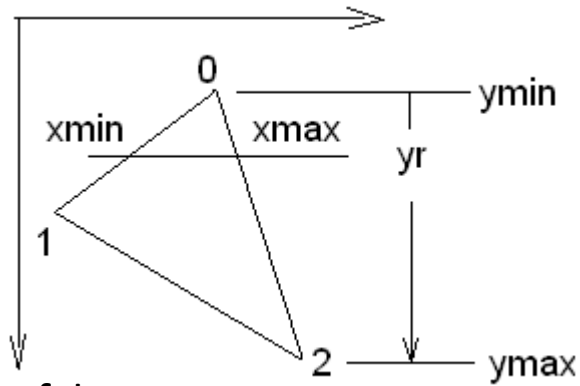
Rasterizarea suprafetelor triunghiulare

Prof. univ. dr. ing. Florica Moldoveanu

Rasterizarea suprafetelor triunghiulare- algoritmul



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. y0 = ymin;

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 ymin-triunghi la ymax-triunghi si se memoreaza in 2 tablouri, XMIN, XMAX;
5. Se afiseaza pixelii interiori triunghiului (intre xmin si xmax pe fiecare linie raster); return true;

}

Algoritmul: pasii 1 si 2

1. Se sorteaza varfurile triunghiului crescator dupa coordonata y, a.i. y0 = ymin

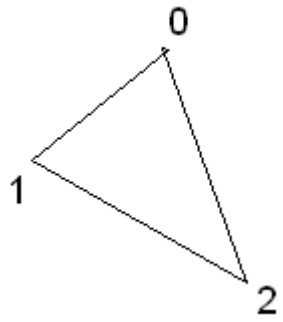
if(y1 < y0) { t= y1; y1 = y0; y0 =t;}

if(y2 < y0) { t= y2; y2 = y0; y0 =t;}

2. Daca triunghiul este degenerat (varfurile sunt coliniare), return false

$$\det = \begin{vmatrix} x1 - x0 & y1 - y0 \\ x2 - x0 & y2 - y0 \end{vmatrix}$$

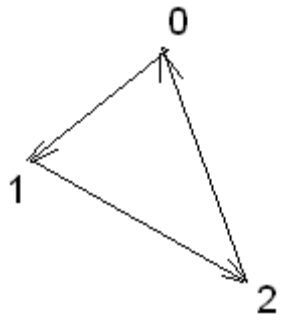
$$= (x1 - x0)(y2 - y0) - (x2 - x0)(y1 - y0)$$



if(det==0) return false

Pasul 3: orientarea conturului

3. Se orienteaza conturul in sens trigonometric (0-1-2: sens trigonometric)



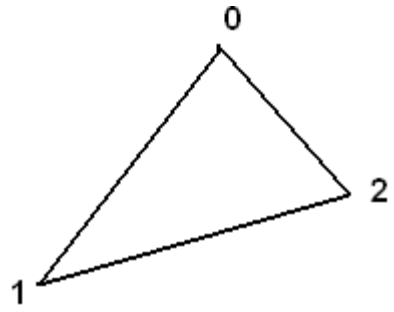
sens trigonometric

dx0 = x1 – x0; dx1 = x2 – x0; dy0 = y1 – y0; dy1 = y2 – y0;

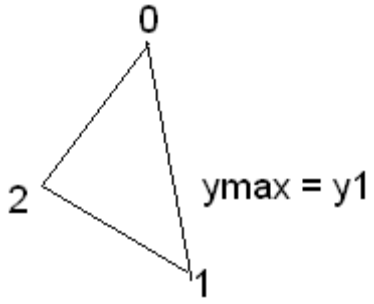
x1 < x0 , x2 > x0

dx0 < 0, dx1 > 0, dy0 > 0, dy1 > 0

det = dx0*dy1- dx1*dy0 **< 0**



sens trigonometric



sens anti-trigonometric

x1> x0, x2 < x0

dx0>0, dx1<0, dy0>0, dy1>0

det = dx0*dy1- dx1*dy0 **>0**

Se inverseaza varful 1 cu 2

Pasul 4: calculul extremitatilor segmentelor interioare(1)

4. Se calculeaza extremitatile segmentelor interioare suprafetei triunghiului

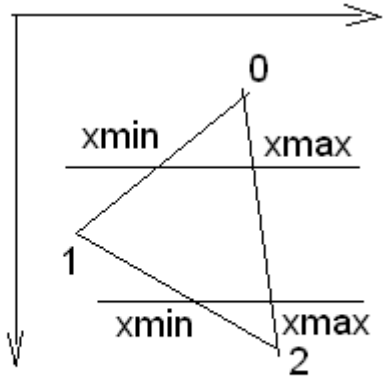
- Coordonatele xmin, xmax ale extremitatilor segmentelor interioare suprafetei triunghiului se memoreaza in 2 tablouri:

  int XMIN[H], XMAX[H]; // H este numarul de linii imagine

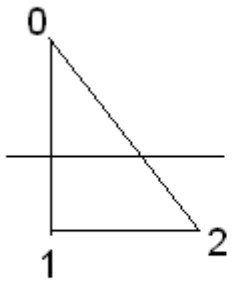
- Calculul extremitatilor este efectuat in functiile **CalculXmin()** si **CalculXmax()**;

