Université Paris 8 - Vincennes à Saint-Denis

Programmation C/C++ et OpenGL

Farès Belhadj

Date de MAJ : 28 octobre 2020

email : mailto:amsi@up8.edu
github : https://github.com/noalien/GL4Dummies
web [GL420] : http://api8.fr/GL4D/

Table des matières

Liste des codes source

Objectifs du support

1. Une base minimale d'architecture des machines – §1 ;
2. Une base minimale des langages C/C++ (Visual Studio) – §2 ;
3. La représentation mémoire d'une image – §2 ;
4. Les premières primitives de dessin – §2 ;
5. L'application graphique (GL4D) – §**??** ;
6. Continuons avec les primitives de dessin ;
7. L'animation / modèle physique de base ;
8. L'interaction : clavier, souris, son, caméra (OpenCV / OpenNI) ;
9. Introduction à l'architecture OpenGL [KSS16] : les données et les programmes (shaders) ;
10. Les Filtres 2D / La Modélisation 3D ;
11. Démo ?

Chapitre 1

Survol de l'architecture des machines

Avez-vous besoin de connaître ? Points abordés en cours selon les connaissances du public :

- La représentation des données en machine (la base 2) ?
- L'adressage mémoire : le stockage des données en mémoire ?
- Qu'est-ce qu'un programme, comment il s'exécute ?
- Les types de données (ex. caractères, entiers et flottants) ?
  Voir par exemple : `https://fr.wikipedia.org/wiki/C_(langage)#Types`
  Ou encore : `https://en.wikibooks.org/wiki/C_Programming/stdint.h`
- L'UAL, le FPU, le GPU ?

Chapitre 2

Introduction rapide au C/C++ & Visual Studio

2.1 Le saviez-vous ? (rapide)

Quelques points pour lesquels il faudra en connaître un minimum :
— Compiler Vs Interpréter ;
— Le préprocesseur, le compilateur, l'éditeur de liens ;
— Une variable, une fonction (proto. & déf.), les objets (plus tard) ;
— Les branchements, les boucles, la récursivité ;
— Les opérateurs (voir Table 2.1).

Les Tables 2.1 et 2.1 donnent respectivement des informations sur la priorité des opérateurs en C/C++ et les types standards et tailles de données en mémoire.

Priorité	Opérateur	Description	Exemple
0	**( )**	appel de fonction, associativité	foo() ; a = (b + c) * d ;
	[ ]	indexation	int tab[3] ; tab[0] = tab[1] = tab[2] = 0 ;
	.	nommage d'un champ	obj.cdr = NULL ;
	− >	nommage indirect de champ	pt− >cdr = NULL ;
1	**!**	négation	$(!a)$ est vraie si a est fausse
	~	complément à 1	a & (~a) = 0
	-	opposé	
	++	incrémentation	i++ ; ++i ;
	- -	décrémentation	i- - ; - -i ;
	&	adresse	int i, *pt ; pt = &i ;
	*****	valeur indirecte	*pt = 0 ; /* donne i = 0 */
	(type_de_donnée)	force le type (cast)	int i = (int)1.5 ;
	sizeof	taille en octets	i = sizeof i ;
2	*****	Multiplication	
	/	Division	
	%	Modulo	
3	+	Addition	
	-	Soustraction	
4	<<	Décalage à gauche	
	>>	Décalage à droite	
5	<	Strictement inférieur	
	<=	Inférieur ou égal	
	>	Strictement supérieur	
	>=	Supérieur ou égal	
6	==	Egal	
	! =	Différent	
7	**&**	“et” binaire	
8	^	“ou” exclusif binaire	
9	|	“ou” binaire	
10	**&&**	“et” logique	
11	||	“ou” logique	
12	**? :**	conditionnelle	`c = (a < b) ? a : b;`
13	= *= /= %= += -= ^= &= <<= >>= |=	Affectations	
14	**,**	Séquence	

TABLE 2.1 – Table des priorités des opérateurs C/C++.

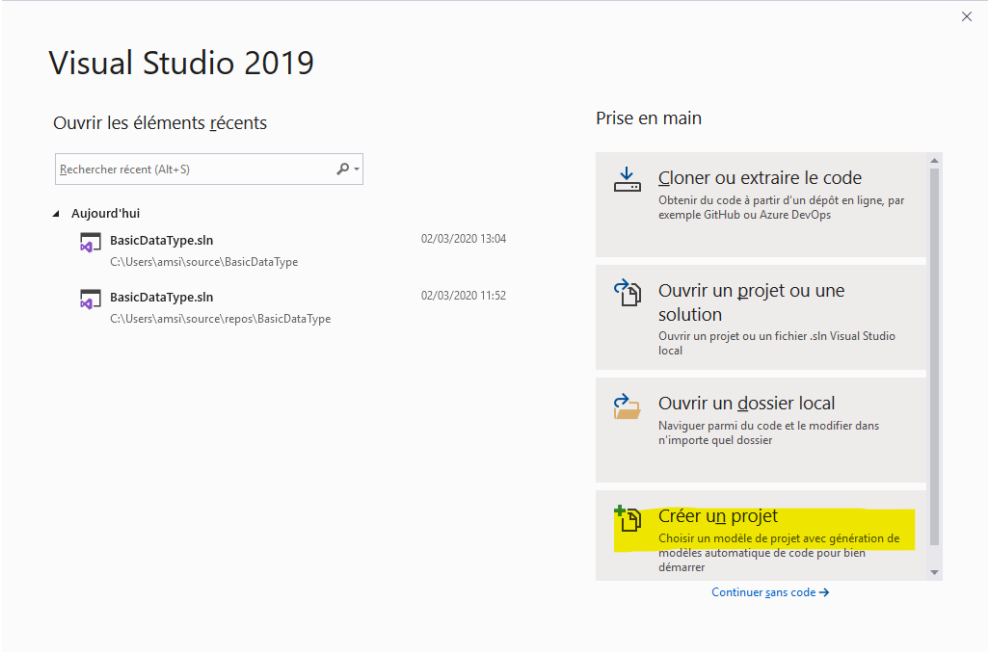
Type	**Occupation mémoire**	**Plage de valeurs**
char	1 octet	-128 à 127
unsigned char	1 octet	0 à 255
int	2 ou 4 octets	Selon l'architecture
unsigned int	2 ou 4 octets	Selon l'architecture
short	2 octets	-32768 à 32767
unsigned short	2 octets	0 à 65535
long	4 octets	-2147483648 à 2147483647
unsigned long	4 octets	0 à 4294967295
long long	8 octets	-2^{63} à $2^{63}-1$
unsigned long long	8 octets	0 à $2^{64}-1$
Type à virgule flottante		
float	4 octets	3.4×10^{-38} à 3.4×10^{38} (IEEE 754)
double	8 octets	1.7×10^{-308} à 1.7×10^{308} (IEEE 754)
long double	10 octets	3.4×10^{-4932} à 3.4×10^{4932} (IEEE 754)

TABLE 2.2 – Types de données standards (à l'époque).

