Algorithmes de tracé de segment

Jean-Luc Mari *

—

IREM d'Aix-Marseille

—

Mai 2010

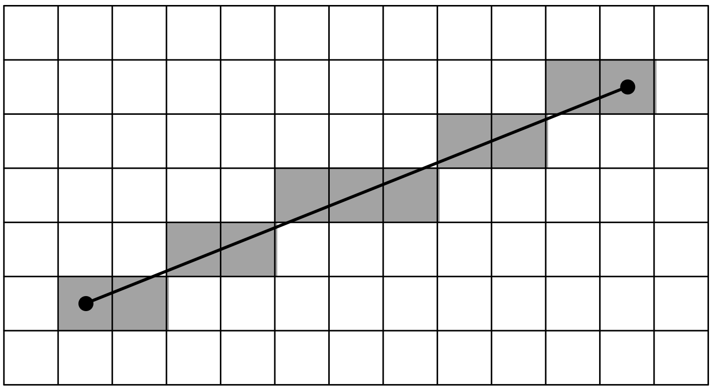


Figure 1: Illustration du résultat de l'algorithme de tracé de segment de Bresenham.

1 Introduction

1.1 Contexte et principe

Les algorithmes de tracé de segment présentés dans ce document sont au nombre de trois. Le premier algorithme est dit *naïf* car

*Département d'Informatique, Faculté des Sciences de Luminy, Université de la Méditerranée — Contact : `mari@univmed.fr` — Web : `http://www.dil.univ-mrs.fr/~mari/` — Site de l'IREM d'Aix-Marseille : `http://www.irem.univ-mrs.fr`

il permet de tracer un segment sans chercher à tirer avantage du caractère intrinsèquement discret d'une image numérique (section 4). Le deuxième algorithme, dit *incrémental*, est fondé sur une relation de récurrence qui permet d'améliorer le temps de calcul (section 5). Cependant, le troisième algorithme (dit du *point milieu*) est encore plus efficace car il ne manipule que des entiers en effectuant un minimum d'opérations couteuses (section 6).

On doit cette dernière approche à Jack E. Bresenham qui, en mai 1962, travaillait dans un laboratoire d'informatique d'IBM. Il a mis au point son algorithme de tracé de segment alors qu'il cherchait à piloter un traceur attaché à une console texte. Cet algorithme a été présenté à la convention de l'ACM en 1963, puis publié en 1965 dans la revue *IBM Systems Journal*.

1.2 Utilisations

L'algorithme détermine quels sont les points d'un plan discret qui doivent être tracés afin de former une approximation de segment de droite entre deux points donnés (voir figure 1). Cet algorithme est souvent utilisé pour dessiner des segments de droites sur l'écran d'un ordinateur ou une image calculée pour l'impression. Il est considéré comme l'un des premiers algorithmes découverts dans le domaine de la synthèse d'image.

Le principe du calcul est largement documenté et a depuis été repris pour tracer de façon incrémentale n'importe quelle courbe conique (cercle, ellipse, arc, parabole, hyperbole) ou courbes de Bézier grâce aux propriétés de leur fonction polynomiale de définition, dont les dérivées permettent de calculer les orientations de segments élémentaires avec de simples opérations entières. On peut même l'adapter à l'approximation de courbes dont on ne connaît qu'un développement limité (dont on ne prendra que les premiers termes de faible degré), utilisable avec une bonne précision sur un domaine suffisant par rapport à la résolution (sinusoïdes, exponentielles, puissances non entières).

L'algorithme est également facilement adaptable au calcul de courbes et surfaces dans un espace discret de plus de deux dimensions (notamment pour le pilotage de machines outils). Même en deux dimensions seulement, on peut discrétiser avec cet algorithme une courbe avec une fonction de lissage prenant en compte l'erreur commise entre deux points candidats afin d'ajuster leur couleur, l'erreur incrémentale étant convertible en coefficient de transparence, ce qui permet de conserver la graisse (épaisseur visuelle) d'une courbe tout en limitant l'effet d'escalier (crénelage).

Cet algorithme intervient aussi dans le lissage de rendus de textures 2D appliquées sur des surfaces d'une scène 3D où la texture subit des réductions ou agrandissements. On l'emploie aussi pour le lissage d'agrandissements photographiques, ou pour l'interpolation de couleurs intermédiaires sur une échelle discrète.

2 Rappels sur les droites et segments dans le plan

2.1 Repère, coordonnées cartésiennes dans le plan

On considère le plan muni d'un repère orthonormé $(O, \overrightarrow{i}, \overrightarrow{j})$. Ceci signifie notamment que les vecteurs $\overrightarrow{i}$ et $\overrightarrow{j}$ sont de norme 1 et sont orthogonaux entre eux. Chaque point M du plan est ainsi muni de deux coordonnées (x_M, y_M) qui déterminent sa position : $\overrightarrow{OM} = x_M \overrightarrow{i} + y_M \overrightarrow{j}$. Les nombres x_M et y_M sont respectivement l'abscisse et l'ordonnée du point M. En écrivant $M = (x_M, y_M)$, on identifie le point M à ses coordonnées : le plan s'identifie à l'ensemble $\mathbb{R}^2$ des couples (x, y) de nombres réels.

