1. Asemanarile si deosebirile dintre efectele proiectiilor perspectiva si cele ale proiectiilor axonometrice.

Asemanari:

- ambele redau mai multe fete ale obiectului
- modifica felul in care arata poza

  - Perspectiva:
    - **Efectul de micsorare :** *Dimensiunea obiectului in proiectia 2D este invers proportionala cu distanta de la centrul de proiectie la obiect.*
    - **Modifica unghiurile dintre dreptele care nu sunt paralele cu planul de proiectie :** *Proiectiile lor converg catre un punct de convergenta.*

  - Axonometrice: se face scalarea laturilor :
    - Izometrice : laturile sunt scalate cu factori egali pe cele 3 axe
    - Dimetrice : laturile sunt scalate cu factori egali pe 2 axe
    - Trimetrice : laturile sunt scalate cu factori diferiti pe axe

Deosebiri:

- Perspectiva:
  - centrul de proiectie la distanta finita de planul de proiectie
  - proiectorii = drepte convergente in centrul de proiectie

- Axonometrice:
  - centrul de proiectie este la infinit
  - proiectorii = paralele ce trec prin varfurile obiectului proiectat si au directia specificata.

2. Care sunt parametrii care definesc volumul vizual al proiectiei perspectiva in OpenGL? Care este planul in care se efectueaza proiectia? Forma geometrica a volumului vizual in proiectia perspectiva.

- Parametrii: planul din fata (near), planul spate (far), pozitia observatorului.
- Se efectueaza proiectia pe planul din fata, cu centrul de proiectie in pozitia observatorului.
- Trunchi de piramida (frustum) cu baza mica in pozitia observatorului si delimitat de planurile far & near.

3. BSP (Binary Space Partitioning)

Cum se construieste arborele bsp si ce se memoreaza in noduri

♣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 faţa 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.
♣ Primitivele din semi-spatiul “faţă” formeaza “lista-faţă”, care va genera subarborele “faţă” al nodului.
♣ Primitivele din semi-spatiul “spate” formeaza “lista-spate”, care va genera subarborele “spate” al nodului.
♣ Se alege un poligon din lista-faţă si se creaza nodul radacina al subarborelui “faţă”, etc.

Cand trebuie reconstruit arborele BSP?

Nu trebuie reconstruit la fiecare cadru imagine : avantaj pentru scenele statice.

Cum se afiseaza o scena reprezentata prin BSP?

-Tine cont de pozitia observatorului
-Afisare “back-to-front”: se incepe cu primitiva cea mai indepartata 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.
- In fiecare nod: se avanseaza in subarborele nodului care este opus observatorului fata de plan.

Functia de afisare BSP

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

Cum poate fi utilizat algoritmul pentru eliminarea obiectelor nevizibile din banda grafica?

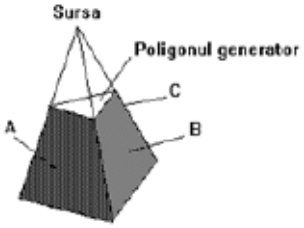
Daca planul de partitionare al unui nod nu intersecteaza volumul vizual, atunci numai subarborele aflat de aceeasi parte cu volumul vizual va fi afisat, celalalt subarbore fiind eliminate din banda grafica.

4. Introducerea umbrelor in imagini prin metoda volumelor de umbra (fara implementare)

❖ Cea mai populara metoda de introducere a umbrelor in imagini.

❑ Sursa de lumină este considerată punctiformă iar scena 3D alcatuita din poligoane.

❑ Un **volum de umbră** este definit de o sursă de lumină şi un poligon luminat (vizibil din poziţia sursei de lumină), pe care-l vom numi **poligonul generator**.



❑ Volumul infinit determinat de o sursă şi un poligon generator este delimitat de o faţă care reprezintă poligonul generator scalat. Această faţă este situată la o distanţă faţă 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.

❑ Fiecare faţă laterală a volumului este numită **poligon de umbră**. Ea este determinată de o latură a poligonului generator şi de cele două drepte care pleacă din sursa de lumină, fiecare trecând printr-un vârf al laturii.

❑ Normalele la feţele laterale punctează înspre exteriorul volumului.

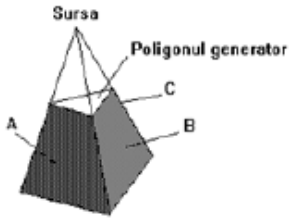
❑ Poligoanele de umbră se folosesc pentru determinarea umbririi produse de poligonul generator în scenă.

Notăm cu :

– PUV poligoanele de umbră care sunt vizibile din poziţia observatorului (A şi B în figura) şi cu

– PUN poligoanele de umbră care nu sunt vizibile din poziţia observatorului (de exemplu, poligonul C).