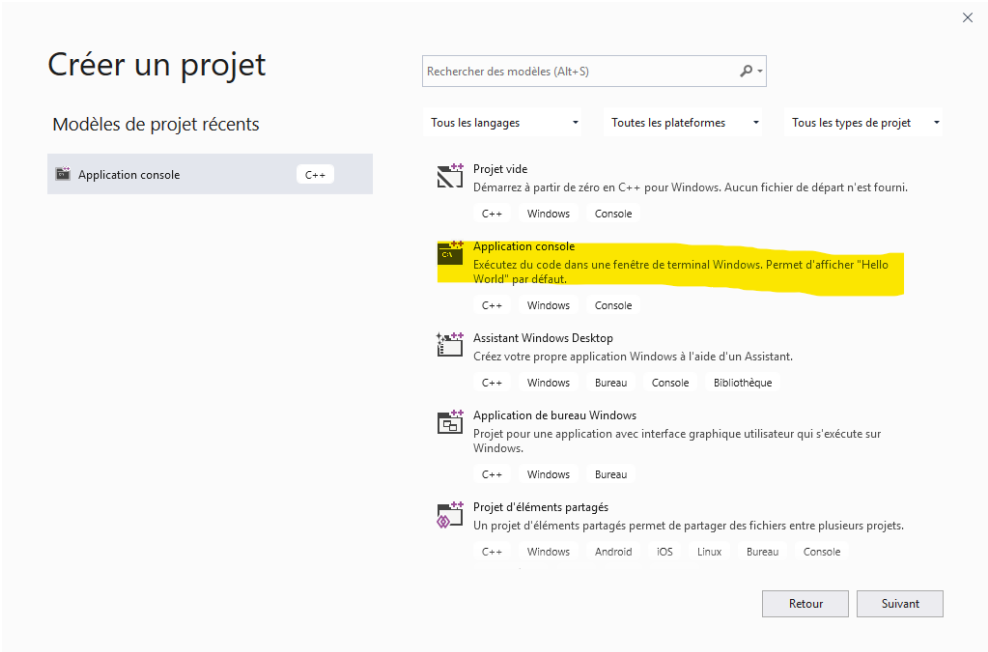
En pratique, les types de données évoluent et changent en fonction de l'architecture, nous verrons cela dans les exemples données dans les sections suivantes.

2.2 Créer une application en mode console sous Visual Studio

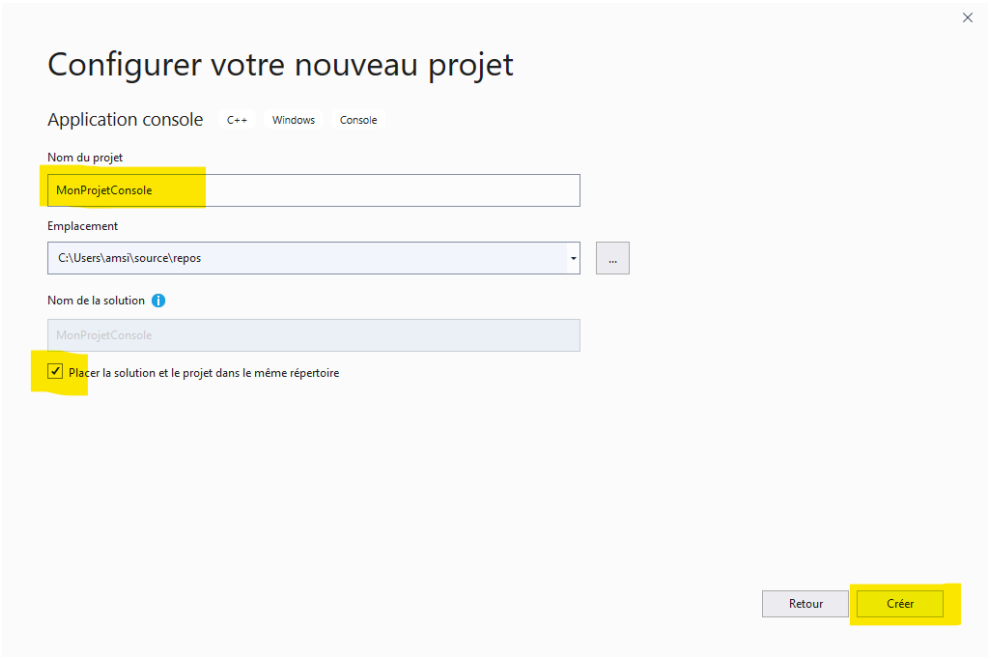
En ouvrant Visual Studio (ici Community 2019) l'outil vous propose une action à mener :



Sélectionnez “Créer un projet” puis là sélectionnez “Application console” :



Entrez le nom de votre projet, cochez la case “Placer la solution et le projet dans le même répertoire“ et cliquez sur créer :



2.3 Les exemples (à développer plus tard)

2.3.1 Récupération d'informations sur les types de données standards

```
#include <iostream>        // pour utiliser std::cout
#include <iomanip>         // pour utiliser std::setprecision
#define _USE_MATH_DEFINES  // pour utiliser les définitions dans math.h
#include <math.h>          // la bibliothèque de fonctions mathématiques
#include <stdio.h>         // pour utiliser printf/fprintf ... les puristes C++ n'apprécieront pas

int main(void) {
  char c = 'A';
  short int s = 1024;
  int i = 100000;
  long int l = 0x7FFFFFFF;
  long long int ll = 0x7FFFFFFFFFFFFFFF;
  float f = (float)M_PI;
  double d = M_PI;
  long double ld = M_PI;
  std::cout << "Hello Data Type avec std::cout !\n";
  std::cout << "c contient " << c << " (" << (int)c << "), sa taille est "
      << sizeof c << " octet(s) et son adresse est " << (void*)&c << "\n";
  std::cout << "s contient " << s << ", sa taille est "
      << sizeof s << " octet(s) et son adresse est " << (void*)&s << "\n";
  std::cout << "i contient " << i << ", sa taille est "
      << sizeof i << " octet(s) et son adresse est " << (void*)&i << "\n";
  std::cout << "l contient " << l << ", sa taille est "
      << sizeof l << " octet(s) et son adresse est " << (void*)&l << "\n";
  std::cout << "ll contient " << ll << ", sa taille est "
      << sizeof ll << " octet(s) et son adresse est " << (void*)&ll << "\n";
  std::cout << std::setprecision(20);
  std::cout << "f contient " << f << ", sa taille est "
      << sizeof f << " octet(s) et son adresse est " << (void*)&f << "\n";
  std::cout << "d contient " << d << ", sa taille est "
      << sizeof d << " octet(s) et son adresse est " << (void*)&d << "\n";
  std::cout << "ld contient " << ld << ", sa taille est "
      << sizeof ld << " octet(s) et son adresse est " << (void*)&ld << "\n";
  fprintf(stdout, "Hello Data Type avec fprintf !\n");
  fprintf(stdout, "c contient %c (%d), sa taille est %zu octet(s) et son adresse est %p\n", c, c, sizeof c,
      &c);
  fprintf(stdout, "s contient %hd, sa taille est %zu octet(s) et son adresse est %p\n", s, sizeof s, &s);
  fprintf(stdout, "i contient %d, sa taille est %zu octet(s) et son adresse est %p\n", i, sizeof l, &l);
  fprintf(stdout, "l contient %ld, sa taille est %zu octet(s) et son adresse est %p\n", l, sizeof l, &l);
  fprintf(stdout, "ll contient %lld, sa taille est %zu octet(s) et son adresse est %p\n", ll, sizeof ll,
      &ll);
  fprintf(stdout, "f contient %1.20f, sa taille est %zu octet(s) et son adresse est %p\n", f, sizeof f, &f);
  fprintf(stdout, "d contient %1.20f, sa taille est %zu octet(s) et son adresse est %p\n", d, sizeof d, &d);
  fprintf(stdout, "ld contient %1.20Lf, sa taille est %zu octet(s) et son adresse est %p\n", ld, sizeof ld,
      &ld);
  return EXIT_SUCCESS;
}
```

Code source 2.1 – L'exemple BasicDataType

L'exemple donné dans CODE SOURCE 2.1 illustre de deux manières différentes comment récupérer et afficher correctement le contenu, la taille et l'adresse mémoire de plusieurs variables, chacune correspond à un type de donnée standard en C/C++. D'une part, l'affichage se fait par le biais d'indirections (le symbole "`<<`") vers l'objet `cout` de la bibliothèque standard C++ `std` (inclure `iostream`), et d'autre part, il se fait par le biais de la fonction `printf` (ou `fprintf`) de la bibliothèque standard C (inclure `stdio.h`). La famille de fonctions `printf` est à nombre d'arguments variable, le type de la donnée affichée est indiqué dans la chaîne de caractères par le symbole de formatage "`%...`". La Table 2.3 donne quelques exemples de formats ainsi que le type attendu en face.

