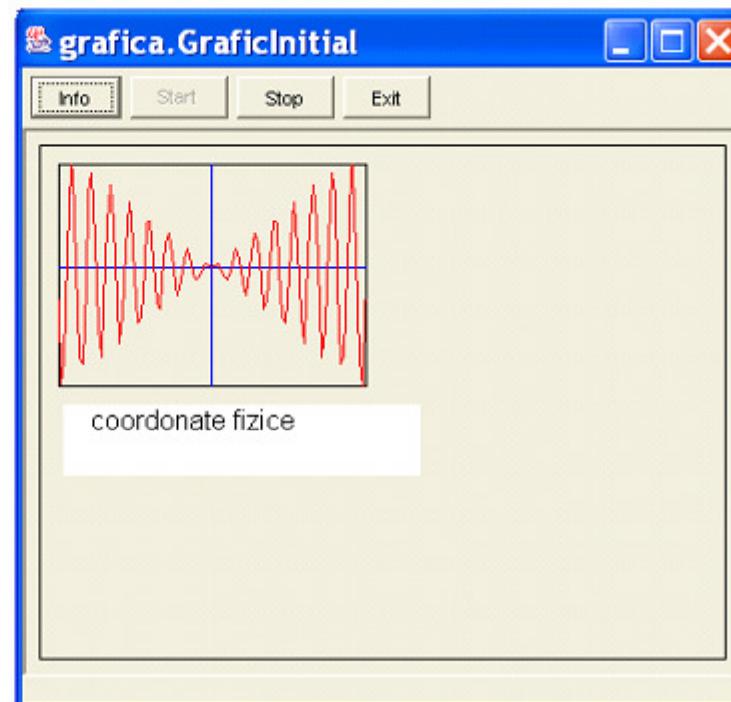
coordonate fizice(dispozitiv)
(pixeli, milimetri, s.a.)

Sistemul de coordonate dispozitiv

Transformarea de vizualizare 2D(2)



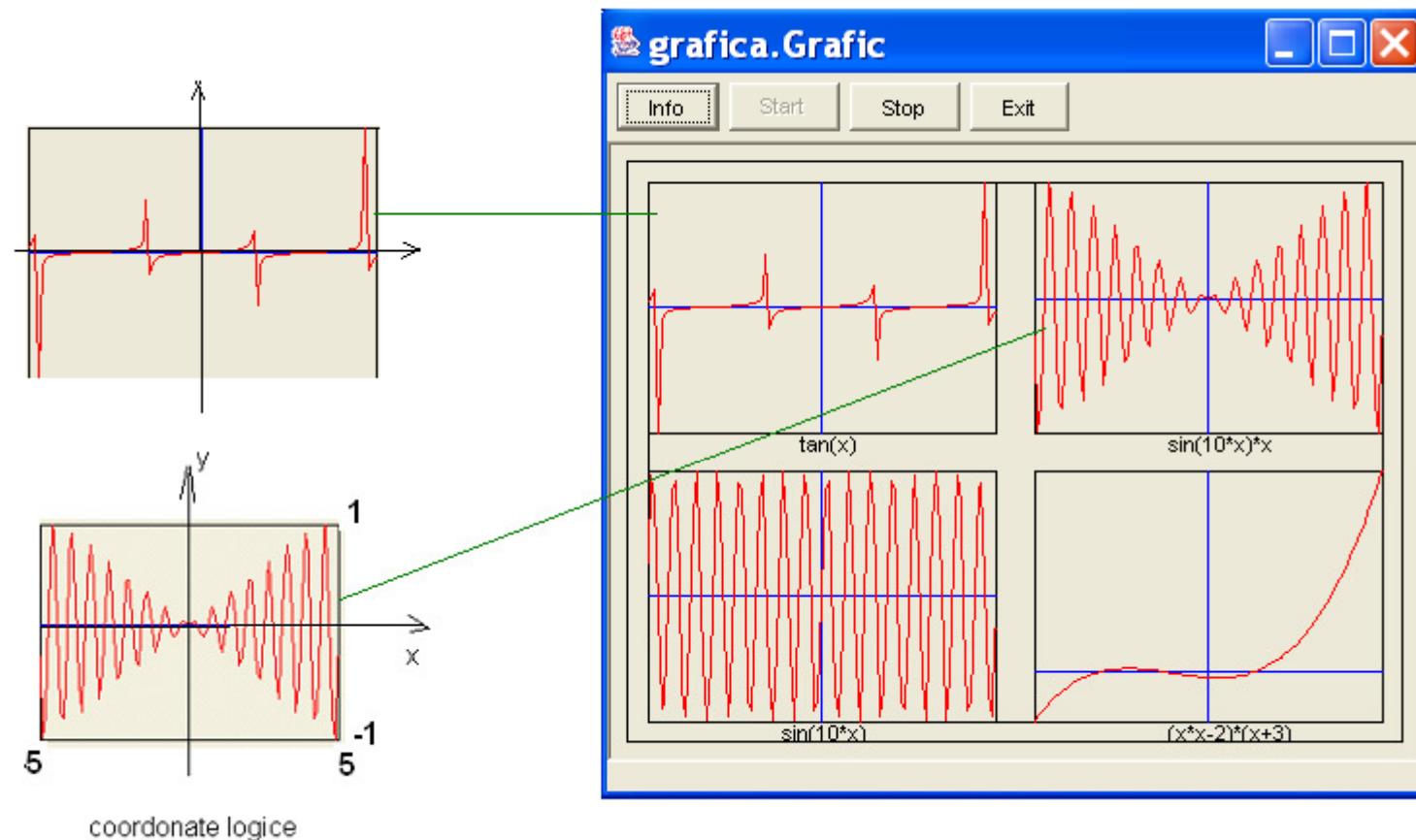
Aplicatia defineste graficul functiei in sistemul de coordonate utilizator (logice).



Coordonatele punctelor de pe graficul afisat pe ecran sunt adrese de pixeli.

Coordonatele logice trebuie transformate in coordonate fizice.

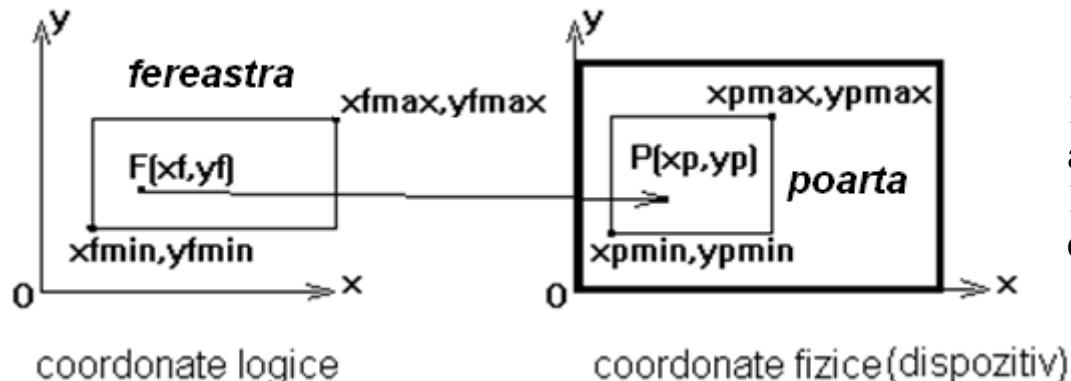
Transformarea de vizualizare 2D(3)



Este necesara o transformare generală!

Transformarea de vizualizare 2D(4)

- transformarea “fereastra-poarta”-



- **fereastra:** zona din spatiul coord. logice al carei continut se va afisa
- **poarta:** zona din spatiul coord. fizice in care se va afisa continutul ferestrei

- Transformarea este definita folosind 2 dreptunghiuri, definite in cele 2 sisteme de coordonate, numite: **fereastra (de vizualizare)** si **poarta (de afisare)**.
- Transformarea se mai numeste **fereastra-poarta**.

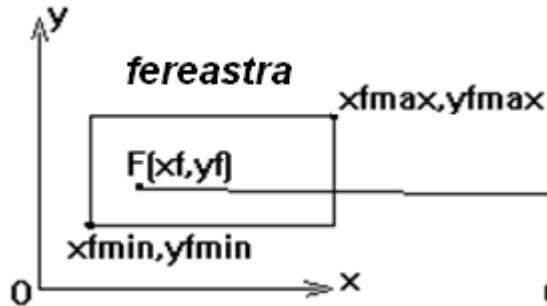
F: un punct din fereastra.

P: punctul in care se transforma F prin transformarea fereastra-poarta.

➤ Transformarea fereastra-poarta trebuie sa asigure ca:

“pozitia relativa a lui P in poarta de afisare este aceeasi cu pozitia relativa a lui F in fereastra”.

Transformarea de vizualizare 2D(5)



$$\frac{x_p - x_{p\min}}{x_{p\max} - x_{p\min}} = \frac{x_f - x_{f\min}}{x_{f\max} - x_{f\min}}$$

$$\frac{y_p - y_{p\min}}{y_{p\max} - y_{p\min}} = \frac{y_f - y_{f\min}}{y_{f\max} - y_{f\min}}$$

$$sx = \frac{x_{p\max} - x_{p\min}}{x_{f\max} - x_{f\min}}$$

$$sy = \frac{y_{p\max} - y_{p\min}}{y_{f\max} - y_{f\min}}$$

- s_x, s_y reprezinta factorii de scalare ai transformarii
- t_x, t_y depind de pozitiile celor 2 ferestre fata de originea sistemului de coord. in care sunt definite

și $t_x = x_{p\min} - sx * x_{f\min}$ $t_y = y_{p\min} - sy * y_{f\min}$

$$xp = x_f * sx + tx$$

$$yp = y_f * sy + ty$$

← Exprimarea matematica a transformarii fereastra – poarta, considerand o aceeasi orientare a axelor celor 2 sisteme de coordonate.

Exemple:

Transformarea de vizualizare 2D(6)

Efectele transformarii

- Marire / micsorare, in functie de dimensiunile ferestrei si ale portii
- Deformare: fereastra si poarta nu sunt dreptunghiuri asemenea
- Pentru scalare uniforma: $s = \min(s_x, s_y)$
- Afisare centrata in poarta: translatie suplimentara pe axa ox sau pe axa oy:

$$tx = (x_{pmax} - x_{pmin} - s * (x_{fmax} - x_{fmin})) / 2$$

$$ty = (y_{pmax} - y_{pmin} - s * (y_{fmax} - y_{fmin})) / 2$$

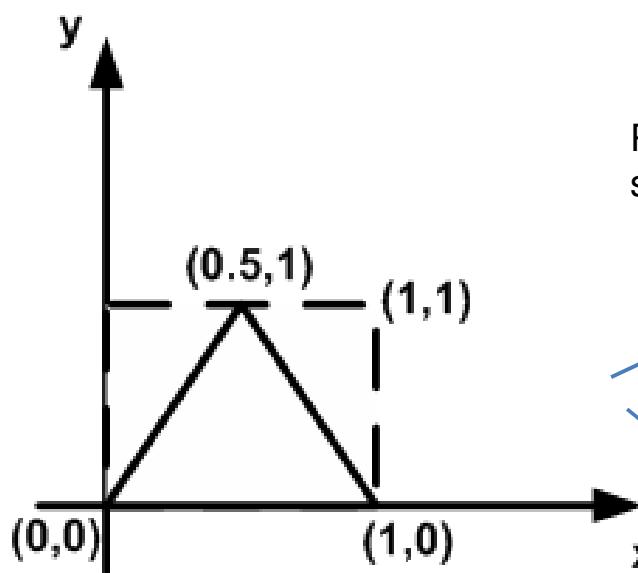
Translatia totala: $tx + ts_x, ty + ts_y$

- Decuparea primitivelor aflate in afara ferestrei vizuale

- Exemple:**

Transformarea de vizualizare 2D(7)

Afisarea centrata in poarta de vizualizare

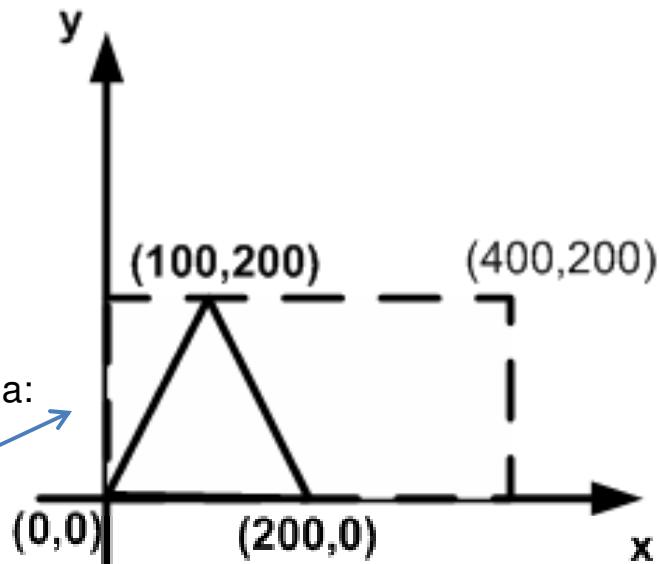


$$sx = 400, sy = 200$$

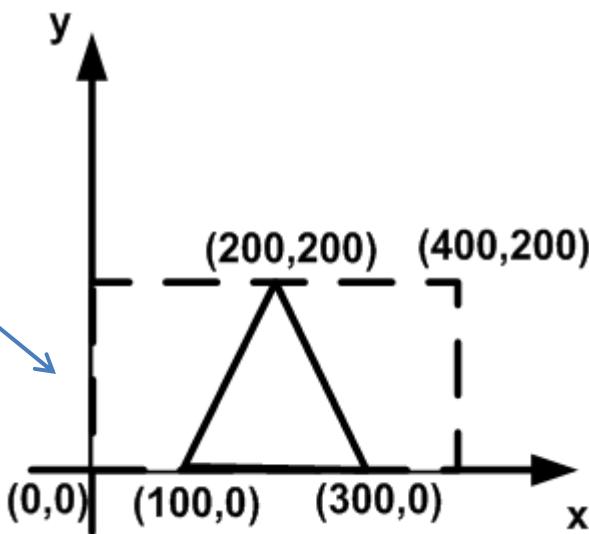
Pentru scalare uniforma:
 $s = \min(sx, sy)$

$$tx=0, ty=0$$

$$\begin{aligned} tsx &= (xpmax - xpmmin - s * (xfmax - xfmin)) / 2 \\ &= (400 - 200 * 1) / 2 = 100 \end{aligned}$$

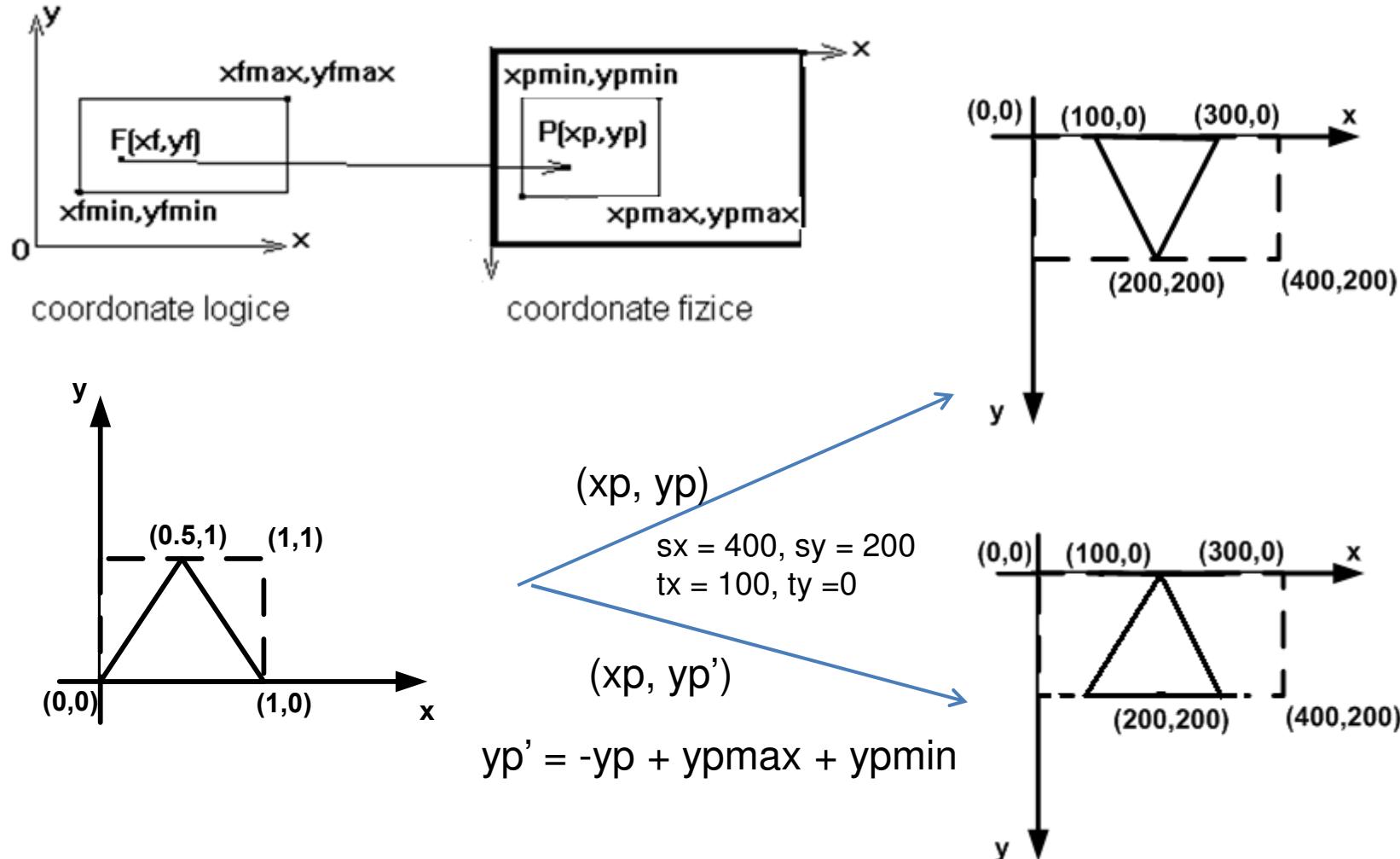


$$\begin{aligned} tsx &= 100, \\ tsy &= 0 \end{aligned}$$



Transformarea de vizualizare 2D(8)

Corectia coordonatei yp tinand cont de orientarea axelor sist. coord fizice



Transformarea de vizualizare 2D(9)

Formulele finale ale transformarii fereastra - poarta:

$$xp = xf * sx + tx$$

$$yp = ypm \text{min} + ypm \text{ax} - (yf * sy + ty) = -yf * sy + ypm \text{min} + ypm \text{ax} - ty$$

Exprimarea matriciala a transformarii fereastra-poarta

$$M = \begin{bmatrix} sx & 0 & tx \\ 0 & -sy & ypm \text{min} + ypm \text{ax} - ty \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} xp \\ yp \\ 1 \end{bmatrix} = M * \begin{bmatrix} xf \\ yf \\ 1 \end{bmatrix}$$

Proiectii

Prof. univ. dr. ing. Florica Moldoveanu

Proiectii

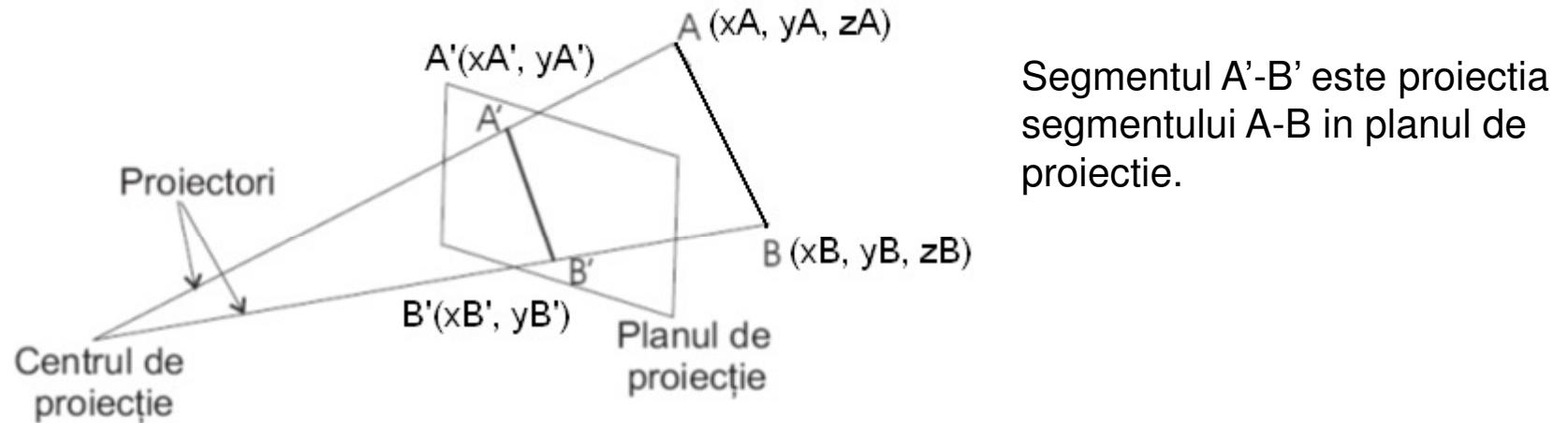
- ❖ Transformari dintr-un spatiu n-dimensional intr-un spatiu k-dimensional, k< n.

Proiectii $\mathbf{R}^3 \rightarrow \mathbf{R}^2$

- Transformari din spatiul tri-dimensional intr-un spatiu bi-dimensional
- Se aplica varfurilor obiectului
- Nu modifica legaturile dintre varfuri

Proiectii $\mathcal{R}^3 \rightarrow \mathcal{R}^2$

- Proiectia unui varf 3D intr-un plan 2D este punctul de intersectie dintre plan si projectorul care pleaca din **centrul de proiectie** si trece prin varf



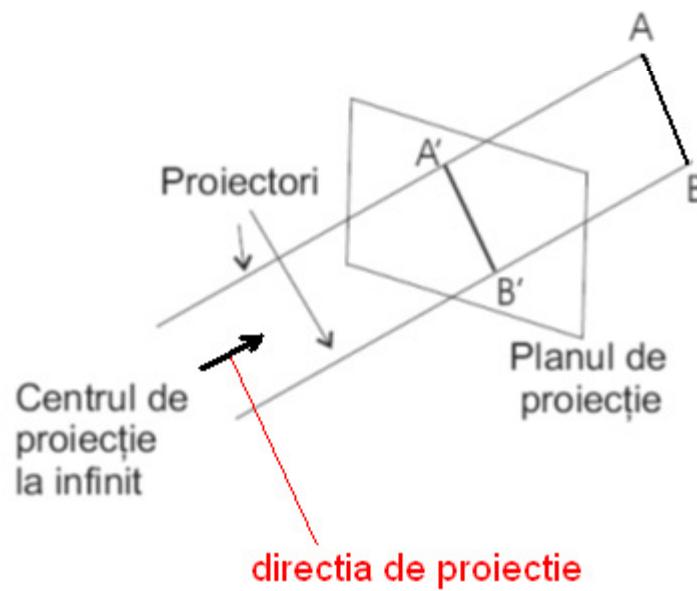
Segmentul $A'-B'$ este proiectia segmentului $A-B$ in planul de proiecție.

Exista 2 clase de proiectii:

- **Proiectii perspectiva**: centrul de proiectie este situat la distanta finita fata de planul de proiectie (ca in figura de mai sus); projectorii sunt drepte convergente in CP.

Proiectii $\mathcal{R}^3 \rightarrow \mathcal{R}^2$

- **Proiectii paralele** - centrul de proiectie este la infinit; projectorii sunt linii paralele care trec prin varfurile obiectului proiectat si au directia specificata (vectorul **directie de proiectie**)



Proiectii perspectiva(1)

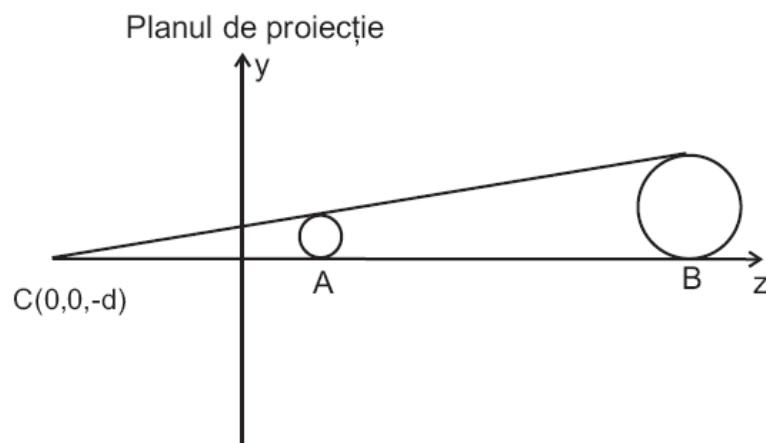
- Produc imagini asemanatoare celor obtinute cu aparatele de fotografiat.

Caracteristici:

1. Efectul de micsorare.

Dimensiunea obiectului in proiectia 2D este invers proportionala cu distanta de la centrul de proiectie la obiect.

Exemplu:



Proiectia sferelor A si B este un cerc in planul de proiectie.

$$r_B = 2 * r_A$$

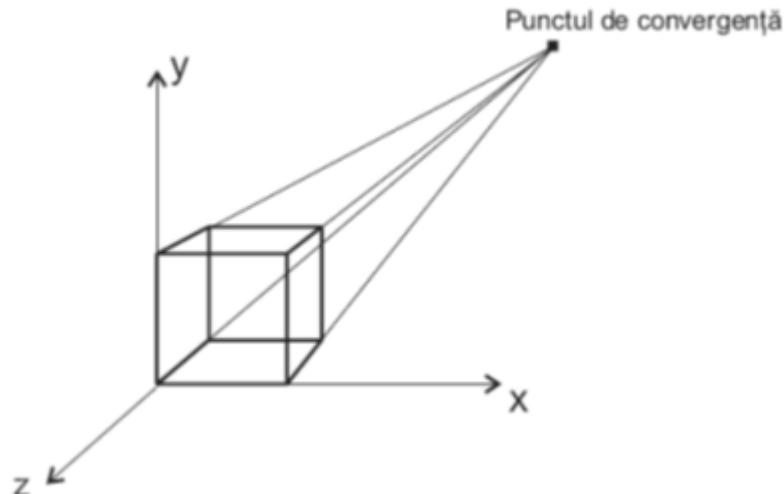
$$\text{Dist}(B-C) = 2 * \text{Dist}(A-C)$$

Proiectii perspectiva(2)

2. Modifica unghiurile dintre dreptele care nu sunt paralele cu planul de proiectie

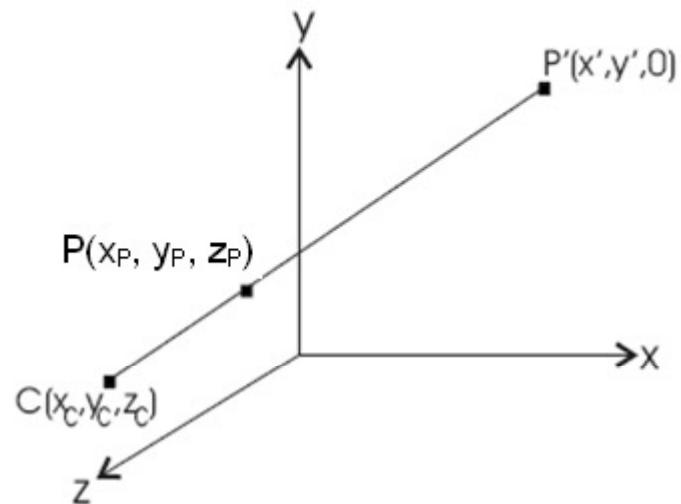
Proiectiile liniilor paralele care nu sunt paralele cu planul de proiectie converg catre un punct din plan, numit punct de convergenta

- Punctul catre care converg liniile paralele cu una dintre axele principale: **punct de convergenta principal**



Pot fi efectuate proiectii cu unul, doua sau trei puncte de convergenta principale.

Proiectia perspectiva in planul XOY cu un centru de proiectie oarecare(1)



C – centru de proiectie
P – punctul proiectat
P' – proiectia lui P in XOY

Ecuatiile parametrice ale projectorului care trece prin P:

$$x = x_c + (x_p - x_c) * t$$

$$y = y_c + (y_p - y_c) * t$$

$$z = z_c + (z_p - z_c) * t$$

$0 \leq t \leq 1$, pentru punctele aflate pe segmentul C-P

$$0 = z_c + (z_p - z_c) * t' \rightarrow t' = -z_c / (z_p - z_c) \rightarrow x' = (x_c * z_p - z_c * x_p) / (z_p - z_c)$$

$$y' = (y_c * z_p - z_c * y_p) / (z_p - z_c)$$

Proiectia perspectiva in planul XOY cu un centru de proiectie oarecare(2)

Forma matriciala

$$[x' \ y' \ z' \ 1] = [xp \ yp \ zp \ 1] * P_{perspectiva}$$

- matricea proiectiei depinde de coordonata z a punctului proiectat →
- se doreste o matrice de proiectie independenta de punctul proiectat pentru a se putea folosi pentru toate varfurile unui obiect

$$P_{perspectiva} = \begin{bmatrix} \frac{-z_c}{zp - z_c} & 0 & 0 & 0 \\ 0 & \frac{-z_c}{zp - z_c} & 0 & 0 \\ \frac{x_c}{zp - z_c} & \frac{y_c}{zp - z_c} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$[x'_w \ y'_w \ z'_w \ w] = [xp \ yp \ zp \ 1] * P_{perspectiva}$$

$$w = zp - zc$$

$$P_{perspectiva} = \begin{bmatrix} -zc & 0 & 0 & 0 \\ 0 & -zc & 0 & 0 \\ xc & yc & 0 & 1 \\ 0 & 0 & 0 & -zc \end{bmatrix}$$

Coordonatele carteziene ale punctului proiectat:

$$x' = x'_w / w \quad y' = y'_w / w : \text{impartirea perspectiva}$$

$$[x'_w \ y'_w \ z'_w \ w] = [xp \ yp \ zp \ 1] * P_{perspectiva}$$

$$w = -(zp - zc) / zc$$

$$P_{perspectiva} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{x_c}{z_c} & -\frac{y_c}{z_c} & 0 & -\frac{1}{z_c} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proiectia perspectiva in planul XOY cu un centru de proiectie oarecare(3)

Pentru conservarea informatiei de adancime (necesara pentru eliminarea din imagine a partilor nevizibile ale scenei 3D) se foloseste matricea :

$$P_{perspectiva} = \begin{bmatrix} -zc & 0 & 0 & 0 \\ 0 & -zc & 0 & 0 \\ xc & yc & 1 & 1 \\ 0 & 0 & 0 & -zc \end{bmatrix}$$

$$z' = zp / (zp - zc)$$

Proiectia perspectiva standard:

$$X_c = 0, Y_c = 0, Z_c = -d$$

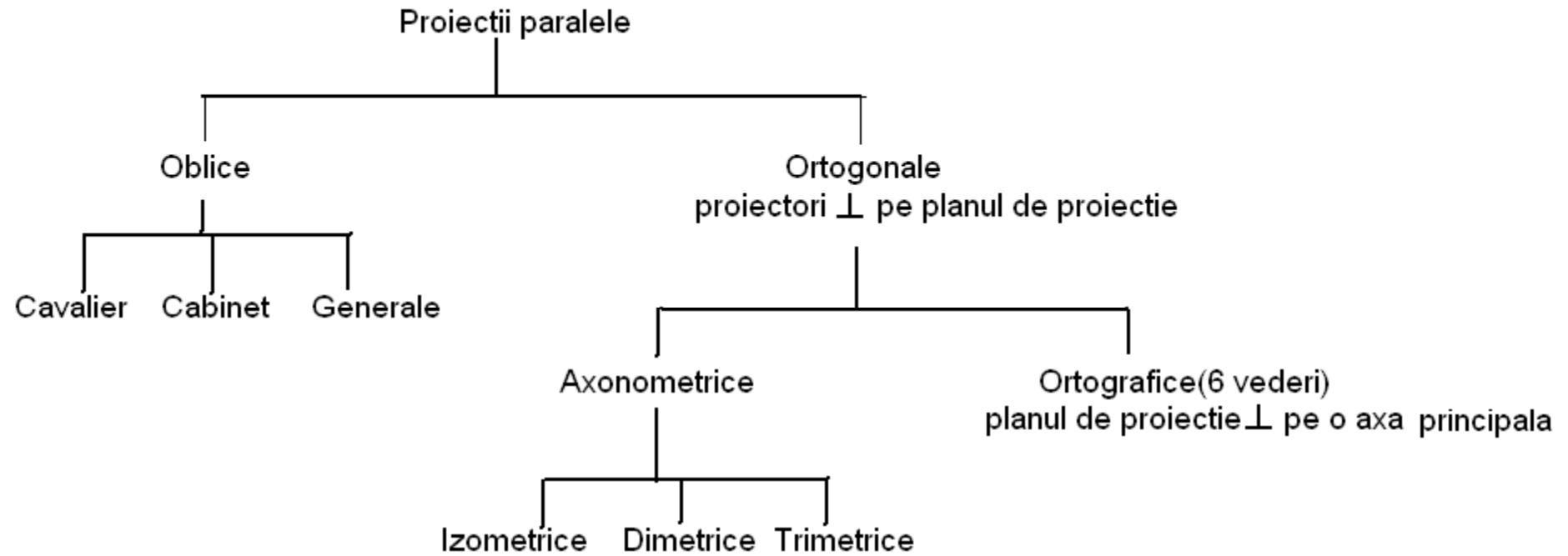
$$P_{perspectiva} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{d} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proiectii paralele(1)

Caracteristici

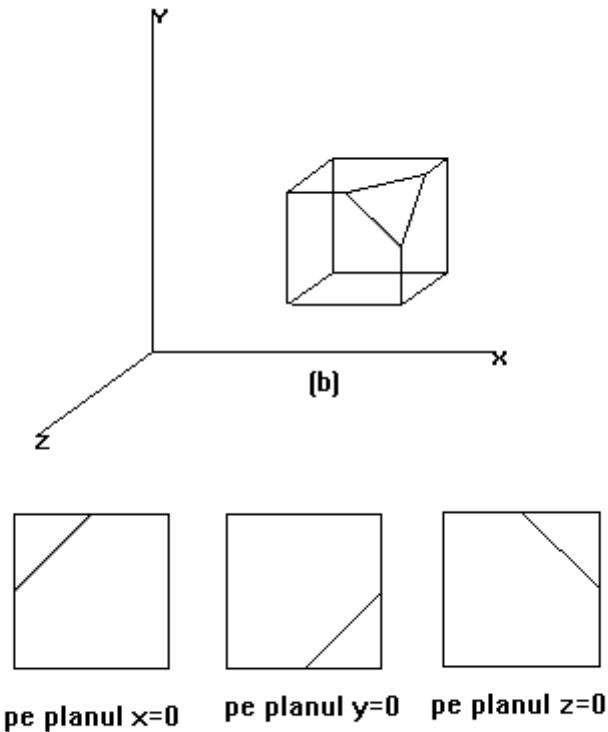
- Proiectorii sunt drepte paralele de directie data: directia de proiectie
- Sunt transformari afine (conserveaza paralelismul liniilor)
- Unghiurile se conserveaza doar pentru fețele obiectului paralele cu planul de proiectie.
- Clasificare dupa unghiul dintre projectorii si planul de proiectie:
 - *Proiectii ortogonale*: projectorii sunt perpendiculari pe planul de proiectie.
 - Ortografice: planul de proiectie este perpendicular cu un plan principal
 - Axonometrice: planul de proiectie este oarecare
 - *Proiectii oblice*: projectorii nu sunt perpendiculari pe planul de proiectie – cazuri particulare:
 - proiectii Cavalier (unghiul dintre projectorii si plan = 45 grade)
 - proiectii Cabinet (unghiul dintre projectorii si plan = 63.43 grade)

Proiectii paralele(2) - clasificare



Proiectii paralele(3)

1. Proiectii ortogonale - ortografice



Proiectii ortografice ale unui cub cu un colt taiat

Ortografice (Vederi): proiectii in planele sistemului de coordonate (planul de proiectie perpendicular pe o axa principală)

- in YOZ: *vederea din stanga, vederea din dreapta,*
- in XOZ : *vederea de sus, vederea de jos,*
- in XOY : *vederea din fata, vederea din spate.*

➤ **Conservează lungimile laturilor și unghiurile dintre laturi**

➤ **Utile în desenul tehnic**

Proiectii paralele(4)

2. Proiectii ortogonale - axonometrice

- Planul de proiectie nu este perpendicular pe nici una din axele sistemului de coordonate.
- Redau mai multe fețe ale obiectului proiectat (ca și cele perspective): alegând planul de proiectie se poate controla scalarea laturilor.
- În funcție de unghiurile pe care planul de proiectie le face cu axele sistemului de coordonate:
 - **proiectii izometrice**, planul face unghiuri egale cu toate cele trei axe → **laturile sunt scalate cu factori de scalare egali pe cele 3 axe.**
 - **proiectii dimetrice**, planul face unghiuri egale cu două dintre axe → **laturile sunt scalate cu factori de scalare egali pe 2 axe.**
 - **proiectii trimetrice**, unghiurile dintre cele trei axe și plan sunt diferite → **factorii de scalare ai laturilor pe cele 3 axe sunt diferiti.**

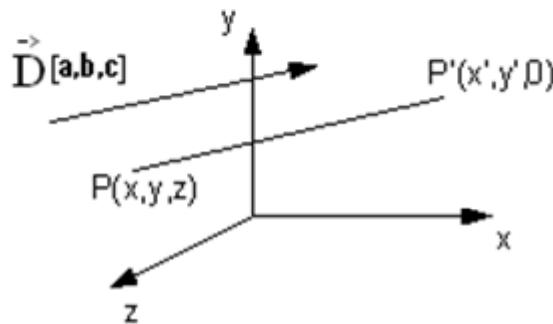
Proiectii paralele(5)

3. Proiectii oblice

- Planul de proiectie este perpendicular pe o axa principala.
- Directia de proiectie nu este perpendiculara pe planul de proiectie.
- Fețele II cu planul de proiectie se proiecteaza fara alterarea unghiurilor si a marimii laturilor
- Alegand directia de proiectie se obtin cazurile particulare:
 - **Proiectii Cavalier** (unghiul projectorilor cu planul de proiectie = 45 grade)
 - Lungimea proiectiei unei laturi perpendiculare pe plan este egala cu lungimea laturii 3D (**consevera lungimea laturilor perpendiculare pe planul de proiectie**)
 - **Proiectii Cabinet** (unghiul projectorilor cu planul de proiectie = 63.43 grade)
 - Lungimea proiectiei unei laturi perpendiculare pe plan este egala cu jumatate din lungimea laturii 3D

Proiectii paralele in planul XOY (6)

Fie P un punct din spatiu, de coordonate (x, y, z) , care se proiecteaza in punctul $P'(x', y')$ din planul de proiectie, directia projectorilor fiind $D[a \ b \ c]$.



Din conditia $D \parallel PP'$, rezulta: $PP' = s^*D$
 $x' - x = s^*a$
 $y' - y = s^*b$
 $0 - z = s^*c \implies s = -z/c$

$$x' = x - (a/c)^* z$$
$$y' = y - (b/c)^* z$$

Proiectii paralele în planul XOY(7)

Exprimarea matricială a proiectiilor paralele este:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] \times P_{paralela} \quad \text{unde} \quad P_{paralela} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ -\frac{a}{c} & -\frac{b}{c} & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

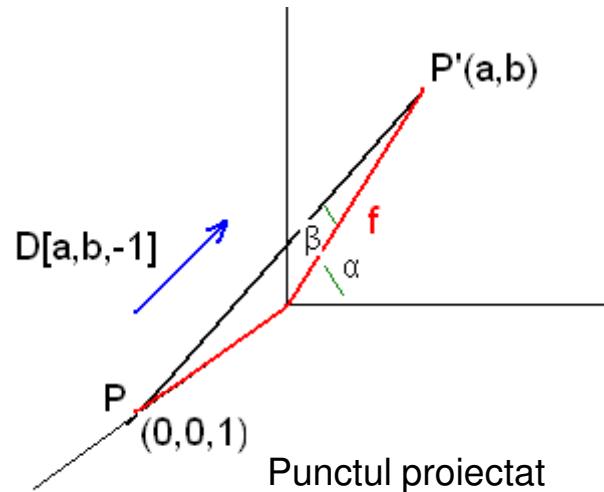
Cazuri particulare

1. *Proiecția ortografică în XOY:*

$$a = 0, b = 0 \rightarrow \quad P_{O-XOY} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Proiectii paralele in planul XOY(8)

2. Proiectii oblice:



$$c = -1 \rightarrow$$

$$P_{OBL} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a & b & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_{OBL} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ f\cos(\alpha) & f\sin(\alpha) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

OP de lungime 1 se proiecteaza in OP' de lungime f: $\rightarrow f$ este factorul de scalare al laturilor perpendiculare pe XOY (OP este perpendicular pe XOY).

$$\operatorname{tg}(\beta) = 1/f$$

$f = 0 \rightarrow \beta = 90^\circ$: proiectie ortografica

α este un parametru liber

$f = 1 \rightarrow \beta = 45^\circ$: proiectie Cavalier

in mod usual, $\alpha = 30^\circ$ sau 45°

$f = 0.5 \rightarrow \beta = 63.43^\circ$: proiectie Cabinet

Proiectii paralele in planul XOY(9)

3. Proiectii axonometrice(1)

2 posibilitati de exprimare matematica:

- Proiectie ortogonală într-un plan oarecare – **obiect fix în spatiu, alegere plan** astfel încât să obtinem cele 3 cauzuri (izometrice, dimetrice, generale):

$$[xp \ y_p \ z_p \ 1] = [x \ y \ z \ 1]^* P_{ax}(N, R_0)$$

- Transformare geometrică (TA) urmata de proiectie ortografică în XOY (Po-xoy) – cu efect echivalent – **plan de proiectie fix (XOY), poziționare obiect în spatiu**:

$$[xp \ y_p \ z_p \ 1] = [x \ y \ z \ 1]^* TA^* Po-xoy = [x \ y \ z \ 1]^* PA$$

TA constă din rotații în jurul axelor OX și OY. Particularizând unghиurile de rotație obtinem cauzurile de proiectii axonometrice: izometrice, dimetrice, generale.

Proiectii paralele in planul XOY(10)

Proiectii axonometrice(2)

Exemplu:

$$TA = Ry(uy) * Rx(ux)$$

$$PA = TA * Po-xoy = \begin{bmatrix} \cos(uy) & \sin(ux)^*\sin(uy) & 0 & 0 \\ 0 & \cos(ux) & 0 & 0 \\ \sin(uy) & -\cos(uy)^*\sin(ux) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Se aplica transformarea PA versorilor axelor principale:

$$U * PA = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} * PA = \begin{bmatrix} X_{ux} & Y_{ux} & 0 & 1 \\ X_{uy} & Y_{uy} & 0 & 1 \\ X_{uz} & Y_{uz} & 0 & 1 \end{bmatrix}$$

Factorii de scalare a laturilor pe cele 3 axe sunt:

$$sx = (X_{ux}^2 + Y_{ux}^2)^{0.5}$$

$$sy = (X_{uy}^2 + Y_{uy}^2)^{0.5}$$

$$sz = (X_{uz}^2 + Y_{uz}^2)^{0.5}$$

Proiectii paralele în planul XOY(11)

Proiectii axonometrice(3)

1. Proiectii izometrice:

- Se impune conditia: $s_x = s_y = s_z$
- Rezulta: $u_x = +/- 35.26$ grade ; $u_y = +/- 45$ grade

2. Proiectii dimetrice: doi dintre factorii de scalare sunt egali

- Daca se impune conditia: $s_x = s_y$, rezulta:

$$u_x = \arcsin(+/- s_z \sqrt{2}), \quad u_y = \arcsin(+/- s_z \sqrt{2 - s_z^2})$$

- s_z poate fi ales intre 0 si 1;
- pentru fiecare valoare a lui s_z exista 4 proiectii dimetrice; de exemplu:
- pentru $s_z = 0.5$: $u_x = +/- 20.705$ grade, $u_y = 22.208$ grade

Transformari geometrice

3D

Prof. univ. dr. ing. Florica Moldoveanu

Transformari geometrice 3D



Sisteme de coordonate carteziene 3D

i, j, k: versorii directiilor axelor sistemului de coordonate

$$\text{dreapta: } \mathbf{k} = (\mathbf{i} \times \mathbf{j}) / |\mathbf{i} \times \mathbf{j}|$$

$$\text{stanga: } \mathbf{k} = (\mathbf{j} \times \mathbf{i}) / |\mathbf{j} \times \mathbf{i}|$$

Sistemul de coordonate in care este descrisa scena intr-o aplicatie: sistem de coordonate carteziene 3D dreapta.

Transformarile geometrice 3D elementare(1)

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1]^* M \quad \text{Transformare geometrica 3D in coordonate omogene}$$

Matricile de transformare folosind vectori linie pentru reprezentarea punctelor din spatiu

Translatia

$$[T] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tx & ty & tz & 1 \end{bmatrix} \quad \begin{aligned} T[tx, ty, tz] - & \text{vectorul} \\ & \text{de translatie} \\ x' = & x + tx \\ y' = & y + ty \\ z' = & z + tz \end{aligned}$$

Scalarea fata de origine

$$[S] = \begin{bmatrix} sx & 0 & 0 & 0 \\ 0 & sy & 0 & 0 \\ 0 & 0 & sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{aligned} sx, sy, sz: & \text{factorii de scalare} - \\ & \text{numere reale} \\ x' = & x * sx \\ y' = & y * sy \\ z' = & z * sz \end{aligned}$$

Rotatiile in jurul axelor principale

$$[R_z] = \begin{bmatrix} \cos \alpha & \sin \alpha & 0 & 0 \\ -\sin \alpha & \cos \alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [R_x] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \alpha & \sin \alpha & 0 \\ 0 & -\sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad [R_y] = \begin{bmatrix} \cos \alpha & 0 & -\sin \alpha & 0 \\ 0 & 1 & 0 & 0 \\ \sin \alpha & 0 & \cos \alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} x' = & x * \cos(\alpha) - y * \sin(\alpha) \\ y' = & x * \sin(\alpha) + y * \cos(\alpha) \\ z' = & z \end{aligned}$$

Deducerea matricilor de rotatie in jurul axelor principale → curs

Transformările geometrice 3D elementare (2)

Oglindirea fata de un plan al sistemului de coordonate

$$[O_z] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Forfecarea fata de origine

- De-a lungul axei OZ: modifica x si y proportional cu z

- $x' = x + Fx^*z$
- $y' = y + Fy^*z$
- $z' = z$



- Analog pentru forfecarea de-a lungul axei OX si a axei OY

- Cazul general (forfecarea pe toate cele 3 axe)

$$x' = x + y^*d + z^*g$$

$$y' = x^*b + y + z^*i$$

$$z' = x^*c + y^*f + z$$

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^* \begin{vmatrix} 1 & b & c & 0 \\ d & 1 & f & 0 \\ g & i & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

Transformari geometrice 3D compuse

Exemple:

1. Scalarea fata de un punct oarecare, $F(x_f, y_f, z_f)$.
2. Rotatia in jurul unei axe paralele cu o axa a sistemului de coordinate:
 1. Translatia obiectului astfel incat axa de rotatie sa se suprapuna peste o axa a sistemului de coordinate.
 2. Rotatia obiectului in jurul axei sistemului de coordinate.
 3. Translatia inversa celei din pasul 1.

Rezulta:

$$[x' \ y' \ z' \ 1] = [x \ y \ z \ 1] * M$$

sau

$$M = T(-xd, -yd, -zd) * R(u) * T(xd, yd, zd)$$

u : unghiul de rotatie

xd, yd, zd : un punct de pe axa de rotatie

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = MC * \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$MC = T(xd, yd, zd) * R(u) * T(-xd, -yd, -zd)$$

Transformari geometrice 3D compuse - exemplu

Rotatia cu un unghi u in jurul unei drepte oarecare

Se considera dreapta data printr-un punct (x_d, y_d, z_d) si directia sa, $D[a, b, c]$.

1. Translatie care face ca dreapta sa treaca prin origine: $T(-x_d, -y_d, -z_d)$
2. Alinierea dreptei cu una dintre axele principale, de ex. cu axa OZ:
 - 2.1. Rotatie in jurul axei OX, cu un unghi u_x , prin care dreapta ajunge in planul XOZ: $R_{OX}(u_x)$
 - 2.2. Rotatie in jurul axei OY, cu un unghi u_y , prin care dreapta se suprapune pe axa OZ: $R_{OY}(u_y)$
3. Rotatia cu unghiul dat, u , in jurul axei pe care s-a aliniat dreapta: rotatie in jurul axei OZ : $R_{OZ}(u)$
4. Transformarea inversa celei din pasul 2:
 - 4.1. Rotatie in jurul axei OY, cu unghiul $-u_y$: $R_{OY}(-u_y)$
 - 4.2. Rotatie in jurul axei OX, cu unghiul $-u_x$: $R_{OX}(-u_x)$
5. Transformarea inversa celei de la pasul 1: $T(x_d, y_d, z_d)$

Exercitiu: Sa se deduca: $\cos(u_x)$, $\sin(u_x)$, $\cos(u_y)$, $\sin(u_y)$.

