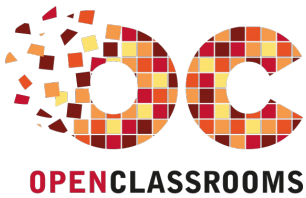
L'écran graphique en TI-Basic

Par myst6re



www.openclassrooms.com

Sommaire

L'écran graphique en TI-Basic

Par  myst6re

Mise à jour : 01/01/1970
Difficulté : Facile

(cc) BY-NC-SA

Vous cherchez désespérément comment faire un programme en TI-Basic qui soit **beau** ? N'en avez-vous pas assez de cet écran HOME qui ne permet que d'afficher du texte en gros caractères ? Ce tutoriel est là pour vous aider...

Certaines notions sur le TI-Basic ne seront pas abordées ici, pour tout ce qui n'est pas graphique lisez le tutoriel dédié au TI-Basic. Je tiens aussi à ajouter que ce tutoriel porte sur un ti-basic qu'on appelle le "83 basic", pour ceux qui ont une TI-89 (ou plus gros) il y aura peut-être de grosses différences, vous êtes prévenus !

Partie 1 : Le graphique

Vous allez tout apprendre dans cette partie : initiation à l'écran graphique, apprentissage des fonctions de dessin, et plus si vous êtes un bon zér0 !

Les bases de l'écran graphique

Voici la base, THE base, il faut en passer par là pour être à l'aise par la suite. Ne vous inquiétez pas, ce n'est pas compliqué du tout .

L'écran du graphe

Sur votre calculatrice, vous avez trois écrans distincts :

- l'écran Home ;
- l'écran Table ;
- l'écran Graph.

Le premier type d'écran se compose de 8 lignes et 16 colonnes, c'est là que n'importe quel utilisateur fait ses calculs (1+1 ; cos(28²)+e^(25)). Le second est utilisé pour afficher des valeurs de manière organisée (sous forme de tableau).

Mais ce qui nous intéresse le plus dans ce tutoriel, c'est le dernier : l'écran Graph ! Celui-ci est utilisé pour afficher des fonctions ou bien des graphiques statistiques. Si vous ne voyez pas de quoi je parle, appuyez sur le bouton TRACE de votre calculatrice .

Ah... il est beau non ?

Il y a plein de trucs sur mon écran : des axes, un graphe qui s'est tracé...

C'est sûr il va falloir faire le ménage avant d'y installer nos propres dessins . Pour cela je vous propose le premier programme du chapitre !

Code : Autre

```
:AxesOff
:GridOff
:FnOff
:PlotsOff
```

Petite description de chaque fonction (avec les touches et les menus d'accès entre crochets) :

- AxesOff [2ND + ZOOM] : n'affiche pas les axes sur l'écran du graphe ;
- GridOff [2ND + ZOOM] : n'affiche pas la grille de points sur la fenêtre du graphe ;
- FnOff [VARS -> Y-VARS -> 4:On/Off...] : désélectionne toutes les fonctions (Y1, Y2...) ;
- PlotsOff [2ND + Y=] : désélectionne tous les graphiques (Plot1, Plot2, Plot3).

Alors ? Vous suivez toujours ? Allez-y, créez un programme et recopiez celui que je viens de faire. Petite astuce pour la suite : toutes les fonctions de la calculatrice (enfin... presque) peuvent se trouver dans le catalogue [2ND + 0].

Voilà c'est fait, mais quand j'exécute, à part "Done" rien ne se passe !

Oui c'est vrai que mon programme n'affiche pas le graphe à la fin . Il existe des commandes pour se rendre au type d'écran que l'on veut :

- écran home : *Disp* ;

- écran Graph : *DispGraph* ;
- écran Table.... *DispTable* (au cas où vous ne l'auriez pas deviné) !

En fait si je n'ai pas mis de *DispGraph* dans mon programme c'est pour vous dire que parfois ce n'est pas nécessaire. Par exemple, si vous voulez afficher un point sur l'écran graph, vous allez utiliser *Pxl-On* et cela suffira à aller sur l'écran Graph. Vous voyez ce que je veux dire ? La plupart des fonctions de dessins affichent l'écran graph quand on les utilise, alors à quoi bon s'encombrer avec un *DispGraph* ? (Bon ok ici c'était nécessaire .)