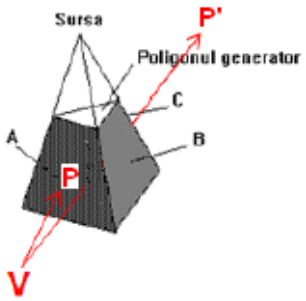
Clasificarea poligoanelor de umbra in PUV si PUN se poate face pe baza normalelor la poligoane (back face culling).



Fie

– un punct P al unei suprafeţe şi

– VP vectorul din poziţia 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.

❑ In general, pentru a determina dacă un punct este în umbră, se poate folosi un contor în care iniţial se memorează numărul de volume de umbră care conţin poziţia observatorului.

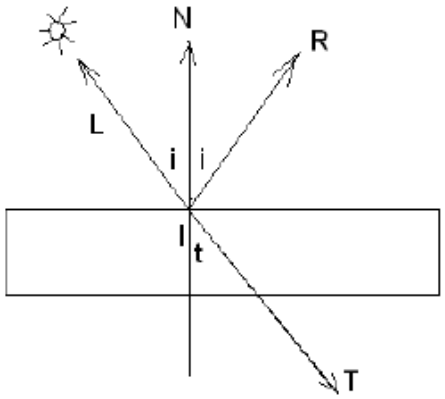
– Se asociază poligoanelor de tip PUV valoarea +1 iar celor de tip PUN valoarea -1.

– 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.

➢ Volumul de calcul presupus de acest algoritm poate fi redus dacă în loc să se calculeze volumul de umbră pentru fiecare poligon vizibil din poziţia sursei, se calculează un singur volum de umbră pentru o suprafaţă poliedrală. In acest scop, se determină poligoanele de umbră numai pentru laturile care fac parte din silueta suprafeţei, văzută din poziţia sursei de lumina.

5. Care sunt razele folosite in calculul culorii unui pixel in algoritmul Ray-Tracing? Cum se calculeaza directia lor? Formula de calcul a culorii unui pixel?



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:

$n1/n2 = \sin(t)/\sin(i)$, n1,n2: indicii de refractie

$T = L*(n1/n2) - (\cos(t) + (n1/n2)*(L \cdot N))*N$

✓*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.

Calculul culorii luminii în punctul I:

$$I_\lambda(I) = I_{local\lambda}(I) + Ks*R_\lambda(I) + Kt*T_\lambda(I)$$

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)

Ks – 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

Kt – 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

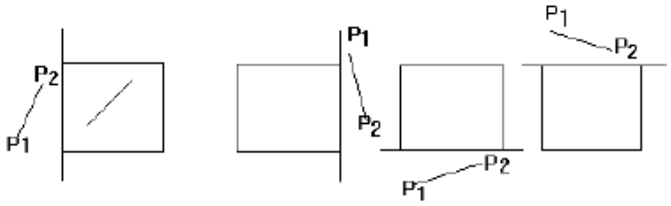
$0 <= ks, kt <= 1$

6. Algoritmul Cohen-Sutherland pentru decuparea vectorilor 2D
a) descriere
b) pasii alg pentru decuparea vect din fig
c) implementare in C

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: pentru punctele din stanga dreptunghiului,
- b1=1: dreapta, b2=1: sub dreptunghi, b3=1: deasupra dreptunghiului

Fie **cod1** codul binar al punctului P1 si **cod2** codul binar al punctului P2.



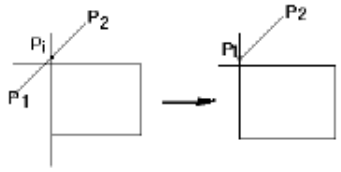
1) cod1 ==0 && cod2==0 :
segment acceptat trivial

2) (cod1 & cod2) != 0: segment rejectat trivial

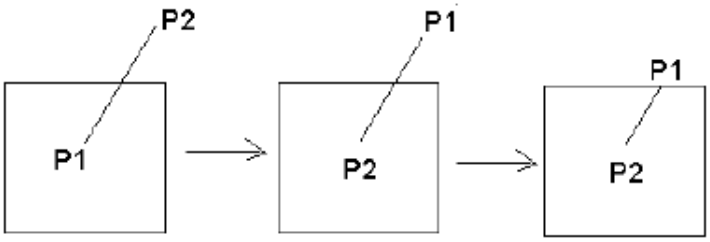
3) altfel: se intersecteaza segmentul P1-P2 cu dreptunghiul de decupare

- Alegerea laturii cu care se intersecteaza segmentul P1-P2: pe baza codului varfului P1, care trebuie sa fie situat in afara dreptunghiului de decupare.

