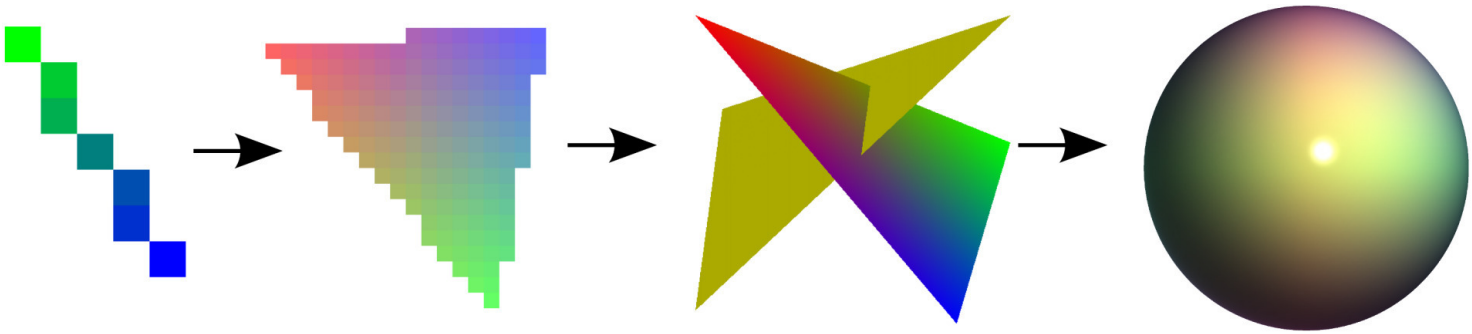
Rendu projectif



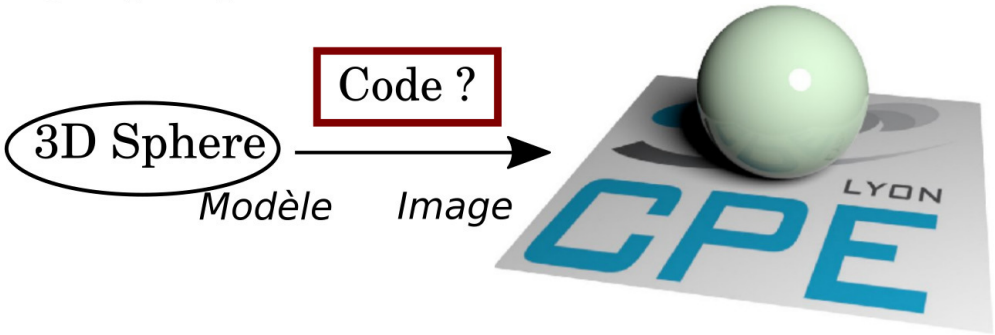
Objectif

Etre capable de réaliser une image à partir de données

sans bibliothèques:

~~Matlab~~
~~VTK~~
~~OpenGL~~
~~SDL~~
...

Etre capable de coder entièrement ce que fait une carte graphique



Ce que l'on sait faire

Structure de données d'images
= tableau de couleurs de taille Nx Ny

couleurs

pratique

```
struct couleur
{
    float r,g,b;
};
```

compact

```
struct couleur
{
    int r,g,b;
};
```

niveau de gris

```
struct couleur
{
    unsigned char r,g,b;
};
```

binaire

```
struct couleur
{
    float value;
};
```

```
struct couleur
{
    bool value;
};
```

tableau

statique

```
struct image
{
    couleur data[Nx][Ny];
};
```

```
struct image
{
    couleur data[Nx*Ny];
};
```

dynamique

```
struct image
{
    int Nx,Ny;
    std::vector<couleur> data;
};
```

Ce que l'on sait faire

Image avec accesseurs

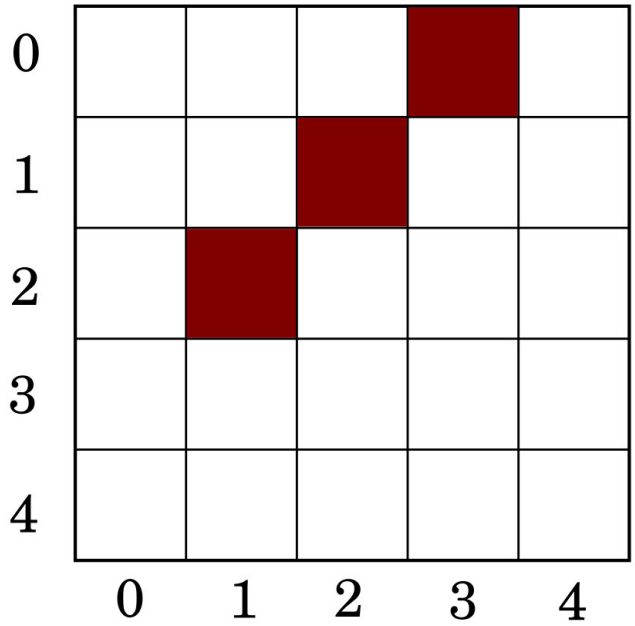
```
struct couleur
{
    float r,g,b;
};
struct image
{
    int Nx,Ny;
    std::vector<couleur> data;

    pixel const& operator()(int kx,int ky) const { return data[kx+Nx*ky] };
    pixel&       operator()(int kx,int ky)       { return data[kx+Nx*ky] };
};


int main()
{
    image im;
    // chargement image ...
    couleur p = im(4,7);          //acces
    im(4,7) = {0.5f,1.0f,0.0f};   //ecriture
    //...
}
```

Export en fichier .ppm pour visualisation

Ce que l'on sait faire

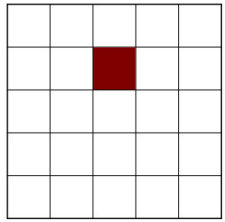
Quelles formes sait-on dessiner correctement sur une image

= savoir allumer précisément les bonnes coordonnées
en valeurs entières



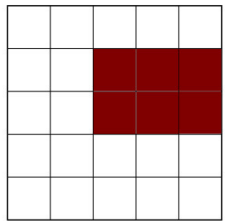
Ce que l'on sait faire

Un point



```
im(kx,ky)=...
```

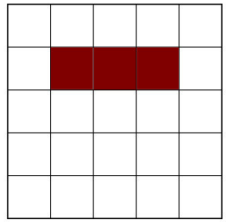
Un rectangle
(/toute l'image)



```
for(int kx=x0 ; kx<x0+Lx ; ++kx)
    for(int ky=y0 ; ky<y0+Ly ; ++ky)
        im(kx,ky)=...
```

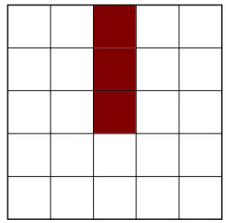
Ce que l'on sait faire

Ligne horizontale



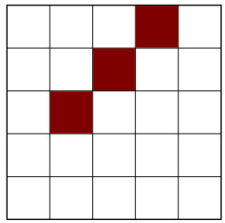
```
for(int kx=x0 ; kx<x1 ; ++kx)
    im(kx,y)=...
```

