Tracer de segments - Optimisation

PROPRIÉTÉ

Une droite du plan peut être caractérisée selon une équation de la forme :

* x=c si cette droite est parallèle à l'axe des ordonnées (« verticale »)
* y=mx+p si cette droite n'est pas parallèle à l'axe des ordonnées.

Dans le second cas, m est appelé coefficient directeur et p ordonnée à l'origine.

REMARQUES

L'équation d'une droite peut s'écrire sous plusieurs formes.   
Par exemple y = 2x–1 est équivalente à y−2x+1=0 ou 2y−4x+2=0, etc.  
Les formes *x*=*c* et *y*=*mx*+*p* sont appelées équation réduite de la droite.

Cette propriété indique que toute droite qui n'est pas parallèle à l'axe des ordonnées est la représentation graphique d'une fonction affine.

Une droite parallèle à l'axe des abscisses a un coefficient direct m égal à zéro.

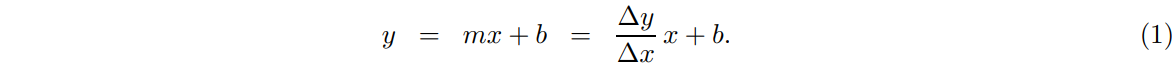
Son équation est donc de la forme y=*p*. C'est la représentation graphique d'une fonction constante.

PROPRIÉTÉ

Soient A et B deux points du plan tels que x​A​​≠x​B​​.   
Le coefficient directeur de la droite (AB) est : *m* =​ *y*​*B*​​−*y*​*A*​​​​ / *x*​*B*​​−*x*​*A*​​​​

ALGORITHME Primitif

Pour écrire l'algorithme, nous commençons par la formulation commune d'interception de pente d'une ligne, où la pente est simplement le rapport de la variation de y à la variation de x.



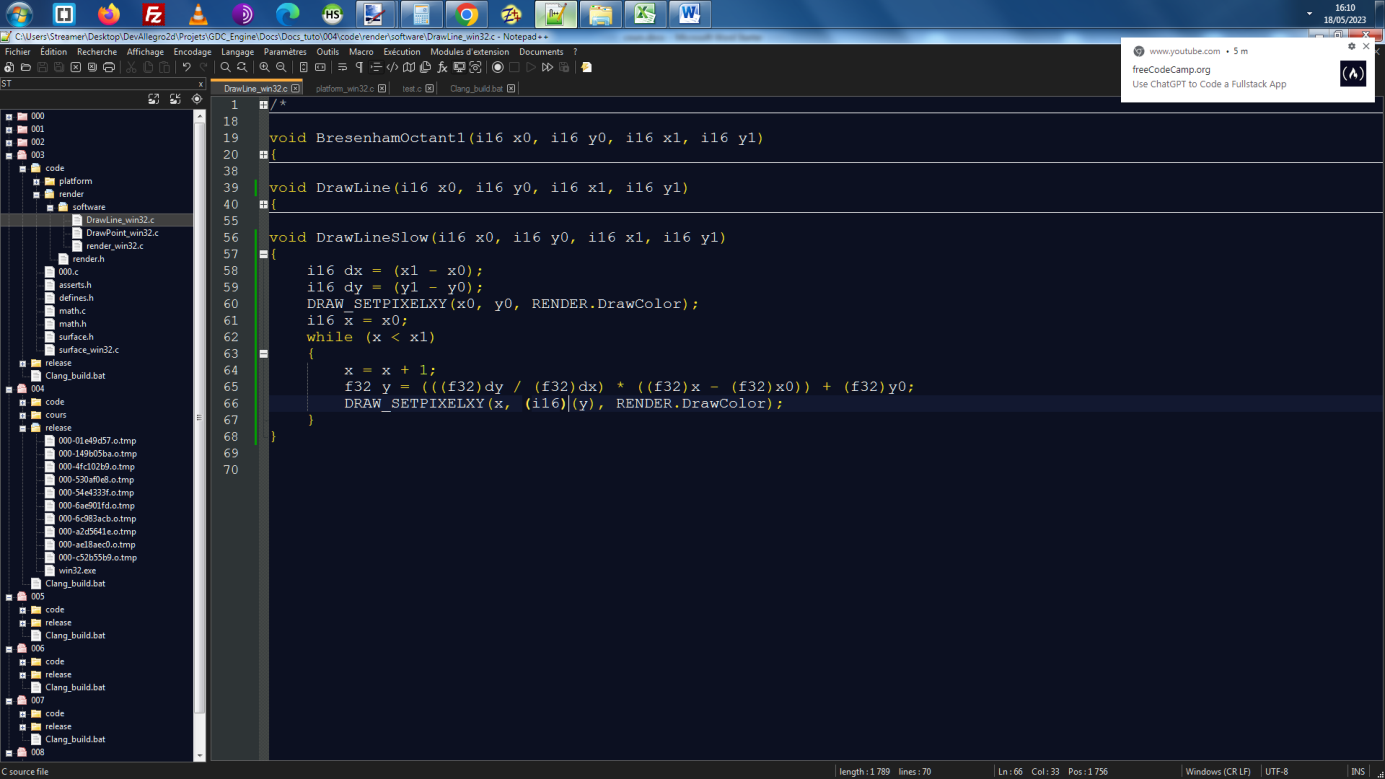
m est à la pente de la droite. Dans notre cas: m = ∆y / ∆x