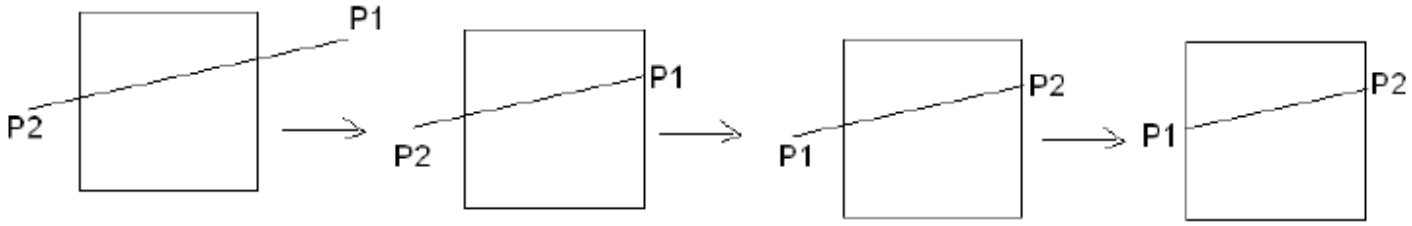
cod1 & 1 != 0



- Se muta P1 in punctul de intersectie
- In iteratia urmatoare cod1&cod2 != 0
  → segmentul este rejectat



cod1 & 4 != 0



Calculul punctelor de intersectie

Intersectia cu latura situata pe dreapta y = ymax

ymax=y1+t(y2-y1) de unde, t=(ymax-y1)/(y2-y1)

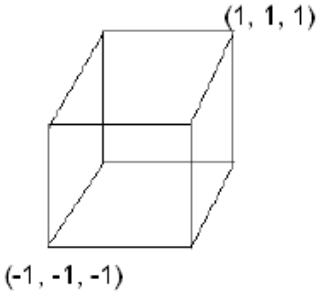
x=x1+t(x2-x1) → **xi=x1+(ymax-y1)/m**

Intersectia cu latura situata pe dreapta x = xmin

xmin=x1+t(x2-x1), deci t=(xmin-x1)/(x2-x1)

y=y1+t(y2-y1) → **yi=y1+(xmin-x1)*m**

Generalizarea algoritmului Cohen-Sutherland pentru decuparea vectorilor 3D(1)



Faţă 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 visual canonic sunt necesari 6 biti.

De ex. se poate face conventia:

b0 = 1 pentru puncte (x,y,z) cu x < -1

b1 = 1 pentru puncte (x,y,z) cu x > 1

b2 = 1 pentru puncte (x,y,z) cu y < -1

b3 = 1 pentru puncte (x,y,z) cu y > 1

b4 = 1 pentru puncte (x,y,z) cu z < -1

b5 = 1 pentru puncte (x,y,z) cu z > 1

daca: cod(P1) ==0 && cod(P2)==0
    segment acceptat trivial

daca (cod(P1) & cod(P2)) != 0
    segment rejectat trivial
altfel
    se intersecteaza segmentul cu volumul

1. Diferentele dintre proiectiile oblice si axonometrice

Asemanari :

- Proiectorii sunt drepte paralele de directie data: directia de proiectie
- Sunt transformari afine (conserva paralelismul liniilor)
- Unghiurile se conserva doar pentru feţele obiectului paralele cu planul de proiectie.

Oblice :

- planul de proiectie este perpendicular pe o axa principala
- proiectorii nu sunt perpendiculari pe planul de proiectie
- fetele paralele cu planul de proiectie se proiecteaza fara alterare unghiuri & laturi

Axonometrice :

- planul de proiectie este oarecare
- proiectorii = paralele ce trec prin varfurile obiectului proiectat si au directia specificata => sunt perpendiculari
- redau mai multe fete ale obiectului proiectat
- sunt de 3 tipuri: izonometrice, dimetrice, trimetrice

2. Care sunt parametrii care definesc sistemul de coordonate observator. Care sunt valorile implicite ale acestor parametrii in OpenGL?

Transformare vizuala: coordonate globale -> coordonate observator.

Sistemul de coordonate observator: atasat camerei virtuale

❖ **Sistem de coordonate 3D dreapta, definit prin 3 parametri**:

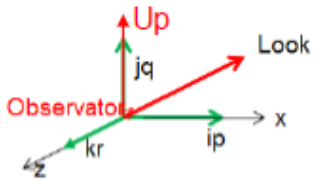
- Originea sa: **pozitia camerei** in spatiul coordonatelor globale **(xo, yo, xo).**
- **Directia in care priveste observatorul** (spre scena 3D) **– vectorul Look**, reprezinta normala la planul de vizualizare (**N**).
- Directia “sus” a planului de vizualizare (**vectorul Up**), care determina directia axei OY a sist. de coordonate observator: reprezinta **rotatia camerei in jurul axei privirii (Look)**

Functia OpenGL: glm::lookAt(glm::vec3(xo,yo,zo), glm::vec3(Px,Py,Pz), glm::vec3(Upx,Upy,Upz));

Valorile implicite:

Sistemul de coordonate observator implicit in OpenGL

- poziţia observatorului (camerei): **originea sistemului de coordonate globale**
- direcţia în care priveşte observatorul: **direcţia negativa a axei OZ**
- direcţia sus a planului de vizualizare: **direcţia pozitivã a axei OY**



Cu aceste valori implicite, transformarea de vizualizare este transformarea identica.