Ligne verticale



```
for(int ky=y0 ; ky<y1 ; ++ky)
    im(x,ky)=...
```

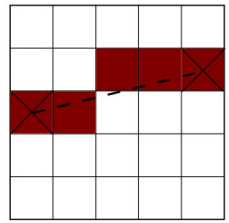
Ligne diagonale



```
for(int k=k0 ; k<k1 ; ++k)
    im(k,k)=...
```

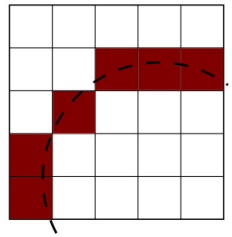
Moins facile

Segment quelconque



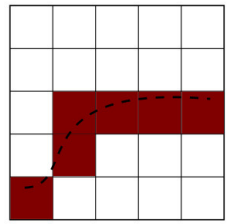
$$y = a\,x + b$$

Cercles



$$(x - x_0)^2 + (y - y_0)^2 = r^2$$

Courbes quelconque



$$f(x, y) = 0$$

Sans sur-echantillonner:

$$x^n = x^{n-1} + \Delta x$$
$$y^n = f(x^n)$$
$$im\,(int(x^n), int(y^n))$$

fonctionne
mais non optimal

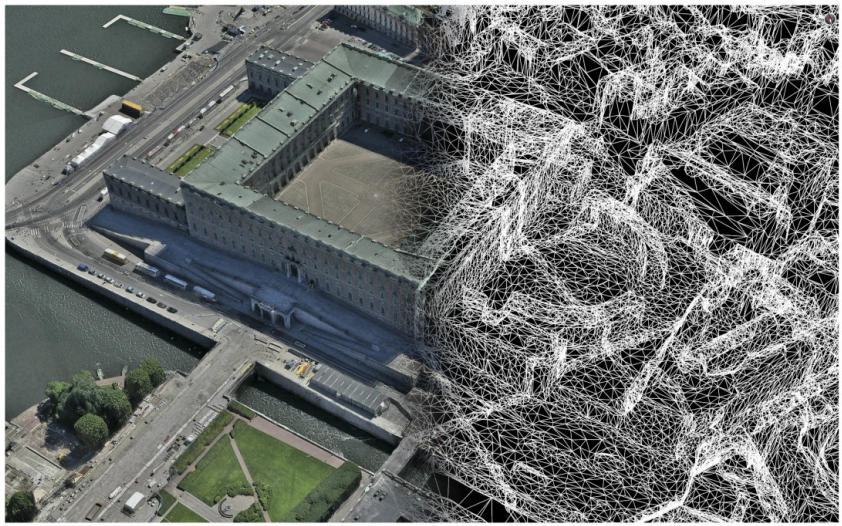
De quoi à t-on besoin
pour afficher une scène 3D ?

Scène 3D

Scène 3D = ensemble de maillages 3D

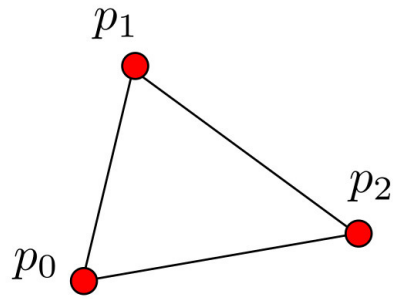
Maillage 3D = ensemble de triangles 3D

colorés, texturés, etc.



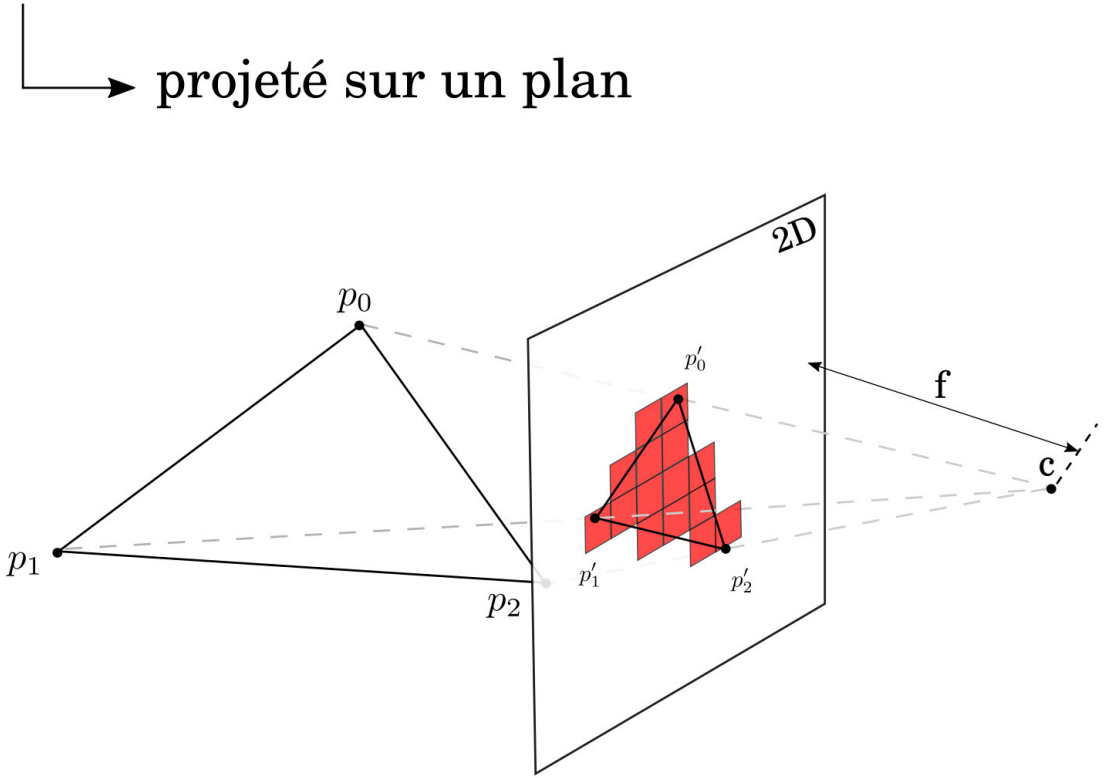
Il faut savoir afficher un triangle "3D"

