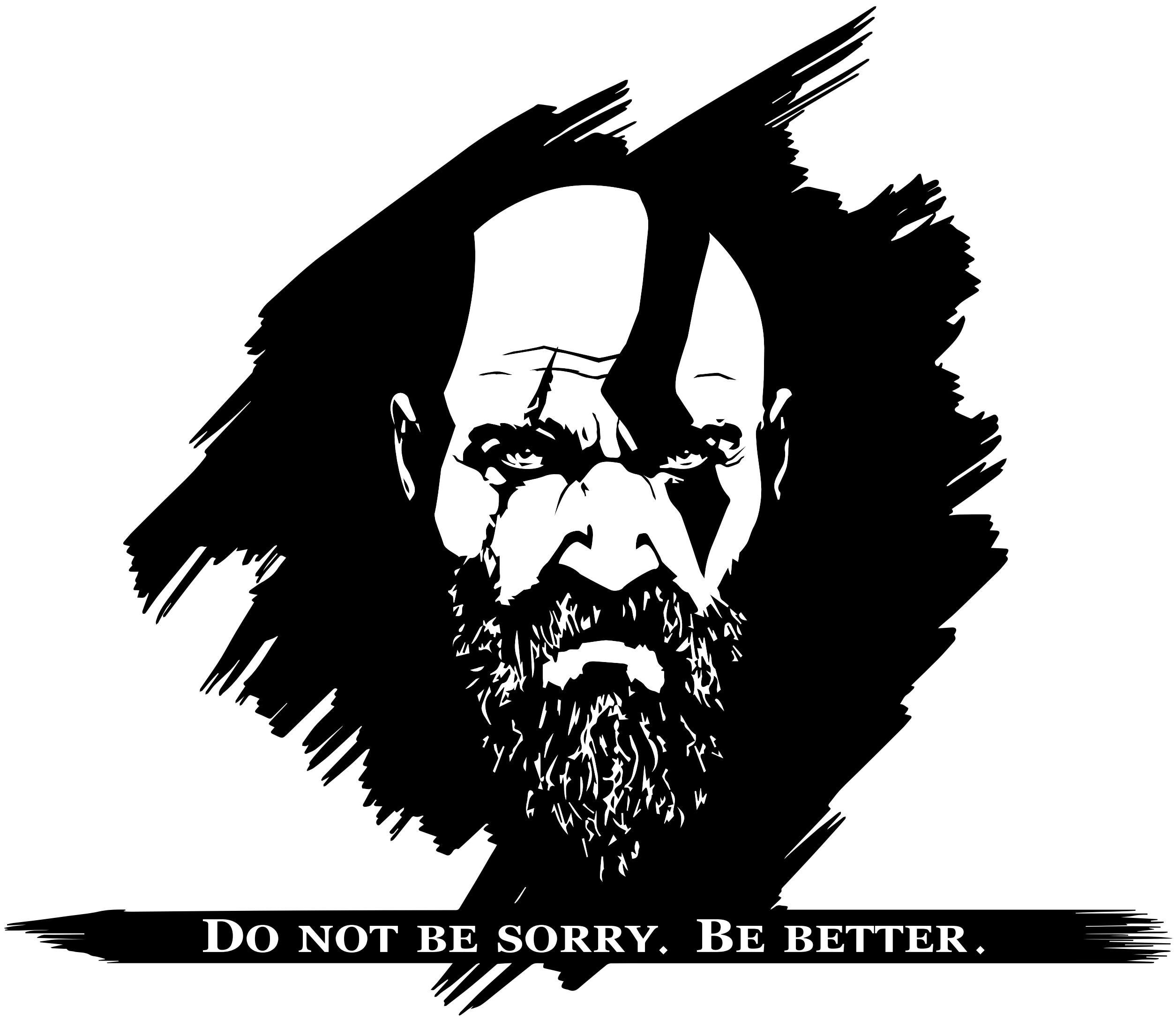
RUSH C — Tutorial

version **#deploy-rush-c-2021-v0.3**



ASSISTANTS C/UNIX 2019 `<assistants@tickets.assistants.epita.fr>`

Copyright

Contents

[*]. https://intra.assistants.epita.fr

1 General indications

In a game, you will most likely be using textures. Be careful to load the textures only once. Loading a sprite everytime you draw it will only critically slow down your application.

https://wiki.libsdl.org/APIByCategory

2 Multiple Buffering

When you want to draw a sprite or a tile, it wouldn't be optimized to update the image everytime. Instead multiple images are loaded, in our case two are loaded. One is shown on the screen, while the other is being updated. Every update is done on this second image. When the images' updates are done, this second image is shown, and the first one becomes the one to be updated.