Oh ! Une autre chose très importante (mais où ai-je la tête ?!) : si vous voulez effacer l'écran Graph, utilisez *ClrDraw* (et bien entendu *ClrHome* pour l'écran Home).

Il est aussi possible de mettre le paramètre "Full" au début du programme, car si la calculatrice est en mode HORIZ ou G-T, l'écran du graphe ne va pas s'afficher correctement (allez dans MODE et testez par vous-mêmes !)

Mini récapitulatif

- Pour afficher l'écran GRAPH, utilisez *DispGraph*.
- Pour effacer l'écran GRAPH, utilisez *ClrDraw*.
- Pour nettoyer l'écran GRAPH avant de l'utiliser, mettez ceci au début de votre programme :

Code : Autre

```
:AxesOff
:GridOff
:FnOff
:PlotsOff
:ClrDraw
```

Les repères

Maintenant que vous avez appris à "initialiser" votre écran graphe, il faut le paramétrer. Pour cela il faut que je vous parle des repères :

- le repère absolu ;
- le repère mathématique.

Repère Absolu

Le repère absolu c'est votre écran : il est composé de 95*63 pixels comme ceci :

L'écran est composé de 96*64 pixels mais la dernière ligne et la dernière colonne de pixels ne sont pas utilisables en TI-Basic.

Repère Mathématique

Le repère mathématique est celui que vous utilisez quand vous tracez un graphe. Il varie en fonction des paramètres de la fenêtre (touche WINDOW pour voir ces paramètres).

Il faut ABSOLUMENT que je vous apprenne à paramétrer la fenêtre . Sachez que l'utilisateur de la calculatrice peut modifier ces paramètres "à la main" en appuyant sur la touche WINDOW, donc pour que vous soyez sûrs du repère que vous allez avoir quand vous allez exécuter votre programme, il faut que vous mettiez au début de votre code ces lignes :

Code : Autre

```
:-10→Xmin
:10→Xmax
:-10→Ymin
:10→Ymax
```

Bien sûr vous pouvez mettre autre chose que -10 et 10 (c'est comme vous voulez !).

Le zoom ZStandard fait la même chose que ton code, ou je me trompe ?

Mais oui bien vu ! Il existe des paramètres prédéfinis que l'on peut appeler grâce aux fonctions de Zoom (touche ZOOM de la calculatrice), c'est plus léger . Ainsi pour mon code précédent on aurait tout aussi bien pu mettre :

Code : Autre

```
:ZStandard
```

Mise en relation des deux repères

Si je vous apprends les deux repères de la calculatrice, c'est d'abord pour que vous puissiez tout savoir, et ensuite car on a des fonctions pour le repère mathématique (*Line(*, *Pt-On(*...) et des fonctions pour le repère absolu (*Text(*, *Pxl-On(*...) !

Ce ne serait pas mal si on pouvait facilement passer d'un repère à un autre ! Ce que je veux dire c'est qu'il est souvent mieux de mettre le repère mathématique aux dimensions du repère absolu. Ne posez pas de question, vous allez comprendre, écrivez votre repère comme suit :

Code : Autre

```
:0→Xmin
:94→Xmax
:-62→Ymin
:0→Ymax
```

Mais... Mais il y a un problème avec les Y ! Pourquoi Ymin est-il négatif ?

Pour la simple raison ma bonne dame qu'on ne peut pas faire autrement , si on avait mis Ymin égal à 0 et Ymax égal à 62 ça aurait été dans le sens inverse du repère absolu (mais non ne faites pas cette tête je suis sûr que vous avez compris). Ici, le repère mathématique se confond avec le repère absolu, à l'exception de l'axe des ordonnées qui est inversé ([10,10] en repère absolu devient ici [10,-10] en repère mathématique).

Avec cette initialisation du repère ça devrait être plus simple pour la suite. Mais bon rien ne vous empêche de mettre un repère mathématique différent.

Assimilez bien toutes ces notions de repère, sans elles vous ne pourrez pas comprendre la suite du tutoriel.