Afisarea obiectelor 3D

Operatii asupra varfurilor:

- Transformari geometrice
- Proiectia 3D-2D
- Transformarea fereastra-poarta asupra varfurilor 2D

Afisare:

- Wire-frame (prin laturi) → reprezentare prin coordonate varfuri si laturi(legaturi intre varfuri)
- Cu fețe opace → reprezentare prin:
 - coordonate varfuri
 - contururi fețe (cycluri de varfuri): topologia obiectului

Reprezentarea obiectelor 3D

```
class Face
{
public:
    vector<int> contour;
public:
    Face(vector<int> list)
    { for (int i = 0; i < list.size(); i++)
        contour.push_back(list[i]);
    }
};
```

```
class Object3D
{
public:
    vector<Point3D*> vertices;
    vector<Point3D*> transf_vertices;
    vector<Face*> faces;
    Color color;
    bool fill; //tipul de interior
    .....
};
```

```
class PolyLine3D:public Object3D
{
public:
    PolyLine3D(vector<Point3D*> listV,Color _color)
    {
        .....
        color = _color;
        fill = false;
    }
};
```

```
class Transform3D
{
public:
    static float ModelMatrix[4][4]; // matrice in care se acumuleaza transformarile geometrice succesive
    static float ProjectionMatrix[4][4]; // matricea de proiectie
    static float MVPMatrix[4][4]; // acumuleaza transformarea de modelare cu cea de proiectie
    static bool proj_type; //false=proiectie paralela, true=proiectie perspectiva

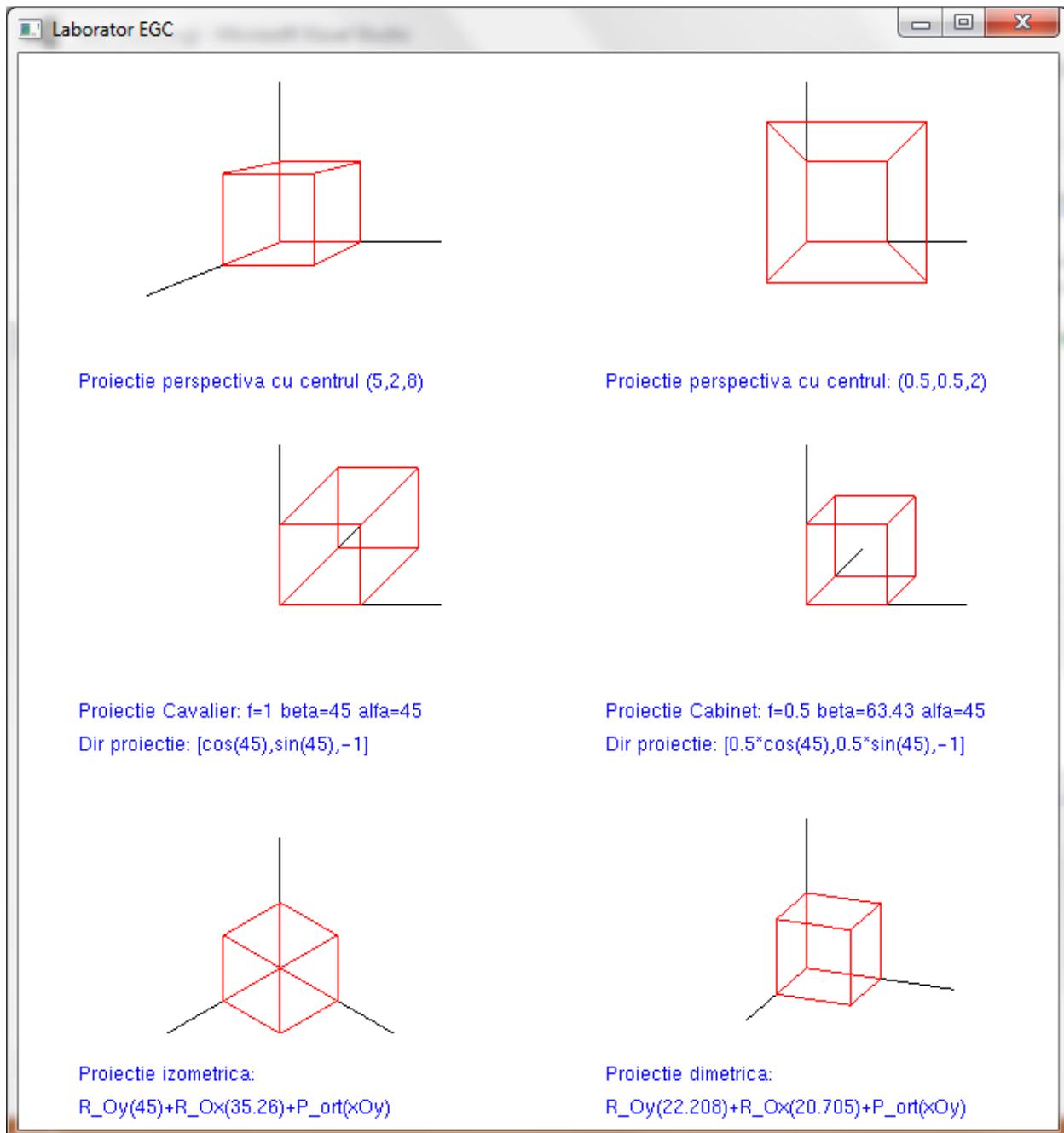
    static void loadIdentityModelMatrix(); //matricea curenta de modelare devine matricea identitate
    static void multiplyModelMatrix(float matrix[4][4]); //inmulteste matricea matrix cu matricea de
    modelare
    static void translateMatrix(float tx, float ty, float tz);
    static void scaleMatrix(float sx, float sy, float sz);
    static void rotateMatrixOx(float u);
    static void rotateMatrixOy(float u);
    static void rotateMatrixOz(float u);
    static void applyTransform(Object3D *o); // calculeaza MVPMatrix si o aplica varfurilor obiectului
    static void applyTransform(Point3D *p, Point3D *transf_p); // aplica MVPMatrix unui singur varf

    static void parallelProjectionMatrix(float a, float b, float c); // calculeaza matricea proiectiei paralele
                                                                // si o memoreaza in ProjectionMatrix
    static void perspectiveProjectionMatrix(float xc, float yc, float zc); // calculeaza matricea proiectiei
                                                                // perspectiva si o memoreaza in ProjectionMatrix
    static void obliqueProjectionMatrix (float f, float alfa);
    static void axonometricProjectionMatrix(float ux, float uy);

};
```

Program care afișează în fereastra aplicației 6 proiecții ale unui cub

În spațiul 3D cubul are laturile egale cu 1, fețele paralele cu planele principale ale sistemului de cordonate și colțul de (xmin, ymin, zmin) în originea sistemului de coordinate.



```

// Cele 6 proiectii se creaza in 6 contexte vizuale
Visual2D v2d1,v2d2,v2d3,v2d4,v2d5,v2d6;

Object3D cube;
PolyLine3D axa_ox, axa_oy, axa_oz;
Text text1, text2, text3, text4, text5, text6, text32, text42, text52, text62;
float n=1; // latura cubului

//functia care permite adaugarea de obiecte
void DrawingWindow::init()
{
    //adaugare contexte vizuale

    v2d1.init(-2.5,-2,2.5,2.5,0,0,DrawingWindow::width/2,DrawingWindow::height/3);
    v2d2.init(-2.5,-
2,2.5,2.5,DrawingWindow::width/2,0,DrawingWindow::width,DrawingWindow::height/3);
    v2d3.init(-2.5,-2,2.5,2.5,0,DrawingWindow::height/3,DrawingWindow::width/2, 2 *
                                              DrawingWindow::height/3);

    v2d4.init(-2.5,-
2,2.5,2.5,DrawingWindow::width/2,DrawingWindow::height/3,DrawingWindow::width, 2*
                                              DrawingWindow::height/3);

    v2d5.init(-2.5,-2,2.5,2.5,0,2 *
DrawingWindow::height/3,DrawingWindow::width/2,DrawingWindow::height);
    v2d6.init(-2.5,-2,2.5,2.5,DrawingWindow::width/2,2 *
DrawingWindow::height/3,DrawingWindow::width,DrawingWindow::height);

    v2d1.tipTran(true);
    v2d2.tipTran(true);
    v2d3.tipTran(true);
    v2d4.tipTran(true);
    v2d5.tipTran(true);
    v2d6.tipTran(true);

//creare cub
vector <Point3D> vertices;
vector <Face> faces;

//varfurile de jos
vertices.push_back( Point3D(0,0,0));
vertices.push_back( Point3D(n,0,0));

```

```
vertices.push_back( Point3D(n,0,n));
vertices.push_back( Point3D(0,0,n));
//varfurile de sus
vertices.push_back( Point3D(0,n,0));
vertices.push_back( Point3D(n,n,0));
vertices.push_back( Point3D(n,n,n));
vertices.push_back( Point3D(0,n,n));

//cele 6 fete
vector <int> contour;
//fata jos
    contour.clear();
    contour.push_back(0);
    contour.push_back(1);
    contour.push_back(2);
    contour.push_back(3);
    faces.push_back( Face(contour));
//fata sus
    contour.clear();
    contour.push_back(4);
    contour.push_back(5);
    contour.push_back(6);
    contour.push_back(7);
    faces.push_back( Face(contour));
//fata fata
    contour.clear();
    contour.push_back(0);
    contour.push_back(1);
    contour.push_back(5);
    contour.push_back(4);
    faces.push_back( Face(contour));
//fata dreapta
    contour.clear();
    contour.push_back(1);
    contour.push_back(2);
    contour.push_back(6);
    contour.push_back(5);
    faces.push_back( Face(contour));
//fata spate
    contour.clear();
```

```

contour.push_back(2);
contour.push_back(3);
contour.push_back(7);
contour.push_back(6);
faces.push_back( Face(contour));

//fata stanga
contour.clear();
contour.push_back(3);
contour.push_back(0);
contour.push_back(4);
contour.push_back(7);
faces.push_back( Face(contour));

cube.init(vertices,faces,Color(1,0,0),false);

// adauga cubul in cele 6 contexte vizuale
addObject3D_to_Visual2D(cube,v2d1);
addObject3D_to_Visual2D(cube,v2d2);
addObject3D_to_Visual2D(cube,v2d3);
addObject3D_to_Visual2D(cube,v2d4);
addObject3D_to_Visual2D(cube,v2d5);
addObject3D_to_Visual2D(cube,v2d6);

//Creaza axele sistemului de coordonate 3D
vector <Point3D> vertices_Ox;
vector <Point3D> vertices_Oy;
vector <Point3D> vertices_Oz;

vertices_Ox.push_back( Point3D(0,0,0));
vertices_Ox.push_back( Point3D(2,0,0));

vertices_Oy.push_back( Point3D(0,0,0));
vertices_Oy.push_back( Point3D(0,2,0));

vertices_Oz.push_back( Point3D(0,0,0));
vertices_Oz.push_back( Point3D(0,0,2));

axa_ox.init(vertices_Ox,Color(0,0,0));
axa_oy.init(vertices_Oy,Color(0,0,0));
axa_oz.init(vertices_Oz,Color(0,0,0));

```

```

// adauga axele in cele 6 contexte vizuale
    addObject3D_to_Visual2D(axa_ox,v2d1);
    addObject3D_to_Visual2D(axa_oy,v2d1);
    addObject3D_to_Visual2D(axa_oz,v2d1);

    addObject3D_to_Visual2D(axa_ox,v2d2);
    addObject3D_to_Visual2D(axa_oy,v2d2);
    addObject3D_to_Visual2D(axa_oz,v2d2);

    addObject3D_to_Visual2D(axa_ox,v2d3);
    addObject3D_to_Visual2D(axa_oy,v2d3);
    addObject3D_to_Visual2D(axa_oz,v2d3);

    addObject3D_to_Visual2D(axa_ox,v2d4);
    addObject3D_to_Visual2D(axa_oy,v2d4);
    addObject3D_to_Visual2D(axa_oz,v2d4);

    addObject3D_to_Visual2D(axa_ox,v2d5);
    addObject3D_to_Visual2D(axa_oy,v2d5);
    addObject3D_to_Visual2D(axa_oz,v2d5);

    addObject3D_to_Visual2D(axa_ox,v2d6);
    addObject3D_to_Visual2D(axa_oy,v2d6);
    addObject3D_to_Visual2D(axa_oz,v2d6);

//text
    text1.init("Proiectie perspectiva cu centrul (5,2,8)",Point2D(-2.5,-1.8),Color(0,0,1),BITMAP_HELVETICA_12);
    text2.init("Proiectie perspectiva standard: (0,0,-d)",Point2D(-2.5,-1.8),Color(0,0,1),BITMAP_HELVETICA_12);
    text3.init("Proiectie Cavalier: f=1 beta=45 alfa=45",Point2D(-2.5,-1.4),Color(0,0,1),BITMAP_HELVETICA_12);
    text32.init("Dir proiectie: [-cos(45),-sin(45),-1]",Point2D(-2.5,-1.8),Color(0,0,1),BITMAP_HELVETICA_12);
    text4.init("Proiectie Cabinet: f=0.5 beta=63.43 alfa=45",Point2D(-2.5,-1.4),Color(0,0,1),BITMAP_HELVETICA_12);
    text42.init("Dir proiectie: [-0.5*cos(45),-0.5*sin(45),-1]",Point2D(-2.5,-1.8),Color(0,0,1),BITMAP_HELVETICA_12);

```

```

text5.init("Proiectie izometrica:",Point2D(-2.5,-
1.4),Color(0,0,1),BITMAP_HELVETICA_12);
text52.init("R_Oy(45)+R_Ox(35.26)+P_ort(xOy)",Point2D(-2.5,-
1.8),Color(0,0,1),BITMAP_HELVETICA_12);
text6.init("Proiectie dimetrica:",Point2D(-2.5,-
1.4),Color(0,0,1),BITMAP_HELVETICA_12);
text62.init("R_Oy(22.208)+R_Ox(20.705)+P_ort(xOy)",Point2D(-2.5,-
1.8),Color(0,0,1),BITMAP_HELVETICA_12);

addText_to_Visual2D(text1,v2d1);
addText_to_Visual2D(text2,v2d2);
addText_to_Visual2D(text3,v2d3);
addText_to_Visual2D(text32,v2d3);
addText_to_Visual2D(text4,v2d4);
addText_to_Visual2D(text42,v2d4);
addText_to_Visual2D(text5,v2d5);
addText_to_Visual2D(text52,v2d5);
addText_to_Visual2D(text6,v2d6);
addText_to_Visual2D(text62,v2d6);

int i;
Transform3D::loadIdentityModelMatrix();

//primul cub - proiectie perspectiva cu centru oarecare de proiectie
Transform3D::perspectiveProjectionMatrix(5,2,8);
for (i = 0; i < v2d1.visual2D_objects3D.size(); i++)
{
    Transform3D::applyTransform(v2d1.visual2D_objects3D[i]);
}

//al doilea cub - proiectie perspectiva cu centru oarecare de proiectie
Transform3D::perspectiveProjectionMatrix(0.5,0.5,2);
for (i = 0; i < v2d2.visual2D_objects3D.size(); i++)
{
    Transform3D::applyTransform(v2d2.visual2D_objects3D[i]);
}

//al treilea cub - proiectie Cavalier: f = 1, alfa = 45 grade => a = cos(PI/4),
// b = sin(PI/4), c = - 1

```

```

Transform3D::parallelProjectionMatrix(cos(PI/4), sin(PI/4), -1);
// sau Transform3D::obliqueProjectionMatrix (1, 45);
for (i = 0; i < v2d3.visual2D_objects3D.size(); i++)
{
    Transform3D::applyTransform(v2d3.visual2D_objects3D[i]);
}

// al patrulea cub - proiectie Cabinet: f = 0.5, alfa = 45 grade => a = 0.5 * cos(PI/4),
// b = 0.5 * sin(PI/4), c = - 1
Transform3D::parallelProjectionMatrix(0.5 * cos(PI/4), 0.5 * sin(PI/4), -1);
// sau Transform3D::obliqueProjectionMatrix (0.5, 45);
for (i = 0; i < v2d4.visual2D_objects3D.size(); i++)
{
    Transform3D::applyTransform(v2d4.visual2D_objects3D[i]);
}

//al cincilea cub - proiectie izometrica
//rotatie fata de Oy (cu unghiul 45 grade),
// urmata de rotatie fata de Ox (cu unghiul 35.26 grade)
// urmata de proiectie ortografica in planul xOy
Transform3D::loadIdentityModelMatrix();
Transform3D::rotateMatrixOy(45.0f * PI/180.0f);
Transform3D::rotateMatrixOx(35.26f * PI/180.0f);
Transform3D::parallelProjectionMatrix(0,0,1);
// sau Transform3D::axonometricProjectionMatrix(35.26, 45);
for (i = 0; i < v2d5.visual2D_objects3D.size(); i++)
{
    Transform3D::applyTransform(v2d5.visual2D_objects3D[i]);
}

//al saselea cub - proiectie dimetrica
//rotatie fata de Oy (cu unghiul 22.208 grade),
// urmata de rotatie fata de Ox (cu unghiul 20.705 grade)
// urmata de proiectie ortografica in planul xOy
Transform3D::loadIdentityModelMatrix();
Transform3D::rotateMatrixOy(22.208f * PI/180.0f);
Transform3D::rotateMatrixOx(20.705f * PI/180.0f);
Transform3D::parallelProjectionMatrix(0,0,1);
// sau Transform3D::axonometricProjectionMatrix(20.705, 22.208);

for (i = 0; i < v2d6.visual2D_objects3D.size(); i++)

```

```

    {
        Transform3D::applyTransform(v2d6.visual2D_objects3D[i]);
    }

// adauga cele 6 contexte vizuale in fereastra curenta
addVisual2D(v2d1);
addVisual2D(v2d2);
addVisual2D(v2d3);
addVisual2D(v2d4);
addVisual2D(v2d5);
addVisual2D(v2d6);

}

//functia care se apeleaza la redimensionarea ferestrei
void DrawingWindow::onReshape(int width,int height)
{
    v2d1.poarta(0,0,width/2,height/3);
    v2d2.poarta(width/2,0,width,height/3);

    v2d3.poarta(0,height/3,width/2,2 * height/3);
    v2d4.poarta(width/2,height/3,width,2 * height/3);

    v2d5.poarta(0,2 * height/3,width/2,height);
    v2d6.poarta(width/2,2 * height/3,width,height);
}

int main(int argc, char** argv)
{
    //creare fereastra
    DrawingWindow dw(argc, argv, 600, 600, 200, 50, "Laborator EGC");
    //se apeleaza functia init() - in care s-au adaugat obiecte
    dw.init();
    //se intra in bucla principala de desenare
    dw.run();
    return 0;
}

```

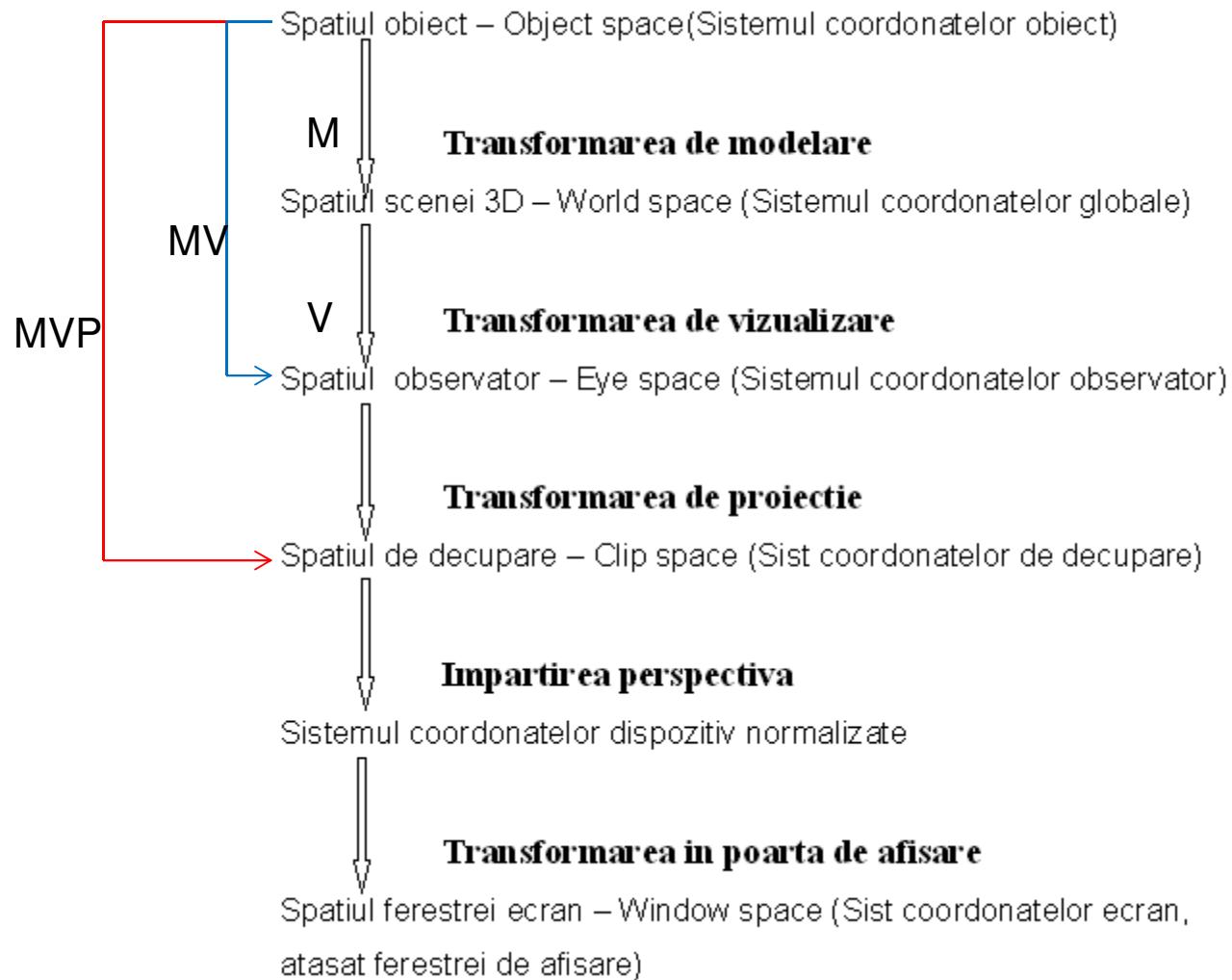
Transformarea de vizualizare din spațiu 3D

Prof. univ. dr. ing. Florica Moldoveanu

Transformarea de vizualizare din spatiul 3D

- Are ca scop transformarea modelului unei scene 3D intr-o imagine redată pe o suprafață de afisare (ecran).
- Poate fi comparată cu transformarea efectuată de un aparat foto asupra lumii reale pentru a se obține imaginea dintr-o fotografie:
 - Imaginea obținută depinde de:
 - poziția ochiului (observatorului)
 - direcția în care priveste (observatorul)
 - ceea ce vede ochiul prin vizorul aparatului foto (volumul vizual)
- Este o transformare compusă din mai multe transformări, între care și o proiecție $R^3 \rightarrow R^2$
- Se efectuează asupra varfurilor obiectelor din scena 3D.
- Include și operația de decupare (clipping): eliminarea partilor din scena 3D care sunt în afara volumului vizual.
 - OpenGL efectuează automat transformarea de vizualizare asupra varfurilor obiectelor 3D descrise într-o aplicatie, existând înțotdeauna o matrice curentă a transformării de vizualizare din spatiul 3D.

Transformarea varfurilor în OpenGL



Transformarea de modelare (1)

Spatiul obiect

- În mod uzual, fiecare obiect este definit în propriul sau sistem de coordonate (sistemul coord locale – 3D « dreapta »); exemple: cubul, sfera, scheletul unui personaj animat, s.a.
- Avantaje: usurinta modelarii si reutilizarea modelelor obiectelor 3D

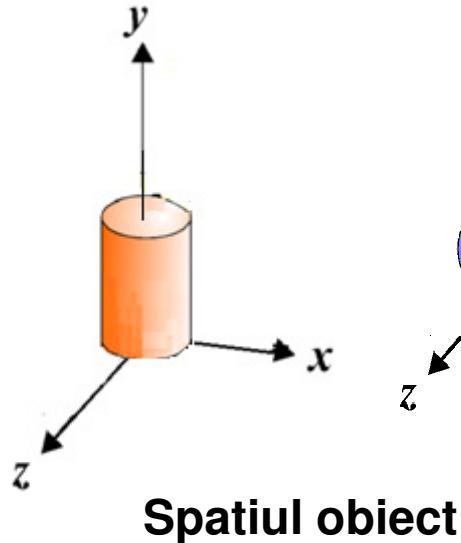
Spatiul scenei 3D (Spatiul lumii reale -World space)

- Spatiul în care este compusa scena 3D
- Raportat la un sistem de referinta global – sist de coord carteziene 3D “dreapta”
- Originea sistemului de coord este stabilita de modelatorul scenei 3D

Transformarea de modelare

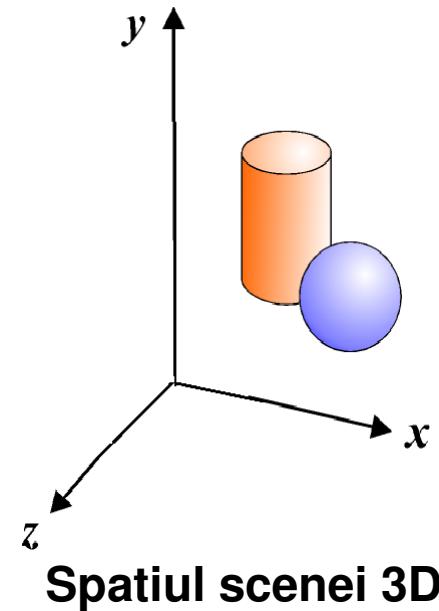
- ❖ Scop: dimensionarea si poziționarea obiectelor în scena 3D.
- ❖ Este o transformare geometrica compusa.

Transformarea de modelare (2)



Obiecte definite în sisteme de coordonate proprii (sisteme de coordonate locale)

Transformarea de modelare



Obiecte definite fata de același sistem de coordonate (sistemul de coordonate globale)

Translatie, Rotatie, Scalare

Transformarea de vizualizare (1)

Spatiul observator (Eye space)

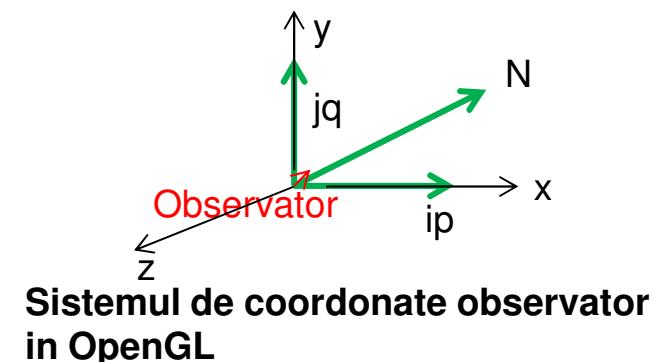
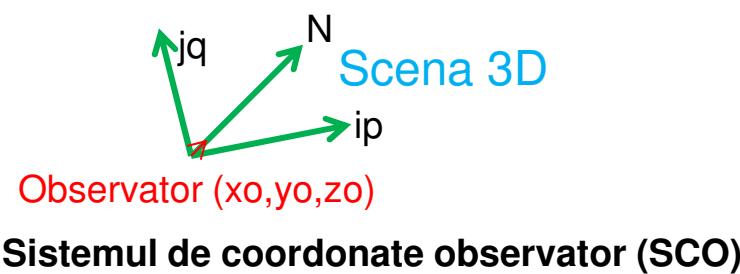
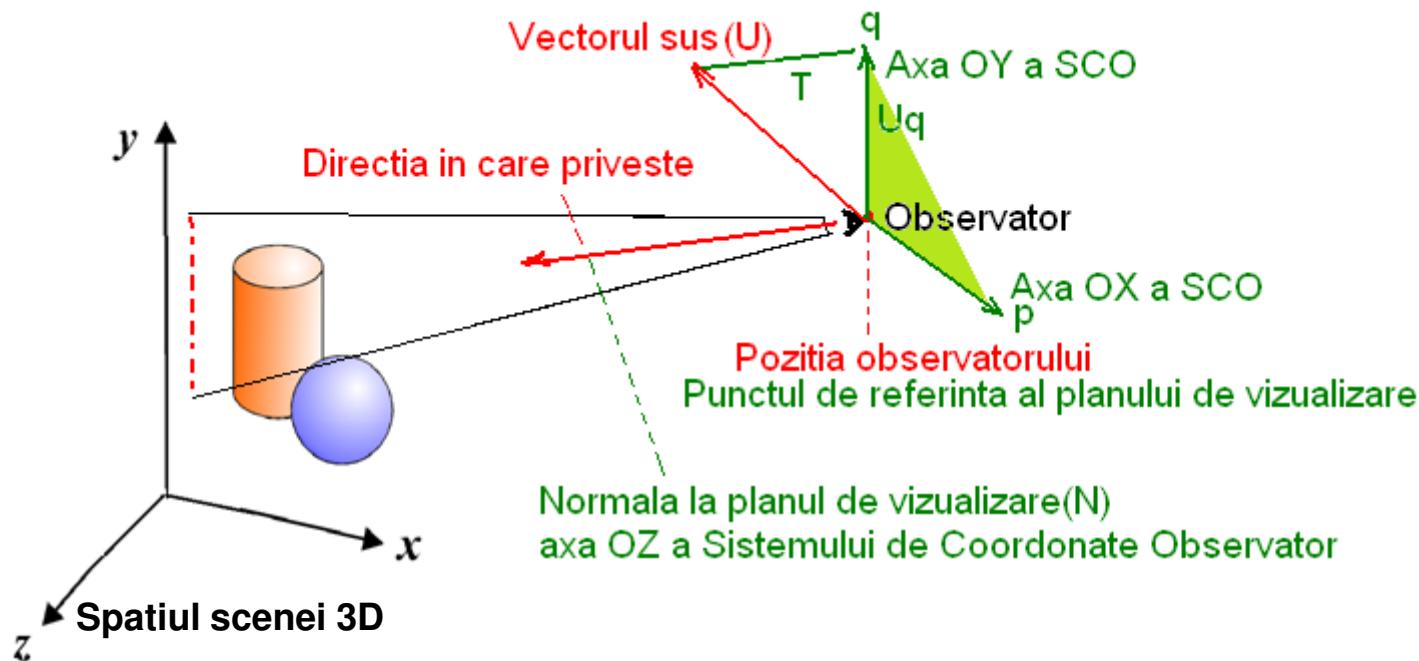
- O imagine este o vedere obtinuta dintr-un punct al spatiului 3D, similara unei fotografii:
- Depinde de:
 - Pozitia observatorului in spatiul scenei,
 - Directia in care priveste si ceea ce observatorul reușește să vada prin vizorul aparatului de fotografiat.
 - Rotatia aparatului fata de axa sa.

Sistemul de coordonate observator: atasat planului de vizualizare

- ❖ **Sistem de coordonate 3D stanga, definit prin 3 parametri:**
 - Originea sa: pozitia observatorului (ochiului) – aflata in planul de vizualizare
 - Normala la planul de vizualizare : directia axei OZ a sistemului de coordonate observator- este directia in care priveste observatorul (centru scenei 3D)
 - Directia sus a planului de vizualizare, care determina axa OY a sist. de coordonate

Transformarea de vizualizare (2)

din sistemul de coordonate globale \rightarrow în sistemul de coordonate observator



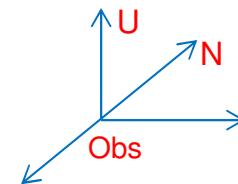
Transformarea de vizualizare (3)

Determinarea sistemului de coordonate observator folosind cei 3 parametri: CURS

In OpenGL, valorile **implicite** ale parametrilor care definesc sistemul de coordonate observator sunt:

- poziția observatorului (camera de luat vederi): **originea sistemului de coordonate globale**
- direcția în care privește observatorul: **direcția negativa a axei OZ**
- direcția sus a planului de vizualizare: **direcția pozitivă a axei OY**

Cu aceste valori **implicite**, transformarea de vizualizare este
transformarea « dreapta-stanga »: $x' = x$, $y' = y$, $z' = -z$.



Funcția **gluLookAt()** permite modificarea valorilor **implicite** ale parametrilor transformării de vizualizare (care definesc sistemul de coordonate observator):

```
void gluLookAt(GLdouble eyex,GLdouble eyey,GLdouble eyez, GLdouble centrx, GLdouble centry,  
               GLdouble centerz, GLdouble upx,GLdouble upy, GLdouble upz);
```

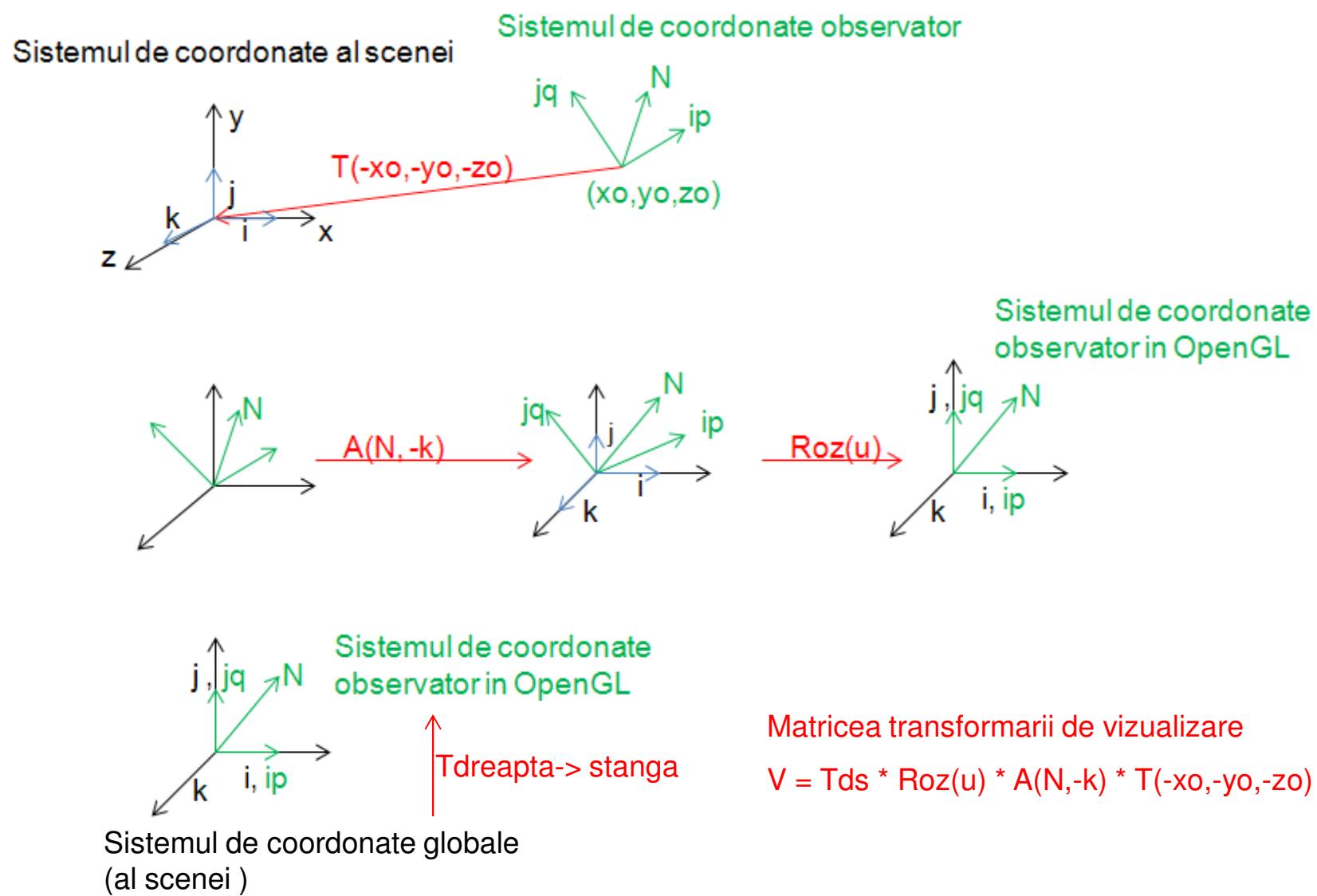
unde:

eyex, eyey, eyez reprezintă **poziția observatorului**

centerx, centrey, centerz reprezintă **direcția în care se privește observatorul (centrul scenei 3D)**

upx, upy, upz reprezintă **direcția vectorului « sus » al planului de vizualizare**

Transformarea de vizualizare (4)



Transformarea de modelare si vizualizare

din sistemul de coordonate obiect → în sistemul de coordonate observator

- ❖ Transformare compusa (model-view) reprezentata printr-o singura matrice in OpenGL

$$VM = V \bullet M$$

M – matricea de modelare curentă

V – matricea de vizualizare curentă

- ❖ Înmulțirea celor 2 matrici este efectuată automat de OpenGL: există în totdeauna o matrice VM!

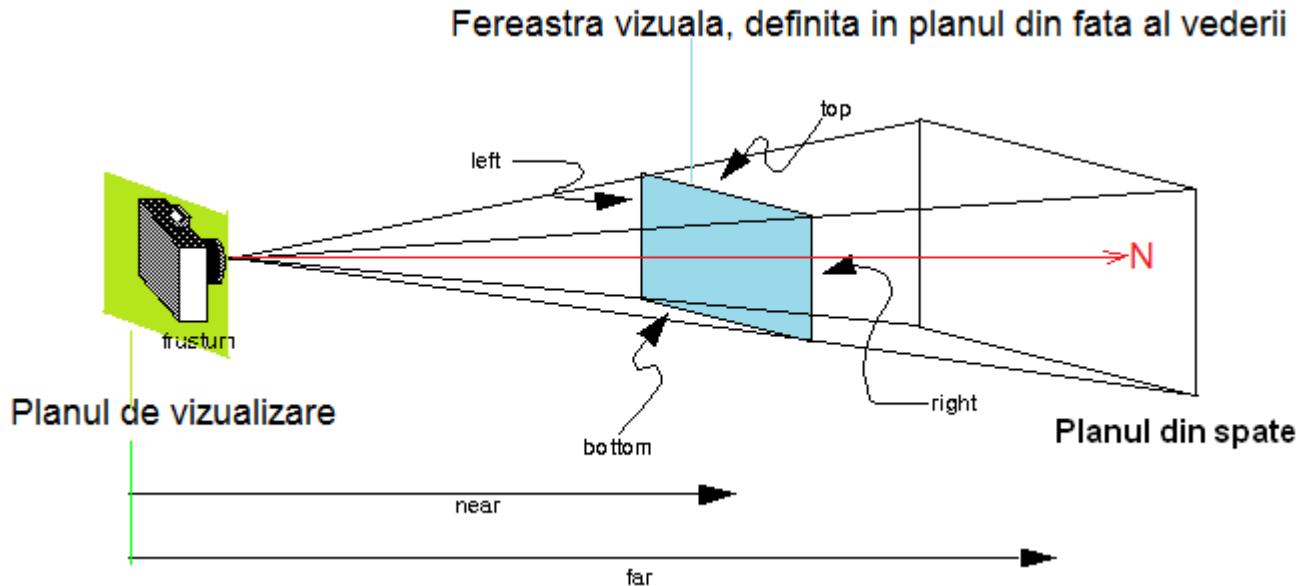
$$[xe \; ye \; ze \; we]^T = VM \bullet [xo \; yo \; zo \; wo]^T$$

Transformarea de proiectie (1)

❖ Determinata de **volumul vizual** definit de programator:

- trunchi de piramida pentru proiectia perspectiva
- paralelipiped dreptunghic pentru proiectia paralela

Volumul vizual al proiectiei perspectiva: trunchi de piramida delimitat de planul din fata (near) si planul din spate (far) al vederii; near si far – valori pozitive masurate pe axa de profunzime a planului de vizualizare.



Transformarea de proiectie (2)

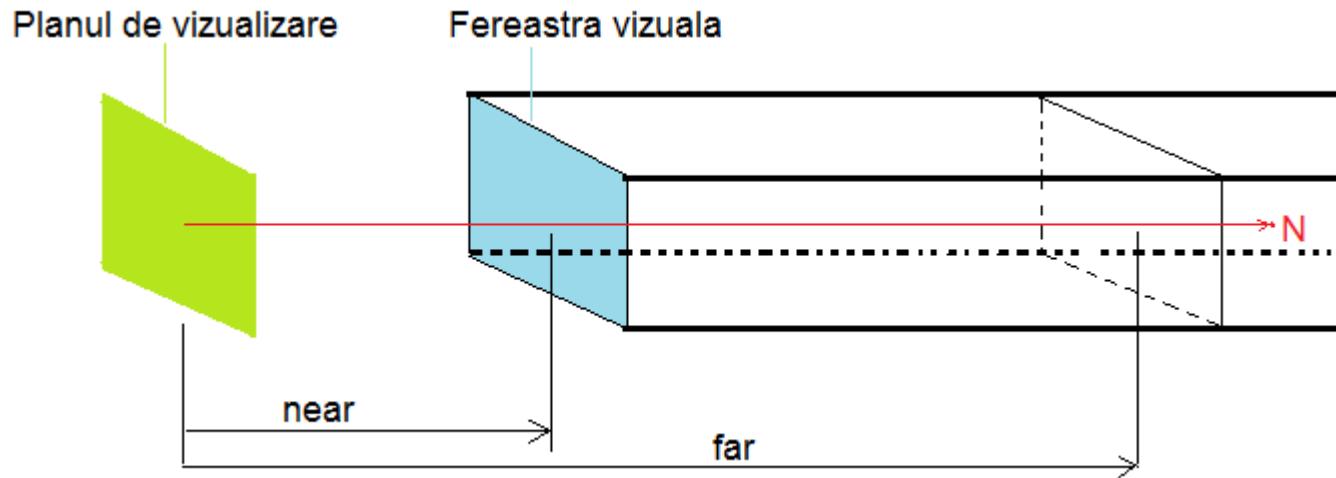
In OpenGL, volumul vizual al proiectiei perspectiva poate fi specificat prin apelul functiei:

`void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);`

-Funcția definește o proiecție perspectivă cu centrul de proiecție în poziția observatorului .

Volumul vizual al proiectiei ortografice

- Directia de proiectie este directia axei N a planului de vizualizare
- Volumul vizual este un paralelipiped dreptunghic cu laturile paralele cu directia de proiectie, delimitat in adancime de planul din fata (near) si planul din spate (far) al vederii



Transformarea de proiectie (3)

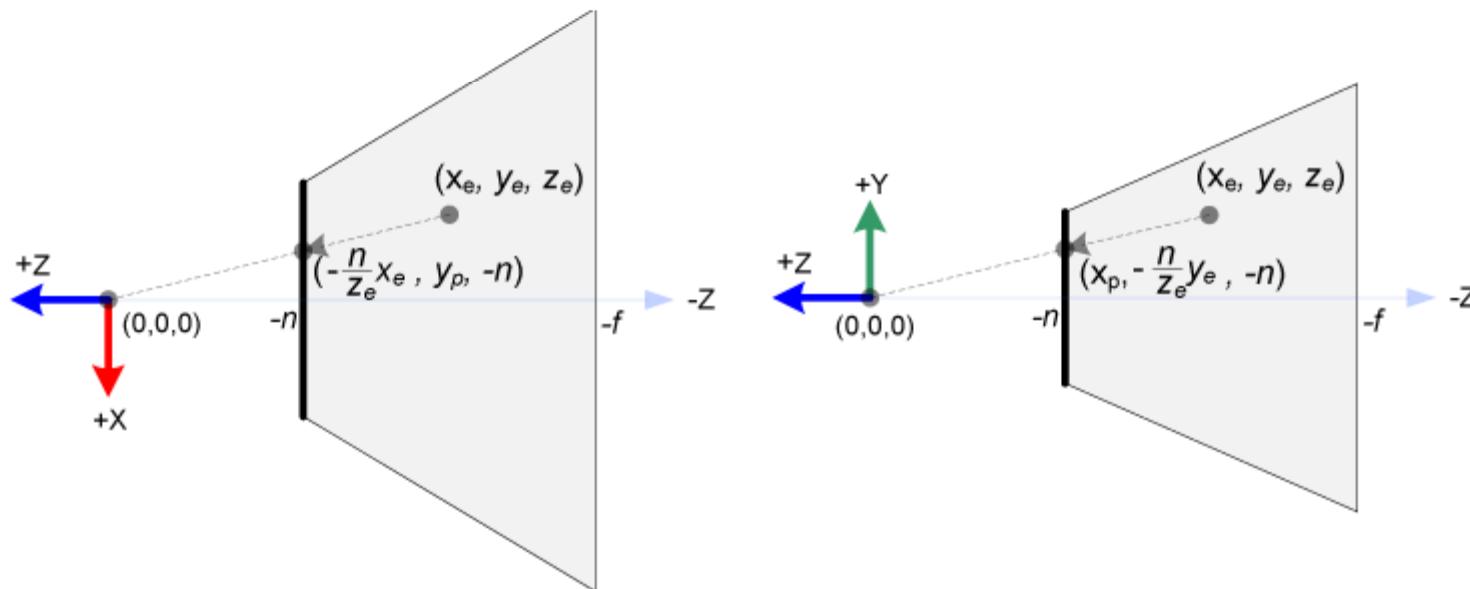
In OpenGL, volumul vizual al proiectiei ortografice poate fi specificat prin apelul functiei:

`void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);`

❖ Pentru ambele tipuri de proiectii, proiectia se efectueaza in (fereastra din) planul din fata:

$z=-\text{near}$

Proiectia perspectiva a unui punct in coordonate observator, (x_e, y_e, z_e) , pe planul din fata al vederii, cu centrul de proiectie in $(0,0,0)$.



$x_p = (-\text{near}/z_e)*x_e, \quad y_p = (-\text{near}/z_e)*y_e$ – se pot obtine plecand de la formulele proiectiei perspectiva in XOY sau din asemanarea triunghiurilor: $x_p/x_e = -n/z_e, \quad y_p/y_e = -n/z_e$.

Transformarea de proiectie (4)

Proiectia ortografica a unui punct in coordonate observator, (x_e, y_e, z_e), pe planul din fata al vederii:

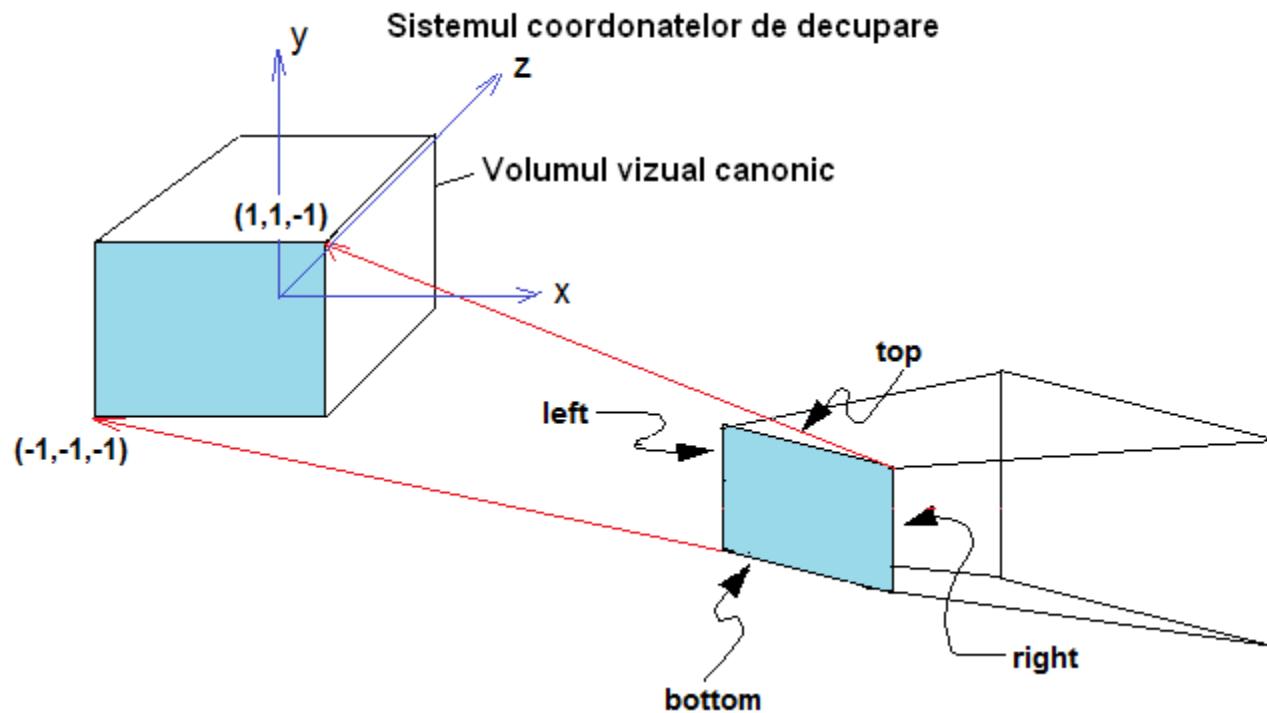
$$x_p = x_e, \quad y_p = y_e, \quad z_p = -\text{near}.$$

- ❖ Transformarea de proiectie este insotita de operatia de decupare a primitivelor grafice la marginile volumului vizual.
- ❖ Pentru simplificarea calculelor de decupare, volumul vizual definit de programator este transformat (de OpenGL) in **volumul vizual canonic**: cub cu latura de 2 unitati, centrat in originea sistemului de coordonate de decupare, cu laturile paralele cu axe.
- ❖ Transformarea volumului vizual in volumul vizual canonic se numeste **transformarea de normalizare** = transformarea de proiectie, in OpenGL.

Dupa transformarea varfurilor prin transformarea de proiectie, primitivele grafice 3D sunt decupate la marginile volumului vizual canonic.

Transformarea de proiecție (5)

- *Sistemul coordonatelor de decupare*: sistem de coordonate 3D stanga
- Prin transformarea de proiecție, colțurile ferestrei 2D din planul de proiecție, (left, bottom,-near) și (right, top,-near) sunt mapate pe colțurile stânga-jos și dreapta-sus ale ferestrei 2D din sistemul coordonatelor de decupare, adică (-1,-1,-1) și (1,1,-1).



