Première partie
présentation

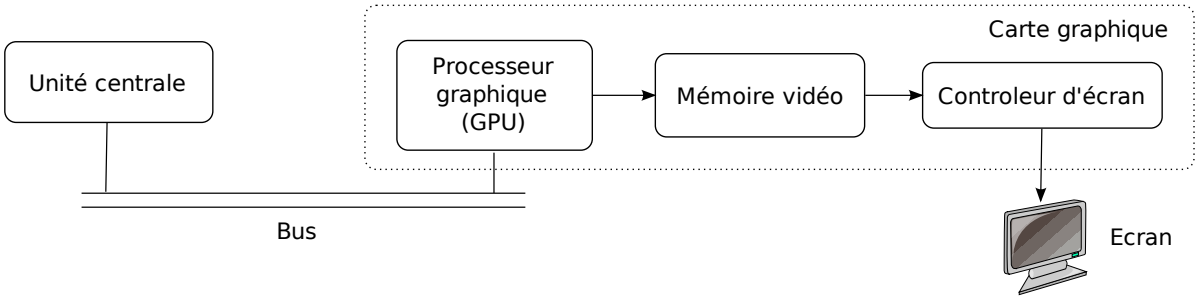
1 Introduction

Motivations

- De nos jours, les ordinateurs possèdent tous un périphérique de visualisation permettant d'afficher des images.
- Ce périphérique se matérialise par un écran (CRT ou LCD) ou par la projection d'une image à l'aide d'un vidéo-projecteur.
- Les images visualisées sont synthétisées par l'ordinateur, et donc par un programme informatique.
- Comment les images synthétisées par un programme s'affiche sur un écran ?

1.1 Architecture matériel

Architecture matérielle



Les ordinateur modernes ou les consoles de jeux vidéo possèdent tous un **processeur graphique** (GPU pour Graphical Processing Unit) permettant au processeur centrale de se libérer des calcules spécifiques à l'affichage. Il est relié à l'unité centrale via un **bus de données** (Pour les PC : ISA, PCI, AGP, PCI Express, USB, ...) Les images traité par le processeur sont stocké dans la **mémoire vidéo**. Un **contrôleur d'écran** se charge de transmettre l'image de la mémoire vidéo à l'écran :

- Sortie analogique (comme VGA) : l'image est convertie en signal analogique par le RAMDAC (Random Access Memory Digital-to-Analog Converter)
- Sortie numérique (comme DVI) : l'image numérique est envoyé directement à l'écran.

Dans l'architecture d'un PC, l'ensemble des trois composants (Processeur graphique, Mémoire vidéo, Contrôleur d'écran) est regroupé dans ce qu'on appel la **carte graphique**.

La mémoire vidéo

A l'origine, la mémoire avait pour seul tache de stocker l'**image courante** à afficher. De nos jour, leur grande capacités (se comptant en giga-octet), est utilisé pour stocker d'autre données :

- Images à afficher plus tard (frame-buffer)
- Texture d'objets
- Objets 3D

Note : Une partie de la mémoire centrale de l'ordinateur peut être utilisé de mémoire vidéo. C'est souvent le cas si la carte graphique est intégré à la carte mère (chipset vidéo).

1.2 L'image en Informatique

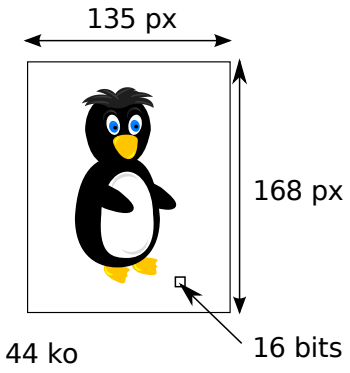
Qu'est qu'une image en informatique ?

Image : Surface rectangulaire divisée en unités élémentaires appelées **pixel**

Caractéristique d'une image :