$$(x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2)$$



Triangle 3D

Un triangle 3D est un triangle 2D vue en perspective

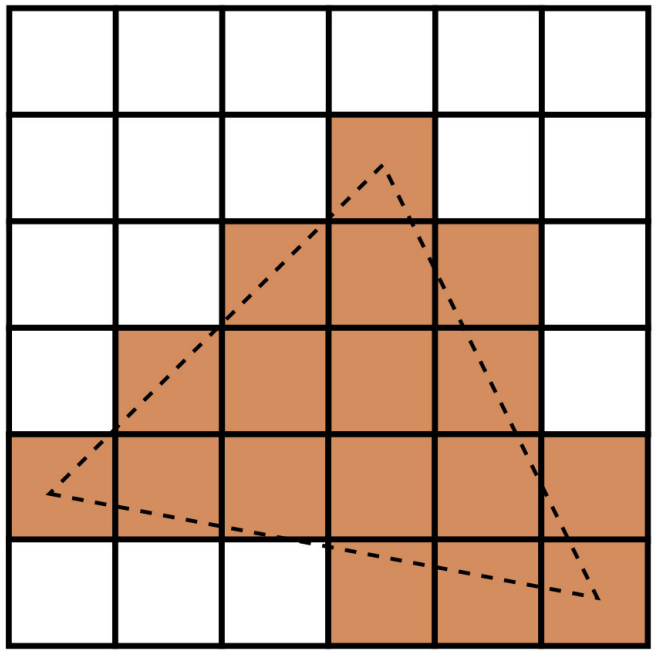


$p' = \mathrm{M}\, p$ M: matrice de projection

Triangle 2D

Il faut savoir afficher un triangle en 2D

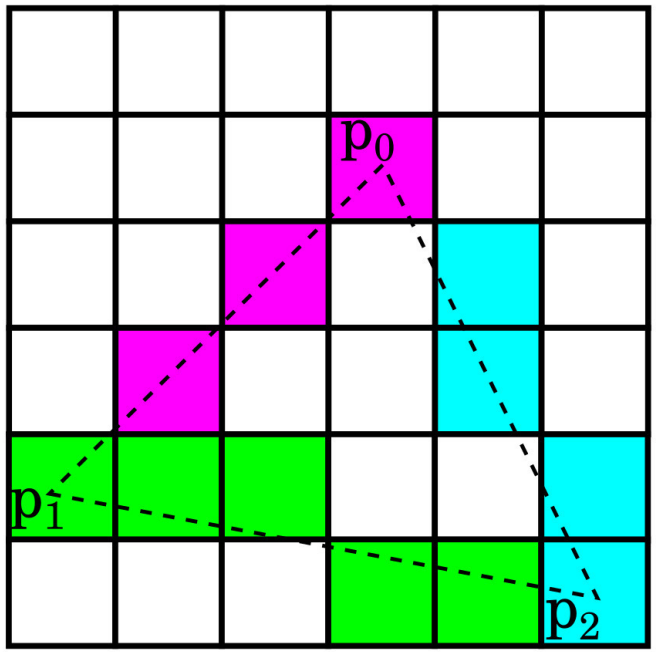
$(x_0, y_0), (x_1, y_1), (x_2, y_2)$



Triangle 2D

Afficher un triangle 2D

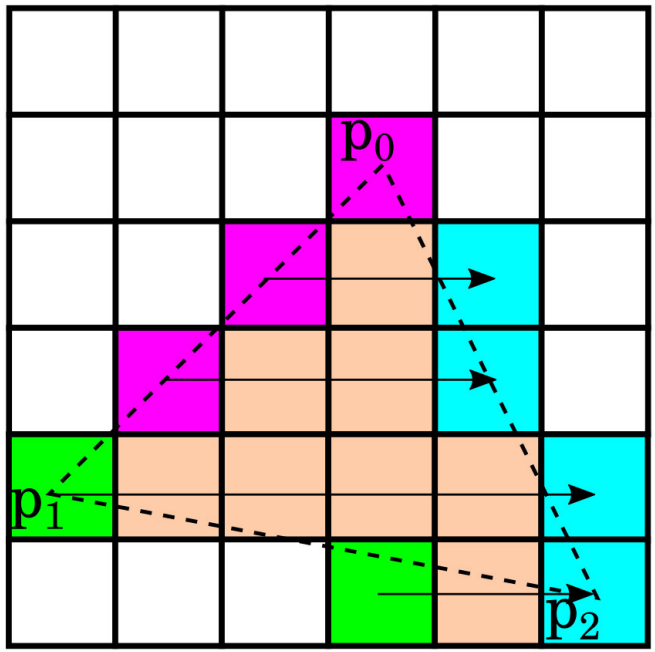
Etape 1: Afficher les segments des cotés $[p_0, p_1]$ $[p_1, p_2]$ $[p_2, p_0]$



Triangle 2D

Afficher un triangle 2D

Etape 2: Combler avec segments horizontaux



Triangle 2D

Afficher un triangle 2D

1. Afficher segments quelconques ✗
2. Afficher segments horizontaux ✓

=> Il faut savoir tracer un segment $(x_a, y_a), (x_b, y_b)$

=> Base de l'affichage en 3D

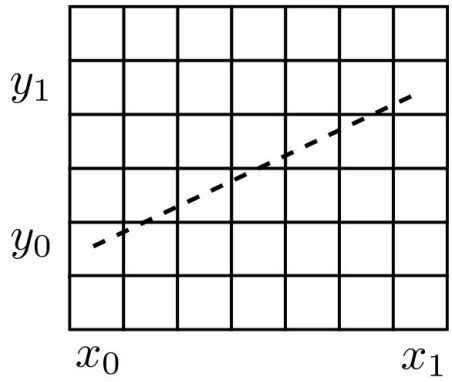
A faire 3x pour chaque triangles, 25x par secondes
=> Doit être efficace

Algorithme de Bresenham

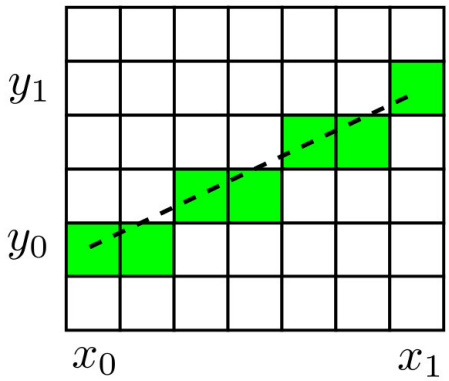
Tracé de segment discret

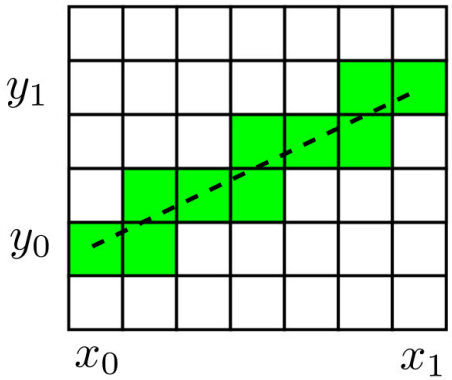
Segment discret

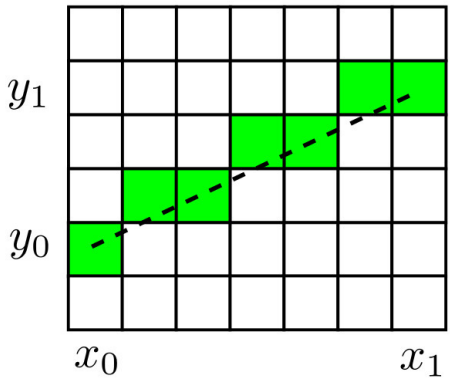
Plusieurs possibilités



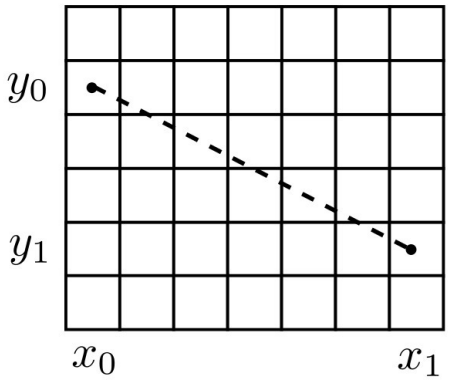
Quel est le "vrai" segment discret?