Format	**Type de donnée**	
%d	Entier	`short, int`
%i	Entier	`short, int`
%u	Entier non signé	`unsigned short, unsigned int`
%zu	taille	`size_t`
%o	Entier octal	`short, int`
%x, %X	Entier hexadécimal	`short, int`
%l, %ld, %li, %lu, %lo, %lx, %lX	Entier long	`long`
%lld, %lli, %llu, %llo, %llx, %llX	Entier long 64 bits	`long long`
%c	Caractère ASCII	`char, unsigned char`
%f, %F, %g, %G	Flottant	`float, double`
%e, %E	Flottant (exponentiel)	`float, double`
%Lf, %LF, %Lg, %LG, %Le, %LE	long double	`long double`
%s	Chaîne de caractères	`char *`
%p	Pointeur	`(type) *`

TABLE 2.3 – Quelques formats pris en charges par la famille de fonctions `printf`

Enfin, de manière générale, la Table 2.4 donne quelques séquences d'échappements (caractères spéciaux tels que "retour à la ligne") que nous pouvons retrouver dans des chaînes de caractères en C et C++.


```
Sous Windows 10 (x86)
Hello Data Type avec std::cout !
c contient A (65), sa taille est 1 octet(s) et son adresse est 0093F99B
s contient 1024, sa taille est 2 octet(s) et son adresse est 0093F98C
i contient 100000, sa taille est 4 octet(s) et son adresse est 0093F980
l contient 2147483647, sa taille est 4 octet(s) et son adresse est 0093F974
ll contient 9223372036854775807, sa taille est 8 octet(s) et son adresse est 0093F964
f contient 3.1415927410125732422, sa taille est 4 octet(s) et son adresse est 0093F958
d contient 3.141592653589793116, sa taille est 8 octet(s) et son adresse est 0093F948
ld contient 3.141592653589793116, sa taille est 8 octet(s) et son adresse est 0093F938
Hello Data Type avec fprintf !
c contient A (65), sa taille est 1 octet(s) et son adresse est 0093F99B
s contient 1024, sa taille est 2 octet(s) et son adresse est 0093F98C
i contient 100000, sa taille est 4 octet(s) et son adresse est 0093F974
l contient 2147483647, sa taille est 4 octet(s) et son adresse est 0093F974
ll contient 9223372036854775807, sa taille est 8 octet(s) et son adresse est 0093F964
f contient 3.1415927410125732419, sa taille est 4 octet(s) et son adresse est 0093F958
d contient 3.1415926535897931160, sa taille est 8 octet(s) et son adresse est 0093F948
ld contient 3.1415926535897931160, sa taille est 8 octet(s) et son adresse est 0093F938

Sous Windows 10 (x64)
Hello Data Type avec std::cout !
c contient A (65), sa taille est 1 octet(s) et son adresse est 000000587433FA04
s contient 1024, sa taille est 2 octet(s) et son adresse est 000000587433FA24
i contient 100000, sa taille est 4 octet(s) et son adresse est 000000587433FA44
l contient 2147483647, sa taille est 4 octet(s) et son adresse est 000000587433FA64
ll contient 9223372036854775807, sa taille est 8 octet(s) et son adresse est 000000587433FA88
f contient 3.1415927410125732422, sa taille est 4 octet(s) et son adresse est 000000587433FAA4
d contient 3.141592653589793116, sa taille est 8 octet(s) et son adresse est 000000587433FAC8
ld contient 3.141592653589793116, sa taille est 8 octet(s) et son adresse est 000000587433FAE8
Hello Data Type avec fprintf !
c contient A (65), sa taille est 1 octet(s) et son adresse est 000000587433FA04
s contient 1024, sa taille est 2 octet(s) et son adresse est 000000587433FA24
i contient 100000, sa taille est 4 octet(s) et son adresse est 000000587433FA64
l contient 2147483647, sa taille est 4 octet(s) et son adresse est 000000587433FA64
ll contient 9223372036854775807, sa taille est 8 octet(s) et son adresse est 000000587433FA88
f contient 3.1415927410125732419, sa taille est 4 octet(s) et son adresse est 000000587433FAA4
d contient 3.1415926535897931160, sa taille est 8 octet(s) et son adresse est 000000587433FAC8
ld contient 3.1415926535897931160, sa taille est 8 octet(s) et son adresse est 000000587433FAE8


Sous mac os X
Hello Data Type avec std::cout !
c contient A (65), sa taille est 1 octet(s) et son adresse est 0x7ffeef977847
s contient 1024, sa taille est 2 octet(s) et son adresse est 0x7ffeef977844
i contient 100000, sa taille est 4 octet(s) et son adresse est 0x7ffeef977838
l contient 2147483647, sa taille est 8 octet(s) et son adresse est 0x7ffeef977828
ll contient 9223372036854775807, sa taille est 8 octet(s) et son adresse est 0x7ffeef977830
f contient 3.1415927410125732422, sa taille est 4 octet(s) et son adresse est 0x7ffeef97783c
d contient 3.141592653589793116, sa taille est 8 octet(s) et son adresse est 0x7ffeef977820
ld contient 3.141592653589793116, sa taille est 16 octet(s) et son adresse est 0x7ffeef977800
Hello Data Type avec fprintf !
c contient A (65), sa taille est 1 octet(s) et son adresse est 0x7ffeef977847
s contient 1024, sa taille est 2 octet(s) et son adresse est 0x7ffeef977844
i contient 100000, sa taille est 8 octet(s) et son adresse est 0x7ffeef977828
l contient 2147483647, sa taille est 8 octet(s) et son adresse est 0x7ffeef977828
ll contient 9223372036854775807, sa taille est 8 octet(s) et son adresse est 0x7ffeef977830
f contient 3.1415927410125732419, sa taille est 4 octet(s) et son adresse est 0x7ffeef97783c
d contient 3.1415926535897931160, sa taille est 8 octet(s) et son adresse est 0x7ffeef977820
ld contient 3.1415926535897931160, sa taille est 16 octet(s) et son adresse est 0x7ffeef977800
```

Code source 2.2 – Résultats d'exécutions de BasicDataType sur différents OS / Architectures

Ce code et ses fichiers de configuration sont disponibles en téléchargement à l'adresse :

`https://expreg.org/amsi/C/PCCPPGL2021S1/exemples/00_basiques/BasicDataType.zip`

Après exécution, le programme affiche les résultats donnés dans le Code source 2.2. Ici trois configurations différentes ont été testée : VS Windows 32 bits, VS Windows 64 bits et Mac OS X en 64 bits. Nous remarquons que les tailles "standards" ne le sont pas tant que ça, que des différences existent entre la norme et les différentes instances de chaque architecture.