Récapitulatif

Voilà ! Ouf c'est fini pour l'initialisation ! Enfin je crois... Ah oui j'ai encore une notion importante à vous donner ! Bon on fait le point et je vous en parle d'accord ?

Votre programme doit ressembler à ça (et je rappelle que le repère mathématique peut être fait comme on veut) :

Code : Autre

```
:Full
:AxesOff
:GridOff
:FnOff
:PlotsOff
:0→Xmin
:94→Xmax
```

```
:-62→Ymin
:0→Ymax
:ClrDraw
:DispGraph
```

Et votre écran graphique devient tout blanc, peu importe quels étaient les paramètres graphiques.

Tout ça est bien beau mais lorsque je quitte le programme, les paramètres que j'ai spécifiés restent ! Comment je fais moi si je veux conserver tous mes paramètres, garder mes graphes et tout et tout ?

Justement c'est de ça que j'allais vous parler : comment bien arrêter un programme ? Pour sauvegarder tous les réglages **graphiques** de l'utilisateur, il existe un truc : le GDB (*Graph DataBase*). Le fonctionnement est simple : on utilise la fonction StoreGDB (qui sauvegarde les paramètres) au tout début du programme et la fonction RecallGDB (qui rappelle les paramètres sauvegardés) tout à la fin. Vous pouvez créer dix variables de sauvegarde (GDB) maximum en mettant simplement un numéro entre 0 et 9 après les deux fonctions :

Code : Autre

```
:StoreGDB 0   //GDB0 est créé
:Full
:AxesOff
:GridOff
:FnOff
:PlotsOff
:0→Xmin
:94→Xmax
:-62→Ymin
:0→Ymax
:ClrDraw
:DispGraph
(Votre code...)
:RecallGDB 0   //Le contenu de GDB0 est rappelé
:DelVar GDB0   //Et n'oubliez pas d'effacer la sauvegarde ! (appuyez sur VARS, p
:Disp          //On retourne à l'écran Home
```

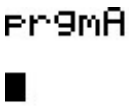
Si vous exécutez cela, miracle ! C'est comme si rien n'avait été changé !

Le GDB ne contient que tout se qui a rapport avec les **graphes**, donc le paramètre **Full** reste après l'exécution du programme.

Dernière petite astuce

Lorsque vous quittez le programme et que vous retournez à l'écran HOME, le mot "Done" apparait. Il existe plusieurs astuces pour le faire disparaitre. En voici deux :

Le disp amélioré

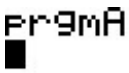


On ajoute une ligne avec un guillemet après le *Disp*. Cette méthode a un petit inconvénient : une ligne vierge s'affiche sur l'écran HOME.

Code : Autre

```
:Disp
:"
```

L'output parfait



On utilise cette fois-ci un *output* qui n'affiche rien (on met un guillemet et c'est tout). Cette méthode utilise deux fois plus d'octets que la précédente, mais elle présente l'avantage de n'afficher aucune ligne blanche en trop.

Code : Autre

```
:Output(1,1,"
```

Peut-être aimeriez-vous maintenant faire du concret ? Alors passez au chapitre suivant !

Les fonctions utiles

Des cercles, des lignes, des points, le Ti-basic est très... basique pour faire du dessin, c'est simple à apprendre donc.

Les pixels et les points

Dans le menu **DRAW>POINTS** vous avez peut-être remarqué qu'il y a deux méthodes pour afficher un point : Pt et Pxl. La différence entre les deux ? Le premier se place dans le repère mathématique, et le second se place dans le repère absolu. Mais elles ont aussi toutes deux des particularités...

Les points

Les points se placent dans le repère mathématique.

Pt-On(X,Y[,type]

Affiche un point de type ***type*** à la position (***X***,***Y***).

Il existe trois types :

- *1* : un point tout simple ;
- *2* : un petit carré ;
- *3* : un petit plus.

Vous n'êtes pas obligés de mettre le type quand vous voulez que ce soit un point tout simple.

Pt-Off(X,Y[,type]

Même chose que *Pt-On(* sauf que le point est effacé.