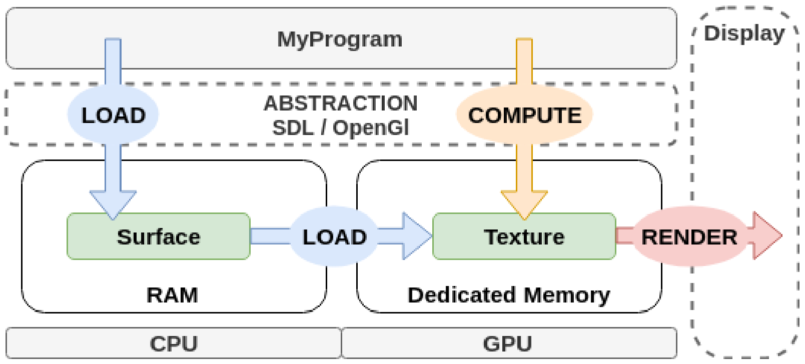
This change of image being shown is done with a call to the function `SDL_RenderPresent` explained below.

3 Compilation

```
#include <SDL.h>
```

```
CFLAGS += $(shell sdl2-config --cflags)
LDLIBS += $(shell sdl2-config --libs)
```

4 Data structures



4.1 Surface

SDL_Surface (https://wiki.libsdl.org/SDL_Surface) contains a collection of Pixels;

4.2 Texture

SDL_Texture (https://wiki.libsdl.org/SDL_Texture) is a structure containing an efficient representation of a pixel data.

4.3 Shape

SDL_Rect (https://wiki.libsdl.org/SDL_Rect) is the representation of a rectangle with (x, y) the origin at the upper left, w the width and h the height.

5 General functions

5.1 Initialization

```
int SDL_Init(Uint32 flags);
```

SDL_Init (https://wiki.libsdl.org/SDL_Init) must be called before everything else, it is used to initialize the SDL internally. Also please note that depending on the architecture, the call to this function may not be needed. It should always be called for portability reasons though.

5.2 Delay

```
void SDL_Delay(Uint32 ms);
```

SDL_Delay (https://wiki.libsdl.org/SDL_Delay) does nothing during the amount of time given as a parameter. This function is important to handle the display of the game. It is a good idea to add at least a delay of 1ms between two frames if you don't want the SDL to update and draw the image too quickly. You can also use this function to implement a different delta timing method.

5.3 Time

```
Uint64 SDL_GetPerformanceFrequency(void);
```

SDL_GetPerformanceFrequency (https://wiki.libsdl.org/SDL_GetPerformanceFrequency) is used to get the count per second of the high resolution counter.

```
Uint64 SDL_GetPerformanceCounter(void);
```

SDL_GetPerformanceCounter (https://wiki.libsdl.org/SDL_GetPerformanceCounter) is used this function to get the current value of the high resolution counter.

5.4 Window

```
SDL_Window *SDL_CreateWindow(const char *title, int x, int y, int w, int h,
                             Uint32 flags);
```

SDL_Window* SDL_CreateWindow (https://wiki.libsdl.org/SDL_CreateWindow) creates a window for the game.

5.5 Renderer

```
SDL_Renderer *SDL_CreateRenderer(SDL_Window *window, int index, Uint32 flags);
```

SDL_Renderer* SDL_CreateRenderer (https://wiki.libsdl.org/SDL_CreateRenderer) creates the renderer for the window that must be previously created.

5.6 Error handling

```
const char *SDL_GetError(void);
```

SDL_GetError (https://wiki.libsdl.org/SDL_GetError) is used to retrieve a message about the last error that occurred.

6 Drawing functions

6.1 Texture

```
SDL_Texture *SDL_CreateTextureFromSurface(SDL_Renderer *renderer,
                                          SDL_Surface  *surface);
```

SDL_CreateTextureFromSurface (https://wiki.libsdl.org/SDL_CreateTextureFromSurface) creates a texture from a renderer and a surface.

6.2 Draw

```
int SDL_RenderCopy(SDL_Renderer   *renderer, SDL_Texture *texture,
                   const SDL_Rect *src_rect, const SDL_Rect *dst_rect);
```

SDL_RenderCopy (https://wiki.libsdl.org/SDL_RenderCopy) adds the texture or a portion of the texture using the given renderer. The texture will not be shown on the screen. In order to update the screen you must update the screen with SDL_RenderPresent.

If you want a character with two sprites, instead of having two textures you can just load one containing both. You just have to move `src_rect` by the size of a sprite.

6.3 Display

```
void SDL_RenderPresent(SDL_Renderer *renderer);
```

SDL_RenderPresent (https://wiki.libsdl.org/SDL_RenderPresent) updates the screen with all the rendering performed since the previous call.

6.4 Get texture informations

```
int SDL_QueryTexture(SDL_Texture *texture,
                     Uint32      *format,
                     int         *access,
                     int         *w,
                     int         *h);
```

SDL_QueryTexture (https://wiki.libsdl.org/SDL_QueryTexture) is a function used to query textures. w and h are the width and height parameters of the texture. This function is needed because you cannot directly access to the members of the `SDL_Texture` class as this class is dependant on the drivers on your computer.

