

SDL Dokumentation Marcel Römer

Dienstag, 26. April 2016

22:30

Includierungen:	
-----------------	--

<SDL2/SDL.h> SDL

<SDL_image.h> SDL_Image

<SDL_ttf.h> TTF Text

<SDL_mixer.h>

Offizielle Dokumentation mit allen Funktionen

<https://wiki.libsdl.org/CategoryAPI>

SDL Initialisierung und Beendigung

SDL_Init(Uint32 flag);

Flags:

SDL_INIT_TIMER	timer subsystem
SDL_INIT_AUDIO	audio subsystem
SDL_INIT_VIDEO	video subsystem. Automatically initializes the SDL_INIT_EVENTS subsystem
SDL_INIT_JOYSTICK	joystick subsystem
SDL_INIT_HAPTIC	haptic (force feedback) subsystem
SDL_INIT_GAMECONTROLLER	controller subsystem. Automatically initializes the SDL_INIT_JOYSTICK subsystem
SDL_INIT_EVENTS	events subsystem
SDL_INIT EVERYTHING	all of the above subsystems
SDL_INIT_NOPARACHUTE	compatibility; this flag is ignored

SDL_Quit();

Window und Surface erstellen und Löschen

Window erstellen

SDL_Window* window = NULL;

Surface erstellen

SDL_Surface* surface = NULL;

Window löschen

SDL_DestroyWindow(window);

Surface löschen

SDL_FreeSurface(surface);

Window intialisieren

Window = SDL_CreateWindow("SDL Tutorial", SDL_WINDOWPOS_UNDEFINED, SDL_WINDOWPOS_UNDEFINED, SCREEN_WIDTH, SCREEN_HEIGHT, SDL_WINDOW_SHOWN);

SDL_CreateWindow Parameter sind:

1. Name des Windows
- 2&3. X,Y Koordinaten zum Bildschirm,
- 4.Flags

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 (\geq SDL 2.0.1)
SDL_WINDOW_MOUSE_CAPTURE	window has mouse captured (unrelated to INPUT_GRABBED, \geq SDL 2.0.4)

Es wird nicht direkt auf das Window gerendert, sondern auf das Surface des Windows.

Surface zum Surface des Windows initialisieren

```
surface = SDL_GetWindowSurface (window);
```

surface ist nun das Surface des Windows.

Normales Surface Initialisieren

```
SDL_Surface* HelloWorld = SDL_LoadBMP("HelloWorld.bmp");
```

Als PNG Laden

SDL_image.h muss inkludiert und gelinkt werden

IMG_Init(IMG_INIT_PNG); zum Beenden muss **IMG_Quit();** ausgeführt werden.

```
SDL_Surface* HelloWorld = IMG_Load("HelloWorld.png");
```

Surface auf dem Window darstellen

Da nicht direkt auf das Window sondern auf das Surface des Windows gerendert wird, muss ein anderes Surface auf das Window Surface gerendert werden.

```
SDL_BlitSurface ( HelloWorld, NULL, surface, NULL);
```

Danach muss das Window noch aktualisiert werden

```
SDL_UpdateWindowSurface(window);
```

Parameter sind:

1. Quellsurface
2. SDL_Rect mit Struktur des zu kopierenden Quellsurfaces oder NULL, um das komplette Quellsurface zu kopieren
3. Zielsurface
4. SDL_Rect mit Struktur des Zielsurfaces oder NULL, für das komplette Zielsurface

Surface in das gleiche Format wie das Window konvertieren, für bessere Performance

```
SDL_Surface* betterhelloworld = SDL_ConvertSurface( helloworld, windowSurface->format, 0);
```

Parameter:

1. Source Surface
2. Surface des Windows->format
3. 0, NULL

Surface auf anderes Surface stretchen

```
SDL_Rect stretchRect;  
stretchRect.x = 0;  
stretchRect.y = 0;  
stretchRect.w = SCREEN_WIDTH;  
stretchRect.h = SCREEN_HEIGHT;
```

```
SDL_BlitScaled( gStretchedSurface, NULL, gScreenSurface, &stretchRect );
```

Parameter:

1. Das Zielsurface
2. NULL
3. Das Quellsurface, das gestretched wird
4. Adresse eines SDL_Rect mit den neuen Werten

Surface speicher löschen

```
SDL_FreeSurface(SDL_Surface*);
```

Renderer und Textures

Renderer Deklarieren und Definieren für Geometrie oder anderes

```
SDL_Renderer* Renderer = SDL_CreateRenderer ( Window, -1, SDL_RENDERER_ACCELERATED );
```

