Computer Programming

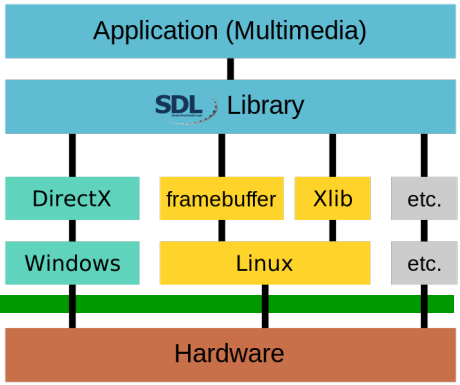
Third-Party Library

"Non-Standard" Library Functions

- Third-party library
  - Many libraries are designed to perform different tasks
    - Graphics, mathematics, sound, gaming, …
  - Typically distributed with binary *libraries* (for linker) as well as *header files* (for compiler)
    - Some are open-source libraries (distributed with source codes) that allow you to make changes to these functions and/or to "compile" the binary files for your own platform
  - ☞ Static library vs. shared library
    - Static library (.a or .lib): statically embedded to your program
    - Shared (dynamically linked) library (.so or .dll): not embedded in your program, but needs to be present in specific directories when your program runs

For DLL, an import library (.lib) is also provided for the linker to resolve references

① The SDL Library



- Simple DirectMedia Layer (SDL)
  - A *cross-platform* development library designed to provide low-level access to audio, keyboard, mouse, joystick, and graphics hardware via OpenGL and Direct3D
  - It is used in many commercial games
  - It supports Windows, macOS, Linux, iOS, and Android
  - It is written in C and works natively with C++
- Features
  - Video: 3D graphics
  - Audio: 5.1 surround sound
  - Input: multitouch gestures
  - Many extension libraries

Installation (Windows)

Search directories and link libraries can be set as the build options of a project (*per-project* settings)

- Download link

  https://www.libsdl.org/release/SDL2-devel-2.0.10-mingw.tar.gz

Replace <ROOT> with **any name** you like

It is okay to download the latest version of SDL for your project

- Unzip the directory to any location (call it *<ROOT>*)
- Add the search directory for header files (for "include")
  - *<ROOT>/SDL2-2.0.10/i686-w64-mingw32/include/SDL2*
- Add the search directory for the library (for linking)
  - *<ROOT>/SDL2-2.0.10/i686-w64-mingw32/lib*
- Add the link options for built-in libraries
  - *-lmingw32 -lSDL2main -lSDL2*
- Copy SDL2.dll to the directory where your program is
- (Optional) Install additional extension libraries
  - SDL_image (for different image formats), SDL_mixer (for audio), SDL_ttf (for font), …

i686-w64-...: 32-bit
x86_64-w64-...: 64-bit

`http://www.libsdl.org/projects`

Installation (macOS)

You can use Xcode or other IDEs (e.g. CLion) to develop SDL2 programs

- Download link

  https://www.libsdl.org/release/SDL2-2.0.10.dmg

  - Open the dmg file
  - Copy file "SDL2.framework" to /Library/Frameworks
    - *Choose from menu "Go::Go To Folder" to choose the folder*
  - Re-sign the SDL2 framework
    - *Open the Terminal application*
    - *codesign -f -s - /Library/Frameworks/SDL2.framework/SDL2*
  - Open a new project and continue the settings

  ☞ Check the following website for detailed instructions

  http://lazyfoo.net/tutorials/SDL/01_hello_SDL/mac/index.php

SDL Tutorials

- Check the tutorials provided

  http://lazyfoo.net/tutorials/SDL/

  - Show image
  - Animation
  - Keyboard & mouse control
  - Sound effects
  - …

  ☞ Demo files (in the tar file): *<ROOT>/SDL2-2.0.10/test*

  ☞ Newer version of SDL may be available – check the official website for your reference

  https://www.libsdl.org

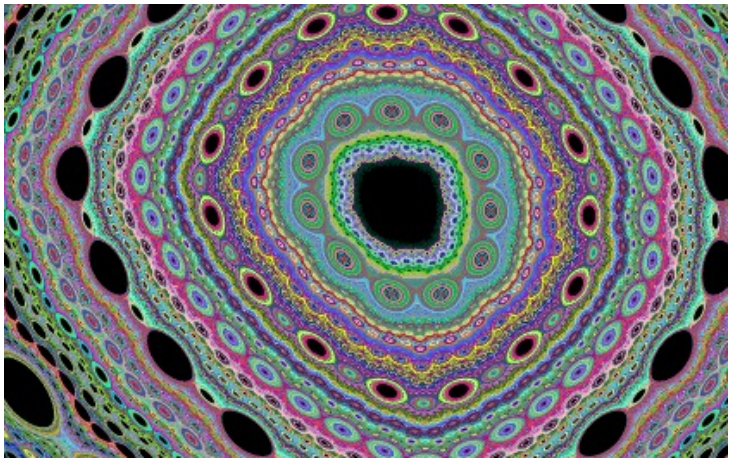
Very useful for you to get started *steps by steps* towards your final project