2.2 Pente d'un segment

On considère deux points distincts $A = (x_A, y_A)$ et $B = (x_B, y_B)$ du plan. Par ces deux points passe une seule droite, que l'on nommera D. Notons aussi $dx = x_B - x_A$ et $dy = y_B - y_A$ les différences entre les coordonnées de A et de B. On suppose que dx est non nul, ce qui revient à dire que la droite D n'est pas verticale (voir figure 2). On note alors $m = \frac{dy}{dx}$ le nombre que l'on appelle la *pente* de la droite D.

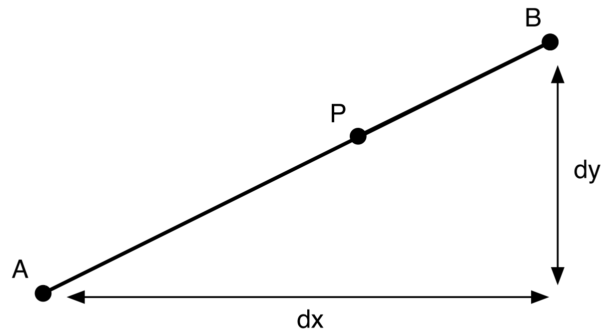


FIGURE 2: Le segment $[A, B]$.

2.3 Equation cartésienne $F(x, y) = 0$

On considère l'équation (E) suivante :

$$(E): \quad F(x, y) = 0, \text{ avec } F(x, y) = dy(x - x_A) - dx(y - y_A)$$

L'équation (E) est l'équation *cartésienne* d'une droite : l'ensemble de tous les points du plan vérifiant (E) est une droite.

On a $F(x_A, y_A) = dy.0 - dx.0 = 0$ et $F(x_B, y_B) = dy.dx - dx.dy = 0$, c'est-à-dire que les points A et B vérifient l'équation (E), cette équation est donc précisément l'équation de la droite D passant par A et B.

2.4 Position d'un point par rapport à un segment

Soit $M = (x_M, y_M)$ un point du plan qui ne se trouve pas sur la droite D. Pour répondre à la question "*le point* M *se situe-t-il à droite ou à gauche du segment* $[A, B]$ *orienté de* A *vers* B *?*", on peut considérer l'équation (E) de la droite D donnée ci-dessus et évaluer $F(x_M, y_M)$. Si $F(x_M, y_M) < 0$, le point M se trouve à gauche du segment $[A, B]$, et si $F(x_M, y_M) > 0$, le point M se situe à droite de $[A, B]$. On peut noter que ce critère est également vrai si le segment $[A, B]$ est vertical, c'est-à-dire si $dx = 0$. Dans le cas où $dx > 0$, ce critère permet de savoir si le point M est "au dessus" de la droite D (on a dans ce cas $F(x_M, y_M) < 0$) ou "au dessous" de D (et dans ce cas, on a $F(x_M, y_M) > 0$).

2.5 Augmentation de y de long d'un segment

Soit un point $P = (x_P, y_P)$ de la droite D ayant pour abscisse :

$$x_P = x_A + \Delta x$$

Etant sur la droite D, le point P doit vérifier l'équation (E) de la droite D, c'est-à-dire que $F(x_P, y_P) = 0$. On a donc :

$$\begin{aligned} & F(x_P, y_P) = 0 \\ \Rightarrow \quad & dy(x_P - x_A) - dx(y_P - y_A) = 0 \\ \Rightarrow \quad & dy(x_A + \Delta x - x_A) - dx(y_P - y_A) = 0 \end{aligned}$$

On en déduit que $dy.\Delta x = dx(y_P - y_A)$, c'est-à-dire que $y_P = y_A + \frac{dy}{dx}.\Delta x$, ou encore :

$$y_P = y_A + m.\Delta x$$

L'augmentation de y le long de la droite D est proportionnel à l'augmentation de x, le coefficient de proportionnalité étant égal à la pente m de D.

3 Position du problème

Les objets géométriques à dessiner dans une image numérique, dans notre cas des segments de droites, sont des approximations discrètes d'objets idéaux continus. On cherche des méthodes efficaces pour passer de l'objet idéal à sa discrétisation, laquelle sera représentée pixel par pixel sur l'image.

Il faut savoir que l'opération qui consiste à tracer des segments de droites est si fréquente que sa version optimisée (que nous étudions dans la section 6) est implémentée dans tous les processeurs graphiques actuels, dans le but de ne pas avoir un tracé "logiciel" (trop lent) mais "matériel" (de manière à être encore plus rapide).

