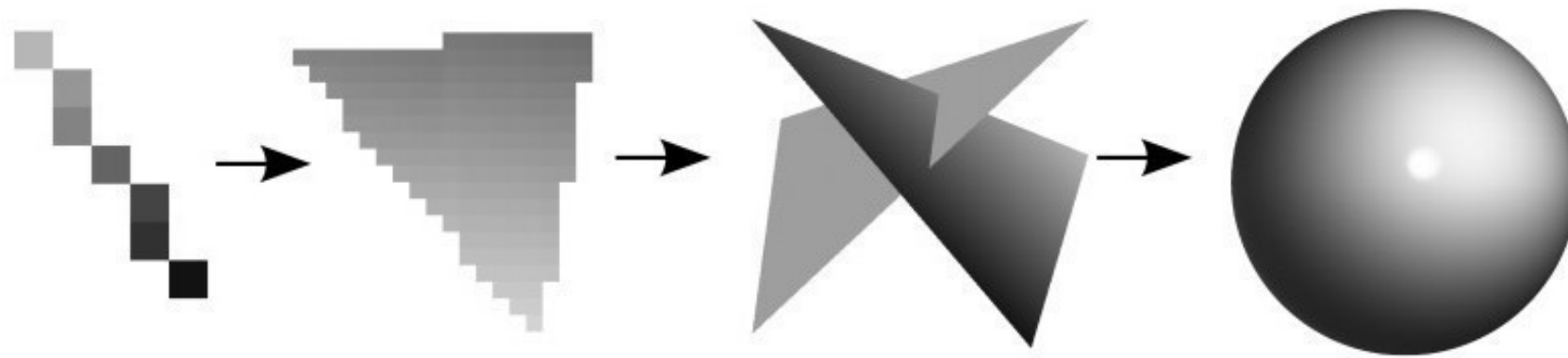


Rendu projectif



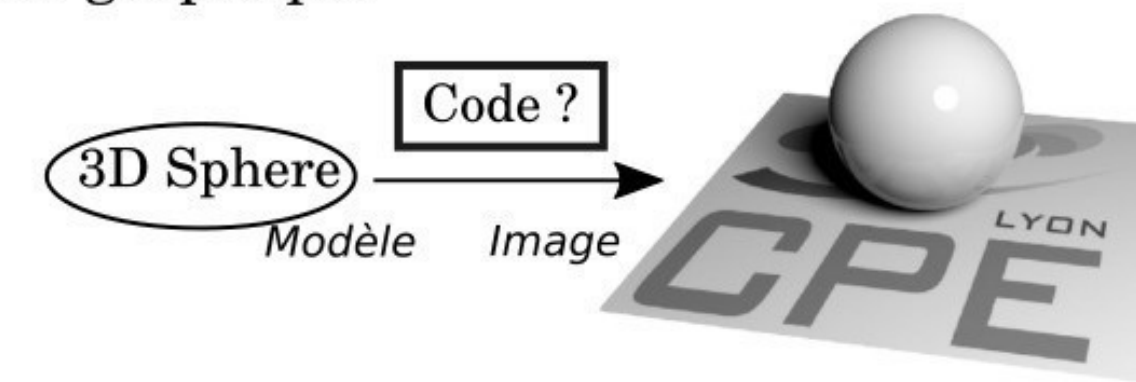
000

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



001

Ce que l'on sait faire

Structure de données d'images

= tableau de couleurs de taille Nx Ny

couleurs

```
struct couleur
{
    float r,g,b;  pratique
};
struct couleur
{
    int r,g,b;
};
struct couleur  compact
{
    unsigned char r,g,b;
};
struct couleur  niveau de gris
{
    float value;
};
struct couleur  binaire
{
    bool value;
};
```

tableau

```
struct image
{
    couleur data[Nx][Ny];
};
struct image  statique
{
    couleur data[Nx*Ny];
};
struct image  dynamique
{
    int Nx,Ny;
    std::vector<couleur> data;
};
```

002

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
    //...
}
```

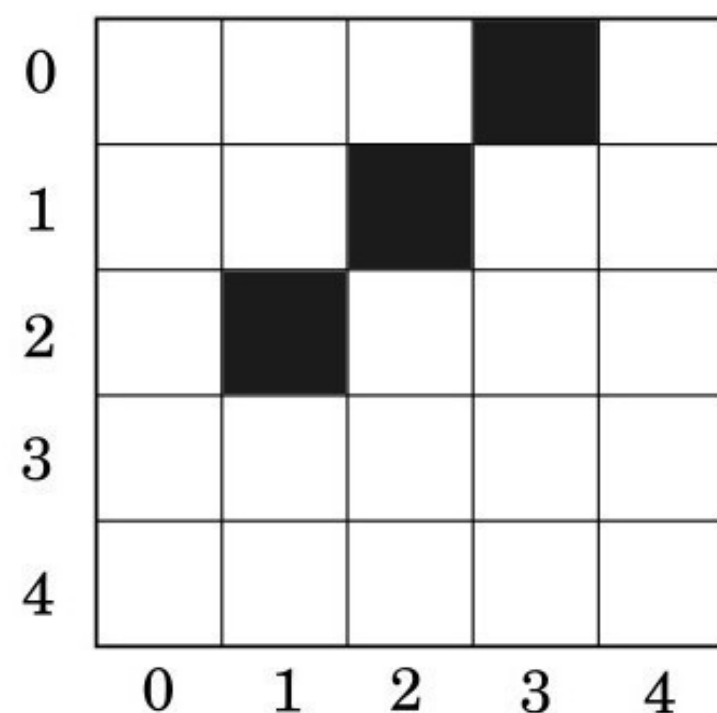
Export en fichier .ppm pour visualisation

003

Ce que l'on sait faire

Quelles formes sait-on dessiner correctement sur une image

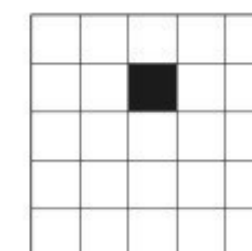
= savoir allumer précisément les bonnes coordonnées
en valeurs entières



004

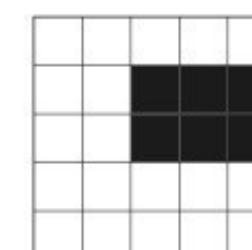
Ce que l'on sait faire

Un point



$im(kx, ky) = \dots$

Un rectangle
(/toute l'image)

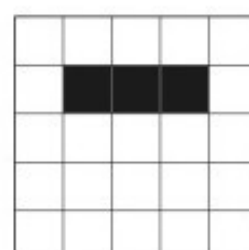


```
for(int kx=x0 ; kx<x0+Lx ; ++kx)
  for(int ky=y0 ; ky<y0+Ly ; ++ky)
    im(kx,ky)=...
```

005

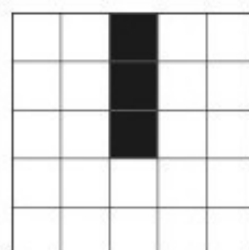
Ce que l'on sait faire

Ligne horizontale



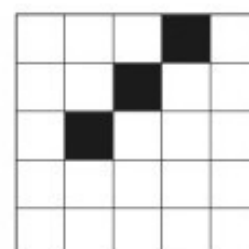
```
for(int kx=x0 ; kx<x1 ; ++kx)
  im(kx,y)=...
```

Ligne verticale



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

Ligne diagonale

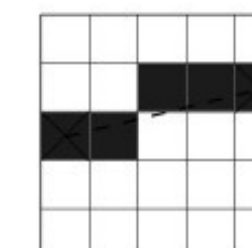