Parameter:

1. SDL_Window* auf das gerendert werden soll
2. Index des Render Treibers oder -1 für den ersten, der unterstützt wird (kommt auf die flags an)
3. Flags

SDL_RENDERER_SOFTWARE	the renderer is a software fallback
SDL_RENDERER_ACCELERATED	the renderer uses hardware acceleration
SDL_RENDERER_PRESENTVSYNC	present is synchronized with the refresh rate
SDL_RENDERER_TARGETTEXTURE	the renderer supports rendering to texture

Farbe, mit der gerendert wird definieren

```
SDL_SetRenderDrawColor ( Renderer, 0xFF, 0xFF, 0xFF, 0xFF );
```

Parameter:

1. Der Renderer, der benutzt wird (SDL_Renderer*)
2. 2/3/4 RGB Farben
3. Alpha Wert der benutzt werden soll

Texture Deklarieren und Definieren

`SDL_Texture* Texture = SDL_CreateTextureFromSurface (Renderer, Surface);` Texture hat nun die Eigenschaften des Surfaces.

Parameter:

1. Renderer, der benutzt werden soll
2. Surface, von welchem die Werte übernommen werden sollen (x,y,w,h usw.)

Window Clearen

`SDL_RenderClear(Renderer);` cleared Window mit der bei `SetRenderDrawColor` definierten Farbe.

Render Viewport einstellen

Der Render Viewport bestimmt, wo gerendert wird.

`SDL_RenderSetViewport (Renderer, &topLeftViewport);`

Parameter:

1. Der Renderer, mit dem gerendert wird
2. Der Bereich, auf dem gerendert werden soll (`SDL_Rect`)

int	x	the x location of the rectangle's upper left corner
int	y	the y location of the rectangle's upper left corner
int	w	the width of the rectangle
int	h	the height of the rectangle

Farbe, die nicht gerendert werden soll festlegen (ColorKey)

`SDL_SetColorKey (SDL_Surface* surface, int flag, Uint32 key)`

Bsp:

`SDL_SetColorKey (surface, SDL_TRUE, SDL_MapRGB (surface->format, 0, 0, 0));`

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

Parameter:

1. `SDL_Surface*`, bei dem das ColorKey gesetzt werden soll
2. `SDL_TRUE` oder `SDL_FALSE` (aktivieren und deaktivieren des ColorKeys)
3. `SDL_MapRGB (SDL_MapRGB Parameter: 1. SDL_PixelFormat des Surface , 2/3/4 RGB FARBEN des ColorKeys)+`

Farben einer Textur verändern

Mit `SDL_SetTextureColorMod(SDL_Texture* t, Uint8 r, Uint8 g, Uint8 b);` können die Farben einer textur beeinflusst werden.

Parameter:

1. `SDL_Texture*` Textur, die verändert werden soll

2. 2/3/4 RGB Farben

Transparenz einer Textur verändern

Mit `SDL_SetTextureAlphaMod (SDL_Texture* t, Uint8 alpha)` kann die Transparenz eingestellt werden.

Parameter:

1. `SDL_Texture*`, bei der die Transparenz verändert werden soll
2. `Uint8` Alpha wert (0 = 0%, 255 = 100%)

Mit `SDL_SetTextureBlendMode(SDL_Texture* t, SDL_BlendMode)` kann eingestellt werden, ob die Textur durch andere Texturen Transparent ist.

Parameter:

1. `SDL_Texture*`
2. `SDL_BlendMode`

<code>SDL_BLENDMODE_NONE</code>	no blending $\text{dstRGBA} = \text{srcRGBA}$
<code>SDL_BLENDMODE_BLEND</code>	alpha blending $\text{dstRGB} = (\text{srcRGB} * \text{srcA}) + (\text{dstRGB} * (1 - \text{srcA}))$ $\text{dstA} = \text{srcA} + (\text{dstA} * (1 - \text{srcA}))$
<code>SDL_BLENDMODE_ADD</code>	additive blending $\text{dstRGB} = (\text{srcRGB} * \text{srcA}) + \text{dstRGB}$ $\text{dstA} = \text{dstA}$
<code>SDL_BLENDMODE_MOD</code>	color modulate $\text{dstRGB} = \text{srcRGB} * \text{dstRGB}$ $\text{dstA} = \text{dstA}$

Textur Rendern

`SDL_RenderCopy(Renderer, Texture, NULL, NULL);`

Parameter:

Renderer, auf den gerendert wird

Textur, die gerendert wird

`SDL_Rect*` um zu bestimmen, was von der Textur gerendert wird

`SDL_Rect*` um zu bestimmen, auf welchen Bereich des Renderers gerendert wird

Window, Renderer Updaten

`SDL_RenderPresent(gRenderer);`

Speicher Löschen:

`SDL_DestroyTexture(Texture);`

`SDL_DestroyRenderer(Renderer);`

Animationen, Rotation und Vsync

Vsync aktivieren

Man aktiviert Vsync, in dem beim initialisieren des Renderers `SDL_RENDERER_PRESENTVSYNC` als flag angibt.