On suppose que le langage de programmation utilisé (éventuellement au sein d'une bibliothèque graphique) dispose d'une primitive :

DessinePixel (x, y, couleur : **entier**)

qui permet d'afficher, de dessiner le pixel de couleur `couleur` aux coordonnées `(x,y)` dans une fenêtre graphique.

Cette primitive correspondrait à la ligne suivante sous AlgoBox [1] :

```
TRACER_POINT (x,y)
```

et aurait le prototype suivant en langage C :

```
void DessinePixel (int x, int y, unsigned char couleur)
```

On se limitera dans ce document aux segments de droites d'épaisseur 1. De manière à obtenir un ensemble de pixels adjacents, pour une pente comprise entre -1 et 1, on veut éditer un pixel sur chaque verticale. Pour des pentes de valeur absolue supérieure à 1, on veut un pixel sur chaque horizontale. Il nous faut donc une méthode rapide pour parcourir cet ensemble de pixels. Une discrétisation de droite est représentée sur la figure 3. Les pixels y sont représentés par des gros points sur une grille.

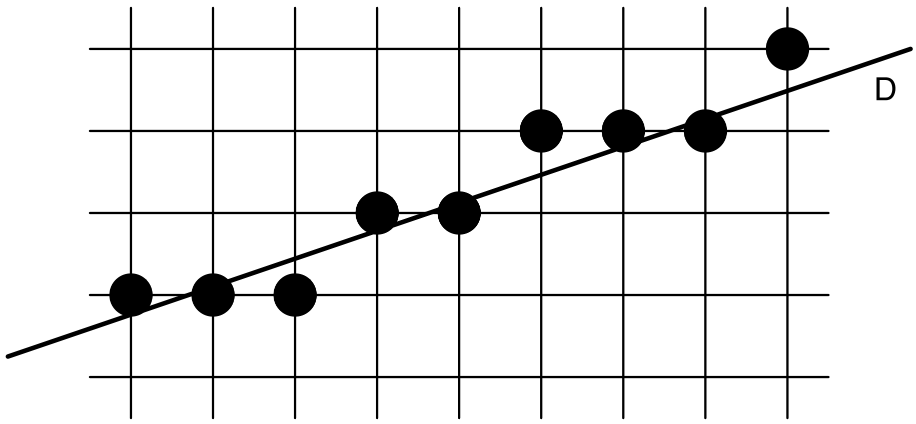


Figure 3: Exemple de segment discrétisé.

1. AlgoBox est un logiciel libre, multi-plateforme et gratuit d'aide à l'élaboration et à l'exécution d'algorithmes dans l'esprit des nouveaux programmes de mathématiques du lycée.
Il peut être téléchargé ici : `http://www.xm1math.net/algobox/`

Pour notre étude, on se place dans le premier octant (voir figure 4), c'est-à-dire dire dans le cas d'une pente comprise entre 0 et 1. On pourra se ramener aux autres cas par rotation et symétrie (cf. section 7). Dans le cas dans lequel on se place, on doit donc afficher exactement un pixel sur chaque verticale.

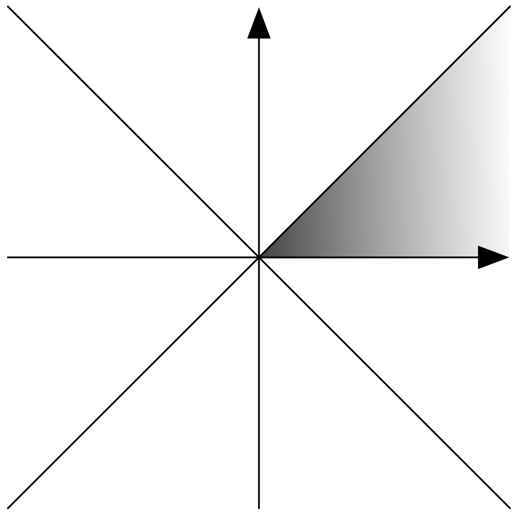


Figure 4: En gris, le premier octant.

On suppose que l'on dispose d'une fonction d'arrondi. En langage C, elle pourrait s'écrire de la manière suivante qui associe à x la partie entière de $x + 0,5$:

```
int Round (float x)
{
    return (int) (x + 0.5);
}
```

On cherche à dessiner un segment de droite d'un point $A = (x_0, y_0)$ à un point $B = (x_0 + dx, y_0 + dy)$, avec x_0, y_0, dx et dy entiers.

4 Algorithme naïf

4.1 Principe

On note $m = \frac{dy}{dx}$ la pente réelle de la droite D passant par les points A et B. L'algorithme naïf consiste à calculer, pour chaque valeur entière de l'abscisse x (comprise entre x_0 et $x_0 + dx$), la valeur de y correspondante par une opération d'arrondi.

```
Fonction TraceSegmentNaif(x0, y0, dx, dy : entier,
couleur : caractère) :
    i, x, y : entier
    m : flottant
    m ← dy / dx
    Pour i de 0 à dx faire
        x ← x0 + i
        y ← Arrondi(y0 + m * i)
        DessinePixel (x, y, couleur)
    Fin Pour
Fin
```

Cette méthode comporte de gros défauts : pour chaque pixel à dessiner, on effectue une multiplication en virgule flottante, une opération d'arrondi et une addition. Les deux opérations les plus couteuses sont la multiplication et l'arrondi. Dans la variante décrite dans la section suivante, nous allons nous attacher à supprimer la multiplication en virgule flottante.