Pt-Change(X,Y

Cette fonction agit comme un interrupteur : s'il y a un pixel allumé à la position (***X***,***Y***), celui-ci s'éteint, et vice-versa. Remarquez qu'il n'y a qu'un seul type de point ici : le point tout simple.

Les pixels

Les pixels se placent dans le repère absolu.

Rappel des limites du repère :

- ligne : de 0 à 62 ;
- colonne : de 0 à 94.

Pxl-On(ligne,colonne

Allume un pixel à la position (***ligne***,***colonne***).

Pxl-Off(ligne,colonne

Efface un pixel à la position (***ligne***,***colonne***).

Pxl-Change(ligne,colonne

Cette fonction agit comme un interrupteur : s'il y a un pixel allumé à la position (***ligne***,***colonne***), celui-ci s'éteint, et vice-versa.

Pxl-Test(ligne,colonne

Retourne 1 si le pixel est allumé à la position (***ligne***,***colonne***) et sinon retourne 0.

Cette fonction est très utile pour savoir dans quel état est un pixel à une position donnée, par exemple on peut mettre dans un programme un test comme celui-ci :

Code : Autre

```
:If Pxl-Test(0,0 //Ceci est l'écriture condensée pour dire "si Pxl-
Test(0,0 existe"
:Then
:Text(10,0,"IL Y A UN PIXEL EN (0,0)
:Else
:Text(10,0,"IL N'Y A PAS DE PIXEL EN (0,0)
:End
```

Dessiner

Maintenant que vous êtes familiarisés avec la fonction *Text(*, on peut dire que les fonctions qui vont suivre vont couler de source, je vais donc faire simplement la liste de diverses fonctions de dessin.

Toutes les fonction de cette partie sont dans le **repère mathématique**.

Line(X1,Y1,X2,Y2[,0]

Trace (ou efface) une ligne entre deux points.

Paramètres :

- *X1* et *Y1* : position X et Y d'une extrémité de la ligne.
- *X2* et *Y2* : position X et Y de l'autre extrémité de la ligne.
- *0* (facultatif) : si 0 est spécifié, la ligne est effacée entre (*X1*,*Y1*) et (*X2*,*Y2*).

Horizontal Y

Trace une ligne horizontale.

Paramètres :

- *Y* : position Y où est tracée la ligne.

Vertical X

Trace une ligne verticale.

Paramètres :

- *X* : position X où est tracé la ligne.

Circle(X,Y,rayon[,{i]

Trace un cercle.

Paramètres :

- *X* et *Y* : position X et Y du centre du cercle.
- *rayon* : taille du rayon du cercle.
- *{i* (facultatif) : si "*{i*" est spécifié, le cercle se trace beaucoup plus vite. (Attention : ne fonctionne pas sur calculatrice TI83.)

Afficher et effacer du texte

Affichage

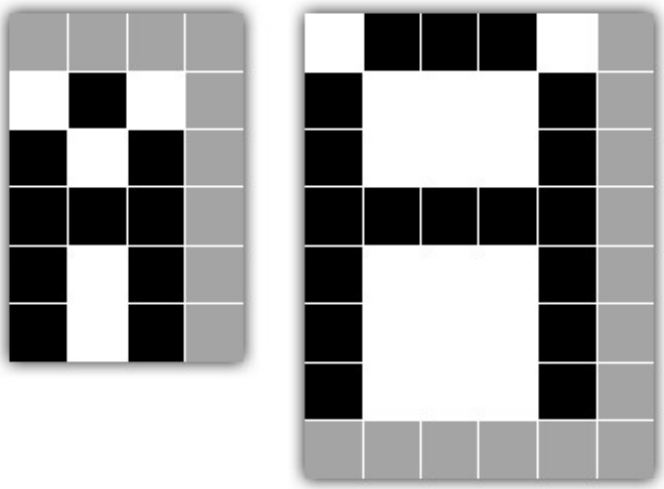
Pour afficher du texte dans l'écran de graphe, il faut utiliser la fonction *Text(* :

Citation : Description de la fonction

Text([*-1*,]*Y*,*X*,*texte*

- *Y* et *X* : ligne et colonne en pixels.
- *texte* : texte à afficher, peut être une chaîne de caractères (ex : "SALUT"), un nombre (ex : 999), une variable (ex : A ou Str1) ou même tout à la fois, séparé par des virgules (ex : "LA VARIABLE A =",A)
- *-1* (facultatif) : si on met -1, le texte est agrandi ! (Attention : ne fonctionne pas sur calculatrice TI83.)