Transformarea de proiectie (6)

Impunand conditia $(left, right] \rightarrow [-1, 1]$ si $[bottom, top] \rightarrow [-1, 1]$, rezulta transformarea punctelor din planul din fata, (xp, yp) , in sistemul coordonatelor de decupare (**clip coordinates**):

$$xd = 2 * xp / (right - left) - (right + left) / (right - left)$$

$$yd = 2 * yp / (top - bottom) - (top + bottom) / (top - bottom)$$

Inlocuind xp si yp rezulta:

$$xd = (xe * 2 * near / (right - left) + ze * (right + left) / (right - left)) / (-ze)$$

$$yd = (ye * 2 * near / (top - bottom) + ze * (top + bottom) / (top - bottom)) / (-ze)$$

Pentru transformarea coordonatelor z se impun conditiile: $-near \rightarrow -1$, $-far \rightarrow 1$;

Rezulta:

$$zd = (ze * (far + near) / (far - near) - 2 * far * near / (far - near)) / (-ze)$$

Punctul in coordonate omogene: (xc, yc, zc, w)

$$xd = xc / w; \quad yd = yc / w; \quad zd = zc / w; \quad w = -ze$$

Transformarea de proiecție (7)

Matricea de proiecție pentru proiecția perspectivă este:

$$P = \begin{bmatrix} \frac{2 * \text{near}}{\text{right} - \text{left}} & 0 & A & 0 \\ 0 & \frac{2 * \text{near}}{\text{top} - \text{bottom}} & B & 0 \\ 0 & 0 & C & D \\ 0 & 0 & -1 & 0 \end{bmatrix} \quad \begin{aligned} A &= \frac{\text{right} + \text{left}}{\text{right} - \text{left}} & B &= \frac{\text{top} + \text{bottom}}{\text{top} - \text{bottom}} \\ C &= -\frac{\text{far} + \text{near}}{\text{far} - \text{near}} & D &= -\frac{2 * \text{far} * \text{near}}{\text{far} - \text{near}} \end{aligned}$$

Pentru proiecția ortografică

$$\begin{aligned} xp = xe \text{ și } yp = ye \rightarrow xd &= 2 * xe / (\text{right} - \text{left}) - (\text{right} + \text{left}) / (\text{right} - \text{left}) \\ yd &= 2 * ye / (\text{top} - \text{bottom}) - (\text{top} + \text{bottom}) / (\text{top-bottom}) \end{aligned}$$

Matricea de proiecție pentru proiecția ortografică este:

$$P = \begin{bmatrix} \frac{2}{\text{right} - \text{left}} & 0 & 0 & -A \\ 0 & \frac{2}{\text{top} - \text{bottom}} & 0 & -B \\ 0 & 0 & \frac{-2}{\text{far} - \text{near}} & C \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Transformarea de proiectie (6)

din coordonate observator → in coordonate de decupare

$$[x_c \ y_c \ z_c \ w]^T = P \cdot [x_o \ y_o \ z_o \ w_o]^T$$

Impartirea perspectiva:

$$x_d = x_c/w, \quad y_d = y_c/w, \quad z_d = z_c/w$$

(x_d, y_d, z_d): coordonate dispozitiv normalizate

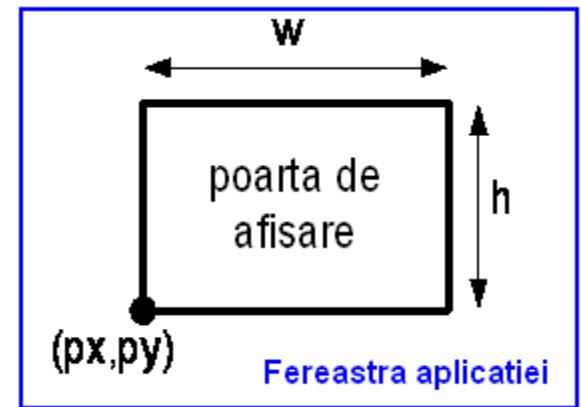
Transformarea de modelare-vizualizare-proiectie

- ❖ Matricile transformarilor de modelare, vizualizare si proiectie sunt inmultite de OpenGL, rezultand transformarea compusa numita “model-view-projection”, a.i. asupra varfurilor definite in sistemul coordonatelor obiect este efectuata o singura transformare:

$$PVM = P \cdot V \cdot M \quad (\text{matricea ModelViewProjection})$$

Transformarea în poarta de afisare

- Transformare fereastra-poarta
- Fereastra este fereastra 2D din sistemul coordonatelor de decupare, având colțurile în $(-1, -1, -1) - (1, 1, -1)$
- Poarta este definită de programator folosind funcția:
`void glViewport(GLint px, GLint Py, GLsizei width, GLsizei height);`
(px, py) reprezintă coordonatele în fereastra ecran a aplicatiei ale colțului stânga jos al porții de afișare (în pixeli). Valorile implicate : $(0,0)$.
width, height reprezintă lățimea, respectiv înălțimea porții de afișare.
Valorile implicate sunt date de lățimea și înălțimea ferestrei aplicatiei.
- Transformarea în poarta de afisare este definită astfel:
 $xw = ox + (width/2)xd$
 $yw = oy + (height/2)yd$
 $zw = ((f-n)/2)zd + (n+f)/2$ // conservă coordonata z pentru eliminarea partilor nevizibile
 (ox, oy) reprezintă coordonatele centrului porții de afișare
width, height reprezintă lățimea, respectiv înălțimea porții de afișare
n și f au valorile implicate 0.0 respectiv 1.0, dar pot fi modificate la valori cuprinse în intervalul $[0,1]$ folosind funcția **DepthRange()**



Eliminarea partilor nevizibile ale scenelor 3D din imagini

Prof. univ. dr. ing. Florica Moldoveanu

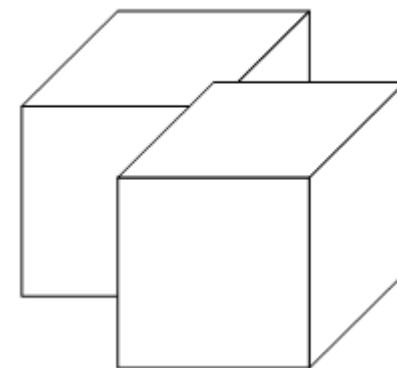
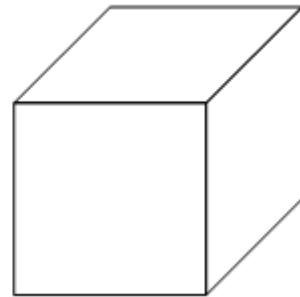
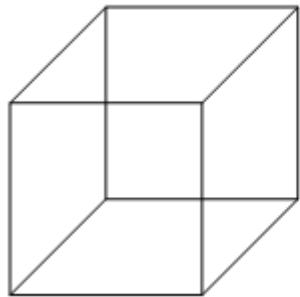
Eliminarea partilor nevizibile ale scenelor 3D, din imagini

Vizibilitatea obiectelor dintr-o scena 3D depinde de:

- Pozitia observatorului
- Poziionarea obiectelor din scena unul fata de celalalt
- Volumul vizual

Partile nevizibile ale unui obiect 3D sunt:

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



Eliminarea partilor nevizibile din scenele 3D

Terminologie:

- **Object culling** – eliminarea din banda grafica a obiectelor sau grupurilor de obiecte care sunt in afara volumului vizual
 - nu este efectuata de OpenGL
 - poate fi efectuata prin algoritmi implementati intr-o biblioteca folosita de aplicatia grafica 3D (motorul grafic)
- **Back face culling** – eliminarea din banda grafica a fețelor auto-obturate ale obiectelor (poate fi efectuata de OpenGL)
- **Hidden surface removal** – eliminarea partilor nevizibile ale fețelor obiectelor:
 - este efectuata de GPU, ca operatie raster la nivel de fragment (z-buffer)
 - poate fi implementata si la nivel de poligoane, prin algoritmi implementati intr-o biblioteca folosita de aplicatia grafica 3D (motorul grafic)

Determinarea fetelor auto-obturate pentru poliedre convexe (Back face culling)(1)

- 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.

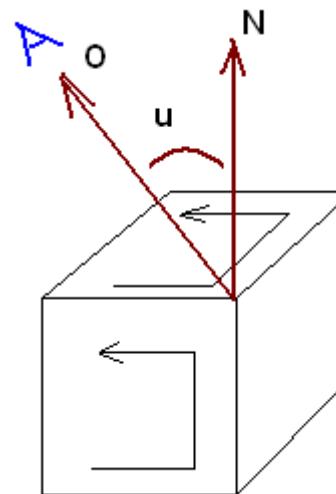
Fața este vizibila daca : $0 \leq u < 90^\circ$

sau

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

N: normala la față

O: vectorul orientat catre observator



Determinarea fetelor auto-obturate pentru poliedre convexe(2)

Produsul scalar:

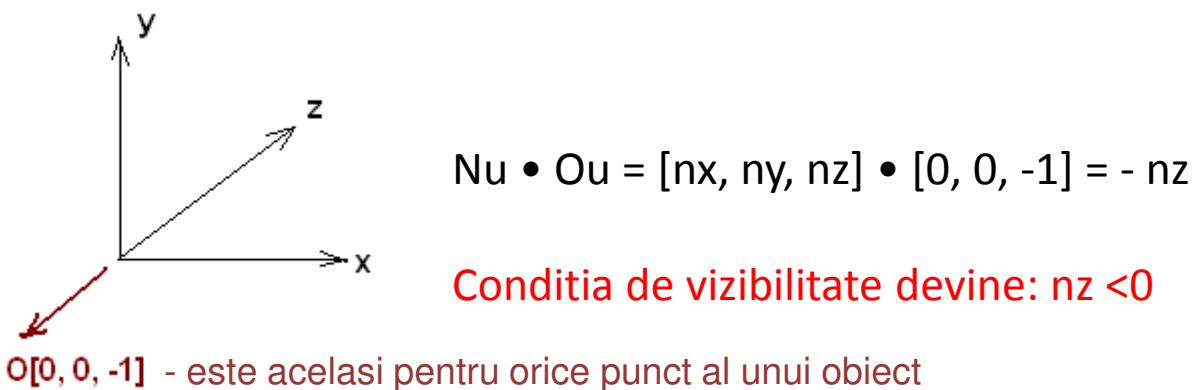
$$\mathbf{N} \cdot \mathbf{O} = ||\mathbf{N}|| * ||\mathbf{O}|| * \cos(u) \rightarrow \cos(u) = \mathbf{N} \cdot \mathbf{O} / (||\mathbf{N}|| * ||\mathbf{O}||) = \mathbf{N} \cdot \mathbf{O} / \mathbf{O}$$

Conditia de vizibilitate: $\mathbf{N} \cdot \mathbf{O} > 0$ (produsul scalar la versorilor)

Determinarea fetelor auto-obturate poate fi efectuata:

1. In sistemul coordonatelor globale (in care este definit si observatorul)
2. In sistemul de coordonate observator
3. In sistemul coordonatelor de decupare (dupa aplicarea transformarii "model-view-projection")

In cazul 3:



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

Observatorul este la infinit, pe axa z negativa, si priveste in directia axei Z pozitive

Determinarea fetelor auto-obturate pentru poliedre convexe(3)

Calculul normalei la o față a poliedrului

Fie $V_1(x_1, y_1, z_1)$, $V_2(x_2, y_2, z_2)$, $V_3(x_3, y_3, z_3)$, 3 varfuri succesive ale conturului feței.

Normala la față se poate obține calculând produsul vectorial: $(V_1-V_2) \times (V_2-V_3)$

$$\begin{aligned} \mathbf{N} = (V_1-V_2) \times (V_2-V_3) = & [(z_3-z_2)*(y_2-y_1) - (z_2-z_1)*(y_3-y_2)]*\mathbf{i} \\ & - [(x_2-x_1)*(z_3-z_2) - (x_3-x_2)*(z_2-z_1)]*\mathbf{j} \\ & + [(x_2-x_1)*(y_3-y_2) - (x_3-x_2)*(y_2-y_1)]*\mathbf{k} \end{aligned}$$

unde \mathbf{i} , \mathbf{j} , \mathbf{k} sunt versorii direcțiilor axelor sistemului de coordonate carteziene 3D

Normalele:

- Pot fi atașate varfurilor în aplicația OpenGL → sunt transformate de OpenGL în sistemul coordonatelor de decupare.
- Nu sunt atașate varfurilor: sunt calculate de OpenGL (dacă s-a cerut eliminarea fetelor auto-obturate) folosind varfurile transformate în sistemul coordonatelor de decupare.

Determinarea fetelor auto-obturate pentru poliedre convexe(4)

Pentru eliminarea fetelor auto-obturate de catre OpenGL:

```
glCullFace(GL_FRONT/ GL_BACK/GL_FRONT_AND_BACK);
```

- eliminare fete: din fata / din spate / toate (nu vor fi afisate fețe, ci numai alte primitive: puncte, linii)

```
glFrontFace(GL_CCW/GL_CW);
```

- specifica orientarea fetelor din fata(in spatiul coordonatelor globale): trigonometrica/sensul acelor de ceas

Daca ciclul de varfuri al fiecarei fete este orientat trigonometric atunci cand obiectul este privit din exteriorul sau:

▪ atunci cand observatorul este in spatiul exterior obiectului se va apela **glFrontFace(GL_CCW)**

▪ atunci cand observatorul este in spatiul interior obiectului se va apela **glFrontFace(GL_CW)**

(vazute din interior, fetele obiectului au conturul orientat anti-trigonometric)

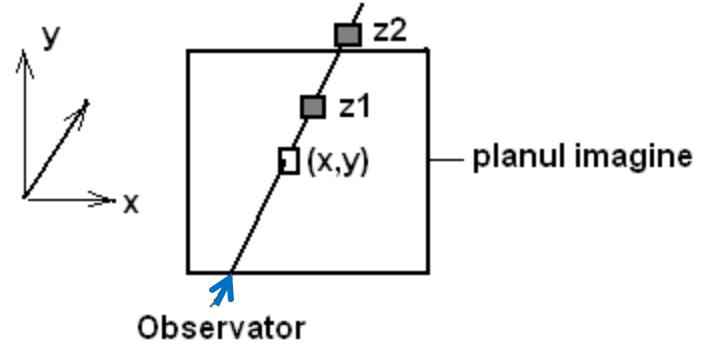
Algoritm z-buffer(1)

- Algoritm de eliminare a partilor nevizibile ale poligoanelor (fețelor obiectelor)
- Executat in timpul rasterizarii primitivelor (dupa transformarea de proiectie), de GPU:

- Observatorul este la infinit, pe axa z negativa
- Asupra primitivelor se efectueaza proiectie ortografica in planul XOY

Doua fragmente aflate pe acelasi projector, rezultate din rasterizarea a doua primitive diferite, au coordonata z diferita:

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



Algoritmul z-bufer(2)

- Primitivele grafice (poligoane, linii) in care a fost descompusa scena 3D sunt rasterizate in ordinea in care au fost transmise de programul de aplicatie.
- Buffer-ul imagine 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
- in memoria placi grafice

Algoritmul z-bufer(3)

Algoritmul z-buffer:

- * Initializeaza buffer-ul imagine la culoarea de fond
- * Initializeaza 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

daca $z < Z\text{-buffer}[y][x]$ atunci

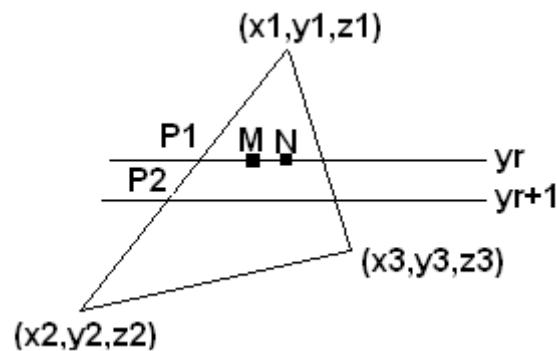
$Z\text{-buffer}[y][x] = z$

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

Algoritmul z-bufer(4)

Calculul coordonatei z a unui fragment

-Ultima transformare din lantul de transformari efectuate asupra varfurilor 3D conservă coordonata z a fiecarui varf.



Coordonatele z ale punctelor (fragmentelor) P_1 , P_2 , M , N se obțin prin calcul incremental, în algoritmul de rasterizare.

Algoritm z-bufer(5)

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

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

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

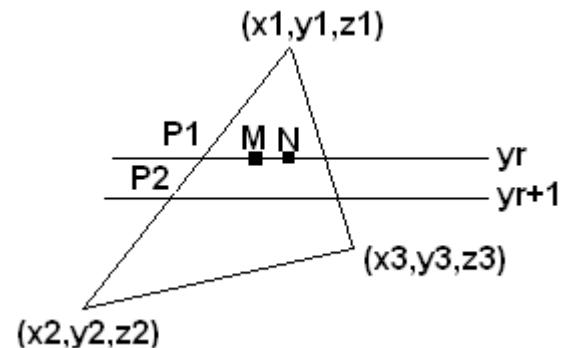
$A*x + B*y + C*z + D = 0$ ec planului poligonului

$zP1 = (-A*xP1 - B*yr - D)/C$

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

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

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



Algoritm z-bufer(6)

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 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) “Depth complexity”: numarul 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-bufer) sunt reprezentate prin numere intregi: 16, 32 biti → pierdere precizie

Algoritmul BSP (Binary Space Partitioning)(1)

Poate fi folosit pentru:

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

Intrarea: lista poligoanelor care compun scena 3D

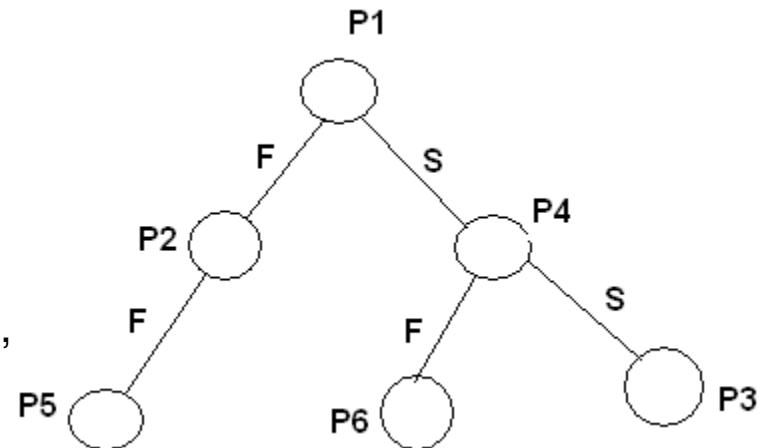
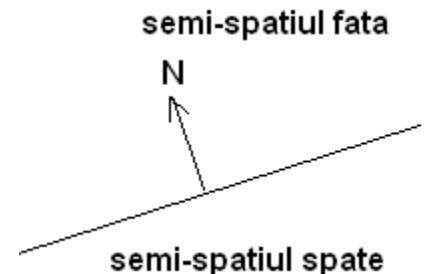
(nu are importanta din care obiect face parte fiecare poligon)

- Scena 3D este reprezentata printr-un arbore binar: arborele BSP
- Arborele BSP al scenei este independent de pozitia observatorului (reprezinta scena in sistemul coordonatelor globale).

Algoritmul BSP (2)

Construirea arborelui BSP al unei scene

- Fiecare nod al arborelui corespunde unui plan de partitionare a spatiului 3D (de regula, planul unui poligon al scenei)
- Fiecare plan de partitionare imparte spatiul in 2 semi-spatii:
 - cel din fata planului (de aceeasi parte cu normala la plan)
 - cel din spatele planului
- Se incepe cu un poligon oarecare din lista (de regula primul), pentru care se creaza nodul radacina al arborelui
- Poligoanele din semi-spatiul “față” formeaza “lista-față”, care va genera subarborele “față” al nodului
- Poligoanele din semi-spatiul “spate” formeaza “lista-spate”, care va genera subarborele “spate” al nodului
- Se alege un poligon din lista-față si se creaza nodul radacina al subarborelui “față, etc.



Algoritmul BSP (3)

Afisarea arborelui BSP al unei scene

- Tine cont de pozitia observatorului
- Afisare “back-to-front”: se incepe cu poligonul cel mai indepartat de observator
- Se porneste din radacina arborelui si se avanseaza in arbore pana la frunze, in functie de pozitia observatorului fata de planul atasat fiecarui nod
- Exemplu: se adauga observatorul in scena 3D a arborelui din figura precedenta

Eliminarea obiectelor nevizibile din banda grafica (Object culling):

- Daca planul de partitionare al unui nod nu intersecteaza volumul vizual, atunci numai subarborele aflat de aceeasi parte cu volumul vizual va fi afisat, celalalt subarbore fiind eliminat din banda grafica.

Algoritmul BSP (4)

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 BSP (5)

```
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);}
}
```

Algoritmul BSP (6)

Aprecieri:

- Arborele nu trebuie sa fie construit pentru fiecare cadru imagine: avantaj pentru scenele statice
- Generarea unui cadru = executia functiei de afisare
- Permite eliminarea din banda grafica a unui numar mare de poligoane care sunt in afara volumului vizual(Object culling)
- Diverse adaptari ale arborelui pentru cazul obiectelor in miscare in scena 3D
- Numarul de suprascrieri ale unui pixel imagine este mai redus decat in algoritmul Z-buffer

- Sistemele grafice actuale, OpenGL si Direct3D, nu construiesc arborele BSP al unei scene.
- Construirea si afisarea arborelui BSP sunt functii de management al scenei 3D, implementate de regula intr-un motor grafic 3D.

Algoritmul Warnock - bazat pe divizarea spatiului de afisare(1)

Intrare: lista poligoanelor care compun scena 3D, transformate in spatiul imagine

- **Față de o zona dreptunghiulară a spatiului de afisare, poligoanele din lista curentă pot fi în una dintre categoriile:**

1. Poligon acoperitor, care acopera intreaga zona
2. Poligon inclus in zona
3. Poligon disjunct cu zona
4. Poligon intersectat de zona: se divide in poligon inclus + poligon/poligoane disjunct/disjuncte



- **In zona curentă sunt vizibile:**

- Poligoanele incluse si unul dintre cele acoperitoare

- **Lista poligoanelor vizibile se sorteaza crescator dupa coordonata zmin a fiecarui poligon:** primul din lista este cel mai apropiat de observator → toate poligoanele care urmeaza in lista dupa unul acoperitor pot fi eliminate din lista.

- **Algoritmul este recursiv:** la un apel se transmit functiei de afisare: zona de afisare si lista de poligoane de afisat. Iesirea din recursitate:

- lista de poligoane este vida
- lista are un singur element, care este un poligon inclus in zona
- primul in lista este un poligon acoperitor al zonei
- zona are un singur pixel

Algoritmul Warnock(2)

Daca iesirea din recursivitate nu este posibila, se imparte zona curenta in 4 subzone egale si se apeleaza functia recursiv pentru fiecare zona, cu lista poligoanelor vizibile.

void Warnock(lista_poligoane L, RECT zona)

{ se formeaza lista V, a poligoanelor vizibile in zona, plecand de la L

daca V este vida return;

daca V are un singur element, P, atunci:

daca P este inclus in zona, atunci:

- se afiseaza zona in culoarea fondului

- se afiseaza poligonul in culoarea proprie

altfel (P este poligon acoperitor)

- se afiseaza zona in culoarea poligonului

return;

daca zona are un singur pixel (x,y), atunci:

- se calculeaza coordonata z in pixelul (x,y) pentru toate poligoanele din lista V;

- se afiseaza pixelul in culoarea poligonului cu zmin in (x,y);

return;

Algoritmul Warnock(3)

**se sorteaza lista V crescator dupa zmin poligoane (primul in lista este cel mai apropiat de observator);
daca primul in lista V este un poligon acoperitor, atunci:**

- se afiseaza zona in culoarea poligonului;
- return;**

se elimina din V toate poligoanele care urmeaza dupa primul poligon acoperitor;

```
Warnock( V, z1);
Warnock( V, z2);
Warnock( V, z3);
Warnock( V, z4);
return;
}
```

zona curenta	
z1	z2
z3	z4

Aprecieri:

- Principalul efort de calcul este clasificarea poligoanelor fata de zona curenta: intersectii poligoane – zona pentru calcul varfuri si informatii asociate varfurilor (normale, culori, coord textura)
- La fiecare apel recursiv, lista poligoanelor scade foarte mult (deci si efortul de clasificare)
- Algoritmul poate fi paralelizat cu usurinta.
- Nu sunt suprascrisi pixelii imaginii (ca in Z-buffer) – exceptand cazul poligon inclus inscris peste fond.

Algoritmul pictorului(1)

–Intrare: lista poligoanelor care alcătuiesc scena 3D, transformate în spațiul de afisare

Forma generală a algoritmului

1. Se calculează “extensia” fiecarui 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 fiecarui poligon: primul în lista va fi cel mai apropiat de observator
3. Se descompun poligoanele ale căror extenții pe axa OZ se suprapun, astfel încât extenziile lor pe axa OZ să fie disjuncte
4. Se afisează (transmit în banda grafică) poligoanele începând cu ultimul din lista

Algoritmul pictorului(2)

Pasul 3. Se descompun poligoanele ale caror extensiile pe axa OZ se suprapun

- Este necesar numai daca extensiile pe axa OZ se suprapun.
- Sunt multe aplicatii in care acest pas nu este necesar, poligoanele fiind amplasate in plane de Z-constant: cartografie, generarea straturilor circuitelor imprimante, etc.
- Poate fi optimizat, stiind ca nu intotdeauna atunci cand extensiile pe axa OZ se suprapun este necesara descompunerea poligoanelor; testele se efectueaza progresiv, in functie de complexitatea calculelor presupuse
- **Principalul efort de calcul:** sortarea listei de poligoane si descompunerea, daca este necesara.

Tehnici de eliminare a obiectelor nevizibile (1) *(Object culling/Visibility culling)*

- ❖ Scopul: de a impiedica rasterizarea obiectelor care sunt complet nevizibile
- 1) Eliminarea fețele auto-obturate
 - 2) Se dorește ca eliminarea să se efectueze la nivel de obiecte sau grupuri de obiecte → aceasta presupune reprezentarea scenei printr-o structură de date spatială.

Structuri de date spatiale pentru managementul scenei 3D

- Ierarhie de volume încadratoare
- Arbore BSP
- Arbore octal

Tehnici de eliminare a obiectelor nevizibile (2)

1. Scena- Ierarhie de volume incadratoare

- ❖ **Scena este reprezentata printr-un arbore avand in noduri **volume incadratoare** :** sfera, paralelipipedul aliniat cu axele, poliedrul convex minimal incadrator, cilindrul, elipsoidul.
 - Volumele incadratoare ale obiectelor sunt frunzele arborelui
 - Se grupeaza volumele incadratoare ale obiectelor, rezultand noduri ale arborelui, fiecare avand asociat volumul incadrator al grupului
 - Se continua gruparea pana ce se ajunge la nodul radacina, al carui volum incadrator incadreaza intreaga scena 3D
- ❖ **Eliminarea obiectelor nevizibile:**
 - teste de vizibilitate incep cu radacina arborelui
 - daca volumul incadrator al unui nod este complet in afara volumului vizual, atunci intregul grup de obiecte din subarborele nodului este exclus din banda de redare.

Tehnici de eliminare a obiectelor nevizibile (3)

2. Scena - Arbore BSP(1)

- ❖ **Planele de partitionare sunt planele poligoanelor**
- Arborele scenei se construieste ca in algoritmul BSP pentru eliminare suprafetelor nevizibile
- ❖ **Eliminarea obiectelor nevizibile:**

daca planul de divizare atasat unui nod nu intersecteaza volumul vizual, atunci volumul vizual este situat de o parte a planului iar subarborele atasat nodului care este situat de cealalta parte a planului este complet nevizibil si poate fi eliminat din procesul de redare.
- Eliminarea se poate face in spatiul observator, folosind volumul vizual definit de aplicatie (trunchi de piramida sau paralelipiped) sau in spatiul coordonatelor de decupare, inainte de decupare.

Tehnici de eliminare a obiectelor nevizibile (3)

3. Scena - Axis Aligned BSP Tree

- ❖ Planele de partitionare sunt aliniate cu axele sistemului de coordonate 3D -perpendiculare pe cele 3 axe: **Axis-Aligned BSP Tree (AA-BSP)**
- ❖ AA-BSP tree este un k-d tree (*k-dimensional tree*): arbore binar in care fiecare nod corespunde unei coordonate intr-un spatiu k-dimensional. Pentru $k=3$, un nod va corespunde uneia dintre coordonatele x, y sau z. De exemplu, daca planul de partitionare al unui nod este perpendicular pe OX, atunci nodul este asociat coordonatei x a planului.
- ❖ **Construirea arborelui:**

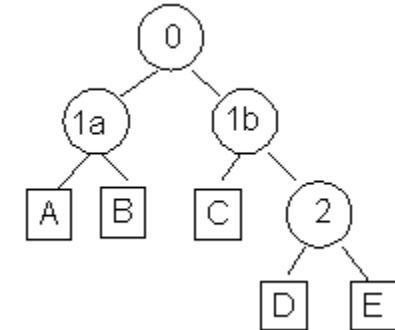
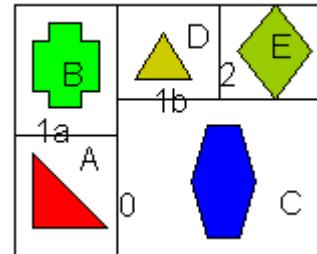
Scena este divizata recursiv prin plane paralele cu planele sistemului de coordonate, intr-o ordine fixa, tinand cont si de geometria scenei; de exemplu, in pasul 1 se alege un plan paralel cu XOY, in pasul 2 un plan paralel cu YOZ, in pasul 3 un plan paralel cu XOZ, in pasul 4 un plan paralel cu XOY, s.a.m.d.

- Se pleaca de la volumul incadrator al scenei (aliniat cu axele)
- In fiecare pas rezulta 2 volume incadratoare mai mici.
- Alegerea planelor se face astfel incat numarul de obiecte din fiecare semispatiu sa fie acelasi sau diferit cu 1, pentru ca arborele sa fie echilibrat
- Alegerea planelor se poate face astfel incat numarul de intersectii dintre planele de partitionare si obiectele scenei sa fie minimizat.

Tehnici de eliminare a obiectelor nevizibile (3)

3. Scena- Axis Aligned BSP Tree (cont)

Exemplu de divizare
cu plane aliniate cu axe, in 2D:



Pasul 1: divizare cu un plan vertical: rezulta sub-arborii 1a si 1b

Pasul 2: - subarborele 1a:

divizare cu un plan orizontal: rezulta frunzele A, B

seiese din recursivitate

- subarborele 1b:

divizare cu un plan orizontal: rezulta nodul frunza C si nodul 2

seiese din recursivitate

Pasul 3: - subarborele 2:

divizare cu un plan vertical: rezulta frunzele D, E

seiese din recursivitate

❖ **Eliminarea obiectelor nevizibile** : la fel ca in cazul ierarhiei de volume incadratoare

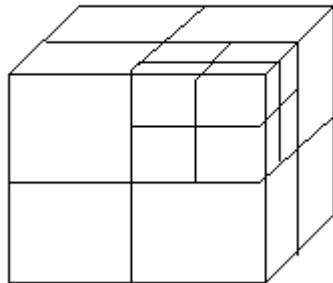
➤ daca volumul incadrator al unui nod este complet in afara volumului vizual, intregul grup de obiecte reprezentate prin subarborele nodului este exclus din banda grafica

Tehnici de eliminare a obiectelor nevizibile (3)

4. Scena - Arbore octal

❖ Intrarea: lista poligoanelor scenei

- In fiecare pas se divide volumul/subvolumul incadrator in 8 subvolume egale.



- Fiecare nod al arborelui are 8 copii.
- Subdivizarea spatiului are loc pana cand numarul de poligoane din fiecare nod frunza este mai mic decat un prag dat.
- Poligoanele sunt memorate in nodurile frunza; un poligon poate fi asociat mai multor noduri, care corespund unor subcuburi adiacente.

❖ Arborele octal se poate folosi pentru reprezentarea ierarhica a scenei 3D: fiecare obiect este asociat cu cel mai mic subcub al arborelui octal, care-l incadreaza complet.

❖ Eliminarea obiectelor nevizibile din banda grafica se efectueaza intersectand subcuburile arborelui cu volumul vizual:

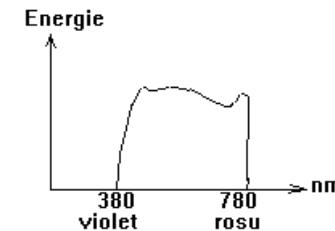
- se elimina toate obiectele din subcuburile care sunt in afara volumului.
- testele incep de la radacina arborelui: daca subcubul unui nod este in afara volumului vizual se elimina toate obiectele din subarborele nodului.

Redarea luminii în imagini

Prof. univ. dr. ing. Florica Moldoveanu

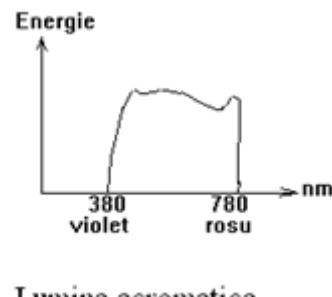
Proprietățile luminii(1)

- Lumina este energie electromagnetică (spectrul vizibil) →

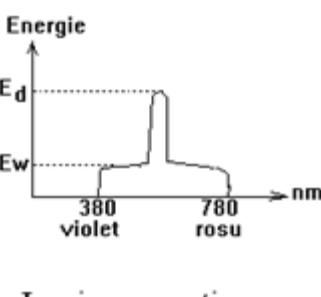


- Atunci când cade pe suprafața unui obiect, ea poate fi: absorbită, reflectată sau transmisă

Sistemul vizual uman percepă: lumina provenită direct de la o sursă și
lumina reflectată sau transmisa de obiectele din mediul.



Lumina acromatică



Lumina cromatică

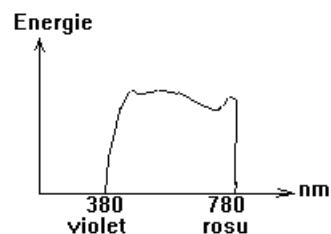
- Lumina care conține toate lungimile de undă din spectrul vizibil în proporții aproximativ egale se numește **acromatică**.
- Lumina în care predomină anumite lungimi de undă se numește **cromatică**.

Proprietățile luminii(2)

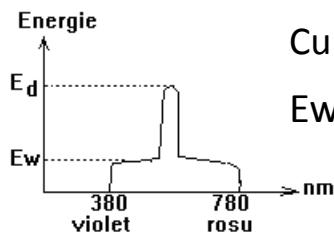
- Culoarea unui obiect, perceputa de ochiul uman, depinde atât de distribuția lungimilor de undă în lumina care cade pe obiect cât și de caracteristicile fizice ale suprafetei obiectului: reflectă/absoarbe anumite lungimi de undă.
 - Energia electromagnetică nu are culoare!
 - Culoarea este rezultatul unui proces psihofiziologic.
- **Definiția psihofiziologică a unei culori cuprinde:** **nuanta, luminozitatea (stralucirea), saturatia**
 - Nuanța:** roșu, galben, verde, etc. -determinată de lungimea de undă dominantă a distribuției spectrale a luminii.
 - Luminozitatea sau strălucirea** - reprezintă intensitatea luminii.
 - Luminozitatea: caracteristică a unui obiect emițător de lumină
 - Strălucirea: caracterizează un obiect neemițător, care reflectă lumina.

Proprietățile luminii(3)

- **Saturația sau puritatea** - o măsură a amestecului de alb într-o culoare pură;
 - permite să se facă distincție între roșu și roz, între albastru și bleu, etc.
 - O culoare pură are saturăția 100%.
 - Lumina acromatică are saturăția 0%.



Lumina acromatică



Lumina cromatică

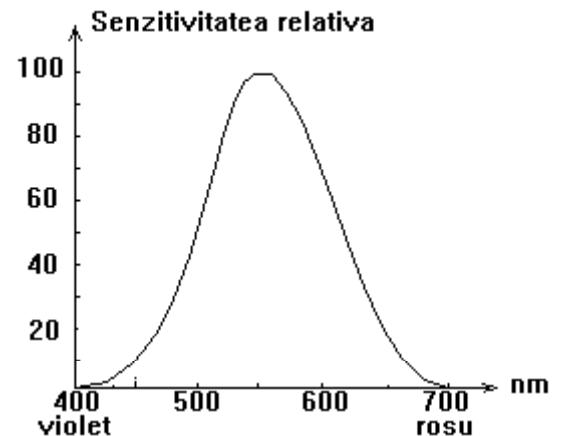
Culoarea este data de lungimea de undă dominantă.

$E_w=0$ – culoarea are puritatea 100% : monocromatica
(lungime de undă de un nanometru)

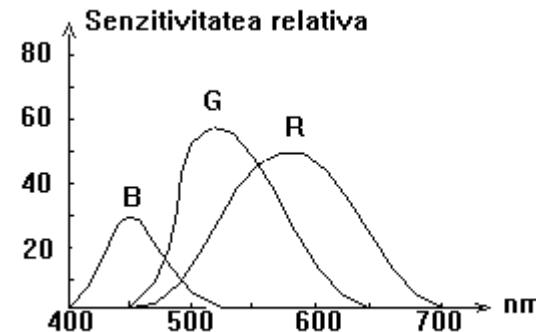
- Cea mai cunoscută dintre teoriile privind formarea culorilor în sistemul ochi-creier este aceea conform căreia în retina ochiului uman există trei tipuri de conuri, fiecare fiind sensibil la una dintre culorile roșu, verde și albastru.

Proprietățile luminii(4)

- Culorile percepute de ochi sunt amestecuri de culori pure.



Curba de luminozitate
Suma celor 3 curbe de sensibilitate



Raspunsul ochiului la cele 3 culori primare

Senzitivitatea maximă corespunde luminii cu lungimea de undă în jur de 550 nm, percepută ca galben-verde.

- Culorile roșu (Red), verde(Green) și albastru (Blue) se numesc **culori primare**.
 - Prin amestecul lor în proporții egale se obține alb.
- Două culori care prin amestec produc lumină albă se numesc **complementare**.
 - Culorile complementare culorilor primare sunt : cian (Cyan), magenta (Magenta), galben (Yellow).

Proprietățile luminii(5)

- Amestecând două culori primare în proporții egale se obține culoarea complementară celei de a treia:

R	G	B
C	M	Y

- De exemplu, albastru+verde=cian, roșu+verde=galben, roșu+albastru= magenta.
- Culorile **roșu, verde și albastru** se numesc și **“primitive aditive”** deoarece ele permit formarea de nuanțe prin **adunarea** lor în diferite proporții.
- Acest mod de definire a culorilor corespunde echipamentelor emițătoare de lumină (dispozitive de vizualizare cu ecran).

Proprietatile luminii(6)

- Culoarele **cian, magenta, galben** se numesc "primitive extractive".
 - Se obtin prin **extragerea** culorilor primare din lumina alba:

- **Se folosesc pentru a defini culorile reflectate de un document imprimat:** pigmentii existenți în cernelurile tipografice absorb culorile complementare acelora ale pigmentilor.

Ex: pigmentul de culoare magenta absoarbe din lumina incidentă componentele corespunzătoare luminii verde, iar cel de culoare galben, componentele corespunzătoare luminii albastre.

➤ O suprafață care conține pigmenti magenta și galben va reflecta (sau transmite) lumină roșie.

- C, M, Y permit specificarea de nuanțe prin extragerea lor în diferite proporții din alb. Scăzându-le în proporții egale din alb se obțin diferite nuanțe de gri.

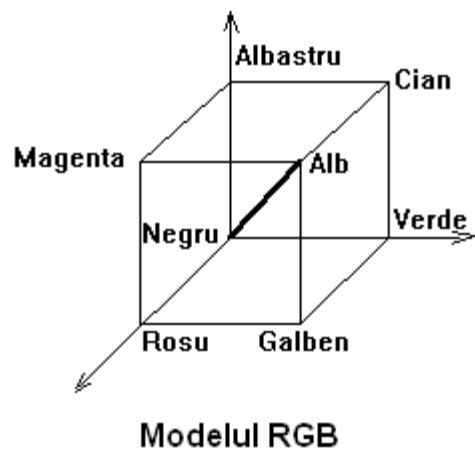
Modele de culoare(1)

- Orientate către echipamente
 - se bazează pe culorile primare folosite de echipamente pentru redarea culorilor:
 - RGB, CMY și YIQ.
- Orientate către utilizator
 - se bazează pe proprietățile psihofiziologice ale culorilor :
 - HSV și HLS.

Modele de culoare(2)

- Un model de culoare specifică un sistem de coordonate 3D și un subspațiu al culorilor în sistemul de coordonate respectiv.
- Fiecare culoare se reprezintă printr-un punct în subspațiul culorilor.

Modelul RGB (Red, Green, Blue)



Rosu: 1,0,0

Verde: 0,1,0

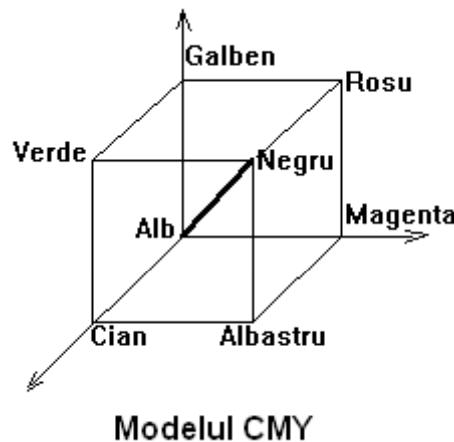
Albastru: 0,0,1

Alb: 1,1,1; Negru: 0,0,0

Cian: 0, 1, 1 - culoarea complementara culorii Rosu !

Modele de culoare(3)

Modelul CMY (Cyan, Magenta, Yellow)



Cian: 1,0,0

Magenta: 0, 1, 0

Galben: 0,0,1

Alb: 0,0,0

Negru: 1,1,1

Rosu: 0, 1, 1 - Complementara culorii Cian!

Coversia RGB \leftrightarrow CMY

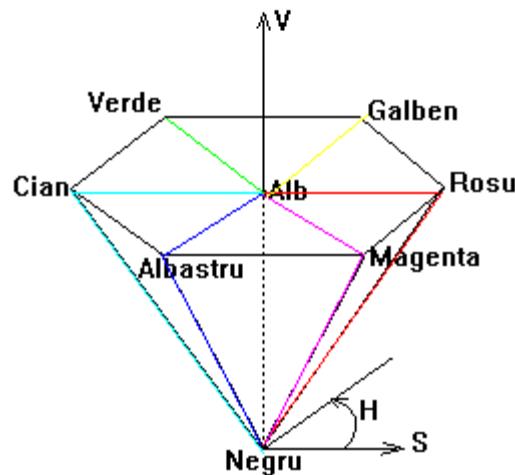
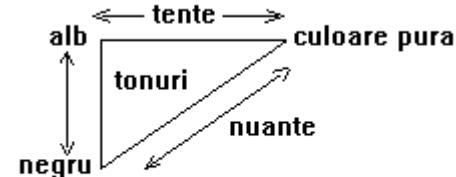
$$[C\ M\ Y] = [1\ 1\ 1] - [R\ G\ B]$$

$$[R\ G\ B] = [1\ 1\ 1] - [C\ M\ Y]$$

Modele de culoare(4)

Modelul HSV (Hue, Saturation, Value)

- Model de culoare orientat catre utilizator
- Artiștii specifică culorile prin **tente**, **nuanțe** și **tonuri**



$$0 \leq V \leq 1, \quad 0 \leq S \leq 1, \quad 0 \leq H \leq 360$$

$V=1$ – baza hexaconului

$S=0$ - pe axa hexaconului

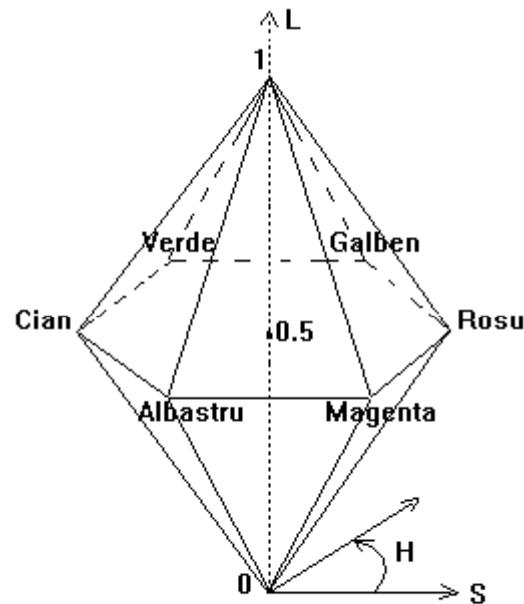
$H=0$ - culoarea Rosu

Culorile complementare – la 180 grade pe baza hexaconului

- Hexaconul din figura reprezinta subspatiul culorilor in modelul HSV

Modele de culoare(5)

Modelul HLS (Hue, Lightness, Saturation)



Modelul HLS

Culorile primare cu saturatie maximă și complementarele lor sunt reprezentate prin $S=1$, $L=0.5$.

Modele de culoare(6)

Interpolarea în spațiul culorilor

- Rezultatul interpolării între două culori depinde de modelul de culoare în care sunt specificate.
- Fie două culori specificate în modelul RGB, $C1=(1,0,0)$ și $C2=(0,1,0)$. Le interpolăm cu ponderi egale în modelele RGB și HSV:
 - Interpolare în modelul RGB:
 $C=t*C2 + (1-t)*C1$ unde $t=0.5$, se obține
 $C=(0.5, 0.5, 0)$, care convertită în HSV ne dă **(60, 1, 0.5)**
 - Interpolare în modelul HSV:
 $C1$ se reprezintă în HSV prin $(0, 1, 1)$ iar
 $C2$ prin $(120, 1, 1)$
 $C= 0.5*(0,1,1) + 0.5*(120,1,1) = \textcolor{red}{(60, 1, 1)}$

Modele empirice pentru calculul reflexiei luminii(1)

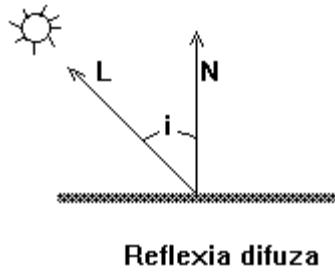
- Caracteristicile luminii reflectate de suprafața unui obiect depind de :
 - lungimile de undă conținute în lumina incidentă,
 - direcția și geometria sursei luminoase,
 - orientarea suprafetei
 - proprietățile materialului din care este construită suprafața.
- Expresia care modeleză intensitatea luminii reflectate într-un punct al unei suprafete este definită pentru o suprafață nicolorată și o **lumină incidentă monocromatică**, deci caracterizată printr-o anumită lungime de undă, λ .
- În cazul general, **lumina reflectată nu este monocromatică**, de aceea pentru calculul său expresia ar trebui să fie evaluată continuu pe întregul domeniu al spectrului de modelat.
- În practică, expresia se evaluatează separat pentru cele trei culori de bază, R, G, B.

Modele empirice pentru calculul reflexiei luminii(2)

Aproximarea reflexiei difuze intr-un punct al unei suprafete 3D

- Lumina reflectată difuz de o suprafață este dispersată regulat în toate direcțiile.
- **Legea lui Lambert** definește reflexia luminii provenite de la o sursă punctiformă, de către un difuzor perfect:

$$Id = I_{\text{sursa}} * kd * \cos(i) \quad 0 \leq i \leq \pi/2$$



I_{sursa} – este intensitatea luminii incidente (provenita de la sursa de lumina)

kd – este coeficientul de difuzie a luminii incidente, dependent de materialul suprafetei

$$0 \leq kd \leq 1.$$

- Dacă i este mai mare ca $\pi/2$, suprafața nu primește lumină de la sursă (sursa de lumină se află în spatele suprafetei).

Modele empirice pentru calculul reflexiei luminii(3)

Lumina ambianta

- Modeleaza lumina provenind de la celelelte obiecte ale scenei 3D: sursa de lumina distribuita uniform in spatiu

$$Id = Ia * ka + Isursa * kd * \cos(i)$$

$$0 \leq i \leq \pi/2$$

unde

Ia – este intensitatea luminii ambiante iar

($0 \leq ka \leq 1$) este coeficientul de difuzie a luminii ambiante, dependent de materialul suprafetei.

Sursa directională: sursa aflata la distanta foarte mare (infinit) de suprafata

- Vectorul L este acelasi in orice punct al suprafetei
- Pentru 2 suprafete paralele, cu aceleasi proprietati de material, rezulta aceeasi Id → daca proiectiile lor se suprapun, nu se vor distinge in imagine.

Modele empirice pentru calculul reflexiei luminii(4)

Modelarea distantei de la sursa de lumina la suprafata

Intensitatea luminii descrește proporțional cu inversul pătratului distanței de la sursa de lumină la obiect

$$I_d = I_a * k_a + f_{at} * I_s * k_d * \cos(i)$$

f_{at} = 1/d² este o funcție de atenuare a luminii provenita de la o sursă

d este distanța de la sursă la punctul de pe suprafață considerat.

- Corecția nu satisfac cazurile în care sursa este foarte îndepărtată: atenuare prea mare
- Dacă sursa este la distanță foarte mică de scenă, intensitățile obținute pentru două suprafete cu același unghi i, între L și N, vor fi mult diferite.

Se foloseste:

$$f_{at} = \min\left(\frac{1}{c_1 + c_2 * d + c_3 * d^2}, 1\right)$$

c₁, c₂ și c₃ sunt trei constante care se asociază sursei de lumină.

- Constanta c₁ se alege astfel încât numitorul să nu devină prea mic atunci când sursa este foarte apropiată.
- Valoarea funcției este limitată la 1 pentru a se asigura atenuarea.

Modele empirice pentru calculul reflexiei luminii(5)

- Lumina incidenta poate contine mai multe lungimi de unda: → reflectate in mod diferit de o suprafata
- Suprafata poate fi colorata → absorbtia, transmisia, reflexia depind de culoarea suprafetei.