4.2 Ecriture avec AlgoBox

La transcription de l'algorithme naïf entre les points $(x_0 = 1, y_0 = 1)$ et $(x_1 = 11, y_1 = 5)$ avec AlgoBox est la suivante :

```
VARIABLES
  x0 EST_DU_TYPE NOMBRE
  y0 EST_DU_TYPE NOMBRE
  dx EST_DU_TYPE NOMBRE
```

```
    dy EST_DU_TYPE NOMBRE
    x EST_DU_TYPE NOMBRE
    y EST_DU_TYPE NOMBRE
    m EST_DU_TYPE NOMBRE
    i EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    x0 PREND_LA_VALEUR 1
    y0 PREND_LA_VALEUR 1
    dx PREND_LA_VALEUR 10
    dy PREND_LA_VALEUR 4
    m PREND_LA_VALEUR dy/dx
    POUR i ALLANT_DE 0 A dx
      DEBUT_POUR
      x PREND_LA_VALEUR x0+i
      y PREND_LA_VALEUR round(y0+m*i)
      TRACER_POINT (x,y)
      FIN_POUR
FIN_ALGORITHME
```

4.3 Implémentation en C

```
void TraceSegmentNaif (int x0, int y0, int dx, int dy,
                       unsigned char couleur)
{
    int i, x, y;
    float m = ((float) dy / ((float) dx);
    for (i = 0 ; i < dx ; i++)
    {
        x = x0 + i;
        y = Round(y0 + m * i);
        DessinePixel (x, y, couleur);
    }
}
```

5 Algorithme incrémental de base

5.1 Principe

En posant $y_i = y_0 + m.i$ et $x_i = x_0 + i$, pour obtenir un couple d'entiers qui définissent des coordonnées discrètes dans une fenêtre graphique, on doit afficher les pixels de la forme : $(x_i, \text{Round}(y_i))$ pour $i = 0, \ldots, dx$.

Cependant, on peut remarquer que :

$$\begin{aligned} y_{i+1} &= y_0 + m(i+1) \\ &= y_0 + m.i + m \\ &= y_i + m \end{aligned}$$

On obtient ainsi la relation de récurrence :

$$y_{i+1} = y_i + m$$

On peut donc se servir de y_i pour calculer y_{i+1}, c'est ce que l'on appelle un *algorithme incrémental* : on calcule une valeur (ici une coordonnée) à partir de la précédente. On obtient l'algorithme ci-dessous.

```
Fonction TraceSegmentIncremental(x0, y0, dx, dy : entier,
couleur : caractère) :
    y : flottant
    m : flottant
    x1 : entier
    y ← y0
    m ← dy / dx
    x1 ← x0 + dx
    Pour x de x0 à x1 faire
        DessinePixel (x, Arrondi(y), couleur)
        y ← y + m
    Fin Pour
Fin
```

Avec cet algorithme, on a supprimé le plus couteux : la multiplication entre flottants à chaque étape. Cependant, il reste une addition en virgule flottante et un calcul d'arrondi. L'objet de la prochaine approche (l'algorithme du point milieu) est de supprimer le calcul d'arrondi et l'utilisation de variables en virgule flottante.

5.2 Ecriture avec AlgoBox

La transcription de l'algorithme incrémental de base entre les points ($x_0 = 1, y_0 = 1$) et ($x_1 = 11, y_1 = 5$) avec AlgoBox est la suivante :

```
VARIABLES
  x0 EST_DU_TYPE NOMBRE
  y0 EST_DU_TYPE NOMBRE
  dx EST_DU_TYPE NOMBRE
  dy EST_DU_TYPE NOMBRE
  x EST_DU_TYPE NOMBRE
  y EST_DU_TYPE NOMBRE
  m EST_DU_TYPE NOMBRE
  x1 EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
  x0 PREND_LA_VALEUR 1
  y0 PREND_LA_VALEUR 1
  dx PREND_LA_VALEUR 10
  dy PREND_LA_VALEUR 4
  y PREND_LA_VALEUR y0
  m PREND_LA_VALEUR dy/dx
  x1 PREND_LA_VALEUR x0+dx
  POUR x ALLANT_DE x0 A x1
    DEBUT_POUR
    TRACER_POINT (x,round(y))
    y PREND_LA_VALEUR y+m
    FIN_POUR
FIN_ALGORITHME
```

5.3 Implémentation en C

```
void TraceSegmentIncremental (int x0, int y0, int dx, int dy,
                              unsigned char couleur)
{
```

```
    float y = y0;
    float m = ((float) dy) / ((float) dx);
    int x1 = x0 + dx;
    for (x = x0 ; x <= x1 ; x++)
    {
        DessinePixel (x, Round(y), couleur);
        y = y + m;
    }
}
```

6 Algorithme du point milieu

6.1 Principe

Cette section décrit un algorithme incrémental permettant de calculer chaque pixel à afficher en fonction du précédent en utilisant des opérations peu couteuses d'arithmétique entière.