Notre premier point (x0, y0) appartient à la droite, il suit aussi l'équation: y0 = m\*x0 + b

Maintenant si on soustrait cette équation à la précédente, on obtient: y - y0 = m\*(x - x0) + b – b et b disparaît.

Après une petite réécriture, on obtient une nouvelle formule pour notre ligne: y = m \* (x - x0) + y0

Fonction DrawLine(x0, y0, x1, y1)

 1 ∆x🡨x1-x0

2 ∆y🡨y1-y0

3 SetPixel(x0, y0)

4 x🡨x0

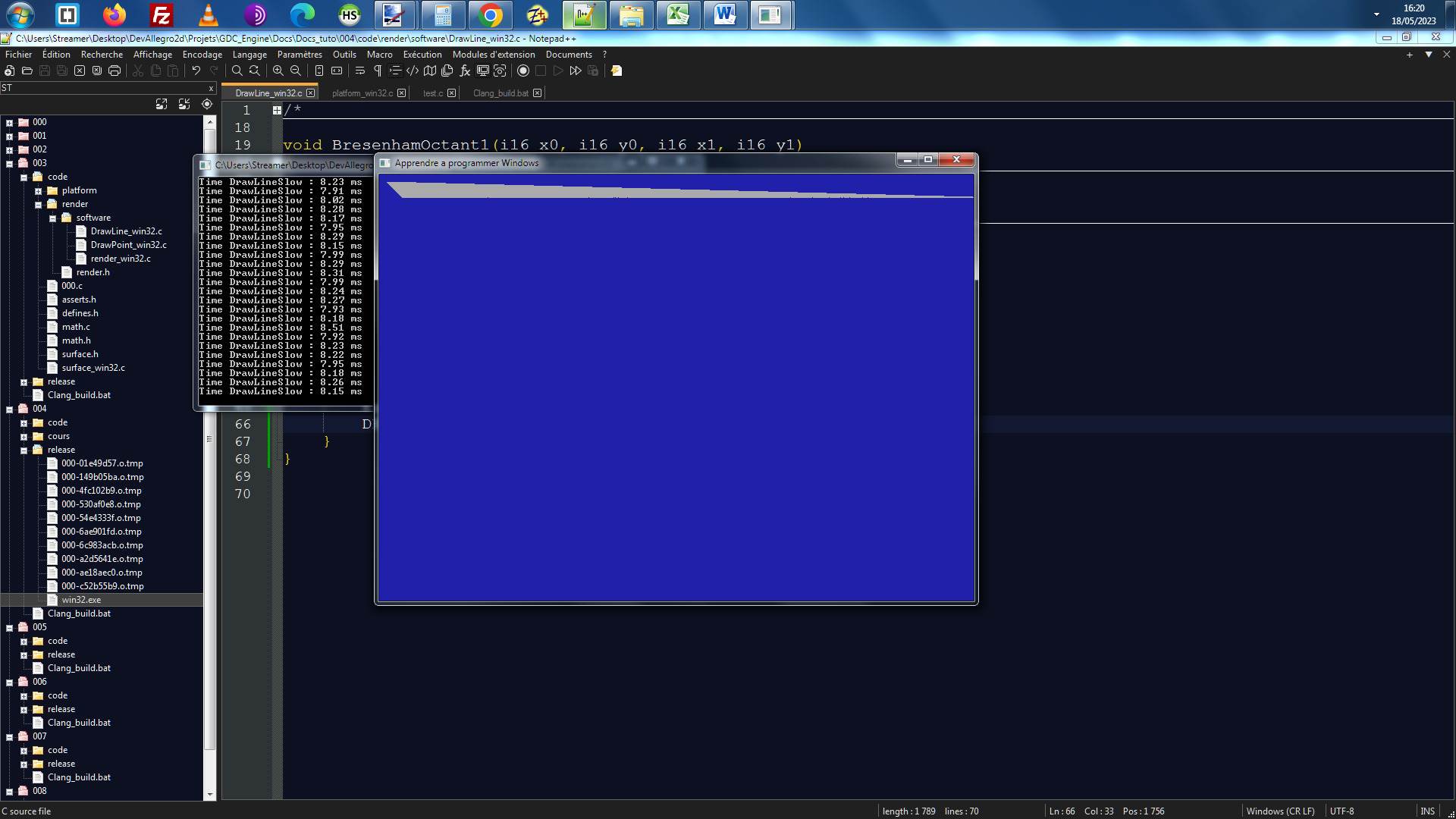
5 **while**(x<x1) **do**

6 x🡨x+1

7 y🡨(∆y/∆x)\*(x-x0)+y0

8 SetPixel(x,y)

9 **endwhile**



L’algorithme s’ouvre de deux lacunes : Les calculs avec des variables réelles engendrent des imprécisions, celui-ci peut être optimisé afin de gagner en temps d’exécution.

OPTIMISATIONs

**Sortons de la boucle un maximum de calculs.**

Le calcul de la pente qui est constant.

Le y0 est constant.