➤ Expresia lumintii reflectate se evalueaza separat pentru cele 3 componente ale lumintii incidente, R,G,B:

$$Id\lambda = Ia\lambda * ka\lambda + fat*Iursa\lambda * kd\lambda * \cos(i)$$

λ: lungimea de unda

- Se calculeaza IdR, IdG, IdB. Ex:

$$IdR = IaR * kaR + fat*IursaR * kdR * \cos(i)$$

kaR, kdR – coeficientii de difuzie a componenteii Rosu din lumina ambianta si lumina incidenta

- In OpenGL, ka (kaR, kaG, kaB) si kd sunt numite: **culoarea ambianta si culoarea difusa a materialului.**

Stiind că

$$\cos(i) = L \cdot N / (|L| \cdot |N|) = Lu \cdot Nu$$

rezulta: $I\lambda = Ia\lambda * ka\lambda + fat * Iursa\lambda * kd\lambda * (Lu \cdot Nu)$

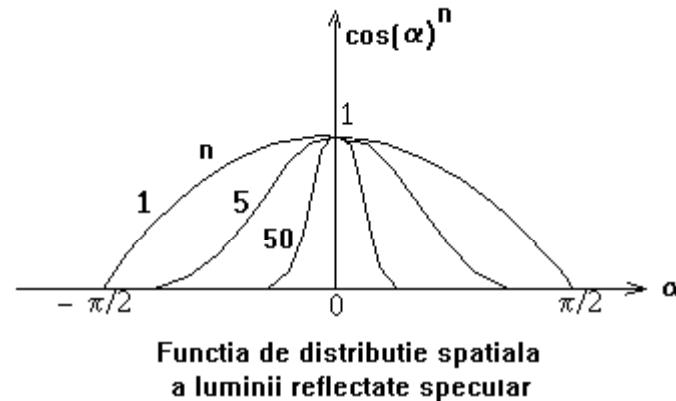
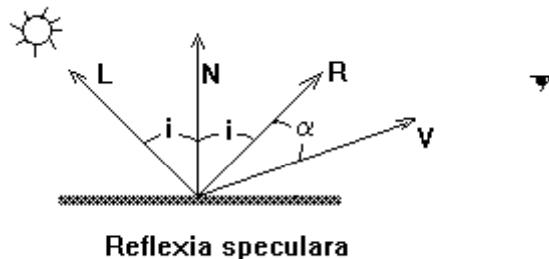
Pentru a include si cazul in care $i > \pi/2$ (lumina de la sursa nu ajunge in punctul considerat):

$$I\lambda = Ia\lambda * ka\lambda + fat * Iursa\lambda * kd\lambda * \max((Lu \cdot Nu), 0)$$

Modele empirice pentru calculul reflexiei luminii(6)

Reflexia speculară

- Un **reflector perfect**, de exemplu o oglindă, reflectă lumina numai într-o singură direcție, R, care este simetrică cu L față de normală la suprafață → numai un observator situat exact pe direcția respectivă va percep raza reflectată:



- Pentru **materialele imperfect reflectante** cantitatea de lumină care ajunge la observator depinde de **distribuția spațială a luminii reflectate specular**:
 - la suprafețele netede (ex. metale) distribuția este dreaptă și focalizată;
 - la suprafețele cu rugozități (ex. hartia) ea este dispersată.
 - se aproximează prin $\cos(\alpha)^n$ (modelul Bui-Tuong Phong) unde **n este exponentul de reflexie speculară** al materialului.

Modele empirice pentru calculul reflexiei luminii(7)

Modelul Phong pentru aproximarea reflexiei speculare într-un punct al unei suprafete 3D

$$Is\lambda = Isursa\lambda * w(i,\lambda) * \cos(\alpha)^n$$

$w(i, \lambda)$ este funcția de reflectanță, i - unghiul de incidentă iar λ -lungimea de undă a luminii incidente

În practică, $w(i, \lambda)$ este înlocuită cu o constantă determinată experimental, numită **coeficientul de reflexie speculară al materialului**.

$$\cos(\alpha) = R \cdot V / (|R| \cdot |V|) = R_u \cdot V_u$$

Rezulta:

$$Is\lambda = Isursa\lambda * fat * ks\lambda * (R_u \cdot V_u)^n$$

Pentru a include și cazul în care $\alpha = 90^\circ$:

$$Is\lambda = Isursa\lambda * fat * ks\lambda * \max((R_u \cdot V_u)^n, 0)$$

Modelul de iluminare locală(1)

Reflexia speculară nu poate avea loc dacă în punctul considerat nu se primește lumina de la sursa:

$$I_{s\lambda} = lum * I_{sursa\lambda} * fat * ks\lambda * \max((R_u \cdot V_u)^n, 0)$$

$$lum = 1 \text{ dacă } (L_u \cdot N_u) > 0$$

$$= 0 \text{ altfel}$$

Modelul de iluminare locală:

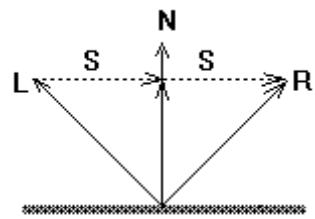
$$I_\lambda = I_{a\lambda} * k_a \lambda + fat * I_{sursa\lambda} [k_d \lambda * \max((L_u \cdot N_u), 0) + lum * k_s \lambda * \max((R_u \cdot V_u)^n, 0)]$$

Dacă scena 3D este luminată de m surse de lumină:

$$I_\lambda = I_{a\lambda} * k_a \lambda + \sum_{1 \leq i \leq m} fat_i * I_{sursa\lambda_i} * [k_d \lambda * \max((L_{u_i} \cdot N_u), 0) + lum_i * k_s \lambda * \max((R_{u_i} \cdot V_u)^n, 0)]$$

Modelul de iluminare locală(2)

Calculul directiei luminii speculare



- Vectorul R este simetricul vectorului L față de N .
- Proiecția lui L_u pe N este: $N_u \cdot \cos(i)$

$$R = N_u \cdot \cos(i) + S, \quad N_u \cdot \cos(i) = L_u + S \quad \rightarrow \quad S = N_u \cdot \cos(i) - L_u$$

Rezulta:

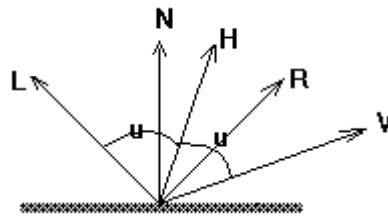
$$R = 2N_u \cdot \cos(i) - L_u = 2N_u \cdot (L_u \cdot N_u) - L_u$$

Modelul de iluminare locală(3)

O altă formulare a modelului Phong se bazează pe **vectorul median**, notat cu **H** în figura.

El face unghiuri egale cu **L** și cu **V**.

Dacă suprafața ar fi orientată astfel încât normala să aibă direcția lui **H**, atunci observatorul ar percepe lumina speculară maximă (deoarece ar fi pe direcția razei reflectate specular).



Termenul care exprimă reflexia speculară este în acest caz:

$$Is\lambda = Isursa\lambda * fat * ks\lambda * (Nu \cdot Hu)^n, \quad \text{unde } Hu = (Lu + Vu) / |(Lu + Vu)|$$

Atunci când sursa de lumină și observatorul sunt la infinit, utilizarea termenului $Nu \cdot Hu$ este avantajoasă deoarece Hu este constant (același în toate punctele unei suprafete).

Modelul de iluminare locală în OpenGL și Direct3D(1)

Modelul de iluminare de baza

Culoarea unui obiect într-un punct al unei suprafețe

$$= \text{culoarea_emisa} + \text{culoarea_ambianta} + \text{culoarea_difuză} + \text{culoarea_speculară}$$

Culoarea emisa :

- lumina emisa de suprafata
- independenta de sursele de lumina
- aceeasi in orice punct al suprafetei
- nu lumineaza obiectele din jur
- culoarea_emisa = Ke, constanta

Culoarea ambiantă

- nu depinde de surse
- culoarea_ambianta = Ka*globalAmbient (Culoarea luminii ambiante: GL_AMBIENT)

Modelul de iluminare locală în OpenGL și Direct3D(2)

Culoarea difuză

$$\text{culoarea_difuză} = \text{kd} * \text{lightColor} * \max((\text{Lu} \cdot \text{Nu}), 0)$$

- lightColor – culoarea luminii de la sursă

Culoarea speculară

$$\text{culoarea_speculară} = \text{ks} * \text{lightColor} * \text{facing} * \max((\text{Nu} \cdot \text{Hu})^n, 0)$$

$$\text{facing} = 1 \text{ daca } \text{Lu} \cdot \text{Nu} > 0$$

$$= 0 \text{ altfel}$$

- Ka , kd , ks sunt constante (proprietăți de material) care pot fi alese de programator:
`GL_AMBIENT`, `GL_DIFFUSE`, `GL_SPECULAR` – fiecare are 3-4 componente, deoarece ecuația care modelează lumina reflectată se evaluează separat pentru fiecare componentă a culorii (RGB sau RGBA).
- Lumina incidentă este omnidirectională: aceeași intensitate în toate direcțiile.

Modelul de iluminare locală în OpenGL și Direct3D(3)

Modelul de iluminare extins adauga:

- atenuarea luminii incidente cu distanța
- efecte de spot

Atenuarea luminii incidente

$$\text{factor_atenuare} = 1/(kc + kl*d + kq*d^2)$$

coeficientii de atenuare: kc – atenuare constantă

kl - atenuare liniară

kd – atenuare patratice

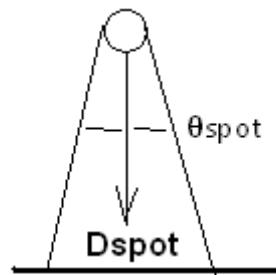
Kc, kl, kd pot fi specificate de programator ca proprietăți asociate sursei de lumina
(GL_CONSTANT_ATTENUATION, GL_LINEAR_ATTENUATION, GL_QUADRATIC_ATTENUATION)

Culoarea unui obiect într-un punct al unei suprafete

$$\begin{aligned} &= \text{culoarea_emisa} + \text{culoarea_ambianta} + \\ &\quad \text{factor_atenuare} * (\text{culoarea_difuză} + \text{culoarea_speculară}) \end{aligned}$$

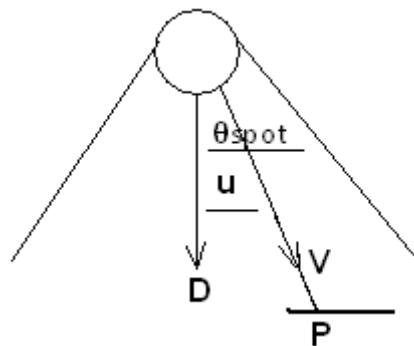
Modelul de iluminare locală în OpenGL și Direct3D(4)

Efectul de spot



Conul de imprastiere a luminii se calculeaza folosind:

- Pozitia spotului
- Directia spotului
- Imprastierea luminii spotului (θ_{spot})



Un punct P al unei suprafete primește lumina de la spot daca $u < \theta_{spot}$:

$$\cos(u) \geq \cos(\theta_{spot})$$

sau

$$V_u \cdot D_u > \cos(\theta_{spot})$$

Calculul culorii fragmentelor la redarea suprafețelor 3D (Shading models) (1)

Suprafața se presupune a fi compusă din fațete poligonale. Ea poate fi reprezentarea exactă a unui corp poliedral sau reprezentarea aproximativă a unei suprafețe curbe.

Modelul LAMBERT(1) (Iluminare constantă sau plată)

- Pentru fiecare poligon(fațetă) se calculează o singură intensitate, folosind expresia care modelează intensitatea într-un punct, de regulă considerând numai componenta difuză:

$$I_{d\lambda} = I_{a\lambda} * k_{a\lambda} + f_{at} * I_{surse\lambda} * [k_{d\lambda} * \max((N_u \cdot L_u), 0) + f_{acing} * \max((N_u \cdot H_u)^n, 0)]$$

N_u este vectorul unitate normal la poligon

L_u este versorul direcției sursei de lumină,

Calculul culorii fragmentelor la redarea suprafetelor 3D (2)

Modelul LAMBERT(2)

Modelul se bazează pe următoarele presupuneri:

- sursa de lumină este la infinit (produsul scalar $(\mathbf{N}_u \cdot \mathbf{L}_u)$ este atunci constant pe întreaga suprafață a poligonului);
- observatorul este la infinit ($(\mathbf{N}_u \cdot \mathbf{H}_u)$ este constant pe suprafața poligonului);
- poligonul face parte din suprafața de vizualizat și nu este o aproximare a unui petic de suprafață curbă.
- Dacă primele două cerințe nu sunt satisfăcute, se poate adopta o convenție de calcul al vectorilor \mathbf{L} și \mathbf{V} pentru un întreg poligon. De exemplu, cei doi vectori pot fi calculați în centrul poligonului.

Calculul culorii fragmentelor la redarea suprafetelor 3D (3)

Modelul LAMBERT (3)

- Dacă ultima cerință nu este îndeplinită, intensitățile calculate pentru fațete adiacente cu orientare diferită vor fi diferite, evidențiindu-se aproximarea suprafetei curbe prin rețea de fațete poligonale.
- Percepția diferenței de intensitate dintre fațetele adiacente este accentuată de efectul de bandă Mach (descoperit de Mach în 1865), cauzat de inhibiția laterală a receptorilor din ochi: cu cât un receptor primește mai multă lumină cu atât mai mult receptorul va inhiba răspunsul receptorilor adiacenți lui.
- Efectul de bandă Mach mărește percepția schimbării de intensitate pe laturile fațetelor adiacente.

Calculul culorii fragmentelor la redarea suprafetelor 3D (4)

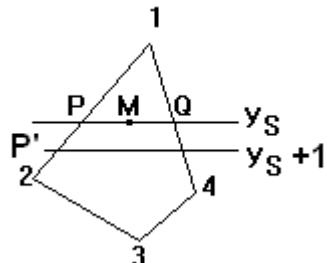
Modelul GOURAUD(1)

- Se calculează o culoare în fiecare vârf al suprafetei de vizualizat pe baza unui model de iluminare local.
 - Culorile fragmentelor interioare suprafetei sunt obținute prin interpolarea liniară a culorilor din vârfuri, pe parcursul rasterizării fiecarei fatete a retelei poligonale.
1. **Culoarea intr-un varf se calculeaza folosind normala in varf:**
 - calculata din ecuația analitică a suprafetei care a fost descompusă în rețea poligonală și atașată fiecarui varf în programul de aplicatie, sau
 - aproximată ca medie aritmetică a normalelor la fatetele adiacente în varf
 2. **La rasterizarea unei fatete a retelei, se calculeaza culoarea fiecarui fragment rezultat astfel:**
 - prin interpolarea liniară a culorilor varfurilor, pentru fragmentele de pe laturi
 - prin interpolare liniară între culorile capetelor fiecarui segment interior, pentru fragmentele interioare fatetei

Calculul culorii fragmentelor la redarea suprafetelor 3D (5)

Modelul GOURAUD(2)

Interpolarea liniara



$$y = y_1 + t(y_2 - y_1) \rightarrow tP = (y_S - y_1)/(y_2 - y_1)$$

$$IP = I_1 + tP(I_2 - I_1) = I_1 + (I_2 - I_1)(y_S - y_1) / (y_2 - y_1)$$

$$IQ = I_1 + (I_4 - I_1)(y_S - y_1) / (y_4 - y_1)$$

$$x = xP + t(xQ - xP) \rightarrow tM = (x_M - x_P) / (x_Q - x_P)$$

$$IM = IQ + tM(IQ - IP) = IQ + (IQ - IP)(x_M - x_P) / (x_Q - x_P)$$

Calculul incremental al culorilor:

$$IP' = I_1 + (I_2 - I_1)(y_{S+1} - y_1) / (y_2 - y_1) = IP + (I_2 - I_1) / (y_2 - y_1)$$

IP' = IP + C₁₋₂, C₁₋₂ – o constantă a laturii 1-2

analog,

$$IQ' = IQ + C_{1-4}$$

$$M' (x_M + 1, y_S)$$

$$IM' = IQ + (IQ - IP)(x_M + 1 - x_P) \rightarrow IM' = IM + C_{P-Q}, C_{P-Q} – o constantă a segmentului P-Q$$

Calculul culorii fragmentelor la redarea suprafetelor 3D (6)

Modelul GOURAUD (3) – aprecieri

- Se integreaza foarte usor in algoritmii de rasterizare
- Calcule simple pentru culoarea unui fragment (calcul incremental) → rapid
- Se lucreaza numai cu componenta difusa a luminii reflectate
- Nu permite calculul luminii reflectate speculare pentru fragmentele interioare unei fatete (culorile fragmentelor interioare nu pot fi mai mari decat cele din varfuri)
- Nu elibera complet efectul de banda Mach:
 - Asigura continuitatea numerica a valorilor intensitatilor la traversarea laturilor poligoanelor adiacente (culorile pe latura de adiacenta a 2 poligoane sunt aceleasi)
 - Nu tine cont de tangentele la fetele adiacente pe o latura (culorile de pe laturi se calculeaza folosind numai culorile din varfuri)
 - Efectul de banda Mach poate fi observat in vecinatatea siluetei suprafetei si a zonelor de curbura mare

Calculul culorii fragmentelor la redarea suprafetelor 3D (7)

Modelul PHONG(1)

In acest model:

- Se calculeaza o normala in fiecare varf al suprafetei, la fel ca in modelul Gouraud
- Pentru fiecare fragment rezultat din rasterizarea (fatetelor) suprafetei se calculeaza o normala prin interpolare liniara intre normalele varfurilor
- Culoarea pentru fiecare fragment interior suprafetei se obtine pe baza normalei interpolate, folosind un model de iluminare local

La rasterizarea unei fatete a retelei poligonale se calculeaza o normala pentru fiecare fragment astfel:

- prin interpolarea liniara a normalelor varfurilor, pentru fragmentele de pe laturi (la fel ca in modelul Gouraud, pentru culori);
- prin interpolare liniara intre normalele capetelor fiecarui segment interior, pentru fragmentele interioare fatetei (la fel ca in modelul Gouraud, pentru culori)

Calculul culorii fragmentelor la redarea suprafetelor 3D (8)

Modelul PHONG (2)

- Componentele Nx, Ny, Nz ale normalei unui fragment se pot obține printr-un calcul incremental (analog cu cel folosit pentru calculul culorilor în modelul Gouraud) dar, pentru folosirea în calculul culorii, normala trebuie să fie normalizată:

$$N_u = N / |N|, \text{ unde } |N| = (Nx^2 + Ny^2 + Nz^2)^{0.5}$$

→ Calcule mai complexe la nivel de fragment decât în modelul Gouraud

- Permite redarea reflexiei speculare în orice fragment al suprafetei
 - Reduce mult efectul de banda Mach
- Modelul Gouraud este implementat pe placa grafică și poate fi selectat din OpenGL apelând:
`glShadeModel(GL_SMOOTH)`
- Modelul Phong poate fi implementat într-un program “fragment shader” scris de programator.

*TRANSPARENȚĂ, CEATĂ SI UMBRE
IN MODELELE DE ILUMINARE
LOCALĂ*

Prof. univ. dr. ing. Florica Moldoveanu

MODELAREA TRANSPARENȚEI (1)

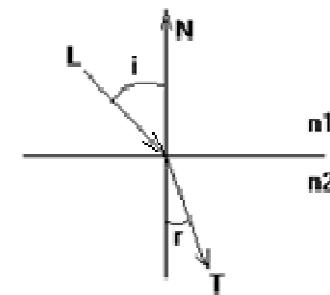
Unele obiecte ale scenei sintetizate pot fi construite din materiale transparente sau translucide.

Transmisia luminii prin obiectele transparente este speculară, în timp ce prin cele translucide este difuză.

- Atunci când lumina trece dintr-un mediu într-altul (de exemplu, din aer în apă), direcția sa se modifică datorită refracției.
- Relația dintre unghiul razei incidente, i , și cel al razei refractate, r , este dată de legea lui Snell:

$$\sin(i)/\sin(r) = n_1/n_2$$

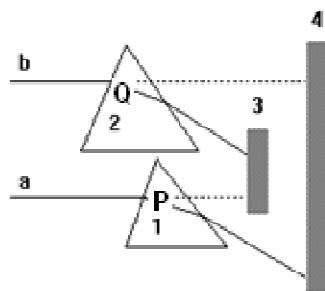
n_1 și n_2 sunt indicii de refracție ai celor două medii (materiale) traversate de lumină.



MODELAREA TRANSPARENTEI (2)

Indicele de refracție al unui material este dependent de lungimea de undă a luminii incidente și chiar de temperatură, dar în modelele de iluminare el este considerat constant.

Culoarea vizibila intr-un punct al unei suprafete transparente provine de la obiectul aflat pe directia razei transmise.



3 si 4 sunt poligoane opace

1 si 2 sunt poligoane transparente, aflate in fata poligoanelor 3 si 4
a si b sunt raze de lumina provenind de la surse, incidente in P si Q

Tinand cont de refracție:

- Culoarea in P este data de poligonul 4
- Culoarea in Q este data de poligonul 3

Neglijand refractia

- Culoarea in P este data de poligonul 3
- Culoarea in Q este data de poligonul 4

MODELAREA TRANSPARENȚEI (3)

- Refracția produce, de asemenea, o distorsionare a obiectelor, asemănătoare cu aceea produsă de o proiecție perspectivă: → dacă se dorește obținerea de imagini realiste, trebuie să se țină cont de refracție.
- Multe metode practice de modelare a transparenței ignoră refracția, astfel încât obiectele vizibile printr-o suprafață transparentă sunt cele aflate pe direcția razei incidente.

Motivul ignorării:

- reducerea volumului de calcule;
- obținerea realismului fotografic în totalitate (fără deformare)

Atunci când suprafața vizibilă într-un pixel este transparentă, culoarea în care va fi afișat pixelul se poate obține combinând culoarea suprafeței vizibile cu aceea a suprafeței aflată imediat în spatele său, folosind următoarea formulă de interpolare:

MODELAREA TRANSPARENȚEI (4)

$$I_{\lambda} = (1-k_{t_1}) * I_{\lambda_1} + k_{t_1} * I_{\lambda_2}$$

Coeficientul de transmisie, k_{t_1} măsoară transparenta suprafetei vizibile în pixel,

$$0 \leq k_{t_1} \leq 1.$$

$k_{t_1} = 0 \rightarrow$ suprafața vizibilă este opacă și deci pixelul va fi afișat în culoarea sa, I_{λ_1} ;

$k_{t_1} = 1 \rightarrow$ suprafața vizibilă este perfect transparentă și nu contribuie la culoarea pixelului.

Pixelul va fi afișat în culoarea suprafetei din spate, I_{λ_2}

Dacă $k_{t_1}=1$ și suprafața din spatele celei vizibile este la rândul său transparentă, metoda de calcul se aplică recursiv, până când se întâlnește o suprafață opacă sau fondul.

Aproximarea liniară din model nu dă rezultate bune pentru suprafetele curbe: în apropierea laturilor siluetei unei suprafete curbe (de exemplu, o vasă sau o sticlă) grosimea materialului reduce transparenta.

Solutia propusa de Kay: k_t se calculeaza in functie de normala la suprafata in punctul considerat.

MODELAREA TRANSPARENȚEI (5)

$$k_t = k_{t_{\min}} + (k_{t_{\max}} - k_{t_{\min}})(1 - (1 - N_z)^m)$$

unde

$k_{t_{\min}}$ și $k_{t_{\max}}$ reprezintă transparenta minimă și cea maximă a suprafeței,

N_z este componenta z a normalei normalizate la suprafață în punctul pentru care se calculează k_t , iar

m este un exponent ce caracterizează transparenta. Valorile uzuale pentru m sunt 2 și 3.

- **Majoritatea algoritmilor de determinare a vizibilității suprafetelor la afișarea scenelor 3D pot fi adaptați pentru a îngloba transparenta.**
- În algoritmii care afișează poligoanele scenei 3D în ordinea “din spate în față” (back to front), de ex. algoritmul Pictorului și BSP, I_{λ_1} corespunde poligonului care se rasterizează la un moment dat iar I_{λ_2} este valoarea existentă în memoria imagine pentru pixelul considerat.

MODELAREA TRANSPARENȚEI (6)

- Adăugarea efectului de transparență în algoritmul Z-Buffer este mai dificilă, deoarece poligoanele sunt rasterizate în ordinea în care sunt transmise în banda grafică, neținându-se cont de distanța lor față de observator.
-Se poate realiza folosind mai multe memorii buffer-imagine și generând imaginea în mai multe etape, imaginea finală obținându-se prin combinarea imaginilor poligoanelor opace cu cele ale poligoanelor transparente

In OpenGL, transparenta poate fi simulată prin amestecul dintre culoarea fragmentului curent și cea a pixelului în care se afisează, folosind modelul (R, G, B, A);

A – opacitatea: 0 – transparent; 1 – opac

Se folosesc funcțiile:

glEnable(GL_ALPHA_TEST) – activează testul alfa

MODELAREA TRANSPARENTEI (7)

glAlphaFunc(func, ref) – specifica functia de comparare a valorii A a fragmentului cu valoarea de referinta; exemplu:

func = GL_GREATER – fragmentul este afisat daca opacitatea sa este > decat valoarea ref

glEnable(GL_BLEND) - activeaza amestecul culorilor

glBlendFunc(..) – specifica modul de amestec al culorilor. Exemplu:

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA):

noua_culoare_pixel = Culoare_fragment * A_fragment + Culoare_pixel * (1 – A_fragment)

Pentru obtinerea unui efect corect ar trebui ca poligoanele sa fie transmise in banda grafica in ordinea descrescatoare a opacitatii lor.

MODELAREA CETEI (1)

Ceața adaugă realism imaginii și ajută în eliminarea defectelor produse de decuparea la nivelul planului din spate al volumului vizual:

- Fără ceață, pe măsură ce observatorul se îndepărtează de un obiect, obiectul se apropie de planul din spate și deci este decupat atunci când planul din spate îl intersectează.
- Cu ceață, dacă intensitatea cetii crește proporțional cu distanța de la poziția observatorului, se crează efectul de distanță pentru obiectele îndepărtate.
- Dacă densitatea cetii crește până la opacitate la nivelul planului din spate, decuparea față de planul din spate nu mai este necesară căci obiectele dispar datorită opacității.

Cfog – culoarea de ceață

Cfragment – culoarea calculată pentru un fragment

$0 \leq \Phi \leq 1$ – factorul de ceață, proporțional cu distanța de la observator la punctul din spațiu 3D vizibil în pixel ($\Phi=1$: opacitate)

Atunci, Cpixel, culoarea de afișare a pixelului corespunzător fragmentului este:

$$\text{Cpixel} = (1-\Phi) * \text{Cfragment} + \Phi * \text{Cfog}$$

MODELAREA CETEI (2)

- Factorul de ceata, Φ , se calculeaza, in mod uzual, folosind coordonata z a fragmentului, necesara si in algoritmul Z-buffer (interpolata de GPU).
- Factorul de ceata liniar (obtinut prin interpolare liniara) actioneaza intre 2 distante definite in sistemul coordonatelor observator, z_0 si z_1 , masurate pe directia de observare.
- Factorul de ceata este calculat in timpul rasterizarii, pentru fiecare fragment, folosind coordonata z a fragmentului si distantele z_0 si z_1 transformate in coordonate ale spatiului de afisare:
 - z: coordonata z a fragmentului
 - $\Phi = 0$ pentru $z < z_0$
 - $\Phi = (z - z_0) / (z_1 - z_0)$ pentru $z_0 \leq z \leq z_1$
 - $\Phi = 1$ pentru $z > z_1$
- Valorile implice pentru z_0 si z_1 sunt 0 si 1, in spatiul de afisare: coordonatele z ale planului din fata si planului din spate al vederii, dupa transformarea in poarta de afisare.
- Factorul de ceata moduleaza culoarea fragmentului dupa texturare.

MODELAREA CEȚEI (3)

In OpenGL, factorul de ceata poate avea o crestere liniara sau exponentiala.

- Factorul de ceata exponential este calculat in functie de coordonata z a fragmentului si de densitatea cetii astfel:

$$\Phi = e^f$$

$f = (- \text{densitate} * z)$ - in modul GL_EXP

$f = (- \text{densitate} * z)^2$ - in modul GL_EXP2

```
glEnable(GL_FOG); // activeaza modul in care se calculeaza ceata  
float FogCol[3]={0.5f,0.5f,0.5f, 1}; // Se defineste culoarea de ceata  
glFogfv(GL_FOG_COLOR,FogCol); // Se seteaza culoarea de ceata  
glFogi(GL_FOG_MODE, GL_LINEAR); // Seteaza factorul de ceata liniar  
glFogf(GL_FOG_START, 10.f); //seteaza distanta z0  
glFogf(GL_FOG_END, 20.f); // seteaza distanta z1
```

MODELAREA CEȚEI (4)

Pentru modurile GL_EXP si GL_EXP2:

GLfloat density = 0.3;

glFogf(GL_FOG_DENSITY, density);

Se poate selecta precizia calculelor de ceata:

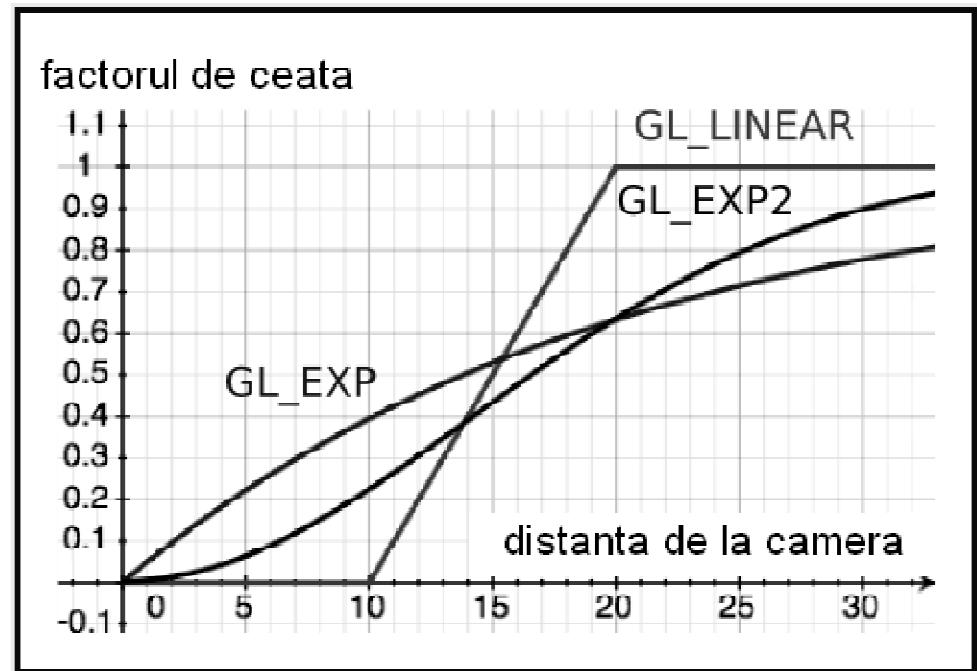
la nivel de varf sau la nivel de fragment.

gl.glHint(GL_FOG_HINT, GL_DONT_CARE);

GL_DONT_CARE,

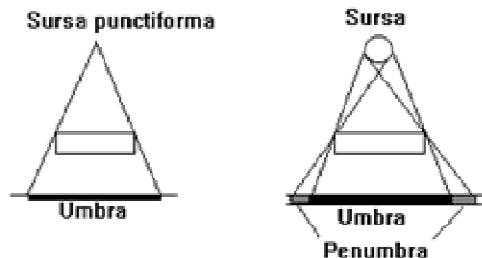
GL_NICEST (la nivel de fragment) sau

GL_FASTEST (la nivel de varf)



INTRODUCEREA UMBRELOR IN IMAGINI(1)

- Atunci când un observator privește o scenă 3D luminată de o sursă de lumină dintr-o poziție diferită de aceea a sursei de lumină, va vedea umbrele produse de obiectele scenei.
- Umbrele au o contribuție însemnată la realismul imaginii, îmbunătățind percepția profunzimii.



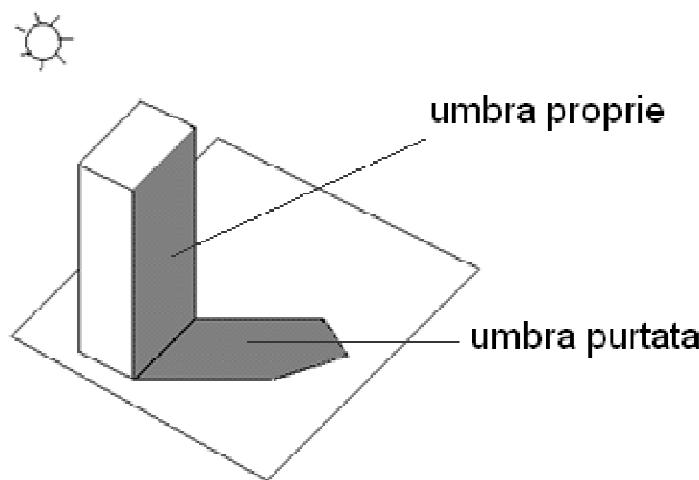
In general, se considera numai umbra pură: calcule mai simple.

Volumul de calcule depinde și de poziția sursei de lumină:

- Dacă sursa este la infinit calculele sunt mai simple.
- În cazul unei surse situate la distanță finită dar în afara câmpului vizual, este necesară o proiecție perspectivă din poziția sursei.
- Cazul cel mai dificil este acela în care sursa este situată în câmpul vizual.

INTRODUCEREA UMBRELOR IN IMAGINI(2)

- Problema determinării umbrelor este similară acelei de determinare a vizibilității obiectelor: suprafetele care nu sunt vizibile din poziția sursei de lumină sunt în umbră.
→ Crearea unei imagini cu umbre presupune rezolvarea de două ori a problemei suprafetelor nevizibile: o dată privind scena din poziția fiecărei surse de lumină, a doua oară privind-o din poziția observatorului.
- Sunt două tipuri de umbre: **umbre proprii și umbre purtate**.



INTRODUCEREA UMBRELOR IN IMAGINI(3)- POLIGOANE DETALIU

- Umbrele proprii sunt generate de obiectul însuși, care împiedică lumina să ajungă la unele dintre fețele sale: fețele umbrite de umbra proprie sunt fețele auto-obturate atunci când scena este văzută din poziția sursei de lumină.
- Umbra purtată este umbra pe care un obiect o produce pe alte părți ale scenei, la care lumina nu ajunge din cauza obiectului.

Poligoane detaliu

- Umbrele purtate se pot determina proiectând în scenă, din poziția sursei, toate poligoanele (fețele) neumbrite de umbra proprie.
 - Rezultă un set de **poligoane de umbră**, care se adaugă la reprezentarea scenei.
- Atât fetele umbrite de umbra proprie cat și poligoanele de umbra se folosesc ca poligoane-detaliu la redarea scenei 3D.
- Poligoanele detaliu sunt poligoane coplanare cu poligoanele scenei, care se folosesc în calculul culorii fragmentelor.

INTRODUCEREA UMBRELOR IN IMAGINI(4)- POLIGOANE DETALIU

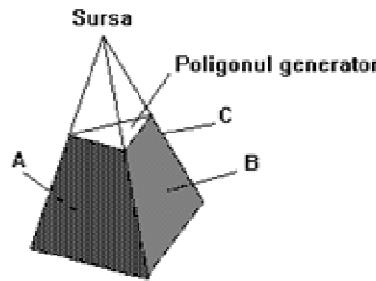
- Numărul de poligoane de umbră este mai mic, dacă în loc să se proiecteze fiecare față luminată de sursă, se proiectează silueta fiecărui obiect (văzută din poziția sursei de lumină).
- După adăugarea poligoanelor de umbră la reprezentarea scenei, se generează imaginea văzută din poziția observatorului.
- Pot fi generate mai multe vederi fără a recalculta umbrele, deoarece umbrele depind numai de poziția sursei (surselor) de lumină.
- Netinând cont de umbre, un punct P al unei suprafețe care este vizibil din poziția observatorului dar nu și din poziția unei surse se afișează cu intensitatea rezultată din reflexia luminii ambiante sau cu o intensitate care rezultă din iluminarea sa de către alte surse existente în scenă:

$$I_\lambda = I_{a_\lambda} * k_{a_\lambda} + \sum_{1 \leq i \leq m} S_i * f_{at_i} * I_{\lambda_i} * [k_{d_\lambda} * (L_{ui} \cdot N_u) + k_{s_\lambda} * (R_{ui} \cdot V_u)^n]$$

$S_i = 0$ dacă lumina de la sursa i nu ajunge în punctul P ;
 $= 1$ dacă lumina de la sursa i ajunge în punctul P : $(L_{ui} \cdot N_u) > 0$

INTRODUCEREA UMBRELOR IN IMAGINI(5) - VOLUME DE UMBRE

- Sursa de lumină este considerată punctiformă iar obiectele ca având fațete poligonale.
- Un **volum de umbră** este definit de o sursă de lumină și un poligon luminat (vizibil din poziția sursei de lumină), pe care-l vom numi **poligonul generator**.



- Fiecare față laterală a volumului este numită **poligon de umbră**. Ea este determinată de o latură a poligonului generator și de cele două drepte care pleacă din sursa de lumină, fiecare trecând printr-un vârf al laturii.
- Normalele la fețele laterale punctează înspre exteriorul volumului.

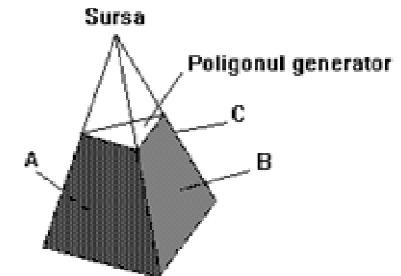
INTRODUCEREA UMBRELOR IN IMAGINI(6) - VOLUME DE UMBRE

- Volumul infinit determinat de o sursă și un poligon generator este delimitat de o față care reprezintă poligonul generator scalat. Această față este situată la o distanță față de sursă dincolo de care intensitatea luminii sursei este neglijabilă, deci orice punct aflat dincolo de această limită este umbrit.
- Volumul de umbră poate fi decupat la marginile volumului vizual.
- Poligoanele de umbră se folosesc pentru determinarea umbririi produse de poligonul generator în scenă.

Notăm cu :

- PUV poligoanele de umbră care sunt vizibile din poziția observatorului (A și B în figura) și cu
- PUN poligoanele de umbră care nu sunt vizibile din poziția observatorului (de exemplu, poligonul C).

Clasificarea poligoanelor de umbra in PUV si PUN se poate face pe baza normalelor la poligoane (back face culling).



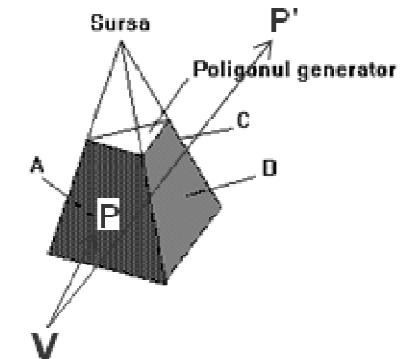
INTRODUCEREA UMBRELOR IN IMAGINI(7) - VOLUME DE UMBRE

Fie

- un punct P al unei suprafețe și
- VP vectorul din poziția observatorului (V) în punctul P.

Punctul P este umbrit dacă numărul de poligoane de tip PUV intersectate de vectorul VP este mai mare decât numărul de poligoane de tip PUN intersectate de vector.

Acesta este singurul caz, atunci când punctul V nu este în umbră.



- În general, pentru a determina dacă un punct este în umbră, se poate folosi un contor în care inițial se memorează numărul de volume de umbră care conțin poziția observatorului.
 - Se asociază poligoanelor de tip PUV valoarea +1 iar celor de tip PUN valoarea -1.
 - Atunci când vectorul VP traversează un poligon de umbră, se adună la contor valoarea asociată poligonului.
 - Punctul P este umbrit dacă valoarea contorului este pozitivă în P.

INTRODUCEREA UMBRELOR ÎN IMAGINI(8) - VOLUME DE UMBRE

- Volumul de calcul presupus de acest algoritm poate fi redus dacă în loc să se calculeze volumul de umbră pentru fiecare poligon vizibil din poziția sursei, se calculează un singur volum de umbră pentru o suprafață poliedrală. În acest scop, se determină poligoanele de umbră numai pentru laturile care fac parte din silueta suprafeței, văzută din poziția sursei.

- Silueta unei suprafețe, corespunzătoare unui punct de observare, este un set conectat de laturi care aparțin poligoanelor vizibile din punctul de observare.
- O latură de siluetă este fie o latură de margine a unei suprafețe deschise, fie o latură care separă un poligon vizibil de unul nevizibil.

- Pentru determinarea laturilor de siluetă, este necesar să se folosească o structură de date care reflectă adiacența poligoanelor. Cunoscându-se poligonul adjacents pe fiecare latură a fiecărui poligon vizibil din poziția observatorului, se pot determina rapid laturile de siluetă.

INTRODUCEREA UMBRELOR IN IMAGINI(9) - VOLUME DE UMBRE

Implementarea metodei folosind stencil-buffer(1)

- Buffer-ul de marcate (stencil-buffer) este prezent pe majoritatea placilor grafice actuale, în memoria placii, alături de buffer-ul imagine (frame-buffer) și buffer-ul de adâncime (z-buffer).
 - este o memorie de numere întregi asociată pixelilor imaginii, de regulă un byte/pixel;
 - poate fi utilizat pentru a implementa diferite operații grafice, de regulă în combinație cu z-buffer, cum este și cazul implementării volumelor de umbra;
 - valorile din stencil-buffer pot fi incrementate/ decrementate automat pentru fiecare fragment care trece/ nu trece testul de adâncime.
- Există mai multe metode de a utiliza stencil-buffer pentru implementarea metodei volumelor de umbre.

INTRODUCEREA UMBRELOR IN IMAGINI(1o) - VOLUME DE UMBRE

Implementarea metodei folosind stencil-buffer(2)

- În metoda prezentată în continuare, se crează în stencil-buffer o masă a umbrelor din imagine: orice pixel de umbra, (x,y), va fi reprezentat în stencil-buffer printr-o valoare #0.
- **Pixelii de umbra sunt afișați în culorile fragmentelor vizibile, calculate cu formula.**
Cfragment = Cemisiva + Cambianta = ke + ka*globalAmbient
ke, ka – proprietăți de material
 - Se presupune că s-au calculat poligoanele de umbra.
 - Conturul fiecarui poligon de umbra trebuie să fie orientat în sens trigonometric atunci când este privit din exteriorul volumului de umbra. În acest fel, poligoanele PUV sunt cele “din față” observatorului, iar poligoanele PUN sunt cele “din spate”.

INTRODUCEREA UMBRELOR IN IMAGINI(11) - VOLUME DE UMBRE

Implementarea metodei folosind stencil-buffer(3)

Se considera cazul in care observatorul nu este in umbra.

1.

Initializeaza z-buffer si stencil-buffer; // nu trebuie programate, acestea sunt
// initializeaza in mod automat pentru fiecare cadru imagine

Activeaza scrierea in z-buffer si dezactiveaza scrierea in stencil-buffer ; //implicite

glDisable(GL_LIGHTING); // Dezactiveaza luminile din scena 3D

*** Rasterizeaza scena (folosindu-se algoritmul z-buffer)**

Efectul:

– in frame-buffer se obtine imaginea scenei in umbra (fara lumini): pentru fiecare fragment vizibil intr-un pixel se memoreaza o culoare:

Cfragm = culoarea emisiva + culoarea ambientala;

– la sfarsitul acestui pas, z-buffer contine coordonatele z ale fragmentelor vizibile in imagine.

INTRODUCEREA UMBRELOR IN IMAGINI(12) - VOLUME DE UMBRE

Implementarea metodei folosind stencil-buffer(4)

2.

```
glDepthMask(GL_FALSE); // Dezactiveaza scrierea in z-buffer  
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE); // Dezactiveaza scrierea in frame-buffer  
glFrontFace(GL_CCW); //fetele “din fata” sunt orientate trigonometric  
glCullFace(GL_BACK); // nu sunt rasterizate poligoanele “din spate”  
glStencilFunc(GL_ALWAYS, 0x0, 0xff);  
glStencilOp(GL_KEEP, GL_INCR, GL_KEEP);  
*rasterizeaza poligoanele de umbra
```

Efectul:

Sunt rasterizate poligoanele de umbra de tip PUV. Pentru fiecare fragment rezultat, f, se face testul:

```
daca f.z < z-buffer[f.y, f.x] // fragmentul f in fata fragmentului din scena 3D vizibil in pixelul (x,y)  
atunci stencil-buffer[f.y, f.x]++; // se numara poligoanele PUV aflate intre observator si  
//obiectele scenei
```

INTRODUCEREA UMBRELOR IN IMAGINI(13) – VOLUME DE UMBRE

Implementarea metodei folosind stencil-buffer(5)

3.

glCullFace(GL_FRONT); // nu sunt rasterizate poligoanele “din fata”

glStencilFunc(GL_ALWAYS, 0x0, 0xff);

glStencilOp(GL_KEEP, GL_DECR, GL_KEEP);

***rasterizeaza poligoanele de umbra**

Efectul:

Sunt rasterizate poligoanele de umbra de tip PUN. Pentru fiecare fragment rezultat, f, se face testul:

daca $f.z < z\text{-buffer}[f.y, f.x]$ // f este in fata fragmentului din scena 3D vizibil in pixelul (x,y)

atunci stencil-buffer[f.y, f.x]--;

In stencil-buffer s-a obtinut o masca cu “gauri” in punctele care primesc lumina: valorile din stencil-buffer sunt diferite de zero pentru pixelii care trebuie afisati in umbra.

INTRODUCEREA UMBRELOR IN IMAGINI(14) - VOLUME DE UMBRE

Implementarea metodei folosind stencil-buffer(6)

4.

```
glEnable(GL_STENCIL_TEST); // Activeaza testul stencil  
glStencilOp(GL_KEEP, GL_KEEP, KEEP); // Dezactiveaza scrierea in stencil-buffer  
glStencilFunc(GL_EQUAL, 0x0, 0xff); // Specifica testul stencil  
glDepthMask(GL_TRUE); // Activeaza scrierea in z-buffer  
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE); // Activeaza scrierea in frame buffer  
glEnable(GL_LIGHTING); // Activeaza toate luminile:  
    *rasterizeaza scena (folosindu-se algoritmul zbuffer)
```

Efectul:

Se rasterizeaza scena modificandu-se in frame-buffer numai culorile pixelilor (x,y) pentru care
stencil-buffer[y, x] = 0 (fragmentul nu este in umbra)

INTRODUCEREA UMBRELOR IN IMAGINI(15) – VOLUME DE UMBRE

Implementarea metodei folosind stencil-buffer(7)

Cazul general:

Se initializeaza stencil-buffer cu o valoare reprezentand numarul de volume de umbra care contin pozitia observatorului.

Aprecieri asupra metodei volumelor de umbra

Pozitive:

- Este generala si poate fi aplicata pentru orice scena, pentru oricate surse de lumina.
- Produce forme de umbre exacte.

Negative:

- Complexa din punct de vedere al timpului de calcul.
- Sursele de lumina sunt considerate punctiforme, de aceea nu pot fi obtinute penumbre.

INTRODUCEREA UMBRELOR IN IMAGINI(16)- Z-BUFFER

Williams [WiLL78] a propus o metodă de generare a umbrelor bazată pe execuția de două ori a algoritmului Z-Buffer:

- In prima etapă se construiește buffer-ul Z al scenei văzută din poziția sursei (în transformarea de proiecție a scenei se consideră poziția sursei).

Vom nota acest buffer cu ZS. El mai este numit și « shadow map », de unde și denumirea sub care este cunoscută metoda în prezent.

- În etapa a doua se construiește imaginea văzută din poziția observatorului, utilizând ZS pentru determinarea punctelor aflate în umbra.

Implementarea modernă a metodei este numita “shadow mapping”.

INTRODUCEREA UMBRELOR IN IMAGINI(17)-Z-BUFFER

Etapa I.

Se transforma varfurile poligoanelor cu o matrice « model-view-projection », MS, calculata cu observatorul in pozitia sursei ;

- ❑ Se activeaza scrierea in z-buffer si dezactiveaza scrierea in frame-buffer
- ❑ Se rasterizeaza scena:

Pentru fiecare poligon al scenei

Pentru fiecare fragment $f(x,y,z)$ al unui poligon

Dacă $z < ZS[y][x]$ atunci

//fragmentul este vizibil în pixelul (x,y) din poziția sursei

$ZS[y][x] = z$

La sfarsitul acestei etape, in ZS sunt memorate coordonatele z ale fragmentelor care primesc lumina de la sursa.

INTRODUCEREA UMBRELOR IN IMAGINI(18)-Z-BUFFER

Etapa II.

Se calculeaza matricea « model-view-projection» cu observatorul in pozitia sa: matricea MO ;

Se asociaza fiecarui varf: pozitia sa in spatiul obiect si pozitia sa transformata cu MS ;

Se rasterizeaza scena:

Pentru fiecare poligon al scenei

Pentru fiecare fragment de coordonate (x,y,z) , al unui poligon

Dacă $z < Z\text{-Buffer}[y][x]$ atunci //fragmentul este vizibil în pixelul (x,y) , din poziția observatorului

(a) $Z\text{-Buffer}[y][x] = z$

(b) Se calculează cordonatele (x',y',z') ale fragmentului în vederea din pozitia sursei (interpoland intre cordonatele varfurilor transformate cu matricea MS)

(c) Dacă $ZS[y'][x'] < z'$ atunci //fragmentul (x,y,z) este în umbră

afișează pixelul (x,y) în intensitatea corespunzătoare umbrei

altfel

afișează pixelul (x,y) în culoarea fragmentului (calculata folosind un model de iluminare)

In cazul mai multor surse de lumină, se utilizează câte un buffer ZS pentru fiecare sursă.