Remarquez que l'on est dans le **repère absolu** (voir chapitre précédent) et qu'il y a deux tailles de texte :



La plupart du temps, la petite police de caractère fait 4*6 pixels et la grande police de caractère fait 6*8 pixels. Je dis la plupart du temps car ce n'est pas toujours le cas pour la petite police ! Par exemple l'espace fait 1*6 pixels, on peut aussi citer le caractère "≠" qui fait 6*6 pixels !

On en vient donc au fait : dans la fonction *Text(*, Y et X ne peuvent pas dépasser certaines valeurs.

- Y : 0->57 pour le petit texte et 0->56 pour le grand.
- X : 0->91 pour le petit texte et 0->90 pour le grand.

Vous pouvez aller jusqu'à 94 pour la valeur du X, mais le texte risque de ne plus s'afficher après les valeurs que j'ai indiquées.

Effacement

L'effacement d'un texte n'est pas ce qu'il y a de plus intuitif... Vous connaissez déjà *ClrDraw* qui efface tout l'écran du Graphe, mais cela n'efface pas localement. La méthode consiste à utiliser un texte avec des espaces, comme ceci :

Code : Autre

```
:Text(0,0,"COUCOU!
:Text(0,0,"                              "
```

Y'a pas trop d'espaces là ?

En fait comme je l'ai dit plus haut, un espace fait 1 pixel de largeur, pas facile avec ça d'effacer un texte ! Utilisons donc la grande police pour agrandir les espaces (6 pixels de largeur). Le seul problème est que ces espaces sont aussi plus hauts ! Et là je dis : "Bah il va falloir faire avec ! "

Code : Autre

```
:Text(0,0,"COUCOU!
:Text(-1,0,0,"      "
```

Et voilà le travail ! Au fait, comprenez bien que cette technique d'effacement n'est pas valable uniquement pour les textes, si vous voulez effacer une zone de l'écran, cette méthode est même la meilleure qui existe (une autre méthode plus précise est possible, mais elle est beaucoup, beaucoup trop lente) !
Vous voyez ce n'était pas compliqué ! Je vous laisse apprendre tout ça (par coeur) !

Techniques (plus) avancées

Vous allez maintenant apprendre à tracer des zones (des carrés, des rectangles, ...) et à vous servir des captures d'écran.

Capture d'écran

Vous avez bien lu, on peut faire des captures d'écran en Ti-Basic, et on ne va pas s'en priver .

Capture et rappel

Si vous voulez capturer l'écran actuel, vous devez utiliser la commande StorePic suivi d'un numéro d'image (entre 0 et 9).
Par exemple :

Code : Autre

```
:StorePic 0
```

Et pour revoir cette image ?

Code : Autre

```
:RecallPic 0
```

Quand vous créez une image, l'écran est stocké dans la variable PicX (avec X le numéro que vous avez indiqué). Si vous voulez effacer cette variable, faites DelVar PicX (PicX se trouve dans VARS > 4:Picture...).

Rafraîchir l'écran

Il y a quand même une limitation fâcheuse : quand on appelle une image, celle-ci se superpose aux pixels déjà présents sur l'écran !

On va mettre *ClrDraw* avant *RecallPic*, c'est ça ?