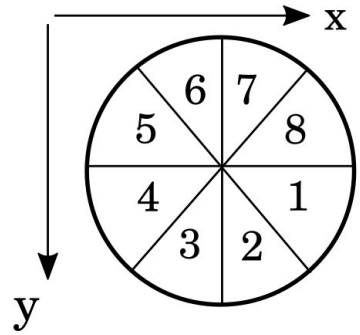


Segment discret



On suppose un segment du 1er quadrant

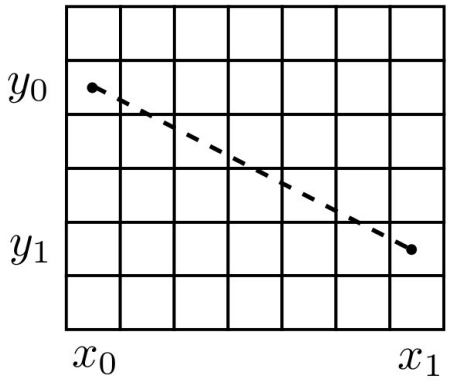
$$x_1 > x_0$$
$$y_1 > y_0$$
$$x_1 - x_0 > y_1 - y_0$$



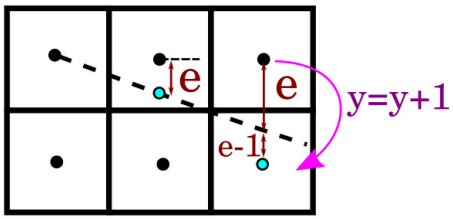
$$y = a\,(x - x_0) + y_0$$
$$a = \Delta y / \Delta x \qquad \Delta y = y_1 - y_0$$
$$\Delta x = x_1 - x_0$$

```
int dx = x1-x0;
int dy = y1-y0;

float a = dy/dx;

for(int x=x0 ; x<=x1 ; ++x)
{
    float y_true = a*(x-x0)+y0;
    int y = int(y_true+0.5f);
    im(x,y) = ...
}
```

Segment discret




```
int dx = x1-x0;
int dy = y1-y0;

float a = dy/dx;

for(int x=x0 ; x<=x1 ; ++x)
{
    float y_true = a*(x-x0)+y0;
    int y = int(y_true+0.5f);
    im(x,y) = ...
}
```



approximation flottant vers entier.

Idée: Analyser l'erreur e
Augmenter y de 1 lorsque l'erreur est trop grande

```
int dx = x1-x0;
int dy = y1-y0;

float a = dy/dx;
float e = 0.0f;

int y = y0;
for(int x=x0 ; x<=x1 ; ++x)
{
    im(x,y) = ...
    e += a;

    if( a>=0.5f )
    {
        y = y+1;
        e = e-1.0f;
    }
}
```

Segment discret

Optimiser en enlevant les calculs en flottants

```
int dx = x1-x0;
int dy = y1-y0;

float a = dy/dx;
float e = 0.0f;

int y = y0;
for(int x=x0 ; x<=x1 ; ++x)
{
    im(x,y) = ...
    e += a;

    if( a>=0.5f )
    {
        y = y+1;
        e = e-1.0f;
    }
}
```

Initialise l'erreur à -0.5 (comparaison à 0)
Multiplie tout par 2 dx (dénominateur)

```
int dx = x1-x0;
int dy = y1-y0;

int a = 2*dy;
int e = -dx;

int y = y0;
for(int x=x0 ; x<=x1 ; ++x)
{
    im(x,y) = ...
    e += a;

    if( e >= 0 )
    {
        y = y+1;
        e = e-2*dx;
    }
}
```

Algorithme de bresenham

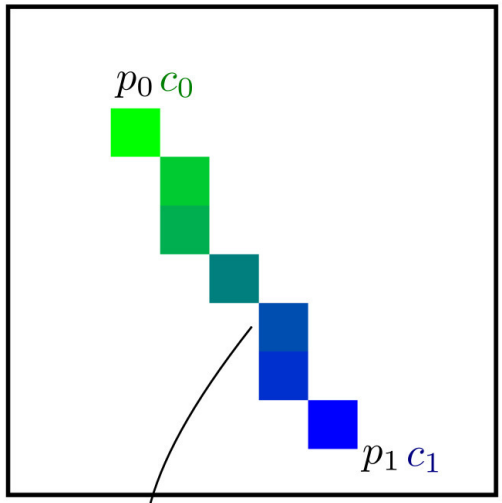
Avec prise en compte des quadrants

```
(x0,y0,x1,y1) = conversion_vers_1er_quadrant(x0_in,y0_in,x1_in,y1_in)
int const dx = x1_q-x0;
int const dy = y1-y0;

int const a = 2*dy;
int const coeff = 2*dx;

int erreur = -dx;
int y = y0;
for(int x=x0 ; x<=x1 ; ++x)
{
    (x_out,y_out) = conversion_vers_quadrant_origine(x,y);
    im(x_out,y_out) = ...

    erreur += a;

    if( erreur >= 0 )
    {
        ++y;
        erreur -= coeff;
    }
}
```

peut être optimisé pour les diagonales, horizontales, verticales, point, etc.

Segments en couleurs

On interpole entre les deux extrémités

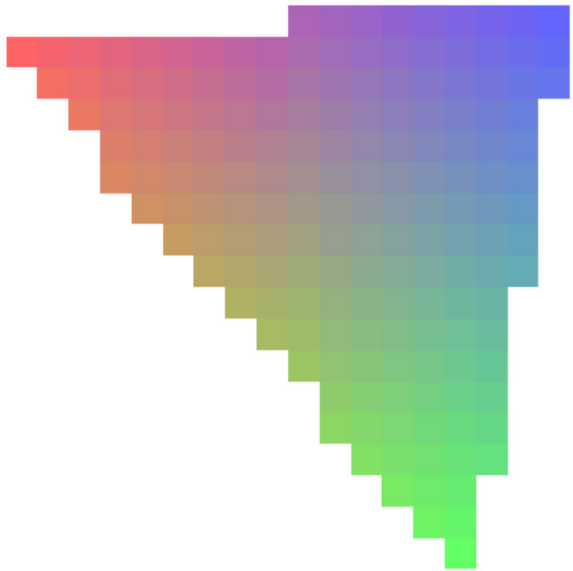


$$p_0 = (x_0, y_0, z_0)$$
$$p_1 = (x_1, y_1, z_1)$$
$$c_0 = (r_0, g_0, b_0)$$
$$c_1 = (r_1, g_1, b_1)$$

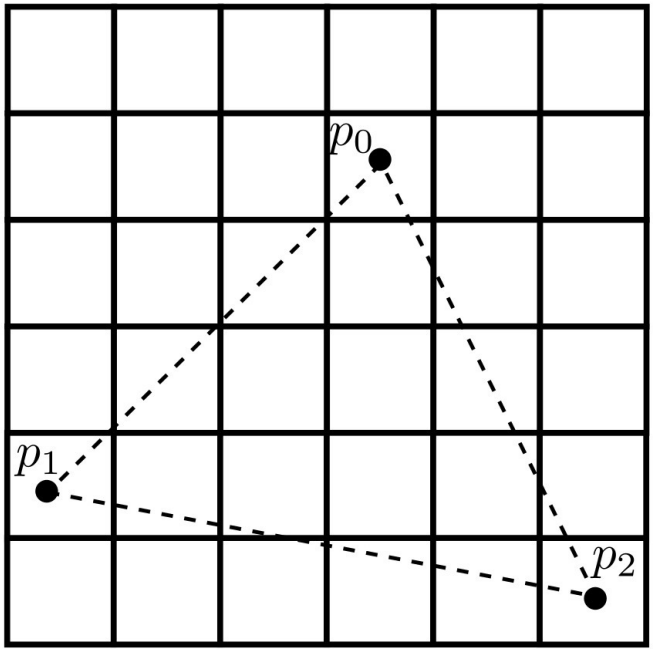
$$c = (1 - \alpha)\, c_0 + \alpha\, c_1 \qquad \alpha = \frac{\|p - p_0\|}{\|p_1 - p_0\|}$$