Illuminarea globală *-Ray Tracing-*

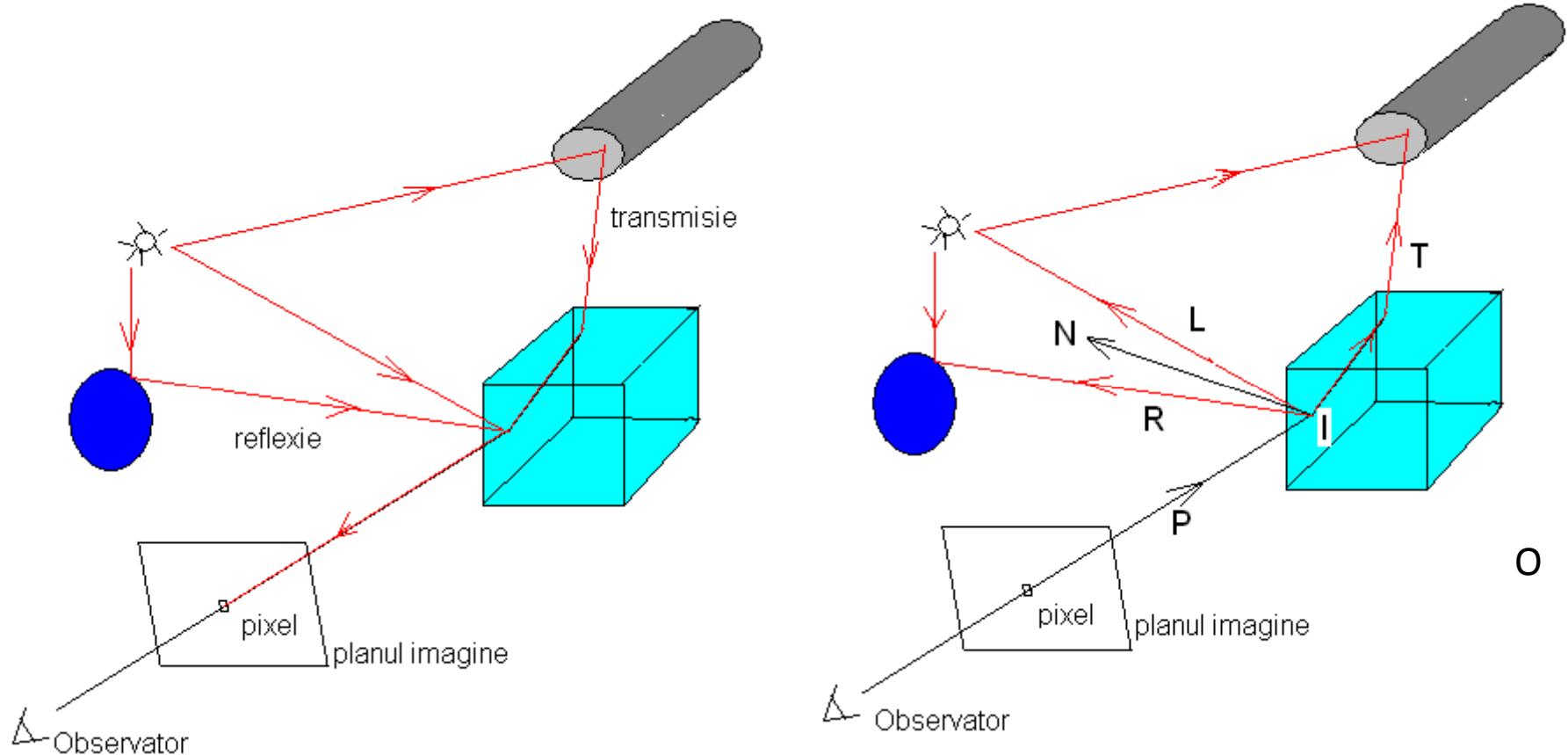
Prof. univ. dr. ing. Florica Moldoveanu

Illuminarea globală în Ray tracing(1)

Imbina:

- Eliminarea partilor nevizibile
- Calculul reflexiei luminii
- Calculul refractiei luminii
- Calculul umbrelor
- Interactiunea globală a reflexiei și refractiei luminii la nivelul întregii scene 3D

Illuminarea globală în Ray tracing(2)



Raze de lumina: directă și indirectă,
de la o sursă de lumina pe o suprafață

In calculul culorii pixelului se consideră raze cu
direcția inversă celor din realitate

Algoritmul Ray tracing

Imaginea se calculeaza pixel cu pixel, pornind de la spatiul imagine.

Considerand o singura sursa de lumina:

Pentru fiecare pixel al imaginii:

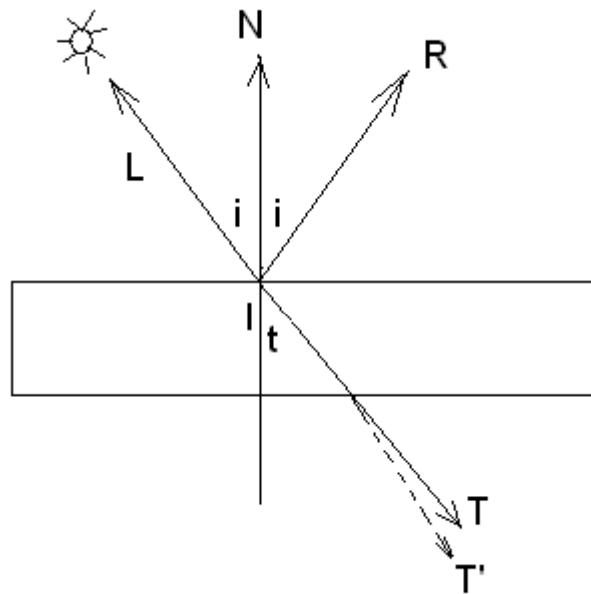
- Se calculeaza raza primara, P , care porneste din pozitia observatorului si trece prin centrul pixelului, in planul imaginii
- Se determina punctul de intersectie, I , al razei primare cu cel mai apropiat obiect de observator
- **Daca** raza nu intersecteaza nici un obiect al scenei,
 - Se afiseaza pixelul in culoarea fondului

altfel

- Se calculeaza culoarea obiectului in punctul I , tinand cont de:
 - Reflexia luminii provenita direct de la sursa de lumina, raza L
 - Lumina provenita in I prin reflexie speculara de la alte obiecte ale scenei, raza R
 - Lumina provenita in I prin transmisie de la alte obiecte ale scenei, raza T
- Afiseaza pixelul in culoarea obtinuta prin combinarea contributiei celor trei raze

Ray tracing - calculul razelor

Calculul razelor care contribuie la reflexia luminii in punctul de intersectie cu raza primara



L: raza din I catre sursa de lumina

R: raza reflectata specular, simetrica, fata de N, cu L

$$R = 2(N \cdot L) \cdot N - L$$

T: raza transmisa, calculata pe baza legii lui Snell:

$$\frac{n_1}{n_2} = \frac{\sin(t)}{\sin(i)}, \quad n_1, n_2: \text{indicii de refractie}$$

$$T = L * (\frac{n_1}{n_2}) - (\cos(t) + (\frac{n_1}{n_2}) * (L \cdot N)) * N$$

✓ Razele sunt considerate infinit subtiri

✓ Reflexia speculara si refractia au loc fara imprastiere (sunt perfect focalizate)

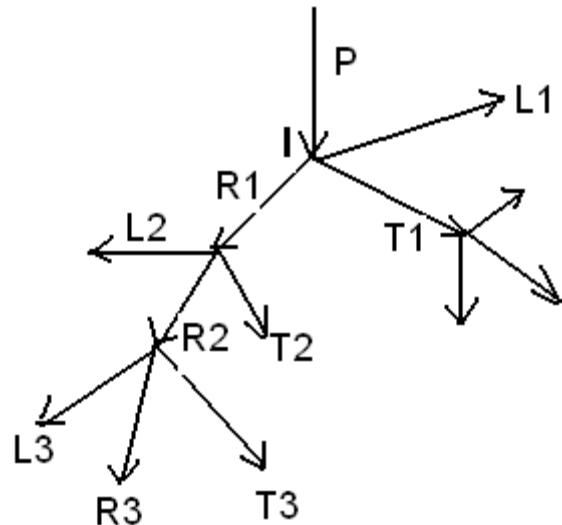
➤ **Efect:** obiectele din imaginea produsa sunt de regula stralucitoare, producand reflexii multiple focalizate.

Ray tracing - arborele de raze

Algoritmul Ray tracing este recursiv:

Lumina provenita in punctul I prin reflexie speculara de la un obiect O1 sau prin transmisie de la un obiect O2, poate fi compusa din:

- Reflexia luminii provenita direct de la o sursa, de catre O1/O2
- Reflexia speculara a altor obiecte de catre O1/O2
- Transmisia luminii prin O1/O2 provenita de la alte obiecte ale scenei



Arborele de raze

➤ Pentru obtinerea culorii pixelului, se evaluateaza arborele de raze de la frunze catre radacina

Ray tracing- calculul reflexiei luminii(1)

Aproximarea reflexiei luminii in punctul I:

$$I_{\lambda}(I) = I_{local\lambda}(I) + K_s * R_{\lambda}(I) + K_t * T_{\lambda}(I)$$

unde:

λ – reprezinta lungimea de unda: expresia se evalueaza pentru R,G,B

$I_{local\lambda}(I)$ – reprezinta componenta rezultata prin reflexia luminii provenite direct de la sursele de lumina din scena 3D (calculata folosind modelul de iluminare locala)

K_s – este coeficientul de reflexie speculara al materialului obiectului

$R_{\lambda}(I)$ – reprezinta lumina provenita prin reflexie speculara in punctul I:

se obtine prin evaluarea arborelui de raze

K_t – este coeficientul de transmisie, specific materialului obiectului

$T_{\lambda}(I)$ - reprezinta lumina provenita prin transmisie (refractie)in punctul I:

se obtine prin evaluarea arborelui de raze

$0 \leq k_s, k_t \leq 1$

Ray tracing- calculul reflexiei luminii(2)

Modelul de iluminare locală:

$$I_{\text{local}\lambda}(I) = Ia_\lambda * K_a + Isursa_\lambda * fat * s * [kd * (N \cdot L) + lum * K_s * (N \cdot H)^n]$$

Ia_λ – reprezinta intensitatea luminii ambiante

K_a – este coeficientul de difuzie a luminii ambiante, specific materialului obiectului, $0 \leq K_a \leq 1$

$Isursa_\lambda$ – reprezinta intensitatea luminii provenite de la sursa

L – este vesorul directiei din punctul I catre pozitia sursei de lumina

N – este normala in punctul I (versor)

H – este vesorul directiei bisectoare a unghiului dintre L si vectorul din I catre observator (raza P)

fat – este factorul de atenuare a luminii de la sursa, proportional cu distanta de la sursa la punctul I

$0 \leq s \leq 1$,

$s = 0$, daca lumina de la sursa nu ajunge in punctul I : vectorul L intersecteaza un obiect opac al scenei

$s = 1$, daca vectorul L nu intersecteaza un alt obiect al scenei

$0 < s < 1$, daca vectorul L intersecteaza un obiect transparent (sau mai multe)

Punctul I primeste lumina de la sursa daca:

- produsul scalar $(N \cdot L) > 0$ ($lum = 1$, altfel $= 0$)
 - raza L nu intersecteaza un obiect opac al scenei
- Daca I nu primeste lumina de la sursa, se afiseaza in culoarea luminii ambiante.

Ray tracing- calculul reflexiei lumini (3)

- Daca in scena 3D exista mai multe surse de lumina, fiecare poate contribui in mod diferit la $I_{local\lambda}(I)$:

$$I_{local\lambda}(I) = I_a * K_a + \sum_i I_{surse,i,\lambda} * f_{at,i} * s_i * [kd * (N \cdot L_i) + I_{lum,i} * K_s * (N \cdot H_i)^n]$$
$$i = 1, n$$

- Pentru evaluarea componentelor $R_\lambda(I)$ si $T_\lambda(I)$ din calculul culorii in I ,

$$I_\lambda(I) = I_{local\lambda}(I) + K_s * R_\lambda(I) + K_t * T_\lambda(I)$$

se coboara in arborele de raze pana la un numar pre-specificat de nivele (energia luminoasa scade destul de repede!).

Algoritmul Ray tracing recursiv(1)

Algoritmul Ray tracing recursiv

Pentru fiecare pixel al imaginii

```
{    *calculeaza raza primara, P;  
    *culoare_pixel = TraseuRaza(P, 1);  
    *afiseaza pixelul in culoare_pixel;  
}
```

Culoare TraseuRaza(Raza R, int n)

```
{    // n este nivelul in arborele de raze  
    *calculeaza intersectiile razei R cu obiectele scenei;  
    daca (nu exista intersectii), atunci  
        return (culoare_fond);  
    altfel  
        *fie I punctul de intersectie cel mai apropiat de observator si O obiectul intersectat;  
        *calculeaza normala N, in punctul I;  
        return CuloarePunct(O, R, I, N, n);  
}
```

Algoritmul Ray tracing recursiv(2)

Culoare CuloarePunct(Obiect O, Raza R, Punct I, Normala N, adancime_arbore n)

{ Culoare culoare;

 culoare = culoare_ambianta;

pentru fiecare sursa de lumina S execută

 *calculeaza vectorul L, din I catre S

daca ((Nu·Lu) > 0 si vectorul L nu interscteaza un obiect opac al scenei) **atunci**

 *calculeaza contributia sursei S la culoare, CS

 culoare = culoare + CS

daca (n< nivel_max) **atunci**

daca (obiectul O produce reflexii speculare) **atunci**

 *calculeaza raza reflectata in punctul I, RS;

 culoare = culoare + Ks* TraseuRaza(RS, n+1);

daca (obiectul O este transparent) **atunci**

 *calculeaza raza transmisa (refractata) in punctul I, RT;

 culoare = culoare + Kt* TraseuRaza(RT, n+1);

return (culoare)

}

Algoritmul Ray tracing recursiv(3)

Deficiențele algoritmului

1. Efecte de aliasing: razele sunt convergente și infinit subțiri

- Defecți ale marginilor suprafețelor (marginile nu sunt drepte, au aspect zimțat)
- Obiecte mici și subțiri pot să apară și să dispare din imagine, la schimbarea poziției observatorului

Imbunătățire:

- marirea rezoluției esantionării spațiului 3D; ex. 4 raze primare/pixel → crește complexitatea algoritmului

2. Complexitatea computațională

- buna din punct de vedere teoretic:
 - pentru o raza/pixel este de $O(p \cdot n)$, unde p este nr. de pixeli - constant (independent de scenă)
 $= O(n)$, unde n este nr. de obiecte din scenă
- Creaște liniar cu numărul de raze/pixel
- Complexitatea calculelor de intersecție depinde de geometria obiectelor intersectate

Imbunătățire:

- Reducerea calculelor de intersecție, prin folosirea de volume încadratoare și gruparea obiectelor din scenă 3D
- Complexitatea se poate reduce la $O(\log n)$

Algoritmul Ray tracing - optimizari(1)

Optimizari:

1. Imbunatatirea calitatii imaginii(1)

Marirea numarului de raze primare

1) Uniforma:

- 4raze/pixel, care trec prin colturile suprafetei pixelului
- culoarea pixelului se determina ca medie a culorilor obtinute cu cele 4 raze primare
- razele primare utilizate pentru un pixel contribuie la culoarea pixelilor adjacenti
- pentru o imagine de mxn pixeli: $(m+1) \times (n+1)$ raze \rightarrow numarul de raze creste cu $(m+n+1)$

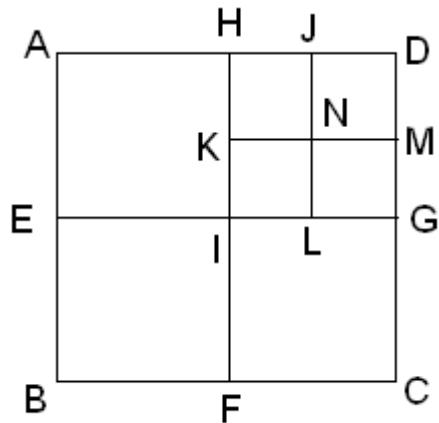
2) Adaptiva (Whitted):

- Se creste rezolutia esantionarii spatiale numai in zonele in care este necesara operatia de anti-aliasing:
- Daca diferența dintre culorile celor 4 raze primare pentru un pixel este mare, se subdivizeaza suprafata pixelului in 4 subzone
- Se aplica acelasi criteriu de comparatie intre culorile celor 4 raze primare corespunzatoare unei subzone
- Subdivizarea se continua recursiv pana la un nivel maxim prestabil sau pana cand diferența dintre cele 4 culori scade sub un prag dat
- Culoarea pixelului se obtine ca o medie ponderata a culorilor subzonelor in care a fost divizata suprafata pixelului

Algoritmul Ray tracing - optimizari(2)

Imbunatatirea calitatii imaginii(2)

Exemplu de subdivizare adaptiva



CA, CB, CC, CD, CE, CF, CG, CH, CI, CJ, CK,
CL, CM, CN
- sunt culorile obtinute cu razele primare care trec prin
punctele respective

Culoare pixel (A-B-C-D) =

$$\begin{aligned} & \frac{1}{4}((CA+CE+CI+CH)/4 + (CE+CB+CF+CI)/4 + (CI+CF+CC+CG)/4 + \\ & \frac{1}{4}((CH+CK+CN+CJ)/4 + (CK+CI+CL+CN)/4 + (CJ+CN+CM+CD)/4 + \\ & (CN+CL+CG+CM)/4)) \end{aligned}$$

Algoritmul Ray tracing - optimizari(3)

Optimizari:

2. Reducerea complexitatii computationale(1)

2.1 Utilizarea de volume incadratoare la nivel de obiect/ grup de obiecte

- Testarea intersectiei raza-volum incadrator in loc de raza-obiect:
 - Daca raza nu intersecteaza volumul incadrator → nu se va calcula intersectia cu obiectul/obiectele din volum
- Eliminarea unui intreg grup de obiecte care nu este intersectat de raza
- Evitarea calculelor de intersectie cu obiectele scenei, care pot avea geometrie complexa

Volume incadratoare: sfera, paralelipipedul cu fetele paralele cu planele principale, elipsoidul, cilindrul.

- Calculul intersectiei raza-volum incadrator trebuie sa fie mai simplu decat calculul intersectiei cu obiectul (de ex. o retea poligonală)
- Volumul incadrator al unui obiect se alege in functie de forma obiectului

Algoritmul Ray tracing - optimizari(4)

Reducerea complexitatii computationale(2)

Calcule de intersectie raza-volum incadrator(1)

Ecuatia razei: $r(t) = P_0 + t*D$,

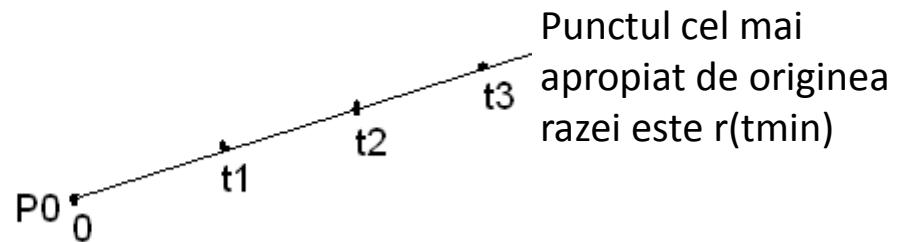
P_0 – este originea razei

D – este directia razei

$$x = x_0 + t*dx$$

$$y = y_0 + t*dy$$

$$z = z_0 + t*dz$$



1) Intersectia cu sfera

$$(x-a)^2 + (y-b)^2 + (z-c)^2 = r^2$$

- Se inlocuiesc in ec sferei x , y , z , cu cele din ecuatia razei
- Rezulta o ecuatie de grad 2 in t
- Se calculeaza discriminantul D , al ecuatiei:
 - $D < 0$ – raza nu intersecteaza sfera
 - Daca sfera este obiect al scenei (intereseaza punctul de intersectie):
 - $D=0$ – raza este tangent la sfera
 - $D > 0$ – se calculeaza radacinile, t_1 , t_2
 - punctul de intersectie mai apropiat de observator este $r(t_{min}(t_1,t_2))$

Algoritmul Ray tracing - optimizari(5)

Reducerea complexitatii computationale(3)

Calcule de intersectie raza-volum incadrator(2)

2) Intersectia cu un paralelipiped cu fetele paralele cu planele principale

Planele care delimitaaza volumul:

$$x = x_{\min}, x = x_{\max}, y = y_{\min}, y = y_{\max}, z = z_{\min}, z = z_{\max}$$

Intersectia cu $x=x_{\min}$ si $x=x_{\max}$

$$x_{\min} = x_0 + t * dx \rightarrow t_{1x} = (x_{\min} - x_0) / dx$$

$$x_{\max} = x_0 + t * dx \rightarrow t_{2x} = (x_{\max} - x_0) / dx$$

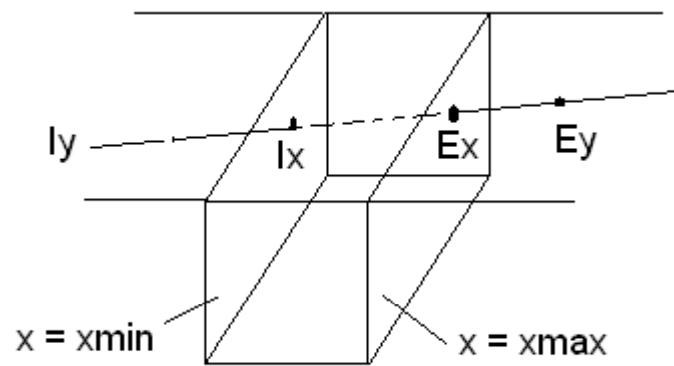
$$t_{ix} = \min(t_{1x}, t_{2x}), t_{ex} = \max(t_{1x}, t_{2x})$$

$$I_x = r(t_{ix}), E_x = r(t_{ex})$$

Analog pentru intersectia cu planele $y=y_{\min}$ si $y=y_{\max}$:

$$t_{iy} = \min(t_{1y}, t_{2y}), t_{ey} = \max(t_{1y}, t_{2y})$$

$$I_y = r(t_{iy}), E_y = r(t_{ey})$$



Raza intersecteaza volumul daca:

$$\max(t_{ix}, t_{iy}) < \min(t_{ex}, t_{ey})$$

Algoritmul Ray tracing - optimizari(6)

Reducerea complexitatii computationale(4)

Calcule de intersectie raza-volum incadrator(3)

2) Intersectia cu un paralelipiped cu fetele paralele cu planele principale (continuare)

daca $\max(t_{ix}, t_{iy}) > \min(t_{ex}, t_{ey})$ → raza nu intersecteaza volumul
altfel, se calculeaza intersectia cu $z=z_{\min}$ si $z=z_{\max}$

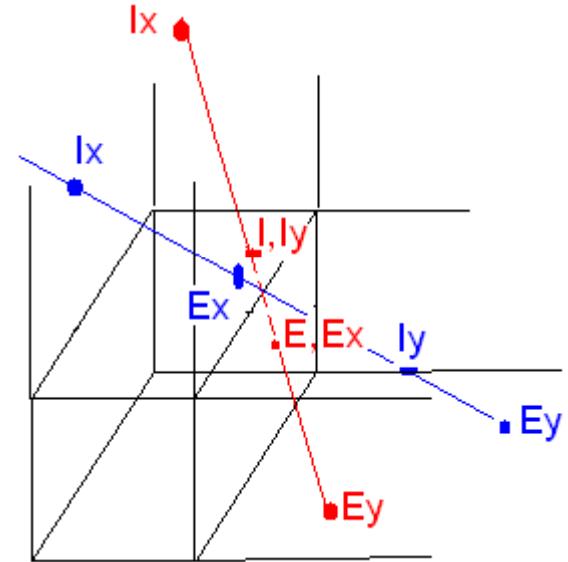
Punctele de intrare si iesire din volum sunt:

$$I = r(\max(t_{ix}, t_{iy}, t_{iz}))$$

$$E = r(\min(t_{ex}, t_{ey}, t_{ez}))$$

daca $\max(t_{ix}, t_{iy}, t_{iz}) < \min(t_{ex}, t_{ey}, t_{ez})$

atunci raza intersecteaza volumul



$$\max(t_{ix}, t_{iy}) = t_{iy}, \min(t_{ex}, t_{ey}) = t_{ex}$$

$$\max(t_{ix}, t_{iy}) > \min(t_{ex}, t_{ey})$$

$$\max(t_{ix}, t_{iy}) < \min(t_{ex}, t_{ey})$$

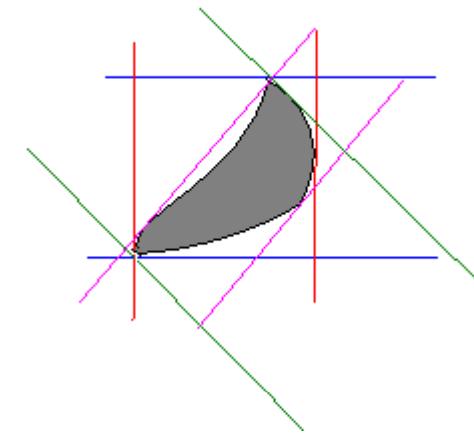
Algoritmul Ray tracing - optimizari(7)

Reducerea complexitatii computationale(5)

Calcule de intersectie raza-volum incadrator(4)

3) Intersectia cu un volum incadrator poliedru convex

Volumul incadrator (Kay, Kajiya[1986]) este un poliedru convex format din intersectiile a 4 perechi de plane paralele, inclinate la 0, 45, 90, 135 grade fata de planul orizontal.



Ec. unui plan: $A*x + B*y + C*z + D = 0$

- Fie t_{11} si t_{12} - valorile pentru intersectiile cu 2 perechi de plane paralele
 $t_{1min} = \min(t_{11}, t_{12})$ corespunde punctului de intersectie mai apropiat de observator
 - Fie t_{21}, t_{22} - valorile pentru intersectiile cu urmatoarele 2 perechi de plane paralele
 - daca $\max(t_{1min}, t_{2min}) > \min(t_{1max}, t_{2max})$ raza nu intersecteaza volumul
altfel, se continua cu intersectia urmatoarei perechi de plane
- Punctele de intersectie cu volumul: $I = r(\max(t_i, min))$, $E = r(\min(t_i, max))$

Algoritmul Ray tracing - optimizari(8)

Reducerea complexitatii computationale(6)

2.2. Divizarea scenei in volume incadratoare(1)

Divizarea regulata a scenei

- se porneste de la paralelipipedul incadrator al scenei, care se divizeaza recursiv in (8) subvolume egale, numite voxeli, pana la o anumita rezolutie
- la divizare nu se tine cont de structura scenei
- pentru fiecare voxel se memoreaza lista obiectelor pe care le contine
- daca raza nu intersecteaza un voxel → ea nu intersecteaza nici un obiect atasat voxelului
- volumul de voxeli poate fi reprezentat eficient, tinand cont de coerența spatială a voxelilor
- traseul razei prin volumul de voxeli poate fi calculat eficient printr-un algoritm DDA 3D

Algoritmul Ray tracing - optimizari(9)

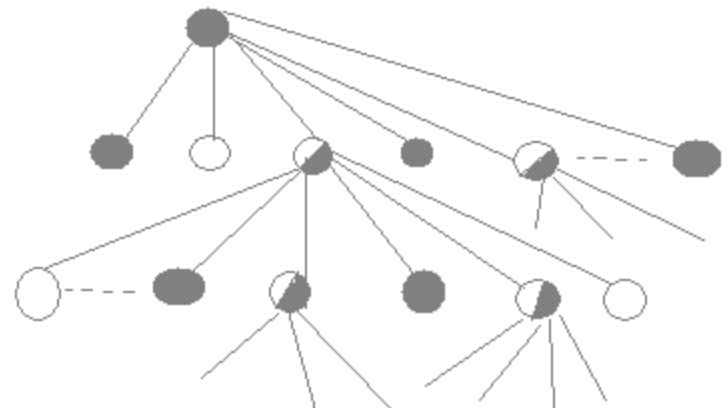
Reducerea complexitatii computationale(7)

2.2 Divizarea scenei in volume incadratoare (2)

Divizarea adaptiva a scenei

- Se porneste de la paralelipipedidul incadrator al scenei, care se divide in 8 subvolume egale, in mod recursiv, divizarea unui subvolum terminandu-se daca el nu contine nici un obiect al scenei sau contine un singur obiect.
- Scena se reprezinta printr-un arbore octal, fiecare nod fiind asociat unui subvolum.
- Pentru intersectia razei cu scena se fac teste de intersectie de la radacina spre frunze.
- Daca un subvolum nu este intersectat de raza, atunci nici unul dintre subvolumele (obiectele) din subarborele nodului subvolumului nu va fi intersectat.

Arborele octal al scenei



Algoritmul Ray tracing - optimizari(10)

Reducerea complexitatii computationale(8)

2.3 Ierarhie de volume incadratoare de grupuri de obiecte

- Scena 3D este reprezentata printr-un arbore (arborele scenei) in care fiecare nod are un parinte si un numar oarecare de copii
- Frunzele contin obiectele scenei, celelalte noduri contin grupuri de obiecte
- Fiecarui nod ii este atasat volumul incadrator al grupului de obiecte
- Pentru intersectia razei cu scena este parcurs arborele de la radacina spre frunze

Algoritmi de generare a cercurilor și elipselor în spațiul discret

Prof. univ. dr. ing. Florica Moldoveanu

Aproximarea cercurilor in spatiul discret(1)

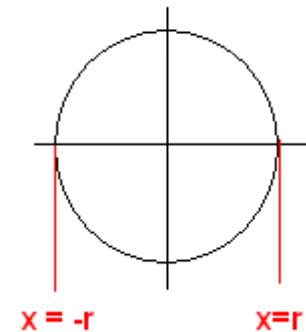
1) Folosind ecuatia cercului in coordonate carteziene

$$(x - xc)^2 + (y - yc)^2 = r^2$$

Generarea cercului cu centrul in (xc, yc) se reduce la generarea cercului centrat in origine, plus o translatie cu (xc, yc) :

$$x^2 + y^2 = r^2$$

```
for x = -r; x<=r; x++)
{   y = (+/-) sqrt(r^2 - x^2);
    putpixel (xc + x, yc + y, culoare); }
```



Dezavantaje:

- Spatierea inegală a punctelor de pe cerc afisate
- Calcule complexe cu numere reale

Aproximarea cercurilor in spatiul discret(2)

2) Folosind ecuatia cercului in coordonate polare

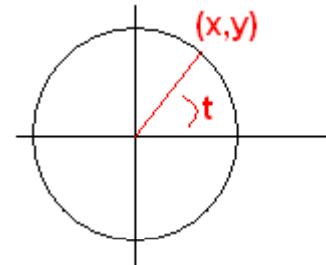
$$x = xc + r^* \cos(t)$$

$$y = yc + r^* \sin(t) \quad 0 \leq t \leq 2\pi$$

Generarea se reduce la generarea cercului centrat in origine, plus o translatie cu (xc, yc) :

$$x = r^* \cos(t)$$

$$y = r^* \sin(t) \quad 0 \leq t \leq 2\pi$$



Cercul poate fi aproximat printr-o polilinie:

- daca se calculeaza N puncte de pe cerc, egal distante, t creste cu pasul $2\pi/N$

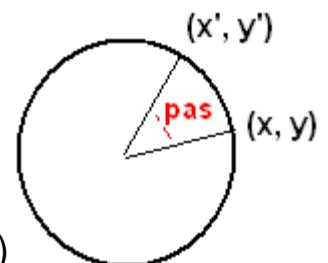
Calculul punctelor succesive de pe cerc, pe baza unei relatii de recurrenta:

- (x, y) : punctul curent

- (x', y') : punctul urmator

$$x' = r^* \cos(t + \text{pas}) = r(\cos(t) * \cos(\text{pas}) - \sin(t) * \sin(\text{pas})) = x * \cos(\text{pas}) - y * \sin(\text{pas})$$

$$y' = r^* \sin(t + \text{pas}) = r(\cos(t) * \sin(\text{pas}) + \sin(t) * \cos(\text{pas})) = x * \sin(\text{pas}) + y * \cos(\text{pas})$$



Aproximarea cercurilor in spatiul discret(3)

Avantajul calculului prin recurenta: nu este necesar sa se calculeze sin si cos in fiecare iteratie!

// Aproximarea unui cerc printr-o linie poligonală

```
void Cerc( int xc, int yc, int r, int N)

{ double pas = 2*M_PI /N ; //pasul unghiular

double u, x, y, x1,y1, s, c;

c=cos(pas); s=sin(pas); // sin si cos se calculeaza o singura data

for( int i = 1, x=r, y=0; i<N; i++)

{ x1 =x*c - y*s; // x'

y1 =x*s + y*c; // y'

line(xc+(int)(x+0.5), yc+(int)(y+0.5), xc+(int)(x1+0.5), yc+(int)(y1+0.5));

x=x1 ; y= y1 ;

}

line(xc+(int)(x+0.5), yc+(int)(y+0.5),xc+r,yc);

}
```

Aproximarea cercurilor in spatiul discret(4)

Simetria punctelor de pe cerc

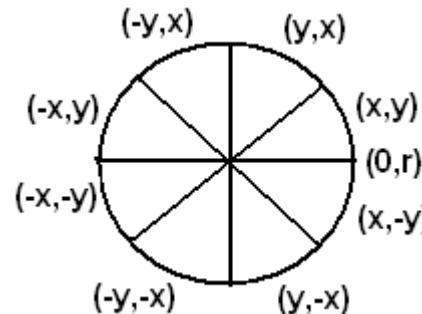
- Fiecare punct de pe cerc are 7 puncte simetrice pe cerc:

- Este suficient sa se calculeze adresele punctelor dintr-un singur octant al cercului, celelalte puncte de pe cerc fiind calculate prin simetrie

// Afiseaza punctele simetrice de pe cerc

```
void punct_simetric(int xc,int yc,int x,int y, int culoare)
```

```
{ putpixel(xc+x,yc+y,culoare);  
putpixel(xc+x,yc-y,culoare);  
putpixel(xc-x,yc-y,culoare);  
putpixel(xc-x,yc+y,culoare);  
putpixel(xc+y,yc+x,culoare);  
putpixel(xc+y,yc-x,culoare);  
putpixel(xc-y,yc-x,culoare);  
putpixel(xc-y,yc+x,culoare);  
}
```



Aproximarea cercurilor in spatiul discret(5)

- Pentru afisarea punctelor succesive, pasul unghiular trebuie sa fie aprox. (1/ raza).

// Calculeaza numai punctele din primul octant si afiseaza toate punctele simetrice

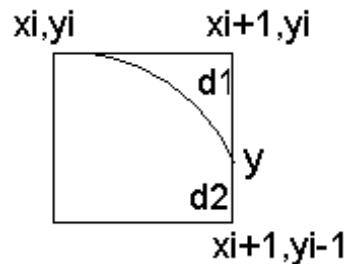
```
void Cerc_sim( int xc, int yc, int r, int culoare)
{
    double pas, u, x, y, xx, s, c;
    pas=1.0/r; // pas unghiular pentru obtinerea de puncte succesive de pe cerc
    c=cos(pas); s=sin(pas);
    putpixel(xc+r,yc,culoare);
    putpixel(xc-r,yc,culoare);
    putpixel(xc,yc+r,culoare);
    putpixel(xc,yc-r,culoare);

    for(u=pas, x=r, y=0; u<M_PI_4; u+=pas)
    {
        xx=xx;
        x =xx*c - y*s;
        y =xx*s + y*c;
        punct_simetric(xc, yc,(int)(x+0.5), (int)(y+0.5), culoare);
    }
}
```

Algoritmul Bresenham pentru rasterizarea cercurilor(1)

- Consideram cercul cu centrul în originea sistemului de coordonate și raza r .
- În cadrul algoritmului **se calculează numai punctele din octantul al 2-lea**, celelalte obținându-se prin simetrie.
- Se calculează punctele de pe cerc începând cu $(x=0, y=r)$ până la $(x=y)$
- Fie (x_i, y_i) ultimul punct al spațiului discret, ales pentru aproximarea cercului.

Urmatorul punct va fi unul dintre (x_i+1, y_i) și (x_i+1, y_i-1) :



Fie y ordonata punctului de pe cercul teoretic, cu abscisa (x_i+1) .

$$y^2 = r^2 - (x_i+1)^2$$

Fie d_1 și d_2 distanțele de la cele 2 puncte ale spațiului discret la punctul de pe cercul teoretic.

Interesează semnul diferenței $(d_1 - d_2)$:

- dacă $(y_i - y) > (y - (y_i - 1))$ atunci

$$(y_i^2 - y^2) > (y^2 - (y_i - 1)^2)$$

$$d_1' = y_i^2 - y^2 = y_i^2 - r^2 + (x_i+1)^2$$

$$d_2' = y^2 - (y_i - 1)^2 = r^2 - (x_i+1)^2 - (y_i - 1)^2$$

Algoritmul Bresenham pentru rasterizarea cercurilor(2)

- Notam cu t_i eroarea de aproximare în pasul curent:

$$t_i = d1' - d2' = y_i^2 + 2*(x_i+1)^2 + (y_i-1)^2 - 2*r^2$$

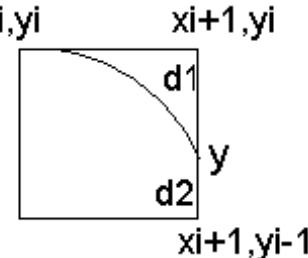
- Se obtine o relatie de recurenta pentru calculul erorii de aproximare in pasul urmator:

$$t_{i+1} = y_{i+1}^2 + 2*(x_{i+1}+1)^2 + (y_{i+1}-1)^2 - 2*r^2$$

(1) Daca $t_i < 0$ ($d1' < d2'$) atunci $x_{i+1} = x_i + 1$ si $y_{i+1} = y_i$. Deci,

$$t_{i+1} = y_i^2 + 2*((x_i+1)+1)^2 + (y_i-1)^2 - 2*r^2$$

sau $t_{i+1} = t_i + 4*x_i + 6$



(2) Daca $t_i >= 0$ atunci $x_{i+1} = x_i + 1$ si $y_{i+1} = y_i - 1$. Deci,

$$t_{i+1} = (y_i-1)^2 + 2*((x_i+1)+1)^2 + ((y_i-1)-1)^2 - 2*r^2$$

sau $t_{i+1} = t_i + 4*(x_i - y_i) + 10$

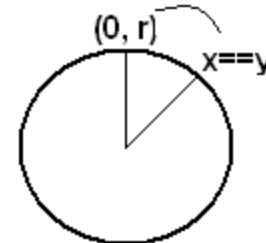
Algoritmul Bresenham pentru rasterizarea cercurilor(3)

- Valoarea erorii de aproximare în primul pas:

$$t_i = y_i^2 + 2*(x_i+1)^2 + (y_i-1)^2 - 2*r^2 : \quad x_1=0, y_1=r. \text{ Rezultă, } t1 = 3-2*r$$

```
void Bres_cerc(int xc, int yc, int r, int culoare)
```

```
{ int x,y,t;  
t=3-(r<<1);  
putpixel(xc+r,yc,culoare); putpixel(xc-r,yc,culoare);  
putpixel(xc,yc+r,culoare); putpixel(xc,yc-r,culoare);  
for(x=1,y=r; x<y; x++)  
{  
    if(t<0)  
        t+=6+(x<<2);  
    else  
        { t+=10+((x-y)<<2); y--;}  
    punct_simetric(xc,yc,x,y,culoare);  
}  
putpixel(xc+x,yc+y,culoare); putpixel(xc+x,yc-y,culoare);  
putpixel(xc-x,yc+y,culoare); putpixel(xc-x,yc-y,culoare);  
}
```

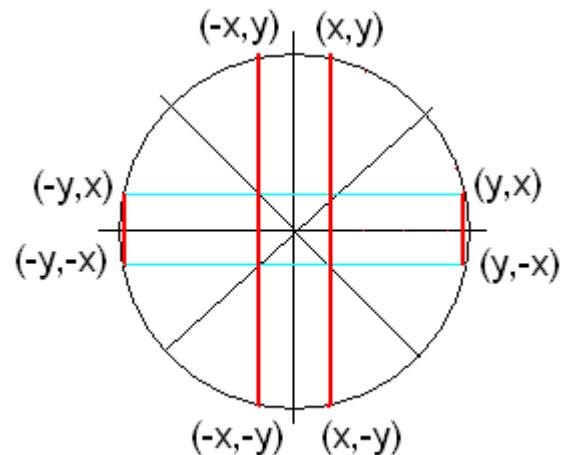


Generarea unei suprafete circulare

EXERCITIU:

- Modificati algoritmul Bresenham pentru generarea cercurilor astfel incat sa se obtina o suprafata circulara.

```
void Bres_cerc(int xc, int yc, int r, int culoare)
{ int x,y,t;
  t=3-(r<<1);
  line(xc,yc+r,xc, yc-r, culoare); putpixel(xc-r,yc,culoare); putpixel(xc+r,yc,culoare);
  for(x=1,y=r; x<y; x++)
  {
    if(t<0)
      t+=6+(x<<2);
    else
      { t+=10+((x-y)<<2); y--;}
    line_simetric(xc,yc,x,y,culoare);
  }
  putpixel(xc+x,yc+y,culoare); putpixel(xc+x,yc-y,culoare);
  putpixel(xc-x,yc+y,culoare); putpixel(xc-x,yc-y,culoare);
}
```



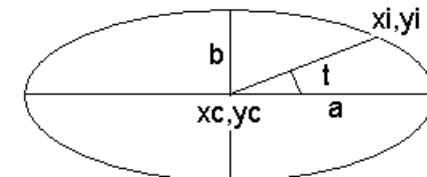
Aproximarea elipselor în spațiul discret(1)

- ❖ Ecuatiile parametrice ale elipsei sunt:

$$x = xc + a * \cos(t)$$

$$y = yc + b * \sin(t) \quad 0 \leq t \leq 2\pi$$

unde (xc, yc) este centrul elipsei, iar a și b sunt marimile semiaxelor elipsei.



- ❖ Se poate obține o secvență de puncte care aproximează circumferința elipsei dându-i lui t valori de la 0 la 2π cu un pas constant.

- Fie (x_i, y_i) ultimul punct calculat de pe circumferința elipsei:

$$x_i = xc + a * \cos(t) = xc + a * dx \quad dx = \cos(t)$$

$$y_i = yc + b * \sin(t) = yc + b * dy \quad dy = \sin(t)$$

- Următorul punct, (x_{i+1}, y_{i+1}) , este distanțat de cel curent cu pasul unghiular:

$$x_{i+1} = xc + a * \cos(t + \text{pas}) = xc + a * dx'$$

$$y_{i+1} = yc + b * \sin(t + \text{pas}) = yc + b * dy'$$

$$dx' = \cos(t + \text{pas}) = \cos(t) * \cos(\text{pas}) - \sin(t) * \sin(\text{pas}) = dx * c - dy * s$$

$$dy' = \sin(t + \text{pas}) = \cos(t) * \sin(\text{pas}) + \sin(t) * \cos(\text{pas}) = dx * s + dy * c$$

unde $c = \cos(\text{pas})$, $s = \sin(\text{pas})$ sunt constante

Aproximarea elipselor in spatiul discret(2)

Rezultă:

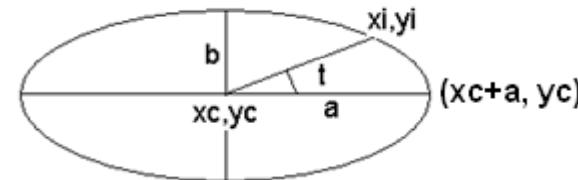
$$x_{i+1} = xc + a \cdot dx' = xc + a \cdot (dx \cdot c - dy \cdot s)$$

$$y_{i+1} = yc + b \cdot dy' = yc + b \cdot (dx \cdot s + dy \cdot c)$$

// Generarea unei elipse cu axele paralele cu axele sistemului de coordonate carteziene 2D

void Elipsa(int xc, int yc, int a, int b, int N)

```
{ double c,s,t,dx,dy,u, pas = 2*M_PI/N; // N: numarul de puncte calculate pe circumferinta elipsei
int x, y, x1, y1 ;
c=cos(pas); s=sin(pas);
x= xc +a; y = yc; // dx = cos(0) = 1; dy = sin(0) = 0
for(i=1, dx=1, dy=0; i<N; i++)
{
    t=dx;
    dx=dx*c - dy*s; // dx'
    dy=t*s + dy*c; //dy'
    x1 = xc +(int)(a*dx+0.5); y1 = yc+(int)(b*dy+0.5);
    line(x, y, x1, y1);
    x = x1; y = y1;
} line(x, y, xc+a,yc);
}
```



Aproximarea elipselor în spațiul discret(3)

Generarea unei elipse rotite

- ❖ Fie R, o elipsă cu centrul în originea sistemului de coordonate și semiaxele a, b rotite cu unghiul u.
 - ❖ Fie E, elipsa cu centrul în origine și semiaxele a și b suprapuse peste axele sistemului de coordonate.
 - ❖ Punctele de pe elipsa R pot fi obținute rotind punctele de pe elipsa E în jurul originii cu unghiul u.
 - ❖ Dacă centrul elipsei R este în (xc, yc) , atunci după rotație se aplică fiecărui punct translația cu (xc, yc) .
-
- Fie: $x' = a * dx'$, $y' = b * dy'$ punctul curent de pe elipsa E, unde vectorul $[dx', dy']$ s-a obținut prin aplicarea formulei de recurență.
 - Notăm: $dx1 = a * dx'$, $dy1 = b * dy'$
 - Punctul corespunzător de pe elipsa R este:

$$xr = xc + dxr, yr = yc + dyr$$

unde (dxr, dyr) se obțin prin rotatia vectorului $(dx1, dy1)$ în jurul originii

$$dxr = dx1 * \cos(u) - dy1 * \sin(u), dyr = dx1 * \sin(u) + dy1 * \cos(u)$$

Aproximarea elipselor in spatiul discret(4)

// Generarea unei elipse cu axele rotite fata de axele sistemului de coordonate carteziene 2D

```
void Elipsa_rot(int xc, int yc, int a, int b, int N, double u)
{ double c,s,t,dx,dy,dx1,dy1,cu,su, pas = 2*M_PI/N;
  c=cos(pas);s=sin(pas);
  cu=cos(u) ;su=sin(u);
  x= xc+(int)(a*cu+0.5); y= yc+(int)(a*su+0.5); // punctul initial de pe elipsa rotita
  for(i=1, dx=1,dy=0; i< N; i++)
  {
    t=dx;
    dx=dx*c-dy*s; //dx'
    dy=t*s+dy*c; // dy'
    dx1=a*dx; dy1=b*dy; // punctul de pe elipsa nerotita cu centrul in origine
    x1 = xc+ (int)(dx1*cu-dy1*su+0.5); y1 = yc + (int)(dx1*su+dy1*cu+0.5);
    line(x, y, x1, y1);
    x = x1; y = y1;
  }
  line(x, y, xc+(int)(a*cu+0.5),yc+(int)(a*su+0.5));
}
```

Rasterizarea vectorilor

Prof. univ. dr. ing. Florica Moldoveanu

Rasterizarea

- Primitivele geometrice (linii, cercuri, poligoane) sunt definite intr-un plan analogic, XOY, prin coordonate.
- Suprafata de afisare este un spatiu discret, alcătuit din celule de afisare (pixeli) adresate prin coordonate intregi, $(0,0) \leq (x,y) \leq (xmax,ymax)$
 - Pentru afisare, este necesara descompunerea primitivelor în fragmente care se afisează în celulele suprafetei de afisare.
- **Rasterizarea** = operația prin care o primitivă grafică este descompusă în fragmente
 - = aproximarea în spațiul discret a unei primitive definite analitic

Un vector este definit analitic prin coordonatele extremitatilor sale : $(x_1, y_1) - (x_2, y_2)$

- **Rasterizarea unui vector** = determinarea pixelilor spațiului discret care sunt cei mai apropiati de vectorul analitic.

Rasterizarea vectorilor

Ecuatia unui vector: $y = m*x + b$

$$m = (y_2 - y_1) / (x_2 - x_1) \text{ și } b = y_1 - m * x_1$$

Fie:

culoare : culoarea in care se afiseaza fragmentele care alcataiesc vectorul

putpixel(x,y, culoare);// scrie culoarea in celula din memoria imagine(frame buffer), corespunzatoare pixelului de adresa (x,y)

Afisarea vectorului:

```
putpixel(x1,y1, culoare);
for(x=x1+1; x<x2; x++)
{ y=m*x+b ;
  putpixel(x, (int)(y+0.5),culoare);
}
putpixel(x2,y2,culoare);
```

Dezavantaje algoritm :

- Nu se tine cont de panta dreptei: vectorii cu panta mare sunt aproximati prin cativa pixeli!
- Calculul fiecarei adrese de pixel de pe traseul vectorului contine operatii cu numere reale

Algoritmul Digital Differential Analyser (DDA)

- Tine cont de panta vectorului
- Coordonatele pixelilor de pe traseul vectorului se obtin printr-un calcul incremental (eficient)

Fie (x', y') si (x'', y'') - 2 puncte succesive de pe vector

- $(y'' - y') / (x'' - x') = (y_2 - y_1) / (x_2 - x_1) = m$
 $|m| < 1$: incrementare x : $x'' = x' + 1 \rightarrow y'' = y' + m$
 $|m| > 1$: incrementare y: $y'' = y' + 1 \rightarrow x'' = x' + 1/m$

```
void DDA(int x1, int y1, int x2, int y2, int culoare)
{double m,r; int x, y;
if(x1==x2) //vector vertical
{if(y1>y2)
{y=y1; y1=y2; y2=y;}
for(y=y1; y<=y2;y++)
putpixel(x1,y,culoare);
return;
}
```

Algoritmul DDA(2)

```
if(y1==y2) //vector orizontal
{if(x1>x2)
 {x=x1; x1=x2; x2=x;}
 for(x=x1; x<= x2; x++) putpixel(x,y1,culoare);
 return;
}
m=(double)(y2-y1)/(x2-x1); r=abs(m);
// se genereaza vectorul de la (x1,y1) la (x2,y2)
if(r<=1 && x1>x2 || r>1 && y1>y2)
 {x=x1; x1=x2; x2=x; y=y1; y1=y2; y2=y;}
putpixel(x1, y1, culoare);           Dezavantaj: calcule cu numere reale
if( r<=1)
 for(x=x1+1, r=y1; x<x2; x++)
 {r+=m; putpixel(x, (int)(r+0.5), culoare); } // y = y + m
else
 {m=1/m;
 for(y=y1+1, r=x1; y<y2; y++)
 {r+=m; putpixel((int)(r+0.5), y, culoare); } // x = x +1/m
 }
putpixel(x2, y2, culoare);
}
```

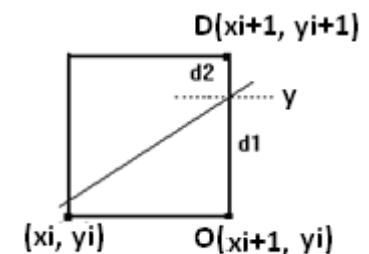
Algoritmul Bresenham -pentru vectori din primul octant- (1)

- Contine numai operatii cu numere intregi
- Calcul incremental al coordonatelor pixelilor de pe traseul vectorului
- Pentru fiecare valoare a lui x se alege acel punct al spațiului discret care este mai apropiat de punctul de pe vectorul teoretic

- Fie $m = (y_2 - y_1) / (x_2 - x_1)$ panta vectorului,
- (x_i, y_i) ultimul punct al spațiului discret ales în procesul de generare a vectorului.
- d_1 distanță de la punctul de pe vectorul teoretic, $(x_i + 1, y)$, la punctul $O(x_i + 1, y_i)$
- d_2 distanță de la punctul de pe vectorul teoretic la punctul $D(x_i + 1, y_i + 1)$.
- **O si D sunt adrese de pixeli (punkte ale spatiului discret)**

➤ Următorul punct al spatiului discret ales pentru aproximarea vectorului va fi:

- O dacă $d_1 < d_2$, D în caz contrar.
- Dacă $d_1 = d_2$ se poate alege oricare dintre cele două puncte.