**Supprimons la multiplication.**

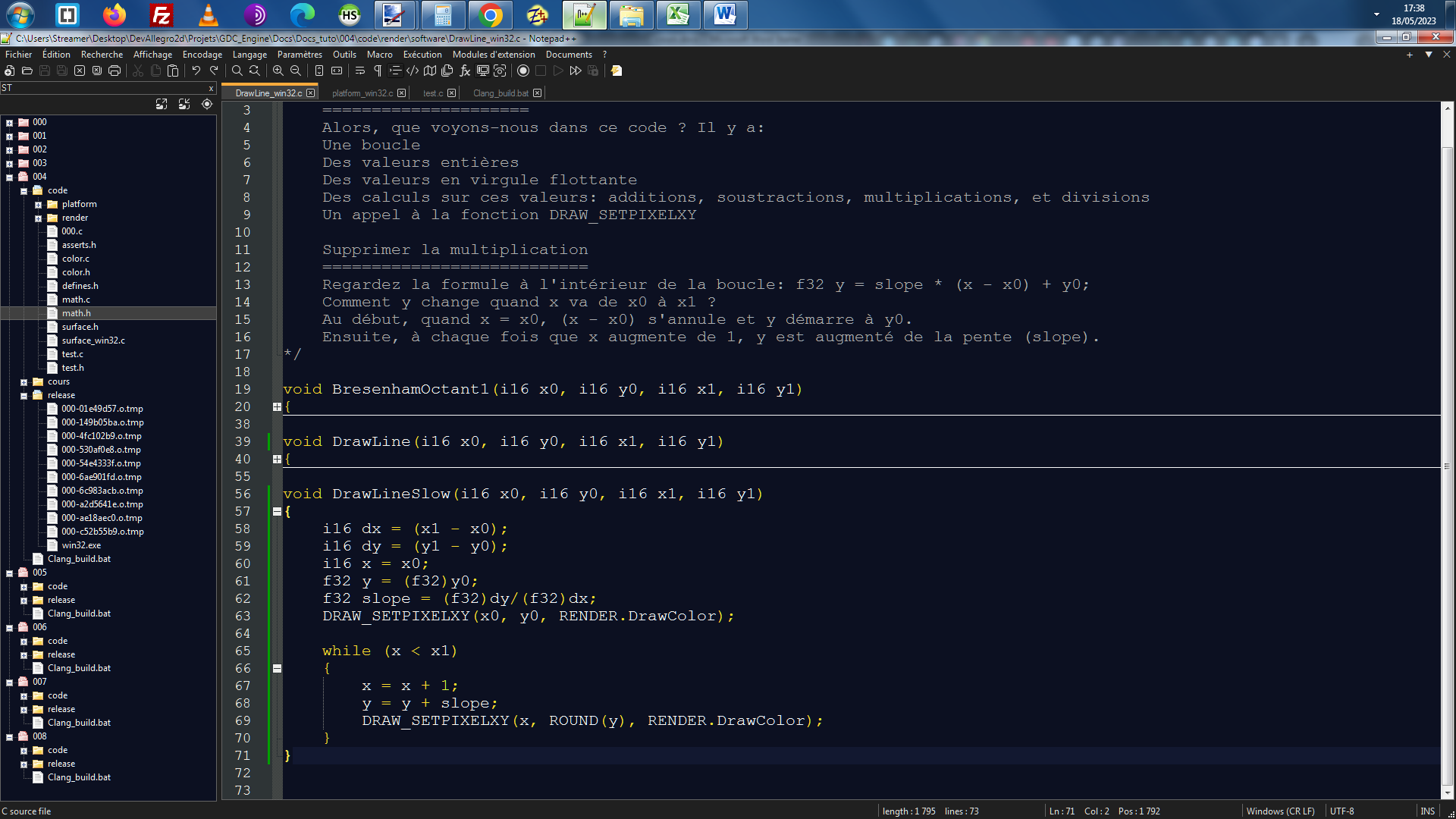
Regardez la formule à l'intérieur de la boucle: y = m\*(x - x0) + y0

Comment y change quand x va de x0 à x1 ?

Au début, quand x = x0, (x - x0) s'annule et y démarre à y0.

Ensuite, à chaque fois que x augmente de 1, y est augmenté de la pente (slope).

Fonction DrawLine(x0, y0, x1, y1)

 1 ∆x🡨x1-x0

2 ∆y🡨y1-y0

.3 Slope 🡨 ∆y/∆x

.4 SetPixel(x0, y0)