```
for(int k=k0 ; k<k1 ; ++k)
  im(k,k)=...
```

006

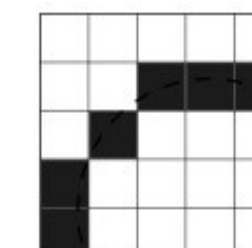
Moins facile

Segment quelconque



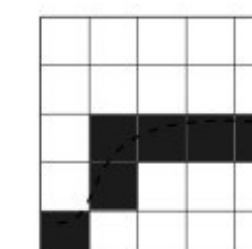
$y = ax + b$

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

007

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

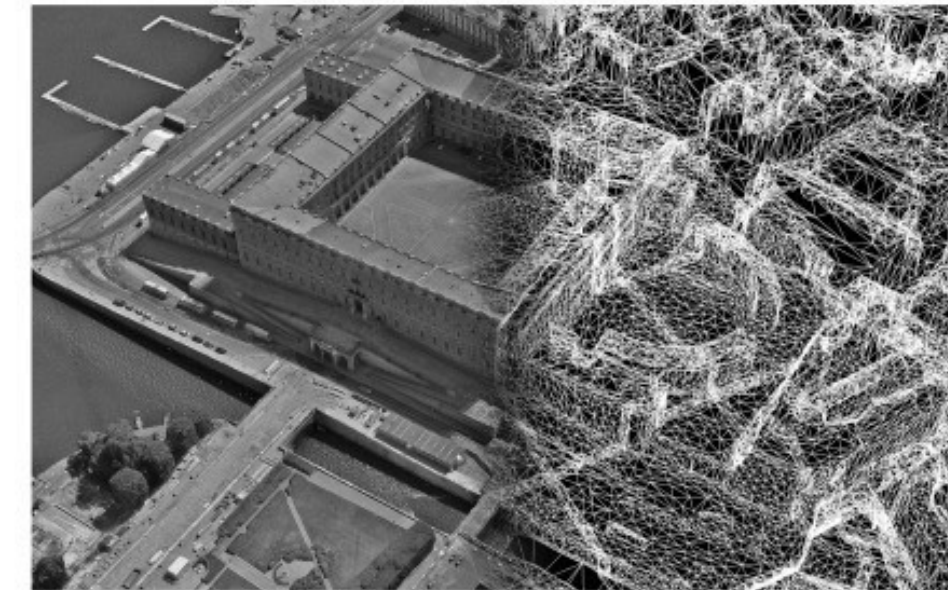
008

Scène 3D

Scène 3D = ensemble de maillages 3D

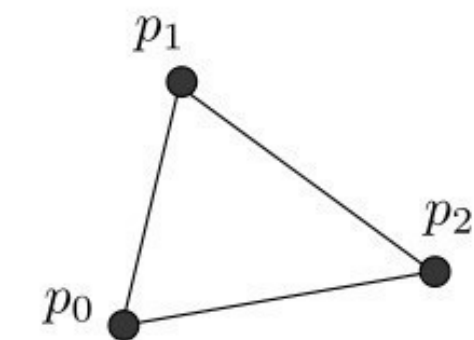
Maillage 3D = ensemble de triangles 3D

colorés, texturés, etc.



Il faut savoir afficher un triangle "3D"

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

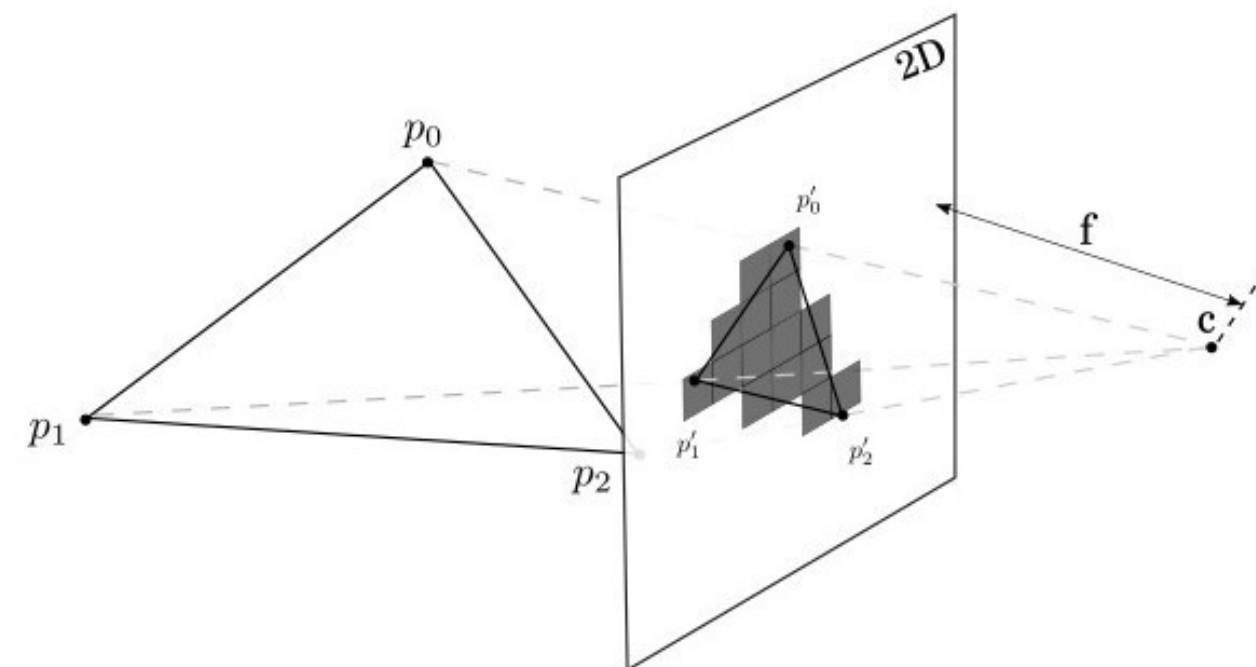


009

Triangle 3D

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

→ projeté sur un plan



$$p' = M p$$

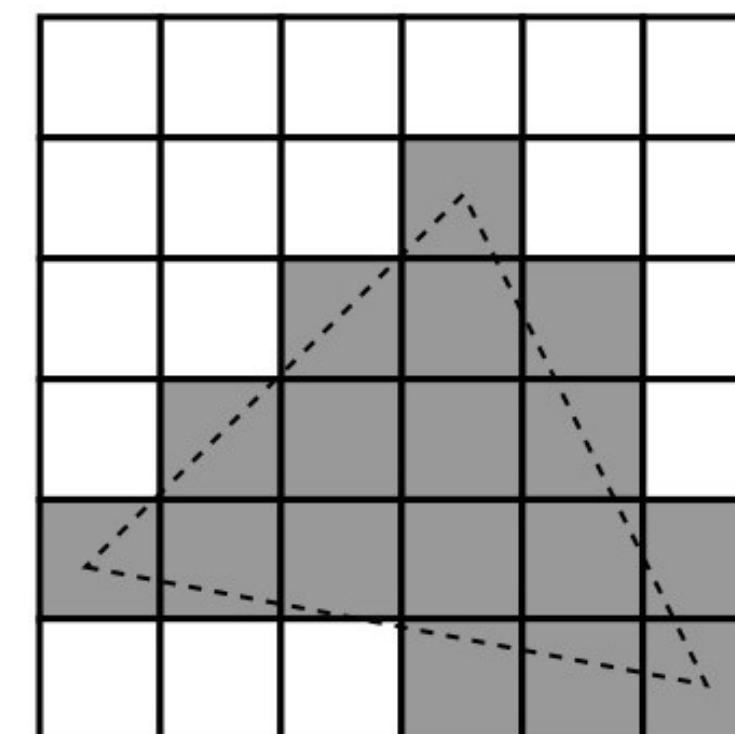
M: matrice de projection

010

Triangle 2D

Il faut savoir afficher un triangle en 2D

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

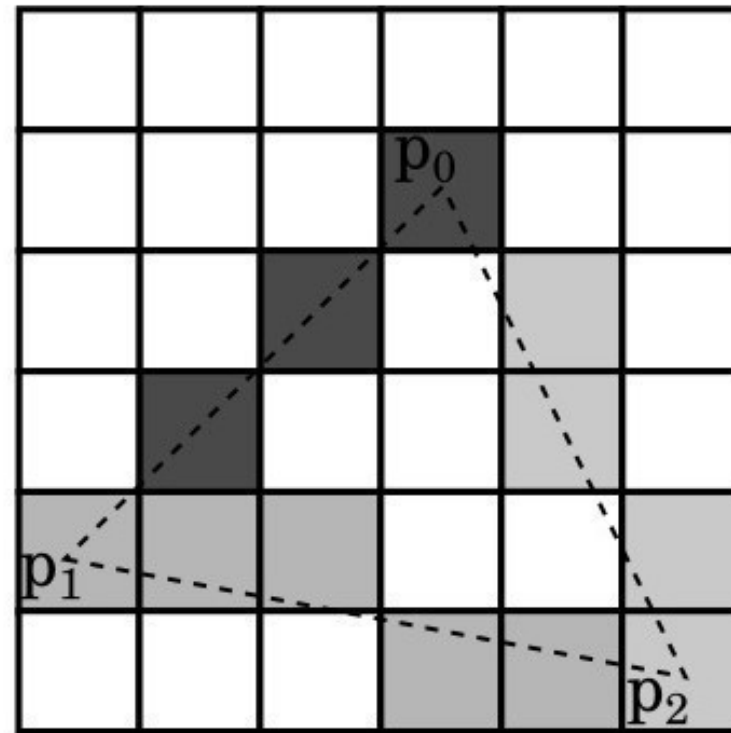