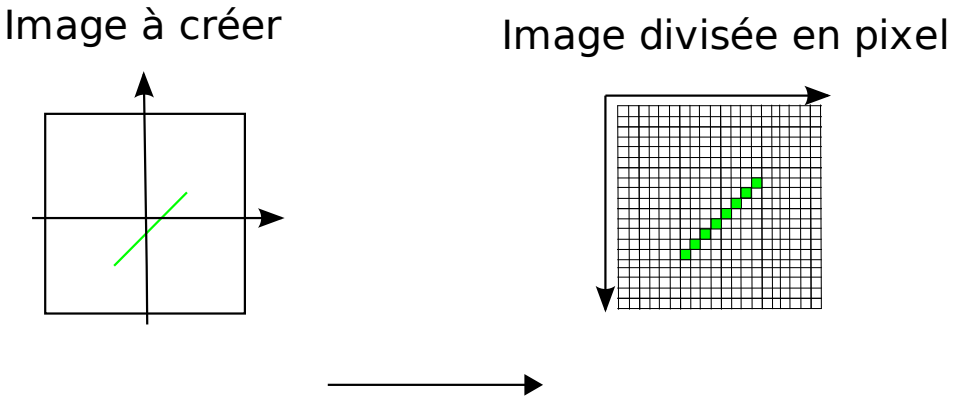
- Taille : nombre de pixel en longueur (length) et en hauteur (height). Ex : 1024x768.
- Profondeur des couleurs : dépend du nombre de bits utilisé pour coder un pixel :
  - 1 bit : 2 couleurs (Noir et blanc)
  - 8 bits : $2^8 = 256$ couleurs (Couleur)
  - 16 bits : 65536 couleurs (Haute couleur)
  - 24 bits : 16 millions de couleur (Vraie couleur), 8 bits pour chaque composante (Rouge,Vert,Bleu).
  - 32 bits : 8 bits pour chaque composante + 8 bit pour le niveau de transparence (Alpha).

Poids d'une image en mémoire



- On calcule le poids d'une image par : $longueur \times hauteur \times profondeur$
- Exemple pour une image 135x168 en 16 bits : $135 \times 168 \times 16 = 362880 bits \approx 44 ko$.
- A l'époque où la mémoire vidéo était limité, la taille de l'image affiché à l'écran était limité. Pour afficher une image de 1024x768 en vraie couleur (24 bits) la mémoire vidéo doit être donc d'au moins 2.25 Mo.

Synthèse d'une image

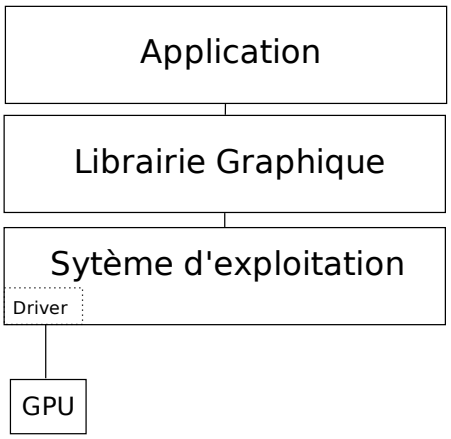


- Pour créer une image, il faut determiner chaque pixel qui la compose

– Nous devons projeter l'image calculer sur la surface divisée en pixel.
– Le repère d'une image est différent de celui classiquement utilisé
– C'est le rôle des différents algorithmes que vous verrez en cour.