5 y🡨y0

6 x🡨x0

.7 **while**(x<x1) **do**

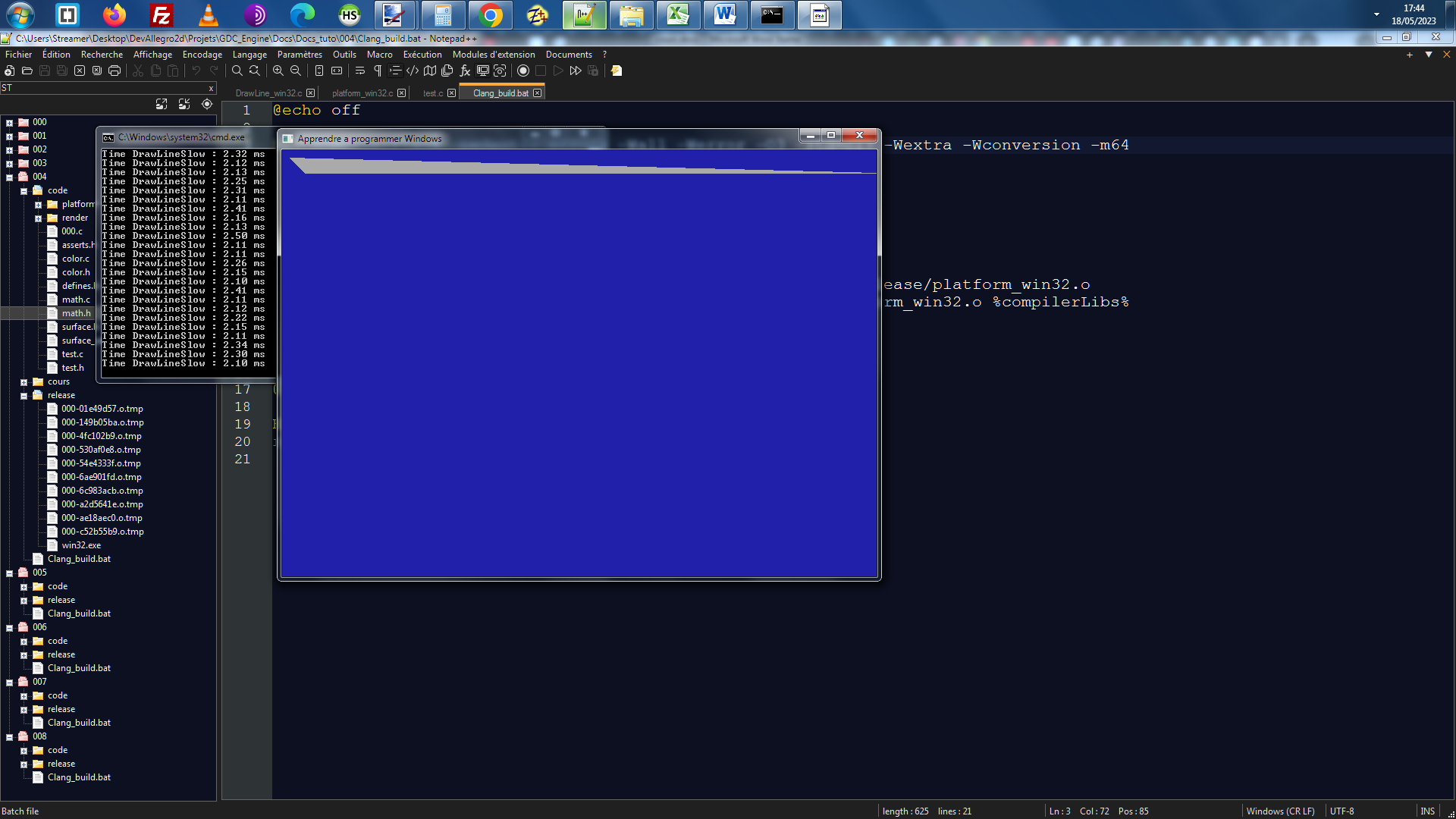
8 x🡨x+1

9 y🡨y+Slope

10 SetPixel(x,y)

11 **endwhile**

Reste à corriger la perte d’information lors de la conversion de y en entier, pour cela il faut calculer l'arrondi entier au plus proche de la valeur spécifiée. Nous allons créer la macro ROUND spécifique à nos entiers : #define ROUND(x) ((x)>=0 ? (i16) ((x)+0.5) : (i16) ((x)-0.5))



Nous avons gagné après ces optimisations près de 6 ms, voyons maintenant comment faire pour aller encore plus loin sur l’optimisation.

Algorithme de Bresenham

Ces notes décrivent un algorithme de tramage de ligne classique, initialement publié en 1965 dans un article sous le titre « Algorithm for Computer Control of a Digital Plotter par Jack Bresenham d'IBM ».

L'algorithme prend deux paires ordonnées d'entiers, représentant les extrémités d'un segment de ligne, et détermine la collection de "pixels", également représentés par des coordonnées entières, qui se rapproche le mieux de la ligne mathématique reliant les deux points d'extrémité.

En général, seuls les deux points finaux se situent exactement sur la ligne, l'algorithme doit donc décider quels pixels intermédiaires sont "suffisamment proches" pour être considéré sur la ligne.

La beauté de l'algorithme de Bresenham est qu'il fonctionne entièrement en entier arithmétique, et est donc bien adapté au matériel graphique de bas niveau.