3. Algoritmul z-buffer. Descriere. Cand se executa? Depth complexity

- Algoritm de eliminare a partilor nevizibile ale primitivelor grafice.

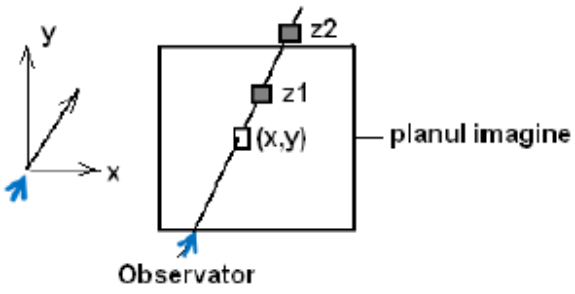
- Executat de GPU, fiind integrat in procesul de rasterizare a primitivelor (dupa transformarea în poarta de afişare a vârfurilor)

  - Observatorul este la infinit, pe axa z negativa

  -Asupra primitivelor se efectueaza proiectie ortografica in planul XOY

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

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



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

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

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

* calculeaza culoarea fragmentului

//testul de vizibilitate (adancime)

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

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

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

}

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

4. Reflexia difuza a luminii intr-un punct al unei suprafete 3D

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

$$I_d = I_{sursa} * k_d * \cos(i) \qquad 0 <= i <= \pi/2$$

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

k_d este coeficientul de difuzie a luminii incidente

I_d este intensitatea luminii reflectate difuz de suprafata

- Dacă i este mai mare ca π/2, suprafaţa nu primeşte lumină de la sursă (sursa de lumină se află în spatele suprafeţei).

- Stiind că

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

rezulta ecuatia care modeleaza lumina reflectata difuz:

- $Id\lambda = Ia\lambda * ka\lambda + fat * Isursa\lambda * kd\lambda * (Lu \cdot Nu)$

$Cd = [IdR, IdG, IdB]$ – culoarea luminii reflectate difuz

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

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

5. Comparatie intre modelul Gourraud si modelul Phong.

Dezavantaje Phong fata de modelul Gouraud

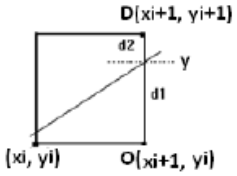
* Componentele Nx, Ny, Nz ale normalei unui fragment se obţin printr-un calcul incremental (analog cu cel folosit pentru calculul culorilor in modelul Gouraud) dar, pentru folosirea în calculul culorii, normala trebuie să fie normalizată.
* De asemenea, vectorii L (catre sursa de lumina) si V (catre observator), folositi in modelul de iluminare local la nivel de fragment, sunt calculati si normalizati in programul fragment shader pentru a fi folositi in calculul culorii fragmentului.
* Calcule mai complexe la nivel de fragment decat in modelul Gouraud

Avantaje Phong fata de modelul Gouraud

* Permite redarea reflexiei speculare in orice fragment al unui poligon
* Reduce mult efectul de banda Mach

6. Algoritmul Bresenham pentru rasterizarea vectorilor. Alegerea punctelor spatiului discret.Deducerea valorii expresiei de test. Implementare in C.

- Contine numai operatii cu numere intregi. **Este definit pentru vectori din primul octant**
- Un vector face parte din"primul octant" daca panta sa, m, satisface conditia 0<m<1.
- Coordonatele pixelilor de pe traseul vectorului se obtin prin calcul incremental.
- Pentru fiecare valoare a lui x, de la xmin_vector la xmax_vector, se alege acel punct al spaţiului discret care este mai apropiat de punctul de pe vectorul teoretic.

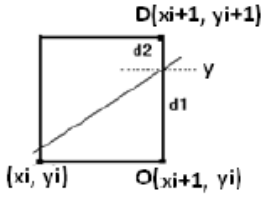


- Fie m=(y2-y1)/(x2-x1) panta vectorului,
- (x_i, y_i) ultimul punct al spaţiului discret ales la executia algoritmului
- d1 distanţa de la punctul de pe vectorul teoretic, (x_i+1, y), la punctul $O(x_i+1,y_i)$
- d2 distanţa de la punctul de pe vectorul teoretic la punctul $D(x_i+1,y_i+1)$.
- O si D sunt adrese de pixeli (puncte ale spatiului discret)

➢ Următorul punct al spatiului discret ales pentru aproximarea vectorului va fi:

- O dacă d1<d2, D în caz contrar.
- Dacă d1=d2 se poate alege oricare dintre cele două puncte.