Séquence d'échappement	**Action**
\n	Nouvelle ligne (`new line`)
\t	Tabulation horizontale
\v	Tabulation verticale
\b	Retour d'un caractère arrière (`backspace`)
\r	Retour chariot (`carriage return`)
\f	Saut de page (`form feed`)
\a	Signal sonore (`alarm`)
\'	Affiche une apostrophe
\"	Affiche un guillemet
\\	Affiche un `Backslash`
\ddd	Affiche les codes ASCII en octale
\xddd	Affiche les codes ASCII en hexadécimale

TABLE 2.4 – Séquences d'échappement dans les chaînes de caractères

2.3.2 Le contenu, le contenant, les pointeurs et les références

```
#include <iostream>
static void swap0(int a, int b) {
    int c = a;
    a = b;
    b = c;
}
static void swap1(int *p, int *q) {
    int c = *p;
    *p = *q;
    *q = c;
}
static void swap2(int& x, int& y) {
  int c = x;
  x = y;
  y = c;
}
static void testerUnSwap(int whichOne) {
    int a = 2, b = 4;
    std::cout << "Avant␣swap" << whichOne << ",␣a␣=␣" << a << ",␣b␣=␣" << b << std::endl;
    switch (whichOne) {
    case 0:
        swap0(a, b);
        break;
    case 1:
        swap1(&a, &b);
        break;
    case 2:
        swap2(a, b);
        break;
    default:
        std::cerr << "Parametre␣incorrect" << std::endl;
        return;
    }
    std::cout << "Apres␣swap" << whichOne << ",␣a␣=␣" << a << ",␣b␣=␣" << b << std::endl;
}
int main(void) {
    testerUnSwap(0);
    testerUnSwap(1);
    testerUnSwap(2);
    return EXIT_SUCCESS;
}
```

Code source 2.3 – Echange de contenu de variables à l'aide du programme Swap

Dans l'exemple donné dans CODE SOURCE 2.3, nous proposons trois fonctions différentes (`swap0, swap1` et `swap2`) qui tentent d'échanger le contenu de deux

variables `a` et `b` locales (ou appratenant) à `testerUnSwap`. Après exécution du programme nous constatons que `swap0` a échoué à effectuer l'échange entre les deux variables (les contenants) de `testerUnSwap`, et que `swap1` et `swap2` ont réussi. Pour la première, c'est les valeurs (le contenu) qui sont transmises à la fonction `swap0` et seuls les variables locales à `swap0` sont échangées ; aucun intérêt car ces variables n'ont plus de validité après la fin de cette fonction. Pour la seconde et la troisième, nous avons respectivement utilisé les "pointeurs" et les "références" vers les deux variables `a` et `b` locales à `testerUnSwap`. Un pointeur est l'adresse en mémoire (numéro de l'octet) à partir de laquelle on retrouve la variable pointée. Une référence est une forme cachée et simplifiée d'utilisation des pointeurs (pas besoin de déréférencement à l'aide du symbole `*`).

→ **Notions à connaître :** adressage mémoire, équiquetage ou labellisation, *l-value* et *r-value*, déclaration et utilisation des pointeurs, déclaration et utilisation des références.

Ce code et ses fichiers de configuration sont disponibles en téléchargement à l'adresse :
`https://expreg.org/amsi/C/PCCPPGL2021S1/exemples/00_basiques/Swap.zip`

Exercice

Pour `a` et `b`, 2 variables du même type, on peut les échanger sans utiliser une 3ème variable :

```
a = a ^ b;
b = a ^ b;
a = a ^ b;
```

Testez cette variante en modifiant `swap1` et `swap2`.
Pour comprendre le fonctionnement de l'opérateur `^` (ou XOR binaire), essayez l'échange avec deux valeurs telles que :
$a = (7)_{10} = (0111)_2$ et
$b = (12)_{10} = (1100)_2$.

2.3.3 Utilisation d'un tableau de taille statique

```c
#include <iostream>
static int myRand(int n) {
    return (int)(n * (rand() / (RAND_MAX + 1.0)));
}
static void init(int* ptr, int n) {
    int i;
    for (i = 0; i < n; ++i) {
        ptr[i] = myRand(n);
    }
}
static void print(int* ptr, int n) {
    for (int i = 0; i < n; ++i)
        std::cout << ptr[i] << "␣";
    std::cout << std::endl;
}
int main(void) {
    int t[32];
    /* Remarque : si on écrit int *p = t; on peut utiliser p
       'a la place de t, sauf tout ce qui concerne sizeof */
    int* p = t; /* imprimer les sizeof des deux pour voir la différence */
    init(t, sizeof t / sizeof * t); // ici t
    print(p, sizeof t / sizeof * t); // ici p
    return EXIT_SUCCESS;
}
```

Code source 2.4 – Initialisation et affichage d'un tableau statique

Le Code source 2.4 montre un exemple de déclaration d'un tableau d'entiers de 32 éléments, son initialisation aléatoire (voir `myRand` et `rand`) via une fonction `init` et son affichage à l'aide de la fonction `print`.

→ **Notions à connaître :** déclaration, indexation, parcours indexé, taille (via `sizeof`) d'un tableau statique, adresse et pointeur vers un tableau statique (différence de taille et de comportement), parcours à l'aide d'un pointeur (non-indexé).

Ce code et ses fichiers de configuration sont disponibles en téléchargement à l'adresse :
`https://expreg.org/amsi/C/PCCPPGL2021S1/exemples/00_basiques/StaticArrayOfInt.zip`

Exercices

1. Écrire la fonction `int maxv(int* ptr, int n)` qui retourne la valeur maximale du tableau pointé par `ptr`.
2. Écrire la fonction `int minv(int* ptr, int n)` qui retourne la valeur minimale du tableau pointé par `ptr`.
3. Écrire la fonction `int maxi(int* ptr, int n)` qui retourne l'indice de la valeur maximale du tableau pointé par `ptr`.
4. Écrire la fonction `int minv(int* ptr, int n)` qui retourne l'indice de la valeur minimale du tableau pointé par `ptr`.
5. Écrire la fonction `void inv(int* ptr, int n)` qui inverse l'ordre des éléments du tableau pointé par `ptr`.

2.3.4 Utilisation d'un tableau bidimensionnel de taille statique

```
#include <iostream>
/* écrire 21 en dur sur l'une des dimensions de l'image est problématique,
 * cette écriture est rigide et ne permet pas de gérer des images de
 * dimensions variables.
 */
static void dessinerCroix(char ptr[][21], int w, int h) {
  int x, y;
  for (y = 0; y < h; ++y)
    for (x = 0; x < w; ++x) {
      if (x == y || (w - 1) - x == y)
  ptr[y][x] = 'X';
      else
  ptr[y][x] = '␣';
    }
}
/* solution */
static void dessinerCarre(char * ptr, int w, int h) {
  int x, y;
  for (y = 0; y < h; ++y)
    for (x = 0; x < w; ++x) {
      if (x == 0 || x == (w - 1) || y == 0 || y == (h - 1))
  ptr[y * w + x] = 'X';
      else
  ptr[y * w + x] = '␣';
    }
}
static void afficher(char ptr[][21], int w, int h) {
  int x, y;
  for (y = 0; y < h; ++y) {
    for (x = 0; x < w; ++x) {
      std::cout << ptr[y][x];
    }
    std::cout << std::endl;
  }
}
int main(void) {
  char image[21][21];
  dessinerCroix(image, 21, 21);
  afficher(image, 21, 21);
  dessinerCarre((char *)image, 21, 21);
  afficher(image, 21, 21);
  return EXIT_SUCCESS;
}
```

Code source 2.5 – Dessiner (une croix puis un carré) sur une "image" (tableau bidimensionnel) puis afficher l'image (print)

Le CODE SOURCE 2.5 montre un exemple d'utilisation d'un tableau d'octets (donc de type `char`) bidimensionnel. Ce tableau peut être considéré comme une image de 21×21. La fonction `croix` utilise la double indexation (y puis x) pour dessiner une croix, cette double indexation est problématique car nous oblige à spécifier en dur l'une des deux dimentions du tableau. La fonction `carre` utilise une simple indexation via une formule de converion (la seule correspondant à la réalité) pour dessiner un carré.