Pourrait être optimisé pour limiter le calcul flottant

Afficher un triangle plein



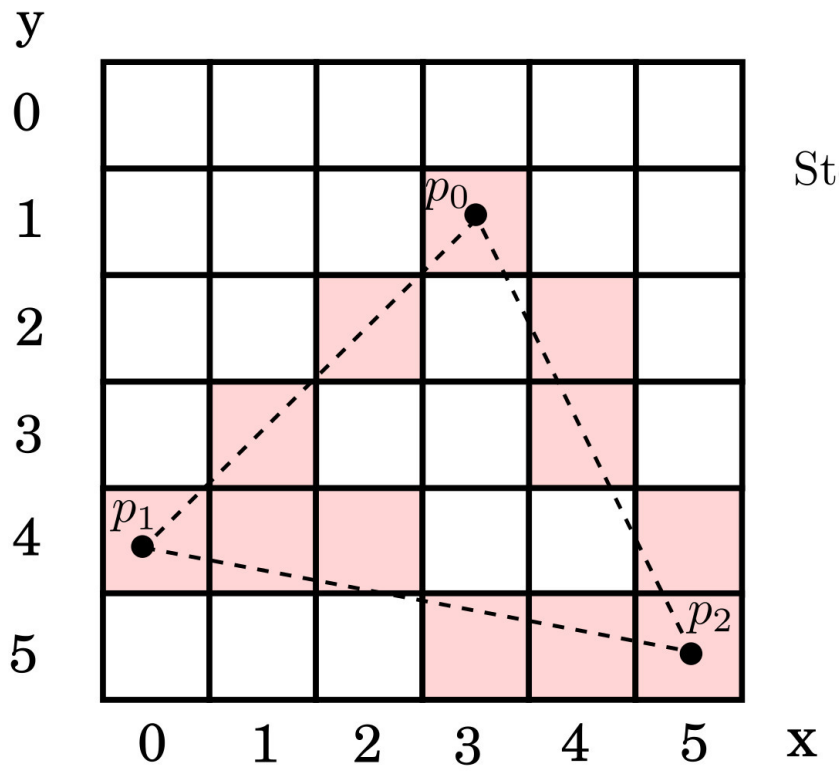
Algorithme scanline



Algorithme scanline

1. Parcours les 3 segments

$[p_0, p_1]$
$[p_1, p_2]$
$[p_2, p_0]$

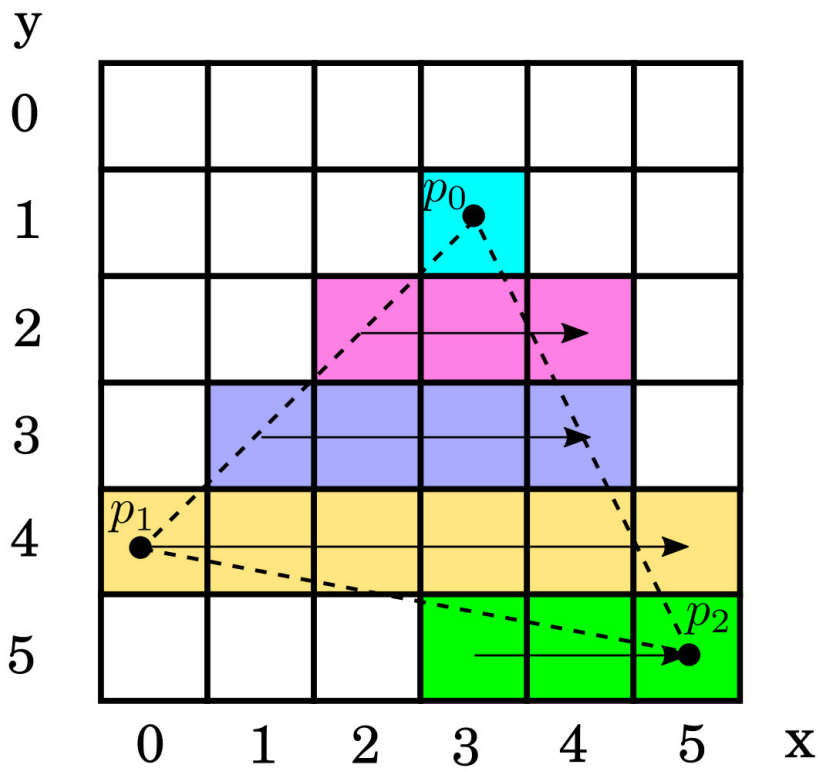


Stocke pour y donné, (x_{min}, x_{max})

y	x_{min}	x_{max}
1 :	3	3
2 :	2	4
3 :	1	4
4 :	0	5
5 :	3	5

Algorithme scanline

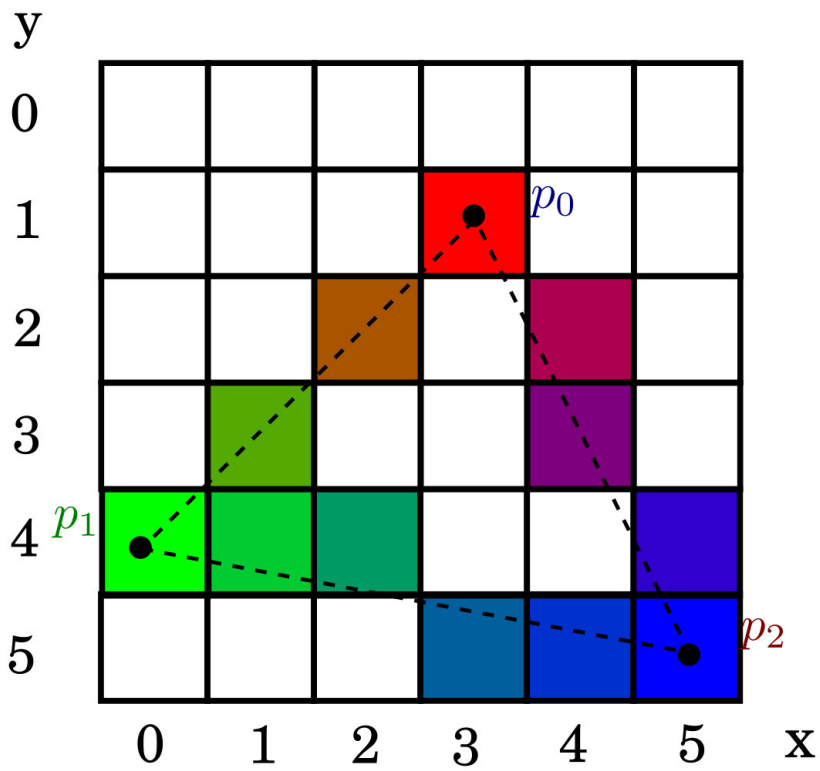
2. Affiche tous les segments horizontaux $[x_{min}, y]$ $[x_{max}, y]$