- Exprimăm diferetţa d1-d2:



$y=m*(x_i+1)+b$ este ordonata punctului de pe vectorul teoretic

$d1= y - y_i= m*(x_i+1)+b-y_i$

$d2= y_i + 1-y= y_i+1-m*(x_i+1)-b$

$d1- d2= 2*m*(x_i+1)-2*y_i+2*b -1$

- Se înlocuieşte m cu dy/dx apoi se înmulţeşte în ambele părţi cu dx. Rezultă:

$t_i= (d1-d2)*dx= 2*dy*(x_i+1)-2*dx*y_i+2*b*dx-dx = 2*dy*x_i - 2*dx*y_i + 2*b*dx - dx + 2*dy$

- t_i reprezintă eroarea de aproximare în pasul i: pe baza sa se alege urmatorul punct al spatiului 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:

$$t_{i+1} = 2*dy*x_{i+1} - 2*dx*y_{i+1} + 2*b*dx - dx +2*dy$$

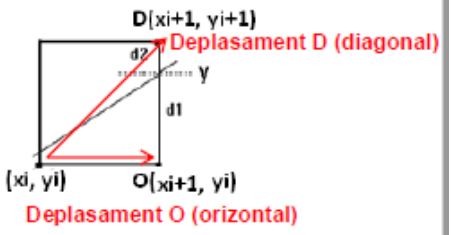
$t_i=(d1-d2)*dx = 2*dy*x_i - 2*dx*y_i + 2*b*dx-dx + 2*dy$, $(dx>0)$

(1) Dacă $t_i <= 0$, se alege punctul O, deci $x_{i+1} = x_i+1$ şi $y_{i+1} = y_i$

Rezultă:

$t_{i+1} = 2*dy*(x_i+1) - 2*dx*y_i + 2*b*dx - dx + 2*dy$

sau **$t_{i+1} = t_i + 2*dy$**



(2) Dacă $t_i > 0$, se alege punctul D, deci $x_{i+1} = x_i + 1$ şi $y_{i+1} = y_i + 1$

Rezultă:

$t_{i+1} = 2*dy*(x_i+1) - 2*dx*(y_i+1) + 2*b*dx - dx + 2*dy$

sau **$t_{i+1} = t_i + 2*dy - 2*dx$**

- Valoarea variabilei de test se obtine prin calcul incremental: adunarea unei constante intregi

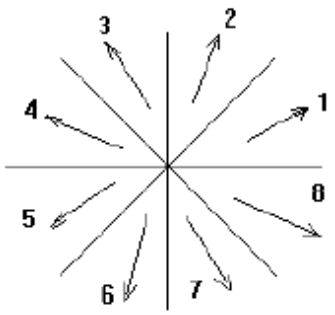
Implementare in C:

```
void Bres_vect(int x1, int y1, int x2, int y2)
{ //pentru vectori cu panta cuprinsă între 0 şi 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);
  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);
   }
  putpixel(x2,y2);
}
```

Generalizarea algoritmului Bresenham pentru vectori de orice panta

▪ Vectorii definiţi în planul XOY pot fi clasificaţi, pe baza pantei, în opt clase geometrice, numite **"octanţi"**

▪ Un vector care apartine unui octant O are 7 vectori simetrici in ceilalti 7 octanti



dx=x2 – x1 şi dy=y2 – y1

octantul 1: dx>0 şi dy>0 şi dx>=dy;

octantul 2: dx>0 şi dy>0 şi dx<dy;

octantul 3: dx<0 şi dy>0 şi abs(dx)<dy;

octantul 4: dx<0 şi dy>0 şi abs(dx)>=dy;

octantul 5: dx<0 şi dy<0 şi abs(dx)>=abs(dy);

octantul 6: dx<0 şi dy<0 şi abs(dx)<abs(dy);

octantul 7: dx>0 şi dy<0 şi dx<abs(dy);

octantul 8: dx>0 şi dy<0 şi dx>=abs(dy);

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),

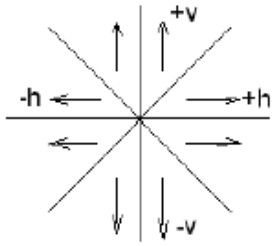
-v, deplasamentul vertical în jos (în sensul descrescător al axei y),

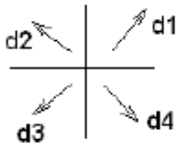
d1, deplasamentul diagonal dreapta-sus,

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

Corespondenţa între alegerea curentă într-un pas al algoritmului Bresenham (punctul O sau punctul D) şi deplasamentul echivalent în fiecare octant:

1. **Care dintre urmatoarele afirmatii sunt corecte:**

a. Algoritmul z-buffer necesita rasterizarea poligoanelor scenei 3D in ordinea descrescatoare a distantei lor fata de observator
Fals, sunt rasterizate in ordinea in care sunt trimise.

b. Bufferul z memoreaza coordonatele z ale fragmentelor de primitive care sunt vizibile intr-o imagine
Adevarat, scopul algoritmului este de a elimina partile nevizibile ale primitivelor grafice.

c. Bufferul z se actualizeaza pentru fiecare fragment rezultat din rasterizarea primitivelor dintr-un cadru imagine
Fals, se actualizeaza doar daca trece testul de adancime.