Algoritmul Bresenham(2)

- Exprimăm diferența $d_1 - d_2$:

$y = m \cdot (x_i + 1) + b$ este ordonata punctului de pe vectorul teoretic

$$d_1 = y - y_i = m \cdot (x_i + 1) + b - y_i$$

$$d_2 = y_i + 1 - y = y_i + 1 - m \cdot (x_i + 1) - b$$

$$d_1 - d_2 = 2 \cdot m \cdot (x_i + 1) - 2 \cdot y_i + 2 \cdot b - 1$$

- Se înlocuiește m cu dy/dx apoi se înmulțește în ambele părți cu dx . Rezultă:

$$t_i = (d_1 - d_2) \cdot dx = 2 \cdot dy \cdot (x_i + 1) - 2 \cdot dx \cdot y_i + 2 \cdot b \cdot dx - dx =$$

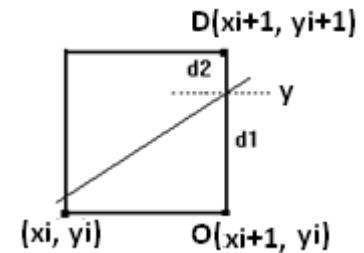
$$2 \cdot dy \cdot x_i - 2 \cdot dx \cdot y_i + 2 \cdot b \cdot dx - dx + 2 \cdot dy$$

- t_i reprezintă eroarea de aproximare în pasul i : pe baza sa se alege urmatorul punct al spațiului discret

Notăm cu (x_{i+1}, y_{i+1}) punctul care se va alege în pasul curent.

- Expresia erorii de aproximare pentru pasul următor este:

$$t_{i+1} = 2 \cdot dy \cdot x_{i+1} - 2 \cdot dx \cdot y_{i+1} + 2 \cdot b \cdot dx - dx + 2 \cdot dy$$



Algoritmul Bresenham(3)

$$t_i = (d_1 - d_2) * dx \quad (dx > 0)$$

(1) Dacă $t_i \leq 0$, se alege punctul O, deci $x_{i+1} = x_i + 1$ și $y_{i+1} = y_i$

Rezultă:

$$t_{i+1} = 2 * dy * (x_i + 1) - 2 * dx * y_i + 2 * b * dx - dx + 2 * dy$$

sau $\mathbf{t_{i+1} = t_i + 2 * dy}$

(2) Dacă $t_i > 0$, se alege punctul D, deci $x_{i+1} = x_i + 1$ și $y_{i+1} = y_i + 1$

Rezultă:

$$t_{i+1} = 2 * dy * (x_i + 1) - 2 * dx * (y_i + 1) + 2 * b * dx - dx + 2 * dy$$

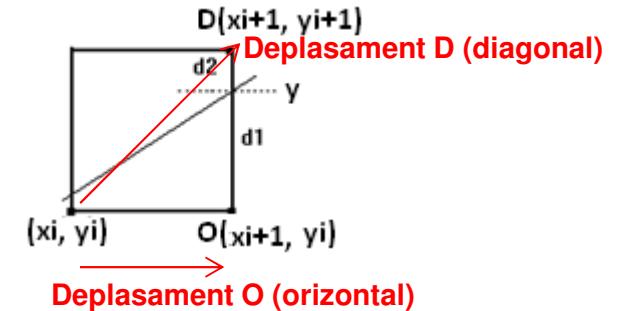
sau $\mathbf{t_{i+1} = t_i + 2 * dy - 2 * dx}$

➤ Valoarea variabilei de test se obține prin calcul incremental: adunarea unei constante întregi

Eroarea de aproximare pentru primul pas: $x_i = x_1$ și $y_i = y_1$:

$$t_1 = 2 * dy * x_1 - 2 * dx * y_1 + 2 * dy - dx + 2 * dx(y_1 - (dy/dx) * x_1)$$

sau $\mathbf{t_1 = 2 * dy - dx}$



Algoritmul Bresenham(4)

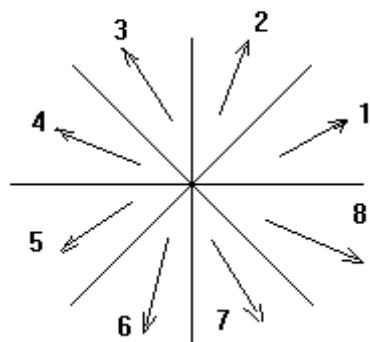
Implementare in C:

```
void Bres_vect(int x1, int y1, int x2, int y2, int culoare)
{ //pentru vectori cu panta cuprinsă între 0 și 1
    int dx, c1, c2, x, y, t;
    dx=x2-x1;
    c1=(y2-y1)<<1; // 2*dy
    c2=c1-(dx<<1); // 2*dy – 2*dx
    t=c1-dx; // 2*dy - dx
    putpixel(x1, y1, culoare);
    for(x=x1+1, y=y1; x<x2; x++)
    {if(t<0) t+=c1; //deplasament O
     else { t+=c2; y++;} // deplasament D
     putpixel(x,y,culoare);
    }
    putpixel(x2,y2,culoare);
}
```

Algoritmul Bresenham- generalizare(1)

Generalizarea algoritmului Bresenham pentru vectori de orice pantă

- Vectorii definiți în spațiul 2D pot fi clasificați, pe baza pantei, în opt clase geometrice, numite “octanți”
- Un vector care aparține unui octant O are 7 vectori simetrici în ceilalți 7 octanți



$$dx = x_2 - x_1 \text{ și } dy = y_2 - y_1$$

- octantul 1: $dx > 0$ și $dy > 0$ și $dx \geq dy$;
- octantul 2: $dx > 0$ și $dy > 0$ și $dx < dy$;
- octantul 3: $dx < 0$ și $dy > 0$ și $\text{abs}(dx) < dy$;
- octantul 4: $dx < 0$ și $dy > 0$ și $\text{abs}(dx) \geq dy$;
- octantul 5: $dx < 0$ și $dy < 0$ și $\text{abs}(dx) \geq \text{abs}(dy)$;
- octantul 6: $dx < 0$ și $dy < 0$ și $\text{abs}(dx) < \text{abs}(dy)$;
- octantul 7: $dx > 0$ și $dy < 0$ și $dx < \text{abs}(dy)$;
- octantul 8: $dx > 0$ și $dy < 0$ și $dx \geq \text{abs}(dy)$;

Algoritmul Bresenham -generalizare(2)

Notam cu:

+h, deplasamentul orizontal spre dreapta (în sensul crescător al axei x),

-h, deplasamentul orizontal spre stânga (în sensul descrescător al axei x),

+v, deplasamentul vertical în sus (în sensul crescător al axei y),

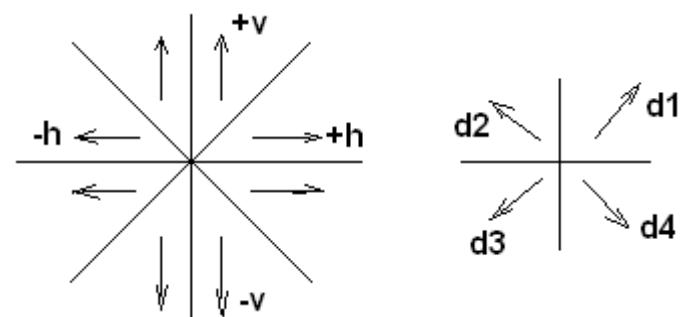
-v, deplasamentul vertical în jos (în sensul descrescător al axei y),

d1, deplasamentul diagonal dreapta-sus,

d2, deplasamentul diagonal stânga-sus,

d3, deplasamentul diagonal stânga-jos,

d4, deplasamentul diagonal dreapta-jos.



Octant	1	2	3	4	5	6	7	8
Deplas O	+h	+v	+v	-h	-h	-v	-v	+h
Deplas D	d1	d1	d2	d2	d3	d3	d4	d4

Corespondența între alegerea curentă într-un pas al algoritmului Bresenham (punctul O sau punctul D) și deplasamentul echivalent în fiecare octant:

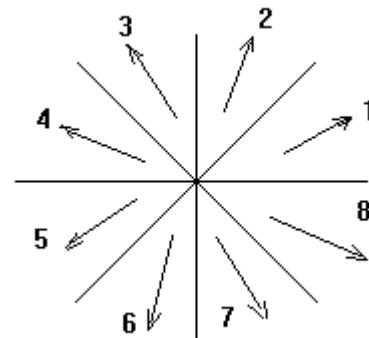
Algoritmul Bresenham - generalizare(3)

Algoritmul Bresemham generalizat -implementare in C

```
void Bres_general(int x1, int y1, int x2, int y2, int culoare)
{ int x, y, i, oct, dx, dy, absdx, absdy, c1, c2, t;
  if(x1==x2)
    //vertical
  {if(y1>y2){y=y1; y1=y2; y2=y;}
   for(y=y1; y<=y2;y++)
     putpixel(x1,y,culoare);
   return;
  }
  if(y1==y2)
    //orizontal
  {if(x1>x2) {x=x1; x1=x2; x2=x;}
   for(x=x1; x<= x2; x++)
     putpixel(x,y1,culoare);
   return;
  }
```

Algoritmul Bresenham - generalizare (4)

```
dx=x2-x1; dy=y2-y1;  
absdx=abs(dx); absdy=abs(dy);  
if(dx>0)//oct=1,2,7,8  
{if(dy>0)// oct=1,2  
 if(dx>=dy) oct=1; else oct=2;  
 else  
 if(dx>=absdy) oct=8; else oct=7;  
}  
else//3,4,5,6  
{if(dy>0)// oct=3,4  
 if(absdx>=dy) oct=4; else oct=3;  
 else  
 if(absdx>=absdy) oct=5; else oct=6;  
}  
// Numărul de pași la execuția algoritmului generalizat este maxim(abs(dx), abs(dy))  
// Dacă abs(dy) > abs(dx), se inversează rolul variabilelor dx și dy în calculul constantelor c1 și c2  
if(absdy>absdx) // adresele de pe traseul vectorului se obțin prin incrementarea lui y  
 {x=absdx; absdx=absdy;absdy=x;}  
c1=absdy<<1; c2=c1-(absdx<<1);  
t=c1-absdx;
```



```

putpixel(x1,y1,culoare);
for(i=1,x=x1,y=y1; i<absdx; i++)
{ if(t<0) // deplasament O
  {t+=c1;
  switch(oct)
  {case 1: case 8:x++; break; // +h
  case 4: case 5:x--;break; // -h
  case 2: case 3:y++;break; //+v
  case 6: case 7:y--;break; // -v
  }
  }
else
{t+=c2 // deplasament D
switch(oct)
{case 1: case 2: x++;y++;break; // d1
  case 3: case 4: x--;y++;break; // d2
  case 5: case 6: x--;y--;break; // d3
  case 7: case 8: x++;y--;break; // d4
  }
}
putpixel(x,y,culoare);
} putpixel(x2,y2,culoare);
}

```

Algoritmul Bresenham - generalizare (5)

Generarea liniilor intrerupte

```
void Bres_gen(int x1, int y1, int x2, int y2, int şablon)
{ // şablon: intreg pe 16 biti
    int val, bit, biţi[16] ;
    .....
    for(int i=0, val=1; i<16; i++)
    {   biţi[i] = şablon & val;
        val = val << 1;
    }
    .....
    if(biţi[0]) putpixel(x1,y1,culoare);
    for(i=1,x=x1,y=y1, bit=1; i<absdx; i++)
    { .....
        if(biţi[(bit++) % 16]) putpixel(x,y,culoare);
    }
    if(biţi[bit % 16]) putpixel(x2,y2,culoare);
}
```

Generarea suprafețelor definite prin contur sau prin interior

Prof. univ. dr. ing. Florica Moldoveanu

Suprafete definite prin contur sau interior

Suprafetele definite prin contur sau interior sunt suprafete oarecare.

(1) Suprafete definite prin contur

- se cunoaște culoarea pixelilor care alcătuiesc conturul suprafetei și un punct interior;
- la generarea suprafetei conturul trebuie să fie inscris în memoria imagine;
- pornind din punctul interior cunoscut, se modifică culoarea

tuturor pixelilor interiori conturului în culoarea și conform şablonului care sunt date.

(2) Suprafete definite prin interior

- se cunoaște culoarea pixelilor interiori suprafetei și un punct interior;
- la generarea suprafetei pixelii săi trebuie să aibă culoarea data (să fie încrucișati în memoria imagine);
- pornind din punctul interior cunoscut, se modifică culoarea tuturor pixelilor suprafetei în culoarea și cu şablonul care sunt date.

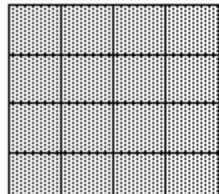
Algoritmi flood fill (1)

Există două tipuri de algoritmi folosiți la generarea suprafețelor definite prin contur și a celor definite prin interior:

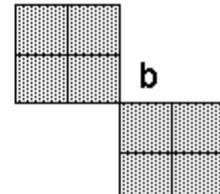
- (1) Algoritmi recursivi, bazați pe verificarea culorii punctelor vecine punctului curent (algoritmi “**flood fill**”) și modificarea culorii lor pana la atingerea conturului;
- (2) Algoritmi bazați pe parcurgerea liniilor raster care traversează suprafața.

În algoritmii **flood fill** se tine cont de tipul suprafeței: **conexă de ordin 4 sau conexă de ordin 8**:

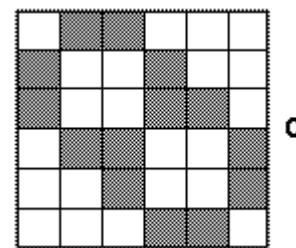
- Fiecare pixel al unei suprafețe conexe de ordinul 4 poate fi atins, pornind dintr-un punct interior, printr-o combinație de deplasări în numai patru direcții: stânga, dreapta, sus, jos (suprafața **a** din figura).
- Un pixel al unei suprafețe conexe de ordinul 8 poate fi atins printr-o combinație de deplasări orizontale, verticale și diagonale (suprafețele **b** și **c** din figura)



Suprafețe definite prin interior



b



Suprafața definită prin contur

Algoritm flood fill (2)

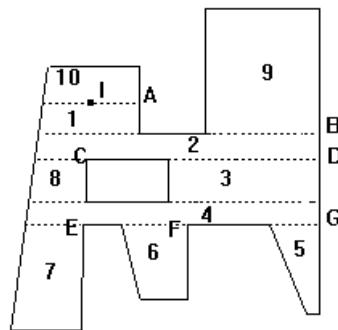
Functie de colorare a unei suprafete conexe de ordin 4, definita prin contur.

```
void sup_conex_4(int x, int y, int cul_interior, int cul_contur)
{
    int culoare = getpixel(x,y); // se citeste culoarea pixelului (x,y) din memoria imagine
    if(culoare != cul_contur && culoare != cul_interior)
    {
        putpixel(x,y,cul_interior); // scrie cul_interior pt pixelul (x,y)
        sup_conex_4(x+1, y, cul_interior, cul_contur);
        sup_conex_4(x -1, y, cul_interior, cul_contur);
        sup_conex_4(x, y+1, cul_interior, cul_contur);
        sup_conex_4(x, y-1, cul_interior, cul_contur);
    }
}
```

- ❖ Algoritmul necesită un spațiu mare de memorare pentru stiva program.
- ❖ De aceea, pentru generarea suprafețelor definite prin contur sau prin interior sunt preferați algoritmii bazați pe parcurserea liniilor imagine care traversează suprafața.

Generarea unei suprafete prin parcurgerea liniilor raster(1)

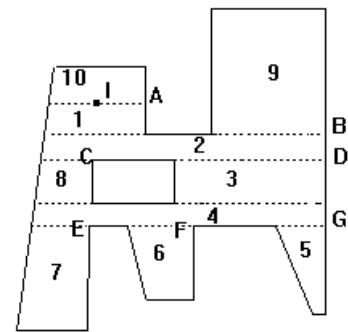
- Consideram suprafața definită prin contur.
- O suprafață poate fi convexă sau concavă și poate avea găuri.
- Deci, o linie orizontală care o traversează poate întâlni conturul suprafetei de mai multe ori:



- Un grup compact de pixeli situați pe aceeași linie raster, care nu au culoarea conturului și nici culoarea de umplere, formează un interval; de exemplu, E, F și G sunt extremitățile din dreapta a trei intervale de pe aceeași linie raster.
- Fie $I(x_i, y_i)$ punctul interior dat. El este primul punct de start pentru colorarea suprafetei.
- Punctele de start pentru colorare se memorează într-o stivă.

Generarea unei suprafete prin parcurgerea liniilor raster(2)

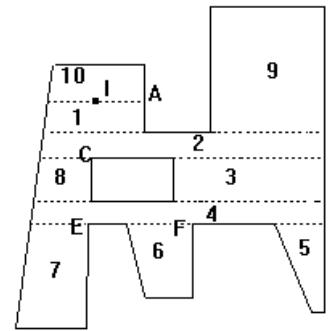
- Se introduce în stivă punctul interior dat, I.
- Cât timp stiva nu este vida:
 - (1) Se extrage din vârful stivei punctul de start pentru colorare
 - (2) Pornind din punctul de start, se colorează toți pixelii din dreapta să până când se ajunge la contur, apoi toți pixelii din stânga să până când se ajunge la contur. Se memorează extremitatea dreaptă a intervalului colorat, xmax, și extremitatea sa stângă, xmin.
 - (3) Se examinează pixelii din intervalul $xmin < x < xmax$ de pe linia de deasupra celei curente, pentru a se determina dacă conține numai pixeli de contur sau pixeli deja colorați. Intervalul poate conține mai multe subintervale. Se memorează extremitatea din dreapta a fiecărui subinterval, în stivă.
- Aceeași prelucrare se execută pentru linia de sub cea curentă.
- Deoarece ultimul punct introdus în stivă este de pe linia de sub cea curentă, prelucrarea se continua cu această linie.



Generarea unei suprafete prin parcurgerea liniilor raster(3)

Continutul stivei:

1. $I(x_i, y_i)$; se extrage I din stiva;
2. Dupa parcurgerea liniei $y = y_l$ si procesarea liniei de deasupra si a celei de dedesupră:
se adauga $A(x_A, y_A+1), (x_A, y_A-1); (x_A, y_A-1)$ se extrage in urmatoarea iteratie
3. Dupa parcurgerea fiecarei linii din zona 1 (liniile de deasupra sunt deja colorate): se adauga (x_A, y) care se extrage in urmatoarea iteratie
4. Dupa parcurgerea primei linii din zona 2: **A, B**
5. Dupa parcurgerea celorlalte linii din zona 2: **A, (xB, y)**; la ultima linie: **A, B,C,D**
6. Dupa parcurgerea fiecarei linii din zona 3: **A, B, C, (xD, y)**
7. Dupa parcurgerea fiecarei linii din zona 4: **A, B, C, (xD, y)**; la ultima linie: **A, B, C, E, F, G**
8. Dupa parcurgerea fiecarei linii din zona 5: **A, B, C, E, F, (xG, y)**



Generarea unei suprafete prin parcurgerea liniilor raster(4)

- Colorarea poligonului din figura începe din punctul I.

- După colorarea liniei $y=y_I$, vor fi colorate liniile din zona 1.

- La prima linie din zona 2 se schimbă x_{max} ; la parcurgerea liniei de deasupra sa se memoreaza B

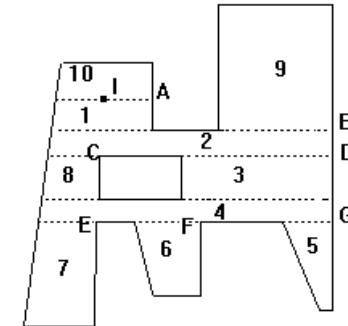
- Atunci când se examinează pixelii de pe linia ce conține latura de sus a

dreptunghiului interior poligonului se vor memora în stivă punctul C și apoi punctul D.

- Punctul D este extras imediat din stivă, devenind punct de start. Parcurgând linia $y=y_D$, spre stânga, se va întâlni latura din dreapta a dreptunghiului, modificându-se x_{min} .

- Colorarea se continuă cu liniile din zona 3, apoi cu cele din zona 4 și din zona 5. Când se ajunge la limita de jos a zonei 5, colorarea nu poate continua nici pe linia de deasupra nici pe cea de sub ea: nu se adauga punct în stivă

- În iterată urmatoare punctul din vârful stivei este F, de aceea colorarea se continuă cu zona 6, și aşa mai departe.



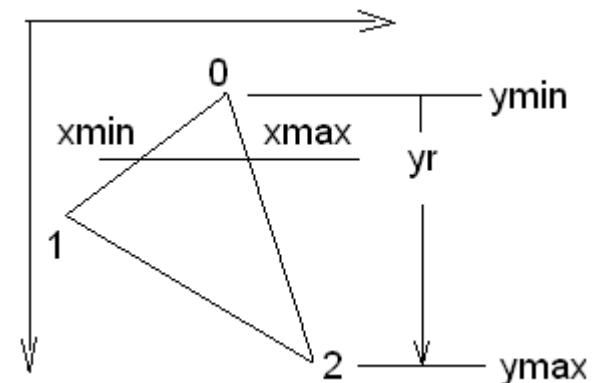
Rasterizarea suprafețelor triunghiulare

Prof. univ. dr. ing. Florica Moldoveanu

Rasterizarea suprafetelor triunghiulare-algoritm

Bool SupTri(int x0, int y0, int x1, int y1, int x2, int y2)

- { 1. Se sorteaza varfurile triunghiului crescator dupa
coordonata y, a.i. $y_0 = y_{\min}$;
 - 2. Daca triunghiul este degenerat (varfurile sunt coliniare), return false;
 - 3. Se orienteaza conturul in sens trigonometric (0-1-2: sens trigonometric);
 - 4. Se calculeaza extremitatile segmentelor interioare suprafetei triunghiului, pentru toate
liniile raster de la y_{\min} -triunghi la y_{\max} -triunghi si se memoreaza in 2 tablouri, XMIN, XMAX;
 - 5. Se afiseaza pixelii interiori triunghiului (intre x_{\min} si x_{\max} pe fiecare linie raster);
return true;
- }



Algoritmul: pasii 1 si 2

1. Se sorteaza varfurile triunghiului crescator dupa coordonata y, a.i. $y_0 = y_{\min}$

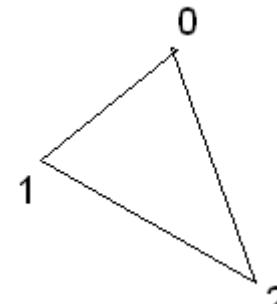
```
if(  $y_1 < y_0$ ) { t=  $y_1$ ;  $y_1 = y_0$ ;  $y_0 = t$  ;}
```

```
if(  $y_2 < y_0$ ) { t=  $y_2$ ;  $y_2 = y_0$ ;  $y_0 = t$  ;}
```

2. Daca triunghiul este degenerat (varfurile sunt coliniare), return false

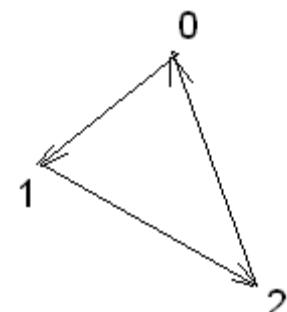
$$\begin{aligned} \det &= \begin{vmatrix} x_1 - x_0 & y_1 - y_0 \\ x_2 - x_0 & y_2 - y_0 \end{vmatrix} \\ &= (x_1 - x_0)(y_2 - y_0) - (x_2 - x_0)(y_1 - y_0) \end{aligned}$$

```
if( $\det == 0$ ) return false
```



Pasul 3: orientarea conturului

3. Se orienteaza conturul in sens trigonometric (0-1-2: sens trigonometric)



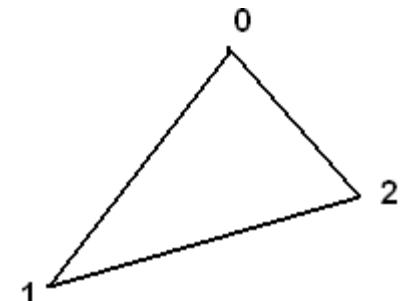
sens trigonometric

$$dx_0 = x_1 - x_0; \quad dx_1 = x_2 - x_0; \quad dy_0 = y_1 - y_0; \quad dy_1 = y_2 - y_0;$$

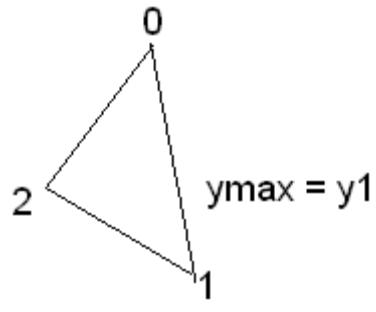
$$x_1 < x_0, \quad x_2 > x_0$$

$$dx_0 < 0, \quad dx_1 > 0, \quad dy_0 > 0, \quad dy_1 > 0$$

$$\det = dx_0 * dy_1 - dx_1 * dy_0 < 0$$



sens trigonometric



sens anti-trigonometric

$$x_1 > x_0, \quad x_2 < x_0$$

$$dx_0 > 0, \quad dx_1 < 0, \quad dy_0 > 0, \quad dy_1 > 0$$

$$\det = dx_0 * dy_1 - dx_1 * dy_0 > 0$$

Se inverseaza varful 1 cu 2

Pasul 4: calculul extremitatilor segmentelor interioare(1)

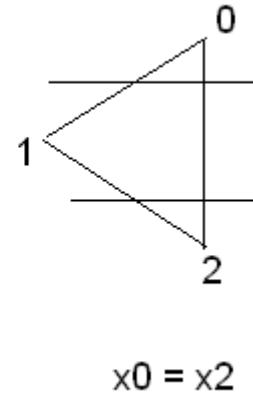
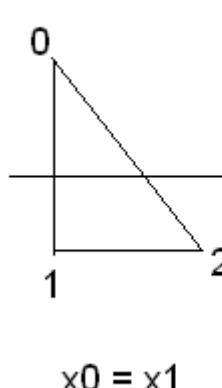
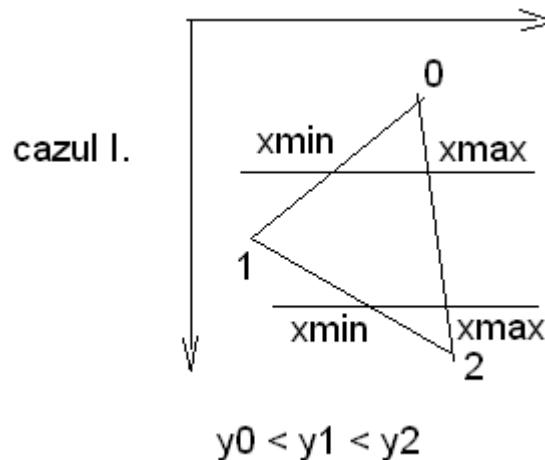
4. Se calculeaza extremitatile segmentelor interioare suprafetei triunghiului

- Coordonatele x_{min} , x_{max} ale extremitatilor segmentelor interioare suprafetei triunghiului se memoreaza in 2 tablouri:

```
int XMIN[H], XMAX[H]; // H este numarul de linii imagine
```

- Calculul extremitatilor este efectuat in functiile **CalculXmin()** si **CalculXmax()**;

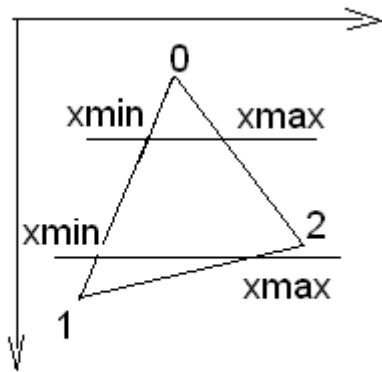
Cazuri:



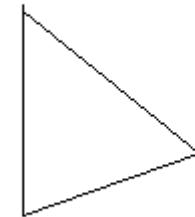
CalculXmin(x_0, y_0, x_1, y_1);
CalculXmin(x_1, y_1, x_2, y_2);
CalculXmax(x_0, y_0, x_2, y_2);

Pasul 4: calculul extremitatilor segmentelor interioare (2)

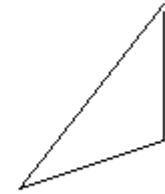
cazul II.



$$y_0 < y_2 < y_1$$



$$x_0 = x_1$$



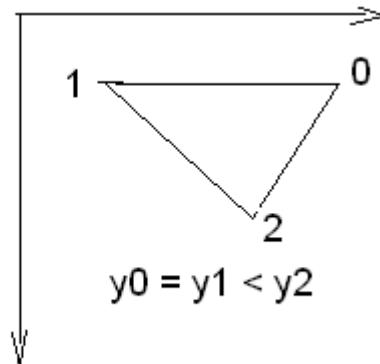
$$x_0 = x_2$$

`CalculXmin(x0, y0, x1, y1);`

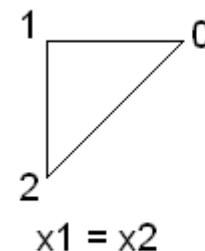
`CalculXmax(x0, y0, x2, y2);`

`CalculXmax(x2, y2, x1, y1);`

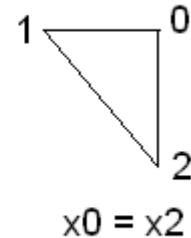
cazul III.



$$y_0 = y_1 < y_2$$



$$x_1 = x_2$$

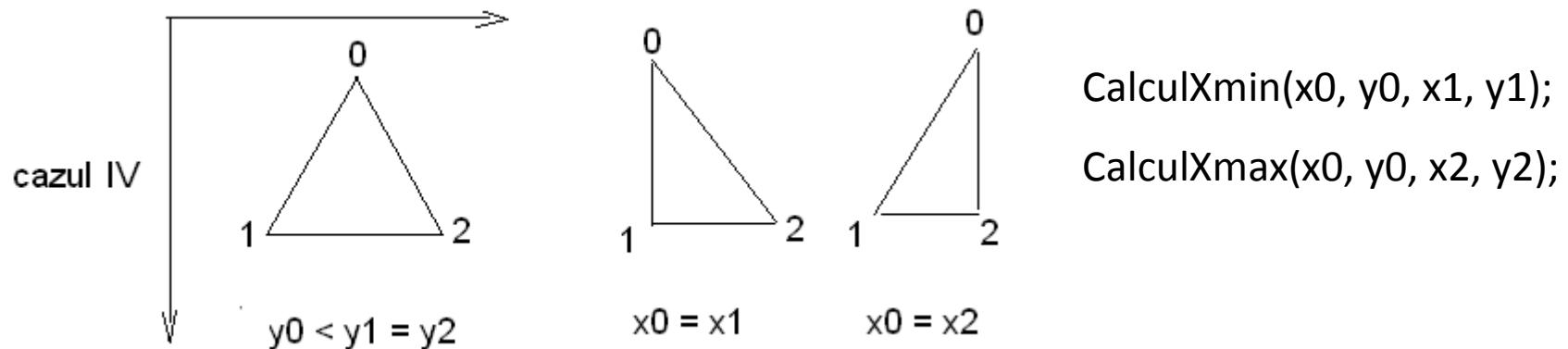


$$x_0 = x_2$$

`CalculXmin(x1, y1, x2, y2);`

`CalculXmax(x0, y0, x2, y2);`

Pasul 4: calculul extremitatilor segmentelor interioare (3)



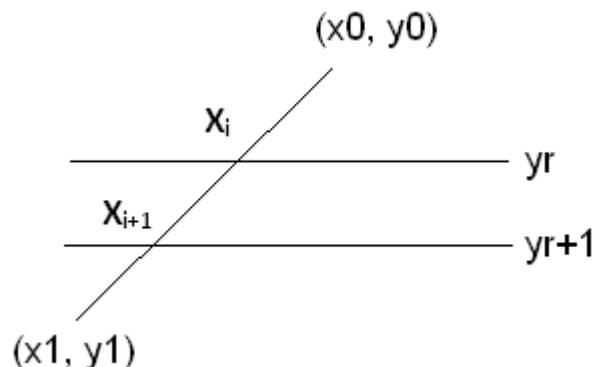
Calculul coordonatelor x_{\min} , x_{\max}

$$m = (y_1 - y_0) / (x_1 - x_0) \quad - \text{panta laturii}$$

$$(y_{r+1} - y_r) / (x_{i+1} - x_i) = m \rightarrow x_{i+1} = x_i + (1/m)$$

$dx = 1.0/m$ - constanta a laturii

$$x_{i+1} = x_i + dx \rightarrow \text{calcul incremental}$$



Pasul 4: calculul extremitatilor segmentelor interioare (4)

```
int XMIN[H], XMAX[H];  
  
void Calcul Xmin(int x0, int y0, int x1, int y1)  
{ // y0 < y1  
    int y;  
    if( x0==x1)  
        {for(y =y0; y<=y1;y++)  
            XMIN[y] = x0;  
        return;  
    }  
    float dx = (float)(x1 - x0) / (y1 - y0);  
    XMIN[y0] = x0; XMIN[y1] = x1;  
    for(y = y0 +1; y<y1; y++)  
        XMIN[y] = XMIN[y-1] + dx;  
}
```

void CalculXmax(int x0, int y0, int x1, int y1)
{ - similara cu functia CalculXmin
 - memoreaza punctele successive de pe latură
 in tabloul XMAX
}

Pasul 5: afisarea pixelilor suprafetei triunghiulare

5. Se afiseaza pixelii interiori triunghiului, intre xmin si xmax pe fiecare linie raster

```
ymax = y1; if (y2>y1 )ymax = y2;
```

```
for(int y= y0; y<= ymax; y++)
```

```
    for( int x = XMIN[y]; x <= XMAX[y]; x++)
```

```
        putpixel(x, y, culoare(x,y)); // afisare fragment rezultat din rasterizarea suprafetei  
                                    // triunghiulare
```

Functia **culoare (int x, int y)** calculeaza culoarea fragmentului care se afiseaza in pixelul (x,y), tinand cont de:

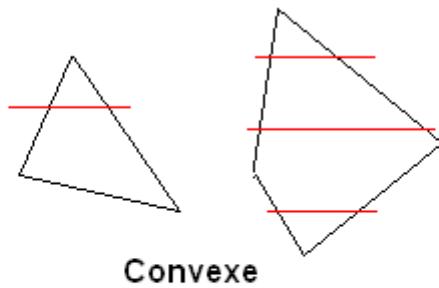
- sursele de lumina
- proprietatile de material ale suprafetei din care face parte triunghiul
- umbre (fragmentul poate fi in umbra)
- ceata

Functia **culoare (int x, int y)** este implementata in “fragment shader”

Rasterizarea suprafețelor poligonale

Prof. univ. dr. ing. Florica Moldoveanu

Rasterizarea suprafetelor poligonale (1)

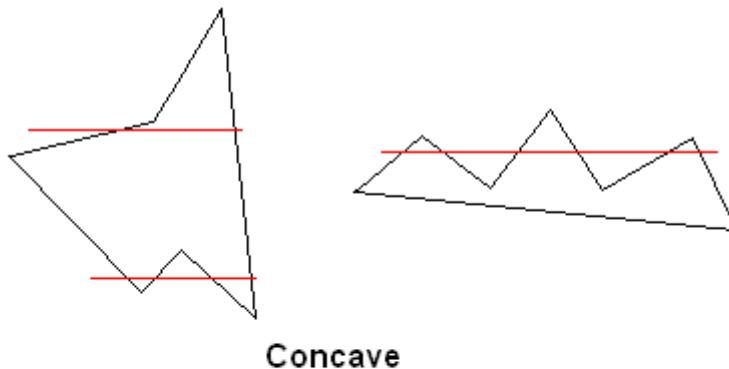


Convexe

Forma generala a algoritmului

pentru toate liniile raster care intersecteaza suprafata poligonului

- calculeaza punctele de intersectie dintre linia raster si laturile poligonului
- coloreaza pixelii interiori poligonului, de pe fiecare linie raster



Concave

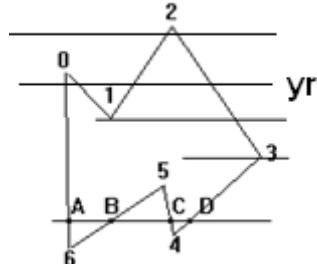
Probleme

- 1) O linie raster nu intersecteaza toate laturile poligonului.
- 2) Intersectia dintre o linie raster si un poligon concav produce mai multe segmente de pixeli interiori.



Autointersectant

Rasterizarea suprafețelor poligonale (2)

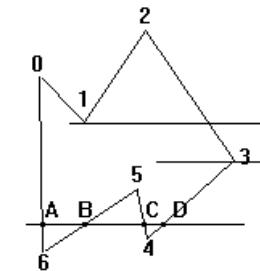


- 1) O linie raster nu intersecteaza toate laturile poligonului.
- se determina pentru fiecare linie raster setul laturilor intersectate:
setul laturilor active ($y_{min_latura} \leq yr < y_{max_latura}$)
- 2) Intersectia dintre o linie raster si un poligon concav produce mai multe segmente de pixeli interiori.
- Ordenează punctele de intersecție crescător după abscisă: x_1, x_2, x_3, x_4 . Exemplu: A, B, C, D.
 - Colorează pixelii de pe segmentele de dreaptă interioare suprafetei poligonului:
 $(x_1, yr) - (x_2, yr); (x_3 - y_3) - (x_4, yr)$. Exemplu: (A-B, C-D).
 - ❖ Punctele de pe segmentele $(x_1, yr) - (x_2, yr); (x_3 - y_3) - (x_4, yr)$, .., respecta **definitia punctului interior unui poligon**: "*un punct este interior unui poligon dacă orice semidreaptă dusă din punctul respectiv intersectează poligonul într-un număr impar de puncte*"

Rasterizarea suprafetelor poligonale (3)

❖ Cazuri particulare: liniile raster care traverseaza varfurile poligonului:

- daca fiecare varf apartine la 2 laturi, atunci, o semidreapta orizontala dusa dintr-un punct interior poligonului
 - va produce 3 puncte de intersectie daca traverseaza varful 1
 - va produce 2 puncte de intersectie daca traverseaza varful 3
- Varfurile 0, 1, 2, 4, 5, 6 sunt varfuri de minim / maxim local



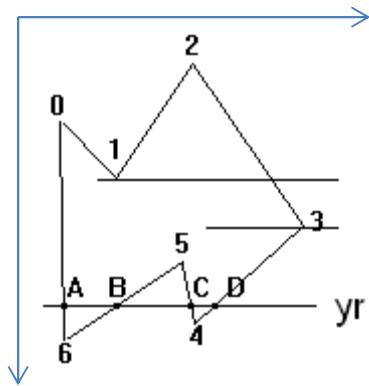
Conventie:

- la traversarea unui varf de minim / maxim local trebuie sa rezulte 0 sau 2 puncte de intersectie
- la traversarea oricarui alt varf trebuie sa rezulte un singur punct de intersectie

Implementarea conventiei: se considera fiecare latura un interval inchis la un capat si deschis la celalalt, de exemplu: [ymin_latura – ymax_latura); Atunci:

-un vîrf de maxim local nu aparține nici uneia dintre laturile care-l formează iar un vîrf de minim local aparține ambelor laturi.

Rasterizarea suprafețelor poligonale (4)



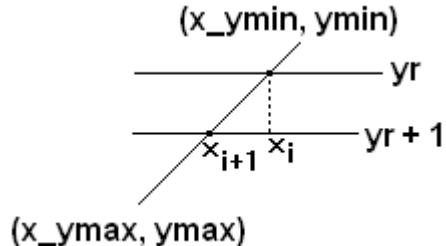
Algoritmul de rasterizare

- 1) Se crează lista laturilor poligonului, ordonată crescător după ordonatele minime ale laturilor: 1-2, 2-3, 0-1, 0-6, 3-4, 4-5, 5-6
- 2)

```
for( yr = ymin_poligon; yr < ymax_poligon; yr++ )
```

 - Determină setul laturilor active
 - Calculează intersecțiile dreptei $y=yr$ cu laturile active. De exemplu: A, D, C, B.
 - Ordenează punctele de intersecție crescător după abscisă: A, B, C, D.
 - Colorează pixelii de pe segmentele de dreaptă interioare suprafeței poligonului (A-B, C-D).

Rasterizarea suprafețelor poligonale (5)



Calculul intersectiilor liniilor raster cu laturile poligonului

$$m = (y_{max} - y_{min}) / (x_{ymax} - x_{ymin}) = (y_r + 1 - y_r) / (x_{i+1} - x_i)$$

$$x_{i+1} = x_i + 1/m \text{ sau } x_{i+1} = x_i + dx$$

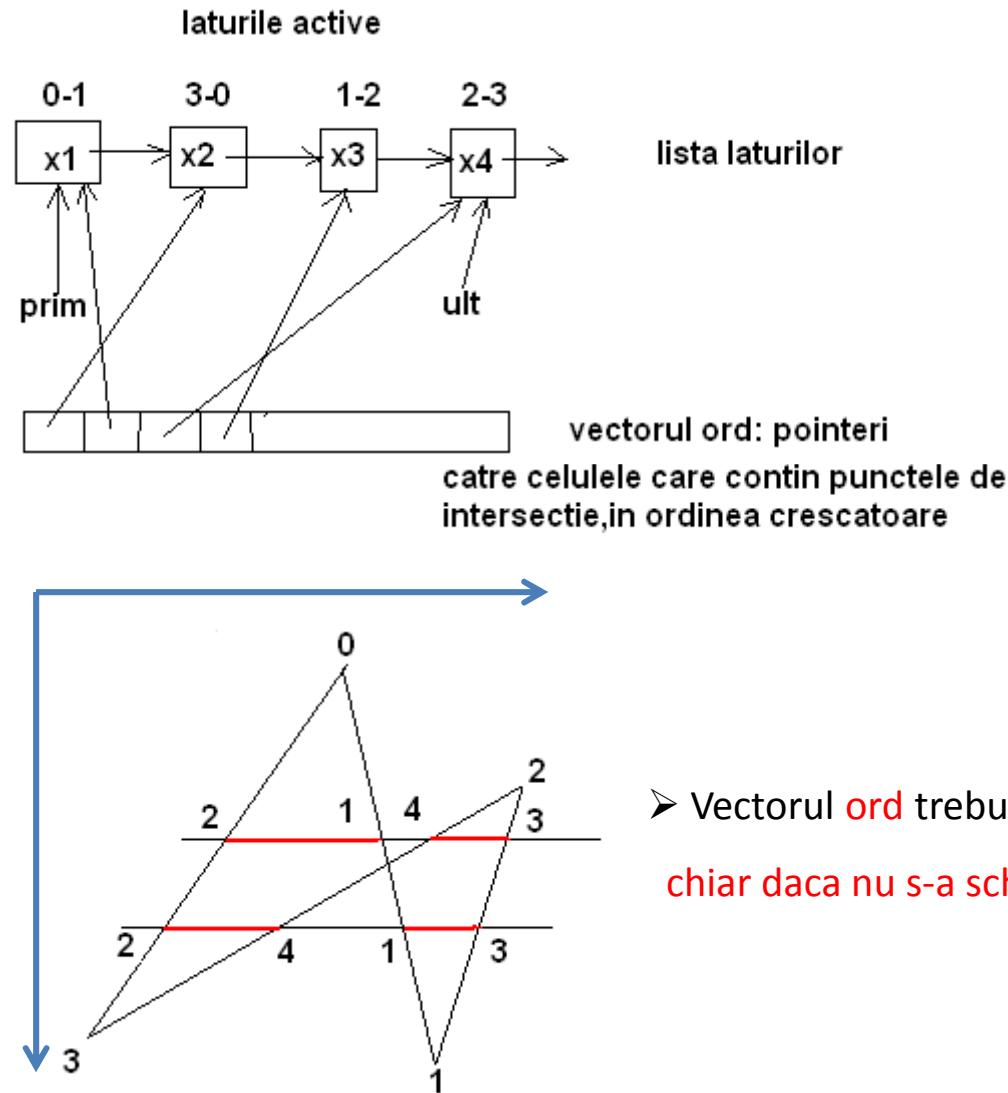
$$x_1 = x_{ymin} - dx \text{ (inainte ca latura sa devina activa)}$$

O implementare(1)

Reprezentarea unei laturi în lista de laturi:

- y_{max_latura} și y_{min_latura} , folosite în testul de latură activă;
- x_i - abscisa ultimului punct de intersecție al laturii; initial, x_i = abscisa varfului de $y_{min_latura} - dx$
- dx - constantă folosită în calculul incremental al absciselor punctelor de intersecție.

O implementare(2)



```

typedef struct{int x,y;}virf;

struct latura{ int ymax,ymin;
float xi,dx;
struct latura * urm;
};

typedef struct latura latura;

latura * prim,*ult;

latura **ord;

```

➤ Vectorul **ord** trebuie ordonat pentru fiecare linie raster,
chiar daca nu s-a schimbat setul laturilor active.

O implementare (3)

Determina lista laturilor active
pentru noua linie raster

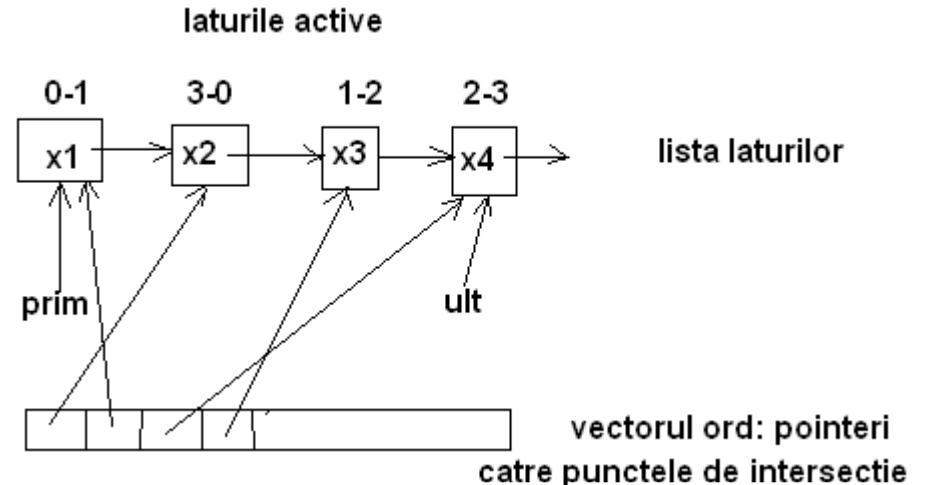
int set_activ()

{

* Adaugă laturile care devin active
(avans **ult** cat timp $yr \geq y_{min_latura}$)

* Elimină din lista laturilor, intre prim si ult, laturile care au devenit inactive
($yr > y_{max_latura}$)

return 1 daca s-a modificat setul laturilor active,
0 altfel;
}



O implementare (4)

```
int colorare_poligon(int nrV, virf poly[])
{
    // nrV este numarul de vârfuri ale poligonului; poly este vectorul vârfurilor
    latura **ord; latura * k; int l, k, modif, x;

    * Creaza lista de laturi ordonata crescator dupa ymin_laturi;
    ult = prim;

    // Genereaza interiorul poligonului prin intersectii cu linii raster, de la ymin_poligon la
    ymax_poligon

    for(yr=ymin_poligon; yr < ymax_poligon; yr++)
    {
        modif=set_activ(); // calculeaza setul laturilor active
        * Calculeaza punctele de intersecție cu laturile active (memorate in celulele listei de
        laturi): xi = xi + dx
        if(modif) // s-a modificat setul laturilor active
            * Memoreaza in vectorul ord pointeri catre laturile active
```

O implementare (5)

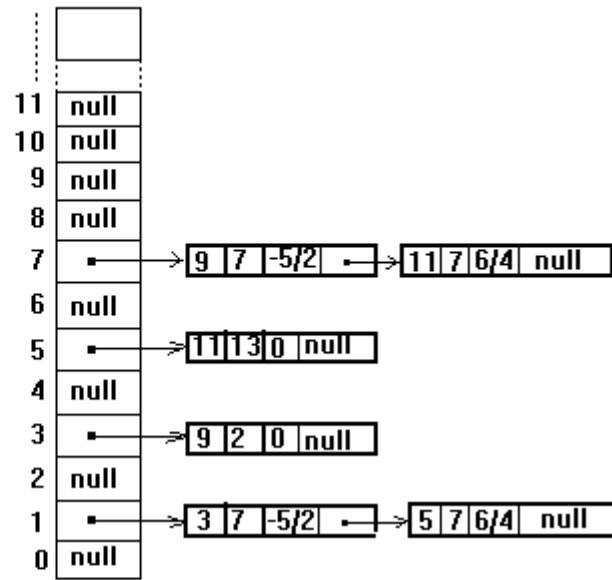
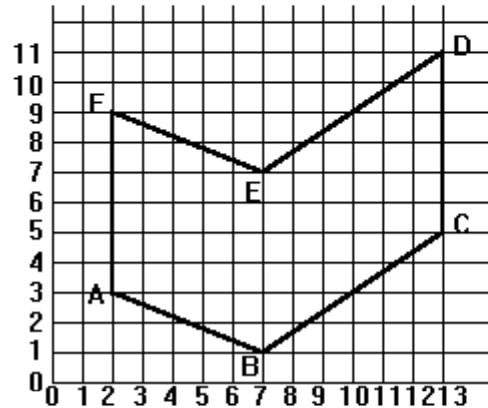
// Ordeneaza vectorul ord a.i. sa punteze catre punctele de intersecție in ordinea crescatoare a absciselor

```
do {o=0;  
    for(i=0; i<nrint-1; i++)  
        if(ord[i]->xi > ord[i+1]->xi) {k=ord[i]; ord[i]=ord[i+1]; ord[i+1]=k;o=1;}  
    }while(o);
```

//Coloreaza pixelii interiori poligonului de pe linia curenta

```
for(i=0; i<nrint-1; i+=2)// nrint este numarul de puncte de intersectie  
    for(x = (int)(ord[i]->xi+0.5); x<=(int)(ord[i+1]->xi+0.5); x++)  
        putpixel(x, yr, culoare(x,yr)); // afisare fragment  
    } //for yr  
return 1;
```

Varianta de reprezentare a listei de laturi(1)



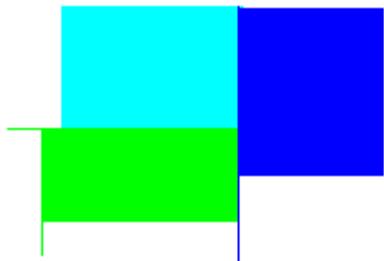
- Lista de laturi este un vector de pointeri cu un număr de intrări egal cu numărul de linii imagine, implicit ordonată crescător după ordonatele minime ale laturilor .
- Fiecare intrare punctează către lista laturilor care au ordonată minimă egală cu indicele intrării
- Lista laturilor cu același y_{min} este ordonată crescător după abscisa vârfului cu ordonata y_{max} .
- Pentru fiecare latură se memorează ordonata maximă , abscisa vârfului de ordonată minimă și incrementul ($1/m$) folosit în calculul punctelor de intersecție.