y	x_{min}	x_{max}
1 :	3	3
2 :	2	4
3 :	1	4
4 :	0	5
5 :	3	5

Triangle en couleurs

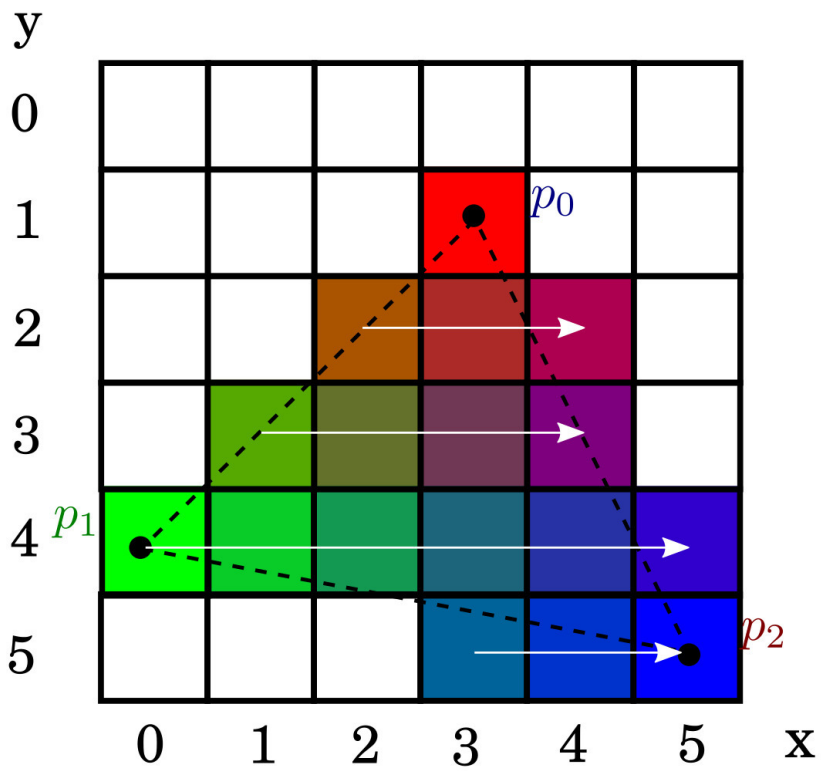
1. Enregistre en + les valeurs de couleur interpolés



y	x_{min}	x_{max}	v_{min}	v_{max}
1 :	3	3	c_0	c_0
2 :	2	4	v_{22}	v_{24}
3 :	1	4	v_b	v_{34}
4 :	0	5	c_1	v_{45}
5 :	3	5	v_{53}	c_2

Triangle en couleurs

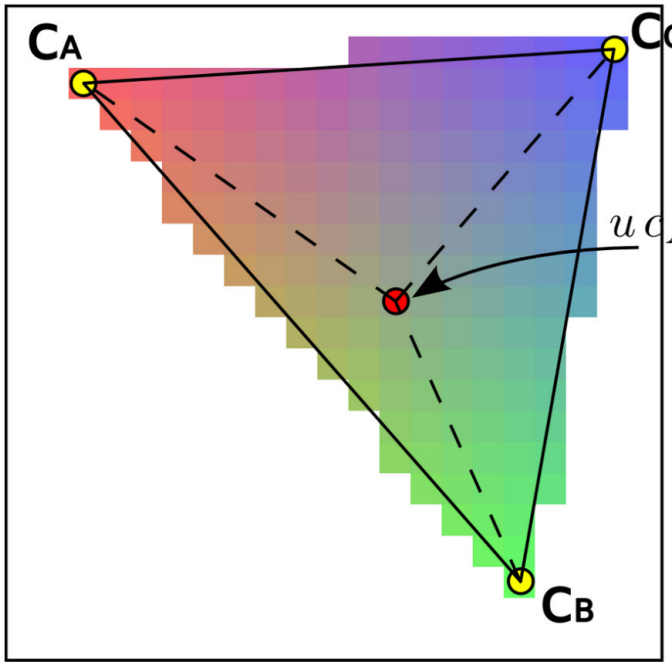
2. Segments horizontaux avec couleurs



y	x_{min}	x_{max}	v_{min}	v_{max}
1 :	3	3	c_0	c_0
2 :	2	4	v_{22}	v_{24}
3 :	1	4	v_b	v_{34}
4 :	0	5	c_1	v_{45}
5 :	3	5	v_{53}	c_2

Triangle en couleur

Autre solution possible: interpolation barycentrique



$$u\,c_A + v\,c_B + w\,c_C$$

$$\begin{cases} A = aire(\mathbf{x}_B - \mathbf{x}_A, \mathbf{x}_C - \mathbf{x}_A) \\ A_1 = aire(\mathbf{x}_C - \mathbf{x}_B, \mathbf{x} - \mathbf{x}_B) \\ A_2 = aire(\mathbf{x}_A - \mathbf{x}_C, \mathbf{x} - \mathbf{x}_C) \\ A_3 = aire(\mathbf{x}_B - \mathbf{x}_A, \mathbf{x} - \mathbf{x}_A) \end{cases}$$

avec $aire(\mathbf{v}_0, \mathbf{v}_1) = 1/2\|\mathbf{v}_0 \times \mathbf{v}_1\|$

$$\Rightarrow \begin{cases} \alpha = A_1/A \\ \beta = A_2/A \\ \gamma = A_3/A \end{cases}$$

Triangles en 3D

Rappel transformation affines

$$p' = \mathrm{R}\,p + \mathrm{t}$$

Rotation 3×3 — Translation (t_x, t_y, t_z)

$$p = (p_x, p_y, p_z)$$
$$p' = (p'_x, p'_y, p'_z)$$

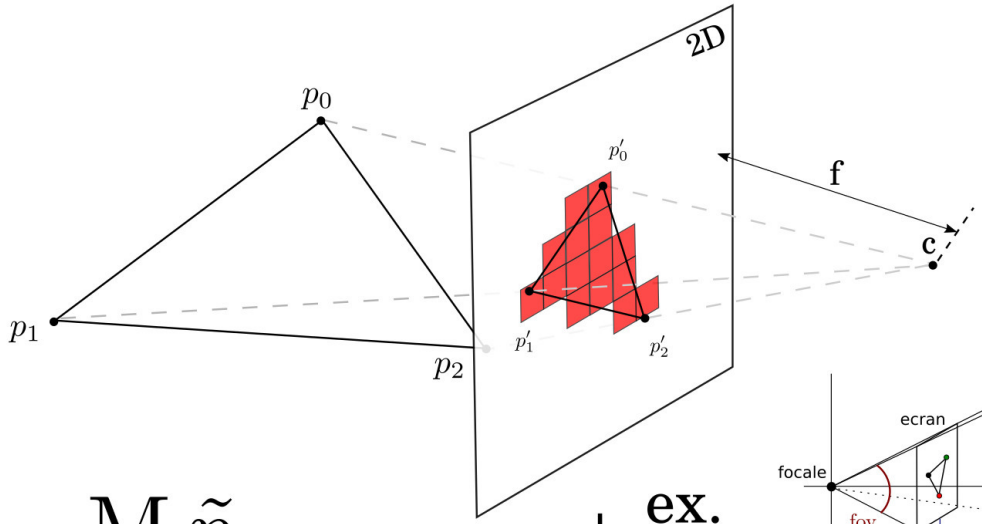
$$\tilde{p}' = \mathrm{M}\,\tilde{p}$$

4×4

$$\tilde{p} = (\tilde{p}_x, \tilde{p}_y, \tilde{p}_z, 1)$$
$$\tilde{p}' = (\tilde{p}'_x, \tilde{p}'_y, \tilde{p}'_z, 1)$$

$$\mathrm{M} = \left(\begin{array}{ccc|c} \multicolumn{3}{c|}{\text{rotation/scaling}} & \text{translation} \\ m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ \hline 0 & 0 & 0 & 1 \end{array}\right)$$