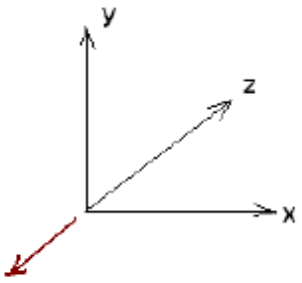
d. Algoritmul z-buffer este exectutat de procesoarele placii grafice
Adevarat, este executat pe GPU si integrat in procesul de rasterizare a primitivelor.

2. **Determinarea fetelor auto-obturate ale poliedrelor convexe. Conditia de vizibilitate atunci cand determinarea este efectuata:**

a) in coordonate de decupare

3. In sistemul coordonatelor de decupare (dupa aplicarea transformarii "model-view-projection")

In cazul 3:



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

Conditia de vizibilitate devine: nz <0

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

Observatorul este la infinit, pe axa z negativa.

b) in coordonate globale

Produsul scalar:

$N \bullet O = |N| * |O| * \cos(u) \rightarrow \cos(u) = N \bullet O / (|N| * |O|) = Nu \bullet Ou$

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

3. Aproximarea reflexiei speculare a luminii intr-un punct al unei

Pentru eliminarea feţelor auto-obturate se apeleaza urmatoarele funcţii 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 feţelor din faţă (in spatiul coordonatelor globale): trigonometrica/sensul acelor de ceas

suprafete 3D, luminata de o sursa punctiforma (modelul empiric)

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

$$I_{s\lambda} = I_{sursa\lambda} * w(i,\lambda) * \cos(\alpha)^n$$

w(i, λ) este funcţia de reflectanţă, i - unghiul de incidenţă iar λ - lungimea de undă a luminii incidente

In practica, w(i, λ) este înlocuită cu o constantă determinată experimental, numită **coeficientul de reflexie speculară al materialului, notat** $k_{s\lambda}$.

Modelul practic al reflexiei speculare:

$$I_{s\lambda} = I_{sursa\lambda} * k_{s\lambda} * \cos(\alpha)^n$$

$\cos(\alpha) = R \cdot V / (|R| \cdot |V|) = Ru \cdot Vu$

Rezulta:

$$I_{s\lambda} = I_{sursa\lambda} * fat * k_{s\lambda} * (Ru \cdot Vu)^n$$

Pentru a include si cazul in care α = 90:

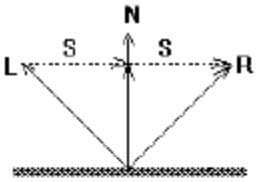
$$I_{s\lambda} = I_{sursa\lambda} * fat * k_{s\lambda} * \max((Ru \cdot Vu)^n, 0)$$

Reflexia speculara nu poate avea loc daca in punctul considerat nu se primeste lumina de la sursa: $Is\lambda = lum * Isursa\lambda * fat * ks\lambda * \max((Ru \cdot Vu)^n, 0)$

$lum = 1$ daca $(Lu \cdot Nu) > 0 \quad (0 <= i < \pi/2)$

$= 0$ altfel

Calculul directiei luminii speculare



- Vectorul R este simetricul vectorului L faţă de N.
- Proiecţia lui Lu pe N este: Nu * cos(i)
- R = Nu*cos(i) + S ,
- Nu*cos(i) = Lu + S → S = Nu*cos(i) – Lu

Rezulta:

$R = 2Nu*\cos(i) - Lu = 2Nu*(Lu \cdot Nu) - Lu$

4. Calculul culorilor fragmentelor la afisarea suprafetelor 3D folosind modelul Phong. Avantajele si dezavantajele modelului Phong in comparatie cu alte modele.

- La momentul rasterizarii, fiecarui vârf al unui poligon îi este asociata normala în vârf , care poate fi calculata la fel ca pentru modelul Gouraud. Normalele sunt calculate si asociate varfurilor în programul de aplicatie.
- Pentru fiecare fragment rezultat din rasterizarea unui poligon, GPU calculează o normala prin interpolare liniara intre normalele varfurilor.
- Culoarea pentru fiecare fragment interior poligonului se obtine pe baza normalei interpolate, folosind un model de iluminare local.
- La rasterizarea unui poligon se calculeaza o normala pentru fiecare fragment astfel:

a) prin interpolarea liniara a normalelor varfurilor, pentru fragmentele de pe laturi (la fel ca in modelul Gouraud, pentru culori);

b) prin interpolare liniara intre normalele capetelor fiecarui segment interior, pentru fragmentele interioare poligonului (la fel ca in modelul Gouraud, pentru culori).

1. **Care dintre urmatoarele NU este o transformare de proiectie afina? (afina = conserva paralelismul liniilor)**

   a) Proiectia oblica
   b) Proiectia perspectiva
   c) Proiectia axonometrica
   d) Proiectia ortografica

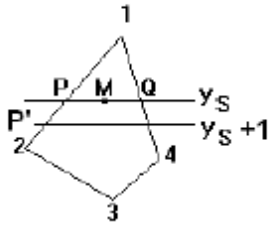
2. **Calculul culorilor fragmentelor la afisarea suprafetelor 3D folosind modelul Gourand. Avantajele si dezavantajele modelului Gourand in comparatie cu alte modele.**

2. La rasterizarea unui poligon, GPU calculeaza culoarea fiecarui fragment rezultat astfel:

   - prin interpolarea liniara a culorilor varfurilor, pentru fragmentele de pe laturi

   - prin interpolare liniara intre culorile capetelor fiecarui segment interior, pentru fragmentele interioare poligonului.