L'approche proposée est différente de celles vues précédemment. L'idée est la suivante : étant donné que l'on travaille dans le premier octant et que la pente de la droite est comprise entre 0 et 1, le pixel à dessiner sera forcément l'un des deux suivants (parmi les 8 pixels adjacents possibles) : soit il s'agira de celui situé à la droite du pixel courant (que l'on appellera le point *Est*, ou E), soit il s'agira de celui situé en haut à droite (que l'on appellera le point *Nord-Est*, ou NE).

Pour chaque verticale $x = x_0 + i$, on considère le point d'intersection Q entre la droite réelle D que l'on veut discrétiser et la droite verticale d'équation $x = x_0 + i$ (voir figure 5). Ce point Q se trouve sur un segment vertical caractérisé par deux valeurs entières successives de y. Soit M le milieu de ce segment vertical. On veut dessiner le pixel centré sur l'extrémité du segment vertical qui se trouve du même côté que le point Q relativement au point M.

Connaissant le pixel précédent $P = (x_P, y_P)$, le calcul du pixel (x_{P+1}, y_{P+1}) à afficher se déroule de la manière suivante (voir figure 6). Comme dit plus haut, il existe deux possibilités pour le pixel (x_{P+1}, y_{P+1}) :

– le pixel $E = (x_P + 1, y_P)$ (c'est-à-dire le point *Est*) ;
– le pixel $NE = (x_P + 1, y_P + 1)$ (c'est-à-dire le point *Nord-Est*).

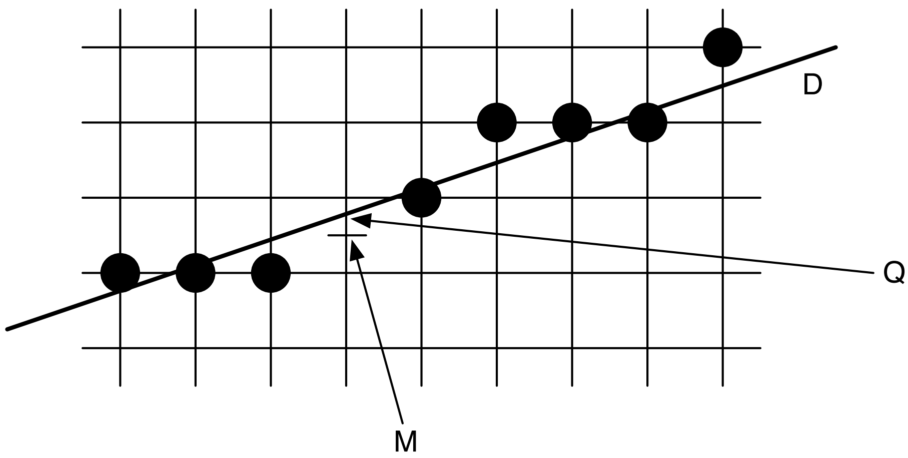


FIGURE 5: Choix du point à éditer pour une verticale donnée.

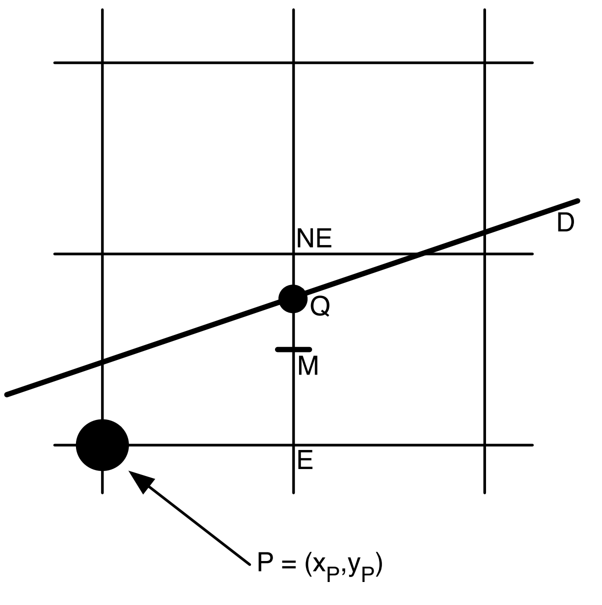


FIGURE 6: Détermination de (x_{P+1}, y_{P+1}) à partir de (x_P, y_P).

