Using SDL

The simple direct media layer or SDL “is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer.”

The main website is www.libsdl.org and has two major versions 1.x and 2.x. The newer 2. version handles most major platforms and OpenGL context creation and we will be using this for the following exercises.

sdl2-config

When sdl is installed a script is installed in the /usr/bin/ called sdl2-config. This script can be run to get the correct build flags for sdl there is also an sdl-config for earlier SDL 1.x libs as well.

```
SDL2-CONFIG --CFLAGS --LIBS
-I/USR/LOCAL/INCLUDE/SDL2 -I/USR/X11R6/INCLUDE -D_THREAD_SAFE
-L/USR/LOCAL/LIB -LSDL2
```

This will give the installed location for the SDL libraries and include files and we can add this to either our Makefile or Qt project.

Initialising SDL

To use SDL we first #include <SDL.h> then call the main init function as follows

```
#include <SDL.h>

int main()
{

    if (SDL_Init(SDL_INIT_EVERYTHING) == -1)
    {
        std::cerr<<SDL_GetError()<<"\n";
        return EXIT_FAILURE;
    }
}
```

The following flags can be or’ed together ( | ) to combine them
Creating a Window

To create a window we use the following code.

```c
const static int WIDTH=720;
const static int HEIGHT=576;

SDL_Window *win = 0;
win = SDL_CreateWindow("Drawing", 100, 100, WIDTH, HEIGHT,
    SDL_WINDOW_SHOWN | SDL_WINDOW_RESIZABLE);
if (win == 0)
{
    std::cerr<<SDL_GetError()<<"\n";
    return EXIT_FAILURE;
}
```

The function parameters are as follows

```c
SDL_Window* SDL_CreateWindow(const char* title,
    int x,
    int y,
    int w,
    int h,
    Uint32 flags)
```
| **title** | the title of the window, in UTF-8 encoding |
| **x** | the x position of the window, SDL_WINDOWPOS_CENTERED, or SDL_WINDOWPOS_UNDEFINED |
| **y** | the y position of the window, SDL_WINDOWPOS_CENTERED, or SDL_WINDOWPOS_UNDEFINED |
| **w** | the width of the window |
| **h** | the height of the window |
| **flags** | 0, or one or more SDL_WindowFlags |

Windows flags can be combined together using the logical or (|) operator.

| SDL_WINDOW_FULLSCREEN | fullscreen window |
| SDL_WINDOW_FULLSCREEN_DESKTOP | fullscreen window at the current desktop resolution |
| SDL_WINDOW_OPENGL | window usable with OpenGL context |
| SDL_WINDOW_SHOWN | window is visible |
| SDL_WINDOW_HIDDEN | window is not visible |
| SDL_WINDOW_BORDERLESS | no window decoration |
| SDL_WINDOW_RESIZABLE | window can be resized |
| SDL_WINDOW_MINIMIZED | window is minimized |
| SDL_WINDOW_MAXIMIZED | window is maximized |
| SDL_WINDOW_INPUT_GRABBED | window has grabbed input focus |
| SDL_WINDOW_INPUT_FOCUS | window has input focus |
| SDL_WINDOW_MOUSE_FOCUS | window has mouse focus |
| SDL_WINDOW_FOREIGN | window not created by SDL |
| SDL_WINDOW_ALLOW_HIGHDPI | window should be created in high-DPI mode if supported (>= SDL 2.0.1) |
| SDL_WINDOW_MOUSE_CAPTURE | window has mouse captured (unrelated to INPUT_GRABBED, >= SDL 2.0.4) |
SDL Renderer

All access to rendering within SDL is done via the renderer. The renderer is associated with a window and is abstracted so one renderer type can handle both Hardware, software and OpenGL rendering.

To create a renderer we need to pass the active window to the call to create the renderer as shown.

```c
SDL_Renderer *ren = 0;
ren = SDL_CreateRenderer(win, -1, SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);
// check to see if this worked
if (ren == 0)
{
    std::cerr<<SDL_GetError()<<"\n";
    return EXIT_FAILURE;
}
```

The 2nd flag is the index of the rendering driver to initialise, or -1 to initialise the first one supporting the requested flags.

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDL_RENDERER_SOFTWARE</td>
<td>the renderer is a software fallback</td>
</tr>
<tr>
<td>SDL_RENDERER_ACCELERATED</td>
<td>the renderer uses hardware acceleration</td>
</tr>
<tr>
<td>SDL_RENDERER_PRESENTVSYNC</td>
<td>present is synchronized with the refresh rate</td>
</tr>
<tr>
<td>SDL_RENDERER_TARGETTEXTURE</td>
<td>the renderer supports rendering to texture</td>
</tr>
</tbody>
</table>

Render Functions

The SDL Renderer has many render functions we can use, however for the following exercises we are going to concentrate on only setting a pixels colours and placing a pixel at an x,y position on the screen.

```c
// set draw colour to white
SDL_SetRenderDrawColor(ren, 255, 255, 255, 255);
SDL_RenderClear(ren);
```

The above code sets the Renderer draw colour to white, the function takes the renderer as the first parameter then 4 unsigned int parameters to represent an RGBA colour tuple, with 0 as off and 255 as fully on for each channel.
The colour state set with this call will be persistent until another call to SetRenderDrawColor.

**DrawPixel**

```c
int SDL_RenderDrawPoint(SDL_Renderer* renderer,
                        int x,
                        int y)
```

To draw a single pixel in the current colour we use the SDL_RenderDrawPoint passing in the x,y co-ordinates.

The window has (0,0) at the top left corner of the screen, with (WIDTH,HEIGHT )at the bottom right.

The following code draws a single line a pixel at a time.

```c
drw::Colour4i c={255,0,0,255};
drw::Point2i p={0,200};
SDL_SetRenderDrawColor(ren,c.r,c.g,c.b,c.a);
for(int x=0; x<WIDTH; ++x)
{
   SDL_RenderDrawPoint(ren,x,p.y);
}
```

**Finish Drawing**

Each of the render functions work by filling in the back buffer, to swap the buffers so the back buffer is visible we use the following code.

```c
SDL_RenderPresent(ren);
```
SDL Event processing

The usual method of creating an SDL program is to do initial configuration such as creating the windows / renderer and setup any parameters required by the program.

Once this is done we go into an infinite loop and process events, followed by drawing. This is shown in the following code segment.

```c
while (quit != true) {
    SDL_Event event;
    // grab the SDL even (this will be keys etc)
    while (SDL_PollEvent(&event)) {
        // look for the x of the window being clicked and exit
        if (event.type == SDL_QUIT)
            quit = true;
        // check for a key down
        if (event.type == SDL_KEYDOWN) {
            switch (event.key.keysym.sym) {
                // if we have an escape quit
                case SDLK_ESCAPE : quit=true; break;
            }
        }
    }
    // draw and process
    SDL_RenderPresent(ren);
    SDL_Delay(15);
}
```

Typically we can put a pause at the end of this loop to stop it thrashing the processor.