   Culoarea fragmentului este intrare pentru programul fragment shader.



Interpolarea liniara

$y = y1 + t(y2 - y1) \rightarrow tP = (yS - y1)/(y2 - y1)$

$IP = I1 + tP(I2 - I1) = I1 + (I2 - I1)(yS - y1) / (y2 - y1)$

$IQ = I1 + (I4 - I1)(yS - y1) / (y4 - y1)$

$x = xP + t(xQ - xP) \rightarrow tM = (xM - xP)/(xQ - xP)$

$IM = IQ + tM(IQ - IP) = IQ + (IQ - IP)(xM - xP) / (xQ - xP)$

Calculul incremental al culorilor:

$IP' = I1 + (I2 - I1)(yS+1 - y1) / (y2 - y1) = IP + (I2 - I1) / (y2 - y1)$

$IP' = IP + C_{1\text{-}2}$, $C_{1\text{-}2}$ – o constanta a laturii 1-2

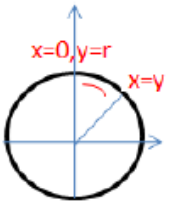
analog, **$IQ' = IQ + C_{1\text{-}4}$**

M' (xM + 1, ys)

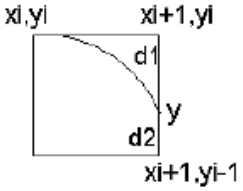
$IM' = IQ + (IQ - IP)(xM + 1 - xP) \rightarrow$ **$IM' = IM + C_{P\text{-}Q}$, $C_{P\text{-}Q}$ – o constanta a segmentului P-Q**

3. Algoritmul Bresenham pentru rasterizarea cercurilor: alegerea punctelor in spatiul discret, implementare in C, functia de afisare a punctelor de pe cerc

- Consideram cercul cu centrul în originea sistemului de coordonate si raza r.
- In cadrul algoritmului **se calculeaza numai punctele din octantul al 2-lea,** celelalte obtinandu-se prin simetrie.



- Se calculeaza punctele de pe cerc incepand cu (x=0,y=r) pana la (x=y), prin incrementarea lui x.
- Fie (x_i,y_i) ultimul punct al spatiului discret, ales pentru aproximarea cercului. Urmatorul punct va fi unul dintre (x_i+1,y_i) si (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 d1 si d2 distantele de la cele 2 puncte ale spatiului discret la punctul de pe cercul teoretic.

Intereseaza semnul diferentei (d1-d2):

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

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

$$d1' = y_i^2 - y^2 = y_i^2 - r^2 + (x_i+1)^2$$

$$d2' = y^2 - (y_i-1)^2 = r^2 - (x_i+1)^2 - (y_i-1)^2$$

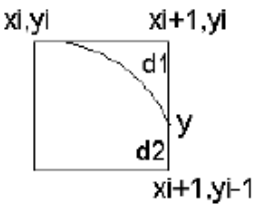
- Notam cu t_i eroarea de aproximare în pasul curent:

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

- Se obtine o relatie de recurenţă 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,

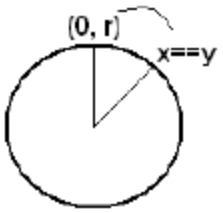
$$t_{i+1} = (y_i-1)^2 + 2*((x_i+1)+1)^2 + ((y_i-1)-1)^2 -2*r^2$$

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

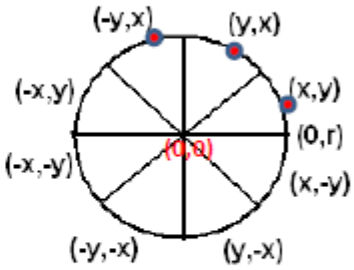
- Valoarea erorii de aproximare în primul pas:

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

```
void Bres_cerc(int xc, int yc, int r, RGB culoare)
{ int x,y,t;
  t=3-(r<<1);
  point(xc+r,yc,culoare); point(xc-r,yc,culoare);
  point(xc,yc+r,culoare); point(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);
  }
// pentru x==y se afiseaza 4 puncte simetrice
  point(xc+x,yc+y,culoare); point(xc+x,yc-y,culoare);
  point(xc-x,yc+y,culoare); point(xc-x,yc-y,culoare);
}
```