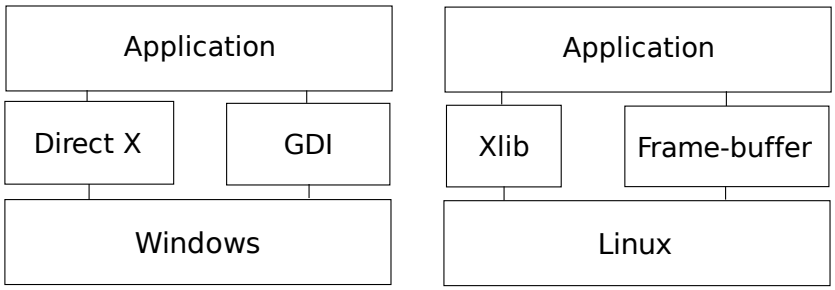
1.3 Architecture logiciel

Outils logiciel pour la programmation graphique



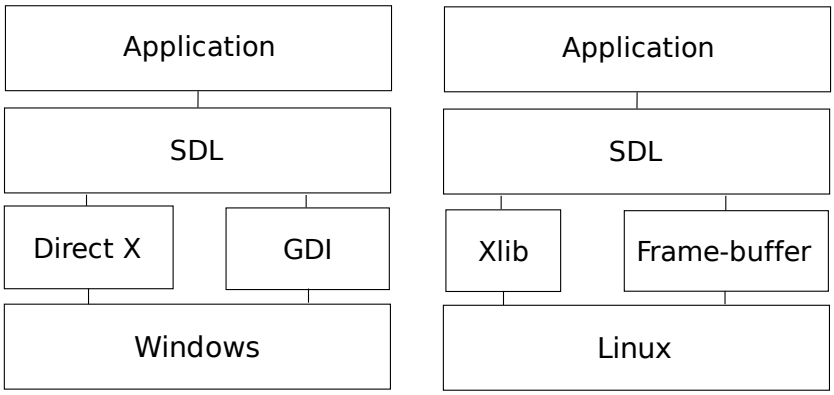
– Nous avons vu qu'une image générée doit être envoyée au processeur graphique avant d'être affichée à l'écran
– Pour faciliter la tâche des programmeurs, le système d'exploitation joue le rôle d'interface entre le matériel et le logiciel
– Les périphériques de visualisation vidéo sont donc gérés par le système d'exploitation (via les drivers)

Librairies graphique de bas niveau



– Les systèmes d'exploitation ou un système X associés proposé une librairie graphique permettant d'afficher des images à l'écran.
– Sous Windows il existe GDI (dans l'API Win32) pour l'affichage orienté bureautique ou les petites animations multimédias, et DirectX pour l'affichage d'animation multimédia complexe et les jeux vidéo 3D.
– Sous GNU/Linux, XLib est utilisé pour l'affichage orienté bureautique. Mais pour des applications exigeantes en performances, le framebuffer Linux (fbdev) est utilisé.

Librairies graphique de haut niveau



Au dessus d'une librairie bas niveau, il existe des librairies graphiques de plus haut niveau, permettant à la fois de simplifier la tache de programmeur, mais également le développement d'application multi-plateforme.

- Pour réaliser une interface graphique : GTK, Qt, etc.
- Pour réaliser des applications multimédias il existe **SDL**.

2 SDL (Simple DirectMedia Layer)

Présentation de SDL

- La librairie SDL est très utilisée dans le domaine de la création de jeux vidéo et d'application multimedia. Elle peut être utilisée avec une multitude le language (C, C++, C#, Java, Python, etc.) sur de nombreux système d'exploitation.
- Elle est disponible en OpenSource sous la licence GNU/GPL.
- Elle permet de gérer :
  - L'affichage vidéo : 2D et 3D avec OpenGL,
  - Les événements : souris, clavier, joystick,
  - L'audio : jouer de la musique,
  - Le multi-threding : Exécuter plusieurs fonctions "simultanément",
  - Le temps : mesure d'intervalle de temps précis, attente.

La librairie

- Comme toute librairie, SDL est composée d'un **ensemble de fonctions**.
- Les fonctions sont regroupées dans **9 catégories** :
  - Général,
  - Vidéo,
  - Gestion des fenêtres,
  - Événement,
  - Joystick,
  - Audio,
  - CD-ROM,
  - Multi-threading,
  - Gestion du temps.
- Le nom d'une fonction SDL commence par **SDL_**

2.1 Exemple 1

Exemple 1 : fonction principale

```
int main(){

    SDL_Surface *screen;
    SDL_Event event;
    int step=0;

    /* Initialise la libraire SDL */
    SDL_Init(SDL_INIT_VIDEO);

    /* netoye au moment ou le programme quitte*/
    atexit(SDL_Quit);

    /* Initialise l'affichage en 640x480 avec une profondeur de 8 bits */
    screen = SDL_SetVideoMode(640, 480, 8, SDL_SWSURFACE);

    //dessine la scene
    draw_scene(screen);

    /*boucle d'evenement*/
    while ( SDL_WaitEvent(&event) >= 0 ) {
        switch (event.type) {

            case SDL_QUIT:
              exit(0);
              break;

        }
    }
    return 0;
}
```

Exemple 1 : fonction draw_scene

```
void draw_scene(SDL_Surface *screen){
    int x, y;
    Uint32 yellow;
    SDL_Rect rect;

    /* calcul la position du millieu de l'ecran  */
    x = screen->w / 2;
    y = screen->h / 2;

    rect.x = x - 5;
    rect.y = y - 5;
    rect.w = rect.h = 10;

    /* retrouve la valeur d'un pixel jaune */
    yellow = SDL_MapRGB(screen->format, 0xff, 0xff, 0x00);

    /* Dessine un rectangle */
    SDL_FillRect(screen, &rect, yellow);

    /* Actualise juste la surface ou le rectangle est dessine */
    SDL_UpdateRect(screen,rect.x, rect.y, rect.w , rect.h);

    return;
}
```