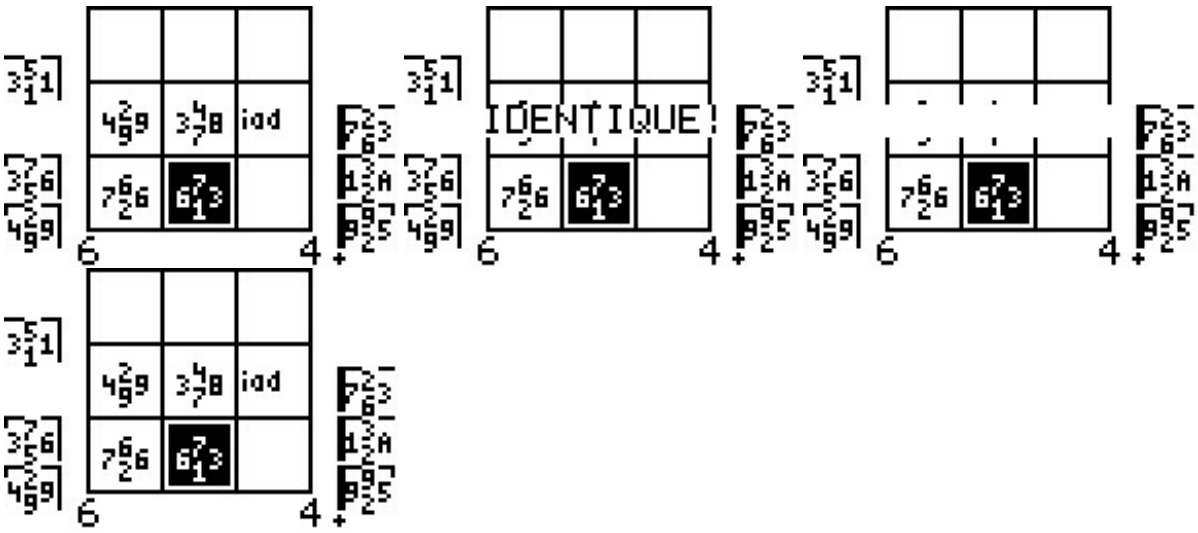
Parfois oui, mais parfois... non . C'est évident, si vous voulez rafraichir tout l'écran d'un coup vous allez mettre *ClrDraw*, dans ce cas vous allez avoir une sorte de petit flash : l'écran deviendra blanc pendant assez longtemps pour que cela se remarque. Si vous le pouvez, je vous conseille donc fortement de faire des effacements locaux.

Alors dans quel cas peut-on faire des effacements locaux ? Simplement quand l'image que l'on veut afficher ressemble à l'écran actuel. On efface juste la zone où il y a du changement grâce à un *Text(* puis on appelle l'image.

Un... *Text(* ?

Rappelez-vous aux chapitres précédents je vous ai dit que la seule méthode pour effacer localement était d'utiliser un texte avec des espaces (et de préférence un grand texte du genre *Text(-1,1,1," ")*).

Voulez-vous un exemple ? Je vais vous en faire un scindé en 4 étapes :



- On sauvegarde l'écran de départ (avec StorePic).
- On affiche le texte "IDENTIQUE!" en gros.
- On efface localement.
- Et on rappelle l'image sauvegardée (avec RecallPic).

L'effacement local est rapide comparé à un *CrlDraw*, c'est son atout majeur .

Dessiner des carrés pleins

Il n'existe pas de fonction à proprement parler pour remplir toute une zone de pixels noirs. Nous allons donc utiliser ce que vous avez probablement vu dans le tutoriel sur le ti-basic : **les boucles For**.

Zone rectangulaire

Si vous n'avez pas déjà oublié ce que l'on a fait précédemment, vous vous souvenez sûrement des *Line*. Je suis certain que vous voyez où je vais, non ? Eh bah nous allons mettre plusieurs lignes côte à côte !

Bon alors je fais ça :

Code : Autre

```
//mon repère mathématique
:0→Xmin
:94→Xmax
:-62→Ymin
:0→Ymax

//mon carré
:Line(1,-1,4,-1
:Line(1,-2,4,-2
:Line(1,-3,4,-3
:Line(1,-4,4,-4
```

Et j'aurais un beau carré de 4 par 4 pixels sur mon écran !

Non ! Enfin oui si vous voulez que ce soit vite exécuté. Mais ce code est long, surtout si on veut faire un GRAND rectangle. C'est là que la boucle *For(* intervient.

Code : Autre

```
:For(A,1,4
```

```
:Line(1,-A,4,-A
:End
```

Ce code va répéter 4 fois *Line(1,-A,4,-A)* avec **A** qui va prendre la valeur 1 au premier tour, 2 au second, 3 au troisième, 4 au quatrième. À chaque tour donc, **A** va s'incrémenter de 1 (on ajoute 1) jusqu'à atteindre la valeur 4.