011

Triangle 2D

Afficher un triangle 2D

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

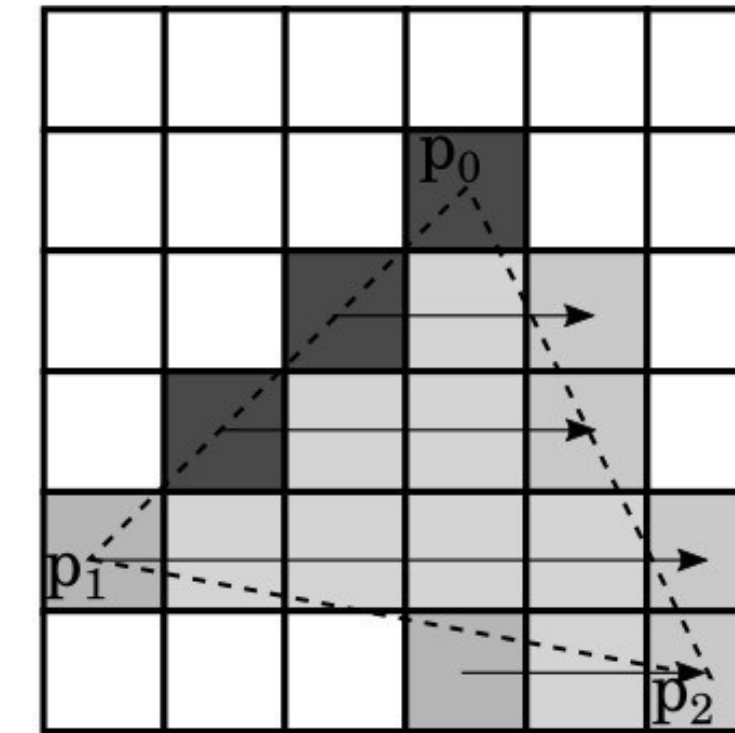


012

Triangle 2D

Afficher un triangle 2D

Etape 2: Combler avec segments horizontaux



013

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

014

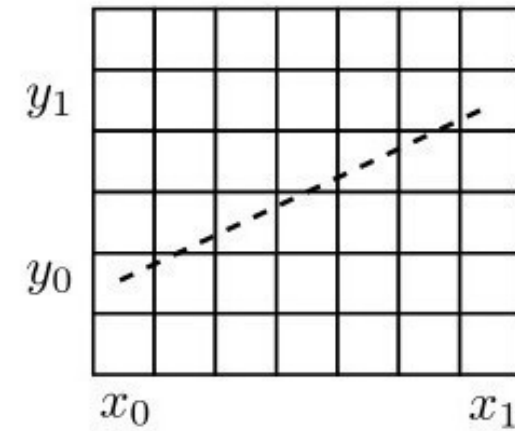
Algorithme de Bresenham

Tracé de segment discret

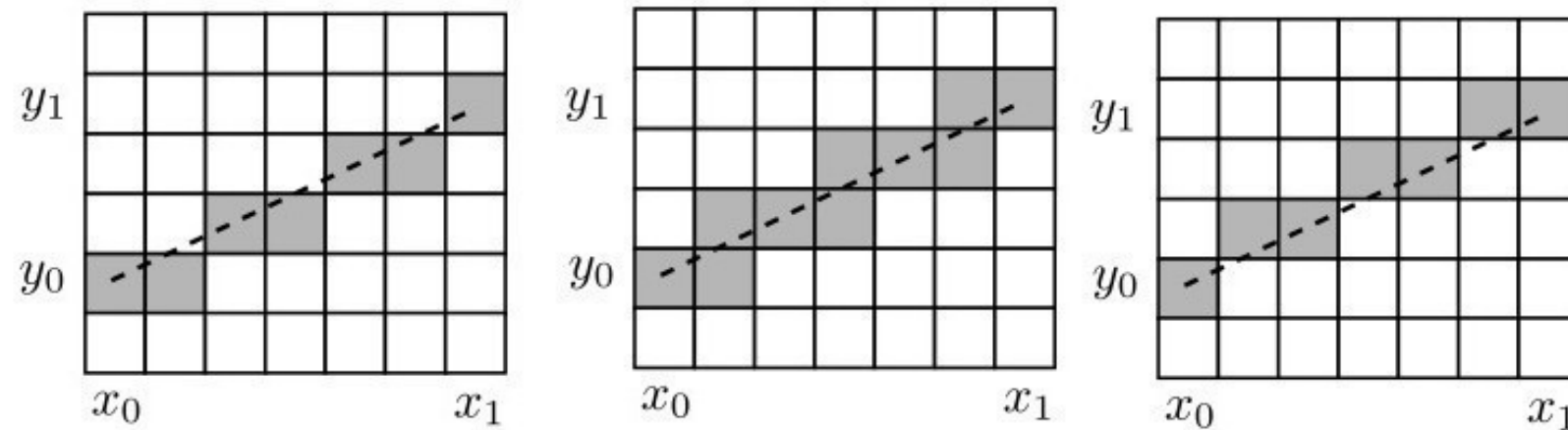
015

Segment discret

Plusieurs possibilités

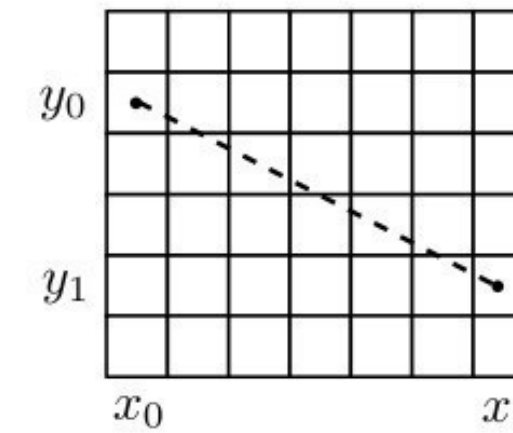


Quel est le "vrai" segment discret?



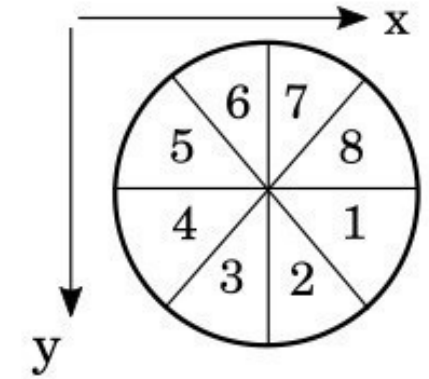
016

Segment discret



On suppose un segment du 1er quadrant

$$\begin{aligned} x_1 &> x_0 \\ y_1 &> y_0 \\ x_1 - x_0 &> y_1 - y_0 \end{aligned}$$



$$\begin{aligned} y &= a(x - x_0) + y_0 \\ a &= \Delta y / \Delta x \quad \Delta y = y_1 - y_0 \\ &\quad \Delta x = x_1 - x_0 \end{aligned}$$

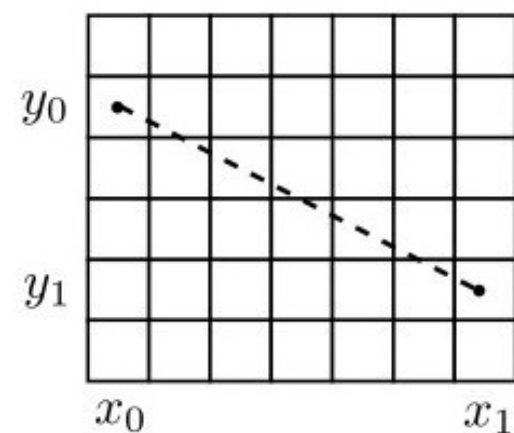
```
int dx = x1-x0;
int dy = y1-y0;