Varianta de reprezentare a listei de laturi(2)

- Lista laturilor active este reprezentată separat, printr-o listă înlățuită, în care laturile sunt memorate în ordinea crescătoare a absciselor punctelor de intersecție cu linia curentă.
- Având în vedere organizarea listei de laturi, timpul necesar construirii sale este mai mic decât în soluția prezentată anterior.
- Actualizarea listei de laturi active la trecerea de la o linie raster la alta poate fi efectuată mai rapid.
- Soluția este dezavantajoasă în privința spațiului de memorie ocupat de lista laturilor, știind că în majoritatea cazurilor numai un număr mic dintre intrările sale sunt folosite (în prezent acesta nu mai constituie un dezavantaj!).

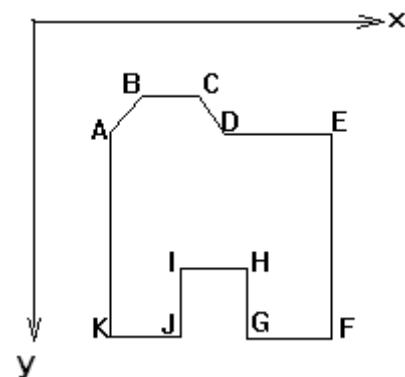
Generarea pixelilor strict interiori

Trebuie evitata suprascrierea pixelilor de pe laturile comune poligoanelor .



Conventie:

- Pixelii de pe laturile de x_{min} si y_{min} apartin poligonului (se afiseaza)
- Pixelii de pe laturile de x_{max} si y_{max} nu se afiseaza



Datorita conventiei folosite pentru traversarea varfurilor poligonului:

- Liniile orizontale care unesc 2 varfuri de minim local vor fi afisate
- Liniile orizontale care unesc 2 varfuri de maxim local nu vor fi afisate
 - In acest fel se respecta conventia

Generarea pixelilor strict interiori

Pentru punctele de xmin si xmax ale segmentelor interioare poligonului:

- fiecare segment interior poligonului este un interval inchis in punctul de abscisa minima si deschis in punctul de abscisa maxima, deci se respecta conventia.

```
for(x = (int)(ord[i]->xi+0.99999); x<=(int)(ord[i+1]->xi-0.000001); x++)  
    putpixel(x, yr, culoare(x,yr));
```

- ❖ Generarea numai a pixelilor strict interiori nu dă rezultate bune atunci când unghiul dintre două laturi este foarte mic. Este posibil ca între cele două laturi să fie afișat un singur pixel sau chiar niciunul.

Generarea interiorului unui poligon folosind un şablon

Modificarea implementarii anterioare pentru generarea interiorului folosind un sablon

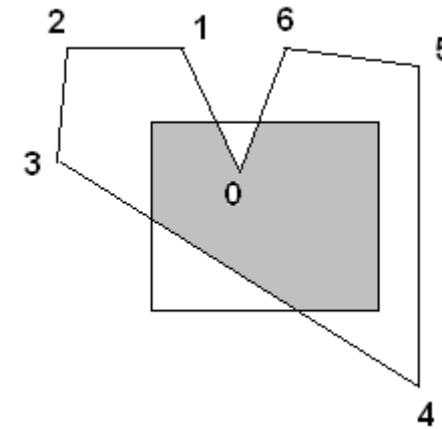
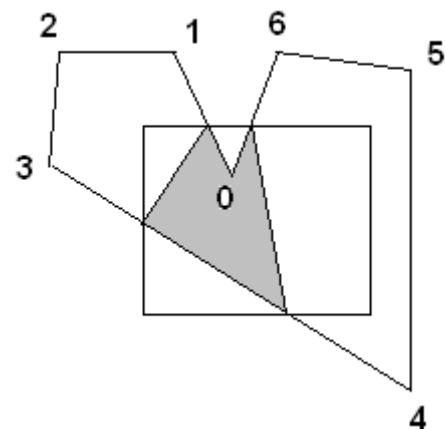
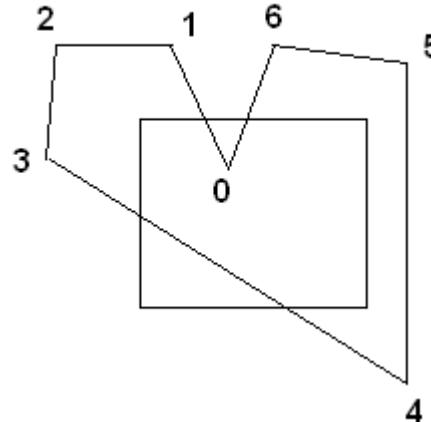
```
int sablon[8][8]={255,0,0,0,0,0,255, 0,255,0,0,0,255,0, 0,0,255,0,0,255,0,0,  
 0,0,0,255,255,0,0,0,0,0,255,255,0,0,0, 0,0,255,0,0,255,0,0,  
 0,255,0,0,0,255,0,255,0,0,0,0,0,0,255  
 }; //matrice de nivele de intensitate  
int hs=8, ls=8, lin, x;  
// sablonul se aplica incepand din (x=0, y=0): originea imaginii  
void colorare_poligon(int nrV, virf poly[] )  
{  
for(yr=ymin_poligon; yr < ymax_poligon; yr++)  
{ .....  
lin=yr%hs; // linia din matricea sablon folosita la colorare  
for(i=0;i<nrint-1;i+=2)  
 for(x=(int)(ord[i]->xi+0.5; x<=(int)(ord[i+1]->xi+0.5); x++)  
 putpixel(x,yr,sablon[lin][x%ls]);  
}
```

Cazul general: sablonul se mapeaza pe un dreptunghi dat, iar aplicarea sablonului consta in multiplicarea dreptunghiului sablon pe suprafata poligonului, incepand dintr-un punct dat.

Decuparea poligoanelor

Prof. univ. dr. ing. Florica Moldoveanu

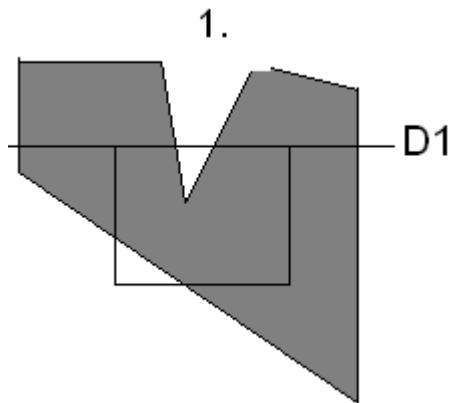
Decuparea poligoanelor 2D



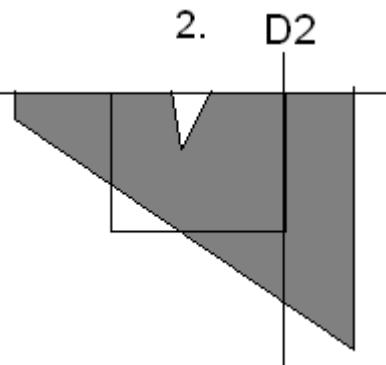
Rezultatul obtinut daca se decupeaza pe rand
laturile poligonului folosind algoritmul de decupare
vectori.

Rezultatul corect.

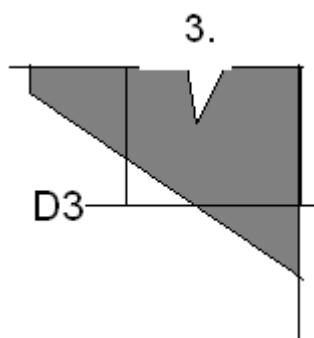
Algoritmul Sutherland-Hodgman(1)



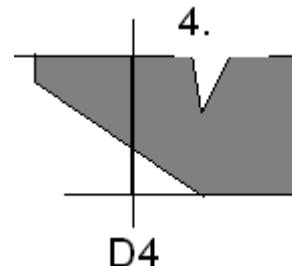
1. Se decupeaza laturile poligonului fata de dreapta pe care se afla o latura a dreptunghiului de decupare, D1
- Se creaza lista varfurilor poligonului rezultat, V1



2. Se decupeaza poligonul cu varfurile V1 fata de dreapta D2
- Se creaza lista varfurilor poligonului rezultat ,V2



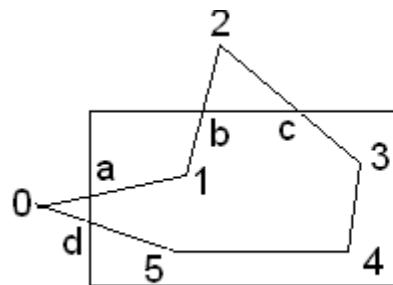
3. Se decupeaza poligonul cu varfurile V2 fata de dreapta D3
- Se creaza lista varfurilor poligonului rezultat, V3



4. Se decupeaza poligonul cu varfurile V3 fata de dreapta D4
- Se creaza lista varfurilor poligonului rezultat ,V4

Algoritmul Sutherland-Hodgman(2)

Calculul varfurilor poligonului rezultat intr-o etapa a decuparii

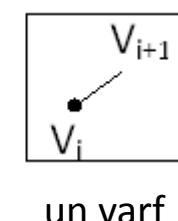
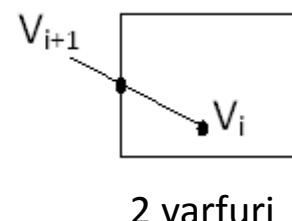
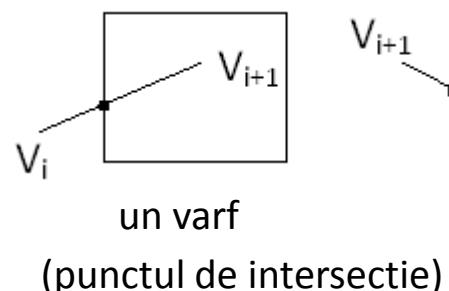
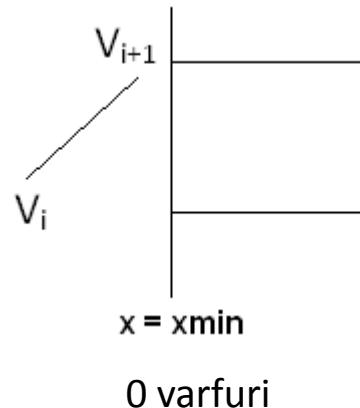


- 0-1: a, 1 --> a, 1
- 1-2: 1,b --> a, 1, 1, b
- 2-3: c,3 --> a, 1, 1, b, c, 3
- 3-4: 3,4 --> a,1, 1, b, c, 3, 3, 4
- 4-5: 4,5 --> a,1, 1, b, c, 3, 3, 4, 4, 5
- 5-0: 5,d --> a,1, 1, b, c, 3, 3, 4, 4, 5, 5, d

Lista corecta: a, 1, b, c, 3, 4, 5, d

Conventie:

- Fiecare latura a poligonului se considera un vector: $V_i - V_{i+1}$
- Numai varful V_i apartine laturii



Algoritmul Sutherland-Hodgman(3)

```
int DecupLatura(P2D v1, P2D v2, P2d * v1d, P2D *v2d, int latura)
// decupeaza o latura a poligonului, v1-v2, fata de dreapta pe care se afla o latura a drept. de decupare
{ // functia intoarce numarul de varfuri rezultate din decupare
    switch(latura)
        { case 1: // dreapta y = ymax
            if(v1.y > ymax && v2.y > ymax) return 0; // latura in afara drept. decupare
            if(v1.y <= ymax && v2.y <= ymax) { *v1d = v1; return1;} // latura in interiorul drept. decupare
            if(v1.y <= ymax && v2.y > ymax) / laturaiese din drept. de decupare
                { * v1d = v1; x_intersectie = ...; v2d->x = x_intersectie; v2d->y = ymax; return 2;}
            else// v1.y > ymax && v2.y <= ymax // latura intra in drept. de decupare
                { x_intersectie = ....; v1d->x = x_intersectie; v1d->y = ymax; return 1;}
        case 2: // latura x=xmax
            .....
        }
}
```

Algoritmul Sutherland-Hodgman(4)

```
void DecupLaturi(int nv, P2D *vrf, int *nd, P2D *vrfd, int latura)
{ // decupeaza toate laturile poligonului fata de dreapta pe care se afla o latura a dreptunghiului
  // vrf: lista varfurilor poligonului de decupat
  // nv: numarul de varfuri ale poligonului de decupat
  // vrfd: lista varfurilor poligonului rezultat din decupare
  // nd: numarul de varfuri ale poligonului rezultat din decupare

  *nd = 0;

  for(int i =0; i< nv; i++)
    *nd+= DecupLatura(vrf[i], vrf[i+1], &vrfd[*nd], &vrfd[*nd+1], latura)
    *nd+= DecupLatura(vrf[nv-1], vrf[0], &vrfd[*nd], &vrfd[*nd + 1], latura);

}
```

Algoritmul Sutherland-Hodgman(5)

```
void DecupPoligon(int nv, P2D * vrf, int *nd, P2D **vrfd)

{ // vrf: lista varfurilor poligonului de decupat

// vrfd: lista varfurilor poligonului rezultat din decupare

P2D * v1 = new P2D[2*nv]; P2D * v2 = new P2D[2*nv]; // se folosesc 2 liste de varfuri

memcpy(v1, vrf, nv*sizeof(P2D));

for(int latura =1 ; latura <= 4; latura+=2)

{ DecupLaturi(nv, v1, nd, v2, latura);

  DecupLaturi(*nd, v2, &nv, v1, latura+1);

}

*vrfd = new P2D[*nd];

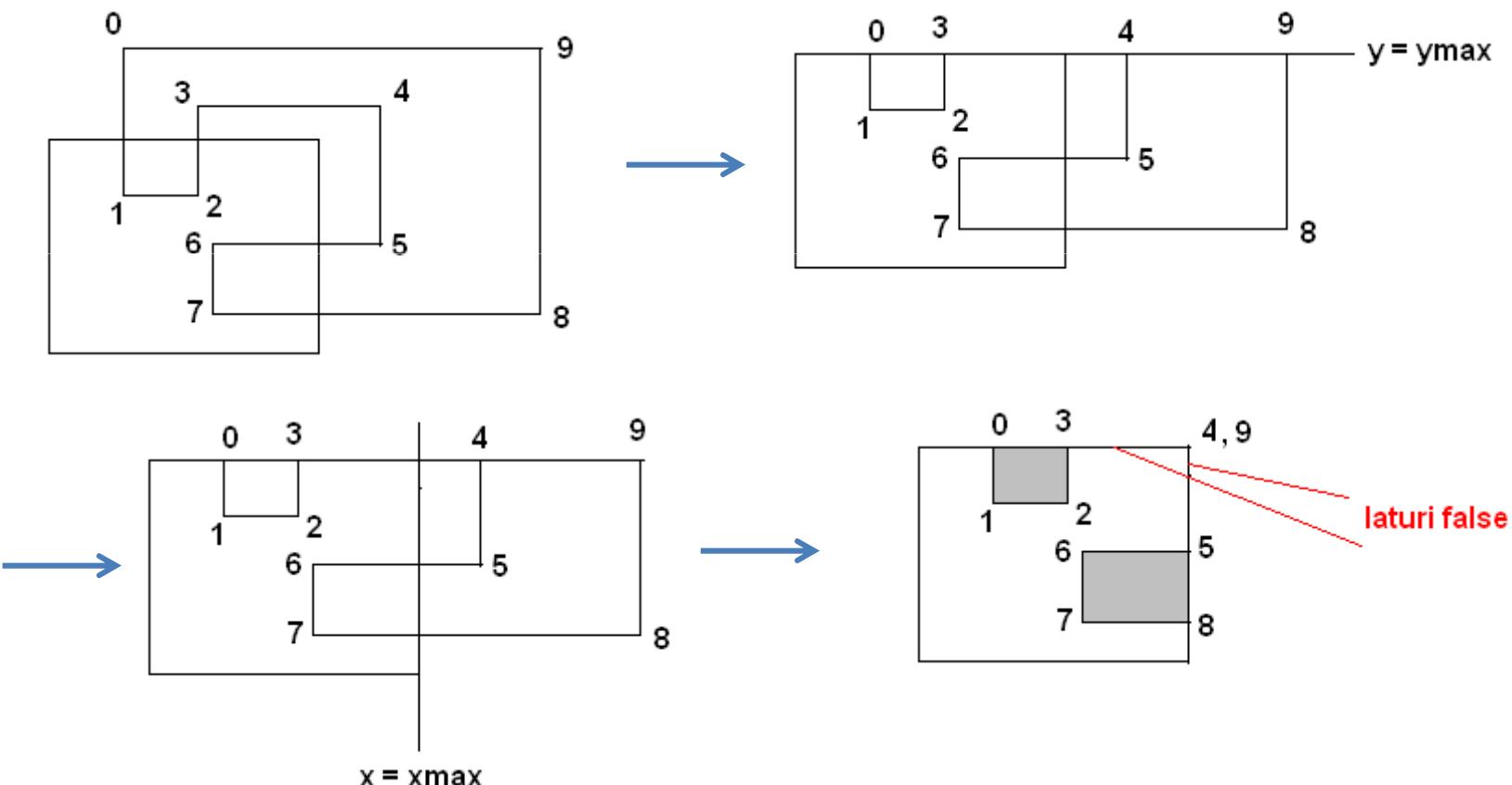
memcpy(*vrfd, v1, (*nd)*sizeof(P2D));

delete v1; delete v2;

}
```

Algoritmul Sutherland-Hodgman(6)

Limitarea algoritmului Sutherland-Hodgman: la decuparea poligoanelor concave, atunci cand din decupare pot rezulta mai multe poligoane. Algoritmul produce un singur poligon, cu posibile laturi false.



Algoritmul Weiler - Atherton(1)

Permite decuparea unui poligon oarecare (poligonul subiect) fata de un poligon oarecare (poligonul de decupare). Notam cu:

PS- poligonul subiect

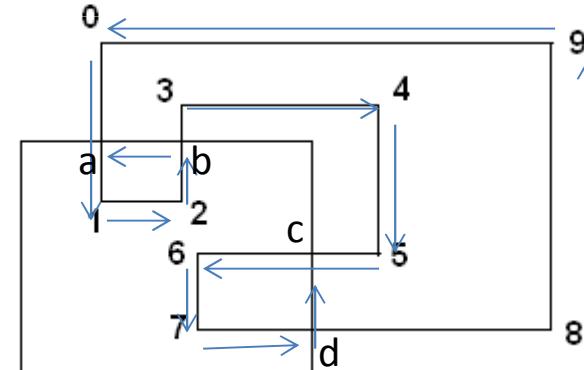
PD-poligonul de decupare

PR- poligonul rezultat – acesta poate fi alcătuit din mai multe cicluri de varfuri(poligoane)

LS – lista de varfuri a PS

LV - lista de varfuri a PR

Cicluri de varfuri: a, 1, 2, b, a; c, 6, 7, d, c



Algoritmul Weiler - Atherton(2)

Se initializeaza LV. Se alege un varf al PS exterior PD. Acesta este Vstart (varful din care incepe parcurgerea conturului poligonului PS).

Vcurrent <- Vstart; gata = false.

repeta

*Se parurge conturul poligonului PS incepand din Vcurrent, pana cand se ajunge in Vstart sau intr-un punct Pi de intersectie PS cu PD.

daca s-a ajuns in Vstart

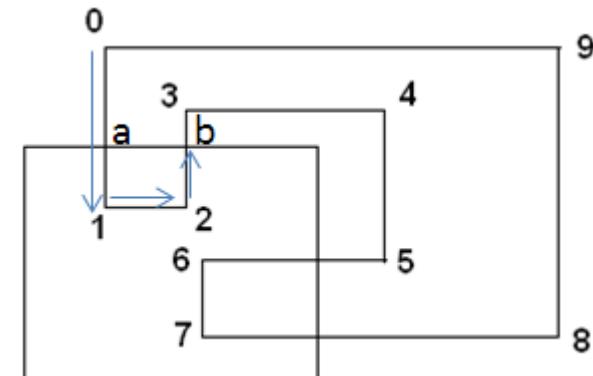
gata = true; break;

altfel

1. Se adauga Pi in LV si se continua

parcurgerea pe conturul PS pana intr-un

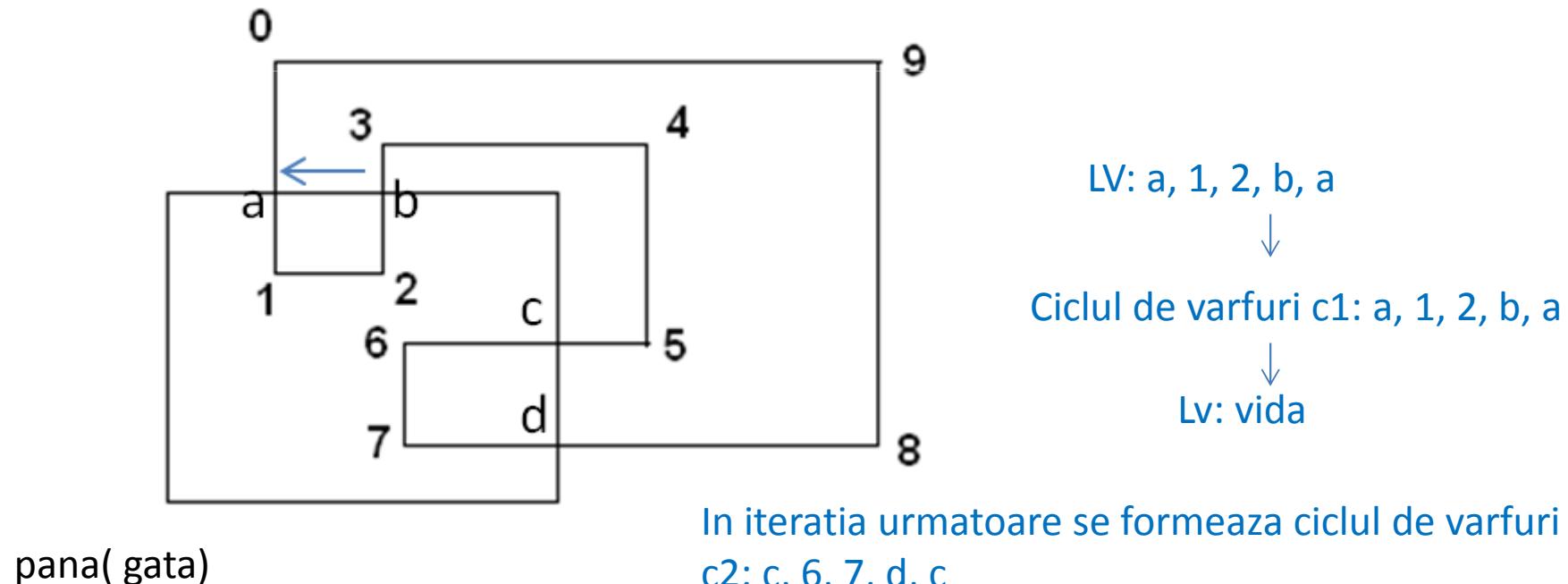
punct Pe de iesire din PD, memorand in LV varfurile PS intalnite pe parcurs si punctul Pe.



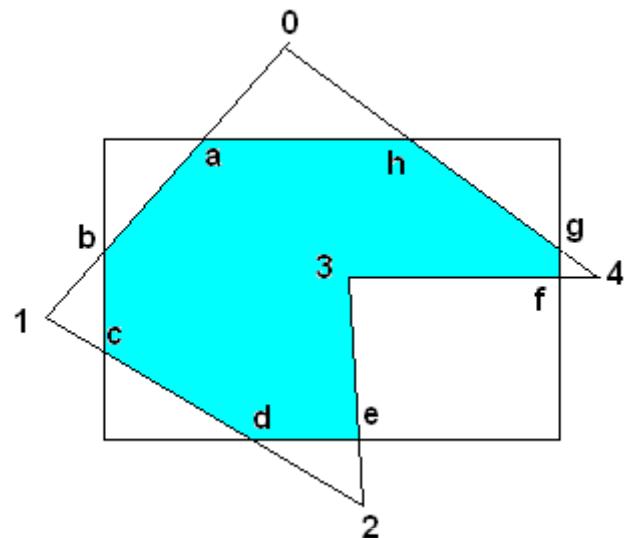
LV: a, 1, 2, b

Algoritmul Weiler - Atherton(3)

2. Vcurrent<-varful exterior al laturii PS care intersecteaza PD in Pe. // vrf. 3
3. Se continua parcurgerea pe conturul PD din Pe pana cand se ajunge intr-un punct, Pi, de intersectie cu PS. Se adauga in LV varfurile PD intalnite pe parcurs si punctul Pi.
4. Daca s-a format un ciclu de varfuri, se memoreaza ciclul de varfuri si se initializeaza LV. altfel, Vcurrent<-Pi (in inlocuieste in LS vrf. 3 cu Pi).



Algoritmul Weiler - Atherton(4)



Se porneste din varful 0

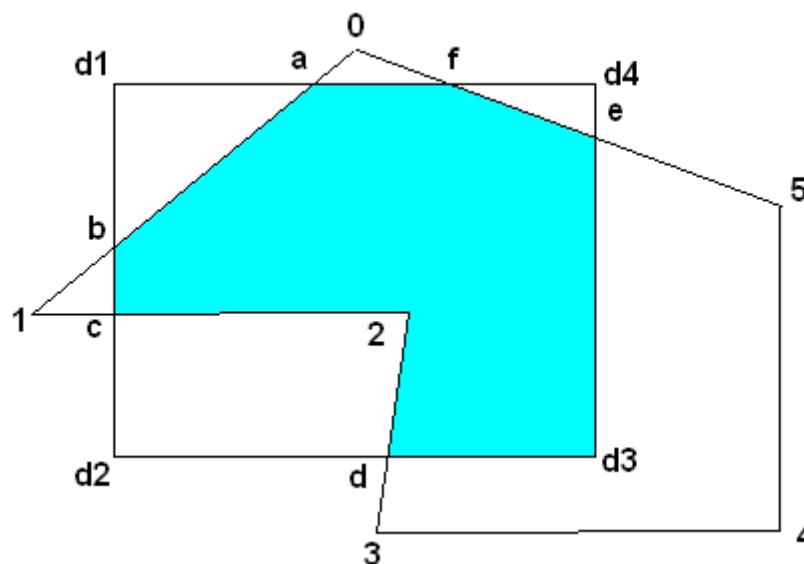
LV: a, b

a, b, c (se inlocuieste in LS vrf. 1 cu c)

a, b, c, d, e (se inlocuieste in LS vrf. 2 cu e)

a, b, c, d, e, 3, f, g (se inlocuieste in LS vrf. 4 cu g)

a, b, c, d, e, 3, f, g, h, a



Se porneste din varful 0

LV: a, b

a, b, c (se inlocuieste in LS vrf. 1 cu c)

a, b, c, 2, d, d3, e (se inlocuieste in LS vrf. 5 cu e)

a, b, c, 2, d, d3, e, f, a

Cazuri particulare

Cazurile particulare:

- dreptunghiul/poligonul de decupare inclus in poligonul decupat (subiect) si
- poligonul decupat inclus in dreptunghiul /poligonul de decupare

se trateaza separat, prin teste efectuate inainte de executia algoritmului de decupare.

1. Daca decuparea se efectueaza față de suprafata unui dreptunghi cu laturile paralele cu axele, atunci pentru a verifica daca poligonul de decupat este inclus in dreptunghiul de decupare, se efectueaza asupra fiecarui varf al sau, $V[i]$, testul:

$$xmin_drept \leq V[i].x \leq xmax_drept \quad \&\quad ymin_drept \leq V[i].y \leq ymax_drept$$

2. Pentru a verifica daca dreptunghiul/poligonul de decupare este inclus in poligonul de decupat se efectueaza asupra fiecarui varf al dreptunghiului/poligonului de decupare testul de “punct interior unui poligon” (Curs 11: Rasterizarea suprafetelor poligonale).

Analog se verifica daca poligonul de decupat este inclus in poligonul de decupare.

Decuparea vectorilor

Prof. univ. dr. ing. Florica Moldoveanu

Decuparea primitivelor grafice

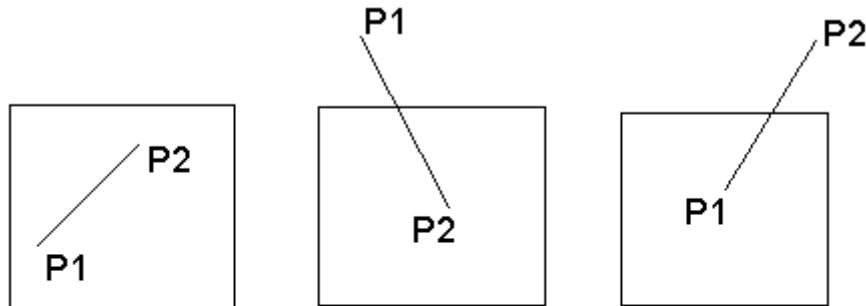
Intr-un sistem grafic 2D:

- Decupare la frontiera unui dreptunghi din spatiul de afisare, avand laturile paralele cu cele ale sistemului de coordonate carteziene 2D (poarta de afisare)
- Decupare la frontiera unui poligon oarecare din spatiul de afisare
- Decuparea se poate efectua:
 - Analitic, prin intersectia primitivelor grafice cu frontiera dreptunghiului/poligonului de decupare, inainte de rasterizare
 - La momentul rasterizarii primitivelor, prin testul de apartenență a fiecarui fragment la dreptunghiul/poligonul de decupare
- Pentru linii și poligoane, decuparea se efectuează analitic (mai eficient).

Intr-un sistem grafic 3D:

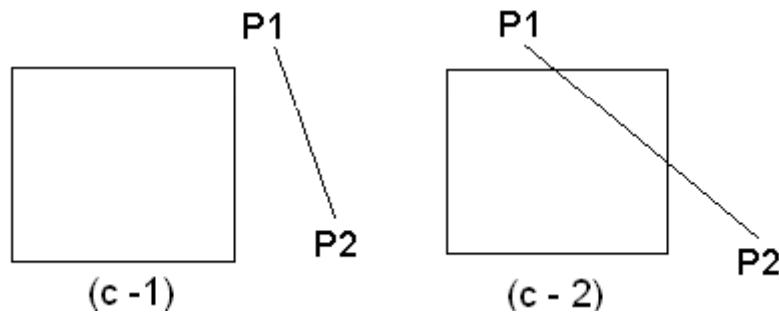
- Decupare la frontiera volumului vizual canonic, după transformarea de proiecție.

Decuparea vectorilor 2D(1)



(a)

(b)



(c - 1)

(c - 2)

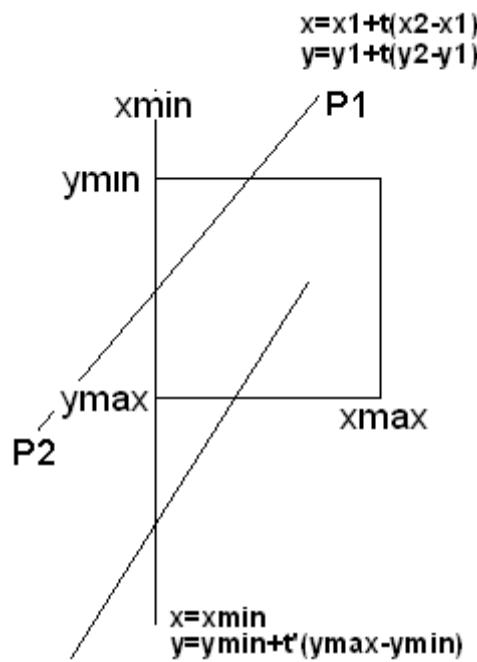
Pozitia vectorului fata de dreptunghiul de decupare

Optimizari: evitarea calculelor de intersectie inutile

- Evitarea calculelor de intersectie in cazurile (a) si (c-1)
- Reducerea calculelor de intersectie in celelalte cazuri, prin evitarea calculelor de intersectie cu laturile neintersectate.

Decuparea vectorilor 2D(2)

Calculul intersectiilor cu laturile dreptunghiului de decupare



Intersectia segmentului $P1-P2$ cu latura $x = xmin$ a dreptunghiului:

1) $xmin=x_1+t(x_2-x_1) \rightarrow t=(xmin-x_1)/(x_2-x_1)$

2) este $0 \leq t \leq 1$???

da: punctul de intersectie se afla pe segmentul $P1-P2$

- se calculeaza $y_i=y_1+(xmin-x_1)*(y_2-y_1)/(x_2-x_1)$

- este $ymin \leq y_i \leq ymax$??

da: exista intersectie intre vectorul $P1-P2$ si latura $x=xmin$

Decuparea vectorilor 2D(3)

Algoritmul Cohen-Sutherland(1)

1001	1000	1010
0001	0000	0010
0101	0100	0110

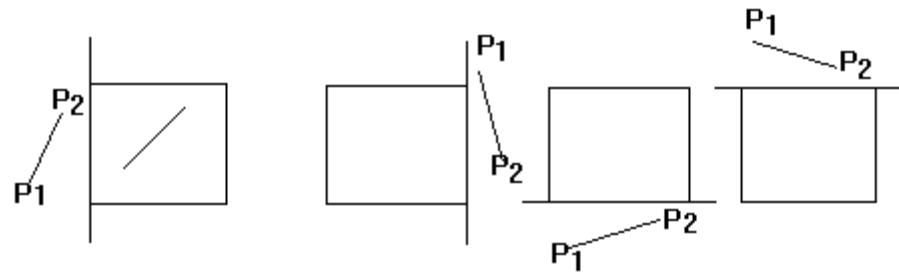
- Punctele din plan sunt codificate prin 4 biti, in functie de pozitia lor fata de dreptunghiul de decupare.

De exemplu: b3b2b1b0

b0=1: stanga, b1=1: dreapta,

b2=1: sub dreptunghi, b3=1: deasupra dreptunghiului

P1: cod1, P2: cod2



1) $\text{cod1} == 0 \ \&\& \ \text{cod2} == 0$:
segment acceptat trivial

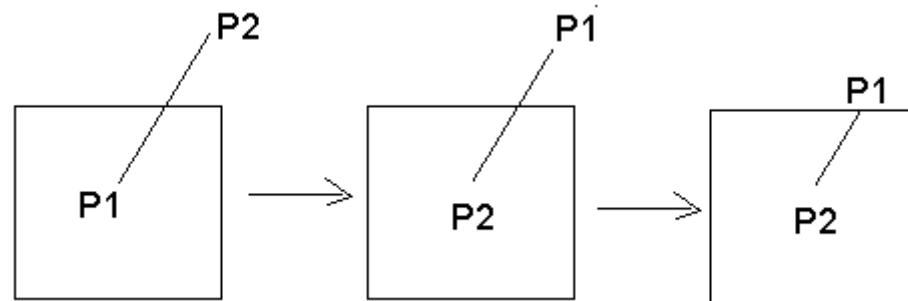
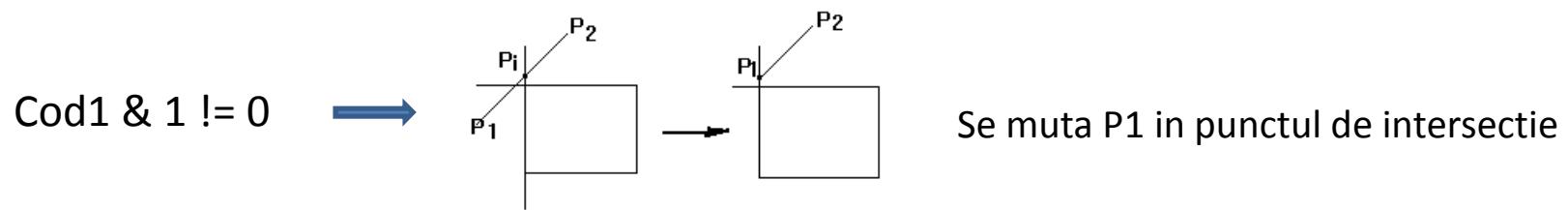
2) $(\text{cod1} \ \& \ \text{cod2}) != 0$: segment rejectat trivial

3) altfel: se intersecteaza segmentul P1-P2 cu dreptunghiul de decupare

Decuparea vectorilor 2D(4)

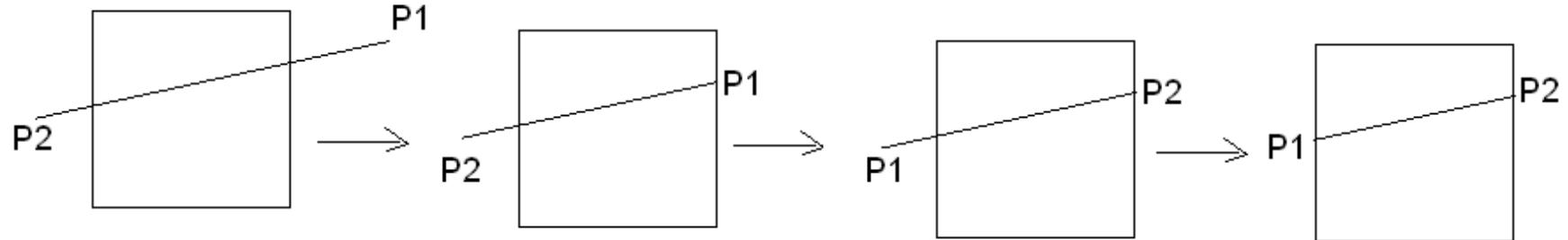
Algoritmul Cohen-Sutherland(2)

- Alegera laturii cu care se intersecteaza segmentul P1-P2: pe baza codului varfului P1, care trebuie sa fie situat in afara dreptunghiului de decupare



Decuparea vectorilor 2D(5)

Algoritm Cohen-Sutherland(3)



Calculul punctelor de intersectie

Intersectia cu latura situata pe dreapta $y = y_{max}$

$$y_{max} = y_1 + t(y_2 - y_1) \text{ de unde, } t = (y_{max} - y_1) / (y_2 - y_1)$$

$$x = x_1 + t(x_2 - x_1) \rightarrow x_i = x_1 + (y_{max} - y_1) / m$$

Intersectia cu latura situata pe dreapta $x = x_{min}$

$$x_{min} = x_1 + t(x_2 - x_1), \text{ deci } t = (x_{min} - x_1) / (x_2 - x_1)$$

$$y = y_1 + t(y_2 - y_1) \rightarrow y_i = y_1 + (x_{min} - x_1) * m$$

Decuparea vectorilor 2D(6)

Algoritmul Cohen-Sutherland(4)

```
int xmin, ymin, xmax, ymax; // colturile dreptunghiului de decupare
```

```
// functia calculeaza codul binar al unui punct din plan
```

```
int codif(int x, int y)
```

```
{
```

```
    int cod;
```

```
    cod=(x < xmin) ? 1:0;
```

```
    if(x > xmax) cod=2;
```

```
    if(y < ymin) cod |=4;
```

```
    if(y > ymax) cod |=8;
```

```
    return cod;
```

```
}
```

Decuparea vectorilor 2D(7)

Algorimul Cohen-Sutherland (5)

```
int Cohen_Suth(int x1i,int y1i,int x2i,int y2i, int* x1d,int* y1d,int* x2d,int* y2d)
{ // primeste coord. capetelor vectorului de decupat si intoarce coord. partii incluse in
    dreptunghi, daca exista
    int acceptat,rejectat,vertical;
    float m, x1=x1i, y1=y1i, x2=x2i, y2=y2i;
    int cod1, cod2, t;
    acceptat=rejectat=vertical=0;
    if (x1i != x2i)
        m=(y2-y1)/(x2-x1);
    else
        vertical=1;
    do
    { cod1=codif(x1,y1); cod2=codif(x2,y2);
        if(cod1==0 && cod2==0) { acceptat=1; break;}
        if((cod1 & cod2) !=0 ) { rejectat=1; break;}
    }
```

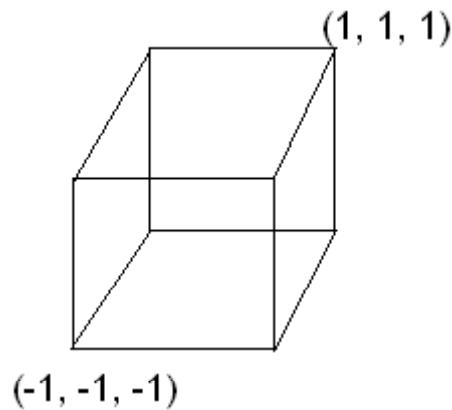
Decuparea vectorilor 2D(8)

```
if(cod1==0) //inversare capete
{t=x1;x1=x2;x2=t; t=y1;y1=y2;y2=t; cod1=cod2;}
if(cod1 & 1)
{ y1+=(xmin-x1)*m; x1=xmin;}
else
if(cod1 & 2)
{ y1+=(xmax-x1)*m; x1=xmax; }
else
if(cod1 & 4)
{ if (!vertical) x1+=(ymin-y1)/m;
  y1=ymin;
}
else
{if (!vertical) x1+=(ymax-y1)/m;
  y1=ymax;
}
} while ( acceptat==rejectat);
```

Decuparea vectorilor 2D(9)

```
if(acceptat)  
{  
    *x1d=x1; *y1d=y1; *x2d=x2; *y2d=y2;  
    return 1;  
}  
return 0;  
}  
}
```

Generalizarea algoritmului Cohen-Sutherland pentru decuparea vectorilor 3D(1)



Fata de volumul vizual canonic, un punct din spatiu se poate afla: in interior, in stanga, in dreapta, sub volum, deasupra volumului, in fata, in spate:
→pentru codificarea pozitiei unui punct din spatiu fata de volumul canonic sunt necesari 6 biti.

De ex. se poate face conventia:

$b_0 = 1$ pentru puncte (x,y,z) cu $x < -1$

daca: $\text{cod}(P1) == 0 \ \&\& \ \text{cod}(P2) == 0$
segment acceptat trivial

$b_1 = 1$ pentru puncte (x,y,z) cu $x > 1$

daca $(\text{cod}(P1) \ \&\& \ \text{cod}(P2)) != 0$
segment rejectat trivial
altfel

$b_2 = 1$ pentru puncte (x,y,z) cu $y < -1$

se intersecteaza segmentul cu volumul

$b_3 = 1$ pentru puncte (x,y,z) cu $y > 1$

$b_4 = 1$ pentru puncte (x,y,z) cu $z < -1$

$b_5 = 1$ pentru puncte (x,y,z) cu $z > 1$

Generalizarea algoritmului Cohen-Sutherland pentru decuparea vectorilor 3D(2)

Intersectiile segmentului cu volumul vizual

Ec. parametrice ale segmentului

$$x = x_1 + t(x_2 - x_1)$$

$$y = y_1 + t(y_2 - y_1)$$

$$z = z_1 + t(z_2 - z_1)$$

daca cod(P1) & 8 !=0

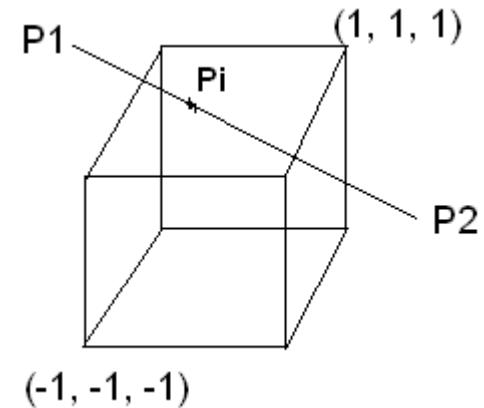
se efectueaza intersectia cu planul $y=1$:

$$1 = y_1 + t(y_2 - y_1) \rightarrow t_{\text{inters}} = (1-y_1)/(y_2-y_1)$$

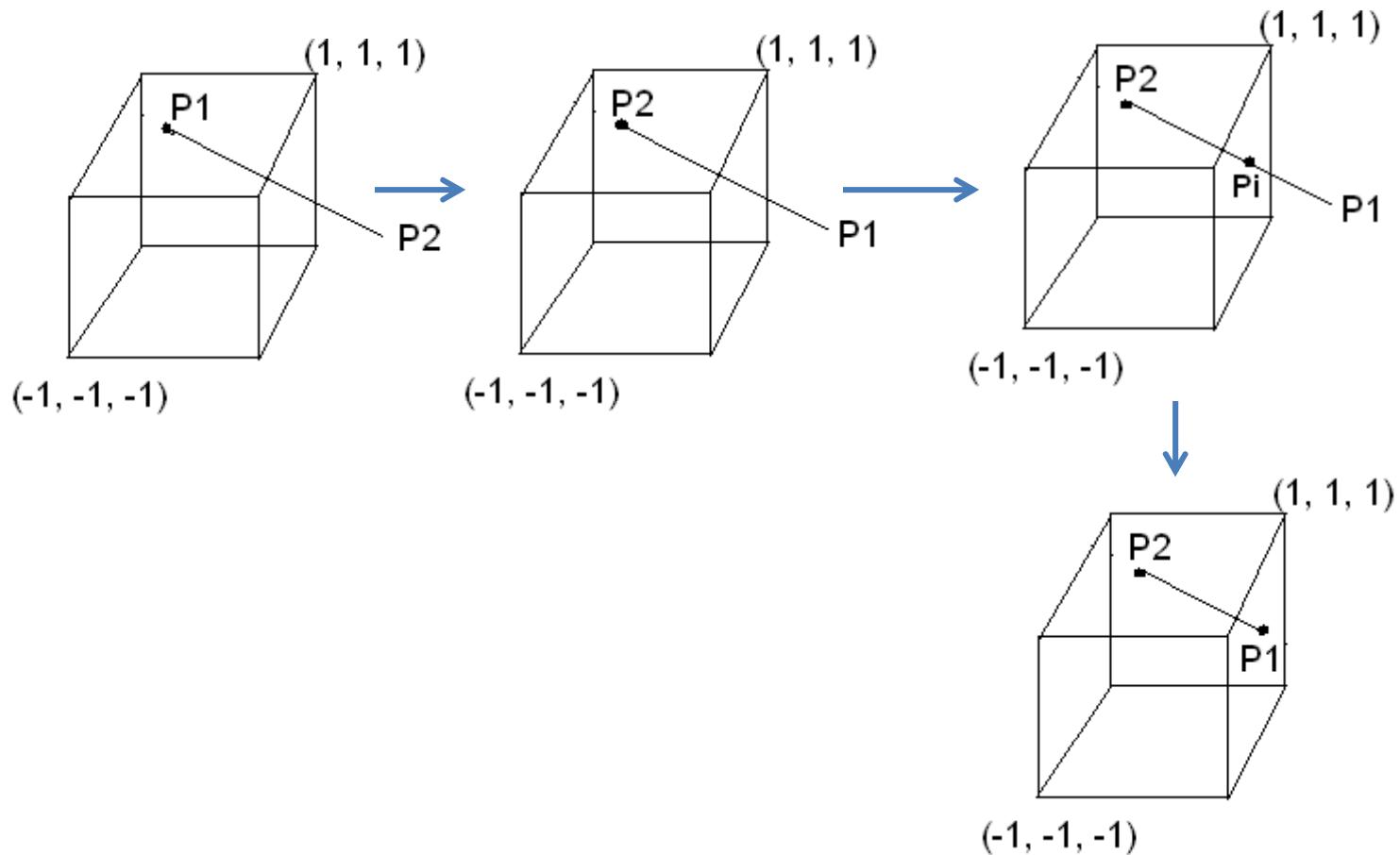
$$x_i = x_1 + t_{\text{inters}}(x_2 - x_1)$$

$$y_i = 1;$$

$$z_i = z_1 + t_{\text{inters}}(z_2 - z_1)$$



Generalizarea algoritmului Cohen-Sutherland pentru decuparea vectorilor 3D(3)



Redarea suprafețelor 3D folosind texturi

Prof. univ. dr. ing. Florica Moldoveanu

Texturi (1)

- Sunt două tipuri de texturi:

1. Texturi reprezentate prin imagini sau şabloane care se aplică pe o suprafaţă netedă, suprafaţa ramanând netedă și după aplicarea texturii.

➤ Imaginea se aplică (« mapeaza ») pe suprafața în timpul generării imaginii suprafetei



➤ Imaginea mediului înconjurător redată pe suprafața unui obiect prin simularea reflexiei și a refractiei.

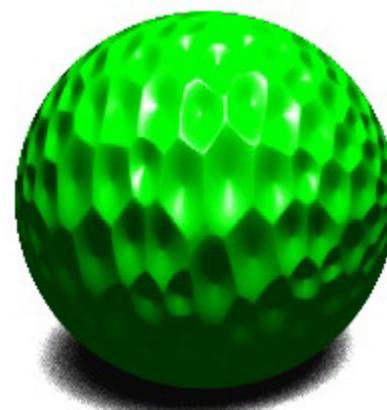


Texturi (2)

2. Texturi prin care se redau detalii ale geometriei suprafetei.

Se folosesc diferite metode:

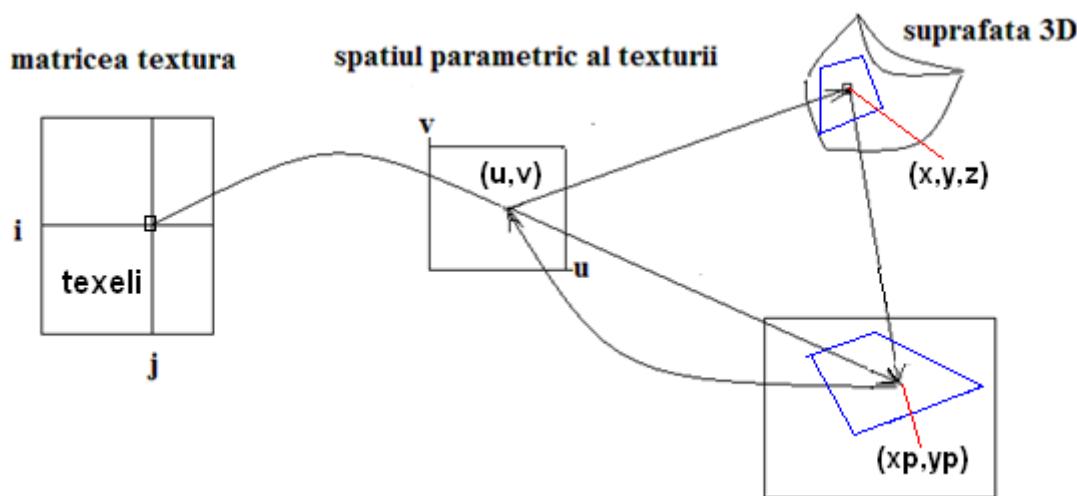
- perturbarea normalelor in punctele suprafetei
- metode fractale
- metode spectrale (functia textura este definita pentru componentele ei pe diferite frecvente)
- si altele



Redarea detaliilor geometrice prin perturbarea normalei la suprafata.