Cazuri:

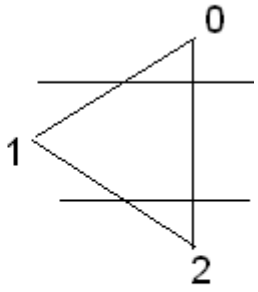
cazul I.



y0 < y1 < y2



x0 = x1



x0 = x2

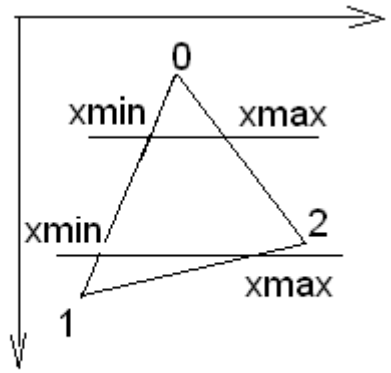
CalculXmin(x0, y0,x1, y1);

CalculXmin(x1, y1, x2, y2);

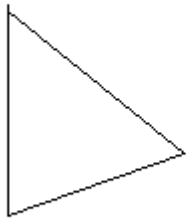
CalculXmax(x0, y0, x2, y2);

Pasul 4: calculul extremitatilor segmentelor interioare (2)

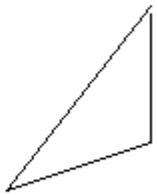
cazul II.



y0 < y2 < y1



x0 = x1



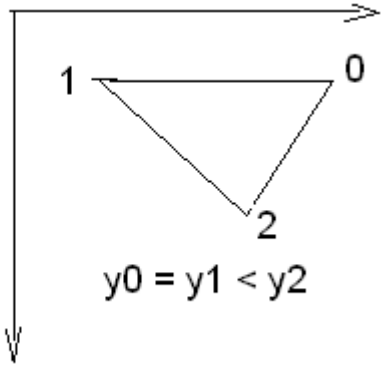
x0 = x2

CalculXmin(x0, y0, x1, y1);

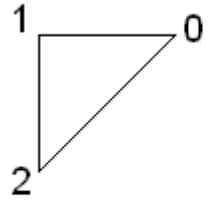
CalculXmax(x0, y0, x2, y2);

CalculXmax(x2, y2, x1, y1);

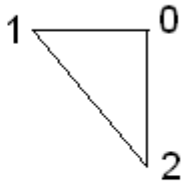
cazul III.



y0 = y1 < y2



x1 = x2

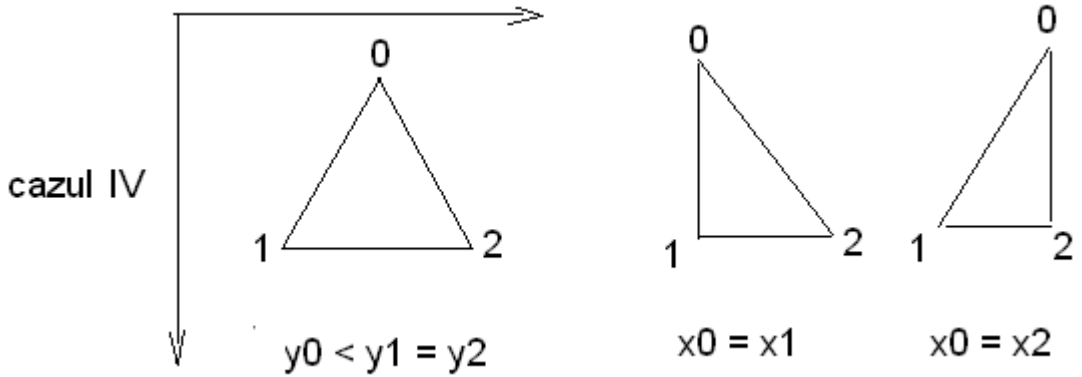


x0 = x2

CalculXmin(x1, y1, x2, y2);

CalculXmax(x0, y0, x2, y2);

Pasul 4: calculul extremitatilor segmentelor interioare (3)



CalculXmin(x0, y0, x1, y1);

CalculXmax(x0, y0, x2, y2);

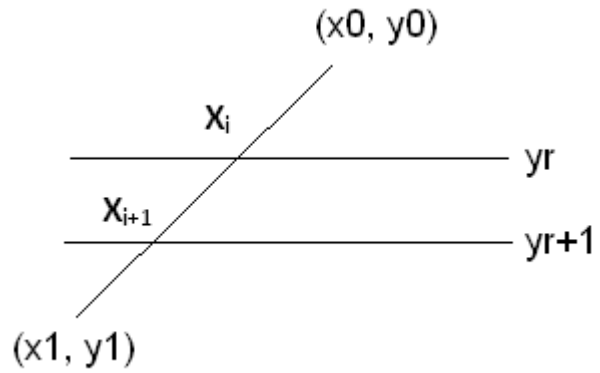
Calculul coordonatelor xmin, xmax

m = (y1 - y0)/ (x1 – x0) - panta laturii

$(yr+1 - yr)/ (x_{i+1} - x_i) = m \rightarrow x_{i+1} = x_i + (1/m)$

dx = 1.0/m - constanta a laturii

$x_{i+1} = x_i + dx \rightarrow$ 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 succesive 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”