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) = ...
}
```

017

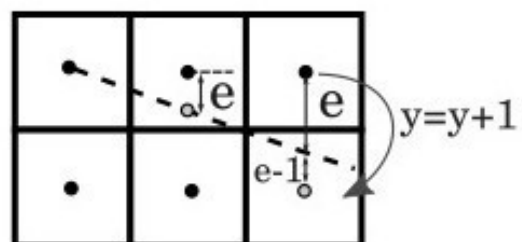
Segment discret



```
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;
    }
}
```

018

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;
    }
}
```

019

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;

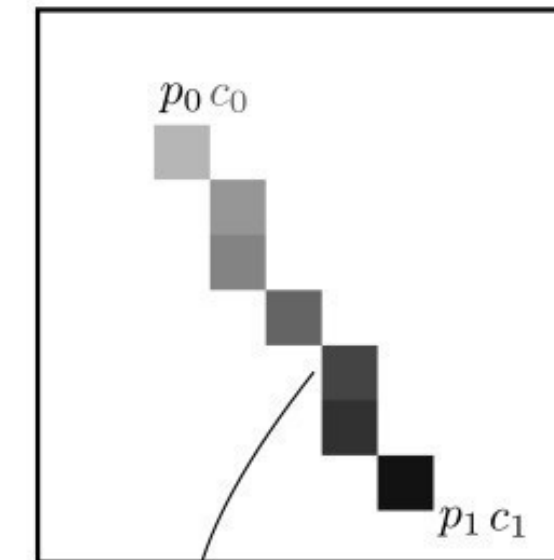
    if( erreur >= 0 )
    {
        ++y;
        erreur -= coeff;
    }
}
```

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

020

Segments en couleurs

On interpole entre les deux extrémités



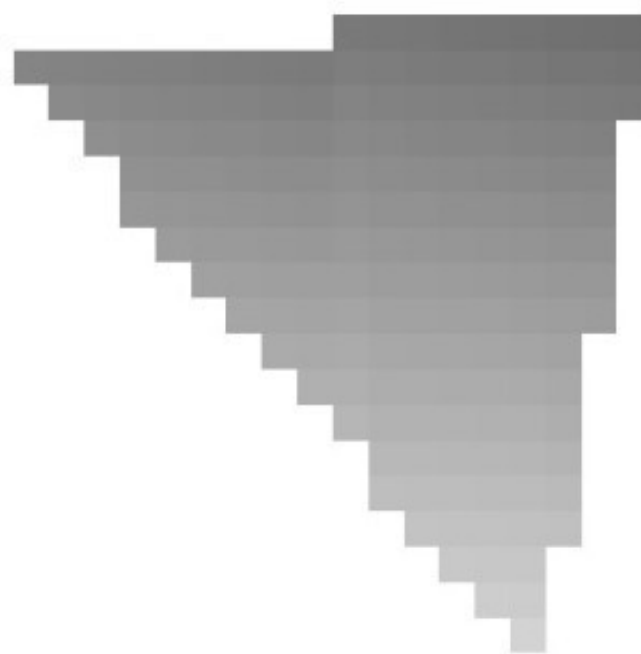
$$\begin{aligned} p_0 &= (x_0, y_0, z_0) \\ p_1 &= (x_1, y_1, z_1) \\ c_0 &= (r_0, g_0, b_0) \\ c_1 &= (r_1, g_1, b_1) \end{aligned}$$

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

Pourrait être optimisé pour limiter le calcul flottant

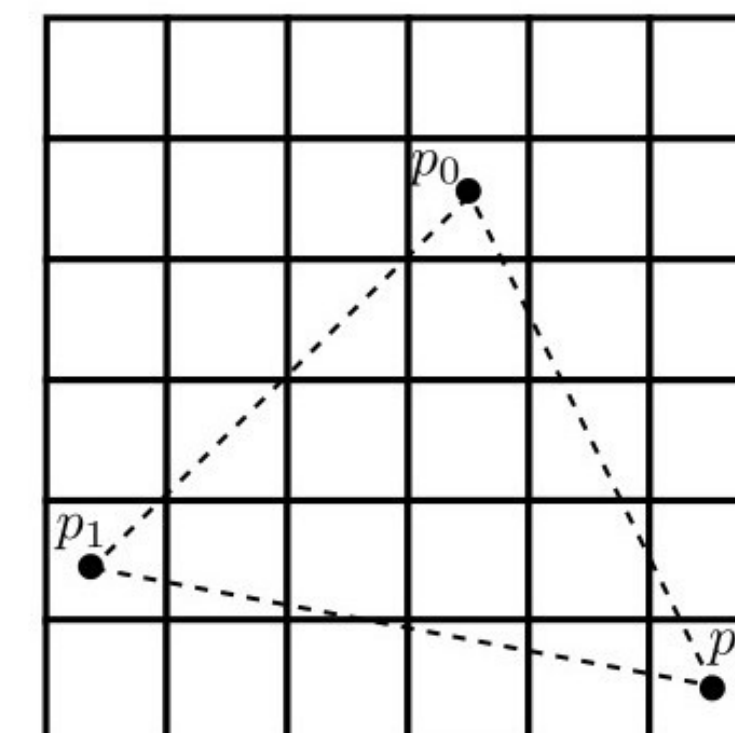
021

Afficher un triangle plein



022