L’algorithme de Bresenham, et des variantes de celui-ci sont couramment implémentées dans les puces d'accélérateur graphique d'aujourd'hui.

Observez que l'équation (1) représente y en fonction de x, et peut exprimer chaque ligne dans le plan à l'exception d'une ligne verticale, qui correspond essentiellement à une pente infinie. (Nous nous occuperons des lignes verticales comme un cas particulier.)

Alternativement, nous pouvons réorganiser l'équation afin qu'elle fournisse une représentation implicite de la ligne. En particulier, on peut écrire :



Cette fonction spécifie la même ligne, mais implicitement ; c'est-à-dire tout point (x, y) qui satisfait F(x, y) = 0 est sur la ligne, mais F ne précise pas comment calculer y étant donné x, ou vice versa.

En plus de spécifier la ligne, la fonction F suppose un signe différent aux points qui se trouvent sur différents côtés de la ligne.

En particulier, F(x, y) < 0 lorsque (x, y) se situe au-dessus de la droite, et F(x, y) > 0 lorsque (x, y) se situe au-dessous de la droite.

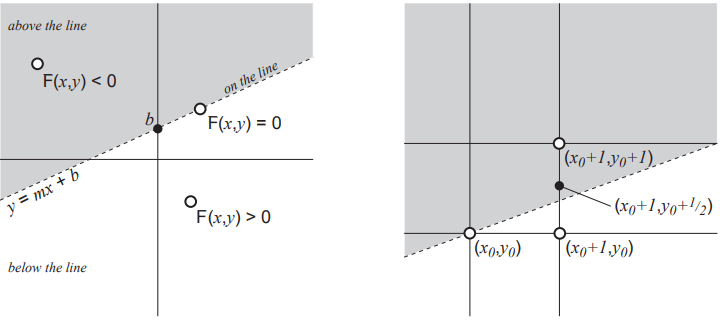
(F1)

Figure 1 : (À gauche) La fonction implicite F qui définit la ligne divise également le plan en trois régions ; F(x, y) < 0 au-dessus de la ligne, F(x, y)=0 sur la ligne et F(x, y) > 0 en dessous de la ligne. (À droite) À partir du point (x0, y0) le prochain point à pixelliser lors du dessin de la ligne sera soit (x0 + 1, y0) soit (x0 + 1, y0 + 1). Le choix dépend du fait que F(x0 + 1, y0 + 1/2) est positif ou négatif.

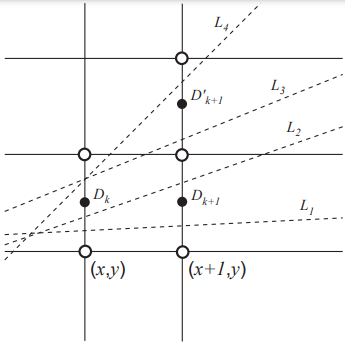
(F2)

Figure 2 : Au fur et à mesure que le processus de rastérisation progresse, nous calculons chaque variable de décision de manière incrémentielle à partir de la variable de décision précédente.

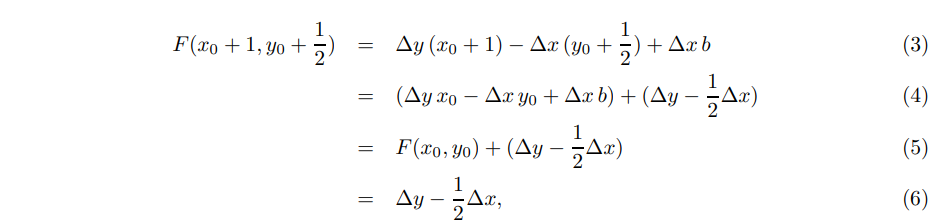
L'augmentation dépend de la décision précédente qui a été prise. Les deux lignes L1 et L2 ci-dessus entraînerait un incrément pour passer de Kk à Dk+1, et les lignes L3 et L4 entraîneraient dans un incrément différent en passant de Dk à D’k+1.