7 Memory handling

7.1 Surface

```
void SDL_FreeSurface(SDL_Surface *surface);
```

SDL_FreeSurface (https://wiki.libsdl.org/SDL_FreeSurface) frees an RGB surface.

7.2 Texture

```
void SDL_DestroyTexture(SDL_Texture *texture);
```

SDL_DestroyTexture (https://wiki.libsdl.org/SDL_DestroyTexture) frees the given texture.

7.3 SDL

```
void SDL_Quit(void);
```

SDL_Quit (https://wiki.libsdl.org/SDL_Quit) cleans up the system. This function should be called upon exit.

7.4 Renderer

```
void SDL_DestroyRenderer(SDL_Renderer *renderer);
```

SDL_DestroyRenderer (https://wiki.libsdl.org/SDL_DestroyRenderer) destroys the renderer passed in parameter.

7.5 Window

```
void SDL_DestroyWindow(SDL_Window *window);
```

SDL_DestroyWindow (https://wiki.libsdl.org/SDL_DestroyWindow) destroys a window.

8 Handling User input

Here are two ways of getting user's inputs. SDL_PollEvent shoud be the go-to function when trying to get user's inputs.

However SDL_GetKeyboardState can also be useful, particularly for the movement inputs because the input will be continous as long as the key is pressed down (when pressing down a key, there will be no delay before repeating the event). This is the "good" way.

8.1 Events

```
int SDL_PollEvent(SDL_Event *event);
```

SDL_PollEvent (https://wiki.libsdl.org/SDL_PollEvent) polls for the next event and fills the event structure in parameter. `SDL_Event` is a structure containing a union of the different events types.

Example:

```
SDL_Event event;
while (SDL_PollEvent(&event))
{
  if (event.type == SDL_QUIT)
  {
    // Exit
  }
}
```

8.2 Inputs

```
const Uint8 *SDL_GetKeyboardState(int *numkeys);
```

SDL_GetKeyBoardState (https://wiki.libsdl.org/SDL_GetKeyboardState) can be used to get the keyboards inputs of the user.

```
Uint32 SDL_GetMouseState(int *x, int *y);
```

SDL_GetMouseState (https://wiki.libsdl.org/SDL_GetMouseState) can be used to get the mouse inputs (cursor and buttons).

9 Image

9.1 SDL_Image

This is a simple library to load images of various formats as SDL surfaces. The SDL has `SDL_LoadBMP`, but it allows us to load only BMP, so we will use `SDL_Image` to load other format.

https://www.libsdl.org/projects/SDL_image/docs/SDL_image.html

10 Compilation

```
#include <SDL_image.h>
```

```
LDLIBS += -lSDL2_image
```

11 Data types

11.1 Surface

`SDL_Surface` (https://wiki.libsdl.org/SDL_Surface) contains a collection of Pixels.

12 General functions

12.1 Initialization

```
int IMG_Init(int flags);
```

`flags` represents which file format will be loaded. it can be several values OR'd together.

We can use it like this:

```
IMG_Init(IMG_INIT_PNG | IMG_INIT_JPG);
```

12.2 Closing

```
void IMG_Quit();
```

12.3 Loading

```
SDL_Surface *IMG_Load(const char *file);
```

12.4 Error handling

```
char *IMG_GetError();
```

This returns the reason of the last error.

Usually used like this:

```
int flags = IMG_INIT_JPG | IMG_INIT_PNG;
int initialized = IMG_Init(flags);
if (initialized & flags != flags)
{
  printf("IMG_Init: %s\n", IMG_GetError());
  exit(2);
}
```

13 Audio

13.1 SDL_Mixer

It's a simple multi-channel audio mixer. It supports 8 channels of 16 bits stereo audio, plus a single channel of music. You can play multiple samples at once, but you can only play one music at a time.

https://www.libsdl.org/projects/SDL_mixer/docs/SDL_mixer.html

14 Compilation

```
#include <SDL_mixer.h>
```

```
LDLIBS += -lSDL2_mixer
```

15 Data types

15.1 Music

`Mix_Music` represents a piece of music, something that can be played for an extended period of time, usually repeated.

15.2 Chunk

`Mix_Chunk` represents a sample, or in other words a sound effect.

16 General functions

16.1 Initialization

```
int Mix_Init(int flags);
```

`flags` represents which file format will be loaded. it can be several values OR'd together. SDL has to be initialized with `SDL_INIT_AUDIO`.

```
int Mix_OpenAudio(int frequency, Uint16 format, int channels, int chunksize);
```

This must be called before using other functions in this library. Don't forget to initialize SDL with the flag `SDL_INIT_AUDIO` before this call.