Bsp: `SDL_Renderer* renderer = SDL_CreateRenderer(window, -1, SDL_RENDERER_ACCELERATED | SDL_RENDERER_PRESENTVSYNC);`

Animationen erstellt man, indem sprites mit den Animationen nacheinander gerendert werden (z.B. jede 10 frames).

Textur Rotiert und geflipped* Rendern

Man Rendert eine Textur Rotiert und geflipped mit der Funktion `SDL_RenderCopyEx`

Parameter:

1. `SDL_Renderer*` mit dem gerendert wird
2. Textur, die gerendert wird
3. `SDL_Rect*` des bereichs der Textur, der gerendert werden soll
4. `SDL_Rect*` des bereichs des Renderers, auf den gerendert werden soll
5. Double in degrees der Rotation
6. `SDL_Point*` des punktes, an dem Rotiert wird
7. `SDL_RenderFlip` horizontal oder vertikal Spiegeln (`SDL_FLIP_NONE`, `SDL_FLIP_HORIZONTAL`, `SDL_FLIP_VERTICAL`)

*= Horizontal, Vertikal spiegeln

Text mit Fonts rendern

SDL_ttf.h muss inkludiert werden

TTF muss Initialisiert werden mit `TTF_Init();`

Außerdem muss es am ende beenden werden mit `TTF_Quit();`

Um text mit fonts zu rendern braucht man ein `TTF_Font` . Dieses Font Intialisiert man mit `TTF_Font* font = TTF_OpenFont (std::string ttfPath, int pSize)`

Parameter:

1. String mit dem Pfad zur TTF datei des fonts
2. Schriftgröße als int

Text mit Font erstellen

Der Text wird einem surface übergeben und danach kann es falls nötig wie gewohnt in ein Texture umgewandelt werden.

`SDL_Surface* textSurface = TTF_RenderText_Solid(font, text, color);`

Parameter:

1. `TTF_Font*`, das benutzt werden soll
2. Der Text als string
3. `SDL_Color` mit der Farbe

`SDL_Color` member sind:

Uint8	r	the red component in the range 0-255
Uint8	g	the green component in the range 0-255
Uint8	b	the blue component in the range 0-255
Uint8	a	the alpha component in the range 0-255

Font löschen

```
TTF_CloseFont( TTF_Font* font);
```

Geometrie Rendern

Es wird mit der **SDL_SetRenderDrawColor** definierten Farbe gerendert

Gefülltes Rechteck

Rect für Position und Größe des Rechtecks

```
SDL_Rect fillRect = { SCREEN_WIDTH / 4, SCREEN_HEIGHT / 4, SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2 };
```

Form des Rects rendern

```
SDL_RenderFillRect( gRenderer, &fillRect );
```

Nur Umrandung des Rechtecks rendern

```
SDL_Rect outlineRect {SCREEN_WIDTH / 6, SCREEN_HEIGHT / 6, SCREEN_WIDTH * 2 / 3, SCREEN_HEIGHT * 2 / 3};
```

```
SDL_RenderDrawRect( Renderer, &outlineRect);
```

Linie Rendern

```
SDL_RenderDrawLine( Renderer, 0, SCREEN_HEIGHT / 2, SCREEN_WIDTH, SCREEN_HEIGHT / 2 );
```

Punkt Rendern

```
SDL_RenderDrawPoint( Renderer, SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2 );
```

Events

In SDL, Events dienen dazu Tastatureingaben, Mauseingaben, Joystickeingaben oder ähnliches zu speichern, um auszuwerten.