Soit Q le point d'intersection de la droite réelle D et de la droite verticale $x = x_P + 1$, et soit M le milieu du segment $[E, NE]$. Pour savoir si l'on choisit le point E ou le point NE, il suffit de déterminer si le point M se trouve au-dessus ou au-dessous du point Q sur ce segment $[E, NE]$. Si le point M se trouve au-dessus de Q, on choisit le point E, sinon on choisit le point NE. Cela revient à déterminer la position du point M par rapport à la droite D. On peut répondre à cette question en utilisant l'équation cartésienne de la droite D (voir section 2.4) :

$$F(x,y) = 0, \text{ avec } F(x,y) = dy(x - x_0) - dx(y - y_0)$$

Pour savoir si le point M est au-dessus ou au-dessous de la droite D, il suffit de calculer le signe de la fonction F au point M dont les coordonnées sont $\left(x_P + 1, y_P + \frac{1}{2}\right)$.

On pose :

$$d_P = 2F\left(x_P + 1, y_P + \frac{1}{2}\right)$$

Cette variable d_P s'appelle la *variable de décision*. Elle est du même signe que la valeur de F au point M. On a :

– si $d_P \leq 0$ alors on choisit le point E ;
– si $d_P > 0$ alors on choisit le point NE.

De plus, d_P est un nombre entier car le rationnel $\frac{1}{2}$ que l'on trouve dans la deuxième composante de F est compensé par le facteur 2 appliqué à l'expression entière.

Le calcul de d_P étant toujours trop couteux, on peut souhaiter faire un calcul incrémental, c'est-à-dire calculer d_{P+1} en fonction de d_P. Pour ce faire, on distingue deux cas, selon que (x_{P+1}, y_{P+1}) est égal à E ou NE.

Premier cas : $(x_{P+1}, y_{P+1}) = E$. On a alors :

$$\begin{aligned}
\frac{d_{P+1}}{2} &= F\left(x_{P+1} + 1, y_{P+1} + \frac{1}{2}\right) \\
&= F\left(x_P + 2, y_P + \frac{1}{2}\right) \\
&= dy\left(x_P + 2 - x_0\right) - dx\left(y_P + \frac{1}{2} - y_0\right)
\end{aligned}$$

et

$$\begin{aligned}
\frac{d_P}{2} &= F\left(x_P + 1, y_P + \frac{1}{2}\right) \\
&= dy\left(x_P + 1 - x_0\right) - dx\left(y_P + \frac{1}{2} - y_0\right)
\end{aligned}$$

d'où :

$$d_{P+1} = d_P + 2dy$$

On pose donc :

$$\delta_E = 2dy$$

Deuxième cas : $(x_{P+1}, y_{P+1}) = NE$. On a alors :

$$\begin{aligned}\frac{d_{P+1}}{2} &= F\left(x_{P+1}+1, y_{P+1}+\frac{1}{2}\right)\\ &= F\left(x_P+2, y_P+\frac{3}{2}\right)\\ &= dy\left(x_P+2-x_0\right) - dx\left(y_P+\frac{3}{2}-y_0\right)\end{aligned}$$

et

$$\frac{d_P}{2} = dy\left(x_P+1-x_0\right) - dx\left(y_P+\frac{1}{2}-y_0\right)$$

d'où :

$$d_{P+1} = d_P + 2(dy-dx)$$

On pose donc :

$$\delta_{NE} = 2(dy-dx)$$

Il est donc possible de déterminer d_{P+1} en fonction de d_P par une addition et en distinguant deux cas. La valeur initiale est :

$$d_0 = 2F\left(x_0+1, y_0+\frac{1}{2}\right) = 2dy - dx$$

Lors de l'implémentation de l'algorithme, on peut représenter la variable de décision d_P par une variable de type `int` (entière), ce qui accélère considérablement les calculs.

L'algorithme du point milieu est donné ci-dessous. A chaque étape, on effectue un test du signe de d_P, une addition entre entiers et une ou deux incrémentations d'entiers, ce qui est beaucoup moins couteux que l'opération d'arrondi et l'addition en virgule flottante que l'on avait avec l'algorithme incrémental de base.

Fonction TraceSegmentPointMilieu(x_0, y_0, x_1, y_1 : **entier**, couleur : **caractère**) :

dx, dy, dp, deltaE, deltaNE, x, y : **entier**
$dx \leftarrow x_1 - x_0$
$dy \leftarrow y_1 - y_0$
$dp \leftarrow 2*dy - dx$ *(Valeur initiale de dp)*
$deltaE \leftarrow 2 * dy$
$deltaNE \leftarrow 2 * (dy - dx)$
$x \leftarrow x_0$
$y \leftarrow y_0$
DessinePixel (x, y, couleur)
Tant que $(x < x_1)$ **faire**
  Si $(dp \leq 0)$ **Alors**
    (On choisit le point E)
    $dp \leftarrow dp + deltaE$
    $x \leftarrow x + 1$
  Sinon
    (On choisit le point NE)
    $dp \leftarrow dp + deltaNE$
    $x \leftarrow x + 1$
    $y \leftarrow y + 1$
  Fin Si
  DessinePixel (x, y, couleur)
Fait
Fin