We can use it like this:

```
Mix_Init(MIX_INIT_OGG | MIX_INIT_MP3);
Mix_OpenAudio(MIX_DEFAULT_FREQUENCY, MIX_DEFAULT_FORMAT, 2, 1024);
```

16.2 Closing

```
void Mix_CloseAudio();
```

Shutdown and cleanup the mixer API.

```
void Mix_Quit();
```

16.3 Loading

```
Mix_Music *Mix_LoadMUS(char *path);
Mix_Chunk *Mix_LoadWAV(char *path);
```

16.4 Freeing

```
void Mix_FreeMusic(Mix_Music *music);
void Mix_FreeChunk(Mix_Chunk *chunk);
```

16.5 Playing music

```
int Mix_PlayMusic(Mix_Music *music, int loops);
```

`loops` is the number of times to play the track. (use `-1` to repeat indefinitly). You can `Halt`, `Resume`, `Rewind` and more, but all is described in the documentation.

To check if a Music is currently played, you can use:

```
int Mix_PlayingMusic();
```

16.6 Playing chunk

```
int Mix_PlayChannel(int channel, Mix_Chunk *chunk, int loops);
```

By default use `-1` for `channel`, it will use the first free channel. `loops` is the number of times to repeat the track. (`-1` is an infinite loop).

16.7 Error handling

```
char *Mix_GetError();
```

This returns the reason of the last error.

Usually used like this:

```
int flags = MIX_INIT_OGG | MIX_INIT_MP3;
int initted = Mix_Init(flags);
if (initted & flags != flags)
{
  printf("Mix_Init: %s\n", Mix_GetError());
  exit(2);
}
```

17 TTF

17.1 SDL_ttf

This library allows you to use TrueType fonts to render text in SDL applications.

https://www.libsdl.org/projects/SDL_ttf/docs/SDL_ttf.html

18 Compilation

```
#include <SDL_ttf.h>
```

```
LDLIBS += -lSDL2_ttf
```

19 Data types

19.1 Font

`TTF_Font` represents a loaded font.

20 General functions

20.1 Initialization

```
int TTF_Init();
```

This must be called before using other functions in this library.

20.2 Closing

```
void TTF_Quit();
```

20.3 Loading

```
TTF_Font *TTF_OpenFont(const char *file, int ptsize);
```

20.4 Freeing

```
void TTF_CloseFont(TTF_Font *font)
```

20.5 Displaying font

We need to get an `SDL_Surface`.

```
SDL_Surface *TTF_RenderText_Solid(TTF_Font *font, const char *text, SDL_Color fg);
SDL_Surface *TTF_RenderText_Shaded(TTF_Font *font, const char *text, SDL_Color fg,␣
↪SDL_Color bg);
SDL_Surface *TTF_RenderText_Blended(TTF_Font *font, const char *text, SDL_Color fg);
```

Example:

```
int texW;
int texH;
SDL_Color white = { 255, 255, 255, 255 };

TTF_Font *font = TTF_OpenFont("resources/arial.ttf", 50);
SDL_Surface *surface = TTF_RenderText_Solid(font, "Text to display", white);
SDL_Texture *texture = SDL_CreateTextureFromSurface(renderer, surface);
SDL_QueryTexture(texture, NULL, NULL, &texW, &texH);
SDL_Rect dst_rect = { 0, 0, texW, texH };

// Inside the loop
SDL_RenderClear(renderer);
SDL_RenderCopy(renderer, texture, NULL, &dst_rect);
SDL_RenderPresent(renderer);
```

This method is good for a text that will always be the same (ie title). For a text that will oftenly change (ie score, dialogs), this is not efficient because you will have to regenerate surface and texture.

To do this a better solution is to generate a texture with all the glyphe you need (ASCII table).

Then you will compute the offset of each character. You can do it with:

```
int TTF_GlyphMetrics(TTF_Font *font,
                     Uint16   ch,
                     int      *minx,
                     int      *maxx,
                     int      *miny,
                     int      *maxy,
                     int      *advance)
```

Finally you can create a function that will look like this:

```
void display(char *string)
{
  int position = 0;
  while (string)
  {
    int offset = offset_of(*string);
    int length = offset_of(*string + 1) - offset;
```

(continues on next page)

(continued from previous page)

```
    SDL_Rect src = { offset, 0, length, texture.height };
    SDL_Rect dst = { position, 0, lenght, texture.height };
    SDL_RenderCopy(renderer, texture, &src, &dst);

    position += length;
  }
}
```

20.6 Error handling

```
char *TTF_GetError();
```

This returns the reason of the last error.

Usually used like this:

```
if (TTF_Init() == -1)
{
  printf("TTF_Init: %s\n", TTF_GetError());
  exit(2);
}
```

Do not be sorry. Be better.