Et si je veux que **A** s'incrémente de 2 ?

Il faut faire ça :

Code : Autre

```
:For(A,1,4,2
```

Voilà vous avez appris à mettre des *Line(* côte à côte, simple non ?

Mes exemples proposent d'aligner des lignes noires, mais vous pouvez faire des lignes blanches (si si rappelez-vous *Line(X1,Y1,X2,Y2,0)* avec le zéro à la fin).

Zone rectangulaire inversée

Attention, cette partie est plus compliquée. Prenons un exemple concret : j'ai une lettre en noir sur fond blanc, et je veux transformer ça en lettre blanche sur fond noir. La seule fonction qui existe pour inverser un pixel c'est.... *Pxl-Change* (et *Pt-Change*) !

Et là c'est embêtant, il va falloir d'abord construire une ligne de pixels, puis répéter cette ligne de pixels pour construire la zone. Double boucle For ! Bon créez un programme comme celui-ci vous allez voir c'est magique :

Code : Autre

```
:StoreGDB 0
:AxesOff
:GridOff
:FnOff
:PlotsOff
```

```
:For(A,1,4
:For(B,1,4
:Pxl-Change(A,B
:Pause
:End
:End
:Pause
:RecallGDB 0
:DelVar GDB0
:Disp
```

En exécutant ça vous allez voir qu'à chaque fois que vous appuyez sur ENTER (à cause du premier *Pause*) un nouveau point apparaît. En fait, arrivé à la première boucle For, A=1, puis à la deuxième boucle For, B prend successivement les valeurs 1, 2, 3, et 4 et à chaque fois Pxl-Change est tracé :
Pxl-Change(1,1)
Pxl-Change(1,2)
Pxl-Change(1,3)
Pxl-Change(1,4)

Quand B arrive à 4, on sort de la deuxième boucle For, et on retourne sur la première. À ce moment-là, A=2... vous avez compris la suite ?

C'est fini ! Je vous donne un programme qui ressemble au précédent mais qui montre la toute puissance de Pxl-Change :

Code : Autre

```
:StoreGDB 0
:AxesOff
:GridOff
:FnOff
:PlotsOff
:Text(1,2,"A
:For(A,1,7
:For(B,1,5
:Pxl-Change(A,B
:End
:End
:Pause
:RecallGDB 0
:DelVar GDB0
:Disp
```

Les boucles For sont plutôt longues à s'exécuter (surtout quand on doit tracer une zone point par point), c'est du basique après tout. Sachez qu'il est plus rapide (mais beaucoup plus lourd) de marquer tous les pixels manuellement.

C'est fini pour aujourd'hui ! J'espère que vous avez tout compris, je n'y reviendrai pas  .

C'est tout ce que j'avais à dire sur le graphique.

Partie 2 : Fonction mathématique et mouvement

Cette partie (qui n'est pas terminée), va couvrir des sujets moins utilisés en TI-Basic.

Fonctions mathématiques

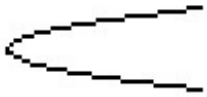
Fonctions pour les courbes

Dans le menu DRAW (2ND+PRGM), vous avez accès à plusieurs fonctions pour tracer des pixels à l'écran à l'aide d'un courbe. Là aussi le traçage est long.

DrawF fonction

Trace la *fonction*. La fonction peut-être une variable de fonction (Y1 par exemple) ou une expression sans guillemets (DrawF X^2 par exemple).

DrawInv fonction

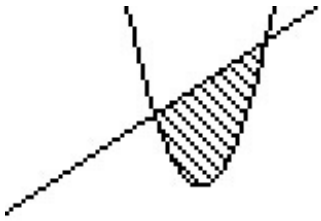


Trace la réciproque de la *fonction*. La fonction peut être une variable de fonction (Y1 par exemple) ou une expression sans guillemets (DrawInv X^2 par exemple).

Tangent(fonction,point

Trace la tangente de la *fonction* en un *point*.

Shade(fct inf,fct sup[,X gauche,X droite,motif,résolution]



Trace une ombre entre les courbes *fct inf* et *fct sup*.