6.2 Ecriture avec AlgoBox

```
VARIABLES
  x0 EST_DU_TYPE NOMBRE
  y0 EST_DU_TYPE NOMBRE
  x1 EST_DU_TYPE NOMBRE
  y1 EST_DU_TYPE NOMBRE
  dx EST_DU_TYPE NOMBRE
  dy EST_DU_TYPE NOMBRE
  dp EST_DU_TYPE NOMBRE
  deltaE EST_DU_TYPE NOMBRE
```

```
    deltaNE EST_DU_TYPE NOMBRE
    x EST_DU_TYPE NOMBRE
    y EST_DU_TYPE NOMBRE
  DEBUT_ALGORITHME
    x0 PREND_LA_VALEUR 1
    y0 PREND_LA_VALEUR 1
    x1 PREND_LA_VALEUR 11
    y1 PREND_LA_VALEUR 5
    dx PREND_LA_VALEUR x1-x0
    dy PREND_LA_VALEUR y1-y0
    dp PREND_LA_VALEUR 2*dy-dx
    deltaE PREND_LA_VALEUR 2*dy
    deltaNE PREND_LA_VALEUR 2*(dy-dx)
    x PREND_LA_VALEUR x0
    y PREND_LA_VALEUR y0
    TRACER_POINT (x,y)
    TANT_QUE (x<x1) FAIRE
      DEBUT_TANT_QUE
      SI (dp<=0) ALORS
        DEBUT_SI
        dp PREND_LA_VALEUR dp+deltaE
        x PREND_LA_VALEUR x+1
        FIN_SI
        SINON
          DEBUT_SINON
          dp PREND_LA_VALEUR dp+deltaNE
          x PREND_LA_VALEUR x+1
          y PREND_LA_VALEUR y+1
          FIN_SINON
      TRACER_POINT (x,y)
      FIN_TANT_QUE
  FIN_ALGORITHME
```

En sortie, on obtient le graphique de la figure 7.

6.3 Implémentation en C

```c
void TraceSegmentPointMilieu (int x0, int y0, int x1, int y1,
                              unsigned char couleur)
{
```

```
    int dx = x1 - x0;
    int dy = y1 - y0;
    int dp = 2 * dy - dx;        /* Valeur initiale de dp */
    int deltaE = 2 * dy;
    int deltaNE = 2 * (dy - dx);
    int x = x0;
    int y = y0;
    DessinePixel (x, y, couleur);
    while (x < x1)
    {
        if (dp <= 0)             /* On choisit le point E */
        {
            dp = dp + deltaE;    /* Nouveau dp */
            x++;                 /* Calcul de x_p+1 */
                                 /* y_p+1 = y_p */
        }
        else                     /* On choisit le point NE */
        {
            dp = dp + deltaNE;   /* Nouveau dp */
            x++;                 /* Calcul de x_p+1 */
            y++;                 /* Calcul de y_p+1 */
        }
        DessinePixel (x, y, couleur);
    }
}
```

7 Algorithme dans le cas général de segments quelconques

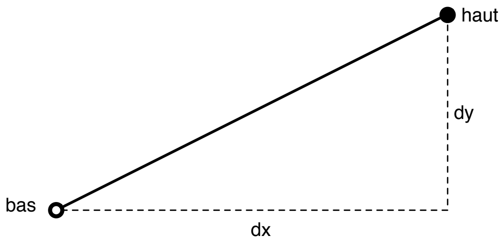
Nous n'avons jusqu'à présent traité que le cas des segments dont la pente était comprise entre 0 et 1 (c'est-à-dire les segments contenus dans le premier octant). Cette section a pour objet l'adaptation de l'algorithme du point milieu au cas des segments quelconques.

Dans le but de réduire le nombre de cas, on considère que le segment à discrétiser est composé de deux extrémités (x_{bas}, y_{bas}) et (x_{haut}, y_{haut}) et que l'on a $y_{bas} \leq y_{haut}$ (quitte à échanger les rôles des deux extrémités du segment si nécessaire). Les deux principaux cas sont donc les suivants (chaque cas étant partagé en deux sous-cas, ce qui fait au total quatre cas).

7.1 Premier cas : si $x_{haut} \geq x_{bas}$

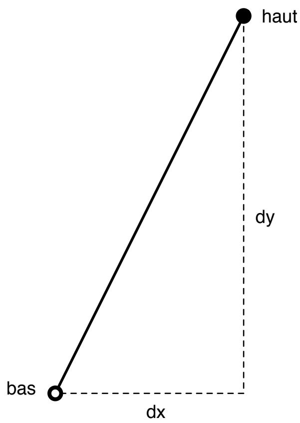
On a alors $dx = x_{haut} - x_{bas}$ et $dy = y_{haut} - y_{bas}$.