② The BGI Library atop SDL

- SDL_bgi

  `https://sdl-bgi.sourceforge.io`

  - BGI: Borland Graphics Interface
    - BGI is originally developed for the DOS environment
    - It allows *pixel-by-pixel* control of the windows
    - *cf.* character-by-character control in the console mode
  - It is a *wrapper* atop SDL2 for graphics programming
  - It provides functions such as
    - `circle(), ellipse(), drawpoly(), line(), putpixel(), setcolor(), setfillpattern(), outtextxy(), putimage(), …`



  - We will *use it later in the lectures* on C++

Installation

You need to install SDL2 first before proceeding with SDL_bgi installation

- Simple package distribution

  https://sourceforge.net/projects/libxbgi/files/SDL_bgi-2.3.0.tar.gz

  - Essential files (under directory *src*)
    - `graphics.h` (for including as the main header file)
    - `SDL_bgi.c`, and `SDL_bgi.h` (for building the library)

SDL_bgi is written in C (no C++ syntax)

- Building options
  - Built as a library (binary) and linked to your program
    - A C library (created by a C compiler/linker) cannot be directly used in a C++ program
    - *Some syntax* needs to be provided for function declarations
  - Added to your project (source code) for compiling together with your own code
    - It needs to be *converted into C++ codes* (.cpp) first

Recall Function Overloading

Different compilers may have different rules for name mangling

- Function overloading
  - In C++, it is possible to define two or more functions with the same name but with different argument lists
  - A C++ compiler uses "name mangling" to encode each function differently to distinguish overloaded functions

Use nm to find the symbol table

```
int maximum(int, int);
int maximum(int, int, int);
double maximum(double, double);
```

→ `__Z7maximumii`

→ `__Z7maximumiii`

→ `__Z7maximumdd`

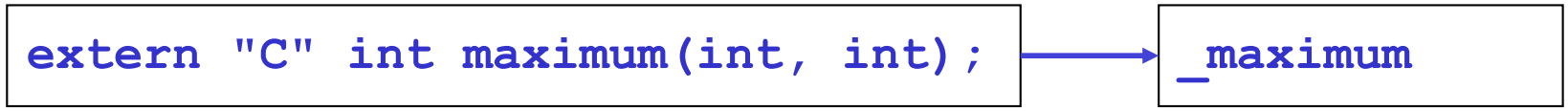
  - C does not support function overloading and hence it does not perform name mangling as C++ does

☞ Problems exist when a C++ program needs to call functions written (compiled) in C

Enter `extern "C"`

If SDL_bgi.c is converted to C++, there is no need (but okay) to use extern "C" in the header (SDL_bgi.h)

- Function overloading
  - By declaring a function with `extern "C"`, the C++ compiler does not add the extra mangling information to the symbol (just like C does)



  - Use `extern "C" {}` to enclose multiple functions
  - Note the use of extern "C" {} in *SDL_bgi.h*
- Alternatively, SDL_bgi can be converted to C++
  - Rename *SDL_bgi.c* as *SDL_bgi.cpp*
  - Change *char ** to *const char **
  - Explicit cast (*void **) to other types (e.g. *int ** or *Uint32 **)

An Example with Animation

```
#include <graphics.h>

int main(int argc, char*argv[])
{
    int x = 0, y = 0, dx = 1, dy = 1, r = 100;
    initwindow(640, 480);                    // window size
    while (1)
    {
        if (kbhit()) break;                  // a key is hit
        x += dx;  y += dy;
        if (x < r) dx = 1;   if (x >= getmaxx() - r)  dx = -1;
        if (y < r) dy = 1;   if (y >= getmaxy() - r)  dy = -1;
        cleardevice();                       // for redraw
        setcolor(GREEN);
        setfillstyle(SOLID_FILL, BLUE);
        fillellipse(x, y, r, r);
        refresh(); delay(10);                // update screen
    }
    closegraph();
}
```

Use **#include "graphics.h"** if it is in the local directory

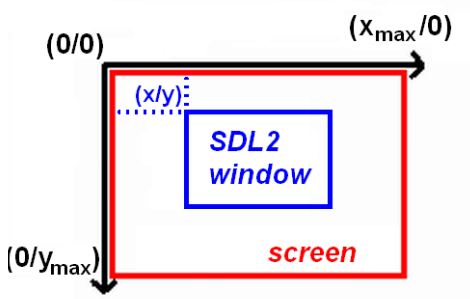
Needed by some compilers for creating an SDL2 program