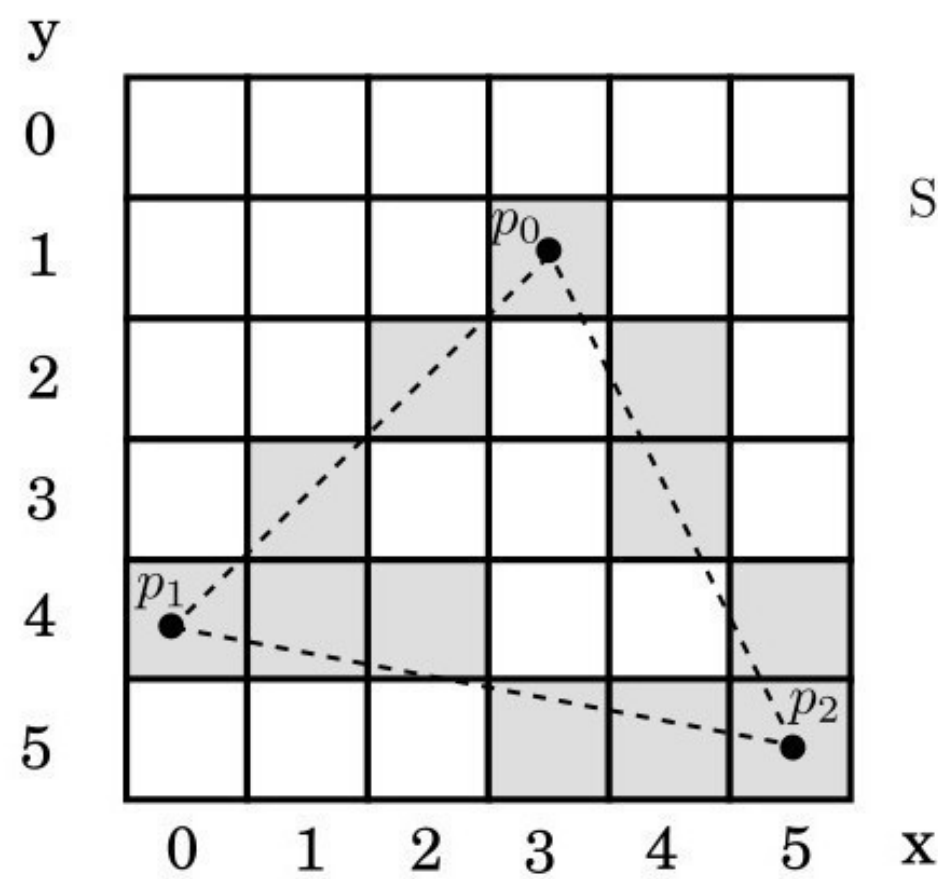
Algorithme scanline



023

Algorithme scanline

1. Parcoure les 3 segments



$[p_0, p_1]$

$[p_1, p_2]$

$[p_2, p_0]$

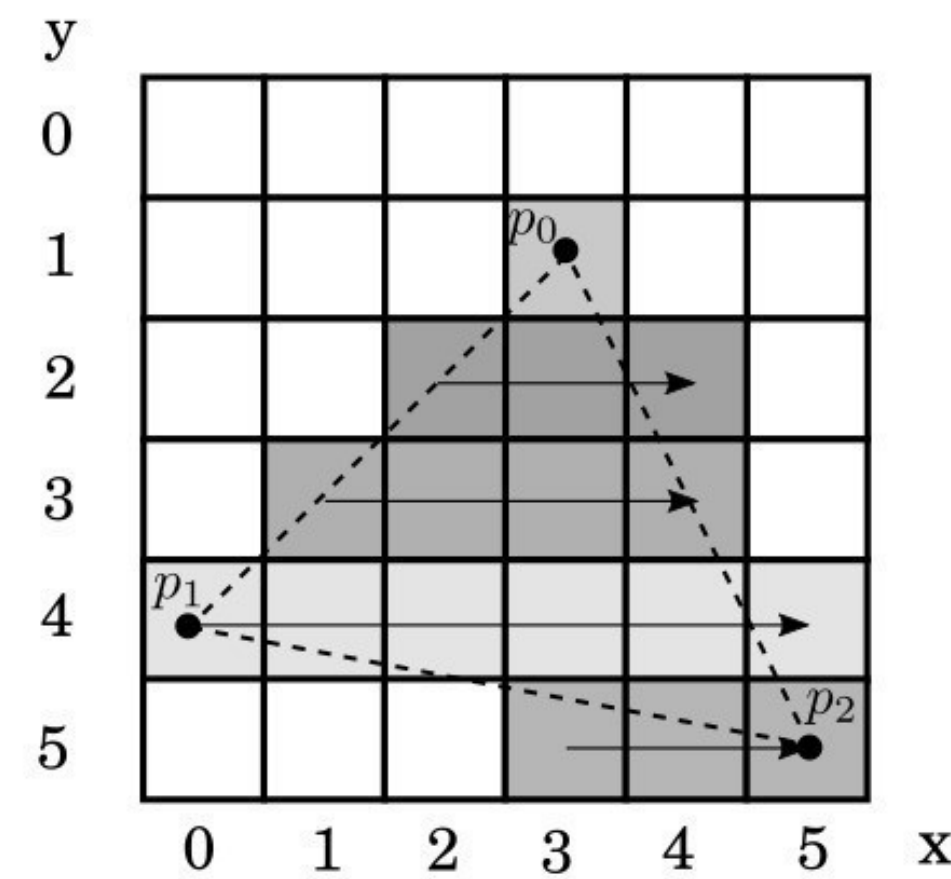
Stocke pour y donné, (x_{min}, x_{max})

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

024

Algorithme scanline

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

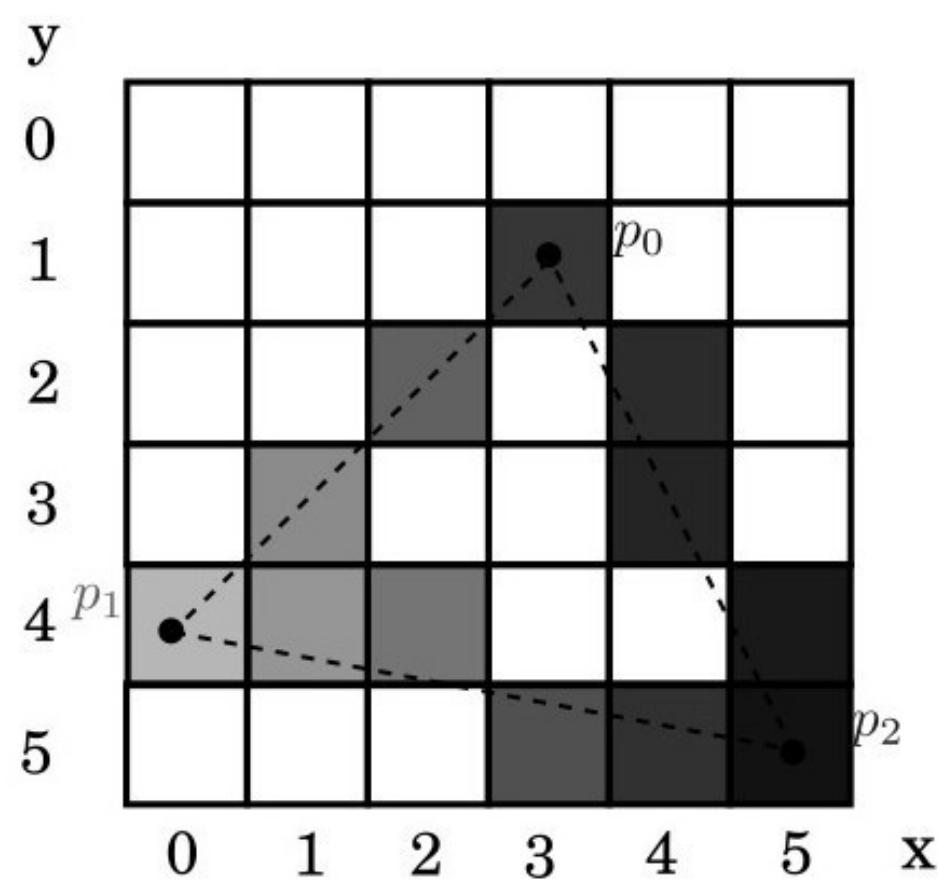


y	x_{min}	x_{max}
1 :	3	3
2 :	2	4
3 :	1	4
4 :	0	5
5 :	3	5

025

Triangle en couleurs

1. Enregistre en + les valeurs de couleur interpolés

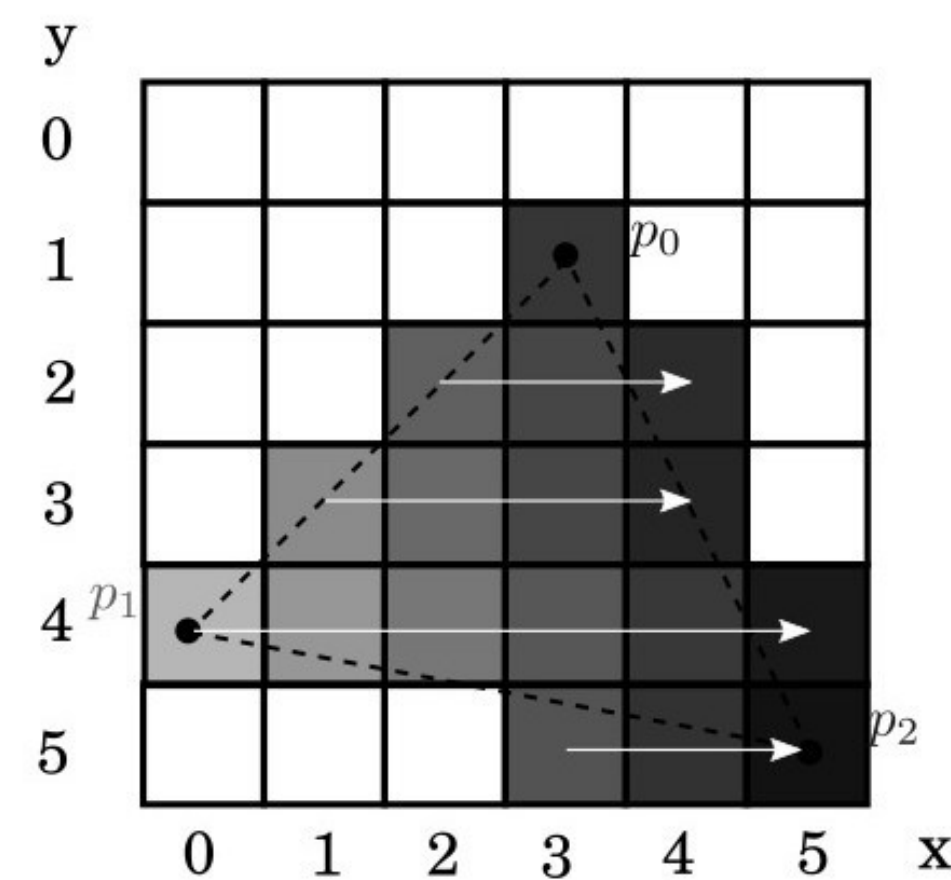


y	x_{min}	x_{max}	v_{min}	v_{max}
1 :	3	3	C_0	C_0
2 :	2	4	V_{22}	V_{24}
3 :	1	4	V_b	V_{34}
4 :	0	5	C_1	V_{45}
5 :	3	5	V_{53}	C_2

026

Triangle en couleurs

2. Segments horizontaux avec couleurs

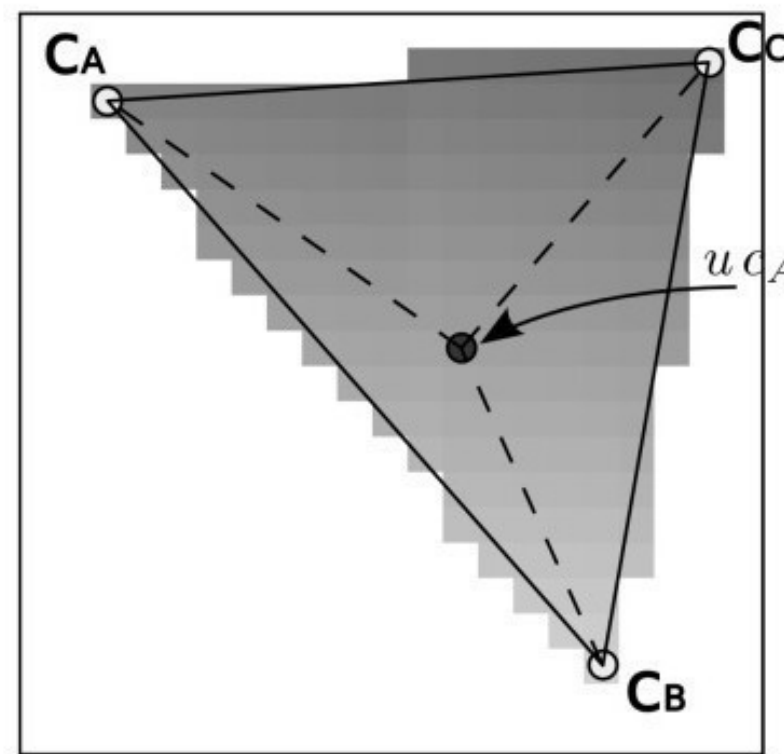


y	x_{min}	x_{max}	v_{min}	v_{max}
1 :	3	3	C_0	C_0
2 :	2	4	V_{22}	V_{24}
3 :	1	4	V_b	V_{34}
4 :	0	5	C_1	V_{45}