Rappel projection



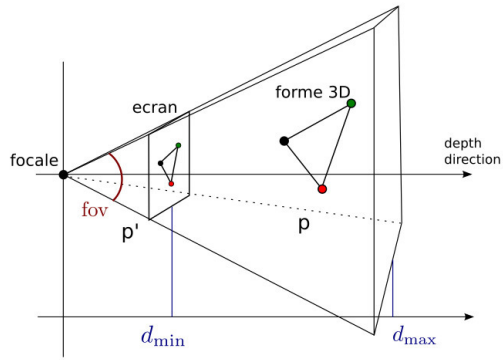
$$\tilde{p}' = \mathrm{M}\,\tilde{p}$$

$$\mathrm{M} = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix}$$

$$\tilde{p} = (\tilde{p}_x, \tilde{p}_y, \tilde{p}_z, 1)$$
$$\tilde{p}' = (\tilde{p}'_x, \tilde{p}'_y, \tilde{p}'_z, \tilde{p}'_w)$$

$\tilde{p}'$ point dans l'espace projectif

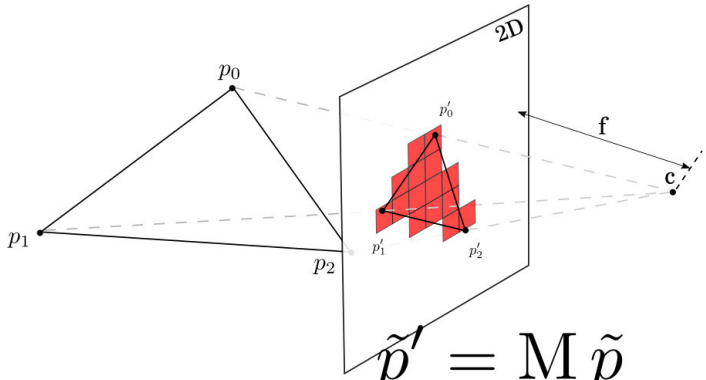
ex.



$$\mathrm{M} = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & C & D \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

$$f = \mathrm{cotan}(\mathrm{fov}/2)$$
$$f_x = f/a$$
$$C = \frac{d_{\max} + d_{\min}}{d_{\max} - d_{\min}}$$
$$D = -2\frac{d_{\max} d_{\min}}{d_{\max} - d_{\min}}$$

Rappel projection



$$\tilde{p}' = \mathrm{M}\,\tilde{p}$$

Pour revenir dans l'espace 3D

Normalization par le paramètre projectif

$$\tilde{p}'' = (\tilde{p}'_x/\tilde{p}'_w, \tilde{p}'_y/\tilde{p}'_w, \tilde{p}'_z/\tilde{p}'_w, 1)$$

$$\tilde{p}'' = (p''_x, p''_y, p''_z, 1)$$

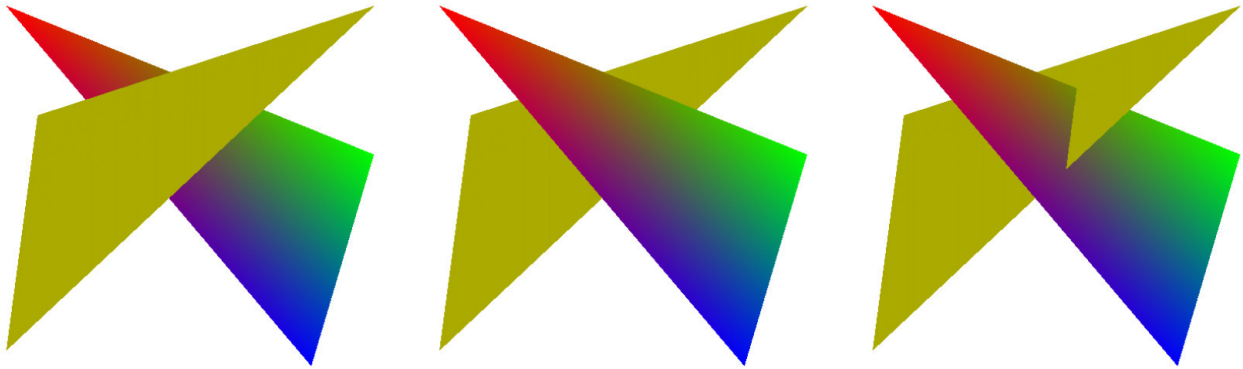
(p''_x, p''_y) sont les coordonnées dans l'espace écran $\in [-1, 1]$

p''_z est la *profondeur* du sommet $\in [-1, 1]$

Gestion de la profondeur

Comment savoir si un triangle doit être affiché devant un autre?

↳ tri par triangle ?



Tampon de profondeur

Tampon de profondeur (depth buffer ou z-buffer)
est une seconde *image* contenant les profondeurs affichées

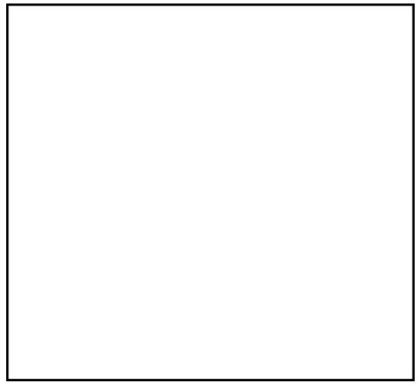
Algorithme:

Function Affiche(position,couleur,profondeur,image,zbuffer)

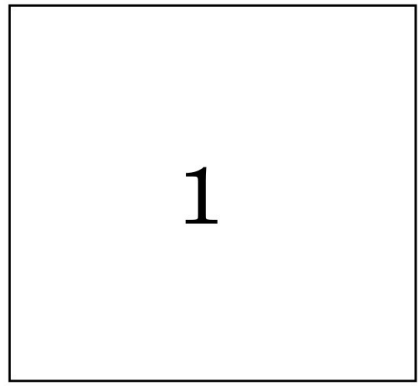
Si zbuffer(position) < profondeur:
    image(position) = couleur
Sinon
    ne rien afficher

Tampon de profondeur

ex.