Another Example with Full Screen

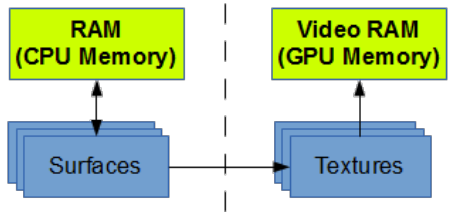
```
#include <graphics.h>
#include <ctime>

int main(int argc, char*argv[])
{
     initwindow(0, 0);    // full screen
     setcolor(RED);
     settextstyle(DEFAULT_FONT, HORIZ_DIR, 2);
     time_t now;          tm* timeinfo;
     char buffer [80];
     for (;;) {
        if (kbhit()) break;
        cleardevice();
        time (&now);      timeinfo = localtime(&now);
        strftime(buffer, 80 ,"Now it's %I:%M:%S%p.", timeinfo);
        outtextxy(random(getmaxx()), random(getmaxy()), buffer);
        refresh(); delay(1000);
     }
     closegraph();
}
```

Use **#include "graphics.h"** if it is in the local directory

Back to the SDL Library



- Window and renderer
  - The window (SDL_Window) is the entity that is showing the graphic output and the renderer (SDL_Renderer) is the "machine" that is generating the output to be shown in the window
  - Rendering is the process of *synthesizing* the final image on your screen from the data stored in memory
    - Software rendering: Image data stored in RAM (SDL_Surface)
    - Hardware rendering: Image data stored in VRAM (SDL_Texture)
    - Hardware rendering is faster and preferred whenever possible
  - ☞ In SDL, a renderer can be created for a *window* to show the *texture*
    - SDL_Surface is more accessible and it can be used to convert to SDL_Texture



A Simple Window

You may need to replace <SDL.h> with <SDL2/SDL.h> in Xcode depending on your search directory settings

```
#include <SDL.h>

const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;

int main(int argc, char*argv[])
{
    if(SDL_Init(SDL_INIT_VIDEO) < 0) return 0;
    SDL_Window* gWindow = SDL_CreateWindow(
            "SDL Tutorial",
            SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
            SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);
    if(gWindow == NULL) return 0;
    SDL_Surface* gScreenSurface = SDL_GetWindowSurface(gWindow);

    SDL_FillRect(gScreenSurface, NULL,
    SDL_MapRGB(gScreenSurface->format, 0x77, 0xA8, 0x48));
    SDL_UpdateWindowSurface(gWindow);
    SDL_PumpEvents();
    SDL_Delay(2000);                        // wait for 2 seconds
    SDL_DestroyWindow(gWindow);
    SDL_Quit();
}
```

Initialization code to be **used again in later examples**

Loading an Image

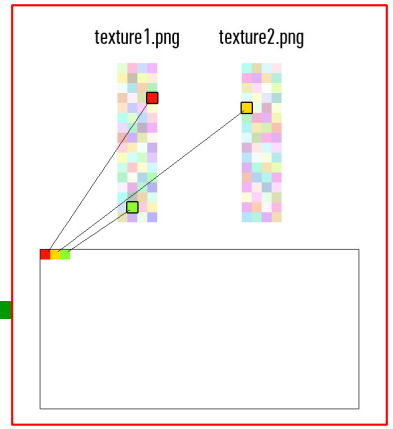
Remember to put the "hello.bmp" file in the same directory as the compiled program

```
#include <SDL.h>
#include <iostream>
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;

int main(int argc, char*argv[])
{
    // Put the initialization code
    // in the previous example here

    SDL_Surface *gHelloSurface = SDL_LoadBMP("hello.bmp");
    if(gHelloSurface == NULL) {
        std::cout << "Unable to load image!"; return 0;
    }

    SDL_BlitSurface(gHelloSurface, NULL, gScreenSurface, NULL);
    SDL_UpdateWindowSurface(gWindow);

    SDL_Delay(2000);                    // wait for 2 seconds
    SDL_FreeSurface(gHelloSurface);
    SDL_DestroyWindow(gWindow);
    SDL_Quit();
}
```