5 :	3	5	V_{53}	C_2

027

Triangle en couleur

Autre solution possible: interpolation barycentrique



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

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

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

028

Triangles en 3D

029

Rappel transformation affines

$$p' = R p + t$$

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

$$p = (p_x, p_y, p_z)$$

$$p' = (p'_x, p'_y, p'_z)$$

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

4 × 4

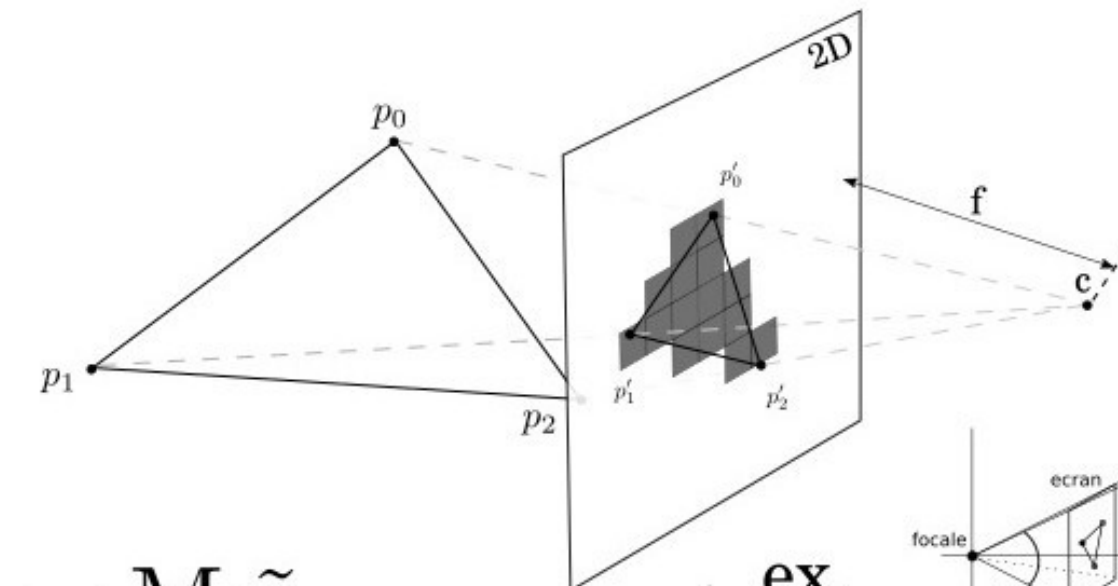
$$\tilde{p} = (\tilde{p}_x, \tilde{p}_y, \tilde{p}_z, 1)$$

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

$$M = \begin{pmatrix} \text{rotation/scaling} & \text{translation} \\ m_{00} & m_{01} & m_{02} & t_x \\ m_{10} & m_{11} & m_{12} & t_y \\ m_{20} & m_{21} & m_{22} & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

030

Rappel projection



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