(x, y) se trouve en dessous de la ligne. Voir Figure 1. C'est la propriété que l'algorithme de Bresenham exploite pour déterminer quels pixels se rapprochent le mieux d'une ligne donnée.

Considérons maintenant un segment de droite dont la pente est comprise entre 0 et 1, allant de « gauche » à « droite » ; c'est-à-dire un segment dont les extrémités (x0, y0) et (x1, y1) sont telles que x0≤x1, y0≤y1, et y1−y0 ≤ x1−x0. Contraindre la ligne en tant que telle nous permet de faire quelques hypothèses simplificatrices qui rendent la dérivation de l'algorithme de Bresenham très simple. Une fois que nous avons géré ce cas particulier, toutes les autres lignes seront faciles à manipuler de la même manière.

Considérons maintenant la figure à droite de la figure 1. Elle représente le premier pixel du segment de ligne, en (x0, y0), et la première "décision" que l'algorithme doit prendre ; à savoir, le pixel suivant doit-il être (x0 + 1, y0), ou (x0 + 1, y0 + 1). Ce sont les deux seuls choix logiques, étant donné que la pente de la droite est comprise entre 0 et 1. Pour prendre cette décision, il suffit de considérer le point qui se trouve à mi-chemin entre les deux possibilités et de déterminer si ce point est "au-dessus" ou "en dessous" de la ligne; c'est-à-dire que nous regardons le signe de F(x0 + 1, y0 + 1/2).

Nous appellerons la valeur résultante de F la variable de décision, car il sera utilisé pour décider entre les deux pixels possibles à sélectionner ensuite. Notez que la grandeur de la variable de décision n'est pas pertinente seul son signe est nécessaire pour prendre la décision. Brancher le coordonnées du point intermédiaire en F et en simplifiant, on a :



La simplification finale est due au fait que F(x0, y0) = 0. Enfin, puisque c'est seulement le signe de cette valeur qui nous intéresse, pas sa grandeur, nous pouvons simplifiez-en multipliant par 2.

Ainsi, notre première variable de décision est :

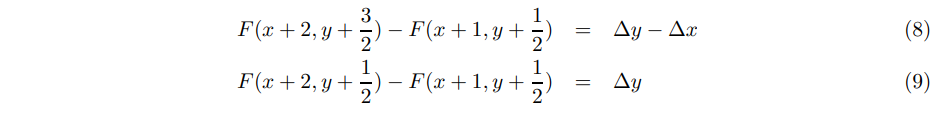


La variable de décision dans l'équation (7) nous permet de prendre la première décision c'est-à-dire si le seconde pixel a la même valeur y que le premier, ou si il est supérieur de 1 ?

Mais qu'en est-il des décisions ultérieures ? Dans tous les cas, nous pourrions simplement calculer F au point intermédiaire et regarder son signe. Cependant, il existe une méthode légèrement plus efficace, à savoir le calcul incrémental. Au lieu de calculer la valeur de F à chaque point de décision, nous calculons simplement en quoi elle diffère de la décision précédente.

La figure 2 illustre trois points de décision potentiels et leurs variables de décision correspondantes, Dk, Dk+1 et D’k+1. En supposant que Dk a déjà été calculé, lequel de Dk+1 et D’k+1 devrait être calculé ensuite, et en quoi diffèrent-ils de Dk ?

Les différences sont faciles à calculer, comme suit :



Pour garder les incréments cohérents avec notre définition précédente de D, nous devons à nouveau les multiplier par deux. Nous pouvons maintenant donner l'algorithme complet. Pixellisez la ligne de (x0, y0) à (x1, y1), où toutes les coordonnées sont des entiers qui satisfont x0≤x1, y0≤y1 et y1−y0≤x1- x0.

Ainsi, la ligne est supposée aller de gauche à droite et ont une pente comprise entre 0 et 1 (c'est-à-dire entre 0 et 45 degrés). Les lignes de toutes les autres pentes peuvent être manipulées en utilisant des variations mineures de ce code.

x1 > x0: x augmente. y1 > y0 : y augmente.

(x1-x0) > (y1-y0) : la ligne est plus horizontale que verticale.