→ **Notions à connaître :** déclaration, double indexation, simple indexation, conversion $1D \rightarrow 2D$, conversion $2D \rightarrow 1D$.

Conversions $1D \leftrightarrow 2D$

Pour une image à une composante 8 bits `img` et aux dimensions constantes $W \times H$, nous pouvons la déclarer à l'aide de l'instruction :


```
char img[H][W];
```

Ainsi, nous pouvons passer d'une double indexation (x, y) à une simple indexation i, et inversement, à l'aide des conversions suivantes :

$$(x, y) \rightarrow i = y * W + x$$

$$i \rightarrow \begin{cases} x = i\%W \\ y = \lfloor i/W \rfloor \end{cases}$$

Ce code et ses fichiers de configuration sont disponibles en téléchargement à l'adresse :
`https://expreg.org/amsi/C/PCCPPGL2021S1/exemples/00_basiques/Static2DArray.zip`

Exercices

1. Modifier la fonction afficher pour qu'elle ne soit plus "rigide" et faire que le parcours se fasse en une simple boucle de 0 à $n-1$ où $n = w \times h$. Attention à ne pas oublier le saut de ligne à fin de chacune des lignes de l'image.
2. Écrire une fonction permettant de dessiner un damier en alternant deux symboles (exemple `#` et `0`).

2.3.5 Commençons avec des primitives de dessin

Le Code source 2.6 montre un exemple d'utilisation de `drawLine`, un embryon de fonction de dessin de droites discrètes [1] dans un tableau d'octets non signés (*i.e.* `unsigned char`). Ce dernier est sauvegardé comme une image de dimensions $W \times H$ dans un fichier BMP à l'aide la fonction `imageSaveBMP` [2].

1. `http://fr.wikipedia.org/wiki/G%C3%A9om%C3%A9trie_discr%C3%A8te`
2. Cette fonction est définie dans le fichier `images_io.c`, son utilisation est possible dans le présent fichier à l'aide du prototype déclaré dans `images_io.h`.


```c
#include <iostream>
#include <string.h> /* pour memset */
#include "images_io.h"

#define W 80
#define H 60

static void drawLine(unsigned char * image, int w, int x0, int y0, int x1, int y1, unsigned char color) {
  float y = 0.0f, pente;
  int u = x1 - x0, v = y1 - y0;
  pente = v / (float)u;
  for (int x = 0; x <= u; ++x) {
    image[(y0 + ((int)y)) * w + x + x0] = color;
    y += pente;
  }
}
int main(void) {
  /* on part sur une image 8 bits / 256 niveaux de gris */
  unsigned char image[H * W];
  /* on met tous les pixels 'a zéro */
  memset(image, 0, sizeof image);
  /* on dessine une droite de (5, 10) 'a (70, 50)
   * la couleur utilisée est le blanc (255)
   */
  drawLine(image, W, 5, 10, 70, 50, 255);
  imageSaveBMP("resu.bmp", image, W, H, 1, 8);
  return EXIT_SUCCESS;
}
```

Code source 2.6 – Dessiner un segment de droite

→ **Notions à connaître :** `memset`, la pente d'une droite, la discrétisation, un octant (huitième de plan).

Ce code et ses fichiers de configuration sont disponibles en téléchargement à l'adresse :

`https://expreg.org/amsi/C/PCCPPGL2021S1/exemples/00_basiques/DrawLineInBMP.zip`

Exercices

1. Dessiner le segment (75, 10, 0, 50, 255), cela ne fonctionne pas. Corriger `drawLine`.
2. Dessiner le segment (10, 10, 20, 50, 255), cela ne fonctionne pas. Corriger `drawLine`.
3. Dessiner le segment (-10, -10, 120, 150, 255), cela ne fonctionne pas. Corriger `drawLine`.
4. Dessiner (si ça marche) toutes les positions d'une aiguille trotteuse en variant les intensités de gris. Vous pouvez utiliser les fonction `cos` et `sin` (inclure `math.h`) tels que :

   ```c
   for (float angle = 0.0f, rayon = 20.0f; angle < 2.0f * M_PI; angle += 0.5f)
    drawLine(image, W, W/2, H/2, W/2 + rayon * cos(angle), W/2 + rayon * sin(angle), rand()&0xFF);
   ```

   La Figure 2.1 donne un exemple de ce qui est attendu.
5. Si l'équation d'un cercle est donnée par la formule :
   r^2 `= (x - x0) * (x - x0) + (y - y0) * (y - y0);`
   Écrire `drawCircle`.

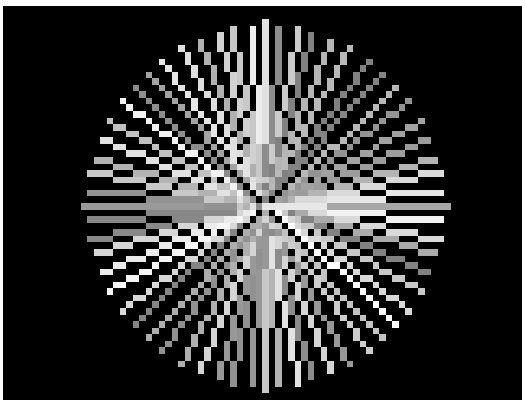


FIGURE 2.1 – Exemple de résultat attendu concernant la résolution de l'exercice 4 basé sur le code `DrawLineInBMP`

Chapitre 3

Aide sur certains exercices donnés

3.1 Aide sur l'exercice 4 de DrawLineInBMP

Cette aide porte sur la capacité à dessiner une multitude de droites à l'aide de votre propre fonction de dessin de droites discrètes. L'ensemble est sauvegardé dans un fichier BMP ;

3.1.1 Dessinons plein de droites dans un BMP

L'idée ici est de vous proposer une aide pas-à-pas qui vous amène jusqu'au résultat illustré par la Figure 2.1, soit la réponse complète à la question 4 de la série d'exerices donnés en section 2.3.5.