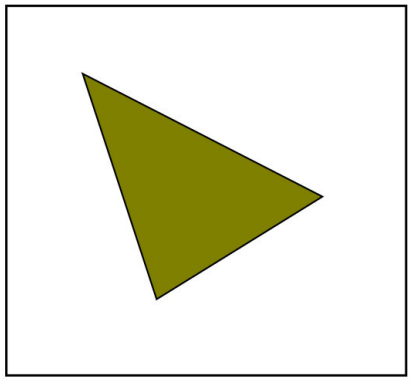


image

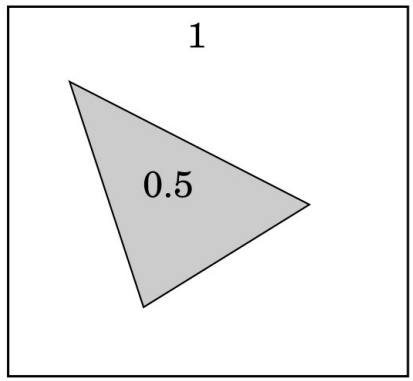


zbuffer

1er triangle de profondeur constante 0.5



image

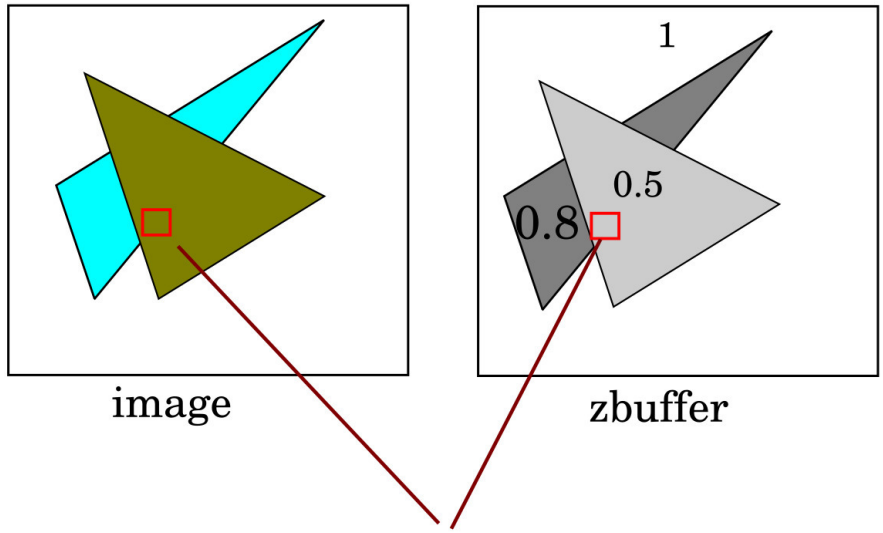


zbuffer

Tampon de profondeur

ex.

2ème triangle de profondeur constante 0.8



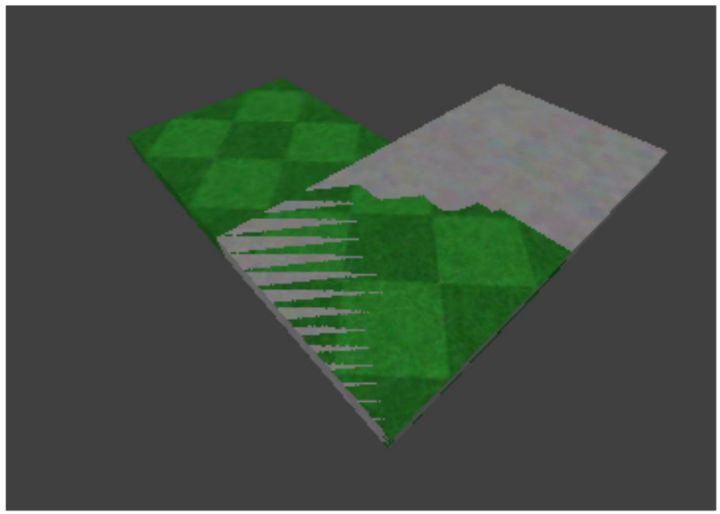
couleur des pixels non modifiée
car zbuffer(p)=0.5 < 0.8

Tampon de profondeur

Gère génériquement toute intersection entre triangle

Zbuffer possède une précision limitée

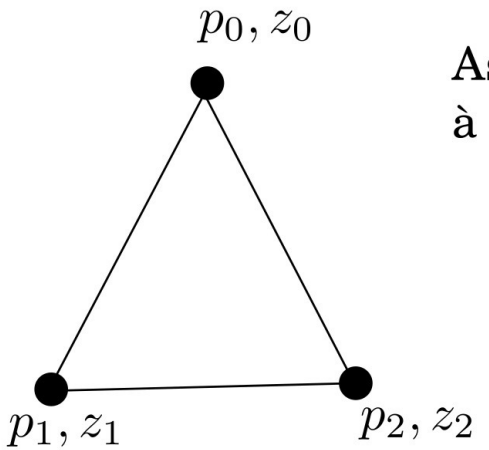
Ne peut pas différencier 2mm d'écart à 1km de distance
nombres flottants ~égaux



Z-fighting

Tampon de profondeur

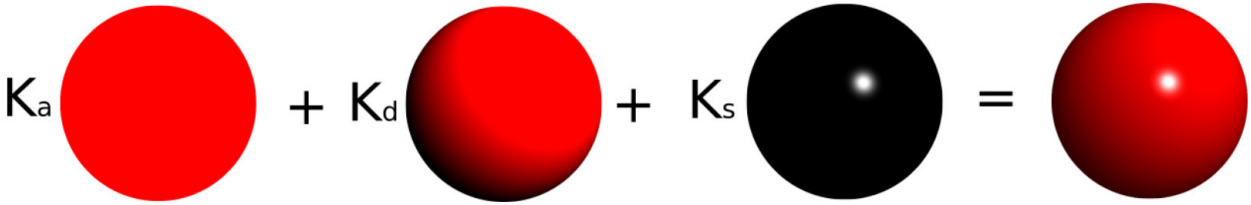
Rem. pour gérer le tampon de profondeur



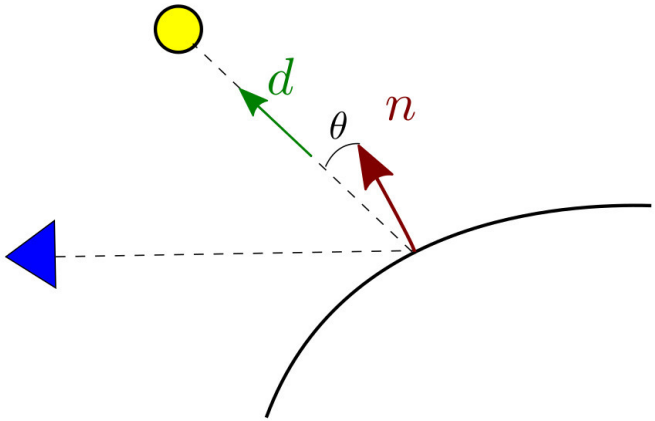
Associe une profondeur z
à tous les sommets 2D

On construit l'image de profondeur en interpolant les valeurs z
tout comme une couleur.

Rappel sur l'illumination



Diffuse coefficient $c_d = \cos(\theta)$



Specular coefficient $c_s = \cos(\psi)^n$