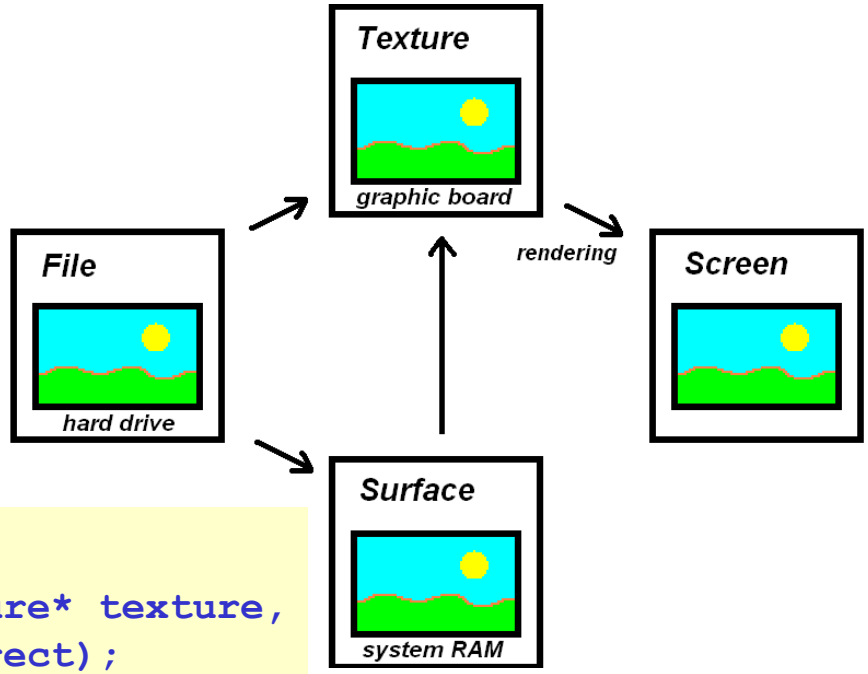
Any bitmap file will do

blit is the operation to combine multiple bitmaps into one using some Boolean function (AND/OR/XOR/...)

Rendering Texture



- Creating texture
  - ① Creating a surface from an image file and then creating a texture from the surface
  - ② Creating a texture directly from an image file (*SDL_image* extension library is required)
- Rendering texture
  - A texture can be rendered into a smaller region of the target
  - A texture can be partially rendered to the target




```
int SDL_RenderCopy(
SDL_Renderer* renderer, SDL_Texture* texture,
SDL_Rect* srcrect, SDL_Rect* dstrect);
```

Rendering an Image

Error checking is removed in this example for lack of space – add for yourself

```
#include <SDL.h>
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;

int main(int argc, char*argv[])
{
    SDL_Init(SDL_INIT_VIDEO);
    SDL_Window* gWindow = SDL_CreateWindow(
        "SDL Tutorial", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED,
        SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);
    SDL_Renderer *gRenderer = SDL_CreateRenderer(gWindow, -1, 0);

    SDL_Surface *gSurface = SDL_LoadBMP("foo.bmp");
    SDL_Texture *gTexture = SDL_CreateTextureFromSurface(gRenderer,
     gSurface);

    SDL_Rect rect = {5, 5, 320, 240};
    SDL_RenderCopy(gRenderer, gTexture, NULL, &rect);
    //SDL_RenderCopy(gRenderer, gTexture, NULL, NULL);
    SDL_RenderPresent(gRenderer); SDL_Delay(2000);

    SDL_DestroyTexture(gTexture); SDL_FreeSurface(gSurface);
    SDL_DestroyRenderer(gRenderer); SDL_DestroyWindow(gWindow);
    SDL_Quit();
}
```

Initialization code to be used again in later examples

Animated Image

Remember to put the "foo.bmp" file in the same directory as the compiled program

```
#include <SDL.h>
const int SCREEN_WIDTH = 640;
const int SCREEN_HEIGHT = 480;

int main(int argc, char*argv[])
{
    // Put the initialization code in the previous example here
    SDL_Surface *gSurface = SDL_LoadBMP("foo.bmp");
    SDL_Texture *gTexture = SDL_CreateTextureFromSurface(gRenderer,
     gSurface);
    SDL_SetRenderDrawColor(gRenderer, 0, 255, 255, 255);
    SDL_Rect src[4] = {{0, 0, 64, 205}, {64, 0, 64, 205}, {128, 0, 64,
     205}, {196, 0, 64, 205}};
    SDL_Rect dst = {0, 200, 64, 205};
    for (int i=0;i<100;i++) {
        SDL_RenderClear(gRenderer);
        SDL_RenderCopy(gRenderer, gTexture, &src[i%4], &dst);
        SDL_RenderPresent(gRenderer); SDL_PumpEvents(); SDL_Delay(400);
        dst.x  = (dst.x + 64)%SCREEN_WIDTH;
    }
    SDL_DestroyTexture(gTexture);   SDL_FreeSurface(gSurface);
    SDL_DestroyRenderer(gRenderer); SDL_DestroyWindow(gWindow);
    SDL_Quit();
}
```

Get it from the web

Review

- Standard library
  - Random number generation and seed
  - Time and clock functions
  - C-style string functions
- Third-party library
  - Graphics mode application
  - Game programming