SDL_Event ist ein 8 bit Union mit diesen Members:

Quelle: <https://www.libsdl.org/release/SDL-1.2.15/docs/html/sdlevent.html>


```
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_ExposeEvent expose;
    SDL_QuitEvent quit;
    SDL_UserEvent user;
    SDL_SysWMEvent syswm;
} SDL_Event;
```

Member Bedeutungen:

<i>type</i>	The type of event
<i>active</i>	<u>Activation event</u>
<i>key</i>	<u>Keyboard event</u>
<i>motion</i>	<u>Mouse motion event</u>
<i>button</i>	<u>Mouse button event</u>
<i>jaxis</i>	<u>Joystick axis motion event</u>
<i>jball</i>	<u>Joystick trackball motion event</u>
<i>jhat</i>	<u>Joystick hat motion event</u>
<i>jbutton</i>	<u>Joystick button event</u>
<i>resize</i>	<u>Application window resize event</u>
<i>expose</i>	<u>Application window expose event</u>
<i>quit</i>	<u>Application quit request event</u>
<i>user</i>	<u>User defined event</u>
<i>syswm</i>	<u>Undefined window manager event</u>

Type enumerations, um das Event zu bestimmen

Event type	Event 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_VIDEOEXPOSE	SDL_ExposeEvent
SDL_USEREVENT	SDL_UserEvent

Mit `SDL_PollEvent(SDL_Event* e)` kann ein Event abgefrag werden.

Beispiel:

```
Bool quit = false;
```

```
SDL_Event e;
```

```
While (quit==false){
```

```
While ( SDL_PollEvent(&e) != 0 ){
```

```
    If (e.type == SDL_QUIT){
```

```
        Quit = true;
```

```
    }
```

```
}
```

```
}
```

Tastaturabfragen mit Events

SDL mit ESC beenden

```
Bool quit = false;
```

```
SDL_Event e;
```

```
While (quit==false){
```

```
While ( SDL_PollEvent(&e) != 0 ){
```

```
    If (e.type == SDL_KEYDOWN){
```

```
        Switch (e.key.keysym.sym){
```

```
            Case SDLK_ESCAPE: quit = true;
```

```

    }

}

}

}

```

Infos:

https://wiki.libsdl.org/SDL_KeyboardEvent

https://wiki.libsdl.org/SDL_Keysym

https://wiki.libsdl.org/SDL_Keycode

Mouse Event

Um ein Mausevent abzufragen ist ein `SDL_Event*` nötig. Mit `SDL_PollEvent` muss dann das Event abgefangen werden und danach kann man die Maus Events abfragen.

```

Bsp: if ( e->type == SDL_MOUSEMOTION || e->type == SDL_MOUSEBUTTONDOWN || e->type ==
SDL_MOUSEBUTTONUP){
...
}

```

`SDL_MOUSEMOTION` = Maus wurde bewegt

`SDL_MOUSEBUTTONDOWN` = Maustaste wurde gedrückt

`SDL_MOUSEBUTTONUP` = Maustaste wurde losgelassen

Mausposition abfragen

Die Mausposition wird mit der Funktion `SDL_GetMouseState (int* x, int* y)`

Die Funktion übergibt den zwei Parametern die Koordinaten.

Gedrückte Maustaste erkennen

Dies funktioniert auch mit dem Event.

```

Bsp: if ( e->button.button == SDL_BUTTON_LEFT){
Cout<<"linke maustaste wurde gedrückt"<<endl;

```

`E->button` ist ein member des `SDL_Event` unions.

Es hat selbst member:

```

typedef struct{
    Uint8 type;
    Uint8 button;
    Uint8 state;
    Uint16 x, y;
} SDL_MouseButtonEvent;

```

Type ist entweder `SDL_BUTTONDOWN` oder `SDL_BUTTONUP` also zur erkennung ob eine Taste gedrückt oder losgelassen wird

Button ist zur erkennung welche Maustaste gedrückt wurde also `SDL_BUTTON_LEFT / MIDDLE / RIGHT`

Dies kann man alles in der offiziellen Dokumentation nachlesen:

<https://www.libsdl.org/release/SDL-1.2.15/docs/html/sdlmousebuttonevent.html>

Keyboard abfragen ohne Events

Auch wenn die Tastatur nicht mit Events abgefragt wird, muss SDL_PollEvent vorhanden sein, um die abfrage zu aktualisieren.

Mit der Funktion SDL_GetKeyboardState (int* numkeys) kann eine Tastaturabfrage gemacht werden. Diese Funktion returned ein Zeiger auf das Array mit den Werten der einzelnen Tasten. Dies ist ein const Uint8* .

Bsp: `const Uint8* status = SDL_GetKeyboardState(NULL);`

```
If (state[SDL_SCANCODE_RIGHT] == 1) {  
    Cout<<"Right key is pressed";  
}
```

Alle Scancodes https://wiki.libsdl.org/SDL_Scancode

Zeitfunktionen

Zeit abwarten

`SDL_Delay(Uint32 milliseconds);`