2.2 Fonctions d'Initialisation

Initialisation : SDL_Init

```
#include "SDL.h"

int SDL_Init(Uint32 flags);
```

- Initialise SDL.
- Elle doit être appelée avant tout autre fonction SDL.
- Le paramètre *flags* spécifie quel partie de SDL initialiser :
  - SDL_INIT_TIMER : Initialise le sous-système “timer ”
  - SDL_INIT_AUDIO : Initialise le sous-système“audio”
  - SDL_INIT_VIDEO : Initialise le sous-système “video”
  - SDL_INIT_CDROM : Initialise le sous-système “cdrom”
  - SDL_INIT_JOYSTICK :Initialise le sous-système “joystick”
  - SDL_INIT_EVERYTHING :Initialise tout.

2.3 Fonctions vidéo

Structure : SDL_Surface

```
typedef struct SDL_Surface {
        Uint32 flags;                           /* Read-only */
        SDL_PixelFormat *format;                /* Read-only */
        int w, h;                               /* Read-only */
        Uint16 pitch;                           /* Read-only */
        void *pixels;                           /* Read-write */

        /* clipping information */
        SDL_Rect clip_rect;                     /* Read-only */

        /* Reference count -- used when freeing surface */
        int refcount;                           /* Read-mostly */

        /* This structure also contains private fields not shown here */
} SDL_Surface;
```

- SDL_Surface représente une zone mémoire “graphique” dans lequel on peut dessiner dedans.
- Le framebuffer vidéo est retourné comme une SDL_Surface par SDL_SetVideoMode et SDL_GetVideoSurface

Structure : SDL_Surface

- Le paramètre *flags* est une combinaison des valeur suivante :
  - SDL_SWSURFACE Crée un surface vidéo dans la mémoire sytème
  - SDL_HWSURFACE Crée un surface vidéo dans la mémoire vidéo
  - SDL_DOUBLEBUF Active le double buffering ; seulement valide avec SDL_HWSURFACE.
  - SDL_FULLSCREEN Mode plein écran
  - ...
- format : Format du pixel (Profondeur, entre autre)
- w, h : Largeur et hauteur de la surface
- pitch : Taille d’une ligne de la surface
- pixels : Pointeur vers le tableau de pixel
- clip_rect : Rectangle sur la surface à actualiser

Initialisation de l’affichage : SDL_SetVideoMode

```
#include "SDL.h"

SDL_Surface *SDL_SetVideoMode(int width, int height, int bpp, Uint32 flags);
```

- Spécifie le mode vidéo avec sa taille (width et height) et la profondeur (bpp).
- Si bpp est 0, alors la profondeur est celle de l’affichage courant.
- Le paramètre *flags* est le même que dans la structure SDL_Surface.

Zone à actualiser : SDL_UpdateRect

```
#include "SDL.h"

void SDL_UpdateRect(SDL_Surface *screen, Sint32 x, Sint32 y, Sint32 w, Sint32 h);
```

- Fait en sorte d’actualiser la zone de l’écran spécifié
- La zone est spécifié par un rectagle situé à (x,y) de longueur w et de hauteur h.

Dessiner un rectangle : SDL_FillRect

```
#include "SDL.h"

int SDL_FillRect(SDL_Surface *dst, SDL_Rect *dstrect, Uint32 color);
```

- Dessine un rectangle *dstrect* dans la surface *dst* avec une couleur *color*.
- Si dstrect est NULL, toute la surface est remplie par la couleur *color*.
- La couleur doit être compatible avec le format du pixel, et peut être généré par la fonction SDL_MapRGB.

Créer un couleur : SDL_MapRGB

```
#include "SDL.h"

Uint32 SDL_MapRGB(SDL_PixelFormat *fmt, Uint8 r, Uint8 g, Uint8 b);
```

- Retourne la valeur du pixel correspondant la plus possible à la couleur spécifié par r,g et b
- r,g et b sont les valeur des composantes rouge, vert et bleu (Red, Green, Blue)
- *fmt* peut être obtenu par le champ *format* de la structure SDL_Surface.

2.4 Boucle d'évenement

Boucle d'évenement

```
   SDL_Event event;
   while ( SDL_WaitEvent(&event) >= 0 ) {
       switch (event.type) {

           case SDL_QUIT:
             exit(0);
             break;
       }
   }
```

- La boucle d'événement sert à récupérer une événement pouvant se produire à tout moment
- Lorsque SDL génère un événement il l'ajoute à une file d'attente
- La fonction SDL_WaitEvent permet de récupérer l'événement dans la file pour le traiter
- Si la file est vide, SDL_WaitEvent bloque le programme en attendant la survenue d'un nouvelle événement
- Si SDL_WaitEvent renvoi 1 si aucune erreur.