Comprendre l'embryon de droite discrète `drawLine`

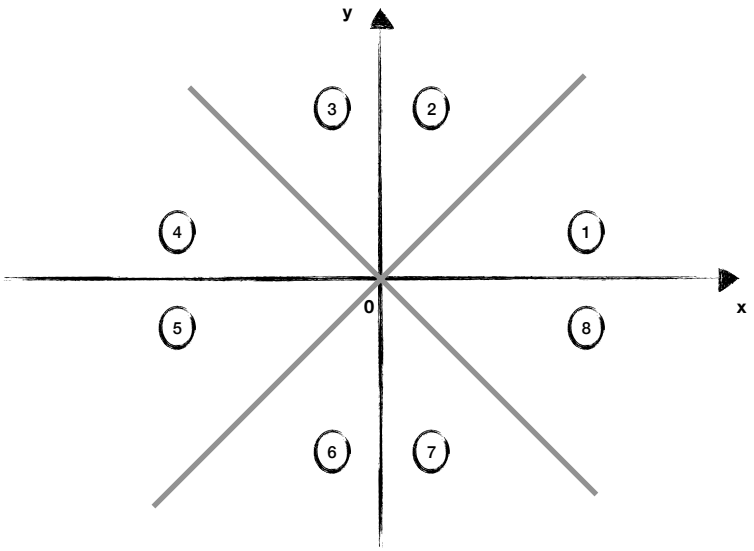
Notre base de travail [1] pour cet exercice est téléchargeable à l'adresse :
`https://expreg.org/amsi/C/PCCPPGL2021S1/exemples/00_basiques/DrawLineInBMP.zip`

Je vous ai donné une version simple et embryonnaire d'une fonction de dessin de droites discrètes qui, pour l'instant, ne sait correctement dessiner que des segments bien définis à l'interieur de l'image (*i.e.* entre ses bornes x de 0 à `w - 1`, et y de 0 à `h - 1`, où `w` et `h` sont respectivement la largeur et la hauteur de l'image) et appartenant au premier octant.

1. Ce travail est lié à la section 2.3.5.

```
static void drawLine(unsigned char * image, int w, int x0, int y0, int x1, int y1, unsigned char color) {
  float y = 0.0f, pente;
  int u = x1 - x0, v = y1 - y0;
  pente = v / (float)u;
  for (int x = 0; x <= u; ++x) {
    image[(y0 + ((int)y)) * w + x + x0] = color;
    y += pente;
  }
}
```

Un octant est un huitième de plan, les huit octants sont illustrés ci-après :



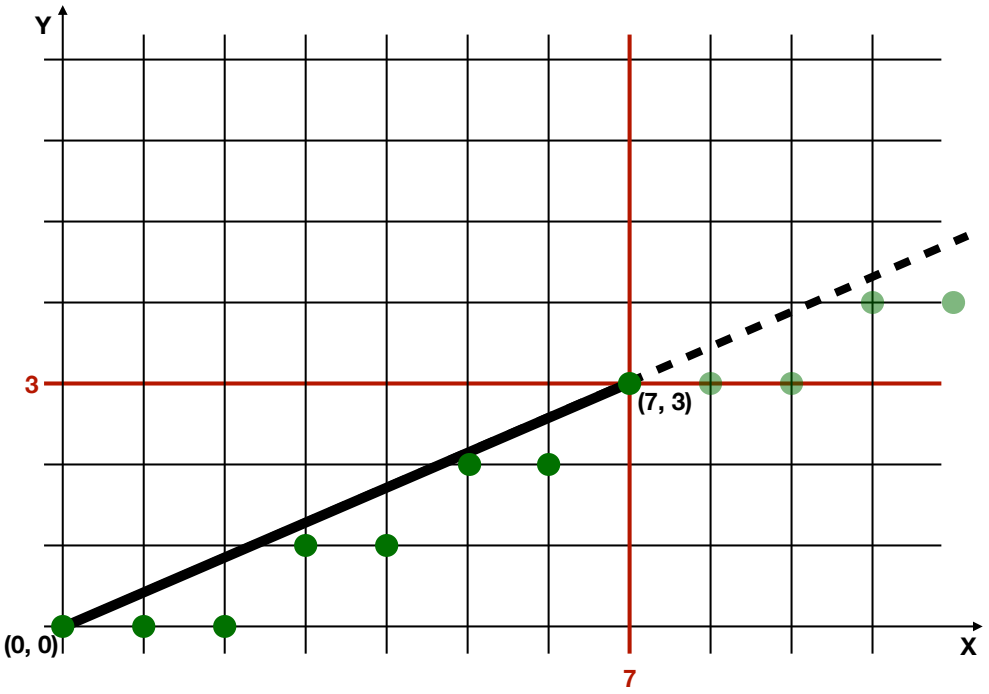
Voici les propriétés de chaque octant si on imagine une droite qui “progresse” dedans, la pente est le rapport entre la progression en y et la progression en x :

1. x augmente, y augmente mais x au moins plus vite que y, la pente est dans l'intervalle $[0, 1]$;
2. x augmente, y augmente mais y au moins plus vite que x, la pente est dans l'intervalle $[1, +\infty]$;
3. x diminue, y augmente mais y au moins plus vite que x, la pente est dans l'intervalle $[-\infty, -1]$;
4. x diminue, y augmente mais x au moins plus vite que y, la pente est dans l'intervalle $[-1, 0]$;
5. x diminue, y diminue mais x au moins plus vite que y, la pente est dans l'intervalle $[0, 1]$ (comme pour 1) ;
6. x diminue, y diminue mais y au moins plus vite que x, la pente est dans l'intervalle $[1, +\infty]$ (comme pour 2) ;
7. x augmente, y diminue mais y au moins plus vite que x, la pente est dans l'intervalle $[-\infty, -1]$ (comme pour 3) ;

8. x augmente, y diminue mais x au moins plus vite que y, la pente est dans l'intervalle $[-1, 0]$ (comme pour 4) ;

Pour comprendre le processus de discrétisation, regardons de plus près deux exemples de segments de droites discrètes dessinées sur une grille [2] de 8×8 pixels.

Le premier segment de droite appartient au premier octant et part du point à coordonnées entières $(0, 0)$ vers le point à coordonnées entières $(7, 3)$. Elle est illustrée par l'image ci-après :



Les disques verts opaques indiquent les points discrets (donc les pixels) qui seront "colorés" (ou "allumés") pour représenter ce segment de droite sur un écran à pixels [3]. Les disques verts translucides indiques la suite de points

2. Une image est une grille où les pixels sont représentés par les intersections des lignes horizontales et verticales de la grille.

3. Par opposition aux écrans vectoriels où la problématique de discrétisation n'a pas lieu d'être. Pour plus d'informations, je vous invite à consulter :
   — `https://en.wikipedia.org/wiki/Vector_monitor`
   — `https://en.wikipedia.org/wiki/Vectrex`

discrets si on continuait dans la même direction.