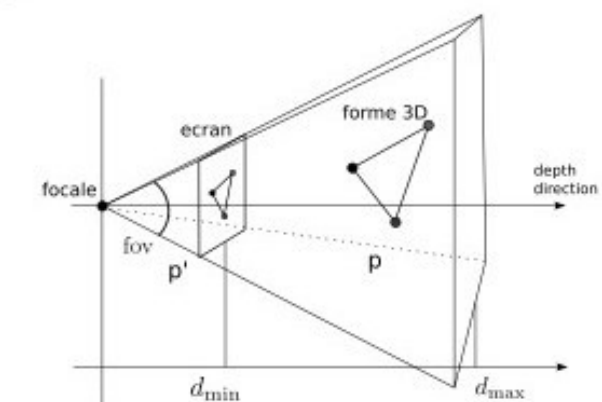
$$M = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{pmatrix}$$

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

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

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

ex.

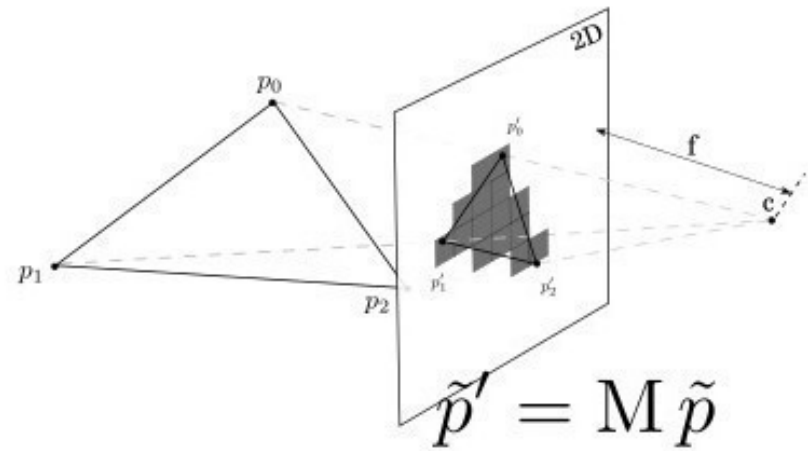


$$M = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & C & D \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

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

031

Rappel projection



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
 $\in [-1, 1]$

032

Gestion de la profondeur

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

↳ tri par triangle ?



033

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)

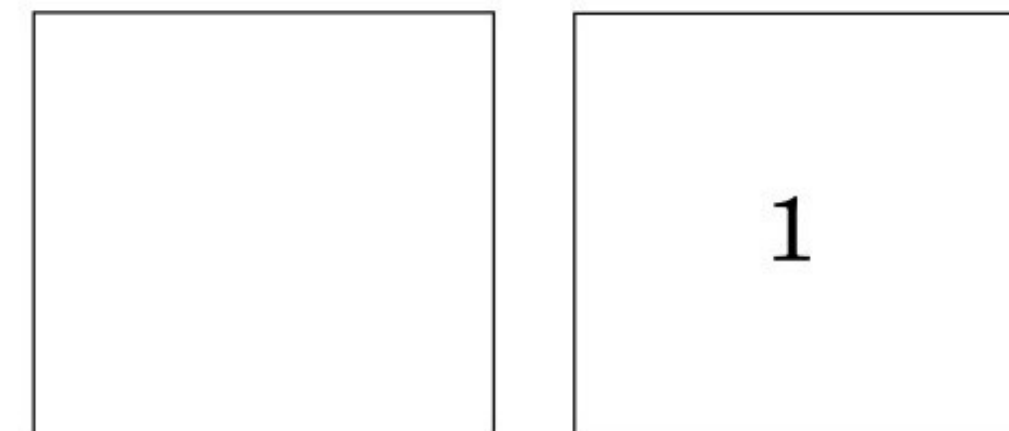
Si zbuffer(position) < profondeur:
 image(position) = couleur

Sinon
 ne rien afficher

034

Tampon de profondeur

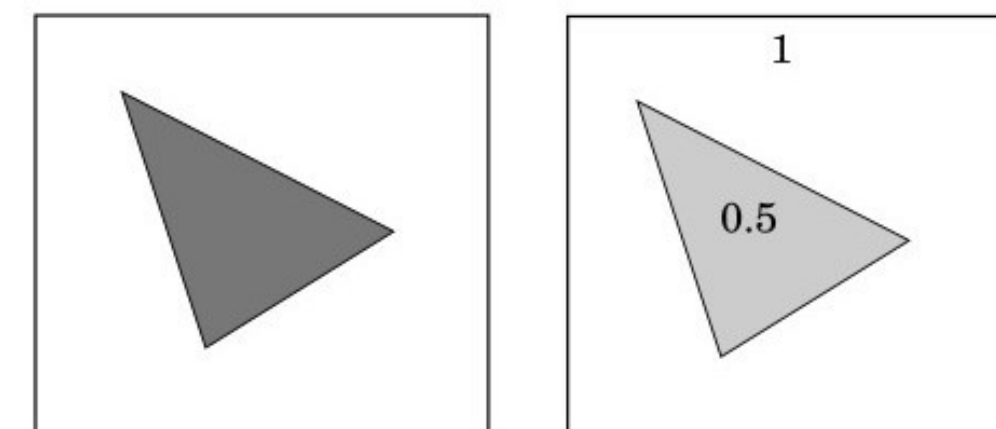