Structure SDL_Event

```
  typedef union{
    Uint8 type;
    SDL_ActiveEvent active;
    SDL_KeyboardEvent key;
    SDL_MouseMotionEvent motion;
    SDL_MouseButtonEvent button;
    SDL_JoyAxisEvent jaxis;
    SDL_JoyBallEvent jball;
    SDL_JoyHatEvent jhat;
    SDL_JoyButtonEvent jbutton;
    SDL_ResizeEvent resize;
    SDL_QuitEvent quit;
    SDL_UserEvent user;
    SDL_SywWMEvent syswm;
  } SDL_Event;
```

- L'union SDL_Event est le cœur de tout événement traité par SDL.
- C'est la structure la plus importante après SDL_Surface
- SDL_Event est l'union de toutes les structure d'événement de SDL. On utilise la structure correspondant au champ *type*.

type	Structure
SDL_ACTIVEEVENT	SDL_ActiveEvent
SDL_KEYDOWN/UP	SDL_KeyboardEvent
SDL_MOUSEMOTION	SDL_MouseMotionEvent
SDL_MOUSEBUTTONDOWN/UP	SDL_MouseButtonEvent
SDL_JOYAXISMOTION	SDL_JoyAxisEvent
SDL_JOYBALLMOTION	SDL_JoyBallEvent
SDL_JOYHATMOTION	SDL_JoyHatEvent
SDL_JOYBUTTONDOWN/UP	SDL_JoyButtonEvent
SDL_QUIT	SDL_QuitEvent
SDL_SYSWMEVENT	SDL_SysWMEvent
SDL_VIDEORESIZE	SDL_ResizeEvent
SDL_USEREVENT	SDL_UserEvent

Structure SDL_KeyboardEvent

```
typedef struct{
  Uint8 type;
  Uint8 state;
  SDL_keysym keysym;
} SDL_KeyboardEvent;
```

- type : SDL_KEYDOWN ou SDL_KEYUP
- state : SDL_PRESSED ou SDL_RELEASED
- keysym : information sur la touche pressée

Structure SDL_keysym

```
typedef struct{
  Uint8 scancode;
  SDLKey sym;
  SDLMod mod;
  Uint16 unicode;
} SDL_keysym;
```

- scancode : code materiel de la touche
- sym : identifiant SDL de la touche
- mod : touches modificatrices pressées (CTRL, ALT, etc.).
- unicode : touche d'un caractère UNICODE

Exemple avec la touche Echap

```
  SDL_Event event;
  while ( SDL_WaitEvent(&event) >= 0 ) {
    switch (event.type) {
      case SDL_KEYDOWN:
        switch(event.key.keysym.sym){
          case SDLK_ESCAPE:
            printf("Bye␣bye...\n");
            exit(0);
            break;
        }
        break;
    }
  }
```

2.5 Exemple 2

Exemple 2 : modifier un pixel

2.6 Compilation

Fichier d'entete et compilation

Fichier d'en-tete à ajouter :

```
  #include "SDL.h"
```

Compilation avec gcc :

```
  gcc -lSDL -o mon_programme mon_programme.c
```

Documentation de SDL sur le site : http ://www.libsdl.org/

3 Exercices

Tracer un dessin avec les flèches

1. Ajouter deux variables globales *pos_x* et *pos_y* initialisées à 0
2. En gérant les événements liées aux touches des flèches du clavier :
   - ajouter 1 à pos_x quand la touche “flèche droite” est pressé
   - enlever 1 à pos_x quand la touche “flèche gauche” est pressé
   - ajouter 1 à pos_y quand la touche “flèche bas” est pressé
   - enlever 1 à pos_y quand la touche “flèche haut” est pressé
3. Quand la touche ”Entrée“ est pressée, un point doit être dessiné à l'emplacement (pos_x,pos_y). *pos_x* et *pos_y* doivent être toujours positif et respectivement inférieur à la largeur et à la hauteur de l'écran.
4. Un point doit se déplacer à l'écran lorsque les flèches du clavier sont pressées.
5. Au final, on doit pouvoir tracer une ligne verticales ou horizontales en déplaçant le point et en pressant simultanément la touche ”Majuscule“.