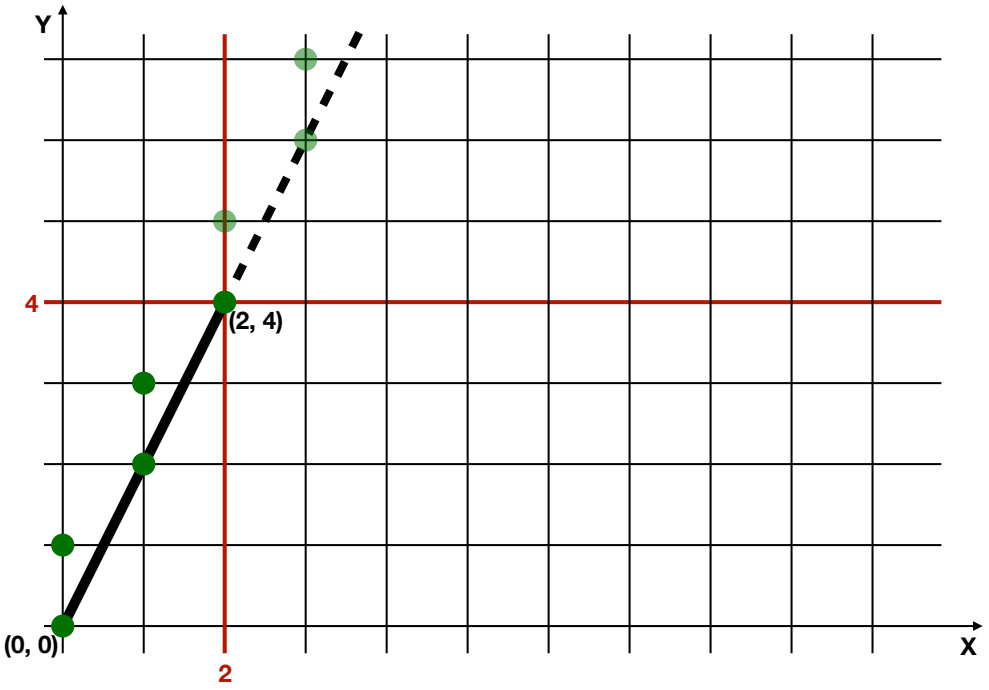
On remarque que dans ce cas "$7 > 3$" (différentiel en x égal à 7 et différentiel en y égal à 3), où nous sommes dans le premier octant, que les points discrets avancent toujours vers la droite. Parfois, ils restent à la même hauteur réalisant un déplacement **horizontal**, et parfois, il leur arrive de monter d'un cran réalisant un déplacement **oblique**. On peut aussi souligner que nous avons fait le choix d'allumer le pixel dont la position en y est atteinte ou légèrement dépassée, ce qui correspond à prendre que la partie entière d'une valeur y (flottante) qui a été calculée.

En résumé, on peut donc dire que dans cette situation (premier octant) les x augmentent de 1 tout le temps (donc boucle `for` sur les x avec incrément) et que y augmente d'une quantité inférieure à 1, qui en s'accumulant produit un saut (*i.e.* un $+1$) quand sa partie décimale produit une retenue (de 1) qui se propage sur les unités de la partie entière.

En conclusion, ici la pente est de $\frac{3}{7}$, elle est inférieure (ou égale) à 1, ceci implique que dans ce cas x est entier et augmente toujours de 1 et y est flottant et augmente d'une valeur inférieure à 1. Selon l'équation d'une droite ce différentiel est donné par la pente [4], donc $\frac{3}{7}$. Enfin, avant chaque variation de x et de y nous allumons le pixel se trouvant à x et la partie entière de y, soit $(x, \lfloor y \rfloor)$ ou, en C/C++, `(x, (int)y)`.

4. Pour une droite passant en $(0, 0)$ son équation peut être simplifiée à $y = p \times x$ où p est la pente.

Le second segment de droite appartient au deuxième octant et part du point à coordonnées entières $(0, 0)$ vers le point à coordonnées entières $(2, 4)$. Cette droite est illustrée par l'image ci-après :



On remarque que dans ce cas "$2 < 4$" (différentiel en x égal à 2 et différentiel en y égal à 4), où nous sommes dans le deuxième octant, que les points discrets avancent toujours vers le haut. Parfois, ils restent à la même abscisse réalisant un déplacement **vertical**, et parfois, il leur arrive de bouger d'un cran vers la droite réalisant un déplacement **oblique**. On peut aussi souligner que nous avons fait le choix d'allumer le pixel dont la position en x est atteinte ou légèrement dépassée, ce qui correspond à prendre que la partie entière d'une valeur x (flottante) qui a été calculée.

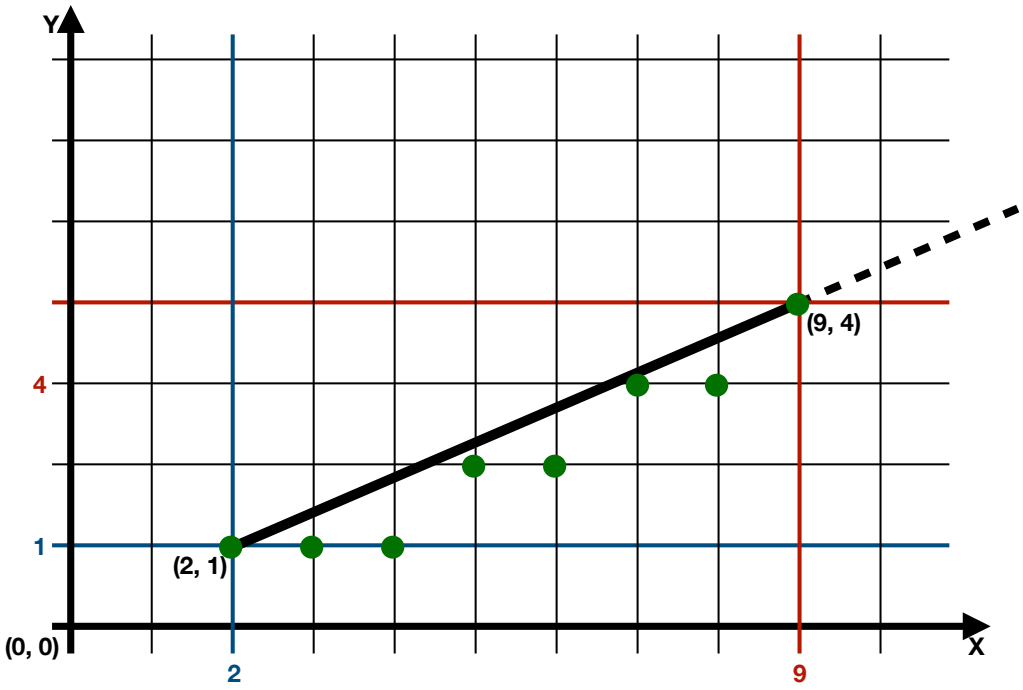
En résumé, on peut donc dire que dans cette situation (premier octant) les x augmentent de 1 tout le temps (donc boucle `for` sur les x avec incrément) et que y augmente d'une quantité inférieure à 1, qui en s'accumulant produit un saut (*i.e.* un $+1$) quand sa partie décimale produit une retenue (de 1) qui se propage sur les unités de la partie entière.

En conclusion, ici la pente est de $\frac{4}{2} = 2$, elle est donc supérieure (ou égale) à 1, soit son inverse (qui est valable pour x en fonction de y) $\frac{2}{4} = 0.5$ est

inférieur (ou égal) à 1, ceci implique que dans ce cas y est entier et augmente toujours de 1 et x est flottant et augmente d'une valeur inférieure à 1. Selon l'équation d'une droite ce différentiel est donné par l'inverse de la pente[5], donc $\frac{2}{4}$. Enfin, avant chaque variation de x et de y nous allumons le pixel se trouvant à la partie entière de x et y, soit $(\lfloor x \rfloor, y)$ ou, en C/C++, `((int)x, y)`.

Revenons donc à notre fonction `drawLine`, pour qui on sait maintenant pour quelle raison elle ne fonctionne que dans le premier octant. Il nous reste deux choses à régler, la première est que notre segment ne démarre pas en $(0, 0)$ pour aller vers une position que nous appellerons (u, v) mais démarre en (x_0, y_0) pour aller vers (x_1, y_1), la seconde, sera comment calculer la pente du segment $(x_0, y_0) \rightarrow (x_1, y_1)$.