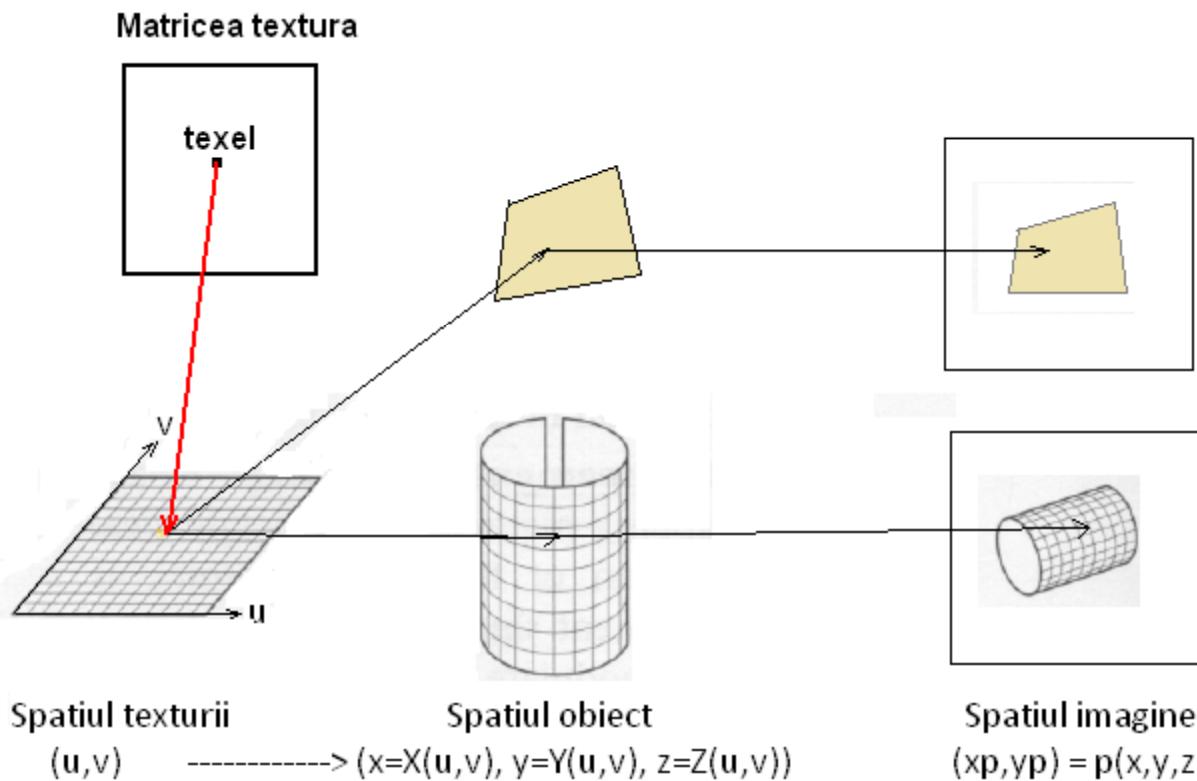
Metode de aplicare a imaginilor textura

- Imaginea textura este memorata intr-o matrice ale carei elemente sunt culori si sunt adreseate prin intermediul unui spatiu parametric (u,v) , $0 \leq u,v \leq 1$
- Elementele matricei textura sunt numite « texeli ».



- ❖ Sunt 2 metode de aplicare a imaginii textura pe imaginea unei suprafete 3D:
 1. “maparea directă” (forward mapping)
 2. “maparea inversă”(inverse mapping)

Maparea directă(1)



Maparea directa(2)

- Matricea textura este parcursa linie cu linie si pentru fiecare texel se calculeaza adresa pixelului in care se reprezinta.
- Imaginea de iesire este completa numai dupa ce intreaga imagine textura a fost procesata. De aceea, este necesar un buffer in care sa fie formata imaginea de iesire.
- Metoda este adevarata atunci cand imaginea textura se obtine in timpul creerii imaginii de iesire, linie cu linie.

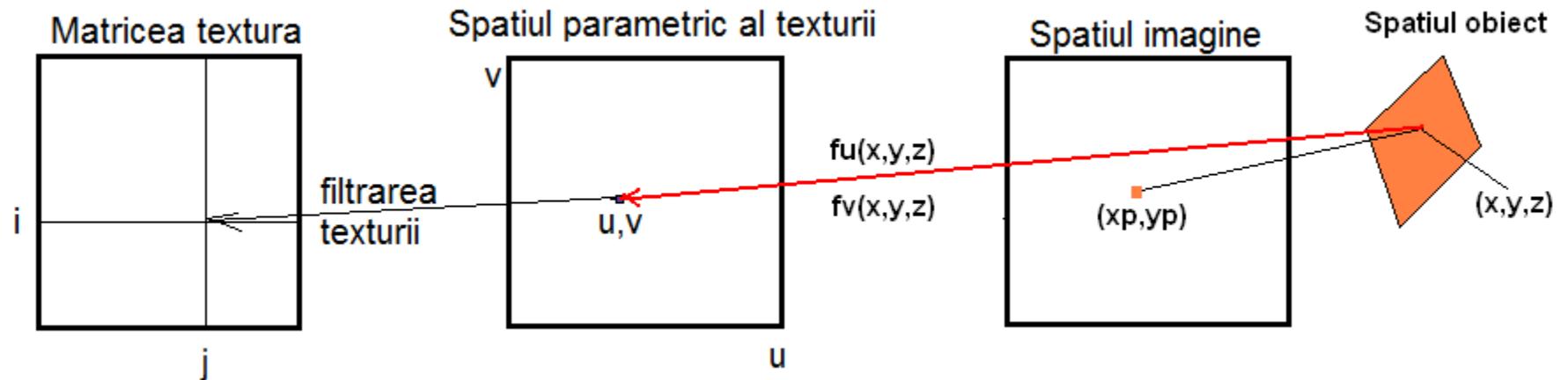
Dezavantaje

- Pixelii sunt afisati intr-o ordine arbitrara si nu se poate garanta ca toti pixelii imaginii sunt asignati texturii.
- Daca se considera fiecare texel (pixel al imaginii de intrare) ca o suprafata, atunci el se mapeaza in imaginea de iesire pe o suprafata de forma unui patrulater → sunt necesare calcule de intersectie dintre patrulater si celulele (pixelii) imaginii de iesire.

Maparea inversa(1)

- Este metoda uzuala de mapare a texturilor pe suprafetele 3D.
- Se pleaca de la spatiul imagine, aplicarea texturii fiind inglobata in procesul de rasterizare a primitivelor in care este descompusa suprafata 3D:
 - pentru fiecare pixel (xp, yp) in care se afiseaza un fragment al unei primitive se calculeaza adresa texelului/texelilor care trebuie folosit(i) in calculul culorii pixelului.

Adresa texelului se obtine folosind **functii de mapare inversa si « filtrarea texturii »**



Maparea inversa(2)

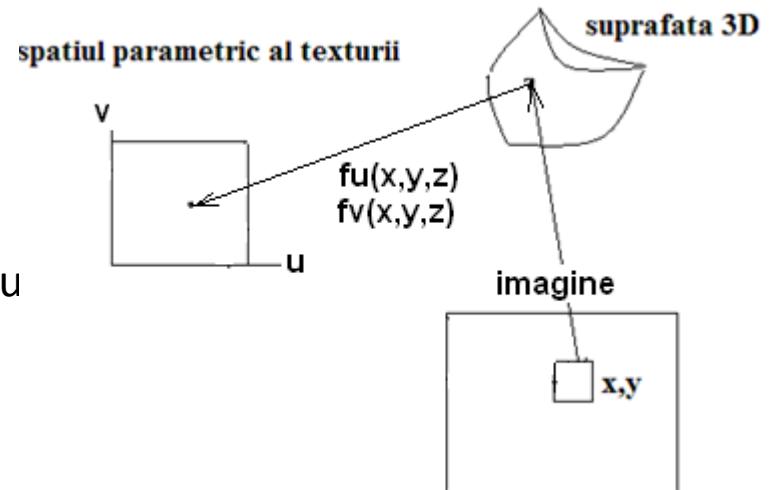
Maparea inversa poate fi descompusa in doua etape :

1. Calculul punctului (x,y,z) de pe suprafata 3D

corespunzator pixelului (xp,yp) in care se aplica textura

2. Calculul adresei (u,v) din spatiul texturii,

corespunzatoare punctului de pe suprafata 3D.



La generarea imaginii se efectueaza o proiectie ortografica asupra scenei 3D, deci $x = xp$, $y = yp$ iar z se calculeaza in functie de tipul suprafetei;

Functiile $f_u(x,y,z)$ si $f_v(x,y,z)$ care definesc corespondenta dintre punctele suprafetei 3D si punctele spatiului (u,v) se numesc functii de mapare inversa:

$$u=f_u(x,y,z); v=f_v(x,y,z)$$

Există functii de mapare inversă numai pentru anumite tipuri de suprafete; ele se numesc **functii de mapare standard**.

Functii de mapare standard(1)

1. Maparea plana (maparea pe un plan)

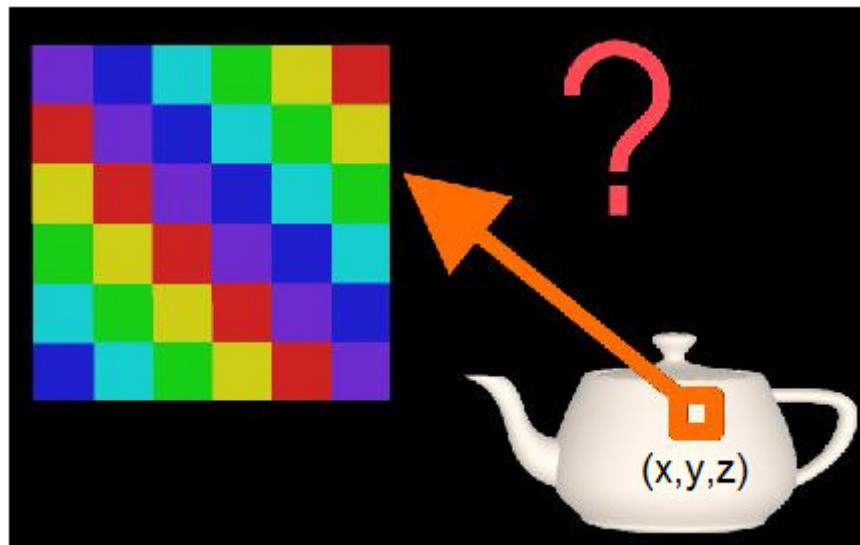
Ecuatiile care descriu aceasta mapare sunt:

$$\begin{cases} u = s_u \cdot x - o_u & x \in X \\ v = s_v \cdot y - o_v & y \in Y \end{cases} \quad 0 \leq u, v \leq 1$$

unde : s_u, s_v sunt factorii de scalare pentru transformarea spatiului (X, Y) in (u, v) iar

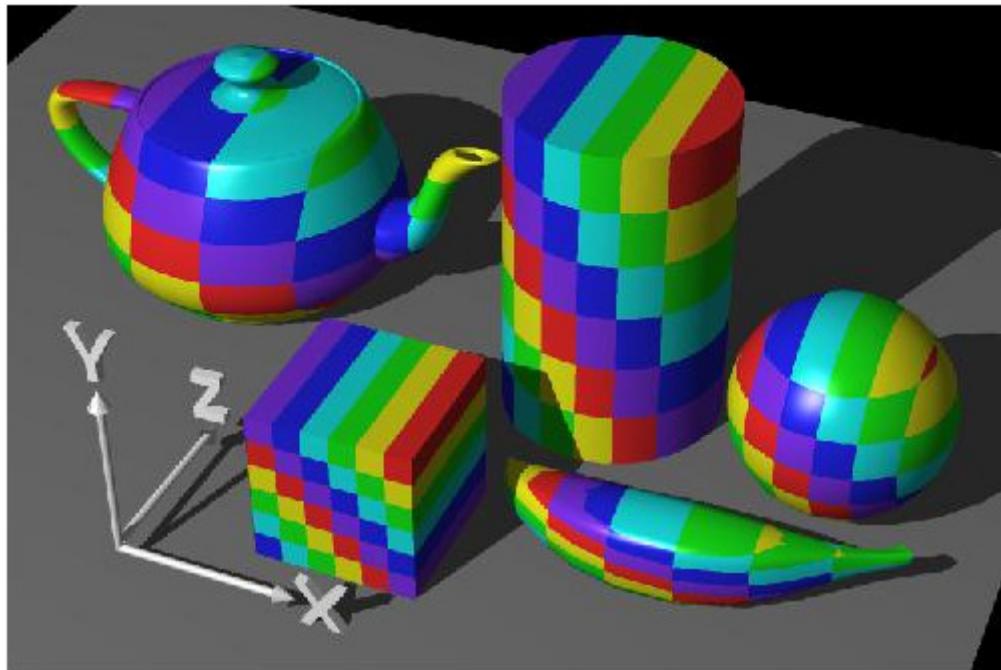
(o_u, o_v) este originea texturii: determina modul in care se aseaza textura pe suprafata.

$(u, v) \leftarrow (x, y) \leftarrow (x, y, z)$ - se renunta la coordonata z !



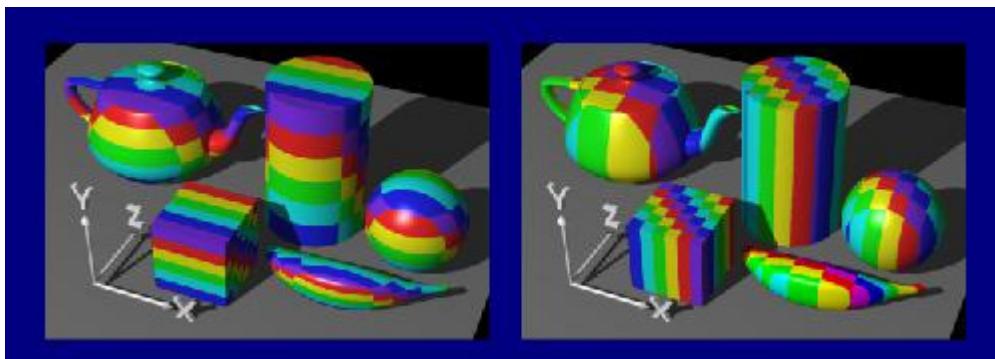
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm

Exemple de mapari plane



Maparea plana prin renuntarea la coordonata z: toate punctele suprafetei cu aceleasi coordonate (x,y) au aceeasi culoare.

http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm



Maparea plana prin renuntarea la coordonata x, respectiv y.

Functii de mapare standard(2)

2. Maparea cilindrica

Coordonatele (x,y,z) de pe suprafata sunt convertite in coordonate cilindrice, (θ, w) .

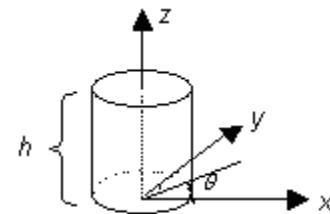
Ecuatiile parametrice ale unei suprafete cilindrice:

$$X = r \cdot \cos(\theta)$$

$$Y = r \cdot \sin(\theta)$$

$$Z = w \cdot h$$

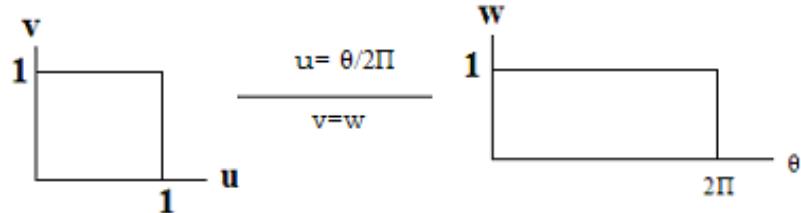
unde $0 < \theta < 2\pi$, $0 < w < 1$



-Se trece din coordonate (x, y, z) in coordonate cilindrice

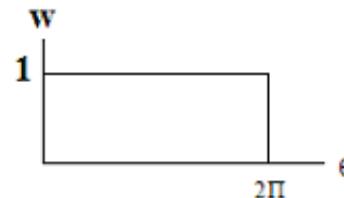
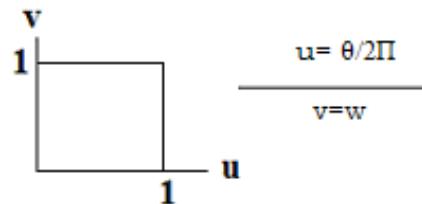
-Se pune in corespondenta spatiul parametric al texturii, (u,v) , spatiului parametric al suprafetei:

$$\begin{cases} x = r \cdot \cos \theta \\ y = r \cdot \sin \theta \\ z = h \cdot w \end{cases} \Rightarrow \begin{cases} \theta = \arctg \frac{x}{y} \\ w = \frac{z}{h} \end{cases}$$



Functii de mapare standard(3)

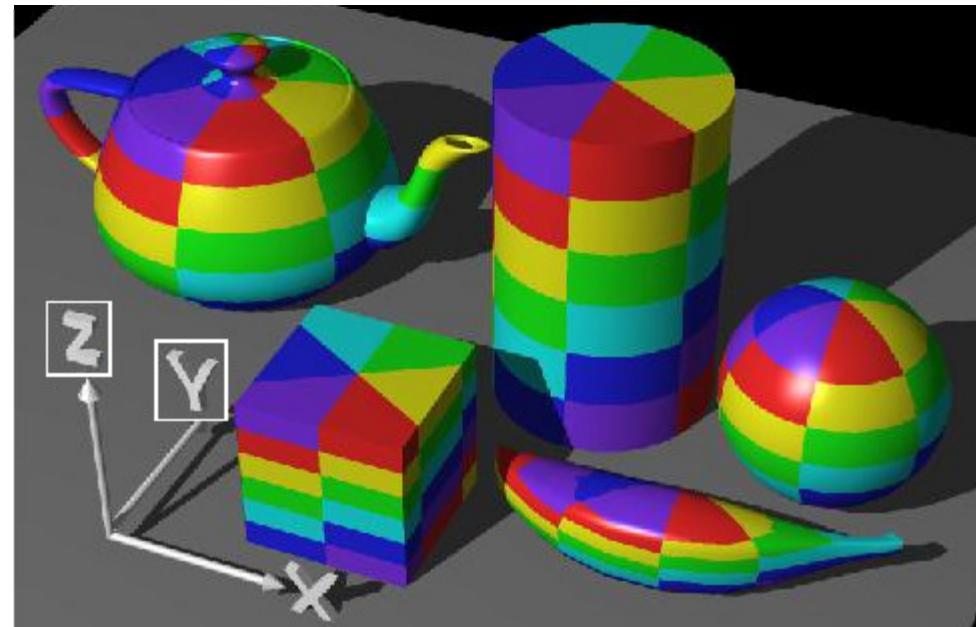
$$\begin{cases} u = \frac{\theta}{2\pi} \\ v = w \end{cases}$$



$$\begin{cases} u = \frac{1}{2\pi} \cdot \arctg\left(\frac{y}{x}\right) \\ v = \frac{z}{h} \end{cases}$$

Functia de mapare cilindrica

Transformarea patratelor texturii in punctele suprafetei de $z=z_{\min}$ si $z=z_{\max}$

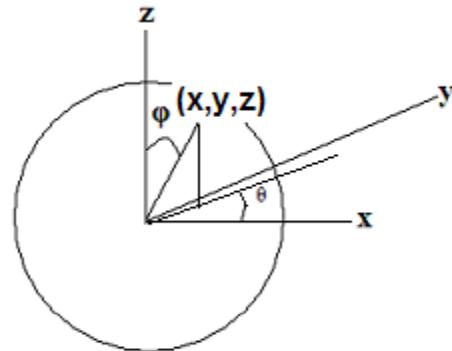


Functii de mapare standard(4)

3. Maparea sferica

- Se pleaca de la ecuatiiile parametrice ale sferei
- Se trece din coordonate (x, y, z) in coordonate sferice

$$\begin{cases} x = r \cos \theta \sin \varphi \\ y = r \sin \theta \sin \varphi, \text{ cu} \\ z = r \cos \varphi \end{cases} \begin{cases} 0 < \theta < 2\pi \\ 0 < \varphi < \pi \end{cases} \Rightarrow \begin{cases} \theta = \arctg \left(\frac{y}{x} \right) \\ \varphi = \arccos \left(\frac{z}{r} \right) \end{cases}$$



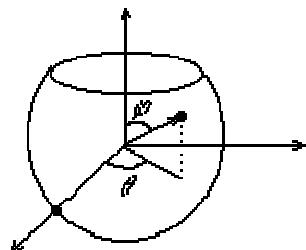
Rezulta

$$\begin{cases} u = \frac{\theta}{2\pi} \\ v = \frac{\varphi}{\pi} \end{cases} \quad \text{sau} \quad \begin{cases} u = \frac{1}{2\pi} \arctg \left(\frac{y}{x} \right) \\ v = \frac{1}{\pi} \arccos \left(\frac{z}{r} \right) = \frac{1}{\pi} \arccos \frac{z}{\sqrt{x^2 + y^2 + z^2}} \end{cases}$$

Functia de mapare sferica

Functii de mapare standard(5)

Maparea pe un sector de sferă



$$\begin{cases} 0 \leq \theta \leq \frac{\pi}{2} \\ \frac{\pi}{4} \leq \phi \leq \frac{\pi}{2} \end{cases}$$

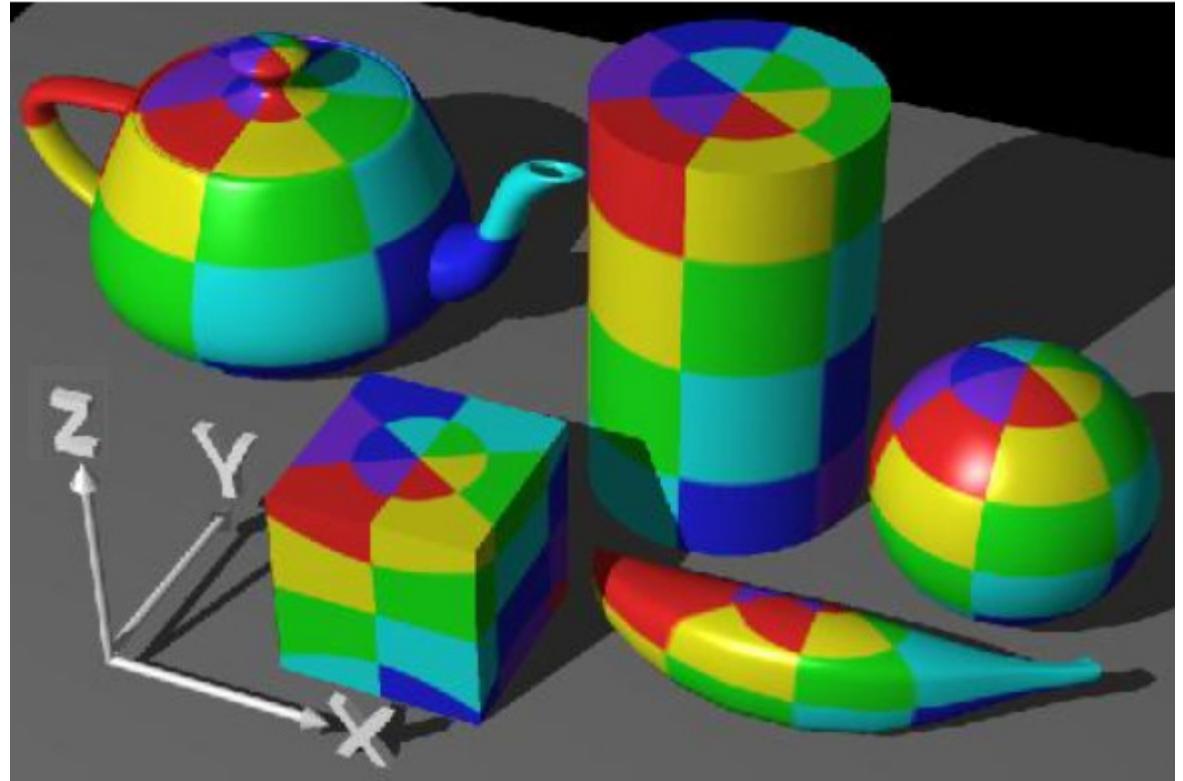
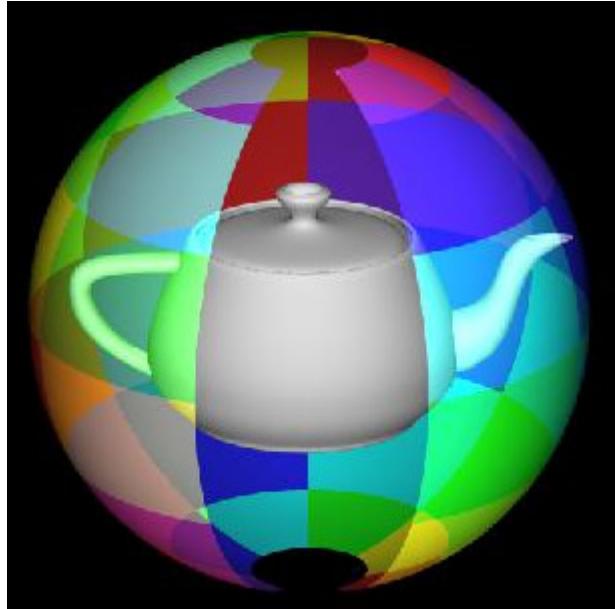
$$u = \theta / (\pi/2)$$

$$v = ((\pi/2) - \phi) / (\pi/4)$$

Functia de mapare

Exemple de mapari sferice

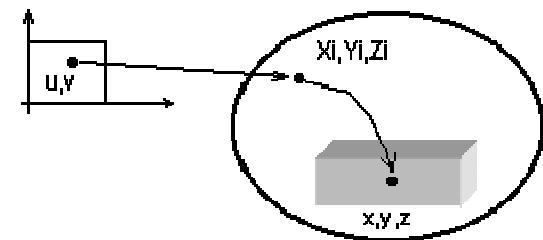
Maparea sferică “întinde” patratele texturii la nivelul ecuatorului și le “comprimă” proporțional cu apropierea de poli.



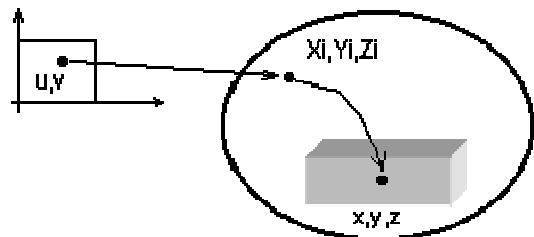
http://www.siggraph.org/education/materials/HyperGraph/mapping/r_wolfe/r_wolfe_mapping_2.htm

Maparea în 2 pași(1)

- Este o tehnica de mapare independenta de forma obiectului pe care se aplica textura. Metoda a fost definita de Bier si Sloan (1986).
- Daca pentru suprafata texturata nu exista o functie de mapare inversa, atunci, se stabileste o corespondenta intre punctele suprafetei texturate si punctele unei suprafete 3D pentru care exista o functie de mapare inversa, numita in continuare « suprafata intermediara ».
- Aplicarea texturii este descompusa in doua operatii, al caror efect este :
 - **mularea texturii pe suprafata intermediara;**
 - **mularea suprafetei intermediare pe suprafata de texturat.**



Maparea în 2 pași(2)



a) **mapare S (mapare inversă):**

$$u = f_u(x_i, y_i, z_i)$$

$$v = f_v(x_i, y_i, z_i)$$

b) $(x_i, y_i, z_i) \leftarrow \text{mapare O} \leftarrow (x, y, z)$

- a) Mularea texturii pe suprafata intermediara se realizeaza prin functia de mapare inversa. Aceasta este numita « **mapare S** », adica mapare pe *suprafata intermediara*.
- b) Mularea suprafetei intermediare pe suprafata de texturat este numita « **mapare O** ». Ea trebuie sa puna in corespondenta fiecarui punct (x, y, z) al suprafetei (*obiectului*) de texturat, un punct (x_i, y_i, z_i) al suprafetei intermediare.

Maparea în 2 pasi(3)

Maparea S

Se folosesc 4 tipuri de suprafete intermediare :

1. un plan cu o orientare oarecare;
2. sferă;
3. cilindrul;
4. fețele unui cub.

Alegerea tipului suprafetei intermediare este dependenta de forma obiectului pe care trebuie sa fie aplicata textura.

Maparea O

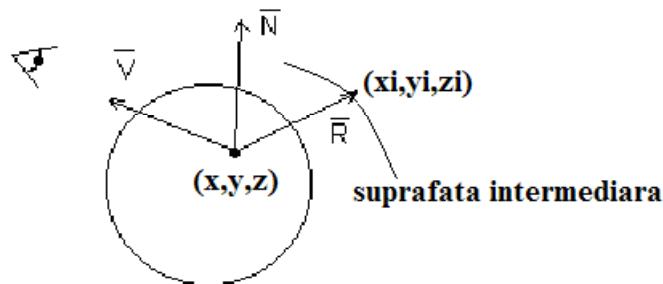
Autorii metodei au definit 4 tipuri de mapari O:

Maparea în 2 pași(4)

Mapari O

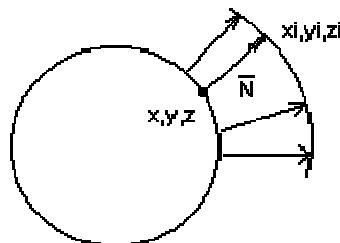
(x,y,z) – un punct al suprafetei , pentru care se dorește calculul coordonatelor (u,v)

(x_i, y_i, z_i) – punctul corespunzător lui (x,y,z) pe suprafata intermediara



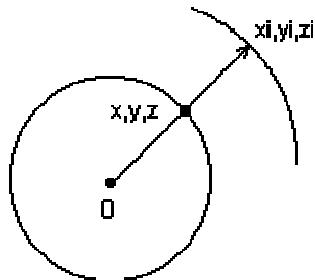
1. (x_i, y_i, z_i) , se obtine intersectand suprafata intermediara cu vectorul reflectat al vederii in punctul (x,y,z) . R este simetric cu V fata de N.

Metoda este dependenta de pozitia observatorului (se foloseste la « maparea mediului inconjurator »)

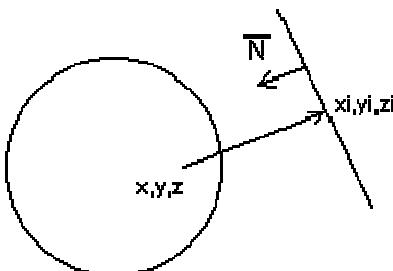


2. (x_i, y_i, z_i) este intersectia suprafetei intermediare cu normala la obiect in (x,y,z) .

Maparea in 2 pasi(5)



3. (x_i, y_i, z_i) este intersectia suprafetei intermediare cu dreapta care trece prin (x, y, z) si centroidul obiectului.



4. (x_i, y_i, z_i) este intersectia suprafetei intermediare cu dreapta care trece prin (x, y, z) si are directia normalei la suprafata intermediara.

Acest tip de mapare O este corelat cu o mapare S pe un plan sau pe fetele unui cub.

Combinatia « mapare S pe cilindru » - « mapare O de tip 4 » este numita « shrinkwrap ».

Aplicarea texturilor pe fetele unei rețele poligonale 3D (1)

In sistemele grafice actuale, orice suprafață 3D este descompusa, în vederea redarii sale, într-o rețea de fațete poligonale/triunghiuri: textura se aplică pe poligoane la momentul rasterizării lor.

Problema mapării inverse apare deci pentru fragmentele rezultate din rasterizarea poligoanelor.

Textura, definită în spațiul parametric pentru $0 \leq u, v \leq 1$, trebuie "întinsă" peste întreaga rețea de fațete poligonale.

Există două tipuri de **metode**:

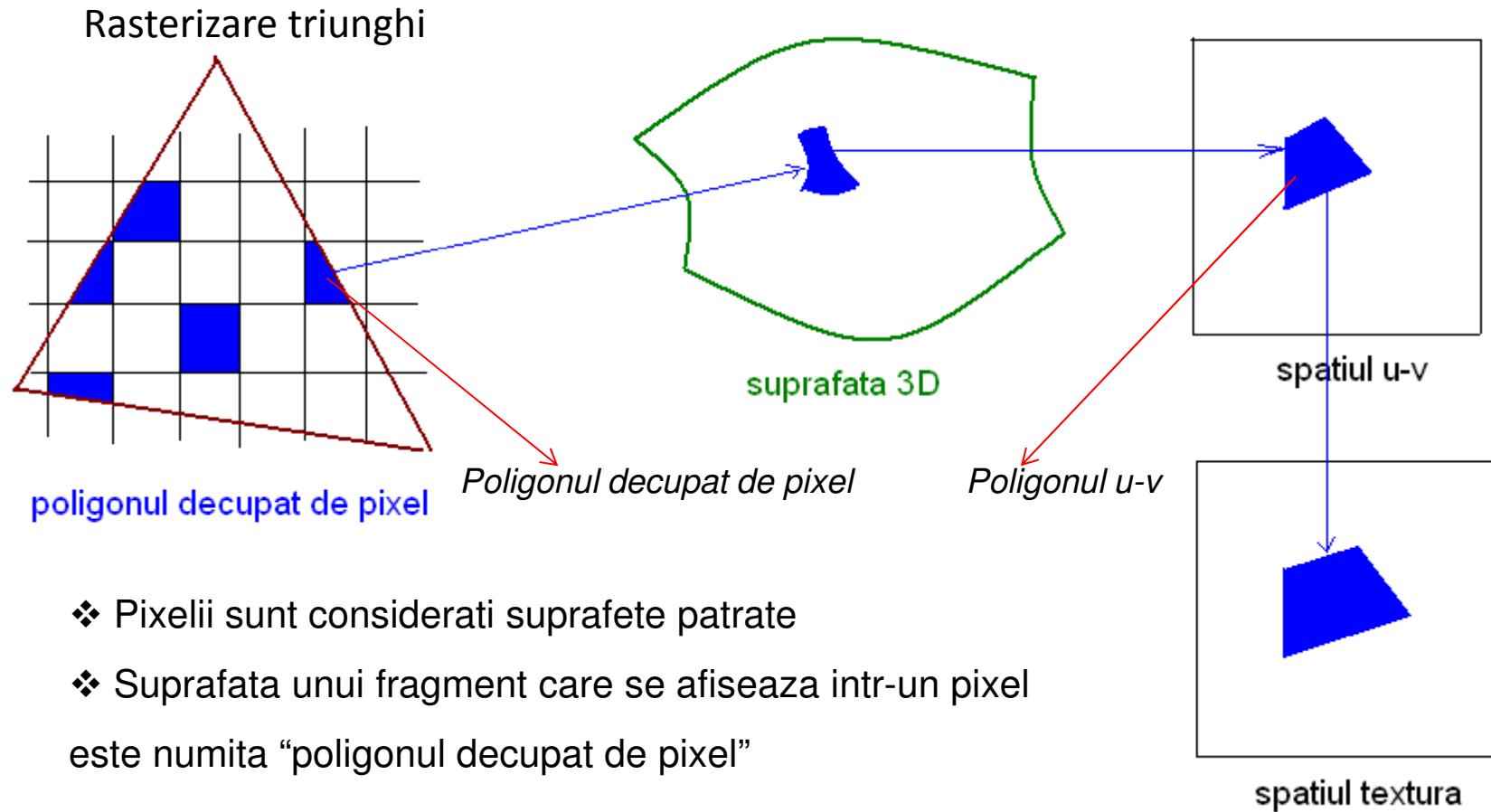
Exacte: se calculează (u, v) exact pentru fiecare fragment al unei fațete

Aproximative:

- se calculează (u, v) pentru fiecare varf al unei fațete
 - pentru o suprafață definită parametric $S(u, v) = (x(u, v), y(u, v), z(u, v))$ se cunoaște (u, v) pentru fiecare punct de pe suprafață
 - pentru o suprafață oarecare, (u, v) se poate obține printr-o *mapare in doi pasi*.
- se calculează (u, v) pentru fiecare fragment prin interpolare între valorile (u, v) asociate varfurilor fațetei.

Aplicarea texturilor pe fetele unei rețele poligonale 3D (2)

Metode exacte(1)



Aplicarea texturilor pe fetele unei retele poligonale 3D(3)

Metode exacte (2)

Algoritm:

pentru fiecare fragment rezultat din rasterizarea unui poligon:

- determina varfurile poligonului decupat de pixelul in care se afiseaza, (x_i, y_i) ;
- determina varfurile peticului de suprafata 3D corespunzator poligonului decupat de pixel, (x'_i, y'_i, z_i) , aplicand transformarea de vizualizare inversa;
- calculeaza varfurile (u_i, v_i) ale poligonului corespunzator in spatiul $(u, v) - \text{poligonul } u-v$, utilizand o functie de mapare inversa sau o mapare in 2 pasi:

$$(u_i, v_i) = (f_u(x'_i, y'_i, z_i), f_v(x'_i, y'_i, z_i))$$

- determina culoarea de afisare a pixelului folosind culorile texelilor corespunzatori *poligonului u-v* prin diferite metode.

Aplicarea texturilor pe fetele unei retele poligonale

3D(4)

Metode exacte (3) - Calculul culorii pixelului : metoda 1

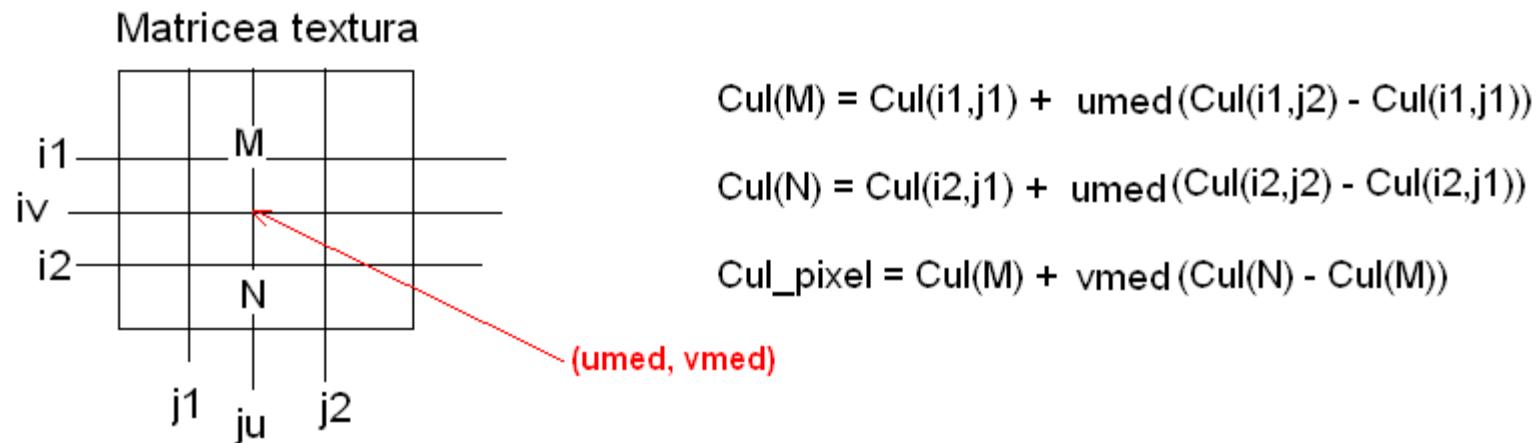
- Se calculeaza o valoare medie a coordonatelor varfurilor poligonului u-v :

$$u_{med} = \text{suma}(u_i) / \text{nr. varfuri}; \quad v_{med} = \text{suma}(v_i) / \text{nr. varfuri};$$

- **Se calculeaza culoarea pixelului folosind texelii din matricea textura, prin interpolare biliniara.**

Daca (ju, iv) sunt adresele reale corespunzatoare valorilor (u_{med}, v_{med}) in spatiul textura,

$ju = u_{med}/\text{nr_col_matrice_textura}$, $iv = v_{med}/\text{nr_linii_matrice_textura}$, atunci:



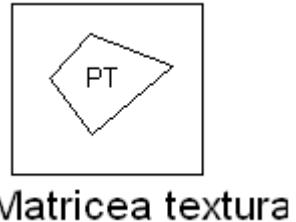
i_1, i_2, j_1, j_2 sunt indici de linie/coloana in matricea textura

$j_1 \leq ju \leq j_2, \quad i_1 \leq iv \leq i_2$

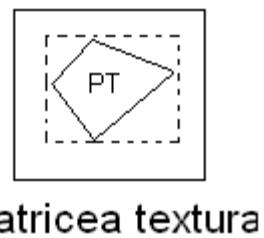
Aplicarea texturilor pe fetele unei retele poligonale 3D(5)

Metode exacte (4) - Calculul culorii pixelului: metoda 2

- Se calculeaza poligonul corespunzator *poligonului u-v* in matricea textura, PT



- Se aproximeaza poligonul textura PT printr-un dreptunghi



- Se calculeaza media valorilor texelilor din dreptunghi.

Aplicarea texturilor pe fetele unei retele poligonale

3D(6)

Metode aproximative

- Se cunoaste (u, v) pentru fiecare varf al poligonului rasterizat.
- Pentru fiecare fragment rezultat din rasterizarea poligonului se calculeaza o adresa textura (uf, vf) , prin interpolarea adreselor textura, (ui, vi) asociate varfurilor (operatie integrata in algoritmul de rasterizare).
 - Pentru rezultate corecte este necesara efectuarea unei interpolari perspectiva. Interpolarea liniara este mai rapida dar nu produce efectul de perspectiva la aplicarea unei texturi pe suprafata unui poligon inclinat fata de planul de vizualizare.
- In cazul utilizarii mai multor texturi la redarea poligonului, fiecarui varf ii sunt asociate un acelasi numar de perechi de coordonate textura (ui, vi) , $1 \leq i \leq n$, pentru accesarea celor n texturi; n este in general limitat la 8.
- Se calculeaza culoarea fragmentului folosind (uf, vf) sau (uf_i, vf_i) printr-o metoda de “filtrare a texturii”. Operatia este efectuata in “fragment shader”.

Filtrarea texturii(1)

- O textura este definita pe o latice discreta de puncte in timp ce coordonatele textura sunt valori reale.
In general, o pereche (uf, vf) nu corespunde unui texel al texturii.
- **Metoda de calcul a culorii dintr-o textura, folosind coordonatele (uf, vf) , se numeste "filtrarea texturii".**

Cazul $0 \leq uf, vf \leq 1$

Sunt doua moduri standard de filtrare:

1. Se considera culoarea texelului cel mai apropiat in spatiul laticei (se trunchiaza coordonatele): rapid dar poate produce defecte in imagine
2. Se efectueaza o interpolare biliniara intre culorile celor mai apropiati texeli in spatiul laticei:
 - scade viteza de calcul a culorii fragmentului
 - poate produce un efect de incetosare

Filtrarea texturii(2)

Cazul uf, vf in afara domeniului $[0,1]$

- Coordonatele (ui, vi) asociate varfurilor pot fi in afara domeniului $[0,1]$. Rezulta:
coordonatele (uf, vf) ale unui fragment pot fi in afara domeniului $[0,1]$.
- Coordonatele (uf, vf) rezultate in procesul de rasterizare sunt transformate in coordonate
 $0 \leq u, v \leq 1$
- Cele doua moduri standard de transformare a coordonatelor textura in domeniul $[0,1]$ sunt denumite “clamping” si “wrapping (sau “repeating”): ele permit utilizarea eficienta a texturilor si obtinerea de efecte interesante.

Clamping (Comprimare)

Coordonatele (uf, vf) sunt transformate in (u, v) astfel:

$$(u, v) = (\min(\max(0, uf), 1), \min(\max(0, vf), 1))$$

Deci: daca uf sau $vf < 0$, atunci $u = 0$, respectiv $v = 0$; daca uf sau $vf > 1$, atunci $u = 1$, respectiv $v = 1$

daca $0 \leq uf, vf \leq 1$, atunci $u, v = uf, vf$.

Filtrarea texturii(3)

Clamping (Comprimare)



Aplicarea acestei transformari are ca efect propagarea culorii din imaginea textura de-a lungul directiei in care coordonata textura depaseste limita de 0 sau de 1.

Filtrarea texturii(4)

Wrapping (Repetare)

- Pentru accesarea texturii se calculeaza (u,v) cu formula:

$$(u,v) = (uf - [uf]), vf - [vf]),$$

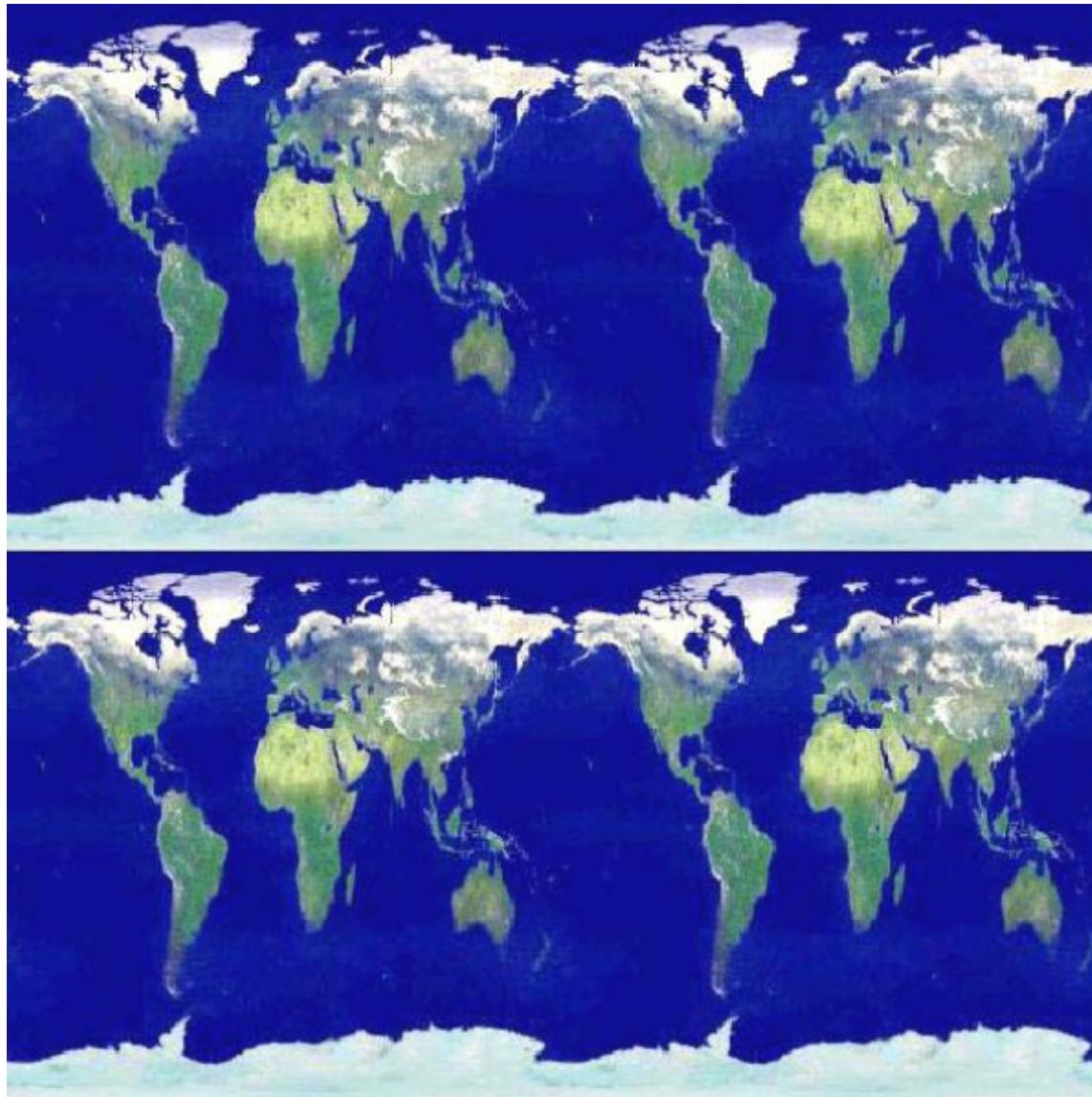
unde $[w]$ reprezinta cel mai mare intreg mai mic sau egal cu w .

De exemplu, $(-0.1, 1.5)$ se transforma in $(-0.1 - (-1), 1.5 - (1)) = (0.9, 0.5)$.

- **Acest mod de transformare permite repetarea unei texturi, producand un efect periodic.**
- La utilizarea acestui mod este necesar ca laturile stanga/dreapta, respectiv sus/jos ale imaginii textura sa se potriveasca, astfel incat sa nu se observe marginile texturii.
- Daca pentru una dintre coordonate se utilizeaza transformarea “comprimare” iar pentru cealalta “repetare”, textura se numeste “cilindrica”.
- Daca pentru ambele coordonate se utilizeaza modul “repetare”, textura se numeste “toroidală”.

Filtrarea texturii(5)

Wrapping (Repetare)



Transparenta texturii

- Imaginea textura poate avea o componenta aditionala, numita “canalul alfa”, prin care se controleaza transparenta sau opacitatea texturii aplicate.
- Fiecare texel este in acest caz reprezentat prin 4 componente: (R,G,B,A), unde A reprezinta opacitatea.
 - daca $A = 1$, atunci texelul este complet opac; daca $A=0$, texelul este complet transparent.
- Pentru $0 \leq A \leq 1$, culoarea texelului moduleaza culoarea fragmentului, determinata pe baza geometriei (modelul Gouraud sau modelul Phong).
- Interfetele de programare OpenGL si Direct3D permit specificarea modului de combinare intre culoarea unui fragment determinata pe baza geometriei primitivei si culoarea rezultata din filtrarea texturii (texturilor), folosind operatori care se aplica culorilor (RGB) si operatori care se aplica opacitatilor A.
- Combinarea dintre culoarea determinata pe baza geometriei si culoarea texturii poate fi programata in “fragment shader”.