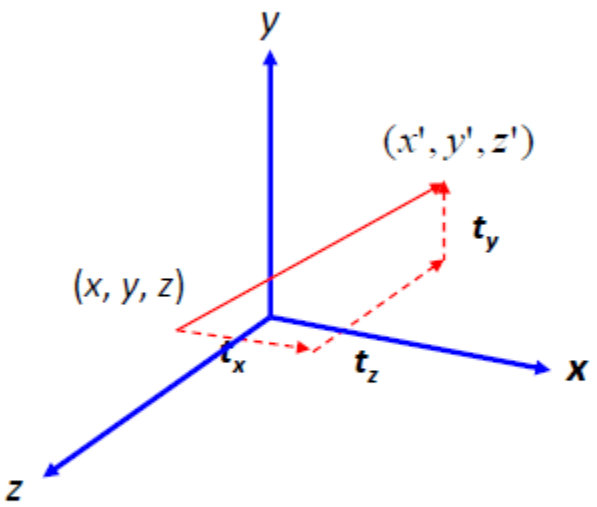

```
void punct_simetric(int xc,int yc,int x,int y, RGB culoare)
{ point(xc+x,yc+y,culoare);
  point(xc+x,yc-y,culoare);
  point(xc-x,yc-y,culoare);
  point(xc-x,yc+y,culoare);
  point(xc+y,yc+x,culoare);
  point(xc+y,yc-x,culoare);
  point(xc-y,yc-x,culoare);
  point(xc-y,yc+x,culoare);
}
```



4. Sa se scrie produsul matriceal prin care se poate exprima rotatia unui punct din spatiul 3D, in jurul unei drepte oarecare. Se va preciza efectul fiecarei matrici.

Se considera dreapta data printr-un punct (xd, yd, zd) si directia sa, D[a, b, c].
1. Translatie care face ca dreapta sa treaca prin origine: T(-xd, -yd, -zd)
2. Alinierea dreptei cu una dintre axele principale, de ex. cu axa OZ:
2.1. Rotatie in jurul axei OX, cu un unghi ux, prin care dreapta ajunge in planul XOZ: Rox(ux)
2.2. Rotatie in jurul axei OY, cu un unghi uy, prin care dreapta se suprapune pe axa OZ: Roy(uy)
3. Rotatia cu unghiul dat, u, in jurul axei pe care s-a aliniat dreapta: rotatie in jurul axei OZ : Roz(u)
4. Transformarea inversa celei din pasul 2:
4.1. Rotatie in jurul axei OY, cu unghiul –uy: Roy(-uy)
4.2. Rotatie in jurul axei OX, cu unghiul –ux: Rox(-ux)
5. Transformarea inversa celei de la pasul 1: T(xd, yd, zd)

Translaţia



$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$

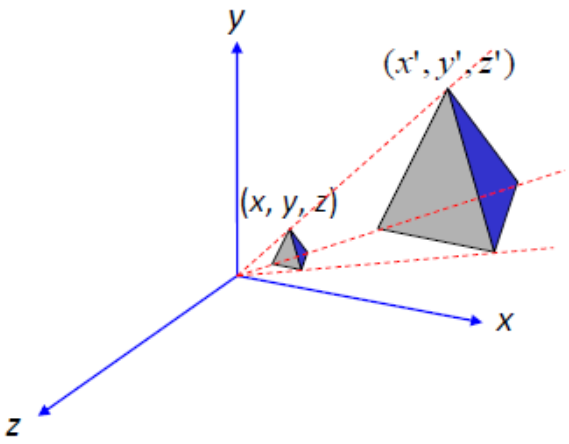
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$T(t_x, t_y, t_z)$

Scalarea faţă de origine

$S_x = S_y = S_z \quad \Rightarrow \quad$ scalare uniforma

altfel, scalare neuniforma

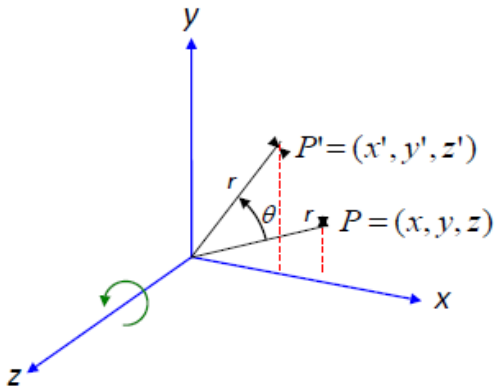


$$x' = x \cdot s_x$$
$$y' = y \cdot s_y$$
$$z' = z \cdot s_z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$S(s_x, s_y, s_z)$

Rotaţia pozitivă (trigonometrica) în jurul axei oz



Este o rotaţie într-un plan de z constant (prin rotaţie nu se modifică coordonata z).

In planul XOY:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z = 0$$

In general:

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$R_z(\theta)$

2. **Rotatia in jurul unei axe paralele cu o axa a sistemului de coordonate**:
   1. Translatia obiectului astfel incat axa de rotatie sa se suprapuna peste o axa a sistemului de coordonate.
   2. Rotatia obiectului in jurul axei sistemului de coordonate.
   3. Translatia inversa celei din pasul 1.

   Rezulta:

   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

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