Paramètres optionnels :

- *X gauche et droite* : permet de délimiter l'ombre à gauche et à droite ;
- *motif* : différent types d'ombres :
  - 1 : lignes verticales ;
  - 2 : lignes horizontales ;
  - 3 : lignes obliques (haut gauche vers bas droit) ;
  - 4 : lignes obliques (bas gauche vers haut droit).
- *résolution* : vous pouvez mettre une valeur de 1 à 8, plus la valeur est forte, plus les lignes sont écartées.

Tracer une courbe

Vous avez peut-être eu (ou vous êtes en train d'avoir) un professeur de mathématiques vous apprenant à vous servir de votre calculatrice, notamment à tracer la courbe d'une fonction. Je vais vous apprendre la même chose, mais en TI-Basic !

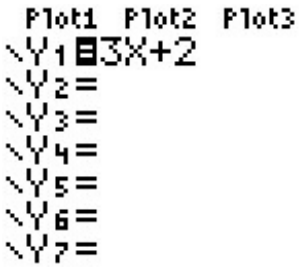
Créer un équation

Tout d'abord, il vous faut spécifier la fonction, par exemple :

Code : Autre

```
:"3X+2->Y1
```

Vous trouverez toutes les variables de fonctions (Y1, Y2, r1, XT1...) en appuyant sur **VARS**, onglet **Y-VARS**. Ici j'ai utilisé la première fonction Y1 et je lui ai affecté l'équation 3X+2. On a donc une simple affectation. Remarquez qu'après cette affectation, votre équation apparaît sur l'écran **Y=**.



Tracer ou ne pas tracer

Dans **VARS** -> **Y-VARS**, dans le menu "**4:On/off...**" vous avez les deux états de la courbe de votre équation : tracée (*FnOn*) et pas affichée (*FnOff*).

Pour afficher votre courbe, vous devez être dans le bon MODE :

- **FUNC** : fonctions (Y1...) ;
- **PAR** : fonctions paramétriques (X1T, Y1T...) ;
- **POL** : coordonnées polaires (r1...).

Allez pour cela dans le menu de la touche MODE.

Faites des tests par vous-mêmes, vous remarquerez que tracer une courbe durant l'exécution d'un programme est vraiment long. Utiliser des courbes dans un programme n'est donc pas adapté du tout. Je vais quand même vous en dire plus.

Les paramètres du graphe

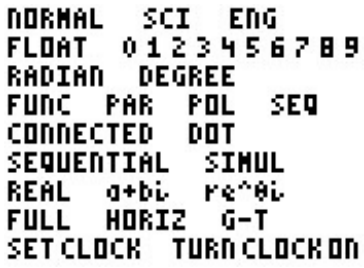
J'ai expliqué au premier chapitre comment manipuler la fenêtre (WINDOW), je ne vais pas y revenir. Regardons plutôt ce que nous offre le menu 2ND+ZOOM (FORMAT) :

- **Grid** : affiche ou non la grille qui permet de mieux visualiser le repère mathématique ;
- **Axes** : affiche ou non les axes du repère mathématique ;
- **Label** : affiche ou non les noms des axes (nécessite que les Axes soient affichés).

Pour les autres paramètres, que vous les mettiez sur un état ou sur un autre, cela ne changera rien durant l'exécution de votre programme.

D'autres paramètres

Dans le menu de la touche MODE, vous avez le choix entre plusieurs paramètres :



- style de la courbe :
  - CONNECTED : la courbe est tracée en tous ses points ;
  - DOT : la courbe est tracée en pointillées ;
- technique de traçage de la courbe :
  - SEQUENTIAL : les courbes sont tracées les unes après les autres (s'il y en a plusieurs) ;
  - SIMUL : les courbes sont toutes tracées en même temps ;
- paramètre de l'écran :
  - FULL : l'écran GRAPHE prend tout l'écran de la calculatrice ;
  - HORIZ : la moitié supérieure de l'écran est l'écran GRAPHE, la moitié inférieure de l'écran est l'écran HOME ;
  - G-T : à gauche l'écran GRAPHE, à droite, l'écran TABLE.

Voilà je crois que je vous en ai dit suffisamment pour que vous reveniez avec un programme tout beau tout mignon  .