Laboratory work 1

1 Objectives

The objective of this laboratory is to describe briefly the SDL library and to exemplify the development of a basic SDL based application.

2 Create a window using SDL

SDL (Simple DirectMedia Layer) library manages the access to graphics hardware (via OpenGL and Direct3D libraries) and also to audio, keyboard and mouse (independent on the underlying operating system). It supports Windows, Mac OS X and Linux and various other platforms. During this laboratory we will be using the C++ programming language but there are available bindings to other languages (such as C# or Python).

In order to create an SDL window the following steps are required:

1. Initialize the SDL library by calling the **SDL_Init()** function with SDL_INIT_VIDEO as argument because we are using only the video subsystem of the SDL.

```
SDL_Init(SDL_INIT_VIDEO);
```

2. Create the window using the **SDL_CreateWindow()** function. The arguments are the window title, position, width, height and some flags (for example to create a fullscreen window or a resizable window).

```
SDL_CreateWindow("SDL Hello World Example", SDL_WINDOWPOS_UNDEFINED,
SDL_WINDOWPOS_UNDEFINED, WINDOW_WIDTH, WINDOW_HEIGHT, SDL_WINDOW_SHOWN |
SDL_WINDOW_ALLOW_HIGHDPI);
```

Before closing the application we need to deallocate all the resources we created:

1. Destroy the window by calling the **SDL_DestroyWindow()** function and passing as the argument the pointer to the window.

```
SDL_DestroyWindow(window);
```

2. Call the **SDL_Quit()** function that is responsible to clean up all initialized subsystems (for our example we are using only the video subsystem).

```
SDL_Quit();
```

3 SDL Surface

In order to draw something onto the screen we need a canvas. In SDL the canvas can be represented by **SDL_Surface** or **SDL_Texture**. SDL_Surface is used in software rendering, instead SDL_Texture is used in hardware rendering, the main difference being the location of data (pixel) buffers. In this laboratory we will be using SDL_Surface because we will implement the graphics pipeline from scratch.

In SDL a surface is a structure containing a collection of pixels. Each window has attached such a surface and in addition we can create new surfaces and apply some operations on them (for instance we can load an image in a surface and copy the content to the window). The **SDL_Surface** stores the format of the pixels, the dimension (width and height), a pointer to the actual pixel data and other relevant information. We will use a 32-bit pixel representation, in this situation we store 1 byte per channel (red, green, blue and alpha).

If we want to draw a rectangle we need to specify the starting position of the rectangle (the x and y coordinates) and the dimension (width and height). In order to specify the color we are using the **SDL_MapRGB()** function using the pixel format of the surface and the Red, Green and Blue values. The **SDL_FillRect()** function will update the surface with the specified rectangle and color.

```
SDL_Rect rectangleCoordinates = {100, 100, 200, 200};
Uint32 rectagleColor = SDL_MapRGB(windowSurface->format, 255, 0, 0);
SDL_FillRect(windowSurface, &rectangleCoordinates, rectagleColor);
```

For each color channel we specify the values between 0 and 255. Look at the following table for some color channel values, by combining the channel values we get different colors.

Color name	Red channel	Green channel	Blue channel	Color
White	255	255	255	⬜
Red	255	0	0	🟥
Green	0	255	0	🟩
Blue	0	0	255	🟦
Yellow	255	255	0	🟨
Black	0	0	0	⬛

4 Process events

In SDL the events could be things like pressing a keyboard key or mouse motion. All the events are stored in a queue in the order in which they occurred. By using the **SDL_WaitEvent()** function we get the next event from the queue.

4.1 Mouse pressed event

We can check if the left mouse button is pressed using the following piece of code, first we check the event type to be SDL_MOUSEBUTTONDOWN and the pressed button to be SDL_BUTTON_LEFT. The SDL_GetMouseState() function gets the coordinates of the mouse current position (x and y).

```
if(currentEvent.type == SDL_MOUSEBUTTONDOWN)
   if(currentEvent.button.button == SDL_BUTTON_LEFT)
```

```
        SDL_GetMouseState(&mouseX, &mouseY);
```

4.2 Mouse move event

We can check if the left mouse button is pressed while the mouse is moving using the following piece of code, first we check the event type to be SDL_MOUSEMOTION and the pressed button to be SDL_BUTTON_LEFT.

```
if(currentEvent.type == SDL_MOUSEMOTION)
   if(currentEvent.button.button == SDL_BUTTON_LEFT)
      SDL_GetMouseState(&mouseX, &mouseY);
```

4.3 Keyboard event

We can get the key pressed by using the following piece of code, first we check the event type to be SDL_KEYDOWN and then we process the desired keys.

```
if(currentEvent.type == SDL_KEYDOWN)
   switch(currentEvent.key.keysym.sym)
   {
      case SDLK_UP:
         //process UP key
      break;

      case SDLK_r:
          //process R key
      break;
      ...
      default:
         //default process
      break;
   }
```

5 Further reading

- Setting up SDL on various platforms – http://lazyfoo.net/SDL_tutorials/lesson01/index.php
- SDL Tutorials - http://lazyfoo.net/SDL_tutorials/

6 Assignment

Download and run the application from the laboratory website. Try to understand the basic example and then extend the application with the following functionality:

- Change the rectangle color by selecting the color channel using the R (for red), G (for green) and B (for blue) keys and by selecting the channel value by pressing the UP and DOWN keys.
- Interactively display a rectangle, the first set of coordinates is retrieved at left mouse button down event and the second set of coordinate is retrieved at every mouse move event (but only if the left button is still pressed).