Imaginons qu'on souhaite dessiner le segment $(2, 1) \rightarrow (9, 4)$ sur la grille précédente tel que :



On remarque que ce segment présente exactement le même pattern que le segment $(0, 0) \rightarrow (7, 3)$. En pratique les deux sont les mêmes à une trans-

5. Pour une droite passant en $(0, 0)$ son équation peut être simplifiée à $y = p \times x$ où p est la pente, mais on peut aussi écrire $x = \frac{1}{p} \times y$.

lation près. En ajoutant respectivement 2 et 1 aux abscisses et ordonnées du segment $(0,0) \to (7,3)$ on obtient le segment $(2,1) \to (9,4)$. Donc, en pratique on soustrait (x_0, y_0) aux coordonnées initiales pour revenir à un segment démarrant en $(0,0)$, on fait l'ensemble des calculs **mais** juste au moment où on dessine on remet ce que l'on a soustrait, c'est à dire (x_0, y_0). **En résumé**, pour dessiner le segment $(x_0, y_0) \to (x_1, y_1)$, on fait comme si on dessinait le segment $(x_0 - x_0, y_0 - y_0) \to (x_1 - x_0, y_1 - y_0)$, qui donne $(0,0) \to (x_1 - x_0, y_1 - y_0)$, puis on ajoute (x_0, y_0) aux (x, y) calculés afin d'allumer le point (mettre `color = 255`, valeur maximale de luminance, dans le tableau). Enfin, en posant $u = x_1 - x_0$ et $v = y_1 - y_0$, pour le segment $(0,0) \to (u, v)$ on sait que la pente est donnée par $\frac{v}{u}$. On fera attention à ce que $u \neq 0$, ce qui ne devrait pas arriver dans le premier octant (car $u \geq v$) sauf si $v = 0$ donnant le cas du segment particulier $(x_0, y_0) \to (x_0, y_0)$, c'est à dire un point (donc pas grave).

En se rappelant que pour accéder à une case (x, y) il faut utiliser $y \times w + x$, on arrive au code donnée plus haut. Ouf !

Première extension : Maintenant que le principe général acquis, il est facile d'imaginer comment étendre la fonction `drawLine` pour dessiner les segments du premier, quatrième, cinquième et huitième octant. Pour cela, il suffit de prendre en compte si une variation pour x est un incrément ou en décrément, ainsi nous passons du premier octant au quatrième. En considérant maintenant la variation du y, positive ou négative, nous étendons respectivement notre fonction au huitième et cinquième octants.

Je propose d'appliquer cette modification au code :

```
int u = x1 - x0, v = y1 - y0;
float y = 0.0f, pente = v / (float)abs(u);
int pas = u < 0 ? -1 : 1;
for (int x = 0; x != u; x += pas) {
  image[(y0 + ((int)y)) * w + x + x0] = color;
  y += pente;
}
```

Deuxième extension : Nous n'avons toujours pas de possibilité de dessiner un segment appartenant au deuxième octant. On remarquera que la différence entre premier et deuxième octant, c'est la même qu'entre quatrième et troisième, cinquième et sixième, huitième et septième. Donc, en ajoutant cette extension au deuxième octan en plus de la première extension, nous aurons automatiquement la possibilité de dessiner aussi sur le troisième, sixième et septième octants (toujours à l'aide des signes pour incrément ou décrément). **La différence** entre le premier et le deuxième octant est uniquement liée à

une permutation en x et y, ce qui fait qu'au second octant on devrait avoir à peu près le même code mais juste en permutant tout ce qui est abscisse avec tout ce qui est ordonnée (attention à ne pas permuter x_0 et y_0 lorsque vous les utiliserez pour allumer le pixel). Aussi, pour savoir si nous sommes sur un cas du deuxième octant il suffit de tester si $v > u$.

Voici les grandes lignes d'ajouts / restructuration que je propose :

```
  int u = x1 - x0, v = y1 - y0;
  if(abs(v) > abs(u)) {
    /* 2eme octant, les x c'est des y et inversement !!
       mais aussi u et v !
       SAUF quand on ecrit dans le pixel, la chacun reprend sa place
     */
  } else {
    /* 1er octant rien ne change pour lui
       donc remettre le meme bloc juste sans la
       ligne "int u = x1 - x0, v = y1 - y0;"
     */
  }
}
```

Ne pas dépasser les bornes : Eh oui ! Que se passe-t-il si nous donnons des coordonnées négatives ou des coordonnées allant plus loin que le $w-1$ en x et le $h-1$ en y ? Dans le meilleur des cas, un résultat bizarre, dans le pire un plantage du programme suite à une erreur de segment mémoire (violation d'accès) [6]. Il faut donc tester toute coordonnée – si elle est dans les bonnes bornes – avant de l'utiliser pour dessiner un point.

Les segments complètement hors image : Dernier cas à gérer, il y bien des segments qui peuvent être définis de manière à ne jamais intersecter la zone de l'image. Ceux là doivent simplement être ignorés en court-circuitant la fonction.

Voici donc le code à ajouter en tout début de la fonction :

```
  if((x0 < 0  && x1 < 0)  ||
     (y0 < 0  && y1 < 0)  ||
     (x0 >= w && x1 >= w) ||
     (y0 >= h && y1 >= h)   )
    return;
```

6. Il y a encore pire, il arrive qu'un débordement mémoire ne provoque pas d'erreur de segmentation car on serait tombé sur une zone mémoire où nous avons encore le droit de lire et/ou d'écrire. Dans ce cas, ceci provoque une réécriture du code lui-même, alterant parfois des instructions ou des données, et ce genre de situations provoque un comportement "aléatoire" de la part de votre programme. Des experts en programmation / système peuvent utiliser ce type de bug comme une faille de sécurité pour pénétrer les systèmes qui utiliseraient votre code.

Pour finir en beauté et tester une multitude de segments exploitant notre fonction “complète” `drawLine`, je vous propose un bout de code de ce type à ajouter dans votre fonction `main` (n'oubliez pas le `#include <math.h>` pour avoir accès à `cos` et `sin` :

```
for(int i = 0; i < 360; i += 6)
  drawLine(image, W, H, W >> 1, H >> 1,
     (W >> 1) + (int)(29.0f * cos(i * M_PI / 180.0f)),
     (H >> 1) + (int)(29.0f * sin(i * M_PI / 180.0f)), 128 + (rand()&127));
/* x >> 1 se lit "x decale en binaire une fois a droite"
 * et signifie qu'on divise x par 2
 * en binaire un decalage a droite est une division par 2
 * deux decalages, une division par 4, trois, par 8, ...
 *
 * en base 10, un decalage a droite est une division par 10,
 * deux, par 100, trois, par 1000, ...
 *
 * voir et comprendre les decalages a gauche.
 */
```

Remarque : Nous avons pour habitude de représenter un repère (x, y) avec l'axe des x (ou des abscisses) comme une flèche vers la droite, et l'axe des y (ou des ordonnées) comme une flèche vers le haut. Généralement, dans les interfaces graphiques (sauf OpenGL®) ou les formats de fichiers images, l'axe des ordonnées est orienté vers le bas, ce qui met le point $(0, 0)$ de l'image en haut à gauche et le point $(w - 1, h - 1)$ en bas à droite.

Aller plus loin : Si vous souhaitez aller plus loin dans cette voie, je vous invite à vous renseigner sur l'analyse discrète différentielle et particulièrement, pour commencer, l'algorithme du tracé de droites de Bresenham :

`http://fr.wikipedia.org/wiki/Algorithme_de_trac%C3%A9_de_segment_de_Bresenham`

Bibliographie

[GL420] GL4D. Introduction à gl4dummies et documentation de référence. `https://api8.fr/GL4D`, 2020.

[KSS16] John Kessenich, Graham Sellers, and Dave Shreiner. *OpenGL Programming Guide : The Official Guide to Learning OpenGL, Version 4.5 with SPIR-V*. Addison-Wesley, 2016.