Cas 1.a : si $dx \geq dy$



- la valeur initiale d_0 est égale à $2dy - dx$;
- $\delta_E = 2dy$ et $\delta_{NE} = 2(dy - dx)$;
- à chaque étape, choisir le point E revient à incrémenter x et choisir le point NE revient à incrémenter x et y.

Cas 1.b : si $dx < dy$

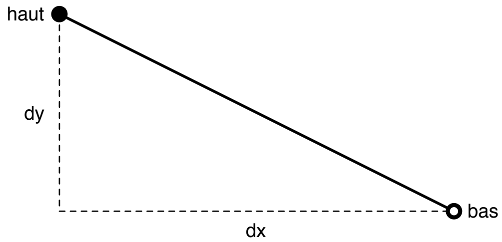


- la valeur initiale d_0 est égale à $2dx - dy$;
- $\delta_E = 2dx$ et $\delta_{NE} = 2(dx - dy)$;
- à chaque étape, choisir le point E revient à incrémenter y et choisir le point NE revient à incrémenter x et y.

7.2 Deuxième cas : si $x_{haut} < x_{bas}$

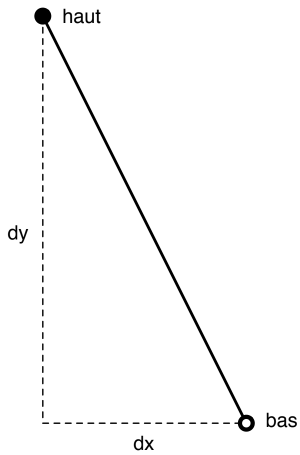
On a alors $dx = x_{bas} - x_{haut}$ et $dy = y_{haut} - y_{bas}$.

Cas 2.a : si $dx \geq dy$



- la valeur initiale d_0 est égale à $2dy - dx$;
- $\delta_E = 2dy$ et $\delta_{NE} = 2(dy - dx)$;
- à chaque étape, choisir le point E revient à décrémenter x et choisir le point NE revient à décrémenter x et incrémenter y.

Cas 2.b : si $dx < dy$



- la valeur initiale d_0 est égale à $2dx - dy$;
- $\delta_E = 2dx$ et $\delta_{NE} = 2(dx - dy)$;
- à chaque étape, choisir le point E revient à incrémenter y et choisir le point NE revient à décrémenter x et incrémenter y.

8 Références

Ce document est principalement inspiré des trois références suivantes :

[1] Rémy Malgouyres. *Algorithmes pour la synthèse d'images et l'animation 3D*. Collection "Sciences Sup", pp. 11–19, Dunod, 2005.
[2] Wikipedia. *Algorithme de tracé de segment*. `http://fr.wikipedia.org/wiki/Algorithme_de_trace_de_segment`
[3] Wikipedia. *Algorithme de tracé de segment de Bresenham*. `http://fr.wikipedia.org/wiki/Algorithme_de_trace_de_segment_de_Bresenham`

Le lecteur pourra également se tourner vers les ressources suivantes, pour approfondir le sujet :

[4] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hugues et Richard L. Phillips. *Introduction à l'infographie*. Addison-Wesley, 1995.

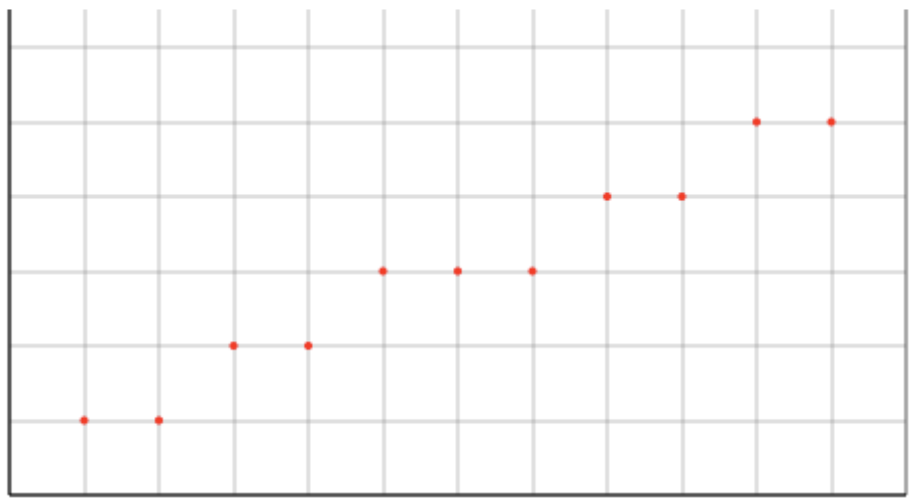


FIGURE 7: Sortie graphique de l'algorithme du point milieu sous AlgoBox, appliqué aux points ($x_0 = 1, y_0 = 1$) et ($x_1 = 11, y_1 = 5$).