ex.



image

zbuffer

1er triangle de profondeur constante 0.5



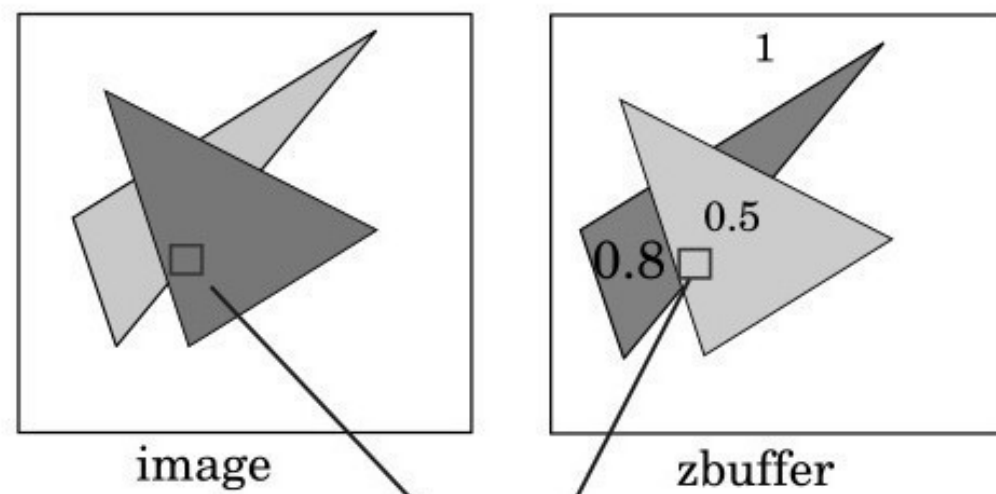
image

zbuffer

035

Tampon de profondeur

ex.
2ème triangle de profondeur constante 0.8



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

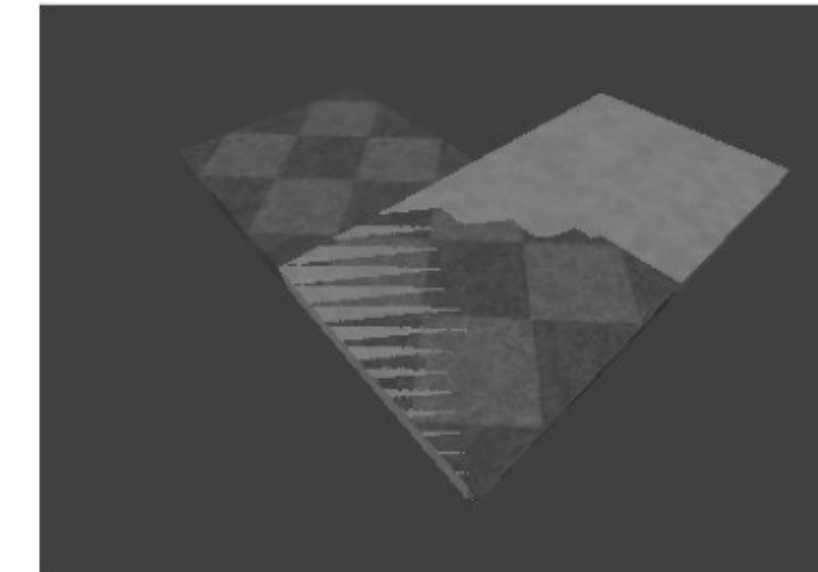
036

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*

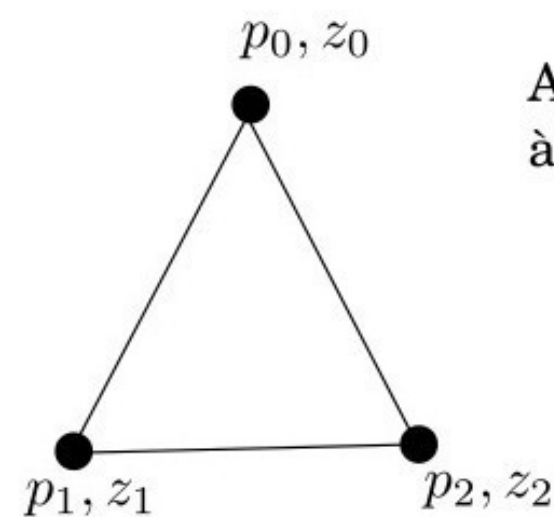


Z-fighting

037

Tampon de profondeur

Rem. pour gérer le tampon de profondeur



Associe une profondeur z
à tous les sommets 2D

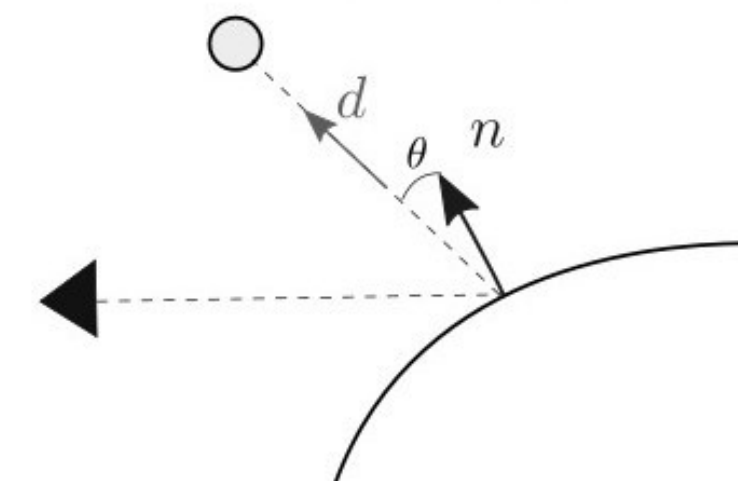
On construit l'image de profondeur en interpolant les valeurs z
tout comme une couleur.

038

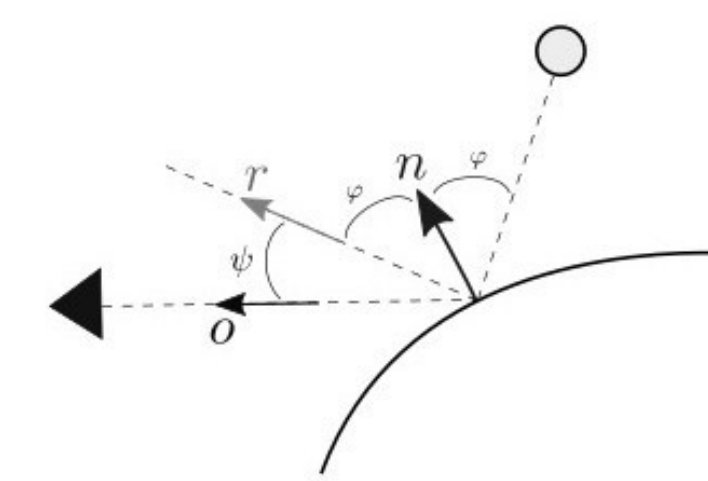
Rappel sur l'illumination

$$K_a \text{ (sphere)} + K_d \text{ (sphere)} + K_s \text{ (sphere)} = \text{ (sphere)}$$

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



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



039